unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

RAFAEL LOPES BONFIM

DESENVOLVIMENTO DE EXTENSÕES PARA UM SISTEMA DE CARREGAMENTO E DESCARREGAMENTO DE NAVIOS

Guaratinguetá
2013

RAFAEL LOPES BONFIM

DESENVOLVIMENTO DE EXTENSÕES PARA UM SISTEMA DE CARREGAMENTO E DESCARREGAMENTO DE NAVIOS

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Licenciatura em Matemática da faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Licenciatura em Matemática.

Orientador: Prof. Dr. Galeno José de Sena

Guaratinguetá
2013

Bonfim, Rafael Lopes
B713d Desenvolvimento de extensões para um sistema de carregamento e descarregamento de navios / Rafael Lopes Bonfim– Guaratinguetá : [s.n], 2013.
42 f. : il.
Bibliografia: f. 41

Trabalho de Graduação em Licenciatura em Matemática – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2013.
Orientador: Prof. Dr. Galeno José de Sena

1. Simulated annealing (Matemática) 2. Otimização matemática
I. Título

CDU 519.863

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

DESENVOLVIMENTO DE EXTENSÕES PARA UM SISTEMA DE CARREGAMENTO E DESCARREGAMENTO DE NAVIOS

RAFAEL LOPES BONFIM

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM LICENCIATURA EM MATEMÁTICA"**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM LICENCIATURA EM MATEMÁTICA.

Profª. Drª. ANA PAULA MARINS CHIARADIA
Coordenadora

BANCA EXAMINADORA:

Prof. Dr. GALENO JOSÉ DE SENA
Orientador/UNESP-FEG

Prof. Dr. JOSÉ FEDERICO VIZCAINO GONZÁLEZ
UNESP-FEG

Prof. Dr. MARCOS ANTONIO PEREIRA
UNESP-FEG

Dezembro de 2013

Aos meus pais

AGRADECIMENTOS

Primeiramente a Deus.

Aos meus pais, por todo o apoio.

Ao orientador, Prof. Dr. Galeno José de Sena, por toda a paciência e disponibilidade, como também por contribuir de maneira eficaz no meu desenvolvimento acadêmico.

A todos os professores da FEG e funcionários.

“A fé é a expectativa certa de coisas esperadas, a demonstração evidente de realidades, embora não observadas.”

Hebreus 11:1

BONFIM, R. L. **Desenvolvimento de extensões para um sistema de carregamento e descarregamento de navios.** 2013. Trabalho de Graduação (Graduação em Licenciatura em Matemática) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2013.

RESUMO

Um dos motivos que influencia a eficiência de um terminal portuário especializado em movimentação de contêineres está no processo de carregamento e descarregamento dos mesmos. A ideia fundamental é determinar um plano, para cada porto, que visa minimizar o número de movimentos total de carregamentos e descarregamentos necessários durante o percurso do navio em todos os portos, a fim de se economizar tempo e conseqüentemente reduzir custos. O objetivo desse trabalho foi apresentar melhorias em um programa, escrito em linguagem C, utilizado para resolver o Problema de Carregamento e Descarregamento de Navios, um problema NP-Completo, através da meta-heurística Simulated Annealing. Com as melhorias empregadas conseguiu em média uma redução de 50% no tempo necessário para resolver o problema em 45 instancias analisadas, sendo que em um grupo específico dessas instancias a redução média atingiu 77%, e em 10 instancias houve inclusive uma melhora na solução.

PALAVRAS-CHAVE: Otimização Combinatória. Meta-heurísticas. Recozimento Simulado. Problema de Carregamento de Contêineres em Terminais Portuários. PCCTP.

BONFIM, R. L. **Developing extensions for a system for loading and unloading ships.** 2013. Graduate Work (Licentiate degree in Mathematics) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2013.

ABSTRACT

One of the reasons that influences the efficience of a port terminal that is especialized in movement of containers is in the process of their loading and unloading. The fundamental idea is to determine a plan, to each port in order to decrease the number of total movement of loading and unloading that are necessary during the ship route to every port so that they save time and consequently reduce costs. The goal of this research was to present improvement in the program, written in language C, used to solve to problem of loading and unloading of ships, a NP-complete problem, through the meta-heuristics Simulated Annealing. With the improvement made, an average of 50% reduction of necessary time was gotten to solve the problem in 45 analised instances and in a specific group of these instances the average reduction reached 77 and in 10 instances there was even an improvement in the solution.

KEYWORDS: Combinatorial Optimization. Metaheuristics. Simulated Annealing.Container Ship Stowage Problem.CSSP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

P	– Tempo Polinomial Determinístico
NP	– Tempo Polinomial não Determinístico
PCCTP	– Problema de Carregamento de Contêineres em Terminais Portuários

SUMÁRIO

1 INTRODUÇÃO

O problema de carregamento e descarregamento de navios é composto por um conjunto de subproblemas que passam desde a distribuição de contêineres no pátio até o modo como os mesmos serão alocados no navio.

Dado o tamanho dos navios hoje em dia, o número de movimentos necessários para se alocar os contêineres pode se chegar a ordem de milhares e um plano de alocação se faz necessário a fim de diminuir o número de movimentos e o tempo que o navio permanece em um porto. Para isso, diversos estudos são realizados a fim de determinar um plano que se mostre ótimo para realizar a alocação, porém conforme mostrado em Avriel, Penn e Shpirer (2000), por se tratar de um problema NP-Completo, uma solução exata em um tempo viável ainda não existe.

Dessa forma, encontramos na literatura soluções aproximadas para o problema utilizando diferentes meta-heurísticas e, por se tratar de um problema que envolve muitas variáveis, não só a solução se mostra importante, como também a abordagem que se dá ao problema a fim de simplificar a linguagem utilizada.

Uma abordagem inovadora é encontrada em Azevedo et al. (2011) na qual a solução do problema é tratada como sendo um conjunto de regras de carregamentos e descarregamentos aplicadas em cada porto. Pode se entender por regra, um conjunto de instruções diretas que ditam como os contêineres serão carregados ou descarregados. Neste caso, uma solução ótima seria encontrar, para cada porto, a melhor regra de carregamento e descarregamento a fim de minimizar o número desses movimentos em todo o percurso do navio.

Naquele trabalho, uma solução é apresentada utilizando a meta-heurística *Simulated Annealing*[1], modelada por um programa escrito em linguagem C. Um dos objetivos do presente trabalho, descrito mais detalhadamente a seguir, é melhorar o algoritmo daquele programa a fim de diminuir o tempo computacional necessário para se obter as soluções.

1.1 OBJETIVOS

Este trabalho tem por objetivo geral apresentar uma extensão a um programa criado em linguagem C para resolver um problema de Otimização Combinatória, a saber, o problema de

[1] Recozimento simulado em Português. Decidiu-se por manter a designação original no texto, por ser amplamente utilizada nos meios acadêmicos, mesmo em publicações em português.

carregamento de contêineres em terminais portuários, ou simplesmente PCCTP, o qual será apresentado no próximo tópico. Este programa utilizou a abordagem por Simulated Annealing mencionado anteriormente e foi utilizado em Azevedo et al. (2011). Especificamente as modificações propostas são:

– Distribuir em arquivos distintos partes principais do código, como funções e estruturas destinadas às regras de carregamento, às regras de descarregamento e à meta-heurística, a fim de simplificar futuras modificações.

– O encapsulamento de execução das instâncias em *threads*, com o objetivo de paralelizar as diferentes chamadas para uma mesma instância, obtendo um ganho de desempenho nos computadores com múltiplos núcleos.

– A introdução de uma nova regra de carregamento, baseada na otimização do descarregamento do porto seguinte.

– Introdução de uma memória na função objetivo, a fim de agilizar o cálculo da mesma quando se modifica, através da escolha de algum vizinho, as regras de carregamento e descarregamento de um dado porto.

1.2 ORGANIZAÇÃO DO TRABALHO

Esse trabalho foi organizado de modo que o capítulo 2 apresenta uma introdução histórica e conceitual dos assuntos utilizados, tais como, pesquisa operacional, complexidade computacional, otimização e meta-heurística.

O capítulo 3 apresenta o problema de carregamento e descarregamento de navios em si, objeto de estudo deste trabalho, bem como o modelo matemático do problema e a abordagem por meio de regras de Azevedo et al. (2011). Neste capítulo, também há uma explicação detalhada por meio de exemplos de cada uma das regras utilizadas.

O capítulo 4 apresenta as modificações efetuadas no programa original e os resultados obtidos com o novo programa e também aqueles obtidos com o programa original a título de comparação.

O capítulo 5 conclui o trabalho com as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PESQUISA OPERACIONAL E OTIMIZAÇÃO

A Pesquisa Operacional teve início por volta da década de quarenta (Johnson; Mowry, 2005) com o advento da aplicação de métodos científicos por militares britânicos e americanos para criar estratégias de defesa terrestres e aéreas, minimização de custos e até mesmo maximizar as perdas inimigas,utilizando de forma eficaz recursos limitados, tais como tropas, munição, remédios, etc.

