Análise de Desempenho do Algoritmo TSP

Luiz Marcio Faria de Aquino Viana, M.Sc.

Engenharia de Sistemas e Computação

COPPE / UFRJ - Universidade Federal do Rio de Janeiro
Ilha do Fundão – Rio de Janeiro – RJ – Brasil

lmarcio@cos.ufrj.br

Resumo. *Este artigo apresenta a avaliação de desempenho de dois algoritmos usados na solução do Problema Simétrico do Caxeiro Viajante (TSP) em ambientes multiprocessados. Neste estudo verificamos que o desempenho do algoritmo de Força Bruta (FB) possui grande escalabilidade com o aumento no número de núvcleos de processamento. Porém, a quantidade de caminhos que precisam ser analisados aumenta muito com o aumento no número de nós do grafo. Desta forma, também analisamos o problema usando uma heurística míope, Metodo da Inserção Mais Próxima (H), que apresentou ótimo desempenho computacional, boa aproximação do resultado final, mas com pouca escalabilidade devido ao excesso de regiões críticas e concorrência nos dados.*

1. Introdução

O objetivo desta pesquisa é analisar o desempenho dos algoritmos paralelos de solução do Problema Simétrico do Caixeiro Viajante (TSP) usando as bibliotecas OpenMP e OpenMPI em ambientes multiprocessados.

O Problema Simétrico do Caxeiro Viajante (TSP) ocorre com frequência no nosso cotidiano, e possui diversos algorítimos determinísticas conhecidos. Estes algoritmos avaliam todas as permutações dos N nós do grafo, isto é, avaliam o custo dos N! diferentes caminhos possíveis de visita aos nós.

Este problema possui complexidade $O(N^2 \times Log_2 N)$, e existem algoritmos determinísticos que resolvem este problema com complexidade $O(N^3)$.

Nesta pesquisa, analisamos este problema usando um algorítimo de Força Bruta (FB) de complexidade $O(N^3)$ que avalia o custo de todos os caminhos possíveis e seleciona o de menor custo,

Em seguida, implementamos um algoritmo que utiliza uma Heurística Míope de complexidade $O(N^2)$, e comparamos o desempenho deste algoritmo em ambientes multiprocessados.

2. Conceitos

O Problema Simétrico do Caxeiro Viajante (TSP) consiste em um conjunto de nós de um grafo, onde o número de nós é maior ou igual à três, e existe pelo menos um caminho que passa por todos os nós do grafo.

Este caminho, é denominado de Caminho Hamiltoniano, e todo o grafo que possui pelo menos um Caminho Hamiltoniano, é denominado de Grafo Hamiltaniano.

2.1. Definição do Problema

Seja G = (N, M), um grafo com no mínimo três nós, onde as cada arésta possui um custo Cij associado. Este grafo possui pelo menos um caminho que permite visitar todos os seus nós, sendo desta forma um Grafo Hamiltaniano.

O Problema Simétrico do Caxeiro Viajante (TSP) consiste em encontrar um percurso com custo mínimo que passe por todos os nós.

Na *Figura 1*, podemos ver um exemplo de Grafo Hamiltoniano, G, que pode ser usado no Problema Simétrico do Caxeiro Viajante (TSP). Neste grafo o custo para se deslocar entre os nós N_1 e N_5 é o mesmo custo para se deslocar de volta de N_5 para N_1, e desta forma, este é considerado um problema simétrico.

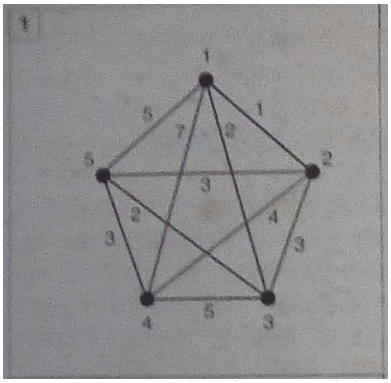

Figura 1: Exemplo de Grafo Hamiltoniano G.

A *Figura 2*, apresenta uma das possíveis soluções de custo mínimo encontrada para o Problema Simétrico do Caxeiro Viajante (TSP) para o grafo *G*.

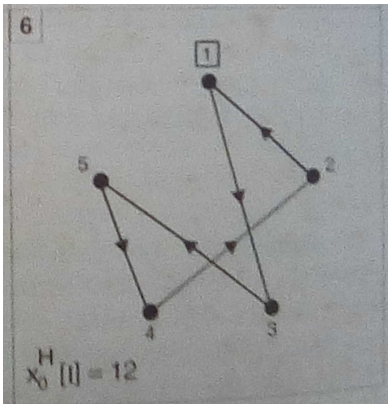

Figura 2: Solução do Problema Simétrico do Caixairo Viajante (TSP) para o Grafo G.

