

Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM

**Um Algoritmo Heurístico Aplicado a Produção em Sistemas
de Manufatura Flexíveis**

Bolsista: Túlio Neme de Azevedo

Orientador: Prof. Dr. Marco Antonio Moreira de Carvalho – DECOM/UFOP

Nota: Relatório final referente ao período de 01/03/2016 a 28/02/2017, apresentado à Universidade Federal de Ouro Preto, como parte das exigências do programa PIVIC-1S.

Local: Ouro Preto - Minas Gerais - Brasil

Data: 20 de novembro de 2017

Um Algoritmo Heurístico Aplicado a Produção em Sistemas de Manufatura Flexíveis

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 representa prejuízo. 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), que é caracterizado por definir a menor rota entre um conjunto de cidades, passando uma vez em cada cidade e retornando à cidade inicial. Desta forma, foi revisitada a modelagem do MTSP via PCV, definindo cada tarefa como uma cidade e a distância entre as tarefas sendo relacionada à estimativas do número de trocas caso as tarefas sejam processadas em sequência. Cinco métricas diferentes, encontradas na literatura, foram consideradas para este fim. Além disso, o Concorde, famoso resolvedor exato para o PCV foi utilizado considerando cada uma das referidas métricas. Experimentos computacionais foram realizados e os resultados comparados com a melhor solução presente na literatura. Os tempos reportados pelos métodos desenvolvidos se mostraram muito competitivos, entretanto, em termos de qualidade das soluções, todos os métodos foram inferiores ao estado da arte.

Assinatura do orientador(a): _____

Marco Antonio Moreira de Carvalho

Assinatura do bolsista: _____

Túlio Neme de Azevedo

Sumário

1	Introdução	1
1.1	Contextualização	1
1.2	Descrição do Problema	2
2	Objetivos	5
3	Fundamentos Teóricos e Revisão	6
4	Materiais e Métodos	10
4.1	Modelagem do Problema	10
4.2	O Resolvedor Concorde	12
4.3	O Problema de Minimização de Trocas de Ferramentas e o Problema do Caixa- xeiro Viajante	12
5	Resultados e Discussões	15
5.1	Instâncias de Yanasse et al. (2009)	15
5.2	Instâncias de Crama et al. (1994)	18
5.3	Instâncias de Catanzaro et al. (2015)	20
5.4	Discussão	22
6	Conclusão	23

Capítulo 1

Introdução

Neste capítulo é apresentado uma introdução ao tema abordado neste trabalho, bem como uma descrição formal do Problema de Minimização de Trocas de Ferramentas, com exemplos.

1.1 Contextualização

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. Desta maneira cada troca exige que a máquina seja desligada, interrompendo a linha de produção, ocasionando prejuízo, aumento de custo de produção, lentidão, etc.

Ao iniciar a fabricação dos produtos, trocas de ferramentas serão necessárias, para que a capacidade não seja ultrapassada e o produto subsequente seja processado. Assim, ao trocar as ferramentas da máquina, a linha de produção é interrompida inteiramente, gerando perda de tempo. Um dos problemas que está presente nas indústrias que empregam máquina flexíveis, é o *Problema de Minimização de Troca de Ferramentas* (MTSP), um problema de otimização combinatória. 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 MTSP 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 and 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. O método *Keep Tool Needed Soonest* (KTNS) (Tang and Denardo, 1988) soluciona este problema em tempo deter-

minístico polinomial. Quando troca de ferramentas são necessárias, a política KTNS garante que as ferramentas que serão necessárias mais brevemente pelas tarefas ainda não realizadas 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 as ferramentas possuem tamanho e custo uniformes, ou seja, não importa qual a posição as ferramentas são alocadas na máquina flexível e o tempo de instalação é o mesmo para todas ferramentas.

1.2 Descrição do Problema

Formalmente, o MTSP é definido da seguinte forma: dados uma máquina flexível com capacidade de comportar até C ferramentas, um conjunto de tarefas $T = [1, \dots, n]$, um conjunto de ferramentas $F = [1, \dots, m]$, o subconjunto de ferramentas F_i ($F_i \in F$) necessárias para realizar 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 execução de cada uma das tarefas. A Tabela 1.1 apresenta os dados de um cenário relacionado ao MTSP em que tem-se $n = 5$, $m = 5$, $C = 3$.

Tabela 1.1: 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 1.1, 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 1.2. As linhas representam as ferramentas e as colunas representam as tarefas, ambas numeradas de 1 a 5. Os elementos q_{ij} da matriz Q são preenchidos da seguinte maneira: $q_{ij} = 1$ caso a ferramenta i ($i \in F$) for necessária para processar a tarefa j ($j \in T$). Caso contrário, $q_{ij} = 0$.

Tabela 1.2: 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

Modelado o problema, uma solução é dada por um sequenciamento ϕ das colunas de Q . Por exemplo, considere $\phi = [3, 4, 2, 1, 5]$, representada pela Tabela 1.3. Esta solução resulta em 7 troca de ferramentas, explicitadas a seguir:

Tabela 1.3: 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 1.4, 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 1.4, 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. Por exemplo, na troca da tarefa 3 para a tarefa 4, a ferramenta 2 foi inserida na máquina, portanto sublinhada, e a ferramenta 5 foi mantida na máquina mesmo sem ser utilizada na tarefa 4, portanto em negrito.

Tabela 1.4: Matriz R^ϕ .		0	3	4	2	1	5
Ferramentas \ ϕ							
1		0	<u>1</u>	1	1	0	0
2		0	0	<u>1</u>	1	1	1
3		0	0	0	<u>1</u>	1	1
4		0	<u>1</u>	0	0	0	<u>1</u>
5		0	<u>1</u>	1	0	<u>1</u>	0

O valor dos elementos r^ϕ são definidos de acordo com a Equação 1.2.

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

Uma solução representada pela matriz R^ϕ é avaliada de acordo com a Equação 1.1, 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, por exemplo, vide os dados contidos na Tabela 1.4, cujo número de trocas é 7.

$$Z_{MTSP}^\phi(R) = \sum_{j \in T} \sum_{i \in F} r_{ij}^\phi (1 - r_{ij-1}^\phi) \quad (1.1)$$

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 1.2.

$$\min_{\phi \in \Phi} Z_{MTSP}^{\phi}(Q) \quad (1.2)$$

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).

Capítulo 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 resolvidor exato Concorde para sua solução.

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 resolvidor Concorde;
4. Propor um método heurístico para solução do Problema de Minimização de Trocas de Ferramentas baseado na solução exata do Problema do Caixeiro Viajante; e
5. Avaliar o método implementado considerando as métricas disponíveis na literatura e problemas teste publicamente disponíveis, realizando uma análise crítica considerando outros métodos da literatura.

Capítulo 3

Fundamentos Teóricos e Revisão

Os conceitos básicos deste trabalho foram fundamentados através dos autores com os respectivos trabalhos referentes ao MTSP mencionados a seguir. A revisão da literatura é apresentada em ordem cronológica.

