Pós-Graduação em Ciência da Computação

ANDRESSON DA SILVA FIRMINO

Métodos de Otimização Aplicados ao Problema de Recuperação de Contêineres

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

ANDRESSON DA SILVA FIRMINO

Métodos de Otimização Aplicados ao Problema de Recuperação de Contêineres

Tese apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciências da Computação.

Área de Concentração: Análise de Algoritmos e Complexidade de Computação.

Orientador: Prof. Dr. Ricardo Martins de Abreu Silva.
Coorientador: Prof.ª Dr.ª Valéria Cesário Times.

Recife
2019

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

F525m Firmino, Andresson da Silva
Métodos de otimização aplicados ao problema de recuperação de contêineres / Andresson da Silva Firmino. – 2019.
114 f.: il., fig., tab.

Orientador: Ricardo Martins de Abreu Silva.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.
Inclui referências.

1. Ciência da computação. 2. Otimização. 3. Algortimos. I. Silva, Ricardo Martins de Abreu (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2019-028

ANDRESSON DA SILVA FIRMINO

“Métodos de Otimização Aplicados ao Problema de Recuperação de Contêineres”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Aprovado em: 22/02/2019.

_____
Orientador: Prof. Dr. Ricardo Martins de Abreu Silva

BANCA EXAMINADORA

_____
Prof. Dr. Silvio de Barros Melo (Examinador Interno)
Centro de Informática /UFPE

_____
Prof. Dr. Tsang Ing Ren (Examinador Interno)
Centro de Informática /UFPE

_____
Prof. Dr. Kelvin Lopes Dias (Examinador Interno)
Centro de Informática / UFPE

_____
Prof. Dr. Jones Oliveira de Albuquerque (Examinador Externo)
Departamento de Estatística e Informática/ UFRPE

_____
Prof. Dr. Thiago Ferreira de Noronha (Examinador Externo)
Departamento de Ciência da Computação/UFMG

Dedico o resultado deste esforço aos meus pais, Firmino e Creusa, à minha amada esposa, Etiane, e à nossa princesa Sarah.

AGRADECIMENTOS

Primeiramente, agradeço ao ser mais importante em minha história. Aquele me fez existir, que me deu saúde, sabedoria e todos os instrumentos necessários para enfrentar os percalços da vida, e assim chegar onde hoje estou: *Deus*. Agradeço aos meus pais José Firmino e Creusa Firmino. Sou grato por suas posições firmes que modelaram meu caráter, me mostraram o caminho e por todo seu imenso carinho e amor, a mim dedicados.

À minha irmã Andresa Firmino, que sempre torceu por mim e me ajudou em várias questões durante toda a minha jornada acadêmica. À Etiane, minha amada esposa, a minha gratidão por seu amor, compreensão, companheirismo, e apoio incondicional nos momentos felizes e difíceis. Para você, um beijo de muito amor. À Sarah, minha querida filha, que nasceu no último ano desse doutorado, obrigado pelas suas interrupções, pois elas foram, por mais incoerente que isso pareça, uma constante fonte de carinho, reflexão, alívio e motivação para continuar esse projeto.

Aos meus colegas de trabalho Rafael Belém, Diogo Guimarães, Diogo Anderson. Meu obrigado pela amizade, pelo companheirismo e pelas produtivas discussões. David Sotello e Diogo Anderson, muito obrigado pelas vossas colaborações no enriquecimento deste trabalho. Ao meu chefe Rodrigo Panza, muito obrigado pela assistência e compreensão. Aos meus irmãos na Fé, Leandro Santos, Gilvan Soares, Rodrigo Santos e Pr. Cristiano Carvalho, obrigado pela amizade, compreensão, torcida e orações.

Sou muito grato aos professores do Centro de Informática da UFPE que contribuíram de alguma forma para meu crescimento. Em especial, os professores Fernando Fonseca e Ana Carolina Salgado, pessoas fundamentais na construção da minha carreira acadêmica, minha profunda gratidão pelos seus ensinamentos, não só como docente, mas como exemplo de ser humano. Também agradeço os professores Thiago Noronha (UFMG), Tsang Ren e Silvio Melo pela vossas valorosas contribuições na banca de exame da qualificação.

Ao Prof.º Ricardo Martins, meu orientador, deixo registrado aqui meus sinceros agradecimentos pela confiança em mim depositada na realização deste trabalho. Sou grato por seus direcionamentos, auxílios, desafios e cobranças. Tudo isso, me fez crescer e tornar uma pessoa e um profissional melhor. Prof.º Ricardo, um forte abraço de agradecimento.

Agradeço em especial à Prof.ª Valéria Times, um presente de Deus nesta minha jornada acadêmica. Minha eterna gratidão, pelo seu cuidado, incentivo (até quando eu não cria), entusiasmo, respeito, bom humor, disponibilidade, confiança e longas discussões no Skype. As suas reflexões e críticas, surgidas do seu saber, experiência e competência profissional, me fizeram um ser humano melhor e guiam-me até hoje. Deus te recompense pelo bem que fizeste a mim e a todos que tiveram a honra de ser seu orientando.

Aos demais familiares e amigos, bem como, a todos que contribuíram direta ou indiretamente para mais essa vitória, meus sinceros agradecimentos.

RESUMO

O Problema de Recuperação de Contêineres (PRC) é um importante campo de pesquisa que visa alcançar eficiência operacional em pátios de um sistema de terminal de contêineres. O PRC envolve encontrar uma sequência ótima de operações para o guindaste, permitindo que ele recupere todos os contêineres de uma baia de acordo com uma ordem predefinida. Uma sequência ótima de operações é obtida reduzindo o tempo de operação despendido pelo guindaste, e esse tempo é geralmente inferido pelo número de realocações de contêineres realizadas. Embora este critério seja a principal função objetivo discutida na literatura, minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste, como tem sido comprovado neste trabalho. Assim, neste estudo, uma trajetória do guindaste é definida como métrica para computar os percursos realizados pelo guindaste a fim de retirar sequencialmente todos os contêineres da baia. A partir dessa métrica, o tempo de operação do guindaste é diretamente computado, em contraste à métrica baseada no número de realocações, onde, indiretamente, obtém-se o tempo de operações guindaste. Além disso, este trabalho propõe métodos de otimização exatos e aproximados para as duas classes de problemas que dividem o PRC segundo a forma de realocação permitida (i.e., PRC restrito e PRC irrestrito). Estas duas classes definem o contexto de resolução do problema: restrito ou irrestrito. Comparando estes dois contextos, no contexto irrestrito, soluções com menor tempo de operação do guindaste podem ser encontradas, mas um tempo computacional maior pode ser requerido para encontrar estas soluções. Portanto, este trabalho investiga a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito em comparação à resolução no contexto restrito. Os resultados experimentais mostram que os métodos de otimização propostos podem fornecer melhores soluções em curto espaço de tempo, quando comparados a outros métodos de otimização na literatura, contribuindo assim com resultados importantes para a área.

Palavras-chaves: Terminal de Contêiner. Otimização. Algoritmos. Heurísticas. Meta-heurísticas.

ABSTRACT

The Container Retrieval Problem (CRP) is an important field of research that aims to achieve high yard operational efficiency in a container terminal system. The CRP involves finding an optimal sequence of operations for the crane, enabling it to retrieve all the containers from the bay according to a predefined order. An optimal sequence of operations is obtained by reducing the operating time spent by the crane, and this time is usually inferred by the number of container relocations made. Although this criterion is the primary objective function discussed in the literature, minimizing the number of relocations does not ensure the solution with the minimal working cost, as has been proven in this work. Thus, in this study, a crane's trajectory is defined as a metric to compute the pathways performed by the crane in order to sequentially remove all the containers from the bay. From this metric, the crane's operating time is directly computed, in contrast to the metric based on the number of relocations, where, indirectly, the crane operations time is obtained. Moreover, this work proposes exact and approximate optimization methods for the two class of problem which divides the CRP according to the relocation fashion allowed (i.e., CRP restricted and CRP unrestricted). These two classes define the problem resolution context: restricted or unrestricted. Comparing these two contexts, in the unrestricted context, solutions with shorter crane's operating time can be found, but longer computational time may be required to find these solutions. Therefore, this work investigates the cost-benefit ratio among the decrease rate in the crane's operating time and the increase rate in algorithms runtime, both related to the resolution of the PRC in the unrestricted context as compared to the resolution in the restricted context. The experimental results show that the proposed optimization methods can provide better solutions in a short time, when compared to others optimization methods in the literature, thus contributing to important results for the area.

Keywords: Container Terminal. Optimization. Algorithms. Heuristics. Metaheuristics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A*	*A Star*
BB	*Branch-and-Bound*
BRP	*Blocks Relocation Problem*
BS	*Beam Search*
CM	*Corridor Method*
CRP	*Container Retrieval Problem*
ENAR	*Expected Number of Additional Relocations*
GPT	Guindaste Portuário sobre Trilhos
GRASP	*Greedy Randomized Adaptive Search Procedure*
IDA	*Iterative Deepening A**
LADI	*Lowest Absolute Difference Index*
LRC	Lista Restrita de Candidatos
LSI	*Lowest-Slot Index*
MESPG	*Mutual Exclusion Scheduling for Permutation Graphs*
MNI	*Min-Max Index*
PLI	Programação Linear Inteira
PM	*Pilot Method*
PRC	Problema de Recuperação de Contêineres
RGRASP	*Reactive Greedy Randomized Adaptive Search Procedure*
RI	*Reshuffle Index*
RIL	*Reshuffle with Look-Ahead Index*
TEU	*Twenty feet Equivalent Unit*

SUMÁRIO

1 INTRODUÇÃO

Esta seção contextualiza e discute a importância do Problema de Recuperação de Contêineres (PRC) para a eficiência operacional em terminais de contêineres, motiva o estudo sobre novos métodos de otimização para o PRC, lista os objetivos gerais e específicos almejados e descreve como esta tese está organizada.

1.1 CONTEXTUALIZAÇÃO

Contêiner é um recipiente metálico de transporte e armazenamento de bens que se tornou padrão no fretamento marítimo do mercado internacional. Isto se deve ao fato de que o uso de recipientes como barris, sacas e paletes se tornaram ineficientes na movimentação de mercadorias, pois aumentavam o tempo de carga e descarga, o número de roubos e avarias e o custo de mão-de-obra para manuseios devido às utilizações direta de mão-de obra. O uso do contêiner permite a utilização de equipamentos eficientes de manipulação como guindastes no transporte entre caminhões, trens e principalmente navios. Além disso, o contêiner possibilita a estocagem e transporte de mercadorias em áreas descobertas sob condições climáticas adversas e possui dispositivos de segurança previstos por regimentos nacionais e internacionais. A unidade padrão de capacidade de um contêiner é o *Twenty feet Equivalent Unit* (TEU) que está associado a um contêiner de 20'(pés) de comprimento por 8' de largura e por 8' de altura. Contêineres se diferem por capacidade (1/2, 1, 2 TEU) e por peso de carga suportado (10, 20, 30 toneladas). Ademais, contêineres podem ser especializados, tais como: térmicos (aquecimento e resfriamento de mercadorias), tanques (transporte de líquidos ou gases), granel (transporte de grãos).

O tráfego portuário de contêineres no mundo tem crescido fortemente, passando de 224,7 milhões de TEU movimentados no ano de 2000 para 701,4 milhões de TEU no ano de 2016 (World Bank, 2018). Além disso, as economias em desenvolvimento crescem e atingiram cerca de 71,9% na cota mundial na movimentação portuária de contêineres (UNCTAD, 2015). Isto é uma tendência de crescimento gradual que é ratificada pelo aumento das relações comercias entre os países em desenvolvimento do hemisfério sul (UNCTAD, 2015). Como resultado deste crescimento, o desempenho dos portos e terminais é importante porque afeta a competitividade comercial de um país.

Durante as últimas décadas, melhorar a logística do terminal de contêineres se tornou um campo de destaque na literatura científica (SILVA et al., 2018). A principal motivação por essa melhoria é a necessidade de redução dos custos de transporte e aumento da utilização dos recursos de um terminal de contêineres. O terminal de contêineres lida com a transferência de contêineres entre os diferentes tipos de veículos (trens, navios e caminhões), e o armazenamento de contêineres até o momento de retirada. A gestão

eficiente do tráfego e armazenamento de contêineres no terminal também vem se tornado cada vez mais importante devido às crescentes necessidades de importação e exportação (UNCTAD, 2017). Por limitação da área disponível para armazenamento nos terminais, os contêineres são empilhados verticalmente, formando várias pilhas. As pilhas são agrupadas em blocos e estes são distribuídos ao longo do pátio de contêineres e separados pelas faixas de tráfego de veículos. Um bloco consiste de um grupo paralelo de baias, e uma baia consiste em um grupo paralelo de pilhas, como ilustrado na Figura 1. Uma secção longitudinal de contêineres em uma baia define uma camada, e a altura máxima das pilhas determina o número de camadas de uma baia.

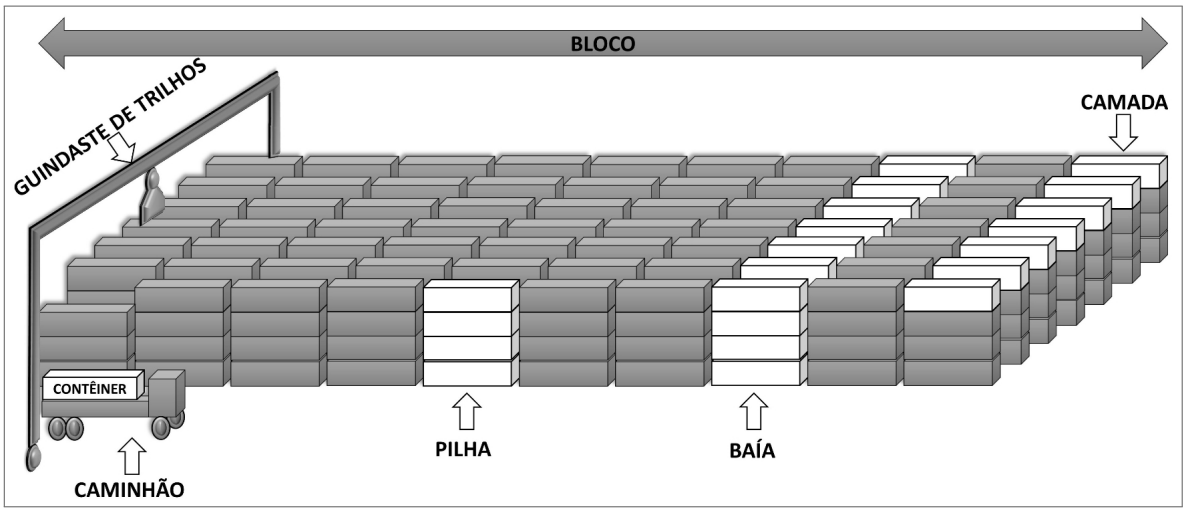


Figura 1 – Exemplo de um pátio e um bloco de contêineres.

Um Guindaste Portuário sobre Trilhos (GPT) é comumente utilizado na movimentação de contêineres dentro de um bloco. No entanto, os contêineres não são transferidos de uma baia para outra pelo GPT. Dado que a maioria dos GPTs não pode deslocar-se através das baias de um bloco enquanto transporta um contêiner (LIN; LEE; LEE, 2015), por exemplo, por razões de segurança (CARLO; VIS; ROODBERGEN, 2014). Portanto, a movimentação de contêineres pelo guindaste ocorre entre pilhas de mesma baia e os contêineres são acessados pelos lados da baía onde situam os caminhões. A retirada de um contêiner que não está no topo da pilha implica na transferência dos contêineres acima dele para outras pilhas. Estes movimentos adicionais consomem tempo e dinheiro, consequentemente reduzem a produtividade no processo de saída de um contêiner. Maximizar a produtividade na retirada de contêineres é conhecido como o Problema de Recuperação de Contêineres (PRC) ou *Container Retrieval Problem* (CRP).

No PRC, cada contêiner é atribuído a um valor correspondente à sua prioridade de saída e todos os contêineres, sequencialmente, são retirados da baia pelo guindaste de acordo com as suas prioridades. As prioridades são comumente atribuídas nos planos de estiva dos navios, onde os contêineres são geralmente classificados de acordo com seus pesos e portos de destino (LEHNFELD; KNUST, 2014). Esta é uma maneira de garantir a

estabilidade do navio e as sequências de recuperação dos contêineres exigidas pelos portos de destino que o navio visitará. Duas operações são realizadas pelo guindaste a fim de recuperar todos os contêineres de uma baia. Uma é a **realocação** pela qual um contêiner situado no topo de um pilha é movido para o topo de outra pilha; e a outra é **retirada** pela qual o contêiner de maior prioridade na baia está situado no topo de uma pilha e é movido para fora da baia. A Figura 2 ilustra a resolução de uma instância do PRC, durante o processo de recuperação dos contêineres, quatro realocações foram realizadas e todos os contêineres foram retirados sequencialmente. O contêiner 8 foi realocado para ter acesso ao contêiner 1 (1ª realocação) e depois o contêiner 1 foi retirado. O contêiner 4 foi realocado para ter acesso ao contêiner 2 (2ª realocação) e após o contêiner 2 foi retirado. O contêiner 6 foi realocado para ter acesso ao contêiner 3 (3ª realocação) e em seguida os contêineres 3, 4 e 5 são retirados nesta ordem. Por fim, o contêiner 8 foi realocado para ter acesso ao contêiner 7 e posteriormente os contêineres 7 e 8 são retirados.

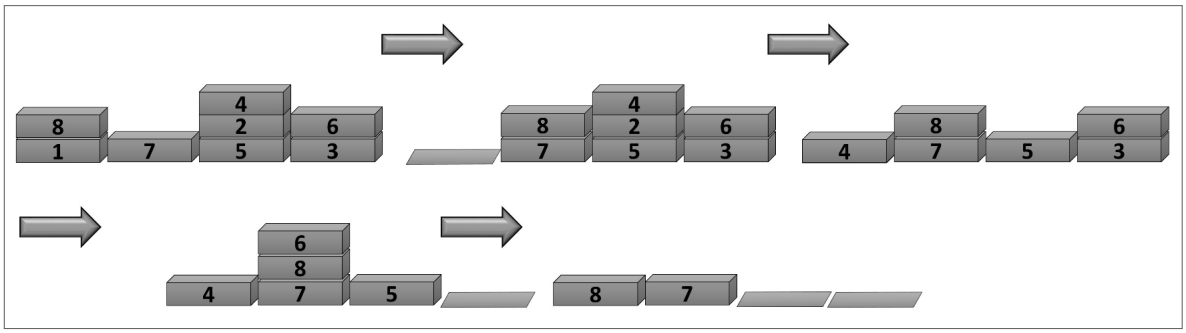


Figura 2 – Exemplo de resolução do Problema de Recuperação de Contêineres.

O trabalho de pesquisa aqui descrito está inserido no contexto de otimização em problemas de operações logísticas em terminais de contêineres e aborda particularmente o PRC. Este problema envolve encontrar uma sequência ótima de operações (i.e., realocações e retiradas) para o guindaste, permitindo que ele recupere todos os contêineres de uma baia de acordo com uma ordem predefinida (i.e., a prioridade de saída). Uma sequência ótima de operações é obtida minimizando o tempo de operação despendido pelo guindaste. O PRC é um problema de otimização combinatória pertencente à classe NP-difícil (LEHNFELD; KNUST, 2014) e de suma importância para terminais de contêineres.

1.2 MOTIVAÇÃO

O transporte marítimo é uma área estratégica e prioritária na agenda de políticas de comércio e desenvolvimento internacional. Além disso, o transporte marítimo é o motor de crescimento para o desenvolvimento na economia mundial. Uma vez que mais de 80% em volume e mais de 70% em valor do comércio global são transportados a bordo de navios e movimentado por portos marítimos em todo o mundo (UNCTAD, 2017). Em vista disso, a necessidade de otimização em operações de terminais de contêineres tornou-se cada vez

mais inevitável nos últimos anos. Sistemas de logística, especialmente em terminais de contêineres de grande porte, já atingiram um grau de complexidade que melhorias são requeridas para otimizar o fluxo de operações. Além disso, as características dinâmicas de operações do terminal de contêineres requerem otimizações e tomadas de decisão em tempo real. Por estas razões, diversos estudos são desenvolvidos para elaboração de algoritmos rápidos e eficazes, a fim de otimizar as atividades em sistemas de terminais de contêineres.

Nesse aspecto, vários trabalhos têm sido desenvolvidos para o PRC com o objetivo de encontrar soluções ótimas ou próximas do ótimo (FIRMINO; SILVA; TIMES, 2018b). Considerando a abordagem de otimização utilizada, estes trabalhos podem ser classificados em abordagens exatas e aproximadas. As abordagens exatas garantem soluções ótimas, mas, na maioria dos casos, requerem elevado tempo computacional. Assim, soluções ótimas são encontradas em tempo computacional viável apenas para pequenas instâncias do problema. Nas abordagens aproximadas, não há garantia de encontrar soluções ótimas para o problema. Mas, soluções próximas da solução ótima podem ser encontradas com um curto tempo computacional até mesmo para as instâncias maiores do problema. Assim, é evidente o desafio de pesquisa existente nas duas abordagens de otimização para o PRC. Na abordagem exata, o desafio é encontrar soluções ótimas para instâncias maiores dentro de um tempo computacional viável. Já, na abordagem aproximada, o desafio é produzir, em tempo reduzido, soluções tão próximas quanto possível da solução ótima.

Tradicionalmente, o principal objetivo de otimização do PRC é minimizar o número de realocações para retirar sequencialmente todo os contêineres de uma baia. Dado que, empiricamente, tendo um menor número de realocações implica um menor tempo de operação do guindaste para remover todos os contêineres da baia, aumentando assim a produtividade e a eficiência nas operações de empilhamento. Entretanto este trabalho tem por hipótese que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste. Desta forma, a trajetória do guindaste é proposta como uma métrica para mensurar os percursos realizados pelo guindaste a fim de retirar sequencialmente todos os contêineres da baia. A partir dessa métrica o tempo de operação do guindaste será diretamente computado em vez da métrica baseada no número de realocações, onde, indiretamente, obtém-se o tempo de operações guindaste. Portanto, melhores soluções serão obtidas para o PRC em termos de tempo de operação do guindaste. Tendo em mente que o tempo é um dos principais fatores utilizados para determinar a performance dos portos (UNCTAD, 2017).

Vale ressaltar que embora alguns poucos trabalhos na literatura tenham o objetivo de minimizar o tempo de operação do guindaste, eles diferem na maneira como o tempo de operação é calculado. Assim, às vezes os tempos de operação do guindaste não são diretamente comparáveis, como mencionado em (LIN; LEE; LEE, 2015). Além disso, alguns destes trabalhos não especificam formalmente como o tempo de operação é computado.

Portanto, é evidente a necessidade de um referencial para o cálculo do tempo de operação do guindaste por meio de uma especificação formal deste cálculo. Além disso, visando minimizar o tempo de operação do guindaste, até então, não se tem conhecimento sobre a aplicação dos algoritmos de busca em caminhos *Dijkstra* (DIJKSTRA, 1959) e *A Star* (A*) (HART; NILSSON; RAPHAEL, 1968), de um modelo matemático de Programação Linear Inteira (PLI) e de algoritmos baseados nas meta-heurísticas *Reactive Greedy Randomized Adaptive Search Procedure* (RGRASP) (PRAIS; RIBEIRO, 2000) e *Pilot Method* (PM) (VOßS; FINK; DUIN, 2005).

Na literatura, o PRC se divide em duas classes segundo a forma de realocação permitida: **PRC restrito** e **PRC irrestrito** (LEHNFELD; KNUST, 2014). No PRC restrito, a pilha de origem de cada movimento de realocação é limitada à pilha na qual o contêiner de maior prioridade na baia está atualmente colocado. Já no PRC irrestrito, a pilha de origem de cada movimento de realocação pode ser qualquer pilha. Como exemplo, a Figura 2 exibe a resolução de uma instância do PRC restrito, a classe do PRC mais abordada na literatura (KU; ARTHANARI, 2016). Consequentemente o PRC restrito, na maioria dos trabalhos, é referenciado simplesmente como PRC.

O PRC restrito é predominante nos trabalhos realizados na literatura porque sua complexidade é menor em comparação ao PRC irrestrito. Isto devido ao espaço de soluções menor no PRC restrito, pois o número de realocações possíveis no PRC restrito é menor quando comparado ao PRC irrestrito. Embora, o PRC irrestrito seja mais complexo que o PRC restrito, o PRC irrestrito gera mais oportunidades de otimização do que o PRC restrito (TRICOIRE; SCAGNETTI; BEHAM, 2018). Em outras palavras, melhores soluções podem ser encontradas no PRC irrestrito, embora este, devido a sua maior complexidade, possa consumir um custo computacional maior na busca de uma solução quando comparado ao PRC restrito. Entretanto, de acordo com o nosso conhecimento, nenhuma das propostas na literatura investiga a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito (i.e., PRC irrestrito) em comparação à resolução no contexto restrito (i.e., PRC restrito). O valor desta razão custo-benefício indica se a resolução do PRC em contexto irrestrito é compensatória, onde valores próximos a zero indicam que a resolução no contexto restrito não é compensatória.

Diante desses fatos, as principais motivações para o desenvolvimento do trabalho aqui proposto são baseadas na percepção das seguintes necessidades: (1) Estudar a relação entre o número de realocações e o tempo de operação do guindaste como objetivos de otimização para o PRC, e, desse modo, demonstrar que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste. (2) Definir formalmente a trajetória do guindaste e utilizá-la para formular o tempo de operação do guindaste; (3) Analisar em métodos exatos e aproximados a razão custo-benefício inerente a resolução do PRC no contexto irrestrito em comparação ao contexto restrito; e (4) Pesquisar novos

métodos de otimização (exatos e aproximados) visando encontrar soluções para o PRC com menor tempo de operação do guindaste.

1.3 PROBLEMA DE PESQUISA E HIPÓTESES

O problema de pesquisa que norteia o desenvolvimento desta tese procura responder as seguintes questões:

- *Questão 1: Será que minimizar o número de realocações garante a solução para o PRC com o tempo mínimo de operação do guindaste?*
- *Questão 2: O PRC pode ser modelado como um problema de busca de caminho mínimo, de modo que as soluções com o tempo mínimo de operação do guindaste sejam obtidas por meios dos algoritmos de busca em caminhos Dijkstra e A* ?*
- *Questão 3: Um modelo matemático de PLI pode ser proposto para encontrar, no PRC, as soluções com o tempo mínimo de operação do guindaste?*
- *Questão 4: Como métodos aproximados para solucionar o PRC, é possível desenvolver algoritmos, baseados nas meta-heurísticas RGRASP e PM, capazes de encontrar, em curto tempo computacional, soluções com tempo de operação do guindaste menor do que nas soluções encontradas por métodos existentes na literatura?*
- *Questão 5: Em quais métodos (exatos e aproximados) a resolução do PRC no contexto irrestrito é compensatória, em comparação ao contexto restrito, conforme os valores da razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução dos algoritmos?*

Para o problema de pesquisa aqui investigado, formulou-se as seguintes hipóteses:

- *Hipótese 1: Minimizar o número de realocações não garante a solução para o PRC com o tempo mínimo de operação do guindaste.*
- *Hipótese 2: Há um mapeamento do PRC ao problema de busca de caminho mínimo e a partir dele os algoritmos Dijkstra e A* podem ser aplicados para encontrar as soluções no PRC com o tempo mínimo de operação do guindaste.*
- *Hipótese 3: Algoritmos baseados nas meta-heurísticas RGRASP e PM podem ser aplicados ao PRC, de modo a serem capazes de encontrar, em curto tempo computacional, soluções com tempo de operação do guindaste menor do que nas soluções encontradas por métodos existentes na literatura.*
- *Hipótese 4: A resolução do PRC no contexto irrestrito em comparação ao contexto restrito deve ser compensatória para métodos aproximados e não compensatória em*

métodos exatos, conforme a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução dos algoritmos.

Conhecidas as questões centrais de investigação, descreve-se na seção seguinte os objetivos gerais e específicos desta tese.

1.4 OBJETIVOS

Para guiar o desenvolvimento desta tese, alguns objetivos foram definidos. O objetivo geral desta tese é propor novos métodos de otimização (exatos e aproximados) visando encontrar soluções para o PRC com menor tempo de operação do guindaste. Isto inclui a formalização da trajetória do guindaste como métrica para computar diretamente o tempo de operação do guindaste no PRC, como também, a análise custo-benefício da resolução do PRC no contexto irrestrito. Para contribuir com o objetivo geral desse trabalho, os objetivos específicos são:

- Especificar formalmente a trajetória do guindaste como uma métrica para mensurar os percursos realizados pelo guindaste e demonstrar que essa nova métrica é mais adequada para computar o tempo de operação do guindaste do que a métrica tradicional baseada no número de realocações.
- Definir e calcular o tempo de operação do guindaste a partir da trajetória do guindaste. Com isso, é possível mensurar e minimizar de maneira direta o tempo de operação do guindaste no PRC.
- Comprovar que minimizar o número de realocações não garante a solução para o PRC com o tempo mínimo de operação do guindaste.
- Formular um modelo matemático de PLI para o PRC, tendo por função objetivo minimizar o tempo de operação do guindaste. O intuito deste modelo é fornecer uma descrição matemática do problema e também possibilitar a resolução do problema por meio de um *solver*[1].
- Modelar o PRC como um problema de busca de caminho mínimo. Com isto, os algoritmos de busca em caminhos *Dijkstra* e A* serão aplicadas para solucionar otimamente o PRC.
- Especificar e implementar algoritmos de otimização para solucionar o PRC e estes baseados nas meta-heurísticas RGRASP e PM.

[1] Um *solver* é um pacote de software matemático capaz de solucionar problemas de otimização a partir de modelos matemáticos.

- Avaliar e analisar, por meio dos experimentos, os resultados obtidos pelos diversos métodos de otimização propostos para solucionar o PRC. Ademais, analisar o custo-benefício na resolução do PRC no contexto irrestrito em comparação ao contexto restrito sobre métodos de otimização exatos e aproximados.

1.5 METODOLOGIA DE PESQUISA

Para a elaboração desta tese, utilizou-se como metodologias de pesquisa a revisão bibliográfica e a pesquisa experimental. A primeira identificou os principais trabalhos correlatos no estado da arte, enquanto que a segunda foi utilizada para avaliar a qualidade das soluções produzidas por diversos métodos de otimização desenvolvidos para solucionar o PRC.

1.6 PUBLICAÇÕES RELACIONADAS A TESE

Esta seção lista as publicações e apresentações produzidas e indica como elas se relacionam com os objetivos deste trabalho.

Definir e implementar algoritmo de otimização baseado na meta-heurística RGRASP.

- FIRMINO, A. S.; SILVA, R. M. A. Reactive grasp to container stacking problem. In: *III Congresso de Matemática Aplicada e Computacional - Sudeste*. [s.n.], 2015. Apresentação no Mini-simpósio: Otimização de Operações Portuárias. Disponível em: <www.sbmac.org.br/arquivos/CMAC-2015-PROGRAMA.pdf>.

- FIRMINO, A. S.; TIMES, V. C.; SILVA, R. M. A.; MATEUS, G. R. Reactive grasp with path relinking for selecting olap views. In: *XLVIII SBPO (2016)*. [s.n.], 2016. Disponível em: <www.sbpo2016.iltc.br/pdf/156026.pdf>.

Especificar matematicamente a trajetória do guindaste e minimizar a trajetória do guindaste em vez do número de realocações. Demais, modelar o PRC como um problema de busca de caminho mínimo, e aplicar os algoritmos de busca em caminhos: *Dijkstra* e A*.

- FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. An exact approach for the container retrieval problem to reduce crane's trajectory. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2016. p. 933–938. ISBN 978-1-5090-1889-5. ISSN 2153-0009.

Constatar que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste. Ademais, computar o tempo de operação do guindaste por meio da trajetória do guindaste. Além disso, desenvolver modelo matemático de otimização para o PRC. Por fim, especificar e implementar algoritmos de otimização baseados nas meta-heurísticas RGRASP e PM.

- FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. A reactive grasp metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics*, p. 1–33, 9 2018. ISSN 1381-1231.

- FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. Optimising the crane's working time with the pilot method. Artigo em processo de revisão e submetido ao *International Journal of Production Research*. 2018.

1.7 ORGANIZAÇÃO DO TRABALHO

As seções restantes desta tese encontram-se estruturadas da seguinte forma.

2 Problema de Recuperação de Contêineres: descreve com maiores detalhes o problema de otimização abordado neste trabalho de pesquisa e introduz conceitos básicos relacionados ao problema que são utilizados para a elaboração deste trabalho.

3 Trabalhos Relacionados: apresenta os trabalhos realizados pela comunidade científica no desenvolvimento de métodos de otimização para solucionar o PRC.

4 Novos Métodos de Otimização para o Problema: explana sobre os métodos de otimização exatos e aproximados propostos nesta tese.

5 Análise Experimental: descreve e discute os resultados dos experimentos efetuados para análise e avaliação dos métodos de otimização desenvolvidos, e para análise investigativa sobre o custo-benefício entre o PRC irrestrito e o PRC restrito.

6 Conclusão: apresenta as considerações finais sobre os principais tópicos abordados nesta tese, incluindo as contribuições alcançadas e as indicações de trabalhos futuros.

2 PROBLEMA DE RECUPERAÇÃO DE CONTÊINERES

2.1 INTRODUÇÃO

Esta seção contém a descrição detalhada do problema de otimização abordado neste trabalho de pesquisa e discorre sobre conceitos básicos atrelados a este problema.

Esta seção está organizada como segue. A Seção 2.2 descreve o problema. A Seção 2.3 explana sobre os índices de decisão que são utilizados na construção de soluções para o problema. A Seção 2.4 apresenta as classes de problemas relativas ao problema. Já a demonstração da complexidade computacional do problema é apresentada na Seção 2.5. Por fim, algumas conclusões estão dispostas na Seção 2.6.

2.2 DESCRIÇÃO DO PROBLEMA

No terminal de contêineres, os contêineres são empilhados na baia, geralmente, na sequência de suas chegadas ao pátio de contêineres. No entanto, esta sequência de empilhamento pode conflitar com a sequência de recuperação dos contêineres da baia. A sequência de recuperação é definida pelos planos de cargas dos navios e determina a prioridade de saída de cada contêiner da baia. A situação torna-se mais conflitante, quando os primeiros contêineres postos na baia ocuparam, provavelmente, as posições mais inferiores nas pilhas, e precisam ser recuperados antes dos contêineres situados acima deles. Assim, um conflito surge quando um contêiner de maior prioridade de saída da baía não está situado no topo de uma pilha. Isto implica na transferência dos contêineres acima dele para outras pilhas, ou seja, implica nas realocações dos contêineres. Estes movimentos adicionais do guindaste promovem custos e consequentemente reduzem a produtividade na recuperação dos contêineres. O problema que visa potencializar esta produtividade por meio da minimização do tempo de operação do guindaste, é conhecido como Problema de Recuperação de Contêineres (PRC), ou *Container Retrieval Problem* (CRP).

O PRC consiste em encontrar uma sequência ótima de operações para o guindaste, permitindo que ele recupere sequencialmente todos os contêineres de uma baia de acordo com as suas prioridades de saída, onde a sequência ótima de operações é aquela com o tempo mínimo de operação despendido pelo guindaste. Sabendo que as dimensões de uma baia (i.e., $W \times H$) são dadas pelo número de pilhas (W) e pelo número de camadas (H), ou seja, a baia tem a altura máxima H e largura máxima W. Uma baia inicialmente possui N contêineres rotulados de 1 até N, conforme a prioridade de saída do contêiner, onde um valor menor significa uma prioridade mais alta. Além disso, cada espaço ocupável por um contêiner dentro da baia é indicado pela coordenada (i, j), onde $i \in \{1, \ldots, W\}$ e $j \in \{1, \ldots, H\}$ indicam, respectivamente, a pilha e a camada dentro da baia. Por último, a sequência de operações do guindaste é formada pelas operações de realocação e retirada.

Estas operações consomem tempo de operação do guindaste e estão sujeitas as seguintes restrições.

- Cada operação efetuada pelo guindaste produz uma configuração de baia e a próxima operação do guindaste deve ser realizada considerando essa nova configuração.
- A operação de retirada sobre o contêiner N define a última operação do guindaste.
- Um contêiner só pode ser movimentado pelo guindaste se ele estiver no topo de uma pilha.
- Um contêiner não só pode ser realocado para um espaço vazio, ou seja, deve ser colocado em cima de outro contêiner ou no piso da baia.
- Um contêiner não pode ser realocado de uma pilha i para uma mesma pilha i.
- A realocação deve respeitar a largura máxima da baia, então um contêiner não pode ser realocado para uma pilha $i > W$.
- A realocação deve respeitar a altura máxima da baia, então um contêiner não pode ser realocado para uma camada $j > H$.
- A operação de retirada tem como destino a coordenada $(\hat{W}, \hat{H})$, onde $\hat{W} = W + 1$ e $\hat{H} = H + 1$, e, assim, toda retirada de contêiner ocorre pelo canto superior direito da baia.
- A garra do guindaste move horizontalmente os contêineres na altura $\hat{H}$ por razões de segurança.

Em resumo, o PRC pode ser descrito por meio da seguinte questão: *Dado um conjunto de contêineres armazenados em uma baia, onde a cada contêiner é atribuído um valor correspondente à sua prioridade de saída da baia, qual o tempo mínimo de operação do guindaste para ele recuperar sequencialmente todos os contêineres da baia de acordo com as suas prioridades de saída?*. Como exemplo de resolução de uma instância do PRC, a Figura 3 ilustra uma solução para uma instância do PRC com 8 contêineres, 4 pilhas e altura máxima de 3 contêineres (i.e., $W = 4$, $H = 3$ e $N = 8$). Neste exemplo (Fig. 3), os contêineres são retirados pelo canto superior direito da baia e as seguintes 12 operações (4 realocações e 8 retiradas), enumeradas de **A** à **L**, são realizadas pelo guindaste:

A) O contêiner 8 é realocado da 1ª pilha para a 2ª pilha para ter acesso ao contêiner de maior prioridade da baia, o contêiner 1.

B) O contêiner 1 é retirado da 1ª pilha para fora da baia.

C) O contêiner 4 é realocado da 3ª pilha para a 2ª pilha para ter acesso ao contêiner de maior prioridade da baia, o contêiner 2.

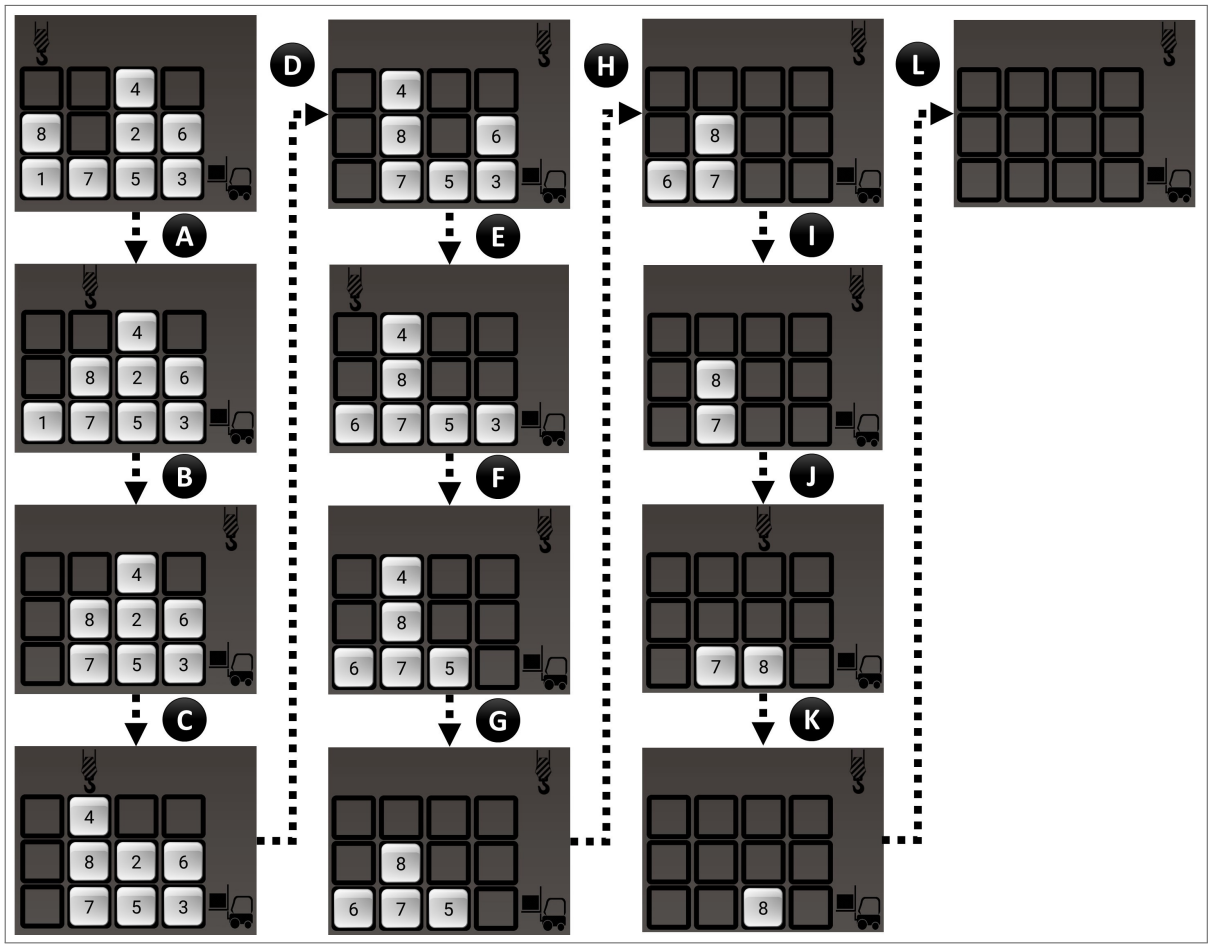


Figura 3 – Exemplo de solução para uma instância do PRC.

D) O contêiner 2 é retirado da 3ª pilha para fora da baia.

E) O contêiner 6 é realocado da 4ª pilha para a 1ª pilha para ter acesso ao contêiner de maior prioridade da baia, o contêiner 3.

F) O contêiner 3 é retirado da 4ª pilha para fora da baia.

G) O contêiner 4 é retirado da 2ª pilha para fora da baia.

H) O contêiner 5 é retirado da 3ª pilha para fora da baia.

I) O contêiner 6 é retirado da 1ª pilha para fora da baia.

J) O contêiner 8 é realocado da 2ª pilha para a 3ª pilha para ter acesso ao contêiner de maior prioridade da baia, o contêiner 7.

K) O contêiner 7 é retirado da 2ª pilha para fora da baia.

L) O contêiner 8 é retirado da 3ª pilha para fora da baia.

