LAYLA MIRANDA DA SILVEIRA

Orientador: Marco Antonio Moreira de Carvalho

UM ESTUDO SOBRE MÉTODOS HEURÍSTICOS APLICADOS AO SEQUENCIAMENTO DA PRODUÇÃO EM SISTEMAS DE MANUFATURA FLEXÍVEIS

Ouro Preto

Dezembro de 2019

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

UM ESTUDO SOBRE MÉTODOS HEURÍSTICOS APLICADOS AO SEQUENCIAMENTO DA PRODUÇÃO EM SISTEMAS DE MANUFATURA FLEXÍVEIS

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.

LAYLA MIRANDA DA SILVEIRA

Ouro Preto
Dezembro de 2019

S381e Silveira, Layla Miranda da.
Um estudo sobre métodos heurísticos aplicados ao sequenciamento da produção em sistemas de manufatura flexíveis [manuscrito] / Layla Miranda da Silveira. - 2019.

x, 43f.: il.: tabs.

Orientador: Prof. Dr. Marco Antônio Moreira de Carvalho.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação.

1. Problema do caixeiro viajante. 2. Otimização combinatória . 3. Pesquisa operacional. I. Carvalho, Marco Antônio Moreira de. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 004.023

Catalogação: ficha.sisbin@ufop.edu.br

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Um Estudo Sobre Métodos Heurísticos Aplicados ao Sequenciamento da Produção em Sistemas de Manufatura Flexíveis

LAYLA MIRANDA DA SILVEIRA

Monografia defendida e aprovada pela banca examinadora constituída por:

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

Dr. Joubert de Castro Lima
Universidade Federal de Ouro Preto

Dr. Túlio Ângelo Machado Toffolo
Universidade Federal de Ouro Preto

Ouro Preto, Dezembro de 2019

Resumo

O Problema de Minimização de Trocas de Ferramentas (SSP, do inglês *Job Sequencing and Tool Switching Problem*) consiste em determinar uma ordem para o processamento de tarefas em uma máquina flexível. Cada tarefa requer um subconjunto de ferramentas específico dentre todas as que estão disponíveis para utilização. A máquina flexível é capaz de armazenar as ferramentas requisitadas por uma tarefa, mas não tem capacidade suficiente para armazenar todas as ferramentas disponíveis simultaneamente. Devido a esta restrição de capacidade, pode ser necessário efetuar trocas de ferramentas entre tarefas, que exigem a paralisação da linha de produção. Desta forma, sequenciar as tarefas de forma otimizada leva a um menor número de trocas de ferramentas e consequentemente, menor tempo ocioso da linha de produção. O SSP pertence ao conjunto de problemas classificados como $\mathcal{NP}$-Difícil. Neste trabalho, o SSP é modelado como Problema do Caixeiro Viajante Generalizado (GTSP, do inglês *Generalized Traveling Salesman Problem*). Deseja-se solucionar tal modelagem por meio dos métodos heurísticos busca local iterada e *Generalized Lin-Kernighan Helsgaun*. Os resultados obtidos demonstram que a modelagem é promissora, embora esteja limitada a solucionar instâncias cujos arquivos de entrada devidamente transformados tenham tamanhos compatíveis com a memória primária disponível. Esta limitação se deve à relação direta entre o número de vértices gerados e o tamanho dos arquivos de entrada gerados para os métodos heurísticos. Para as instâncias que foram devidamente solucionadas, foi possível encontrar soluções que distam percentualmente das melhores soluções conhecidas entre 0 e 10%.

Abstract

The *Job Sequencing and Tool Switching Problem* (SSP) consists of finding the processing order of jobs in a flexible machine. Each job requires a specific subset of tools from the set of all available tools. The flexible machine can store all tools required to process a job, but cannot store all available tools simultaneously. Owing to this capacity constraint, it may be necessary to switch tools between the processing of a pair of consecutive jobs. In order to switch tools, the machine must be turned off, which causes idle production time. Thus, scheduling jobs in an optimized way may lead to less tool switches and a shorter idle times. The SSP belongs to the $\mathcal{NP}$-Hard class of problems. This study models the SSP as a *Generalized Traveling Salesman Problem* (GTSP). The resulting problem will be solved using the *Iterated Local Search* (ILS) and *Generalized Lin-Kernighan-Helsgaun* (GLKH). The results show that this modeling is viable, though it is limited to solve instances whose transformed input files have sizes compatible with the primary memory available. The limitation is due to the direct relation between the number of vertexes generated and the size of the input files created for the heuristic methods. For the instances that were properly solved, it was possible to find solutions identical to the best know solutions or at most 10% worse.

Dedico este trabalho a minha família por todo o apoio e carinho ao longo destes anos. Palavras não são suficientes para dimensionar o amor e gratidão que sinto.

Agradecimentos

Agradeço a minha mãe, Sonia, por toda compreensão, todos os conselhos e todo o esforço que faz diariamente para me ajudar a alcançar meus sonhos. Esta conquista é definitivamente por você e para você.

Agradeço ao meu avô Dionísio por me lembrar todos os dias de que mesmo geograficamente distantes, um estava sempre cuidando do outro.

Aos melhores amigos que eu poderia ter encontrado no bacharelado, Palloma, Bruno e Marco, meu modesto muito obrigada. Vocês ajudaram e muito a tornar estes quatro anos mais leves!

A minha amada república Peça Rara por me mostrar que sim, todos temos nosso lugar em Ouro Preto. E que não precisamos mudar nossa personalidade nem nosso caráter para pertencer a qualquer lugar.

Ao DECOM, especialmente aos professores Guilherme, Marcelo e Saul. Cada um de vocês me ensinou um pouco sobre computação e muito sobre o que é ser um bom profissional.

Ao professor Joubert, pelos vários conselhos que me deu ao longo da graduação. Alguns destes conselhos me fizeram permanecer firme e forte até o final deste curso.

De forma alguma eu poderia esquecer de agradecer ao meu orientador, Marco. Gostaria de lhe dizer $\Theta(n!)$ obrigadas, para um n que tende ao infinito, por acreditar no meu potencial e aceitar me orientar. Por ser um professor comprometido com o ensino e ter plantado a semente do gosto pela pesquisa em minha vida.

Por fim, agradeço ao CNPq pela bolsa de Iniciação Científica que tive. Esta bolsa me possibilitou pesquisar boa parte do conteúdo contido neste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Siglas

ATSP *Asymmetric Traveling Salesman Problem*. vii, 23, 29–31, 37

FMS *Flexible Manufacturing System*. 1, 3, 5

GLKH *Generalized Lin-Kernighan-Helsgaun*. ii, 2, 3, 14, 23, 24, 32–34, 36–38

GTSP *Generalized Traveling Salesman Problem*. i, ii, vii, viii, 2, 9, 18–21, 23, 25–32, 34, 37, 38

ILS *Iterated Local Search*. ii, 2, 3, 7, 14, 21, 22, 34, 36–38

KTNS *Keep Tool Needed Soonest*. 2, 4, 5, 7, 8, 10, 12–14

LKH *Lin-Kernighan-Helsgaun*. 2, 23, 24, 33, 37

SSP *Job Sequencing and Tool Switching Problem*. i, ii, vii, viii, 2–16, 25–28, 31, 34, 38

TSP *Traveling Salesman Problem*. vii, viii, 4, 8–13, 17–21, 23, 25

Capítulo 1

Introdução

No século XX surgiu o conceito de sistema de manufatura flexível, ou FMS, do inglês *Flexible Manufacturing System* (Sethi e Sethi, 1990). Trata-se de um sistema produtivo que permite que fábricas sejam capazes de aumentar a gama de produtos disponibilizados para comercialização, sem a necessidade de linhas de produção adicionais. Este sistema se contrapõe ao modelo de linhas de produção dedicadas, que são capazes de produzir apenas um tipo de produto por linha, devido ao posicionamento fixo das máquinas dedicadas. De forma breve, um FMS corresponde a um conjunto de máquinas interligadas capazes de serem adaptadas às necessidades de produção. Consequentemente, esta adaptabilidade favorece a flexibilização da produção.

Uma máquina integrante de um FMS é denominada *máquina flexível* e é capaz de realizar diferentes operações sobre a matéria-prima ou produtos semiacabados (Crama, 1997). Estas operações são genericamente denominadas *tarefas*. Diferentes tarefas podem ser *processadas* em uma máquina flexível de acordo com diferentes configurações da mesma. Uma *configuração* é realizada através da inserção das ferramentas necessárias para se processar uma tarefa em um compartimento específico da máquina, chamado *magazine*. Uma *ferramenta* pode ser uma lâmina de corte, uma broca de perfuração, uma politriz, etc. e ocupa apenas um espaço do *magazine* chamado *slot*, cujos tamanhos são considerados uniformes.

É importante destacar que o *magazine* apresenta capacidade limitada, suficiente para armazenar as ferramentas necessárias para processar uma tarefa isoladamente, mas não para armazenar todas as ferramentas existentes simultaneamente. Portanto, entre o processamento de diferentes tarefas pode ser necessário remover uma ferramenta presente no *magazine* para inserir outra que seja necessária ao processamento da próxima tarefa. Este processo caracteriza uma *troca de ferramentas* e para sua realização a produção deve ser paralisada, gerando ociosidade.

No contexto genérico de um FMS, não se aplicam conceitos de precedência ou prioridade entre as tarefas. Desta forma, surge a importância de determinar uma ordem para o processamento visando a conclusão de todas as tarefas ao longo da produção. Sequenciar as tarefas

implica em também determinar em quais momentos interromper a produção para efetuar as trocas de ferramentas. A determinação destes dois itens é representada pelo *plano de produção*. Em vista disto, definir um plano de produção otimizado pode minimizar o número de trocas de ferramentas e consequentemente, reduzir o tempo ocioso das linhas de produção.

Surge então o Problema de Minimização de Trocas de Ferramentas (SSP, do inglês *Job Sequencing and Tool Switching Problem*). De acordo com a literatura, este problema pode ser dividido em duas partes:

1. Determinar o sequenciamento das tarefas para formar o plano de produção;
2. Determinar quando efetuar as trocas de ferramentas, dada uma sequência fixa de tarefas.

O problema mencionado no item 1 é o objeto de estudo deste trabalho. Existem diversas versões do SSP abordadas na área de Pesquisa Operacional. Este trabalho aborda a versão que busca determinar o sequenciamento de tarefas em uma única máquina. Considera-se também uma abstração da realidade, na qual as ferramentas possuem tamanho e custo de inserção e remoção uniformes, não sofrem desgaste, nem quebram. Tendo em vista tais especificidades para o problema, esta versão é encontrada na literatura como SSP *uniforme*.

Para o problema apresentado no item 2 encontra-se na literatura uma solução exata, em tempo determinístico polinomial, amplamente adotada. Tang e Denardo (1988) introduziram a política KTNS (do inglês, *Keep Tool Needed Soonest*) que determina o melhor plano possível de trocas de ferramentas para uma única máquina, dada uma sequência fixa de tarefas. Brevemente, o KTNS busca manter na máquina as ferramentas que serão necessárias mais cedo para executar as próximas tarefas do plano de produção.

Com o objetivo de abordar o SSP é utilizada uma modelagem pouco explorada na literatura, que resulta em uma transformação para o Problema do Caixeiro Viajante Generalizado (GTSP, do inglês *Generalized Traveling Salesman Problem*). Para solucionar o problema resultante são utilizadas abordagens heurísticas da literatura, incluindo uma implementação de *busca local iterada* (ILS, do inglês *Iterated Local Search*) e uma versão da heurística *Lin-Kernighan-Helsgaun* direcionada ao GTSP, chamada GLKH. A referida modelagem consta na literatura sobre o SSP, entretanto, não há registro de sua implementação e solução efetiva.

1.1 Motivação

O SSP é classificado como um problema $\mathcal{NP}$-Difícil, ou seja, ainda não se conhecem métodos para solucioná-lo em tempo determinístico polinomial, em máquinas determinísticas. Tal demonstração se encontra em Crama et al. (1994). Por se tratar de um problema de alta complexidade computacional e de relevância para o setor industrial, é pertinente investigar o problema, objetivando-se encontrar soluções de qualidade.

Ao abordar o conceito de FMS, com foco nos sistemas que utilizam de máquinas flexíveis, Crama (1997) ressalta a importância da utilização de tais sistemas em indústrias. O autor aponta que sequenciar de forma otimizada as tarefas, implica na diminuição no número de trocas das ferramentas e consequentemente, aumenta o tempo produtivo das máquinas flexíveis. Espera-se que, ao produzir por mais tempo, seja possível finalizar mais produtos dentro dos prazos determinados e também reduzir os custos de produção.

1.2 Objetivos

Este trabalho tem como objetivo geral abordar o problema de sequenciamento enunciado pelo SSP de forma heurística. Há também o objetivo de comparar os resultados obtidos com os existentes na literatura de forma a determinar se a modelagem utilizada é promissora. Os objetivos específicos deste trabalho são:

1. Avaliar a modelagem do SSP por meio do Problema do Caixeiro Viajante Generalizado;
2. Utilizar de implementações das heurísticas ILS e GLKH que constam na literatura, para abordar o problema resultante da modelagem proposta;
3. Comparar os resultados obtidos com os melhores resultados conhecidos ao solucionar instâncias pertencentes à literatura para o problema.

1.3 Organização do Trabalho

Este trabalho está dividido em 6 capítulos. O Capítulo 2 apresenta uma revisão da literatura abordando SSP. O Capítulo 3 contém a fundamentação teórica do problema abordado, explicita a versão em estudo, fornecende o embasamento necessário sobre a modelagem empregada e descreve os métodos heurísticos de solução utilizados. O Capítulo 4 descreve como o problema foi abordado e quais as estratégias de solução aplicadas. O Capítulo 5 apresenta os resultados obtidos através dos testes realizados sobre a modelagem proposta. Por fim, o Capítulo 6 abrange as conclusões obtidas através deste trabalho.

Capítulo 2

Revisão da Literatura

O Problema de Minimização de Trocas de Ferramentas (SSP) é abordado por muitos autores na área de Pesquisa Operacional. Este capítulo apresenta uma breve revisão de trabalhos anteriores sobre o problema. Para uma análise mais profunda sobre o tema deste trabalho e problemas correlatos, recomenda-se consultar a revisão sistemática de Calmels (2018).

