TÚLIO NEME DE AZEVEDO

Orientador: Marco Antonio Moreira Carvalho

UM ALGORITMO HEURÍSTICO APLICADO A PRODUÇÃO EM SISTEMAS DE MANUFATURA FLEXÍVEIS

Ouro Preto

Agosto de 2017

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

UM ALGORITMO HEURÍSTICO APLICADO A 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.

TÚLIO NEME DE AZEVEDO

Ouro Preto
Agosto de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Um Algoritmo Heurístico Aplicado a Produção em Sistemas de Manufatura Flexíveis

TÚLIO NEME DE AZEVEDO

Monografia defendida e aprovada pela banca examinadora constituída por:

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

Dr. Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Dr. Puca Huachi Vaz Penna
Universidade Federal de Ouro Preto

Ouro Preto, Agosto de 2017

Resumo

O Problema de Minimização de Troca de Ferramentas (MTSP) é um problema combinatório encontrado em linhas de produção industriais que se caracteriza por processar um conjunto de diferentes tarefas em uma única máquina flexível. O processamento de cada tarefa requer que um conjunto específico de ferramentas seja alocado na máquina flexível, que possui uma capacidade fixa para comportá-las. Como as tarefas necessitam de conjuntos de ferramentas diferentes, é necessário que as ferramentas alocadas na máquina sejam trocadas durante o processamento das mesmas, respeitando a capacidade máxima da máquina, de forma que todas as tarefas sejam concluídas. A cada troca de ferramenta realizada, a máquina deve ser desligada, interrompendo a linha de produção, o que não é desejável. Assim, é necessário sequenciar as tarefas de forma a minimizar o número de trocas de ferramentas necessárias, minimizando o tempo ocioso da máquina flexível. O MTSP foi definido como $\mathcal{NP}$-Difícil, e vários autores tentaram solucionar o problema utilizando diversas modelagens. Um tipo de modelagem bastante utilizada é a que envolve o clássico Problema do Caixeiro Viajante (PCV). Neste trabalho, foi revisitada a modelagem do MTSP via PCV, considerando cinco definições de custo diferentes, encontradas na literatura. Para realizar uma análise precisa desta modelagem, o problema resultante foi resolvido de maneira exata utilizando-se o resolvedor Concorde.

Abstract

The Minimization of Tool Switches Problem (MTSP) is a combinatorial problem found in industrial production lines, which is characterized by processing a set of different tasks in a single flexible machine. The processing of each task requires that a specific set of tools be allocated to the flexible machine, which has a fixed capacity to support them. As the tasks require different tool sets, it is necessary that the tools allocated in the machine be changed during their processing, respecting the maximum capacity of the machine, so that all the tasks are completed. At each tool change, the machine must be switched off, interrupting the production line, which is undesirable. Therefore, tasks need to be sequenced in order to minimize the number of tool changes required, minimizing the downtime of the flexible machine. The MTSP was defined as $\mathcal{NP}$-Hard, and several authors tried to solve the problem using several models. A very popular type of modeling is the one that involves the classic Traveling Salesman Problem (TSP). In this work, the MTSP modeling via TSP is revisited considering five cost definitions found in the literature. In order to perform a precise evalutarion of this modeling, the resulting problem was exactly solved by means of the Concorde TSP solver.

Reescrever o abstract. Está muito “aportuguesado”

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com a evolução frequente da tecnologia, as indústrias que utilizam máquinas flexíveis em suas linhas de produção recorrem a métodos computacionais para resolverem problemas operacionais. Por sua vez, esses métodos, com suas respectivas especificidades, podem contribuir significativamente para otimização da eficiência das linhas de produção.

Uma *máquina flexível* tem como característica proporcionar dinamismo à linha de produção ao fabricar diversos tipos de produtos não relacionados entre si. Este tipo de máquina comporta um número fixo de *ferramentas* instaladas, por exemplo, chaves de fenda, alicates e brocas, entre outras. Diferentes conjunto de ferramentas produzem diferentes tipos de produtos. Cada produto exige que um determinado conjunto de ferramentas esteja instalado no momento de sua produção.

A capacidade da máquina de produção é suficiente para suportar o conjunto de ferramentas necessárias para a fabricação de cada produto isoladamente, porém, não comporta todas as ferramentas necessárias para fabricar todos os produtos simultaneamente. Durante a fabricação dos produtos, trocas de ferramentas serão necessárias para que a capacidade não seja ultrapassada e o produto subsequente seja processado. Desta maneira, cada troca exige que a máquina seja desligada, interrompendo a linha de produção, ocasionando aumento de custo de produção e ociosidade.

Um dos problemas presentes nas indústrias que empregam máquina flexíveis, é o *Problema de Minimização de Troca de Ferramentas* (MTSP). Este problema consiste em determinar a sequência de fabricação dos produtos a cada estágio nas linhas de produção. A fabricação de um produto é considerada uma *tarefa* a ser realizada por uma máquina flexível. O objetivo do MSTP é otimizar a linha de produção pela minimização das trocas de ferramentas necessárias. Desta forma, a produção é interrompida minimamente, tornando-a mais eficiente.

O MTSP se divide em duas partes: o problema de carregamento de ferramentas e o problema de sequenciamento de tarefas, como definido por Tang e Denardo (1988). O problema de carregamento de ferramentas, dada uma sequência fixa de tarefas, consiste em determinar o menor número de trocas de ferramentas para processar o conjunto de tarefas. O método *Keep*

Tool Neeeded Soonest (KTNS), proposto por Tang e Denardo (1988), soluciona este problema em tempo determinístico polinomial. Quando troca de ferramentas são necessárias, a política KTNS garante que as ferramentas que serão necessárias mais brevemente pelas tarefas ainda não processadas sejam mantidas na máquina. O problema de sequenciamento de tarefas é $\mathcal{NP}$-Difícil (Crama et al., 1994), significando que não se conhece algoritmo eficiente para sua solução. Sendo assim se faz necessário resolver o problema de sequenciamento das tarefas.

O MTSP se apresenta de diferentes formas na literatura. Este trabalho aborda o caso geral, no qual o tempo de instalação é o mesmo para todas ferramentas e o tamanho e custo das ferramentas são uniformes, ou seja, não importa qual a posição as ferramentas são alocadas na máquina flexível.

Falta uma ligação com o parágrafo anterior

Este trabalho propõe a análise crítica de uma modelagem amplamente adotada na literatura. Em trabalhos anteriores, após a modelagem utiliza-se métodos heurísticos para solução do problema, o que impede a análise precisa da modelagem adotada. Para conferir precisão à analise, a modelagem é realizada conforme reportado em diferentes trabalhos da literatura e resolvida pela primeira vez de maneira exata. A introdução de um resolvedor exato permite que conclusões inequívocas sobre a qualidade da modelagem sejam realizadas.

1.1 Motivação

O problema considerado neste trabalho possui ampla aplicação prática em sistemas de manufatura flexível. Além da aplicabilidade prática, trata-se de um problema $\mathcal{NP}$-Difícil, o que gera interesse teórico.

Especificamente, a modelagem considerada neste trabalho é utilizada em diversos trabalhos da literatura, porém, sem a realização de uma análise aprofundada de sua qualidade. Desta maneira, não é possível determinar se os resultados reportados na literatura são devidos à modelagem utilizada ou à qualidade dos métodos empregados para solução da mesma.