Vale ressaltar que, convencionalmente, o tempo de operação realizado pelo guindaste é inferido pelo número de operações de realocação realizadas pelo guindaste. Visto que,

empiricamente, tendo um menor número de realocações implica um menor tempo de operação do guindaste. Por isso, o PRC também é conhecido em outros trabalhos da literatura como o Problema de Realocação de Blocos, ou *Blocks Relocation Problem*. A resolução do PRC, por meio da redução do número de realocações, tornou-se tão tradicional na literatura, que alguns direcionadores foram propostos para auxiliar na construção das soluções para o PRC, de modo a produzir soluções com o menor número de realocações. Este direcionadores são descritos na seção a seguir.

2.3 ÍNDICES DE DECISÃO

Muitos dos algoritmos de otimização, propostos na literatura para solucionar o PRC, são regidos por uma regra ou direcionador que especifica quais realocações devem ser executadas na resolução do problema. Tais direcionadores e regras, encontrados na literatura, foram vistos por este trabalho de pesquisa como indicadores para a construção de soluções para o PRC. Estes indicadores são denominados, neste trabalho, por *Índices de Decisão* e são aplicáveis no seguinte contexto. Durante a resolução do PRC, quando um contêiner precisa ser realocado, é necessário selecionar entre as pilhas elegíveis qual será a pilha de destino dessa realocação. Nessa tomada de decisão, para cada pilha elegível (i.e., não atingiu a altura máxima de baia), o índice de decisão é calculado e seu valor é usado para selecionar a pilha de destino, e, assim, determinar a realocação que deve ser executada. Essa tomada de decisão ocorre repetidas vezes até que a baia esteja vazia.

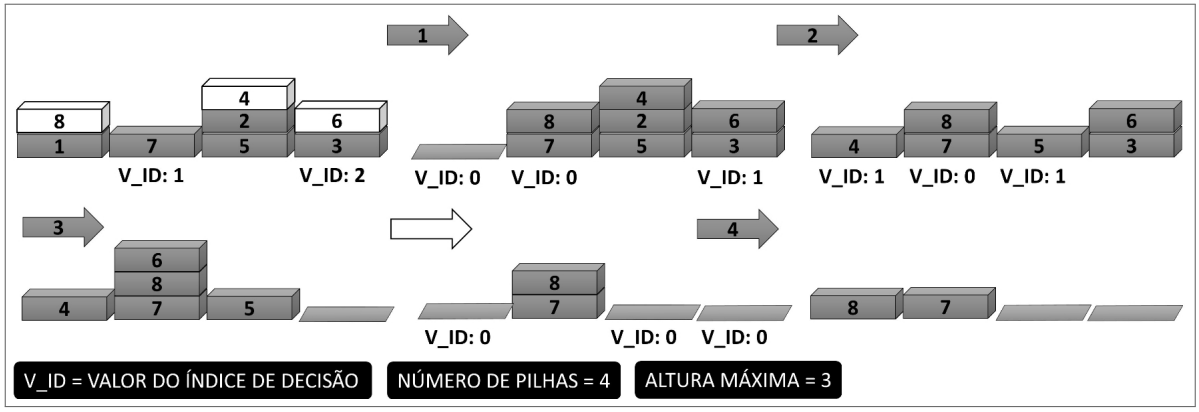


Figura 4 – Exemplo de uso do índice de decisão na resolução de uma instância do PRC.

A Figura 4 ilustra o uso de um índice de decisão na construção de uma solução para uma instância do PRC com 8 contêineres, 4 pilhas e altura máxima de 3 contêineres. Este índice de decisão computa o número de bloqueios produzido na pilha caso o contêiner seja realocado para ela. Um bloqueio ocorre quando um contêiner de prioridade mais baixa está acima de um contêiner de prioridade mais alta, em que um valor menor significa uma prioridade mais alta. Por exemplo, os contêineres com prioridade 8, 4 e 6, destacados na figura, produzem bloqueios, pois eles estão respectivamente sobre os contêineres 1, 2, 3

que possuem prioridade de saída respectivamente maior. Neste exemplo (Fig. 4), por meio do índice de decisão, quatro realocações, descritas a seguir, foram selecionadas.

1. O contêiner com prioridade oito, localizado no topo da primeira pilha, precisa ser realocado para possibilitar a retirada do contêiner com prioridade um. Para cada pilha elegível, o índice de decisão é calculado. A segunda pilha foi selecionada porque possui o menor valor de índice, o que significa que a segunda pilha produziria um bloqueio, a terceira pilha é inelegível (i.e., está com a altura máxima) e a quarta pilha produziria dois bloqueios.

2. O contêiner com prioridade quatro, localizado no topo da terceira pilha, precisa ser realocado para possibilitar a retirada do contêiner com prioridade dois. A primeira pilha foi selecionada, mas a segunda pilha também poderia ser selecionada porque ambas não produziriam bloqueios (i.e., $V\_ID : 0$), enquanto que a quarta pilha produziria um bloqueio.

3. O contêiner com prioridade seis, localizado no topo da quarta pilha, precisa ser realocado para possibilitar a retirada do contêiner com prioridade três. A segunda pilha foi selecionada porque não produziria bloqueio, enquanto a primeira pilha e a terceira pilha produziriam um bloqueio.

4. Após a remoção dos contêineres com prioridade 4, 5 e 6, o contêiner com prioridade oito, localizado no topo da segunda pilha, precisa ser realocado para possibilitar a retirada do contêiner com prioridade sete. A primeira pilha foi selecionada, mas a terceira e a quarta pilha também poderiam ser selecionadas porque ambas não produziriam bloqueios.

Os principais índices de decisão encontrados na literatura são descritos abaixo.

1. ***Lowest-Slot Index* (LSI)** (ZHANG, 2000): computa a altura da pilha. A realocação selecionada é aquela em que a pilha de destino tem a menor altura. Esse índice é baseado na premissa de que aumentar a altura da pilha promove um aumento no número de realocações futuras.

2. ***Reshuffle Index* (RI)** (MURTY et al., 2005): computa o número de bloqueios produzido na pilha de destino, caso o contêiner seja realocado para ela. Assim, é selecionada a realocação para pilha que produzirá o menor número de bloqueios. Neste índice, assume-se que muitos bloqueios produzem muitas realocações futuras. Vale mencionar que este índice de decisão é o mesmo utilizado no exemplo fornecido pela Figura 4.

3. ***Reshuffle with Look-Ahead Index* (RIL)** (WU; TING, 2010): é uma extensão do RI. A extensão consiste em incluir uma regra de desempate quando dois ou

mais valores do RI, computados para cada pilha elegível, forem iguais. Esta regra é denominada por *look-ahead* e funciona da seguinte maneira: para cada pilha de destino s, o valor da maior prioridade de saída presente na pilha (v_s) é computado. Em seguida, seleciona-se a operação de realocação cuja pilha de destino s tem o valor mais alto para v_s, i.e., $\{s \mid v_s = Max\,(v_1, \ldots, v_n)\}\,, \forall s = 1, \ldots, n$. O propósito desta regra é selecionar a realocação cuja pilha de destino possua o contêiner de menor prioridade (i.e., o valor mais alto para v_s). Desta forma, esta regra posterga realocação futura na pilha, pois ela detém o contêiner de menor prioridade cuja retirada será a mais tardia. Com isto, a realocação futura na pilha ocorrerá em um momento com menos contêineres na baia, de modo que uma realocação sem bloqueios seja mais facilmente encontrada.

4. ***Expected Number of Additional Relocations*** **(ENAR)** (KIM; HONG, 2006): computa o número esperado de realocações adicionais para uma configuração de baia b, i.e., $ENAR(b)$. O $ENAR(b)$ é computado pelo somatório de $e_s(k, n)$ para cada pilha s presente na configuração de baia b. $e_s(k, n)$ é o número esperado de realocações adicionais para a pilha s, considerando os seus k espaços disponíveis para empilhamento e o contêiner n no topo da pilha. Este número é obtido pela probabilidade de mover contêineres de outras pilhas cujo valor de prioridade é maior que n (i.e., contêineres que produzem bloqueios em n) para as k posições disponíveis da pilha. A finalidade deste índice é selecionar a realocação com a menor probabilidade de produzir realocações adicionais futuras, e é aplicado da seguinte maneira. Dada uma configuração inicial de baia b, para cada realocação elegível é feita uma simulação da configuração futura da baia b' resultante da operação de realocação. Em seguida, para cada configuração futura b' gerada, é obtida a contribuição para realocações adicionais, subtraindo $ENAR(b') - ENAR(b)$ e adicionando 1 se um bloqueio foi gerado pela realocação. Assim, seleciona-se a realocação com a menor contribuição.

5. ***Lowest Absolute Difference Index*** **(LADI)** (WU; TING, 2012): para cada pilha de destino s, o valor da prioridade mais alta presente na pilha (v_s) é computado, no caso da pilha s estar vazia, $v_s = N + 1$, tal que N é o número de contêineres na configuração inicial da baia. Em seguida, para cada v_s calcula-se a seguinte diferença $dif_s = v_s - c$, tal que c é o valor da prioridade do contêiner a ser realocado. Posteriormente, seleciona-se a realocação cuja pilha de destino s satisfaz as seguintes condições. Se existirem pilhas com $dif_s > 0$, a pilha com menor dif_s é escolhida. Caso contrário, todas as pilhas possuem $dif_s < 0$, então é escolhida a pilha com menor $\mid dif_s \mid$. O objetivo deste índice é preferencialmente selecionar a realocação que não produz bloqueios (i.e., $dif_s > 0$) e casos todas realocações produzam bloqueio optar pela realocação cuja pilha de destino possua o valor da prioridade mais alta (v_s) e

o mais próximo do valor de prioridade do contêiner a ser realocado, e então reduzir o número de realocações futuras.

6. ***Min-Max Index* (MNI)** (JOVANOVIC; VOSS, 2014): é uma melhoria para o LADI. Sua abordagem é substituir v_s por $v'_s = -N - v_s$, quando, na pilha de destino, a realocação produzirá um bloqueio (i.e., $v_s < c$) e a tornará cheia (altura máxima atingida). Lembrando que N é o número de contêineres inicial da baia e c é o valor da prioridade do contêiner a ser realocado. Esta substituição é para evitar realocações que produzam bloqueios e encham a pilha, de modo a não permite mais movimentações nesta pilha.

Vale mencionar que todos os índices de decisão listados acima foram projetados para minimizar o número de realocações na resolução do PRC, e criados em trabalhos que abordavam o PRC restrito. Todavia, as estratégias relacionadas a estes índices podem ser ampliadas ou combinadas para definir regras para aplicação no PRC irrestrito. Mais informações sobre essas e outras classes de problema do PRC são dadas na seção a seguir.

2.4 CLASSES DO PROBLEMA

O PRC consiste em encontrar uma sequência de operações para o guindaste visando o tempo mínimo de operação, permitindo que ele recupere sequencialmente todos os contêineres armazenados em uma baia de acordo com as suas prioridades de saída. Esta é a problemática central do PRC, mas este problema possui algumas especializações no que diz respeito a distribuição dos valores de prioridade e a forma de realocações permitas. Estas especializações correspondem às classes do problema.

Quanto às prioridades de saída, no PRC existem duas classes do problema: a de **prioridades repetidas** e a de **prioridades distintas**. Na primeira, dois ou mais contêineres têm a mesma prioridade de saída. Na última, mais comumente tratada na literatura, cada contêiner possui um valor único de prioridade saída. Como exemplo, a Figura 4 ilustra o PRC com prioridades distintas, e esta classe do problema é abordada neste trabalho. Vale ressaltar que o PRC de prioridades repetidas pode ser solucionado com o PRC de prioridades distintas. Em (TANAKA; TAKII, 2016), prioridades repetidas são tratadas pela atribuição de prioridades distintas entre contêineres de mesma prioridade.

O PRC também pode ser classificado em outras duas classes de acordo com a forma de realocação permitida no problema: **restrito** ou **irrestrito** (LEHNFELD; KNUST, 2014). No PRC restrito, a pilha de origem de cada operação de realocação é limitada à pilha na qual o contêiner de maior prioridade na baia está atualmente colocado, enquanto no PRC irrestrito não há essa restrição. A Figura 5 ilustra duas soluções para uma mesma instância do PRC com 8 contêineres, 3 pilhas e altura máxima de 4 contêineres (i.e., 4 camadas). A primeira solução é sobre o PRC restrito e outra solução, logo abaixo, sobre o PRC irrestrito. Na figura, o contêiner destacado em preto indica o contêiner de

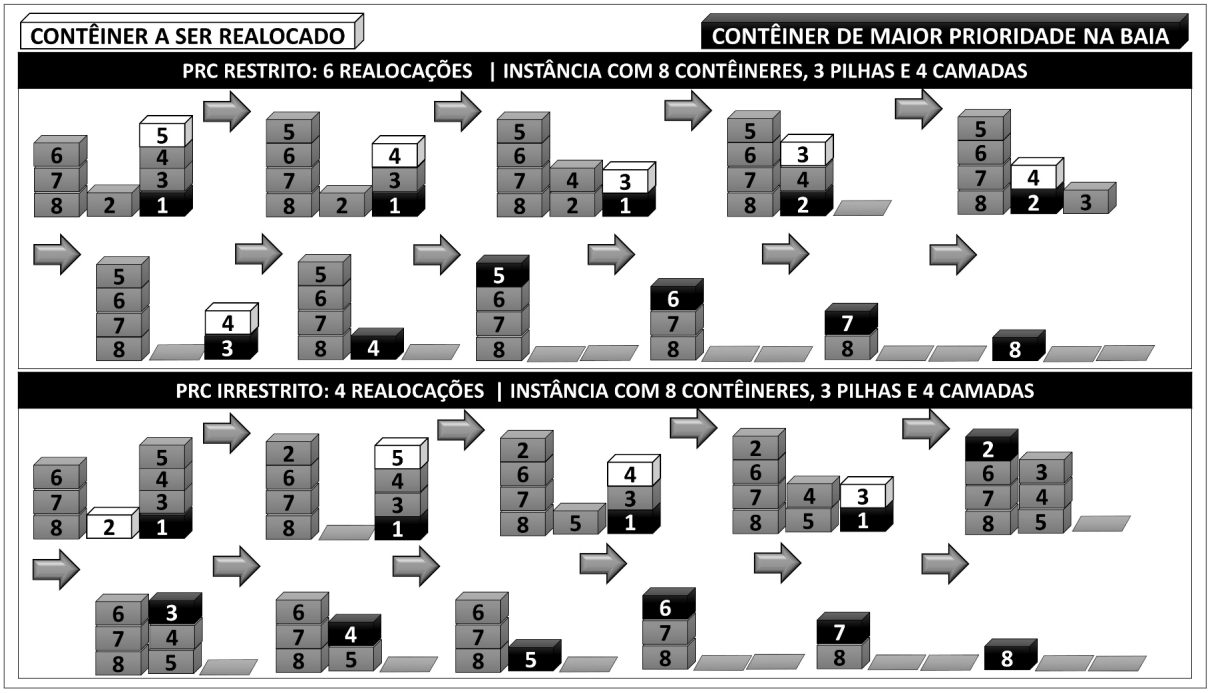


Figura 5 – Exemplo de PRC restrito e PRC irrestrito.

maior prioridade na baia e o contêiner destacado em branco indica o contêiner que será realocado. Neste exemplo (Fig. 5), durante o processo de recuperação dos contêineres, seis realocações foram realizadas na solução sobre o PRC restrito e quatro realocações foram realizadas na solução sobre o PRC irrestrito. Estas duas soluções são descritas a seguir.

- **Solução PRC Restrito**: O contêiner 5 foi realocado para ter acesso ao contêiner 1 (1ª realocação). O contêiner 4 foi realocado para ter acesso ao contêiner 1 (2ª realocação). O contêiner 3 foi realocado para ter acesso ao contêiner 1 (3ª realocação) e depois o contêiner 1 foi retirado. O contêiner 3 foi realocado para ter acesso ao contêiner 2 (4ª realocação). O contêiner 4 foi realocado para ter acesso ao contêiner 2 (5ª realocação) e após o contêiner 2 foi retirado. O contêiner 4 foi realocado para ter acesso ao contêiner 3 (6ª realocação) e, em seguida, o contêiner 3 foi retirado. Por fim, os contêineres 4, 5, 6, 7 e 8 são retirados nesta ordem.

- **Solução PRC Irrestrito**: O contêiner 2 foi realocado para cima do contêiner 6 (1ª realocação). O contêiner 5 foi realocado para ter acesso ao contêiner 1 (2ª realocação). O contêiner 4 foi realocado para ter acesso ao contêiner 1 (3ª realocação). O contêiner 3 foi realocado para ter acesso ao contêiner 1 (4ª realocação) e depois o contêiner 1 foi retirado. Por fim, os contêineres 2, 3, 4, 5, 6, 7 e 8 são retirados nesta ordem.

As duas soluções apresentadas no exemplo da Figura 5 são respectivamente as soluções ótimas para as duas classes do problema. Observando estas duas soluções para a problemática central do PRC, percebe-se que o PRC irrestrito gera mais oportunidades

de otimização do que o PRC restrito, uma vez que a solução ótima no PRC irrestrito foi composta de 4 realocações e a solução ótima no PRC restrito foi composta de 6 realocações. Por outras palavras, melhores soluções podem ser encontradas no PRC irrestrito, como reportado em (TRICOIRE; SCAGNETTI; BEHAM, 2018). Isso ocorre, principalmente, pela flexibilidade em realizar realocações prévias antes de realocações que produziriam bloqueios caso estas realocações prévias não fossem realizadas. Por exemplo, no início da solução para o PRC irrestrito (Fig. 5), contêiner 2 foi previamente realocado no intuito de evitar a formação de bloqueios proveniente da realocação dos contêineres 5, 4 e 3 para cima do contêiner 2.

Contudo, essa flexibilidade do PRC irrestrito têm o seu ônus. O espaço de busca no PRC irrestrito é maior que no PRC restrito, pois o número de realocações possíveis no PRC restrito é menor quando comparado ao PRC irrestrito. Por exemplo, no início das duas soluções presente na Fig. 5, o número de possíveis realocações no PRC restrito é 2 (i.e., mover o contêiner 5 para a 1ª ou 2ª pilha) já no PRC irrestrito este número é 4 (i.e., mover o contêiner 5 para a 1ª ou 2ª pilha; ou mover o contêiner 2 para a 1ª pilha; ou mover o contêiner 6 para a 2ª pilha). O espaço de busca maior implica em um custo computacional maior para encontrar uma solução. Portanto, este trabalho de pesquisa investiga a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução no PRC irrestrito em comparação à resolução no no PRC restrito.

2.5 COMPLEXIDADE DO PROBLEMA

A complexidade computacional de um problema estabelece uma classificação dos problemas segundo a “dificuldade” da sua resolução, ou seja, com base na complexidade dos algoritmos que os resolvam. Além disso, um problema computacional pode ser abordado como um problema de otimização ou como um problema de decisão. Nos problemas de otimização, suas soluções visam maximizar ou minimizar uma função objetivo. Já nos problemas de decisão, as suas soluções são “Sim” ou “Não”. Por exemplo, o "*problema do caminho mínimo*"(HART; NILSSON; RAPHAEL, 1968), numa abordagem de otimização, consiste em encontrar o caminho mais curto entre dois pontos em um grafo, mas já, numa abordagem de decisão, o problema consiste em descobrir se existe um caminho de comprimento menor que uma constante k.

O PRC é um problema de complexidade NP-Difícil, onde esta complexidade é comprovada, em (CASERTA; SCHWARZE; VOSS, 2012), por meio da redução do problema *Mutual Exclusion Scheduling for Permutation Graphs* (MESPG) para o PRC. Essa redução é feita por meio das versões de decisão do MESPG e do PRC, onde o problema MESPG na sua versão de decisão é comprovado ser NP-Completo (JANSEN, 1998). Ademais, essa redução aplica-se tanto para o PRC restrito quanto ao PRC irrestrito. Desta forma, como existe uma redução de um problema NP-Completo para a versão de decisão do PRC, então o

PRC é um problema NP-Difícil. Isto implica em uma maior dificuldade para encontrar, eficientemente, soluções ótimas para o problema, e a necessidade de elaborar algoritmos que lidem com a maior parte das instâncias do problema e encontrem soluções o mais próximo possível da solução ótima. Portanto, a demostração da redução do problema MESPG para o PRC é apresentada a seguir.

2.5.1 Prova da Complexidade do Problema

Considere n tarefas que precisam ser executadas em um recurso. Este recurso só consegue executar um conjunto de no máximo m tarefas por vez, e a quantidade de vezes que o recurso é utilizado para executar um conjunto de tarefas é contabilizado por t. Além disso, entre as tarefas existem dependências de modo que uma tarefa não pode ser executada antes do término de execução de outra tarefa. Com isto, surge o seguinte problema de decisão: *existe uma solução onde todas as* $\boldsymbol{n}$ *tarefas sejam executadas, o recurso seja utilizado* $\boldsymbol{t}$ *vezes e, em cada vez, só pode ser executado um conjunto com no máximo* $\boldsymbol{m}$ *tarefas independentes?* Este problema decisão é o *Mutual Exclusion Scheduling for Permutation Graphs* (MESPG).

O PRC, tradicionalmente, como problema de otimização, visa minimizar o número de realocações efetuadas. Como um problema de decisão, o PRC é dado pela seguinte questão: *dada um baia com um número finito de* $\boldsymbol{W}$ *pilhas,* $\boldsymbol{H}$ *camadas e* $\boldsymbol{N}$ *contêineres, existe uma solução que requeira exatamente* $\boldsymbol{n}$ *realocações para recuperar sequencialmente todos os contêineres da baia de acordo com as suas prioridades de saída?* Conhecidas as versões de decisão dos dois problemas (i.e., PRC e MESPG), o próximo passo é realizar a redução do problema de decisão MESPG para o problema de decisão PRC. Para isso, é preciso transformar o PRC no problema MESPG e utilizar a solução produzida no PRC para solucionar o MESPG.

Neste propósito, considere uma baia com a seguinte configuração: $W = t{+}1$, $H = n{+}1$, $N = (W \times H) - (t \times m)$ e a distribuição dos contêineres definida pela Tabela 1. A Figura 6 exibe três configurações de uma baia com $W = 4$ e $H = 4$: (A) com $N = 4$, (B) com $N = 7$ e (C) com $N = 10$. Estes três exemplos ilustram configurações de baia que atendem os requisitos descritos na Tabela 1. Observe que, por meio da Tabela 1, instâncias do PRC são definidas a partir de instâncias do MESPG, pois o número de pilhas W o número de camadas H e o número de contêineres N são atribuídos de acordo com três parâmetros de entrada do MESPG (i.e., n, t, m). Além disso, seja o Lema 1 a afirmativa inerente a resolução do MESPG por meio da resolução do PRC, e, portanto, a prova deste afirmativa comprova a redução do MESPG para o PRC.

Lema 1. *Uma solução para o PRC com exatamente* $\boldsymbol{n}$ *realocações existe se e somente se* $\boldsymbol{n}$ *tarefas puderem ser executadas por um recurso, utilizado* $t = W - 1$ *vezes e, em cada vez, é executado um conjunto com no máximo* $\boldsymbol{m}$ *tarefas independentes.*

Prova [Lema 1]. Note que todas as combinações possíveis de configuração de baia, em que $W = t+1$, $H = n+1$, $N = (W \times H) - (t \times m)$ e atenda a distribuição dos valores de prioridade especificada na Tabela 1, terá pelo menos $H - 1 = n$ realocações necessárias para remover o contêiner 1. Além disso, os n contêineres acima do contêiner 1 assemelham-se às n tarefas no MESPG, cujas relações de dependência estão atreladas aos valores de prioridade. Assim, nestes n contêineres, existe a seguinte relação de dependência: um contêiner não pode ser realocado para a mesma pilha na qual outro contêiner que estava previamente acima dele foi realocado. Uma vez que infringir esta relação de dependência implica na produção de realocações adicionais. Por exemplo, nos exemplos da Figura 6, se o contêiner 2 for realocado para a mesma pilha em que os contêineres 3 ou 4 forem postos, isso produzirá um bloqueio e, assim, mais uma realocação será necessária para solucionar o problema.

Portanto, encontrar a atribuição dos n contêineres (postos acima do contêiner 1) para as $t = W - 1$ pilhas com no máximo m atribuições por pilha e sem produzir bloqueios, de modo a obter uma solução do PRC com apenas n realocações, é equivalente ao problema de executar n tarefas em um recurso que é utilizado $t = W - 1$ vezes e, em cada vez, é executado um conjunto com no máximo m tarefas independentes. □

Tabela 1 – Distribuição dos valores de prioridade dos contêineres na baia.

PILHA	**CAMADA**	**PRIORIDADE**
$i = 1$	$j = 1$	1
$i = 1$	$j = 2, \ldots, H$	j
$i = 2, \ldots, W$	$j = 1, \ldots, H - m$	$N + 1 - (j + (i - 2)(H - m))$
$i = 2, \ldots, W$	$j = H - m + 1$	sem contêiner

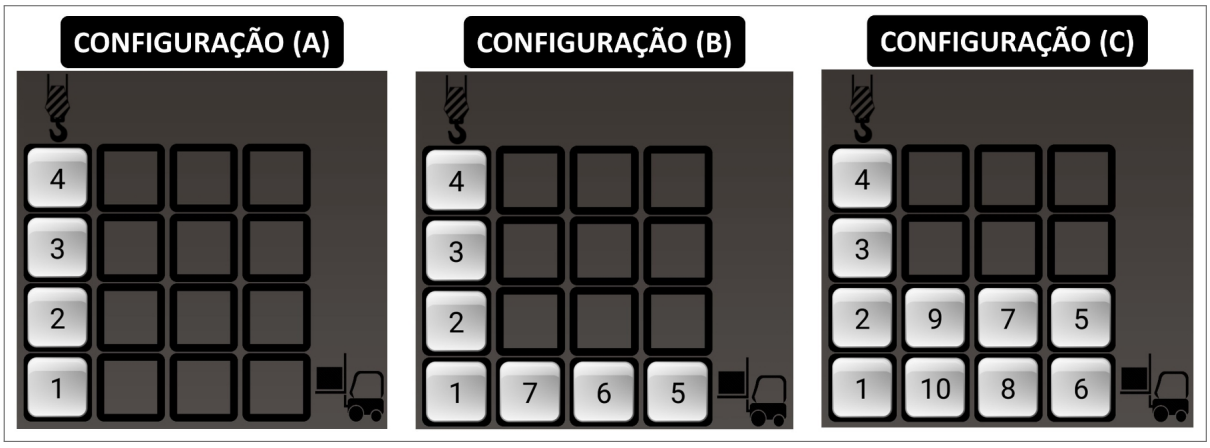


Figura 6 – Três exemplos de configuração de baia usados na prova de complexidade.

2.6 CONSIDERAÇÕES FINAIS

Esta seção descreveu em pormenor o PRC, problema de otimização abordado neste trabalho. Foram apresentados os índices de decisão que são indicadores para auxiliar na construção de soluções para o PRC, e estes serão avaliados e empregados em métodos de otimização propostos neste trabalho de pesquisa. Em seguida, definiram-se as principais classes de problema relativa ao PRC, e apresentou-se a carência de uma avaliação custo-benefício entre o PRC restrito e PRC irrestrito. Por fim, foi demostrado a complexidade computacional do PRC como um problema NP-Difícil, atestando que o PRC está entre os problemas computacionais mais difíceis.

Na Seção 3, apresentam-se os trabalhos relacionados e identificados no estado da arte na área de desenvolvimento de métodos de otimização para solucionar o PRC.

3 TRABALHOS RELACIONADOS

3.1 INTRODUÇÃO

Diversos trabalhos têm sido realizados como objetivo de desenvolver métodos de otimização para solucionar o Problema de Recuperação de Contêineres (PRC). Estes métodos de otimização, segundo a abordagem utilizada, visam encontrar soluções ótimas (i.e., abordagem exata) ou soluções próximas da solução ótima (i.e., abordagem aproximada). Ademais, alguns destes trabalhos abordam variações do PRC no que diz respeito às restrições do problema e à função objetivo. Sabendo que miniminizar o número de realocações é a principal função objetivo abordada pela literatura. Desta forma, para uma análise sistêmica das abordagens encontradas na literatura, esta seção agrupa os trabalhos para o PRC de acordo com abordagem utilizada. Portanto, a Seção 3.2 discorre sobre os trabalhos de abordagem exata, a Seção 3.3 explana sobre os trabalhos de abordagem aproximada, e a Seção 3.4 relata os trabalhos que tratam variações do problema. Por fim, uma análise comparativa entre os trabalhos investigados e as considerações finais são dispostas na Seção 3.5.

3.2 ABORDAGENS EXATAS

As abordagens exatas garantem a solução ótima para o problema. No entanto, na maioria dos casos, elevado tempo computacional pode ser requerido por tais abordagens para encontrar a solução ótima. Nesse aspecto, pioneiramente, Kim e Hong (2006) propôs um algoritmo *Branch-and-Bound* (BB) para solucionar o PRC restrito. Posteriormente, em (ÜNLÜYURT; AYDIN, 2012), também foi proposto um algoritmo BB para o PRC restrito. Este algoritmo BB alcançou melhores resultados que o algoritmo BB em (KIM; HONG, 2006). Mais estudos foram propostos a fim de encontrar um algoritmo BB capaz de reportar as soluções ótimas em menor tempo computacional. Recentemente, Expósito-Izquierdo, Melián-Batista e Moreno-Vega (2015) e Tanaka e Takii (2016) propuseram novos algoritmos BB. Em (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015), uma nova proposta de algoritmo BB reportou um baixo tempo computacional através da inclusão de uma estratégia inteligente para explorar os nós mais promissores no grafo de busca usada no BB. Em (TANAKA; TAKII, 2016), para o PRC restrito, um eficiente algoritmo BB foi proposto por meio da formulação de um novo limite inferior para busca de soluções no BB. Também, visando reduzir o tempo computacional na busca de soluções ótimas, Ku e Arthanari (2016) desenvolveu um algoritmo que reduz o espaço de busca, permitindo, assim, solucionar otimamente instâncias do PRC de tamanho pequeno a médio.

Técnicas de otimização de busca em caminhos também foram aplicadas ao PRC para

encontrar soluções ótimas para o problema. Primeiramente, Zhang et al. (2010) estudou como algoritmo de busca *A Star* (A*) pode ser aplicado para resolver PRC restrito. Depois disso, Zhu et al. (2012) propôs dois algoritmos de busca *Iterative Deepening A** (IDA) um para resolver o PRC restrito e outro para resolver o PRC irrestrito. O algoritmo IDA para solucionar PRC restrito conseguiu resolver otimamente mais instâncias que o algoritmo em (ZHANG et al., 2010). A razão para este número maior de instâncias solucionadas, se deve, principalmente, pela definição de um novo limite inferior para a busca de soluções. Vale mencionar que também foram propostas versões dos dois algoritmos IDA com um limitador de tempo computacional, no intuito de solucionar mais instâncias do problema em um tempo computacional viável. Por último, Expósito-Izquierdo, Melián-Batista e Moreno-Vega (2014) propuseram novos algoritmos de busca A* para resolver o PRC restrito e o PRC irrestrito. Os autores utilizaram estes novos algoritmos com o objetivo de encontrar as soluções ótimas para serem aplicadas em seus estudos comparativos com outros algoritmos, incluindo o próprio algoritmo de abordagem aproximada também proposto nesse trabalho.

Em abordagens exatas, existem trabalhos baseados em modelagens matemáticas que também asseguram a solução ótima, embora o tamanho das instâncias que tais abordagens podem processar, em geral, é menor em comparação com a outras abordagens exatas. Inicialmente em (WAN; LIU; TSAI, 2009), uma modelagem de Programação Linear Inteira (PLI) para o PRC restrito foi proposto. Depois, em (CASERTA; SCHWARZE; VOSS, 2012) foram criadas duas modelagens de PLI uma para o PRC restrito e outra para o PRC irrestrito. A primeira resolve otimamente pequenas instâncias do problema e a segunda não foi avaliada pelas seguintes razões: (1) possui um grande espaço de busca, tornando inviável o tempo computacional para encontrar a solução; (2) esta modelagem precisa de um constante T para limitar o número máximo de realocações antes da retirada do contêiner de maior prioridade, contudo o uso dessa constante é impraticável, pois quanto maior o valor de T maior é o espaço de busca, e um valor menor para T pode gerar soluções distantes da solução ótima. Mais à frente, a modelagem matemática de PLI para o PRC irrestrito, em (CASERTA; SCHWARZE; VOSS, 2012), foi melhorada em (PETERING; HUSSEIN, 2013). Este modelo melhorado utiliza menos variáveis de decisão e resultou em um melhor desempenho quando comparado a (CASERTA; SCHWARZE; VOSS, 2012). Todavia, os resultados dos experimentos indicaram que a abordagem proposta por modelagem matemática para o PRC irrestrito mostrou-se insuficiente para instâncias do mundo real, e, assim, corroborou com as afirmações ditas em (CASERTA; SCHWARZE; VOSS, 2012).

Adiante, mais formulações matemáticas foram propostas, por exemplo, os trabalhos em (AZARI, 2015; EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015; ZEHENDNER et al., 2015; GALLE; BARNHART; JAILLET, 2018). Em particular, os trabalhos em (AZARI, 2015; EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015; ZEHENDNER et al., 2015) corrigem e melhoram o modelo matemático de PLI para o PRC restrito

que é retratado em (CASERTA; SCHWARZE; VOSS, 2012). Por fim, Galle, Barnhart e Jaillet (2018) formula um novo modelo PLI com melhores resultados que os três anteriores.

3.3 ABORDAGENS APROXIMADAS

Obter soluções próximas do ótimo em curto tempo computacional tem sido a meta de vários trabalhos até então. Estes trabalhos estão associados a uma abordagem aproximada (ou heurística), uma vez que boas soluções podem ser encontradas, mas não há garantia de obter soluções ótimas. A abordagem aproximada é uma alternativa às dificuldades encontradas para desenvolver métodos de abordagem exata com alto desempenho, porquanto tais métodos estão atados a uma elevada carga computacional. Por exemplo, em (KIM; HONG, 2006), devido ao alto tempo de processamento do algoritmo BB, uma heurística baseada no índice de decisão ENAR foi desenvolvida. Similarmente, em (WAN; LIU; TSAI, 2009), o elevado tempo computacional requerido pela abordagem com modelos matemáticos de PLI levou à construção de duas heurísticas gulosas: uma usando modelos reduzidos de PLI e a outra utilizando o índice de decisão RI. O mesmo se aplica aos trabalhos em (CASERTA; SCHWARZE; VOSS, 2012; PETERING; HUSSEIN, 2013; OLSEN; GROSS, 2014; EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2014; TRICOIRE; SCAGNETTI; BEHAM, 2018), onde heurísticas baseadas em um conjunto de regras de realocação foram desenvolvidas, em contrapartida aos métodos de abordagem exata. Visto que estes não são adequados para solucionar instancias maiores devido ao seu alto custo computacional. Em particular, Tricoire, Scagnetti e Beham (2018) se concentraram apenas no PRC irrestrito e propuseram um algoritmo BB e, alternativamente, propuseram uma heurística baseada na meta-heurística *Pilot Method* (PM).

Métodos de otimização baseados em pesquisa em grafos de busca também tem sido propostos para o PRC, como visto em (WU; TING, 2010; FORSTER; BORTFELDT, 2012; JIN; ZHU; LIM, 2015; TING; WU, 2017). Inicialmente, em (WU; TING, 2010), três algoritmos *Beam Search* (BS) para o PRC restrito são propostos, cada algoritmo usa um índice de decisão entre {LSI, RI e RIL}. Os índices de decisão são utilizados pelos algoritmos BS para selecionar os nós do grafo de busca. Posteriormente, em (TING; WU, 2017), também foi proposto, para o CRP restrito, um novo algoritmo BS incorporado com uma heurística para avaliar os nós no grafo de busca. Forster e Bortfeldt (2012) propuseram um algoritmo heurístico baseado em grafo de busca para o PRC irrestrito. Mais tarde, Jin, Zhu e Lim (2015) também propuseram um algoritmo heurístico baseado em grafo de busca, cujos resultados superaram o algoritmo em (FORSTER; BORTFELDT, 2012).

Algoritmos baseados em heurísticas inspiradas em comportamentos da natureza também têm sido desenvolvidos. Carraro e Castro (2011) investigaram um algoritmo de seleção clonal para o PRC restrito. Entretanto, experimentos adicionais que considerem mais instâncias (principalmente média e grandes instâncias), são necessários para avaliar a eficácia desse algoritmo. Em (CASERTA; VOSS; SNIEDOVICH, 2011), foi proposto um algoritmo heu-

rístico fundamentado na meta-heurística *Corridor Method* (CM). Este algoritmo utiliza uma técnica para reduzir o espaço de busca, e ela é baseada em restrições (denominadas *exogenous constraints*) sobre as realocações elegíveis.

Ao contrário das outras heurísticas para o PRC restrito que selecionam a próxima realocação considerando apenas o contêiner mais acima (i.e., do topo da pilha) do contêiner de maior prioridade da baia ($\hat{c}$), em (WU; TING, 2012) é proposta uma heurística que seleciona a próxima realocação, considerando o contêiner do topo e os demais contêineres acima $\hat{c}$. Assim a próxima realocação é determinada considerando os possíveis destinos das próximas realocações até a retirada de $\hat{c}$. Além disso, a heurística utiliza o LADI como seu índice de decisão. Esta nova heurística apresentou melhor resultado comparado a uma heurística que considera apenas o contêiner do topo, entretanto, esta nova heurística reportou maior custo computacional. Na mesma direção de considerar mais de um contêiner para selecionar a próxima realocação, (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2014) produziu um algoritmo heurístico para o PRC irrestrito baseado em um conjunto te regras denominado *conhecimento de domínio específico*. Este algoritmo heurístico foi proposto, em contrapartida ao algoritmo A* também proposto em (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2014), devido ao alto custo computacional do A* para resolver instâncias de médio e grande porte.

Por fim, existem trabalhos que propuseram algoritmos heurísticos, mas estes não são baseados em meta-heurísticas, como os testemunhado em (CASERTA; SCHWARZE; VOSS, 2012; OLSEN; GROSS, 2014; JOVANOVIC; VOSS, 2014). Uma heurística simples baseada em conjunto de regras de realocação foi proposta em (CASERTA; SCHWARZE; VOSS, 2012), e utilizada para solucionar instancias maiores, visto que a abordagem com modelos matemáticos de PLI proposta resolvia viavelmente apenas instâncias menores. Olsen e Gross (2014) apresentaram uma heurística para o PRC restrito similar ao de (CASERTA; SCHWARZE; VOSS, 2012) e por meio de análise probabilística formal demostrou que para determinados tipos de instâncias, com esta heurística, é possível obter soluções ótimas. Em (JOVANOVIC; VOSS, 2014), foi definido o *Min-Max Index* (MNI) e foi proposta uma nova heurística chamada de *Chain Heuristic*, que utiliza o MNI na seleção de futuras realocações. A particularidade desta heurística é considerar os efeitos futuros associados à realocação a ser executada. A decisão sobre qual realocação realizar é embasada nos impactos de uma cadeia de realocações futuras. A heurística *Chain* foi comparada com as heurísticas de (WAN; LIU; TSAI, 2009; WU; TING, 2010; CASERTA; SCHWARZE; VOSS, 2012), e obteve os melhores resultados em praticamente todos as instâncias avaliadas, principalmente, para as instâncias de maior porte.

3.4 VARIAÇÕES DO PRC

Enquanto a maioria dos estudos sobre o PRC lida com o número de realocações, existem alguns outros estudos que lidam com outros objetivos de otimização. Ünlüyurt e Aydin

(2012) estendeu a heurística gulosa em (KIM; HONG, 2006) para minimizar a distância total percorrida horizontalmente pelo guindaste. Uma redução no consumo de combustível é tratada em (HUSSEIN; PETERING, 2012). Cada contêiner possui um peso (em toneladas), e a função objetivo é minimizar a energia total consumida pelo guindaste para manipular os contêineres, considerando os seus respectivos pesos. A fim de resolver esta variação do PRC, um algoritmo genético foi proposto. Contudo, Hussein e Petering (2012) reconhecem que mais experimentos com diferentes tamanhos de instâncias são necessários para avaliar a eficácia do algoritmo.

Existem também outras variações do PRC que não estão atreladas às alterações na função objetivo do problema. Por exemplo, alguns estudos assumem que novos contêineres podem entrar na baia enquanto o processo de recuperação do contêiner ocorre. Essa variante dinâmica do PRC é encontrada em (AKYÜZ; LEE, 2014; CASEY; KOZAN, 2012). Outro exemplo, em (SILVA et al., 2018), mais uma variação do PRC foi abordada. Nesta variante, a peculiaridade está em ter apenas dois grupos de prioridade de contêineres: o grupo-alvo a ser recuperado primeiro e o outro grupo formado por todos os contêineres restantes.

Ainda sobre variações do PRC em termos de alterações na função objetivo, existem trabalhos, tais como (LI; YU, 2010; LEE; LEE, 2010; FORSTER; BORTFELDT, 2012; LIN; LEE; LEE, 2015; KIM; KIM; LEE, 2016), que propõe métodos de otimização que objetivam minimizar o número de operações de contêineres, bem como, o tempo de operação do guindaste. Em particular, os trabalhos em (LI; YU, 2010; LIN; LEE; LEE, 2015) reduzem o número de realocações mais utilizam uma função de penalidade para minimizar o tempo total de operação do guindaste. (FORSTER; BORTFELDT, 2012) propuseram uma heurística baseada em grafo de busca e apresentou resultados melhores do que a heurística em (LEE; LEE, 2010). Por fim, Kim, Kim e Lee (2016) propuseram uma nova heurística com base nas circunstâncias presentes em uma baia de contêiner. A heurística classifica as circunstâncias em quatro casos e usa esses casos para selecionar as operações da solução. Esta heurística foi comparada com a heurística em (LEE; LEE, 2010; LIN; LEE; LEE, 2015) e mostrou bons resultados na redução do número de realocações e no tempo de operação do guindaste.

Mesmo embora esses estudos tenham o mesmo propósito de minimizar o tempo de operação do guindaste, eles diferem na maneira como o tempo de operação é calculado. Assim, às vezes os tempos de operação do guindaste não são diretamente comparáveis, como mencionado em (LIN; LEE; LEE, 2015). Por exemplo, Galle, Barnhart e Jaillet (2018) propuseram uma modelagem matemática de PLI para minimizar o tempo de operação do guindaste. No entanto, este trabalho considera apenas o tempo de deslocamento horizontal do guindaste, ao contrário deste e de outros estudos, uma vez que o deslocamento vertical dos contêineres não é considerado nos cálculos. Além disso, a distância de deslocamento necessária para alinhar a garra do guindaste antes de executar a operação de realocação também não é considerada. Também vale ressaltar que, embora esses estudos abordem o

tempo de operação do guindaste, o tempo de operação é minimizado pela minimização do número de operações de contêineres, mas o número de realocações e o tempo de operação não se refletem necessariamente, como relatado em (LIN; LEE; LEE, 2015).