Este capítulo foi dividido em 5 seções. A Seção 2.1 contém os trabalhos que caracterizam a base teórica do SSP. A Seção 2.2 descreve como o SSP foi abordado na literatura por meio de métodos exatos e heurísticos. A Seção 2.3 agrupa trabalhos que modelam e resolvem o SSP como o Problema do Caixeiro Viajante (ou TSP, do inglês *Traveling Salesman Problem*). As Seções 2.4 e 2.5 apresentam, respectivamente, aplicações reais do problema, por exemplo na indústria, e variações do problema original.

2.1 Trabalhos seminais

Uma das primeiras contribuições sobre o SSP na literatura foi apresentada por Tang e Denardo (1988). Este foi também o primeiro trabalho a modelar e resolver o SSP como TSP. Esta modelagem se tornou recorrente em trabalhos subsequentes e está descrita em detalhes na Seção 2.3. Com o objetivo de determinar a ordem ótima de trocas das ferramentas, dada uma sequência fixa de tarefas, é introduzida a política KTNS. Esta política analisa uma dada sequência de tarefas e, ao concluir o processamento de uma tarefa específica, mas antes de se iniciar o da tarefa seguinte, analisa o conjunto de ferramentas presentes na máquina. Caso seja necessário remover alguma ferramenta já equipada, opta-se por manter na máquina aquelas que serão necessárias mais cedo dentro do plano de produção. Esta política determina o plano ótimo de trocas de ferramentas em tempo determinístico polinomial.

Adiante, Crama et al. (1994) provam que o SSP pertence à classe $\mathcal{NP}$-Difícil, para qualquer capacidade do magazine maior ou igual a duas ferramentas. Tal complexidade independe da quantidade total de ferramentas utilizadas pelas tarefas. Esta prova é feita baseada nas afirmações apresentadas por Tang e Denardo (1988).

Dentro do escopo do problema abordado, Crama (1997) apresenta formalmente a definição do conceito de Sistema de Manufatura Flexível. Brevemente, um FMS corresponde a um conjunto de máquinas que podem ser configuradas de acordo com as tarefas que devem processar. Baseado neste conceito, o autor expõe que, determinar a melhor forma de uso deste sistema, buscando maximizar a produtividade, é um problema de grande relevância para o setor industrial.

2.2 Métodos exatos e heurísticos

Dado o problema de determinar uma ordem de processamento das tarefas que minimize o número total de trocas de ferramentas, Bard (1988) apresenta uma implementação de *branch-and-bound*. O método resolve uma relaxação lagrangiana proposta para o problema, por um determinado número de iterações. Em seguida, é utilizada a política KTNS para determinar o plano ótimo de trocas de ferramentas para a sequência encontrada. A partir da solução encontrada pelo *branch-and-bound*, o método proposto intensifica as buscas na vizinhança da solução. Para isso, são feitas trocas entre duas tarefas, de posições não necessariamente contíguas, na solução corrente. Este procedimento resulta em uma solução ótima local para o problema, devido à limitação no número de iterações da execução do *branch-and-bound*.

Em uma abordagem metaheurística, Al-Fawzan e Al-Sultan (2003) encontram soluções para o SSP por meio de uma implementação de busca tabu. A lista tabu é associada a uma memória de longo prazo, que atribui uma frequência para cada movimento. Os movimentos utilizados são deslocar um bloco de tarefas e trocar de posição duas tarefas. Se o melhor movimento de uma iteração estiver na lista tabu, sua frequência é atualizada e a função objetivo reavaliada, aplicando uma penalização. Somente o melhor movimento não tabu de uma iteração será inserido na lista. A implementação apresenta também uma memória de curto prazo para armazenar a melhor solução até a iteração atual. Por fim, o método utiliza o KTNS como função de avaliação para obter o número de trocas de ferramentas.

Zhou et al. (2005) apresentam uma implementação de *beam search*. Nesta implementação, cada nó interno da árvore de busca corresponde a um sequenciamento parcial para as tarefas e consequentemente, os nós folhas representam uma solução factível completa. Ao percorrer a árvore de busca, o método explora a árvore em largura e avalia cada nó de acordo com duas funções de avaliação. A primeira função verifica quantas ferramentas existem em comum entre a última tarefa inserida na solução e as tarefas candidatas a inserção, escolhendo-se o nó com maior número de ferramentas em comum. Caso o resultado após o uso da primeira função seja de apenas uma ferramenta em comum, utiliza-se a segunda função. Esta função efetua a união dos conjuntos de ferramentas da última tarefa inserida na solução e das tarefas candidatas a inserção restantes e verifica quantas trocas de ferramentas serão necessárias. Por fim, é selecionado o nó mais promissor e realizada uma poda na árvore.

O uso de busca tabu ocorre novamente em Konak et al. (2008), em duas implementações diferentes. Cada tarefa é tratada como um instante s_i da produção e deseja-se agrupar as tarefas de forma a diminuir a cardinalidade de S, o plano de produção. A primeira implementação seleciona duas tarefas, i e j e tenta inserir j no mesmo instante de i. Caso não ocorra nenhuma troca de ferramentas entre o processamento de ambas as tarefas, estas são agrupadas no mesmo instante e o movimento inserido na lista tabu. A segunda implementação corresponde a uma busca tabu convencional sem critérios de aspiração ou retroalimentação da lista tabu. Ambas implementações produziram melhorias nos resultados conhecidos à época. Desta forma, os autores não destacam uma implementação específica como a melhor.

Amaya et al. (2008), Amaya et al. (2011) e Amaya et al. (2013) utilizam de algoritmos meméticos, que são uma combinação de algoritmo genético a um ou mais métodos de busca local, para solucionar o SSP. Nos três trabalhos os melhores indivíduos gerados na iteração substituem os piores indivíduos da população. A mutação altera um bloco de genes do indivíduo e a seleção dos indivíduos para recombinação é feita através de torneio binário. Amaya et al. (2008) utiliza a heurística de *descida rápida*. A seleção é feita através da escolha dos dois melhores indivíduos, a recombinação através da alternância na escolha dos genes dos melhores indivíduos. O cruzamento é feito através do operador OX e da escolha dos genes através de alternância entre pais. Amaya et al. (2011) utiliza a *descida com escolha do melhor vizinho* e uma implementação de busca tabu semelhante a de Al-Fawzan e Al-Sultan (2003). O método alterna a utilização destas heurísticas após um determinado número de iterações sem melhora. A recombinação de indivíduos é feito pelo operador APX. Por último, Amaya et al. (2013) difere de Amaya et al. (2011) apenas na utilização de entropia cruzada para a geração das populações em cada iteração e na utilização de um operador de recombinação que consiste em uma função probabilística para a escolha de cada gene a ser atribuído ao indivíduo filho.

Senne e Yanasse (2009); Yanasse et al. (2009) abordam o SSP lidando com subproblemas menores e inserindo tarefas na solução construtivamente. Para isso, os autores consideram uma sequência de tamanho k, tal que k seja menor que o tamanho do conjunto de tarefas, que está ordenada sob o critério de gerar o menor número de trocas de ferramentas. Em seguida, é inserida nesta sequência uma tarefa j na posição que resulta no menor número possível de trocas. O limite superior do número de trocas é obtido através de uma implementação de *beam search* com três critérios diferentes para o armazenamento da árvore na memória, sendo eles, manter apenas os melhores ramos, manter s ramos que resultem na menor quantidade de trocas de ferramentas ou manter no máximo t nós.

Chaves et al. (2012) buscam solucionar o SSP com uma modelagem em grafos na qual os vértices representam as ferramentas. Uma aresta $\{i, j\}$ indica que um par de ferramentas é utilizado de forma simultânea no processamento de uma tarefa. Rótulos nas arestas indicam as tarefas, podendo haver arestas paralelas. As arestas não apresentam pesos. O método proposto apresenta duas fases. A fase construtiva da solução busca sequenciar as tarefas

através da remoção de arestas do grafo. Para isso, escolhe-se o vértice de menor grau, maior que zero, e escolhe-se uma de suas arestas. O rótulo da aresta indica qual tarefa deverá ser processada e os demais arcos com este mesmo rótulo são removidos do grafo. Este processo se repete até se processar todas as tarefas. A segunda fase, chamada de refinamento, utiliza de uma implementação de ILS partindo da solução da fase anterior. O movimento utilizado troca a posição de duas tarefas dentro de uma solução.

Uma alternativa ao KTNS é apresentada por Adjiashvili et al. (2015). Além de resolver o problema de determinar a quantidade e os instantes da produção no qual ocorrem as trocas de ferramentas, dada uma sequência fixa de tarefas, este método também é capaz de resolver o problema para uma sequência variável de tarefas. O método apresentado busca minimizar os instantes de paradas de máquina, haja vista que considera que toda troca de ferramenta demanda que a linha de produção seja paralisada.

Novos modelos de programação linear inteira são apresentados por Catanzaro et al. (2015). A relaxação e os modelos propostos são baseados nos apresentados por Laporte et al. (2004). Porém, as novas restrições e as novas relaxações propostas pelos autores geram melhorias nas soluções encontradas quando comparadas aos modelos predecessores.

Chaves et al. (2016) combinam a heurística *biased random key genetic algorithm* (BRKGA) à busca em *clusters*. Para isso, um conjunto de soluções é gerado e cada solução é utilizada como ponto central de um *cluster*. Em seguida, um novo conjunto de soluções é gerado através do BRKGA. Cada solução é inserida em um *cluster*, baseado em uma métrica, como a quantidade máxima de movimentos necessária para transformar a solução no ponto central de um *cluster*. Por fim, é feita uma busca nos *clusters* para encontrar aquele que recebeu mais soluções naquela iteração e aplicada a heurística de *descida em vizinhança variável* no ponto central. A heurística utiliza movimentos de *shift* de uma ou duas posições de uma tarefa ou a troca em pares de uma ou duas tarefas de posição na solução.

O estado da arte em métodos heurísticos para o SSP corresponde ao trabalho apresentado por Paiva e Carvalho (2017). Este trabalho apresenta uma nova modelagem em grafos para o problema. Tal modelagem relaciona os vértices com as ferramentas, inovando na forma de determinar o peso para as arestas. As arestas ocorrem entre pares de ferramentas e possuem o peso determinado como a frequência com que aquele par de ferramentas é requisitado por diferentes tarefas na entrada do problema. Após modelar a entrada como este grafo, é aplicada uma implementação de busca em largura para determinar uma ordem de disponibilização das ferramentas. A partir desta sequência de ferramentas, é realizado um sequenciamento guloso das tarefas. Disponibiliza-se uma ferramenta a cada instante; se existir tarefa capaz de ser processada com as ferramentas disponíveis até determinado instante, então esta é inserida na solução. Por fim, os autores apresentam uma implementação de ILS que entre as buscas locais, utiliza um método de busca inédito, que percorre a matriz binária de permutação buscando agrupar blocos consecutivos de uns.

2.3 Modelagens através do problema do caixeiro viajante

Na literatura foi possível encontrar uma grande quantidade de trabalhos que buscam resolver o SSP através de uma transformação deste problema no TSP. Esta modelagem foi encontrada pela primeira vez, nesta revisão, em Tang e Denardo (1988). Frequentemente os vértices do grafo representam as tarefas que devem ser processadas e as arestas indicam sequenciamento das tarefas, ou seja, se uma tarefa i é executada imediatamente após uma tarefa j. O grafo desta modelagem é completo. Desse modo, deseja-se encontrar um caminho ou um ciclo hamiltoniano no grafo, para contemplar o processamento de todas as tarefas. Os trabalhos que aqui foram agrupados divergem na forma de encontrar este caminho e/ou na maneira de determinar os pesos das arestas do grafo.

O trabalho apresentado por Hertz et al. (1998) agrupa as diferentes métricas para determinação dos pesos das arestas mais frequentes na literatura. As métricas são aqui apresentadas de forma enumerada, para posterior referência. Adicionalmente, os autores introduziram duas novas métricas, correspondentes às métricas 4 e 5, que não foram amplamente utilizadas na literatura em trabalhos posteriores. Sejam c a capacidade do magazine, i e j tarefas quaisquer, T_i o conjunto de ferramentas requerido pela tarefa i, n a quantidade de tarefas a serem processadas, $\Lambda(i,j)$ o número de tarefas diferentes de i e de j que precisam das ferramentas que pertencem a $|T_i \cup T_j|$ e θ um parâmetro entre $[0,1]$. As métricas para determinação do peso das arestas são:

1. $c - |T_i \cap T_j|$;
2. $|T_i \cup T_j| - |T_i \cap T_j|$;
3. $\max\{0, |T_i \cup T_j| - c\}$;
4. $\max\left\{0, |T_i \cup T_j| - \left[\theta\left(\frac{\Lambda(i,j)}{(n-2)|T_i \cup T_j|}\right]\right]\right\}$
5. $\left(\left[\frac{c+1}{c}\right]|T_i \cup T_j| - |T_i \cap T_j|\right)\left[\frac{(n-2)|T_i \cup T_j|}{\max\{\Lambda(i,j), 0.5\}}\right]$

Ao agrupar estas métricas, os autores realizaram experimentos aplicando as cinco métricas em três heurísticas diferentes populares para o TSP. Para isso foram utilizadas implementações da heurística de inserção da aresta de menor custo e do algoritmo *GENI* de Gendreau et al. (1992). Também foi implementado um algoritmo que se assemelha à inserção da aresta de menor custo, mas que computa o custo de tal inserção utilizando o KTNS. A partir dos resultados que foram obtidos, os autores concluem que a métrica 3 apresenta os melhores resultados.

Em Crama et al. (1994), são apresentadas duas possíveis modelagens para o grafo correspondente ao problema. Na primeira, busca-se encontrar um caminho hamiltoniano de custo mínimo. O peso atribuído às arestas é dado pela métrica 3. Esta métrica busca determinar

um limitante inferior para o número de trocas de ferramentas e previne a obtenção de valores negativos. Para a segunda modelagem, ocorre a inserção de dois vértices artificiais para indicar início e fim do processamento das tarefas e o grafo resultante é direcionado. Nesta modelagem deseja-se encontrar um ciclo hamiltoniano de custo mínimo. O peso dos arcos é dado pelo limite superior UB $= |T_i \setminus T_j|$. Para solucionar o problema, ambas as modelagens utilizam de heurísticas construtivas ou de estratégias de melhoria da solução tradicionais para o TSP.