1.2 Objetivos

O trabalho de pesquisa abordado neste projeto se propõe a estudar o Problema de Minimização de Trocas de Ferramentas em sistemas de manufatura flexíveis, revisitando sua modelagem pelo Problema do Caixeiro Viajante e utilizando o resolvedor exato Concorde para sua solução. Desta forma, é possível avaliar precisamente a qualidade da modelagem empregada amplamente na literatura sobre o MTSP. São os objetivos específicos:

1. Realizar pesquisa para geração de embasamento teórico e revisão bibliográfica sobre o tema tratado;

2. Realizar pesquisa sobre a modelagem do Problema de Minimização de Trocas de Ferramentas como o Problema do Caixeiro Viajante;

3. Realizar pesquisa para geração de embasamento teórico a respeito do resolvedor Concorde;

4. Implementar a modelagem e solução exata do Problema de Minimização de Trocas de Ferramentas baseado Problema do Caixeiro Viajante; e

5. Avaliar o método implementado considerando as definições disponíveis na literatura e problemas teste publicamente disponíveis, realizando uma análise crítica considerando outros métodos da literatura.

1.3 Organização

O restante deste trabalho está organizado como descrito a seguir. O Capítulo 2 relata os trabalhos apresentados na literatura. Os fundamentos teóricos do MTSP e do PCV, juntamente com a descrição do resolvedor Concorde são apresentados no Capítulo 3. O Capítulo 4 detalha as definições de custo propostas na literatura para a modelagem do MTSP baseada no PCV. O planejamento de atividades restantes para a conclusão da Monografia como um todo é apresentado no Capítulo 5. Por fim, as conclusões são apresentadas no Capítulo 6.

Capítulo 2

Revisão da Literatura

Tang e Denardo (1988) foram os primeiros a relatarem na literatura uma abordagem ao MTSP, sugerindo a modelagem do problema como o Problema do Caminho Hamiltoniano Mínimo em um grafo completo em que os vértices representam as tarefas e cujos pesos das arestas foi definido como um limitante inferior para o número de troca de ferramentas entre duas tarefas específicas. Contudo, a principal contribuição deste trabalho foi a política ótima *Keep Tool Needed Soonest* (KTNS) para troca de ferramentas. Dada uma sequência fixa de tarefas a serem realizadas, esta política define o menor número de troca de ferramentas correspondente. Se trocas forem necessárias, a política garante que as ferramentas que serão necessárias mais brevemente pelas tarefas ainda não realizadas sejam mantidas na máquina flexível.

No mesmo ano, Bard (1988) formulou o MTSP como um problema de programação inteira não linear e o resolveu com uma heurística baseada na relaxação dual do modelo. O conjunto de instâncias utilizados nos experimentos não teve sua origem relatada. Todavia, os resultados demonstraram a necessidade de um tempo de execução relativamente pequeno para determinar boas soluções locais.

Crama et al. (1994) relacionaram o MTSP com o Problema do Caixeiro Viajante (PCV), o que inovou a modelagem do MTSP. Neste trabalho foram aplicadas heurísticas específicas para o PCV na resolução do MTSP. Também foi definida a complexidade computacional do problema, $\mathcal{NP}$-Difícil para $C \geq 2$. O que é C?

Posteriormente, Hertz et al. (1998) utilizaram a mesma modelagem, embutindo a política KTNS em heurísticas existentes para solução do PCV. Estas heurísticas incluem GENI e GENIUS (Gendreau et al., 1992) e *Farthest Insertion* (Golden e Stewart, 1985). As heurísticas GENI e *Farthest Insertion* consistem em métodos construtivos, ou seja, geram soluções gradativamente, baseando-se em métodos de inserção. A heurística GENIUS utiliza o mesmo princípio de GENI, porém, com uma fase de pós-otimização, denominadas US. Neste mesmo trabalho foram apresentadas definições formais para o cálculo das distâncias entre as tarefas, utilizadas como pesos das arestas nos métodos de Crama et al. (1994) e foi definida uma fun-

ção objetivo para o PCV que considera o KTNS como um de seus componentes. Dentre os métodos citados, a versão do GENIUS que incorpora o KTNS obteve o melhor desempenho.

O primeiros trabalho a utilizar dados reais oriundos das indústrias é devido a Shirazi e Frizelle (2001), provando que os métodos existentes na literatura eram melhores que os utilizados nas indústrias. Observou-se que as instâncias reais possuíam máquinas com capacidade elevada em relação as instâncias da literatura, de forma que as instâncias reais possuíam solução mais fácil do que as da literatura.

Diferentes problemas de sequenciamento de produção foram relacionados por Linhares e Yanasse (2002), o Problema de Pilhas Abertas (*Minimization of Open Stacks Problem*, MOSP), o Problema de Minimização de Descontinuidades (*Minimization of Discontinuities Problem*, MDP) e Problema de Minimização de Espalhamento de Ordens (*Minimization of Order Spread Problem*, MORP). Entre os resultados teóricos deste trabalho, destaca-se a prova de que MTSP e MOSP, apesar de relacionados, não são equivalentes entre si.

Al-Fawzan e Al-Sultan (2003) propuseram implementações de Busca Tabu para solução do MTSP, utilizando diferentes estratégias e métodos probabilísticos para reduzir o espaço de busca. Adicionalmente, foram utilizados dois métodos para geração de soluções aleatórias, *Random Swapping*, que realiza trocas entre pares de tarefas, e *Random Block Insertion*, que realiza trocas entre pares de blocos de tarefas consecutivas. Comparados com a Busca Tabu original, a versão mista, ou seja, a que utilizou estratégias probabilísticas e de oscilação, apresentou melhores resultados.

Os métodos *branch-and-cut* e *branch-and-bound* foram aplicados à solução do MTSP por Laporte et al. (2004). Neste trabalho foi proposta uma nova função objetivo, no intuito de melhorar a relaxação linear de um modelo baseado no PCV. Essa nova proposta introduziu restrições oriundas do PCV à formulção do MTSP reportada por Tang e Denardo (1988), resultando assim na nova função objetivo. O *branch-and-bound* foi capaz de solucionar instâncias com até 25 tarefas e 25 ferramentas em tempo computacional aceitável. Um novo *branch-and-bound* foi a base para a implementação do *beam search* de Zhou et al. (2005), que reduziu o espaço de busca ao limitar o número de nós na árvore de busca. Os resultados apresentados foram melhores que os resultados reportados por Bard (1988).

Privault e Finke (2000) seguiram a linha de pesquisa que utiliza métodos específicos para resolução do PCV para solucionar o MTSP. Foi proposta uma adaptação de um método para o problema de localização de k-servidores móveis para grandes lotes de requisições, gerando o método *Bulk-Service Partitioning Algorithm* (BSPA). Desta forma, os servidores foram representados pelos *slots* de ferramentas da máquina e os pedidos em grandes lotes pelos conjuntos de ferramentas necessárias a uma tarefa. Os resultados obtidos foram comparados aos de métodos específicos para o PCV, sendo a heurística *Farthest Insertion* utilizada para instâncias esparsas, e os clássicos *2-opt* e *Shortest Edge* aliados a um limite inferior utilizados para instâncias densas. O método BSPA apresentou boas soluções, porém o tempo computacional

foi muito maior que os comparados.

Djellab et al. (2000) propuseram uma modelagem do MTSP através de hipergrafos, associada ao método *Iterative Best Insertion*, além de dois métodos baseados no KTNS para definição do menor número de troca de ferramentas dada a sequência de tarefas. Em seus experimentos, foram consideradas apenas instâncias geradas pelos próprios autores, sem comparação com nenhum outro método da literatura.

