DOUGLAS MATUZALEM PONTES BELO LANÇA

Orientador: Marco Antonio Moreira de Carvalho

UM ALGORITMO HEURÍSTICO APLICADO À MINIMIZAÇÃO DO ESTOQUE INTERMEDIÁRIO EM SISTEMAS INDUSTRIAIS

Ouro Preto

Março de 2017

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

UM ALGORITMO HEURÍSTICO APLICADO À MINIMIZAÇÃO DO ESTOQUE INTERMEDIÁRIO EM SISTEMAS INDUSTRIAIS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

DOUGLAS MATUZALEM PONTES BELO LANÇA

Ouro Preto
Março de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Um Algoritmo Heurístico Aplicado à Minimização do Estoque Intermediário em Sistemas Industriais

DOUGLAS MATUZALEM PONTES BELO LANÇA

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Marco Antonio Moreira de Carvalho – Orientador
Universidade Federal de Ouro Preto

Dr. Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Dr. Puca Huachi Vaz Penna
Universidade Federal de Ouro Preto

Ouro Preto, Março de 2017

Resumo

Em ambientes industriais, diferentes processos produtivos são implementados nas várias linhas de produção. Estas linhas são divididas em estágios, de acordo com o tipo de produto fabricado. Em geral, é necessário que os processos produtivos sejam otimizados de acordo com diferentes objetivos, como, por exemplo, a minimização da utilização de matéria prima e a economia de energia. Outro objetivo comumente considerado é que a sequência em que os produtos são processados em cada estágio das linhas de produção seja otimizada para reduzir o uso de estoque intermediário, de maneira a aumentar o ritmo de produção e também o ritmo de entrega dos produtos fabricados aos consumidores. Existem diversos aspectos que interferem na eficiência das linhas de produção, dando origem aos problemas de otimização industriais. A otimização destes problemas na maioria das vezes não é trivial, portanto, exige estudo e investimento. Neste trabalho será descrito com detalhes o Problema de Minimização de Espalhamento de Ordens (*Minimization of Order Spread Problem - MORP*), que tem como foco a diminuição do uso de estoque intermediário nas indústrias. Para a resolução deste problema é proposto um método heurístico em duas fases para a geração da solução inicial e a melhoria da mesma utilizando o método Busca Local Iterada, ambos métodos já existentes na literatura. Os experimentos computacionais foram realizados a partir de um grande conjunto de instâncias propostas na literatura e foram obtidos novos melhores resultados para algumas instâncias. Tais resultados são comparados com o método considerado o estado da arte em relação ao MORP.

Abstract

In industrial environments, different production processes are implemented in various production lines. These lines are divided into stages according to the type of product manufactured. In general, it is necessary that the production processes are optimized according to different purposes, such as, for example, minimizing the use of raw material and energy saving. It is also necessary to optimize the sequence in which the products are processed at each stage of production lines, in order to reduce the use of intermediate stock, to increase the production rate and also the rate of delivery of the manufactured products to customers. For different processes there are several aspects that affect the efficiency of production lines, giving rise to industrial optimization problems. The optimization of these problems most of the time is not trivial, therefore, it requires study and investment. This work describes in details the Minimization of Order Spread Problem, which is focused on reducing the use of intermediate stock in industries. To solve this problem we propose a two phase heuristic method for generating an initial solution and then improving it using an Iterated Local Search, both methods from the literature. The experiments considered a large set of instances from the literature and the results are compared to the current state-of-the-art method. The proposed method was able to generate new best solutions.

Dedico este trabalho ao meu pai Wellington e a minha mãe Rita que me ajudaram em todos os momentos de dificuldade, ao meu irmão Yuri que na maioria das vezes foi o "alívio cômico", demonstrando vital importância para a conclusão do trabalho. Estas três pessoas acreditaram em mim quando nem eu mesmo acreditava.

Agradecimentos

Agradeço a minha namorada que gastou todo meu dinheiro, fazendo com que não sobrasse para as bebidas. Agradeço aos meus amigos que me fizeram gastar com bebidas e não sobrou dinheiro para a namorada. Agradeço ao pessoal do Stilingue que fizeram com que não sobrasse tempo para namorada nem amigos. Terminei sem tempo e sem dinheiro, mas terminei! Agradeço também ao meu orientador Marco que teve muita paciência comigo ao longo do projeto.

Sumário

1 Introdução **1**
1.1 Motivação . . . 2
1.2 Objetivos . . . 3
1.3 Organização do Trabalho . . . 3

2 Revisão da Literatura **4**
2.1 O Trabalho de Madsen (1988) . . . 4
2.2 O Trabalho de Foerster e Wäscher (1998) . . . 5
2.3 O Trabalho de Fink e Voß (1999) . . . 6
2.4 O Trabalho de De Giovanni et al. (2013) . . . 7
2.5 O Trabalho de Kim et al. (2016) . . . 8
2.6 Trabalhos Relacionados . . . 9

3 Fundamentação Teórica **11**
3.1 O Problema de Minimização de Espalhamento de Ordens . . . 11
3.1.1 Pré-Processamento Por Dominância . . . 14
3.1.2 Limitante Inferior . . . 14
3.2 Métodos para Geração da Solução Inicial . . . 15
3.2.1 *Cheapest Insertion* . . . 15
3.2.2 *Best Insertion* . . . 16
3.2.3 Busca em Largura . . . 17
3.3 Métodos de Busca Local e Perturbação . . . 18
3.3.1 *k-swap* . . . 19
3.3.2 *2-opt* . . . 19
3.3.3 Agrupamento de *1-blocks* . . . 20
3.4 Busca Local Iterada . . . 20

4 Desenvolvimento **23**
4.1 Geração da Solução Inicial . . . 23
4.1.1 Heurística em Grafos . . . 23

4.1.2 *Cheapest Insertion* Aplicado ao MORP 27
4.1.3 Uma Heurística em Duas Fases Aplicada ao MORP 28
4.2 ILS Aplicada ao MORP 30

5 Experimentos Computacionais **32**
5.1 Experimentos Preliminares 32
5.1.1 Critério Guloso da BFS 32
5.1.2 Escolha do Método para Geração da Solução Inicial 33
5.1.3 Escolha da Busca Local e Perturbação 33
5.1.4 Aleatoriedade no Método de Agrupamento de *1-blocks* 34
5.1.5 Método de Agrupamento de *1-blocks* Utilizando *Best Insertion* 34
5.1.6 Estratégia de Intensificação 34
5.1.7 Ajuste de Parâmetros 35
5.2 Comparação de Resultados 35
5.2.1 Instâncias e Comparações com Fink e Voß (1999) 36
5.2.2 Comparação com De Giovanni et al. (2013) 37
5.2.3 Instâncias de VLSI Hu e Chen (1990). 39
5.2.4 Instâncias do *First Constraint Modeling Challenge* (Smith e Gent, 2005) 40
5.2.5 Instâncias de Chu e Stuckey (2009) 42
5.2.6 Instâncias SCOOP 42

6 Conclusões **44**

Referências Bibliográficas **45**

Lista de Figuras

3.1 Exemplo da execução do *Cheapest Insertion*. 16
3.2 Exemplo da execução do *Best Insertion*. 17
3.3 Busca em Largura 18
3.4 Exemplo da execução do *k-swap*, em que, k igual a 2. 19
3.5 Exemplo da execução do *2-opt*, momento da escolha das arestas para remoção. . . 20
3.6 Exemplo da execução do método *2-opt* após trocar arestas. 20
3.7 Exemplo de execução da ILS no espaço de busca. 21
3.8 Exemplo da execução da ILS durante e após a perturbação. 22

4.1 Grafo referente a Tabela 4.1 24
4.2 Busca em Largura aplicada ao MORP 25
4.3 Exemplo da execução do *Best Insertion*. 29

Lista de Tabelas

3.1 Instância MORP. 12
3.2 Exemplos de soluções em forma da matriz Q^{π}. 13
3.3 Exemplo de instância (a) antes do pré-processamento e (b) após a aplicação do pré-processamento por dominância. 14
3.4 Limitante inferior. 15

4.1 Representação em grafo. 24
4.2 Construção da fila de acordo com a execução da BFS. 25
4.3 Construção do vetor solução π a partir do vetor ϕ. 27

5.1 Ajuste de parâmetros usando *irace*. 35
5.2 Comparação com o melhor resultado de Fink e Voß (1999). 36
5.3 Comparação com o melhor resultado de De Giovanni et al. (2013). 38
5.4 Resultados encontrados para as instâncias VLSI. 39
5.5 Resultados encontrados para as instâncias do *First Constraint Modeling Challenge*. 41
5.6 Resultados encontrados para as instâncias Chu e Stuckey (2009). 42
5.7 Resultados encontrados para as instâncias SCOOP. 43

Capítulo 1

Introdução

Com os avanços da tecnologia, as indústrias tendem cada vez mais a investir na melhoria da qualidade dos produtos e dos métodos de produção. É de interesse destas indústrias minimizar os gastos relacionados ao consumo de energia, mão de obra, maquinário, manipulação e estoque de produtos ou qualquer outro gasto relacionado com as linhas de produção. Estas variáveis podem ser trabalhadas para otimizar o processo de produção industrial e no intuito de maximizar a lucratividade. Desta forma, surge a mobilização de profissionais de engenharia e ciências exatas para analisar e otimizar todos os aspectos das linhas de produção de uma indústria, desde o recebimento da matéria prima até a entrega do produto acabado ao consumidor final.

Os produtos são classificados como semi-acabados quando faltam poucas etapas do processo produtivo para que eles sejam terminados e enviados aos seus respectivos destinos finais, ou seja, em parte dos casos, os produtos passam por processos de acabamento estético antes de serem enviados ao consumidor final. Os produtos que são classificados como acabados já passaram por todas as etapas do processo produtivo ligadas a sua fabricação e portanto podem ser enviados ao consumidor final.

Uma linha de produção é uma forma de produção em série, onde vários funcionários, com ajuda de maquinário, trabalhando de forma sequencial, fabricam produtos semi-acabados ou acabados. A forma mais característica, a da montagem em série, foi inventada por Henry Ford, empresário estadunidense do setor automobilístico. A produção em massa, popularizada por Ford no início do século XX, se tornou um modo de produção muito difundido, pois permitia altas taxas de produção por trabalhador e ao mesmo tempo disponibilizava produtos a preços baixos.

Neste trabalho, considera-se uma indústria que fabrica vários produtos diferentes e diferentes clientes que possuem diferentes ordens de compra, ou seja, demandas por conjuntos de produtos. É necessário atender estas ordens o mais rápido possível, de maneira a diminuir o tempo de atendimento de todos os pedidos, fazendo com que o pedido seja despachado tão rápido quanto possível, reduzindo o uso do estoque intermediário. O estoque intermediário

consiste no armazenamento de produtos fabricados, porém, ainda não preparados para entrega. A utilização deste tipo de estoque implica em utilização de mão de obra e maquinário adicionais para manipulação dos produtos e área física disponível para tanto. Além disto, os produtos podem ser danificados devido a acidentes ou intempéries.

A fabricação dos produtos é realizada em estágios, sendo que em cada um dos estágios um tipo de produto é fabricado. O intervalo entre o início e o fim da fabricação dos produtos de uma mesma ordem de compra é chamado de espalhamento de ordem, e é medido pelo número de estágios necessários para concluir a fabricação de todos os produtos de uma ordem de compra específica de um cliente.

O Problema de Minimização de Espalhamento de Ordens (Minimization of Order Spread, MORP) é um problema de sequenciamento de produção proposto no contexto de operações de corte na indústria de vidro, no qual a estocagem intermediária das peças é custosa. A operação de corte consiste em, dadas unidades maiores de matéria prima, que podem ser barras, bobinas ou chapas de papel, madeira, vidro, ou metal, cortá-las em unidades menores de acordo com um padrão pré-estabelecido. Este padrão é a forma como as unidades menores estão dispostas nas unidades maiores.

Considerando a produção em larga escala, existem vários padrões diferentes a serem cortados, e diferentes sequências de corte otimizam determinados objetivos das indústrias. Entre estes objetivos cita-se a homogeneidade das características físicas do produto final, o ritmo da produção, o ritmo da entrega dos pedidos de compra, a economia com armazenamento intermediário e o gasto com mão de obra.

Quando o espalhamento das ordens de compra é minimizada, o tempo em que os clientes aguardam pelo atendimento de seus pedidos – e consequentemente a utilização do estoque intermediário – é minimizado. Existe, também, a tendência a minimizar a heterogeneidade das características físicas dos produtos.

Para solução do problema apresentado, este trabalho propõe uma heurística construtiva simples para geração de uma solução inicial e também a aplicação da meta-heurística Busca Local Iterada (*Iterated Local Search* – ILS) em uma fase posterior de aprimoramento. A ILS foi escolhida para este problema pois nunca foi aplicada ao problema e demonstrou bons resultados em outros problemas de minimização.

1.1 Motivação

O Problema de minimização de espalhamento de ordens possui aplicação industrial ampla; portanto, o estudo deste problema pode contribuir para a otimização das linhas de produção de diversas indústrias. O problema tratado neste trabalho tem equivalência com outros problemas na literatura, por exemplo, MOSP, MDP e *Talent Scheduling*, fazendo com que este estudo possa servir de base para estudos de outros problemas relacionados. Este problema é

classificado como NP-Difícil (Garey e Johnson, 2002); portanto, existe também a motivação teórica.

1.2 Objetivos

De uma maneira geral, este trabalho consiste em realizar pesquisa para geração de embasamento teórico para compreensão dos conceitos relacionados ao problema de minimização de espalhamento de ordens e aos problemas de sequenciamento de produção para elaborar uma heurística consistente que alcance bons resultados e se aplique a contextos reais dos processos de produção das industrias nacionais. São objetivos específicos:

1. Realizar pesquisa para geração de embasamento teórico e revisão bibliográfica sobre problema de minimização de espalhamento de ordens (MORP);
2. Elaborar uma heurística consistente que possa ser utilizada no contexto do problema tratado que permita a obtenção rápida de soluções próximas ou melhores que as melhores soluções conhecidas anteriormente sem que se perca a vantagem da busca sistemática – inicialmente considerando problemas específicos, mas com uma possibilidade de generalização;
3. Pesquisar técnicas para melhoria fina de soluções (*polishing*).

