O Problema do Caixeiro Viajante

12/2022

Manuel Castanares

Luciano Soares

Super-computação

Índice:

Introdução:

O problema do caixeiro viajante é um dos problemas mais populares dentro da área da supercomputação, por conta de sua complexidade computacional. O enunciado do problema é o seguinte: “Dada uma lista de cidades, representadas por suas coordenadas, o objetivo é encontrar o caminho fechado (acaba na cidade que começa) que visite todas as cidades e tenha o menor comprimento possível.” Tendo isso em conta, foram propostas três soluções possíveis: fazer uma busca com heurística, uma busca local e por fim, uma busca global. Assim, foi possível comparar o desempenho de cada solução. Além disso, foram testadas as diferenças de tempo entre uma busca local sequencial, paralelizada na CPU e paralelizada na GPU. Para fazer isso, cada programa recebia as mesmas entradas, que continha o número de cidades, seguido pelas coordenadas de cada cidade.

O propósito de ter entradas iguais para cada solução é justamente para poder fazer uma comparação justa. Ainda mais, cada solução possui dez testes, cada um sendo diferente dos outros. Por exemplo, existem alguns testes que possuem somente quatro cidades enquanto outro teste conta com cem. A intenção disso, é de verificar quanto o número de cidades impacta o desempenho de cada solução em relação ao tempo e também à precisão. A única exceção, ocorre com os testes da busca exaustiva. Por conta do elevado tempo de execução de uma busca exaustiva em testes que tem um número grande de cidades, os testes dessa solução são menores. Porém, ainda foi possível fazer as comparações necessárias, para poder extrair algumas conclusões importantes.

Depois de ter criado os códigos para todas as soluções mencionadas anteriormente, foram exploradas algumas perguntas essenciais para o problema. Juntando todos os testes, as seguintes perguntas foram respostas: “Qual é o melhor método para resolver o problema?”, “É melhor investir em uma CPU com mais cores, ou uma GPU potente?” e “Vale a pena esperar pelo resultado da busca exaustiva?”. Essas respostas serão demonstradas nas seções a seguir, amparadas pelos resultados dos testes realizados.

Qual é o melhor método para resolver o problema?

Em questão de precisão, a busca exaustiva se apresenta como a melhor solução possível. Porém, o problema desse método é justamente o tempo que demora para o método encontrar a solução. O segundo método mais preciso é o da heurística, pois este escolhe a melhor rota, de acordo com uma teoria de base muito efetiva. A heurística oferece uma solução adequada para o problema, sem precisar de muito tempo. A busca local, apesar de ser mais rápida que a busca exaustiva, é um pouco mais devagar que a heurística. Além disso, a heurística apresenta resultados melhores que os da busca local, especialmente na medida que o número de cidades aumenta. Portanto, o melhor método para resolver o problema é a heurística, pois este oferece bons resultados, em um tempo curto.

Número de entradas	Método		
	Heurística	Busca Local	Busca Exaustiva
4	0,002s	0,002s	0,003s
5	0,002s	0,002s	0,003s
10	0,002s	0,003s	1,813s
15	0,002s	0,005s	1hr45min

Tabela 1.1: Diferença de tempo de execução por método.

Essa tabela demonstra a diferença de tempo entre todos os métodos propostos. Faz-se evidente a perca de tempo que ocorre quando é utilizado o método de busca exaustiva. Como mostra a tabela, com uma entrada de 15 cidades, a busca exaustiva já possui uma demora de uma hora e quarenta e cinco minutos. Por outro lado, a busca local e a heurística demoram 0.005 segundos e 0.002 segundos respectivamente. Portanto, se a precisão não for extremamente essencial, não se justifica o uso de uma busca exaustiva. Como a busca local e a heurística demoram aproximadamente o mesmo tempo para serem executados, é necessário observar qual dos métodos resulta em menores distâncias.

Número de entradas	Método	
	Heurística	Busca Local
4	4	4
5	8	8
10	357,375	331,347
15	395,41	471,371
30	532,987	1146,2
50	619,624	1776,44
100	984,7	3848,76

Tabela 1.2: Diferença de tamanho de distância, por método.

