UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Danilo Dias Pena

Infraestrutura de testes de algoritmos de otimização para reorganização de estoque em centros de distribuição

RECIFE
2017
UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Danilo Dias Pena

Infraestrutura de testes de algoritmos de otimização para reorganização de estoque em centros de distribuição

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Ricardo Martins de Abreu Silva.

RECIFE
2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Danilo Dias Pena

Algoritmos de otimização para reorganização de estoque em centros de distribuição

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em 14 de Dezembro de 2017.

Banca Examinadora:

_____ Orientador
Ricardo Martins de Abreu Silva
Doutor

_____ Examinador
Silvio de Barros Melo
Doutor

AGRADECIMENTOS

Agradeço a Deus e à toda a minha família. Agradeço a Universidade Federal de Pernambuco e aos professores do CIn-UFPE pelos conhecimentos compartilhados e todo aprendizado de vida adquirido enquanto estudante de graduação.

Agradeço aos amigos pelo apoio durante os momentos que abdiquei da presença deles para me dedicar ao curso e em especial aos que estavam ao meu lado em algum momento durante esse trajeto. Agradeço em especial os meus pais, pelo acompanhamento e o auxílio em todos os momentos.

Aos meus tutores e professores que me incentivaram ao aprendizado técnico e pessoal, por ensinamentos muito além do universo da engenharia, computação ou das ciências.

A Leonardo Bueno, por contribuir no desenvolvimento deste trabalho e pelos conhecimentos transmitidos.

Enfim, a todas as pessoas que direta ou indiretamente participaram dessa etapa da minha vida acadêmica que agora concluo.

RESUMO

A otimização da reorganização de pacotes em grandes armazéns impacta diretamente e de forma positiva nos seus rendimentos. Grandes armazéns necessitam, em grande frequência, de reorganizações por motivos sazonais, de mercado, logísticos, etc. Determinados tipos de produtos saem mais em uma época do ano do que em outras, produtos postos em promoção vão ser liquidados e vão sair do estoque mais rapidamente, novos produtos são recebidos constantemente nos depósitos, todos esses são exemplos que demandam por uma reorganização frequente no estoque.

Imprecisões e falhas de projeto e operação de armazéns podem resultar em grandes atrasos na entrega de produtos e até na falta de itens em inventários de clientes finais. Entre as causas principais de falta de inventário se encontram: incongruência entre a capacidade e a frequência de abastecimento; infrequência, atraso, ou inexistência de reposição de artigos em prateleiras; inventário inexato ou errado; armazenamento com organização inadequadas, rompimento de embalagens e pouca disponibilidade; mal projeto do armazenamento e baixa qualidade dos serviços operacionais.

Devido à sua importância estratégica, a gestão eficiente de um estoque de um centro de distribuição contém diversos problemas que podem ser resolvidos via métodos de otimização. Neste universo, são frequentemente explorados pela literatura os problemas de: Dimensionamento de ambientes, organização (e layout) de departamentos, organização ou layout de estoque, padrão de empilhamento, metodologia de armazenamento e recuperação de produtos.

Determinar a forma otimizada de estocagem de produtos é um problema que vem sido estudado há décadas, porém, a necessidade de mudança nos estoques, em grande frequência, trouxe um novo problema à tona, a estratégia de obtenção de uma estocagem em particular, dada a situação atual do estoque (o estado atual das cargas no estoque). Neste cenário, o estudo atual se propõe a implementar uma infraestrutura de testes capaz de comparar técnicas já apresentadas por Héctor J. Carlo e Germán E. Giraldo, em sua tese de mestrado, através da preparação de benchmarks e simulação de cenários maiores.

Palavras-chave: Otimização, Realocação de objetos, Reorganização de pacotes

Sumário

Lista de figuras

Tabela de siglas

AS/RS	***Automated Storage/Retrieval System***
SLAP	***Storage location assignment policy***
SLAP/NI	***Storage location assignment policy - No information***
SLAP/II	***Storage location assignment policy - Item information***
RWW	***Rearrange While Working***

1 INTRODUÇÃO

1.1 Motivação

A gestão de um estoque de um centro de distribuição contém diversos problemas que podem ser resolvidos via métodos de otimização. Devido à sua relevância no tempo de entrega de produtos, centros de distribuição de grandes redes de correios, vendedores de comércio eletrônico, e fabricantes de produtos, apresentam enorme interesse na melhoria das suas operações. Neste universo, são frequentemente explorados pela literatura os problemas de: Dimensionamento de ambientes, organização (ou layout) de departamento, organização ou layout de estoque, padrão de empilhamento, metodologia de armazenamento e recuperação de produtos.

Figura 1: Cenário de armazém.

Um dos problemas pouco explorados na literatura, que se tornou mais importante com o advento de sistemas automáticos para recuperação e armazenamento de veículos e a crescente demanda nos setores de comércio eletrônico é a reorganização periódica do estoque.