3.5 CONSIDERAÇÕES FINAIS

Esta seção apresentou os principais trabalhos realizados pela comunidade cientifica na área de desenvolvimento de métodos de otimização para solucionar o PRC. A partir destes trabalhos investigados, apresenta-se a seguir uma análise sobre as principais características de cada trabalho e discute-se a particularidade desta tese dentre os demais trabalhos. Para auxiliar esta análise, a Tabela 2 lista os trabalhos e apresenta os seguintes critérios de comparação entre os trabalhos: (1) os métodos de otimização exatos propostos; (2) os métodos de otimização aproximados propostos; (3) as classes do PRC abordadas (i.e., restrito e irrestrito); (4) as métricas de otimização consideradas; (5) se a problemática é uma variação do PRC.

Como mostra a Tabela 2, um grupo menor de trabalhos aborda variações no PRC e este grupo ainda torna-se menor quando a variação no PRC não ocorre por causa de alterações na métrica de otimização. A respeito da métrica de otimização, há uma predominância nos trabalhos em minimizar o número de realocações. Poucos são os trabalhos que visam minimizar diretamente algum custo de operação, tal como, o consumo de combustível, o tempo de operação ou alguma métrica que mensure melhor o tempo de operação do guindaste em vez do número de realocações. Ademais, entre estes trabalhos, que objetivam minimizar diretamente o tempo de operação, há divergências na computação do tempo de operação do guindaste. Isto se deve pela ausência de definição formal e explicita de uma métrica para representar todo o percurso realizado pelo guindaste, e assim por meio dessa métrica computar de maneira direta e unívoca o tempo de operação do guindaste.

Já quanto aos métodos de otimização propostos para o PRC, os métodos aproximados são dominantes e são poucos os trabalhos que propõe métodos exatos e aproximados. Assim, a maior parte dos trabalhos foca em apenas uma abordagem: exata ou aproximada. Por fim, pela Tabela 2, pode-se observar que a maioria dos trabalhos aborda a classe PRC restrito, e os poucos trabalhos, que abordam as duas classes, não realizaram uma análise sobre a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito. Em vista disso, este trabalho de pesquisa apresenta algumas particularidades quando comparado com trabalhos anteriores encontrados na literatura. Especificamente, uma variação do PRC é abordada. Nesta variação, o objetivo é minimizar tempo de operação do guindaste e computá-lo de maneira direta e unívoca a partir da trajetória do guindaste, uma nova métrica para mensurar os percursos do guindaste. Assim, evitar a inferência de um menor tempo de operação do guindaste a partir do número de realocações, como na maioria dos estudos. Neste trabalho de pes-

quisa, para esta variante do PRC, novos métodos de otimização exatos e aproximados são desenvolvidos para as duas classes do PRC, e uma análise investigativa sobre o custo-benefício dentre estas duas classes do PRC também é realizada. Portanto, as principais contribuições deste trabalho (i.e., a definição de novas métricas e de novos métodos de otimização) são descritas na Seção 4.

Tabela 2 – Análise comparativa dos Trabalhos Correlatos.

TRABALHO	MÉT. OTIMIZAÇÃO EXATOS	MÉT. OTIMIZAÇÃO APROXIMADOS	CLASSES PRC[a]	MÉTRICAS OTIMIZAÇÃO[b]	VARIAÇÃO PRC
(KIM; HONG, 2006)	ALGORITMO BB	HEURÍSTICA	R	NR	NÃO
(WAN; LIU; TSAI, 2009)	MODELAGEM PLI	HEURÍSTICA	R	NR	NÃO
(LEE; LEE, 2010)		HEURÍSTICA	R	NR E TO	SIM
(LI; YU, 2010)		HEURÍSTICA	R	NR E TO	SIM
(WU; TING, 2010)		ALGORITMO BS	R	NR	NÃO
(ZHANG et al., 2010)	ALGORITMO A*		R	NR	NÃO
(CARRARO; CASTRO, 2011)		ALG. GENÉTICO	R	NR	NÃO
(CASERTA; VOSS; SNIEDOVICH, 2011)		META-HEURÍSTICA CM	R	NR	NÃO
(CASERTA; SCHWARZE; VOSS, 2012)	MODELAGEM PLI	HEURÍSTICA	R e IR	NR	NÃO
(CASEY; KOZAN, 2012)		META-HEURÍSTICA TABU-SEARCH	R	NR	SIM
(FORSTER; BORTFELDT, 2012)		HEURÍSTICA	R e IR	NR E TO	SIM
(HUSSEIN; PETERING, 2012)		ALG. GENÉTICO	R	CC	SIM
(WU; TING, 2012)		HEURÍSTICA	R	NR	NÃO
(ÜNLÜYURT; AYDIN, 2012)	ALGORITMO BB	HEURÍSTICA	R	NR E DH	NÃO
(ZHU et al., 2012)	ALGORITMO IDA	IDA C/ LIMITADOR	R e IR	NR	NÃO
(PETERING; HUSSEIN, 2013)	MODELAGEM PLI	HEURÍSTICA	R e IR	NR	NÃO
(AKYÜZ; LEE, 2014)	MODELAGEM PLI	HEURÍSTICA	R	NR	SIM
(JOVANOVIC; VOSS, 2014)		HEURÍSTICA	R	NR	NÃO
(OLSEN; GROSS, 2014)		HEURÍSTICA	R	NR	NÃO
(EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2014)	ALGORITMO A*	HEURÍSTICA	R e IR	NR	NÃO
(AZARI, 2015)	MODELAGEM PLI		R	NR	NÃO
(EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015)	ALGORITMO BB e MODELAGEM PLI		R	NR	NÃO
(JIN; ZHU; LIM, 2015)		HEURÍSTICA	IR	NR	NÃO
(LIN; LEE; LEE, 2015)		HEURÍSTICA	IR	NR E TO	SIM
(ZEHENDNER et al., 2015)	MODELAGEM PLI		R	NR	NÃO
(KIM; KIM; LEE, 2016)		HEURÍSTICA	IR	NR E TO	SIM
(KU; ARTHANARI, 2016)	ALG. DE BUSCA		R	NR	NÃO
(TANAKA; TAKII, 2016)	ALGORITMO BB		R	NR	NÃO
(GALLE; BARNHART; JAILLET, 2018)	MODELAGEM PLI		R	NR E TO	SIM
(TING; WU, 2017)		ALGORITMO BS	R	NR	NÃO
(SILVA et al., 2018)	ALGORITMO BB	ALGORITMO BS	IR	NR	SIM
(TRICOIRE; SCAGNETTI; BEHAM, 2018)	ALGORITMO BB	META-HEURÍSTICA PM	IR	NR	NÃO

[a] **R**: RESTRITO; IR: IRRESTRITO. [b] **NR**: NÚMERO DE REALOCAÇÕES; **TO**: TEMPO DE OPERAÇÃO; **CC**: CONSUMO DE COMBUSTÍVEL; **DH**: DISTÂNCIA PERCORRIDA HORIZONTALMENTE.

4 NOVOS MÉTODOS DE OTIMIZAÇÃO PARA O PROBLEMA

4.1 INTRODUÇÃO

Problemas de otimização surgem em diversas áreas do conhecimento. Eles emergem quando a tarefa é encontrar a melhor solução dentre as inúmeras soluções para um dado problema, onde uma solução é dita melhor que outra por meio de uma métrica de otimização que determina o valor ou qualidade de uma solução. Além disto, a métrica de otimização é usada para computar a função objetivo do problema, e está função, a depender do problema, deve ser minimizada ou maximizada para se encontrar a melhor solução do problema de otimização. A resolução, em tempo computacional viável, de problemas de otimização pertencentes a classe de complexidade NP-Difícil, tais como o Problema de Recuperação de Contêineres (PRC), é um dos grandes desafios computacionais. Diante disso, pesquisadores têm concentrado esforços em usar técnicas de otimização, cada vez mais elaboradas, no intuito de encontrar soluções ótimas (i.e., métodos de otimização exatos) ou soluções próximas da solução ótima (i.e., métodos de otimização aproximados) em problemas dessa magnitude dentro de um tempo computacional factível.

Portanto, esta seção discorre sobre os métodos de otimização aplicados à variante do PRC tratada neste trabalho, cujo objetivo é minimizar o tempo de operação executado pelo guindaste, onde o tempo de operação é computado a partir da trajetória do guindaste. Esta seção está organizado da seguinte forma. Inicialmente, na Seção 4.2, são definidas as novas métricas de otimização para computar diretamente o tempo de operação do guindaste, pois, como formalmente é evidenciado nessa seção, minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste. Em seguida, a Seção 4.3 discorre sobre os métodos de otimização exatos desenvolvidos, e ela inclui a elaboração de um modelo matemático para o problema. Depois, a Seção 4.4 explana sobre os métodos de otimização aproximados que são propostos. Por fim, a Seção 4.5, apresenta algumas considerações finais sobre os métodos de otimização construídos neste trabalho de pesquisa.

4.2 MÉTRICAS E OBJETIVOS DE OTIMIZAÇÃO

Devido à pressão competitiva nos portos, é essencial melhorar os níveis de desempenho dos terminais de contêineres por meio da otimização das operações logísticas, da redução de custos e da eficiência no tempo. O tempo, em especial, é um dos principais fatores usados para determinar a performance dos portos (UNCTAD, 2017). Por conseguinte, este trabalho estuda o uso da trajetória do guindaste em vez do número de realocações para melhor mensurar o tempo de operação do guindaste no PRC, e, assim, melhorar a qualidade das soluções. Desta forma, este trabalho utiliza a trajetória do guindaste para

computar diretamente o tempo operacional do guindaste, e como resultado, aumenta-se a produtividade e eficiência nas operações de empilhamento de contêineres pela otimização do tempo de operação.

Tradicionalmente, na literatura, o principal objetivo de otimização do PRC é minimizar o número de realocações, pois, empiricamente, tendo um menor número de realocações implica um menor tempo de operação. Contudo, a trajetória do guindaste é proposta, neste estudo, como uma métrica geral simples para mensurar os percursos realizados pelo guindaste a fim de retirar sequencialmente os contêineres da baia. Ademais, a métrica proposta pode ser estendida para computar diretamente outros indicadores operacionais do guindaste, por exemplo, energia, mão-de-obra, emissão de poluentes, aquecimento, data de manutenção. A trajetória do guindaste consiste na distância total percorrida (na horizontal e na vertical) pelo guindaste para retirar ordenadamente todos os contêineres da baia. A distância percorrida pelo guindaste para executar qualquer operação de empilhamento (i.e., realocação ou retirada), é calculada segundo os passos dados a seguir.

Inicialmente, para uma melhor didática, algumas notações e premissas serão apresentadas. A baia é composta de W pilhas e de H camadas. Cada espaço ocupável por um contêiner dentro da baia é indicado pela coordenada (i, j), onde $i \in \{1, \ldots, W\}$ e $j \in \{1, \ldots, H\}$ indicam, respectivamente, a pilha e a camada dentro da baia. Toda retirada de contêiner ocorre pelo canto superior direito da baia, ou seja, tem como destino a coordenada $(\hat{W}, \hat{H})$, onde $\hat{W} = W + 1$ e $\hat{H} = H + 1$. Além disso, a garra do guindaste está inicialmente posta acima da pilha 1 (i.e., na coordenada $(1, \hat{H})$) e o guindaste move horizontalmente os contêineres na altura $\hat{H}$ por razões de segurança. A Figura 7 ilustra a definição das coordenadas em uma baia com 4 pilhas e 3 camadas. Note que as coordenadas externas a baia $(1, 4)$ e $(5, 4)$ são, respectivamente, a coordenada inicial da garra do guindaste (i.e., $(1, \hat{H})$) e a coordenada de destino de uma retirada (i.e., $(\hat{W}, \hat{H})$).

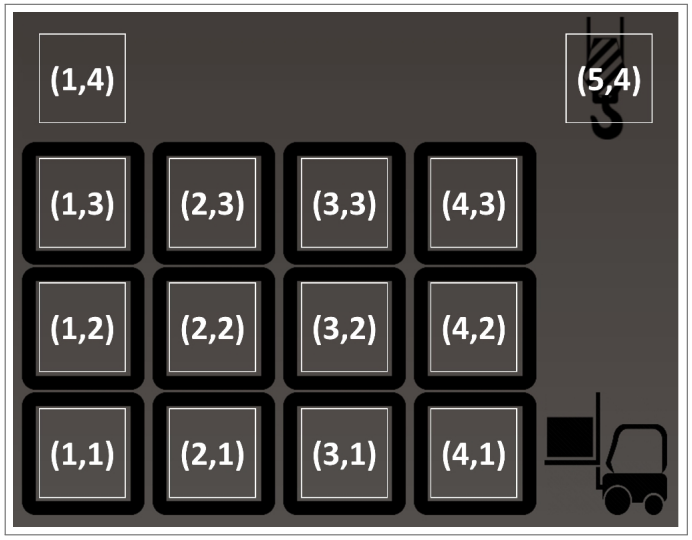


Figura 7 – Exemplo da definição de coordenadas em uma baia com 4 pilhas e 3 camadas.

A distância percorrida pelo guindaste para mover um contêiner da coordenada (i, j) para coordenar (k, l) é definida pela função $d(i, j, k, l)$ (Equação 4.1). Esta função é calculada somando três distâncias: (1) a distância vertical resultante de elevar o contêiner da camada j para $\hat{H}$ (i.e., $\mid j - \hat{H} \mid$); (2) a distância horizontal resultante de mover o contêiner da pilha i para a posição k (i.e., $\mid i - k \mid$); e (3) a distância vertical resultante de descer o contêiner de $\hat{H}$ para a camada l (i.e., $\mid l - \hat{H} \mid$). Perceba que a função $d(i, j, k, l)$ computa apenas a distância percorrida pelo guindaste enquanto ele sustenta o contêiner. No entanto, para se obter a função que computa a distância total percorrida pelo guindaste para executar qualquer operação de empilhamento, i.e., a função $t(i, j, k, l, f)$ (Equação 4.2), é necessário considera mais três percursos realizados pelo guindaste enquanto ele não sustenta o contêiner. O primeiro percurso consiste na distância horizontal percorrida para alinhar a garra do guindaste antes de iniciar uma operação. Noutras palavras, a distância horizontal resultante de mover a garra do guindaste da coordenada $(f, \hat{H})$ para a coordenada (i, j) (i.e., $\mid f - i \mid$), onde $(f, \hat{H})$ indica em que coordenada estava a garra do guindaste anteriormente. Os outros dois percursos são inerentes à distância vertical resultante de descer a garra do guindaste de $\hat{H}$ para a camada j no intuito de coletar o contêiner (i.e., $\mid j - \hat{H} \mid$) e à distância vertical resultante de retornar a garra do guindaste ao topo após ter posto o contêiner na camada l (i.e., $\mid l - \hat{H} \mid$).

$$d(i, j, k, l) = \underbrace{\mid j - \hat{H} \mid}_{(1)} + \underbrace{\mid i - k \mid}_{(2)} + \underbrace{\mid l - \hat{H} \mid}_{(3)} \tag{4.1}$$

$$\begin{aligned} t(f, i, j, k, l) = \; & \underbrace{|f - i|}_{\text{alinhar garra}} + \underbrace{|j - \hat{H}|}_{\text{descer garra}} + \underbrace{|j - \hat{H}|}_{\text{elevar contêiner}} + \\ & \underbrace{|i - k|}_{\text{mover contêiner}} + \underbrace{|l - \hat{H}|}_{\text{descer contêiner}} + \underbrace{|l - \hat{H}|}_{\text{subir garra}} \end{aligned} \tag{4.2}$$

A função $t(i, j, k, l, f)$ computa a trajetória do guindaste para uma operação de empilhamento. Já, para computar a trajetória do guindaste de uma solução do PRC basta apenas adicionar a trajetória do guindaste de cada operação que compõem a solução. Comparando as duas métricas de otimização para o PRC, a trajetória do guindaste mensura melhor o tempo de operação do guindaste em uma solução do que o número de realocações realizadas, e isso pode ser provado, considerando os seguintes argumentos. Em primeiro lugar, qualquer operação de empilhamento (i.e., realocação ou retirada) está associada a uma ou mais unidades de distância de uma trajetória do guindaste, então a trajetória do guindaste é mais precisa que o número de realocações. Além do que, a trajetória do guindaste considera, em sua computação, as realocações e as retiradas enquanto que a métrica baseada no número de realocações considera apenas as realocações. Em último lugar, minimizar o número de realocações não garante a solução para o PRC

com o tempo mínimo de operação do guindaste e a prova desta sentença será descrita mais adiante. Outra vantagem da trajetória do guindaste, em relação à métrica baseada no número de realocações, é que outros indicadores operacionais do guindaste podem ser diretamente computados observando as distâncias percorridas e certos parâmetros do guindaste, tais como, velocidades, taxas por unidade de distância, etc.

Consequentemente, o tempo de operação do guindaste pode ser calculado utilizado a trajetória do guindaste e as velocidades horizontais (i.e., velocidades do *trolley*[1]) e as velocidades verticais (i.e., velocidades da garra do guindaste). Para o guindaste, quatro velocidades são assumidas: a velocidade horizontal segurando um contêiner (ν_1), a velocidade horizontal não segurando um contêiner (ν_2), a velocidade vertical segurando um contêiner (ν_3) e a velocidade vertical não segurando um contêiner (ν_4). Uma vez dadas essas velocidades, a função $w(i, j, k, l, f)$ (Equação 4.3) define o tempo de operação que o guindaste consome para realizar qualquer operação de empilhamento. Portanto, análogo ao mencionado para a trajetória do guindaste, o tempo de operação do guindaste em uma solução do PRC é computado adicionando os tempos de operação despendidos em cada operação que compõem a solução.

$$\begin{aligned} w(f,i,j,k,l) = \underbrace{|f-i|/\nu_2}_{\text{alinhar garra}} + \underbrace{|j-\hat{H}|/\nu_4}_{\text{descer garra}} + \underbrace{|j-\hat{H}|/\nu_3}_{\text{elevar contêiner}} + \\ \underbrace{|i-k|/\nu_1}_{\text{mover contêiner}} + \underbrace{|l-\hat{H}|/\nu_3}_{\text{descer contêiner}} + \underbrace{|l-\hat{H}|/\nu_4}_{\text{subir garra}} \end{aligned} \tag{4.3}$$

Observe que, em cenários onde os valores das velocidades do guindaste são iguais a 1, os valores da trajetória e do tempo de operação do guindaste também são iguais, i.e., $t(f, i, j, k, l) = w(f, i, j, k, l)$. Além disso, nos cenários onde os valores de todas as velocidades do guindaste são idênticos, a trajetória e o tempo de operação são equiparáveis. Assim, nessas condições, minimizar a trajetória do guindaste equivale-se a minimizar o tempo de operação do guindaste. O mesmo se aplica a qualquer outro indicador operacional derivado a partir trajetória do guindaste, desde que os valores dos parâmetros inerentes ao indicador sejam idênticos. Isto significa que esses parâmetros não são determinantes para encontrar o valor mínimo do indicador operacional. Portanto, nessas condições, minimizar a trajetória do guindaste equivale-se a minimizar o tempo de operação guindaste.

Em vista disso, tem-se mais um argumento para afirmar que a trajetória do guindaste é uma métrica melhor para computar indiciadores operacionais do guindaste, no PRC, do que a métrica baseada em número de realocações realizadas. A respeito dessa afirmação, apresenta-se, a seguir, a prova de outro argumento favorável ao uso da trajetória e citado anteriormente. Além disso, esta aprova ratifica a Hipótese 1 desta tese.

[1] O *troley* é o dispositivo do guindaste que se move horizontalmente na baia e desloca a garra do guindaste sobre as pilhas da baia.

Lema 2. *Minimizar o número de realocações não garante a solução para o PRC com o tempo mínimo de operação do guindaste.*

Prova [Lema 2]. Para a comprovação do Lema 2, basta apenas encontrar duas soluções s_1 e s_2, tal que o número de realocações em s_1 seja menor que o em s_2 e o tempo de operação do guindaste em s_1 seja maior que o em s_2. Desta forma, seja o tempo de operação do guindaste mensurado por meio da Equação 4.3, e as velocidades do guindaste sejam $\nu_1 = 1$, $\nu_2 = 2$, $\nu_3 = 1$ e $\nu_4 = 2$. Ademais, sejam s_1 e s_2 as soluções apresentadas respectivamente pela Figura 8 e Figura 9. Ambas as soluções são para uma instância do PRC com uma configuração inicial de baia consistindo de 12 contêineres, 6 pilhas ($W = 6$) e altura máxima de 4 contêineres ($H = 4$).

As Tabelas 3 e 4 descrevem, respectivamente, o tempo de operação do guindaste para as soluções s_1 e s_2. A solução s_1 é composta de 2 realocações e 12 retiradas, resultando em um tempo de operação igual a 140 unidades de tempo. A solução s_2 é composta de 3 realocações e 12 retiradas, resultando em um tempo de operação igual a 134 unidades de tempo. Assim, o número de realocações em s_1 é menor que em s_2, mas o tempo de operação do guindaste em s_1 é maior que em s_2. Portanto, o Lema 2 está provado. Com isto, não é consistente inferir ou computar o tempo de operação do guindaste a partir do número de realocações realizadas na resolução do PRC. Além disso, esta prova corrobora que a trajetória do guindaste é uma métrica melhor para computar indiciadores operacionais do guindaste, em particular, o tempo de operação do guindaste. Diante disso, a trajetória do guindaste e indiciadores operacionais derivados a partir dela são relevantes objetivos de otimização a serem estudados ao PRC. Em particular, esse trabalho estuda o tempo de operação do guindaste computado a partir da trajetória.

□

A partir do entendimento de como as novas métricas, propostas neste trabalho de pesquisa, contribuem para a melhora da qualidade das soluções para o PRC em termos de tempo de operação, descreve-se, nas demais seções deste documento, os métodos de otimização propostos para solucionar o PRC, visando minimizar o tempo de operação despendido pelo guindaste. Sabido que o tempo de operação é mensurado a partir da trajetória do guindaste, e não em termos de número de realocações realizadas.

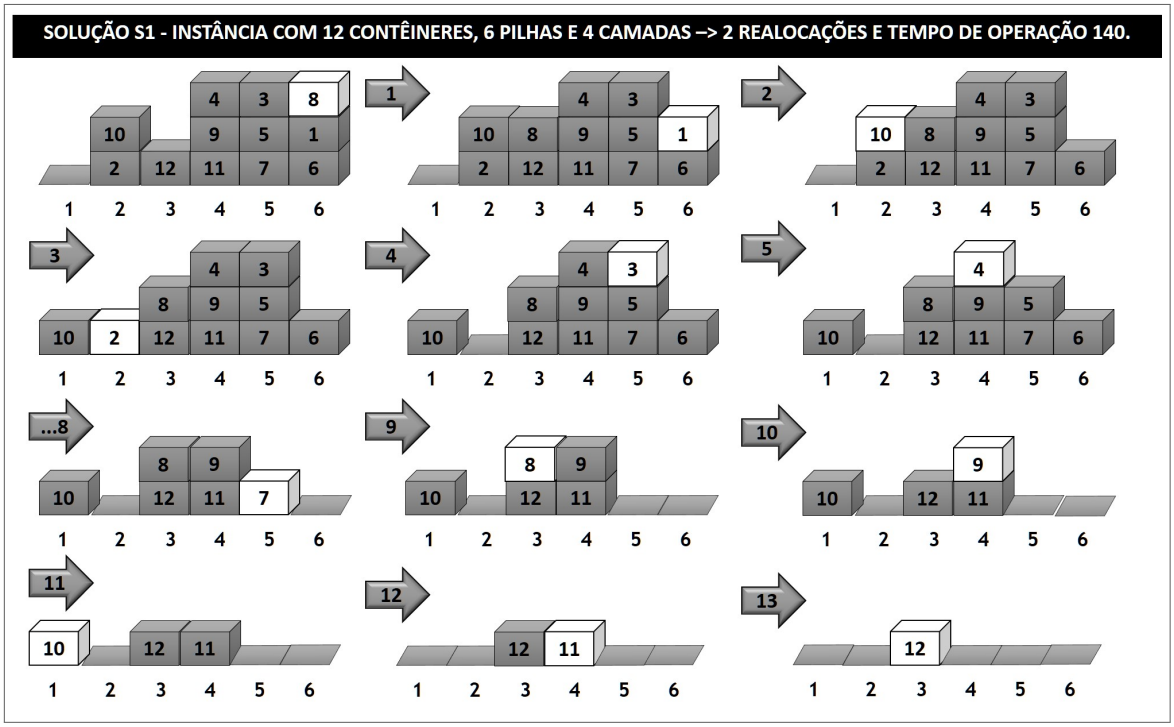


Figura 8 – Solução s_1, menor número de realocações com maior tempo de operação.

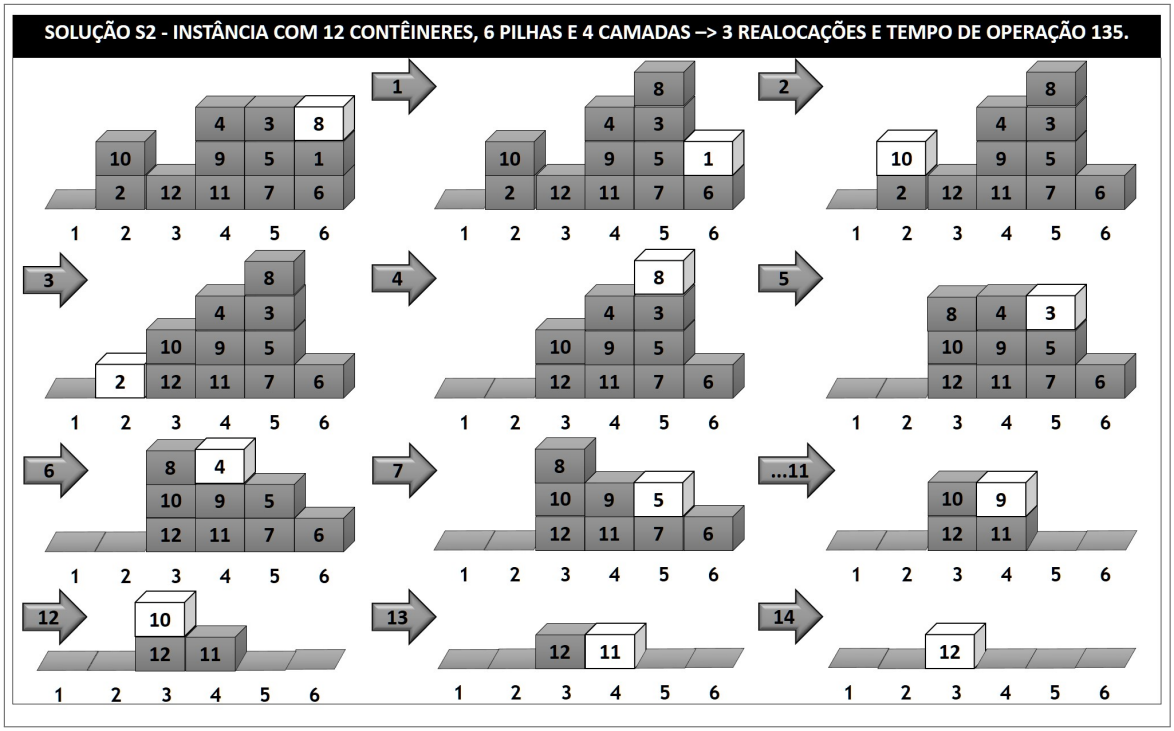


Figura 9 – Solução s_2, maior número de realocações com menor tempo de operação.

Tabela 3 – Tempo de operação do guindaste na solução s_1.

OPERAÇÃO (CONTÊINER)	**COORDENADAS (i,j) (k,l) (f,$\hat{H}$)**	**TEMPO DE OPERAÇÃO** $\lvert i-k\rvert/\nu_1 + (\lvert j-\hat{H}\rvert + \lvert l-\hat{H}\rvert)/\nu_3 +$ $\lvert f-i\rvert/\nu_2 + (\lvert j-\hat{H}\rvert + \lvert l-\hat{H}\rvert)/\nu_4$
REALOCAÇÃO(8)	(6,3) (3,2) (1,5)	$\lvert 6-3\rvert/1 + (\lvert 3-5\rvert + \lvert 2-5\rvert)/1 +$ $\lvert 1-6\rvert/2 + (\lvert 3-5\rvert + \lvert 2-5\rvert)/2 = 13$
RETIRADA(1)	(6,2) (7,5) (3,5)	$\lvert 6-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 3-6\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 7$
REALOCAÇÃO(10)	(2,2) (1,1) (7,5)	$\lvert 2-1\rvert/1 + (\lvert 2-5\rvert + \lvert 1-5\rvert)/1 +$ $\lvert 7-2\rvert/2 + (\lvert 2-5\rvert + \lvert 1-5\rvert)/2 = 14$
RETIRADA(2)	(2,1) (7,5) (1,5)	$\lvert 2-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 1-2\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 11,5$
RETIRADA(3)	(5,3) (7,5) (7,5)	$\lvert 5-7\rvert/1 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/2 = 6$
RETIRADA(4)	(4,3) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(5)	(5,2) (7,5) (7,5)	$\lvert 5-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(6)	(6,1) (7,5) (7,5)	$\lvert 6-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-6\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(7)	(5,1) (7,5) (7,5)	$\lvert 5-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 9$
RETIRADA(8)	(3,2) (7,5) (7,5)	$\lvert 3-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-3\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 10,5$
RETIRADA(9)	(4,2) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 9$
RETIRADA(10)	(1,1) (7,5) (7,5)	$\lvert 1-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-1\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 15$
RETIRADA(11)	(4,1) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 10,5$
RETIRADA(12)	(3,1) (7,5) (7,5)	$\lvert 3-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-3\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 12$
TOTAL: 2 REALOCAÇÕES E TEMPO DE OPERAÇÃO 140		

Tabela 4 – Tempo de operação do guindaste na solução s_2.

OPERAÇÃO (CONTÊINER)	**COORDENADAS (i,j) (k,l) (f,$\hat{H}$)**	**TEMPO DE OPERAÇÃO** $\lvert i-k\rvert/\nu_1 + (\lvert j-\hat{H}\rvert + \lvert l-\hat{H}\rvert)/\nu_3 +$ $\lvert f-i\rvert/\nu_2 + (\lvert j-\hat{H}\rvert + \lvert l-\hat{H}\rvert)/\nu_4$
REALOCAÇÃO(8)	(6,3) (5,4) (1,5)	$\lvert 6-5\rvert/1 + (\lvert 3-5\rvert + \lvert 4-5\rvert)/1 +$ $\lvert 1-6\rvert/2 + (\lvert 3-5\rvert + \lvert 4-5\rvert)/2 = 8$
RETIRADA(1)	(6,2) (7,5) (5,5)	$\lvert 6-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 5-6\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 6$
REALOCAÇÃO(10)	(2,2) (3,2) (7,5)	$\lvert 2-3\rvert/1 + (\lvert 2-5\rvert + \lvert 2-5\rvert)/1 +$ $\lvert 7-2\rvert/2 + (\lvert 2-5\rvert + \lvert 2-5\rvert)/2 = 12,5$
RETIRADA(2)	(2,1) (7,5) (3,2)	$\lvert 2-7\rvert/1 + (\lvert 1-2\rvert + \lvert 5-2\rvert)/1 +$ $\lvert 3-2\rvert/2 + (\lvert 1-2\rvert + \lvert 5-2\rvert)/2 = 11,5$
REALOCAÇÃO(8)	(5,4) (3,3) (7,5)	$\lvert 5-3\rvert/1 + (\lvert 4-5\rvert + \lvert 3-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 4-5\rvert + \lvert 3-5\rvert)/2 = 7,5$
RETIRADA(3)	(5,3) (7,5) (3,3)	$\lvert 5-7\rvert/1 + (\lvert 3-3\rvert + \lvert 5-3\rvert)/1 +$ $\lvert 3-5\rvert/2 + (\lvert 3-3\rvert + \lvert 5-3\rvert)/2 = 6$
RETIRADA(4)	(4,3) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(5)	(5,2) (7,5) (7,5)	$\lvert 5-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(6)	(6,1) (7,5) (7,5)	$\lvert 6-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-6\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 7,5$
RETIRADA(7)	(5,1) (7,5) (7,5)	$\lvert 5-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-5\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 9$
RETIRADA(8)	(3,3) (7,5) (7,5)	$\lvert 3-7\rvert/1 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-3\rvert/2 + (\lvert 3-5\rvert + \lvert 5-5\rvert)/2 = 9$
RETIRADA(9)	(4,2) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 9$
RETIRADA(10)	(3,2) (7,5) (7,5)	$\lvert 3-7\rvert/1 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-3\rvert/2 + (\lvert 2-5\rvert + \lvert 5-5\rvert)/2 = 10,5$
RETIRADA(11)	(4,1) (7,5) (7,5)	$\lvert 4-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-4\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 10,5$
RETIRADA(12)	(3,1) (7,5) (7,5)	$\lvert 3-7\rvert/1 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/1 +$ $\lvert 7-3\rvert/2 + (\lvert 1-5\rvert + \lvert 5-5\rvert)/2 = 12$
TOTAL: 3 REALOCAÇÕES E TEMPO DE OPERAÇÃO 134		

4.3 MÉTODOS DE OTIMIZAÇÃO EXATOS

Nesta seção são apresentados os métodos de otimização propostos para solucionar o PRC otimamente, ou seja, a solução ótima para o problema é encontrada. Em todos os métodos propostos, a solução ótima é aquela com o tempo mínimo de operação do guindaste, onde este tempo é computado a partir da trajetória do guindaste. Desta forma, esta seção está organizada da seguinte maneira. Na Seção 4.3.1, o PRC é formulado por uma modelagem matemática, pela qual um modelo de Programação Linear Inteira (PLI) para o problema é proposto. Na Seção 4.3.2, é proposto um algoritmo baseado no algoritmo de Dijkstra. Em seguida, o algoritmo é estendido para um algoritmo *A Star* (A*), na Seção 4.3.3.

4.3.1 Modelo de Programação Linear Inteira

Para modelar matematicamente um problema de otimização é necessário identificar a função objetivo, ou seja, a função a ser medida e otimizada. Depois é preciso identificar as características peculiares ao problema que influenciam como essa a função objetivo será otimizada. Em outras palavras, é preciso identificar as constantes, as variáveis de decisão e as restrições do problema. Desta maneira, a otimização do problema consistirá em encontrar os valores das variáveis de decisão de modo a maximizar ou minimizar a função objetivo e a atender a todas as restrições impostas ao problema. Após a modelagem matemática do problema, um modelo de otimização é produzido. Quando a função objetivo do modelo é uma função linear e as restrições desse modelo (i.e., equações e/ou inequações) também são lineares, o modelo de otimização é dito ser um modelo de *Programação Linear*. Se, além disso, as variáveis de decisão do modelo assumem apenas valores inteiros, tem-se um modelo de Programação Linear Inteira (PLI).

Modelos de PLI podem ser solucionados otimamente, dentro de limites computacionais impostos, por meio de um *solver*. Um *solver* é um pacote de software matemático composto por algoritmos e técnicas de otimização projetados para solucionar modelos matemáticos. Existem diversos *solvers* comerciais e gratuitos disponíveis para resolver modelos de programação linear. Em geral, eles diferem entre si pelos métodos de otimização implementados e tipos de modelos matemáticos que são capazes de resolver.

Em vista disso, um modelo de PLI fornece uma descrição matemática de um problema de otimização e possibilita a resolução ótima deste problema por meio de um *solver*. Deste modo, a seguir, descreve-se matematicamente o PRC por meio de um modelo de PLI. Mas antes, para facilitar o entendimento, são apresentados as premissas, notações e terminologias adotadas no modelo.

A baia tem a largura máxima W e altura máxima H, e seja $\hat{W} = W + 1$ e $\hat{H} = H + 1$. Inicialmente, a baia contém N contêineres rotulados de $1, \ldots, N$, de acordo com os seus respectivos valores de prioridade de saída, tal que n indica a prioridade de um contêiner, i.e., $n \in \{1, \ldots, N\}$. Cada espaço ocupável por um contêiner dentro da baia é indicado

pela coordenada (i, j), onde $i \in \{1, \ldots, W\}$ e $j \in \{1, \ldots, H\}$ indicam, respectivamente, a pilha e a camada dentro da baia. Além disso, toda retirada de contêiner ocorre pelo canto superior direito da baia (i.e., $(\hat{W}, \hat{H})$) e a garra do guindaste está inicialmente acima da pilha 1. Por fim, o horizonte de tempo é discretizado pela introdução de períodos de tempos $t \in \{1, \ldots, T\}$, onde cada período de tempo t é definido pelo conjunto completo de operações necessárias para retirar um contêiner $n = t$, o conjunto inclui as operações de realocação, se necessárias, e a retirada do contêiner n.

O modelo de otimização formulado, neste trabalho, utilizou como referência os modelos matemáticos desenvolvidos em (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015; ZEHENDNER et al., 2015). A principal diferença entre os modelos está no objetivo de otimização. O modelo proposto visa minimizar o tempo de operação do guindaste, enquanto os demais visam minimizar o número de operações de realocação. Além disso, o modelo proposto possui novos parâmetros, variáveis de decisão e restrições, inerentes ao tempo de operação, a fim de encontrar o tempo mínimo de operação do guindaste.

A respeito das variáveis do modelo proposto, ele possui duas variáveis de decisão binárias. A primeira variável (b_{ijnt}) indica onde os contêineres estão localizados em dado período de tempo t. A última variável ($x_{ijklfnt}$) representa a operação realizada pelo guindaste, que pode ter sido uma realocação ou uma recuperação, lembrando que toda operação de retirada tem como coordenada de destino ($k = \hat{W},\ l = \hat{H}$). Em particular, o modelo proposto representa operações de realocação e recuperação em uma única variável de operação (i.e., $x_{ijklfnt}$), uma vez que ambas as operações compõem o tempo de operação do guindaste. Além disso, a variável de operação considera a posição horizontal que estava a garra do guindaste antes de realizar a operação (i.e., f) para calcular o tempo de operação.

$$b_{ijnt} = \begin{cases} 1, & \text{se o contêiner } n \text{ está na } (i,j) \text{ no período de tempo } t, \\ 0, & \text{caso contrário;} \end{cases}$$

$$x_{ijklfnt} = \begin{cases} 1, & \text{se o contêiner } n \text{ é movido da } (i,j) \text{ para } (k,l) \text{ no período de tempo } t \\ & \text{e a garra do guindaste estava anteriormente na posição horizontal } f, \\ 0, & \text{caso contrário;} \end{cases}$$

Oito parâmetros são usados no modelo proposto. O primeiro (v_{nt}) determina se um contêiner n já foi recuperado antes do período de tempo t. O segundo (s_{ijn}) indica se o contêiner n está na coordenada (i, j) na configuração inicial da baia. O terceiro (w_{ijklf}) retorna o tempo de operação realizado pelo guindaste para mover um contêiner da coordenada (i, j) para a coordenada (k, l). Este tempo de operação é computado por meio da trajetória do guindaste e suas 4 velocidades, conforme definido pela Equação 4.3, na Seção 4.2. Estas quatro velocidades são respectivamente o quarto (ν_1), o quinto (ν_2), o sexto

(ν_3) e o sétimo (ν_4) parâmetro do modelo, e são utilizados para computar o parâmetro w_{ijklf}. O último parâmetro (p) indica, na configuração inicial da baia, em que camada está o topo da pilha em que o contêiner 1 está situado. Este parâmetro é utilizado no modelo para identificar a coordenada do primeiro contêiner a ser movimentado na baia. Vale pontuar que os últimos sete parâmetros, aqui apresentados, são os novos parâmetros, mencionados anteriormente, e foram elaborados para este modelo.

$$
\begin{aligned}
v_{nt} &= \begin{cases} 0, & \text{se } n \geq t, \\ 1, & \text{caso contrário;} \end{cases} \\
s_{ijn} &= \begin{cases} 1, & \text{se o contêiner } n \text{ está inicialmente na } (i,j) \\ 0, & \text{caso contrário;} \end{cases} \\
w_{ijklf} &= (|i-k|)/\nu_1 + (|j-\hat{H}| + |l-\hat{H}|)/\nu_3 + \\
&\quad (|f-i|)/\nu_2 + (|j-\hat{H}| + |l-\hat{H}|)/\nu_4 \\
p &= j + \sum_{j'=j+1}^{H} \sum_{n=2}^{N} s_{ij'n}, \text{ quando } s_{ij1} = 1
\end{aligned}
$$

Definidas as variáveis de decisão e as constante do modelo proposto, apresenta-se, abaixo, o conjunto de restrições, enumeradas de (**A**) à (**N**), para garantir que as transições entre as possíveis configurações da baia sejam válidas. Em algumas dessas restrições, utiliza-se a notação $\bar{M}$ para indicar um número muito grande face às grandezas da equação ou inequação em questão. Vale ressaltar que apenas restrições (B), (D) e (E) são idênticas àquelas encontradas em (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015), as demais restrições foram elaboradas para este modelo.