Tang and Denardo (1988) foram os primeiros a relataram na literatura uma abordagem ao MTSP, utilizando modelagem por grafos. Porém, 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 $C \geq 2$. Posteriormente, nesta mesma modelagem, Hertz et al. (1998) embutiram a política KTNS em heurísticas existentes para solução do PCV: duas heurísticas de Gendreau et al. (1992), GENI, e GENIUS, e uma de Golden and Stewart (1985), *Farthest Insertion*. A heurística GENI consiste em um método construtivo, ou seja, gera 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. O método *Farthest Insertion* tem como característica inserir na solução o elemento cuja distância seja a maior em relação aos elementos presentes na solução corrente. Adicionalmente, 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 função objetivo para o PCV que considera o KTNS como um de seus componentes. Das heurísticas citadas, a que apresentou melhores resultados foi a versão do GENIUS que incorpora o KTNS.

Os primeiros a utilizarem dados reais oriundos das indústrias foram Shirazi and Frizelle (2001), provando que os métodos existentes na literatura eram melhores que os utilizados nas indústrias. Observou-se que as instâncias reais possuíam uma capacidade C 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 em Linhares and Yanasse (2002), como o GMLP, 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 and Al-Sultan (2003) propuseram uma busca tabu, utilizando diferentes estratégias e métodos probabilísticos para reduzir o espaço de busca. Adicionalmente, foram utilizados dois métodos para geração de soluções aleatórias: o *Random Swapping*, que realiza trocas entre pares de tarefas, e o *Random Block Insertion*, que realiza trocas entre pares de blocos de tarefas consecutivas. Comparados com a Busca Tabu original, a heurística mista, ou seja, a que utilizou estratégias probabilísticas e de oscilação, foi a que apresentou melhores resultados.

Os métodos *branch-and-cut* e *branch-and-bound* foram aplicados à solução do MTSP por Laporte et al. (2004). Novamente, a formulação baseou-se no PCV, utilizando o método GENIUS. O *branch-and-bound* foi capaz de solucionar instâncias com até 25 tarefas e 25 ferramentas em tempo computacional aceitável. Adicionalmente, foi proposta uma nova função objetivo também baseada no PCV. 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 and Finke (2000) seguiram a linha de pesquisa que utiliza métodos específicos para resolução do PCV para solucionar o MTSP, e adaptaram o método *Partitioning Algorithm*, algoritmo desenvolvido especificamente para o problema de localização de k -servidores, aplicando-o ao 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, eles propuseram as suas próprias instâncias e não compararam com nenhum outro método da literatura.

Novamente modelando o MTSP como outros problemas combinatórios, Yanasse and Pinto (2002) propuseram uma modelagem, baseada em grafos, 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 parece ser promissora, principalmente para instâncias pequenas, independente da capacidade da máquina. Posteriormente, Yanasse and Lamosa (2005) enunciaram uma proposta de modelagem utilizando o Problema do Caixeiro Viajante Generalizado, implementada por Yanasse and 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 por Tzur and Altman (2004). Neste trabalho, foi proposto o método *Aladdin*, baseado no melhor método existente até então para resolver o MTSP original, o GENIUS, utilizado anteriormente

por Hertz et al. (1998). Comparou com os outros métodos existentes para o MTSP e implementados por ele. Nessas implementações utilizou-se o método *Keep Smaller Tools Needed Soonest* (KSTNS), que é quem determina o número de troca de ferramentas para a extensão do MTSP abordada. Esta mesma extensão do MTSP foi considerada por Crama et al. (2007). Neste trabalho, 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. Contudo, este método apresenta um alto custo exponencial na resolução do problema.

Um algoritmo memético foi utilizado por Amaya et al. (2008), cuja implementação utilizou uma variação de algoritmo genético e a busca local *Hill Climbing*. Além disso, uma implementação de algoritmo memético que utiliza a busca local e o algoritmo genético. Os resultados foram comparados com o *Beam Search* de Zhou et al. (2005) e foram melhores, principalmente os obtidos pelo algoritmo memético.

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, não foi capaz de resolver outras instâncias, como alguns casos com 25 tarefas. Baseados neste *branch-and-bound*, Senne and Yanasse (2009) empregaram o método *beam search*, que seleciona as regiões promissoras, com o objetivo de determinar limitantes superiores para acelerar os métodos já existentes, ou seja, reduzir o espaço de busca para os algoritmos. 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 and Nevalainen (2015), considerando posições fixas para as ferramentas na máquina, isto é, 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 NP-difícil no sentido forte. No caso em que o número de módulos alimentadores e o número de módulos alimentadores forem fixos, pode-se encontrar a solução em tempo determinístico polinomial, porém, o expoente polinomial é muito alto.

Dois modelos de programação linear inteira específicos para solucionar o MTSP foram implementados por Catanzaro et al. (2015). Os resultados reportados indicam que ambos os métodos apresentam melhor relaxação linear do que os métodos que tem a mesma linha de raciocínio anteriores. Uma política alternativa ao KTNS foi recentemente proposta por Adjashvili 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.

Uma nova abordagem realizada sobre a estratégia de busca local iterada (ILS) aliada com o método *2-opt* foi proposta implementada por Paiva and Carvalho (2016). Em seus estudos, foram embutidos nos métodos mencionados uma nova proposta de representação em grafos para o problema, uma nova heurística construtiva para a geração de soluções iniciais e um novo método de busca local baseada no agrupamento de *1-blocks*. 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). O novo método foi capaz de igualar ou superar todos os resultados anteriores para essas instâncias, sendo considerado o atual estado da arte para o problema abordado.

Capítulo 4

Materiais e Métodos

Neste capítulo é abordado, em subseções, a modelagem adotada para o Problema de Minimização de Trocas de Ferramentas e como este se relaciona com 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.

4.1 Modelagem do Problema

Conforme apresentado no Capítulo 3, diferentes autores relacionaram o MTSP com um problema clássico da literatura, o *Problema do Caixeiro Viajante* (PCV), em inglês *Traveling Salesman Problem*.

Em seu livro (Cook, 2012), William J. Cook, uma referência sobre o estudo do PCV, relata que a origem do problema não possui datamento correto, 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 4.1 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.

Computacionalmente, uma instância do PCV pode ser modelada por uma matriz W $n \times n$, exemplificada na Tabela 4.1. 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 ($i, j = 1 \dots n$). Note que $w_{ij} = w_{ji}$ no caso simétrico e quando $i = j$, w_{ij} possui valor nulo.

Dado o exemplo da Tabela 4.1, 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 o ciclo começa e termina na cidade 0, visitando as cidades 2, 1 e 3, nesta ordem, durante o percurso. O

Figura 4.1: Representação de uma instância do PCV.

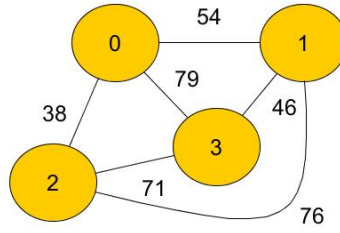


Tabela 4.1: Matriz W .

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

custo total da solução S é 239, calculada pela soma das distâncias entre as cidades, de acordo com a Tabela 4.1, 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 4.1, 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, 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 de 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 resolvidor *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 resolvidor 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.

¹<http://www.math.uwaterloo.ca/tsp/concorde>

²<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>

4.2 O Resolvedor Concorde

O resolvedor *Concorde* é um código desenvolvido para solução do PCV disponível também para vários problemas de otimização relacionados. O código pode ser utilizado 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 and Lin, 1970) e solução aleatória.

