revista principia
Vol. 61, N. 1
REVISTA DE DIVULGAÇÃO CIENTÍFICA E TECNOLÓGICA DO INSTITUTO FEDERAL DE CIÊNCIA, EDUCAÇÃO E TECNOLOGIA DA PARAÍBA

**DOI:** http://dx.doi.org/10.18265/1517-0306a2022id6730
ARTIGO ORIGINAL

**SUBMETIDO** 11/03/2022
**APROVADO** 09/06/2022
**PUBLICADO ON-LINE** 06/08/2022
**PUBLICADO** 10/01/2024
**EDITOR ASSOCIADO**
Francisco Petrônio Alencar de Medeiros

# *Framework* para apoiar especialistas no problema de sequenciamento de contêiner em terminais portuários

Mayrton Dias de Queiroz [1]*
Ricardo Martins de Abreu Silva [2]

[1] mdq@cin.ufpe.br
[2] rmas@cin.ufpe.br

*Centro de Informática, Universidade Federal de Pernambuco (UFPE), Brasil*

*Autor para correspondência.

**RESUMO:** Com a utilização de contêineres em terminais portuários, surge o desafio de manipulá-los com intuito de obter um menor tempo para realização das operações. Entre os problemas encontrados nesse contexto, é possível destacar o problema de sequenciamento de contêiner, que busca determinar uma sequência de operações com os contêineres realizadas entre o navio e o pátio através dos guindastes. Frente à quantidade de algoritmos de otimização encontrados na literatura, é necessária uma estratégia que possibilite a construção de soluções. O objetivo deste trabalho consiste em identificar uma alternativa capaz de auxiliar os especialistas na construção de soluções. Com a revisão da literatura, foi possível observar a necessidade de um *framework* que fosse genérico e de fácil integração. Para verificar o comportamento do *framework*, foi integrado a este a meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), que tem uma boa aceitação na literatura. Nos resultados, foi possível visualizar as soluções encontradas pela técnica e o tempo gasto ao executar as instâncias. Através do *framework*, foi possível visualizar e analisar as soluções geradas, o que permitiu verificar o impacto das soluções aplicadas ao problema.

**Palavras-chave:** *framework*; problema de sequenciamento de contêiner; terminais portuários.

# *Framework to support specialists in container sequencing problem in port terminals*

**ABSTRACT:** *With the use of containers in port terminals comes the challenge of handling them with the purpose of taking less time in conducting the operations. Among the problems found in this context, it is possible to highlight the container sequencing problem, which seeks to determine a sequence of container operations conducted between the vessel and the yard by using cranes. In view of the amount of optimization algorithms found in the literature, a strategy enabling the construction of solutions becomes necessary. The purpose*

CC BY NC SA
ISSN (on-line): 2447-9187

*of this study is to identify an alternative capable of supporting specialists in the construction of solutions to this problem. The literature review revealed the need for a generic, easily integrated framework. The metaheuristic GRASP (Greedy Randomized Adaptive Search Procedure), which is well accepted in the literature, was integrated into the framework with the aim of investigating its behavior. The results showed both the solutions found by the technique and the time spent in solving the instances. The framework enabled visualization and analysis of the generated solutions, which in turn allowed to measure the impact of the solutions applied to the problem.*

**Keywords:** *container sequencing problem; framework; port terminals.*

## 1 Introdução

Segundo a Comex Stat (Brasil, 2021), em agosto de 2021, o Brasil teve um crescimento de 67,5% no valor das exportações, em comparação ao mesmo mês do ano anterior. Já o crescimento da taxa de importações foi de 56,2%. Assim, faz-se necessária a busca por soluções que reduzam o tempo de realização das operações nos terminais portuários. Com a adoção dos contêineres, foi possível padronizar o espaço para inserir os produtos e as matérias-primas a serem transportados, evitando extravios. Os contêineres podem ter o tamanho de 20 ou 40 pés e, comumente, possuem o formato de um paralelepípedo reto retangular, mas podem variar de formato, caso o conteúdo a ser inserido seja formado por grãos ou líquidos (ABNT, 2002).

Destacamos, no presente trabalho, o problema de sequenciamento de contêiner, qual seja: encontrar um conjunto de operações que o guindaste realiza com os contêineres partindo de uma configuração inicial até a obtenção da configuração de partida desejada. Ao analisar esse problema, – que, segundo Lee, Liu e Chu (2015), pertence à classe NP-difícil –, nota-se que existem diversas possibilidades de operações com os contêineres. Em Meisel e Wichmann (2010) e em Ding *et al*. (2017), os autores usaram heurísticas ou algum método exato, no entanto, é importante uma estratégia que permita integrar e validar as soluções encontradas.

Portanto, faz-se necessária a adoção de uma estratégia capaz de apoiar a construção de soluções para o problema citado. Diante desse contexto, o objetivo deste trabalho consiste em encontrar uma estratégia, um *framework*, para auxiliar os especialistas na construção de soluções para o problema de sequenciamento de contêiner em terminais portuários.

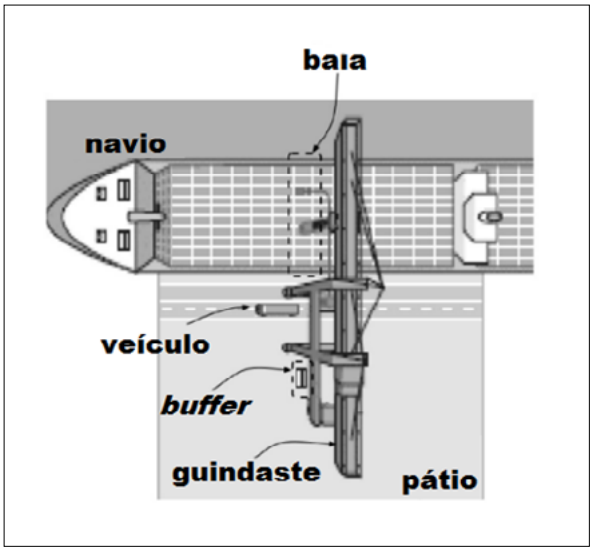
O artigo está organizado da seguinte forma: a seção 2 apresenta o referencial teórico com as principais informações sobre o problema de sequenciamento de contêiner; a seção 3 mostra os passos e resultados da revisão da literatura; a seção 4, por sua vez, apresenta os detalhes da metodologia; por fim, a seção 5 mostra os resultados obtidos e a seção 6, a conclusão e trabalhos futuros.

## 2 Referencial teórico

O problema de sequenciamento de contêiner (*Container Sequencing Problem* – CSP) consiste em encontrar uma sequência de operações com os contêineres, partindo de uma configuração inicial (*Arrival Configuration* – AC) até obter a configuração de partida (*Departure Configuration* – DC). Na Figura 1, é possível ter uma visão geral das

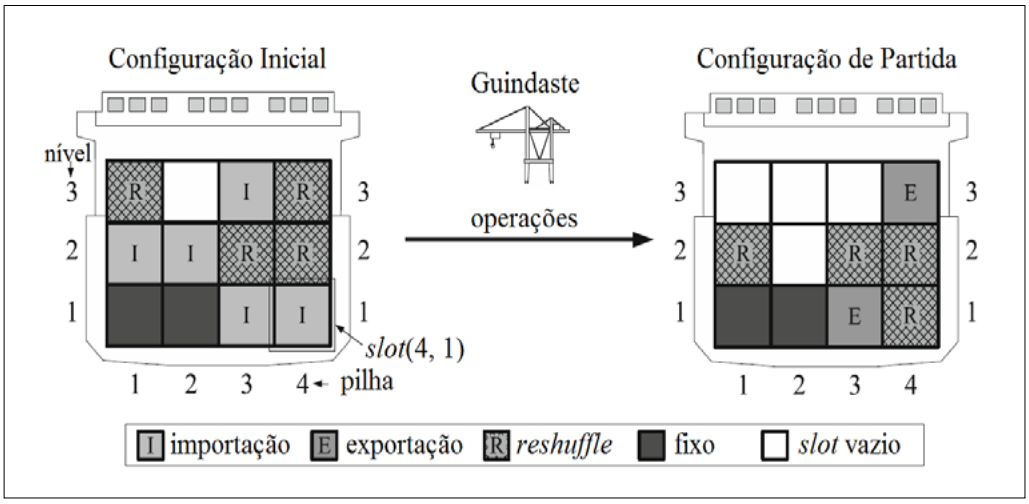
principais áreas disponíveis para a realização das operações com o guindaste (*crane*). O navio (*vessel*) contém diversos contêineres organizados por baias (*bay*), conforme é mostrado na Figura 1. Com relação ao pátio (*yard*), é importante destacar a região em que os veículos transitam, auxiliando no recebimento e disponibilização dos contêineres, conforme a operação realizada pelo guindaste. A região do *buffer* é utilizada para realizar operações com o contêiner de forma temporária, ou seja, é possível colocar um contêiner nessa região, executar outras operações e depois recuperar o contêiner e colocá-lo no local desejado.

**Figura 1 ▶**
Visão geral do guindaste e das regiões disponíveis para realização das operações.
*Fonte: adaptada de Liu et al.* (2015b).




As operações realizadas pelo guindaste consideram uma baia composta por *m* pilhas e *n* níveis. A configuração inicial e a configuração de partida representam o estado desejado para a baia antes e depois das operações, respectivamente. Na Figura 2, é possível observar um exemplo de configuração inicial e de partida; nesse exemplo, nota-se que uma configuração é formada por quatro pilhas e três níveis. Cada contêiner possui uma posição formada pelo par ordenado *slot(i, j)*, sendo $1 \leq i \leq m$ e $1 \leq j \leq n$.