(A) No período de tempo $t = 1$, a posição inicial dos contêineres na baia (i.e., b_{ijn1}) é definida de acordo com os valores do parâmetro s_{ijn}, que indicam a configuração inicial da baia. Assim, se $s_{ijn} = 1$ então obrigatoriamente $b_{ijn1} = 1$.

$$s_{ijn} \leq b_{ijn1}, \quad i = 1, \ldots, W, \ j = 1, \ldots, H, \ n = 1, \ldots, N \tag{4.4}$$

(B) Em cada período de tempo, há apenas duas possibilidades de localização para cada contêiner $n \in \{1, \ldots, N\}$. O contêiner n está em uma pilha dentro da baia (i.e., $\sum_{i=1}^{W} \sum_{j=1}^{H} b_{ijnt} = 1$) ou ele está fora da baia (i.e., $v_{nt} = 1$), pois, ele já foi removido. Além disso, garante-se que, em cada período de tempo, há no máximo um contêiner dentro da baia com prioridade n, porque o valor máximo de $\sum_{i=1}^{W} \sum_{j=1}^{H} b_{ijnt}$ é 1.

$$\sum_{i=1}^{W} \sum_{j=1}^{H} b_{ijnt} + v_{nt} = 1, \ n, t = 1, \ldots, N \tag{4.5}$$

(C) As operações de retirada (i.e., $x_{ijklfnt'}$ onde $k = \hat{W}$ e $l = \hat{H}$) são realizadas de acordo com os valores do parâmetro v_{nt} e, assim, elas atendem a discretização do

horizonte de tempo. Noutras palavras, a operação de retirada do contêiner n ocorre apenas uma vez, e essa vez ocorre no período de tempo $t' = n$. Isto acontece por duas razões: (1) quando $v_{nt} = 0$, então retiradas sobre o contêiner n em um tempo $t' < n$ não são permitidas; e (2) quando $v_{nt} = 1$, a retirada sobre o contêiner n só ocorre no máximo uma vez dentre todos os períodos de tempo (i.e., $t' \in \{1, \ldots, N\}$) e obrigatoriamente deve ocorrer um retirada quando $t' = n = t - 1$.

$$v_{nt} = \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{f=1}^{\hat{W}} \sum_{t'=1}^{t-1} x_{ijklfnt'},$$
$$n = 1, \ldots, N,\ t = 2, \ldots, N+1,\ k = \hat{W},\ l = \hat{H} \tag{4.6}$$

(D) Em cada período de tempo, cada espaço ocupável por um contêiner na baia, indicado pela coordenada (i, j), deve ser ocupado por um contêiner no máximo.

$$\sum_{n=1}^{N} b_{ijnt} \leq 1,\ i = 1, \ldots, W,\ j = 1, \ldots, H,\ t = 1, \ldots, N \tag{4.7}$$

(E) Nenhum espaço vazio é permitido entre contêineres de uma pilha, ou seja, todo contêiner deve estar na base de uma pilha ou posto em cima de outro contêiner. Assim, se não existe um contêiner no espaço (i, j) (i.e., $b_{ijnt} = 0$), então obrigatoriamente não pode existir um contêiner no espaço $(i, j + 1)$ (i.e., $b_{ij+1nt} = 0$). Em resumo, esta restrição garante que não haja contêineres acima de um espaço vazio, ou seja, flutuando na baia.

$$\sum_{n=1}^{N} b_{ijnt} \geq \sum_{n=1}^{N} b_{ij+1nt},\ i = 1, \ldots, W,\ j = 1, \ldots, H-1,\ t = 1, \ldots, N \tag{4.8}$$

(F) Em cada período de tempo t, uma nova configuração viável no período de tempo $t+1$ é alcançada através da aplicação de um conjunto válido de operações. Por exemplo, se um contêiner n encontra-se na posição (i, j) no tempo $t + 1$ (i.e., $b_{ijnt+1} = 1$), então um destes dois fatos pode ter ocorrido sobre o contêiner n no período t: (1) o contêiner n estava na posição (i, j) e nenhuma operação ocorreu sobre ele, i.e., $b_{ijnt1} = 1$, $\sum_{k=1}^{W} \sum_{l=1}^{H} \sum_{f=1}^{\hat{W}} x_{klijfnt} = 0$ e $\sum_{k=1}^{\hat{W}} \sum_{l=1}^{\hat{H}} \sum_{f=1}^{\hat{W}} x_{ijklfnt} = 0$; ou (2) o contêiner n estava na posição (k, l) e foi realocado para a posição (i, j), i.e., $b_{ijnt} = 0$, $\sum_{k=1}^{W} \sum_{l=1}^{H} \sum_{f=1}^{\hat{W}} x_{klijfnt} = 1$ e $\sum_{k=1}^{\hat{W}} \sum_{l=1}^{\hat{H}} \sum_{f=1}^{\hat{W}} x_{ijklfnt} = 0$. Outro cenário, se um contêiner n não se encontra na posição (i, j) no tempo $t+1$ (i.e., $b_{ijnt+1} = 0$), então um destes dois fatos pode ter ocorrido sobre o contêiner n no período t: (1) o contêiner n estava na posição (i, j) e foi retirado ou realocado, i.e., $b_{ijnt1} = 1$, $\sum_{k=1}^{W} \sum_{l=1}^{H} \sum_{f=1}^{\hat{W}} x_{klijfnt} = 0$ e $\sum_{k=1}^{\hat{W}} \sum_{l=1}^{\hat{H}} \sum_{f=1}^{\hat{W}} x_{ijklfnt} = 1$; ou (2) o contêiner n não estava na posição (i, j) e nenhuma operação ocorreu sobre ele. i.e., $b_{ijnt} = 0$,

$\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{\hat{W}} x_{klijfnt} = 0$ e $\sum_{k=1}^{\hat{W}}\sum_{l=1}^{\hat{H}}\sum_{f=1}^{\hat{W}} x_{ijklfnt} = 0$.

$$b_{ijnt+1} = \left\{ b_{ijnt} + \sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{\hat{W}} x_{klijfnt} - \sum_{k=1}^{\hat{W}}\sum_{l=1}^{\hat{H}}\sum_{f=1}^{\hat{W}} x_{ijklfnt} \right\},$$
$$i = 1,\ldots,W,\ j = 1,\ldots,H,\ t = 1,\ldots,N-1,\ n = 1,\ldots,N \tag{4.9}$$

(G) Garantir, entre uma sequência de realocações efetuadas, a política do primeiro a entrar é o último a sair da pilha. Portanto, se no período t, o contêiner n é movido da posição (i,j) para a posição (k,l) (i.e., $\sum_{n=t+1}^{N}\sum_{f=1}^{\hat{W}} x_{ijklfnt} = 1$), então não pode existir realocações (i.e, $\sum_{n=t+1}^{N}\sum_{j'=j+1}^{H}\sum_{l'=l+1}^{H}\sum_{f=1}^{\hat{W}} x_{ij'kl'fnt} = 0$), no período t, que movam um contêiner localizado acima da posição (i,j) (i.e., posição (i,j') tal que $j' > j$) para uma posição acima de (k,l) (i.e., posição (k,l') tal que $l' > l$). Além disso, é possível realocar no máximo $H-1$ contêineres de uma pilha sempre que $\sum_{n=t+1}^{N}\sum_{f=1}^{\hat{W}} x_{ijklfnt} = 0$, ou seja, é possível realocar todos os contêineres acima de n se este não será realocado no tempo t.

$$(H-1)\left(1 - \sum_{n=t+1}^{N}\sum_{f=1}^{\hat{W}} x_{ijklfnt}\right) \geq \sum_{n=t+1}^{N}\sum_{j'=j+1}^{H}\sum_{l'=l+1}^{H}\sum_{f=1}^{\hat{W}} x_{ij'kl'fnt},$$
$$i,k = 1,\ldots,W,\ j = 2,\ldots,H,\ l = 1,\ldots,H,\ t = 1,\ldots,N-1 \tag{4.10}$$

(H) Apenas as realocações sobre os contêineres que estão acima do contêiner de maior prioridade na baia são permitidas. Assim, em cada período de tempo t, quando o contêiner de maior prioridade (i.e., $n = t$) está localizado na coordenada (i,j), i.e., $b_{ijtt} = 1$, o lado esquerdo da inequação abaixo torna-se zero, e, logo, assegura que as operações de realocação associadas aos espaços abaixo de (i,j) não sejam permitidas (i.e., $\sum_{j'=1}^{j-1}\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{W+1}\sum_{n=1}^{N} x_{ij'klfnt} = 0$) e que também não sejam autorizadas realocações oriundas de outras pilhas além da pilha i (i.e., $\sum_{\substack{i'=1\\ i'\neq i}}^{W}\sum_{j'=1}^{H}\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{W+1}\sum_{n=1}^{N} x_{i'j'klfnt} = 0$).

$$\bar{M}\,(1 - b_{ijtt}) \geq \sum_{j'=1}^{j-1}\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{\hat{W}}\sum_{n=1}^{N} x_{ij'klfnt} + \sum_{\substack{i'=1\\ i'\neq i}}^{W}\sum_{j'=1}^{H}\sum_{k=1}^{W}\sum_{l=1}^{H}\sum_{f=1}^{\hat{W}}\sum_{n=1}^{N} x_{i'j'klfnt},$$
$$i = 1,\ldots,W,\ j = 1,\ldots,H,\ t = 1,\ldots,N \tag{4.11}$$

(I) Em cada período de tempo t, se o contêiner de maior prioridade (i.e., $n = t$) está localizado na coordenada (i,j), i.e., $b_{ijtt} = 1$, então o lado esquerdo da inequação abaixo torna-se zero, e, assim, assegura que as operações de retirada com coordenada de origem diferente de (i,j) não são permitidas.

$$\bar{M}\,(1 - b_{ijtt}) \geq \sum_{\substack{j'=1\\ j'\neq j}}^{H}\sum_{f=1}^{\hat{W}}\sum_{n=1}^{N} x_{ij'klfnt} + \sum_{\substack{i'=1\\ i'\neq i}}^{W}\sum_{j'=1}^{H}\sum_{f=1}^{\hat{W}}\sum_{n=1}^{N} x_{i'j'klfnt},$$
$$i = 1,\ldots,W,\ j = 1,\ldots,H,\ k = \hat{W},\ l = \hat{H},\ t = 1,\ldots,N \tag{4.12}$$

(J) Nenhum contêiner pode ser movido para um espaço na coordenada (k, l), se este espaço está acima de um espaço vazio em um dado período de tempo t. Portanto, se o espaço na coordenada $(k, l-1)$ está vazio (i.e., $\sum_{n=1}^{N} b_{kl-1nt} = 0$) e nenhum contêiner foi realocado para a coordenada $(k, l-1)$ (i.e., $\sum_{n=1}^{N}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{f=1}^{W+1} x_{ijkl-1fnt} = 0$), então o lado direito da inequação abaixo torna-se zero e assegura que as operações de realocação com coordenada de destino (k, l) não sejam permitidas, pois, o espaço $(k, l-1)$ está vazio no período de tempo t.

$$\sum_{n=1}^{N}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{f=1}^{\hat{W}} x_{ijklfnt} \leq \sum_{n=1}^{N} b_{kl-1nt} + \sum_{n=1}^{N}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{f=1}^{\hat{W}} x_{ijkl-1fnt},$$
$$k = 1, \ldots, W,\ l = 2, \ldots, H,\ t = 1, \ldots, N \tag{4.13}$$

(K) As seguintes operações de realocação não são permitidas: (1) de uma pilha para a mesma pilha (i.e., $i = k$); (2) para a coordenada (k, l) tal que $k = \hat{W}$ e $l \leq H$, ou seja, respeita-se a largura máxima da baia; (3) para a coordenada (k, l) tais que $k \leq W$ e $l = \hat{H}$, ou seja, respeita-se a altura máxima da baia; e (4) com coordenada de origem sobre a primeira camada (i.e., $(i, 1)$), porque se um contêiner está na base de uma pilha (i.e., $j = 1$) ele não precisa ser realocado, apenas precisa ser removido.

$$\underbrace{x_{ijilfnt}}_{(1)} = 0;\ \underbrace{x_{ijk'lfnt}}_{(2)} = 0;\ \underbrace{x_{ijkl'fnt}}_{(3)} = 0;\ \underbrace{x_{i1klfnt}}_{(4)} = 0;$$
$$i, k = 1, \ldots, W,\ j, l = 1, \ldots, H,\ f = 1, \ldots, W+1,\ n, t = 1, \ldots, N$$
$$k' = \hat{W},\ l' = \hat{H} \tag{4.14}$$

(L) A próxima operação do guindaste, após um operação de realocação $x_{ijklfnt}$, sempre ocorre na mesma pilha i, no mesmo período de tempo t, em uma camada abaixo $(j-1)$, e com a posição horizontal anterior da garra f igual a k. Desse modo, se $x_{ijklfnt} = 1$, então obrigatoriamente deve existir uma operação do guindaste subsequente $x_{ij-1k'l'kn't}$.

$$x_{ijklfnt} \leq \sum_{k'=1}^{\hat{W}}\sum_{l'=1}^{\hat{H}}\sum_{n'=1}^{N} x_{ij-1k'l'kn't},\ i, k = 1, \ldots, W,\ j = 2, \ldots, H,$$
$$l = 1, \ldots, H, f = 1, \ldots, \hat{W},\ t = 1, \ldots, N-1,\ n = t+1, \ldots, N \tag{4.15}$$

(M) A próxima operação do guindaste, após uma operação de retirada $x_{ijklftt}$, deve acontecer no período de tempo $(t+1)$ e com a posição horizontal anterior da garra f igual a k. Dessa forma, se $x_{ijklftt} = 1$, então obrigatoriamente deve existir uma operação do guindaste subsequente $x_{i'j'k'l'kn't+1}$.

$$x_{ijklftt} \leq \sum_{i'=1}^{W}\sum_{j'=1}^{H}\sum_{k'=1}^{\hat{W}}\sum_{l'=1}^{\hat{H}}\sum_{n'=1}^{N} x_{i'j'k'l'kn't+1},\ i = 1, \ldots, W,\ j = 1, \ldots, H,$$
$$k = \hat{W},\ l = \hat{H},\ f = 1, \ldots, \hat{W},\ t = 1, \ldots, N-1 \tag{4.16}$$

(N) A garra do guindaste deve estar inicialmente na posição horizontal 1. Consequentemente, a primeira operação do guindaste ocorre no período de tempo ($t = 1$), com a posição horizontal anterior da garra ($f = 1$), e terá como coordenada de origem (i, j'), onde i é a pilha onde está o contêiner 1 e $j' = p$. Desse modo, se o contêiner 1 está na coordenada (i, j) (i.e., $b_{ij11} = 1$), então a primeira operação (i.e., $x_{ij'kl1n1}$), obrigatoriamente, será sobre o contêiner localizado no topo da pilha i.

$$b_{ij11} \leq \sum_{k=1}^{\hat{W}} \sum_{l=1}^{\hat{H}} \sum_{n=1}^{N} x_{ij'kl1n1},\ i = 1, \ldots, W,\ j = 1, \ldots, H,\ j' = p \tag{4.17}$$

O modelo matemático de PLI proposto para o PRC é apresentado abaixo. Este modelo tem por objetivo minimizar o tempo de operação do guindaste, permitindo que ele recupere sequencialmente todos os contêineres armazenados em uma baia de acordo com as suas prioridades de saída.

$$Minimizar \left(\sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{\hat{W}} \sum_{l=1}^{\hat{H}} \sum_{f=1}^{\hat{W}} \sum_{n=1}^{N} \sum_{t=1}^{N} x_{ijklfnt} \times w_{ijklf} \right)$$
$$Sujeito\ a\ (Eq.\ 4.4) - (Eq.\ 4.17).$$

Vale salientar que o modelo aqui proposto é aplicado para o PRC restrito, como evidenciado em algumas restrições do modelo, onde apenas realocações restritas são consideradas. A restrição mais enfática a esse respeito é a restrição (H), pela qual são permitidas, no modelo, apenas as realocações sobre os contêineres que estão acima do contêiner de maior prioridade na baia. Este trabalho de pesquisa não propôs um modelo matemático para o PRC irrestrito pelas seguintes razões. Como reportado em (CASERTA; SCHWARZE; VOSS, 2012) e (PETERING; HUSSEIN, 2013), um modelo matemático para o PRC irrestrito possui um grande espaço de busca, tornando inviável o tempo computacional para encontrar a solução. Além disso, um modelo para o PRC irrestrito precisa de um constante T para limitar o número máximo de realocações antes da retirada do contêiner de maior prioridade, mas o uso dessa constante é impraticável, porque quanto maior o valor de T maior é o espaço de busca, e um valor menor para T pode gerar soluções distantes da solução ótima. Consequentemente, Petering e Hussein (2013) afirma, segundo os experimentos realizados, que um modelo matemático para o PRC irrestrito mostrou-se insuficiente para instâncias do mundo real. Desde então, nenhum modelo matemático para o PRC irrestrito foi experimentado, mas apenas modelos para o PRC restrito, como observado em (AZARI, 2015; EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2015; ZEHENDNER et al., 2015; GALLE; BARNHART; JAILLET, 2018), e similarmente neste estudo.

Felizmente, quando não é possível formular um modelo matemático para um problema ou até mesmo quando modelo formulado é insuficiente em casos reais. Algoritmos podem ser usados para resolver otimamente problemas de otimização. Portanto, algoritmos exatos para solucionar o PRC restrito e o PRC irrestrito são apresentados nas seções a seguir.

4.3.2 Algoritmo de Dijkstra

Alguns problemas de otimização podem ser representados por um grafo de busca, de maneira que encontrar o caminho mais curto entre um nó inicial e um nó alvo do grafo equivale-se a encontrar a solução ótima do problema. A Figura 10, ilustra um exemplo do problema de encontrar o caminho mínimo entre cidades A e G. Este problema está representado por um grafo de busca, onde cada nó do grafo é uma cidade e as conexões (i.e. arestas) entre as cidades têm uma distância em Km. Além disso, o nó inicial e nó alvo nesse grafo de busca são respectivamente a cidades A e G, e o caminho mínimo desse problema consiste no percurso passando, nesta ordem, pelas seguintes cidades A, C, D, B, E e G. Este caminho mínimo resultou no custo mínimo de 16 Km.

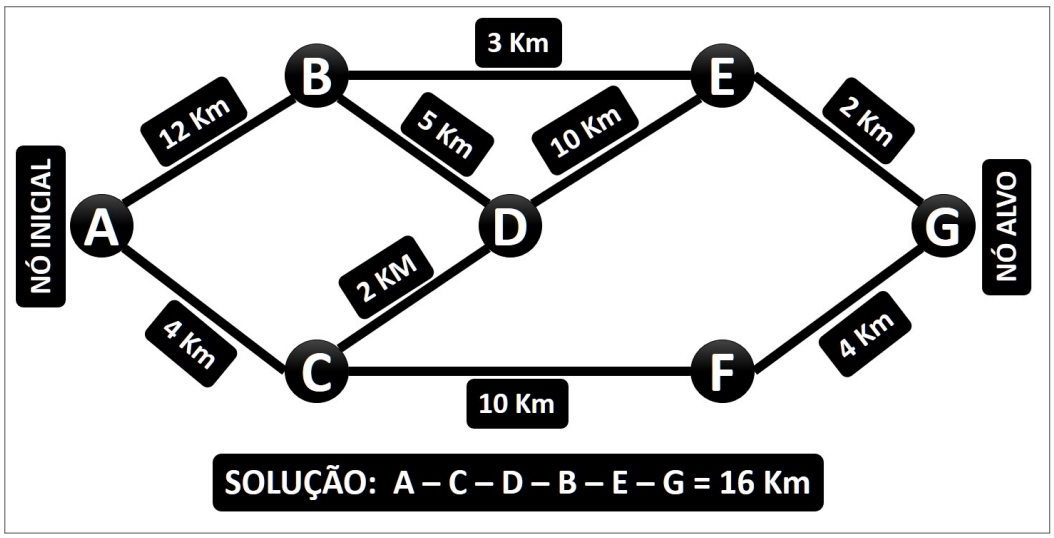


Figura 10 – Representação do problema do caminho mínimo num grafo de busca.

Os algoritmos especializados em solucionar o problema do caminho mínimo são casualmente chamados de *algoritmos de busca de caminhos*. O algoritmo de Dijkstra (DIJKSTRA, 1959) é um deles, e este algoritmo é referência na geração de soluções ótimas para uma grande lista de problemas de otimização com objetivo de encontrar o caminho mais curto. Este algoritmo de busca recebe um grafo de busca composta: do nó inicial, do nó alvo e dos demais nós, das arestas entre os nós e os custos de cada aresta, onde o custo obrigatoriamente precisa ter valor positivo. Cada nó n no grafo é avaliada por uma função de pontuação $f(n)$. Esta função de pontuação é usada para definir a ordem de visitação dos nós no grafo por meio de uma fila de prioridade, de modo que o nó com o menor valor de $f(n)$ é visitado primeiro dentre os demais. A fila de prioridade é uma estrutura de dados usada pelo algoritmo para acelerar o tempo computacional no armazenamento dos nós a serem visitados e sobretudo na obtenção do nó com menor valor de $f(n)$. Além disto, a função de pontuação é definida como $f(n) = g(n)$, onde $g(n)$ é uma função de pontuação do caminho, e ela indica o custo de ir a partir do nó inicial até o nó n. Por exemplo, os valores da função $g(n)$ para os nós B e C, na Figura 10, são, respectivamente, 12 e 4.

O algoritmo de Dijkstra é um processo iterativo, no qual cada iteração consiste, resumidamente, em repetir 5 passos até que o nó alvo seja atingido, como ilustra a Figura 11. Adicionando-se o nó inicial à fila de prioridade, os seguintes passos são realizados: a) obter nó da fila de prioridade com menor $f(n)$ e que não tenha sido visitado; b) verificar se n é o nó alvo e encerra a busca, senão continua a busca; c) explorar os nós descendentes de n, onde os nós descendentes são os nós conectados por uma aresta a n; d) computar a distância acumulada do nó inicial aos filhos de n (i.e., definir o valor de $g(n)$); e) adicionar os nós filhos de n à fila de prioridade e marca o nó n como visitado.

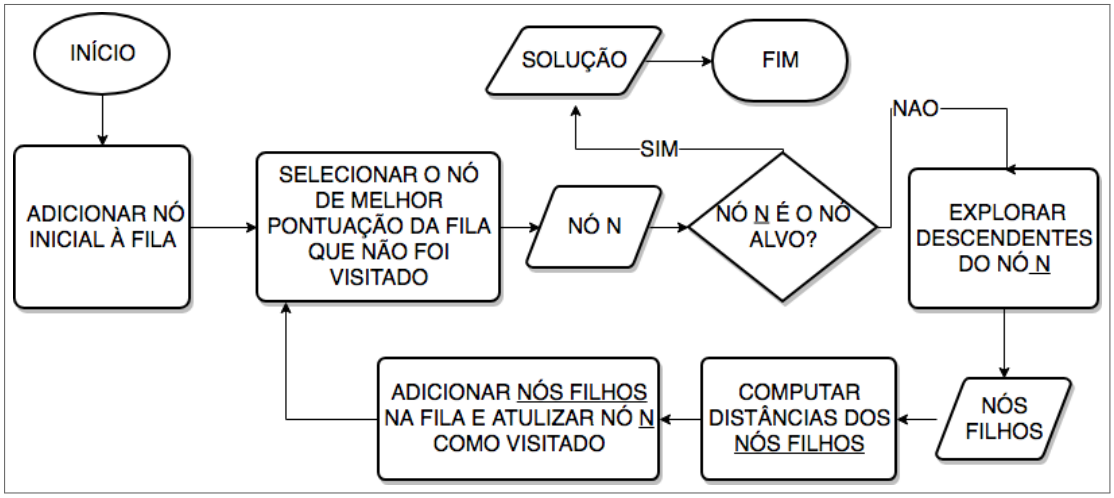


Figura 11 – Fluxograma do algoritmo de Dijkstra.

Modelar o PRC como um problema de busca do caminho mínimo permite aplicar o algoritmo de busca Dijkstra para solucionar otimamente o PRC. O problema de busca do caminho mínimo consiste em encontrar o caminho de menor custo num grafo G conectando um nó inicial n_o até um nó alvo n_a, onde o grafo $G = (N, A)$ possui um conjunto $N = \{n_0, n_1, n_2, \ldots, n_k\}$ de nós e um conjunto A de arestas tal que o custo de cada aresta $\{n_i, n_j\} \in A$ é $c_{ij} \in \mathbb{R}$. Mapeando esse problema ao PRC, n_o e n_a correspondem, respectivamente a configuração inicial da baia e a configuração de baia onde todos os contêineres foram retirados (i.e., a baia está vazia). Os demais nós do grafo G representam todas as demais configurações de baia possíveis, considerando todas as permutações dos contêineres nos espaços ocupáveis da baia. As arestas do grafo equivalem às operações de empilhamento do guindaste (i.e., realocações e retiradas) e o custo de uma aresta corresponde ao tempo de operação do guindaste para realizar a operação, conforme definido pela Equação 4.3, na Seção 4.2. Por fim, a função de pontuação do caminho $g(n)$, utilizada pelo algoritmo Dijkstra, é computada pelo tempo de operação despendido em todas as operações realizadas pelo guindaste para alcançar a configuração da baia, que é representada pelo nó n, a partir do nó inicial n_o.

Embora o algoritmo de Dijkstra, convencionalmente, requeira a entrada de todos os nós e arestas do grafo busca, produzir antecipadamente todos os nós e arestas para instâncias

do PRC é algo inviável computacionalmente. Dado que o número de configurações possíveis para uma baia com n pilhas, r contêineres e m camadas, cresce exponencialmente à medida que n, r, e m aumentam. Isto foi reportado em (KU; ARTHANARI, 2016), por meio da função $P(n, r, m)$ (Equação 4.18) que computa número de configurações possíveis para uma baia. Por exemplo, uma simples baia com 5 pilhas, 8 contêineres e 3 camadas tem o número de 6.249.600 configurações possíveis. Mas, se nesta mesma baia, incrementamos em 1 o número de contêineres (i.e., $r = 9$), o número passa para 48.988.800. Portanto, o algoritmo de busca de caminhos proposto não gera todos os nós e arestas previamente, mas os nós e arestas são gerados a medida que os nós são visitados pelo algoritmo.

$$P(n,r,m) = \left[\binom{n+r-1}{r} - \sum_{j=1}^{\lfloor r/(m+1) \rfloor} (-1)^{j+1} \times \binom{n}{j} \times \binom{n+r_{(j)}-1}{r_{(j)}} \right] \times r! \qquad (4.18)$$
$$\text{onde } r_{(j)} = r - j \times (m+1),\ j = 1, 2, \ldots$$

O algoritmo de busca de caminhos, criado neste trabalho para solucionar otimamente o PRC e baseado no algoritmo Dijkstra, tem seu pseudocódigo descrito pelo Algoritmo 1.

Algoritmo 1: Algoritmo de Busca de Caminhos para solucionar o PRC

Entrada: *baia*, configuração inicial da baia
Saída: solução exata

```
1: solucao ← ∅; fila ← ∅
2: v_no ← nó inicial gerado por meio de baia.
3: adiciona v_no to fila
4: enquanto fila não está vazia faça
5:    v_no ← o nó selecionado de fila
6:    se v_no não foi visitado então
7:       se v_no é o nó alvo então
8:          solucao ← operações efetuadas para alcançar o v_no
9:          retorne solucao
10:      else
11:         v_nos ← ∅;
12:         se ĉ está no topo em v_no então
13:            n ← novo nó obtido pela operação de retirada de ĉ
14:            v_nos ← v_nos ∪ n
15:         else
16:            v_nos ← novos nós obtidos pelas realocações elegíveis.
17:         fim do se
18:         adiciona v_nos à fila
19:         atualiza v_no como visitado
20:      fim do se
21:   fim do se
22: fim do enquanto
23: retorne solucao
```

No início do algoritmo, o caminho mais curto do nó inicial para o nó alvo, ou seja, a solução do problema (*solucao*) e a fila de prioridade ($fila$) são inicializada com um conjunto vazio (linha 1). Em seguida, o nó inicial do grafo de busca é gerado a partir da configuração inicial da baia (*baia*), que é passada com entrada do algoritmo, na sequência este nó é adicionado a fila de prioridade (linhas 2 e 3). O processo de busca itera enquanto houver pelo menos um nó ativo para explorar, ou seja, a fila de prioridade não está vazia (linhas 4–22). Em cada etapa, um nó é selecionado com o objetivo de expandir o grafo de busca. A fila de prioridade seleciona o nó com o menor valor de função de pontuação e o retorna, removendo-o da fila. Então, verifica-se o nó selecionado (v_no) não foi visitado ainda pelo processo de busca (linhas 5–6). Em seguida, se o nó selecionado (v_no) é o nó alvo (ou seja, todos os contêineres foram retirados), a solução é construída a partir de todas as operações efetuadas, desde o nó inicial até o v_no, e em seguida é retornada (linhas 7–9). Caso contrário, um conjunto com os nós descendentes de v_no é gerado por meio do procedimento de expansão e v_no é marcado como visitado (linhas 11–19). Finalmente, quando o nó alvo não pode ser alcançado, o algoritmo retorna uma solução vazia (linha 23). A Figura 12 exibe o fluxograma do Algoritmo 1, nele é apresentando o passo a passo do processo de busca do algoritmo, conforme relatado acima.

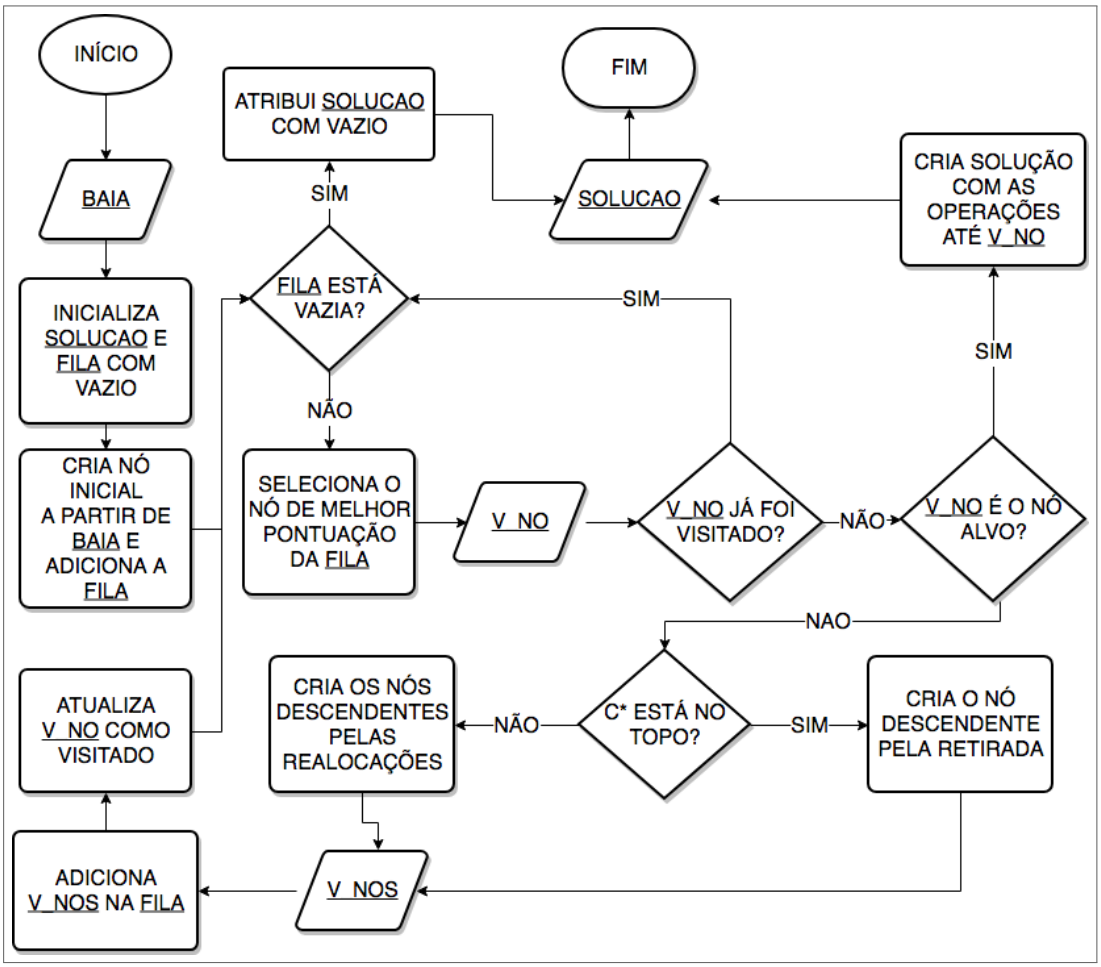


Figura 12 – Fluxograma do Algoritmo de Busca de Caminhos para solucionar o PRC.

O procedimento de expansão de um nó (v_no) funciona da seguinte maneira. O procedimento verifica, para configuração da baia representada pelo nó v_no, se o próximo contêiner a ser retirado, $\hat{c}$, está atualmente situado no topo de uma pilha. E se assim, um novo nó (n) é gerado pela operação de retirada (linha 13). Caso contrário, um conjunto com todas as operações elegíveis de realocação são identificadas de acordo com a classe do PRC a ser abordada (i.e., PRC restrito ou PRC irrestrito). A partir desse conjunto de realocações um conjunto de novos nós é gerado (linha 16). Por último, os nós descendentes de v_no (i.e., v_nos) são adicionados na fila de prioridade ($fila$). Vale destacar que um nó n' poderia estar presente na fila de prioridade mais que uma vez e com diferentes valores de função de pontuação, ou seja, mais de um instância do nó n', dado que vários caminhos podem levar a mesma configuração de baia representada por n'. Portanto, a fim de reduzir o número de instâncias de um nó, apenas uma instância de um nó n' é mantida na fila de prioridade. Para isso, a instância do nó n' somente é substituída na fila de prioridade por uma nova instância, quando o valor da função de pontuação da nova instância é menor que o da instância presente na fila de prioridade.

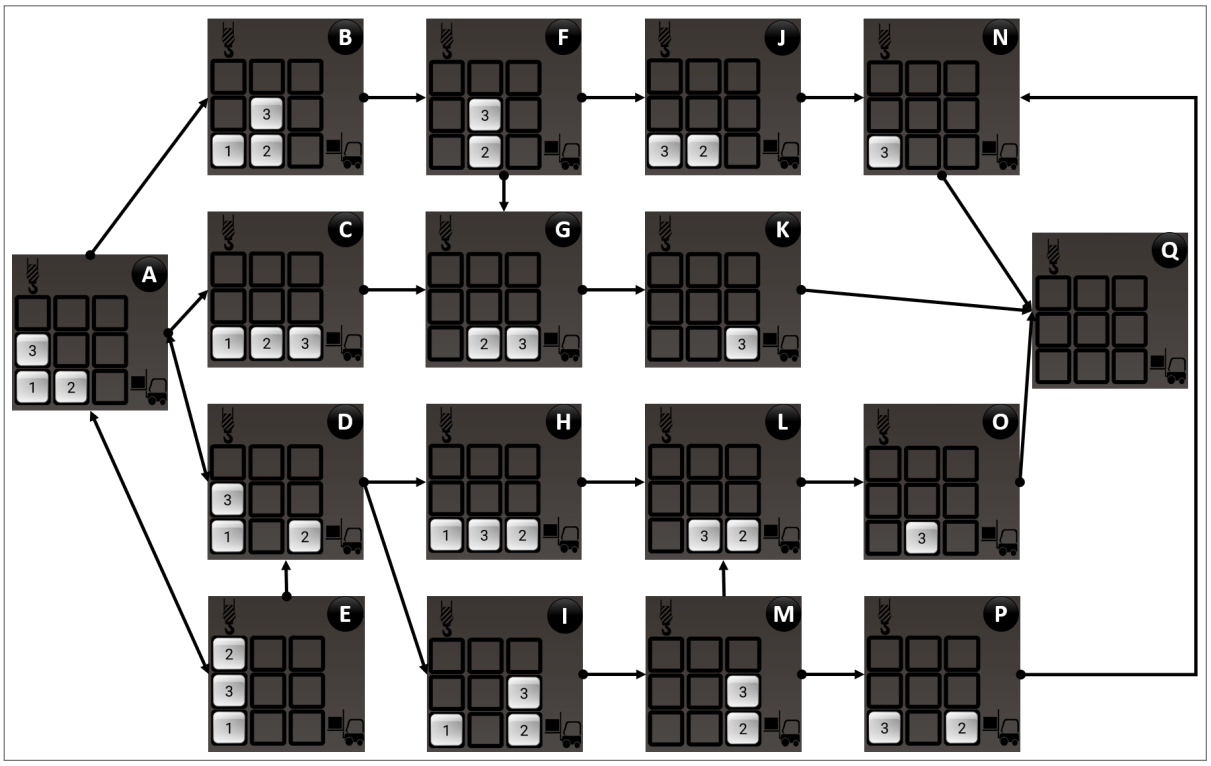


Figura 13 – Demostrações do procedimento de expansão de nós.

A Figura 13 ilustra o procedimento de expansão de nós para o PRC irrestrito. A expansão inicia no nó A, que representa um baia com 3 pilhas, 3 camadas e 3 contêineres, e explora também os nós intermediários até atingir o nó Q, que representa a configuração de baia vazia. Na visita ao nó A, quatro operações são possíveis: (1) realocar o contêiner 3 para pilha 2; (2) realocar o contêiner 3 para pilha 3; (3) realocar o contêiner 2 para pilha 3; e (4) realocar o contêiner 2 para a pilha 1. Estas quatro realocações produzem

respectivamente os nós descendentes B, C, D e E. Em seguida, suponha que o nó E está sendo visitado, e assim mais uma instância do nó D é gerada como descendente do nó E. Caso esta nova instância do nó D possua o valor da função de pontuação menor que o da instância do nó D gerada como descendente do nó A, então a fila de prioridade será atualizada para substituir a instância atual do nó D pela nova instância descendente do nó E. Por fim, note que os nós B, C, G, H, I, J, K, L, M, N, O e P ao serem visitados produzem apenas um nó descendente, e este nó é resultante de uma operação de retirada, visto que naqueles nós o contêiner de maior prioridade da baia está no topo de um pilha.

4.3.3 Algoritmo de Busca A*

O algoritmo de busca *A Star* (A*) (HART; NILSSON; RAPHAEL, 1968), assim como o algoritmo de Dijkstra, também é um algoritmo de busca de caminhos. O algoritmo de busca A* proposto neste trabalho é uma extensão do algoritmo de busca descrito no Algoritmo 1. A extensão consiste em uma alteração na função de pontuação $f(n)$, onde ela agora é definida como $f(n) = g(n) + h(n)$. Note que nessa alteração, a função $h(n)$ foi adicionada com uma parcela da função de pontuação $f(n)$. $h(n)$ é uma função heurística que estima o custo mínimo restante para atingir o nó alvo do grafo de busca a partir do nó n.

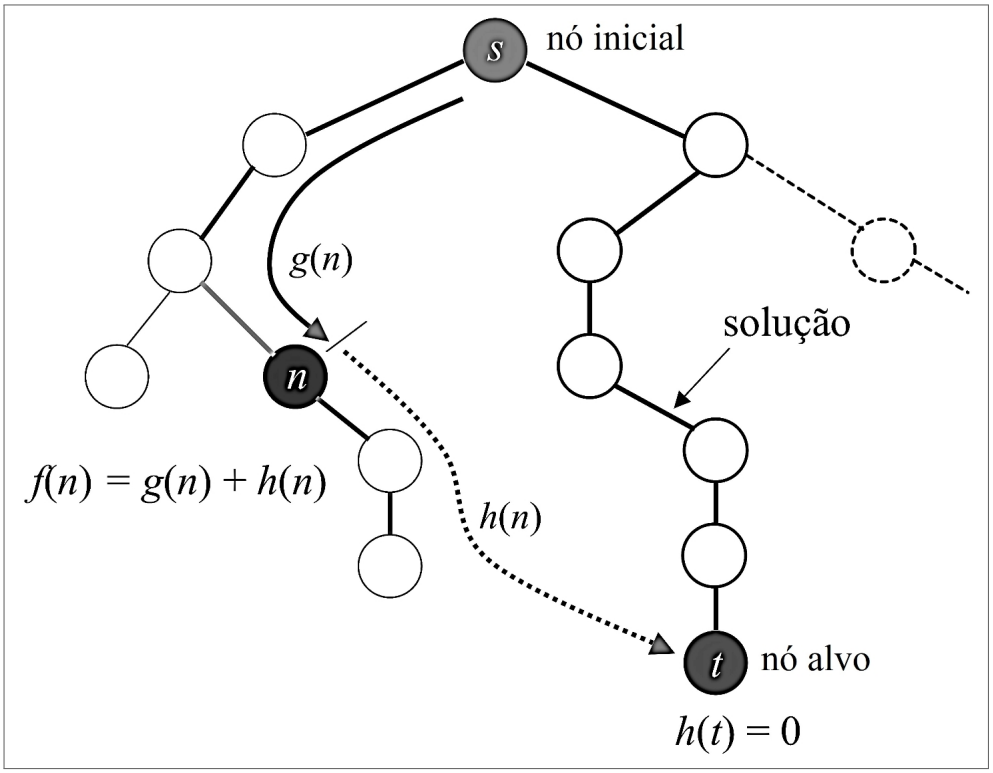


Figura 14 – Função de pontuação $f(n)$ usada no algoritmo de busca A*.

A Figura 14 ilustra como é composta a função de pontuação $f(n)$ para encontrar o caminho mínimo entre o nó inicial s e o nó alvo t. Perceba, que apesar da solução

está no caminho formado pelos descendentes do lado direito do nó s, o algoritmo de busca explorou também os descendentes de s do lado esquerdo, e, em um dado momento, visitou o nó n. Nesse momento, se o algoritmo de busca utilizado fosse o de Dijkstra, a função de pontuação seria o custo de ir a partir do nó inicial até o nó n. i.e., $f(n) = g(n)$. Mas, no caso do algoritmo de busca A*, a função de pontuação consiste no custo do caminho do nó inicial até o nó n mais a estimativa do custo do caminho mínimo do nó n até o nó alvo, i.e., $f(n) = g(n) + h(n)$.

A adoção da função $h(n)$ a função de pontuação $f(n)$ agiliza o processo de busca, pois além de $g(n)$, há outro indicativo para informa que o nó n parece fazer parte do melhor caminho, i.e., $h(n)$. Por exemplo, na Figura 14, suponha que o $g(n)$ para o nó n seja um valor bem pequeno, isso, pela função $f(n) = g(n)$, indicaria que o nó n seria um nó promissor para ser explorado. Contudo, se o valor de $h(n)$ para o nó n é um valor muito grande, o nó n, pela função $f(n) = g(n) + h(n)$, não é mais um nó promissor. Assim, não é necessário perder tempo explorando o nó n, pois ele no melhor caso, segundo $h(n)$, possui um custo muito alto para atingir o nó alvo. Diante disso, para o algoritmo A* é proposta uma função heurística $h(n)$ que estima o menor tempo de operação do guindaste de modo a esvaziar a baia, representada por n.

Antes da função heurística proposta ser apresentada, algumas notações serão introduzidas para melhor compreensão da função. Assim, inicialmente, a baia contém N contêineres rotulados de $1, \ldots, N$, de acordo com os seus respectivos valores de prioridade de saída, tal que c indica a prioridade de um contêiner (i.e., $c \in \{1, \ldots, N\}$) e $\hat{c}$ o próximo contêiner a ser retirado de uma baia, ou seja, o contêiner de maior prioridade. Também, seja $s(c)$ e $t(c)$, respectivamente, a pilha e camada na qual o contêiner com prioridade c está atualmente localizado. Além disso, seja $w'(i, j, k, l)$ (Equação 4.19) a função que computa o tempo de operação que o guindaste consome para realizar qualquer operação de empilhamento. Mas, diferente da função $w(f, i, j, k, l)$ definida pela Equação 4.3 na Seção 4.2, a função $w'(i, j, k, l)$ não contempla o custo para alinhar a garra do guindaste antes de iniciar a operação (i.e., $|f - i|/\nu_2$).

$$w'(i,j,k,l) = \underbrace{|j - \hat{H}|/\nu_4}_{\text{descer garra}} + \underbrace{|j - \hat{H}|/\nu_3}_{\text{elevar contêiner}} + \underbrace{|i - k|/\nu_1}_{\text{mover contêiner}} + \underbrace{|l - \hat{H}|/\nu_3}_{\text{descer contêiner}} + \underbrace{|l - \hat{H}|/\nu_4}_{\text{subir garra}} \tag{4.19}$$

O valor da função $h(n)$ consiste no tempo de operação requerido pelo guindaste para realizar as operações de retiradas sobre todo os contêineres restantes na configuração de baia do nó n (i.e., $\{\hat{c}, \ldots, N\}$), onde o tempo de operação cada operação é calculado por meio da função $w'(i, j, k, l)$. Assim, seja a função heurística $h(n)$ definida pela Equação 4.20, lembrando que a coordenada $(\hat{W}, \hat{H})$ é a coordenada de destino de toda operação de retirada. Note que, em $h(n)$, as operações de realocação não fazem parte do tempo de

operação, mas apenas as operações de retirada, ou seja, mover o contêiner de sua coordenada $(s(c), t(c))$ para fora da baia $(\hat{W}, \hat{H})$. Além do que, nestas operações de retirada não é considerado o tempo para alinhar a garra do guindaste.

$$h(n) = \sum_{c=\hat{c}}^{N} w'(s(c), t(c), \hat{W}, \hat{H}) \tag{4.20}$$

De acordo com o princípio de otimização computacional da busca A*, o algoritmo A* é um método de otimização exato quando a função heurística $h(n)$ é uma função admissível, ou seja, o custo estimado por $h(n)$ é sempre menor ou igual ao custo real para atingir o nó alvo a partir do nó n. Em outras palavras, $h(n)$ deve ser um limite inferior para o custo do problema. Desta forma, o caminho mais curto para o nó alvo é sempre alcançado, por conseguinte, a solução ótima para o problema é retornada.