Novamente modelando o MTSP com base em outros problemas combinatórios, Yanasse e Pinto (2002) propuseram uma modelagem. Desta vez, a modelagem proposta foi associada ao Problema de Fluxo de Custo Mínimo em Redes. Não foram realizados experimentos computacionais, porém, foram realizadas observações sobre algumas instâncias presentes na literatura e relatou-se que a proposta demonstra ser promissora. Posteriormente, Yanasse e Lamosa (2005) enunciaram uma proposta de modelagem utilizando o Problema do Caixeiro Viajante Generalizado, implementada por Yanasse e Lamosa (2006), porém, experimentos computacionais não foram realizados, sendo este apenas um artigo teórico.

Uma extensão do MTSP que considera ferramentas de tamanho não uniforme, foi abordada poe Tzur e Altman (2004). Neste trabalho, foi proposto o método *Aladdin*, baseado no melhor método existente até então para resolver o MTSP original, o GENIUS. Este mesmo método fora utilizado anteriormente por Hertz et al. (1998). Neste mesmo trabalho foi proposta a política *Keep Smaller Tools Needed Soonest*, para determinação do número de troca de ferramentas para a extensão do MTSP abordada.

Posteriormente, Crama et al. (2007) consideraram esta mesma extensão do MTSP. Concluiu-se que, quando a capacidade de ferramentas na máquina é fixa, o problema pode ser resolvido em tempo determinístico polinomial, utilizando um método proposto no mesmo trabalho. Apesar da complexidade polinomial, este método apresenta alto custo (polinômio de ordem alta) para resolução do problema.

Um algoritmo memético foi utilizado por Amaya et al. (2008), cuja implementação utilizou uma variação de algoritmo genético e *Hill Climbing*. Os resultados foram comparados com o *Beam Search* de Zhou et al. (2005) e considerados melhores.

Outro *branch-and-bound* foi proposto por Yanasse et al. (2009), juntamente com um método para determinar limitantes inferiores para o valor das soluções. Os resultados foram comparados com os resultados obtidos por Laporte et al. (2004). O *branch-and-bound* foi capaz de resolver instâncias não resolvidas anteriormente pelo método comparado. Entretanto, este método não foi capaz de resolver outras instâncias, como alguns casos com 25 tarefas.

Baseados neste último *branch-and-bound*, Senne e Yanasse (2009) empregaram o método *Beam Search*, que seleciona regiões promissoras do espaço de busca, com o objetivo de determinar limitantes superiores para acelerar os métodos já existentes. Os resultados obtidos foram comparados com os obtidos por Laporte et al. (2004). O método foi capaz de obter 92% de soluções entre ótimas e melhores soluções existentes.

A pesquisa em algoritmos meméticos teve sequência por Amaya et al. (2012). Neste trabalho, o algoritmo memético foi relacionado com técnicas utilizadas em problemas de otimização em redes. Os resultados foram comparados com os de Amaya et al. (2008), superando-os.

Uma nova modelagem em grafos, porém, não relacionada ao PCV, foi apresentada por Chaves et al. (2012). A solução do MTSP foi dividida em duas fases: construtiva e refinamento. Na fase construtiva gera-se uma solução viável e, posteriormente, a fase de refinamento é responsável por encontrar o máximo local para a solução inicial encontrada na primeira fase, através de uma Busca Local Iterada (*Iterated Local Search*,ILS). As soluções encontradas foram utilizadas como limitantes superiores para o algoritmo *Branch-and-Bound*, proposto anteriormente por Yanasse et al. (2009). O algoritmo apresentou uma diminuição de 74% dos nós gerados na árvore de busca e de 76% no tempo computacional.

Novamente, Amaya et al. (2013) propuseram diferentes formulações para algoritmos meméticos e genéticos. Desta vez, os algoritmos foram combinados com entropia cruzada e diferentes buscas locais. Os resultados foram comparados com os reportados em Amaya et al. (2012), superando-os. Foi também relatado que os algoritmos que utilizam a entropia cruzada são melhores que os que não utilizam, não havendo um método dominante.

Outra extensão do MTSP foi estudado por Raduly-Baka e Nevalainen (2015), considerando posições fixas para as ferramentas na máquina. Desta forma, uma ferramenta não pode ser instalada em qualquer posição livre não máquina flexível. Foi provado que esse problema é mais complexo que o MTSP original, sendo $\mathcal{NP}$-difícil no sentido forte. No caso em que o número de módulos alimentadores é fixo, pode-se encontrar a solução em tempo determinístico polinomial, porém, a ordem do polinômio é alta.

Dois modelos de programação linear inteira específicos para solucionar o MTSP foram propostos por Catanzaro et al. (2015). Os resultados reportados indicam que ambos os métodos apresentam melhor relaxação linear do que os métodos anteriores da literatura, que têm a mesma base conceitual. Uma política alternativa ao KTNS foi recentemente proposta por Adjiashvili et al. (2015). Embora o novo método exija tempo determinístico polinomial para determinar um plano de troca de ferramentas, assim como o KTNS, não é realizada comparação entre os dois.

Chaves et al. (2016) apresentaram um método baseado em *Clustering Search* e Algoritmo Genético de Chaves Aleatórias Viciadas (*Biased Random Key Genetic Algorithm* – BRKGA). A metaheurística *Clustering Search* identifica regiões promissoras dentro do espaço de busca e realiza a busca local pelo método Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND) nessas regiões. A metaheurística BRKGA foi utilizada para gerar soluções dentro da região do espaço de busca determinada pelo *Clustering Search*. Os resultados obtidos por Chaves et al. (2012) nas instâncias propostas por Yanasse et al. (2009) e Crama et al. (1994) foram comparados nos experimentos computacionais conduzidos. O novo método foi capaz de igualar ou superar todos os resultados anteriores para estas instâncias.

Mais recentemente, uma nova abordagem realizada sobre a estratégia de busca local iterada foi proposta por Paiva e Carvalho (2017). Neste trabalho foram propostos uma nova representação em grafos para o problema, uma nova heurística construtiva para a geração de soluções iniciais e um novo método de busca local. Os resultados do ILS foram comparados aos resultados obtidos por Chaves et al. (2016) nas instâncias de Yanasse et al. (2009) e Crama et al. (1994), além dos melhores resultados obtidos para as instâncias de Catanzaro et al. (2015). O novo método foi capaz de igualar ou superar todo os resultados anteriores para essas instâncias, sendo considerado o atual estado da arte para solução do MTSP.

Capítulo 3

Fundamentação Teórica

Neste capítulo são descritos formalmente o Problema de Minimização de Trocas de Ferramentas e o clássico Problema do Caixeiro Viajante. Também é apresentado brevemente o método exato que representa o estado da arte para solução do Problema do Caixeiro Viajante.

3.1 O Problema de Minimização de Trocas de Ferramentas

Formalmente, o MTSP é definido como descrito a seguir. Dados uma máquina flexível com capacidade de comportar até C ferramentas, um conjunto de tarefas $T = \{1, \ldots, n\}$, um conjunto de ferramentas $F = \{1, \ldots, m\}$, o subconjunto de ferramentas F_i ($F_i \in F$) necessárias para processar a tarefa i ($i \in T$), é necessário determinar uma permutação ϕ dos elementos de T tal que o número de trocas de ferramentas, obtido pela aplicação do algoritmo KTNS, seja minimizado.

Uma instância do MTSP apresenta informações sobre as ferramentas necessárias para o processamento de cada uma das tarefas. A Tabela 3.1 apresenta os dados de um cenário relacionado ao MTSP em que tem-se $n = 5$, $m = 5$, $C = 3$.

Tabela 3.1: Exemplo de instância MTSP.