2.2. Resolução do Problema por Força Bruta (FB)

Para encontrar a solução ótima por Força Bruta (FB), precisamos avaliar a permutação dos N nós do grafo G, isto é, precisamos avaliar cerca de N! caminhos possíveis para visitar todos os nós do grafo.

Nesta análise usamos um algorítimo de Força Bruta (FB) com complexidade $O(N^3)$ que avalia todas as permutações e seleciona a de menor custo. A *Tabela 1*, apresenta uma lista de permutações possíveis para o grafo G que possui cinco nós, totalizando: (5 – 1)! = 4! = 24 caminhos.

N(I)	***N_1***	***N_2***	***N_3***	***N_4***		***N(I)***	***N_1***	***N_2***	***N_3***	***N_4***
1	2	3	4	5		1	4	2	3	5
1	2	3	5	4		1	4	2	5	3
1	2	4	3	5		1	4	3	2	5
1	2	4	5	3		1	4	3	5	2
1	2	5	3	4		1	4	5	2	3
1	2	5	4	3		1	4	5	3	2
1	3	2	4	5		1	5	2	3	4
1	3	2	5	4		1	5	2	4	3
1	3	4	2	5		1	5	3	2	4
1	3	4	5	2		1	5	3	4	2
1	3	5	2	4		1	5	4	2	3
1	3	5	4	2		1	5	4	3	2

Tabela 1: Lista de caminhos possíveis para o Grafo G (4! = 24 caminhos).

Quando usamos um algoritmo de Força Bruta (FB) para obter uma solução determinística para este problema, esbarramos no rápido crescimento no número de caminhos que precisam ser avaliados em função do aumento no número de nós do grafo.

Desta forma, a solução deste problema com o uso de algoritmos de Força Bruta (FB), embora seja extremamente escalável em máquinas multiprocessadas, esbarra no crescimento acelerado do número de percursos que precisam ser avaliados quando aumentamos o número de nós, tornando inviável o uso prático destes algoritmos.

Exemplo do aumento no número de caminhos e o tempo aproximado de computação necessário para avaliar todas as possibilidades em uma máquinas com 2 núcleos de processamento.

1 + 5 = 6 nós...	120 possibilidades...	0,3 ms
1 + 6 = 7 nós...	720 possibilidades...	1,7 ms
1 + 7 = 8 nós...	5.040 possibilidades...	11,6 ms
1 + 8 = 9 nós...	40.320 possibilidades...	92,4 ms
1 + 9 = 10 nós...	362.880 possibilidades...	831,6 ms
1 + 10 = 11 nós...	3.628.800 possibilidades...	8,3 segundos
1 + 11 = 12 nós...	39.916.800 possibilidades...	91,5 segundos
1 + 12 = 13 nós...	6.227.020.800 possibilidades...	3,9 horas
1 + 13 = 14 nós...	87.178.291.200 possibilidades...	55,5 horas

2.3. Resolução do Problema usando Heurísticas

Para tornar computacionalmente viável o Problema do Caxeiro Viajante (TSP), implementamos um algoritmo usando uma Heurísticas Míope que procura obter um melhor desempenho em relação aos algoritmos determinísticos, mas retornando um caminho com custo igual ou bastante próximo ao caminho de menor custo do grafo.

As soluções que usam heurísticas obtem grande melhoria de desmpenho em grafos com poucos números de nós, e este desempenho aumenta exponencialmente com o aumento no número de nós. Nesta pesquisa, verificamos um ganho de 400 vezes em relação ao algoritmo determinístico usado para grafos pequenos de até 1 + 5 = 6 nós.

A Figura 3, apresenta um exemplo de solução para o problema usando o Método Heurístico das Economias, que mantém uma lista de custo dos nós ordenada de forma crescente, permitindo a inclusão dos nós que acrescentam o menor custo ao caminho em formação.

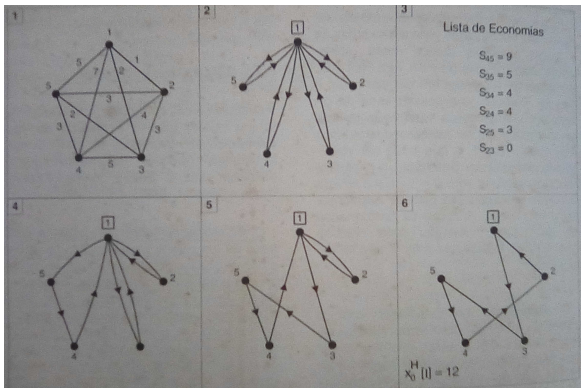

Figura 3: Solução do Problema Simétrico do Caxeiro Viajante (TSP) usando o método das economias.