Djellab et al. (2000) modelam o SSP por meio de um hipergrafo, um tipo de grafo no qual uma aresta pode ligar dois ou mais vértices de forma simultânea. Nesta modelagem, as tarefas que devem ser processadas são representadas como os vértices. Já as hiperarestas relacionam as tarefas que necessitam de uma mesma ferramenta durante o processamento. Assim, o peso de uma hiperaresta representa o custo de se trocar determinada ferramenta. Este peso é dado pela cardinalidade da diferença entre os conjuntos de ferramentas requisitados pelas tarefas associadas àquela hiperaresta. Este cálculo é similar à métrica 3 descrita em Hertz et al. (1998). Para solucionar o problema, utiliza-se uma heurística gulosa, de inserção da hiperesta de menor custo no caminho hamiltoniano. Por fim, são efetuadas trocas nas posições dos vértices na solução buscando minimizar o custo total.

Em seguida, Shirazi e Frizelle (2001) investigam se a utilização de heurísticas poderia gerar melhorias nos resultados obtidos pela indústria. Para isso, utilizam seis métodos heurísticos gulosos frequentes na literatura para resolver o TSP, como a inserção da aresta de menor custo, início múltiplo e inserção mais distante, por exemplo. Como esperado, ao aplicar as heurísticas sobre instâncias disponibilizadas pela indústria, foram obtidas melhorias expressivas nos resultados, quando comparados aos das técnicas populares no setor industrial.

Como exemplo de pesquisa de solução através de métodos exatos, Laporte et al. (2004) aborda o SSP utilizando dois métodos de programação linear inteira para solucionar o TSP. Para viabilizar esta modelagem, são inseridos dois vértices artificiais que sinalizam início e fim do plano de produção. Cada implementação proposta utiliza de uma métrica para o peso das arestas. Na implementação de *branch-and-cut* o peso da aresta $\{i, j\}$ é igual a 1 se, e somente se, uma tarefa j for executada imediatamente após a tarefa i e 0 em caso contrário. Já na implementação de *branch-and-bound*, o peso da aresta é dado pela métrica 3 de Hertz et al. (1998).

Com o objetivo de explorar outras modelagens Yanasse e Lamosa (2006a) e Yanasse e Lamosa (2006b) apresentam a possibilidade de se modelar o SSP como um GTSP. Para isso, cada vértice do grafo corresponde a uma configuração do *magazine* da máquina. As tarefas são representadas por *clusters* que agrupam todos os vértices que são capazes de processar uma tarefa. Em seguida, são propostas, porém não implementadas, várias possibilidades de exploração desta modelagem como efetuar reduções ao TSP e resolvê-lo de maneira heurística ou utilizar de métodos exatos para resolver o GTSP correspondente.

Ghiani et al. (2007) apresentam uma nova formulação para resolver o SSP de forma exata. Esta formulação considera que analisar o plano de produção de forma direta ou reversa implica no mesmo número de trocas de ferramentas. A partir desta perspectiva, apresentam melhorias no *branch-and-bound* apresentado por Laporte et al. (2004). Além da formulação, o grafo resultado deste trabalho é direcionado e o peso do arco (i, j) é 1 caso a tarefa j seja executada imediatamente após a tarefa i e zero em caso contrário.

Adiante Ghiani et al. (2010) reduzem o SSP ao problema de encontrar um ciclo hamiltoniano de custo mínimo. Buscando viabilizar a modelagem, um vértice artificial é inserido no grafo. O custo de uma aresta $\{i, j\}$ é calculado pelo número de trocas de ferramentas feitas para se sair do vértice artificial e chegar em i, acrescido das trocas de i para j. Este cálculo é feito através de sucessivas chamadas ao KTNS. A partir deste grafo, é utilizada uma implementação de *branch-and-cut*.

Azevedo e Carvalho (2017) modelam o SSP como TSP, utilizando as métricas descritas em Hertz et al. (1998). Para resolver o TSP, foi utilizado o resolvedor exato *concorde*, que foi capaz de solucionar todas as instâncias em tempo computacional desprezível. Porém, ao comparar os resultados obtidos com aqueles apresentados no estado da arte, de Paiva e Carvalho (2017), constatou-se que os resultados obtidos eram consideravelmente piores. Ressalta-se que a métrica 3, mais frequente na literatura, obteve os piores resultados nos experimentos. Assim, foi possível concluir que a modelagem do SSP como TSP, utilizando as métricas de Hertz et al. (1998), não é eficiente. A falha desta modelagem pode estar, além da não equivalência entre os problemas, nas métricas de peso para as arestas.

Contrariando a constatação apresentada pelo trabalho imediatamente acima, Ahmadi et al. (2018) utiliza da modelagem geral apresentada nesta seção. Para solucionar o TSP os autores apresentam uma implementação do algoritmo genético *q-learning*. A modelagem apresentada é chamada de TSP de segunda ordem, pois mede o número de trocas de ferramentas futuras considerando as tarefas i, j, k, sendo i a tarefa anterior já incluída no ciclo, j a tarefa a ser incluída e k a sua sucessora. Apesar de apresentar leves melhorias quando comparado ao trabalho que caracteriza o estado da arte (Paiva e Carvalho, 2017), os autores não deixam clara a métrica utilizada para definir o peso das arestas.

2.4 Aplicações práticas

Assim como é possível encontrar na literatura modelagens do SSP como TSP, há também a modelagem de outros problemas como o SSP. Esta seção é dedicada a mencionar alguns dos trabalhos que o fizeram, indicando em cada caso quais elementos são modelados como ferramentas e como tarefas.

Privault e Finke (2000) modelam o problema de k-servidores com requisições em massa como o SSP, no caso em que as tarefas não são totalmente conhecidas ao início da produção.

O problema de k-servidores consiste em posicionar k servidores, em um grafo de $n \geq k$ vértices com o objetivo de minimizar o custo de atender a demanda de um subconjuntos de vértices chamada de requisição em massa. Portanto, cada vértice corresponde a um cliente e uma demanda pode ser a solicitação de uso de rede, por exemplo. Para permitir a modelagem, as tarefas são relacionadas aos servidores e as ferramentas aos clientes. Desta forma, atender um conjunto de requisições corresponde a processar uma tarefa que exige um conjunto de ferramentas. Em seguida, é feita uma modelagem do SSP como o TSP, utilizando a métrica 3 citada na Seção 2.3. Por fim, uma implementação da heurística chamada *bulk-service partitioning* é utilizada para agrupar as tarefas que podem ser sequenciadas sem trocar ferramentas.

Denizel (2003) reduz o SSP ao Problema de Empacotamento com Compartilhamento. Este problema consiste em determinar como armazenar diversos itens em contêineres, tendo em vista que há itens que não podem ser armazenados juntos. Porém, armazenar dois itens que podem ser armazenados juntos no mesmo contêiner implica em custos menores que armazená-los separadamente. A redução é feita ao relacionar as tarefas aos itens e as ferramentas aos contêineres. Uma troca de ferramentas neste contexto implica em utilizar mais um contêiner para armazenar uma tarefa. A modelagem foi abordada por uma implementação de *branch-and-bound*.

Ghrayeb et al. (2003) e Salonen et al. (2006) buscam resolver em seus trabalhos o problema de montagem de circuitos impressos encapsulados, que consiste em sequenciar a ordem de utilização das fitas de componentes eletrônicos, de forma a produzir diferentes tipos de circuitos de forma simultânea com o menor número de trocas de componentes possível. Este problema se relaciona à versão do SSP que utiliza múltiplas máquinas, na qual cada ferramenta representa uma fita de componentes eletrônicos e as trocas de ferramentas indicam troca do componente eletrônico. As tarefas a serem processadas são relacionadas à confecção de um circuito completo por meio da modelagem via TSP por ambos os autores. A diferença entre estes trabalhos se encontra na métrica utilizada para determinar o peso da aresta. Ghrayeb et al. (2003) utiliza a métrica 3 e uma implementação mista que alia uma heurística à programação linear não inteira. Salonen et al. (2006) utiliza a métrica 2, que consiste em observar a quantidade de trocas de ferramentas entre as tarefas i e j e apresenta uma heurística que nomeou como *grouping with minimum setup algorithm*.

Passado algum tempo, a modelagem por SSP para solucionar outros problemas é utilizada por Marvizadeh e Choobineh (2013). Os autores apresentam o problema de estamparia de placas metálicas. O processo de estamparia consiste em moldar ou deformar chapas metálicas através do uso de prensas, geralmente a frio. A relação entre os problemas é feita ao vincular os moldes às ferramentas. Logo, o processamento de uma tarefa corresponde à obtenção da moldagem desejada em uma placa metálica. Após apresentarem a modelagem, os autores descrevem três heurísticas para solucionar o problema, sendo uma implementação de algoritmo genético, *set merging algorithm* e *randomized set merging algorithm*.

Burger et al. (2015) buscam minimizar o número de trocas de cartuchos de tinta utilizados para imprimir embalagens plásticas. Minimizar as trocas de cartuchos é importante pois ao trocar de uma cor para outra ocorrem variados custos de limpeza dos equipamentos. A transformação deste problema no SSP é feita associando-se os cartuchos de tinta às ferramentas e cada tarefa a uma embalagem completamente finalizada. É apresentada uma nova formulação de programação linear inteira tendo como base o trabalho de Tang e Denardo (1988).

Raduly-Baka e Nevalainen (2015) abordam novamente o problema de montagem de circuitos impressos, porém com a utilização de módulos para armazenamento e utilização dos componentes eletrônicos. A modelagem se assemelha à de Ghrayeb et al. (2003) e Salonen et al. (2006). A contribuição deste trabalho é manifestada através da prova de que agrupar as ferramentas em módulos, mesmo que exista uma quantidade fixa de módulos de capacidade fixa, não faz com que o problema deixe de ser classificado como $\mathcal{NP}$-Difícil. O método de solução utilizado é o *branch-and-bound*.

2.5 Variações do problema original

Privault e Finke (1995) abordam o SSP com tempos variáveis para trocas das ferramentas. Como contribuição é apresentado um método alternativo ao KTNS capaz de lidar com os tempos variáveis das trocas de ferramentas. Dada uma sequência fixa de tarefas, é criado um grafo no qual os vértices representam ferramentas e as arestas indicam as trocas da ferramenta i pela ferramenta j. O custo da aresta é dado pelo tempo de troca das ferramentas i e j. Então, o SSP é resolvido como o problema de determinar um fluxo máximo com custo mínimo.

Hertz e Widmer (1996) consideram o SSP em um contexto de múltiplas máquinas, cujo tempo de preparo e inserção das ferramentas pode ser variável. Neste trabalho foi utilizada modelagem através do TSP, aos moldes da versão geral apresentada na Seção 2.3. Para solucionar o problema, apresentaram-se duas implementações de métodos de solução. Quando o tempo de preparo e inserção é uniforme, utilizou-se uma implementação de busca tabu. Caso contrário, apresentou-se uma implementação modificada de busca tabu, chamada de *tomato* pelos autores. Sob o contexto de múltiplas máquinas apenas, mais informações podem ser encontradas em outros trabalhos (Fathi e Barnette, 2002; Beezão et al., 2017).

Sem apresentar testes computacionais, Avci e Akturk (1996) demonstram com exemplos numéricos uma maneira de abordar o SSP com restrições sobre as ferramentas, como tempo de vida limitado. Para isso, os autores apresentam um modelo matemático que permite criar cópias virtuais das ferramentas. Estas cópias virtuais podem ser normalmente inseridas no magazine, viabilizando a utilização de um tipo de ferramenta, mesmo após o fim da vida útil de uma unidade de ferramenta daquele tipo.

Em uma abordagem mais realista do problema, Matzliach e Tzur (1998) abordam o SSP para ferramentas de tamanho não uniforme e em um ambiente no qual não se conhecem todas

as tarefas antes do início dos processamentos. Para resolver esta versão do SSP, foram desenvolvidas duas heurísticas. A primeira, denominada *weighted backward distance*, tem como ponto principal a análise de quantos estágios da produção cada ferramenta permanece no magazine. A segunda, *weighted probabilistic*, analisa quantos estágios da produção a ferramenta está sendo efetivamente utilizada quando está no magazine.

Song e Hwang (2002) acrescentam ao SSP uniforme a necessidade de buscar ou levar as ferramentas para uma área de armazenamento. O veículo que efetua este transporte tem capacidade limitada, um fator restritivo. Assim, os autores abordam o problema de determinar o plano ótimo de trocas das ferramentas, minimizando o número de viagens feitas com o veículo. Para isso, é apresentada uma nova função objetivo que busca minimizar o número de viagens do veículo. Por fim a política de trocas, não nomeada, é comparada com o KTNS após a aplicação de uma implementação de início múltiplo guloso. Esta comparação prova que o KTNS não é ótimo para esta versão do problema.

Posteriormente, Tzur e Altman (2004) retomam as pesquisas que abordam tamanho variável das ferramentas. O trabalho busca resolver o problema de determinar o posicionamento das ferramentas no magazine, de forma a tentar minimizar o número de trocas. Para isso, é apresentada a heurística chamada *aladdin*. Esta heurística utiliza uma versão modificada do KTNS, que além de manter as ferramentas requisitadas mais cedo, opta por manter as que ocupam menos espaço também. A heurística também utiliza da modelagem geral por meio do TSP, descrita na Seção 2.3.

Em Konak e Kulturel-Konak (2007), deseja-se minimizar os instantes de parada de máquina para troca de ferramentas. Modela-se o problema através do TSP, nos moldes da Seção 2.3. O peso das arestas é dado de forma semelhante à métrica 3, sendo $C + 1$ acrescido à cardinalidade da união dos conjuntos de todas as ferramentas das tarefas pertencentes ao caminho percorrido até o instante j atual. Como método de solução é apresentada uma implementação da metaheurística otimização de colônia de formigas. O conceito de feromônio é utilizado para dar prioridade ao quanto é desejável executar uma tarefa j após a tarefa i.

Ainda no escopo do último trabalho mencionado, Crama et al. (2007) apresenta uma prova de que o SSP para tamanho variável das ferramentas é $\mathcal{NP}$-Difícil. Esta prova é válida tanto para custo variável ou constante de troca das ferramentas. Porém, se a capacidade do magazine for fixada, este problema pode ser solucionado em tempo polinomial.