Uma organização de um estoque não se mantém ótima por todo o período de vida do armazém, pois existem fatores externos que alteram a demanda por produtos. Sejam mudanças sazonais, surgimento de novos produtos, ou depreciação de produtos

antigos. Por estes motivos, é necessário, mudar a organização do estoque, de forma periódica e de modo que produtos de maior demanda sejam mais rapidamente acessados e distribuídos, reduzindo o tempo médio de entrega de bens.

A determinação da sequência ideal de movimentos para alocação de cargas num armazém tem sido extensivamente estudada na literatura da logística, porém o processo de realocar as cargas de um conjunto de posições inicial para um conjunto de posições final, por ser mais recente, ainda vem sendo aperfeiçoado.

1.2 Objetivo

Dados os problemas da área e os trabalhos recentes realizados por German Giraldo [2] e Jennifer Pazour [4], este projeto tem como objetivo geral a elaboração de um ambiente de testes para problemas de reorganização de estoque e a validação do ambiente com a reprodução dos resultados obtidos pelos trabalhos citados.

Os objetivos específicos são:

- Criar gerador de cenários de teste em C++ para testes de n espaços vazios e estoque de cargas unitárias fora de operação.
- Criar gerador de cenários de teste em C++ para testes de n espaços vazios e estoque de cargas unitárias em operação.
- Transcrever algoritmos de Giraldo Gonzalez [2] em C++
- Reproduzir resultados do trabalho citado no novo ambiente de testes

1.3 Estrutura do trabalho

Para atingir os objetivos, o trabalho está estruturado da seguinte forma: no Capítulo 2 é feito um estudo geral sobre o estado-da-arte atual, no Capítulo 3 são apresentados métodos necessários para criação de cenários de testes e o algoritmo que será testado; no Capítulo 4 são apresentados os resultados obtidos nos experimentos e, no Capítulo 5, a conclusão e a proposta de trabalhos futuros.

2 Fundamentação teórica

2.1 Estado-da-arte

O primeiro trabalho citado na literatura sobre este problema é o trabalho de Christofides [3], que se referiu ao problema como Reajustamento de Depósitos (Warehouse Rearrangement), propôs um algoritmo de programação dinâmica para reorganização de estoques com itens pertencentes a ciclos, o estudo considera apenas itens contidos em ciclos. Ciclos existem quando há dependências entre objetos, em relação a posição atual e final de cada um deles. É assumido que itens em um cíclo são movidos sequencialmente

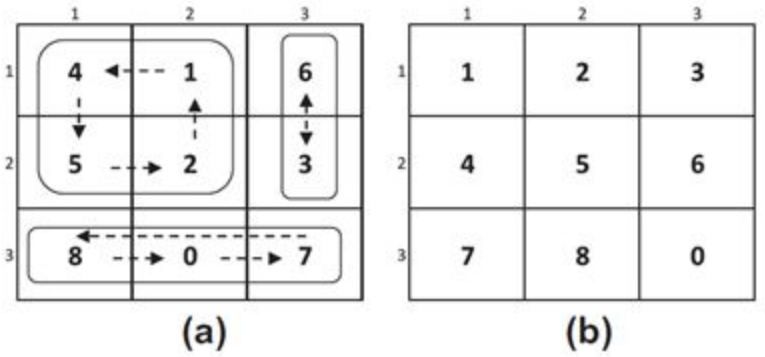

Figura 2: 3 ciclos identificados na realocação (a) e Estado final após realocação (b).

Como apenas ciclos são considerados, e eles são executados separadamente, um após o outro, as posições que não estão sendo ocupadas por nenhum objeto permanecem as mesmas antes e depois do reajuste, ou seja, o estudo assume que as posições não-ocupadas são fixas durante o processo.

Os autores propuseram um algoritmo de dois estágios que constrói a sequência de movimentos carregados que minimiza os custos necessários para o reajuste de um determinado depósito. O primeiro estágio identifica como cada um dos ciclos pode ser reposicionado, enquanto o segundo utiliza programação dinâmica para determinar a sequência na qual os ciclos serão movidos.

A metodologia de Christofides [3], apesar de importantíssima na área, não é capaz de resolver problemas de maior complexidade e mais próximos de casos reais. Como o próprio autor indica, seu algoritmo só encontra o caso ótimo em situações

específicas, quando os itens estão em ciclo, se movem sequencialmente e onde os espaços vazios permanecem fixos.

Um estudo semelhante foi apresentado por Linn and Wysk (citação), definido como Política de Restauração (Restoring Policy), propôs uma política de ajuste por AS/RS usando carregamento baseado em classes aonde reposições eram feitas no tempo ocioso, movendo itens para mais perto do ponto de entrada e saída. O estudo foi limitado ao conceito da política de ajuste e não apresenta resultados computacionais.