Entretanto, encontrar uma função heurística para resolver um problema é um desafio, pois ela depende das características do problema. Por exemplo, no problema de encontrar o menor caminho entre cidades, um limite inferior para o custo entre duas cidades pode ser a distância euclidiana (i.e., uma reta) entre as duas cidades. Visto que a reta sempre é a menor distância entre dois pontos. Portanto, encontrar uma função admissível para um problema é encontrar um limite inferior para o custo do problema.

Em vista disso, neste trabalho, é necessário a prova de que a função $h(n)$ proposta é um limite inferior para o tempo mínimo de operação do guindaste entre o nó n e o nó alvo. A prova é dada como se segue.

Lema 3. *$h(n)$ é um limite inferior para o tempo mínimo de operação despendido pelo guindaste para retirar, sequencialmente, todos os contêineres da baia representada pelo nó n.*

Prova [Lema 3]. Seja o tempo mínimo de operação consumido pelo guindaste para retirar, sequencialmente, todos os contêineres da baia n definido como $m(n) = x(n)+y(n)$, onde as duas parcelas $x(n)$ e $y(n)$ são os tempos de operação do guindaste relacionados, respectivamente, às operações de realocação e retirada. Agora, suponha que o problema foi relaxado tal que a restrição sobre a sequência de retirada dos contêineres não exista e o tempo para alinhar a garra do guindaste não é considerado. Desse modo, as realocações não mais necessárias, logo $x(n) = 0$, e o tempo de operação despendido pelo guindaste para efetuar uma operação é definido pela função $w'(i, j, k, l)$, em vez de $w(f, i, j, k, l)$. Assim $m(n)$ é reduzido e passar a ser definido pela Equação 4.21.

$$m(n) = \sum_{c=\hat{c}}^{N} w'(s(c), t(c), \hat{W}, \hat{H}) \tag{4.21}$$

Observe que, por essa relaxação do problema, $m(n)$ torna-se idêntico à função $h(n)$ (Equação 4.20). Este processo é análogo ao traçar uma reta entre duas cidades para se estimar o custo entre elas, uma vez que, para isso, são desconsiderados alguns obstáculos existentes no caminho mínimo entre duas cidades. Nessa analogia, os obstáculos correspondem às restrições do problema que foram descartadas por meio da relaxação, $h(n)$ corresponde ao custo numa reta, (i.e., percurso sem obstáculos) e $m(n)$ representa custo mínimo real do percurso.

Em vista disso, a função $h(n)$ é um limite inferior para o tempo mínimo de operação despendido pelo guindaste para retirar, sequencialmente, todos os contêineres da baia representada pelo nó n, ou seja, $h(n)$ é um limite inferior para $m(n)$. Isto devido ao fato de que $h(n)$ nunca excede $m(n)$ (i.e., $h(n) \leq m(n)$), pois $h(n)$, diferentemente de $m(n)$, não contempla todas as parcelas que formam o tempo mínimo de operação do guindaste. As parcelas de tempo não contempladas em $h(n)$ são: os tempos para efetuar operações de realocação e os tempos para alinhar a garra do guindaste antes de executar uma operação. □

Diante da demostração dessa prova, a função heurística $h(n)$ proposta neste trabalho é uma função admissível. Isto garante que o algoritmo de busca A* proposto encontra o caminho mais curto do nó inicial até o nó alvo. Consequentemente, a solução ótima, aquela com o tempo mínimo de operação do guindaste, é retornada pelo algoritmo de busca A* proposto, solucionando o PRC restrito ou o PRC irrestrito.

4.4 MÉTODOS DE OTIMIZAÇÃO APROXIMADOS

A resolução de problemas de otimização pertencentes a classe de complexidade NP-Difícil, tais como o PRC, por meio de métodos exatos está atrelada a uma elevada carga computacional. Embora algumas instâncias de pequeno a médio porte desses problemas possam ser bem resolvidas por métodos exatos, instâncias de médio a grande porte não se espera que elas possam ser resolvidas por métodos exatos sobre recursos computacionais viáveis. Mesmo assim, os problemas de otimização necessitam de soluções mesmo que não sejam de valor ótimo (de preferência que tenham valor próximo do ótimo), em contrapartida a não obtenção de soluções ótimas por métodos exatos. Diante disso, métodos de otimização aproximados, casualmente chamados de métodos heurísticos, são propostos para atender essa necessidade.

Um método de otimização heurístico é a abordagem de otimização que faz uso de uma heurística para solucionar problemas. O termo heurístico tem sua origem na palavra grega *"heuriskein"* que significa *"encontrar ou descobrir"*. Uma heurística é um conjunto de regras e métodos derivado de conhecimentos intuitivos sobre como encontrar boas soluções para um problema. No método heurístico, não existe a garantia de encontrar soluções ótimas, bem como garantir quão próximas estão as soluções obtidas da solução

ótima. Mas, espera-se, por meio de uma heurística, encontrar soluções boas, mas não necessariamente ótimas. O desafio é produzir, em tempo reduzido, soluções melhores, ou seja, tão próximas quanto o possível da solução ótima. Muitos esforços têm sido feitos nesta direção e heurísticas eficientes e flexíveis foram desenvolvidas para diversos problemas. Além disso, heurísticas mais generalistas visando resolver de forma genérica problemas de otimização, também são desenvolvidas e denominadas de meta-heurísticas.

Uma meta-heurística pode ser vista como uma heurística geral que pode ser aplicada a diferentes problemas de otimização com relativamente poucas modificações para torná-la adaptada para um problema específico. As meta-heurísticas definem um conjunto de regras e métodos gerais para explorar o espaço de soluções, i.e., conjunto de soluções viáveis de um problema. Mas, para isso, elas incorporaram heurísticas especializadas sobre um problema específico de modo a encontrar as soluções do problema investigado de maneira mais eficientemente. As meta-heurísticas têm recebido cada vez mais popularidade e seu uso, em diversas aplicações, mostra sua eficiência e eficácia para resolver problemas grandes e complexos (TALBI, 2009).

Em vista disso, esta seção discorre sobre algoritmos de otimização propostos, baseados em meta-heurísticas, para solucionar o PRC. A Seção 4.4.1, apresenta a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) e descreve um algoritmo de otimização baseado numa versão reativa desta meta-heurística. Em seguida, mais uma meta-heurística é relatada, a *Pilot Method* (PM), e os algoritmos desenvolvidos para a otimização do problema por meio dessa meta-heurística são explanados na Seção 4.4.2.

4.4.1 Algoritmo Reactive GRASP

Esta seção está organizada como segue. Inicialmente, na Seção 4.4.1.1, apresenta-se a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) e sua versão reativa: *Reactive Greedy Randomized Adaptive Search Procedure* (RGRASP). Em seguida, a Seção 4.4.1.2 descreve o algoritmo baseado na meta-heurística RGRASP e proposto para solucionar o PRC. Por fim, as duas fases do algoritmo proposto são explanadas na Seção 4.4.1.3 e Seção 4.4.1.4.

4.4.1.1 Meta-heurística Reactive GRASP

Dentre as meta-heurísticas existentes, a meta-heurística GRASP tem se destacado como uma das mais competitivas em termos da qualidade das soluções alcançadas (RESENDE; RIBEIRO, 2016). A meta-heurística GRASP é um processo iterativo, no qual cada iteração consiste, resumidamente, em repetir dois passos até que certo critério de parada seja atingido, como ilustra a Figura 15. O primeiro passo, a fase construtiva, consiste em gerar uma solução inicial de boa qualidade por meio de uma heurística gulosa aleatorizada. O segundo passo, a fase de busca local, consiste em melhorar a solução inicial por meio de uma busca local e atualizar a melhor solução encontrada até então.

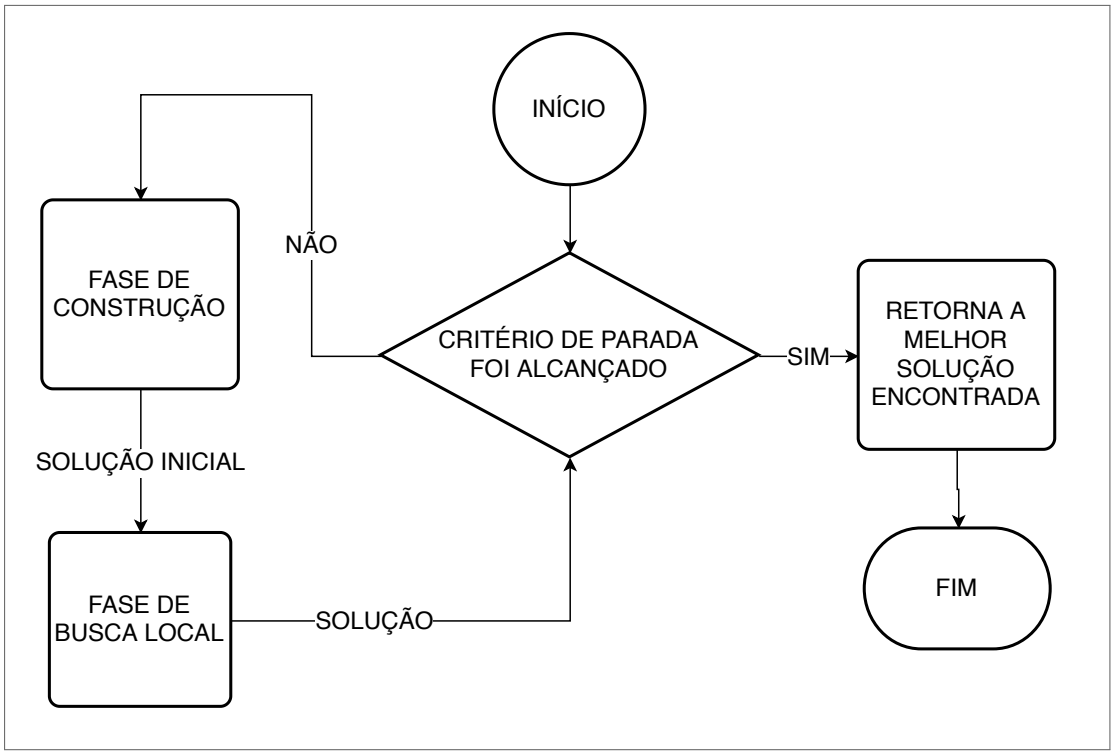


Figura 15 – Fluxograma da Meta-heurística GRASP.

Os algoritmos baseados na meta-heurística GRASP recebem, além dos dados de entrada peculiares ao problema sendo otimizado, os valores dos parâmetros de configuração do algoritmo. Estes valores regulam a execução eficiente do algoritmo segundo os dados de entrada do problema. Desta forma, para cada conjunto de dado de entrada distinto, é necessária a realização da calibração destes parâmetros. Visando eliminar o custo associado à calibração dos parâmetros, são propostos em (RIOS-MERCADO; FERNANDEZ, 2009; DENG; BARD, 2011; FIRMINO et al., 2016), versões reativas da meta-heurística GRASP: a meta-heurística RGRASP. Estas versões propõem mecanismos de autoconfiguração dos parâmetros de entrada que regulam os procedimentos do algoritmo ao longo da execução. No contexto do PRC, a versão reativa melhora a usabilidade da solução proposta, pois não é necessária, uma etapa para determinar os melhores valores para os parâmetros de configuração do algoritmo. O método reativo, adotado no algoritmo RGRASP proposto, utilizou como referência o procedimento definido em (FIRMINO et al., 2016). Mais detalhes desse procedimento reativo serão dados na seção a seguir.

4.4.1.2 Algoritmos GRASP e RGRASP

O algoritmo *Reactive GRASP* (RGRASP), proposto nesta seção, faz uso dos índices de decisão, descritos na Seção 2.3, para obter boas soluções iniciais na primeira fase do algoritmo. Na segunda fase do algoritmo, a solução inicial (gerada pela fase anterior) é melhorada visando reduzir o tempo de operação do guindaste por meio de um procedimento de busca local. Portanto, este algoritmo tem um parâmetro de entrada para

informar qual índice de decisão deve ser usado pelo algoritmo para construir as soluções iniciais (i.e., ξ).

Antes de descrever o algoritmo RGRASP, será apresentando a versão não reativa do algoritmo: o algoritmo GRASP. Este é inicialmente apresentado como uma base para melhor compreensão do algoritmo RGRASP proposto. O Algoritmo 2 descreve o pseudocódigo do algoritmo GRASP. No início do algoritmo, a melhor solução é inicializada com um valor máximo de custo (indicando o pior caso) a ser comparado com a primeira solução encontrada por este algoritmo (linha 1). As iterações do algoritmo GRASP são executadas até que o critério de parada for atendido (linhas 2–7). A variável *baia* é copiada para *baia′* no início de cada nova iteração (linha 3), de modo que a configuração inicial da baia seja a mesma em todas as iterações, pois a configuração da baia é alterada durante a execução dos procedimentos *FaseConstrução* e *FaseBuscaLocal*. Uma solução inicial é obtida através da fase de construção e é armazenada na variável *solucao* (linha 4). Em seguida, durante a fase de busca local, uma nova solução, que é igual ou melhor do que a solução anterior, é obtida e armazenada na variável *solucao* (linha 5). O procedimento *AtualizarSolução* é chamado para comparar a melhor solução corrente (*melhorSolucao*) e a solução obtida na linha anterior *solucao*. Posteriormente, atribui a solução com a menor tempo de operação do guindaste para *melhorSolucao* (linha 6). Recordando que o tempo de operação do guindaste em uma solução é dada pela soma do tempo de operação consumido pelo guindaste em cada operação presente na solução. Ademais, o tempo de operação em cada operação é computado a partir da trajetória do guindaste, conforme definido pela Equação 4.3, na Seção 4.2. Por fim, no final de todas as iterações, a melhor solução até então encontrada, que é essencialmente a sequência de operações (realocações e retiradas) com menor tempo de operação do guindaste, é retornada (linha 8).

Algoritmo 2: Algoritmo GRASP para solucionar o PRC

Entrada: *baia*, configuração inicial da baia
Entrada: ξ, índice de decisão a ser utilizado
Entrada: α, grau de gulosidade.
Saída: melhor solução encontrada.

1: *melhorSolucao* ← valor muito grande
2: **enquanto** critério de parada não for satisfeito **faça**
3: *baia′* ← *baia*
4: *solucao* ← FaseConstrução(*baia′*, ξ, α)
5: *solucao* ← FaseBuscaLocal (*baia′*, *solucao*)
6: AtualizarSolução(*solucao*, *melhorSolucao*)
7: **fim do enquanto**
8: **retorne** *melhorSolucao*

Uma vez definido o algoritmo GRASP, o algoritmo RGRASP é apresentado a seguir. Seu pseudocódigo é dado no Algoritmo 3, que é similar ao Algoritmo 2. Eles diferem apenas sobre o grau de gulosidade α, o qual era previamente obtido como um parâmetro

de entrada, agora é calculado pelo procedimento *ObterGrauGulosidade*, que calcula o valor reativamente.

Algoritmo 3: Algoritmo Reactive GRASP para solucionar o PRC

Entrada: *baia*, configuração inicial da baia
Entrada: ξ, índice de decisão a ser utilizado
Saída: melhor solução encontrada.

```
1: melhorSolucao ← valor muito grande
2: enquanto critério de parada não for satisfeito faça
3:    baia' ← baia
4:    α ← ObterGrauGulosidade
5:    solucao ← FaseConstrução(baia', ξ, α)
6:    solucao ← FaseBuscaLocal (baia', solucao)
7:    AtualizarSolução(solucao, melhorSolucao)
8: fim do enquanto
9: retorne melhorSolucao
```

O procedimento *ObterGrauGulosidade* funciona da seguinte maneira. Inicialmente, é atribuído um valor inicial entre {0,1} para o grau de gulosidade α. Depois, é definido uma ação inicial que pode ser {incrementar (I) ou decrementar (D)} e um valor v para incremento ou decremento. A cada iteração do algoritmo, há uma verificação para avaliar se solução atual, obtida pelo procedimento *FaseConstrução*, melhorou em relação a melhor solução obtida até o momento pelo procedimento *FaseConstrução*. Em caso afirmativo, é realizado ação corrente, caso contrário é realizado a ação oposta.

Como um exemplo, considere que, inicialmente, $\alpha = 0$, ação atual é I e $v = 0.05$. Na iteração consecutiva i_1, foi verificada uma melhoria na solução gerada, então, foi realizado a ação atual computando $\alpha = \alpha + v \Rightarrow 0 + 0.05 \Rightarrow 0.05$. Na i_2, também foi verificada uma melhoria, então α é incrementado, tal que agora $\alpha = 0.10$. O mesmo acontece na i_3 e i_4, então α é incrementado duas vezes, sendo agora igual a $\alpha = 0.20$. Entretanto, em i_5, a solução gerada piorou, então foi realizado a ação oposta de modo que $\alpha = \alpha - v \Rightarrow 0.20 - 0.05 \Rightarrow 0.15$. Na i_6 e i_7, foram verificadas melhorias nas soluções geradas. Portanto, continua a ação do tipo D, com α decrementado duas vezes, sendo agora igual a $\alpha = 0.05$. Todavia, na i_8, foi gerada uma solução pior, então realizou-se a ação oposta, incrementando o valor de modo que $\alpha = \alpha + v \Rightarrow 0.05 + 0.05 \Rightarrow 0.10$. Assim, comporta-se o algoritmo ao longo de todas as iterações ajustando o valor de α de acordo com a qualidade das soluções geradas em tempo de execução.

Vale ressaltar que no procedimento *ObterGrauGulosidade* uma solução é dita melhor que outra, se o número de operações de realocações de uma solução for menor que o número de operações de realocações da outra. Uma vez que as soluções são geradas na fase de construção por meio dos índices de decisão e estes tem por objetivo direcionar a construção de soluções com menor número de realocações.

4.4.1.3 Fase de Construção

Em cada iteração do algoritmo RGRASP, o procedimento *FaseConstrução*, descrito pelo Algoritmo 4, é chamado. Este procedimento objetiva gerar uma boa solução inicial para ser melhorada na fase seguinte da meta-heurística RGRASP.

Algoritmo 4: Procedimento da Fase de Construção do Algoritmo RGRASP

Entrada: *baia*, configuração inicial da baia
Entrada: ξ, índice de decisão a ser utilizado
Entrada: α, grau de gulosidade.
Saída: solução construída.

```
1:  solucao ← ∅
2:  enquanto baia não está vazia faça
3:    se ĉ está no topo então
4:      operacao ← retirada(baia)
5:    else
6:      LRC ← CriarLRC(baia, ξ, α)
7:      se LRC não está vazia então
8:        operacao ← SelecionarAleatoriamente(LRC, baia)
9:      else
10:       solucao ← ∅
11:       retorne solucao
12:     fim do se
13:   fim do se
14:   Adiciona(operacao, solucao)
15: fim do enquanto
16: retorne solucao
```

No início do procedimento, a solução é inicializada com um conjunto vazio de operações (linha 1). As iterações do algoritmo são realizadas enquanto a baia (*baia*) não estiver vazia (linhas 2-15). A cada início de iteração é verificado se o próximo contêiner a ser retirado, $\hat{c}$, está atualmente situado no topo de uma pilha (linha 3). Em caso positivo, o contêiner $\hat{c}$ é retirado e, por conseguinte, uma operação de retirada é produzida (linha 4). Caso contrário, o contêiner situado no topo da pilha, onde está o $\hat{c}$, precisa ser realocado para o topo de outra pilha. Como pode haver várias opções de realocação, é definida uma Lista Restrita de Candidatos (LRC). Essa lista é formada pelas melhores operações de realocação indicadas pelo índice (ξ) adotado. Mais detalhes sobre a LRC serão dados adiante, na descrição do procedimento *CriarLRC*.

Subsequentemente, é verificado se existem realocações na LRC para serem adicionados à solução (linha 7). Caso exista algum realocação na LRC, o procedimento *SelecionarAleatoriamente* é chamado para realizar a escolha aleatória de uma realocação na LRC. Esta realocação é executada e armazenado na variável *operacao* (linha 8). Essa aleatoriedade é o diferencial do RGRASP em relação às abordagens gulosas, nas quais a solução é construída escolhendo o melhor candidato em cada etapa da solução. Uma vez que, no

RGRASP, a escolha dos candidatos (i.e., operações) que compõem a solução é feita de forma gulosa aleatorizada, ou seja, de forma gulosa, mas também aleatória. Desta forma, são geradas soluções de boa qualidade, devido à característica gulosa, e diversificadas devido à característica aleatória. Se não há realocações na LRC, isto significa que não existem mais operações para construir à solução e, portanto, não existe uma solução. Assim, *solucao* é atribuída com um conjunto vazio de operações e é retornada (linha 11). No fim de cada iteração, a operação gerada na iteração (*operacao*) é adicionada à solução (linha 14). Por fim, quando a baia está vazia, a solução construída é retornada (linha 16).

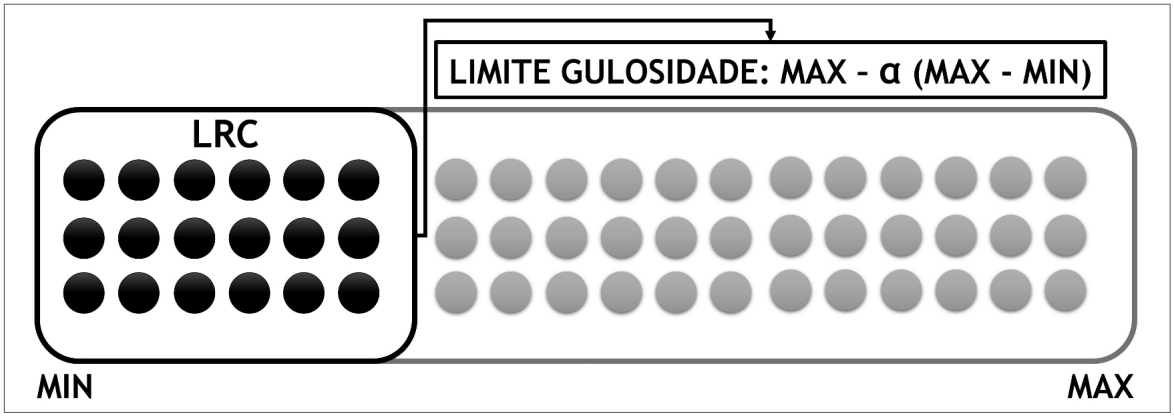


Figura 16 – Exemplo da Lista Restrita de Candidatos - LRC.

O procedimento *CriarLRC* inicializa a LRC que será retornada como uma lista vazia de operações. Em seguida, os valores do índice de decisão para cada realocação elegível são computados. Estes valores são computados de acordo como o índice (ξ) adotado, e o valor máximo e o valor mínimo dentre todos estes valores são atribuídos, respectivamente, a max e min. Os valores de máximo e mínimo são usados para computar o limite de gulosidade, dado por: $LimiteGulosidade\,(\alpha) = max - \alpha\,(max - min)$, onde α é o grau de gulosidade. Este limite de gulosidade define os elementos que irão compor a LRC. Todas as operações de realocação elegíveis são examinadas a fim de que realocações sejam adicionadas à LRC, somente se o valor do índice de decisão correspondente não é maior que o limite de gulosidade. Por fim, o procedimento encerra e retorna a LRC construída.

O processo de obtenção da LRC é ilustrado graficamente na Figura 16, na qual os extremos da lista representam os valores máximo e mínimo encontrados. Os itens da esquerda são os elementos que possuem valores entre o mínimo encontrado e o limite de gulosidade. Em resumo, as operações que comporão a LRC, são aquelas cujos valores estão entre o valor mínimo e o limite de gulosidade.

A Figura 17 exibe uma ilustração do procedimento da fase de construção, relatado acima, para um melhor compreensão do seu funcionamento. Nesta ilustração, o procedimento é aplicado a uma instância do PRC com 6 contêineres, 3 pilhas e altura máxima de 3 contêineres (i.e., $W = 3$ e $H = 3$), e as velocidades do guindaste são $\nu_1 = 1$, $\nu_2 = 2$, $\nu_3 = 1$ e $\nu_4 = 2$. Ademais, $RL(c, s, t)$ indica a operação de realocação do contêiner c da

pilha s para a pilha t e $RT(c, s, t)$ representa a operação de retirada do contêiner c da pilha s para a pilha t, onde t é a posição horizontal de destino de toda retirada, i.e., $t = W + 1$. Além disso, o grau de gulosidade (α) é igual a zero, caracterizando uma abordagem de seleção aleatória, e o índice de decisão usado para eleger as realocações é o RI.

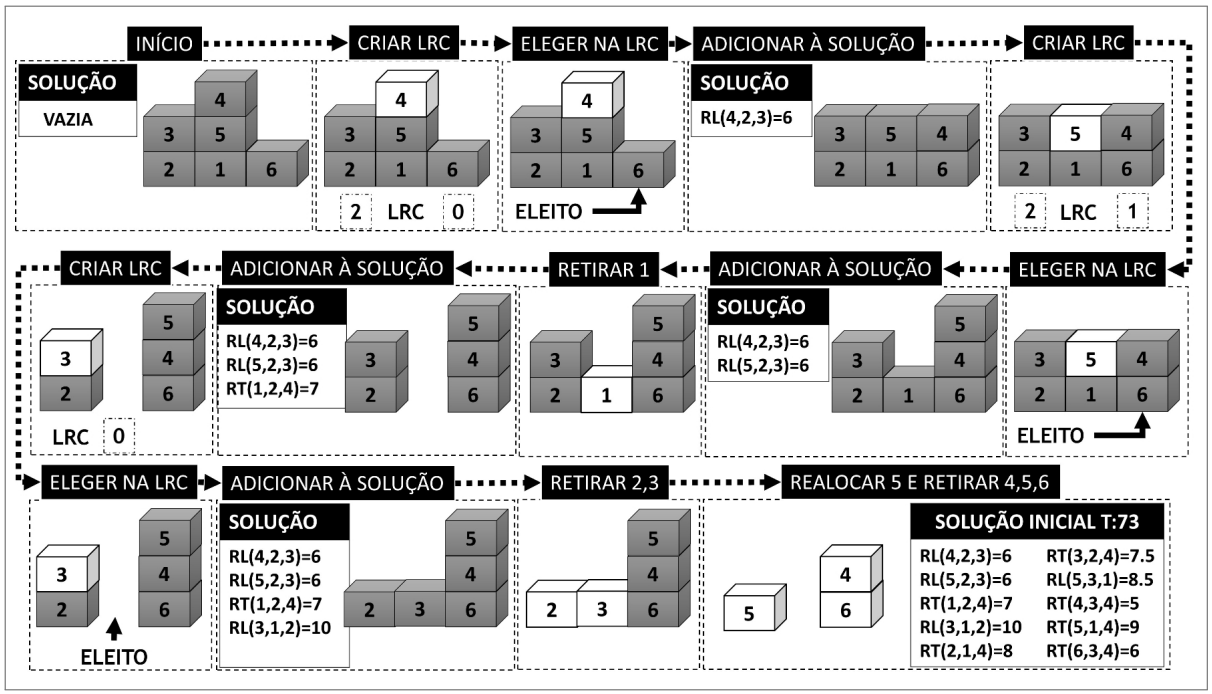


Figura 17 – Demonstração da Fase de Construção.

Na ilustração (Fig. 17), inicialmente, a solução está vazia e a baia não está vazia. Note que o contêiner 4, localizado no topo da segunda pilha, precisa ser realocado para recuperar o contêiner 1. Então, a LRC é definida com as realocações do contêiner 4 para a primeira pilha e para a terceira pilha de acordo com o índice de decisão RI. Em seguida, a realocação do contêiner 4 para a terceira pilha é selecionada aleatoriamente e adicionada à solução. Essa realocação resulta em um tempo de operação igual a 6, i.e., $RL(4, 2, 3) = 6$. Posteriormente, o contêiner 5, no topo da segunda pilha, precisa ser realocado para recuperar o contêiner 1. A LRC é definida, e, então, a realocação do contêiner 5 para a terceira pilha é selecionada aleatoriamente e adicionada à solução. Neste ponto, o contêiner 1 está disponível e sua retirada é adicionada à solução. Depois, o contêiner 3, no topo da primeira pilha, precisa ser realocado para recuperar o contêiner 2. A LRC é criada e a realocação do contêiner 3 para a segunda pilha é selecionada aleatoriamente e adicionada à solução. Neste momento, os contêineres 2 e 3 estão disponíveis e, portanto, suas retiradas são adicionadas à solução. Por último, quatro operações são realizadas e adicionadas à solução: o contêiner 5 é realocado para recuperar o contêiner 4, e sequencialmente os contêineres 4, 5 e 6 são recuperados. No final da fase de construção, a solução construída consiste em 4 realocações e 6 recuperações, resultando em um tempo de operação do guindaste igual a 73 unidades de tempo.

4.4.1.4 Fase de Busca Local

Esta segunda fase do algoritmo RGRASP objetiva por refinar a solução obtida na fase de construção, a fim de encontrar soluções melhores. Dada uma solução *sol*, a busca por melhores soluções é feita por meio da busca por soluções vizinhas, de modo a encontrar soluções melhores que *sol* na sua vizinhança.

A técnica de busca local utilizada, neste trabalho, consiste em desfazer sequencialmente as operações em *sol*, de modo a substituir a operação desfeita por uma outra operação de menor tempo de operação e, portanto, gerar uma nova solução de menor tempo de operação do guindaste. Durante o processo de substituição, a substituição de uma operação requer ajustes nas operações subsequentes (já desfeitas), uma vez que a configuração da baia foi alterada pela substituição. Por esta razão, critérios são adotados na escolha das substituições, a fim de reduzir a quantidade de ajustes, evitar reajustes e não gerar uma solução com tempo de operação pior que *sol* por meio da substituição.

O pseudocódigo do procedimento de busca local é dado no Algoritmo 5. Inicialmente, uma lista de pilhas visitadas (*pilhasVisitadas*) é inicializada com vazio. Esta lista é utilizada para armazenar, ao longo das iterações, as pilhas de destino associadas as operações de realocação já visitadas pelo algoritmo. O objetivo desta lista é evitar reajustes, ou seja, evitar a substituição por uma realocação cuja pilha de destino já foi previamente visitada. Tendo em mente que uma pilha visitada indica que uma realocação para esta pilha já foi examinada e esta foi identificada como a melhor realocação para esta pilha. Ademais, todas as operações subsequentes a esta realocação também estão ajustadas para estabelecer a sequência de operações a parti dela. Desta maneira, não se substitui uma realocação por outra realocação cuja pilha de destino seja uma pilha já visitada pelo algoritmo.

Algoritmo 5: Procedimento da Fase de Busca Local do Algoritmo RGRASP

Entrada: *baia*, configuração atual da baia
Entrada: *sol*, solução construída na fase anterior.
Saída: solução igual ou melhor que (*sol*)

```
1:  pilhasVisitadas ← ∅
2:  para todo operacao em sol, em ordem inversa faça
3:    Desfazer(operacao, baia)
4:    se operacao é uma realocação então
5:      novaOperacao ← Procurar(baia, operacao, pilhasVisitadas)
6:      se novaOperacao existe então
7:        Substituir&Atualizar(operacao, novaOperacao, sol, baia)
8:        Inserir(novaOperacao, pilhasVisitadas)
9:      else
10:       Inserir(operacao, pilhasVisitadas)
11:     fim do se
12:   fim do se
13: fim do para
14: retorne sol
```

Todas as operações da solução são examinadas em ordem inversa, isto é, desde a última operação até à primeira operação realizada pelo guindaste para esvaziar a baia (linhas 2-13). A cada iteração, a operação examinada (*operacao*) é desfeita e a configuração da baia é alterada para o estado anterior à execução da operação *operacao* (linha 3). Neste ponto, há uma busca por uma nova operação de realocação que satisfaça os seguintes critérios (linha 5):

1. Seja diferente de *operacao*;
2. A sua pilha de destino não foi visitada, ou seja, não está em $pilhasVisitadas$;
3. Não produz bloqueios, ou seja, na pilha de destino, todos os contêineres possuem prioridade menor ao contêiner a ser realocado;
4. A sua substituição implica em uma nova solução cujo tempo de operação do guindaste é menor que o da solução anterior à substituição.

Quando uma nova operação (*novaOperacao*) que satisfaz todas as condições acima é encontrada, a solução é atualizada com a substituição da operação corrente (*operacao*) pela nova operação encontrada. A substituição altera a coordenada do contêiner movimentado e, assim, a próxima operação subsequente e a primeira operação subsequente que manipula o mesmo contêiner tornam-se inválidas, uma vez que o contêiner é colocado em uma nova coordenada. Desta forma, estas operações subsequentes são ajustadas às novas coordenadas e os seus tempos de operação são atualizados (linha 7). Além disso, a pilha de destino de *novaOperacao* é marcada como visitada (linha 8). Se uma nova operação não é encontrada, a operação corrente (*operacao*) também terá sua pilha de destino marcada como visitada (linha 10). Isto significa que esta operação é a melhor até agora e não pode mais ser ajustada em substituições futuras. Finalmente, após todas as operações terem sido examinadas, o procedimento retorna a melhor solução encontrada durante a busca local (linha 14).

Visando também uma melhor compreensão do funcionamento deste procedimento, a Figura 18 exibe uma ilustração do procedimento da fase de busca local. Nesta ilustração, o procedimento recebe como entrada a solução que foi construída pela fase de construção no exemplo ilustrado na Figura 17. Inicialmente, as operações presentes na solução são desfeitas, da última operação à primeira operação, até que uma operação de realocação seja desfeita. Então, a realocação do contêiner 5 é alcançada, i.e., $RL(5, 3, 1)$. Neste ponto, uma nova operação de realocação é encontrada, i.e., $RL(5, 3, 2)$. Esta nova realocação satisfaz todas as condições, incluindo a redução do tempo de operação, pois esta nova realocação tem tempo de operação igual a 19,5 enquanto a realocação atual tem o tempo de operação igual a 22,5. Assim, a realocação $RL(5, 3, 1)$ é substituída pela realocação $RL(5, 3, 2)$, a segunda pilha é marcada como visitada e os operações subsequentes ($RT(4, 3, 4)$ e

$RT(5, 1, 4)$) são ajustadas. Neste momento, o tempo de operação do guindaste na solução mudou de 73 para 70.

Posteriormente, as operações continuam sendo desfeitas até que uma operação de realocação seja desfeita, i.e., a realocação $RL(3, 1, 2)$ é alcançada. No entanto, novas operações não são encontradas para substituir $RL(3, 1, 2)$, já que a terceira pilha está na altura máxima. Desta forma, a primeira pilha é marcada como visitada e as operações continuam sendo desfeitas até que a realocação $RL(5, 2, 3)$ seja atingida. Também, neste ponto, não são encontradas operações para substituição, já que a primeira pilha e a segunda pilha estão marcadas como visitadas. Em seguida, a terceira pilha é marcada como visitada e a última realocação $RL(4, 2, 3)$ é desfeita. Novamente, não foram encontradas operações, então $RL(4, 2, 3)$ não foi substituída, visto que todas as pilhas já foram visitadas. No final, todas as operações foram desfeitas e examinadas e a solução melhorada por esse procedimento é retornada.

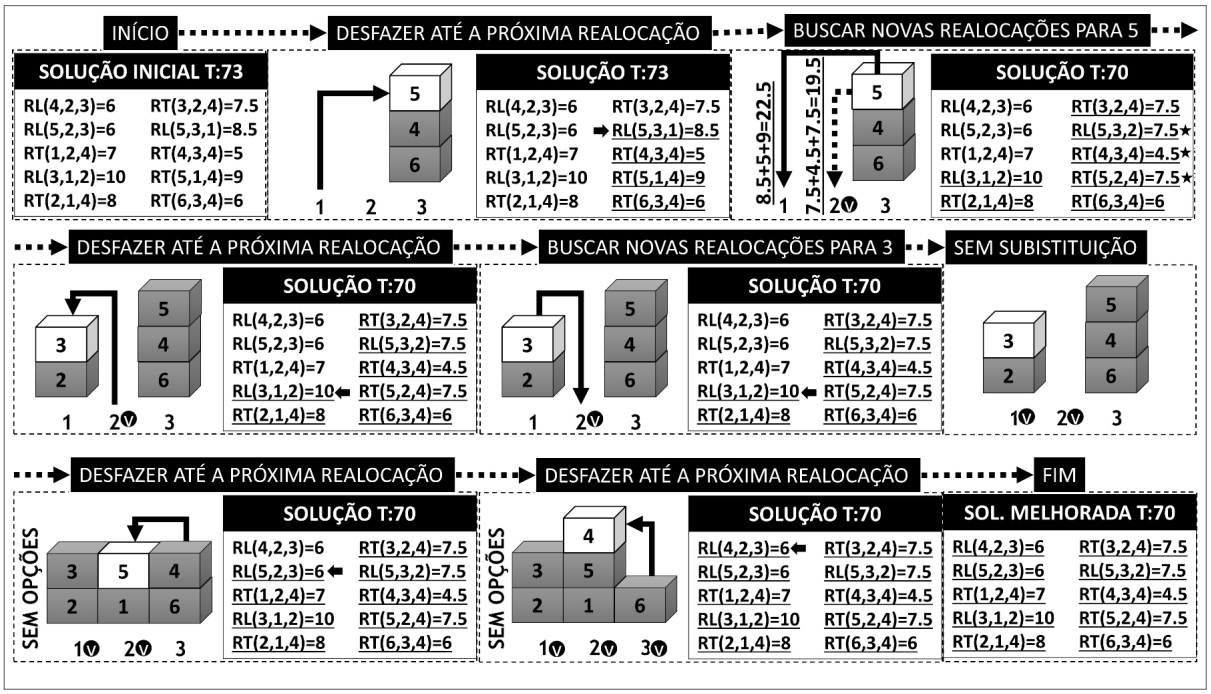


Figura 18 – Demonstração da Fase de Busca Local.

Vale ressaltar que os dois procedimentos do algoritmo RGRASP proposto, fase de construção e fase de busca local, consideram apenas operações de realocação restritas, ou seja, o algoritmo RGRASP aborda apenas o PRC restrito. A razão disso está no fato que o algoritmo RGRASP adota, para a construção das soluções, os índices de decisão definidos na Seção 2.3, e estes foram projetados para o PRC restrito. Portanto, como o algoritmo RGRASP proposto aborda apenas o PRC restrito, na próxima seção, são apresentados algoritmos de otimização que solucionam o PRC restrito e também o PRC irrestrito.

4.4.2 Algoritmo Pilot Method

A *Pilot Method* (PM) é uma meta-heurística que incorpora uma heurística construtiva gulosa h, chamada de *pilot heuristic*, e utiliza essa heurística para avaliar e tomar decisão sobre quais elementos devem compor a solução para o problema. Esta meta-heurística é utilizada para solucionar diversos problemas de otimização, incluindo problemas de complexidade NP-Difícil, e na resolução destes problemas tem apresentados bons resultados (VOßS; FINK; DUIN, 2005). A meta-heurística PM pode ser vista como um processo iterativo no qual cada iteração consiste, resumidamente, em repetir três tarefas até que um determinado critério de parada seja alcançado. A primeira tarefa consiste em avaliar, no estado atual de construção, todos os elementos disponíveis para construir a solução. Cada elemento é avaliado pelo valor obtido na execução completa da heurística h para construir uma solução para o problema, partindo de uma solução parcialmente construída com este elemento e outros elementos escolhidos em etapas anteriores no processo de construção. A segunda tarefa consiste em selecionar o melhor elemento avaliado e adicioná-lo a solução, como resultado, avança-se uma etapa no processo de construção e um novo estado é gerado e definido como o estado atual de construção. A última tarefa é atualizar e armazenar a melhor solução encontrada até o momento.

Neste trabalho de pesquisa, aplica-se a meta-heurística PM para solucionar o PRC. Nesse intuito, realiza-se uma analogia do processo de construção da meta-heurística PM com o processo de navegação num grafo de busca do PRC, definido na Seção 4.3.2. Nesta analogia, cada nó no grafo de busca coincide com um estado de construção na meta-heurística PM, e, assim, o nó inicial e o nó alvo no grafo de busca correspondem, respectivamente, ao estado inicial e final de construção. Além disso, um novo estado de construção é gerado por meio de uma operação de empilhamento (realocação ou retirada), assim como um nó descendente é gerado no grafo de busca. Por fim, a operação de empilhamento condiz ao elemento construtivo na meta-heurística PM e um estado de construção engloba a configuração atual da baia e todos os elementos (i.e., as operações) efetuados para atingir essa configuração a partir da configuração inicial da baia.

O algoritmo *Pilot Method* (PM), baseado na meta-heurística PM e proposto para solucionar o PRC restrito e o PRC irrestrito, tem o seu pseudocódigo descrito pelo Algoritmo 6. Inicialmente, a melhor solução (*melhorSolucao*) e o estado inicial (*estado*) são inicializados, respectivamente, com um conjunto vazio e a configuração inicial da baia (linha 1). As iterações do algoritmo PM são executadas até que o critério de parada for atendido (linhas 2–16). No início de cada iteração, o procedimento *retirarTodos* é chamado para retirar todos o contêineres da baia que podem ser retirados da baia sem a necessidade de uma realocação. Por meio deste procedimento, um novo estado de construção é gerado resultante de todas as operações de retiradas efetuadas nesse procedimento. Então esse novo estado é atribuído como estado de construção atual (linha 3). Em seguida, as variáveis para armazenar o melhor descendente (*melhordescendente*) do estado atual de constru-

Algoritmo 6: Algoritmo Pilot Method para solucionar o PRC

Entrada: *baia*, configuração inicial da baia
Saída: melhor solução encontrada

```
1: melhorSolucao ← ∅; estado ← baia
2: enquanto critério de parada não for satisfeito faça
3:    estado ← retirarTodos(estado)
4:    melhordescendente ← ∅; melhorcusto ← valor muito grande
5:    para todo d ∈ descendentes(estado) faça
6:       solucao ← heuristica(d)
7:       se custo(solucao) < melhorcusto então
8:          melhordescendente ← d
9:          melhorcusto ← custo(solucao)
10:         se melhorSolucao = ∅ ou melhorcusto < custo(melhorSolucao) então
11:            melhorSolucao ← solucao
12:         fim do se
13:      fim do se
14:   fim do para
15:   estado ← melhordescendente
16: fim do enquanto
17: retorne melhorSolucao
```

ção (*estado*) e seu respectivo custo (*melhorcusto*) são inicializadas, respectivamente, com vazio e com um valor máximo de custo (indicando o pior caso) a ser comparado com o custo do primeiro descendente de *estado*.