Zeballos (2010) aborda o SSP através de *constraint programming*, ou seja, mantém seu foco sobre o conjunto de restrições do problema. São acrescentadas restrições de forma a ser possível processar as tarefas em mais de uma máquina, de forma paralela. Para percorrer a árvore de busca utiliza-se a conhecida *busca em largura*.

O último exemplo desta seção, faz menção ao trabalho de Furrer e Mütze (2017). Este trabalho apresenta um *framework* que pode ser configurado para solucionar o SSP sob diferentes óticas, como múltiplas máquinas ou tempo váriavel de preparo das ferramentas.

Capítulo 3

Fundamentação Teórica

Este capítulo contém a descrição formal do SSP. Adicionalmente, com o objetivo de criar a base teórica utilizada pela modelagem adotada neste trabalho, descreve-se formalmente o Problema do Caixeiro Viajante em suas versões tradicional e generalizada. Ao final do Capítulo é apresentada uma descrição do ILS e do GLKH utilizados para gerar os resultados obtidos através da solução da modelagem proposta.

3.1 Problema de Minimização de Trocas de Ferramentas

O SSP uniforme é composto pelo conjunto de tarefas $T = \{1, ..., n\}$ que devem ser processadas em uma única máquina e o conjunto $F = \{1, ..., m\}$ de todas as ferramentas disponíveis para uso durante a produção. Cada tarefa $j \in T$ exige um subconjunto $F_j \subseteq F$ de ferramentas equipadas na máquina, que pode armazenar de forma simultânea C ferramentas, tal que $C \leq m$. Uma solução para o SSP corresponde a uma permutação π dos elementos do conjunto T, que indica a ordem de processamento das tarefas, cada uma em um estágio da produção. A partir de π é possível aplicar a política KTNS para determinar quantas e quais trocas de ferramentas serão necessárias para se obter o plano de produção.

Uma entrada para o SSP pode ser representada por meio de dados como os apresentados na Tabela 3.1. Nesta tabela, representa-se uma instância com 5 tarefas e 6 ferramentas. É possível identificar que a tarefa 1 requer que as ferramentas 3, 5 e 6 sejam equipadas na máquina antes do seu processamento. A tarefa 2 requer as ferramentas 2, 3 e 4 e assim em diante. É possível observar também a limitação da capacidade do *magazine* em até 3 ferramentas equipadas de forma simultânea.

O exemplo contido na Tabela 3.1 pode ser convertido em uma matriz binária A, cujas colunas representam as n tarefas e as linhas representam as m ferramentas. Cada elemento a_{ij} da matriz assumirá o valor 1 se a ferramenta i estiver equipada na máquina durante o processamento da tarefa j e 0 caso contrário. A Tabela 3.2 apresenta a matriz correspondente ao exemplo anterior.

Tabela 3.1: Exemplo de instância do SSP.

Tarefas	1	2	3	4	5
Ferramentas	3	2	1	4	1
	5	3	4	6	2
	6	4	5		4
Capacidade do *magazine: 3*					

Tabela 3.2: Matriz binária correspondente a instância do SSP.

Ferramentas/Tarefas	1	2	3	4	5
1	0	0	1	*1*	1
2	0	1	0	0	1
3	1	1	0	0	0
4	0	1	1	1	1
5	1	0	1	0	0
6	1	0	0	1	0
Capacidade do *magazine: 3*					

A representação por matriz binária também pode ser utilizada como forma de representação de um plano de produção, desde que as colunas obedeçam a ordem estabelecida em uma solução π. Desta forma, obtém-se uma matriz A^{π}. Interpretando a Tabela 3.2 como um plano de produção dado por $\pi = [1, 2, 3, 4, 5]$, as colunas indicam quais ferramentas devem ser equipadas na máquina para o processamento de cada tarefa, preferencialmente mantendo o *magazine* completo. Devido ao fato de a tarefa 4 exigir apenas duas ferramentas no *magazine*, opta-se por manter equipada a ferramenta 1, que está destacada na tabela, evitando-se assim uma troca desnecessária no futuro.

Para analisar o número de trocas de ferramentas resultante do plano de produção, deve-se observar na matriz binária as ocorrências de inversão do valor 0 para 1. Uma inversão ocorre quando em um estágio t uma ferramenta apresenta valor $a_{ij} = 0$ e no estágio $t + 1$ o valor passa a ser $a_{ij} = 1$. Esta contagem pode ser feita através da função de avaliação descrita pela Equação (3.1).

$$Z_{SSP}(A^{\pi}) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}(1 - a_{ij-1}) \tag{3.1}$$

Ao avaliar a solução induzida pela Tabela 3.2 é possível observar as seguintes trocas, considerando que cada par ordenado indica a ferramenta removida e a ferramenta equipada, separadas por estágio do plano de produção:

- Estágio 1: (-, 3), (-, 5), (-, 6);
- Estágio 2: (5, 2), (6, 4);

- Estágio 3: (2, 1), (3, 5);
- Estágio 4: (5, 6);
- Estágio 5: (6, 2);

Ao todo são efetuadas 9 trocas de ferramentas para se processar todas as tarefas, incluindo na contagem as inserções das três ferramentas iniciais. De acordo com o objetivo do problema, deseja-se efetuar o menor número possível de trocas de ferramentas dado o plano de produção. Assim, a função objetivo do problema pode ser representada pela Equação (3.2).

$$\min_{\pi \in \Pi} Z_{\text{SSP}}(A^{\pi}) \tag{3.2}$$

Considerando o exemplo da Tabela 3.1, é possível encontrar uma solução melhor do que a apresentada na Tabela 3.2. Esta solução está descrita na Tabela 3.3, que representa a permutação $\pi = [3, 4, 1, 5, 2]$.

Tabela 3.3: Exemplo de solução ótima para a instância de exemplo do SSP.

Ferramentas/Tarefas	2	5	3	4	1
1	0	1	1	0	0
2	1	1	0	0	0
3	1	0	0	0	1
4	1	1	1	1	0
5	0	0	1	*1*	1
6	0	0	0	1	1
Capacidade do *magazine: 3*					

A solução expressa na Tabela 3.3 resulta em sete trocas de ferramentas apresentadas abaixo e separadas por estágios do plano de produção:

- Estágio 1: (-, 2), (-, 3), (-, 4);
- Estágio 2: (3, 1);
- Estágio 3: (2, 5);
- Estágio 4: (1, 6);
- Estágio 5: (4, 3);

3.2 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante é amplamente estudado na área de Pesquisa Operacional. Este problema é frequentemente abordado na literatura ao se investigar problemas que tratem

da minimização do comprimento de uma rota que inclui elementos modelados genericamente como cidades.

Em um enunciado simples, um caixeiro viajante deseja vender seus produtos em diversas cidades. Para efetuar suas vendas, o caixeiro pretende visitar uma única vez cada cidade, utilizando as estradas que conectam todas as cidades entre si. Ao final do percurso, o caixeiro viajante deve retornar à cidade de origem. O objetivo é definir a rota de visitação das cidades com menor comprimento total.

Formalmente, o TSP pode ser representado através de grafos. Seja $G = (V, E)$ um grafo completo. O conjunto V é o conjunto de vértices, que para o problema correspondem às n cidades que o caixeiro deseja visitar. O conjunto E de arestas corresponde às estradas que conectam as cidades. Assim, o peso de cada aresta d_{ij} é dado por uma métrica de distância entre as cidades i e j. Um exemplo de representação em grafos é apresentado na Figura 3.1, que contém 4 cidades numeradas de 1 a 4. A distância entre as cidades é apresentada junto de cada aresta, ou seja, $d_{12} = 10$, $d_{21} = 10$ e $d_{24} = 5$, por exemplo.

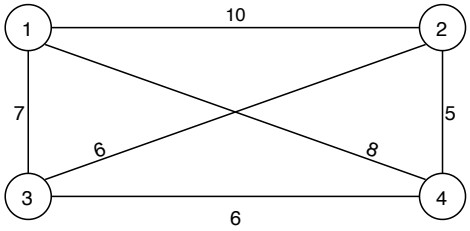


Figura 3.1: Representação gráfica de uma instância do TSP.

A abstração apresentada pode ser representada também por uma matriz de distâncias D, composta pelas distâncias entre as cidades. A conversão da Figura 3.1 em uma matriz é apresentada na Tabela 3.4, na qual linhas e colunas representam as cidades. A diagonal principal foi preenchida com zeros, pois a distância de uma cidade para si mesma é tida como nula.

Tabela 3.4: Matriz de distâncias de uma instância do TSP.

Cidade	1	2	3	4
1	0	10	7	8
2	10	0	6	5
3	7	6	0	6
4	8	5	6	0

Uma solução para o TSP corresponde a uma permutação π do conjunto V de cidades, que representa um ciclo hamiltoniano no grafo G. Esse tipo específico de ciclo requer que cada cidade seja visitada exatamente uma vez, fixando uma cidade como ponto inicial e final do trajeto. A Figura 3.2 ilustra uma possível solução $\pi = [1, 2, 3, 4]$ para o exemplo apresentado

na Figura 3.1. As arestas utilizadas no ciclo estão com traçado contínuo, enquanto as que não estão foram representadas através de linhas tracejadas.

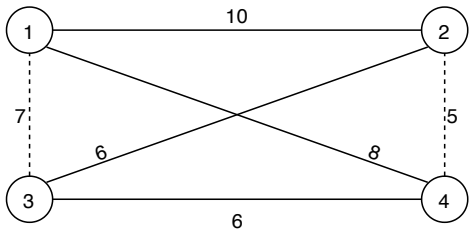


Figura 3.2: Representação gráfica de uma solução para o TSP. As arestas utilizadas no ciclo estão com traçado contínuo.

Dada uma permutação π do conjunto de cidades, se faz necessário calcular a distância total correspondente àquela rota. Para isso, utiliza-se a função de avaliação apresentada na Equação (3.3). Cada posição i da permutação π indica qual a ordem de visitação de cada uma das cidades, ou seja, $i = 1$ indica qual cidade é a origem da rota, $i = 2$ indica qual a segunda cidade visitada, por exemplo. Desta forma, ao observar na matriz de distâncias o valor correspondente à posição $\{\pi_n, \pi_1\}$ e somar todas as distâncias remanescentes, $\{\pi_i, \pi_{i+1}\}$ para todo $i < n$, é possível calcular a distância total percorrida.

$$Z_{TSP}(\pi) = d_{\pi_n \pi_1} + \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}} \tag{3.3}$$

Tendo em vista a Equação (3.3) e a Tabela 3.4 é possível computar a distância total da rota proposta na Figura 3.2: $d_{41} + d_{12} + d_{23} + d_{34} = 30$. Portanto, a rota apresentada para o TSP possui comprimento de 30 unidades de distância. Para satisfazer o objetivo do problema busca-se encontrar o menor valor possível para a Equação (3.4). Assim, a função objetivo do TSP é apresentada na Equação (3.4).

$$\min_{\pi \in \Pi} Z_{\text{TSP}}(\pi) \tag{3.4}$$

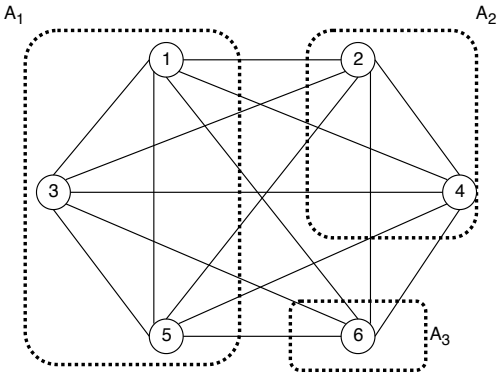
3.3 Problema do Caixeiro Viajante Generalizado

Uma das diversas variantes do TSP é conhecida como Problema do Caixeiro Viajante Generalizado (ou GTSP, do inglês *Generalized Traveling Salesman Problem*). Formalmente, seja $G = (V, E)$ um grafo não direcionado com n vértices e m arestas. Adicionalmente, considera-se um conjunto $A = \{A_1, A_2, ..., A_p\}$ de agrupamentos de elementos do conjunto V. Sem perda de generalidade, se assume que G é um grafo completo, e, portanto, a matriz de distâncias correspondente a ele é simétrica. Os agrupamentos, também denominados *clusters*, contêm os vértices do grafo e apresentam tamanho maior ou igual a um. Caso um vértice

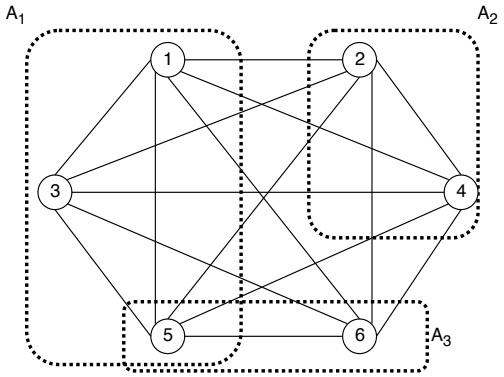
pertença a mais de um *cluster*, trata-se da versão *inclusiva* do GTSP. Em caso contrário, a versão é dita *exclusiva*.

Este agrupamento dos vértices faz com que não se deseje visitar cada vértice exatamente uma vez, mas sim visitar cada *cluster* exatamente uma vez. O objetivo de encontrar um ciclo de custo mínimo originário do TSP é preservado, entretanto, a solução passa a ter tamanho determinado pela quantidade de *clusters*.

A Figura 3.3 contém dois exemplos da representação em grafos de uma entrada para o GTSP. Nela os *clusters* são representados pelos retângulos tracejados, cada vértice por um círculo e cada aresta por uma linha sólida. No grafo contido na Figura 3.3a é possível observar uma representação do GTSP em sua versão *exclusiva*. No grafo contido na Figura 3.3b há a representação de uma instância do GTSP *inclusiva*. Os vértices foram agrupados em três *clusters* sob um critério aleatório. Os pesos das arestas foram omitidos no grafo, porém são apresentados na matriz de distâncias contida na Tabela 3.5. A distância de uma cidade para si mesma é tida como nula e a matriz é simétrica.



(a) Grafo correspondente a uma instância do GTSP exclusivo.



(b) Grafo correspondente a uma instância do GTSP inclusivo.

Figura 3.3: Exemplos de instâncias do GTSP nas quais os *clusters* são representados por retângulos tracejados e os vértices por círculos.

Tabela 3.5: Matriz de Distâncias para uma instância do GTSP.