Além dos objetivos principais, outros produtos deste projeto de pesquisa serão trabalhos publicados em periódicos e eventos nacionais.

1.3 Organização do Trabalho

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta a revisão da literatura de 1988 até 2017. A base conceitual do MORP é relatada no Capítulo 3. A implementação do método proposto é descrita no Capítulo 4 e os experimentos são descritos no Capítulo 5. No Capítulo 6 é apresentado o plano para conclusão do trabalho de pesquisa.

Capítulo 2

Revisão da Literatura

Neste capítulo os trabalhos publicados anteriormente serão analisados em ordem cronológica, colocando em evidência os avanços nas heurísticas e métodos propostos para o Problema de Minimização de Espalhamento de Ordens. Trabalhos que abordam problemas relacionados ao MORP também são abordados. Os trabalhos revisados foram publicados entre 1988 a 2017.

2.1 O Trabalho de Madsen (1988)

O Problema de Minimização de Espalhamento de Ordens foi proposto por Madsen (1988) ao estudar o processo de corte na indústria de vidro. De acordo com este trabalho, o custo de estocagem intermediária em tal indústria é custosa e de alto risco para a integridade física das peças. As peças de vidro não podem ser estocadas em pilhas altas, devem ser manuseadas com cuidado devido a fragilidade do material e quando estocadas existia a dificuldade em diferenciá-las. No intuito de minimizar a utilização de estocagem e o manuseio das peças de vidro, foi proposta uma estratégia de solução em três estágios, descritos a seguir.

No primeiro estágio, resolve-se o problema de corte de estoque (definição dos padrões de corte) sem levar em consideração o espalhamento de ordens. Para este primeiro estágio foi usado o método proposto por Gilmore e Gomory (1966).

No segundo estágio, constrói-se a matriz de distâncias C baseada no resultado do estágio 1. Esta matriz é construída da seguinte forma: Para produtos contidos em ordens de compras *singulares* (ou seja, uma ordem de compra única para um produto específico), denotadas por $p_s, p'_s, p''_s, \ldots$, o valor da $c_{p_s,j}$ e c_{j,p_s} ($j \neq p_s$ e j variando entre 1 e o número máximo de ordens) é definido como 1.000; o valor das entradas $c_{p'_s,p''_s}$ e $c_{p''_s,p'_s}$ é definido como 110; para as ordens i e j ($i \neq j$) que possuem n produtos em comum o valor de $c_{i,j}$ é definido por $100 - 10 \times n$. Tal valor é definido apenas para instâncias que possuem no máximo 10 produtos, ou seja, o valor varia de acordo com o tamanho da instância, caso contrário a matriz pode conter valores negativos. Os elementos da diagonal principal da matriz C possuem valor 10.000 cada.

No terceiro estágio resolve-se o Problema do Caixeiro Viajante, considerando-se os produtos como vértices e a matriz de distâncias C. Para solução, foi empregado o método *3-opt* proposto por Lin (1965). Este método é uma heurística de melhoramento definida da seguinte maneira: dada uma solução inicial em forma de grafo, eliminam-se três arestas – $(k_1, k_2), (j_1, j_2)$ e (i_1, i_2) – e então são testados todas as combinações de ligações novas entre os vértices que estavam conectados por tais arestas. Se existir alguma nova configuração melhor que a anterior, ou seja, se o comprimento total da rota diminuir, mantêm-se a nova rota. Caso contrário, escolha novamente outras três arestas para análise.

A sequência dos vértices na rota estabelecida como solução do Problema do Caixeiro Viajante determina a ordem de fabricação dos produtos, visando minimizar o espalhamento de ordens. O algoritmo atribui um valor alto (10.000) para os elementos da diagonal principal da matriz C, para garantir que cada produto será fabricado apenas uma vez. Os elementos da diagonal principal da matriz representam as arestas que ligam um vértice a ele mesmo. O valor 110 é atribuído as arestas que representam conexões entre produtos (vértices) contidos apenas em ordens de compra singulares para que tais ordens sejam atendidas por último, pois não possuem espalhamento. Os valores atribuídos às demais arestas $(100 - 10 \times n)$ representam quantas ordens (n) os dois produtos (vértices), que estão conectados, possuem em comum. Quanto maior este número, menor é o peso da aresta e mais probabilidade existe dos dois produtos em questão serem alocados em sequência.

O resultado apresentado pelo autor mostrou a diminuição em média de 18% do espalhamento de ordens e de 30% na minimização de descontinuidades em relação aos resultados obtidos com a execução do estágio 1. Uma *descontinuidade* ocorre quando a produção de um produto é iniciada, interrompida e então retomada. O problema de minimização de descontinuidades (*Minimization of Descontinuities* – MDP) tem como objetivo minimizar o número de descontinuidades sem considerar o tamanho da mesma, visando maximizar a homogeneidade das características físicas dos produtos. A aplicação proposta por Madsen (1988) resolveu o problema de minimização de descontinuidades com mais eficiência.

2.2 O Trabalho de Foerster e Wäscher (1998)

Em Foerster e Wäscher (1998), foi proposto o uso da meta-heurística Recozimento Simulado (*Simulated Annealing* – SA) para solução do MORP. O método SA é uma meta-heurística que consistem em uma técnica de busca local probabilística, fundamentada numa analogia com a termodinâmica. O algoritmo deste método substitui a solução atual por uma segunda (situada na vizinhança da primeira), selecionada de acordo com a função objetivo e a variável T (dita Temperatura, por analogia). Quanto maior a temperatura T, maiores as chances da solução selecionada substituir a anterior. A medida que o algoritmo progride, a variável T é decrementada, fazendo com que a solução convirja para um ótimo local.

Os autores implementaram o procedimento de Madsen (1988), o procedimento *3-opt* e o SA utilizando as mesmas instâncias usadas por Madsen (1988). De acordo com a implementação dos autores, o *3-opt* superou o procedimento de Madsen (1988) e o SA superou o *3-opt*. O SA apresentou redução considerável na média dos espalhamentos das ordens, porém, o procedimento de Madsen (1988) foi melhor em relação ao tempo de execução.

2.3 O Trabalho de Fink e Voß (1999)

No ano seguinte Fink e Voß (1999), empregaram diferentes heurísticas para solução do Problema de Minimização de Pilhas Abertas (ou MOSP, um problema correlato) e do MORP. O MOSP consiste em otimizar o uso de estoque primário (ainda no ambiente de produção) visando obedecer restrições que dizem respeito a capacidade de estoque, utilização de mão de obra e preservação da integridade física das peças.

Neste trabalho, foram geradas diferentes soluções iniciais pelos métodos de Inserção Mais Barata (*Cheapest Insertion* – CI) e Inserção Mais Barata do Pior Padrão (*Cheapest Insertion of the Worst Pattern* – CIW). Para a otimização destas soluções iniciais foram utilizados os métodos *2-opt*, uma nova implementação de SA e variações da Busca Tabu.

A ideia básica da Busca Tabu é manter um histórico da busca realizada por um algoritmo específico de busca local e utilizar estas informações para evitar a exploração redundante do espaço de busca, visando também escapar de ótimos locais. Neste trabalho foram implementas as seguintes técnicas de Busca Tabu avançadas, utilizando como solução inicial o CIW:

TSS: Busca Tabu Estática (*Static Tabu Search*). Esta técnica tem como princípio básico proibir a inversão de alterações realizadas pelo algoritmo de busca local por um número fixo de iterações (duração tabu);

TSTA: Busca Tabu Estrita por Trajetória Aproximada (*Strict Tabu Search by Approximate Trajectory*. Esta técnica não permite que trajetórias semelhantes às já realizadas no espaço de busca pelo algoritmo e buscas locais sejam realizadas;

TSTAE: Busca Tabu Estrita por Trajetória Aproximada com Cinco Movimentos Aleatórios de Fuga (*Strict Tabu Search by Approximate Trajectory Including Five Random Escape Moves*). Semelhante a TSTA, entretanto, inclui cinco movimentos aleatórios de fuga, não especificados;

TSTRE: Busca Tabu Reativa (*Reactive Tabu Search Including Five Random Escape Moves*). Esta técnica é uma modificação da TSS para que o critério de duração tabu seja adaptada dinamicamente ao longo da busca, incluindo cinco movimentos aleatórios de fuga, não especificados;

TSTRE5000: Semelhante ao anterior, porém, aplicado a cada 5000 iterações.

Os métodos implementados foram comparados com métodos *3-opt* e a implementação de SA proposta por Foerster e Wäscher (1998). Os melhores resultados foram obtidos pela execução do TSRE5000, melhorando a qualidade de solução inicial em até $49,5\%$. O SA proposto por Foerster e Wäscher (1998) atingiu resultados entre $4,9\%$ e $15,9\%$ distantes dos melhores valores encontrados. As instâncias utilizadas foram geradas aleatoriamente e fornecidas para testes neste trabalho.

2.4 O Trabalho de De Giovanni et al. (2013)

De Giovanni et al. (2013) propuseram dois algoritmos para resolver o Problema de Minimização de Custo de Conexões entre Portas em Circuitos (*Gate Matrix Connection Cost Minimization Problem - GMCCP*). O objetivo dos algoritmos é minimizar o comprimento dos fios na produção de circuitos no intuito de minimizar a área deles.

Os circuitos são definidos por portas conectadas de formas distintas. Cada conexão, denominada rede, envolve um subconjunto de portas. As portas são conectadas por fios e as conexões entre duas portas podem cruzar portas que não estão contidas na rede, por exemplo, seja a sequencia de portas A, B e C, para conectar a rede formada pelas portas A e C pode ser necessário que o fio passe pela porta B, sendo esta não pertencente à rede. Considerando as portas como sendo colunas e as redes sendo as linhas de uma matriz M, é possível que duas redes ocupem a mesma linha da matriz, desde que não ocupem as mesmas colunas. Por exemplo, dado um conjunto de portas de A até D, as redes formadas pela conexão estre as portas A e B e as portas C e D podem ocupar a mesma linha, desde que não haja sobreposição, portanto, as linhas da matriz M são denominadas trilhas, que podem conter uma ou mais redes. O GMCCP consiste em encontrar uma permutação de portas, em que, o comprimento do fio para a conexão seja minimizado.

A minimização do número de trilhas no circuito é correlato ao MOSP, em que, cada trilha representa uma pilha aberta. A minimização do comprimento do fio utilizado para conectar as portas é o MORP, em que, as portas representam produtos e as trilhas que contém redes representam as ordens de compras. O comprimento do fio é calculado sendo a distância, em colunas, entre a primeira e a última porta de cada rede.

Para resolver o problema os autores propuseram dois algoritmos. O primeiro algoritmo é um Algoritmo Genético (AG) e é usado para diminuir o número de trilhas no circuito, ou seja, resolver o MOSP, e gerar uma solução factível para a minimização de custo do fio, ou seja, para o MORP. O segundo algoritmo é proposto uma nova formulação de programação inteira baseada na propriedade dos uns consecutivos. Tal formulação é resolvida por um algoritmo *branch and cut - B&C*.

Os algoritmos foram avaliados em 330 instâncias originais de distintos conjuntos. O primeiro conjunto de 11 instâncias foi proposto por Hu and Chen (1990), da indústria VLSI.

Segundo conjunto composto por 24 instâncias originais de duas empresas de corte de madeira (SCOOP Team, 2009). Estes dois conjuntos são instâncias reais com tamanhos distintos variando linhas e colunas de 10 a 202 e 10 a 141, respectivamente. O restante das instancias foi retirada da literatura referente ao MOSP: 250 instâncias de Faggioli and Bentivoglio (1998), com linhas e colunas variando de 9 a 50 e 10 a 30, respectivamente, e dois conjuntos de instâncias de Smith and Gent (2005), o primeiro nomeado como 25 Shaw e o segundo Wilson. As instâncias pertencentes ao conjunto 25 Shaw possui linhas e colunas igual a 20 e o conjunto Wilson possuem linhas variando de 10 a 100 e colunas de 20 a 100.

Os autores compararam o AG proposto com o algoritmo proposto por Oliveira and Lorena (2002), o AG proposto obteve melhores resultados para algumas das instâncias comparadas. Para as instâncias restantes, não existiam resultados para o MORP na literatura e os resultados obtidos para o MOSP são comparados com o ótimo ou como a fronteira encontrada pelo *B&C*. O AG proposto alcançou os melhores resultados da literatura e o *B&C* encontrou soluções ótimas ou sub-ótimas para as instâncias analisadas.

2.5 O Trabalho de Kim et al. (2016)

No ano de 2016 Kim et al. (2016) propuseram uma formulação matemática e uma heurística para resolver o MORP, o problema de Desequilíbrio de Carga em Máquina (*Machine Load Imbalance*) e o problema de Desperdício de Matéria Prima (*Trim Loss*) na produção de molduras para janelas. Na indústria descrita no trabalho, os produtos são definidos por peças de 4 tipos que, quando montadas, formam uma moldura de janela e as ordens de compra são compostas por tais molduras de janelas.

A fabricação das molduras na indústria citada no trabalho é dividida em 3 estágios: corte, processamento e montagem. O estágio de corte é definido da seguinte forma: a indústria possui vários perfis de matéria prima que podem ser cortados em 4 peças diferentes [superior (A), inferior (B), esquerda (E) e direita (D)] formando molduras de janela de tamanhos definidos pelas ordens dos clientes. A indústria gasta 1 estágio de tempo para cortar 1 perfil. Neste estágio é necessário definir a sequencia em que as peças [A, B, D e E] serão cortadas, minimizando o Desperdício de Matéria Prima. A sequencia definida no primeiro estágio afetará os próximos estágios da produção. No estágio de processamento, as peças D e E são processadas na mesma máquina, pois possuem mesmo tamanho, enquanto as peças A e B são processadas em outra máquina. Neste estágio a ocorrência de peças do tipo A e B ou D e E consecutivas causará desequilíbrio de carga em máquina. No último estágio é necessário obter as 4 peças de uma moldura para montá-lo, portanto, se houver peças de uma mesma moldura cortadas de perfis diferentes ocorrerá *espalhamento de ordem*. Exemplo: se uma moldura pertencente a ordem de compra i possui os lados A e B cortados no estágio de tempo 1 e os lados D e E cortados no estágio de tempo 5, considerando que é necessário 1 estágio de tempo para cortar

1 perfil, o tamanho do *espalhamento de ordem* para a ordem i é 4, ou seja, os lados A e B ficarão no estoque intermediário 4 estágios de tempo até que a moldura ao qual pertencem possa ser montado.