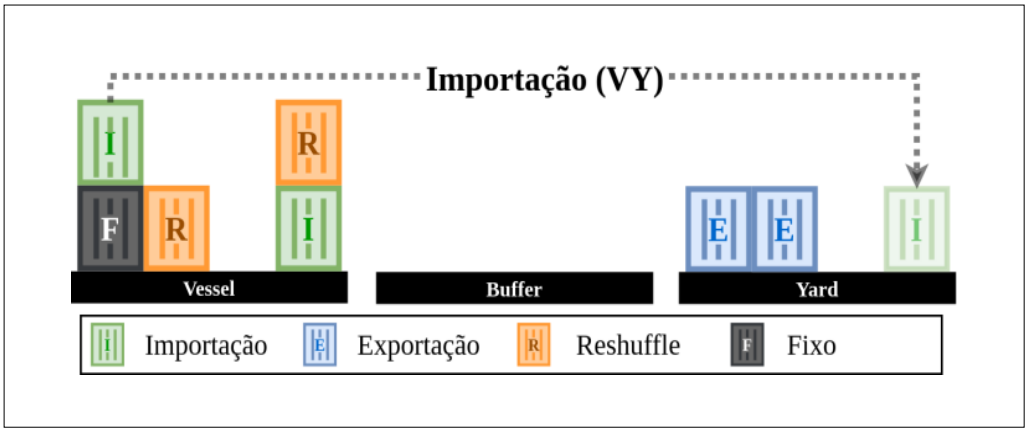
**Figura 2 ▶**
Exemplo de uma configuração inicial e de uma configuração de partida.
*Fonte: adaptada de Meisel e Wichmann (2010)*




Com relação às operações que o guindaste pode realizar com cada tipo de contêiner, é possível destacar:

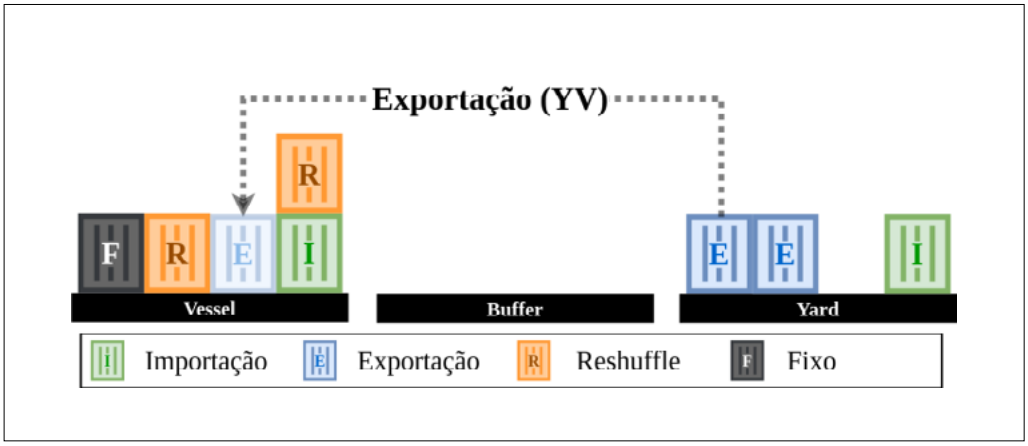
**Importação:** O contêiner de importação contém o conteúdo que será desembarcado. A operação realizada com esse contêiner irá retirá-lo do *vessel* e inseri-lo no *yard*. Na Figura 3, é possível observar a operação realizada com o contêiner de importação, representado pela letra I. Para realizar essa operação, o contêiner precisa estar localizado no topo da pilha e não haver nenhum outro contêiner com nível superior na mesma pilha. No exemplo apresentado, nota-se esse cenário ocorrendo com o contêiner localizado no *slot*(1,2). O contêiner localizado no *slot*(4,1) só estará pronto para realizar a operação após a remoção do contêiner do *slot*(4,2).

Figura 3 ▶
Exemplo de uma operação (VY) com um contêiner de importação.
*Fonte: elaborada pelos autores*




**Exportação:** O contêiner de exportação contém o conteúdo a ser embarcado, de forma que esse contêiner será recuperado do *yard* e transferido pelo guindaste para o *vessel* no *slot* desejado, conforme a configuração de partida. Na Figura 4, é possível observar um exemplo de movimentação com o contêiner de exportação, representado pela letra E. Para a operação de exportação ser realizada, é importante que os contêineres pertencentes à mesma pilha e com o valor de nível inferior ao do *slot* a ser inserido já tenham realizado as operações necessárias – ou seja, que todos os contêineres de importação já tenham sido retirados e todos os contêineres de nível inferior estejam conforme a configuração de partida. Com o *slot* do *vessel* livre, é possível realizar a operação do contêiner. No exemplo da Figura 4, nota-se que o *slot*(3,1) está livre, então o contêiner de exportação do *yard* pode ser movimentado.

Figura 4 ▶
Exemplo de uma operação (YV) com um contêiner de exportação.
*Fonte: elaborada pelos autores*

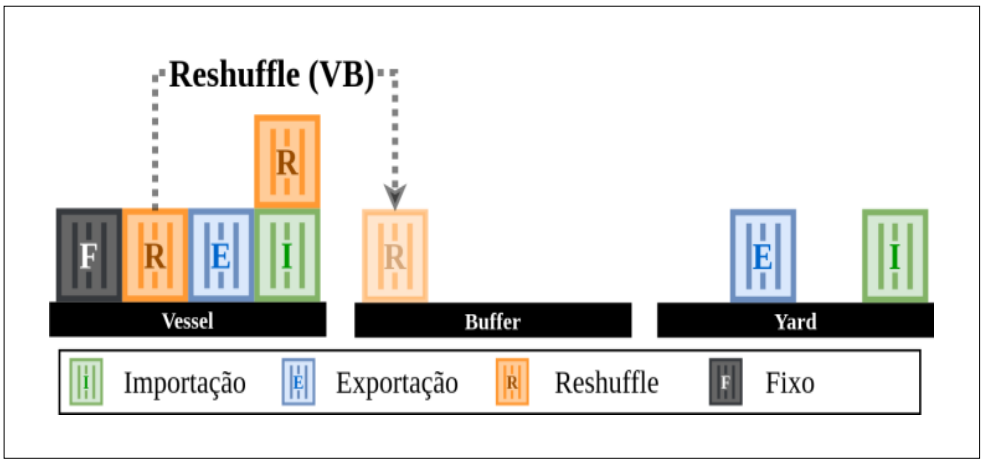



***Reshuffle:*** O contêiner de *reshuffle* contém o conteúdo a ser mantido no *vessel*. Esse contêiner estará presente tanto na configuração de chegada como na configuração

de partida. É possível destacar três operações que podem ser realizadas com esse tipo de contêiner:

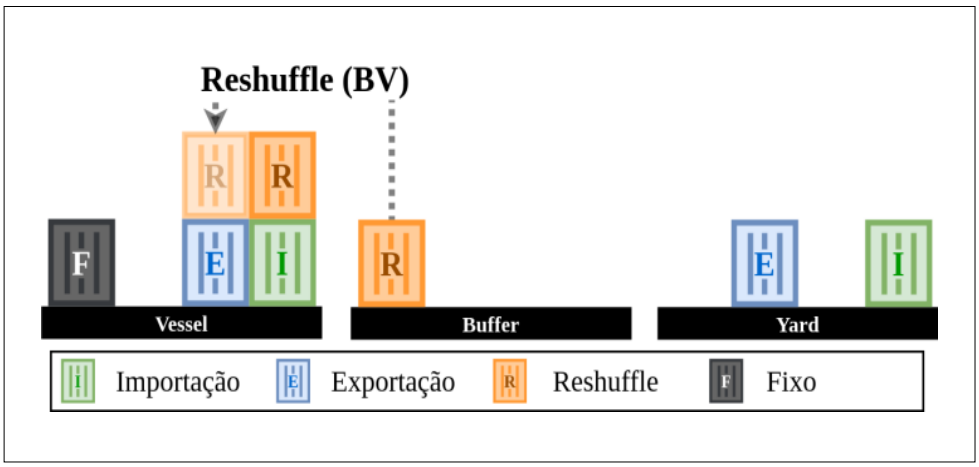
- **Inserir no *buffer*:** O *buffer* é uma região que pode ser utilizada para inserir os contêineres de *reshuffle* enquanto outras operações são realizadas; por exemplo, a operação de importação com os contêineres localizados em níveis inferiores da mesma pilha. Na Figura 5, é possível observar um exemplo de uma operação com o contêiner de *reshuffle*, representado pela letra R. No exemplo, o contêiner do *slot*(2,1) no *vessel* será removido para o *buffer*, pois dessa forma o *slot* ficará livre para inserção de outro contêiner. Já o contêiner do *slot*(4,2) precisa realizar uma operação, visto que ele se encontra acima de um contêiner de importação, que só poderá ser transferido para o *yard* quando todos os *slots* de nível superior estiverem livres;

Figura 5 ▶
Exemplo de uma operação (VB) com um contêiner de *reshuffle*.
*Fonte: elaborada pelos autores*



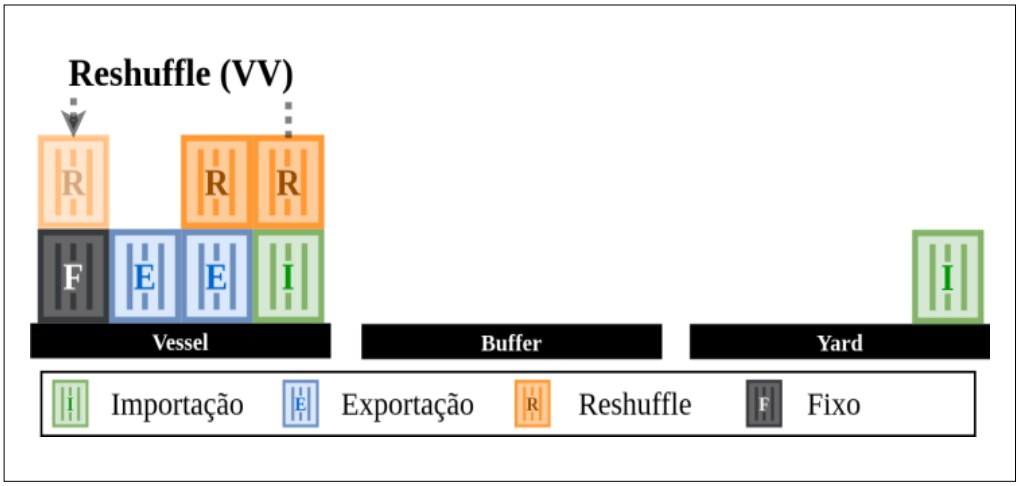

- **Retirar do *buffer*:** Outra operação possível com o contêiner de *reshuffle* é a remoção de um contêiner do *buffer* para o *vessel*, conforme pode ser observado na Figura 6. Após a realização de todas as operações com os contêineres de nível inferior, é possível movimentar o contêiner de *reshuffle* de volta ao *vessel*. No exemplo da figura, o contêiner será retirado do *buffer* e inserido no *slot*(3,2) livre, formando a configuração de partida desejada. É importante destacar que, antes da primeira operação, o *buffer* estava vazio e, após todas as operações – ou seja, chegando à configuração de partida –, nenhum contêiner de *reshuffle* deve permanecer no *buffer*.

Figura 6 ▶
Exemplo de uma operação (BV) com um contêiner de *reshuffle*.
*Fonte: elaborada pelos autores*



- **Movimentação interna:** A movimentação interna envolve o contêiner de *reshuffle* no *vessel*, ou seja, um contêiner em um *slot* será movido para outro *slot* pronto para receber esse contêiner e em conformidade com a configuração de partida. Na Figura 7, é possível observar o exemplo de um contêiner de *reshuffle* sendo retirado do *slot*(4,2) para ser inserido no *slot*(1,2).

Figura 7 ►
Exemplo de uma operação (VV) com um contêiner de *reshuffle*.
*Fonte: elaborada pelos autores*




**Fixo:** O contêiner fixo, representado pela letra F, não sofre ação das operações, pois ele permanece no mesmo *slot* tanto na configuração de chegada como na configuração de partida.

Esses são os tipos de contêineres e as operações realizadas pelo guindaste no CSP. Com relação ao *buffer*, uma solução de uma instância do problema precisa garantir que todos os contêineres inseridos no *buffer* sejam retirados; o *buffer* inicia vazio e, após todas as operações, ele permanece vazio. Já o *yard* inicia com os contêineres de exportação e, após todas as operações, ficarão apenas os contêineres de importação retirados do *vessel*. A configuração e a sequência de recuperação dos contêineres fazem parte do Problema de Recuperação de Contêineres, como pode ser visto no trabalho de Firmino, Silva e Times (2019). Dessa forma, para não sair do escopo do trabalho, o CSP considera a sequência fornecida pelo Problema de Recuperação de Contêineres, pois quando as soluções forem integradas cada um irá contribuir com uma parte do problema.

