Um Algoritmo de Enumeração Implícita Para o Problema do Caixeiro Viajante

Thiago Cantos Lopes
Leandro Magatão

Programa de Pós-Graduação em Engenharia Elétrica e Computação Industrial - CPGEI
Universidade Tecnológica Federal do Paraná - UTFPR
Av. Sete de Setembro, 3165 - Rebouças CEP 80230-901
Curitiba, Brasil

***Resumo*—O do caixeiro viajante é um problema clássico de otimização, amplamente estudado na literatura. Se por um lado há muitos procedimentos exatos e heurísticos para esse problema, por outro pouco foco foi dado a processos de enumeração. Este artigo apresenta um novo procedimento de enumeração implícita de busca de início guloso para o problema do caixeiro viajante. Esse procedimento oferece complexidade quadrática de complexidade em memória em relação ao número de cidades. A enumeração é baseada em um limitante local também aqui descrito. Com o algoritmo, problemas de pequeno tamanho (20 cidades) foram resolvidos em segundos, problemas modestamente maiores (40 cidades) levaram muito mais tempo. Soluções ótimas apresentaram poucas decisões não-gulosas, indicando uma direção para possíveis heurísticas. Se por um lado, o algoritmo pode não ser prático para problemas grandes, por outro ele pode resolver problemas de tamanho prático. Direções para melhora de performance são apresentadas bem como dicas para adaptar a técnica desenvolvida para outros problemas.**

***Palavras-Chave*—Problema do Caixeiro Viajante, Enumeração Implícita, Limitante Local**

I. Introdução

O problema do caixeiro viajante (CV) é um problema clássico de otimização combinatorial [1]. O objetivo é minimizar o custo de um circuito que visita todas as cidades dados os custos de viajem entre cada par de cidades. Há uma ampla variedade de procedimentos de solução para esse problema, desde heurísticos [2], meta-heurísticos [3], até exatos [4].

Variantes exatas para CV são frequentemente baseadas em Branch-and-Cut [5] ou programação dinâmica [6], ambas podendo requerer complexidade exponencia de memória ($O(2^N)$) em relação ao número de cidades N. Procedimentos de Enumeração são frequentemente desconsiderados devido ao número de soluções possíveis: $(N-1)!/2$ para o problema simétrico e $(N-1)!$ para o problema assimétrico. Isso significa que se for possível enumerar 3 bilhões de soluções por segundo, um problema simétrico de 20 cidades demoraria 230 dias para ser enumerado exaustivamente, (já um de 40 cidades requereria 10^{29} anos).

No entanto, não é necessário enumerar todas as soluções se for possível estabelecer um limitante local para soluções parciais. Esse é o conceito chave por trás de algoritmos de enumeração implícita [7]: codificar soluções em partes e estabelecer limitantes para soluções parciais. Isso permite desconsiderar inúmeras soluções sub-ótimas rapidamente, mas requer o cálculo de um limitante local. O presente trabalho é baseado nos princípios de enumeração implícita descritos em [8], comumente referido como método de Balas.

A Seção II apresenta o pseudocódigo do algoritmo desenvolvido e descreve o seu funcionamento. A Seção III descreve em maior detalho o limitante local empregado no processo de enumeração. A Seção IV apresenta um exemplo ilustrativo de um problema de 20 cidades. A Seção V discute os resultados obtidos bem como algumas considerações práticas. A Seção VI descreve direções para trabalhos futuros, melhorias e adaptações. A Seção VII conclui o artigo e resume suas contribuições.