A heurística proposta é baseada no Problema da Mochila (*Knapsack Problem*). O Problema da Mochila consiste em preencher uma mochila com objetos de diferentes pesos e valores de forma a maximizar o valor transportado na mochila respeitando a capacidade de transporte da mesma. O algoritmo proposto seleciona perfis interativamente atribuindo peças A, B, D e E ao perfil selecionado respeitando um critério que minimiza o espalhamento da ordem. Posteriormente, o algoritmo resolve o Problema da Mochila considerando objetos como um conjunto de peças C contidas em ordens de compra que não foram atribuídas ao sequenciamento e a mochila como o restante do perfil, minimizando o desperdício de matéria prima e o espalhamento de ordem. O conjunto C é definido por 4 peças, uma de cada tipo, então, outro Problema da Mochila é resolvido considerando a mochila sendo o conjunto C e os objetos sendo uma peça pertencente a qualquer ordem de compra não atribuída. Alguns procedimentos de melhoria são aplicados e o processo descrito é repetido variando atributos e o melhor resultado é retornado.

Os autores realizaram testes com 20 instâncias reais da indústria de produção de molduras de janelas. O algoritmo obteve resultados eficientes e está sendo usado pela empresa de onde originou-se o estudo.

2.6 Trabalhos Relacionados

O Problema de Escalonamento de Talentos (*Talent Scheduling Problem*) pode ser visto como uma variação do MORP (Cheng et al., 1993). Este problema tem como objetivo minimizar o tempo que atores, equipamentos ou técnicos envolvidas nas produções de filmes permanecem no local da filmagem sem estar trabalhando. Considerando os talentos (atores, equipamentos ou técnicos) sendo ordens de compra e os dias de filmagem sendo produtos, monta-se a matriz M definindo o valor da célula $m_{ij} = 1$ quando o talento i é requisitado para filmar no dia j e $m_{ij} = 0$ caso contrário. Desta forma, para reduzir o tempo ocioso dos talentos aplica-se técnicas para resolver o MORP levando em consideração a matriz M. Este problema se diferencia do MORP devido ao fato de considerar custos de atores, técnicos e equipamentos, ou seja, o espalhamento das ordens referentes aos talendos mais caros tende a ser minimizado com prioridade. Cheng et al. (1993) propôs a utilização do *branch-and-bound* e uma heurística que utiliza o *branch-and-bound* para resolver tal problema. A heurística usada é baseada na escolha da ramificação mais atrativa, ou seja, escolhem sempre o nó de menor valor. Posteriormente,o método *2-opt* é aplicado para trocar o conteúdo dos dias de filmagens, no intuito de aprimorar a solução. A heurística se mostrou mais eficiente em questão ao tempo, ambos chegando a mesma solução. Com base em uma instância real, o custo

da filmagem relativo aos dias de espera dos atores, equipe e equipamentos de um filme foi reduzido de US$36.400 para US$17.900.

Há também o Problema de Escalonamento de Ensaio (*Scheduling a Rehearsal*), que possui como objetivo minimizar o tempo em que os músicos de uma orquestra permanecem no local do ensaio sem que estejam ensaiando. Durante um ensaio, diferentes movimentos são executados, porém, nem todos os músicos tocam em todos os movimentos, gerando ociosidade de alguns músicos de acordo com a estrutura da peça ensaiada. De forma análoga ao *Talent Scheduling Problem*, este problema se assemelha ao MORP e pode ser definido por ordens de compra sendo músicos e produtos sendo cada movimento do ensaio. A característica principal deste problema é levar em consideração o tempo de ensaio, ou seja, os espalhamentos de ordens podem variar de acordo com o tamanho de cada movimento do ensaio. Smith (2003) propôs uma modelagem usando programação por restrições. O autor apresentou apenas uma tabela de instância, tal tabela possui 5 músicos e 9 estágios do ensaio. O autor encontrou o resultado 17 unidades de tempo que, segundo ele, é o resultado ótimo e apresentou a sequencia de escalonamento.

O MORP, MOSP e MDP (problemas mencionados anteriormente neste trabalho) são problemas correlatos. Linhares e Yanasse (2002) estabeleceram relação entre estes afirmando que não são equivalente, ou seja, uma solução ótima para um não é solução ótima para os demais.

O MORP é também um caso especial do Problema de Minimização de Largura de Banda em Matrizes, segundo (Yanasse, 1997). Este problema consiste em permutar as linhas e as colunas de matrizes binárias para minimizar a distância entre os elementos não nulos e a diagonal principal. De acordo com Garey e Johnson (2002) o MORP é NP-Difícil por redução ao Partição de Triângulos.

Capítulo 3

Fundamentação Teórica

Este capítulo tem por objetivo descrever em detalhes o problema abordado e suas propriedades, bem como apresentar em linhas gerais os princípios dos algoritmos de busca local presentes literatura que foram considerados como alternativas durante o desenvolvimento do método proposto neste trabalho, bem como a meta-heurística ILS. Detalhes específicos da aplicação destes métodos ao MORP são apresentados no Capítulo 5.

3.1 O Problema de Minimização de Espalhamento de Ordens

O Problema de Minimização de Espalhamento de Ordens, proposto por Madsen (1988), é relacionado ao contexto das indústrias em que uma parte da linha de produção confecciona um conjunto J de produtos distintos para atender a demanda de um conjunto I de ordens de compra, formadas por tais produtos. A produção é dividida em estágios em que em cada estágio é fabricado um produto diferente.

Na linha de produção todos os produtos já fabricados são armazenados temporariamente no estoque intermediário. Este estoque intermediário é utilizado para armazenar os produtos de uma ordem de compra de um cliente. Uma vez que todos os produtos que compõem uma ordem de compra específica tenham sido fabricados, estes podem ser removidos do estoque intermediário.

O número de estágios necessários para a produção de todos os produtos de uma ordem, desconsiderando o primeiro estágio, é denominado espalhamento de ordem. O primeiro estágio é desconsiderado, porque, ordens de compra que possuem apenas um produto, são despachadas imediatamente sem que tal produto fique armazenado no estoque intermediário, ou seja, o tamanho do espalhamento de ordem quando a mesma possui apenas 1 produto é 0. Quando os produtos de uma mesma ordem não são fabricados consecutivamente, ou seja, existe um intervalo de estágios em que nenhum produto desta ordem é fabricado, caracteriza-se a *descontinuidade*. As descontinuidades influenciam diretamente no espalhamento de ordens. Uma

ordem composta por n produtos em que existem m estágios de descontinuidade em sua produção, tem como espalhamento $(n-1)+m$ estágios.

Uma instância para o MORP pode ser representada como uma matriz binária P que relaciona ordens de compra e produtos. Quando o produto $j \in J$ está presente na ordem de compra $i \in I$, o valor da matriz P na posição p_{ij} é 1, caso contrário, o valor é 0. A Tabela 3.1 é um exemplo de instância para o MORP. As linhas, numeradas de i_1 a i_6, representam as ordens de compras dos clientes. As colunas, numeradas de j_1 a j_6, representam os produtos.

Tabela 3.1: Instância MORP.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	1	1	0	0	0	0
i_2	1	0	1	0	0	0
i_3	0	0	0	1	1	0
i_4	0	0	0	1	0	1
i_5	0	1	0	0	1	0
i_6	0	0	1	0	0	1

Uma solução para o MORP é uma permutação π das $|J|$ colunas da matriz P gerando uma matriz permutação Q^π. É de suma importância para o problema identificar os tamanhos dos espalhamentos das ordens, medidos em número de estágios. Para identificar o comprimento dos espalhamentos de ordens, a matriz Q^π é definida com a propriedade de 1*s consecutivos*. Todo elemento da matriz P, de valor 0, compreendido entre dois elementos de valor 1 em uma mesma linha é preenchido com o valor 1 na matriz Q^π. Esta propriedade indica, na matriz Q^π, por quais estágios se estende a produção que atende cada ordem de compra. A matriz $Q^\pi = \{q_{ij}\}$ é definida por:

$$q_{ij} = \begin{cases} 1 & \text{se } \exists x, \exists y \mid \pi[x] \leq j \leq \pi[y] \text{ e } p_{ix} = p_{iy} = 1 \\ 0, & \text{caso contrário} \end{cases} \tag{3.1}$$

Na Equação (3.1), π denota a permutação dos $|J|$ produtos definindo a sequência em que cada produto será fabricado, ou seja, $\pi[j]$ é o estágio da produção em que o produto j será fabricado.

As Tabelas 3.2 representam exemplos de soluções para o MORP representadas da forma da matriz Q^π. As colunas representam o instante π em que produtos serão fabricados, as linhas representam ordens de compra. Os elementos da matriz de valor 1 em negrito representam as descontinuidades.

A matriz Q^π é definida para auxiliar a Função (3.2) a encontrar qual o maior espalhamento de ordens dentre todas as $|I|$ ordens. A linha da matriz Q^π que possuir o maior número de 1s consecutivos é a ordem com maior espalhamento. O comprimento do espalhamento de uma ordem i é definido pela soma dos valores da linha i da matriz Q^π subtraído de um.

Tabela 3.2: Exemplos de soluções em forma da matriz Q^{π}.

	j_5	j_2	j_4	j_6	j_3	j_1
i_1	0	1	**1**	**1**	**1**	1
i_2	0	0	0	0	1	1
i_3	1	**1**	1	0	0	0
i_4	0	0	1	1	0	0
i_5	1	1	0	0	0	0
i_6	0	0	0	1	1	0

(a)

	j_1	j_6	j_5	j_4	j_3	j_2
i_1	1	**1**	**1**	**1**	**1**	1
i_2	1	**1**	**1**	**1**	1	0
i_3	0	0	1	1	0	0
i_4	0	1	**1**	1	0	0
i_5	0	0	1	**1**	**1**	1
i_6	0	1	**1**	**1**	1	0

(b)

$$Z_{MORP}(Q^{\pi}) = \max_{i \in I} \left(\sum_{j=1}^{|J|} q_{ij} - 1 \right) \tag{3.2}$$

No sequenciamento da Tabela 3.2, letra a, o produto j_5 é o primeiro a ser fabricado, pois j_5 é a primeira coluna da tabela, seguida por j_2, j_4, j_6, j_3, j_1. A ordem de compra i_5 é a primeira a ser despachada para o próximo estágio da produção (ou consumidor final) pois é composta pelos produtos j_5 e j_2, sendo j_2 o último da ordem a ser produzido, porém, produzido no segundo estágio. A ordem i_1 possui dois produtos, j_2 e j_1, sendo o produto j_2 produzido no estágio 2 e o produto j_1 produzido no estágio 6, sendo assim, o espalhamento desta ordem tem tamanho 4. Para a ordem em questão (i_1), o produto j_2 foi armazenado no estoque intermediário até o estágio 6, em que o último produto da ordem é fabricado e então a ordem completa pode ser despachada. A Tabela 3.2(a) apresenta uma melhor solução em relação a solução apresentada na Tabela 3.2(b), pois o maior espalhamento contido na primeira solução é menor que o maior espalhamento contido na segunda solução.

A função objetivo do MORP visa a minimizar o maior espalhamento das ordens, ou seja, minimizar a maior soma em qualquer linha da matriz Q^{π}. Com base na análise da Função (3.2), é necessário definir uma permutação π que a minimize. A função objetivo é definida de acordo com a Função (3.3), em que Π representa o número máximo de permutações possíveis da matriz Q^{π}. Em outras versões, a função objetivo do MORP visa a minimizar o espalhamento médio das ordens de compra.

$$\min_{\pi \in \Pi} \; Z_{MORP}(Q^{\pi}) \tag{3.3}$$

Como mencionado anteriormente, para o MORP, o número de descontinuidades das soluções não é levado em consideração; portanto, uma solução que possui várias descontinuidades menores é melhor que uma solução que possui uma descontinuidade maior. Em suma, o Problema de Minimização de Descontinuidades trata do número de descontinuidades, ao contrário do MORP, que trata da duração das mesmas.

3.1.1 Pré-Processamento Por Dominância

O pré-processamento por dominância é feito para reduzir o tamanho das instâncias removendo os dados redundantes. Os dados são considerados redundantes quando eles podem ser ignorados ao resolver o problema sem que sua estrutura seja afetada. O pré-processamento por dominância aplicado ao MORP remove os produtos que estão presentes em todas as ordens de compra em que um segundo produto também está presente.

Dados dois produtos p_i e p_j, seja $o(p_i)$ uma função que retorna todas as ordens de compra pelo produto p_i. Caso o produto p_j conste exatamente em um subconjunto das ordens de compra em que p_i consta (ou seja $o(p_j) \subseteq o(p_i)$), dizemos que $o(p_j)$ é *dominado* por p_i. Pode-se, sem perda de generalidade, considerar p_i e p_j como sendo um único produto, sequenciando-os consecutivamente na solução.

Antes da aplicação de um método para a solução do MORP, para diminuir o tamanho das instâncias, os produtos dominados podem ser removidos das mesmas. Depois de obtida uma solução para a instância reduzida, os produtos dominados são inseridos imediatamente após seus respectivos produtos dominantes em π, mantendo a otimalidade da solução. A Tabela 3.3 apresenta um exemplo de instância em que é possível reduzir seu tamanho através do processamento por dominância.

Tabela 3.3: Exemplo de instância (a) antes do pré-processamento e (b) após a aplicação do pré-processamento por dominância.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	1	0	0	1	1	0
i_2	1	1	1	0	0	0
i_3	0	0	1	1	0	0
i_4	1	1	1	0	1	0
i_5	0	**1**	0	0	1	**1**
i_6	0	**1**	0	0	0	**1**

(a)

	j_1	j_2	j_3	j_4	j_5
i_1	1	0	0	1	1
i_2	1	1	1	0	0
i_3	0	0	1	1	0
i_4	1	1	1	0	1
i_5	0	1	0	0	1
i_6	0	1	0	0	0

(b)

Analisando a Tabela 3.3 (a) verifica-se que o produto 6 está presente nas ordens 5 e 6, e o produto 2 está contido nas ordens 2, 4, 5, 6, ou seja, o conjunto de ordens de compra que contém o produto 6 é subconjunto das ordens de compra que contêm o produto 2. O produto 6 é dominado por 2, portanto, o produto 6 é removido sem alterar a solução do problema, mostrado na Tabela 3.3 (b).