Cidade	1	2	3	4	5	6
1	0	10	9	6	7	5
2	10	0	8	8	13	11
3	9	8	0	15	7	10
4	6	8	15	0	13	7
5	7	13	7	13	0	9
6	5	11	10	7	9	0

A Tabela 3.6 exemplifica uma representação da pertinência de cada vértice a um determinado *cluster*. Na Tabela 3.6a há a representação correspondente para a Figura 3.3a, indicando

por exemplo que a cidade 1 pertence ao *cluster* A_1. Já a Tabela 3.6b é correspondente à Figura 3.3b e indica por exemplo que a cidade 5 pertence aos *clusters* A_1 e A_3 simultaneamente.

Tabela 3.6: Alocação das cidades por *cluster* em ambas versões do GTSP (a) exclusiva e (b) inclusiva.

Cidade	*Cluster*
1	A_1
2	A_2
3	A_2
4	A_2
5	A_1
6	A_3

(a)

Cidade	*Clusters*
1	A_1
2	A_2
3	A_2
4	A_2
5	A_1/A_3
6	A_3

(b)

Uma solução para o GTSP corresponde a um sequência σ contendo entre um e p vértices, indicando qual vértice é visitado em cada *cluster*, bem como a ordem de visitação dos *clusters*. A Figura 3.4 ilustra uma possível solução para a instância apresentada na Figura 3.3a com $\sigma = [1, 2, 6]$. As cidades visitadas estão indicadas por círculos preenchidos e apenas as arestas utilizadas estão representadas junto de seus custos correspondentes.

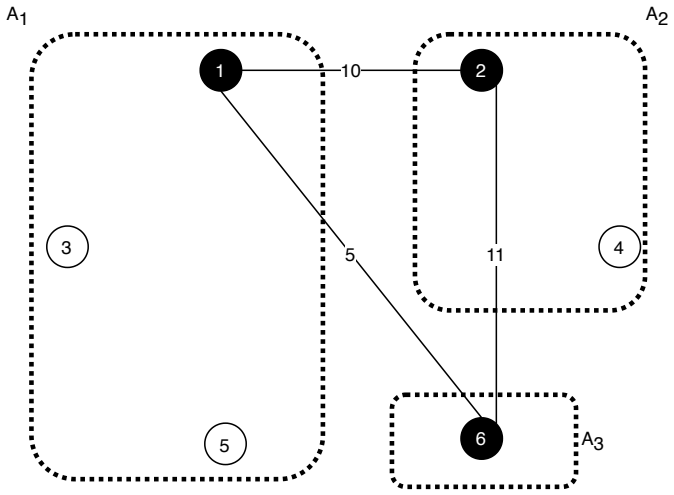


Figura 3.4: Representação gráfica de uma solução para o GTSP.

A avaliação de uma solução do GTSP é feita da mesma forma que a apresentada na Equação (3.3) para o TSP. Para se avaliar uma solução observa-se o peso atribuído a cada aresta e efetua-se o somatório dos pesos das arestas utilizadas no ciclo. Logo, a solução apresentada em σ tem custo dado por $d_{12} + d_{26} + d_{16} = 26$ unidades de distância. Desta forma, a função objetivo apresentada para o TSP na Equação (3.4) também é a mesma utilizada para o GTSP haja vista que se deseja encontrar o ciclo que visite uma única vez todos os *clusters* de menor custo possível.

3.3.1 Busca Local Iterada aplicada ao GTSP

Todas as soluções válidas ou não para um determinado problema combinatório caracterizam uma abstração denominada *espaço de soluções*. As soluções que satisfazem todas as restrições impostas pela natureza do problema abordado caracterizam o espaço de soluções *viáveis*. De forma genérica, efetuar uma *busca local* significa explorar regiões alcançáveis a partir de um ponto de partida definido dentro do espaço de soluções. Se o contexto for definido como um grafo, buscar localmente significa descobrir quais vértices podem ser alcançados, sob um determinado critério, a partir de uma origem determinada.

A partir de uma solução já definida, também é possível mover a busca para outras regiões do espaço de soluções. Para isso, são efetuadas operações, genericamente denominadas *perturbações*. Tais operações causam certos níveis de desordem na solução que permitem alcançar outras regiões do espaço de soluções, anteriormente inalcançáveis pela busca local.

Diferentes meta-heurísticas alternam métodos de busca local e de perturbação para abordagem de problemas combinatórios. Dentre elas, destaca-se a *busca local iterada* (Lourenço et al., 2003). Este método consiste em aplicar uma busca local sobre uma solução inicial, partindo de soluções iniciais diferentes a cada iteração. Quando um mínimo local é encontrado pela busca local, é aplicado algum nível de perturbação sobre esta solução para se reiniciar o procedimento a partir de um outro ponto do espaço de soluções.

Atualmente, a metaheurística estado da arte para abordagem do GTSP é uma implementação de ILS combinada à heurística de *início múltiplo* proposta por Reinsma et al. (2018) e apresentada nos Algoritmos 1 e 2. Os parâmetros homônimos em ambos algoritmos preservam sua significação. Desta forma, $iter_{maxMS}$ representa o número máximo de iterações do início múltiplo, p_{ini} é o índice de perturbação inicial, p_{incr} é o incremento do nível de perturbação, num_{incr} indica quantas vezes o índice de perturbação pode ser aumentado, $iter_{maxSILS}$ corresponde ao número máximo de iterações da ILS e s representa a solução atual.

O início múltiplo, apresentado no Algoritmo 1, consiste em a cada iteração gerar uma solução s (linhas 4-11) e aplicar sobre s a ILS, buscando encontrar novos ótimos locais e eventualmente, o ótimo global. Uma implementação da heurística de *vizinho mais próximo* do TSP, adaptada ao GTSP é utilizada para gerar um novo s a cada início (linha 6), tendo em vista que o primeiro vértice inserido em s é definido sob um critério aleatório. Em seguida, é aplicado o método *Cluster Improvement* sobre s (linha 7). Este método seleciona um vértice v_i pertence a s, identifica a qual *cluster* o vértice pertence e tenta substituí-lo por outro vértice do mesmo *cluster*, optando por aquele que gerar a melhor melhoria para a solução.

Ainda sobre s aplica-se o método *2-opt* (linha 8), que é popular na literatura do TSP e consiste em inverter as posições de vértices dentro de um intervalo definido na permutação que s representa. Novamente é aplicado o método *Cluster Improvement* (linha 9) e depois o método descrito em detalhes no Algoritmo 2 (linha 10). Estas operações se repetem por um número fixo de iterações, o que caracteriza a aplicação da heurística de início múltiplo

repetidas vezes em conjunto com a busca local iterada proposta.

Algoritmo 1: *Multi-Start Smart Iterated Local Search* (MS-SILS)

Entrada: $iter_{maxMS}$, p_{ini}, p_{incr}, num_{incr}, $iter_{maxSILS}$

```
1  início
2      f(s*) ← ∞;
3      iter ← 0;
4      enquanto iter < iter_maxMS faça
5          iter ← iter + 1;
6          s ← NearestNeighbor();
7          s' ← ClusterImprovement(s);
8          s' ← 2 − Opt(s');
9          s' ← ClusterImprovement(s');
10         s'' ← SILS(s', p_ini, p_incr, num_incr, iter_maxSILS);
11         se f(s'') < f(s*) então
12             f(s*) ← f(s'');
13         fim
14     fim
15 fim
```

Saída: s^*

O Algoritmo 2 apresenta a implementação de Reinsma et al. (2018) para a busca local iterada, levemente modificada. Esta versão é dita inteligente por efetuar uma variação gradual no índice de perturbação da solução dado um número de iterações sem melhora (linha 18). Perturbar a solução nesta implementação representa trocar a posição de diferentes elementos da solução. Em um aspecto geral, este método funciona de forma semelhante a uma busca local iterada comum, que efetua perturbações na solução corrente com o objetivo de explorar de forma eficiente o espaço de soluções.

Na linha 3 ocorre o cálculo do índice máximo de perturbação da solução. Enquanto não for atingido o nível máximo de perturbação (linhas 5-20) o laço de repetição interno será executado. O laço interno (linhas 6 - 17) é responsável por executar o número máximo de iterações da ILS. A solução s' é obtida através do embaralhamento da solução s (linha 8). Em seguida (linhas 9-11) é aplicado o método *ClusterImprovement*, seguido de *2-Opt* e novamente de *ClusterImprovement*, para se obter a soluçao s''. As linhas 12 a 16 apresentam o critério de aceitação para uma nova solução, que depende que o resultado da aplicação da função de avaliação sobre s'' seja menor que o da melhor solução conhecida até o momento s. As linhas 18 e 19 preparam o método para a próxima iteração, com novo índice de perturbação.

O código-fonte deste método foi gentilmente cedido pelos atores e será utilizado para resolver a modelagem apresentada em detalhes no capítulo seguinte. Os parâmetros utilizados no Algoritmo 1 foram os recomendados originalmente: $p_{ini} = 10\%$, $p_{incr} = 5\%$, $num_{incr} = 3$, $iter_{maxSILS} = 150$ e $iter_{maxMS} = 50$.

Algoritmo 2: *Smart Iterated Local Search (SILS)*

Entrada: $s, p_{ini}, p_{incr}, num_{incr}, iter_{maxSILS}$

início
- $p \leftarrow p_{ini}$;
- $p_{max} \leftarrow p + (p_{incr} \times num_{incr})$;
- $iter \leftarrow 0$;
- **enquanto** $p \leq p_{max}$ **faça**
  - **enquanto** $iter < iter_{maxSILS}$ **faça**
    - $iter \leftarrow iter + 1$;
    - $s' \leftarrow RandomShuffle(s, p)$;
    - $s'' \leftarrow ClusterImprovement(s')$;
    - $s'' \leftarrow 2 - Opt(s'')$;
    - $s'' \leftarrow ClusterImprovement(s'')$;
    - **se** $f(s'') < f(s)$ **então**
      - $s \leftarrow s''$;
      - $p \leftarrow p_{ini}$;
      - $iter \leftarrow 0$;
    - **fim**
  - **fim**
  - $p \leftarrow p + p_{incr}$;
  - $iter \leftarrow 0$;
- **fim**

fim

Saída: s

3.3.2 Heurística Generalized Lin-Kernighan-Helsgaun

Helsgaun (2015b) propõe um *framework* para solucionar o GTSP exclusivo. De forma simples, o método recebe uma instância do GTSP padronizada de forma semelhante às instâncias contidas na biblioteca TSPlib. O método proposto efetua uma transformação da instância do GTSP para uma instância do Problema do Caixeiro Viajante Assimétrico (ATSP, do inglês *Asymmetric Traveling Salesman Problem*). Esta versão difere do TSP quanto a definição do custo dos arcos entre os vértices i e j, que varia de acordo com o sentido de percurso do arco. A transformação do GTSP em ATSP é descrita brevemente na Seção 4.1.2.

Depois da transformação, o método GLKH utiliza a heurística Lin-Kernighan-Helsgaun (LKH), proposta por Helsgaun (2000). O LKH é capaz de resolver diversas variações do TSP, entre elas a sua versão assimétrica e como consequência da transformação feita sobre a entrada, é possível obter uma solução para o ATSP e transformá-la em uma solução válida e sem perda de generalidade para o GTSP. Este método é conhecido na literatura por obter resultados consistentes e frequentemente ótimos para os problemas em que foi empregado.

De maneira superficial, o LKH utiliza de movimentos do tipo λ-opt para perturbar a solução corrente e ampliar a busca no espaço de soluções. O valor de λ varia entre dois e cinco,

indicando entre uma a quatro sequências de aplicações de trocas por meio de movimentos 2-opt. O *framework* se encontra disponível em Helsgaun (2015a). O GLKH é um executável disponibilizado de forma gratuita para uso acadêmico. Foi possível utilizar a implementação mais recente da heurística LKH de Helsgaun (2000) acoplada ao GLKH.

Capítulo 4

Desenvolvimento

Conforme mencionado no Capítulo 2, frequentemente o SSP é transformado em outros problemas com o objetivo de facilitar a busca por uma solução de qualidade. A transformação do SSP em TSP é recorrente na literatura, apesar não existir equivalência entre os problemas (Azevedo e Carvalho, 2017). Entretanto, não houve larga exploração da redução do SSP ao GTSP em trabalhos anteriores, exceto por Yanasse e Lamosa (2006a), Yanasse e Lamosa (2006b) e Moreira (2016). A modelagem aqui apresentada foi extraída de Moreira (2016) com o objetivo de investigar a qualidade da modelagem através de sua resolução pode meio de métodos heurísticos.

4.1 Redução do SSP ao GTSP

Seja $G = (V, E)$ um grafo completo no qual cada vértice contido em V corresponde a uma possível *configuração do magazine* que sempre requer C ferramentas. As arestas têm peso definido como a quantidade literal de trocas de ferramentas entre cada par de vértices i e j, ou seja, quantas trocas de ferramentas são necessárias para se transformar a configuração i na configuração j do magazine. A matriz de distâncias resultante é simétrica. Todos os vértices com capacidade para processar uma tarefa são agrupados em um único *cluster*. Portanto, se um vértice for suficiente para processar mais de uma tarefa, este estará presente em mais de um *cluster*. Neste grafo, há a inserção de um *cluster* artificial, denotado por A_0, cujo único vértice O corresponde a uma configuração do magazine vazio. A distância deste vértice para todos os demais é zero.

Para solucionar o SSP é necessário determinar a ordem de visitação dos *clusters* que minimiza o número de trocas de ferramentas e qual vértice será visitado em cada *cluster*. A partir destas informações é possível formular o *plano de produção*, que terá início e fim fixado no *cluster* A_0.

A Tabela 4.1, extraída de Moreira (2016), contém uma instância para o SSP na qual $C = 5$, $n = 5$ e $m = 8$. É possível notar que as tarefas 2 e 5 requerem exatamente C ferramentas.

Desta forma, em uma representação em grafos, seus respectivos *clusters* terão apenas um vértice. Porém, para as demais tarefas é possível gerar diversas combinações para completar os espaços não preenchidos do *magazine*.

Tabela 4.1: Exemplo de instância do SSP, extraído de Moreira (2016).

Tarefas	Ferramentas
1	A, B, C
2	A, C, D, E, F
3	A, E
4	B, C, G
5	D, E, F, G, H

A Figura 4.1, adaptada a partir de Moreira (2016), contém uma representação gráfica para a instância da Tabela 4.1. É possível observar que a representação corresponde ao GTSP inclusivo, haja vista que uma configuração do magazine pode ser capaz de processar mais de uma tarefa. Por questão de legibilidade, não são representadas todas as possíveis combinações de vértices contidos em cada *cluster* nem as arestas.