As técnicas desenvolvidas se mantiveram em segredo até 1947 (Mukhopadhyay, 2011), já no pós Guerra, e despertaram o interesse de muitas indústrias em utilizá-las no gerenciamento dos processos de produção diários.

Os fundadores dessas técnicas são o estadunidense George Dantzig, com a introdução do Método Simplex, o húngaro John von Neumann, com a Teoria da Dualidade e o soviético Leonid Kantorovichque desenvolveu trabalhos voltados à Economia e que serviram de base para o Método Simplex de Dantzig.

Olhando por um ângulo matemático, as técnicas desenvolvidas têm por objetivo encontrar pontos de máximos ou mínimos para uma função, dentro de algum domínio e se remetem a um ramo da Matemática chamada Otimização ou Programação Matemática. A natureza desse domínio dividiu a Otimização em três grandes áreas, em que cada uma se restringe a estudar umaclasse de problemas:aqueles cujas variáveis são contínuas; aqueles cujas variáveis são inteiras e, por fim, aqueles que algumas variáveis são inteiras e outras contínuas.

Os problemas de Otimização cujas variáveis são inteiras, uma vez comprovada a finitude deseu domínio, possuem uma maneira trivial de encontrar a solução ótima: testando caso a caso. Porém esta estratégia pode se mostrar inviável nos casos em que o número de possíveis soluções for elevado. Para se ter uma ideia, Dantzig propôs um problema no qual buscava dispor de maneira ótima 70 pessoas em 70 postos de trabalhos (Basu, 2009), o que gera um total de 70! soluções para serem analisadas. Em uma de suas publicações escreveu o seguinte comentário:

> Agora 70! é um número grande, maior do que 10^{100}. Suponha que tivéssemos um IBM 370-168 disponível desde a época do big bangà 15 bilhões de anos atrás. Seria ele capaz de olhar todas as 70! combinações até o ano de 1981? Não! E se ele pudesse examinar 1 bilhão de soluções por segundo? A resposta ainda é não. Mesmo

> que a Terra fosse toda preenchida com esses computadores, todos trabalhando em paralelo, a resposta ainda seria não (Dantzig, 1984, p. 106, tradução nossa).

Em contrapartida, o problema foi resolvido em pouco tempo utilizando o algoritmo Simplex proposto por ele, visto que o algoritmo reduzia drasticamente o número de soluções a serem testadas.

Por esse fato, os matemáticos do século passado perdiam o interesse nesse tipo de problema, visto que buscar um algoritmo que reduzisse o número de passos necessários para resolver o problema de 10^{198} para 10^4eram igualmente impraticáveis de se proceder manualmente (Iusem, 1987).

Posto isto, podemos perceber que a Programação Matemática evoluiu em conjunto com os computadores, visto que neste novo cenário, executar um algoritmo com passos da ordem de 10^4 é um problema que leva menos de um segundo.

2.2 COMPLEXIDADE DE ALGORITMOS[2]

Com a evolução dos computadores, a busca de algoritmos eficientes para se resolver problemas de Otimização Combinatória cujo domínio era finito, tornou-se novamente plausível.

A partir desse novo motivador, outras questões se tornaram relevantes nessa busca para um dado problema, como por exemplo, a existência ou não do próprio algoritmo eficiente; o que fez surgir uma nova área conhecida como Complexidade Computacional.

Assim como em muitas outras situações na História da Matemática, essa nova linha de pesquisa teve como marco inicial um problema proposto pelo estadunidense Stephen Cook (1971) e pelo russo Leonid Levin, trabalhando separadamente em países opostos pela Guerra Fria no início da década de setenta, conhecido como P vs NP. Curiosamente o problema já havia sido levantado por Kurt Gödel em 1956, em uma carta endereçada à John von Neumann que ficou perdida até a década de oitenta (Fortnow, 2013).

P e NP são maneiras de se classificar um problema, de tal forma que, se um problema possui um algoritmo eficiente para resolvê-lo, então ele pertence a P, caso a sua solução possa ser verificada de forma eficiente, então ele está em NP. Mais precisamente:

---

[2] Este é um assunto complexo cujo aprofundamento foge do escopo deste trabalho. Uma discussão mais detalhada é encontrada em Iusem (1987).

"Um problema Q pertence à classe P se existe um algoritmo polinomial A para resolvê-lo, ou seja, tal que $g_A(n)$ está limitado por um polinômio em n." (Iusem, 1987, p. 47)
"A classe NP como formada por aqueles problemas para os quais, dada uma solução, existe um algoritmo polinomial B que confirma que o objeto dado é uma solução." (Iusem, 1987, p. 49)

Neste contexto, $g_A(n)$ é o número máximo de operações que o algoritmo A leva para resolver uma instância do problema cujo tamanho é no máximo n, sendo n o número de variáveis do problema.

A essência do problema P vs NP é justamente determinar se vale ou não aconjectura de que:

$$P = NP \tag{1}$$

Se a equação (1) for verdadeira, então todo problema cuja solução pode ser verificada através de um algoritmo eficiente, também possuirá um algoritmo eficiente para determiná-la. Segundo Fortnow (2013) a veracidade desta conjectura nos levaria a descobrir rapidamente desde curas para doenças terminais, até a natureza do próprio universo.

Em princípio uma demonstração para a equação (1) deveria descobrir um algoritmo eficiente, ou demonstrar a existência deles, para encontrar a solução de cada um dos problemas pertencentes à classe NP, posto que já se sabe que NP é pelo menos P. Cook mostrou que existe ainda outra classe de problemas em NP chamada de NP-Completo, de forma que qualquer problema em NP pode ser reduzido em tempo polinomial a um problema NP-Completo. Assim, se algum problema NP-Completo também pertencer à classe P, a conjectura será verdadeira.

Dado que ainda não se sabe se algum dia a conjectura se mostrará verdadeira ou no caso em que se mostre falsa, Matemática e Computação se uniram para desenvolver técnicas para a obtenção de soluções aproximadas denominadas heurísticas, que fornecem soluções próximas das ótimas em tempo viável para diversos problemas de Otimização.

2.3 META-HEURÍSTICAS E OTIMIZAÇÃO COMBINATÓRIA

Antes de caracterizarmos uma meta-heurística, precisaremos de uma definição mais precisa sobre problemas de Otimização Combinatória. De acordo com Blum e Roli (2003), um problema $P = (S, f)$ desse tipo, pode ser definido como:

- Um conjunto de variáveis $X = \{x_1; \dots; x_n\}$;

- Os domínios dessas variáveis $D_1, \dots, D_n$;
- Restrições às quais as variáveis estão sujeitas;
- Uma função objetivo$f$ para ser minimizada, no qual $f: D_1 \times \dots \times D_n \longrightarrow \mathbb{R}^+$;
- Um conjunto de todas as soluções possíveis:

$$S = \{s = \{(x_1; v_1); \dots; (x_n; v_n)\} | v_i \in D_i, s\ satisfazendo\ as\ restrições\ impostas\}.$$

Geralmente S é conhecido como espaço de busca e, cada um de seus elementos pode ser considerado como um candidato à solução.

Resolver um problema de Otimização Combinatória é encontrar um $s^* \in S$, conhecido como ótimo global de (S, f), tal que:

$$f(s^*) \leq f(s)\ \forall s \in S \tag{2}$$

Além do mais, também podemos definir a noção de vizinhança de s, como sendo uma função do tipo:

$$\mathcal{N}: S \longrightarrow 2^S \tag{3}$$

Responsável por levar cada elemento $s \in S$ a um conjunto de vizinhos $\mathcal{N}(s) \subseteq S$, chamado de vizinhança de s.

Como foi mostrado no final do tópico anterior, ao menos que (1) seja verdadeira, alguns problemas não possuirão algoritmos para se obter uma solução. De posse dessa hipótese, um novo tipo de algoritmo de aproximação surgiu com a ideia de explorar de maneira eficaz o espaço de busca utilizando estratégias de alto nível e diferentes tipos de métodos.A essa ideia deram o nome de meta-heurística.

A origem etimológica dessa palavra provém da composição de duas palavras Gregas, *Heuristic* derivada do verbo *heuriskein* e do sufixo *metá*, que significam respectivamente, “descobrir” e “além, de um nível superior”. O que faz referência ao conceito de meta-heurística por propiciar uma busca em um nível acima, adicionando elementos estratégicos para fugir de mínimos locais.

Utilizando a linguagem introduzida anteriormente, um mínimo local s' é um valor cuja imagem é a menor dentreàs de sua vizinhança, ou seja, que possui a seguinte propriedade:

$$\forall s \in \mathcal{N}(s'): f(s') \leq f(s) \tag{4}$$

2.3.1 Meta-heurística: Simulated Annealing

O Simulated Annealing, ou apenas SA, é uma das meta-heurísticas mais antigas e uma das primeiras a possuir uma estratégia explícita para fugir de mínimos locais (Blum; Roli, 2003).