Examinando o estado atual (*estado*), identifica-se todas as operações de realocação elegíveis para a configuração de baia associada a esse estado. Vale pontuar, que nesse ponto, se o PRC restrito está sendo abordado, apenas as realocações restritas são consideradas. Depois, com esse conjunto de operações identificadas, todos os possíveis estados descendentes de *estado* são gerados e avaliados usando o procedimento *heuristica* (linhas 5-14). Este procedimento retorna uma solução composta por todas as operações selecionadas desde o estado inicial até o estado de construção d e, mais as operações encontradas pela heurística (i.e., *pilot heuristic*) para solucionar a configuração de baia associada ao estado d. O descendente de d cuja solução possui o menor tempo de operação do guindaste (i.e., *custo*(solucao)) é definido como o melhor descendente (linha 8). Caso o tempo de operação da solução associada ao melhor descendente de d seja o menor custo encontrado até o momento, então a melhor solução encontrada (i.e., *melhorSolucao*) é atualizada com a solução do melhor descendente de d. O algoritmo termina quando qualquer um dos critérios de parada a seguir é atendido. O estado final é alcançado (i.e., quando *estado* é vazio), ou um limite de tempo de execução é atingido, ou um número máximo de iterações consecutivas sem ganhos na qualidade da melhor solução é atingido. Finalmente, a melhor solução encontrada, ou seja, a sequência de operações (realocações e retiradas) com menor tempo de operação do guindaste, é retornada (linha 17).

A Figura 19 exibe o fluxograma do Algoritmo 6, nele é apresentando o passo a passo do processo de construção de soluções no algoritmo PM, conforme relatado acima. Um dos passos do algoritmo é aplicar uma heurística, i.e., a *pilot heuristic*, para avaliar os descendentes do estado atual. Contudo, essa heurística não foi apresentada. Portanto, a heurística proposta para ser incorporada ao algoritmo PM é descrita na próxima seção.

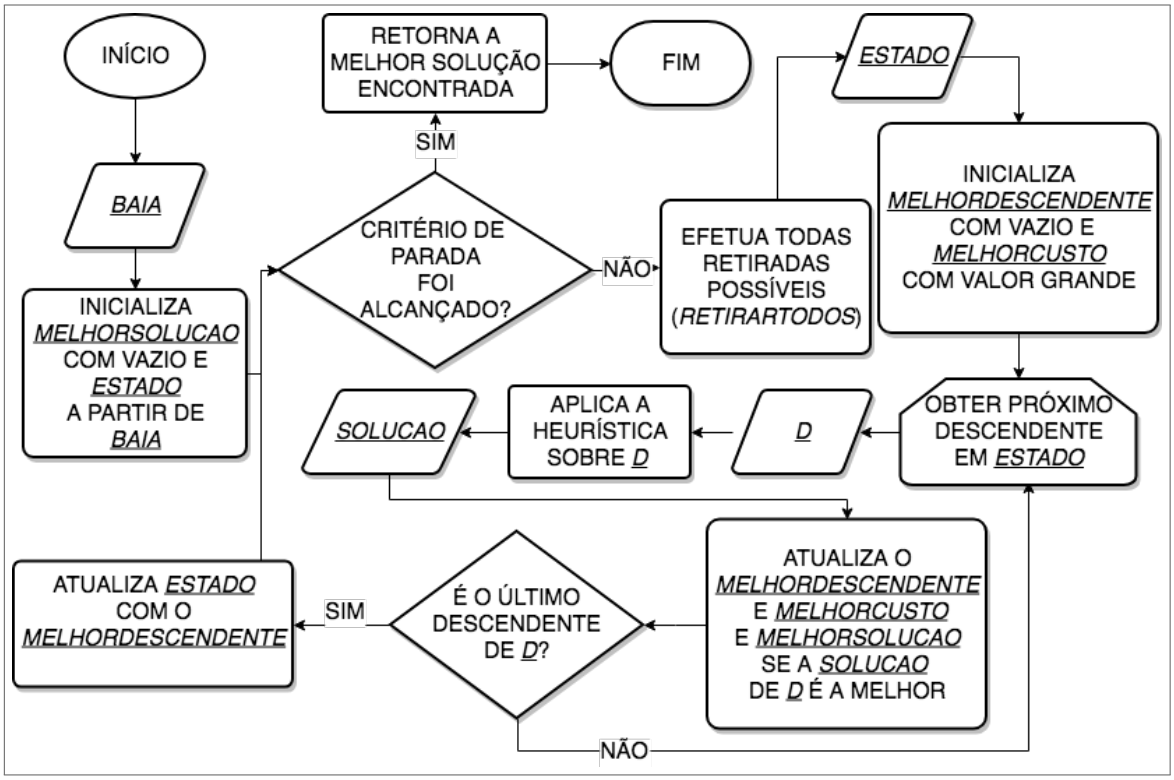


Figura 19 – Fluxograma do Algoritmo PM para solucionar o PRC.

4.4.2.1 Heurística Tríade

Inicialmente, antes que seja descrita a heurística proposta para solucionar o PRC, como também para ser incorporada ao algoritmo PM, algumas notações e termos serão recordados e introduzidos para uma melhor compreensão da heurística.

Uma baia possui um conjunto S de pilhas e a cardinalidade desse conjunto é dada por W, i.e., o número de pilhas na baia. Quando um contêiner é retirado da baia, sua posição horizontal de destino é $W + 1$, pois as retiradas ocorrem pelo canto superior direito da baia. Esta posição é denominada *posição de saída* dos contêineres, e a distância de uma pilha s (i.e., $d(s)$) para a posição de saída é dada por $d(s) = |p(s) - (W + 1)|$, onde $p(s)$ é a posição da pilha s na baia, tal que $p(s) \in \{1, \ldots, W\}$. Além disso, $\check{c}$ e $\hat{c}$ denotam, respectivamente, o contêiner com a prioridade mais baixa e o contêiner com a prioridade mais alta que estão presentes na baia. A pilha que contém o contêiner $\hat{c}$ é denominada *pilha fonte*, de modo que o contêiner localizado no topo desta pilha é denotado por $\vec{c}$. Toda a pilha, cuja altura é menor que altura máxima da baia e é diferente da pilha fonte, é

considerada uma *pilha elegível*, ou seja, pode ser o destino de uma operação de realocação. Sobre as prioridades dos contêineres, o valor da prioridade de um contêiner c é dada por $v(c)$, e $h(s)$ denota o valor da prioridade mais alta presente em uma pilha s, caso a pilha s esteja vazia, $h(s) = v(\check{c}) + 1$. Por fim, um *bloqueio* é o evento onde um contêiner de prioridade mais baixa está acima de um contêiner de prioridade mais alta. Assim, um contêiner bloqueador é aquele que está bloqueando outro contêiner na baia.

A heurística proposta neste trabalho é chamada de *Tríade*. Esta heurística constrói soluções para o PRC por meio de três critérios, aplicados a uma configuração de baia, que determinam quais operações de empilhamento irão compor a solução do problema. Estes três critérios são apresentados abaixo.

Critério 1: Caso existam pilhas elegíveis $s \in S$ tais que $h(s) > v(\vec{c})$, então seleciona a pilha s com o menor $d(s)$ para realocar o contêiner $\vec{c}$, que está impedindo o acesso ao contêiner de maior prioridade da baia. Neste critério, a estratégia de realocar o contêiner $\vec{c}$ para outra pilha é dupla. A primeira consiste em mover o contêiner $\vec{c}$ para uma pilha sem gerar realocações adicionais (i.e., sem bloqueios), independentemente da organização dos contêineres existentes nessa pilha (ou seja, quando $h(s) > v(\vec{c})$). A segunda é transferir $\vec{c}$ para a pilha mais próxima da posição de saída (i.e., menor $d(s)$), diminuindo assim o tempo de operação do guindaste para recuperar $\vec{c}$ no futuro.

Critério 2: Quando o critério 1 é atendido, como resultado, uma pilha t é selecionada para receber o contêiner $\vec{c}$. Se pilha t tem $n + 1$ espaços disponíveis para empilhamento e existe um conjunto C com no máximo n contêineres, de modo que os contêineres em C possam ser pré-realocados para a pilha t antes de realocar o contêiner $\vec{c}$ para a pilha t, então o critério 2 é aplicado da seguinte maneira. Os contêineres $q \in C$ devem satisfazer duas condições: (1) $h(t) > v(q) > v(\vec{c})$ para impedir a ocorrência de um bloqueio; e (2) o contêiner q está localizado no topo de uma pilha e bloqueando algum contêiner. Além disso, os contêineres em C são pré-realocados em ordem crescente por prioridade de modo que o contêiner com a menor prioridade seja pré-realocado primeiro para a pilha t, evitando assim bloqueios. Perceba que esse segundo critério caracteriza a resolução em um PRC irrestrito, porque contêineres, que não estão no situados topo da pilha que se encontra o contêiner de maior prioridade da baia, podem ser realocados. O benefício desta pré-realocação é transferir contêineres, em um momento adequado, para uma pilha onde eles não serão mais realocados, evitando futuras realocações. Além disso, pela condição 2, apenas os contêineres bloqueadores são transferidos, o que significa que estes precisarão ser realocados mais cedo ou mais tarde por causa dos seus bloqueios.

Critério 3: Se o critério 1 não pode ser atendido e houver pilhas elegíveis $s \in S$, selecione a pilha s com o maior $h(s)$ para realocar o contêiner $\vec{c}$. Note que, uma vez que o contêiner $\vec{c}$ tenha sido colocado na pilha selecionada s, um bloqueio é produzido porque o critério 1 não é cumprido (i.e., $h(s) < v(\vec{c})$). Consequentemente, o contêiner $\vec{c}$ deve ser realocado novamente em um futuro tão próximo quanto $\vec{c}$ está próximo de $h(s)$.

Portanto, a finalidade deste critério é atrasar o máximo possível essa segunda realocação do contêiner $\vec{c}$, selecionando a pilha com o maior $h(s)$. Dessa forma, a segunda realocação ocorrerá em um momento com menos contêineres na baia, de modo que uma realocação apropriada seja mais facilmente encontrada. Uma realocação apropriada é aquela que não produz bloqueio e consome baixo tempo de operação do guindaste.

Algoritmo 7: Algoritmo da Heurística Tríade para solucionar o PRC

Entrada: *baia*, configuração inicial da baia
Saída: solução construída.

```
1:  solucao ← ∅
2:  enquanto baia não está vazia faça
3:    se ĉ está no topo então
4:      operacao ← retirar(ĉ,baia)
5:    else
6:      t ← selecione uma pilha segundo o Critério 1
7:      se t não é nula e o PRC irrestrito é abordado então
8:        C ← selecione um conjunto de contêineres segundo o Critério 2
9:        enquanto C não está vazio faça
10:         q ← selecione um contêiner em C segundo o Critério 2
11:         operacao ← realocar(q,t,baia)
12:         adiciona operacao à solucao
13:       fim do enquanto
14:       operacao ← realocar(c⃗,t,baia)
15:     else
16:       t ← selecione uma pilha segundo o Critério 3
17:       operacao ← realocar(c⃗,t,baia)
18:     fim do se
19:   fim do se
20:   adiciona operacao à solucao
21: fim do enquanto
22: retorne solucao
```

Definidos os três critérios utilizados pela heurística Tríade, o Algoritmo 7 descreve o pseudocódigo dessa heurística. No início do algoritmo, a solução é inicializada com um conjunto vazio de operações (linha 1). As iterações do algoritmo são realizadas enquanto a baia (*baia*) não estiver vazia (linhas 2-21). A cada início de iteração é verificado se o próximo contêiner a ser retirado, $\hat{c}$, está atualmente situado no topo de uma pilha (linha 3). Em caso positivo, o contêiner $\hat{c}$ é retirado e, por conseguinte, uma operação de retirada é produzida (linha 4). Caso contrário, o contêiner situado no topo da pilha ($\vec{c}$), onde está o contêiner $\hat{c}$, precisa ser realocado para o topo de outra pilha.

Como pode haver várias opções de realocação, o Critério 1 é aplicado para selecionar pilha t para colocar o contêiner $\vec{c}$. Se houver essa pilha t e o PRC irrestrito estão sendo abordado pelo algoritmo, o Critério 2 será aplicado, e, assim, um conjunto de contêineres C é gerado conforme definido no Critério 2 (linha 8). Em seguida, também segundo o Critério

2, cada contêiner $q \in C$ é realocado para a pilha t produzindo operações de realocação na solução (linhas 10-12). Quando todos os contêineres em C são pré-realocados para a pilha t, o contêiner $\vec{c}$ é realocado para a pilha t e uma operação de realocação é produzida (linha 14). Caso nenhuma pilha seja selecionada pelo Critério 1, o Critério 3 é adotado e uma operação de realocação sobre o contêiner $\vec{c}$ é produzida (linhas 16-17). No fim de cada iteração, a operação gerada na iteração (*operacao*) é adicionada à solução (linha 20). O algoritmo finalmente chega ao fim quando a baia é esvaziada e a solução encontrada é retornada (linha 22).

Vale pontuar que a heurística Tríade utilizou como referência a heurística com função de penalidade definida em (LIN; LEE; LEE, 2015). As principais diferenças entre essas heurísticas são: (1) a condição para pré-realocar um contêiner na heurística Tríade é $h(t) > v(q) > v(\vec{c})$, enquanto na outra é $h(t) > v(q) > v(\vec{c}) > h(t) - 5$; (2) a heurística Tríade não possui fator de penalidade; (3) a heurística Tríade é proposta para abordar o PRC restrito e o irrestrito, enquanto a outra abordada apenas o PRC irrestrito. A restrição $v(\vec{c}) > h(t) - 5$ não foi adotada na heurística Tríade, pois ela limita a quantidade de contêineres que podem ser pré-realocados. Por fim, a Figura 20 exibe o fluxograma do Algoritmo 7, nele é apresentando o passo a passo da heurística Tríade proposta para solucionar o PRC restrito e o PRC irrestrito, conforme relatado acima.

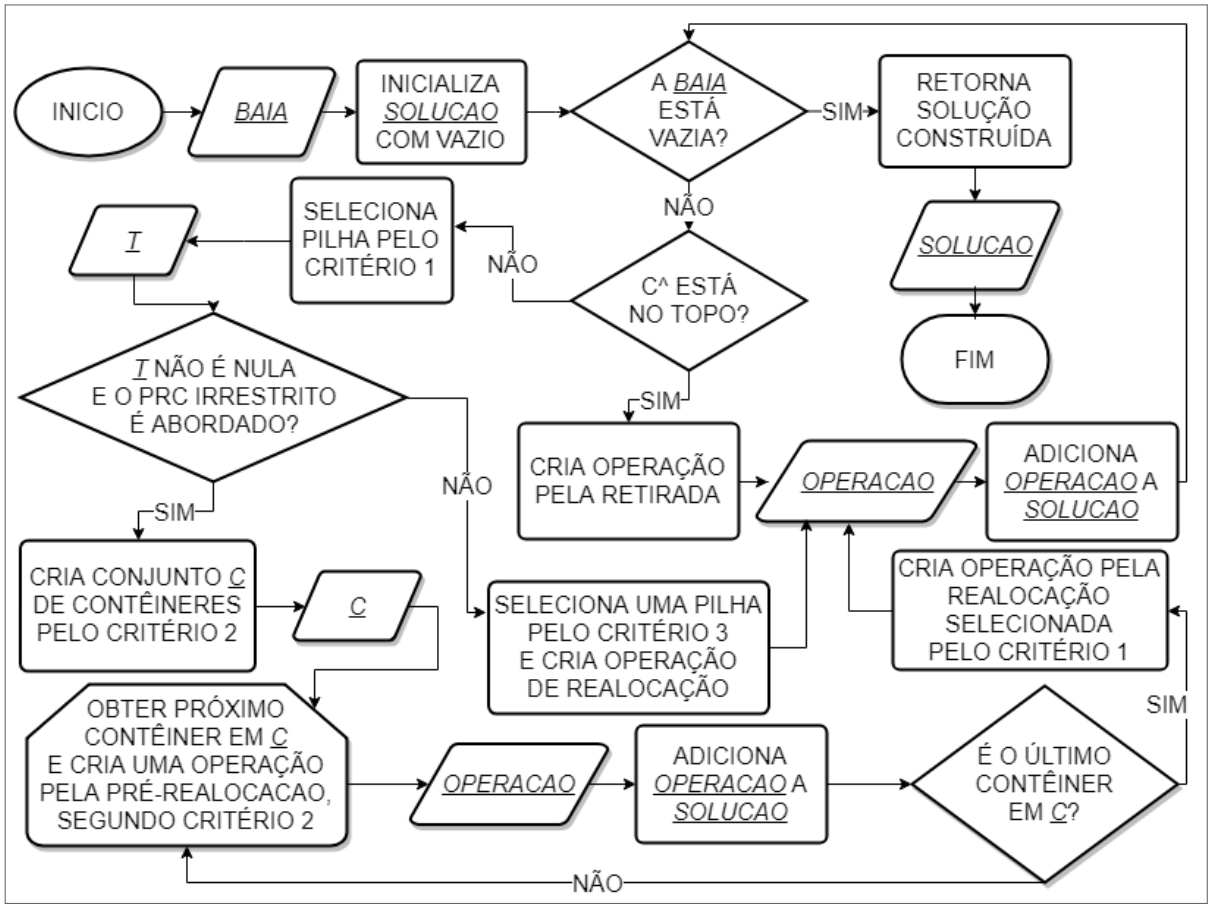


Figura 20 – Fluxograma do Algoritmo da Heurística Tríade para solucionar o PRC.

4.5 CONSIDERAÇÕES FINAIS

Solucionar o PRC é uma questão importante para alcançar eficiência operacional em pátios de um sistema de terminal de contêineres. Estudos anteriores na literatura trataram este problema de otimização por meio de métodos exatos e aproximados minimizando o número de realocações de contêineres a fim de reduzir o tempo de operação do guindaste. Alguns poucos trabalhos até propuseram métodos para otimizar diretamente o tempo de operação do guindaste, mas nestes há divergência ou ausência de definição sobre como o tempo de operação é computado. Em vista disso, nesta seção, foram definidas novas métricas de otimização para computar diretamente o tempo de operação do guindaste, pois, como formalmente foi evidenciado nessa seção, minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste. As novas métricas apresentadas foram a trajetória do guindaste e tempo de operação do guindaste, e esta última é computada a partir da primeira.

Definidas as novas métricas de otimização e demostrado a relevância dessas métricas para qualidade das soluções do PRC, novos métodos de otimização exatos e aproximados para solucionar PRC foram apresentados. Estes métodos visam minimizar o tempo de operação do guidaste, onde este tempo é computado a partir da trajetória do guindaste.

Os métodos de otimização exatos desenvolvidos foram: um modelo matemático de PLI para solucionar o PRC restrito; um algoritmo baseado no algoritmo de Dijkstra para tratar o PRC restrito e o PRC irrestrito; e um algoritmo de busca A* para resolver o PRC restrito e o PRC irrestrito. Para a elaboração do A*, foi proposta uma função heurística para agilizar o processo de busca. Está função foi comprovada ser uma função admissível, por meio da constatação de que a função heurística proposta é um limite inferior para o tempo de operação do guindaste.

Já os métodos de otimização aproximados, que foram propostos neste estudo, são baseados em duas meta-heurísticas: a RGRASP e a PM. Inicialmente, o algoritmo RGRASP é proposto para resolver o PRC restrito. Este algoritmo faz uso de índices de decisão para obter boas soluções iniciais, e, em uma fase posterior, as soluções iniciais são melhoradas para reduzir o tempo de operação do guindaste. Em seguida, o algoritmo PM é proposto para solucionar o PRC restrito e o PRC irrestrito. Este algoritmo faz uso de uma heurística construtiva para avaliar e tomar decisão sobre quais elementos devem compor a solução para o PRC. A heurística construtiva proposta neste trabalho foi denominada Heurística Tríade, e utiliza três critérios para construir as soluções. Esta heurística além de ser incorporada ao algoritmo PM, também soluciona o PRC restrito e o PRC irrestrito.

Por fim, na Seção 5, realiza-se a avaliação e a comparação dos métodos de otimização propostos com outros trabalhos existentes sobre um conjunto de instâncias diversas em termos de tamanho e configuração.

5 ANÁLISE EXPERIMENTAL

5.1 INTRODUÇÃO

Esta seção descreve os resultados dos experimentos realizados para a análise e avaliação dos métodos de otimização desenvolvidos, como também, para a análise investigativa sobre o custo-benefício entre as duas classes do Problema de Recuperação de Contêineres (PRC). Assim, esta seção está organizada da seguinte maneira. A Seção 5.2 descreve a configuração do ambiente de teste utilizado para avaliar os métodos de otimização propostos neste trabalho de pequisa. Em seguida, as avaliações sobre os métodos de otimização exatos são detalhadas na Seção 5.3. Os resultados dos experimentos efetuados sobre os métodos de otimização aproximados foram descritos na Seção 5.4 e na Seção 5.5. Depois, na Seção 5.6, é investigado a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito em comparação à resolução no contexto restrito. Por fim, a Seção 5.7 conclui a seção com uma discussão sobre os resultados obtidos nos experimentos.

5.2 CONFIGURAÇÃO DO AMBIENTE

Os experimentos computacionais foram realizados em um computador com sistema operacional (*Windows 10 Education 64 bits*) e com processador (*Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz*) com 16 GB de memória RAM. O modelo matemático, proposto na Seção 4.3.1, foi codificado na linguagem *Java* e conectado com solver comercial *IBM ILOG CPLEX 12.7* como seu mecanismo de otimização. O gerador de aleatoriedade, utilizados nos algoritmos, foi o *Mersenne Twister* encontrado na biblioteca *Apache Commons Math 3.3* [1]. As sementes aleatórias utilizadas foram obtidas das casas decimais de π (.14159265358979323846264338327...) tomadas de 3 a 3, por exemplo, $semente_1 = 141$, $semente_2 = 592$, $semente_3 = 653$, e assim por diante.

Todos os algoritmos foram implementados na linguagem *Java*, utilizando o ambiente de desenvolvimento *Eclipse Neon*. Em todos os algoritmos analisados, o tempo de operação do guindaste é computado por meio da trajetória do guindaste e das quatro velocidades do guindaste, conforme definido pela Equação 4.3, na Seção 4.2. Os valores adotados nos experimentos para as quatro velocidades do guindaste (2 horizontais e 2 verticais) são baseados nos valores definidos em (KIM; KIM; LEE, 2016; LIN; LEE; LEE, 2015), e descritos a seguir. As velocidades horizontais do guindaste utilizadas foram: 2,4 segundos por largura de contêiner quando carregado e 1,2 segundos por largura de contêiner quando

[1] Biblioteca disponível em <commons.apache.org/proper/commons-math/download_math.cgi>

descarregado (i.e., $\nu_1 = \frac{1}{2.4}$ and $\nu_2 = \frac{1}{1.2}$). Já, as velocidades verticais do guindaste foram: 5,18 segundos por camada quando carregado, e 2,59 segundos por camada quando descarregado (i.e., $\nu_3 = \frac{1}{5.18}$ and $\nu_4 = \frac{1}{2.59}$).

Um cenário de experimentação foi especificado para uma melhor análise sobre o desempenho dos algoritmos avaliados. Neste intuito, foi gerado um conjunto de 1200 instâncias do PRC com diferentes configurações (i.e., distribuição dos contêineres) e tamanhos (i.e., número de pilhas e camadas). Todas essas instâncias do problema foram geradas utilizando um gerador de instâncias (*benchmark*) da literatura, proposto em (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2012).

As instâncias foram geradas de acordo com a combinação do número de pilhas (W) e camadas (H), onde $W \in [4, 8]$ e $H \in [5, 16]$. Esta combinação resultou em 60 tipos de baias, considerando que a capacidade de armazenamento varia de 20 a 128 contêineres. A capacidade de armazenamento é o produto do número de camadas pelo número de pilhas. Por exemplo, uma baia com 6 camadas e 5 pilhas (tipo 5×6) tem uma capacidade de armazenamento igual a $5 \times 6 = 30$. Para cada tipo de baia, um grupo de 20 instâncias distintas foi gerado. Um grupo é constituído por cinco subgrupos, de acordo com a taxa de ocupação da baia: 50%, 60%, 70%, 80% e 90%. Além disso, cada subgrupo compreende de quatro instâncias em que os contêineres têm prioridades distintas e são distribuídos aleatoriamente dentro da baia. Como exemplo, em uma baia tipo 5×6, as quatro instâncias nos subgrupos 50%, 60%, 70%, 80% e 90%, possuem, respectivamente, a quantidade de 15, 18, 21, 24 e 27 contêineres distribuídos na baia. Por fim, cada uma das 1200 instâncias (60 tipos × 20 instâncias por tipo) foi classificada como uma instância *pequena*, *normal* *média* e *grande*, de acordo com suas capacidades de armazenamento, conforme divulgado na Tabela 5. Todas estas 1200 instâncias estão disponíveis em <www.cin.ufpe.br/~asf2/csp/instances/>.

Tabela 5 – As classes de instâncias do problema

CLASSE	CAPACIDADE	TIPOS
PEQUENA	20 to 42	**16 types:** 4×5; 4×6; 4×7; 4×8; 4×9; 4×10; 5×5; 5×6; 5×7; 5×8; 6×5; 6×6; 6×7; 7×5; 7×6; 8×5.
NORMAL	44 to 60	**16 types:** 4×11; 4×12; 4×13; 4×14; 4×15; 5×9; 5×10; 5×11; 5×12; 6×8; 6×9; 6×10; 7×7; 7×8; 8×6; 8×7.
MÉDIA	63 to 80	**14 types:** 4×16; 5×13; 5×14; 5×15; 5×16; 6×11; 6×12; 6×13; 7×9; 7×10; 7×11; 8×8; 8×9; 8×10.
GRANDE	84 to 128	**14 types:** 6×14; 6×15; 6×16; 7×12; 7×13; 7×14; 7×15; 7×16; 8×11; 8×12; 8×13; 8×14; 8×15; 8×16.

5.3 AVALIAÇÃO DOS MÉTODOS EXATOS

Nos experimentos realizados com os métodos de otimização exatos, avaliou-se o desempenho dos métodos propostos considerando-se o tempo computacional (medido em segundos) e a quantidade de instâncias do problema, que foram solucionadas otimamente dentro dos limites computacionais especificados. A este respeito, inicialmente, são comparados os resultados obtidos pelos algoritmos propostos Dijkstra e *A Star* (A*) em solucionar o PRC restrito. Os dois algoritmos foram executados dez vezes para cada uma das 1.200 instâncias e foram executados durante um tempo computacional máximo de 120 segundos.

Neste experimento, os algoritmos Dijkstra e A* resolveram, respectivamente, 147 e 148 das 1200 instâncias. As 147 instâncias solucionadas pelo algoritmo Dijkstra estão contidas nas 148 solucionadas pelo algoritmo A*, ou seja, apenas 1 instância não foi resolvida pelo algoritmo de Dijkstra dentro dos limites computacionais especificados, mas foi resolvida pelo algoritmo A*, como mostra a Tabela 6. Essas 148 instâncias pertencem às classes de instâncias *pequena* e *normal*, e são instâncias com até 28 contêineres.

A Tabela 6 mostra a comparação entre os dois algoritmos de busca. Nesta tabela, os resultados são exibidos de forma agregada pelo número de contêineres em cada instância. A segunda e a terceira coluna mostram, respectivamente, o número de instâncias que foram resolvidas e não foram resolvidas de maneira ótima, dentro do tempo computacional máximo, pelo algoritmo de Dijkstra. A quarta e a quinta coluna mostram o tempo médio em termos de tempo computacional das soluções encontradas pelos algoritmos Dijkstra e A*, respectivamente. Por fim, a última coluna mostra, em média, quantas vezes o tempo computacional requerido pelo Dijkstra foi maior que o tempo computacional requerido pelo algoritmo A* nas instâncias solucionadas por ambos algoritmos.

O algoritmo Dijkstra foi capaz de resolver todas as instâncias com até 18 contêineres, em média de 5 segundos. Todas as instâncias com 20 a 28 contêineres e a maioria das instâncias com 19 contêineres puderam ser resolvidas, mas requereram tempos de solução maiores, em média de 15 segundos. Uma instância com 19 contêineres e todas as instâncias acima de 28 contêineres não puderam ser resolvidas, devido ao limite de tempo computacional. Além disso, o tempo de execução obtido pelo algoritmo Dijkstra foi em média **2,36** vezes maior que o do algoritmo A*, conforme mostrado na Tabela 6.

Os resultados computacionais, apresentados na Tabela 6, destacam o desempenho do algoritmo A* ao resolver as instâncias do cenário avaliado. A eficiência da estratégia de busca no algoritmo A* é evidenciada pelo número de soluções resolvidas e pela redução do tempo computacional quando comparado ao algoritmo Dijkstra. Esta eficiência é promovida pela função heurística proposta neste trabalho, na Seção 4.3.3, que reduz o número de nós gerados e visitados durante o processo de navegação pelo grafo de busca. Analisando o tempo computacional, os dados mostram que as instâncias podem ser resolvidas otimamente dentro de um tempo médio de **9,79** e **4,19** segundos, respectivamente, para o algoritmo Dijkstra e o algoritmo A*. Isso indica que os dois algoritmos propostos podem

Tabela 6 – Resultados computacionais dos algoritmos Dijkstra e A* no PRC restrito.

Instâncias por N.º de Contêineres	**Resolvidas**	**Não Resolvidas**	**Dijkstra Tempo (segundos)**	**A* Tempo (segundos)**	**Razão entre Tempos**
10	4	0	0,06	0,02	2,28
12	12	0	0,11	0,05	2,03
14	12	0	0,36	0,23	1,54
15	12	0	0,4	0,15	2,61
16	16	0	9,36	2,27	4,12
17	9	0	3,63	1,7	2,14
18	17	0	14,44	1,37	10,52
19	9	1	3,94	2,08	3,75
20	7	0	20,65	12,24	1,69
21	23	0	17,65	7,56	2,33
22	9	0	10,5	7,56	1,39
24	10	0	22,5	13,76	1,64
25	5	0	17,79	9,91	1,79
26	1	0	4,34	4,09	1,06
28	1	0	1,52	0,88	1,73
TODAS	147	1	9,79	4,19	2,36

ser aplicados em cenários reais para resolver instâncias de tamanho pequeno e normal, e, em especial, o algoritmo A*, por seu tempo computacional menor.

Com a finalidade de avaliar o método de otimização exato por intermédio de um modelo matemático de Programação Linear Inteira (PLI), o algoritmo A* foi utilizado para validar o modelo de otimização proposto na Seção 4.3.1 por meio de testes comparativos, pois ambos são métodos exatos. As 148 instâncias resolvidas pelo algoritmo A* foram utilizadas para avaliar o modelo proposto. Estas instâncias foram solucionadas pelo solver CPLEX como sendo o mecanismo de otimização para o modelo. O solver CPLEX foi executado cinco vezes para cada instância usando suas configurações padrões. Cada execução foi limitada a um tempo computacional máximo de 16 minutos, que é oito vezes maior que o limite usado nos testes do algoritmo A*.

A Tabela 7 mostra a comparação entre os métodos CPLEX e algoritmo A* na resolução de instâncias sobre o PRC restrito. Nesta tabela, os resultados estão agregados pelo número de contêineres em cada instância, e os número de instâncias resolvidas e não resolvidas pelo CPLEX são dados pela segunda e terceira coluna da tabela. As duas próximas colunas mostram o tempo médio de execução requerido pelo CPLEX e algoritmo A*, respectivamente. A última coluna mostra, em média, quantas vezes o tempo computacional requerido pelo CPLEX foi maior que o tempo requerido pelo algoritmo A* nas instâncias solucionadas por ambos.

Tabela 7 – Resultados computacionais dos métodos CPLEX e A* no PRC restrito.

Instâncias por N.º de Contêineres	**Resolvidas**	**Não Resolvidas**	**CPLEX Tempo (segundos)**	**A* Tempo (segundos)**	**Razão entre Tempos**
10	4	0	37,24	0,02	1503,09
12	12	0	117,26	0,05	2137,89
14	11	1	192,13	0,23	2182,19
15	8	4	407,12	0,15	11956,47
16	8	8	283,22	2,27	750,99
17	0	9	-	1,70	-
18	1	16	238,76	1,37	5657,82
19	1	9	859,09	2,08	25416,75
TODAS ATÉ 19	45	47	228,67	1,10	2021,15
20	0	7	-	12,24	-
21	0	23	-	7,56	-
22	0	9	-	7,56	-
24	0	10	-	13,76	-
25	0	5	-	9,91	-
26	0	1	-	4,09	-
28	0	1	-	0,88	-
TODAS	45	103	228,67	4,19	2021,15

Os resultados indicaram que a resolução pelo modelo matemático de PLI obteve as mesmas soluções produzidas pelo algoritmo A*, e portanto, os testes confirmaram a validade do modelo de otimização proposto. Além disso, a otimização pelo modelo de PLI foi capaz de resolver otimamente apenas 45 das 148 instâncias, porque o CPLEX excedeu o tempo computacional máximo ou esgotou a memória computacional, devido às muitas variáveis de decisão sendo processadas. O modelo matemático foi capaz de resolver todas as instâncias com até 12 contêineres em tempo razoável, em média de 92 segundos. A maioria das instâncias com 14 e 15 contêineres e metade das instâncias com 16 contêineres puderam ser resolvidas, mas requereram tempos de solução mais longos, em média de 283 segundos. Algumas instâncias com 18 e 19 contêineres, todas as instâncias com 17 contêineres e todas as instâncias acima de 19 contêineres não puderam ser resolvidas, principalmente devido ao estouro de memória. Além disso, o tempo de execução obtido pelo CPLEX foi em média **2021** vezes maior que o do algoritmo A*, conforme mostrado na Tabela 7.

Tendo em vista os resultados obtidos, a otimização proposta por modelos matemáticos mostrou-se insuficiente para aplicações em instâncias reais (com mais de 19 contêineres). Os algoritmos de busca de caminhos (i.e., Dijkstra e A*) acabam por ser eficazes e eficientes na resolução de instâncias de pequeno e normal porte, principalmente o A*. Contudo, algumas instâncias, especialmente as de médio e grande porte, não foram resol-

vidas, devido à alta carga computacional dos métodos exatos. Portanto, outra abordagem é necessária para resolver o PRC restrito sobre instâncias de médio e grande porte, como os métodos aproximados, que foram anteriormente apresentados neste documento e avaliados nas próximas seções. Vale ressaltar que, os resultados obtidos pelos algoritmos Dijkstra e A* sobre o PRC irrestrito serão descritos na análise custo-benefício relatada na Seção 5.6.

5.4 AVALIAÇÃO DO ALGORITMO REACTIVE GRASP

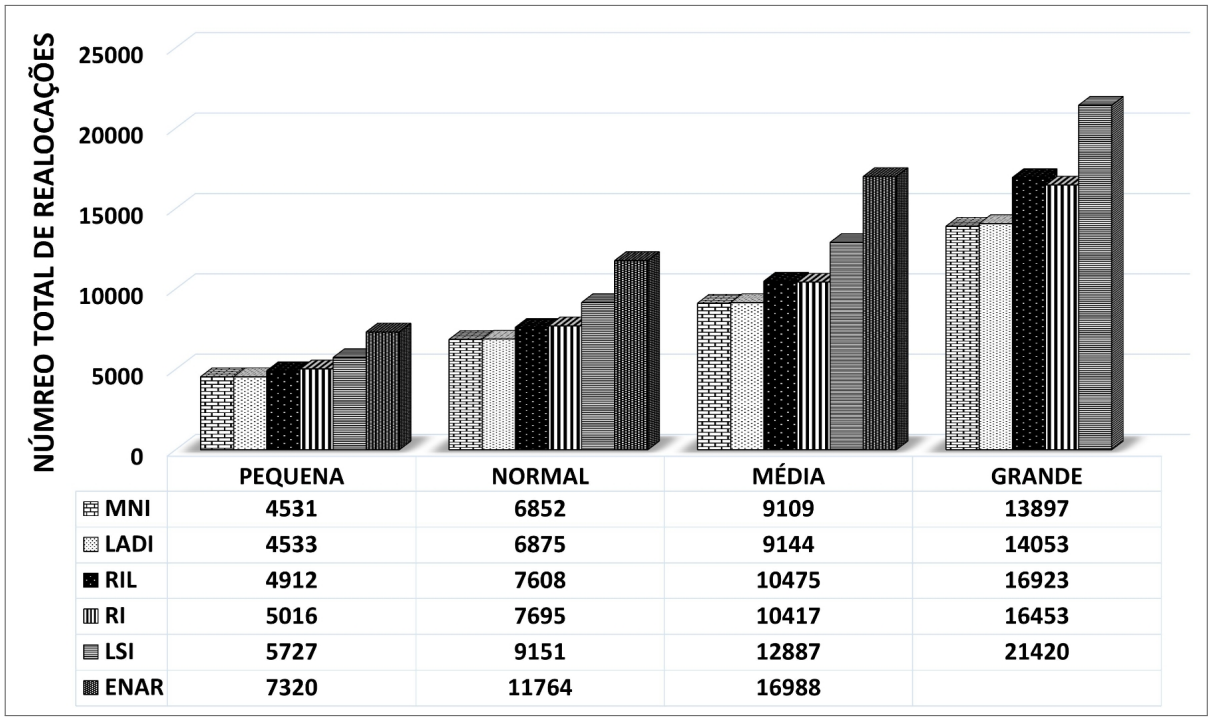
Este trabalho propôs um algoritmo aproximado baseado na meta-heurística *Reactive Greedy Randomized Adaptive Search Procedure* (RGRASP) e este utiliza índices de decisão na construção de soluções. Esta seção é dedicada a analisar o desempenho do algoritmo proposto e discorre sobre os experimentos realizados. Em vista disso, inicialmente, são realizados experimentos sobres os índices de decisão, na Seção 5.4.1, para investigar o melhor índice a ser utilizado em nosso algoritmo. Em seguida, na Seção 5.4.2, algumas propriedades do algoritmo RGRASP são avaliadas e uma análise comparativa entre algoritmo RGRASP e algoritmos existentes na literatura é realizada.

5.4.1 Análise dos Índices de Decisão

Os índices de decisão são utilizados para escolher qual a próxima operação de realocação deve ser realizada de modo a reduzir o número de realocações futuras. O algoritmo proposto *RGRASP* recebe como entrada um indicador informando qual índice deve ser utilizado na construção da solução, durante a primeira fase do algoritmo (i.e., a Fase de Construção). Portanto, é preciso avaliar o impacto dos índices de decisão no desempenho na fase de construção do algoritmo, e, assim, qualificar qual o melhor índice de decisão a ser adotado no algoritmo *RGRASP*. Foram avaliados nos testes experimentais, os seis índices descritos na Seção 2.3: *Lowest-Slot Index* (LSI), *Reshuffle Index* (RI), *Reshuffle with Look-Ahead Index* (RIL), *Expected Number of Additional Relocations* (ENAR), *Lowest Absolute Difference Index* (LADI) e *Min-Max Index* (MNI).

Os índices de decisão avaliados foram codificados no procedimento da Fase de Construção do algoritmo *RGRASP*, procedimento definido pelo Algoritmo 4 na Seção 4.4.1.3. Dessa forma, os experimentos sobre os índices foram realizados por meio da execução desse procedimento. Nos experimentos, o valor do grau de gulosidade(α) adotado foi igual 1 para que a execução seja equivalente a uma abordagem gulosa, uma vez que esses índices são utilizados em abordagens gulosas. Para cada índice de decisão avaliado, foram realizadas 1200 execuções, uma execução para cada instância das 1200 instâncias.

A Figura 21 mostra os resultados dos experimentos sobre a avaliação dos índices de acordo com as classes de instância (*pequena*, *normal*, *média* e *grande*). O gráfico exibe o número total de realocações geradas pela utilização de cada índice. Os resultados mostram



	PEQUENA	NORMAL	MÉDIA	GRANDE
MNI	4531	6852	9109	13897
LADI	4533	6875	9144	14053
RIL	4912	7608	10475	16923
RI	5016	7695	10417	16453
LSI	5727	9151	12887	21420
ENAR	7320	11764	16988	

Figura 21 – Avaliação dos Índices de Decisão sobre as classes de instância.

que o índice que apresentou melhores resultados, ou seja, apresentou o menor número de realocações, em todas as classes de instância, foi o MNI. Além disso, os melhores índices de decisão para reduzir o número de realocações de contêineres foram MNI, LADI, RI, RIL, LSI e ENAR, nesta ordem. Outro ponto favorável ao MNI está no fato que, analisando os resultados ao nível de instância, o índice MNI não alcançou o menor número de realocações em apenas 109 das 1.200 instâncias (ou seja, 9% dos casos). Vale ressaltar, que devido ao alto tempo de processamento dos experimentos, apenas os cinco melhores índices obtidos nas classes *pequena*, *normal* e *média* foram avaliados na classe *grande*. Assim, o índice ENAR não é exibido para a classe *grande*.

Por fim, é importante observar que o índice RIL forneceu melhores resultados que o RI em instâncias normais e pequenas, e esse resultado é consistente com os resultados relatados em (WU; TING, 2010). No entanto, para instâncias com capacidade de armazenamento maior que as avaliadas em (WU; TING, 2010), o índice RI obtém melhores resultados que o RIL, como observado em instâncias de médio e grande porte.

5.4.2 Análise do Algoritmo RGRASP

Nesta seção é analisado o desempenho do algoritmo RGRASP em solucionar o PRC restrito, otimizando o tempo de operação do guindaste. Inicialmente, na Seção 5.4.2.1, descreve-se a configuração de teste utilizada para avaliação do algoritmo nos experimentos, e depois as seguintes propriedades do algoritmo RGRASP são avaliadas: a convergência das soluções produzidas, a contribuição da fase de busca local e a proximidade da

otimalidade. Por fim, na Seção 5.4.2.2, realiza-se uma avaliação comparativa do algoritmo RGRASP com outros algoritmos da literatura.

5.4.2.1 Configuração e Propriedades do Algoritmo RGRASP

Nos experimentos computacionais do RGRASP, o algoritmo denominado *RGRASP_MNI* é avaliado. Este algoritmo é o algoritmo RGRASP proposto, utilizando o MNI como índice de decisão. O MNI foi adotado no algoritmo porque mostrou os melhores resultados nos experimentos realizados na Seção 5.4.1. Em todos os experimentos realizados sobre o algoritmo RGRASP_MNI, ele foi executado com o valor inicial do grau de gulosidade (α) igual 1; a ação inicial, para ajustar reativamente α, igual a D, o valor de incremento para α igual a 0,01; e os seguintes critérios de parada: tempo máximo de execução de 4 segundo ou número máximo de 10.000 iterações consecutivas sem melhoria na qualidade da solução produzida. O curto tempo de 4 segundos foi utilizado como limite de tempo de execução, de modo a possibilitar a comparação entre o RGRASP_MNI e os demais algoritmos avaliados, pois estes executam em curto espaço de tempo.