II. Algoritmo

O algoritmo geral possui a estrutura ilustrada pela Figura 1: Decisões são tomadas em uma busca gulosa em profundidade até que um limitante local determine que as soluções remanescentes são necessariamente piores que a incumbente. Quando isso ocorre essas soluções são consideradas implicitamente enumeradas e descartadas. Decisões são então desfeitas, as últimas primeiro, até que o limitante aponte a possibilidade de novas soluções ou a enumeração esteja concluída. Se a enumeração não estiver concluída, o processo de busca gulosa continua em profundidade.

```
BEGIN
|    Initialization (INI)
|    DO UNTIL D=0
|    |    IF UB > LLB
|    |    |    Greedy Depth First Search Routine (GDFSR)
|    |    ELSE
|    |    |    Undo Last Decisions Routune (ULDR)
|    |    |    Set Decision to Next Option Routine (SDNOR)
|    |    END IF
|    END DO
END
```

Figura 1. Pseudocódigo do Algoritmo Principal, sub-rotinas cujos pseudocódigos estão apresentados estão destacadas en negrito entre parenteses

O processo de inicialização essencialmente carrega a informação do problema, define as cidades como objetos que contem e define listas (chamada Vizinhos) que contem todas as demais cidades, ordenadas pelas distâncias. Esse procedimento é descrito pela Figura 2 e também é responsável por configurar o valor de D como 1. Isso significa que a decisão atual a ser tomada é para onde ir a partir da primeira cidade.

```
BEGIN INI
|     Create Cities
|     Define Distance Matrix
|     FOR EACH City
|     |     Include all (N-1) other Cities in the City's Neighbors List
|     |     Sort this list in order of the closest first
|     NEXT
|     D <- 1
END INI
```

Figura 2. Processo de Inicialização: cada a cidade é dada uma lista de todas as demais cidades ordenada pela distância em relação a cidade que possui a lista. Essas listas são empregadas no processo decisório para simplificar a busca.

O procedimento de tomada de decisões guloso em um profundidade é descrito pela Figura 3. Ele pode ser visto como um passo da heurística do vizinho mais próximo: cada cidade deve se conectar ao vizinho disponível mais próximo. O princípio em jogo aqui é que boas soluções devem seguir essa regra para a maioria das cidades, logo deve-se avaliar essas soluções antes. Isso deve acontecer se o limitante inferior local (LIL) for melhor que a incumbente ou limitante superior (LS), caso contrário as soluções serão garantidamente sub-otimas: quanto mais decisões forem tomadas, mais restrito é o subproblema restante e pior o LIL será.

```
BEGIN GDFSR
|     Set Dth City Decision to Best Remaining Option
|     D <- D+1
|     Update LLB
|     IF Solution is Complete AND LLB<UB
|     |     UB <- LLB
|     |     Set Incumbent to Solution
|     END IF
END GDFSR
```

Figura 3. Sub-rotina gulosa de tomada de decisão em profundidade: Decisões são tomadas nessa parte do código seguindo um passo da heurística do vizinho mais próximo. Se a solução parcial for completa, ela é aqui comparada à incumbente

Quando o limitante local se torna pior ou igual a incumbente, entende-se que as decisões anteriormente tomadas não podem levar a soluções melhores. Infere-se que elas devam ser desfeitas e substituídas, se possível, por alternativas imediatamente "menos gulosas". Para fazer isso, primeiramente se desfazem as últimas escolhas até que o LIL se torne melhor que o LS, ou incumbente. Este procedimento é descrito pela Figura 4.

A sub-rotina ULDR em geral desfaz apenas algumas, ou mesmo uma, decisão no início da execução. No entanto, conforme melhores incumbentes são obtidas e as soluções

```
BEGIN ULDR
|     WHILE LLB >= UB AND D>0
|     |     D <- D-1
|     |     Undo the Dth Decision
|     |     Update LLB
|     END WHILE
END ULDR
```

Figura 4. Procedimento de Desfazer Decisões, sua função é implicitamente enumerar uma série de soluções dominadas

gulosas iniciais substituídas, esse processo desfaz múltiplas decisões simultaneamente, acelerando a enumeração conforme maiores conjuntos de soluções são implicitamente dominados.