Tarefas	Ferramentas
1	2, 3, 5
2	1, 3
3	1, 4, 5
4	1, 2
5	2, 3, 4

Na referida Tabela 3.1, a primeira coluna representa as tarefas a serem processadas pela máquina flexível, enumeradas de 1 a 5. A segunda coluna representa as ferramentas necessárias para que a tarefa referida seja processada. Computacionalmente, esta instância pode ser modelada por uma matriz binária Q, conforme apresentado pela Tabela 3.2. As linhas de Q

representam as ferramentas e as colunas representam as tarefas, ambas numeradas de 1 a 5. Os elementos q_{ij} da matriz Q são definidos $q_{ij} = 1$ caso a ferramenta i ($i \in F$) seja necessária para processar a tarefa j ($j \in T$). Caso contrário, $q_{ij} = 0$.

Tabela 3.2: Matriz Q.

Ferramentas\Tarefas	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	1
3	1	1	0	0	1
4	0	0	1	0	1
5	1	0	1	0	0

Uma solução para este problema é dada por um sequenciamento ϕ das colunas de Q. Por exemplo, considere $\phi = [3, 4, 2, 1, 5]$, representada pela Tabela 3.3. Esta solução resulta em 7 troca de ferramentas, explicitadas a seguir:

Tabela 3.3: Solução MTSP.

Tarefas	Ferramentas carregadas na máquina
3	1, 4, 5
4	1, 2, 5
2	1, 2, 3
1	2, 3, 5
5	2, 3, 4

1. Três trocas para carregar as ferramentas iniciais na máquina (ferramentas 1,4,5);
2. Uma troca entre a tarefa 3 e 4 (ferramenta 4 por 2);
3. Uma troca entre a tarefa 4 e 2 (ferramenta 5 por 3);
4. Uma troca entre a tarefa 2 e 1 (ferramenta 1 por 5); e
5. Uma troca entre a tarefa 1 e 5 (ferramenta 5 por 4).

Considerando a representação por matrizes binárias, a solução ϕ induz uma matriz permutação R^{ϕ}, vide Tabela 3.4, cujas colunas são as colunas de Q na ordem estabelecida por ϕ e uma coluna adicional 0 com todos os elementos nulos. Na referida Tabela 3.4, as ferramentas inseridas a cada instante na máquina estão sublinhadas e as ferramentas que permanecem na máquina, mesmo sem serem utilizadas, estão indicadas em negrito. Na troca da tarefa 3 para a tarefa 4, a ferramenta 2 foi inserida na máquina, portanto sublinhada. A ferramenta 5 foi mantida na máquina mesmo sem ser utilizada na tarefa 4, portanto, em negrito.

Tabela 3.4: Matriz R^{ϕ}.

Ferramentas\ϕ	0	3	4	2	1	5
1	0	1	1	1	0	0
2	0	0	1	**1**	1	1
3	0	0	0	1	1	1
4	0	1	0	0	0	1
5	0	1	**1**	0	1	0

O valor dos elementos r_{ij}^{ϕ} é definido de acordo com a Equação 3.1.

$$r_{ij}^{\phi} = \begin{cases} 1, \text{ se a ferramenta } i \in F \text{ estiver na máquina durante o processamento da tarefa } j \in T \\ 0, \text{caso contrário.} \end{cases}$$

Uma solução representada pela matriz R^{ϕ} é avaliada de acordo com a Equação 3.1, proposta por Crama et al. (1994). Esta função calcula, dada uma sequência de tarefas, o número de inversões de 0 para 1, que representam a inserção de ferramentas na máquina.

$$Z_{MTSP}^{\phi}(R) = \sum_{j \in T} \sum_{i \in F} r_{ij}^{\phi}(1 - r_{ij-1}^{\phi}) \tag{3.1}$$

O objetivo do MTSP é determinar a permutação $\phi \in \Phi$ das colunas da matriz Q que resulte no menor número de trocas de ferramentas, em que Φ é o conjunto de todas as permutações possíveis. A função objetivo correspondente é exibida na Equação 3.2.

$$\min_{\phi \in \Phi} Z_{MTSP}^{\phi}(Q) \tag{3.2}$$

Conforme mencionado anteriormente, o Problema de Minimização de Trocas de Ferramentas pertence à classe $\mathcal{NP}$-Difícil para os casos em que $C \geq 2$ (Crama et al., 1994).

3.2 O Problema do Caixeiro Viajante

Em seu livro, Cook (2012), uma referência sobre o estudo do PCV, relata que a origem do problema não possui datamento preciso, e faz uma análise das origens históricas do mesmo. Um relato indica que em 1925 uma empresa dos EUA, *Page Seed Company*, deixou para seu funcionário, *Mr. Cleveland*, uma lista de cidades para visitar em busca de negócios. *Mr. Cleveland* e a empresa fizeram observações que deixaram claro o objetivo de minimizar o tempo gasto na rota. Outro relato cita que em 1832, um livro alemão fez a descrição do PCV, em que é indicado que a idéia principal do problema é sempre visitar o máximo de cidades possíveis sem ter que visitá-las duas vezes. Um terceiro relato indica que os primeiros estudos matemáticos relacionados ao PCV foram datados de 1930, em Harvard e Viena, apresentando

o problema para a comunidade matemática.

O PCV é um problema combinatório que em sua versão de otimização consiste em, dadas uma lista de n cidades e as distâncias entre elas, determinar uma rota que comece e termine em uma mesma cidade e passe pelas demais cidades exatamente uma vez, com o menor custo possível. A Figura 3.1 demonstra o problema, através de um grafo, no qual os vértices representam as cidades e as arestas representam as ligações entre duas cidades, cujas distâncias são representadas pelos pesos das arestas.

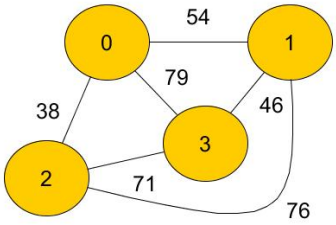

Figura 3.1: Representação de uma instância do PCV.

Computacionalmente, uma instância do PCV pode ser modelada por uma matriz W $n \times n$, exemplificada na Tabela 3.5. Na matriz, as linhas e colunas representam cada uma das n cidades do problema, e cada elemento w_{ij} da tabela, indica a distância entre as cidades i e j $(i, j = 1 \ldots n)$. Note que $w_{ij} = w_{ji}$ no caso simétrico e quando $i = j$, w_{ij} possui valor nulo.

Tabela 3.5: Matriz W.

	0	1	2	3
0	-	54	38	79
1	54	-	76	46
2	38	76	-	71
3	79	46	71	-

Dado o exemplo da Tabela 3.5, uma solução para o exemplo de instância referida é construída da seguinte maneira: seleciona-se uma cidade qualquer, e a partir dela seleciona-se as demais, de forma a construir um ciclo sem repetições de cidades. A primeira cidade escolhida é a que termina o ciclo. Por exemplo, uma solução $S = [0, 2, 1, 3, 0]$ demonstra que a o ciclo começa e termina na cidade 0, visitando as cidades 2, 1 e 3, nesta ordem, durante o percurso. O custo total da solução S é 239, calculada pela soma das distâncias entre as cidades, de acordo com a Tabela 3.5, apresentadas a seguir:

1. Distância da cidade 0 para 2: 38;
2. Distância da cidade 2 para 1: 76 (acumulado, 114);
3. Distância da cidade 1 para 3: 46 (acumulado, 160); e

4. Distância da cidade 3 para 0: 79 (acumulado, 239).

O objetivo do PCV é determinar a rota que possua o menor custo, ou seja, a solução ótima é aquela que minimiza a distância total percorrida. Para a instância referida na Tabela 3.5, a solução ótima é dado pela sequência $S = [0, 1, 3, 2, 0]$, que possui custo total 209.