É possível identificar 5 operações:

- **VY**: contêiner de importação do *Vessel* para o *Yard*;
- **YV**: contêiner de exportação do *Yard* para o *Vessel*;
- **VB**: contêiner de *reshuffle* do *Vessel* para o *Buffer*;
- **BV**: contêiner de *reshuffle* do *Buffer* para o *Vessel*;
- **VV**: contêiner de *reshuffle* do *Vessel* para o *Vessel*.

Cada operação realiza uma movimentação com um contêiner, retirando-o de uma determinada posição e inserindo-o em outra. No entanto, dependendo da região para a qual o contêiner for remanejado, o tempo para realização dessa operação pode variar. Assim, considerando que cada operação $t$ gasta um tempo igual a $d^t$ segundos, na Tabela 1 é possível identificar um exemplo de uma instância conforme Meisel e Wichmann (2010).

Tabela 1 ▶
Tempo necessário para realização de uma operação *t* em segundos. *Fonte: Meisel e Wichmann (2010)*

| $t$ | $d^t$ |
|---|---|
| **VY** | **100** |
| **YV** | **100** |
| **VB** | **100** |
| **BV** | **100** |
| **VV** | **90** |

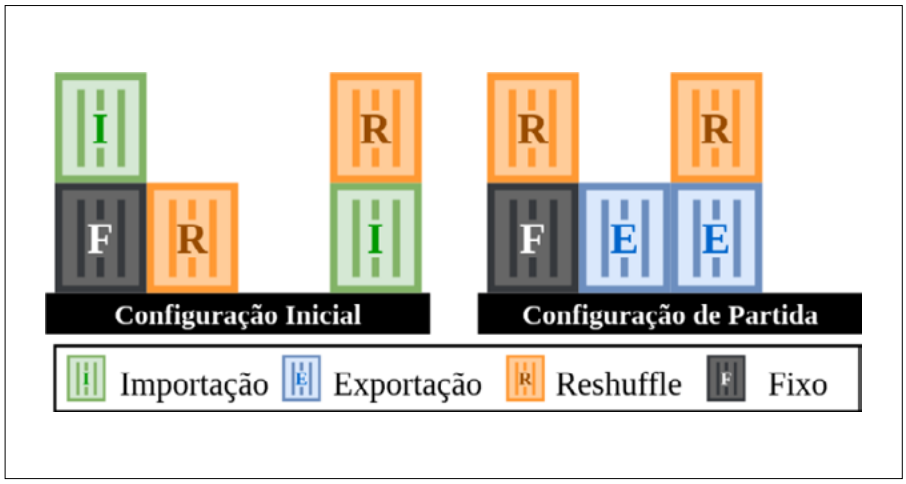
Ao concluir uma operação, o guindaste precisa se posicionar para iniciar a próxima operação. Porém, o tempo de transição do guindaste sem contêiner também pode variar. Na Tabela 2, é possível observar um exemplo, conforme Meisel e Wichmann (2010), que expressa o tempo $d^{tu}$ em segundos, em que $t$ é a operação que está sendo concluída e $u$ é a próxima operação a ser iniciada. Assim, $d^{tu}$ representa o tempo necessário para o guindaste se movimentar sem contêiner a fim de iniciar a próxima operação da solução.

Tabela 2 ▶
Tempo necessário para transição do guindaste de uma operação *t* até uma operação *u* em segundos. *Fonte: Meisel e Wichmann (2010)*

| $t\downarrow u\rightarrow$ | VY | YV | VB | BV | VV |
|---|---|---|---|---|---|
| VY | 20 | 10 | 20 | 10 | 20 |
| YV | 10 | 20 | 10 | 20 | 10 |
| VB | 20 | 10 | 20 | 10 | 20 |
| BV | 10 | 20 | 10 | 20 | 10 |
| VV | 10 | 20 | 10 | 20 | 10 |

Após a descrição das informações principais sobre o CSP, apresenta-se, na Figura 8, um exemplo de solução para uma instância. No exemplo, as configurações possuem 4 pilhas e 2 níveis, dessa forma, $m = 4$ e $n = 2$.

Figura 8 ▶
Exemplo de uma configuração inicial e de uma configuração de partida de uma instância. *Fonte: elaborada pelos autores*



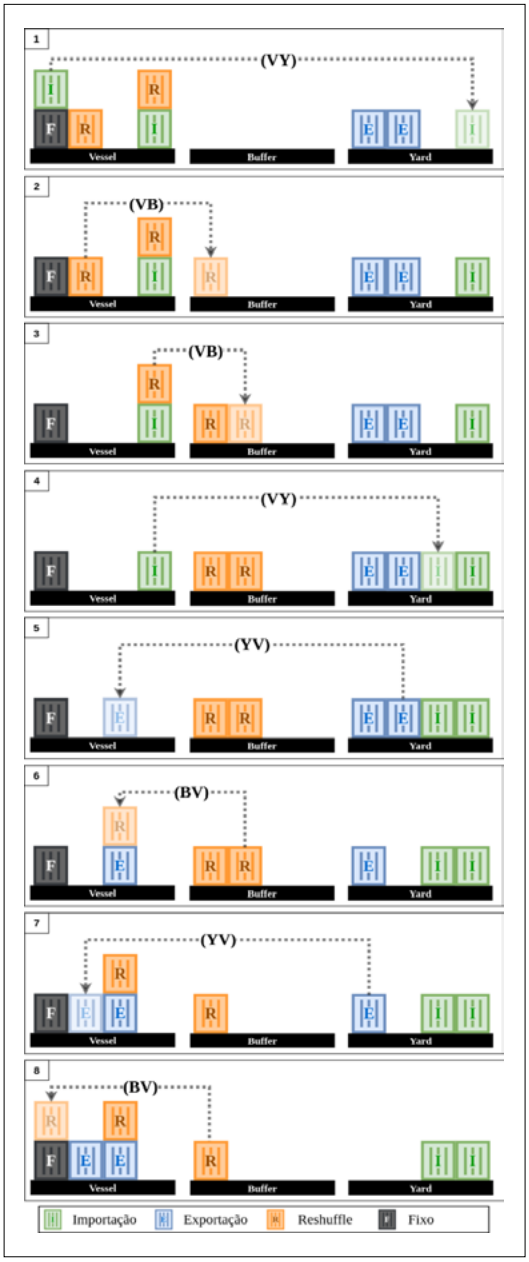

Uma solução possível para esse problema pode ser observada na Tabela 3, que mostra os detalhes das oito operações que compõem uma solução para a instância apresentada na Figura 8.

Tabela 3 ▶
Tempo gasto em cada operação da solução.
*Fonte: dados da pesquisa*

| *N* | $d^{tw}$ | Origem | Destino | Operação | $d^{t}$ |
|---|---|---|---|---|---|
| **1** | **-** | **(1,2)** | Yard | **VY** | **100** |
| **2** | **20** | **(2,1)** | Buffer | **VB** | **100** |
| **3** | **20** | **(4,2)** | Buffer | **VB** | **100** |
| **4** | **20** | **(4,1)** | Yard | **VY** | **100** |
| **5** | **10** | Yard | **(3,1)** | **YV** | **100** |
| **6** | **20** | Buffer | **(3,2)** | **BV** | **100** |
| **7** | **20** | Yard | **(2,1)** | **YV** | **100** |
| **8** | **20** | Buffer | **(1,2)** | **BV** | **100** |
| | | **Total** | | | **930** |

Na Figura 9, é possível obter uma visão geral das modificações realizadas por cada operação.

Figura 9 ▶
Exemplo de uma solução com oito operações.
*Fonte: elaborada pelos autores*



Assim, é possível obter o tempo gasto pelo guindaste para executar essa solução, que, nesse caso, será o somatório de todos os $d^{tu}$ e dos $d^{t}$, o que resulta em 930 segundos. Após a descrição do exemplo de uma solução, é importante destacar que essa é uma das soluções viáveis para a instância apresentada desse problema. Na instância da Figura 8, nota-se a existência de mais de uma possibilidade para realizar a primeira operação: VY com o contêiner de importação do *slot*(1,2); VB com o contêiner de *reshuffle* do *slot*(2,1); YV com o contêiner de exportação para ser inserido no *slot*(3,1); e VB com o contêiner de *reshuffle* no *slot*(4,2). O guindaste tem diversas possibilidades para realizar a próxima operação. Dessa forma, surge o desafio de investigar alternativas possibilitadoras de soluções eficientes, que viabilizem a realização de operações no menor tempo possível. Ao realizar a revisão da literatura, foi possível encontrar trabalhos que propõem a adoção de métodos exatos e heurísticas, conforme será descrito na Seção 3.

## 3 Revisão da literatura

Diante do objetivo do trabalho, foi necessária a realização de uma revisão com o intuito de identificar os principais estudos relacionados a esta pesquisa. Essa fase foi organizada em três etapas (planejamento, condução e resultados), descritas a seguir.

### 3.1 Planejamento

Segundo Kitchenham *et al*. (2009), o planejamento é a etapa de definição de cada passo do protocolo, o qual consiste em: definição das questões de pesquisa; definição da *string* de busca; definição das bases de dados; e definição dos critérios de inclusão e exclusão. Nas subseções 3.1.1 a 3.1.4 são descritos os passos desse protocolo.

#### 3.1.1 Definição das questões de pesquisa

Inicialmente, são definidas as Questões de Pesquisa (QPs), que serão respondidas durante a fase de condução. Neste trabalho, foram definidas cinco QPs:

1) Qual o método de otimização encontrado no trabalho?
    - Exato;
    - Uma heurística;
    - Uma simulação;
    - Outro;
    - Não se aplica.

2) Qual o algoritmo adotado?
    - Programação dinâmica;
    - Programação linear;

- *Greedy Randomized Adaptive Search Procedure* (GRASP);
- Algoritmo híbrido;
- Outro;
- Não se aplica.

3) Quais são as restrições ou características destacadas pelo trabalho analisado?
   - Quantidade de ciclos;
   - Quantidade operações;
   - Outro;
   - Não se aplica.

4) Usa quais tipos de contêiner?
   - Importação;
   - Exportação;
   - *Reshuffle*;
   - Fixo;
   - Outro;
   - Não se aplica.

5) Qual a movimentação do guindaste utilizada no trabalho?
   - Ciclo simples;
   - Ciclo duplo;
   - Não se aplica.

### 3.1.2 Definição da *string* de busca

Após a definição das QPs, definiu-se uma *string* de busca composta com as principais palavras-chave sobre o tema: (*Container Sequencing Problem*) *AND [*(*Quay Crane*) *OR* (*Double Cycling*)].

### 3.1.3 Definição das bases de dados

Em seguida, definiu-se as principais bases de dados, na área de Ciência da Computação, disponíveis no Portal de Periódicos da CAPES. Foi possível identificar sete bases de dados, como apresentado no Quadro 1 (próxima página). Para cada base selecionada, a *string* de busca foi configurada a fim de que atendesse às especificações de cada base.