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 alguns problemas de otimização, 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 completude de resultados. Com efeito, durante os experimentos computacionais, erros de execução e solução foram verificados. Por exemplo, na documentação do resolvedor estão relacionadas duas funções indicadas como equivalentes entre si para solução dos problemas devidamente modelados (as funções *CCtsp_solve_sparse* e *CCtsp_solve_dat*). Porém, durante os experimentos computacionais, constatou-se o contrário. Uma das funções (*CCtsp_solve_sparse*) não foi capaz de resolver boa parte das instâncias consideradas, ao passo que a outra (*CCtsp_solve_dat*) resolveu todas.

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 (*Minimal Working Example*). Este relatório técnico está incluso no Apêndice A.

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

Conforme mencionado no Capítulo 3, o MTSP pode ser modelado como o PCV, ao relacionar o cálculo do número de trocas de ferramentas do primeiro problema com cálculo da distância entre as cidades do segundo problema, em que a minimização dos valores calculados é abordada na função objetivo de ambos os problemas. Nesta seção, revisitamos as diferentes formas de cálculo de distâncias propostas na literatura ao modelar o MTSP como o PCV.

³<http://www.math.uwaterloo.ca/~bico/qsopt>

⁴<http://www.ibm.com/software/commerce/optimization/cplex-optimizer/>

Hertz et al. (1998) reportaram cinco formas para realizar o cálculo das distâncias d_{ij} ($i, j \in T$) dada uma instância do MTSP. As três primeiras são conhecidas da literatura e outras duas novas são 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 ($i \in T$).

A primeira métrica relatada, 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.

$$d_{ij} = C - |F_i \cap F_j| \quad (4.1)$$

Para a referida Equação 4.1, Hertz et al. (1998) citaram 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 $|T_i| < C$ ($T_i \in T$) e ainda assim tem-se algoritmos para o PCV que resultam em bons resultados heurísticos para o problema.

A Equação 4.2 considera 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).

$$d_{ij} = |F_i \cup F_j| - |F_i \cap F_j| \quad (4.2)$$

Uma observação válida sobre as Equações 4.1 e 4.2 é que quando $|F_i| = C$ as equações se tornam equivalentes.

Uma outra métrica foi proposta anteriormente por Tang and Denardo (1988) para cálculo de limitantes inferiores para o número de trocas de ferramentas, apresentada na Equação 4.3. Esta métrica considera o número de ferramentas utilizadas por duas tarefas que excede a capacidade de máquina.

$$d_{ij} = |F_i \cup F_j| - C \quad (4.3)$$

Entretanto, a métrica apresentada na Equação 4.3 pode produzir valores negativos. A terceira métrica enumerada por Hertz et al. (1998) é uma adaptação desta métrica para evitar valores negativos que foi proposta originalmente por Crama et al. (1994). Vale ressaltar que este é o cálculo de distância utilizada pela maioria dos autores que solucionaram o MTSP utilizando a modelagem baseada no PCV. A Equação 4.4 apresenta esta métrica.

$$d_{ij} = \max\{0, |F_i \cup F_j| - C\} \quad (4.4)$$

As três primeiras métricas (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 métricas.

A métrica expressada pela Equação 4.5, aperfeiçoa a Equação 4.4. É adotado 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. Para isso, computa-se a função $\lambda_k(ij)$, resultando no número de tarefas, excluindo-se as tarefas i e j , que requerem da ferramenta $k \in F_i \cup F_j$, e a função $\Lambda(ij) = \sum_{k \in F_i \cup F_j} \lambda_k(ij)$, que 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 métrica subtrai de $|F_i \cup F_j|$ uma quantidade entre $[0, C]$, sendo esta quantidade alta quando as ferramentas de $F_i \cup F_j$ são utilizadas com frequência.

$$d_{ij} = \max \left\{ 0, |F_i \cup F_j| - \left\lceil \theta \frac{\Lambda(ij)}{(n-2) |F_i \cup F_j|} \right\rceil C \right\} \quad (4.5)$$

Nesta mesma lógica, a Equação 4.6 aperfeiçoa a Equação 4.2. O fator $\frac{c+1}{c}$ é um valor entre $[1, 2]$ que proporciona um peso maior para o termo $|F_i \cup F_j|$ caso a capacidade máxima da máquina for pequena, i. e., se mais trocas de ferramentas são prováveis. A segunda parte da equação é no mínimo 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 $\Lambda(ij) = 0$.

$$d_{ij} = \left(\left\lceil \frac{c+1}{c} \right\rceil |F_i \cup F_j| - |F_i \cap F_j| \right) \left\lceil \frac{(n-2) |F_i \cup F_j|}{\max\{\Lambda(ij), 0.5\}} \right\rceil \quad (4.6)$$

Em testes realizados nas heurísticas apresentadas em seu trabalho, Hertz et al. (1998) reportaram que o cálculo da distância realizado pela Equação 4.6 foi a que apresentou melhores resultados.

Capítulo 5

Resultados e Discussões

As cinco diferentes métricas para cálculo da distância na modelagem do MTSP como o problema do Caixeiro Viajante mencionadas no Capítulo 4 foram utilizadas para derivar cinco métodos diferentes para a solução do MTSP. São elas: d_1 (descrita pela Equação 4.1), d_2 (descrita pela Equação 4.2), d_3 (descrita pela Equação 4.4), d_4 (descrita pela Equação 4.5) e d_5 (descrita pela Equação 4.6).

Experimentos preliminares foram conduzidos para ajustar o valor do parâmetro θ , de maneira a gerar os melhores resultados quando utilizada a distância d_4 . Entretanto, conclui-se que a variação do valor de θ no intervalo $[0, 1]$ não gerou alteração significativa nas soluções obtidas para as instâncias consideradas.

Todos os métodos foram implementados utilizando a linguagem C++, além de incorporar as funções do resolvidor Concorde e do resolvidor de programação linear CPLEX, versão 12.6.3.

Os experimentos foram realizados em um computador com processador *Intel i3-2350M Quad Core* de 2.30GHz, 4 GB de RAM e sistema operacional Ubuntu 16.04.02. Os resultados obtidos foram comparados com os obtidos pelo atual estado da arte relacionado ao MTSP, a metaheurística *Iterated Local Search* (ILS) proposta por Paiva and Carvalho (2016). Os resultados desse método foram obtidos em um computador com processador *Intel i5 Quad Core* de 3.2GHz e 8 GB de RAM.

Nas tabelas a seguir são apresentados o número de ferramentas (m), a capacidade da máquina (C), a melhor solução encontrada pelo ILS (S^*), a solução encontrada pelos métodos propostos neste trabalho (S) e o tempo médio de execução de cada método em segundos (T). As siglas d_1 , d_2 , d_3 , d_4 e d_5 indicam qual algoritmo está evidenciado na coluna e as demais informações sobre os grupos de instâncias são definidas no enunciado de cada tabela.

Para análise das soluções foram levados em consideração o *gap* (distância percentual em relação à melhor solução existente), calculado como $100 \times (S - S^*) / S^*$ e o *ranking* ordinal, que estabelece uma ordem entre os métodos comparados, indicando quais deles geram as melhores soluções com mais frequência, embora não indique a qualidade das soluções geradas. Em outras palavras, o *ranking* ordinal atribui valores ordinais aos métodos de acordo com a qualidade das soluções associadas (em caso de empate, o mesmo valor ordinal é atribuído aos métodos).