Muralidharan et al também estudou como reposicionar itens utilizando AS/RS com carregamento baseado em classes, o problema foi definido como Realocação (*Shuffling*) e formulado como *Precedence Constrained Selective Asymmetric Traveling Salesman Problem*, dada a complexidade computacional do problema, os autores propuseram duas heurísticas: Realocação com o vizinho mais próximo *Shuffling with Nearest Neighbor* (SNN) e Realocação com inserção (*Shuffling with insertion*, SI). Através de resultados simulados os autores concluíram que a utilização do tempo ocioso para atualizar configuração do depósito aumenta a eficiência da operação com AS/RS.

O estudo apresentado por Chen [7] foca em realocar itens no depósito através de decisões sobre onde itens devem ser realocados e quais os seus destinos, tendo em vista satisfazer a demanda necessária durante horários de pico. Um modelo matemático para o problema e mais duas heurísticas (*two-stage heuristic* e *Tabu Search*) foram apresentados.

A definição da melhor localização para cada item do armazém com o intuito de minimizar o esforço na manipulação destes itens, conhecido como *storage location assignment problem* (SLAP) (Hausman,Schwarz,& Graves, 1976), também foi bastante explorada na literatura. O SLAP pode ser classificado de acordo com a quantidade de informação obtida sobre as cargas dos depósitos: SLAP/II - *item information*, SLAP/PI - *product information* ou SLAP/NI - *no information*.

O problema SLAP/II assume que todas as informações referentes aos itens do depósito são conhecidas. Uma política comumente utilizada no SLAP/II é a *Duration-of-Stay* (DOS) onde os itens com menor tempo esperado de saída são alocados nos lugares mais acessíveis (Goetschalckx & Ratliff,1990).

No SLAP/PI, as informações disponíveis são a nível de produto, os produtos são classificados em classes, de acordo com características físicas ou requisitos. O objetivo é determinar localizações para cada tipo de classe. Hausman (1976) descreveu o caso onde o número classes é igual ao número de produtos (n) como *Dedicated* (*Fixed slot*)

Storage onde há um conjunto de locais pré-definidos para o depósito. No caso em que há apenas uma classe a política de armazenamento intitulada *Randomized* ou *Floating Slot*, é determinado que qualquer item pode ser armazenado em qualquer localização. Quando o número de classes é entre 2 e n-1, o problema é conhecido como *Class-Based Storage*.

Já no SLAP/NI, onde não há nenhuma informação sobre os itens que serão armazenados, apenas políticas simples de armazenamento podem ser aplicadas.

Além da classificação pela quantidade de informação dos produtos a serem armazenados, o SLAP também pode ser classificado como estático ou dinâmico. No SLAP estático o fluxo de produtos é considerado constante durante o planejamento, já no dinâmico, há um ajuste contínuo nas atribuições de armazenamento baseado no fluxo de produtos. A maioria dos estudos realizados apresenta um foco na versão estática do problema.

Durante os anos, diversas metodologias foram utilizadas para a solução do problema de reorganização. O trabalho mais importante encontrado na área, atualmente (de Carlos e Giraldo (2012) [6]), analisa o problema de forma extensiva e propõe uma metodologia de *Rearrange-While-Working* (RWW) para garantir organização contínua do estoque. Por outro lado, Jennifer e Héctor (2014) [4], analisa a modelagem linear do problema ao longo de toda a literatura, e propõe uma heurística para resolução do problema de reestruturação do armazém (*warehouse reshuffling problem*), conseguindo, através de benchmarks, exibir resultados melhores que o trabalho de German Giraldo [2]. Resultados esses, obtidos por meio da análise do tratamento dos ciclos e pelo uso da estratégia de *double-handling*, que consiste em mover o item mais de uma vez até que chegue ao seu local de destino. *Double-handling* é uma estratégia benéfica na resolução do problema de reestruturação do armazém, pelo fato de ser não-gulosa e por levar em consideração que múltiplos itens podem precisar ser reposicionados.

3 Métodos

3.1 Criação do gerador de cenários de teste

O primeiro passo da implementação foi o desenvolvimento de um gerador de cenários de testes para o problema de realocação de produtos em armazéns. A linguagem utilizada foi a C++.

3.1.1 C++

É uma linguagem de programação compilada multi-paradigma (incluindo orientação a objetos) e de uso geral. Desde os anos 1990 é uma das linguagens comerciais mais populares, sendo bastante usada na academia por seu grande desempenho e base de utilizadores.

3.1.2 Bibliotecas de C++ utilizadas:

- Iostream: Define o fluxo padrão de entrada e saída de objetos.
- Algorithm: Define uma coleção de funções especialmente projetadas para serem usadas em intervalos de elementos. Foi utilizada a função random_shuffle para embaralhar vetores.
- Vector: Vetores são recipientes em sequência, representam arrays que podem mudar de tamanho.
- Ctime: Contém definições de funções para obter e manipular informações de data e hora. Foi utilizada na geração de números aleatórios.
- Cstdlib: Define várias funções de propósito geral, incluindo gerenciamento de memória dinâmica, geração de números aleatórios, comunicação com o meio ambiente, aritmética inteira, pesquisa, triagem e conversão.