Ela foi concebida por Kirkpatrick, Gellat e Vecchi (1983) e faz uma analogia ao resfriamento térmico de materiais expostos a altas temperaturas. O mérito do SA é permitir movimentos de uma solução corrente s para uma solução vizinha $s' \in \mathcal{N}(s)$, mesmo que s' gere um valor de pior qualidade. Essa escolha tem por objetivo fugir de mínimos locais utilizando como critério uma probabilidade que diminui ao longo da busca. Segundo Abensur (2011), o SA pretende alargar o horizonte de alternativas analisadas:

> A questão do horizonte de alternativas pode ser entendido através de um exemplo cotidiano. Imagine que uma pessoa queira comprar uma pizza e resolva fazer uma pesquisa a pé pelo seu bairro. [...] . Após algum tempo essa pessoa faria a melhor escolha sobre o número de alternativas encontradas. No entanto, poderia existir alguma pizzaria com uma melhor oferta do produto um pouco além do limite estabelecido ou talvez num dos trajetos evitados pelo consumidor (Abensur, 2011, p. 1564-1565).

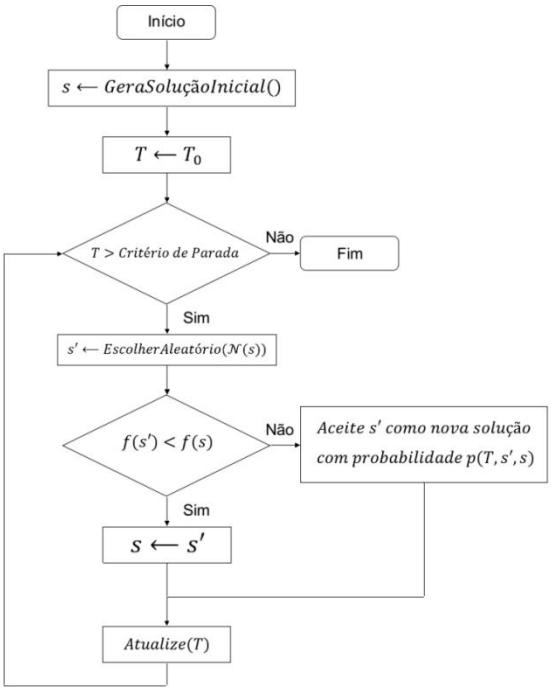
Conforme Figura 1, o algoritmo começa gerando uma solução inicial aleatória e inicializando o parâmetro T através de uma temperatura inicial T_0. Daí, em cada iteração escolhe-se aleatoriamente uma solução vizinha $s' \in \mathcal{N}(s)$ e testa-se a qualidade da mesma.

O primeiro teste é efetuado de maneira direta, se a solução vizinha é melhor doque a solução corrente, então atribui-se este valor ao valor corrente; porém, se a solução vizinha é de pior qualidade, ainda se efetua um segundo teste; aceitando a solução vizinha com uma probabilidade p, calculada de acordo com a distribuição de Boltzmann:

$$p(T, s', s) = e^{-\frac{f(s')-f(s)}{T}} \tag{1}$$

Em se tratando do algoritmo de buscas, isso significa que no início o algoritmo se preocupa mais em explorar o espaço de busca do que em melhorar a solução; permitindo saltos de uma solução corrente para uma solução de pior qualidade com mais facilidade. Porém, à medida que a temperatura diminui, esses saltos ocorrem com uma frequência menor, levando a busca a convergir para um mínimo local.

Figura 1 – Fluxograma do Simulated Annealing



Como pode ser visto pela equação (5) a probabilidade de se aceitar tais saltos é controlada por dois fatores: a temperatura e a diferença entre a solução corrente e a solução vizinha. No início, como a temperatura é alta, grandes diferenças no valor de $f(s)$ e $f(s')$ não influenciam muito no valor de p, o que permite o algoritmo explorar o espaço de busca e, conforme a temperatura vai diminuindo, esses saltos passam a ser cada vez menores, dado que $f(s') - f(s)$ passa a ter um valor significativo, o que leva a busca convergir para um mínimo local.

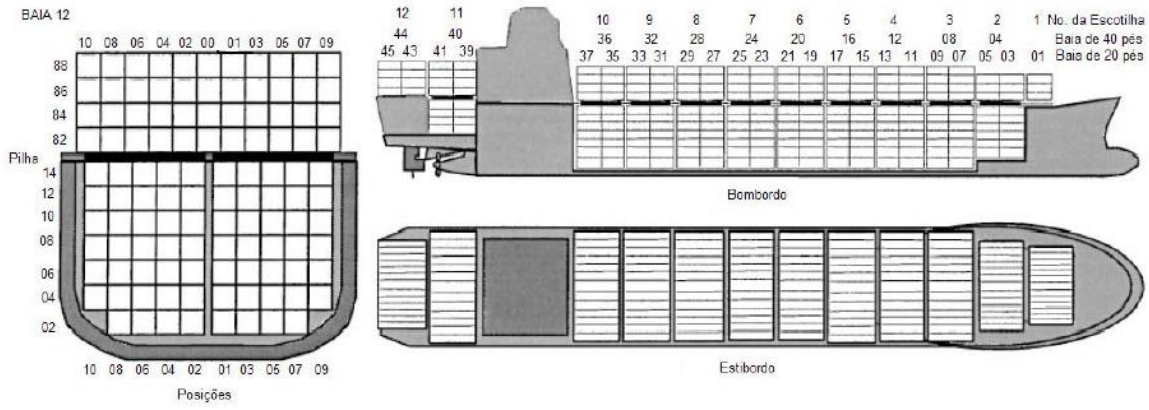
3O PROBLEMA DE CARREGAMENTO DE NAVIOS

3.1 INTRODUÇÃO

Navios porta contêineres possuem sua capacidade medida em TEU, sigla em inglês cuja tradução significa Unidade Equivalente de Vinte pés. Por exemplo, um navio cuja capacidade é de seis mil TEUs pode carregar até seis mil contêineres de vinte pés de comprimento, por oito de largura e oito de altura. Em muitos navios, estes contêineres são alocados em células formando pilhas verticais e, um conjunto dessas pilhas que se estende por toda a largura do navio, recebe o nome de baia.

A figura 2 apresenta um corte frontal do navio no qual é possível visualizar a baia de número 12, junto com suas células dispostas em linhas horizontais e colunas verticais.

Figura 2 – Estrutura celular de um navio



Fonte: (WILSON, I.; ROACH, P, 2000)

Por esse estilo de configuração, descarregar um contêiner que não seja o primeiro em uma pilha, acarreta em descarregar também todos os contêineres acima dele, visto que os contêineres são retirados pelo topo.

O PCCTP visa determinar, de maneira sistemática e respeitando as restrições operacionais dos contêineres, como efetuar o carregamento de um conjunto de contêineres no navio, a fim de minimizar o tempo não só do carregamento em si, mas também dos futuros descarregamentos nos portos subsequentes.

Segundo Avrieletal. (2000), o PCCTP se trata de um problema NP-Completo e, de posse desse fato, soluções viáveispodem ser encontradas utilizando meta-heurísticas.

3.2 MODELO MATEMÁTICO

A formulação matemática apresentada a seguir está de acordo com as restrições operacionais relacionadas aos contêineres e ao navio e pode ser vista mais detalhadamente em Avriel et al. (1998).

Suponha, sem perda de generalidade, um navio inicialmente vazio com uma única baia retangular contendo R linhas e C colunas, numeradas de baixo para cima e da esquerda para a direita, de forma que a linha R corresponda ao topo da baia. Portanto,considerando que esta baia pode ser vista como uma matriz, este navio tem capacidade máxima de $R \times C$ contêineres, os quais serão considerados todos de mesma dimensão.

Suponha ainda que este navio visite N portos dispostos sequêncialmente, de modo que, em cada porto $i = 1, 2, 3, \dots, N$, ele pode receber um carregamento ou descarregar contêineres, com exceção do primeiro porto no qual não há descarregamentos, visto que inicialmente o navio está vazio, e do último porto onde o navio deve descarregar todos os contêineres ainda presentes e não receber nenhum.

Essas operações de carregamento e descarregamento, seguindo as condições impostas, serão dadas de acordo com uma matriz de transporte $T = [T_{ij}]$, quadrada de ordem$(N - 1)$, conhecida antes de se iniciar a rota. Nesta matriz cada elemento T_{ij} indica o número de contêineres com origem no porto i e destino ao porto j. Vale ressaltar que esta matriz é triangular superior, visto que o navio visitará seqüencialmente os portos, e o índice da origem obrigatoriamente estará antes do índice do destino, ou seja, $T_{ij} = 0$, se $i \geq j$.