3.1.2 Limitante Inferior

Um limite inferior para o valor de solução para problemas de minimização, como o MORP, é um valor igual ou menor ao valor da solução ótima. Em métodos de solução compostos por múltiplas iterações, o limite inferior pode ser usado como critério de parada além de critérios

já existentes: uma vez atingida uma solução com valor igual ao limite inferior, não existe forma de melhorá-la, logo, a execução pode ser interrompida. O valor do limite inferior, em alguns casos, pode não ser viável, ou seja, o método para resolução do problema não achará tal resposta pois esta pode quebrar restrições do problema.

O limite inferior pode auxiliar o método de solução de forma que iterações que não podem melhorar a solução não são executadas, diminuindo assim o tempo de execução.

Um limite inferior para MORP é o número de produtos que compõem a maior ordem de compra da instância. Não existe forma de obter espalhamento de ordens menor que este limite inferior sem modificar a instância. A solução da Tabela 3.4 possui limite inferior igual a 4.

Tabela 3.4: Limitante inferior.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	1	1	0	1	1	1
i_2	1	0	1	0	0	0
i_3	0	0	0	1	1	0
i_4	0	0	0	1	0	1
i_5	0	1	0	0	1	0
i_6	0	0	1	0	0	1

A Tabela 3.4 apresenta um exemplo de instância MORP em que o limitante inferior para o espalhamento máximo de uma ordem é dado pela ordem i_1. Esta é a ordem que possui mais produtos, ou seja, é a linha da matriz que possui maior número de uns. Não existe permutação π das colunas da referida instância que possua espalhamento máximo menor que o número de produtos da ordem i_1.

3.2 Métodos para Geração da Solução Inicial

Foram considerados três diferentes métodos para geração de uma solução inicial. Os dois primeiros (*Cheapest Insertion* e *Best Insertion*) são métodos construtivos clássicos da literatura. O terceiro método, entretanto, é um algoritmo clássico de busca em grafos utilizado para dar origem a um método heurístico inédito aplicado ao MORP.

3.2.1 *Cheapest Insertion*

A heurística de Inserção mais Barata (*Cheapest Insertion - CI*) consiste em construir uma solução completa inserindo elementos menos custosos a cada posição considerando a função objetivo. O método funciona da seguinte forma: um elemento i_0 é escolhido para o início da construção, então, para inserir o elemento na segunda posição analisa-se todos os elementos que não estão na solução e escolhe o que melhor atende a função objetivo. Os demais elementos são inseridos de forma análoga ao segundo até que todos elementos estejam inseridos na solução, ou seja, a solução esteja completa.

A Figura 3.1 exemplifica a execução do método *Cheapest Insertion* para a construção de uma solução inicial composta por 4 elementos, $[0, 1, 2, 3]$. O algoritmo começa com o elemento 0, sublinhado no início da execução, então, o método verifica iterativamente dentre todos os elementos restantes qual o melhor elemento para a ser inserido na segunda posição da solução. Para cada tentativa de inserção o valor da função objetivo é calculado e a inserção que obtiver melhor valor é mantida. Neste exemplo, o melhor valor de função objetivo para a segunda posição da solução foi obtido adicionando o elemento 2. Na segunda iteração restaram apenas dois elementos a serem inseridos, os elementos 1 e 3 , portanto o algoritmo experimenta ambos na terceira posição e mantém o elemento 3, pois foi o que obteve melhor valor da função objetivo. Por último, na terceira iteração, o elemento 1 é adicionado na quarta posição terminando a construção da solução inicial executando o método *Cheapest Insertion*.

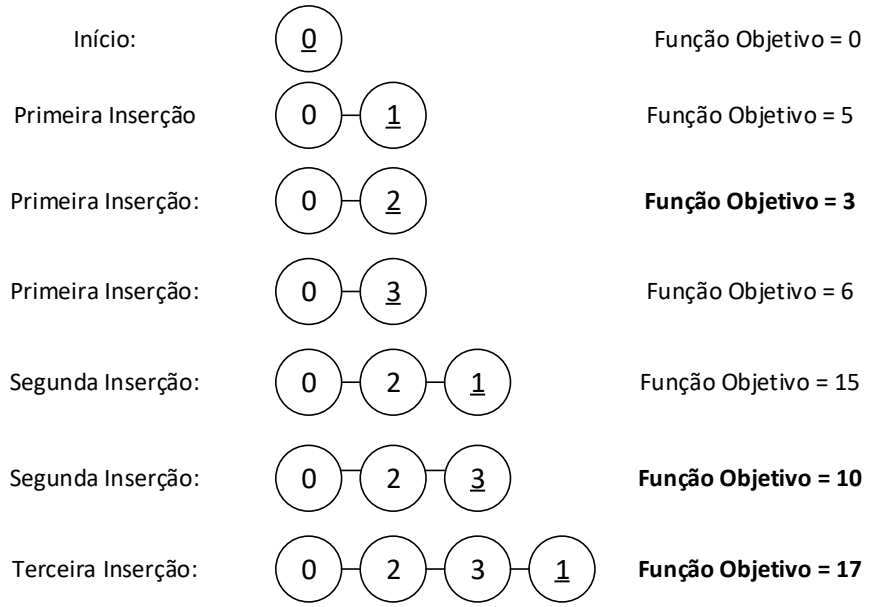


Figura 3.1: Exemplo da execução do *Cheapest Insertion*.

3.2.2 *Best Insertion*

A heurística de *Best Insertion* seleciona um elemento da solução e o re-insere na melhor posição possível de acordo com a função objetivo. A seleção do elemento pode ser feita de forma aleatória ou de acordo com critérios que dependem das características do problema. Este método seleciona os elementos da solução iterativamente, verifica em qual posição o elemento selecionado estaria melhor inserido na solução e assim por diante até que todas as possibilidades de inserção acabem e todos os elementos sejam selecionados.

A Figura 3.2 exemplifica a execução do método *Best Insertion*. O elemento 2, pertencente a solução, sublinhado nesta figura, foi selecionado aleatoriamente, então, o método verifica iterativamente em qual posição este elemento otimiza a função objetivo. Para cada iteração, o valor da função objetivo é calculado e o movimento que obtiver o melhor valor da função objetivo é salvo. Neste exemplo, a segunda iteração obteve o melhor valor da função objetivo, o elemento 2 é movimentado para o a posição 3.

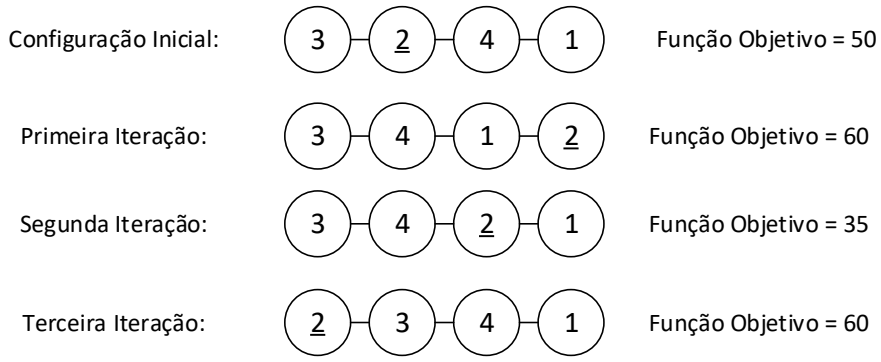


Figura 3.2: Exemplo da execução do *Best Insertion*.

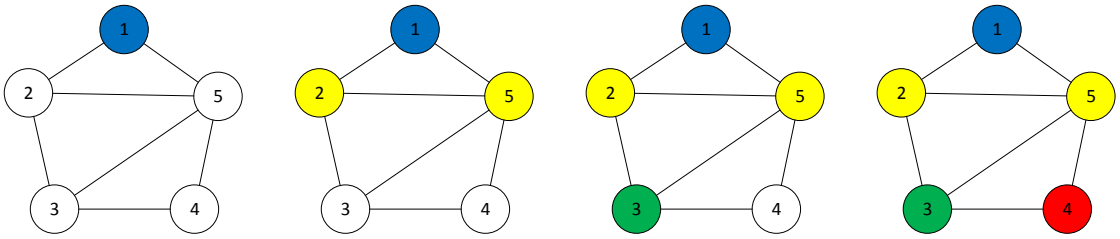
3.2.3 Busca em Largura

A Busca em Largura (ou *Breadth-First Search, BFS*) é um método de busca que explora sistematicamente todos os vértices de um grafo direcionado ou não-direcionado. A BFS é realizada da seguinte forma: dado um vértice inicial, o algoritmo explora todos seus vértices vizinhos. Então, para cada um desses vértices, explora-se todos seus vértices vizinhos que ainda não tenham sido explorados e assim por diante, até que o alvo da busca seja encontrado ou todos os vértices do grafo tenham sido explorados.

O algoritmo garante que nenhum vértice ou aresta seja visitado mais de uma vez. Para isso, utiliza-se uma estrutura de dados fila para garantir a ordem de exploração dos vértices: cada vértice recém-explorado é adicionado ao final da fila, e o próximo vértice a ter sua vizinhança atualizada é o primeiro da fila. A ordem de exploração dos vértices pode ser definida por diferentes critérios para que uma determinada característica seja priorizada, por exemplo, ordem lexicográfica ou peso das aresta de um vértice para os vizinhos.

A Figura 3.3 exemplifica a execução da BFS em um grafo genérico. A fila é inicialmente vazia, então o vértice 1 é adicionado à fila e marcado como explorado. No segundo momento, os vizinhos do vértice 1 (vértices 2 e 5), que não foram explorados, são adicionados a fila em ordem lexicográfica e marcados como explorados. No terceiro momento, os vizinhos do vértice 2 que não foram explorados (vértice 3) são adicionados na fila em ordem lexicográfica. O

vértice 2 possui como vizinhos os vértices 1, 3 e 5, sendo apenas o 3 não explorado, portanto, os vértices 1 e 3 não são adicionados à fila novamente. No quarto momento, os vizinhos do vértice 5 (vértice 4) que não foram explorados são adicionados a fila em ordem lexicográfica. Todos os vértices do grafos estão marcados como explorados, portanto, a busca termina e a fila representa a ordem de exploração do grafo.



(a) (b) (c) (d)

Figura 3.3: Busca em Largura

3.3 Métodos de Busca Local e Perturbação

A *busca local* é definida pela execução de um determinado algoritmo aplicado a um *espaço de busca*, que contém conjunto de todas soluções possíveis. A *vizinhança* é um espaço de busca reduzido que contém todas as soluções que são parecidas entre si. As vizinhanças são induzidas por *movimentos*, ou seja, pequenas modificações nas soluções que as tornam semelhantes estre si. Os algoritmos de busca local iniciam em uma solução candidata e então procuram por soluções na vizinhança aplicando movimentos até que encontre uma solução considerada ótima ou algum *critério de parada* seja obedecido. Ao executar um algoritmo de busca local, o mesmo pode executar durante muito tempo sem obter melhorias significantes para o problema, portanto um critério de parada é definido para economizar tempo. Determinar tempo máximo de execução ou tempo máximo de execução sem melhora na solução são exemplos de critério de parada. O algoritmo *2-opt* é um exemplo de algoritmo de busca local usado em Fink e Voß (1999) aplicado ao MORP.

Quando o algoritmo de busca encontra uma solução ótima, esta pode ser *ótima local* ou *ótima global*. Caso a solução for ótimo global, melhor solução em todas as vizinhanças, o problema está resolvido, caso a solução for ótima local significa que o algoritmo encontrou a melhor solução presente na vizinhança, porém, esta pode não ser a melhor solução para o problema, portanto, precisa procurar por melhores soluções em outras vizinhanças. Para que o algoritmo mude de vizinhança, a operação chamada *perturbação* é aplicada à solução

fazendo com que o algoritmo de busca seja executado em vizinhança distinta. A operação de perturbação é feita da seguinte forma: dada uma solução, modifica-se sua estrutura para que a mesma se pareça menos com sua vizinhança atual e mantenha alguns aspectos de otimalidade, ou seja, parte da solução é modificada resultando em uma nova solução que pertence a uma nova vizinhança e sirva de solução candidata para o algoritmo de busca local.

3.3.1 *k-swap*

O *movimento de troca* (*k-swap*), consiste em, dada uma solução, trocar k elementos de posição. Este movimento pode ser usado para perturbação de solução e para métodos de busca local. O movimento de *k-swap* para busca local consiste em trocar k elementos de uma solução; em caso de melhoria da solução, mantém-se a troca, caso contrário a troca é revertida. O movimento de *k-swap* para a perturbação consiste em trocar k elementos de lugar e manter a troca independentemente do resultado final da solução.

A Figura 3.4 exemplifica a execução de um movimento *k-swap*, em que k é igual a 2. A solução contém os elementos de 1 a 5 na ordem, 3, 2, 4, 1 e 5. Os elementos 2 e 5, sublinhados nesta figura, foram escolhidos aleatoriamente, portanto, são trocados de posição, resultando na solução 3, 5, 4, 1, 2.

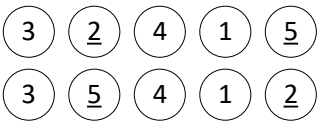


Figura 3.4: Exemplo da execução do *k-swap*, em que, k igual a 2.

3.3.2 *2-opt*

O método *2-opt* foi proposto por Croes (1958) para resolver o Problema do Caixeiro Viajante. O método é definido da seguinte forma: seja uma solução composta por arestas que ligam os vértices k_1, k_2, k_3 e k_4. É necessário eliminar duas arestas da solução e inserir novamente duas arestas, não pertencentes à solução, de forma cruzada, ou seja, se as arestas removidas foram os pares que ligam os vértices (k_1, k_2) e (k_3, k_4), as arestas inseridas devem ligar os vértices na forma (k_1, k_4) e (k_3, k_2). Se esta nova solução for melhor que a anterior, mantém-se esta configuração. Caso contrário, escolhe-se novamente duas arestas para análise. Esta operação é executada para todas as combinações possíveis.