Após a apresentação dos resultados gerados para cada conjunto de instâncias e respectivas análises individuais, a Seção 5.4 apresenta uma discussão geral sobre o desempenho dos métodos.

5.1 Instâncias de Yanasse et al. (2009)

Yanasse et al. (2009) propôs um total de 1.350 instâncias para o MTSP, divididas em 5 grupos

(A, B, C, D, e E). Entre os grupos, varia-se a capacidade da máquina, o número de tarefas, o número de ferramentas e o número de instâncias. A Tabela 5.1 apresenta os resultados obtidos para grupo A, em que o número de tarefas é fixado em 8 para todas as instâncias.

Tabela 5.1: Resultados obtidos para o Grupo A.

<i>m</i>	<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
		<i>S</i> *	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>
15	5	17,00	0,06	17,80	0,00	17,70	0,00	17,90	0,01	18,30	0,00	19,30	0,01
15	10	16,83	0,08	17,83	0,00	17,57	0,00	17,63	0,00	17,50	0,00	17,87	0,00
20	5	21,80	0,06	22,60	0,01	22,50	0,00	22,50	0,00	22,40	0,00	25,70	0,00
20	10	23,07	0,15	24,50	0,00	23,93	0,00	24,00	0,01	23,63	0,00	24,87	0,01
20	15	22,08	0,12	23,03	0,00	22,83	0,00	23,28	0,01	22,60	0,00	22,93	0,00
25	5	25,10	0,03	25,40	0,00	25,20	0,00	25,50	0,00	25,40	0,00	27,80	0,00
25	10	28,20	0,16	29,57	0,00	29,20	0,00	29,20	0,00	29,17	0,00	30,73	0,00
25	15	27,95	0,21	29,53	0,00	28,93	0,00	29,52	0,01	28,83	0,00	29,82	0,00
25	20	26,61	0,15	27,69	0,00	27,58	0,00	27,78	0,01	27,24	0,00	27,78	0,00
<i>gap</i>		0,00%		4,51%		3,33%		4,21%		3,26%		8,89%	
<i>ranking</i>		1,00		1,85		1,63		1,78		1,61		2,32	

Nenhum dos métodos propostos conseguiu igualar os melhores valores já reportados. O *gap* e o *ranking* dos métodos indicam que as melhores métricas de distância foram *d*₄ e *d*₂. O tempo de execução apresentado pelos métodos mostrou agilidade na resolução das instâncias, com média de 0,00 segundos para todos os métodos. Ressalta-se também que *d*₄ alcançou, dentre os métodos comparados, o menor *gap* em 66,67% dos subgrupos, ou seja, em 6 dos 9 subgrupos.

A Tabela 5.2 apresenta os resultados obtidos para as instâncias do grupo B. Neste grupo, as instâncias do grupo possuem o número de tarefas fixo em 15 unidades.

Tabela 5.2: Resultados obtidos para o Grupo B.

<i>m</i>	<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
		<i>S</i> *	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>
15	5	17,20	0,08	18,30	0,01	18,20	0,01	18,20	0,01	18,20	0,01	20,10	0,01
15	10	17,37	0,12	18,53	0,01	18,23	0,00	18,50	0,01	17,93	0,01	18,83	0,01
20	5	22,40	0,09	23,60	0,01	23,00	0,01	23,30	0,01	23,30	0,01	26,30	0,01
20	10	24,17	0,22	25,63	0,01	25,50	0,00	25,83	0,01	25,10	0,01	26,63	0,01
20	15	22,60	0,20	23,87	0,01	23,52	0,01	24,10	0,01	23,45	0,01	23,95	0,01
25	5	25,40	0,05	26,20	0,01	26,10	0,01	26,30	0,01	26,40	0,01	29,60	0,01
25	10	28,77	0,24	30,57	0,01	29,93	0,01	30,17	0,01	29,87	0,01	32,13	0,01
25	15	29,74	0,39	31,68	0,01	31,04	0,01	31,48	0,01	30,82	0,01	31,94	0,01
25	20	27,19	0,24	28,31	0,01	28,10	0,01	28,47	0,02	28,37	0,01	27,19	0,01
<i>gap</i>		0,00%		5,58%		4,18%		5,43%		3,83%		10,99%	
<i>ranking</i>		1,00		2,05		1,79		2,00		1,74		2,61	

Neste grupo, novamente, nenhum dos métodos propostos conseguiu igualar aos valores ótimos reportados pelo ILS. Os respectivos *gap* e *ranking* indicam maior qualidade nos resultados obtidos pelo método *d*₄, quando comparado entre os métodos propostos. O tempo de execução médio de 0,00 segundos, para todos os métodos propostos, indica mais uma vez a agilidade em solucionar o problema. Analogamente ao ocorrido no grupo A, o método *d*₄ apresentou menores valores de *gap* em 6 dos 9 subgrupos.

Tabela 5.3: Resultados obtidos para o Grupo C.

<i>m</i>	<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
		<i>S</i> *	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>
15	5	21,60	0,60	25,50	0,02	24,30	0,01	25,80	0,02	24,70	0,02	25,50	0,01
15	10	19,80	1,02	22,37	0,01	22,20	0,01	23,60	0,04	21,87	0,02	22,13	0,01
20	5	25,60	0,75	28,40	0,01	27,90	0,01	28,40	0,02	28,40	0,02	30,30	0,01
20	10	28,33	1,82	32,17	0,02	31,03	0,01	33,80	0,03	30,97	0,02	32,50	0,01
20	15	25,52	2,00	28,17	0,01	27,90	0,01	29,78	0,04	27,72	0,02	28,12	0,01
25	5	32,50	0,71	35,10	0,02	35,00	0,01	35,00	0,02	34,60	0,02	40,30	0,02
25	10	35,07	2,19	38,33	0,01	37,97	0,01	40,47	0,03	37,67	0,02	40,57	0,02
25	15	34,07	3,49	37,83	0,01	37,22	0,01	39,90	0,04	37,13	0,02	38,68	0,01
25	20	29,66	2,41	32,16	0,01	31,94	0,01	33,98	0,05	31,74	0,02	29,66	0,01
<i>gap</i>		0,00%		14,02%		12,15%		17,72%		12,00%		17,62%	
<i>ranking</i>		1,00		2,85		2,46		3,33		2,42		3,31	

Os resultados obtidos para o grupo C são apresentados na Tabela 5.3. O número de tarefas em cada instância do grupo é fixado em 15.

A qualidade das soluções apresentadas, medidas pelo *gap* e pelo *ranking*, indicam novamente o melhor desempenho do método *d*₄, seguido pelo método *d*₂. O primeiro método, novamente, apresentou os menores valores de *gap* em 77,78% dos subgrupos. Em relação ao tempo de execução, a característica observada nos conjuntos anteriores é mantida, com médias em torno dos 0,03 segundos em todos os métodos.

O grupo D apresenta mais variações entre número de tarefas, número de ferramentas e capacidade da máquina. Os respectivos resultados para este grupo são sumarizados na Tabela 5.4. Nesta tabela, uma linha separa as instâncias que contém 20 e 25 tarefas.