- Math: Declara um conjunto de funções para calcular operações e transformações matemáticas comuns.

- Fstream: Contém fluxos de entrada / saída para operações com arquivos.

O gerador recebe 3 parâmetros como entrada e escreve o cenário de testes em um arquivo de extensão CSV (*comma-separated values*), formato que pode conter valores separados por algum delimitador, ponto e vírgula (;) por exemplo, pode ser criado em qualquer editor de texto e lido em uma planilha de textos, onde cada linha/coluna será as linhas do arquivo, separados por vírgula (,), ponto-e vírgula, etc. O formato foi escolhido por ser amplamente utilizado em meio acadêmico para fins de registro de dados.

Os três parâmetros solicitados pelo gerador são:

- Imax: Número total de posições no armazém (inteiro).

- Tamanho dos vetores Ik e Fk, vetores com a posição inicial e final do item k, respectivamente: Quantidade de produtos que serão realocados (número inteiro, menor ou igual a Imax)

- Porcentagem de organização:Número inteiro (entre 0 e 100), indica a porcentagem de posições iniciais que permanecerão inalteradas no vetor de posições finais.

O gerador então constrói um vetor do tamanho do número total de posições (Imax) em ordem crescente. Um exemplo para o número 10 como entrada:

Ik

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Em seguida embaralha através de um número gerado aleatoriamente, 10*l vezes, sendo l o tamanho do vetor.

Ik

6	**3**	**10**	**1**	**8**	**5**	**2**	**7**	**4**	**9**

O gerador constrói o vetor de posições finais, que inicialmente será igual ao de posições iniciais.

Ik

6	**3**	**10**	**1**	**8**	**5**	**2**	**7**	**4**	**9**

Fk

6	**3**	**10**	**1**	**8**	**5**	**2**	**7**	**4**	**9**

Baseado na porcentagem de organização, será feito um embaralhamento da parte final do vetor de posições finais Fk, a parte inicial dos dois vetores permanecem iguais. Para uma porcentagem de 40%, no exemplo anterior:

Ik

6	**3**	**10**	**1**	**8**	**5**	**2**	**7**	**4**	**9**

Fk

6	**3**	**10**	**1**	**4**	**7**	**8**	**9**	**5**	**2**

Após o embaralhamento da parte final de Fk, as primeiras posições de Ik e Fk são idênticas, respeitando a porcentagem de organização. Para que os vetores tenham uma melhor uniformidade, essas primeiras posições serão trocadas tanto em Ik quanto em Fk, com outras posições aleatórias do vetor, as posições de troca serão as mesmas para os dois vetores.

Ik

2	**8**	**5**	**9**	**3**	**10**	**6**	**7**	**4**	**1**

Fk

8	**4**	**7**	**2**	**3**	**10**	**6**	**9**	**5**	**1**

Observou-se que em alguns casos, mais frequentemente em vetores de tamanhos maiores, após o embaralhamento aleatório, novas posições iguais surgiram, foi implementada então uma verificação da consistência dos cenários gerados, para garantir que todo cenário gerado respeite os parâmetros de entrada.

A matriz dijk, construída no gerador, diz respeito ao custo do item k ser movido da posição i para a posição j, o custo dos itens serem movidos de uma posição para a mesma é definido como 0, os outros custos são gerados aleatoriamente com valores entre 0 e 100.

Os dados então são escritos no arquivo CSV de nome “example.csv”, da seguinte forma:

- Linha 1: Imax
- Linha 3: Ik, com as posições iniciais estando separadas por vírgulas.
- Linha 5: Fk, com as posições finais estando separadas por vírgulas.
- Linha 7 em diante: dijk, com os custos estando separados por vírgulas

3.2 Criação do parser

O segundo passo da implementação foi a criação do parser, que recebe o nome de um arquivo de extensão CSV como entrada e cria o ambiente necessário para a realocação dos itens.

O parser possui um construtor que recebe o nome do arquivo CSV, e a partir dos dados contidos neste arquivo, constrói e define as seguintes variáveis:

- Imax: Conjunto dos locais de armazenamento, indexados em i, j = 0, 1, 2, …, |I| (aonde 0 é o ponto do I/O)
- Omax: Número de posições abertas.
- Kmax: Conjunto de itens, indexados em k = 1, 2, …, |K|.
- Ik: Vetor com as localizações iniciais dos itens.