As Figuras 3.5 e 3.6 exemplificam o método de busca *2-opt* em uma solução $[1, 2, 3, 4, 5, 6, 7]$. A aresta que liga os vértices 2 e 3 e a aresta que liga os vértices 6 e 7 foram selecionadas para remoção e então duas arestas que ligam de forma cruzada os vértices

citados acima são escolhida para compor a nova configuração da solução $[1, 2, 6, 5, 4, 3, 7]$. Diferente do método *k-swap*, ao trocar as arestas, vértices que não possuem ligações diretas com tais arestas mudam de posição. Após as operações, se a nova configuração for melhor que a anterior, esta é mantida, caso contrário, a operação de troca é desfeita.

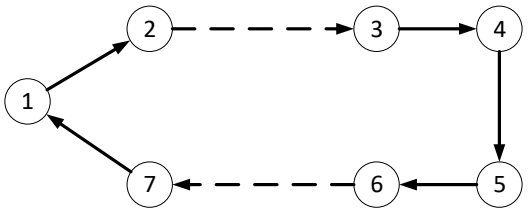


Figura 3.5: Exemplo da execução do *2-opt*, momento da escolha das arestas para remoção.

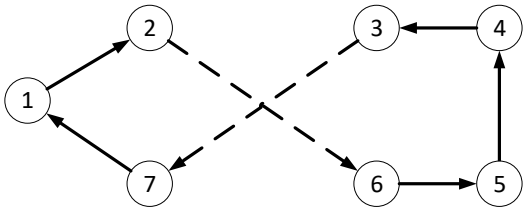


Figura 3.6: Exemplo da execução do método *2-opt* após trocar arestas.

3.3.3 Agrupamento de *1-blocks*

O agrupamento de *1-blocks* é uma busca local que tem como objetivo minimizar o numero de *1-blocks* em cada linha de uma matriz binária. Um *1-block* é definido pelo conjunto de entradas consecutivas de valor igual a 1 em uma mesma linha de uma matriz binária. O agrupamento de *1-block* é definido da seguinte forma: dada uma matriz binária, para cada linha da matriz, encontrar dois ou mais *1-blocks* e então, movimenta-se todas as colunas do primeiro *1-block*, uma a uma, para antes ou depois das colunas do segundo *1-block*. Caso ambos movimentos piorem a solução, tal coluna não é movimentada. Se a linha possuir três *1-blocks*, primeiramente movimenta-se as colunas do primeiro *1-block* para junto do segundo, então movimenta-se as colunas do segundo *1-block* para o terceiro sendo o segundo formado pelas colunas originais e as colunas que pertenciam ao primeiro *1-block* e foram movimentadas.

3.4 Busca Local Iterada

A Busca Local Iterada (*Iterated Local Search*) proposto por Lourenço et al. (2003) é uma metaheurística que combina busca local e perturbação. Este método consiste em aplicar

iteradamente métodos de busca local e perturbações, de modo a explorar com mais facilidade um grande espaço de busca. Ao executar o método de busca local, existe a possibilidade da solução ficar presa em um ótimo local e assim explorar apenas uma pequena parte do espaço de busca. Para melhorar esta característica, a perturbação é aplicada com o objetivo de obter uma nova solução que faça parte de uma nova vizinhança, fazendo com que o espaço de busca e as chances de encontrar um ótimo global aumentem.

A Figura 3.7 mostra o espaço de busca de um problema hipotético. O ponto A representa a solução inicial, ao aplicar uma busca local a solução se encontra no ponto B. Ao continuar a busca local, a solução ficará limitada ao ótimo local, pois só movimento de melhora são efetuados pela busca.

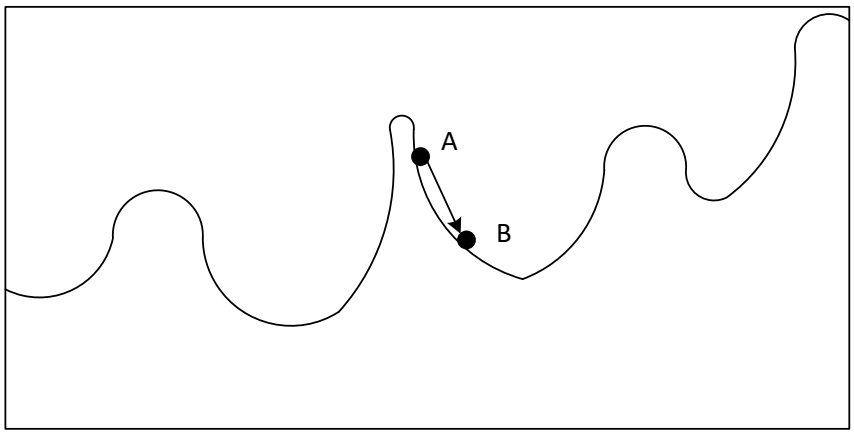


Figura 3.7: Exemplo de execução da ILS no espaço de busca.

Então, após aplicar o método de busca, o algoritmo executa a operação de perturbação na tentativa de gerar uma solução que esteja localizada fora da vizinhança anterior. Desta forma, a nova solução pode estar localizada na vizinha que possua ótimo local melhor que o anterior ou possua o ótimo global. A Figura 3.8 mostra a operação de perturbação. A linha pontilhada representa a perturbação da solução B, que resulta em uma nova solução C. Então, a ILS executa novamente a busca local resultando em uma nova melhor solução representada pelo ponto D. O algoritmo executa durante um número determinado de iterações e a melhor solução encontrada dentre todas as iterações é retornada.

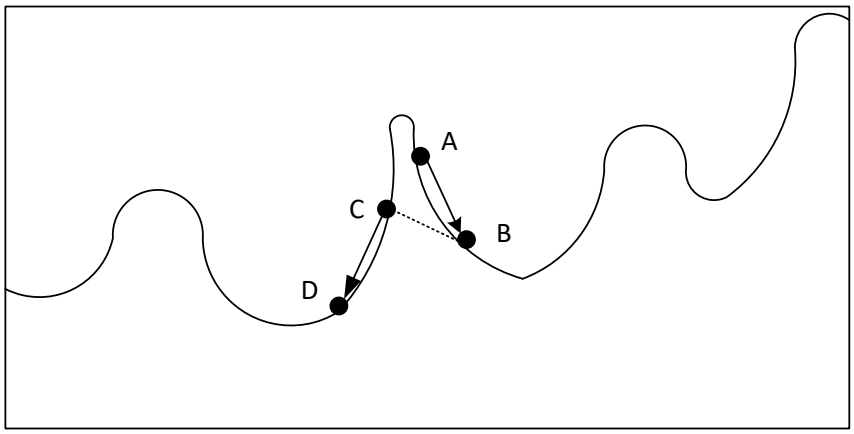


Figura 3.8: Exemplo da execução da ILS durante e após a perturbação.

Capítulo 4

Desenvolvimento

Neste capítulo, são descritos os detalhes da aplicação dos algoritmos, relatados anteriormente de maneira geral, ao MORP. São apresentados os detalhes de implementação ilustrados por pseudo-códigos e exemplos de funcionamento.

4.1 Geração da Solução Inicial

Para a geração da solução inicial foram implementados três métodos. O primeiro método é uma heurística em grafos que propõe uma modelagem do MORP em grafos e então executa o algoritmo BFS, cujo retorno posteriormente é transformado, dando origem a solução inicial para o MORP. O segundo método é a aplicação do algoritmo *Cheapest Insertion*, e o terceiro é uma heurística em duas fases que gera a solução pelo método *Best Insertion* e seguido da aplicação das buscas locais *2-opt* e agrupamento de *1-blocks*.

4.1.1 Heurística em Grafos

Como mencionado anteriormente, uma instância para o MORP é dada em forma de matriz binária, em que as linhas representam ordens de compras e as colunas representam produtos que compõem estas ordens. Para representação computacional do MORP, é definido um grafo G em que vértices representam ordens de compra e uma aresta entre dois vértices existe quando as ordens de compra correspondentes possuem produtos em comum. O peso de cada aresta é definido pela quantidade de produtos em comum entre as ordens de compra.

A Tabela 4.1 apresenta uma instância MORP e a representação em grafo correspondente é apresentada na Figura 4.1.

A ordem de compra i_6 da referida Tabela 4.1 possui todos os produtos disponíveis em sua composição e estes também compõem as demais ordens de compra, portanto, existe uma aresta entre o vértice 6 e todos os demais vértices. Especificamente, as ordens de compra i_1 e i_5 possuem três produtos em comum com a ordem de compra i_6 e um produto em comum

Tabela 4.1: Representação em grafo.

	j_1	j_2	j_3	j_4	j_5	j_6
i_1	1	1	0	0	0	1
i_2	0	0	1	0	0	0
i_3	1	0	0	0	0	1
i_4	0	0	0	1	0	0
i_5	0	1	0	1	1	0
i_6	1	1	1	1	1	1

entre si. Desta forma, o peso das arestas {1, 6} e {1, 5} é três e o peso da aresta {1, 5} é um. O mesmo processo se repete para as demais ordens de compra para construção do grafo.

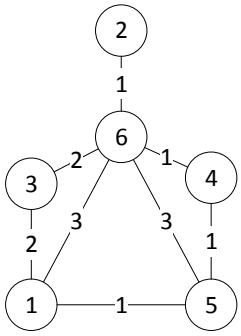


Figura 4.1: Grafo referente a Tabela 4.1

Neste trabalho, a BFS aplicada ao MORP utiliza como vértice inicial aquele de menor grau. A prioridade de exploração dos vértices é determinada pela menor soma dos pesos das arestas incidentes em um vértice. Por exemplo, considere um vértice a de um grafo G que está conectado a dois vértices, b e c. O vértice b possui 4 arestas incidentes e o vértice c possui 3 arestas incidentes. O somatório dos pesos das arestas incidentes de c é 20 e de b é 10, então, o vértice b tem prioridade sobre c. O resultado da Busca em Largura é um vetor ϕ que representa a ordem em que os vértices foram explorados.

A Figura 4.2 exemplifica a execução da BFS no grafo representado na Figura 4.1. O vértice 2 (em azul em 4.2a) é o de menor grau, portanto, é o vértice inicial da busca. O vértice 2 é adicionado à fila e marcado como explorado. O referido vértice possui como vizinho apenas o vértice 6 (em amarelo em 4.2b), portanto, este vértice é adicionado à fila e marcado como explorado. O vértice 6 possui como vizinhos não explorados os vértices 1, 3, 4 e 5, (em verde em 4.2c). Estes vértices são marcados como explorados e adicionados à fila levando em consideração o somatório dos pesos das arestas incidentes em cada vértice em ordem crescente. A ordem em que os vértices são adicionados à fila é 4, 3, 5 e 1. Todos os vértices do grafo estão marcados como explorados, portanto, execução termina. Como mencionado, a fila representa a ordem em que os vértices foram explorados, denominada ϕ, ou seja, cada posição do vetor ϕ

representa uma ordem de compra específica e a sequência em que foram adicionadas representa a relação entre ordens de compra, levando em consideração o número de produtos em comum.

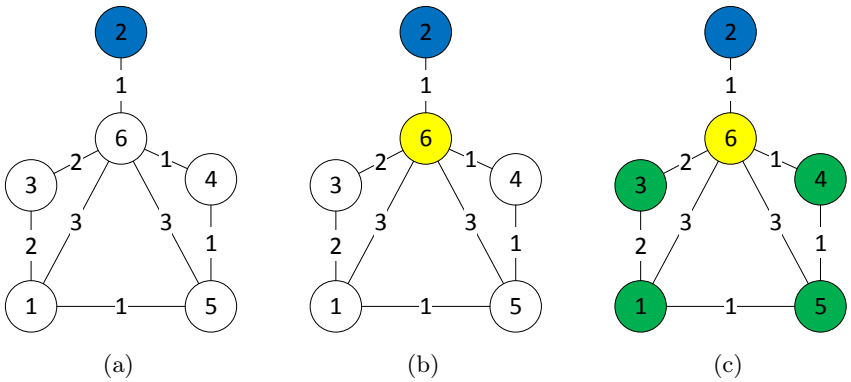


Figura 4.2: Busca em Largura aplicada ao MORP

A Tabela 4.2 exemplifica a construção da fila de acordo com a execução mostrada na Figura 4.2.

Tabela 4.2: Construção da fila de acordo com a execução da BFS.

Iteração	Fila
1	i_2
2	i_2, i_6
3	$i_2, i_6, i_4, i_3, i_5, i_1$

O Algoritmo 1 é o pseudocódigo da BFS modificada aplicada ao MORP desenvolvida neste trabalho. Como mencionado anteriormente, a BFS recebe um grafo G formado por ordens de compra e suas relações marcam o primeiro vértice v, escolhido aleatoriamente, como visitado (linha 3) e adicionado à fila Q (linha 5). Então, enquanto a fila Q não estiver vazia (laço das linhas 6-18), todos os vizinhos do vértice v (laço das linhas 8-13) são marcados como explorados e adicionados a um vetor auxiliar T (linhas 9, 10 e 11). Após isto, o vetor auxiliar T é ordenado (linha 14), em ordem crescente, pelo somatório dos pesos das arestas incidentes em cada vértice e adicionado a lista Q (linha 15). Por fim, o vértice v é adicionado a ϕ (linha 16) e retirado da lista Q (linha 17). Desta forma, a BFS retorna o vetor ϕ como sequencia de ordens de compra.

Algoritmo 1: Pseudocódigo da BFS modificada.

```
1  Entrada: Grafo de ordens G.
2  v ← f(G);
3  Marque v como visitado;
4  φ ← ∅;
5  Fila Q ← v;
6  enquanto Q ≠ ∅ faça
7  |  T ← ∅;
8  |  para todo vértice w vizinho de v hacer
9  |  |  se w é marcado como não explorado então
10 |  |  |  Insira w ao final de T;
11 |  |  |  Marque w como explorado;
12 |  |  fim
13 |  fin
14 |  Ordene T crescentemente levando em consideração o somatório dos pesos das
   |  arestas incidentes em cada vértice;
15 |  Insira todos os elementos de T no final de Q;
16 |  φ ← v;
17 |  Remova v de Q;
18 fim
```

O Algoritmo 2 apresenta o pseudocódigo da função que recebe o vetor de ordens de compra ϕ, resultado da BFS e a partir dele constrói a solução π. Para cada ordem i pertencente a ϕ (laço das linhas 3-8), adiciona-se tal ordem ao vetor auxilia A (linha 4) e verifica se existe algum produto que está contido em todas as ordens de compras de A (condição das linhas 5,6 e 7). Ao final da verificação de todas as ordens de compra, é garantido que todos os produtos estarão contidos em π.