É possível ainda que em um dado nível a decisão atual seja a menos gulosa possível: A primeira cidade pode se conectar a qualquer uma das $N-1$ demais, a segunda à qualquer das $N-2$ restantes, a M^{ma} cidade à uma das $N-M$ restantes. Se o algoritmo chega à $(N-M)^{ma}$ decisão para a M^{ma} cidade, ao invés de buscar a $(N-M+1)^{th}$ alternativa, a decisão do nível anterior deve ser desfeita. Isso deve ocorrer ainda que LIL<LS e é absolutamente necessário para o progresso adequado da enumeração. Isso é umas das funções da sub-rotina de Mudança de Decisão (SDNOR), descrita pela Figure 5.

```
BEGIN SDNOR
|     DO Until Decision is made or Enumeration is complete
|     |     IF There are Remaining Available Options to Dth Decision
|     |     |     Set Dth Decision to Next Available Value
|     |     |     Update LLB
|     |     |     EXIT DO     Decision is made
|     |     ELSEIF D>1
|     |     |     D <- D-1
|     |     |     Undo the Dth Decision
|     |     |     Update LLB
|     |     ELSE
|     |     |     D <- 0
|     |     |     EXIT DO     Enumeration is Complete
|     |     END IF
|     LOOP
END SDNOR
```

Figura 5. Procedimento de Mudança de Decisão, sua função central é realizar a operação após a condição **ELSEIF**: mudar a direção de busca para a alternativa mais gulosa ainda não explorada.

Esse procedimento é o que determinará a conclusão da enumeração quando a última decisão para a primeira cidade for enumerada juntamente ao espaço de busca subsequente: Quando isso ocorrer o problema esta implicitamente enumerado e a condição de otimalidade esta satisfeita.

III. Limitante Inferior Local

Para poder enumerar implicitamente todas as soluções, é preciso obter um limitante seguro da qualidade de todas as possíveis soluções dada uma solução parcial. No caso particular do algoritmo proposto para o CV, uma solução parcial é dada por um conjunto inicial de conexões: Cidade A conecta à B, Cidade B conecta à Cidade C, etc..

O procedimento de conexão segue o algoritmo descrito na seção anterior, no qual as decisões são sequenciais: Elas sempre partem da última cidade conectada, o que facilita a implementação e previne subciclos. Dado um conjunto de conexões realizadas, é seguro afirmar que o "custo estabelecido"por essas conexões é um limitante inferior de soluções contenedoras da atual parcial. No entanto, a dificuldade está em limitar a qualidade do resto da solução, ao invés de sua parte inicial.

Da mesma forma que a escolha das conexões é definida em termos do conjunto ordenado de Vizinhos de cada cidade, o limitante local é calculado com base nesses conjuntos: Todas as cidades somam metade das distancias das duas cidades disponíveis mais próximas. Esse é um uso sinérgico dos procedimentos de ordenação da inicialização, já que a informação obtida com as listas ordenadas é usada tanto na busca gulosa em profundidade quanto no limitante inferior. A Figura 6 ilustra como o limitante local é calculado, somando ao "custo estabelecido"(linhas em preto das decisões tomadas) ao limite do mínimo custo restante das decisões subsequentes (linhas em vermelho).

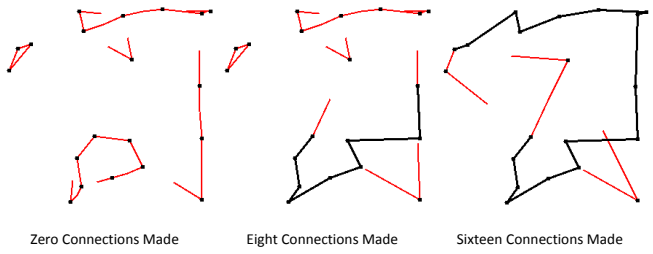

Figura 6. Ilustração do funcionamento do limitante inferior, calculado conforme decisões são tomadas. Sempre que conexões são feitas ou desfeitas, o limitante é atualizado. As linhas em preto representam a solução parcial, enquanto as vermelhas o mínimo custo restante.