Quadro 1 ▶
Bases de dados adotadas.
*Fonte: dados da pesquisa*

| Base de dados | Endereço WEB |
|---|---|
| Arxiv | https://arxiv.org/ |
| EBSCOhost | https://www.ebsco.com/ |
| Google Scholar | https://scholar.google.com/ |
| IEEE | https://ieeexplore.ieee.org/ |
| Science Direct | https://www.sciencedirect.com/ |
| Scopus | https://www.scopus.com/ |
| Springer | https://link.springer.com/ |

### 3.1.4 Definição dos critérios de inclusão e exclusão

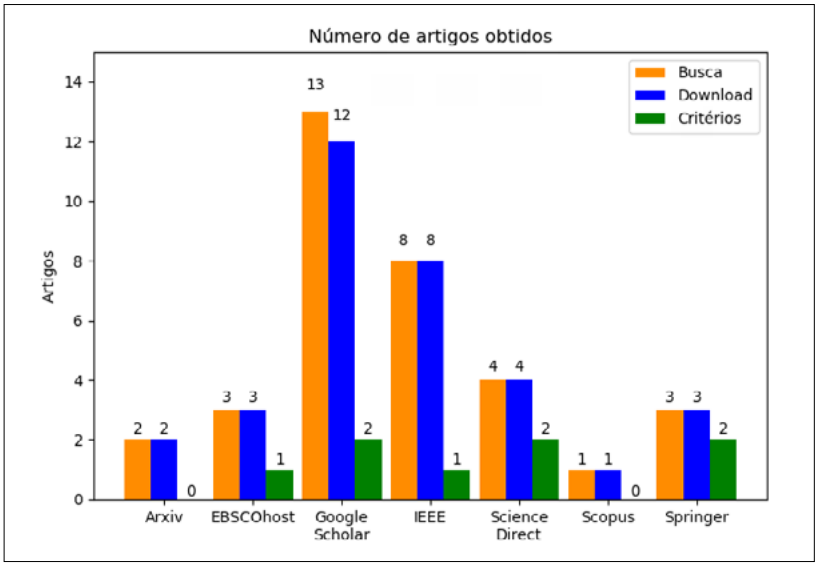
Para concluir a etapa de planejamento, foram definidos os critérios de inclusão e exclusão a serem atendidos por todos os artigos. Os critérios definidos como de inclusão foram: i) estudos completos publicados em revistas ou conferências sobre o CSP; ii) estudos teóricos ou experimentais com o objetivo de apresentar conceitos para o entendimento da área; iii) estudos acessíveis eletronicamente.

Como critérios de exclusão foram adotados: i) estudos que não estejam relacionados ao CSP; ii) estudos que não respondem a nenhuma das QPs; iii) artigos duplicados, ou seja, aqueles encontrados em mais de uma base de dados; iv) artigos convidados, tutoriais e relatórios técnicos não submetidos aos critérios de avaliação das conferências ou revistas.

## 3.2 Condução

Na fase de condução, foi colocado em prática o planejamento definido na seção 3.1. Dessa forma, a *string* de busca foi inserida em cada base de dados, respeitando as especificações de cada base. De posse dos resultados da busca, foi realizado o *download* dos estudos disponíveis. Em seguida, os trabalhos passaram pelos critérios de inclusão e exclusão. Na Figura 10, é possível observar a quantidade de trabalhos encontrados em cada base de dados durante a pesquisa, a realização do *download* e a aplicação dos critérios de inclusão e exclusão.

Figura 10 ▶
Quantidade de artigos em cada base de dados durante as fases.
*Fonte: dados da pesquisa*

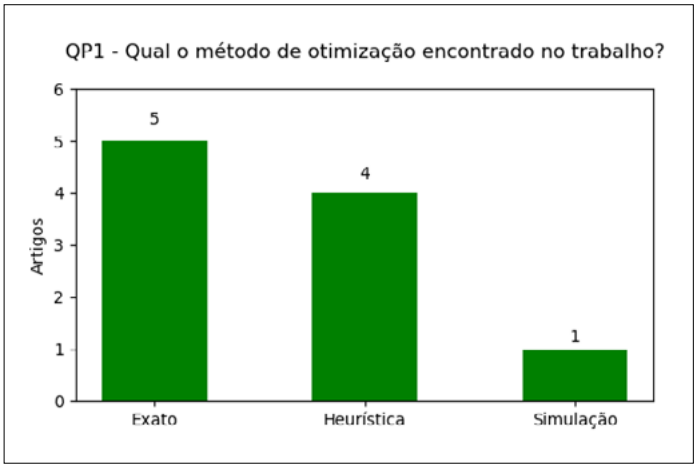


Após a aplicação dos critérios de inclusão e exclusão, foram identificados oito artigos. Em seguida, iniciou-se a leitura dos artigos objetivando responder às cinco questões de pesquisa.

## 3.3 Resultados

Nesta última etapa, serão apresentadas as respostas obtidas para as cinco QPs. Na Figura 11, é possível observar um gráfico de barras correspondente à QP1, que se refere ao método de otimização adotado nos trabalhos analisados. Observando o gráfico, nota-se que cinco trabalhos fizeram uso de métodos exatos, entre eles o trabalho de Zhang e Kim (2009), que utilizaram Programação Linear para realizar comparações com uma heurística que usa Busca Local. Meisel e Wichmann (2010) também utilizaram Programação Linear, porém realizaram comparações com a heurística GRASP. Lee, Liu e Chu (2015) fizeram uso do algoritmo de Sidney, um algoritmo exato que busca minimizar o tempo de operações dos contêineres explorando o duplo ciclo dos guindastes. Liu *et al*. (2015a) usaram uma abordagem híbrida de Programação Inteira com uma heurística chamada *Internal-Reshuffle-Dense* – esse algoritmo tem como objetivo maximizar o número de operações com contêiner de *reshuffle*. Por fim, no trabalho de Zheng *et al*. (2020), os autores utilizaram Programação Dinâmica.

Figura 11 ▶
Resultado da Questão de Pesquisa 1.
*Fonte: dados da pesquisa*

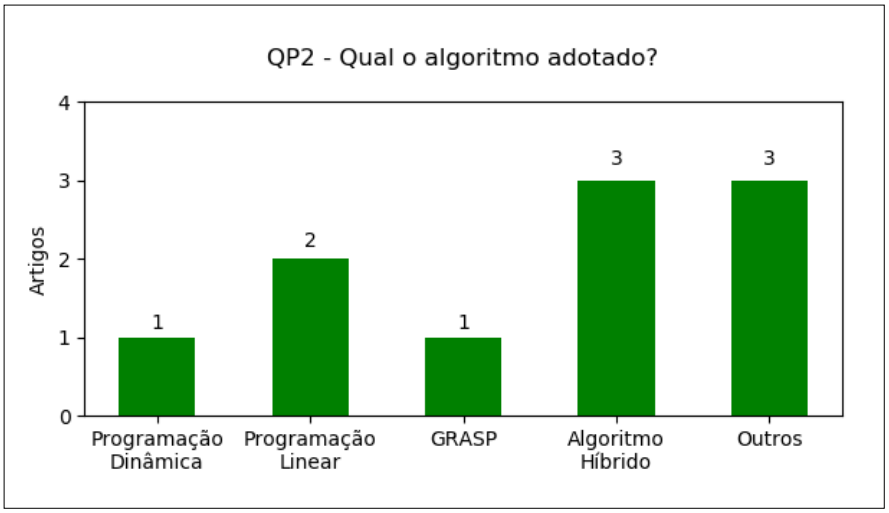



Na alternativa heurística da QP1, foram identificados quatro trabalhos. Conforme mostra o gráfico, os trabalhos adotaram alguma heurística, quais sejam: o trabalho de Meisel e Wichmann (2010), que fez uso do algoritmo GRASP; o trabalho de He, Wang e Zheng (2011), que utilizou uma abordagem híbrida na qual os autores definiram o algoritmo em duas partes – a primeira utiliza a *Johnson's Rule* e a segunda, o *Particle Swarm Optimization*; o trabalho de Liu *et al*. (2015b), no qual foi adotado um algoritmo híbrido que utilizou Programação Inteira e a heurística *Internal-Reshuffle-Dense*, conforme citado anteriormente; e o trabalho de Ding *et al*. (2017), em que se adotou um algoritmo híbrido que possuía duas etapas: inicialmente, foi aplicado o algoritmo *Neighborhood Search* e, em seguida, o algoritmo *Tabu Search*. Na alternativa "Simulação" da QP1, observou-se que o trabalho de Ahmed *et al*. (2021) apresentou um modelo para simular as abordagens de ciclo simples e duplo dos guindastes.

A QP2 foi elaborada para identificar os algoritmos adotados pelos oito trabalhos. No gráfico da Figura 12, é possível observar os principais algoritmos encontrados. O trabalho que utilizou Programação Dinâmica foi Zheng *et al*. (2020). Já os trabalhos de Zhang e

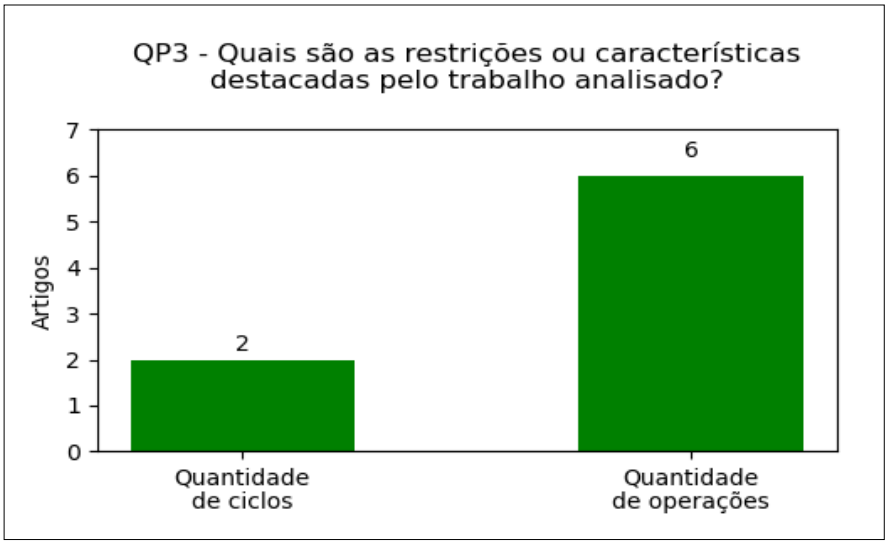
Kim (2009) e Meisel e Wichmann (2010) utilizaram Programação Linear para comparar os resultados com suas heurísticas. Com relação aos Algoritmos Híbridos, foram identificados três trabalhos: o de He, Wang e Zheng (2011), que utilizou o algoritmo *Johnson's Rule* com o algoritmo *Particle Swarm Optimization*; o de Liu *et al*. (2015a), que fez uso do algoritmo de Programação Inteira e do algoritmo chamado *Internal-Reshuffle-Dense*; e o trabalho de Ding *et al*. (2017), que apresentou o algoritmo *Neighborhood Search* combinado com o algoritmo *Tabu Search*. Na alternativa "Outro", foi possível elencar três trabalhos: o trabalho de Zhang e Kim (2009) adotou o algoritmo de busca local para comparar com o algoritmo de Programação Linear; o trabalho de Lee, Liu e Chu (2015) fez uso do *Sidney's algorithm*; e o trabalho de Ahmed *et al*. (2021) elaborou fluxogramas com as etapas das simulações.