Algoritmo 2: Pseudocódigo da construção do vetor π

```
1  Entrada: Lista de ordens φ.
2  A ← ∅;
3  para cada ordem i ∈ φ hacer
4  |  A ← A ∪ i;
5  |  se c(p_i) ⊆ A então
6  |  |  Insira p_i ao final de π;
7  |  fim
8  fin
```

Para obter bons resultados no sequenciamento de produtos é necessário observar tanto

os produtos quanto as ordens de compra, portanto, o vetor ϕ auxilia na criação do vetor de permutações π. A construção de π é feita como descrito a seguir. O vetor ϕ é percorrido analisando-se iterativamente cada ordem de compra na sequência em que são listados. Esta análise consiste em verificar quais produtos estão presentes em cada ordem. Uma vez que todas as ordens de compra de um produto específico tenham sido analisadas, este produto é então adicionado ao final do vetor de solução π. Devido ao fato de todas as ordens de compras estarem presentes no vetor ϕ é garantido que todos os produtos necessários para atender todas as ordens de compra estarão no vetor π.

A Tabela 4.3 demonstra a construção de π a partir de ϕ, gerado pela execução da BFS exemplificada na Figura 4.2. Na primeira iteração, o vetor π está vazio pois não há produto exclusivamente na ordem de compra i_2. Na segunda iteração, o vetor π recebe o produto j_3 pois o mesmo está presente nas ordens de compra i_2 e i_6 simultaneamente. Na terceira iteração e quarta iteração, o vetor π não recebe nenhum novo produto pois não há produto que atenda o critério de seleção. Na quinta iteração o vetor π recebe os produtos j_4 e j_5. O produto j_4 está presente no conjunto de ordens $[i_4, i_5, i_6]$ e o produto j_5 está presente no conjunto de ordens $[i_5, i_6]$, ambos já analisados. O mesmo processo é repetido para as ordens de compra restantes, e por temos $\pi = [j_3, j_4, j_5, j_1, j_2, j_6]$.

Tabela 4.3: Construção do vetor solução π a partir do vetor ϕ.

Iteração	ϕ	π
1	$[i_2]$	$[\emptyset]$
2	$[i_2, i_6]$	$[j_3]$
3	$[i_2, i_6, i_4]$	$[j_3]$
4	$[i_2, i_6, i_4, i_3]$	$[j_3]$
5	$[i_2, i_6, i_4, i_3, i_5]$	$[j_3, j_4, j_5]$
6	$[i_2, i_6, i_4, i_3, i_5, i_1]$	$[j_3, j_4, j_5, j_1, j_2, j_6]$

4.1.2 *Cheapest Insertion* Aplicado ao MORP

O *Cheapest Insertion* aplicado ao MORP, análogo ao *Best Insertion*, constrói um vetor de permutação π considerando cada elemento sendo um produto. Por exemplo, o método inicia com uma permutação parcial para π contendo apenas um produto, então, o segundo produto é escolhido dentre os produtos restantes, ou seja, analisa-se todos os produtos que não estão na solução escolhendo o que resulta em menor espalhamento de ordem. Os demais produtos são inseridos de forma análoga a inserção do segundo até que uma permutação π completa seja construída.

O Algoritmo 3 apresenta o pseudocódigo do *Cheapest Insertion*. O algoritmo recebe a lista de produtos J e retorna uma solução π. Inicialmente, π recebe o primeiro produto (linha 3), escolhido de forma aleatória. Então, para todas as posições que vão de 2 a $|J|$ (laço das linhas 4-13), analisa-se todo produto que pertence a J (laço das linhas 5-10) e que não está contido

em π (condição das linhas 6-9). Desta forma, ao final das $|J|-1$ iterações, o algoritmo constrói a solução inicial completa π.

Algoritmo 3: Pseudocódigo do *Cheapest Insertion*

```
1  Entrada: Lista de produtos J.
2  i ← 1;
3  π ← primeiro produto;
4  enquanto i < |J| faça
5      para cada produto j ∈ J hacer
6          se j ⊄ π então
7              Experimente j na posição π_i;
8              k ← avalie e atualize melhor produto para posição π_i;
9          fim
10     fin
11     π_i ← k;
12     i ← i + 1;
13 fim
```

4.1.3 Uma Heurística em Duas Fases Aplicada ao MORP

A heurística é definida pelas seguintes fases: a primeira consiste em construir uma solução factível para o problema através do método *Best Insertion*, a segunda fase é o refinamento da solução usando as duas buscas locais definidas anteriormente.

O *Best Insertion* aplicado ao MORP considera cada elemento sendo um produto e então constrói uma solução completa, ou seja, constrói um vetor de permutação π. Por exemplo, o método inicia com uma permutação parcial para π contendo apenas um produto, então, o segundo produto é inserido à frente do primeiro, pois não existe diferença de espalhamento se considerarmos apenas dois produtos. Para a inserção do terceiro produto, analisa-se todas as possibilidades de inserção e insere na posição que configura melhor permutação parcial. Os demais produtos são inseridos de forma análoga ao terceiro até que uma permutação π completa seja construída.

A Figura 4.3 exemplifica a construção de uma solução completa composta por três elementos cuja função objetivo é minimização. A construção inicia com o elemento 0, então, o método verifica qual a melhor posição para a inserção do elemento 1, ou seja, antes ou depois do elemento 0 inserindo o elemento 1 na posição que melhor satisfaça a função objetivo. Neste caso é preciso minimizar o problema, portanto foi escolhida a inserção que resultou no valor 3 para a função objetivo. Para a inserção do elemento 2, o método verifica todas as possibilidades de inserção e o mantém na posição que consiste em melhor configuração da solução.

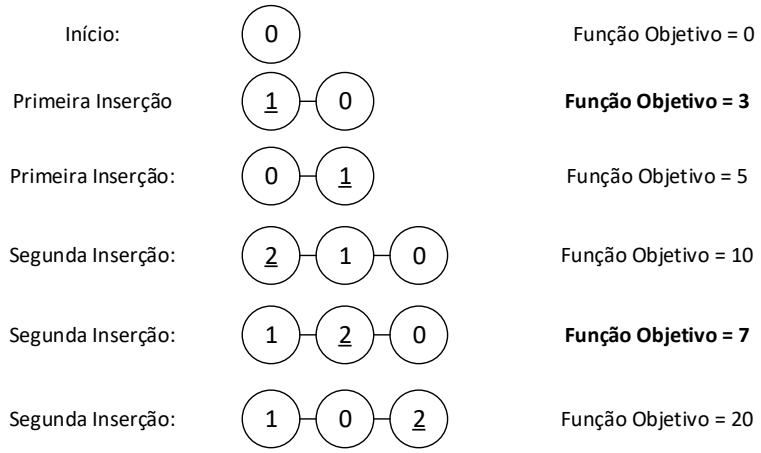


Figura 4.3: Exemplo da execução do *Best Insertion*.

O Algoritmo 4 apresenta o pseudocódigo do *Best Insertion*. O algoritmo recebe a lista de produtos J e atribui a π o primeiro produto (linha 2), escolhido de forma aleatória. Então, todo produto pertencente a J (laço das linhas 3-11) que ainda não esta contido em π (condição das linhas 4-10) é analisado em todas as posições possíveis da solução π em construção (laço das linhas 5-8) e mantido na posição que melhor satisfaça a função objetivo (linha 9), até que todos os elementos de J sejam inseridos no vetor de solução π.

Algoritmo 4: Pseudocódigo do *Best Insertion*

```
1  Entrada: Lista de produtos J.
2  π ← primeiro produto;
3  para cada produto j ∈ J hacer
4      se j ⊈ π então
5          para cada posição i ∈ π hacer
6              Experimente j na posição π_i;
7              k ← avalie e atualize melhor posição para j;
8          fin
9          π_k ← p_j;
10     fim
11 fin
```

Após a construção da solução factível através do *Best Insertion* a mesma é refinada com a execução das buscas locais *2-swap* e agrupamento de *1-blocks*. O refinamento foi aplicado ao *Best Insertion* pois foi o método que gerou as melhores soluções dentre os três métodos propostos. Os testes entre os 3 métodos esta melhor descrito na seção 5.

4.2 ILS Aplicada ao MORP

Neste trabalho, a ILS aplicada ao MORP utiliza como busca local o agrupamento por *1-blocks* e o *2-swap*. Adicionalmente, utiliza-se como mecanismo de perturbação o método *2-opt*. Desta forma, algoritmo executa x iterações em que cada iteração são executadas a busca local por *2-swap*, agrupamento por *1-blocks* e perturbação por *2-opt*. A ILS é aplicada à solução inicial gerada. O critério de parada é obedecido quando o número limite de iterações é atingido.

Para realizar a busca local pelo método *2-swap*, todas combinações de padrões de tamanho 2 são salvas, ou seja, todas as combinações de padrões tomados 2 a 2 são salvas em um vetor. Então, uma porcentagem α de combinações é selecionada para que os respectivos padrões sejam trocados. Por exemplo, se a quantidade de combinações possíveis é 10 e a porcentagem α é igual a 20%, 2 combinações são escolhidas. Cada combinação escolhida contém quais padrões devem ser trocados; portanto, para este exemplo, serão feitas 2 trocas de 2 padrões. Como mencionado anteriormente no Capítulo 3, a troca de padrões é aceita no caso de melhora ou solução igual e descartada caso a solução piore.

Para a execução do método de perturbação *2-opt*, o processo é análogo ao processo da busca local descrito anteriormente. São definidas todas as possíveis combinações de padrões tomados 2 a 2, então é embaralhado e a porcentagem β destes conjuntos é definida para seleção de padrões que serão trocados. A perturbação aceita qualquer movimento independente da melhora ou piora no valor da solução.

O método de agrupamento de *1-block* na ILS aplicada ao MORP, percorre as linhas da matriz de forma aleatória priorizando o gargalo da instância. O gargalo é definido pela linha da matriz que possui o maior número de elementos não nulos. O algoritmo identifica a linha que representa o gargalo e inicia a execução a partir desta. No decorrer da execução do algoritmo, o gargalo da solução pode variar, portanto, quando houver mudança na linha que representa o gargalo, esta é reposicionada para ser a próxima na execução da busca. Para evitar que a busca local analise a mesma linha repetidamente, caracterizando ciclicidade, todos os gargalos são salvos em uma lista e, antes de reposicionar um gargalo, verifica-se se o mesmo está na lista. Caso o gargalo esteja na lista, o mesmo não é reposicionado e a execução segue a ordem aleatória predefinida.

O Algoritmo 5 apresenta o pseudocódigo da ILS proposta aplicada ao MORP. O algoritmo recebe a solução inicial π gerada por algum método de construção de solução inicial e a proporção de aplicação da busca local α. A cada iteração (laço das linhas 3-11), o método executa as operações de perturbação (linha 5) e busca local na proporção β (linhas 6 e 7) um número de iterações pré-determinada, atualizando a solução π em caso de melhora (linhas 8, 9 e 10). Ao final, é retornada a solução π de melhor valor obtido.

Algoritmo 5: Pseudocódigo da Busca Local Iterada aplicada ao MORP.

Entrada: π, β.
Aplique a busca local de agrupamento de 1-blocks
enquanto *o número máximo de iterações não for atingido* **faça**
    $\pi' \leftarrow \pi$;
    Perturbe a solução π' em uma proporção β aplicando *2-opt*;
    Aplique a busca local *2-swap* em π' em uma proporção β;
    Aplique a busca local de agrupamento de *1-blocks* em π';
    se $Z^{\pi'}_{MORP}(Q^{\pi'}) < Z^{\pi}_{MORP}(Q^{\pi})$ **então**
        $\pi \leftarrow \pi'$;
    fim
fim
retorna Q^{π};

Capítulo 5

Experimentos Computacionais

Os experimentos computacionais foram divididos em dois núcleos: experimentos preliminares e comparação com os resultados da literatura. No primeiro núcleo, foram realizados experimentos para definição da composição do método proposto, ao passo que no segundo núcleo os experimentos se dedicam a comparar a versão final do método proposto com os resultados da literatura e também a lançar as bases para futuras comparações.

O ambiente computacional adotado para todos os experimentos consiste em um computador com processador *Intel i5 Quad Core* de 2.27 GHz com 4 GB RAM sob o sistema operacional Windows 10. O código do método proposto foi escrito em C++, compilado com g++ 4.4.1 e a opção de otimização -O3.

5.1 Experimentos Preliminares

Foram implementadas quatro versões de BFS, uma versão do *Cheapest Insertion* e uma versão do *Best Insertion* para geração de solução inicial e comparadas entre si. Foram implementando também os métodos de *k-swap* e *2-opt* que foram comparados e analisados para perturbação e busca local. Para o método de agrupamento de *1-block* foram implementadas quatro variações e comparadas entre si. Todos os experimentos estão descritos nas seções a seguir.

5.1.1 Critério Guloso da BFS

Para a geração da solução inicial foram avaliados quatro critérios para guiar a BFS.

- O primeiro, descrito no Capítulo 4, explora os vértices do grafo priorizando a soma dos pesos das arestas incidentes em um vértice, quanto menor a soma maior a prioridade. Por exemplo, considere um vértice a de um grafo G que está conectado a dois vértices, b e c. O vértice b possui 4 arestas incidentes e o vértice c possui 3 arestas incidentes. O

somatório dos pesos das arestas incidentes de c é 20 e de b é 10, então, o vértice b tem prioridade sobre c;

- O segundo critério prioriza o peso das arestas que ligam o vértice atual aos demais, quanto maior o peso, maior a prioridade;
- O terceiro critério implementado prioriza o grau dos vértices, ou seja, o vértice que possui maior número de vizinhos tem maior prioridade de exploração;
- O quarto critério é implementado priorizando os vértices que tem o menor somatório de pesos de arestas, ou seja, no grafo G citado acima, o vértice c teria prioridade sobre b.