Agora seja a variável binária $x_{ijv}(r, c)$ que assume valor um se existe um contêiner no compartimento (r, c), carregado no navio no porto i, cujo destino final é o porto j e que foi movido no porto v; caso contrário assume valor zero. Considere que o índice v pode assumir os valores:

$$v = i + 1, i + 2, \dots, j - 1, j. \tag{2}$$

Se $v = j$, então o contêiner é descarregado no seu destino final, enquanto que se $v < j$, o contêiner é movido, ou seja, descarregado no porto v e recarregado novamente neste mesmo

porto, passando a ser denotado por $x_{vjv'}(r', c')$, no qual (r', c') pode ser um compartimento diferente.

De modo análogo, também vamos definir a variável binária $y_i(r, c)$ que assume valor 1 se, ao deixar o porto i, o compartimento do navio (r, c) está ocupado por um contêiner e assume valor 0 caso contrário.

Sem perder a generalidade, vamos assumir que o custo de se mover algum contêiner em qualquer um dos portos é sempre o mesmo. Neste caso, o nosso objetivo é minimizar o custo total dos movimentos, mais precisamente:

Minimizar o custo total:

$$\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sum_{v=i+1}^{j-1}\sum_{r=1}^{R}\sum_{c=1}^{C} x_{ijv}(r, c) \tag{3}$$

Sujeito a:

$$\sum_{v=i+1}^{j}\sum_{r=1}^{R}\sum_{c=1}^{C} x_{ijv} - \sum_{k=1}^{i-1}\sum_{r=1}^{R}\sum_{c=1}^{C} x_{kji} = T_{ij} \qquad i = 1, \dots, N-1, j = i+1, \dots, N; \tag{4}$$

$$\sum_{k=1}^{i}\sum_{j=i+1}^{N}\sum_{v=i+1}^{j} x_{kjv}(r, c) = y_i(r, c) \qquad \begin{matrix} i = 1, \dots, N-1, r = 1, \dots, R, \\ c = 1, \dots, C; \end{matrix} \tag{5}$$

$$y_i(r, c) - y_i(r+1, c) \geq 0 \qquad \begin{matrix} i = 1, \dots, N-1, r = 1, \dots, R-1, \\ c = 1, \dots, C; \end{matrix} \tag{6}$$

$$\sum_{i=1}^{j-1}\sum_{p=j}^{N} x_{ipj}(r, c) + \sum_{i=1}^{j-1}\sum_{p=j+1}^{N}\sum_{v=j+1}^{p} x_{ipv}(r+1, c) \leq 1 \qquad \begin{matrix} j = 2, \dots, N, r = 1, \dots, R-1, \\ c = 1, \dots, C; \end{matrix} \tag{7}$$

$$x_{ijv}(r, c) = 0 \; ou \; 1; \; y_i(r, c) = 0 \; ou \; 1. \tag{8}$$

A equação (7) representa a função objetivocusto, a qual retorna o custo total de todos os movimentos em todos os portos. Aqui é assumido que o custo por movimento é igual a 1. Como o custo no porto final de um dado contêiner é obrigatório, então este não é incluso no cálculo da função objetivo, visto que a soma corre sobre $v = i + 1, \dots, j - 1$.

A restrição (8) indica o número de contêineres que serão movidos originalmente do porto i até o porto j. A primeira parcela da subtração guarda a quantidade de contêineres que serão descarregados no porto j e que possuem origem inicial ou não no porto i; dado este fato há a necessidade de se descontar a quantidade de contêineres com origem inicial antes do

porto i e destino em j, no qual passaram a ter origem no porto i por terem sido movidos neste porto.

A restrição (9) garante que ao sair de um dado porto i cada compartimento (r, c) do navio terá no máximo um único contêiner. Isso é feito garantindo que todos os descarregamentos ou movimentos que serão feitos daquele compartimento nos portos subseqüentes, cuja origem foi no porto i ou antes, seja no máximo um.

A restrição (10) impõe que ao sair de um dado porto i, se existir um contêiner em uma coluna qualquer c na posição $r + 1$, então também haverá um contêiner abaixo dele na posição r.

A restrição (11) impõe que se um dado contêiner no compartimento (r, c) for descarregado em j, seja por ser o seu destino final ou apenas um ato de movimento para realocação, não pode existir no navio um contêiner com posição $(r + 1, c)$ que será descarregado somente nos portos subseqüentes, visto que este obrigatoriamente precisará ser descarregado no porto j para que o primeiro também seja.

Ainda de acordo com Avriel, Penn e Wittenboon (1998) se fosse permitido um relaxamento na restrição 12, de modo que um contêiner pudesse ocupar um espaço menor ou maior do que um único compartimento, ou seja:

$$0 \leq x_{ijv}(r, c) \leq 1;\ 0 \leq y_i(r, c) \leq 1, \tag{9}$$

então o valor ótimo neste caso, seria sempre zero,pois como em cada porto, após efetuar o descarregamento dos contêineres, o volume do espaço não ocupado é pelo menos o volume total dos contêineres a serem carregados, bastaria fatiar, em cada porto, os contêineres em R fatias de mesmo tamanho e alocá-los em uma coluna vertical. Isso sempre é possível visto que de acordo com a restrição (13) essa coluna vertical poderia ocupar uma largura além de um único compartimento e sempre haveria espaço suficiente para alocá-los. Sendo assim, essa coluna seria mexida somente nos destinos finais de seus contêineres, ou seja, o número de movimentos extras durante o trajeto seria sempre zero.

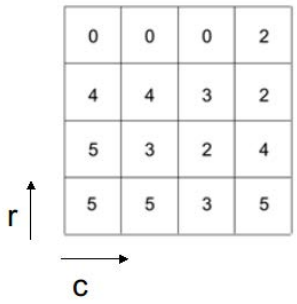
Infelizmente para problemas reais, a formulação dada acima pelas equações de (8) a (12) é inviável dado a grandeza desses problemas e só poderia ser utilizada para encontrar a solução ótima de problemas pequenos.

3.3 REPRESENTAÇÃO MATRICIAL

Conforme a estrutura celular de um navio mostrada na figura 2 e ainda supondo um navio com uma única baia retangular de dimensões $R \times C$, vamos definir uma matrizdeocupação B, que representa o estado de cada um dos compartimentos(r, c) do navio. Nesta matriz$B_{r,c} = j$ representa um contêiner presente no compartimento (r, c) , cujo destino será o porto j. No caso em que $B_{r,c}$ valha zero, isso apenas significará que este compartimento se encontra vazio.

Por exemplo, na figura 3, $B_{2,3}$ é igual a dois, o que significa que no compartimento $(2,3)$ do navio existe um contêiner com destino ao porto dois, enquanto que $B_{4,1} = 0$ significa que o compartimento $(4,1)$ do navio está vazio. Vale ressaltar, como dito anteriormente, que a numeração dos compartimentos é efetuada de baixo para cima e da esquerda para a direita.

Figura 3 – Matriz de Ocupação



Ao longo do trajeto pelos N portos essa matriz sofre modificações devido a entrada e a saída de contêineres. Por exemplo, ao chegar a um determinado porto j os contêineres cujo destino é este porto serão descarregados e os contêineres cujos destino são os portos $j + 1, j + 2, \ldots, N$ serão carregados.

De posse desse fato, existem basicamente duas ações que modificam a matriz de ocupação B em cada um dos portos, o carregamento e o descarregamento. A maneira como eles são efetuados influenciam na quantidade de movimentos que um contêiner pode sofrer ao longo de sua estadia no navio, impactando diretamente no tempo em que o navio demoraria em cada porto e no custo total da operação.

Sendo assim, Azevedo et al. (2011) apresentam uma ideia para abordar o problema do PCCTP, baseada na criação de regras que possuem como objetivo determinar a maneira com a qual estes carregamentos e descarregamentos são efetuados em cada um dos portos.Por

exemplo, em um determinado porto pode ser mais vantajoso efetuar o descarregamento dos contêineres por coluna, enquanto que em algum outro pode ser mais vantajoso efetuar tal descarregamento por linha. De modo análogo também foram criadas regras que determinam como os carregamentos dos contêineres são efetuados. Daí o objetivo é, a partir de uma lista pré-determinada dessas regras, determinar qual par de regras de carregamento e descarregamento se deve utilizar em cada porto, a fim de diminuir o custo total dos movimentos em todo o trajeto.

Para isso, foi apresentado um conjunto de 12 pares de regras, chamadas de Regras_k, resultado da combinação de quatro regras de carregamento, a saber, Rc1, Rc2, Rc3 e Rc4, e três regras de descarregamentos, a saber, Rd1, Rd2 e Rd3, às quais são apresentadas a seguir:

Rc1: Efetua o carregamento dos contêineres por linha da esquerda para a direita, de tal maneira que os contêineres destinados aos portos mais longínquos sejam carregados primeiro.