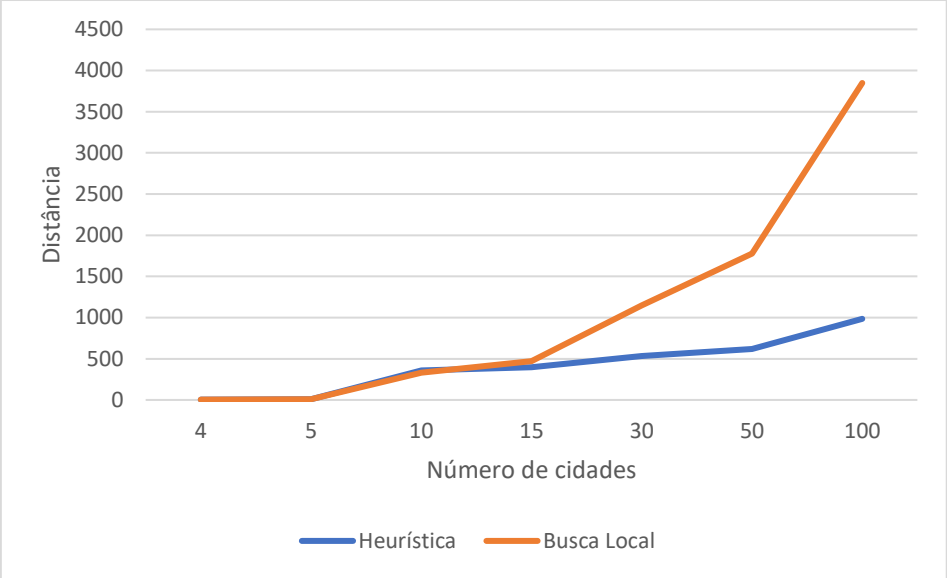

Gráfico 1.1: Plotagem das distâncias comparadas com o número de cidades.

A Tabela 1.2 e o Gráfico acima demonstram a diferença das distâncias entre os métodos de busca local e heurística. Pode ser visto uma grande diferença entre estes números, especialmente a medida em que o número de cidades aumenta. Quando o número de cidades ainda é pequeno, os dois métodos se assemelham em resultados. Porém, quando a entrada começa a ficar maior, podemos ver que a heurística mostra resultados muito superiores aos da busca local. Tendo isso em conta, podemos concluir que o melhor método para esse problema é o de heurística, pois possui um tempo de execução baixo, e um resultado de distância muito bom.

Valeria a pena gastar dinheiro comprando uma CPU com mais cores ou uma GPU potente?

Durante o percurso do projeto, foram feitas três implementações da busca local: uma sequencial, paralelizada na CPU, e paralelizada na GPU. Isso foi feito com o propósito de descobrir se é melhor investir em uma CPU com mais cores ou uma GPU mais potente. Assim, repetimos os testes com cada uma das implementações, e foi possível observar que o melhor tempo foi obtido pela paralelização feita na CPU. Porém, é importante ressaltar que esta versão é mais simples que o algoritmo feito na GPU, pois ela possui menos optimizações e loops. Por conta disso, o tempo da paralelização na CPU ficou menor mas o resultado foi menos preciso que na GPU. Quando o código da GPU foi comparado com o código sequencial, o da GPU se mostrou muito superior em questão de tempo e resultado. Tendo tudo isso em conta, seria melhor gastar dinheiro em uma GPU potente, já que a GPU consegue lidar com uma quantidade de dados muito grande. Apesar de que a paralelização na CPU funcione muito bem com uma quantidade de números pequena, o mesmo não poderia ser dito para uma

quantidade grande. A tendência que apareceu é que, a medida que o número de cidades aumenta, a GPU vai se mostrando superior ao resto.

		Ferramenta		
Teste	**Número de entradas**	**C++**	**Openmp**	**Thrust**
in-0.txt	5	0,002s	0,003s	0,360s
in-1.txt	4	0,002s	0,003s	0,372s
in-2.txt	4	0,002s	0,003s	0,362s
in-3.txt	15	0,005s	0,004s	0,359s
in-4.txt	20	0,008s	0,005s	0,359s
in-5.txt	40	0,037s	0,016s	0,383s
in-6.txt	100	0,418s	0,130s	0,607s
in-7.txt	5	0,002s	0,003s	0,372s
in-8.txt	10	0,003s	0,003s	0,363s
in-9.txt	30	0,018s	0,008s	0,374s
in-10.txt	50	0,064s	0,024s	0,410s
in-11.txt	200	3,039s	0,910s	2,200s
in-12.txt	400	22,744s	6,519s	15,545s

Tabela 2.1: Comparação entre tempo de execução e ferramenta utilizada, tendo em conta o número de entradas.