Os experimentos foram executados utilizando todas as instâncias disponíveis, comparando as soluções e os tempos de execução para cada instância. O critério que obteve os melhores resultados foi o que prioriza os vértices com menor soma dos pesos das arestas incidentes em um vértice, o primeiro critério descrito nesta seção.

5.1.2 Escolha do Método para Geração da Solução Inicial

O método BFS que prioriza a menor soma dos pesos das arestas incidentes em um vértice, o método *Best Insertion* e o método *Cheapest Insertion* foram comparados a fim de determinar o melhor método para a geração da solução inicial. Os testes foram realizados para todas as instâncias disponíveis. Nesses testes, o método BFS obteve 99,63% dos melhores resultados e 73,96% de *gap* em relação ao *Cheapest Insertion*, fazendo com que o método *Cheapest Insertion* fosse descartado. O método *Best Insertion* obteve 93,68% dos melhores resultados e 12,88% de *gap* em comparação com a BFS. Em relação ao tempo, a BFS obteve 84,69% das soluções mais rápidas mas a diferença do tempo médio das gerações das soluções entre os dois métodos é de apenas 0,004 segundos. De acordo com essas informações conclui-se que, o método *Best Insertion* é mais eficiente para a geração da solução inicial porque gera soluções melhores que o método da BFS de forma pouco mais lenta, o método demora em média 0,009 segundos para gerar uma solução.

5.1.3 Escolha da Busca Local e Perturbação

Como descrito anteriormente, a ILS aplicada ao MORP possui 2 métodos de busca local e um método e perturbação. Um dos métodos de busca local é o agrupamento de *1-blocks*, o segundo método de busca local e o método de perturbação foram escolhidos com base em testes realizados em todas as instâncias disponíveis. Os métodos *2-opt* e *k-swap* foram experimentados para executar tal função, sem que o mesmo método fosse repetido, ou seja, se a busca local fosse executada por *2-opt* a perturbação deveria ser executada por *k-swap*, pois caso contrário, a busca local e a perturbação se anulariam. Os testes foram executados para k

igual a 2 e 3, fazendo com que o método *2-swap* fosse escolhido para busca local e o método *2-opt* para perturbação.

5.1.4 Aleatoriedade no Método de Agrupamento de *1-blocks*

O método de agrupamento de *1-blocks* na ILS aplicada ao MORP, percorre a linha da esquerda para a direita, encontra os dois primeiros *1-blocks* e então tenta agrupá-los, movimentando as colunas do primeiro em relação ao segundo. Foi implementada uma versão que insere aleatoriedade neste método, em que duas decisões são tomadas de forma aleatória: primeiro é decidido se a linha será percorrida da direita para esquerda ou da esquerda para direita e depois é decidido se o primeiro *1-block* será movimentado em relação ao segundo ou se o segundo será movimento em relação ao primeiro. Foram realizados experimentos preliminares e os resultados não foram satisfatórios, fazendo com que o uso da aleatoriedade fosse descartada.

5.1.5 Método de Agrupamento de *1-blocks* Utilizando *Best Insertion*

O método de agrupamento de *1-blocks* utilizando *Best Insertion* funciona da seguinte forma: ao identificar dois ou mais *1-blocks* movimenta-se todas as colunas do primeiro *1-block*, uma a uma, para antes ou depois das colunas do segundo *1-block* ou todas as posições localizadas entre a primeira coluna do segundo *1-block* e a última coluna do primeiro *1-block*, ou seja, todas possibilidades de inserção entre os dois *1-blocks*. Este método obteve soluções pouco melhores em relação ao método de agrupamento de *1-blocks* sem utilização do *Best Insertion* com tempo de execução consideravelmente maior, portanto, a utilização do *Best Insertion* no agrupamento de *1-blocks* foi descartado.

5.1.6 Estratégia de Intensificação

A priorização do gargalo foi a estratégia de intensificação aplicada ao agrupamento de *1-block*. Como mencionado anteriormente, o gargalo da instância é a linha da matriz que possui o maior espalhamento de ordem, portanto, para priorizar o gargalo durante a execução do método encontra-se um gargalo e tal linha é reposicionada para ser a próxima na execução da busca. Duas maneiras de tratar o gargalo foram implementadas: a primeira realiza a busca no gargalo e salva em uma lista de gargalos para evitar repetições, a segunda realiza a busca no gargalo enquanto houver melhoria e salva em uma lista, análogo à primeira. A segunda implementação se mostrou a mais eficiente dentre todas as implementações propostas, tanto em relação ao tempo quanto em relação aos resultados.

5.1.7 Ajuste de Parâmetros

A ILS aplicada ao MORP possui três principais parâmetros que influenciam diretamente no desempenho do algoritmo: porcentagem de busca local, porcentagem de perturbação e número de iterações. Por exemplo, se a ILS executar com porcentagem de busca local alta e porcentagem de perturbação baixa, pode ficar presa em ótimo local. Em relação ao número de iterações, a ILS pode estar executando mais que o necessário, portanto, ao calibrar tal parâmetro, é possível reduzir o tempo de execução do algoritmo. Para a escolha dos melhores valores foi utilizado o *irace* (López-Ibáñez et al., 2016), um pacote que implementa uma série de procedimentos de configuração automática, em particular oferece o procedimento de corrida iterada, que vem sendo usado com sucesso para configurar automaticamente vários algoritmos do estado da arte.

Para os parâmetros foram pré-selecionados alguns valores conforme apresentados na Tabela 5.1, em que a coluna *Parâmetros* indica qual parâmetro foi analisado. Na coluna *Valores* é apresentado o conjunto dos diferentes valores possíveis para cada parâmetro. Todas as instâncias foram utilizadas nos testes do irace.

Tabela 5.1: Ajuste de parâmetros usando *irace*.

Parâmetros	*Valores*
Busca Local(%)	$10, 20, 30, 40, 50, 60, 70, 80, 90, 100$
Perturbação(%)	$5, 10, 15, 20, 25, 30, 35, 40, 45, 50$
Iterações	$50, 100, 150, 200$

Como resultado o irace apresentou três configurações que foram as melhores:

- Busca Local = 30%, Perturbação = 20% e Iterações = 100;
- Busca Local = 30%, Perturbação = 25% e Iterações = 100;
- Busca Local = 100%, Perturbação = 20% e Iterações = 100.

É preciso destacar que uma configuração não domina a outra, ou seja, em alguns testes uma configuração obteve melhores resultados enquanto, em outros testes outra configuração se mostrou melhor. Portanto entre as configurações citadas foi escolhida a terceira, dado que o tempo de execução do método nestas configurações é baixo, optou-se pela maior exploração do espaço de busca pela busca local.

5.2 Comparação de Resultados

Devido à utilização de componentes de aleatoriedade, o método proposto foi executado independentemente dez vezes por instância, sendo que os melhores resultados obtidos e a média de todos os resultados obtidos são utilizados nas análises realizadas. Comparam-se

o comprimento dos espalhamento de ordens obtidas e o tempo de execução da heurística proposta com o estado da arte. Todavia, ressalta-se que a comparação direta entre os tempos de execução não deve ser realizada, uma vez que os métodos foram executados em arquiteturas computacionais distintas.

As tabelas de comparação são divididas em três seções. A primeira seção é referente à características das instâncias: nome (*Instância*), número de linhas (m) quando o nome não é informado, o número de colunas (n) e a densidade (v), quando disponível. A segunda seção é referente aos melhores resultados encontrados na literatura: o espalhamento médio (BKS) e o tempo médio de execução (T). A terceira seção é referente aos resultados obtidos pelo método proposto: solução média (S), melhor solução (S^*), distância percentual entre melhor resultado obtido e o melhor resultado da literatura (*gap*), calculado como $100 \times (S^* - BKS)/BKS$), o desvio padrão em dez execuções independentes (σ) e o tempo médio de execução (T), em segundos.

5.2.1 Instâncias e Comparações com Fink e Voß (1999)

Essas instâncias, geradas aleatoriamente, são as mesmas utilizadas por Fink e Voß (1999), gentilmente cedidas pelo autor. O parâmetro v representa o nível de densidade das matrizes, de maneira que, quanto menor o valor de v, mais densa é a instância. O parâmetro m representa o número de linhas, ou seja, número de ordens de compra da instância. Há 4 conjuntos de instâncias para $m \approx 50$ e $v = \{0,25, 0,5, 0,75, 1\}$ com 100 instâncias cada, e 4 conjuntos de instância para $m \approx 60$ com 100 instâncias para $v = \{0,25, 0,5\}$, 98 instâncias para $v = 0,75$ e 99 instâncias para $v = 1$, totalizando 797 instâncias. Para todas as instâncias, o número de colunas (n) é aproximado ao número de linhas.

A Tabela 5.2 expõe os resultados obtidos com a ILS aplicada ao MORP (*ILS-MORP*) comparados ao método Busca Tabu Reativa com 5000 iterações (*TSRE500*), método proposto por Fink e Voß (1999) que obteve os melhores resultados da literatura.

Tabela 5.2: Comparação com o melhor resultado de Fink e Voß (1999).

m	v	*TSRE5000*		*ILS-MORP*				
		BKS	T^1	S	S^*	*gap*	σ	T
50	0,25	14,04	142,6	14,90	14,31	1,92	0,49	0,48
50	0,50	12,43	79,5	15,03	14,56	17,13	0,30	0,44
50	0,75	4,87	58,6	3,04	2,71	-37,57	0,24	0,41
50	1,00	1,38	35,1	1,48	1,29	-6,52	0,19	0,70
60	0,25	16,41	159,5	18,18	17,44	6,27	0,54	0,97
60	0,50	14,61	142,6	16,84	16,35	11,90	0,35	0,80
60	0,75	5,33	98,2	4,18	3,79	-28,89	0,31	0,92
60	1,00	1,45	58,9	1,66	1,32	-8,96	0,29	0,83

Analisando a Tabela 5.2 observa-se que o método *ILS-MORP* obteve bons resultados, alcançando *gap* médio de $-5,59\%$. O método obteve melhores resultados para instâncias esparsas (i.e., $v = 0,75$ e $v = 1,00$), determinando assim novos melhores resultados. Para instâncias esparsas, a solução inicial se igualou à melhor solução em 330 instâncias dentre 397. Para instâncias com densidade $v \in \{0,25, 0,50\}$, o algoritmo demonstrou boa convergência encontrando o melhor resultado após executar, em média, 48,69% das iterações. A melhoria da ILS em relação a solução inicial foi significativa para instâncias mais densas obtendo melhora máxima de 41,22%, suprindo a dificuldade do método de geração da solução inicial nestes casos.

5.2.2 Comparação com De Giovanni et al. (2013)

Conforme descrito anteriormente, De Giovanni et al. (2013) propuseram um algoritmo genético (GAG) para solução de um problema equivalente ao MORP. A Tabela 5.3 apresenta a comparação entre os resultados obtidos por De Giovanni et al. (2013) e o método proposto. Nota-se que o GAG avalia a solução somando todos os espalhamentos de ordens da solução e subtraindo m, portanto, os resultados apresentados na Tabela 5.3 são com base nesta forma de avaliação.

O método proposto se igualou ao algoritmo genético proposto por De Giovanni et al. (2013) em 7 das comparações, encontrou melhor solução para 6 comparações e se mostrou inferior em 18 instâncias com *gap* médio de 3,18%. A solução inicial gerada encontrou melhores resultados para 19 das instâncias. O método demonstrou boa convergência, encontrando o melhor resultado após executar 45,02% das iterações, em média. Além disso, ILS melhorou a solução inicial de forma significativa, com máximo de 34,67%.

[1]O algoritmo *TRSE5000.* foi executado em um computador com processador PentiumII/266
[2]O algoritmo GAG foi executado em um computador com processador 3.2GHz Intel i7-960.

Tabela 5.3: Comparação com o melhor resultado de De Giovanni et al. (2013).

Instância	*GAG*		*ILS-MORP*				
	BKS	T^2	S	S^*	gap	σ	T
wli	24	0,00	24,00	24	0,00	0,00	0,01
wsn	96	0,50	101,10	100	4,17	0,74	0,05
v4000	56	0,20	52,60	51	-8,93	0,97	0,02
v4050	38	0,20	38,80	38	0,00	0,42	0,01
v4090	109	0,80	116,50	113	3,67	1,96	0,07
v4470	237	4,20	278,00	265	11,81	17,85	0,21
x0	298	5,30	334,90	315	5,70	18,80	0,23
w1	39	0,30	44,50	41	5,13	2,76	0,02
w2	233	2,00	254,90	251	7,73	3,70	0,16
w3	668	21,60	815,10	735	10,03	54,24	1,59
w4	1683	29,10	2427,30	2214	31,55	152,41	23,27
Shaw	184,32	0,19	185,82	183,96	-0,20	1,2	0,05
GP5	9341	15,09	9340,80	9340	-0,01	1,03	10,85
GP6	7359	27,23	7360,50	7359	0,00	1,96	8,89
GP7	7366	23,11	7377,10	7367	0,01	21,98	8,14
GP8	5858	37,93	5858,00	5858	0,00	0,00	8,22
NWRS7	316	4,46	364,30	317	0,32	25,37	0,42
NWRS8	591	7,53	613,30	604	2,20	8,51	0,40
SP3	1622	26,28	1767,40	1690	4,19	43,52	1,33
SP4	3450	56,79	3584,40	3390	-1,74	114,67	3,52
A_FA+AA-_1	184	4,08	218,90	194	5,43	14,96	0,44
A_FA+AA-_2	58	0,13	57,90	51	-12,07	3,38	0,07
A_FA+AA-_6	91	0,19	94,60	91	0,00	1,51	0,09
A_FA+AA-_8	112	0,65	130,50	121	8,04	7,09	0,17
A_FA+AA-_11	87	0,4	94,90	89	2,30	3,25	0,20
A_FA+AA-_12	48	0,15	53,70	48	0,00	4,06	0,08
A_FA+AA-_13	304	1,2	327,40	315	3,62	11,03	0,55
A_FA+AA-_15	52	0,1	47,30	44	-15,38	1,70	0,05
B_REVAL_145	129	0,5	175,90	157	21,71	19,11	0,41
Wood A	35,25	0,07	37,25	35,25	0,00	1,1	0,03
Wood B	35,73	0,06	39,84	39,1	9,43	0,62	0,02

5.2.3 Instâncias de VLSI Hu e Chen (1990).

Oriundas do projeto para o *Gate Matrix Connection Cost Minimization Problem*, este conjunto possui 25 instâncias de empresas da Ásia e introduzidas por Hu e Chen (1990). Estas instâncias possuem matrizes de dimensões que variam entre 5×7 até 202×141. A Tabela 5.4 apresenta os resultados obtidos pelo ILS para este conjunto de instâncias.