Rc2: Carrega os contêineres por coluna da esquerda para a direita até uma linha l, calculada da seguinte forma:

$$l = \left\lceil \frac{\sum_{i=1}^{p} \sum_{j=p+1}^{N} T_{ij}}{C} \right\rceil \tag{10}$$

O objetivo disso é deixar as colunas mais homogêneas, procurando diminuir o número de remanejamentos (Bischoff; RATCLIFF, 1995).

Rc3: Esta regra é similar à regra Rc1, porém difere quanto ao sentido do carregamento. De modo que ela efetua o carregamento por linha da direita para a esquerda, começando pelos contêineres cujo destino é mais longínquo.

Rc4: Esta regra é semelhante a regra de carregamento Rc2, diferindo somente pelo sentido do carregamento. De modo que o carregamento é efetuado por coluna da direita para a esquerda até a linha l, calculada também de acordo com a equação (14).

Rd1: Esta regra descarrega todos os contêineres com destino ao porto atual e aqueles que estão acima deles.

Rd2: Esta regra descarrega todos os contêineres das pilhas que possuem pelo menos um contêiner para o porto atual, visando o remanejamento deles através da regra de carregamento adotada para o porto.

Rd3: Esta regra remove todos os contêineres do navio, visando a reorganização total do mesmo no próprio porto atual, segundo a regra de carregamento adotada para este porto.

A tabela a seguir mostra os 12 pares de regras:

Tabela 1 – Regras_k

Regra_k	Regra de Carregamento	Regra de Descarregamento
1	Rc1	Rd1
2	Rc1	Rd2
3	Rc1	Rd3
4	Rc2	Rd1
5	Rc2	Rd2
6	Rc2	Rd3
7	Rc3	Rd1
8	Rc3	Rd2
9	Rc3	Rd3
10	Rc4	Rd1
11	Rc4	Rd2
12	Rc4	Rd3

A figura 4 mostra uma matriz de ocupação T, na qual diferente da matriz de ocupação B, é numerada de cima para baixo e da esquerda para a direita, de modo que o elemento $T_{24} = 1$ significa que existe um único contêiner com origem no porto dois e destino no porto quatro, enquanto que $T_{35} = 5$ significa que existem cinco contêineres com origem no porto três e destino no porto cinco.

Figura 4 – Matriz de transporte T

	D2	D3	D4	D5
O1	4	1	2	3
O2	0	2	1	2
O3	0	0	1	5
O4	0	0	0	2

A fim de ilustrar a utilização dessas regras, vamos supor uma matriz de ocupação B de dimensões 4×4 com capacidade para transportar até 16 contêineres,uma matriz de transporte T conforme a figura 4 e um trajeto contendo cinco portos.

Observe que no primeiro porto a regra de descarregamento é inexistente, assim como a regra de carregamento no último porto. Também podemos afirmar que a regra de descarregamento utilizada no último porto é irrelevante, visto que todos os contêineres serão retirados do navio.

Portanto vamos supor que no primeiro porto seja aplicada a regra um presente na tabela 1. A figura 5a mostra a matriz de ocupação, inicialmente vazia, antes de aplicar a regra de carregamento, enquanto que a figura 5b já mostra como ficou a matriz de ocupação após a regra de carregamento Rc1:

Figura 5 – Matriz *B*no porto 1: (a) antes de aplicar Rc1; (b) após a aplicar Rc1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(a)

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

(b)

No porto dois vamos aplicar a regra quatro da tabela 1:

Figura 6 – Matriz *B* no porto 2: (a) antes de aplicar Rd1; (b) após a aplicar Rd1

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

(a)

0	0	0	0
0	0	0	0
4	3	0	0
5	5	5	4

(b)

Os contêineres, se houver, descarregados neste porto com destino aos portos subsequentes, permanecem no pátio para que sejam carregados junto com os contêineres com origem neste porto.

Figura 7 – Matriz *B* no porto 2: (a) antes de aplicar Rc2; (b) após a aplicar Rc2

0	0	0	0
0	0	0	0
4	3	0	0
5	5	5	4

(a)

0	0	0	0
5	5	3	0
4	3	4	3
5	5	5	4

(b)

No porto três vamos aplicar a regra oito da tabela 1:

Figura 8 – Matriz *B* no porto 3: (a) antes de aplicar Rd2; (b) após a aplicar Rd2

0	0	0	0
5	5	3	0
4	3	4	3
5	5	5	4

(a)

0	0	0	0
5	0	0	0
4	0	0	0
5	0	0	0

(b)

Note que no exemplo da figura 8, três contêineres com destino ao porto cinco e dois contêineres com destino ao porto quatro foram descarregados e serão novamente carregados junto com os contêineres do porto atual para os subsequentes.

Figura 9 – Matriz *B* no porto 3: (a) antes de aplicar Rc3; (b) após a aplicar Rc3

0	0	0	0
5	0	0	0
4	0	0	0
5	0	0	0

(a)

5	0	0	0
5	0	0	0
4	4	4	4
5	5	5	5

(b)

No porto quatro vamos aplicar a regra 12 da tabela 1:

Figura 10 – Matriz *B* no porto 4: (a) antes de aplicar Rd3; (b) após a aplicar Rd3

5	0	0	0
5	0	0	0
4	4	4	4
5	5	5	5

(a)

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(b)

Neste exemplo da figura 10, ficaram no pátio junto com os demais contêineres a serem carregados neste porto, seis contêineres com destino ao porto cinco.

Figura 11 – Matriz *B* no porto 4: (a) antes de aplicar Rc4; (b) após a aplicar Rc4

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(a)

0	0	0	0
0	0	0	0
5	5	5	5
5	5	5	5

(b)

Como dito acima, a regra de descarregamento utilizada no porto cinco é irrelevante, visto que qualquer regra produzirá o mesmo número de movimentos.

Segundo Azevedo et al. (2011) as vantagens de se utilizar esta abordagem por regras permite que:

- O conhecimento prévio do planejador seja incorporado de maneira simples.
- As matrizes de ocupação produzidas sejam factíveis, acarretando também em soluções factíveis.
- As soluções obtidas são vetores de dimensões $N - 1$, desde que N seja o número de portos. O que proporciona uma representação mais compacta se comparada a outras abordagens como aquela utilizada em Avriel, Penn e Shpirer (2000).

4 MODIFICAÇÕES REALIZADAS NO PROGRAMA

4.1 MODIFICAÇÕES ESTRUTURAIS

As diversas funções que compõem o programa foram distribuídas em diferentes arquivos, separadas de acordo com o objetivo que cada uma desempenha, a fim de manter uma maior organização e clareza na leitura do código.

A seguir é apresentado uma breve descrição desses arquivos:

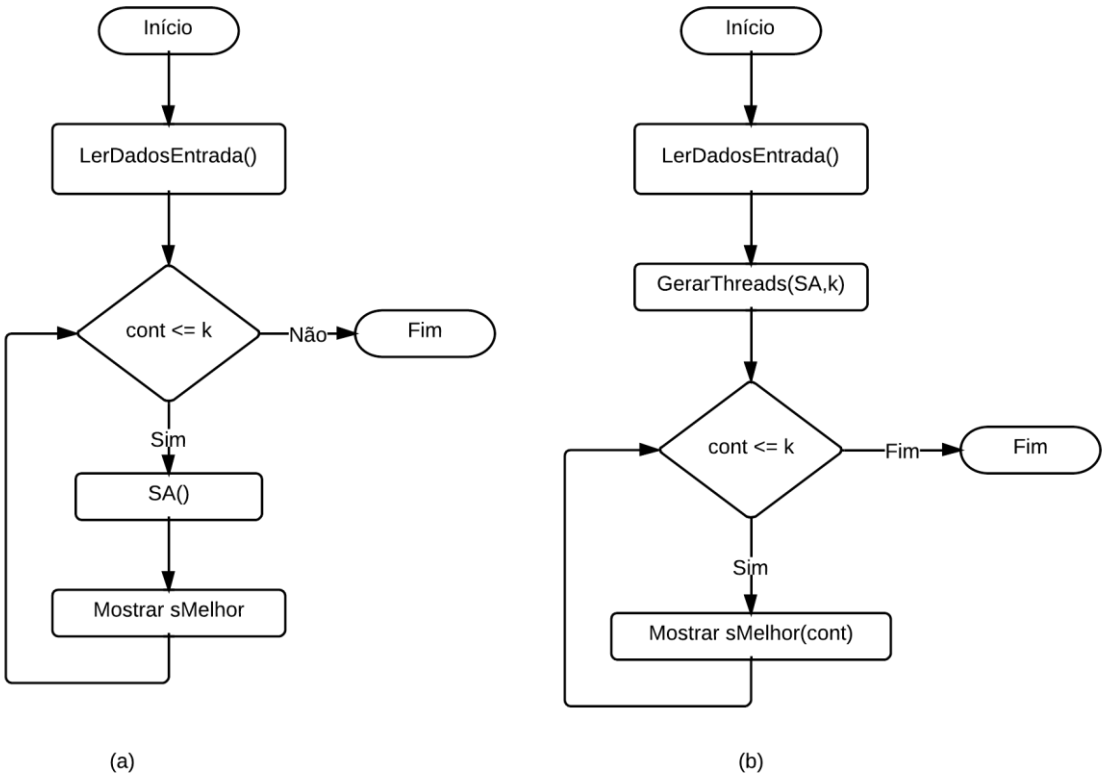
- Biblioteca.h: Possui todas as assinaturas dos métodos e funções, declarações de estruturas e constantes.
- RegrasCarregamento.c: Armazena todas as regras de carregamento presentes no programa. Estas regras são escolhidas através de um bloco *case*, cuja dimensão é atribuída por uma constante Nrc presente no arquivo biblioteca.h.
- RegrasDescarregamento.c: Semelhante ao arquivo anterior, porém contém todas as regras de descarregamento implementadas. Cada um dessas regras também é escolhida através de um bloco *case*, cuja dimensão é atribuída através da constante Nrd presente no arquivo biblioteca.h.
- Metaheuristicas.h: Contém a implementação da meta-heurística utilizada, neste caso, o SimulatedAnnealing.
- FuncObj.c: Contém a implementação da função objetivo.
- Funcoes.c: Neste arquivo estão presentes todas as implementações das demais funções do programa, tais quais, funções de leitura de dados, funções geradoras de números aleatórios, bem como outras funções auxiliadoras.
- Main.c: Possui o método principal do programa, responsável por chamar e fazer o encapsulamento em *thread* do Simulated Annealing.