Esse limitante é baseado em um limitante da literatura definido em programação linear [9], no qual a cada cidade é associado um raio e o objetivo é a maximização da soma dos raios de forma que os mesmos não se sobreponham. Os autores suspeitam que o limitante proposto seja mais forte que sua inspiração, mas nenhum estudo estatístico foi realizado para verificar tal relação. A Figura 7 ilustra um caso em que o limitante proposto é mais forte que o da maximização da soma dos raios em algumas partes e mais fraco em outras.

O limitante da minimização dos raios [9] é dado pela Maximização de $\sum_c R_c$, onde R_i é o raio da cidade i sujeito à $R_i + R_j \leq D_{ij}$ para todo par de cidades diferentes i e j. Há dois problemas com esse limitante: Primeiramente, seria necessário adaptá-lo para oferecer limitantes para soluções parciais. Em segundo lugar, mas mais importante, ele requer um software de resolução de programação linear que precisará ser usado iterativamente muitas vezes. Tecnicas de re-otimização poderiam ser empregadas para não requere reinícios crus, mas requiriria sofisticadas conversões entre linguagens de programação e escrita de arquivos texto.

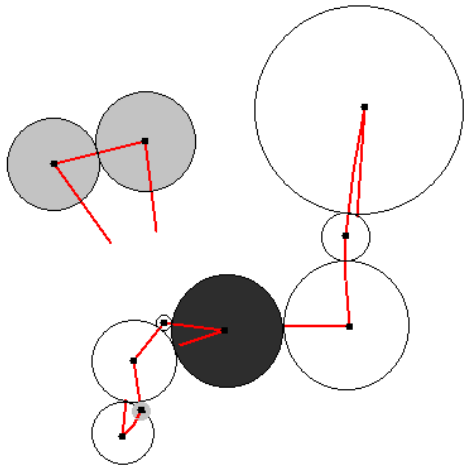

Figura 7. Comparação entre o limitante inferior desenvolvido e sua inspiração, a relaxação linear da maximização da soma radial. O circulo cinza escuro indica um nó para o qual o limitante proposto oferece limite mais fraco. O contrario ocorre nos nós cinza claro e uma equivalência ocorre nos demais.

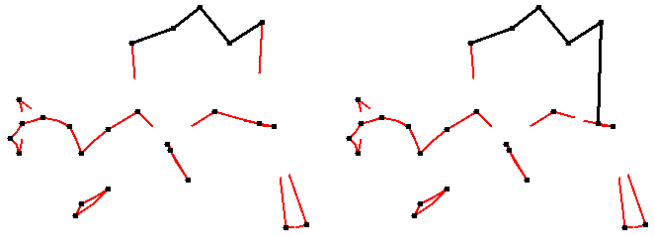

Figura 8. Ilustração de como o limitante local muda apenas para os nós conectados e para os nos que estavam ligados aos conectados pelo limitante inferior.

Essas são duas vantagens do método proposto: Ele pode ser executado na mesma linguagem do algoritmo e incorpora o aspecto local. Uma terceira vantagem esta no fato de que após uma conexão, o limitante só é alterado para os nós relacionados à conexão, conforme ilustrado pela Figura 8: o limitante inferior é o mesmo para praticamente todos os nós. Uma consequência direta desse fato é a oportunidade de, ao invés de recalcular o limitante a cada iteração, simplesmente atualizá-lo nos nós relevantes.

IV. Exemplo Ilustrativo

Ilustramos a execução do algoritmo para um problema simétrico de 20 cidades cujas coordenadas foram aleatoriamente geradas (Tabela I). As distâncias foram calculadas pelo arredondamento da distância euclidiana para o inteiro mais próximo. O problema foi resolvido até otimalidade em menos de 10 segundos de tempo computacional com 14 incumbentes encontradas durante a execução, quatro das quais estão ilustradas na Figura 9, note que essas são sa mesmas cidades cuja solução gulosa é ilustrada em construção na Figura 6.