Figura 12 ▶
Resultado da Questão de Pesquisa 2.
*Fonte: dados da pesquisa*



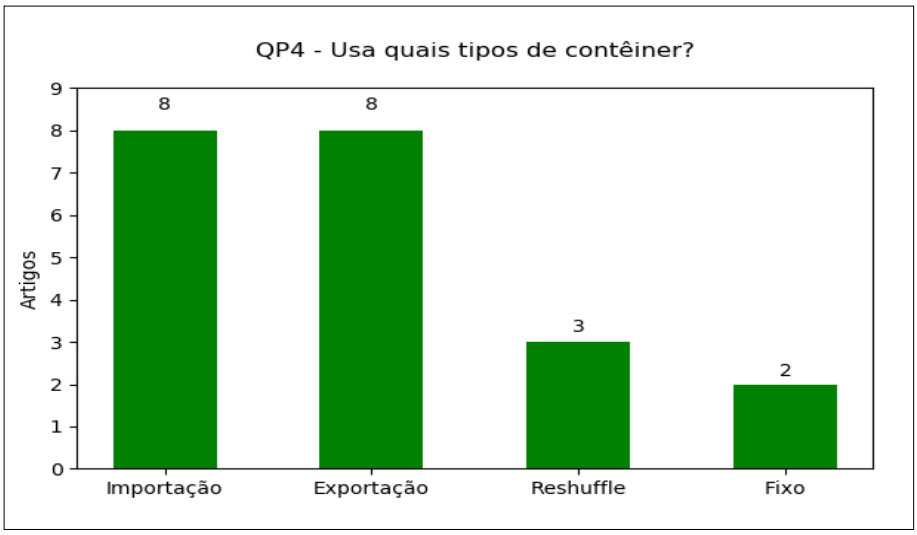

A QP3 foi elaborada para identificar as restrições ou características dos trabalhos. Foi possível destacar duas características principais consideradas pelos trabalhos, como mostra a Figura 13. A primeira foi a quantidade de ciclos realizados pelos guindastes na solução encontrada nos experimentos. Dessa forma, os trabalhos de Zhang e Kim (2009) e Zheng *et al*. (2020) buscaram soluções visando à redução da quantidade de ciclos. A segunda característica identificada foi a quantidade de operações realizadas pelas soluções; essa característica foi investigada pelos seis artigos restantes.

Figura 13 ▶
Resultado da Questão de Pesquisa 3.
*Fonte: dados da pesquisa*



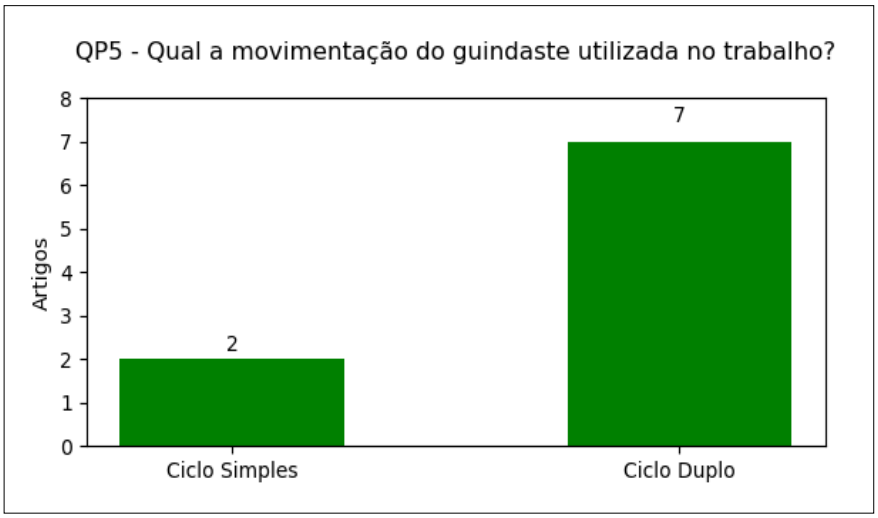
Na QP4, foi verificado o tipo de contêiner utilizado nos trabalhos. Como já era esperado, os oito trabalhos utilizaram os contêineres de importação e exportação, conforme mostra a Figura 14. Já os trabalhos de Meisel e Wichmann (2010) e Liu *et al*. (2015a) utilizaram também os contêineres de *reshuffle* e o fixo; no trabalho de Ding *et al*. (2017), entretanto, os autores consideraram o contêiner de *reshuffle*, mas não descreveram se fizeram uso de contêiner fixo.

Figura 14 ▶
Resultado da Questão de Pesquisa 4.
*Fonte: dados da pesquisa*




Com a QP5, buscou-se identificar quais movimentações realizadas pelos guindastes foram consideradas pelos trabalhos (Figura 15). Nessa questão, foi possível observar que o trabalho de He, Wang e Zheng (2011) utilizou uma abordagem com ciclo simples, enquanto o trabalho de Ahmed *et al*. (2021) realizou simulações fazendo uso do ciclo simples e do ciclo duplo e, nos resultados, fez a comparação de produtividade para cada tipo de movimentação. Com relação ao ciclo duplo, foi possível identificar esse tipo de movimentação nos demais trabalhos.

Figura 15 ▶
Resultado da Questão de Pesquisa 5.
*Fonte: dados da pesquisa*




Com a realização desta revisão da literatura, foi possível identificar os principais trabalhos alinhados ao CSP, bem como as principais técnicas e características adotadas, o que permite obter uma visão mais abrangente para identificar possíveis melhorias.

## 4 Método da pesquisa

Diante do objetivo deste trabalho, que consiste em identificar uma alternativa para apoiar especialistas na construção de soluções para o CSP, e após a revisão da literatura, surgiu a proposta de criação de um *framework* genérico que possibilitasse a integração com outros problemas relacionados à otimização em terminais portuários. O *framework* busca facilitar a construção dos experimentos, possibilitando que os especialistas observem qual o impacto de suas soluções ao aplicá-las no problema, bem como acompanhar cada operação durante a execução da solução. Para a realização dos experimentos, pode-se seguir os passos detalhados na próxima seção.

### 4.1 Passos para realização de um experimento

Visto que o *framework* possibilitará a construção dos experimentos, é importante definir quais os passos necessários:

- **Passo 1 - Definição das configurações da instância:** neste passo, serão definidos os dados de entrada para os algoritmos. Será necessário definir a quantidade de pilhas, o número de níveis e valores como: o tempo para realizar cada operação; o tempo gasto com a transição entre as operações; a configuração inicial e de partida;
- **Passo 2 - Definição do algoritmo de otimização:** diante da quantidade de algoritmos que podem construir soluções para o problema, neste passo será definido qual algoritmo deve ser utilizado no experimento;
- **Passo 3 - Definição dos parâmetros e execução dos algoritmos:** neste passo, serão definidos os valores dos parâmetros dos algoritmos, como quantidade de iterações, condição de parada, número de repetições com sementes distintas, entre outros. Em seguida, os algoritmos serão executados conforme os parâmetros definidos;
- **Passo 4 - Obtenção dos resultados:** no último passo, serão apresentados os resultados adicionais, tais como o comportamento da solução conforme as iterações, os valores obtidos, a média, o desvio padrão, o tempo de execução, a variância e os valores máximo e mínimo.

### 4.2 Etapas da metodologia

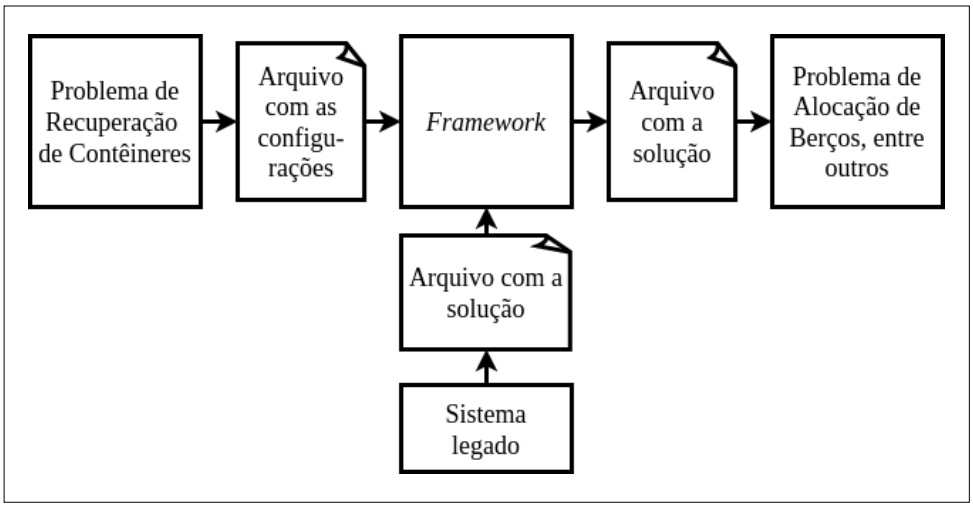
Diante do que foi descrito sobre os experimentos do *framework*, foi necessário definir três etapas a serem realizadas na metodologia, com o intuito de obter o *framework*.

#### 4.2.1 Etapa 1: Descrição conceitual do *framework*

Para possibilitar a integração do *framework* com sistemas legados, algoritmos implementados em outras linguagens de programação e problemas relacionados, foi necessário investigar uma alternativa para possibilitar a comunicação entre esses elementos. Com a revisão da literatura, observou-se que os trabalhos que abordam o CSP adotam as instâncias utilizadas no trabalho de Meisel e Wichmann (2010). Conforme

mostra a Figura 16, o *framework* recebe um arquivo com as configurações que podem utilizar informações geradas por outros problemas, como o Problema de Recuperação de Contêineres. Esse arquivo segue o padrão adotado por Meisel e Wichmann (2010), visto que este é utilizado em trabalhos relacionados para fazer comparações.

Figura 16 ▶
Visão geral da comunicação do *framework* com os demais problemas e sistemas legados.
*Fonte: elaborada pelos autores*




Com relação à técnica de otimização, é importante destacar que novos algoritmos podem ser incorporados ao *framework* ou os algoritmos de sistemas legados podem ser integrados ao *framework* por meio dos arquivos. Nos dois casos, faz-se necessário que os algoritmos sejam capazes de realizar a leitura do arquivo de configuração e retornem o arquivo com a solução. De posse do arquivo, será possível visualizar a solução encontrada executando cada operação, possibilitando aos especialistas observar a ação gerada após a execução de cada operação.

Para verificar o comportamento do *framework*, neste trabalho foi escolhido o algoritmo GRASP (Resende; Ribeiro, 2010), conforme as modificações propostas no trabalho de Meisel e Wichmann (2010), pois esse algoritmo tem uma boa aceitação na literatura para ser comparado com outras propostas de algoritmos. Na Figura 17, é possível notar o algoritmo GRASP alinhado ao CSP.