4.2MODIFICAÇÕES LÓGICAS

O programa original calculava k vezes sequencialmente a mesma instancia. Portanto, a fim de otimizar o tempo, no programa modificado esses cálculos são feitos dentro de *threads.* A vantagem disso é que computadores com múltiplos núcleos efetuam o processamento de mais de uma *thread* por vez, sendo essa quantidade limitada pelo número de núcleos.

As *threads* são processadas a partir do um método GerarThreads, como mostrado na figura 12. A biblioteca utilizada por este método para o processamento das *threads* é a *libpthread* e informações mais detalhadas podem ser encontradas em Hurd (2013).

Figura 12 –*Thread* principal do programa: (a) Original; (b) Modificado



O fluxograma da meta-heurística para ambos os programas é o mesmo e pode ser visto na figura 13 logo a seguir. O algoritmo adotado difere um pouco daquele presente na figura 1, visto que neste caso há também um número de iterações para cada temperatura, chamado de SAMAX, cujo valor inicial é de mil e decai a medida que a mesma diminui. Também se adotou um esquema de resfriamento variado, como pode ser visto no fluxograma, de modo que para temperaturas acima de mil a taxa de resfriamento é de 98%, para temperaturas entre cem e mil a taxa é de 95% e para temperaturas menores a taxa decai para 92%. Desse modo, há uma exploração maior no espaço de busca para temperaturas mais altas. Em todos os casos, a temperatura inicial T_0 adotada foi de cem mil.

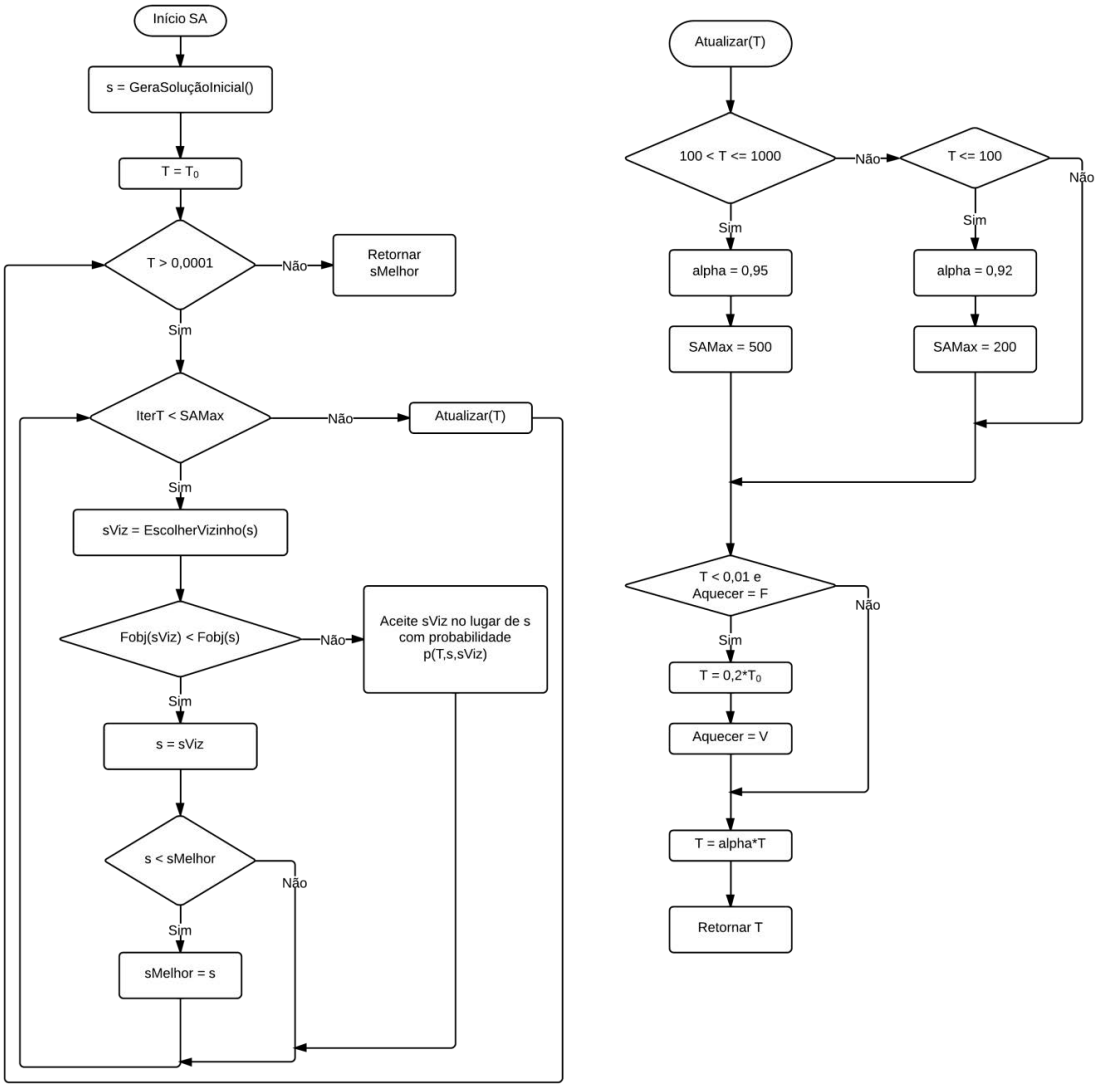
Neste modelo uma solução s é um vetor que associa a cada porto uma das regras presentes na Tabela 1. Uma solução vizinha de uma solução s é uma solução sViz que é

determinada escolhendo de maneira aleatória um único porto para receber, também de maneira aleatória, outra regra presente na Tabela 1. O sorteio da nova regra é efetuado até que se encontre uma regra diferente da já estipulada para o porto escolhido.

A função objetivo, representada por Fobj no fluxograma a seguir, recebe como parâmetro uma solução e retorna o número de movimentos total efetuados em todos os portos. Estes movimentos são calculados de acordo com a regra escolhida para o porto.

Ao final o algoritmo retorna a melhor solução encontrada, denominada sMelhor, que também é um vetor mostrando qual das regras da Tabela 1 deve ser aplicada em cada porto.