Tabela 5.4: Resultados obtidos para o Grupo D.

<i>m</i>	<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
		<i>S</i> *	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>	<i>S</i>	<i>T</i>
15	5	25,90	1,61	30,80	0,03	30,00	0,02	32,60	0,04	29,80	0,03	31,60	0,02
15	10	18,20	2,70	21,30	0,03	21,10	0,01	25,85	0,08	21,20	0,02	21,70	0,02
20	5	29,20	2,14	34,00	0,02	33,10	0,02	35,80	0,03	33,60	0,02	36,40	0,03
20	10	20,60	3,22	23,30	0,02	23,80	0,02	27,90	0,09	23,60	0,03	24,30	0,02
20	15	21,67	4,16	24,00	0,03	24,17	0,01	27,07	0,09	24,00	0,03	23,97	0,02
25	5	35,10	2,35	38,70	0,02	38,60	0,02	40,40	0,03	38,20	0,03	44,30	0,03
25	10	25,40	4,10	27,90	0,03	28,30	0,02	34,50	0,10	28,20	0,03	30,80	0,03
25	15	36,25	9,35	40,18	0,02	39,70	0,01	44,53	0,08	40,18	0,06	40,40	0,02
25	20	26,15	4,17	28,70	0,03	28,40	0,02	31,13	0,10	28,43	0,03	28,75	0,02
15	10	15,90	4,56	18,80	0,03	18,70	0,02	24,80	0,14	18,70	0,04	18,80	0,04
20	10	21,60	11,38	26,70	0,03	26,50	0,03	35,00	0,15	26,10	0,04	27,00	0,04
20	15	22,60	12,50	26,70	0,03	27,50	0,03	31,10	0,16	26,30	0,05	26,70	0,04
25	10	26,60	13,46	32,00	0,04	31,20	0,04	40,10	0,14	31,20	0,04	32,80	0,06
25	15	25,00	7,41	26,80	0,04	27,40	0,03	32,20	0,14	27,50	0,05	27,30	0,06
25	20	25,50	7,67	27,60	0,03	27,53	0,02	30,67	0,16	27,87	0,04	27,83	0,03
<i>gap</i>		0,00%		14,18%		13,90%		33,27%		13,51%		17,73%	
<i>ranking</i>		1,00		2,65		2,57		4,22		2,52		3,13	

O tempo de execução se mantém baixos, sendo altamente competitivos com a média encontrada pelo ILS: 6,05 segundos. Neste grupo de instâncias, os métodos propostos tiveram

a sua pior performance. O menor *gap* médio foi de 13,51%. Particularmente, os métodos d_3 e d_5 obtiveram seus piores valores para o *gap* neste conjunto, com média de 33,27% e 17,63%, respectivamente.

Comparando a qualidade das soluções entre os métodos propostos, d_4 e d_2 possuem os melhores resultados para esse grupo de instâncias. Além disso os métodos d_2 e d_4 apresentaram os menores valores de *gap* em 40,00% dos subgrupos analisados.

Os resultados obtidos para o grupo E são apresentados na Tabela 5.5. Analogamente a introdução da Tabela 5.4, as instâncias do grupo estão divididas por uma linha, que indica as que possuem 10 e 15 tarefas, respectivamente.

Tabela 5.5: Resultados obtidos para o Grupo E.

m	C	ILS		d_1		d_2		d_3		d_4		d_5	
		S^*	T	S	T	S	T	S	T	S	T	S	T
10	4	13,50	0,10	15,50	0,01	15,30	0,01	15,20	0,01	15,20	0,01	15,80	0,01
10	5	11,20	0,10	12,70	0,01	12,00	0,01	12,70	0,01	11,90	0,01	12,40	0,01
10	6	10,30	0,06	10,80	0,01	11,00	0,01	12,50	0,02	10,90	0,01	11,60	0,03
10	7	10,00	0,03	10,30	0,01	10,20	0,01	11,40	0,02	10,30	0,01	10,60	0,01
20	6	27,40	1,01	30,80	0,01	31,30	0,01	32,40	0,02	30,60	0,02	34,20	0,01
20	8	22,20	1,19	24,80	0,02	24,30	0,01	29,20	0,05	24,40	0,08	27,20	0,01
20	10	20,30	0,82	21,70	0,01	21,80	0,01	25,10	0,06	21,60	0,02	23,30	0,01
20	12	20,20	0,73	21,30	0,01	21,50	0,01	24,50	0,06	21,30	0,02	22,00	0,01
<i>gap</i>		0,00%		9,00%		8,29%		19,37%		7,58%		14,61%	
<i>ranking</i>		1,00		2,10		2,01		2,99		1,95		2,73	

Neste conjunto de instâncias, novamente verifica-se o baixo tempo de execução para todos os métodos propostos e a dificuldade dos métodos em obter soluções próximas das melhores conhecidas. Comparando os métodos propostos, novamente d_4 obteve melhor desempenho e os métodos d_3 e d_5 obtiveram desempenhos muito aquém dos demais métodos.

Em suma, os resultados obtidos para as instâncias propostas por Yanasse et al. (2009) demonstraram que, mesmo com bons tempos de execução, nenhum dos métodos foi capaz de atingir algum valor reportados pelo ILS, que são os melhores reportados na literatura. O método d_4 obteve maior destaque entre os métodos propostos, apresentando menores valores de *gap* em 60,00% no total dos subgrupos de instâncias, ou seja, 30 em 50 subgrupos. Destaca-se também o mau desempenho dos métodos d_3 e d_5 . Curiosamente, o método d_3 , conforme mencionado na Seção 3 é o mais utilizado pelos trabalhos da literatura.

5.2 Instâncias de Crama et al. (1994)

Divididas em 4 grupos (C_1 , C_2 , C_3 e C_4), as 160 instâncias propostas por Crama et al. (1994) são diferenciadas pelo número de tarefas, número de ferramentas e capacidade máxima da máquina flexível. Cada grupo possui 40 instâncias.

A Tabela 5.6 apresenta as soluções encontradas para as instâncias do grupo C_1 . Ressalta-se que o número de tarefas e o número de ferramentas é fixado para todas as instâncias do grupo em 10. Sendo assim, varia-se somente a capacidade da máquina.

Apesar de as instâncias neste conjunto possuírem o número de tarefas similar ao das instâncias do grupo A, apresentado anteriormente, o desempenho dos métodos propostos é pior, conforme indicado pelos valores de *gap*. Embora os melhores resultados não tenham sido alcançados, a métrica d_2 apresentou os melhores indicadores, tanto para o *gap* quanto para o *ranking*.

Tabela 5.6: Resultados obtidos para o Grupo C_1 .

C	ILS		d_1		d_2		d_3		d_4		d_5	
	S^*	T	S	T	S	T	S	T	S	T	S	T
4	13,10	0,10	14,70	0,01	14,60	0,01	15,20	0,01	14,30	0,01	15,20	0,01
5	11,20	0,10	12,40	0,01	11,90	0,01	13,20	0,01	12,40	0,01	13,00	0,01
6	10,30	0,06	11,10	0,01	10,90	0,01	12,30	0,02	10,90	0,01	11,40	0,01
7	10,10	0,04	10,40	0,01	10,40	0,01	11,10	0,02	10,50	0,01	10,50	0,01
<i>gap</i>	0,00%		8,42%		6,63%		15,80%		7,42%		11,69%	
<i>ranking</i>	1,00		1,90		1,70		2,55		1,80		2,23	

Em 75,00% dos subgrupos este método dominou os demais. Todos os métodos apresentaram baixo tempo de execução, sempre abaixo de 0,02 segundos.