Essas soluções podem ser codificadas de acordo com suas decisões $X[D]$, cada qual responde a pregunta: Na D^{ma}

Tabela I
COORDENADAS PARA CIDADES NO PROBLEMA ILUSTRATIVO DE CV, ARREDONDE OS CUSTOS PARA O INTEIRO MAIS PRÓXIMO PARA OBTER OS VALORES NA FIGURA 9 E FIGURA 10

N	1	2	3	4	5	6	7	8	9	10
X	46	94	99	38	17	62	95	41	40	35
Y	65	42	8	75	23	67	66	17	88	95
N	11	12	13	14	15	16	17	18	19	20
X	95	54	68	95	77	11	39	63	59	7
Y	9	84	79	94	7	25	8	30	10	35

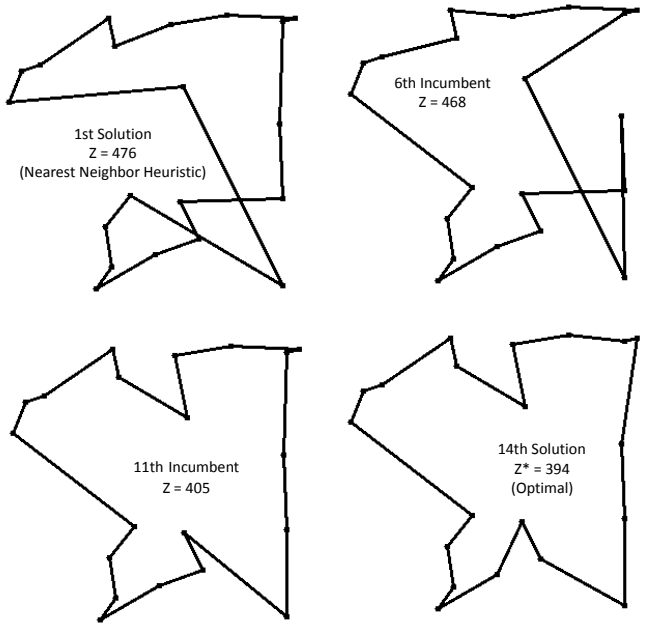

Figura 9. Quatro das quatorze soluções incumbentes obtidas durante a enumeração.

cidade, dentre as alternativas disponíveis, para qual o caixeiro deve is? A resposta dada por $X[D]$ é "Para a X^{ma} mais próxima". Nesse sentido podemos esperar que boas soluções tenham a maioria (mas não todas) $X[D]$ configuradas em valores pequenos ou mesmo em um , indicando a escolha do vizinho mais próximo. De fato, os códigos obtidos para cada uma as soluções tem um grande número de 1s, a resposta ótima foi a que teve o maior número de decisões não gulosas (quatro dentre dezenove). Os códigos correspondentes às quatro soluções mostradas na Figura 9 são mostrados na Figura 10

A codificação apresentada (Figura 10) para a solução ótima oferece uma dica de um possível procedimento heurístico: Para a maioria das cidades a melhor escolha era a escolha gulosa. Para apenas algumas, a resposta era: vá para a segunda ou terceira cidade mais próxima. Uma possível busca heurística seria baseada em iterativamente distribuir exceções à regra do vizinho mais próximo. Isso poderia também ser executado em paralelo ao algoritmo principal para acelerá-lo, pela obtenção de melhores incumbentes.