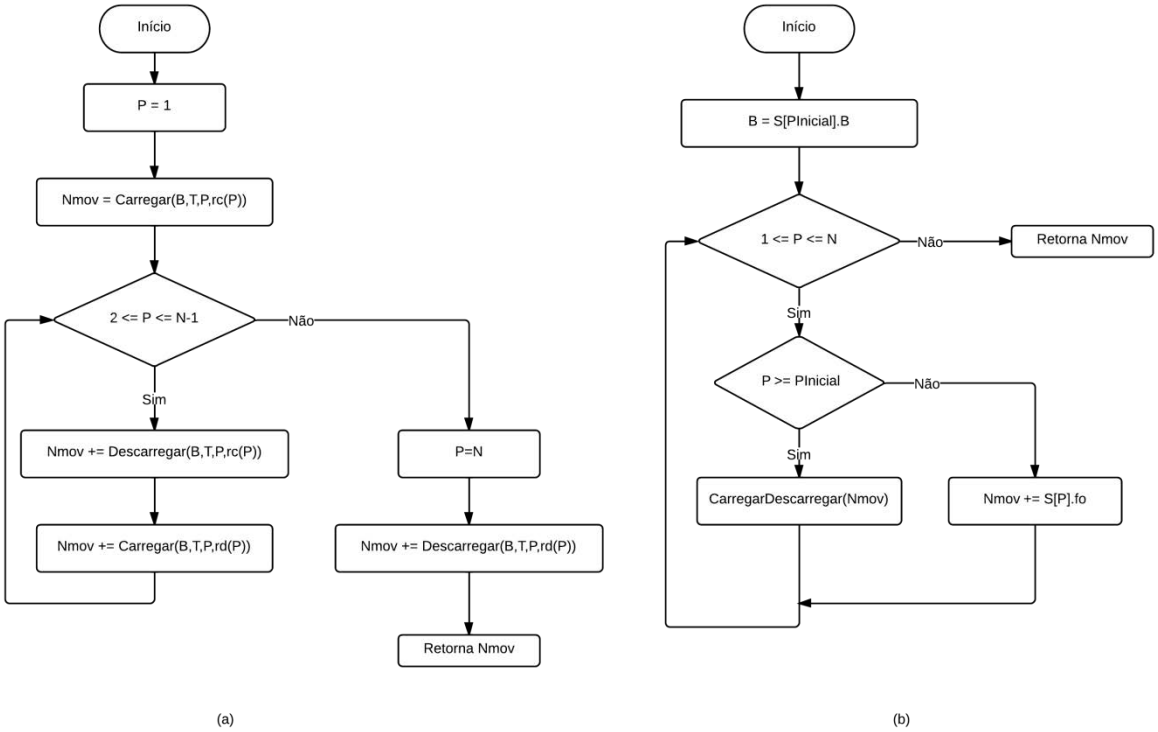
Figura 13 – Fluxograma da meta-heurística Simulated Annealing.



Uma das modificações que provocou mudanças consideráveis no código foi quanto ao cálculo da função objetivo, conforme mostrado na figura 14.

Na versão antiga do programa quando um porto p é escolhido para receber uma solução vizinha, a função objetivo computa novamente os números de movimentos realizados para os portos anteriores a p. Note que isso não é necessário, visto que os portos que serão influenciados por novas escolhas de regras em p são os portos subsequentes a ele.

Figura 14 – Função Objetivo do programa: (a) Original; (b) Modificado



Porém, para se calcular o número de movimentos em p é necessário conhecer o estado da matriz B assim que o navio chega neste porto. E, na versão antiga, só é possível obter este estado efetuando novamente os cálculos para os portos anteriores.

Portanto, a fim de aproveitar essa ideia, uma solução s deixou de ser um vetor que retorna para cada porto uma das regras da Tabela 1 e passou a ser um vetor que, para cada porto i, retorna uma estrutura que guarda a regra de carregamento, a regra de descarregamento, o número total de movimentos executados no porto e o estado da matriz de ocupação B.

Mais precisamente, uma solução S passou a ser um vetor de dimensão N, sendo N o número de portos, cujos elementos são estruturas do tipo:

Figura 15 – Elemento do vetor S.

```
typedef struct {
int rc;
int rd;
int fo;
int B[R][C];
}
```

Nesta estrutura,$S[i].B$ é a matriz de ocupação no momento em que o navio chegou ao porto i e $S[i].rc, S[i].rd, S[i].fo$ são respectivamente a regra de carregamento e descarregamento aplicadas, bem como o número de movimentos total realizados neste porto.

Conforme pode ser visto na figura 14 (b), ao se escolher aleatoriamente um porto PInicial para receber novas regras de carregamento e descarregamento, a matriz de ocupação B é restaurada para o estado em que se encontrava quando o navio chegou pela primeira vez neste porto. E o cálculo do número de movimentos passa a ser efetuado somente para os portos a partir deste, já que os números de movimentos dos portos anteriores se encontram gravados, de acordo com a nova estrutura, em S[i].fo.

Outras modificações menores também foram efetuadas a fim de otimizar o tempo dos cálculos das próprias regras em si. Como a matriz de ocupação $B[L][C]$ possui dimensões $(L+1) \times (C+1)$, a linha $L = 0$ não possui significado para o modelo proposto. Portanto, durante o processo de carregamento do navio, esta linha foi utilizada para armazenar o menor elemento presente em cada coluna, com exceção da posição $B[0][0]$ que retorna zero, caso não existam contêineres a serem descarregados no porto atual.

A primeira informação beneficia as regras de descarregamentos rd1 e rd2, visto que não há necessidade de se fazer uma varredura na coluna procurando pelos contêineres a serem descarregados, se o porto atual é menor do que o menor valor presente naquela coluna. E a última informação dispensa o cálculo das regras de descarregamentos, fazendo com que as mesmas retornem zero como número de movimentos.

Uma nova regra de descarregamento também foi implementada baseada na regra rd1. Esta nova regra chamada de rd4 visa, em cada porto, minimizar o número de movimentos do próximo porto. Para isso, em cada porto p é descarregado por coluna, tanto os contêineres do porto atual como também os do próximo, bem como os contêineres acima deles. O objetivo é reorganizar os contêineres do próximo porto já no porto atual. A Figura 16 ilustra esta regra:

Figura 16 – Matriz *B* no porto 3: (a) antes de aplicar Rd4; (b) após a aplicar Rd4

0	0	0	0	0
5	0	0	0	0
4	5	3	3	4
4	5	3	4	5
3	4	5	5	5

(a)

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	5
0	0	5	5	5

(b)

Conforme será mostrado no próximo tópico, com esta nova regra houve melhora em algumas soluções.

4.3 RESULTADOS

Os resultados de ambos os programas, mostrados a seguir, foram realizados utilizando um computador com processador Core i7-3612QM 2,10 Ghz com 8 GB de memória RAM, rodando o sistema operacional Ubuntu 13.10 com kernel Linux 3.11.0-13. Foram testadas 45 instâncias, as mesmas utilizadas em Azevedo et al. (2011), e que podem ser obtidas no endereço https://sites.google.com/site/projetonavio/home.

As instancias foram divididas em três tipos segundo Avriel, Penn, e Wittenboon (1998), de acordo com a matriz de transporte considerada: 1 – Mista, 2 – Longa distância e 3 – Curta distância. Essa classificação está relacionada com o tempo que os contêineres permanecem no navio antes de serem descarregados. Por exemplo, se os contêineres percorrem longas distâncias antes de serem descarregados, a matriz é do tipo 2, caso percorram curtas distâncias é do tipo 3 e, por fim, a do tipo 1 que combina as duas características anteriores.

A Tabela 2 mostra a solução obtida em número de movimentos Nmov e o tempo em segundos necessário para obtê-la T(s) para os dois programas executando uma única vez cada instancia, além do número I da instância, a capacidade do navio em termos de linhas e colunas RxC, o tipo da matriz de transporte M, o número de portos N, o limitante inferior L. Inf. e a redução percentual no tempo necessário para se obter a solução utilizando o programa modificado em comparação com o original.

Vale ressaltar que o limitante inferior é o dobro do número total de contêineres, ou seja, o caso ideal em que cada contêiner é carregado e descarregado uma única vez, porém este valor nem sempre pode ser obtido. Por exemplo, suponha três portos $a, b\ e\ c$, dispostos em ordem crescente, pertencentes a uma rota com mais de três portos. Se existe no navio um contêiner com destino ao porto b e, no porto a o navio é totalmente carregado com contêineres com destino ao porto c, então ou o contêiner com destino em b será remanejado em um porto anterior a b, ou alguns contêineres com destino em c serão remanejados em b, o que em todo caso já acarreta em mais de dois movimentos por contêiner.

Tabela 2 – Comparação dos resultados entre os dois programas.