- Fk: Vetor de localizações finais, indexado em k.
- Ii: Vetor com os itens em suas respectivas localizações iniciais
- Fi: Vetor com os itens em suas respectivas localizações finais
- OIo: Posições abertas antes do reajuste.
- OFo: Posições abertas depois do reajuste.
- Cmax: Conjunto de ciclos indexados em c = 1, 2, …, |C|
- Cc: Matriz de conjunto dos itens que pertencem ao ciclo c, indexada em c.
- Nmax: Número de itens que não pertencem a um ciclo.
- N: Conjunto de itens que não pertencem a um ciclo.
- Dij: Matriz com os custos de mover os itens da localização i para a j.

O parser começa obtendo todas as informações contidas no arquivo CSV, em seguida faz uma checagem nos parâmetros Ik, Fk, Imax, Omax e Dij, para garantir que não há incoerências nos dados. Os tamanhos de Ik e de Fk devem ser iguais ao (tamanho de Imax - tamanho e Omax), Omax não deve ser maior que Imax e Dij deve ser uma matriz com dimensão do tamanho de Imax, todos os valores precisam ser números inteiros.

Em seguida é iniciada construção das demais variáveis: Ii, Fi, OIo, OFo, L, N e Cc.

parser.h

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

class Parser{
public:
    Parser(string name);
    unsigned imax;                // Set of storage locations, indexed on i, j = 0, 1, 2, . . . ,
                                  //  |I| (where 0 is the I/O point).
    unsigned omax;               // Number of empty positions
    unsigned kmax;               // Set of items, indexed on k = 1, 2, . . . , |K|.
    vector<int> Ik;              // The initial location of item k
    vector<int> Fk;              // The final location of item k;
    std::vector<int> Ii;        // Initial items in each location
    std::vector<int> Fi;       // Final items in each location
    std::vector<int> OIo;        // The open locations
    std::vector<int> OFo;        // The open locations
    unsigned cmax;                //C – set of cycles, indexed on c = 1, 2, . . . , |C|.
    std::vector<std::vector<int> > Cc;        //Cc – set of items that belong to cycle c, indexed on c.
    unsigned nmax;                // Items not in a cycle
    std::vector<int> N;            //N – set of items that do not belong to a cycle (i.e., non-cycle items),
                                   // indexed on k.
    vector<vector<double> > dij;    // Distance to travel from location i to j.
    vector<vector<double> > fields;   // Variable used to store the csv fields
    const vector<int> getIk() const;
    const vector<int> getFk() const;
    const vector<int> getIi() const;
    const vector<int> getFi() const;
    const vector<int> getOIo() const;
    const vector<int> getOFo() const;
    const vector<vector<double> > getDij() const;
    int getImax();
    int getOmax();
    unsigned getKmax();
};
```

Figura 3: Classe parser contida no pacote parser.h.

O parser possui também, métodos de acesso as variáveis necessários para a resolução do problema, como getIk(), getFk(), getIi(), getFi(), getOIo(), getOFo(), getDij(), getImax(), getOmax(), getKmax().

3.3 Implementação do algoritmo

O algoritmo implementado para os testes foi baseado na Heurístico 3 (H3 caso 0): *Distancia más corta* da tese de dissertação de Doutorado de Giraldo [2]. O algoritmo utiliza a distância dos objetos a serem movidos em relação à posição aberta se o espaço vazio estiver na posição desejada pelo objeto, caso não haja nenhuma posição desejada disponível, o objeto mais próximo deve ser movido para a posição vazia. E o

objeto que possui como posição final a nova posição vazia é movido, os passos são repetidos até que todos os objetos estejam na posição final desejada.

Os passos da heurística são:

- Passo 1: Encontre A(0) e B(0) se A(0) é diferente de B(0) e realoque o objeto que se encontra na posição A(B(0)) para a localização A(0) e determine o custo necessário para realizar esta operação. Repita este passo até chegar no arranjo final, ou até quando a posição aberta esteja em sua posição final, neste caso, siga para o Passo 2.

- Passo 2: Mova o objeto que está mais próximo do espaço vazio. Em caso de empate, utilize o objeto com menor índice.

Para calcular o custo, baseado nas literaturas atuais, definiu-se que o custo de carregar o objeto da posição i para a j será definido pela matriz Dij, e o custo de carregamento vazio, onde a máquina vazia, vai da posição i para a j será definido como metade do valor contigo em Dij para os mesmos i e j.

4 Resultado e análise

Foi utilizado inicialmente um cenário de teste do artigo de Christofides [3], o resultado apresentado pelo algoritmo implementado se mostrou bastante próximo do obtido pelo algoritmo de Christofides (1148):