A Tabela 5.7 sumariza os resultados obtidos para as instâncias do grupo C_2 . Novamente, as instâncias variam apenas a capacidade da máquina, uma vez que o número de tarefas e o número de ferramentas são fixadas em 15 e 20, respectivamente.

Tabela 5.7: Resultados obtidos para o Grupo C_2 .

C	ILS		d_1		d_2		d_3		d_4		d_5	
	S^*	T	S	T	S	T	S	T	S	T	S	T
6	26,60	1,03	29,40	0,01	29,30	0,01	32,70	0,02	29,90	0,02	33,00	0,01
8	21,70	1,45	24,50	0,01	25,30	0,01	28,00	0,05	25,30	0,02	26,80	0,01
10	20,10	1,04	21,80	0,01	22,60	0,01	26,00	0,06	22,30	0,02	23,40	0,02
12	19,60	0,61	20,50	0,01	20,70	0,01	24,30	0,06	20,70	0,02	20,70	0,02
<i>gap</i>	0,00%		9,17%		10,22%		25,06%		10,97%		15,40%	
<i>ranking</i>	1,00		2,15		2,35		3,65		2,35		2,93	

Pela primeira vez nos experimentos realizados, o método d_1 obteve os melhores resultados médios, embora todos os métodos tenham apresentado resultados distantes dos melhores conhecidos na literatura. Os métodos d_2 e d_4 obtiveram resultados comparáveis, o que é indicado pelo valor igual para o *ranking* ordinal. O tempo de execução se manteve baixo para todos os métodos, reforçando a eficiência do resolutor Concorde para estas instâncias.

O grupo C_3 fixa o número de tarefas e o número de ferramentas em 30 e 40 respectivamente, variando a capacidade da máquina, o que é indicado pela coluna C . A Tabela 5.8 apresenta todos os resultados obtidos para esse grupo.

Tabela 5.8: Resultados obtidos para o Grupo C_3 .

C	ILS		d_1		d_2		d_3		d_4		d_5	
	S^*	T	S	T	S	T	S	T	S	T	S	T
15	106,40	104,03	121,90	0,06	119,50	0,04	128,00	0,15	118,50	0,12	124,40	0,08
17	88,50	160,88	103,60	0,07	102,80	0,04	115,30	0,08	100,50	0,09	106,60	0,08
20	70,50	228,83	86,10	0,05	83,90	0,05	102,30	0,20	84,30	0,10	87,10	0,08
25	52,90	206,65	64,20	0,07	65,90	0,04	82,10	0,25	64,10	0,10	65,40	0,08
<i>gap</i>	0,00%		18,78%		18,01%		37,72%		16,42%		21,14%	
<i>ranking</i>	1,00		3,40		3,03		5,33		2,58		3,98	

Neste grupo, os métodos propostos apresentam as piores performances para instâncias deste conjunto. Os valores do *ranking* ordinal indicam um alto revezamento dos métodos na obtenção de boas soluções e o alto valor de *gap* indicam a baixa qualidade das soluções geradas. De fato,

nenhuma das métricas foi capaz de gerar soluções próximas das melhores conhecidas, o que indica que a modelagem do MTSP via PCV não produz heurísticas de boa performance para as instâncias *benchmark* atuais. Seguindo a mesma tendência dos conjuntos anteriores, os métodos d_2 e d_4 apresentaram os melhores indicadores, ao passo que os métodos d_3 e d_5 apresentaram os piores.

Assim como no grupo C_3 , as instâncias do grupo C_4 apresentam um nível de dificuldade de resolução alto. A Tabela 5.9 apresenta os resultados obtidos para estas instâncias, que possuem número de tarefas e número de ferramentas fixados em 40 e 60 respectivamente, ao passo em que a capacidade da máquina varia.

Tabela 5.9: Resultados obtidos para o Grupo C_4 .

C	ILS		d_1		d_2		d_3		d_4		d_5	
	S^*	T	S	T	S	T	S	T	S	T	S	T
20	198,70	533,78	222,60	0,13	219,30	0,08	231,80	0,15	221,10	0,27	230,90	0,14
22	174,00	801,76	198,90	0,12	196,20	0,09	222,50	0,12	198,40	0,20	207,00	0,14
25	146,50	1144,95	172,10	0,08	172,40	0,07	206,20	0,26	171,00	0,18	177,50	0,14
30	114,00	1908,61	138,50	0,09	136,20	0,07	172,40	0,35	138,00	0,20	143,60	0,14
<i>gap</i>	0,00%		16,83%		15,57%		34,65%		16,27%		21,10%	
<i>ranking</i>	1,00		3,25		2,63		5,53		3,08		4,25	

Embora estas instâncias possuam um número maior de tarefas e ferramentas, os resultados obtidos foram levemente melhores do que os obtidos para o conjunto anterior. Entretanto, os indicadores se mantêm altos, mantendo a tendência para os grupos de instâncias deste conjunto. A análise comparativa dos métodos propostos também mantém a mesma tendência anterior.

De maneira geral, mesmo com os tempos de execução baixos, sempre em torno de décimos de segundos, os métodos apresentaram baixa qualidade das soluções, mesmo para instâncias de dimensões semelhantes às do primeiro conjunto de instâncias consideradas. A qualidade das soluções se deteriorou de maneira mais acentuada na resolução de instâncias dos grupos C_3 e C_4 .

Comparando-se os métodos propostos, o método d_2 se destacou dos demais apresentando menores soluções em 75,00% dos subgrupos de instâncias, seguido pelo método d_4 . Novamente, a pior performance foi relacionada ao método d_3 , embora o método d_5 também tenha apresentado soluções de baixa qualidade em geral.

5.3 Instâncias de Catanzaro et al. (2015)

As 160 instâncias propostas por Catanzaro et al. (2015) foram divididas em 4 grupos (*datA*, *datB*, *datC* e *datD*) com 40 instâncias cada. Os grupos de instâncias se diferenciam pelo número de tarefas, número de ferramentas e capacidade da máquina.

O grupo *datA* se divide em 4 subgrupos de instâncias. Todas instâncias presentes nesses quatro subgrupos possuem 10 tarefas e ferramentas. A coluna C indica a capacidade da máquina flexível em cada caso. Os resultados obtidos no grupo *datA* são mostrados na Tabela 5.10.

Todos os métodos apresentam um melhor comportamento neste primeiro grupo de instâncias, com leve superioridade do método d_1 sobre os métodos d_2 e d_4 . Os métodos d_3 e d_5 se distanciam dos demais, principalmente ao considerarmos os valores de *gap*. Levando em consideração o tempo de execução, todos os métodos propostos apresentam médias em torno de 0,02 segundos.

Tabela 5.10: Resultados obtidos para o Grupo *datA*.