O PCV, em sua versão de otimização possui complexidade $\mathcal{NP}$-Difícil (Laporte, 1992), significando que não se conhece algoritmo eficiente para sua solução. Embora haja diversas heurísticas clássicas para sua solução, estas heurísticas não possuem bom desempenho à medida em que o número cidades aumenta, tendendo ao aumento rápido do tempo de execução. Da mesma forma, métodos exatos exigem muito tempo para a solução exata, tornando-se impraticáveis.

Novamente, Cook (2012) destaca a evolução dos algoritmos criados para solução do PCV. O primeiro, que permaneceu como estado da arte por 17 anos, resolveu uma instância do problema contendo 49 cidades. Atualmente, o estado da arte é o resolvedor *Concorde*[1], utilizado para resolver o PCV entre outros problemas, como o roteamento de veículos, o mapeamento genético e outros. Em se tratando do PCV, este resolvedor foi utilizado para resolver problemas existentes na *TSPLIB*[2], uma biblioteca específica de casos de testes para o PCV. Em seus resultados, se destaca a solução ótima de todos os 110 casos de teste disponíveis, incluindo um caso de teste com 85.900 cidades, assim sendo o maior número de cidades resolvidas atualmente.

3.3 O Resolvedor Concorde

O resolvedor *Concorde* é uma ferramenta desenvolvida para solução do PCV disponível também para vários problemas de otimização relacionados. A ferramenta pode ser utilizada gratuitamente para fins acadêmicos, havendo também um servidor que permite a resolução de instâncias do problema *online*.

Codificado em ANSI C, apesar de contar com métodos heurísticos em seu código com mais de 700 funções, o Concorde é um método exato. A exatidão do resolvedor é devida à utilização do método de plano de cortes, que por sua vez resolve os problemas utilizando técnicas de relaxação linear. Verifica-se também o uso de técnicas para o cálculo de limitantes inferiores: Triangulação de *Delaunay*, Árvore Geradora Mínima e várias heurísticas do tipo *vizinho mais próximo*. Aliadas a esses métodos, são disponibilizadas também cinco heurísticas geradoras de soluções alternativas para o PCV: gulosa (não especificada), *Boruvka* (Boruvka, 1926), *Quick Boruvka* (Applegate et al., 2003), *Nearest Neighbor*, *Chained Lin-Kernighan* (Kernighan e Lin, 1970) e solução aleatória.

[1] `http://www.math.uwaterloo.ca/tsp/concorde`
[2] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

O resolvedor também possui suporte para programação em ambientes paralelos de memória compartilhada e uma interface gráfica opcional. Estas funções permitem ao usuário a criar códigos especializados para problemas que se relacionam com o PCV, como por exemplo o MTSP e problemas de otimização em redes. Também há suporte para os resolvedores de programação linear *QSopt*[3] e *CPLEX*[4].

Apesar de o resolvedor apresentar resultados expressivos para o PCV, a sua documentação está incompleta e não há suporte oficial para sua utilização, o que dificulta o entendimento e o uso do mesmo. De fato, o pacote é disponibilizado sem garantias de precisão ou corretude de resultados. Uma contribuição deste trabalho é a criação e disponibilização de um relatório técnico que descreve a instalação do resolvedor Concorde utilizando os resolvedores *CPLEX* ou *QSopt* em ambientes Linux e também a modelagem para os problemas e utilização das rotinas básicas para solução dos mesmos. Todos os tópicos são acompanhados de exemplos minimamente funcionais. Este relatório técnico está incluso no Apêndice A.

[3] `http://www.math.uwaterloo.ca/~bico/qsopt`
[4] `http://www.ibm.com/software/commerce/optimization/cplex-optimizer/`

Capítulo 4

O Problema de Minimização de Trocas de Ferramentas e o Problema do Caixeiro Viajante

Uma maneira comum para solucionar problemas de otimização consiste em realizar modelagens de problemas correlatos entre si que possuam métodos de solução de boa qualidade. Desta forma, a solução do problema original é composta da solução gerada pela modelagem para um segundo problema.

Embora o MTSP e o PCV não sejam equivalentes entre si no caso geral, uma modelagem comum, conforme mencionado no Capítulo 2, relaciona ambos os problemas considerando um grafo completo em que cada tarefa é representada por um vértice. Os pesos das arestas podem ser definidos de diferentes maneiras, de forma a converter o número de troca de ferramentas entre duas tarefas processadas contiguamente, característica do MTSP, na distância entre cidades, característica do PCV. Neste capítulo, revisitamos as diferentes formas de cálculo de distâncias propostas na literatura ao modelar o MTSP como o PCV.

Além da dificuldade introduzida pela definição dos pesos das arestas, a análise da literatura realizada na Seção 2 indica que somente métodos heurísticos foram considerados para solução dos modelos. Desta forma, as soluções geradas possuem, além da degradação da modelagem por si só, a degradação oriunda da solução por heurísticas, o que dificulta uma avaliação precisa da modelagem.

Hertz et al. (1998) descrevem cinco definições diferentes para o cálculo das distâncias d_{ij} $(i, j \in T)$ dada uma instância do MTSP. As três primeiras são conhecidas anteriormente na literatura e outras duas são novas propostas. Nas equações a seguir, C é a capacidade máxima da máquina e F_i $(F_i \in F)$ é conjunto de ferramentas necessárias a processar a tarefa i $(i \in T)$.

4.1 Definição 1

A primeira definição, apresentada na Equação 4.1, relaciona a capacidade máxima da máquina e o número de ferramentas em comum entre duas tarefas. Deste modo, determina-se quantas ferramentas permanecerão na máquina e a capacidade restante da máquina é utilizada como um limitante superior para o número de trocas.

$$d_{ij} = C - \mid F_i \cap F_j \mid \tag{4.1}$$

Esta definição foi originalmente proposta por Crama et al. (1994). Hertz et al. (1998) constatam que o MTSP se reduz ao PCV caso todas as tarefas T_i ($T_i \in T$) necessitem de exatamente C ferramentas para serem processadas. Além disso, reportaram que o caso geral apresenta $\mid T_i \mid < C$ ($T_i \in T$) e ainda assim tem-se algoritmos para o PCV que resultam em bons resultados heurísticos para o problema.

4.2 Definição 2

A definição apresentada pela Equação 4.2, também foi proposta originalmente por Crama et al. (1994). Considera-se como limitante superior para o número de troca de ferramentas entre duas tarefas a diferença simétrica entre os conjuntos de ferramentas necessárias para processar duas tarefas diferentes (i.e., a união dos conjuntos subtraída da interseção entre os mesmos).

$$d_{ij} = \mid F_i \cup F_j \mid - \mid F_i \cap F_j \mid \tag{4.2}$$

Uma observação válida sobre as Equações 4.1 e 4.2 é que quando $\mid F_i \mid = C$ as equações se tornam equivalentes. Adicionalmente, ambas resultam em valores altos caso duas tarefas i e j compartilhem poucas ferramentas entre si.

4.3 Definição 3

A terceira definição se baseia na proposta original por Tang e Denardo (1988) para cálculo de limitantes inferiores para o número de trocas de ferramentas, apresentada na Equação 4.3.

$$d_{ij} = \mid F_i \cup F_j \mid - C \tag{4.3}$$

Esta definição considera o número de ferramentas utilizadas por duas tarefas que excede a capacidade de máquina. Entretanto, é possível que valores negativos sejam produzidos, quando duas tarefas i e j exigirem um número baixo de ferramentas quando comparado à capacidade da máquina.