Devido ao facto de que a aleatoriedade pode influenciar o algoritmo RGRASP, para realização das avaliações, o RGRASP_MNI foi executada 21 vezes para cada instância das 1.200 e cada execução usou a semente aleatória correspondente ao seu número de execução. Por exemplo, a 1ª execução usou $seed_1$, cujo valor é de 141, enquanto que a 2ª execução usou $seed_2$, cujo valor é 592 e assim por diante.

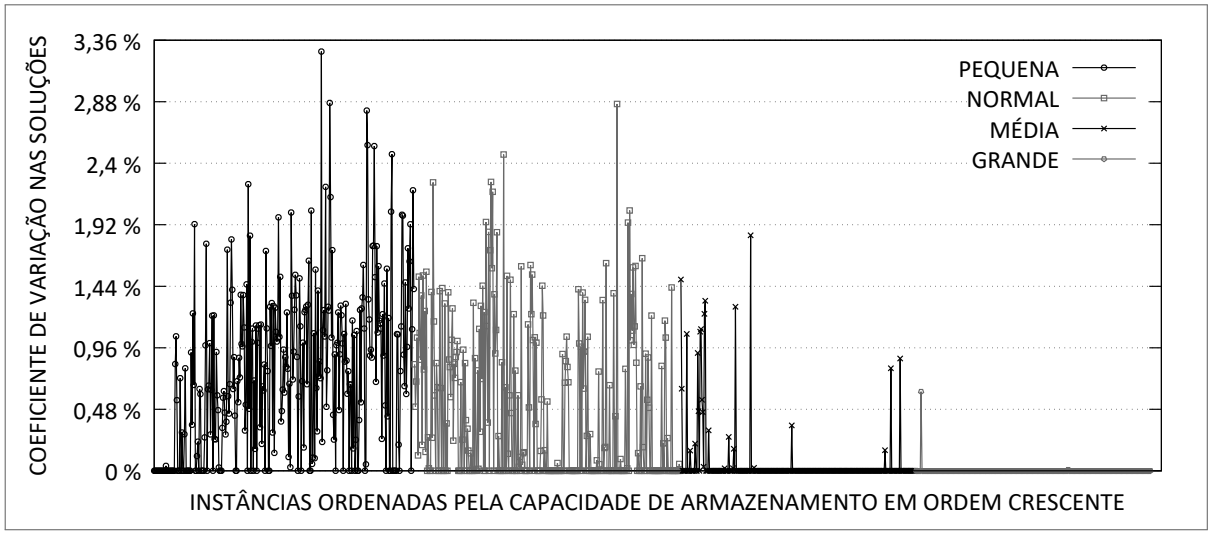
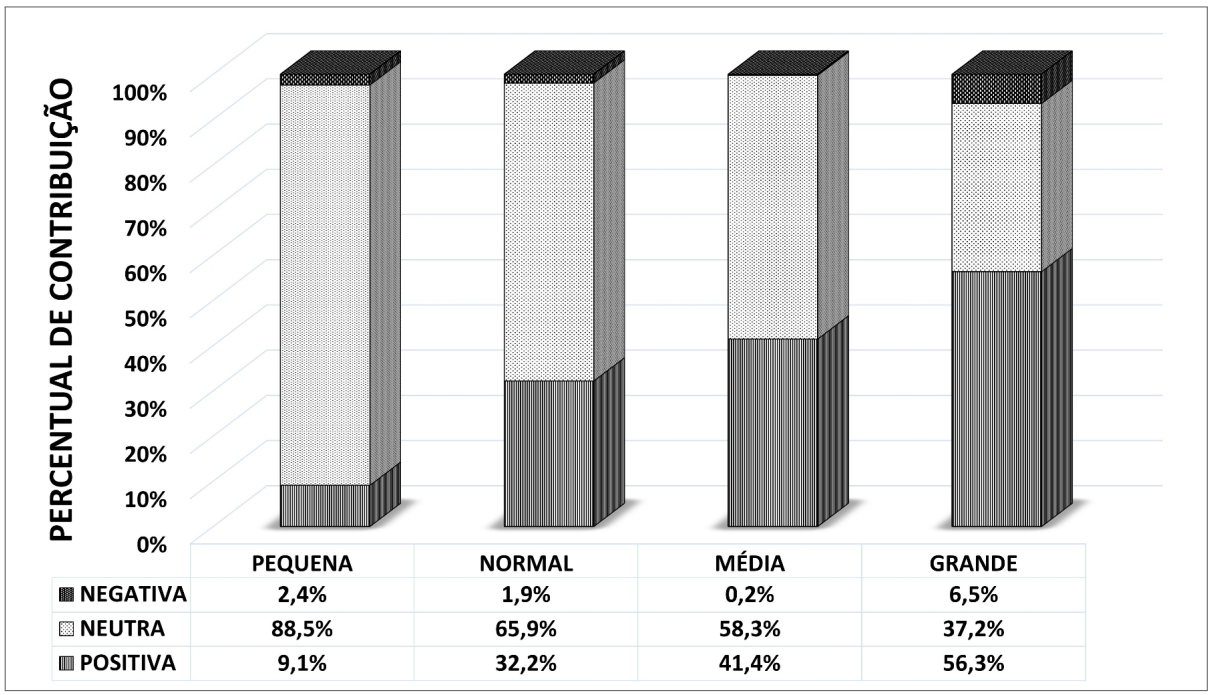


Figura 22 – Coeficiente de variação para as soluções produzidas pelo RGRASP_MNI.

Ainda, no que diz respeito à aleatoriedade, foi investigado a convergência das soluções produzidas em um algoritmo aleatório, como o RGRASP_MNI. Assim, a variabilidade das soluções produzidas pelo RGRASP_MNI é avaliada ao longo das 21 execuções para cada uma das 1200 instâncias. A Figura 22 exibe os coeficientes de variação nas soluções produzidas pelo algoritmo RGRASP_MNI para cada instância avaliada, onde as

instâncias estão ordenadas pela suas respectivas capacidades de armazenamento e estão agrupadas pelas classes de tamanho. Os resultados computacionais apresentados na Fig. 22 reportam que o RGRASP_MNI apresentou uma boa convergência nas soluções produzidas em termos de tempo de operação do guindaste, principalmente nas instâncias de médio e grande porte, onde predomina-se valores muito baixos no coeficiente de variação. A interpretação destes resultados é de que há uma baixa dispersão entre os valores das soluções produzidas, tendo em vista que o maior coeficiente de variação foi de 3,27% e o coeficiente de variação médio foi de 0,33%. Portanto, embora o algoritmo proposto seja uma abordagem aleatória, as suas soluções tendem a convergir mesmo sob diferentes sementes de aleatoriedade.



	PEQUENA	NORMAL	MÉDIA	GRANDE
NEGATIVA	2,4%	1,9%	0,2%	6,5%
NEUTRA	88,5%	65,9%	58,3%	37,2%
POSITIVA	9,1%	32,2%	41,4%	56,3%

Figura 23 – A contribuição da fase de busca local para o RGRASP.

Outro ponto investigado, no algoritmo RGRASP, foi a contribuição da fase de busca local proposta para o algoritmo em termos de melhoria das soluções. Nesse intuito, uma variante do RGRASP_MNI, que não possui o procedimento de busca local, chamada de *RGRASP_S*, foi criada e comparada com RGRASP_MNI. Nesta avaliação comparativa, o algoritmo RGRASP_S, também foi executado 21 vezes e com a mesma configuração de teste usadas pelo RGRASP_MNI. Além disso, as contribuições do procedimento de busca local foram classificadas como *positiva*, *neutra* e *negativa*, de acordo com o desempenho do RGRASP_MNI em relação ao RGRASP_S em termos dos valores da solução (i.e., os tempos de operação do guindaste). A contribuição é dita positiva quando a solução encontrada pelo RGRASP_MNI é melhor que a solução encontrada pelo RGRASP_S. Por conseguinte, a contribuição é negativa quando a solução encontrada pelo RGRASP_S é melhor que a solução encontrada pelo RGRASP_MNI. Se a solução encontrada pelo

RGRASP_MNI possui o mesmo valor que a solução encontrada pelo RGRASP_S, então a contribuição é neutra.

Os resultados computacionais apresentados na Fig. 23 destacam a contribuição do procedimento de busca local proposto para o RGRASP na melhoria da qualidade das soluções. As contribuições positivas foram maiores que as contribuições negativas em todas as classes de instância, sendo a menor porcentagem de contribuições positivas igual a 9,1%. Além disso, as porcentagens de contribuições positivas, neutras e negativas de todas as 1.200 instâncias foram 34,04%, 63,24% e 2,72% respectivamente. Outro fato de destaque é que à medida que os tamanhos das instâncias aumentam, isto é, a complexidade do espaço de busca aumenta, a porcentagem de contribuições positivas tende a aumentar. Esta situação favorável é mais perceptível nas instâncias maiores do problema, onde a porcentagem de contribuições positivas foi de 56,3%. Vale a pena notar que, à medida que os tamanhos das instâncias diminuem, a porcentagem de contribuições neutras tende a aumentar. Isso ocorre porque, em instâncias menores, há poucas soluções sujeitas à melhoria, devido à redução do espaço de busca.

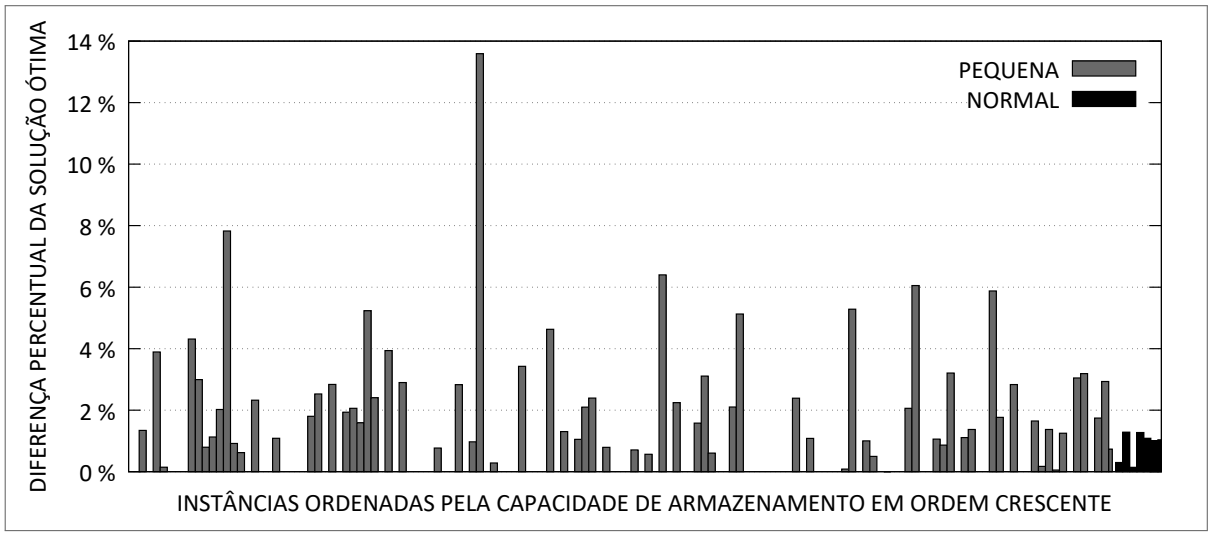


Figura 24 – Os valores da diferença percentual da solução ótima com as soluções produzidas pelo RGRASP_MNI.

A terceira propriedade do algoritmo RGRASP avaliada é a sua proximidade da otimalidade, ou seja, o quão próximo estão as soluções produzidas pelo algoritmo da solução ótima. Neste propósito, uma análise comparativa entre o RGRASP_MNI e um método exato foi realizada. Nesta análise, a proximidade da solução ótima é computada pela diferença percentual entre a solução ótima (*otima*) e a solução produzida (*solucao*), isto é, $proximidade = 100 \times (solucao - otima)/otima$, onde a solução ótima foi obtida pelo algoritmo de busca A* proposto neste trabalho e que solucionou otimamente 148 instâncias. A Figura 24 possibilita perceber que o algoritmo RGRASP proposto apresentou boa proximidade da otimalidade nas soluções produzidas, dado que a maior diferença percentual da solução ótima foi **13,59%**, e na média a diferença percentual foi de **1,15%** sobre

as 148 instâncias avaliadas. Além disso, a figura mostra que essa boa característica de proximidade no algoritmo não é influenciada pelo tamanho das instâncias.

Vale destacar que o coeficiente de variação médio e o desvio percentual médio sobre as soluções produzidas pelo algoritmo RGRASP_MNI foram 0.36% e 0.45%, respectivamente, confirmando a excelente convergência das soluções nas 148 instâncias avaliadas. Portanto, o algoritmo RGRASP é capaz de obter soluções ótimas, pois a solução ótima foi obtida em **48,65%** das 148 instâncias avaliadas, como também, é capaz de obter soluções bem próximas do ótimo em curto tempo computacional. Ademais, o algoritmo é capaz de resolver instâncias maiores não resolvidas por métodos exatos. Estes resultados experimentais confirmam a eficácia do algoritmo proposto RGRASP para encontrar soluções de alta qualidade para o PRC restrito em termos de tempo de operação do guindaste.

5.4.2.2 Avaliação Comparativa do Algoritmo RGRASP

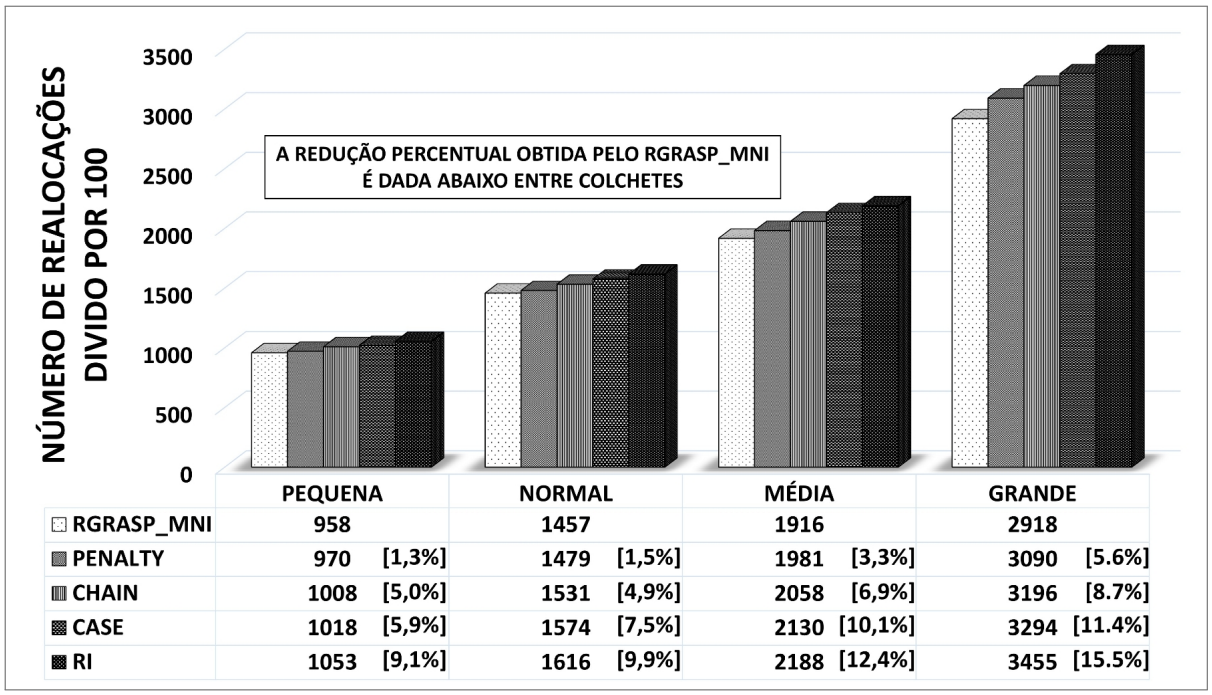
Avaliadas algumas propriedades do algoritmo RGRASP, realiza-se uma análise comparativa com o algoritmo RGRASP_MNI em termos de redução do número de realocações e do tempo de operação do guindaste. Lembrando, que este último é derivado da trajetória do guindaste. Apesar da redução do número de realocações não ser o objetivo de otimização do algoritmo RGRASP_MNI, este objetivo está sendo avaliado pelas seguintes razões. Em primeiro lugar, o algoritmo RGRASP constrói a suas soluções iniciais a partir da redução do número de realocações, assim, isso deve produzir algum efeito, a ser analisado, nas soluções concernente ao número de realocações. Em segundo lugar, é um dos objetivos deste trabalho, investigar, por meios dos experimentos, as relações entre o número de realocações e o tempo dr operação do guindaste (derivado a partir da trajetória do guindaste), de modo avaliar a proporcionalidade entre o número de realocações e o tempo de operação do guindaste.

Desta forma, uma análise comparativa entre o RGRASP_MNI e quatro algoritmos (*RI*, *CHAIN*, *CASE* e *PENALTY*) foi realizada. O primeiro é um algoritmo puramente guloso usando o índice de decisão RI, definido em (WAN; LIU; TSAI, 2009). Este algoritmo é avaliado para comparar uma abordagem puramente gulosa com a proposta de abordagem gulosa-aleatorizada, que é o RGRASP_MNI. O segundo algoritmo é um algoritmo heurístico proposto em (JOVANOVIC; VOSS, 2014), que faz uso do índice MNI e apresentou bons resultados na redução do número de realocações. Os dois últimos algoritmos são algoritmos recentes com bons resultados na redução do número de realocações e do tempo de operação: CASE é o algoritmo proposto em (KIM; KIM; LEE, 2016), e PENALTY é o algoritmo com função de penalidade definido em (LIN; LEE; LEE, 2015).

Vale pontuar, que os algoritmos RI e CHAIN foram projetados para solucionar o PRC restrito e reduzir o número de realocações. Assim estes dois algoritmos são bons comparativos para o RGRASP_MNI em termos da redução do número de realocações. Já os algoritmos CASE e PENALTY foram projetados para solucionar o PRC irrestrito e

minimizar o tempo de operação reduzindo o número de realocações. Estes dois algoritmos possuem, conceitualmente, vantagem na qualidade de suas soluções em relação aos demais algoritmos projetados para PRC restrito (i.e., RGRASP_MNI, RI e CHAIN), pois eles podem considerar operações irrestritas nas suas soluções, tendo assim, maior oportunidade de otimização. Mesmo assim, o algoritmo RGRASP_MNI é comparado com estes dois algoritmos para aferir quão boas são as suas soluções geradas em um contexto restrito (i.e., de operações restritas) versus as soluções geradas em um contexto irrestrito (i.e., de operações irrestritas). Neste mesmo intuito, mais avaliações comparativas serão realizadas, neste trabalho, entre algoritmos projetados para esses dois diferentes contextos do PRC.

Assim como para o RGRASP_MNI, todos os algoritmos foram executados 21 vezes para cada instância das 1.200. Como o algoritmo PENALTY é influenciado pelo fator de penalidade, nos experimentos, o fator de penalidade foi definido para 30. Esse valor foi adotado porque apresentou os melhores resultados nos experimentos realizados em (LIN; LEE; LEE, 2015). Vale mencionar que todos os algoritmos avaliados reportaram um tempo de execução médio abaixo de um segundo nas 1.200 instâncias, portanto, a comparação direta em relação ao tempo médio de execução dos algoritmos não é significativa. Os resultados da comparação dos algoritmos, em termos de número de realocações, são apresentados na Figura 25. A figura compara o número total de realocações obtido por cada algoritmo nas 1200 instâncias e agregado pelas classes de instância. Além disso, essa figura informa, entre colchetes, a redução percentual obtida pelo algoritmo RGRASP_MNI em relação aos valores reportados nos outros algoritmos.



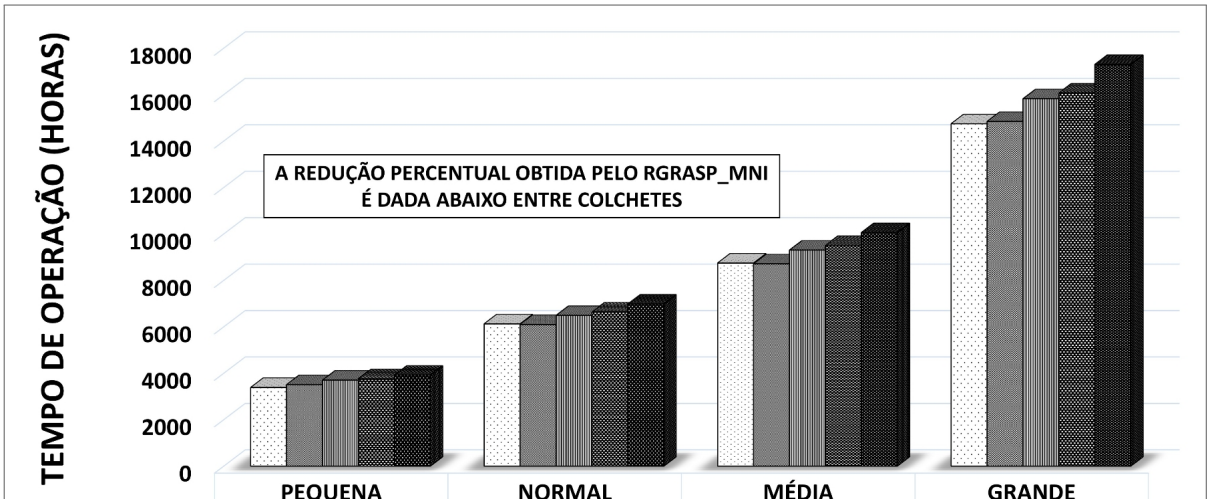
	PEQUENA	NORMAL	MÉDIA	GRANDE
RGRASP_MNI	958	1457	1916	2918
PENALTY	970 [1,3%]	1479 [1,5%]	1981 [3,3%]	3090 [5.6%]
CHAIN	1008 [5,0%]	1531 [4,9%]	2058 [6,9%]	3196 [8.7%]
CASE	1018 [5,9%]	1574 [7,5%]	2130 [10,1%]	3294 [11.4%]
RI	1053 [9,1%]	1616 [9,9%]	2188 [12,4%]	3455 [15.5%]

Figura 25 – Avaliação do RGRASP_MNI por número de realocações.

Os resultados computacionais na Fig. 25 indicam que o algoritmo RGRASP_MNI

obteve melhores resultados que os demais algoritmos em todas as classes de instâncias. O número total de realocações produzidas por RGRASP_MNI foi o menor em comparação com os demais. A redução do número de realocações foi **3,6%** em comparação com o PENALTY, **7,0%** em comparação com o CHAIN, **9,6%** em comparação com o CASE e **12,8%** em comparação com o RI, sabendo que o número total de realocações, nas 1200 instâncias, foram 724.879, 752.052, 779.226, 801.654 e 831.201, para RGRASP_MNI, PENALTY, CHAIN, CASE e RI respectivamente. Além disso, à medida que os tamanhos das instâncias aumentam, ou seja, a complexidade do espaço de busca aumenta, os algoritmos PENALTY, CHAIN, CASE e RI tendem a degradar a qualidade das soluções. Esta situação negativa é mais evidente nas instâncias de grande porte, onde os valores de redução foram, respectivamente, **5,6%**, **8,7%**, **11,4%** e **15,5%**.

Embora projetado para minimizar o tempo de operação em um PRC restrito, os resultados experimentais destacaram a capacidade o algoritmo RGRASP_MNI de encontrar soluções melhores, em termos de número de realocações, mesmo quando comparado com algoritmos que apresentam os seguintes aspectos favoráveis. Os algoritmos CHAIN e RI foram projetados particularmente para minimizar o número de realocações em um PRC restrito. Ademais, os algoritmos PENALTY e CASE contemplam operações irrestritas nas suas soluções. Mesmo diante disso, ainda é necessário avaliar o algoritmo RGRASP_MNI com os demais algoritmos, no que diz respeito ao tempo de operação. Tendo em mente que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste, como já evidenciado neste trabalho. Em vista disso, experimentos comparativos sobre o tempo de operação guindaste é reportado a seguir.



	PEQUENA	NORMAL	MÉDIA	GRANDE
RGRASP_MNI	3388	6129	8748	14746
PENALTY	3512 [3,5%]	6102 [-0,4%]	8710 [-0,4%]	14843 [0,7%]
CHAIN	3706 [8,6%]	6491 [5,6%]	9306 [6,0%]	15822 [6,8%]
CASE	3759 [9,9%]	6639 [7,7%]	9509 [8,0%]	16074 [8,3%]
RI	3920 [13,6%]	6983 [12,2%]	10059 [13,0%]	17300 [13,7%]

Figura 26 – Avaliação do RGRASP_MNI por tempo de operação do guindaste.

Os resultados computacionais apresentados na Figura 26 ressaltam a eficiência do algoritmo RGRASP_MNI ao solucionar as instâncias minimizando o tempo de operação do guindaste. O RGRASP_MNI obteve resultados expressivamente melhores do que os algoritmos RI, CASE e CHAIN em todas as classes. Já, em relação ao algoritmo PENALTY, o RGRASP_MNI foi superior nas classes *pequena* e *grande* e ligeiramente inferior nas classes *normal* e *média*. Além disso, a redução do tempo de operação foi **0,5%** em comparação com o PENALTY, **6,5%** em comparação com o CHAIN, **8,3%** em comparação com o CASE e **13,7%** em comparação com o RI, sabendo que o tempo total de operação em horas, nas 1200 instâncias, foram 33.011, 33.168, 35.324, 35.982 e 38.262, para RGRASP_MNI, PENALTY, CHAIN, CASE e RI respectivamente. Perceba que, embora o algoritmo RGRASP_MNI gere soluções em um contexto restrito, ele apresentou resultados superiores aos de algoritmos competitivos em minimizar o tempo de operação sobre um contexto irrestrito (i.e., CASE e PENALTY).

Como pode ser visto nos resultados das Figuras 25 e 26, não há garantia de proporcionalidade entre o número de realocações e o tempo de operação do guindaste. Uma vez que, os valores de redução apresentados nos dois experimentos (número de realocações e tempo de operação do guindaste) são distintos e também não são proporcionais. Por exemplo, deve-se observar que o algoritmo PENALTY teve um desempenho próximo ao RGRASP_MNI em instâncias de grande porte concernente ao tempo de operação (i.e., redução de 0,7%); todavia, nas mesmas soluções e instâncias do problema, os resultados foram inferiores em termos de número de realocações (i.e., redução de 5,6%).

Por fim, o teste estatístico não paramétrico de *Wilcoxon Rank Sum* (SHESKIN, 2007) foi conduzido para validar a relevância estatística dos resultados obtidos nas avaliações comparativas. Este teste não-paramétrico, com nível de confiança de 99%, revelou que existe uma diferença estatisticamente significativa entre a qualidade das soluções obtidas pelo RGRASP_MNI e as obtidas pelos demais algoritmos nos resultados da avaliação comparativa em termos de número de realocações. Nos resultados da avaliação comparativa em termos de tempo de operação do guindaste, o teste não-paramétrico também revelou, com nível de confiança de 99%, que existe uma diferença estatisticamente significativa entre a qualidade das soluções obtidas pelo RGRASP_MNI e os demais algoritmos, com exceção do algoritmo PENALTY. Portanto, em termos de número de realocações e de tempo de operação, o RGRASP_MNI obteve soluções significativamente melhores que os demais algoritmos, com exceção do algoritmo PENALTY no tocante ao tempo de operação. Visto que a diferença, observada entre as soluções do RGRASP_MNI e do PENALTY, não é estatisticamente significativa, mesmo para níveis de confiança maiores que 53%.

5.5 AVALIAÇÃO DO ALGORITMO PILOT METHOD

Este trabalho propôs um algoritmo baseado na meta-heurística *Pilot Method* (PM) e uma heurística construtiva, denominada Tríade, para solucionar o PRC restrito e irrestrito,

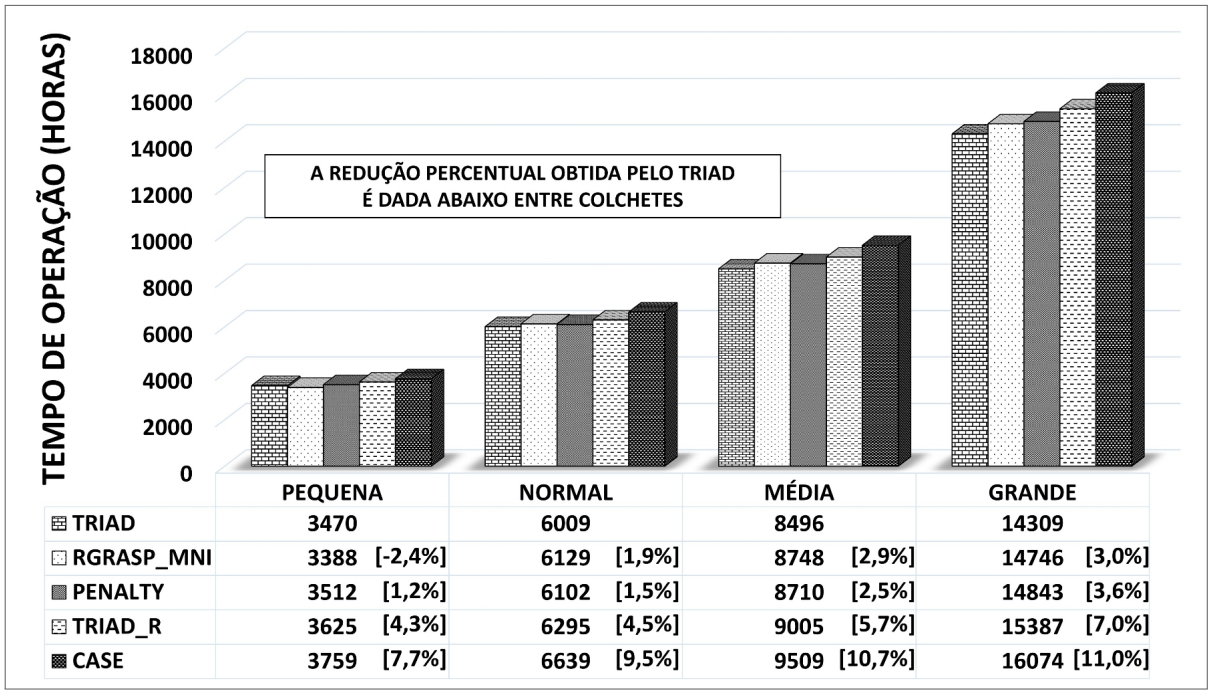
de modo que este algoritmo utiliza essa heurística para avaliar e tomar decisão sobre quais elementos devem compor a solução para o problema. Por meio desse algoritmo e dessa heurística, quatro algoritmos (*PILOT*, *PILOT_R*, *TRIAD* e *TRIAD_R*) são avaliados nesta seção. O primeiro é o algoritmo baseado na meta-heurística PM. O segundo algoritmo equivale ao primeiro algoritmo, mas com abordagem restrita, ou seja, apenas operações restritas são permitidas. O terceiro corresponde ao algoritmo da heurística Tríade e o último algoritmo equivale ao terceiro algoritmo, mas com abordagem restrita.

Esta seção é dedicada a analisar o desempenho desses quatro algoritmos e discorre sobre os experimentos realizados. Em todos os experimentos, os algoritmos PILOT e PILOT_R foram executados com os seguintes critérios de parada: tempo máximo de execução de 4 segundos ou número máximo de 25 iterações consecutivas sem melhoria na qualidade da solução produzida. O curto tempo de 4 segundos foi utilizado como limite de tempo de execução, de modo a possibilitar a comparação com os algoritmos RGRASP_MNI, CASE e PENALTY, porque estes executam em curto espaço de tempo, como já evidenciado na Seção 5.4.2. A seguir, nas seções 5.5.1 e 5.5.2, são descritas as avaliações comparativas, respectivamente, dos algoritmos TRIAD e PILOT.

5.5.1 Análise dos Algoritmo da Heurística Tríade

Uma análise comparativa foi feita entre o algoritmo TRIAD e os algoritmos TRIAD_R, RGRASP_MNI, CASE e PENALTY. O algoritmo TRIAD_R é avaliado nesta comparação para aferir quão boas são as suas soluções geradas pela heurística Tríade proposta em um contexto restrito. Outro algoritmo de operações restritas também é avaliado, o RGRASP_MNI, ele apresentou bons resultados nas avaliações comparativas da Seção 5.4.2.2. Os dois últimos algoritmos também já foram avaliados, e são algoritmos competitivos na literatura em minimizar o tempo de operação sobre um contexto irrestrito. A Figura 27 apresenta os resultados dessa análise, comparando o tempo total de operação do guindaste, agregado pelas classes de instância, e obtido por cada algoritmo na resolução das 1200 instâncias.

Os resultados computacionais na Fig. 27 indicam que o algoritmo TRIAD obteve melhores resultados que os demais algoritmos na maioria das classes de instâncias. O TRIAD não foi superior apenas nas instâncias de pequeno porte em comparação ao algoritmo RGRASP_MNI. Além disso, nas 1200 instâncias, o tempo total de operação do guindaste reportado pelo TRIAD foi o menor em comparação com os demais. A redução do tempo de operação foi **2,2%** em comparação com o RGRASP_MNI, **2,7%** em comparação com o PENALTY, **5,9%** em comparação com o TRIAD_R e **10,3%** em comparação com o CASE, sabendo que o tempo total de operação em horas, nas 1200 instâncias, foram 32.284, 33.011, 33168, 34311 e 35982, para TRIAD, RGRASP_MNI, PENALTY, TRIAD_R e CASE respectivamente. Todos os algoritmos avaliados reportaram um tempo de execução médio abaixo de um segundo nas 1.200 instâncias, assim, uma



	PEQUENA	NORMAL	MÉDIA	GRANDE
TRIAD	3470	6009	8496	14309
RGRASP_MNI	3388 [-2,4%]	6129 [1,9%]	8748 [2,9%]	14746 [3,0%]
PENALTY	3512 [1,2%]	6102 [1,5%]	8710 [2,5%]	14843 [3,6%]
TRIAD_R	3625 [4,3%]	6295 [4,5%]	9005 [5,7%]	15387 [7,0%]
CASE	3759 [7,7%]	6639 [9,5%]	9509 [10,7%]	16074 [11,0%]

Figura 27 – Avaliação do TRIAD por tempo de operação do guindaste.

comparação referente ao tempo médio de execução dos algoritmos não é significativa.

A versão do algoritmo TRIAD para um contexto restrito do PRC, ou seja, o TRIAD_R, não apresentou bons resultados na avaliação comparativa, principalmente, em relação ao algoritmo TRIAD. Todavia, vale destacar que apesar do algoritmo TRIAD_R contemplar apenas operações restritas e ele foi superior ao algoritmo CASE que contempla operações irrestritas. Mais um ponto de destaque para o algoritmo TRIAD está no fato que à medida que os tamanhos das instâncias aumentam, ou seja, a complexidade também aumenta, a qualidade das soluções do algoritmo TRIAD aumentam em relação aos demais. Esta situação positiva é mais evidente nas instâncias de grande porte, onde os valores de redução atingiram, respectivamente, **3,0%**, **3,6%**, **7,0%** e **11,0%**.

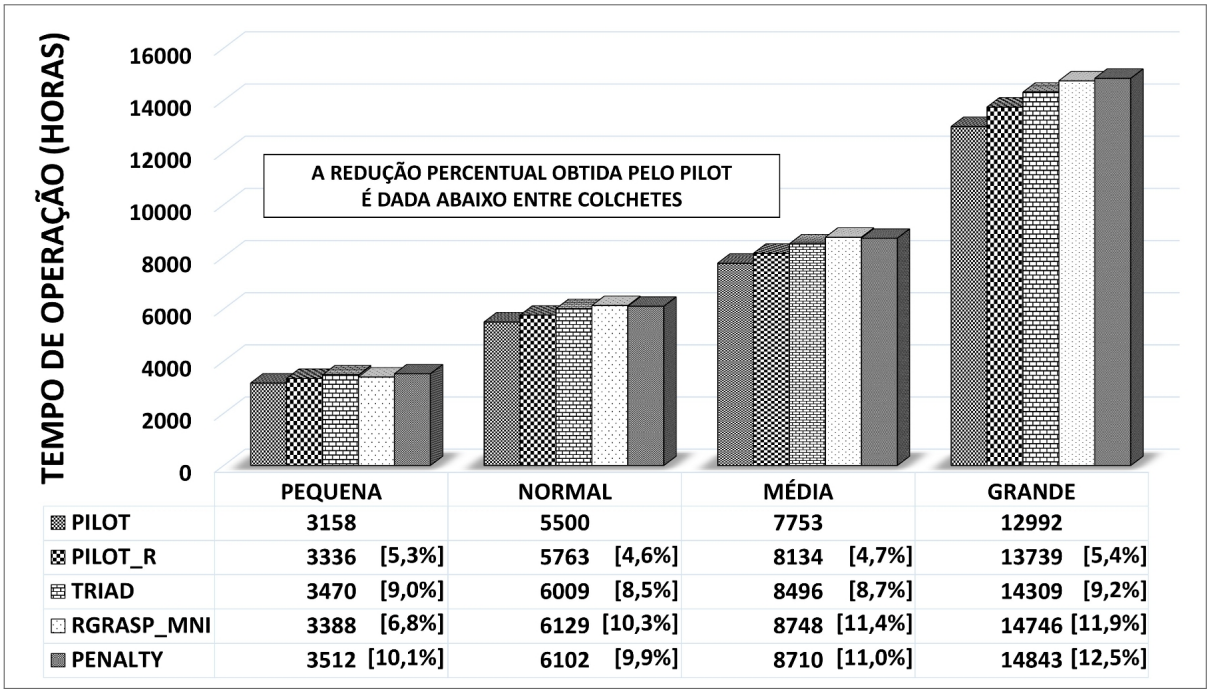
Por fim, o teste estatístico não paramétrico de *Wilcoxon Rank Sum* (SHESKIN, 2007) foi conduzido para validar a relevância estatística dos resultados obtidos nesta avaliação comparativa. O teste revelou, com nível de confiança de 99%, que existe uma diferença estatisticamente significativa entre a qualidade das soluções obtidas pelo algoritmo TRIAD e as obtidas pelos demais algoritmos. Em vista disso, o algoritmo proposto TRIAD reportou soluções significativamente melhores que os demais algoritmos em reduzir o tempo de operação do guindaste.

5.5.2 Análise do Algoritmo PILOT

A *Pilot Method* é uma meta-heurística que aumenta a qualidade da soluções de uma heurística ao incorporá-la, fornecendo um mecanismo inteligente e construtivo para avaliar certas decisões (VOßS; FINK; DUIN, 2005). O algoritmo da heurística Tríade, usada pelo

algoritmo proposto PILOT, reportou, na seção anterior, resultados superiores aos demais algoritmos avaliados em termos da otimização do tempo de operação do guindaste. Dessa maneira, a melhoria na qualidade das soluções promovida pela meta-heurística PM sobre a heurística Tríade é um ponto a ser investigado.

Os dois algoritmos propostos nesse trabalho, baseados na meta-heurística PM, ou seja, PILOT e PILOT_R, foram comparativamente avaliados com os três algoritmos mais bem avaliado nos experimentos referentes a otimização do tempo de operação: TRIAD, RGRASP_MNI e PENALTY. Os cinco algoritmos foram excetuados 21 vezes para cada instância das 1.200. Nestas execuções, todos os algoritmos reportaram um tempo de execução médio abaixo de um segundo nas 1.200 instâncias, logo, a comparação no tocante ao tempo médio de execução dos algoritmos não é significativa. A Figura 28 apresenta os resultados comparativos de cada um dos cinco algoritmo na resolução das instâncias.



	PEQUENA	NORMAL	MÉDIA	GRANDE
PILOT	3158	5500	7753	12992
PILOT_R	3336 [5,3%]	5763 [4,6%]	8134 [4,7%]	13739 [5,4%]
TRIAD	3470 [9,0%]	6009 [8,5%]	8496 [8,7%]	14309 [9,2%]
RGRASP_MNI	3388 [6,8%]	6129 [10,3%]	8748 [11,4%]	14746 [11,9%]
PENALTY	3512 [10,1%]	6102 [9,9%]	8710 [11,0%]	14843 [12,5%]

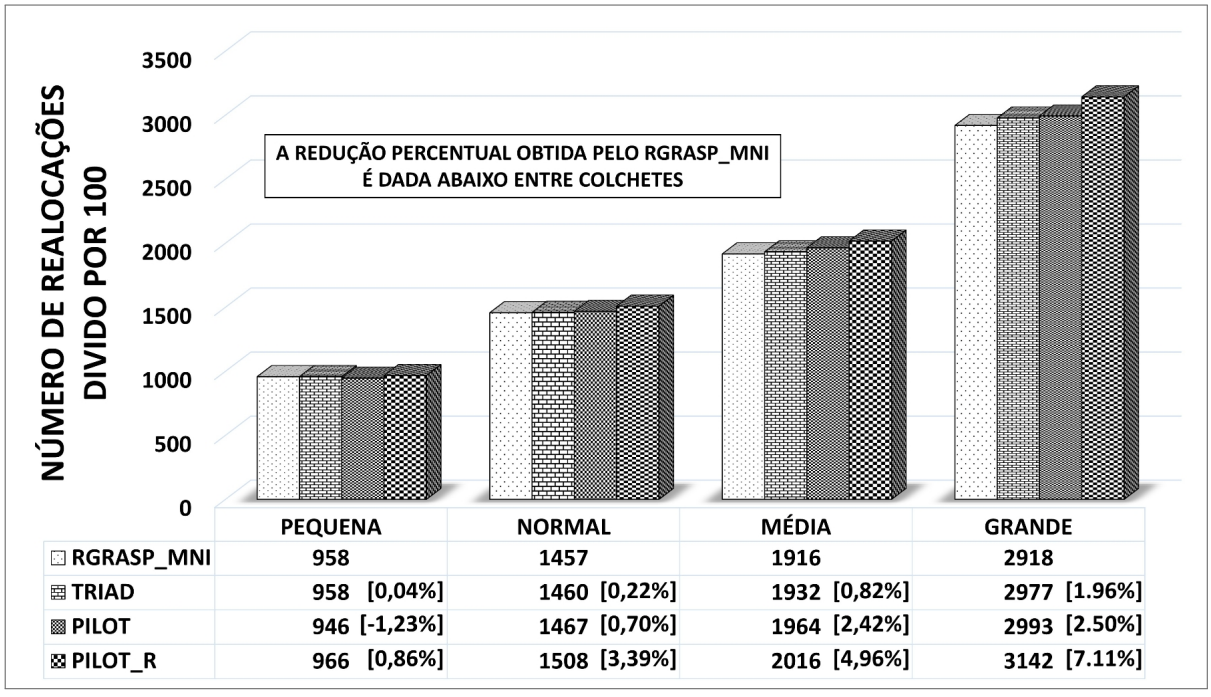
Figura 28 – Avaliação do PILOT por tempo de operação do guindaste.