Figura 17 ▶
Algoritmo GRASP aplicado ao CSP.
*Fonte: elaborada pelos autores*

**Algoritmo 1:** GRASP

**Entrada:** iterações, AC, DC
**Saída:** melhorSolução

```
1  início
2      melhorSolução = Infinity
3      para it de 0 até iterações faça
4          solução = Construção(AC, DC)
5          solução = BuscaLocal(solução)
6          se tempo(solução) < tempo(melhorSolução) então
7              melhorSolução = solução
8          fim
9      fim
10     retorna melhorSolução
11 fim
```

Em linhas gerais, o algoritmo possui a função *Construção* e a função *BuscaLocal*. Na função *Construção*, conforme mostra a Figura 18, a função *construir* identifica quais possibilidades podem ser realizadas para as próximas operações. No exemplo apresentado com a instância da Figura 8, viu-se que existiam quatro possibilidades para a realização da próxima operação: VY com o contêiner de importação do *slot*(1,2); VB com o contêiner de *reshuffle* do *slot*(2,1); YV com o contêiner de exportação para ser inserido no *slot*(3,1); e VB com o contêiner de *reshuffle* no *slot*(4,2). Assim, a função *construir* é responsável por identificar quais são as possibilidades de operações e inseri-las em uma lista de candidatas.

Figura 18 ▶
Função *Construção* do GRASP.
*Fonte: elaborada pelos autores*

**Algoritmo 2:** CONSTRUÇÃO

```
Entrada: AC, DC
Saída: solução
1 início
2     atual = AC
3     enquanto atual <> DC faça
4         construir()
5         selecionar()
6         atualizar()
7     fim
8     retorna solução
9 fim
```

De posse da lista de candidatas, a função *selecionar* irá escolher uma operação entre as possibilidades presentes na lista de candidatas. Inicialmente, a função obtém o valor de *w* referente ao tempo gasto em cada operação somado ao tempo gasto com as operações realizadas. O intuito desse método é selecionar uma operação aleatoriamente da lista de candidatas, de forma que as operações que tenham um menor tempo tenham maior probabilidade de serem selecionadas. Dessa forma, na Equação (1) é possível obter o valor *S* que corresponde ao inverso de *w*. Em seguida, a função aplica a Equação (2), obtendo o valor total de todos os elementos da lista. De posse do valor total, a função irá calcular a probabilidade de cada elemento da lista com base na Equação (3). Ou seja, será feita a divisão do inverso de *w* pelo valor total de todos os elementos.

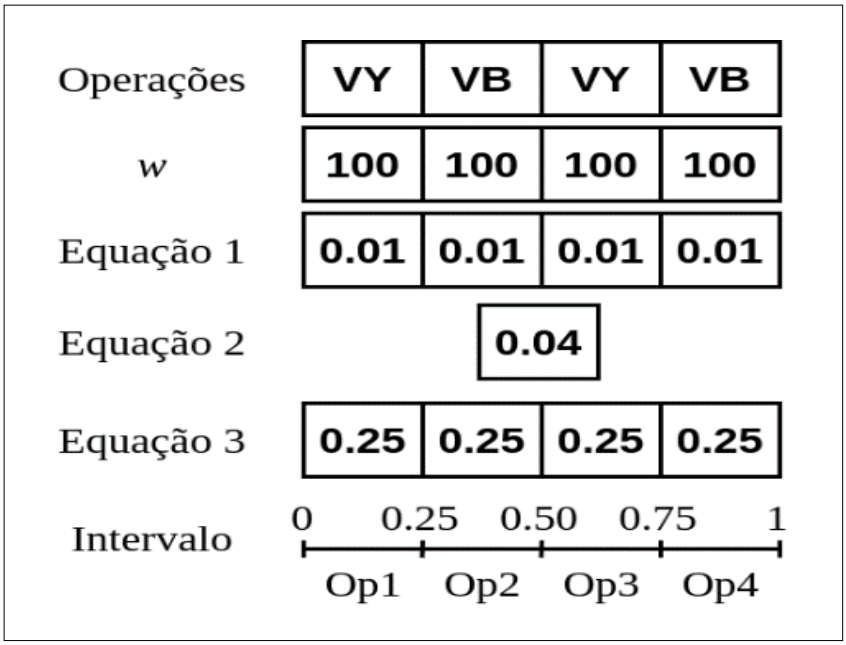
$$S = \frac{1}{w} \tag{1}$$

$$S^T = \sum_{1}^{N} \frac{1}{w_i} \tag{2}$$

$$S_i = \frac{S}{S^T} \tag{3}$$

No exemplo citado, para realizar a primeira operação, a função *construir* identificou quatro possíveis operações, então a função *selecionar* irá aplicar as três equações conforme mostra a Figura 19. Nesse exemplo, a lista de candidatas possui quatro operações, de forma que foram aplicadas as três equações, obtendo-se o intervalo [0,1]. Após a obtenção desse intervalo, é possível selecionar uma das soluções candidatas. Então, ao gerar um número aleatório, será verificado a qual região do intervalo ele pertence e qual a operação correspondente. Suponha-se que o número aleatório obtido foi 0.47, que pertence ao intervalo [0.25, 0.50]; dessa forma, o item 2 da lista de candidatas foi selecionado, ou seja, a Operação 2, que corresponde a VB com o contêiner de *reshuffle* do *slot*(2,1). É importante destacar que essa função possibilita que todas as movimentações possíveis na configuração atual dos contêineres tenham chance de serem selecionadas, mas priorizando as operações que gastam menos tempo para serem realizadas.

Figura 19 ▶
Exemplo de aplicação das equações durante a execução da função *selecionar*.
*Fonte: elaborada pelos autores*




Em seguida, é executada a função *atualizar*, que, após a seleção da operação, será inserida na solução atual e retirada da lista de candidatas. Após a atualização, é verificado se o estado atual representado pela variável *atual* é idêntico à configuração de partida. Caso seja verdadeiro, o *enquanto* finaliza e segue para a próxima função; caso seja diferente, as funções *construir*, *selecionar* e *atualizar* serão executadas novamente até se obter a configuração de partida, pois a condição de serem idênticas indica que a solução com as operações obtidas é uma solução viável para o problema.

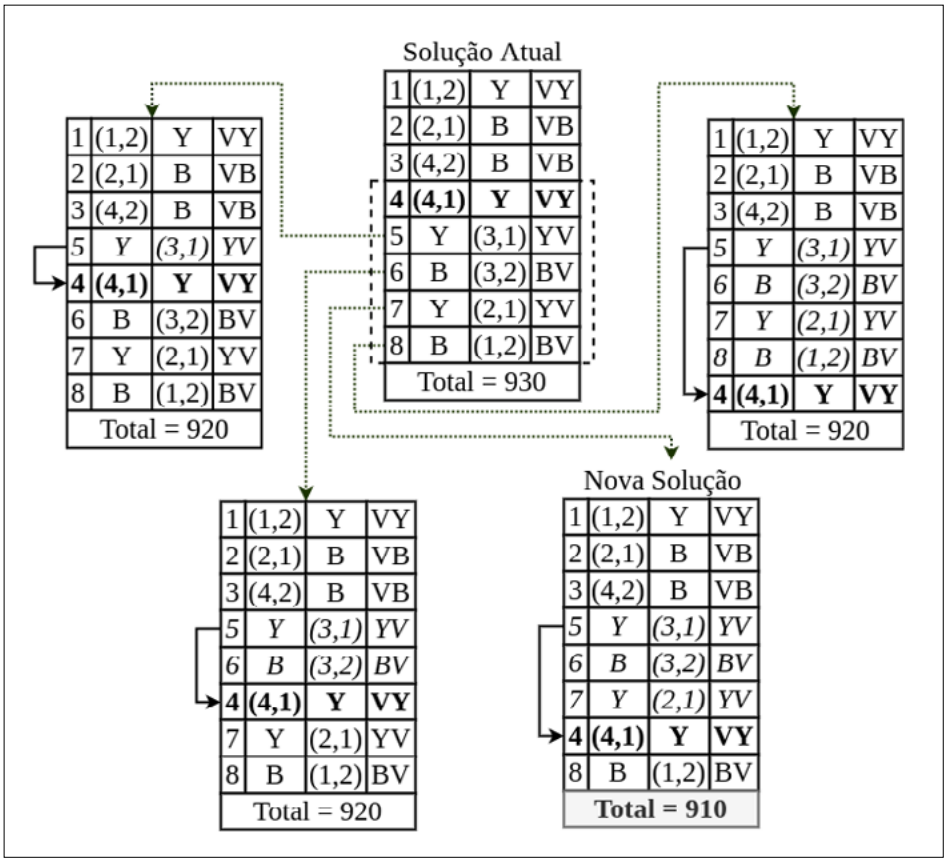
Com a finalização da função *Construção*, o algoritmo possuirá uma solução válida para o problema: ele inicia a função *BuscaLocal*, conforme o algoritmo apresentado na Figura 20 (próxima página), que executa duas funções. A função *mover* realiza alterações na solução atual de forma que uma operação é deslocada para posições anteriores ou posteriores, dentro de um intervalo seguro de movimentações. Esse passo é realizado com cada operação da solução atual; assim que inicia uma determinada operação, o algoritmo identifica um intervalo que essa operação pode percorrer. Para determinar o intervalo que a operação irá percorrer, deve-se observar a pilha dos *slots* da operação atual.

Figura 20 ▶
Função *BuscaLocal* do GRASP.
*Fonte: elaborada pelos autores*

```
Algoritmo 3: BuscaLocal
  Entrada: solução
  Saída: solução
1 início
2 |  solução = mover(solução)
3 |  solução = transformar(solução)
4 |  retorna solução
5 fim
```

Na Figura 21, é possível notar um exemplo do funcionamento desse passo durante a execução da operação 4. Ao analisar a operação 4, é possível identificar que o intervalo será da operação 5 até a operação 8. Esse intervalo foi definido porque, nesse exemplo, a operação 3 fez uma ação na pilha 4, a mesma pilha da operação 4; as operações anteriores à operação 4 não foram consideradas, pois geram uma solução inviável; no entanto, as operações após a operação 4 foram todas consideradas, pois não afetam a pilha 4.

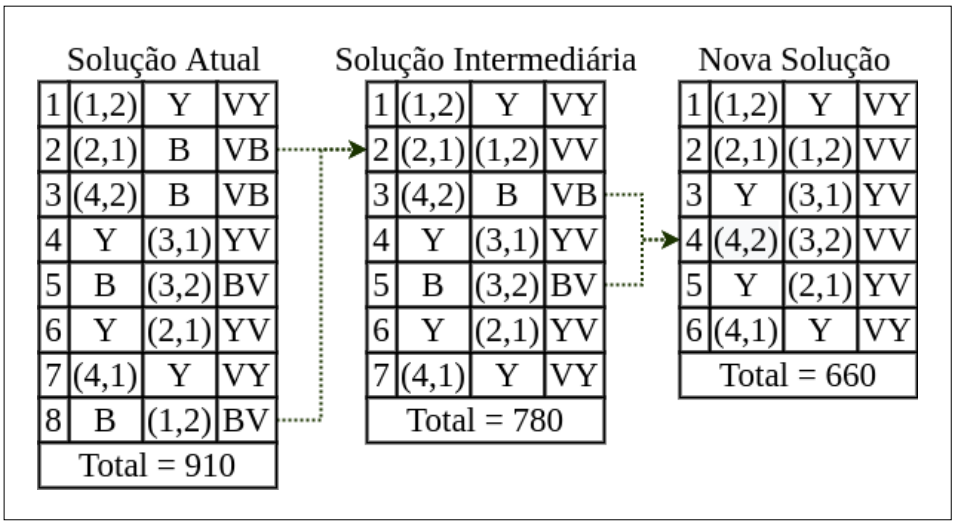
Figura 21 ▶
Exemplo da geração de novas soluções ao mover a operação 4 durante a execução da função *mover*.
*Fonte: elaborada pelos autores*




Visto que o intervalo varia entre 5 e 8, então a função *mover* irá gerar quatro soluções – em cada solução a posição da operação 4 será deslocada em uma unidade, conforme mostra a Figura 21. A solução atual do exemplo possui um tempo total de 930 segundos. Ao realizar os deslocamentos, foram encontradas três soluções com 920 segundos e uma solução com 910 segundos. Então, essa nova solução na qual a operação 4 foi deslocada para a posição da operação 7 será armazenada, pois possui o menor tempo entre as soluções geradas nessa iteração.