É importante remarcar que para processar somas e

Decision Codes	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
Itinerary	1	4	9	10	12	13	6	7	2	11	3	15	19	8	17	5	16	20	18	14	1
1st Incumbent																			Z=	476	
Decision Codes	1	1	1	1	1	1	1	1	6	1	5	1	1	1	2	1	1	1	1		
Itinerary	1	4	9	10	12	13	6	7	2	14	18	11	3	15	19	17	8	5	16	20	1
6st Incumbent																			Z=	468	
Decision Codes	1	1	1	1	1	1	4	1	1	1	1	1	1	3	1	1	1	1	1		
Itinerary	1	4	9	10	12	13	6	14	7	2	11	3	15	19	18	8	17	5	16	20	1
11st Incumbent																			Z=	405	
Decision Codes	1	1	1	1	2	1	2	1	1	3	1	1	1	3	1	1	1	1	1		
Itinerary	1	4	9	10	12	6	13	14	7	2	3	11	15	19	18	8	17	5	16	20	1
Optimal Solution																			Z*=	394	

Figura 10. Quatro dentre as quatorze soluções obtidas durante a Enumeração

subtrações mais rapidamente, foram consideradas distâncias inteiras, levando a resposta de $Z = 394$, mas se custos reais forem considerados, a solução ótima teria valor $Z = 394.248$ e as cidades 9 e 10 teriam suas posições trocadas no circuito ótimo. O mesmo circuito pode ser obtido pelo algoritmo proposto, multiplicando as coordenadas por by 10 obtendo os custos inteiros da mesma forma.

V. RESULTADOS E DISCUSSÕES

Esse método foi aplicado para pequenos problemas de CV gerados aleatoriamente por coordenadas, soluções ótimas foram obtidas em segundos e sua prova ocorreu em mais alguns segundos. O exemplo ilustrativo foi resolvido em menos de 10 segundos até otimalidade.

Casos com mais cidades levaram significativamente mais tempo: CVs com tamanho 30 demoraram mais de uma hora para ser enumerados. O maior caso resolvido foi um CV de 40 cidades implicitamente enumerado em pouco menos que três dias de tempo computacional. Considerando que o problema tem $39!/2$ soluções, dado que o computador usado opera a 2.3GHz, se for possível avaliar uma resposta por operação do relógio então apenas $5.84 \cdot 10^{-32}\%$ das soluções do problema de 40 cidades foram enumeradas, uma performance interessante.

Esse algoritmo naturalmente não compete com algoritmos de branch-and-cut de estado da arte, que já foram usados para resolver instancias com mais de 20000. No entanto para alcançar tais resultados esses algoritmos empregam clusters e supercomputadores requeridos por um longo período de tempo.

Por outro lado um problema de roteamento de um único veiculo que de fato use uma formulação em Caixeiro Viajante no mundo real dificilmente realizará mais de 20 entregas por dia: Com uma entrega a cada 15 minutos, um turno de quatro horas realizará 16 entregas.

O algoritmo apresentado é simples, oferece otimalidade para pequenos problemas, não requer computação paralela ou uma licença de software comercial e tem uma baixa complexidade em memória de O(N^2). Comparados por exemplo com variantes exatas de programação dinâmica, sua complexidade é O(2^N) (isto é exponencial).

VI. TRABALHOS FUTUROS

Cinco direções de possíveis trabalhos futuros foram identificadas pelos autores. A primeira consiste em fortalecer a

informação adquirida pela *matriz de distâncias*. A segunda em acelerar a aquisição de *limites superiores* para o problema. A terceira em fortalecer e acelerar o *limitante inferior*. A quarta na expansão da proposta formulação para *outros problemas* similares. A quinta em uma heurística baseada na distribuição de escolhas não gulosas.

O problema deve ter sua resolução significativamente acelerada se sua *matriz de distâncias* passar por um processo de geração de zeros similar ao ocorrido no problema de designação quando resolvido pelo método Húngaro [10]. A matriz final do referido método oferece informações mais fortes que podem mudar quais cidades estão mais "próximas"uma das outras avaliando as consequências de conexões no problema da designação, que é uma relaxação do problema do caixeiro viajante.

O processo de enumeração tende a ser significativamente melhorado se o *limitante superior* inicial for melhor que a heurística do vizinho mais próximo. Bons limitantes primais podem ainda ser buscados em paralelo ao processo de enumeração, possivelmente inspirados na codificação esperada da solução ótima do problema como mencionado na Seção IV. A enumeração propriamente dita pode também ser paralelizada a partir da primeira cidade.