```
Cycles:
0 2 4
1 9 7
3 5
Non Cycles:
6 8

Ik: 0  1  2  3  4  5  6  7  8  9
Fk: 2  9  4  5  0  3  6  1  8  7
Ii: 0  1  2  3  4  5  6  7  8  9  -1
Fi: 4  7  0  5  2  3  6  9  8  1  -1
Initial open position: 10
Final open position 10
0  60  72  83  51  58  94  64  71  85  90
63  0  71  75  68  85  90  60  78  87  96
58  80  0  71  86  65  68  92  73  83  88
70  69  88  0  93  66  74  79  67  81  75
62  81  55  73  0  78  84  58  92  91  66
71  80  73  71  77  0  82  79  91  83  86
50  63  55  82  94  58  0  64  73  87  78
72  92  78  84  73  81  76  0  66  57  66
50  60  72  58  63  77  82  90  0  85  84
66  65  69  72  80  68  77  87  89  0  98
70  86  85  75  68  92  90  88  69  83  0

item a mover: 4
0  1  2  3  -1  5  6  7  8  9  4  custo: 100.
item a mover: 2
0  1  -1  3  2  5  6  7  8  9  4  custo: 228.
item a mover: 0
-1  1  0  3  2  5  6  7  8  9  4  custo: 331.
item a mover: 4
4  1  0  3  2  5  6  7  8  9  -1  custo: 445.
item a mover: 7
4  1  0  3  2  5  6  -1  8  9  7  custo: 543.
item a mover: 9
4  1  0  3  2  5  6  9  8  -1  7  custo: 671.
item a mover: 1
4  -1  0  3  2  5  6  9  8  1  7  custo: 804.
item a mover: 7
4  7  0  3  2  5  6  9  8  1  -1  custo: 939.
item a mover: 3
4  7  0  -1  2  5  6  9  8  1  3  custo: 1051.
item a mover: 5
4  7  0  5  2  -1  6  9  8  1  3  custo: 1168.
item a mover: 3
4  7  0  5  2  3  6  9  8  1  -1  custo: 1297.
Program ended with exit code: 0
```

Figura 4: Cenário de teste apresentado por Christofides com Imax = 11, 1 posição vazia e 10% de organização. Na figura, temos a identificação dos ciclos, itens que não pertencem a nenhum ciclo. Vetores de localizações iniciais e finais (Ik e Fk respectivamente), vetores com itens em suas

localizações iniciais e finais (Ii e Fi respectivamente). A localização da posição aberta no início e no fim da realocação, a matriz Dij e a execução do algoritmo. O custo é cumulativo.

Em seguida, foram usados cenários feitos no gerador de cenários de teste implementados, com porcentagens de organização e número de espaços vazios variantes.

```
Tap imax:
12
Tap vector length:
10
Tap the percentage of organization:
25
number of equal positions: 2

Cycles:
0 3 7 5 6
Non Cycles:
2 4 9 1 8

Ik: 10  4  1  5  9  2  8  0  3  6
Fk: 5  3  11  0  9  8  10  2  1  6
Ii: 7  2  5  8  1  3  9  -1  6  4  0  -1
Fi: 3  8  7  1  -1  0  9  -1  5  4  6  2
Initial open position: 7  11
Final open position 4  7
0  52  39  41  94  76  35  63  29  99  16  83
95  0  3  10  26  73  50  93  19  11  75  85
30  31  0  76  48  91  24  31  57  24  37  68
87  74  49  0  42  78  81  33  15  48  45  64
60  59  0  46  0  45  85  30  90  21  64  50
83  25  31  28  1  0  52  16  70  58  53  61
32  83  45  69  16  79  0  24  83  2  42  22
45  25  58  0  2  89  52  0  86  98  30  98
11  88  48  52  49  4  64  80  0  55  67  38
47  24  74  29  13  27  4  16  69  0  4  55
51  99  5  42  63  35  31  7  67  5  0  23
94  92  52  15  90  92  48  96  32  76  84  0

item a mover: 2
7  -1  5  8  1  3  9  -1  6  4  0  2  custo: 131.
item a mover: 8
7  8  5  -1  1  3  9  -1  6  4  0  2  custo: 212.
item a mover: 1
7  8  5  1  -1  3  9  -1  6  4  0  2  custo: 271.
item a mover: 5
7  8  -1  1  5  3  9  -1  6  4  0  2  custo: 343.
item a mover: 7
-1  8  7  1  5  3  9  -1  6  4  0  2  custo: 412.
item a mover: 3
3  8  7  1  5  -1  9  -1  6  4  0  2  custo: 540.
item a mover: 0
3  8  7  1  5  0  9  -1  6  4  -1  2  custo: 583.
item a mover: 6
3  8  7  1  5  0  9  -1  -1  4  6  2  custo: 685.
item a mover: 5
3  8  7  1  -1  0  9  -1  5  4  6  2  custo: 806.
Program ended with exit code: 0
```

Figura 5: Cenário de teste com Imax = 12, 2 posições vazias e 25% de organização. Na figura, temos a identificação dos ciclos, itens que não pertencem a nenhum ciclo. Vetores de

localizações iniciais e finais (Ik e Fk respectivamente), vetores com itens em suas localizações iniciais e finais (Ii e Fi respectivamente). A localização da posição aberta no início e no fim da realocação, a matriz Dij e a execução do algoritmo. O custo é cumulativo.