Tabela 5.4: Resultados encontrados para as instâncias VLSI.

Instância	S	S^*	σ	T
v1	2,00	2,00	0,00	0,01
v4000	5,21	5,20	0,03	0,02
v4050	3,08	3,00	0,04	0,01
v4090	5,05	4,96	0,05	0,08
v4470	7,24	6,92	0,19	0,22
vc1	5,81	5,47	0,20	0,03
vw1	2,40	2,40	0,00	0,00
vw2	2,41	2,38	0,06	0,00
w1	2,48	2,33	0,09	0,04
w2	5,41	5,27	0,10	0,14
w3	9,84	9,04	0,63	1,37
w4	11,49	9,92	0,78	23,24
wan	1,70	1,67	0,05	0,00
wli	2,18	2,18	0,00	0,01
wsn	5,89	5,76	0,10	0,03
x0	8,78	8,08	0,46	0,41
x1	5,60	5,60	0,00	0,01
x2	8,50	8,50	0,00	0,01
x3	11,89	11,86	0,06	0,04
x4	0,33	0,33	0,00	0,00
x5	0,33	0,33	0,00	0,03
x6	0,35	0,35	0,00	0,13
x7	3,00	3,00	0,00	0,00
x8	4,45	4,45	0,00	0,02
x9	5,53	5,53	0,00	0,11

De acordo com a Tabela 5.4 o método proposto encontrou os resultados com o tempo médio de 0,85 segundos. O desvio padrão médio entre as execuções independentes do método é de 0,11 segundos, mostrando-se robusto em relação a variação dos valores das soluções. Para este conjunto de instâncias a solução inicial gerada se igualou ao melhor resultado em 13 instâncias, e em geral, convergiu em média após executar 37,70% das iterações. A ILS melhorou a solução inicial em no máximo 18,43%.

5.2.4 Instâncias do *First Constraint Modeling Challenge* (Smith e Gent, 2005)

Este conjunto contém uma seleção de 46 instâncias MOSP geradas aleatoriamente propostas para o *First Constraint Modeling Challenge* (Smith e Gent, 2005), desafio mundial em que vários autores submeteram seus resultados para estas instâncias. Os resultados obtidos são apresentados na Tabela 5.5.

Os resultados obtidos para as instâncias do *First Constraint Modeling Challenge*, mostradas na Tabela 5.5, mostram que o método obteve desvio padrão médio de 0,14 e tempo de execução médio igual a 0,93 segundos. Para este conjunto de instâncias a solução inicial se igualou a melhor encontrada em 11 instâncias. Em geral, o método convergiu após executar 39,95% das iterações, em média. O método da ILS obteve melhoria de no máximo de 31,66% sobre a solução inicial.

Tabela 5.5: Resultados encontrados para as instâncias do *First Constraint Modeling Challenge*.

Instância	S	S^*	σ	T
GP1	43,43	43,38	0,03	0,95
GP2	37,94	37,88	0,04	0,92
GP3	38,21	38,18	0,07	0,89
GP4	28,59	28,56	0,02	0,83
GP5	93,41	93,39	0,01	9,53
GP6	73,64	73,60	0,07	7,09
GP7	73,91	73,67	0,50	7,30
GP8	58,74	58,58	0,26	7,51
Miller_20_40	18,23	18,15	0,05	0,12
NWRS1	4,23	3,90	0,33	0,02
NWRS2	5,70	5,40	0,18	0,02
NWRS3	6,10	6,07	0,04	0,04
NWRS4	8,44	8,20	0,15	0,05
NWRS5	10,41	10,40	0,02	0,09
NWRS6	11,95	11,90	0,06	0,09
NWRS7	15,06	14,44	0,71	0,29
NWRS8	25,17	23,96	0,74	0,43
SP1	3,79	3,60	0,12	0,07
SP2	11,71	11,48	0,24	0,50
SP3	23,44	22,32	0,57	1,43
SP4	36,14	34,55	1,02	3,58
Shaw1	9,33	9,25	0,05	0,05
Shaw2	8,41	8,35	0,08	0,05
Shaw3	9,48	9,40	0,04	0,05
Shaw4	9,44	9,35	0,07	0,05
Shaw5	8,82	8,80	0,03	0,05
Shaw6	9,73	9,60	0,12	0,05
Shaw7	8,91	8,80	0,08	0,05
Shaw8	9,81	9,75	0,06	0,05
Shaw9	9,19	9,10	0,06	0,05
Shaw10	8,99	8,95	0,03	0,05
Shaw11	10,00	10,00	0,00	0,05
Shaw12	9,17	9,05	0,09	0,05
Shaw13	9,21	9,05	0,10	0,04
Shaw14	9,41	9,30	0,07	0,05
Shaw15	9,38	9,30	0,05	0,05
Shaw16	9,40	9,30	0,08	0,05
Shaw17	9,82	9,75	0,05	0,05
Shaw18	9,78	9,65	0,08	0,05
Shaw19	9,45	9,45	0,00	0,05
Shaw20	9,45	9,40	0,04	0,05
Shaw21	7,40	7,40	0,00	0,05
Shaw22	9,29	9,20	0,05	0,05
Shaw23	9,25	9,20	0,04	0,05
Shaw24	9,56	9,50	0,04	0,05
Shaw25	9,31	9,30	0,03	0,05

5.2.5 Instâncias de Chu e Stuckey (2009)

O terceiro conjunto de instâncias proposto por Chu e Stuckey (2009), contém 200 instâncias MOSP de dimensões que variam entre 30×30 até 125×125. Os problemas foram agrupados em subconjuntos de 25 instâncias de acordo com o número de padrões e peças. A Tabela 5.6 apresenta os resultados obtidos.

Tabela 5.6: Resultados encontrados para as instâncias Chu e Stuckey (2009).

m	n	S	S^*	σ	T
30	30	13,45	13,21	0,16	0,11
40	40	17,75	17,39	0,24	0,22
50	50	22,08	21,51	0,38	0,62
75	75	32,01	31,16	0,56	1,53
50	100	56,15	54,90	0,82	1,92
100	50	13,54	13,12	0,26	0,94
100	100	41,40	40,18	0,73	3,83
125	125	50,43	49,26	0,81	8,83

Ao analisar os dados apresentados, observa-se que o método possui média de desvio padrão igual a 0,49, demonstrando baixa variação na qualidade das soluções. O tempo médio de solução das instâncias foi de 2,25 segundos. A solução inicial gerada se igualou a melhor encontrada em 103 instâncias, e em média o método convergiu após executar 49,14% das iterações. A melhoria da ILS em relação a solução inicial foi de no máximo 19,64%.

5.2.6 Instâncias SCOOP

Conjunto de instâncias reais fornecido pelo SCOOP *Consortium*[3], contém 187 instâncias do MOSP. Porém, a maioria destas instâncias são muito pequenas (por exemplo, 2 linhas e colunas) tendo soluções triviais, podendo ser descartadas. Deste conjunto, 24 instâncias com dimensões significativas, variando de 10 linhas e colunas a 49 linhas e 134 colunas, foram selecionadas para serem usadas nos experimentos. A Tabela 5.7 apresenta os valores obtidos para cada instância.

As respostas obtidas para as instâncias SCOOP mostram que o método possui baixa variação na qualidade das respostas e baixo tempo de execução. A média do desvio padrão é de 0,07 e a média do tempo de execução é de 0,11 segundos. Para este conjunto de instâncias, a solução inicial gerada se igualou à melhor solução encontrada em 4 instâncias. O algoritmo convergiu após executar 43,70% das iterações, em média. A ILS melhorou a solução inicial em no máximo 36,10%.

[3]http://www.scoop-project.net

Tabela 5.7: Resultados encontrados para as instâncias SCOOP.

Instância	S	S^*	σ	T
A_AP-9.d_3	1,75	1,70	0,04	0,02
A_AP-9.d_6	1,11	0,97	0,07	0,04
A_AP-9.d_10	1,62	1,60	0,03	0,01
A_AP-9.d_11	1,97	1,81	0,08	0,04
A_FA+AA-_1	2,08	1,85	0,13	0,45
A_FA+AA-_2	0,78	0,75	0,02	0,07
A_FA+AA-_6	1,19	1,15	0,03	0,09
A_FA+AA-_8	1,56	1,40	0,11	0,18
A_FA+AA-_11	1,06	0,93	0,08	0,20
A_FA+AA-_12	0,74	0,71	0,03	0,08
A_FA+AA-_13	2,44	2,35	0,07	0,60
A_FA+AA-_15	0,66	0,66	0,00	0,06
B_12F18_11	1,95	1,71	0,13	0,02
B_12M18_12	2,30	2,23	0,06	0,06
B_18AB1_32	2,33	2,33	0,00	0,01
B_18CR1_33	1,05	1,05	0,00	0,02
B_22X18_50	3,71	3,71	0,00	0,01
B_39Q18_82	1,83	1,79	0,07	0,02
B_42F22_93	0,89	0,89	0,00	0,01
B_CARLET_137	2,58	2,57	0,02	0,01
B_CUC28A_138	1,99	1,78	0,08	0,08
B_GTM18A_139	1,72	1,63	0,11	0,03
B_REVAL_145	2,75	2,22	0,39	0,50

Capítulo 6

Conclusões

Neste trabalho foram introduzidos os conceitos referentes ao Problema de Minimização de Espalhamento de Ordens (*Minimization of Order Spread* – MORP), um problema combinatório de ampla aplicação prática. Também foi revisada a literatura referente a este problema, detalhadas as abordagens empregadas e os resultados obtidos.

O problema foi formalmente descrito e uma implementação de uma heurística foi apresentada. Com base na pesquisa realizada foi possível compreender o problema e gerar uma solução satisfatória, aplicando uma heurística em duas fases para gerar a solução inicial e melhorar tal solução aplicando a Busca Local Iterada (*Iterated Local Serach* – ILS).

Experimentos computacionais extensivos foram realizados considerando 1091 instâncias da literatura, divididas em 5 grupos. Os resultados foram comparados com dois métodos da literatura, reconhecidos por gerarem as melhores soluções para as instâncias consideradas. Outras instâncias da literatura ainda não haviam sido consideradas no contexto do MORP, e portanto, não possuíam resultados reportados na literatura. Os resultados obtidos pelo método proposto para estas instâncias foram apresentados para criar uma base de comparação para possíveis trabalhos futuros.

Particularmente, para dois dos conjunto de instâncias já estabelecidas na literatura sobre o MORP a ILS proposta gerou novos melhores resultados para algumas das instâncias, especificamente as esparsas. Entretanto, não foi possível melhorar os resultados das instâncias densas. Em todos os conjuntos de instâncias considerados, o método proposto demonstrou robustez e baixo tempo de execução, além de rapidez de convergência. Vale ressaltar que a qualidade da solução inicial gerada pela heurística proposta contribui para rápida convergência da ILS. Em alguns casos, a solução inicial gerou soluções próximas do melhor resultado conhecido com um tempo computacional baixo.

Os trabalhos futuros se concentrarão em estudar maneiras de melhorar os resultados para instâncias mais densas, acelerando as técnicas já existentes ou aplicando novas técnicas, como paralelismo.

Referências Bibliográficas

Cheng, T.; Diamond, J. e Lin, B. (1993). Optimal scheduling in film production to minimize talent hold cost. *Journal of Optimization Theory and Applications*, 79(3):479–492.

Chu, G. e Stuckey, P. J. (2009). Minimizing the maximum number of open stacks by customer search. In *International Conference on Principles and Practice of Constraint Programming*, pp. 242–257. Springer.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.

De Giovanni, L.; Massi, G.; Pezzella, F.; Pfetsch, M.; Rinaldi, G. e Ventura, P. (2013). A heuristic and an exact method for the gate matrix connection cost minimization problem. *International Transactions in Operational Research*, 20(5):627–643.

Fink, A. e Voß, S. (1999). Applications of modern heuristic search methods to pattern sequencing problems. *Computers Operations Research*, 26(1):17 – 34.

Foerster, H. e Wäscher, G. (1998). Euro best applied paper competition simulated annealing for order spread minimization in sequencing cutting patterns. *European Journal of Operational Research*, 110(2):272 – 281.

Garey, M. R. e Johnson, D. S. (2002). *Computers and intractability*, volume 29. W. H. Freeman New York.

Gilmore, P. e Gomory, R. (1966). The theory and computation of knapsack functions. *Operations Research*, 14(6):1045–1074.

Hu, Y. H. e Chen, S.-J. (1990). Gm plan: a gate matrix layout algorithm based on artificial intelligence planning techniques. *IEEE transactions on computer-aided design of integrated circuits and systems*, 9(8):836–845.

Kim, B.-I.; Ki, Y.; Son, D.; Bae, B. e Park, J.-S. (2016). An algorithm for a cutting problem in window frame production. *International Journal of Production Research*, 54(14):4327–4339.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal, The*, 44(10):2245–2269.

Linhares, A. e Yanasse, H. H. (2002). Connections between cutting-pattern sequencing, vlsi design, and flexible machines. *Computers & Operations Research*, 29(12):1759–1772.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M. e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Lourenço, H. R.; Martin, O. C. e Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics*, pp. 320–353. Springer.

Madsen, O. B. G. (1988). An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. *Operational Reseach Society Ltd*, 39(3):249 – 256.

Smith, B. e Gent, I. (2005). Constraint modelling challenge 2005. In *IJCAI 2005 Fifth Workshop on Modelling and Solving Problems with Constraints*, pp. 1–8.

Smith, B. M. (2003). Constraint programming in practice: Scheduling a rehearsal. *Re-search Report APES-67-2003, APES group.*

Yanasse, H. H. (1997). On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research*, 100(3):454 – 463.