Em seguida, a função *transformar* inicia a transformação de duas operações, sendo uma VB e outra BV, em uma nova operação VV que será a combinação das duas primeiras. Essa transformação gera uma solução com o tempo total menor e uma solução com um número menor de operações. Na Figura 22, é possível notar um exemplo da aplicação de duas transformações. Inicialmente, o algoritmo identificou que poderia transformar as operações 2 e 8 e inseriu a nova operação na posição 2, pois o *slot*(1,2) não sofre mais alterações após a operação 1, já o *slot*(2,1) sofre ação da operação 6; portanto, a nova operação VV não poderia ficar na posição da operação 8, pois geraria uma solução inviável.

Figura 22 ▶
Exemplo da geração de novas soluções por meio das transformações durante a função *transformar*.
*Fonte: elaborada pelos autores*




Em seguida, foi aplicada a transformação com as operações 3 e 5 da solução intermediária, gerando a operação 4 da nova solução. Nesse caso, a nova operação deve ficar na posição em que estava a operação 5 da solução intermediária, visto que a operação 4 possui o *slot*(3,1) e a nova operação VV possui o *slot*(3,2); então, a nova operação deve ficar após a operação 4 da solução intermediária. Atualizando os identificadores das operações na nova solução, nota-se que esta contém seis operações, em que a operação 2 foi gerada das operações 2 e 8 e a operação 4 foi gerada das operações 3 e 5. Com relação ao tempo total, a solução atual, com 930 segundos, foi atualizada para a nova solução que possui 660 segundos.

Ao executar essas duas funções, a função *BuscaLocal* retorna a solução encontrada, que é armazenada na variável *solução*. Em seguida, será executado o *se*, verificando se a solução armazenada possui um tempo total menor se comparado à solução atual armazenada em *melhorSolução*. Caso seja menor, o valor da solução é atualizado e segue para a próxima iteração do *para*, que irá repetir todos os processos até o valor da variável *iterações*.

### 4.2.2 Etapa 2: Implementação do *framework*

Após a descrição conceitual, o *framework* foi implementado em Python em conjunto com a biblioteca Tkinter para a construção da interface gráfica. A meta-heurística GRASP aplicada ao problema foi desenvolvida em C++ e com a biblioteca Mersenne Twister para geração de números pseudoaleatórios.

### 4.2.3 Etapa 3: Validação do *framework*

Com o intuito de analisar o comportamento das soluções encontradas pelo algoritmo de otimização, nesta etapa foi inserido o exemplo apresentado na Figura 8, analisando-se o passo a passo do *framework*. A seguir, são apresentados os resultados obtidos pelo GRASP com as instâncias adotadas no trabalho de Meisel e Wichmann (2010).

## 5 Resultados da pesquisa

Nesta seção, serão descritos os resultados obtidos com a utilização do *framework* para encontrar uma solução de uma instância; em seguida, este será utilizado com o conjunto de instâncias adotado na literatura.

### 5.1 Inserção de dados de configuração e visualização da solução

Conforme descrito na seção 4.1, para criar um experimento no *framework* faz-se necessário seguir quatro passos. Primeiro, será carregado o arquivo de configurações para esse experimento (Figura 8). O arquivo utilizado pode ser observado no Apêndice A.1. Na Figura 23, é possível observar a tela inicial do *framework* e o menu de opções; no item *Load Configuration*, será inserido o arquivo de configuração citado. Ao carregar o arquivo, o *framework* fará a leitura e apresentará para o especialista a aplicação dos dados inseridos, conforme mostra a Figura 24.

Figura 23 ▶
Tela inicial e menu de opções do *Framework*.
*Fonte: arquivo dos autores*

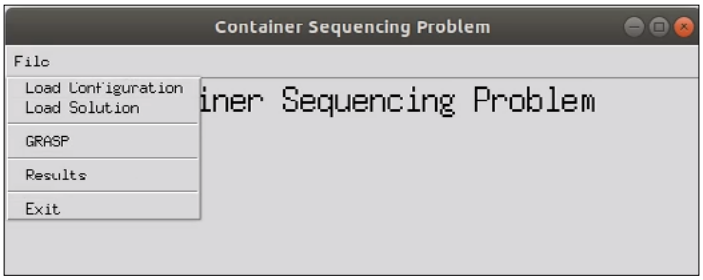
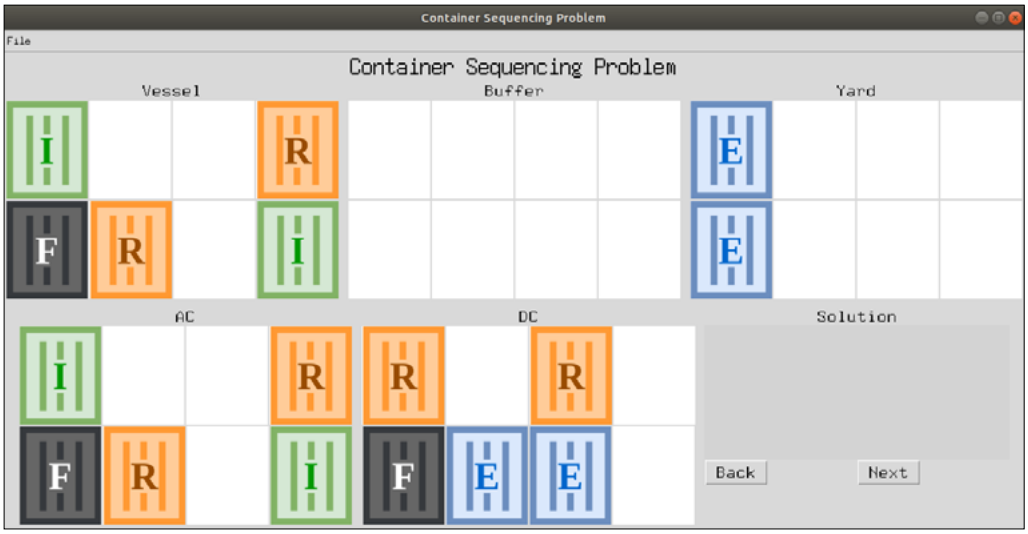
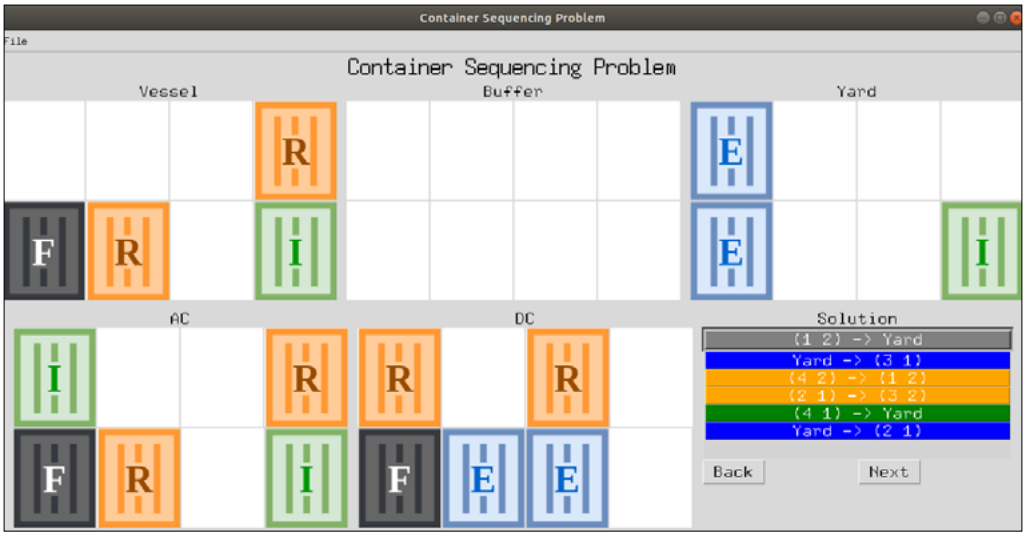



Figura 24 ▶
Tela de visualização do arquivo de configuração.
*Fonte: arquivo dos autores*



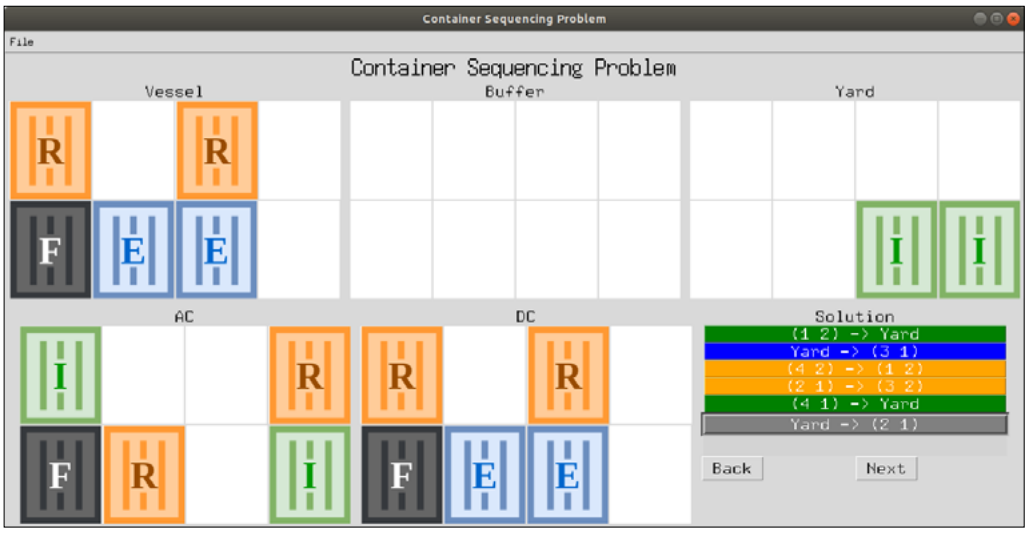
O próximo passo é a escolha do algoritmo de otimização. Neste trabalho, foi adotado o algoritmo GRASP, conforme descrito na Figura 17. No passo seguinte, são determinados os valores dos parâmetros do algoritmo. Para esse experimento, foram utilizadas 100 iterações, com a semente 141592, que se refere às seis primeiras casas decimais do número π. Com a execução do algoritmo, foi gerado o arquivo com a solução conforme o Apêndice A.2; é possível visualizar essa solução clicando no item *Load Solution* do menu. Na Figura 25, é possível observar a tela com a solução, ao clicar no botão *Next*. Assim, o especialista consegue observar a ação que cada operação está realizando com o contêiner.