```
Tap imax:
12
Tap vector length:
10
Tap the percentage of organization:
50
number of equal positions: 5

Cycles:
Non Cycles:
1 2 3 4 5 8 0 6 7 9

Ik: 4  8  1  11  7  5  9  10  3  0
Fk: 9  8  6  11  7  5  1  4  3  10
Ii: 9  2  -1  8  0  5  -1  4  1  6  7  3
Fi: -1  6  -1  8  7  5  2  4  1  0  9  3
Initial open position: 2  6
Final open position 0  2
0  52  14  91  0  54  6  13  72  60  36  9
77  0  95  75  53  46  69  29  38  47  1  56
51  26  0  74  29  92  17  60  39  15  60  6
57  10  28  0  96  57  23  14  72  91  60  83
96  39  19  72  0  53  79  32  56  75  60  92
94  72  40  92  70  0  62  22  69  1  52  72
87  3  63  89  84  64  0  49  70  57  60  93
62  78  26  76  31  7  20  0  90  75  97  98
30  21  83  14  24  13  26  63  0  58  26  90
60  7  31  36  82  74  98  85  13  0  78  44
39  23  59  7  24  41  5  30  50  33  0  25
3  45  43  7  59  67  0  27  48  36  29  0

item a mover: 2
9  -1  -1  8  0  5  2  4  1  6  7  3  custo: 91.
item a mover: 6
9  6  -1  8  0  5  2  4  1  -1  7  3  custo: 126.
item a mover: 0
9  6  -1  8  -1  5  2  4  1  0  7  3  custo: 227.
item a mover: 7
9  6  -1  8  7  5  2  4  1  0  -1  3  custo: 290.
item a mover: 9
-1  6  -1  8  7  5  2  4  1  0  9  3  custo: 374.
Program ended with exit code: 0
```

Figura 6: Cenário de teste com Imax = 12, 2 posições vazias e 50% de organização. Na figura, temos a identificação dos ciclos, itens que não pertencem a nenhum ciclo. Vetores de localizações iniciais e finais (Ik e Fk respectivamente), vetores com itens em suas localizações iniciais e finais (Ii e Fi respectivamente). A localização da posição aberta no início e no fim da realocação, a matriz Dij e a execução do algoritmo. O custo é cumulativo.

```
Tap imax:
12
Tap vector length:
10
Tap the percentage of organization:
10
number of equal positions: 1

Cycles:
Non Cycles:
6 7 0 2 1 3 9 5 4 8

Ik: 8  11  5  0  6  9  10  4  1  3
Fk: 5  0  10  3  8  6  2  4  11  9
Ii: 3  8  -1  9  7  2  4  -1  0  5  6  1
Fi: 1  -1  6  3  7  0  5  -1  4  9  2  8
Initial open position: 2  7
Final open position 1  7
0  0  76  23  58  37  17  88  64  1  63  3
32  0  71  10  92  26  82  44  77  74  90  74
65  90  0  29  81  64  64  51  87  87  24  99
34  82  35  0  76  20  59  15  88  32  84  59
62  47  83  86  0  19  85  9  71  90  53  70
26  49  2  49  28  0  40  10  75  96  45  68
14  50  79  70  84  46  0  0  84  73  21  62
78  49  26  76  42  95  41  0  25  17  50  27
77  24  77  23  39  29  64  53  0  50  16  73
64  39  95  79  97  8  92  21  47  0  75  90
61  32  26  53  49  68  51  51  88  89  0  56
54  61  66  26  71  5  99  97  79  58  49  0

item a mover: 6
3  8  6  9  7  2  4  -1  0  5  -1  1  custo: 50.
item a mover: 2
3  8  6  9  7  -1  4  -1  0  5  2  1  custo: 127.
item a mover: 0
3  8  6  9  7  0  4  -1  -1  5  2  1  custo: 200.
item a mover: 4
3  8  6  9  7  0  -1  -1  4  5  2  1  custo: 304.
item a mover: 5
3  8  6  9  7  0  5  -1  4  -1  2  1  custo: 421.
item a mover: 9
3  8  6  -1  7  0  5  -1  4  9  2  1  custo: 488.
item a mover: 3
-1  8  6  3  7  0  5  -1  4  9  2  1  custo: 543.
item a mover: 1
1  8  6  3  7  0  5  -1  4  9  2  -1  custo: 626.
item a mover: 8
1  -1  6  3  7  0  5  -1  4  9  2  8  custo: 700.
Program ended with exit code: 0
```

Figura 7: Cenário de teste com Imax = 12, 2 posições vazias e 10% de organização. Na figura, temos a identificação dos ciclos, itens que não pertencem a nenhum ciclo. Vetores de localizações iniciais e finais (Ik e Fk respectivamente), vetores com itens em suas localizações

iniciais e finais (Ii e Fi respectivamente). A localização da posição aberta no início e no fim da realocação, a matriz Dij e a execução do algoritmo. O custo é cumulativo.