					Programa Original		**Programa Modificado**		
I	RxC	M	N	L. Inf.	Nmov	T(s)	Nmov	T(s)	Redução(%)
1	6x50	1	10	1322	1356	15,816	1356	12,334	22,02%
2	6x50	2	10	750	1002	12,172	990	11,454	5,90%
3	6x50	3	10	2282	2282	15,506	2282	1,388	91,05%
4	6x50	1	15	1580	1702	26,992	1702	17,697	34,44%
5	6x50	2	15	778	974	20,490	974	15,614	23,80%
6	6x50	3	15	3024	3024	25,102	3024	2,206	91,21%
7	6x50	1	20	1990	2118	41,334	2118	22,874	44,66%
8	6x50	2	20	784	928	27,054	928	20,842	22,96%
9	6x50	3	20	4880	4880	49,248	4880	15,712	68,10%
10	6x50	1	25	1664	1782	45,907	1782	27,242	40,66%
11	6x50	2	25	944	1156	37,049	1152	24,829	32,98%
12	6x50	3	25	5492	5492	67,436	5492	19,819	70,61%
13	6x50	1	30	2262	2410	72,149	2410	33,211	53,97%
14	6x50	2	30	1030	1348	50,529	1348	31,193	38,27%
15	6x50	3	30	6380	6380	94,105	6380	22,317	76,29%
16	6x100	1	10	2878	2906	27,331	2906	19,577	28,37%
17	6x100	2	10	1436	1698	22,179	1698	18,174	18,06%
18	6x100	3	10	5354	5354	31,590	5354	4,501	85,75%
19	6x100	1	15	3532	3690	49,142	3680	29,217	40,55%
20	6x100	2	15	1656	1852	34,996	1852	27,732	20,76%

					Programa Original		Programa Modificado		
I	RxC	M	N	L. Inf.	Nmov	T(s)	Nmov	T(s)	Redução
21	6x100	3	15	6296	6300	52,871	6300	25,940	50,94%
22	6x100	1	20	4138	4278	74,707	4278	40,567	45,70%
23	6x100	2	20	1942	2476	53,003	2474	36,362	31,40%
24	6x100	3	20	8714	8714	90,225	8714	36,530	59,51%
25	6x100	1	25	4390	4624	105,480	4598	49,157	53,40%
26	6x100	2	25	2410	3010	82,419	3010	47,566	42,29%
27	6x100	3	25	12236	12236	141,491	12236	35,427	74,96%
28	6x100	1	30	4654	5066	138,916	5058	59,388	57,25%
29	6x100	2	30	2204	2916	91,106	2916	56,341	38,16%
30	6x100	3	30	14432	14432	193,708	14432	41,432	78,61%
31	6x150	1	10	3222	3472	36,208	3472	27,383	24,37%
32	6x150	2	10	2238	2268	30,530	2268	26,626	12,79%
33	6x150	3	10	5882	5882	40,533	5882	6,711	83,44%
34	6x150	1	15	4562	4754	64,316	4754	40,517	37,00%
35	6x150	2	15	2212	3100	47,722	3050	37,504	21,41%
36	6x150	3	15	7672	7672	67,510	7672	0,674	99,00%
37	6x150	1	20	5470	5652	101,050	5652	55,504	45,07%
38	6x150	2	20	3000	3932	77,359	3846	53,052	31,42%
39	6x150	3	20	13190	13190	131,169	13190	38,475	70,67%
40	6x150	1	25	5516	5902	139,605	5900	69,704	50,07%
41	6x150	2	25	2868	3044	99,584	3044	63,710	36,02%
42	6x150	3	25	16864	16864	197,905	16864	50,016	74,73%
43	6x150	1	30	7130	7402	191,242	7416	82,248	56,99%
44	6x150	2	30	2104	2278	90,029	2278	73,352	18,52%
45	6x150	3	30	21342	21342	286,427	21342	59,549	79,21%

Observando a tabela 2, pode-se notar que em alguns casos houve melhora na solução e em outros a solução encontrada em ambos os programas é de pior qualidade do que aquela em Azevedo et al. (2011). Isso se dá, visto que os dados desta tabela foram gerados rodando uma única vez cada instância, sendo assim, gerou-se novos resultados para estes dados, mostrados na coluna I da Tabela 3, porém rodando 32 vezes cada uma das instancias.:

Tabela 3 – Resultados obtidos rodando 32 vezes a mesma instância.

					Programa Original		**Programa Modificado**		
I	RxC	M	N	NMin	Nmov	T(s)	Nmov	T(s)	Redução
2	6x50	2	10	750	1002	391,161	**990**	89,258	77,18%
4	6x50	1	15	1580	1700	875,802	1700	136,488	84,42%
8	6x50	2	20	784	926	865,045	926	165,712	80,84%
11	6x50	2	25	944	1156	1201,95	**1152**	220,146	81,68%
23	6x100	2	20	1942	2474	1734,843	**2462**	321,273	81,48%
25	6x100	1	25	4390	4586	3504,681	**4580**	439,673	87,45%
28	6x100	1	30	4654	5038	4708,002	**5032**	529,820	88,75%
29	6x100	2	30	2204	2834	3003,185	**2826**	474,711	84,19%
35	6x150	2	15	2212	3100	1708,318	**3050**	336,121	80,32%
37	6x150	1	20	5470	5646	3609,539	5646	484,048	86,59%
38	6x150	2	20	3000	3900	2737,957	**3838**	474,594	82,67%
40	6x150	1	25	5516	5888	4595,058	**5882**	623,437	86,43%
43	6x150	1	30	7130	7394	6428,563	**7390**	329,891	94,87%
44	6x150	2	30	2104	2276	6598,492	2276	327,938	95,03%

4 CONSIDERAÇÕES FINAIS

Este trabalho apresentou modificações no programa escrito em linguagem C utilizado em Azevedo et al. (2011), com o objetivo de resolver o problema de carregamento de contêineres em terminais portuários, conhecido como PCCTP.

Conforme os resultados computacionais apresentados, a redução média no tempo para se encontrar a solução, executando uma única vez, as 45 instâncias foi de aproximadamente 48,56%. Em se tratando especificamente das instancias cuja matriz de transporte era de curta distância, o tempo médio foi reduzido em 77,07%, enquanto que para as mistas a redução foi de 42,30% e, por fim, para as de longa distância a redução foi de 26,32%.

Além disso, com a nova regra introduzida, percebemos que após rodar as instâncias que apresentaram melhoras e aquelas que apresentaram uma solução de pior qualidade, quando comparadas com os resultados em Azevedoet al. (2011), por 32 vezes, obteve-se ou um resultado igual ou de melhor qualidade, porém com um tempo reduzido de 84,14%.

Vale ressaltar que a maioria dos resultados que se obteve melhora foi para matrizes de longa distância, porém em apenas duas se pode concluir que a melhora obtida é significativa, já que nos demais casos a diferença foi de no máximo 12 movimentos.

REFERÊNCIAS

ABENSUR, E.O. Um método heurístico integrado ao Simulated Annealing para a programação de tarefas em uma máquina. In: Simpósio Brasileiro de Pesquisa Operacional, 43., 2011, Ubatuba. **Anais...** SP: SBPO, 2011. p. 1559-1569.

AVRIEL, M.; PENN, M.; SHPIRER, N. Container ship stowage problem: complexity and connection to the coloring of circle graphs. **Discrete Applied Mathematics**, v. 103, p. 271-279, 2000.

AVRIEL, M.; PENN, M.; WITTENBOON, S. Stowage planning for container ships to reduce the number of shifts. **Annals of Operations Research**, v. 76, p. 55-71, 1998.

AZEVEDO, ANIBAL TAVARES; SENA, GALENO JOSÉ; CHAVES, ANTÔNIO AUGUSTO; RIBEIRO, CASSILDA MARIA. O Problema de Carregamento e Descarregamento de Contêineres em Terminais Portuários: Uma abordagem via Simulated Annealing. In: Simpósio Brasileiro de Pesquisa Operacional, 43., 2011, Ubatuba. **Anais...** SP: SBPO, 2011. p. 2023-2034.

BASU, Dipak. **Economic Models Methods, Theory and Applications.** 1st ed. Singapore: World Scientific Publishing Company, 2009. 215 p.

BISCHOFF, E. E.; RATCLIFF, M. S. W. Issues in the development of approaches to container loading. **Omega**, v. 23, n. 4, p. 377-390, 1995.

BLUM, C.; ROLI, A. Methaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys (CSUR),** v. 35, n. 3, p. 268-308, 2003.

COOK, S. A.: The complexity of Theorem Proving Procedures. **Proc. 3rd. ACM Symp. on the Theory of Computing**, p. 151-158, 1971.

DANTZIG, G. B. Reminiscences about the origins of linear programming. In: COTTLE, Richard W. (Ed.). **Mathematical programming: proceedings of the international Congress on Mathematical Programming, Rio de Janeiro, Brazil, 6-8 April, 1981.** Amsterdam: Elsevier Science Ltd, 1984, 6, p. 105-112.

FORTNOW, Lance. **The Golden Ticket: P, NP, and the Search for the Impossible**. Princeton: Princeton University University Press, 2013. 192 p.

HURD, GNU. **POSIX Threading Library**. Disponível em: <http://www.gnu.org/software/hurd/libpthread.html>. Acessado em: 15 ago. 2013.

IUSEM, Alfredo. P = NP ou as Sutilezas da Complexidade Computacional. **Matemática Universitária**, Rio de Janeiro, n. 5, p. 33-60, jun. 1987.

JOHNSON, DAVID B.; Mowry, Thomas A. **Finite Mathematics:** Practical Applications Docutech Version. 1st ed. New York: W. H. Freeman and Company, 2005. 538 p.

KIRKPATRICK, S.; GELLAT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, New York, v. 220, n. 4598, p. 671-680, 1983.

MUKHOPADHYAY, Saumitra. **Linear Programming with Game Theory**. 1st ed. Kolkata: Academic Publishers, 2011. 458 p.

WILSON, I.; ROACH, P. Container stowage planning: a methodology for generating computerised solutions. **Journal of the Operational Research Society,** v. 51, p. 1248-1255,2000.