Figura 25 ▶
Tela com a visualização da operação 1.
*Fonte: arquivo dos autores*




Na Figura 26, nota-se que a solução é viável, pois o *vessel* possui a mesma configuração de DC.

Figura 26 ▶
Tela com a visualização da operação 6.
*Fonte: arquivo dos autores*




Adicionalmente, todos os contêineres de exportação que estavam no *yard* estão no *vessel*, todos os contêineres de importação foram movidos para o *yard* e o *buffer* ficou vazio. Na Tabela 4, nota-se uma solução encontrada pelo algoritmo com tempo total de 630 segundos para essa instância.

Tabela 4 ▶
Tempo gasto em cada operação da solução.
*Fonte: dados da pesquisa*

| N | $d^{tu}$ | Origem | Destino | Operação | $d^{tu}$ |
|---|---|---|---|---|---|
| **1** | **-** | **(1,2)** | Yard | **VY** | **100** |
| **2** | **10** | Yard | **(3,1)** | **YV** | **100** |
| **3** | **10** | **(4,2)** | **(1,2)** | **VV** | **90** |
| **4** | **10** | **(2,1)** | **(3,2)** | **VV** | **90** |
| **5** | **10** | **(4,1)** | Yard | **VY** | **100** |
| **6** | **10** | Yard | **(2,1)** | **YV** | **100** |
| | | **Total** | | | **630** |

Tabela 5 ▼
Média aritmética, tempo (s), desvio padrão e valores máximo e mínimo de cada instância.
*Fonte: dados da pesquisa*

## 5.2 Desempenho do algoritmo de otimização

Para verificar o comportamento do GRASP, a meta-heurística recebeu como entrada o conjunto de instâncias utilizado na literatura, proposto por Meisel e Wichmann (2010). Na Tabela 5, é possível observar a síntese dos resultados obtidos.

| Nome | Média | Tempo | Desvio padrão | Mínimo | Máximo |
|---|---|---|---|---|---|
| CSP_n10_I40_E70_R00 | 12900,4 | 8,3 | 45,5 | 12790 | 13010 |
| CSP_n10_I40_E70_R02 | 13401,8 | 8,9 | 34,5 | 13290 | 13470 |
| CSP_n10_I40_E70_R04 | 13842,2 | 9,6 | 42,7 | 13700 | 13910 |
| CSP_n10_I40_E70_R06 | 14319,9 | 10,3 | 66,4 | 14100 | 14410 |
| CSP_n10_I40_E70_R08 | 14824,8 | 11,0 | 50,5 | 14690 | 14930 |
| CSP_n10_I40_E70_R10 | 15251,3 | 11,8 | 54,2 | 15030 | 15350 |
| CSP_n10_I40_E70_R12 | 16017,2 | 12,9 | 75,6 | 15730 | 16110 |
| CSP_n10_I40_E70_R14 | 16479,5 | 13,7 | 49,9 | 16350 | 16570 |
| CSP_n10_I40_E70_R16 | 16704,9 | 14,1 | 41,3 | 16540 | 16770 |
| CSP_n15_I40_E70_R00 | 29299,4 | 29,7 | 85,1 | 29110 | 29450 |
| CSP_n15_I40_E70_R02 | 30527,8 | 33,4 | 43,2 | 30430 | 30630 |
| CSP_n15_I40_E70_R04 | 31503,8 | 35,9 | 68,9 | 31270 | 31630 |
| CSP_n15_I40_E70_R06 | 32655,8 | 39,5 | 84,9 | 32330 | 32810 |
| CSP_n15_I40_E70_R08 | 33629,0 | 43,6 | 90,2 | 33390 | 33790 |
| CSP_n15_I40_E70_R10 | 34806,4 | 50,0 | 83,1 | 34570 | 34930 |
| CSP_n15_I40_E70_R12 | 36008,6 | 54,2 | 61,6 | 35870 | 36130 |
| CSP_n15_I40_E70_R14 | 37000,0 | 56,2 | 57,7 | 36830 | 37130 |
| CSP_n15_I40_E70_R16 | 37936,0 | 64,2 | 89,0 | 37710 | 38050 |

O nome de cada instância apresenta as informações sobre o número de pilhas e níveis e a quantidade de contêineres. Por exemplo, o arquivo CSP_n10_I40_E70_R02 possui 10 pilhas, 10 níveis e, com relação à quantidade de contêineres, 40 de importação, 70 de exportação e 2 de *reshuffle*. Cada linha da tabela mostra uma instância, que apresenta 10 variações dos contêineres nas configurações, de forma que cada variação foi executada 30 vezes pelo algoritmo com sementes distintas. Para a criação das sementes, foram utilizadas as casas decimais de π, ou seja, a cada 6 casas decimais uma nova semente foi gerada, por exemplo, $s_1 = 141592$, $s_2 = 653589$, e assim sucessivamente.

## 6 Conclusão

Com a elaboração deste trabalho, foi possível observar uma alternativa capaz de auxiliar os especialistas na construção de soluções para o CSP. Através do *framework*, os especialistas podem construir suas soluções e verificar o impacto na solução final, bem como observar a solução de uma forma mais intuitiva. Adicionalmente, o *framework* possibilitou a comunicação com outros problemas de otimização em terminais portuários que podem fornecer novos dados para serem incorporados a novas técnicas de otimização, bem como podem consumir as informações geradas pelo *framework* para usar em outros problemas que tenham ligação com o CSP.

Durante a revisão da literatura, foi possível identificar os principiais trabalhos relacionados, o que possibilitou a construção do *framework* de forma que ele fosse genérico a ponto de se integrar com outras técnicas de otimização ou sistemas legados, além de padronizar os arquivos necessários para a comunicação com outros sistemas.

Quanto às propostas de trabalhos futuros, pode-se citar: a integração com outras técnicas de otimização para obter melhores resultados; a análise do impacto das soluções geradas pelo *framework* em um contexto global em relação ao terminal portuário; e a realização de um estudo para verificar a experiência de usuário com os especialistas.

## Agradecimentos

Os autores agradecem à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e a Universidade Federal de Pernambuco (UFPE) por financiarem a realização deste trabalho.

## Financiamento

Esta pesquisa não recebeu financiamento externo.

## Conflito de interesses

Os autores declaram não haver conflito de interesses.

## Referências

ABNT – ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO 6346**. Contêineres de carga: códigos, identificação e marcação. Rio de Janeiro: ABNT, 2002.

AHMED, E.; EL-ABBASY, M. S.; ZAYED, T.; ALFALAH, G.; ALKASS, S. Synchronized scheduling model for container terminals using simulated double-cycling strategy. **Computers & Industrial Engineering**, v. 154, 107118, 2021. DOI: https://doi.org/10.1016/j.cie.2021.107118.

BRASIL. Ministério da Economia. **Comex Stat**. Exportação e importação geral. Brasília, DF: Ministério da Economia, 2021. Disponível em: http://comexstat.mdic.gov.br/pt/geral. Acesso em: 20 set. 2021.

DING, Y.; WEI, X.-J.; YANG, Y.; GU, T.-Y. Decision support based automatic container sequencing system using heuristic rules. **Cluster Computing**, v. 20, n. 1, p. 239-252, 2017. DOI: https://doi.org/10.1007/s10586-016-0678-2.

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**, v. 25, n. 2, p. 141-173, 2019. DOI: https://doi.org/10.1007/s10732-018-9390-0.

HE, X.; WANG, S.; ZHENG, J. A hybrid heuristic algorithm for integrated large-capacity quay crane scheduling problem. *In*: INTERNATIONAL CONFERENCE ON COMPUTER RESEARCH AND DEVELOPMENT, 3., 2011, Shanghai. **Proceedings** […]. Shanghai: IEEE, 2011. p. 309-312. DOI: https://doi.org/10.1109/ICCRD.2011.5764026.

KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering: a systematic literature review. **Information and Software Technology**, v. 51, n. 1, p. 7-15, 2009. DOI: https://doi.org/10.1016/j.infsof.2008.09.009.

LEE, C.-Y.; LIU, M.; CHU, C. Optimal algorithm for the general quay crane double-cycling problem. **Transportation Science**, v. 49, n. 4, p. 957-967, 2015. DOI: https://doi.org/10.1287/trsc.2014.0563.

LIU, M.; CHU, F.; ZHANG, Z.; CHU, C. A polynomial-time heuristic for the quay crane double-cycling problem with internal-reshuffling operations. **Transportation Research Part E: Logistics and Transportation Review**, v. 81, p. 52-74, 2015a. DOI: https://doi.org/10.1016/j.tre.2015.06.009.

LIU, M.; WANG, S.; CHU, C.; ZHENG, F. A branch-and-price framework for the general double-cycling problem with internal-reshuffles. *In*: IEEE INTERNATIONAL CONFERENCE ON NETWORKING, SENSING AND CONTROL, 12., 2015, Taipei. **Proceedings** […]. Taipei: IEEE, 2015b. p. 304-308. DOI: https://doi.org/10.1109/ICNSC.2015.7116053.

MEISEL, F.; WICHMANN, M. Container sequencing for quay cranes with internal reshuffles. **OR Spectrum**, v. 32, n. 3, p. 569-591, 2010. DOI: https://doi.org/10.1007/s00291-009-0191-6.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: advances, hybridizations, and applications. *In*: GENDREAU, M.; POTVIN, J.-Y. (ed.). **Handbook of metaheuristics**. Boston: Springer, 2010. p. 283-319. DOI: https://doi.org/10.1007/978-1-4419-1665-5_10.

ZHANG, H.; KIM, K. H. Maximizing the number of dual-cycle operations of quay cranes in container terminals. **Computers & Industrial Engineering**, v. 56, n. 3, p. 979-992, 2009. DOI: https://doi.org/10.1016/j.cie.2008.09.008.

ZHENG, F.; PANG, Y.; LIU, M.; XU, Y. Dynamic programming algorithms for the general quay crane double-cycling problem with internal-reshuffles. **Journal of Combinatorial Optimization**, v. 39, n. 3, p. 708-724, 2020. DOI: https://doi.org/10.1007/s10878-019-00508-9.

## Apêndice A

### A.1 Arquivo com as configurações

```
m = 4;
n = 2;

d = [ 90.0, 100.0, 100.0, 100.0, 100.0];

dd = [
[ 10.0, 10.0, 10.0, 20.0, 20.0],
[ 20.0, 20.0, 20.0, 10.0, 10.0],
[ 20.0, 20.0, 20.0, 10.0, 10.0],
[ 10.0, 10.0, 10.0, 20.0, 20.0],
[ 10.0, 10.0, 10.0, 20.0, 20.0]
];

AC = [
[ 1, 0, 0, 3],
[ 4, 3, 0, 1]
];

DC = [
[ 3, 0, 3, 0],
[ 4, 2, 2, 0]
];
```

## A.2 Arquivo com a solução

1 ( 1, 2) (-1, 0) VY [00]
2 (-1, 0) ( 3, 1) YV [10]
3 ( 0, 3) ( 1, 2) VV [10]
3 ( 1, 1) ( 0, 2) VV [10]
1 ( 1, 3) (-1, 0) VY [10]
2 (-1, 0) ( 1, 1) YV [10]