<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>
4	12,50	0,09	13,70	0,02	13,60	0,01	14,00	0,01	13,60	0,01	15,10	0,01
5	10,80	0,09	11,70	0,01	11,90	0,01	13,40	0,02	11,80	0,01	12,90	0,01
6	10,10	0,05	10,80	0,01	10,90	0,01	12,60	0,03	11,00	0,01	11,40	0,01
7	10,00	0,04	10,20	0,01	10,20	0,01	11,50	0,03	10,20	0,01	10,50	0,01
<i>gap</i>	0,00%		6,72%		7,23%		18,96%		7,24%		14,53%	
<i>ranking</i>	1,00		1,70		1,78		2,73		1,78		2,40	

A Tabela 5.11 sumariza os resultados para as instâncias do grupo *datB*. Este grupo de instâncias é dividido em 4 subgrupos, os quais possuem fixados o número de tarefas e o de ferramentas em 15 e 20 unidades, respectivamente.

Tabela 5.11: Resultados obtidos para o Grupo *datB*.

<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>
6	26,50	1,02	29,90	0,02	28,70	0,01	31,70	0,02	30,10	0,02	31,70	0,01
8	21,70	1,38	24,40	0,01	23,50	0,01	28,90	0,05	25,10	0,02	26,20	0,01
10	19,70	1,07	22,30	0,01	21,60	0,01	26,80	0,06	22,00	0,02	22,60	0,02
12	19,20	0,55	20,30	0,01	20,40	0,01	23,10	0,06	20,20	0,02	20,20	0,02
<i>gap</i>	0,00%		10,62%		7,71%		26,83%		11,11%		14,62%	
<i>ranking</i>	1,00		2,35		1,93		3,95		2,53		2,98	

À exceção dos métodos *d*₂ e *d*₅, os demais métodos tiveram uma queda significativa na qualidade das soluções geradas. Apesar de o método *d*₂ se destacar frente aos demais, a mesma análise anterior se aplica a este grupo de instâncias, inclusive com melhor desempenho de *d*₁ em relação a *d*₄.

Os resultados apresentados na Tabela 5.12 são as soluções encontradas para as instâncias do grupo *datC*. Novamente as instâncias do grupo foram divididas em 4 subgrupos, contendo número de tarefas e de ferramentas fixadas em 30 e 40 unidades, respectivamente.

Tabela 5.12: Resultados obtidos para o Grupo *datC*.

<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>	<i>S</i> *	<i>T</i>
15	98,80	111,27	114,60	0,05	112,20	0,04	120,20	0,06	112,60	0,09	118,90	0,07
17	82,60	164,64	97,90	0,09	98,20	0,04	113,60	0,09	96,30	0,12	101,40	0,09
20	66,70	229,75	82,30	0,06	82,30	0,04	100,60	0,23	82,00	0,08	84,20	0,07
25	51,30	191,48	63,10	0,05	63,10	0,04	76,20	0,25	62,30	0,10	63,90	0,06
<i>gap</i>	0,00%		20,23%		19,71%		39,64%		18,74%		23,48%	
<i>ranking</i>	1,00		3,28		3,00		5,40		2,75		3,95	

Neste que é considerado um dos conjuntos de instâncias mais difíceis da literatura, os métodos propostos obtiveram as piores performances em todos os experimentos. Todos os indicadores apontam a baixa qualidade das soluções e a inconsistência dos métodos, característica da comparação de algoritmos de desempenho ruim. Destaca-se o *gap* médio de 36,94% do método *d*₃.

Para o último grupo de instâncias proposto por Catanzaro et al. (2015), as soluções são apresentadas na Tabela 5.13. Mais uma vez, as instâncias são divididas em 4 subgrupos. O

número de tarefas e o número de ferramentas são idênticos para todas as instâncias do grupo, respectivamente 40 e 60.

Tabela 5.13: Resultados obtidos para o Grupo *datD*.

<i>C</i>	<i>ILS</i>		<i>d</i> ₁		<i>d</i> ₂		<i>d</i> ₃		<i>d</i> ₄		<i>d</i> ₅	
	<i>S</i> [*]	<i>T</i>	<i>S</i> [*]	<i>T</i>	<i>S</i> [*]	<i>T</i>	<i>S</i> [*]	<i>T</i>	<i>S</i> [*]	<i>T</i>	<i>S</i> [*]	<i>T</i>
20	198,00	489,23	223,40	0,08	219,20	0,09	230,20	0,15	220,50	0,17	230,90	0,16
22	173,80	721,80	199,50	0,08	196,90	0,12	217,90	0,17	198,00	0,20	206,30	0,19
25	146,50	1144,53	170,90	0,08	170,40	0,11	206,60	0,28	173,00	0,22	178,10	0,15
30	115,00	1853,48	139,30	0,08	139,20	0,13	175,70	0,35	138,80	0,23	144,00	0,16
<i>gap</i>	0,00%		16,40%		15,38%		33,90%		16,06%		20,57%	
<i>ranking</i>	1,00		3,15		2,80		5,35		3,00		4,45	

O comportamento dos métodos propostos, apesar de levemente melhor do que o apresentado em relação ao grupo de instâncias anterior, segue o mesmo padrão de valores altos para *gap* e *ranking* ordinal, e também de posição relativa entre os métodos, em relação à qualidade das soluções geradas.

Neste conjunto de instâncias, de maneira geral, assim como nos demais conjuntos considerados, apesar do baixo tempo de execução, os resultados não são razoavelmente próximos daqueles reportados na literatura recente. Particularmente nos subgrupos *datC* e *datD*, considerados como os mais difíceis atualmente, a performance dos métodos propostos foi muito inferior ao atual estado da arte. Comparando-se os métodos entre si, novamente o método *d*₂ obteve os melhores indicadores, seguido pelo método *d*₄, ao passo que o método *d*₃ obteve os piores indicadores, muito aquém dos demais métodos comparados.

5.4 Discussão

De acordo os experimentos e com as respectivas análises específicas nas seções anteriores, é possível realizar uma análise geral sobre a modelagem do Problema de Minimização de Trocas de Ferramentas como o Problema do Caixeiro Viajante. Devido à utilização do resolvidor exato Concorde, foi possível verificar a consistência e qualidade da modelagem em si, dado que os resultados são ótimos para a modelagem utilizada. Isto não foi possível anteriormente na literatura devido à utilização de métodos heurísticos para solução dos modelos, o que por si só poderia gerar degradação da qualidade da solução.

Os dados indicam que as soluções obtidas utilizando a modelagem proposta e as métricas encontradas na literatura possuem baixa qualidade, já que nenhum dos métodos considerados conseguiu igualar ou se aproximar razoavelmente dos resultados obtidos pelo atual estado da arte, mesmo para instâncias mais fáceis. Muito embora os métodos tenham sido capazes de gerar soluções em tempo de execução desprezível, esta característica não é suficiente para que conferir relevância aos mesmos.

Desta forma, conclui-se que a modelagem do MTSP utilizando PCV foi levada ao limite neste trabalho e que a mesma não é adequada, considerando-se o que já foi proposto neste tema. Todavia, é possível concentrar esforços em novas formas de cálculo de distância que sejam capazes de suprir as deficiências das cálculos propostos anteriormente na literatura.

Capítulo 6

Conclusão

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 tarefa 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.