Crama et al. (1994) propôs uma adaptação da definição de custo de Tang e Denardo (1988) para evitar valores negativos. A adaptação consistem em definir a distância como zero caso a definição resulte em um valor negativo. A Equação 4.4 apresenta esta definição.

$$d_{ij} = \max\{0, \mid F_i \cup F_j \mid -C\} \tag{4.4}$$

Esta definição de distância é utilizada utilizada pela maioria dos autores que solucionaram o MTSP utilizando a modelagem baseada no PCV (Tang e Denardo, 1988; Crama et al., 1994; Hertz et al., 1998; Laporte et al., 2004; Privault e Finke, 2000; Catanzaro et al., 2015). Adicionalmente, Fathi e Barnette (2002) utilizaram esta definição como limitante inferior para o Problema de Sequenciamento em Máquinas Idênticas Paralelas com Restrições de Ferramentas.

As três primeiras definições (Equações 4.1, 4.2 e 4.4) levam em consideração somente duas tarefas subsequentes i e j, sem considerar os conjuntos de ferramentas carregadas na máquina antes do processamento da tarefa i e depois do processamento da tarefa j. Levando em consideração esta análise, Hertz et al. (1998) propôs duas novas definições, descritas nas seções a seguir.

4.4 Definição 4

A definição expressada pela Equação 4.5, aperfeiçoa a definição de custo anterior, adotando o critério de frequência de utilização para a permanência das ferramentas na máquina, carregadas anteriormente. Sendo assim, subtrai-se uma quantidade menor que C de ferramentas caso as ferramentas requeridas pelas tarefas i ou por j não forem necessárias antes de i ou depois de j. Caso as ferramentas requeridas por i ou por j forem requeridas antes de i ou depois de j, o número de ferramentas subtraídas será maior.

$$d_{ij} = \max\left\{0, \mid F_i \cup F_j \mid - \left[\theta \frac{\Lambda(ij)}{(n-2) \mid F_i \cup F_j \mid}\right] C\right\} \tag{4.5}$$

São utilizadas duas funções auxiliares. A primeira $\lambda_k(ij)$, indica o número de tarefas, excluindo-se as tarefas i e j, que requerem da ferramenta $k \in F_i \cup F_j$. A função $\Lambda(ij) = \sum_{k \in F_i \cup F_j} \lambda_k(ij)$, representa o total das frequência de todas as ferramentas utilizadas para processar duas tarefas i e j. Além disso, tem-se o parâmetro $\theta \in [0, 1]$. Assim, a definição subtrai de $\mid F_i \cup F_j \mid$ uma quantidade entre $[0, C]$, sendo esta quantidade alta quando as ferramentas de $F_i \cup F_j$ são utilizadas com frequência.

4.5 Definição 5

Seguindo o mesmo raciocínio da definição anterior, a definição representada pela Equação 4.6 também considera a frequência de utilização de ferramentas.

$$d_{ij} = \left(\left[\frac{c+1}{c} \right] \mid F_i \cup F_j \mid - \mid F_i \cap F_j \mid \right) \left[\frac{(n-2) \mid F_i \cup F_j \mid}{\max\{\Lambda(ij), 0.5\}} \right] \tag{4.6}$$

O fator $\frac{c+1}{c}$ é um valor entre $[1, 2]$ que proporciona um peso maior para o termo $\mid F_i \cup F_j \mid$ caso a capacidade máxima da máquina for pequena, i.e., se mais trocas de ferramentas são prováveis. A segunda parte da equação é no minimo 1, tornando-se maior se as ferramentas presentes em $F_i \cup F_j$ forem raramente utilizadas. O parâmetro 0.5 é utilizado para evitar a divisão por 0, quando $\Lambda(ij) = 0$. Em experimentos realizados com as heurísticas consideradas em seu trabalho, Hertz et al. (1998) reportaram que esta definição para o custo das arestas apresentou os melhores resultados dentre todas as descritas neste capítulo.

Capítulo 5

Plano de Atividades Restantes

Como planejamento para conclusão do projeto de Monografia, na Tabela 5.1 são apresentadas informações sobre atividades subsequentes, a serem realizadas na disciplina Monografia II.

Tabela 5.1: Planejamento de atividades para Monografia II.

Atividades	Mês 1	Mês 2	Mês 3	Mês 4
Implementação da modelagem	X	X		
Realização de experimentos computacionais		X	X	
Descrição dos experimentos			X	
Análise dos experimentos			X	X
Conclusão da Monografia				X

As atividades basicamente se concentram implementar a modelagem considerada neste trabalho e sua solução pelo Concorde. Serão consideradas as 5 definições para o cálculo de distância entre duas tarefas. O método então será submetido a experimentos computacionais e análises adicionais, comparando as definições entre si e com o atual estado da arte para solução do problema de Minimização de Trocas de Ferramentas.

Capítulo 6

Conclusões

Neste trabalho considerou-se o Problema de Minimização de Trocas de Ferramentas (ou MTSP), um problema de sequenciamento de tarefas em linhas de produção industrial. Na tentativa de solucionar este problema de aplicação prática, foram apresentadas diferentes modelagens desde o final da década de 80. Na literatura, uma forma comum de modelar o MTSP baseia-se no clássico Problema do Caixeiro Viajante (PCV). Embora os dois problemas não sejam equivalentes entre si no caso geral, considera-se cada tarefas como uma cidade, tal que existem diferentes formas para adaptação do número de troca de ferramentas, característica do MTSP, para a distância entre cidades, característica do PCV. Na literatura, estas diferentes adaptações foram aplicadas na modelagem do MTSP pelo PCV, que, via de regra, era resolvido por heurísticas clássicas, como 2-*opt* e *nearest-neighbor*, por exemplo.

Propôs-se então revisitar esta tradicional modelagem, constatando que os métodos presentes na literatura utilizam definições para adaptação do número de trocas de ferramentas em distâncias entre cidades. Ao todo são consideradas cinco definições para estas distâncias. Constatou-se também que o atual estado da arte para o PCV é o resolvedor exato Concorde, e que seus resultados possibilitariam uma análise exata sobre a tradicional resolução do MTSP modelado como o PCV.

Para a conclusão da Monografia, serão implementadas a modelagem conforme reportado na literatura e posterior solução utilizando o resolvedor Concorde, considerando as cinco definições de custo descritas neste trabalho. Após as implementações, serão realizados experimentos computacionais e análises precisas acerca da resolução do MTSP utilizando esta tradicional modelagem considerada neste trabalho.

Apêndice A

Guia de instalação e uso do resolvedor Concorde

Guia de instalação e uso do resolvedor Concorde

Túlio Neme de Azevedo - tulioneme10@gmail.com

Marco Antonio Moreira de Carvalho - marco.opt@gmail.com

Departamento de Computação

Universidade Federal de Ouro Preto - MG

20 de janeiro de 2017

Introdução

O *Concorde*[1] é um resolvedor para o clássico Problema do Caixeiro Viajante, do inglês *Traveling Salesman Problem* (TSP), e alguns problemas de otimização relacionados. Desenvolvido em ANSI C, o resolvedor possui os melhores resultados reportados até o momento, resolvendo todas as instâncias disponíveis na *TSPLIB*[2].

Este guia apresenta como instalar e utilizar o *Concorde* em um ambiente Linux, utilizando os resolvedores de programação linear QSOpt ou CPLEX. Todos os passos abaixo foram realizados em um computador com sistema operacional Ubuntu, porém, funcionam também para outras distribuições, incluindo as Mac OS.