2.4. Heurísticas Míopes – Inserção Mais Próxima (H)

Nesta pesquisa, implementamos um algoritmo que usa a Heurística Míope, de Inserção Mais Próxima (H), para formar o caminho de menor custo aproximado para o grafo G. Este algoritmo possui complexidade $O(N^2)$ e o resultado obtido se aproxima bastante do melhor caminho obtido pelos algorítimos determínisticos.

Este algoritmo se mostrou bastante eficiênte na solução deste problema, e permitiu a resolução do Problema Simétrico do Caxeiro Viajante (TSP) para um grafo com até 18.512 nós, em um tempo de 5,3 segundos usando processamento sequencial em um computador simples com 2 núcleos de processamento.

Este algoritmo obteve um ganho de *speedup* acima de 400 vezes para grafos com poucos nós, e ganhos exponenciais para grafos maiores, permitindo o processamento de grafos com grande números de nós.

3. Análise dos Resultados

O experimento foi realizado usando um computador simples com 2 núcleos de processamento de até 4 threads simultâneas, frequencia de operação de 2.8 Ghz, 8 GB de memória DRAM e sistema operacional Linux CentOS 7.

Os experimentos não puderam ser realizados no supercomputador Lobo Carneiro, o que inviabilizou uma análise mais precisa em termos da influencia do canal de comunicação das redes Infiniband e da topologia deste supercomputadorna solução deste problema.

3.1. Algoritmo de Força Bruta (FB)

Nesta pesquisa usando algoritmo de Força Bruta (FB), obtivemos uma excelente escalabilidade com os algoritmos paralelos, devido ao tipo de processamento que precisa gerar todos os caminhos possíveis para visitar todos os nós.

Este conjunto de dados pode ser facilmente separado e distribuído entre os nós de processamento, reduzindo de forma significativa o tempo de execução da aplicação paralela.

Na *Tabela 2* e *3*, podemos verificar que o tempo de execução do algoritmo usando a bilbioteca OpenMP foi reduzido à metade, e que para aplicações OpenMPI obteve um *speedup* maior que 80. Isto ocorreu, devido a dois fatores que influenciaram de forma significativa os resultados.

1. A biblioteca OpenMPI utiliza somente os núcleos reais de processamento e não usa o recurso de Hyperthread dos novos processadores, enquanto a biblioteca OpenMP efetua o escalonamento de forma inderterminada por padrão.

2. A concorrência dos dados compartilhados em memória pelas biblioteca OpenMP geram contenção no processamento, e ao usar OpenMPI o sistema dispara vários processos que são alocados em diferentes núcleos de processamento, que possuem áreas de dados independentes e desta forma processam os dados de forma livre e sem contenção.

TEMPO DE EXECUÇÃO (ms)				
		SEQ	MP2	MPI2
DOMÍNIO	NUM_NÓS	SEQ_FB	MP2_FB	MPI2_FB
5040	8	128	77	2
40320	9	2673	1443	31
362880	10	64603	37678	163

Tabela 2: Tempo de execução do algoritmo de Força Bruta (FB) com o crescimento no número de nós.

SPEEDUP				
DOMÍNIO	NUM_NÓS	SEQ_FB	MP2_FB	MPI2_FB
5040	8	1	2	84
40320	9	1	2	87
362880	10	1	2	396

Tabela 3: *Speedup* do algoritmo de Força Bruta (FB) com o crescimento no número de nós.

Os *Gráficos 1 e 2*, são apresentados os resultados do algoritmo de Força Bruata (FB) usado no processamento de grafos de até 10 nós. Para grafos com quantidades maiores de nós, apesar do algoritmo escalar de forma eficiênte com o acrécimo de núcleos de processamento, o crescimento no volume de dados é muito maior a capacidade computacional o que impede a analise do sistema em grafos com maior número de nós.