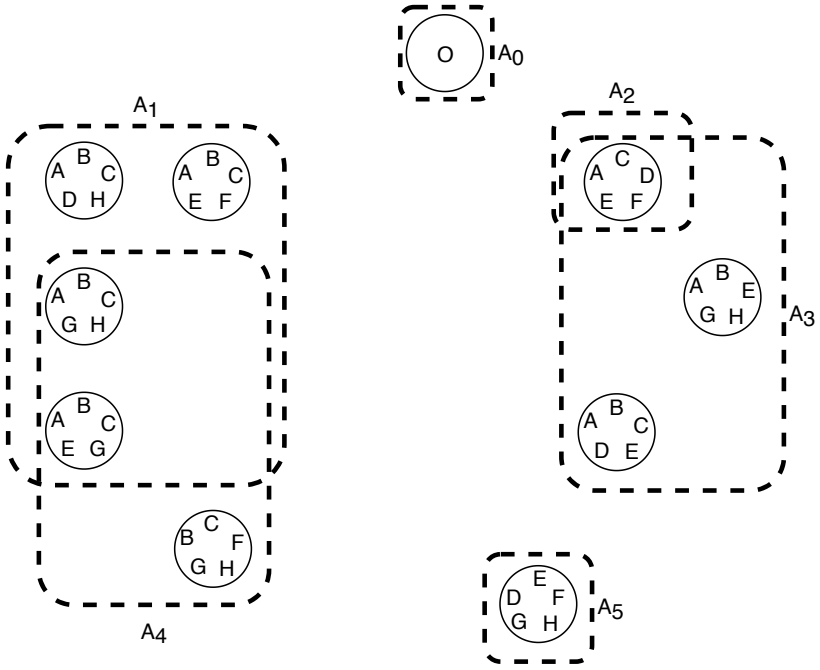


Figura 4.1: Representação gráfica parcial de uma instância do GTSP, adaptada a partir de Moreira (2016).

A Figura 4.2 apresenta a representação de uma possível solução para o GTSP. O peso das arestas é calculado a partir do número literal de trocas de ferramentas necessárias para modificar a configuração do magazine representada pelo vértice i para a configuração representada

pelo vértice j. Esta solução apresenta um total de 11 trocas de ferramentas, acrescidas das $C = 5$ inserções iniciais. Esta adição é realizada ao final do processo para que as arestas do vértice O possuam peso zero e sejam neutras no grafo.

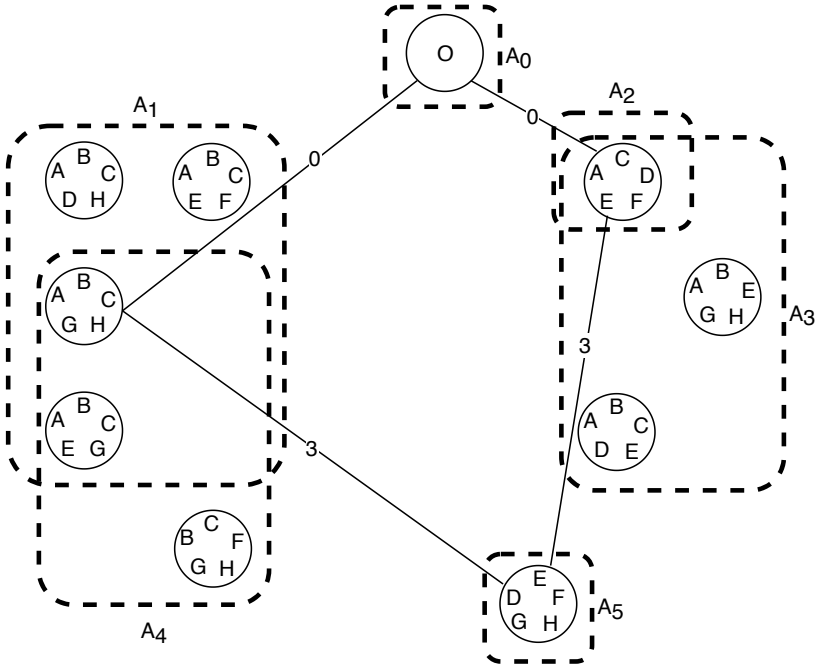


Figura 4.2: Representação gráfica de uma solução para o SSP quando modelado como GTSP, adaptada a partir de Moreira (2016).

O grafo G correspondente a uma instância do SSP modelada como uma instância GTSP frequentemente corresponde à versão inclusiva do GTSP, pois uma configuração do *magazine* pode ser capaz de processar mais de uma tarefa. Considerando esta situação, para a aplicação dos métodos heurísticos considerados neste trabalho se faz necessário transformar uma instância do GTSP inclusivo em uma instância do GTSP exclusivo, na qual não há interseção entre os *clusters*. Esta transformação acontece em uma etapa apresentada na Seção 4.1.1.

4.1.1 Transformação do GTSP inclusivo em GTSP exclusivo

Considere que um grafo G representa uma instância para o GTSP inclusivo. É possível transformar G em $G' = \{V', E'\}$, correspondente ao GTSP exclusivo, sem alteração da equivalência entre as soluções ótimas de G e G' (Lien et al., 1993). Desta forma, para cada vértice $v \in V$ é associado um número q_v que corresponde à quantidade de *clusters* que contem o vértice v. O número q_v também indica a cardinalidade do conjunto I_v, que contém os índices dos *clusters* que contém o vértice v.

A partir destas informações são criadas q_v cópias do vértice v. Os vértices passam a ser

rotulados por um par ordenado $u = (v, i)$ que indica qual é o número do vértice e a qual *cluster* pertence. O grafo permanece completo, portanto, as arestas que conectam cópias de um vértice $v \in V$ apresentam custo nulo. As arestas que conectam vértices diferentes apresentam os mesmos custos do grafo original.

A Figura 4.3 ilustra a transformação de uma instância do GTSP inclusivo em um GTSP exclusivo. O vértice que representa a configuração do *magazine* $\{A, C, D, E, F\}$ estava contido nos *clusters* A_2 e A_3. Desta forma, foram criadas duas cópias deste vértice, sendo uma inserida em cada *cluster* correspondente. Os vértices $\{A, B, C, G, H\}$ e $\{A, B, C, E, G\}$ pertencem simultaneamente aos *clusters* A_1 e A_4, portanto ambos foram duplicados e inseridos em cada *cluster* correspondente.

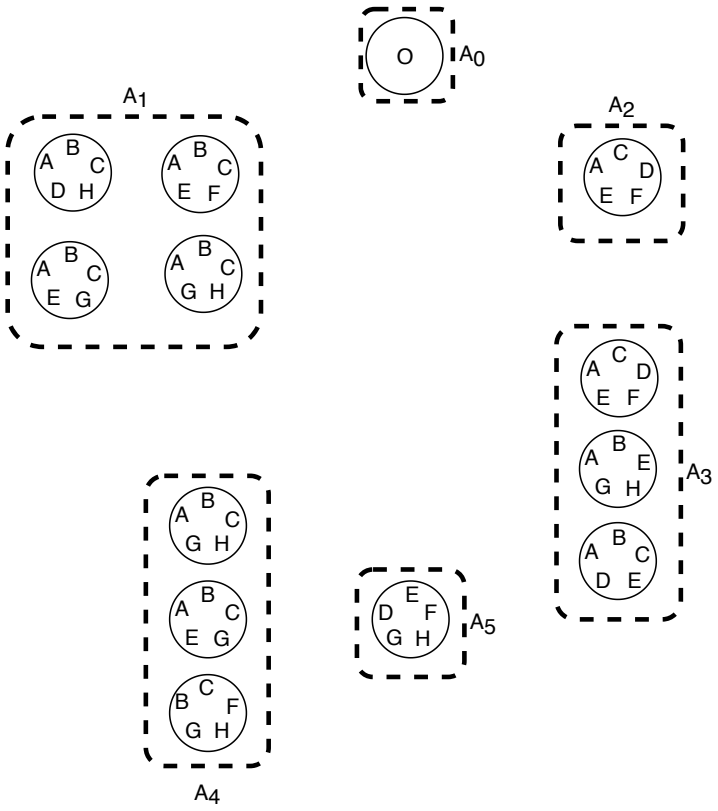
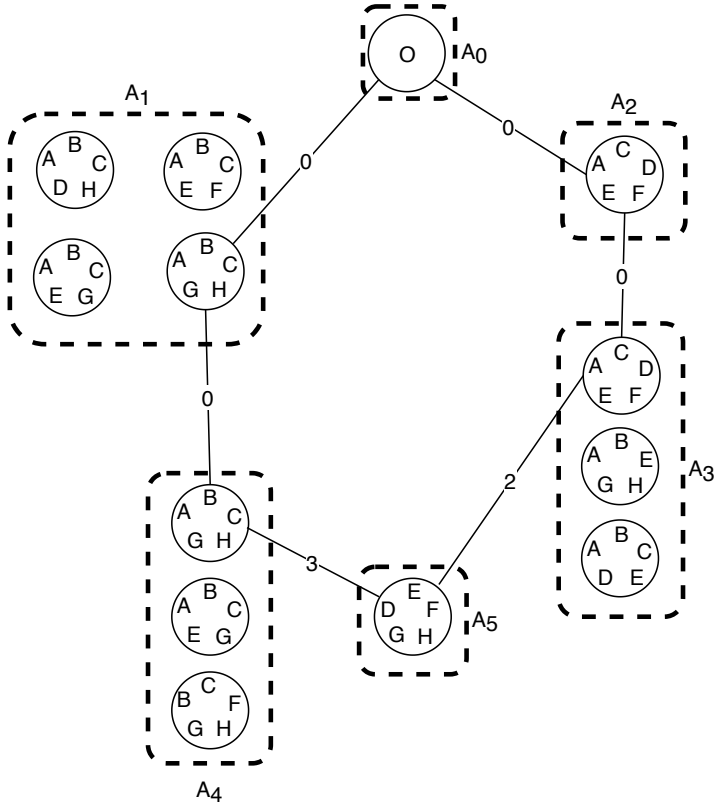


Figura 4.3: Representação gráfica da transformação do GTSP inclusivo em exclusivo.

A Figura 4.4 contém a representação de uma solução para a transformação da instância exposta na Figura 4.3. A solução $\sigma = \{A_0, A_1, A_4, A_5, A_3, A_2\}$ apresenta custo 5.

A solução σ para o GTSP é equivalente à solução π do SSP. Para a instância apresentada, $\pi = \{1, 4, 5, 3, 2\}$ significa remover de σ o *cluster* artificial A_0 e considerar o índice de cada *cluster*, uma vez que cada *cluster* representa uma tarefa e a solução do SSP corresponde ao sequenciamento das tarefas. Também é importante destacar que a solução σ pode incluir a visitação de vértices que representam a mesma configuração do magazine, presentes em

Figura 4.4: Representação gráfica da transformação do GTSP inclusivo em exclusivo.



clusters diferentes em função da transformação do GTSP inclusivo em exclusivo. Um exemplo disto é o vértice $\{A, B, C, G, H\}$ que pode ser utilizado no processamento das tarefas 1 e 4. A escolha deste vértice é vantajosa porque não acarreta nenhuma troca de ferramentas entre o processamento das tarefas 1 e 4, sendo potencialmente favorável à diminuição do número de trocas de ferramentas.

A solução π do SSP tem valor 10, sendo 5 unidades de custo correspondentes às inserções iniciais das ferramentas e outras 5 resultantes das trocas ao longo do processamento do plano de produção, que corresponde ao valor encontrado pela solução do GTSP exclusivo. É importante destacar que a implementação da modelagem utilizada gera diretamente uma instância resultante do GTSP exclusivo e que os métodos de solução utilizados neste trabalho recebem apenas este tipo de grafo como entrada.

4.1.2 Transformação do GTSP exclusivo em ATSP

A partir da transformação apresentada na Seção 4.1.1 é possível obter uma instância do ATSP. Esta transformação é feita de maneira semelhante à exposta em Helsgaun (2015b). Seja G'' o grafo correspondente a uma instância do GTSP exclusivo. Para cada *cluster* com mais de um vértice é criado um circuito (i.e., um ciclo direcionado) que passa por todos os

vértices, em ordem arbitrária de visitação, no qual o custo dos arcos de i para j é igual a zero e j sucede i no ciclo. A Figura 4.5 apresenta esta etapa da transformação. Para fins de visualização não são representados todos os arcos do grafo.