[1] http://www.math.uwaterloo.ca/tsp/concorde

[2] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Instalação e compilação

Antes de tudo, verifique se sua máquina possui os resolvedores de programação linear QSopt ou CPLEX.

Obs: verifique a sua arquitetura de seu processador, faz diferença!

1. Qsopt: 64 bits[3] - 32 bits[4]
2. CPLEX[5] (é necessário fazer um cadastro no site da IBM e você escolhe a arquitetura)

A instalação requer passos diferentes dependendo do resolvedor de programação linear empregado, portanto foram dividas em duas subseções, uma para o QSopt e outra para o CPLEX.

[3] http://www.math.uwaterloo.ca/~bico/qsopt/beta/index.html

[4] http://www.math.uwaterloo.ca/~bico/qsopt/downloads/downloads.htm

[5] http://www-01.ibm.com/support/docview.wss?uid=swg24036489

QSopt

Após o *download* do QSopt, crie um link simbólico para o arquivo qsopt.a. No mesmo diretório dos arquivos do QSopt:

$ sudo ln -s qsopt.a libqsopt.a .

Faça o *download* do código-fonte do Concorde, disponível na seção *Download* do site oficial[6], (última versão disponibilizada em 19 de Dezembro de 2003).

[6] http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

O arquivo está compactado, e para descompactar utilize no terminal:

$ gunzip co031219.tgz

$ tar xvf co031219.tar

Em seguida selecione o diretório "concorde" criado e então configure:

$ cd concorde

$ ./configure –with-qsopt=DIR (diretório contendo "qsopt.h" e "qsopt.a")

Em seguida, rode o makefile do concorde:

$ make

Crie um link simbólico para a biblioteca do Concorde. No mesmo diretório do Concorde:

$ sudo ln -s concorde.a libconcorde.a .

Pronto, agora você pode resolver os problemas do TSP disponíveis na TSPLIB.

Se você quer modificar os parâmetros do resolvedor para resolver problemas *TSP-like*, então terá que corrigir alguns *bugs* eventuais gerados pelo Concorde.

Os desenvolvedores do Concorde adicionaram em algumas funções os parâmetros com identificadores "new" e "class", que em C++ são palavras reservadas, portanto, isso pode gerar erros de compilação.

Para resolver esse erro, basta editar o arquivo "concorde.h", na pasta de instalação do Concorde, e modificar (ou mesmo excluir) os identificadores "new" e "class".

Note que somente apagar as palavras resolve o problema, já que em C++ os *headers* não precisam ter o identificador dos parâmetros, apenas o tipo.

Obs: Tome cuidado para não fazer uma troca automática em todo o código e apagar mais do que deveria.

Para compilar um código que utilize o Concorde, é necessário ajustar alguns parâmetros. Sugiro um arquivo *makefile*, que irá facilitar a sua vida (DIRCONCORDE é o diretório da instalação do Concorde e DIRQSOPT é o diretório que contém os arquivos do QSOPT):

```
CXX= g++

CONCORDE=DIRCONCORDE

QSOPT=DIRQSOPT

CPPFLAGS= -w -m64

LIBFLAGS= -I$(QSOPT) -I$(CONCORDE) -L$(CONCORDE) -L$(QSOPT)
    -lconcorde -lqsopt -lm -lpthread
```

```
all:
    $(CXX) main.cpp -o teste $(LIBFLAGS) $(CPPFLAGS)

    @echo ''-- DONE --''


run:
    ./bin/main
```

Execute o *Makefile* (via comando *make*, no diretório em que se encontrar o *Makefile*) e o seu executável será gerado.

Pronto, tudo ok. SHOW TIME

CPLEX

Após o *download* do CPLEX, faça o *download* do código-fonte do Concorde, disponível na seção *Download* so site oficial do Concorde[7], (última versão disponibilizada em 19 de Dezembro de 2003).

[7] http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

O arquivo está compactado, e para descompactar utilize no terminal:

$ gunzip co031219.tgz

$ tar xvf co031219.tar

No diretório do Concorde, edite o arquivo "Makefile.in". Adicione -lpthread ao final da linha que define as "LIBFLAGS".

No mesmo diretório, selecione o diretório "TSP" e novamente edite o arquivo "Makefile.in" da mesma maneira que o anterior.

Usando o terminal, crie links simbólicos para os *headers* do CPLEX. No diretório de instalação do Concorde:

$ sudo ln -s /opt/ibm/ILOG/CPLEX_Studio1263/cplex/include/ilcplex/*.h .

$sudo ln -s /opt/ibm/ILOG/CPLEX_Studio1263/cplex/lib/x86-64_linux/static_pic/libcplex.a .

Após isto, no diretório "LP", edite o arquivo "lpcplex8.c", incluindo a linha #define CPX_PARAM_FASTMIP 1017. Esta constante era utilizada em versões anteriores do CPLEX, porém, foi removida das versões mais recentes.

Agora, configure o Concorde com o CPLEX, no diretório do Concorde:

$ ./configure –prefix=DIR (diretório do Concorde) –with-cplex=DIR (diretório do Concorde, que agora contém os links para os *headers* do CPLEX)

Rode o Makefile:

$ ./make

Crie um link simbólico para a biblioteca do Concorde. No mesmo

diretório do Concorde:

$ sudo ln -s concorde.a libconcorde.a .

Obs: em meus testes esse passo não foi concluido com sucesso, tendo sido reportado um erro no arquivo criado. Caso ocorra, exclua o arquivo "libconcorde.a" criado, faça uma cópia do arquivo "concorde.a" e o renomeie para "libconcorde.a".

Os desenvolvedores do Concorde adicionaram em algumas funções os parâmetros com identificadores "new" e "class", que em C++ são palavras reservadas, portanto, isso pode gerar erros de compilação.

Para resolver esse erro, basta editar o arquivo "concorde.h", na pasta de instalação do Concorde, e modificar (ou mesmo excluir) os identificadores "new" e "class".

Note que somente apagar as palavras resolve o problema, já que em C++ os *headers* não precisam ter o identificador dos parâmetros, apenas o tipo.

Obs: Tome cuidado para não fazer uma troca automática em todo o código e apagar mais do que deveria.

Para compilar um código que utilize o Concorde, é necessário ajustar alguns parâmetros. Sugiro um arquivo *makefile*, que irá facilitar a sua vida (DIRCONCORDE é o diretório da instalação do Concorde, com os links simbólicos para os *headers* do CPLEX):

```
CXX= g++

CONCORDE=DIRCONCORDE

CPPFLAGS= -w -m64

LIBFLAGS= -I$(CONCORDE) -L$(CONCORDE) -lconcorde -lcplex -lm
    -lpthread

all:
    $(CXX) main.cpp -o teste $(LIBFLAGS) $(CPPFLAGS)

    @echo ''-- DONE --''


run:
    ./bin/main
```

Execute o *Makefile* (via comando *make*, no diretório em que se encontrar o *Makefile*) e o seu executável será gerado.

Pronto, tudo ok. SHOW TIME

Utilização

Para esse guia consideraremos a linguagem C++, utilizando as estruturas presentes na STL.

Como a abordagem do resolvedor é para problemas *TSP-like*, é necessário realizar a modelagem correta do seu problema, considerando um grafo com n vértices.

Para tal se faz necessário a criação de uma matriz de distâncias entre dois vértices do seu problema, desta forma temos uma matriz de tamanho $n \times n$. Além disso, um vetor, de tamanho n, é necessário. Desta forma o resolvedor preencherá o vetor criado com a melhor rota entre os vértices que componham o seu problema.