```
Tap imax:
11
Tap vector length:
10
Tap the percentage of organization:
50
number of equal positions: 5

Cycles:
Non Cycles:
0 2 6 7 8 9 1 3 5 4

Ik: 10  9  5  7  4  0  6  3  8  1
Fk: 10  7  2  0  5  4  6  3  8  1
Ii: 5  9  -1  7  4  2  6  3  8  1  0
Fi: 3  9  2  7  5  4  6  1  8  -1  0
Initial open position: 2
Final open position 9
0  84  90  7  43  48  98  72  45  93  20
53  0  66  11  26  23  33  1  26  42  80
55  32  0  41  55  61  27  58  46  72  63
28  91  43  0  41  45  89  71  85  36  56
72  8  58  45  0  29  4  50  15  16  91
64  66  75  1  17  0  47  15  89  45  75
11  33  21  49  55  15  0  25  34  36  36
50  69  53  74  11  6  30  0  40  4  32
88  54  8  49  61  83  34  99  0  37  64
13  28  41  57  9  66  47  18  80  0  12
19  19  39  37  36  19  28  24  92  60  0

item a mover: 2
5  9  2  7  4  -1  6  3  8  1  0  custo: 84.
item a mover: 4
5  9  2  7  -1  4  6  3  8  1  0  custo: 140.
item a mover: 5
-1  9  2  7  5  4  6  3  8  1  0  custo: 215.
item a mover: 3
3  9  2  7  5  4  6  -1  8  1  0  custo: 290.
item a mover: 1
3  9  2  7  5  4  6  1  8  -1  0  custo: 354.
Program ended with exit code: 0
```

Figura 8: Cenário de teste com Imax = 11, 1 posição vazia e 50% de organização. Na figura, temos a identificação dos ciclos, itens que não pertencem a nenhum ciclo. Vetores de localizações iniciais e finais (Ik e Fk respectivamente), vetores com itens em suas localizações iniciais e finais (Ii e Fi respectivamente). A localização da posição aberta no início e no fim da realocação, a matriz Dij e a execução do algoritmo. O custo é cumulativo.

4.1 Análise

É possível verificar pelos resultados que a porcentagem de organização e o número de posições vazias impactam de maneira evidente na complexidade do problema, ou seja, o número de passos e os custos são afetados diretamente.

5 Considerações finais

5.1 Conclusão

Este trabalho teve como objetivo, explorar de forma geral a literatura da realocação de objetos, focando em problemas de SLAP estático, onde um conjunto de itens precisam ser realocados de suas posições iniciais para suas posições finais, montar um cenário aleatório de testes e utilizar uma técnica já conhecida para resolver o problema. Para tal, foi implementado um gerador de cenário de testes [descrito em 3.1], um parser [descrito em 3.2] e um algoritmo [descrito em 3.3] na linguagem de programação C++.

5.2 Trabalhos futuros

Buscar técnicas para otimizar os resultados já obtidos na literatura, explorando o desempenho das melhores heurísticas analisadas em cenários maiores. Propor novas metodologias para resolução do problema de organização de estoques de cargas unitárias fora de operação.

6 Referências

[1] Gu, J., Goetschalckx, M., & McGinnis, L. F. (2007). Research on warehouse operation: a comprehensive review. European Journal of Operational Research, 177(1), 1–21.

[2] Giraldo, G. E. (2011) Metodología Para la Reorganización Perpetua de Almacenes. Master's thesis. University of Puerto Rico – Mayaguez.

[3] Christofides, N., & Colloff, I. (1973). The rearrangement of items in a warehouse. Operations Research, 21(2), 577–589.

[4] Jennifer A. Pazour, Héctor J. Carlo, Warehouse reshuffling: Insights and optimization, In Transportation Research Part E: Logistics and Transportation Review, Volume73,2015,Pages207-226,ISSN1366-5545,https://doi.org/10.1016/j.tre.2014.11.002.(http://www.sciencedirect.com/science/article/pii/S1366554514001914)

[5] Roodbergen, K. J., & Vis, I. F. A. (2009). A survey of literature on automated storage and retrieval systems. European Journal of Operational Research, 194, 343–362

[6] Carlo, H.J., Giraldo, G.E., 2012. Toward perpetually organized unit-load warehouses. Comput. Ind. Eng. 64 (4), 1003–1012.

[7] Chen, L., Langevin, A., Riopel, D., 2011. A tabu search algorithm for the relocation problem in a warehousing systems. Manage. Sci. 22 (6), 629-638.