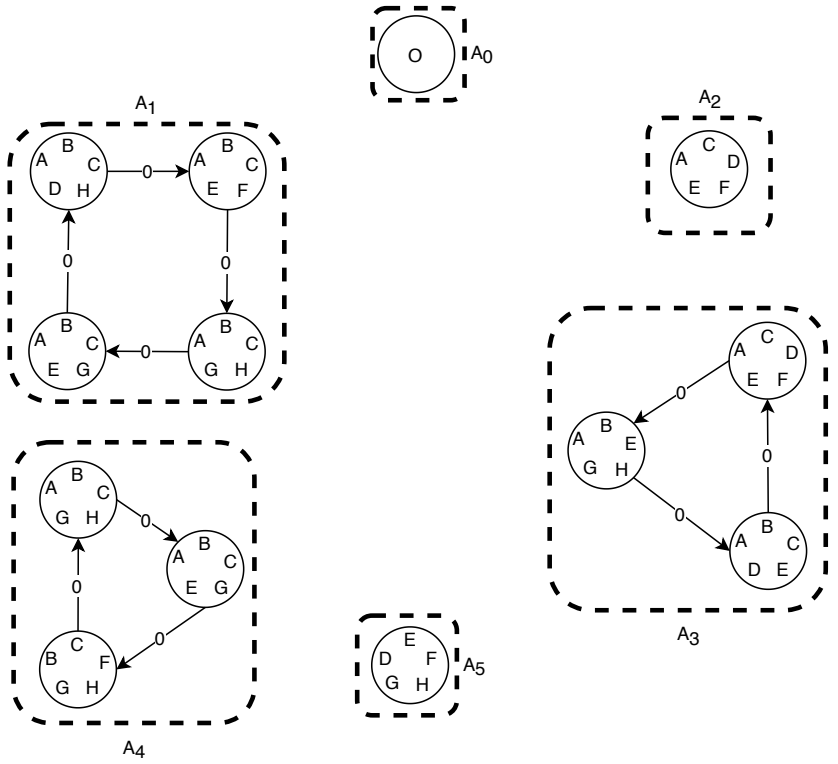
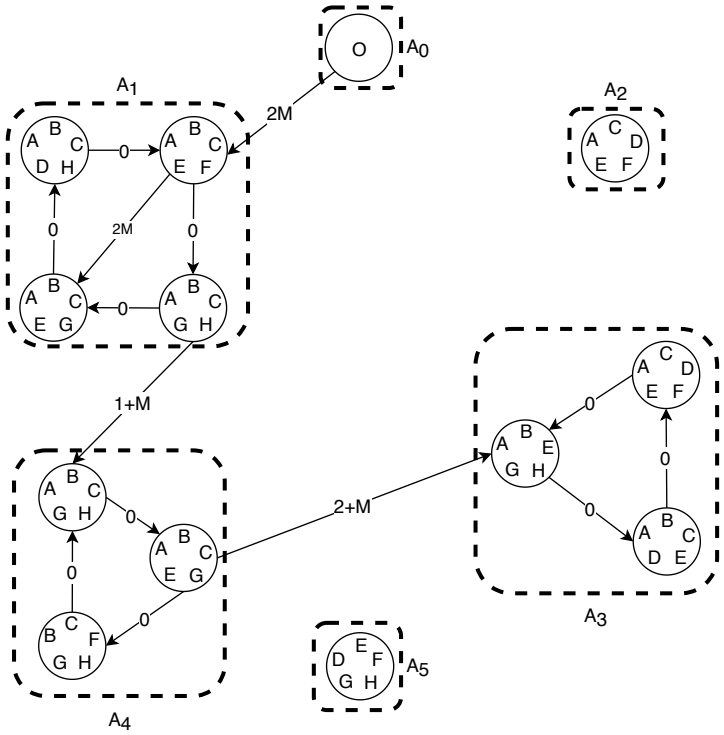


Figura 4.5: Representação gráfica da primeira fase da transformação do GTSP exclusivo em ATSP.

Para vértices i e j que não pertencem ao mesmo *cluster*, define-se o custo do arco de i para j como o custo de ir de k para j mais uma constante suficientemente grande (M), sendo k o vértice que sucede i no circuito anteriormente definido. Para os demais casos, o custo de ir de um vértice a outro é dado por duas vezes a constante M. A Figura 4.6 apresenta esta segunda etapa da transformação. Na figura estão contidos alguns exemplos, das definições de custo dos arcos, como o arco do único vértice do *cluster* A_0 para um vértice de um *cluster* não unitário, um arco para vértices contidos no mesmo *cluster* mas que não se sucedem no ciclo e vértices de *clusters* diferentes.

De acordo com Helsgaun (2015b), esta transformação é válida, pois ao se escolher um vértice qualquer de um determinado *cluster*, há uma sub-rota ótima de custo zero que visita todos os vértices daquele *cluster*. É importante destacar que esta transformação atua apenas sobre a definição dos custos dos arcos. Logo, o grafo G''' resultante apresenta o mesmo número de vértices do grafo G''. Por fim, o grafo G''' resulta em uma instância do ATSP, pois o custo de um vértice i para um vértice j não é mesmo em ambos os sentidos do arco. Por exemplo, o

Figura 4.6: Representação gráfica da transformação do GTSP exclusivo em ATSP.



custo do arco conectando o vértice O contido no *cluster* A_0 a qualquer outro vértice é sempre dado por duas vezes a constante M. Porém, o custo do arco dos demais vértices para O varia de acordo com a composição dos *clusters* e da posição dos vértices nos circuitos criados dentro de cada *cluster*.

4.1.3 Geração de *clusters* de vértices

Para efetivamente transformar o SSP em GTSP é necessário gerar as possíveis combinações de C ferramentas que representam os vértices do grafo. Entretanto, existem configurações do *magazine* que nunca serão capazes de processar qualquer tarefa. Desta forma, apresenta-se o Algoritmo 3, que descreve o funcionamento de uma geração recursiva das possíveis combinações, utilizando do conhecimento das ferramentas requeridas por cada tarefas. Como entrada é necessário fornecer a quantidade f de ferramentas não utilizadas pela tarefa j, a diferença d entre a capacidade C do magazine e quantidade de ferramentas requeridas pela tarefa j, um vetor v auxiliar que armazena temporariamente a combinação corrente e um identificador $numJob$ que indica a tarefa utilizada para gerar as combinações. Como saída é produzido o conjunto de todas as configurações do *magazine*, de tamanho C capazes de processar a tarefa $numJob$ do SSP.

O método recursivo se inicia pela definição da quantidade de ferramentas disponíveis para combinação (linha 2). Em seguida, é iniciado o laço de repetição principal (linhas 3-12),

responsável por inserir a i-ésima ferramenta na penúltima posição disponível de v (linha 4) e chamar recursivamente a função para gerar as demais combinações (linhas 5-7), se houver capacidade remanescente em v. Após a chamada recursiva (linhas 8-11), combina-se o vértice obtido com as ferramentas requeridas pela tarefa j (linha 9), pelo método *UnirFerramentas*, que recebe a tarefa *numJob* e o subconjunto gerado v de forma a obter um vértice com C ferramentas. Por fim, é feita a ordenação das ferramentas contidas no vértice e a inserção do vértice gerado no *cluster* correspondente à tarefa j (linha 11).

Algoritmo 3: GeraVertices

```
   Entrada: f, d, v, numJob
 1 início
 2     i ← m;
 3     enquanto i ≥ d faça
 4         v ← i;
 5         se d > 1 então
 6             GeraVertices(i − 1, d − 1, v, numJob);
 7         fim
 8         senão
 9             UnirFerramentas(numJob, v);
10         fim
11         Cluster ← v;
12     fim
13     i ← i − 1;
14 fim
   Saída: Cluster da tarefa numJob
```

O grafo resultante deste processo de geração de vértices é uma instância do GTSP exclusivo. Portanto, pode ser resolvido utilizando-se a heurística de Reinsma et al. (2018) e o GLKH proposto por Helsgaun (2015b), apresentados no Capítulo 3.

Capítulo 5

Experimentos

Este capítulo apresenta os resultados obtidos por meio dos experimentos computacionais realizados. Todos os experimentos foram realizados em um computador com processador *Intel Core i7* de 3.6GHz com 16 GB de RAM. O sistema operacional foi o Ubuntu 18.04.1 LTS. O GLKH utilizado apresenta implementação em ANSI C, estando na versão 1.0 acoplado ao LKH na versão 3.0.6. Os demais métodos utilizados foram implementados na linguagem C++, na versão 14, e compilados utilizando o *gcc* versão 7.4.0 e *flags* de compilação -O3 e -march=native.

Este capítulo está dividido em três seções. A Seção 5.1 apresenta uma breve descrição de cada conjunto de instâncias utilizado. A Seção 5.2 contém uma análise dos resultados obtidos durante o pré-processamento das entradas para posterior aplicação dos métodos de solução. Por fim, a Seção 5.3 contém os resultados obtidos para as instâncias que foram totalmente solucionadas pelos métodos heurísticos utilizados.

5.1 Descrição dos conjuntos de instâncias utilizados

As instâncias utilizadas são caracterizadas pelo número de tarefas (n), número de ferramentas (m) e quantidade de *slots* no magazine (C). Cada conjunto de instâncias foi rotulado para posterior referência neste capítulo. O primeiro conjunto utilizado foi proposto por Crama et al. (1994). Neste existem 160 instâncias agrupadas em quatro grupos:

- C_1: n=10, m=10, $C \in \{4, 5, 6, 7\}$;
- C_2: n=15, m=20, $C \in \{6, 8, 10, 12\}$;
- C_3: n=30, m=40, $C \in \{15, 17, 20, 25\}$;
- C_4: n=40, m=60, $C \in \{20, 22, 25, 30\}$.

O segundo conjunto de instâncias utilizado foi proposto por Catanzaro et al. (2015). Neste existem 160 instâncias agrupadas em 4 grupos nomeados dat_A, dat_B, dat_C e dat_D respectivamente:

- dat_A: n=10, m=10, $C \in \{4, 5, 6, 7\}$;
- dat_B: n=15, m=20, $C \in \{6, 8, 10, 12\}$;
- dat_C: n=30, m=40, $C \in \{15, 17, 20, 25\}$;
- dat_D: n=40, m=60, $C \in \{20, 22, 25, 30\}$.

O terceiro conjunto de instâncias foi proposto por Yanasse et al. (2009). Neste conjunto há 1350 instâncias, divididas em 5 grupos chamados A, B, C, D e E respectivamente:

- A: n=8, $m \in \{15, 20, 25\}$, $C \in \{5, 10, 15, 20\}$;
- B: n=9, $m \in \{15, 20, 25\}$, $C \in \{5, 10, 15, 20\}$;
- C: n=15, $m \in \{15, 20, 25\}$, $C \in \{5, 10, 15, 20\}$;
- D: n=20, $m \in \{15, 20, 25\}$, $C \in \{5, 10, 15, 20\}$;
- E: $n \in \{10, 25\}$, $m \in \{10, 20\}$, $C \in \{4, 5, 6, 7, 8, 10, 12\}$.

5.2 Análise da fase de geração de vértices e *clusters*

Por se tratar de uma transformação do SSP em GTSP, faz-se relevante uma análise da fase de geração vértices. Como descrito na Seção 4.1.1 são geradas apenas as combinações de tamanho C capazes de processar tarefas. A Tabela 5.1 agrupa os resultados obtidos para cada instância. São apresentados o nome do grupo de instâncias, o menor (v_{min}) e o maior (v_{max}) número vértices gerados para aquele subconjunto de instâncias, o menor (t_{min}), o maior (t_{max}) tempo total para gerar os arquivos de entrada e o maior tamanho aproximado, em *megabytes*, de arquivo gerado. Destaca-se que os tempos compreendem o tempo de geração dos vértices para cada *cluster* acrescido do tempo de escrita do arquivo. Os tempos são expressos em segundos.

Os conjuntos de instâncias estão separados na tabela por uma linha horizontal. Foram ocultados da tabela os conjuntos C_3, C_4, dat_C, dat_D. Estes conjuntos resultaram em instâncias de ordem superior a 200 *gigabytes*, para as quais não é possível obter solução devido a disparidade entre o tamanho dos arquivos e a disponibilidade de memória primária. Claramente, o número de vértices gerados cresce rapidamente com o aumento das dimensões das instâncias e da diferença entre a quantidade de ferramentas exigidas por cada tarefa e a capacidade do magazine. Desta forma, se torna inviável armazenar todas as combinações possíveis e posteriormente utilizar estes valores como entrada para o ILS e o GLKH.

Tabela 5.1: Informações sobre a fase de geração de vértices.

Grupo	v_{min}	v_{max}	T_{min}	T_{max}	Tamanho máximo
C_1	61	485	< 1	< 1	< 1
C_2	3.641	302.302	< 1	6.383,657	86.000
dat_A	82	520	< 1	< 1	< 1
dat_B	2.997	329.758	< 1	7.579,13	102.000
A	203	153.439	< 1	1.534,021	442.000
B	189	162.231	< 1	1.922,608	445.000
C	175	170.964	< 1	2.032,034	451.000
D	230	185.921	< 1	2.435,006	453.000
E	46	282.854	< 1	5.373,641	92.000

Para as instâncias que foram transformadas com sucesso, é possível observar que conforme o número de vértices cresce, também crescem os tempos de geração dos arquivos e o tamanho dos mesmos. Também é possível inferir que quanto maior a diferença entre o tamanho do magazine e quantidade de ferramentas requeridas por cada tarefa, maior será o número de vértices gerados por conjunto, conforme a disparidade entre o número total de ferramentas disponíveis e a capacidade do magazine.

5.3 Análise das soluções obtidas pelas heurísticas

Conforme mencionado na Seção 5.2, devido a limitações de espaço não é possível gerar todos os arquivos de entrada para as heurísticas. Entretanto, mesmo para alguns conjuntos em que foi possível gerar os arquivos de entrada, não é possível manter todas as suas informações em memória primária, o que inviabiliza a aplicação dos métodos de solução. Desta maneira, apenas os conjuntos para os quais foi possível obter solução são reportados nesta seção.

A Tabela 5.2 apresenta a soma das melhores soluções conhecidaw (BKS) para cada grupo de instâncias, de acordo com Paiva e Carvalho (2017) e a quantidade de subconjuntos de cada grupo solucionado. Para as heurísticas consideradas neste estudo, a quantidade I de instâncias solucionadas dentro de cada grupo, a soma das soluções médias obtidas em 10 execuções independentes (S) de cada instância solucionada dividida pela quantidade de instâncias solucionadas, a melhor solução obtida (S^*) na solução de cada instância divida pela quantidade de instâncias solucionadas, o tempo médio de execução (T) em segundos e a distância percentual ($gap\%$) entre S^* e BKS, calculada como $100 \times (S^* - BKS)/BKS$.

Tabela 5.2: Resultados obtidos pela aplicação das heurísticas nas instâncias transformadas

		ILS					GLKH				
Grupo	BKS	I	S	S^*	T	$gap\%$	I	S	S^*	T	$gap\%$
C_1	44,70	40	46,11	45,90	0,00	2,68	40	44,70	44,70	9,83	0,00
C_2	26,60	10	27,91	27,40	0,00	3,01	10	26,78	26,60	211,81	0,00
dat_A	43,40	40	44,32	44,10	0,15	1,61	40	43,40	43,40	10,47	0,00
dat_B	26,50	10	28,07	27,50	0,01	3,77	10	29,32	29,10	241,80	9,81
A	55,63	50	56,56	56,40	0,00	1,38	50	55,63	55,63	98,64	0,00
B	56,97	50	57,83	57,57	0,001	1,38	50	56,97	56,97	225,91	0,00
C	67,00	50	70,30	68,90	0,568	2,84	50	67,00	67,00	272,63	0,00
D	73,30	50	75,60	74,95	0,568	2,25	50	73,30	73,30	265,43	0,00
E	72,40	50	77,72	76,50	0,325	5,56	50	72,40	72,40	273,20	0,00

Em primeiro plano é possível concluir que embora a abordagem utilizada esteja limitada a solucionar instâncias pequenas, quando se refere ao tamanho dos arquivos de entrada gerados, os resultados obtidos para as instâncias são satisfatórios. A diferença percentual entre os resultados obtidos e as melhores soluções conhecidas é pequena, para ambos os métodos. O maior $gap\%$ foi encontrado no grupo dat_B para ambos os métodos, para o subconjunto com $n = 15$, $m = 20$ e $C = 6$, que continha 10 instâncias. Neste caso em específico o ILS obteve uma distância percentual igual 3,77% que é menor que os 9,81% obtidos com o GLKH. Excetuando-se este conjunto em específico, o GLKH foi capaz de encontrar com maior frequência soluções de melhor qualidade, que coincidem ou se aproximam muito das melhores soluções conhecidas.