Crie uma função que contenha como parâmetro a matriz de distâncias e o vetor, sendo o segundo o resultado que o Concorde retorna. Nesta função realizaremos a chamada da função do Concorde que transforma a matriz de distância na estrutura de dados aceita pelo resolvedor, denominada *CCutil_graph2dat_matrix* e em seguida faremos a chamada da função que resolve o problema, chamada *CCtsp_solve_dat*, retornando um inteiro, que possui alguns parâmetros que precisam ser ajustados:

Obs: Os parâmetros estão nomeados de acordo com as descrições[8] [9] relatadas no site do Concorde.

Descrição da função *CCutil_graph2dat_matrix*:

int ncont número total de vértices (n);

int ecount número total de arestas;

*int *elist* *array* indicando o inicio e o fim dos pesos das arestas (em pares);

*int *elen* *array* indicando o peso das arestas;

int defaultelen parametro *default* para o peso das arestas;

*CCdatagroup *dat* estrutura de dados especifica do concorde;

Descrição da função *CCtsp_solve_dat*:

int ncont número total de vértices (n);

*CCdatagroup *dat* estrutura de dados especifica do concorde (preenchida por uma função);

*int *in_tour* fornece uma solução inicial (pode ser NULL);

*int *out_tour* vetor contendo a sequência ótima dos vértices (pode ser NULL);

[8] http://www.math.uwaterloo.ca/tsp/concorde/DOC/util.html#CCutil_graph2dat_matrix

[9] http://www.math.uwaterloo.ca/tsp/concorde/DOC/tsp.html#CCtsp_solve_dat

*double *in_val* define um limite superior inicial para o valor da solução (pode ser NULL);

*double *optval* valor ótimo para a solução do Caixeiro Viajante;

*int *success* retorna 1 caso a função tenha executada corretamente, ou 0 se a execução foi interrompida precocemente (de acordo com alguns limites pre estabelecidos do Concorde);

*int *foundtour* retorna 1 se uma solução foi encontrada (caso o parâmetro *success* for 0, então o caminho pode não ser o ótimo);

*char *name* nomeação dos arquivos que são escritos durante a execução do algoritmo (caso seja NULL, por padrão, "noname" será utilizado, porém, em programas *multithreaded* isto gera problemas, portanto prefira especificar algo);

*double *timebound* limite superior para o tempo de execução;

*int *hit_timebound* retorna 1 se o *timebound* for atingido, ou 0 caso contrário (pode ser nulo);

int silent caso diferente de 0, boa parte da saída será suprimida;

*CCrandstate *rstate* é utilizado pelo gerador de números aleatórios do Concorde;

Obs: Para realizar a chamada do resolvedor é necessária a inclusão da biblioteca "concorde.h".

Faremos um exemplo mínimo funcional:

```
extern "C" { #include <concorde.h> }

//criando funcao para a chamada da funcao presente no Concorde
void solving_tsp_concorde(int[][] distancia, int[] tour){

   //criando uma sequencia qualquer no vetor
   for(int i = 0; i < tour->size(); i++){
      tour->at(i) = i;
   }
   if(dist->size() > 4 ){//TSP somente para mais de 4 elementos
      int rval = 0;
      int semente = rand();
      double szeit, optval, *in_val, *timebound;
      int ncount, success, foundtour, hit_timebound = 0;
      int *in_tour = (int *) NULL;
      int *out_tour = (int *) NULL;
      CCrandstate rstate;
      char *name = (char *) NULL;
      static int silent = 1;
      CCutil_sprand(semente, &rstate);
```

```
      in_val = (double *) NULL;
      timebound = (double *) NULL;
      ncount = dist->size();
      int ecount = (ncount * (ncount - 1)) / 2;
      int *elist = new int[ecount * 2];
      int *elen = new int[ecount];
      int edge = 0;
      int edge_peso = 0;
      for (int i = 0; i < ncount; i++) {
         for (int j = i + 1; j < ncount; j++) {
            if (i != j) {
               elist[edge] = i;
               elist[edge + 1] = j;
               elen[edge_peso] = dist->at(i)[j];
               edge_peso++;
               edge = edge + 2;
            }
         }
      }
      out_tour = CC_SAFE_MALLOC (ncount, int);
      name = CCtsp_problabel(" ");

      CCdatagroup dat;
     CCutil_init_datagroup (&dat);

     rval = CCutil_graph2dat_matrix (ncount, ecount, elist, elen,
         1, &dat);

     rval = CCtsp_solve_dat (ncount, &dat, in_tour, out_tour, NULL,
         &optval, &success, &foundtour, name, timebound,
         &hit_timebound, silent, &rstate);

      for (int i = 0; i < ncount; i++) {
         tour->at(i) = out_tour[i];
      }
      szeit = CCutil_zeit();
      CC_IFFREE (elist, int);
      CC_IFFREE (elen, int);
      CC_IFFREE (out_tour, int);
      CC_IFFREE (probname, char);
   }

}

int main(){

   //inicializacao da matriz e vetor
   solving_tsp_concorde(distancia,tour);

   //verificando resultado dado pelo resolvedor
   for(int i = 0; i < tour->size(); i++){
```

```
      cout<<tour->at(i)<<" ";
   }

   //faca aqui a construcao da solucao do seu problema

   return 0;
}
```

Modele seu problema de acordo com passos deste *HowTo* e resolva seu problema *TSP-like* de forma exata.

Referências

Algumas referências foram utilizadas para criarmos este *HowTo*:

http://rodrigobrito.net/2016/08/14/instalando-concorde-tsp-solver/
http://www.leandro-coelho.com/installing-concorde-tsp-with-cplex-linux/#more-146
http://www.math.uwaterloo.ca/tsp/concorde

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.

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 *Hybrid metaheuristics*, pp. 190–202. Springer.

Amaya, J. E.; Cotta, C. e Fernández-Leiva, A. J. (2012). Solving the tool switching problem with memetic algorithms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(02):221–235.

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.

Applegate, D.; Cook, W. e Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92.

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

Boruvka, O. (1926). O jistem problemu minimaalnim. moravske prirodovedecke spolecnosti 3, 37-58.

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.; Lorena, L.; Senne, E. e Resende, M. (2016). Hybrid method with cs and brkga applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174–183.

Chaves, A. A.; Senne, E. L. F. e Yanasse, H. H. (2012). A new heuristic for the minimization of tool switches problem. *Gestão & Produção*, 19(1):17–30.

Cook, W. (2012). *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press.

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.

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

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

Golden, B. e Stewart, W. (1985). Empirical analysis of heuristics in the travelling salesman problem, e. lawer, j. lenstra, a. rinnooy and d. shmoys.

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.

Kernighan, B. W. e Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

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.

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

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

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.

Senne, E. L. F. e Yanasse, H. H. (2009). Beam search algorithms for minimizing tool switches on a flexible manufacturing system. In *XI WSEAS International Conference on Mathematical and Computational Methods in Science and Engineering, MACMESE*, volume 9, pp. 68–72.

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

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. e Lamosa, M. (2005). 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*, p. 90.

Yanasse, H. e Lamosa, M. (2006). On solving the minimization of tool switches problem using graphs. In *Annals of the XII ICIEOM – International Conference on Industrial Engineering and Operations Management*.

Yanasse, H. H. e Pinto, M. J. (2002). The minimization of tool switches problem as a network flow problem with side constraints. *XXXIV SBPO–Simpósio Brasileiro de Pesquisa Operacional, realizado no Rio de Janeiro, RJ*, 8:86.

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 and Produção*, 13(3).

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.