Os resultados computacionais apresentados na Figura 28 destacam a eficiência dos algoritmos PILOT e PILOT_R ao solucionar as instâncias minimizando o tempo de operação do guindaste. O algoritmo PILOT, em especial, obteve resultados expressivamente melhores do que os algoritmos TRIAD, RGRASP_MNI e PENALTY em todas as classes. Além disso, a redução do tempo de operação foi **5,1%** em comparação com o PILOT_R, **8,9%** em comparação com o TRIAD, **10,9%** em comparação com o RGRASP_MNI e **11,3%** em comparação com o PENALTY, sabendo que o tempo total de operação em horas, nas 1200 instâncias, foram 29.404, 30.972, 32.284, 33.011 e 33.168, para PILOT, PILOT_R, TRIAD, RGRASP_MNI e PENALTY respectivamente. Vale notar que quanto maiores são as instâncias, melhores são as soluções produzidas pelo algoritmo PILOT.

Este aspecto positivo é mais notório nas instâncias de grande porte, onde os valores de redução do PILOT atingiram, respectivamente, **5,4%**, **9,2%**, **11,9%** e **12,5%**.

Embora projetado para minimizar o tempo de operação com soluções de operações restritas, os resultados experimentais destacaram a capacidade o algoritmo PILOT_R de encontrar soluções melhores quando comparado a algoritmos que contemplam operações irrestritas nas suas soluções (i.e., TRIAD e PENALTY). O algoritmo PILOT_R não foi superior ao PILOT, mas foi o segundo algoritmo mais bem avaliado nos experimentos. Outro ponto de destaque do PILOT_R, é que a melhoria na qualidade das soluções promovida pelo PILOT_R sobre algoritmo TRIAD_R foi superior a do algoritmo PILOT sobre o algoritmo TRIAD. Visto que as reduções no tempo de operação foram **9,7%** do PILOT_R para o TRIAD_R, e **8,9%** do PILOT para o TRIAD. Dessa forma, estes resultados evidenciam que os algoritmos PILOT e, em especial, PILOT_R melhoram a qualidade das soluções dos algoritmos que eles incorporam.

Os algoritmos PILOT, PILOT_R, TRIAD foram melhores que o RGRASP_MNI referente a otimização do tempo de operação do guindaste. Contudo, o RGRASP_MNI foi o melhor algoritmo, nos experimentos da Seção 5.4.2.2, concernente a otimização do número de realocações. Por ventura, o algoritmo RGRASP_MNI também seria superior àqueles três algoritmos na otimização do número de realocações. Investigar a relação entre esses dois objetivos de otimização, por meio de experimentos comparativos entre esses quatro algoritmos, pode ajudar ainda mais a corroborar com a hipótese 1 deste trabalho de pesquisa. Desse modo, a Figura 29 apresenta os resultados comparativos desses quatro algoritmos em termos de número de realocações.



	PEQUENA	NORMAL	MÉDIA	GRANDE
RGRASP_MNI	958	1457	1916	2918
TRIAD	958 [0,04%]	1460 [0,22%]	1932 [0,82%]	2977 [1.96%]
PILOT	946 [-1,23%]	1467 [0,70%]	1964 [2,42%]	2993 [2.50%]
PILOT_R	966 [0,86%]	1508 [3,39%]	2016 [4,96%]	3142 [7.11%]

Figura 29 – Avaliação do RGRASP_MNI com PILOT, PILOT_R e TRIAD por número de realocações.

Os resultados computacionais na Fig. 25 indicam que o algoritmo RGRASP_MNI obteve melhores resultados que os demais algoritmos na maioria das classes de instâncias. O RGRASP_MNI não foi superior apenas nas instâncias de pequeno porte em comparação ao algoritmo PILOT. Além disso, nas 1200 instâncias, o número total de realocações reportado pelo RGRASP_MNI foi o menor em comparação com os demais. A redução do número de realocações foi **1,06%** em comparação com o TRIAD, **1,64%** em comparação com o PILOT e **5,02%** em comparação com o PILOT_R, sabendo que o número total de realocações, nas 1200 instâncias, foram 724.879, 732.648, 736.974 e 763.161, para RGRASP_MNI, TRIAD, PILOT e PILOT_R respectivamente.

Como pode ser visto nos resultados das Figuras 28 e 29, novamente, não há garantia de proporcionalidade entre o número de realocações e o tempo de operação do guindaste, dado que os valores de redução também são distintos e não proporcionais. Ademais, os melhores algoritmos em otimizar o tempo de operação, também não são os melhores algoritmos em otimizar o número de realocações. Por exemplo, os algoritmos PILOT, PILOT_R e TRIAD foram os melhores, na Fig. 28, em otimizar o tempo de operação, mas não foram melhores que o RGRASP_MNI em otimizar o número de realocações. Por meio desta observação, mais uma vez reitera-se que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste.

Por fim, o teste estatístico não paramétrico de *Wilcoxon Rank Sum* (SHESKIN, 2007) foi conduzido para validar a relevância estatística dos resultados obtidos nestas avaliações comparativas. O teste revelou, com nível de confiança de 99%, que existe uma diferença estatisticamente significativa entre a qualidade das soluções obtidas pelo algoritmo PILOT e as obtidas pelos demais algoritmos referentes ao tempo de operação do guindaste. Com também, ele revelou com nível de confiança de 99%, que existe uma diferença estatisticamente significativa entre a qualidade das soluções obtidas pelo algoritmo RGRASP_MNI e as obtidas pelos demais algoritmos concernentes ao número de realocações. Em vista disso, os algoritmos propostos PILOT e RGRASP_MNI reportaram soluções significativamente melhores que todos os algoritmos avaliados no tocante à otimização do tempo de operação do guindaste e do número de realocações respectivamente.

5.6 AVALIAÇÃO DO PRC RESTRITO VERSUS PRC IRRESTRITO

Nesta seção, investiga-se, por meio dos experimentos, a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito em comparação à resolução no contexto restrito. Nos experimentos realizados com os métodos de otimização exatos, na Seção 5.3, avaliou-se o desempenho dos métodos propostos apenas em um contexto restrito, ou seja, solucionando o PRC restrito. Assim, para uma análise comparativa desses métodos, em contextos restrito e irrestrito, é preciso, inicialmente, descrever os resultados dos métodos exatos em solucionar o PRC irrestrito. Desse

modo, os algoritmos Dijkstra e A* foram executados dez vezes para cada uma das 1.200 instâncias e foram executados durante um tempo computacional máximo de 120 segundos, similarmente aos experimentos com o PRC restrito.

Neste experimento, os algoritmos Dijkstra e A* resolveram, respectivamente, 3 e 21 das 1200 instâncias. As 3 instâncias solucionadas pelo algoritmo Dijkstra estão contidas nas 21 solucionadas pelo algoritmo A*, e estas 21 estão contidas nas 148 instâncias solucionadas pelo algoritmo A* no contexto restrito. Essas 21 instâncias pertencem à classe de instância *pequena* e são instâncias com até 16 contêineres. Similarmente à Tabela 6 no contexto restrito, a Tabela 8 mostra os dados da comparação entre os dois algoritmos no contexto irrestrito.

Tabela 8 – Resultados computacionais dos algoritmos Dijkstra e A* no PRC irrestrito.

Instâncias por N.º de Contêineres	**Resolvidas**	**Não Resolvidas**	**Dijkstra Tempo (segundos)**	**A* Tempo (segundos)**	**Razão entre Tempos**
10	0	4	-	1,98	-
12	3	7	44,79	17,99	2,10
14	0	4	-	11,06	-
15	0	2	-	62,13	-
16	0	1	-	19,72	-
TODAS	3	18	44,79	17,91	2,10

No PRC irrestrito, o algoritmo A*, novamente, mostrou-se eficiente em relação ao algoritmo Dijkstra devido ao número maior de instâncias solucionadas e o tempo menor de execução. Entretanto, sobre os mesmos limites computacionais, o percentual de instâncias resolvidas pelos algoritmos Dijkstra e A* foram, respectivamente, 2% e 12% em relação às instâncias resolvidas no PRC restrito. Além disso, para solucionar as mesmas instâncias, os algoritmos Dijkstra e A* requereram um tempo de execução muito maior que o requerido no PRC restrito. Por exemplo, o tempo requerido pelo algoritmo A* foi 1069 vezes maior no PRC irrestrito.

Diante disso, percebe-se uma maior carga computacional para solucionar otimamente o PRC irrestrito. Ademais, questiona-se se esta carga computacional elevada do PRC irrestrito justifica o ganho na qualidade das soluções. Noutras palavras, questiona-se se a resolução do PRC em contexto irrestrito é compensatória. Um indicador para esta questão é a razão custo-benefício entre a taxa de decréscimo no tempo de operações do guindaste e a taxa de acréscimo no tempo de execução de algoritmos, ambas inerentes à resolução do PRC no contexto irrestrito em comparação à resolução no contexto restrito. Assim, quando os valores desta razão custo-benefício forem próximos a zero (i.e., inferiores a 0,01), a resolução no contexto restrito não é compensatória, tal que os valores dessa razão custo-benefício estão entre 0 e ∞.

Neste sentido, foram avaliados os algoritmos propostos neste estudo para atender tanto o PRC irrestrito quanto o PRC restrito, ou seja, os algoritmos: Dijkstra, A*, TRIAD e PILOT. As soluções das instâncias solucionadas por estes algoritmos no PRC irrestrito, em contraste com a mesmas instâncias solucionadas no PRC restrito, foram comparadas. A Tabela 9, mostra os resultados desta avaliação custo-benefício para os algoritmos. A segunda e a terceira coluna mostram, respectivamente, a taxa de decréscimo no tempo de operação do guindaste e a taxa de acréscimo no tempo de execução do algoritmo, entre as soluções avaliadas. A quarta e a quinta coluna indicam, respectivamente, o valor da razão custo-benefício e se a resolução no contexto irrestrito mostrou-se compensatória.

Tabela 9 – Resultados avaliação custo-benefício na resolução do PRC irrestrito.

Algoritmo	**Taxa Decréscimo T. Operação**	**Taxa Acréscimo T. Execução**	**Razão Custo-Benefício**	**Resolução PRC Irrestrito Compensatória**
A*	1,027	1069	0,000961	NÃO
DIJKSTRA	1,007	76797	0,000961	NÃO
TRIAD	1,063	1,08	0,984259	SIM
PILOT	1,053	10,2	0,103235	SIM

Para exemplificar o cálculo da razão custo-benefício, observa-se o algoritmo A*. Ele reportou uma taxa de decréscimo no tempo de operação do guindaste igual a 1,027 e uma taxa de acréscimo no tempo de execução do algoritmo igual a 1069 (i.e., consumo de tempo 1069 vezes maior), ambos promovidos pela resolução no contexto irrestrito. Dessa forma, o valor da razão custo-benefício do algoritmo A* é igual a $\frac{1,027}{1069} = 0,000961 \approx 0 < 0.01$, e, logo, a resolução no contexto irrestrito mostrou-se não compensatória. Desta forma, os resultados computacionais indicam que os métodos exatos (i.e., os algoritmos Dijkstra e A*) não apresentaram uma resolução compensatória no contexto irrestrito do PRC, mas os métodos aproximados (i.e., os algoritmos TRIAD e PILOT) mostraram uma resolução compensatória no contexto irrestrito. Portanto, os experimentos realizados apontam indícios que a resolução do PRC no contexto irrestrito não é compensatória para métodos de otimização exatos, mas é compensatória para métodos de otimização aproximados.

Por fim, como os testes sobre o algoritmo A* no PRC irrestrito foram descritos nessa seção, a proximidade da otimalidade nos algoritmos PILOT e TRIAD pode ser avaliada, bem como, nos algoritmos PILOT_R e TRIAD_R. Nesta avaliação, as soluções ótimas foram obtidas das instâncias solucionadas pelo A*: 21 instâncias no PRC irrestrito e 148 no PRC restrito. A Tabela 6 mostra os resultados desta avaliação para todos os algoritmos aproximados que foram propostos neste trabalho. A segunda e a terceira coluna mostram, respectivamente, a maior diferença percentual da solução ótima e a média da diferença percentual da solução ótima, nas instâncias avaliadas. A quarta e a quinta coluna mos-

tram, respectivamente, o percentual de soluções ótimas obtidas nas instâncias avaliadas e quantidade de instâncias avaliadas. Os dados indicam que os algoritmos, especialmente o PILOT e o RGRASP_MNI, reportam boa proximidade da otimalidade nas soluções produzidas, pois a diferença percentual média em todos é inferior a 8%.

Tabela 10 – Resultados computacionais dos algoritmos sobre a proximidade da otimalidade.

Algoritmo	**Maior Diferença Percentual**	**Média Diferença Percentual**	**Percentual de Soluções Ótimas**	**N.º de Instâncias Avaliadas**
PILOT	6,7%	1,5%	52,38%	21
TRIAD	15,1%	4,5%	28,57%	21
RGRASP_MNI	13,59%	1,15%	48,65%	148
PILOT_R	19,4%	2,1%	33,78%	148
TRIAD_R	27,2%	7,4%	6,76%	148

5.7 CONSIDERAÇÕES FINAIS

Esta seção focou na análise dos métodos de otimização propostos neste trabalho, descritos em detalhe na Seção 4. Nesta seção, foi especificado o ambiente de teste utilizado nos experimentos sobre os métodos de otimização propostos e foram descritos os resultados desses experimentos, bem como, foram relatadas conclusões sobre os resultados coletados.

Inicialmente os dois algoritmos exatos (Dijkstra e A*) foram avaliados na otimização do PRC restrito. Os dois algoritmos propostos foram analisados em termos de tempo computacional e a quantidade de instâncias solucionadas otimamente. O algoritmo A* apresentou melhores resultados em todos os termos analisados. Isto se deve a eficácia da estratégia de busca promovida pela função heurística proposta e utilizada no A*. Outro ponto relevante é que devido ao baixo tempo computacional consumido pelos algoritmos exatos, nas instâncias avaliadas, os dois algoritmos podem ser aplicados, otimamente, em cenários reais com baixo tempo computacional (i.e., inferior a 10 segundos), especialmente o algoritmo A*. O mesmo não ocorreu para o modelo matemático de PLI proposto. O tempo de execução requerido pelo modelo foi muito maior que o do algoritmo A*, e a quantidade de instâncias solucionadas otimamente foi menor. Dessa forma, a otimização proposta pelo modelo matemático mostrou-se insuficiente para aplicações em instâncias reais com mais de 19 contêineres. Ao término da avaliação dos métodos exatos, foi reportado que os algoritmos exatos não conseguiram solucionar instâncias maiores. Visto a alta carga computacional destes algoritmos para instâncias maiores.

Em meio a esta lacuna na resolução de instâncias maiores, os algoritmos aproximados foram avaliados. O primeiro algoritmo avaliado foi o RGRASP. Como este algoritmo proposto faz uso de índices de decisão para construção das soluções, primeiramente, tes-

tes experimentais foram executados para identificar os melhores índices de decisão, de modo a ser adotado no algoritmo RGRASP. Nesta avaliação sobres os índices de decisão, os melhores índices em reduzir o número de realocações foram, ordenadamente, MNI, LADI, RI, RIL, LSI e ENAR. Visto que o MNI apresentou o melhor resultado, ele foi adotado no algoritmo RGRASP, sendo referido como RGRASP_MNI, e avaliado com outros algoritmos da literatura.

Antes da avaliação comparativa com outros algoritmos, algumas propriedades do algoritmo RGRASP_MNI foram avaliadas. Os resultados dessa avaliação indicaram que o algoritmo possui boa convergência das soluções; ele é capaz de obter soluções ótimas ou soluções bem próximas do ótimo em curto tempo computacional; e o seu procedimento de busca local contribui expressivamente para a qualidade das soluções produzidas. Nos experimentos comparativos, o algoritmo RGRASP_MNI, em termos de número de realocações e de tempo de operação, obteve os melhores resultados, mesmo sendo comparado com algoritmos projetados para minimizar o número de realocações e com algoritmos de contexto irrestrito.

Os próximos algoritmos avaliados foram os algoritmos ligados à meta-heurística PM e sua heurística construtiva. Dois desses algoritmos são para solucionar o PRC irrestrito, i.e., PILOT e TRIAD, e outros dois para solucionar o PRC restrito, i.e., PILOT_R e TRIAD_R. Nos experimentos comparativos, o algoritmo RGRASP_MNI foi incluído nas comparações, e os resultados indicaram que o algoritmo PILOT apresentou expressivamente melhores resultados em reduzir o tempo de operação do guindaste em relação a todos os algoritmos avaliados e em todas as classes de instância. Mas, na otimização do número de realocações, o algoritmo RGRASP_MNI foi o que apresentou melhores resultados entre todos os avaliados. Vale ressaltar que todos os 5 algoritmos propostos (RGRASP_MNI, PILOT, TRIAD, PILOT_R, TRIAD_R) conseguiram ser superiores a pelo menos um algoritmo competitivo da literatura em reduzir do tempo de operação do guindaste. Ademais, quatro destes cinco algoritmos foram superiores a todos os algoritmos da literatura que foram avaliados nos experimentos.

Na investigação sobre a razão custo-benefício na resolução do PRC irrestrito, os resultados dos testes reportaram uma carga computacional maior para solucionar otimamente o PRC irrestrito quando comparado ao PRC restrito. Além disso, os resultados dos experimentos apontam indícios que a resolução do PRC no contexto irrestrito não é compensatória para métodos de otimização exatos, mas é compensatória para métodos de otimização aproximados. Ao final desta investigação, forma exibidos os resultados sobre a proximidade da otimalidade nos 5 algoritmos aproximados, propostos neste trabalho. Nestes resultados, todos os algoritmos apresentaram boa proximidade da otimalidade, em especial os algoritmos PILOT e RGRASP_MNI.

Por fim, os experimentos reiteraram a hipótese 1 dessa tese. Uma vez que os experimentos destacaram que não há garantia de proporcionalidade entre o número de realocações

e o tempo de operação do guindaste. Além disso, os melhores algoritmos em otimizar o tempo de operação, também não foram os melhores algoritmos em otimizar o número de realocações. As hipótese 3 e 4 também foram confirmadas, visto que os novos métodos propostos conseguiram soluções melhores do que algoritmos competitivos na literatura e em curto tempo computacional, bem como, a resolução do PRC no contexto irrestrito mostra-se compensatória para métodos aproximados e não compensatória em métodos exatos.

6 CONCLUSÃO

Esta seção tem como objetivo apresentar as considerações finais sobre os principais tópicos abordados nesta tese, incluindo as contribuições alcançadas e indicações para trabalhos futuros.

6.1 CONSIDERAÇÕES FINAIS

Esta tese abordou o Problema de Recuperação de Contêineres (PRC), sua resolução é uma importante questão para alcançar eficiência operacional em pátios de um sistema de terminal de contêineres. A revisão literária permitiu o entendimento de várias características relacionadas ao PRC, dentre as quais como o tempo de operação do guindaste é mensurado, o que serviu como base para a realização de um minucioso estudo sobre como melhor mensurar e otimizar o tempo de operação do guindaste. O exame dessas características também foram primordiais para a definição dos objetivos almejados e contribuições deste trabalho para a comunidade acadêmica.

A partir do levantamento feito, verificou-se que, tradicionalmente, o principal objetivo de otimização do PRC é minimizar o número de realocações para retirar sequencialmente todo os contêineres de uma baia. Dado que, empiricamente, tendo um menor número de realocações implica um menor tempo de operação do guindaste. Poucos são os trabalhos que visam minimizar diretamente algum custo de operação, tal como, o consumo de combustível, o tempo de operação ou alguma métrica que mensure melhor o tempo de operação do guindaste em vez do número de realocações. Ademais, entre estes trabalhos, que objetivam minimizar diretamente o tempo de operação, há divergências na computação do tempo de operação do guindaste, devido à ausência de uma definição formal e explicita de uma métrica para computar, de maneira direta e unívoca, o tempo de operação do guindaste. Além disso, não se tem conhecimento de uma análise custo-benefício da resolução do problema entre um contexto restrito ou irrestrito, ou seja, entre as duas classes do problema: PRC restrito e PRC irrestrito.

Tudo isso serviu de estímulo para que o presente estudo definisse novas métricas para computar diretamente o tempo de operação do guindaste, bem como, propusesse novos métodos de otimização exatos e aproximados para encontrar melhores soluções para o PRC restrito e PRC irrestrito. A primeira fase do trabalho consistiu na definição da trajetória do guindaste como métrica para computar os percursos realizados pelo guindaste, e a partir dela definir e computar diretamente o tempo de operação consumido. Além disso, foram apresentados argumentos que atestam que a métrica proposta de trajetória do guindaste mensura melhor o tempo de operação do guindaste. Um destes argumentos é a hipótese 1 desta tese, e sua prova formal foi descrita na Seção 4.2.

A partir da definição das novas métricas, os novos métodos de otimização foram descritos na Seção 4. Primeiramente, os métodos exatos foram apresentados. Estes métodos são um modelo matemático de Programação Linear Inteira (PLI) e dois algoritmos de busca de caminhos: um algoritmo Dijkstra e um algoritmo *A Star* (A*). Este último é uma extensão do algoritmo Dijkstra a partir da definição de uma função heurística, que foi proposta para agilizar o processo de busca. Seguidamente, os métodos aproximados foram retratos, e estes são algoritmos oriundos das meta-heurísticas *Reactive Greedy Randomized Adaptive Search Procedure* (RGRASP) e *Pilot Method* (PM).

A fim de avaliar os métodos de otimização desenvolvidos, foi especificado um cenário de experimentação composto de um conjunto de 1200 instâncias de teste com diferentes configurações e tamanhos. Este conjunto foi construído a partir de um gerador de instâncias da literatura (EXPÓSITO-IZQUIERDO; MELIÁN-BATISTA; MORENO-VEGA, 2012). As instâncias foram geradas de acordo com a combinação das seguintes propriedades de uma baia de contêineres: número de pilhas, número de camadas (ou altura máxima), capacidade de armazenamento e taxa de ocupação.

No cenário de experimentação definido, experimentos e avaliações comparativas foram realizados sobre os algoritmos propostos e algoritmos competitivos na literatura em reduzir o número de realocações e o tempo de operação do guindaste. Em primeiro lugar, os algoritmos Dijkstra e A* foram avaliados na resolução de instâncias sobre o PRC restrito. O algoritmo A* reportou melhores resultados que o algoritmo Dijkstra em termos de número de instâncias solucionadas e tempo de execução. Isto devido pela eficiente função heurística proposta e utilizada no A*. O baixo tempo computacional consumido pelos algoritmos exatos, nas instâncias avaliadas, indicam que os dois algoritmos podem ser aplicados para resolver pequenas e médias instâncias reais do problema, especialmente o A*. O mesmo fato não foi demostrado pela otimização por meio do modelo matemático de PLI. Nesta otimização, o tempo de execução requerido foi muito maior e a quantidade de instâncias solucionadas otimamente foi menor quando comparado ao algoritmo A*. Desse modo, a otimização pelo modelo matemático proposto mostrou-se insuficiente para aplicações em instâncias reais com mais de 19 contêineres.

Posteriormente, foi avaliado, o algoritmo RGRASP_MNI, oriundo da meta-heurística RGRASP e usando do índice de decisão *Min-Max Index* (MNI). Este índice foi adotado no algoritmo pois mostrou os melhores resultados em reduzir o número de realocações dentre os índices avaliados. Os dados da avaliação indicaram que o RGRASP_MNI apresenta boa convergência das soluções embora seja uma abordagem aleatória, e seu procedimento de busca local melhora a qualidade das soluções geradas. Na avaliação comparativa, o RGRASP_MNI, em termos de número de realocações e de tempo de operação, obteve os melhores resultados, mesmo sendo comparado com algoritmos projetados para minimizar o número de realocações e com algoritmos de contexto irrestrito. Sabendo que o RGRASP_MNI foi projetado para otimizar o tempo de operação no PRC restrito.

Os quatro algoritmos provenientes da meta-heurística PM também foram avaliados com algoritmos da literatura, e o algoritmo RGRASP_MNI também foi incluído nessa avaliação. Os algoritmos propostos PILOT e TRIAD correspondem, respectivamente, a meta-heurística PM e a heurística construtiva incorporada nessa meta-heurística. Estes dois foram projetados para solucionar o PRC irrestrito, e os algoritmos PILOT_R e TRIAD_R, respectivamente, condizem com as suas versões para solucionar o PRC restrito. Os resultados computacionais indicaram que o algoritmo PILOT apresentou expressivamente melhores resultados em reduzir o tempo de operação do guindaste em relação a todos os algoritmos avaliados. Mas, na otimização do número de realocações, o algoritmo RGRASP_MNI foi o que apresentou melhores resultados entre todos os avaliados. Os dados também apontaram que os algoritmos da meta-heurística (i.e., PILOT e PILOT_R) melhoraram a qualidade das soluções dos algoritmos que eles incorporaram (i.e., respectivamente, TRIAD e TRIAD_R).

A última avaliação foi sobre a análise custo-benefício da resolução do PRC entre um contexto restrito ou irrestrito. Os resultados dessa avaliação apontam indícios que a resolução do PRC no contexto irrestrito não é compensatória para métodos de otimização exatos, mas é compensatória para métodos de otimização aproximados, confirmando, assim, a hipótese 4 dessa tese. Ao termino desta investigação, os dados destacaram que os 5 algoritmos aproximados, que foram propostos neste trabalho, são capazes de obter soluções ótimas ou soluções bem próximas do ótimo em curto tempo computacional, em especial os algoritmos PILOT e RGRASP_MNI. Além disso, estes algoritmos solucionaram as instâncias maiores, que não puderam ser solucionadas pelos algoritmos exatos.

Por fim, os experimentos realizados nesta tese reiteraram a hipótese 1 da tese, destacando que não há garantia de proporcionalidade entre o número de realocações e o tempo de operação do guindaste, bem como, o melhor algoritmo avaliado em reduzir o número realocações não é o melhor em reduzir o tempo de operação, e vice-versa. As hipóteses 2 e 3 também foram confirmadas pelos experimentos, dado que os novos métodos de otimização propostos conseguiram soluções melhores do que algoritmos competitivos na literatura e em curto tempo computacional. O restante desta seção está organizada como segue. As principais contribuições obtidas neste trabalho são descritas na Seção 6.2. Por fim, os trabalhos futuros que podem dar continuidade ao projeto de pesquisa realizado nesta tese são detalhados na Seção 6.3.

6.2 PRINCIPAIS CONTRIBUIÇÕES

O desenvolvimento desta pesquisa produziu contribuições para as áreas de otimização e sistemas de terminais de contêineres. As principais contribuições deste trabalho são detalhadas a seguir. A ordem em que elas são listadas indica, aproximadamente, a forma como este trabalho foi desenvolvido.

- Definição de novas métricas de otimização para o PRC para melhor mensurar o tempo de operação do guindaste.
- Prova formal de que minimizar o número de realocações não garante a solução com o tempo mínimo de operação do guindaste.
- Um modelo matemático de PLI para fornecer uma descrição matemática do PRC restrito, bem como, possibilitar a resolução ótima deste problema.
- Os algoritmos Dijkstra e A*. Dois algoritmos de busca de caminhos para solucionar otimamente o PRC restrito e irrestrito. Esta contribuição inclui a definição de um função heurística para o algoritmo A*.
- Um algoritmo baseado na meta-heurística RGRASP para solucionar o PRC restrito.
- Os algoritmos TRIAD e TRIAD_R. Dois algoritmos associados a uma heurística construtiva, denominada Tríade, e foram projetados para solucionar, respectivamente, o PRC irrestrito e o PRC restrito.
- Os algoritmos PILOT e PILOT_R. Dois algoritmos baseados na meta-heurística PM para solucionar, respectivamente, o PRC irrestrito e o PRC restrito. Estes dois algoritmos incorporam, respectivamente, os algoritmos TRIAD e TRIAD_R.
- Formalização de um cenário de experimentação formado por um conjunto de 1200 instâncias de teste com diferentes configurações e tamanhos.
- Experimentos de avaliação sobre os principais índices de decisão da literatura.
- Experimentos de avaliação e validação dos novos métodos de otimizados propostos, bem como, comparação com métodos competitivos na literatura.
- Experimentos de avaliação sobre a análise custo-benefício da resolução do PRC entre um contexto restrito ou irrestrito.

6.3 TRABALHOS FUTUROS

Para dar continuidade ao trabalho de pesquisa descrito nesta tese, lista-se, nesta seção, propostas de trabalhos futuros a serem realizadas.

- Avaliar outras métricas de otimização que possam ser computadas a partir da trajetória do guindaste. Deste modo, otimizar no PRC, em conjunto ou não, custos como energia, mão de obra, manutenção, e outras despesas financeiras.
- Estudar e avaliar outros métodos de otimização de maneira a encontrar soluções ainda melhores para o PRC.

- Os métodos de otimização propostos podem ser ampliados de modo que algumas restrições logísticas possam ser consideradas ao problema. Por exemplo, dado contêiner não pode estar próximo outro contêiner, devido a riscos de explosão ou contaminação; duas pilhas vizinhas não podem ter uma certa diferença de altura, devido a algum risco de desmoronamento da pilha vizinha.

- Estender os métodos de otimização propostos para solucionar outras variantes do problema. Por exemplo, a variante trata em (AKYÜZ; LEE, 2014; CASEY; KOZAN, 2012), onde novos contêineres podem entrar na baia enquanto o processo de recuperação do contêiner ocorre. Outro, exemplo, está na variante tratada em (SILVA et al., 2018), onde há apenas dois grupos de prioridade de contêineres: o grupo-alvo a ser recuperado primeiro e o outro grupo formado por todos os contêineres restantes.

- Aplicar as métricas e os métodos de otimização propostos em outros problemas de empilhamento em terminais de contêineres. Por exemplo, o *Premarshalling Problem* (LEHNFELD; KNUST, 2014). Neste problema, dado um conjunto de contêineres armazenados em uma baia, onde cada contêiner é atribuído a um valor correspondente à sua prioridade de saída da baia, o objetivo é reorganizar os contêineres na baia por meio de operações de realocação, de modo que o número de realocações realizadas seja mínimo e ao término de todas realocações necessárias, todos os contêineres podem ser retirados segundo os respectivos valores de prioridade e nenhuma realocação adicional necessita ser efetuada.

- Integrar problemas de otimização sequencialmente relacionados na cadeia de operações em um sistema de terminais de contêineres. Desta forma, o PRC estará atrelado a outros problemas de otimização, e eles serão solucionados de maneira sistêmica e não isolada, diferentemente, de como é realizado na maioria dos estudos da literatura (LEHNFELD; KNUST, 2014).

REFERÊNCIAS

AKYÜZ, M. H.; LEE, C.-Y. A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Nav. Res. Logist.*, v. 61, n. 2, p. 101–118, 2014. ISSN 1520-6750.

AZARI, E. Notes on "a mathematical formulation and complexity considerations for the blocks relocation problem". *Scientia Iranica*, v. 22, n. 6, p. 2722–2728, 2015.

CARLO, H. J.; VIS, I. F.; ROODBERGEN, K. J. Storage yard operations in container terminals: Literature overview, trends, and research directions. *Eur. J. Oper. Res.*, v. 235, n. 2, p. 412–430, 6 2014. ISSN 03772217.

CARRARO, L. A.; CASTRO, L. N. A clonal selection algorithm for the container stacking problem. In: *Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing*. [S.l.]: IEEE, 2011. p. 569–574. ISBN 9781457711237.

CASERTA, M.; SCHWARZE, S.; VOSS, S. A mathematical formulation and complexity considerations for the blocks relocation problem. *Eur. J. Oper. Res.*, v. 219, n. 1, p. 96–104, 5 2012. ISSN 03772217.

CASERTA, M.; VOSS, S.; SNIEDOVICH, M. Applying the corridor method to a blocks relocation problem. *OR Spectr.*, Springer-Verlag, v. 33, n. 4, p. 915–929, 2011. ISSN 0171-6468.

CASEY, B.; KOZAN, E. Optimising container storage processes at multimodal terminals. *J. Oper. Res. Soc.*, Operational Research Society, v. 63, n. 8, p. 1126–1142, 8 2012. ISSN 0160-5682.

DENG, Y.; BARD, J. F. A reactive grasp with path relinking for capacitated clustering. *J. Heuristics*, v. 17, n. 2, p. 119–152, 2011. ISSN 1381-1231.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, v. 1, n. 1, p. 269–271, 1959. ISSN 0029599X.

EXPÓSITO-IZQUIERDO, C.; MELIÁN-BATISTA, B.; MORENO-VEGA, J. M. Pre-marshalling problem: Heuristic solution method and instances generator. *Expert Syst. Appl.*, v. 39, n. 9, p. 8337–8349, 7 2012. ISSN 09574174.

EXPÓSITO-IZQUIERDO, C.; MELIÁN-BATISTA, B.; MORENO-VEGA, J. M. A domain-specific knowledge-based heuristic for the blocks relocation problem. *Adv. Eng. Informatics*, v. 28, n. 4, p. 327–343, 2014. ISSN 14740346.

EXPÓSITO-IZQUIERDO, C.; MELIÁN-BATISTA, B.; MORENO-VEGA, J. M. An exact approach for the blocks relocation problem. *Expert Syst. Appl.*, v. 42, n. 17–18, p. 6408–6422, 2015. ISSN 0957-4174.

FIRMINO, A. S.; SILVA, R. M. A. Reactive grasp to container stacking problem. In: *III Congresso de Matemática Aplicada e Computacional - Sudeste*. [s.n.], 2015. Apresentação no Mini-simpósio: Otimização de Operações Portuárias. Disponível em: <www.sbmac.org.br/arquivos/CMAC-2015-PROGRAMA.pdf>.

FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. An exact approach for the container retrieval problem to reduce crane's trajectory. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2016. p. 933–938. ISBN 978-1-5090-1889-5. ISSN 2153-0009.

FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. Optimising the crane's working time with the pilot method. Artigo em processo de revisão e submetido ao *International Journal of Production Research*. 2018.

FIRMINO, A. S.; SILVA, R. M. A.; TIMES, V. C. A reactive grasp metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics*, p. 1–33, 9 2018. ISSN 1381-1231.

FIRMINO, A. S.; TIMES, V. C.; SILVA, R. M. A.; MATEUS, G. R. Reactive grasp with path relinking for selecting olap views. In: *XLVIII SBPO (2016)*. [s.n.], 2016. Disponível em: <www.sbpo2016.iltc.br/pdf/156026.pdf>.

FORSTER, F.; BORTFELDT, A. A tree search heuristic for the container retrieval problem. In: KLATTE, D.; LÜTHI, H.-J.; SCHMEDDERS, K. (Ed.). *Operations Research Proceedings 2011 SE - 41*. [S.l.]: Springer Berlin Heidelberg, 2012, (Operations Research Proceedings). p. 257–262. ISBN 978-3-642-29209-5.

GALLE, V.; BARNHART, C.; JAILLET, P. A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. *Eur. J. Oper. Res.*, v. 267, p. 467–477, 2018. ISSN 03772217.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A Formal Basis for the Heuristc Dennination of Minimum Cost Paths. *IEEE Trans. on Systems Science and Cyberetics*, v. 4, n. 2, p. 61, 1968. ISSN 0536-1567.

HUSSEIN, M.; PETERING, M. E. H. Genetic algorithm-based simulation optimization of stacking algorithms for yard cranes to reduce fuel consumption at seaport container transshipment terminals. In: *Evolutionary Computation (CEC), 2012 IEEE Congress on*. [S.l.]: IEEE, 2012. p. 1–8. ISBN 978-1-4673-1510-4.

JANSEN, K. The mutual exclusion scheduling problem for permutation and comparability graphs. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.: s.n.], 1998. v. 1373 LNCS, p. 287–297. ISBN 3540642307. ISSN 03029743.

JIN, B.; ZHU, W.; LIM, A. Solving the container relocation problem by an improved greedy look-ahead heuristic. *Eur. J. Oper. Res.*, v. 240, n. 3, p. 837–847, 2015. ISSN 03772217.

JOVANOVIC, R.; VOSS, S. A chain heuristic for the blocks relocation problem. *Comput. Ind. Eng.*, v. 75, p. 79–86, 9 2014. ISSN 03608352.

KIM, K. H.; HONG, G.-P. A heuristic rule for relocating blocks. *Comput. Oper. Res.*, v. 33, n. 4, p. 940–954, 4 2006. ISSN 03050548.

KIM, Y.; KIM, T.; LEE, H. Heuristic Algorithm for Retrieving Containers. *Comput. Ind. Eng.*, 2016. ISSN 03608352.

KU, D.; ARTHANARI, T. S. On the abstraction method for the container relocation problem. *Comput. Oper. Res.*, v. 68, p. 110–122, 2016. ISSN 03050548.

LEE, Y.; LEE, Y.-J. A heuristic for retrieving containers from a yard. *Comput. Oper. Res.*, v. 37, n. 6, p. 1139–1147, 6 2010. ISSN 03050548.

LEHNFELD, J.; KNUST, S. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *Eur. J. Oper. Res.*, v. 239, n. 2, p. 297–312, 12 2014. ISSN 03772217.

LI, J.; YU, H. X. Optimizing retrieval sequencing in container yards. In: *Proceedings - 2010 International Conference on Optoelectronics and Image Processing*. [S.l.]: IEEE, 2010. v. 2, p. 90–92. ISBN 978-1-4244-8683-0.

LIN, D. Y.; LEE, Y. J.; LEE, Y. The container retrieval problem with respect to relocation. *Transp. Res. Part C Emerg. Technol.*, v. 52, p. 132–143, 2015. ISSN 0968090X.

MURTY, K. G.; LIU, J.; WAN, Y.-w.; LINN, R. A decision support system for operations in a container terminal. *Decis. Support Syst.*, v. 39, n. 3, p. 309–332, 5 2005. ISSN 01679236.

OLSEN, M.; GROSS, A. Average case analysis of blocks relocation heuristics. In: GONZÁLEZ-RAMÍREZ, R. G.; SCHULTE, F.; VOSS, S.; DÍAZ, J. A. C. (Ed.). *Computational Logistics. ICCL 2014*. [S.l.]: Springer International Publishing, 2014. p. 81–92. ISBN 978-3-319-11421-7.

PETERING, M. E.; HUSSEIN, M. I. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *Eur. J. Oper. Res.*, v. 231, n. 1, p. 120–130, 11 2013. ISSN 03772217.

PRAIS, M.; RIBEIRO, C. C. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, v. 12, n. 3, p. 164–176, 2000.

RESENDE, M.; RIBEIRO, C. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. [S.l.]: Springer New York, 2016. ISBN 9781493965304.

RIOS-MERCADO, R. Z.; FERNANDEZ, E. A reactive grasp for a commercial territory design problem with multiple balancing requirements. *Comput. Oper. Res.*, v. 36, n. 3, p. 755–776, 2009.

SHESKIN, D. J. *Handbook of Parametric and Nonparametric Statistical Procedures*. 4. ed. [S.l.]: Chapman & Hall/CRC, 2007. ISBN 1584888148, 9781584888147.

SILVA, M. de Melo da; ERDOGAN, G.; BATTARRA, M.; STRUSEVICH, V. The Block Retrieval Problem. *Eur. J. Oper. Res.*, v. 265, n. 3, p. 931–950, 2018. ISSN 03772217.

TALBI, E.-G. *Metaheuristics: From Design to Implementation*. [S.l.]: Wiley Publishing, 2009. ISBN 9780470278581.

TANAKA, S.; TAKII, K. A faster branch-and-bound algorithm for the block relocation problem. *Autom. Sci. Eng.*, v. 13, n. 1, p. 181–190, 1 2016. ISSN 1545-5955.

TING, C. J.; WU, K. C. Optimizing container relocation operations at container yards with beam search. *Transp. Res. Part E Logist. Transp. Rev.*, v. 103, p. 17–31, 2017. ISSN 13665545.

TRICOIRE, F.; SCAGNETTI, J.; BEHAM, A. New insights on the block relocation problem. *Comput. Oper. Res.*, Pergamon, v. 89, p. 127–139, 1 2018. ISSN 0305-0548.

UNCTAD. *Review of Maritime Transport 2015*. UNITED NATIONS PUBLICATION, 2015. ISBN 978-92-1-112892-5. Disponível em: <unctad.org/en/PublicationsLibrary/rmt2015_en.pdf>.

UNCTAD. *Review of Maritime Transport 2017*. UNITED NATIONS PUBLICATION, 2017. ISBN 978-92-1-112922-9. Disponível em: <unctad.org/en/PublicationsLibrary/rmt2017_en.pdf>.

ÜNLÜYURT, T.; AYDIN, C. Improved rehandling strategies for the container retrieval process. *J. Adv. Transp.*, v. 46, n. 4, p. 378–393, 10 2012. ISSN 2042-3195.

VOßS, S.; FINK, A.; DUIN, C. Looking ahead with the pilot method. *Annals of Oper. Res.*, v. 136, n. 1, p. 285–302, Apr 2005. ISSN 1572-9338.

WAN, Y.-w.; LIU, J.; TSAI, P.-C. The assignment of storage locations to containers for a container stack. *Nav. Res. Logist.*, Wiley Subscription Services, Inc., A Wiley Company, v. 56, n. 8, p. 699–713, 12 2009. ISSN 1520-6750.

World Bank. *Container port traffic (TEU: 20 foot equivalent units)*. 2018. [Acessado 16 Dezembro 2018]. Disponível em: <data.worldbank.org/indicator/IS.SHP.GOOD.TU?end=2018&start=2000>.

WU, K.-C.; TING, C.-J. A beam search algorithm for minimizing reshuffle operations at container yards. In: *International Conference on Logistics and Maritime Systems*. [S.l.: s.n.], 2010. p. 15–17.

WU, K.-C.; TING, C.-J. Heuristic approaches for minimizing reshuffle operations at container yard. In: KACHITVICHYANUKUL, H. L. V.; PITAKASO, R. (Ed.). *Asia Pacific Industrial Engineering and Management Systems Conference*. [S.l.: s.n.], 2012. p. 1407–1415.

ZEHENDNER, E.; CASERTA, M.; FEILLET, D.; SCHWARZE, S.; VOSS, S. An improved mathematical formulation for the blocks relocation problem. *Eur. J. Oper. Res.*, v. 245, n. 2, p. 415–422, 2015. ISSN 03772217.

ZHANG, C. *Resource Planning in Container Storage Yard*. Tese (Doutorado) — Hong Kong University of Science and Technology, 2000.

ZHANG, H.; GUO, S.; ZHU, W.; LIM, A.; CHEANG, B. An investigation of ida* algorithms for the container relocation problem. In: GARCÍA-PEDRAJAS, N.; HERRERA, F.; FYFE, C.; BENÍTEZ, J. M.; ALI, M. (Ed.). *Trends in Applied Intelligent Systems SE - 4*. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6096). p. 31–40. ISBN 978-3-642-13021-2.

ZHU, W.; QIN, H.; LIM, A.; ZHANG, H. Iterative deepening a* algorithms for the container relocation problem. *Autom. Sci. Eng.*, v. 9, n. 4, p. 710–722, 10 2012. ISSN 1545-5955.