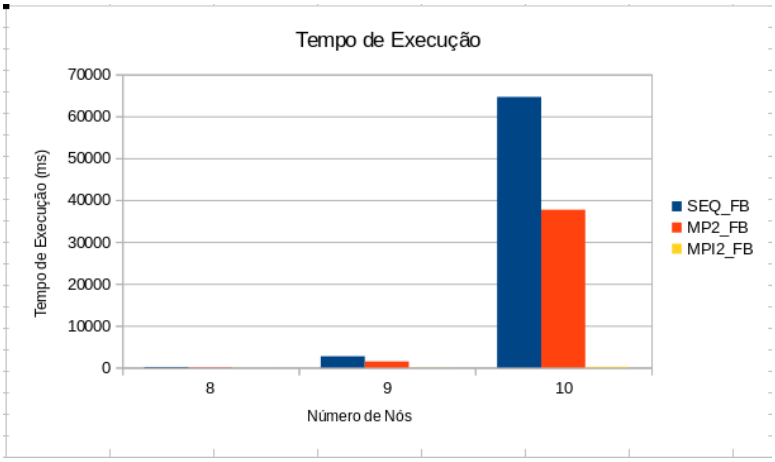

Gráfico 1: Tempo de execução do algoritmo de Força Bruta (FB) com o crescimento no número de nós.

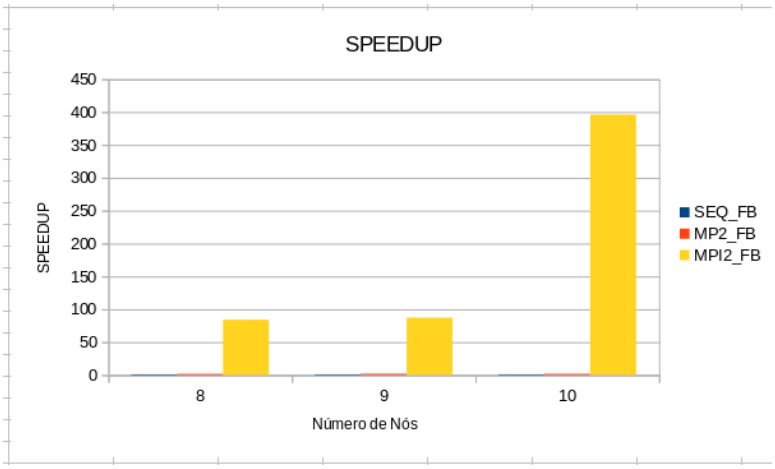

Gráfico 2: *Speedup* do algoritmo de Força Bruta (FB) com o crescimento no número de nós.

3.2. Heurísticas Míopes – Inserção Mais Próxima (H)

A implementação do algoritmo de Heurística Míopes, denominado Método da Inserção Mais Próxima (H), o tempo de processamento se mostrou extremamente eficiênte e conseguimos reduzir o tempo de execução do *Benchmark* em até 400 vezes, conforme os valores obtidos para SEQ_H, MP2_H e MPI2_H na *Tabela 4*.

TEMPO DE EXECUÇÃO (ms)							
		SEQ		MP2		MPI2	
DOMÍNIO	NUM_NÓS	SEQ_FB	SEQ_H	MP2_FB	MP2_H	MPI2_FB	MPI2_H
120	6	1334	3	991	19	265	125
720	7	13802	3	5991	15	540	133
5040	8	127627	2	77320	12	1518	133
40320	9	2673422	3	1443098	15	30683	129
362880	10	64602790	3	37677955	16	163130	194

SPEEDUP							
DOMÍNIO	NUM_NÓS	SEQ_FB	SEQ_H	MP2_FB	MP2_H	MPI2_FB	MPI2_H
720	6	1.0	444.7	1.3	0.2	5.0	0.02
5040	7	1.0	4600.7	2.3	0.2	25.6	0.02
40320	8	1.0	63813.5	1.7	0.2	84.1	0.02
362880	9	1.0	891140.7	1.9	0.2	87.1	0.02
3628800	10	1.0	21534263.3	1.7	0.2	396.0	0.02

Tabela 4: Comparação do tempo de execução e do *speedup* entre os algoritmos implementados.

Na *Tabela 4*, podemos observar que com o uso de uma heurística, obtemos melhor desempenho na resolução do problema e podemos analisar grafos com maior número de nós.

Desta forma, realizamos novas análises usando somente o algoritmo heurístico em grafos com mais de 10.000 nós e obtemos os resultados apresentados na *Tabela 5*.

TEMPO DE EXECUÇÃO (ms)			
NUM_NÓS	SEQ_H	MP2_H	MPI2_H
10512	1828.619	2200.419	2636.169
12512	2411.854	3133.349	3755.239
14512	3226.226	2789.112	5260.098
16512	4229.055	3446.21	6629.24
18512	5333.943	4977.352	8748.858

SPEEDUP			
NUM_NÓS	SEQ_H	MP2_H	MPI2_H
10512	1.0	0.8	0.69
12512	1.0	0.8	0.64
14512	1.0	1.2	0.61
16512	1.0	1.2	0.64
18512	1.0	1.1	0.61

Tabela 5: Tempo de execução e *speedup* do algoritmo de Inserção Mais Próxima (H).