Ainda sobre qualidade das soluções é importante destacar uma maior convergência do GLKH quando comparado ao ILS. Na maior parte das instâncias os resultados obtidos pelo GLKH para S e S^* são os mesmos, fator que indica uma constância na soluções encontradas. Por exemplo, para 10 execuções independentes, o GLKH tende a encontrar com maior frequência o mesmo valor de solução em todas as execuções, enquanto o ILS apresenta valores diferentes de S e S^* em todos os conjuntos solucionados. Este fato pode ser ilustrado com 10 instâncias contidas no conjunto dat_A, com $n = 10$, $m = 10$ e $C = 7$. Ambos os métodos foram capazes de encontrar as melhores soluções conhecidas. No caso deste subconjunto, o BKS é igual a 10 e a melhor solução de cada instância separadamente também é de 10 trocas de ferramentas. O GLKH foi capaz de encontrar o valor 10 tanto para S quanto para S^*. Porém, o ILS encontrou 10,02 para S, fator que indica que em algumas instâncias não se obteve o mesmo resultado nas 10 execuções independentes.

Apesar desta leve disparidade quanto a qualidade das soluções, outro fator deve ser observado ao analisar os resultados obtidos. Mesmo apresentando melhores resultados obtidos, o GLKH precisa de mais tempo que o ILS para encontrar uma solução. Enquanto o tempo médio de solução do ILS é sempre menor ou em torno de 1 segundo para cada execução independente, o GLKH chega a utilizar em algumas instâncias 10 minutos para obter solução

em uma execução independente. Acredita-se que esta diferença no tempo de execução pode ser causada pela natureza da entrada fornecida aos métodos. Ambos os métodos recebem como entrada um grafo correspondente a uma instância do GTSP. Entretanto, enquanto o ILS resolve a instância sem efetuar nenhuma transformação prévia, o GLKH transforma o grafo da entrada em uma instância do ATSP e depois a resolve. Desta forma a razão para a diferença nos tempos de execução pode estar na forma como o GLKH obtém uma solução. Como o GLKH utiliza o LKH para obter uma solução para o ATSP e em seguida a converte para uma solução equivalente do GTSP, há também certo consumo de tempo devido a mais esta conversão.

As instâncias que não foram solucionadas por nenhum método compartilham de uma característica em comum: o tamanho dos arquivos de entrada gerados a partir das instâncias. Mesmo dentro de conjuntos como E ou Cr_2 no qual é possível obter soluções para alguns subconjuntos, há instâncias que geram arquivos maiores do que a memória primária disponível. Como os métodos utilizados dependem de todas as informações dos grafos presentes em memória primária e é inviável fazer vários acessos a disco para verificar ou consultar dados sobre a instância, devido ao alto consumo de tempo destas operações, instâncias maiores que 14 *gigabytes* não foram solucionadas. Quanto maior for a disparidade entre a quantidade de ferramentas disponíveis e a capacidade do magazine ou a diferença entre a quantidade de ferramentas exigidas pelas tarefas e o tamanho do magazine, maior o número de vértices gerados para cada instância. Desta forma, o tamanho dos arquivos tende a crescer muito rápido, ainda que não seja de forma exponencial.

Capítulo 6

Conclusões

Este trabalho abordou o problema de otimização combinatória que busca minimizar o número de trocas de ferramentas em uma máquina flexível, o Problema de Minimização de Trocas de Ferramentas (SSP, do inglês *Job Sequencing and Tool Switching Problem*). É possível encontrar na literatura a modelagem frequente do SSP como um Problema do Caixeiro Viajante. Porém, após o estudo de Azevedo e Carvalho (2017) comprovar que tal modelagem apresenta problemas, faz-se necessário explorar outras possibilidades. Aborda-se então um método de solução alternativo que consiste em modelar o SSP como Problema do Caixeiro Viajante Generalizado (GTSP). A modelagem apresentada é pouco explorada na literatura fazendo com que seja necessário investigá-la.

Por meio de testes utilizando os métodos heurísticos ILS e GLKH foi possível constatar que a modelagem é promissora e que tais métodos são capazes de encontrar soluções de qualidade para as instâncias presentes na literatura. O ILS encontra com menos frequência os valores ótimos conhecidos para as instâncias, porém encontra soluções em baixo tempo computacional e com distância máxima de aproximadamente 6% das melhores soluções conhecidas. Esta distância percentual é baixa e pode ser considerada pouco relevante se o tempo para se obter uma solução for de alta prioridade. Entretanto o GLKH encontra com maior frequência as melhores soluções conhecidas ao custo de um tempo computacional muito maior para se obter uma solução. Desta forma, pode-se concluir que a preferência de um método sobre o outro em um contexto industrial depende da prioridade de tempo ou da qualidade das soluções.

Como trabalhos futuros, pretende-se buscar novas estratégias para gerar os vértices, de forma a diminuir o espaço ocupado pelos arquivos de entrada. Tal estratégia pode por exemplo, levar em consideração a frequência com que as ferramentas são utilizadas por tarefas. Desta forma, talvez seja possível gerar vértices que pareçam ter mais chances de serem utilizados nas soluções. Outra estratégia consiste na utilização de computação distribuída, com o objetivo de alocar um ou alguns *clusters* em diferentes computadores, com o objetivo de permitir a exploração de grafos maiores. Entretanto, os custos de comunicação de rede quanto a tempo podem não favorecer tal ideia.

Referências Bibliográficas

Adjiashvili, D.; Bosio, S. e Zemmer, K. (2015). Minimizing the number of switch instances on a flexible machine in polynomial time. *Operations Research Letters*, 43(3):317–322.

Ahmadi, E.; Goldengorin, B.; Süer, G. A. e Mosadegh, H. (2018). A hybrid method of 2-tsp and novel learning-based ga for job sequencing and tool switching problem. *Applied Soft Computing*, 65:214 – 229.

Al-Fawzan, M. A. e Al-Sultan, K. S. (2003). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering*, 44(1):35–47.

Amaya, J. E.; Cotta, C. e Fernández, A. J. (2008). A memetic algorithm for the tool switching problem. In Blesa, M. J.; Blum, C.; Cotta, C.; Fernández, A. J.; Gallardo, J. E.; Roli, A. e Sampels, M., editores, *H*ybrid Metaheuristics, number 5296 in Lecture Notes in Computer Science, pp. 190–202. Springer.

Amaya, J. E.; Cotta, C. e Fernández-Leiva, A. J. (2011). Memetic cooperative models for the tool switching problem. *Memetic Computing*, 3(3):199–216.

Amaya, J. E.; Cotta, C. e Fernández-Leiva, A. J. (2013). Cross entropy-based memetic algorithms: An application study over the tool switching problem. *International Journal of Computational Intelligence Systems*, 6(3):559–584.

Avci, S. e Akturk, M. S. (1996). Tool magazine arrangement and operations sequencing on cnc machines. *Computers & operations research*, 23(11):1069–1081.

Azevedo, T. N. D. e Carvalho, M. (2017). Uma avaliação precisa da modelagem do problema de minimização de troca de ferramentas como o problema do caixeiro viajante. In *Anais do XLIX Simpósio Brasileiro de Pesquisa Operacional*.

Bard, J. F. (1988). A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, 20(4):382–391.

Beezão, A. C.; Cordeau, J.-F.; Laporte, G. e Yanasse, H. H. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, 257(3):834–844.

Burger, A. P.; Jacobs, C.; van Vuuren, J. H. e Visagie, S. (2015). Scheduling multi-colour print jobs with sequence-dependent setup times. *Journal of Scheduling*, 18(2):131–145.

Calmels, D. (2018). The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends. *International Journal of Production Research*, 0(0):1–21.

Catanzaro, D.; Gouveia, L. e Labbé, M. (2015). Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, 244(3):766 – 777.

Chaves, A. A.; Lorena, L. A. N.; Senne, E. L. F. e Resende, M. G. C. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174–183.

Chaves, A. A.; Yanasse, H. H. e Senne, E. L. F. (2012). Uma nova heurística para o problema de minimização de trocas de ferramentas. *Gestão & Produção*, 19(1):17–30.

Crama, Y. (1997). Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153.

Crama, Y.; Kolen, A. W. J.; Oerlemans, A. G. e Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1):33–54.

Crama, Y.; Moonen, L. S.; Spieksma, F. C. e Talloen, E. (2007). The tool switching problem revisited. *European Journal of Operational Research*, 182(2):952 – 957.

Denizel, M. (2003). Minimization of the number of tool magazine setups on automated machines: A lagrangean decomposition approach. *Operations Research*, 51(2):309–320.

Djellab, H.; Djellab, K. e Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64(1):165–176.

Fathi, Y. e Barnette, K. W. (2002). Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, 40(1):151–164.

Furrer, M. e Mütze, T. (2017). An algorithmic framework for tool switching problems with multiple objectives. *European Journal of Operational Research*, 259(3):1003 – 1016.

Gendreau, M.; Hertz, A. e Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.

Ghiani, G.; Grieco, A. e Guerriero, E. (2007). An exact solution to the tlp problem in an nc machine. *Robotics and Computer-Integrated Manufacturing*, 23(6):645–649.

Ghiani, G.; Grieco, A. e Guerriero, E. (2010). Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks*, 55(4):379–385.

Ghrayeb, O. A.; Phojanamongkolkij, N. e Finch, P. R. (2003). A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. *International journal of production research*, 41(16):3849–3860.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130.

Helsgaun, K. (2015a). Site do código fonte do framework glkh. http://akira.ruc.dk/~keld/research/GLKH/.

Helsgaun, K. (2015b). Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287.

Hertz, A.; Laporte, G.; Mittaz, M. e Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions*, 30(8):689–694.

Hertz, A. e Widmer, M. (1996). An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, 65(1-3):319–345.

Konak, A. e Kulturel-Konak, S. (2007). An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system. In *Computational Intelligence in Scheduling, 2007. SCIS'07. IEEE Symposium on*, pp. 43–48. IEEE.

Konak, A.; Kulturel-Konak, S. e Azizoğlu, M. (2008). Minimizing the number of tool switching instants in flexible manufacturing systems. *International journal of production economics*, 116(2):298–307.

Laporte, G.; Salazar-Gonzalez, J. J. e Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36(1):37–45.

Lien, Y.-N.; Ma, E. e Wah, B. W.-S. (1993). Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem. *Information Sciences*, 74(1):177 – 189.

Lourenço, H. R.; Martin, O. C. e Stützle, T. (2003). Iterated local search. In Glover, F. e Kochenberger, G. A., editores, *H*andbook of Metaheuristics, pp. 320–353. Springer US, Boston, MA.

Marvizadeh, S. Z. e Choobineh, F. F. (2013). Reducing the number of setups for CNC punch presses. *Omega*, 41(2):226–235.

Matzliach, B. e Tzur, M. (1998). The online tool switching problem with non-uniform tool size. *International Journal of Production Research*, 36(12):3407–3420.

Moreira, A. C. B. (2016). *O problema de minimização de trocas de ferramentas*. PhD thesis, Universidade de São Paulo.

Paiva, G. S. e Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers and Operations Research*, 88:208 – 219.

Privault, C. e Finke, G. (1995). Modelling a tool switching problem on a single NC-machine. *Journal of Intelligent Manufacturing*, 6(2):87–94.

Privault, C. e Finke, G. (2000). k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, 96(1-4):255–269.

Raduly-Baka, C. e Nevalainen, O. S. (2015). The modular tool switching problem. *European Journal of Operational Research*, 242(1):100–106.

Reinsma, J.; Penna, P. H. V. e Souza, M. J. F. (2018). Um algoritmo simples e eficiente para resolução do problema do caixeiro viajante generalizado. In *Anais do L Simpósio Brasileiro de Pesquisa Operacional*.

Salonen, K.; Raduly-Baka, C. e Nevalainen, O. S. (2006). A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering*, 50(4):458–465.

Senne, E. L. F. U. e Yanasse, H. H. (2009). Beam search algorithms for minimizing tool switches on a flexible manufacturing system. *Proceedings of The 11th Wseas International Conference on Mathematical and Computational Methods In Science and Engineering (MACMESE '09)*, pp. 68–72.

Sethi, A. K. e Sethi, S. P. (1990). Flexibility in manufacturing: a survey. *International journal of flexible manufacturing systems*, 2(4):289–328.

Shirazi, R. e Frizelle, G. D. M. (2001). Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research*, 39(15):3547–3560.

Song, C.-Y. e Hwang, H. (2002). Optimal tooling policy for a tool switching problem of a flexible machine with automatic tool transporter. *International Journal of Production Research*, 40(4):873–883.

Tang, C. S. e Denardo, E. V. (1988). Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches. *Operations Research*, 36(5):767–777.

Tzur, M. e Altman, A. (2004). Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions*, 36(2):95–110.

Yanasse, H. H. e Lamosa, M. J. P. (2006a). An application of the generalized travelling salesman problem: the minimization of tool switches problem. In *International Annual Scientific Conference of the German Operations Research Society*, pp. 90–100, Bremen, Germany.

Yanasse, H. H. e Lamosa, M. J. P. (2006b). On solving the minimization of tool switches problem using graphs. In *XII International Conference on Industrial Engineering and Operations Management*, pp. 1–9, Fortaleza, Brazil.

Yanasse, H. H.; Rodrigues, R. d. C. M. e Senne, E. L. F. (2009). Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão & Produção*, 16(3):370–381.

Zeballos, L. (2010). A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 26(6):725–743.

Zhou, B.-H.; Xi, L.-F. e Cao, Y.-S. (2005). A beam-search-based algorithm for the tool switching problem on a flexible machine. *The International Journal of Advanced Manufacturing Technology*, 25(9-10):876–882.