Os limites duais, ou *limitante inferior*, podem ser melhorados ou pela implementação de técnicas de atualização (quando comparadas a recalcular o limitante do início) ou pelo emprego de limitantes mais fortes, por exemplo inspirados em problemas de árvore geratriz mínima [11], ou o próprio problema de designação. Provavelmente tais limitantes também poderão ser atualizados de ao invés de totalmente recalculados.

Além disso, *problemas de roteamento de veículos* são candidatos naturais para adaptação dos métodos propostos: Os conceitos são semelhantes, apesar da necessidade de significativas alterações para considerar características como janelas de tempo, múltiplos veículos e limitações de capacidade. Por outro lado, o algoritmo de enumeração é um procedimento de busca em profundidade orientado possibilidade (guloso primeiro) cujos princípios básicos podem ser adaptados a qualquer problema cujas soluções podem ser escritas como composição de uma sequencia de decisões e ordenadas das mais gulosas às menos gulosas

Por fim, foi verificado que soluções ótimas possuem poucas decisões não gulosas. Isso pode ser empregado para desenvolver procedimentos de busca para obter soluções escolhendo aleatoriamente quais cidades devem evitar decisões gulosas.

VII. Conclusões

Um novo procedimento de enumeração implícita para pequenas instancias do problema do caixeiro viajante foi apresentado. O procedimento é baseado em uma busca gulosa em profundidade ao qual um limitante inferior foi incorporado. O algoritmo mostrou-se eficiente na produção de soluções ótimas para problemas com vinte cidades, mas requer muito mais tempo computacional para problemas maiores.

Se por um lado há técnicas mais rápidas para resolver problemas de larga escala (dezenas de milhares de cidades), por outro a técnica desenvolvida oferece uma complexidade em memória enxuta de O(N^2). Além disso, o algoritmo dispensa o uso de licenças de softwares comerciais ou supercomputadores empregados para resolver algumas das mais desafiadoras instancias do caixeiro viajante.

Direções para trabalhos futuros são apresentados, desde melhorias ao procedimento de enumeração em si à aplicação do algoritmo à outros problemas e possíveis procedimentos heurísticos que podem ser desenvolvidos com base no conceito principal deste algoritmo.

Agradecimentos

Ao apoio financeiro da Fundação Araucária (Agreement 141/2015 FA – UTFPR – RENAULT) and CNPq (Grant 305405/2012-8).

Referências

[1] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and Its Variations*. Boston, Mass: Springer, 2007.

[2] D. J. Rosenkrantz and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 5, pp. 563–581, 1977.

[3] S. H. Nasseri and M. H. Khaviari, "Solving tsp by considering processing time: Meta-heuristics and fuzzy approaches," *Fuzzy Information and Engineering*, vol. 3, no. 4, pp. 359–378, 2011.

[4] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *Siam Review*, vol. 33, no. 1, pp. 60–100, 1991.

[5] H. Hernández-Pérez and J. Salazar-González, "A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery," *Discrete Applied Mathematics*, vol. 145, no. 1, pp. 126–139, 2004.

[6] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Indutrial and Applied Mathematics*, vol. 10, no. 1, pp. 196–210, 1962.

[7] E. Balas, "An additive algorithm for solving linear programs with zero-one variables," *Operations Research*, vol. 13, pp. 517–546, 1965.

[8] A. M. Geoffrion, "Integer programming by implicit enumeration and balas' method," *SIAM Review*, vol. 9, pp. 178–190, 1967.

[9] WaterlooUniversity, "Tsp control zones," http://www.math.uwaterloo.ca/tsp/methods/opt/zone.htm, 2005, acessed: 2016-08-10.

[10] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[11] H. Yeo, "The 1-tree lower bound for tsp," http://www.ieor.berkeley.edu/kaminsky/ieor251/notes/3-16-05.pdf, 2005, acessed: 2016-08-10.