Neste trabalho, propôs-se revisitar esta tradicional modelagem e também as cinco métricas tradicionais para conversão do número de trocas de ferramentas em distâncias entre cidades. Entretanto, utilizou-se o resolvidor exato Concorde, para solução do PCV. Desta forma, foi possível avaliar a qualidade da modelagem empregada, dado que a solução pelo Concorde para o PCV é ótima, para cada métrica empregada. Os experimentos computacionais consideraram três grandes conjuntos de dados, em um total de 1.670 instâncias, divididas em treze subgrupos, e os resultados foram comparados ao método *Iterated Local Search* (ILS), considerado o estado da arte para o problema em questão.

Os resultados computacionais reportaram tempo de execução baixo para os métodos implementados, com médias em torno dos décimos de segundos para todos os métodos em todos os grupos de instâncias, sendo sempre menores que as médias reportada pelo ILS. Entretanto, nenhum método proposto foi capaz de igualar ou se aproximar razoavelmente dos melhores resultados reportados na literatura. As análises dos resultados indicam que duas métricas apresentaram melhores resultados e se destacaram das demais e também, que a métrica mais comumente utilizada na literatura obteve os piores indicadores em relação aos demais métodos comparados. De maneira geral, foi possível observar que a modelagem do MTSP pelo PCV não é adequada para as instâncias atuais, quando consideramos as métricas existentes para adaptação do MTSP ao PCV.

Os trabalhos futuros se concentrarão em produzir métricas de cálculo de distância que apresentem resultados com melhor precisão e melhor desempenho. Dentre as opções destaca-se a utilização de métodos evolutivos que, utilizadas em conjunto com o resolvidor Concorde, darão origem a heurísticas matemáticas para resolução do MTSP.

ANEXO A

Guia de instalação e uso do resolvidor Concorde

Guia de instalação e uso do resolvidor 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 resolvidor para o clássico Problema do Caixeiro Viajante, do inglês *Traveling Salesman Problem* (TSP), e alguns problemas de otimização relacionados. Desenvolvido em ANSI C, o resolvidor 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 resolvidores 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 resolvidores de programação linear QSOpt ou CPLEX.

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

1. QSOpt: 64 bits³ - 32 bits⁴
2. CPLEX⁵ (é necessário fazer um cadastro no site da IBM e você escolhe a arquitetura)

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>

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

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 oficial⁶, (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 coo31219.tgz
```

```
$ tar xvf coo31219.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 Concorde⁷, (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 coo31219.tgz
$ tar xvf coo31219.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 concluído 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 descrições⁸ 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

- David Adjashvili, Sandro Bosio, and Kevin Zemmer. Minimizing the number of switch instances on a flexible machine in polynomial time. *Operations Research Letters*, 43(3):317–322, 2015.
- M A Al-Fawzan and K S Al-Sultan. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & industrial engineering*, 44(1):35–47, 2003.
- Jhon Edgar Amaya, Carlos Cotta, and Antonio J Fernández. A memetic algorithm for the tool switching problem. In *Hybrid metaheuristics*, pages 190–202. Springer, 2008.
- Jhon Edgar Amaya, Carlos Cotta, and Antonio J Fernández-Leiva. Solving the tool switching problem with memetic algorithms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(02):221–235, 2012.
- Jhon Edgar Amaya, Carlos Cotta, and Antonio J Fernández-Leiva. Cross entropy-based memetic algorithms: An application study over the tool switching problem. *International Journal of Computational Intelligence Systems*, 6(3):559–584, 2013.
- David Applegate, William Cook, and André Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.
- Jonathan F Bard. A heuristic for minimizing the number of tool switches on a flexible machine. *IIE transactions*, 20(4):382–391, 1988.
- O Boruvka. O jistem problemu minimaalnim. moravske prirodovedecke spolecnosti 3, 37-58, 1926.
- Daniele Catanzaro, Luis Gouveia, and Martine Labbé. Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, 244(3):766–777, 2015.
- AA Chaves, LAN Lorena, ELF Senne, and MGC Resende. Hybrid method with cs and brkga applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174–183, 2016.
- Antônio Augusto Chaves, Edson Luiz França Senne, and Horacio Hideki Yanasse. A new heuristic for the minimization of tool switches problem. *Gestão & Produção*, 19(1):17–30, 2012.
- W. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2012. ISBN 9780691152707. URL https://books.google.com.br/books?id=S3bxbr_-qhYC.

- Yves Crama, Antoon W. J. Kolen, Alwin G. Oerlemans, and Frits C. R. Spieksma. Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1):33–54, 1994. ISSN 1572-9370. doi: 10.1007/BF01324874. URL <http://dx.doi.org/10.1007/BF01324874>.
- Yves Crama, Linda S Moonen, Frits CR Spieksma, and Ellen Talloen. The tool switching problem revisited. *European Journal of Operational Research*, 182(2):952–957, 2007.
- Housni Djellab, Khaled Djellab, and Michel Gourgand. A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64(1):165–176, 2000.
- Michel Gendreau, Alain Hertz, and Gilbert Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.
- B Golden and W Stewart. Empirical analysis of heuristics in the travelling salesman problem, e. lawer, j. lenstra, a. rinnooy and d. shmoys, 1985.
- Alain Hertz, Gilbert Laporte, Michel Mittaz, and Kathryn E Stecke. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE transactions*, 30(8):689–694, 1998.
- Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.
- Gilbert Laporte, Juan Jose Salazar-Gonzalez, and Frederic Semet. Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36(1):37–45, 2004.
- Alexandre Linhares and Horacio Hideki Yanasse. Connections between cutting-pattern sequencing, {VLSI} design, and flexible machines. *Computers & Operations Research*, 29(12):1759 – 1772, 2002. ISSN 0305-0548. doi: [http://dx.doi.org/10.1016/S0305-0548\(01\)00054-5](http://dx.doi.org/10.1016/S0305-0548(01)00054-5). URL <http://www.sciencedirect.com/science/article/pii/S0305054801000545>.
- Gustavo Silva Paiva and Marco Antonio Moreira Carvalho. Um método para planejamento da produção em sistemas de manufatura flexível. In *Anais do XLVIII Simpósio Brasileiro de Pesquisa Operacional*, pages 3717–3724, 2016.
- Caroline Privault and Gerd Finke. k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, 96(1-4):255–269, 2000.
- Csaba Raduly-Baka and Olli S Nevalainen. The modular tool switching problem. *European Journal of Operational Research*, 242(1):100–106, 2015.
- Edson Luiz França Senne and Horacio Hideki Yanasse. 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, pages 68–72, 2009.
- Rabia Shirazi and GDM Frizelle. Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research*, 39(15):3547–3560, 2001.

- Christopher S Tang and Eric V Denardo. Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, 36(5):767–777, 1988.
- Michal Tzur and Avri Altman. Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions*, 36(2):95–110, 2004.
- HH Yanasse and MJP Lamosa. 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*, page 90, 2005.
- HH Yanasse and MJP Lamosa. On solving the minimization of tool switches problem using graphs. In *Annals of the XII ICIEOM – International Conference on Industrial Engineering and Operations Management*, 2006.
- Horacio Hideki Yanasse and Maria José Pinto. 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, 2002.
- Horacio Hideki Yanasse, Rita de Cássia Meneses Rodrigues, and Edson Luiz França Senne. Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão and Produção*, 13(3), 2009.
- Bing-Hai Zhou, Li-Feng Xi, and Yong-Shang Cao. 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, 2005.