Devido à eficiência do algoritmo do Método de Inserção Mais Próxima (H), podemos observar que mesmo para grafos com mais de 10.000 nós o ganho em *speedup* começa a acontecer somente após os 14.512 nós, usando a biblioteca OpenMP, devido a baixa facilidade de paralelização deste algoritmo e a concorrência dos dados em memória.

Para a implementação OpemMPI, o número de nós ainda é muito pequeno e um tempo de processamento significativo ainda é perdido na inicialização dos recursos de comunicação desta biblioteca. Desta forma, podemos assegurar que devido à eficiência do algoritmo na solução do problema, é necessário avaliar o resultado no processamento de grafos com um grande número de nós, muito mais do que os 18.512 nós usados nesta pesquisa.

Os *Gráficos 3 e 4* apresentam o tempo de execução e o *speedup* obtido com o processamento do algoritomo heirístico de forma sequencial e usando as bibliotecas OpenMP e OpenMPI.

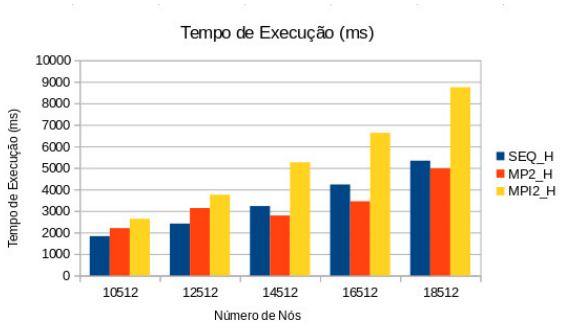

Gráfico 3: Tempo de execução do algoritmo do Método de Inserção Mais Proxima (H).

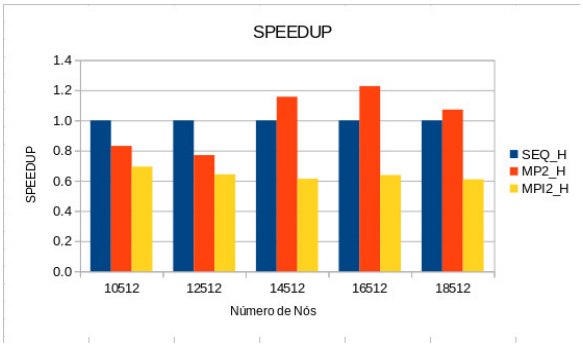

Gráfico 4: *Speedup* do algoritmo do Método de Inserção Mais Proxima (H).

4. Conclusões

O Problema do Caixeiro Viajante (TSP) é um problema que ocorre com frequência no nosso cotidiano, e para encontrar uma solução determinística é necessário avaliar todas as possibilidades, realizando a permutação dos N nós do grafo, gerando cerca de N! caminhos que precisam ser analisados.

Este problema possui complexidade $O(N^2 \times Log_2 N)$ e existem algoritmos que resolvem ele de forma determinística com complexidade $O(N^3)$.

Na pesquisa usamos um algorítimo de Força Bruta (FB) com complexidade $O(N^3)$, que avalia todas as permutações e seleciona a de menor custo, e verificamos que com o aumento no número de nós de processamento, o problema se torna computacionalmente inviável mesmo para supercomputadores com grandes quantidades de núcleos de processamento como o Lobo Carneiro.

Desta forma, para tornar o Problema Simétrico do Caxeiro Viajante (TSP) computacionalmente viável, devemos usar heurísticas que aproximam o resultado final do melhor resultado obtido com um algoritmo determinístico.

Nesta análise, implementamos o algoritmo de Heurística Míope, denominado Método da Inserção Mais Próxima (H), que possui complexidade $O(N^2)$.

Usando está heurística, obtivemos uma redução superior à 400 vezes no tempo de execução do algoritmo em relação ao tempo de processamento usando Força Bruta (FB).

Deste modo, prosseguimos com a análise do algoritmo heurístico para grafos com mais de 10.000 nós, e pudemos observar que o algoritmo heurístico é bastante eficiênte e que devido as contenções existentes ele não escala muito com o aumento no número de núcleos de processamento, e para obtermos uma avaliação mais precisa do algoritmo heurístico será necessário analisar o precessamento de grafos com mais de 18.512 nós.

5. Referências

[1] Ruy Eduardo Campello e Nelson Maculan; Algoritmos e Heurísticas:
Desenvolvimento e Avaliação de Performance; Divisão Gráfica de Furnas; Ano 1994.

[2] Gerhard Reinelt; Tutorial TSPLIB 95; Universitat Heidelberg; Ano 1995.