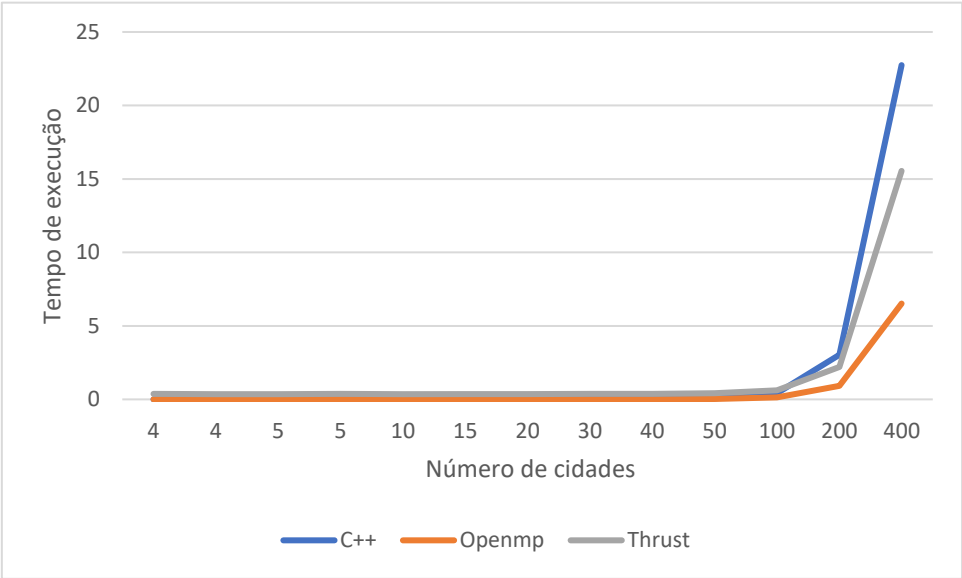

Gráfico 2.1: Tempo de execução por número de cidades.

A tabela e o gráfico acima mostram que o Openmp (paralelização na CPU), é a mais rápida de todas. Com um número de dados baixo, talvez a melhor opção seja investir em mais cores, porém a medida que o número de dados aumenta, é possível ver que a GPU se aproxima ao tempo da CPU paralelizada. O código na GPU só é mais devagar porque ela executa com mais otimizações para pegar um melhor resultado. Além disso, a cópia de dados na CPU para a GPU também aumenta o tempo total. Com uma amostra de dados pequena, esses detalhes fazem com que a paralelização na GPU seja mais devagar que na CPU. Se a entrada de cidades for um número muito grande, como, por exemplo, dez mil cidades, é muito provável que a GPU seja mais rápida. Isso aconteceria porque a GPU consegue paralelizar muito mais do que a CPU.

Vale a pena esperar pelo resultado da busca exaustiva?

Para responder esta pergunta, é necessário pensar em diferentes contextos. Em um cenário onde a precisão exata é crucial, talvez seja necessário fazer uma busca exaustiva. Nesse caso, a única opção seria esperar para que o programa seja executado. Em qualquer outro caso, não vale a pena esperar pelo resultado da busca exaustiva. A grande maioria das vezes, a busca local e a heurística oferecem resultados satisfatórios em um tempo muito reduzido. Portanto, se é necessário ter uma resposta rápida, mas um pouco menos preciso, esses dois métodos são muito melhores. Podemos ver que para todas as entradas abaixo, os métodos alternativos à busca exaustiva demonstraram bons resultados.

Número de entradas	**Método**		
	Heurística	Busca Local	Busca exaustiva
4	4	4	4
5	8	8	8
7	315,167	315,167	315,167
9	371,461	322,194	322,194
12	396,95	372,743	287,804
13	312,481	434,658	311,287
14	363,832	475,871	340,157

Tabela 3.1: Comparação entre os métodos sobre as distâncias obtidas alterando o número de cidades.

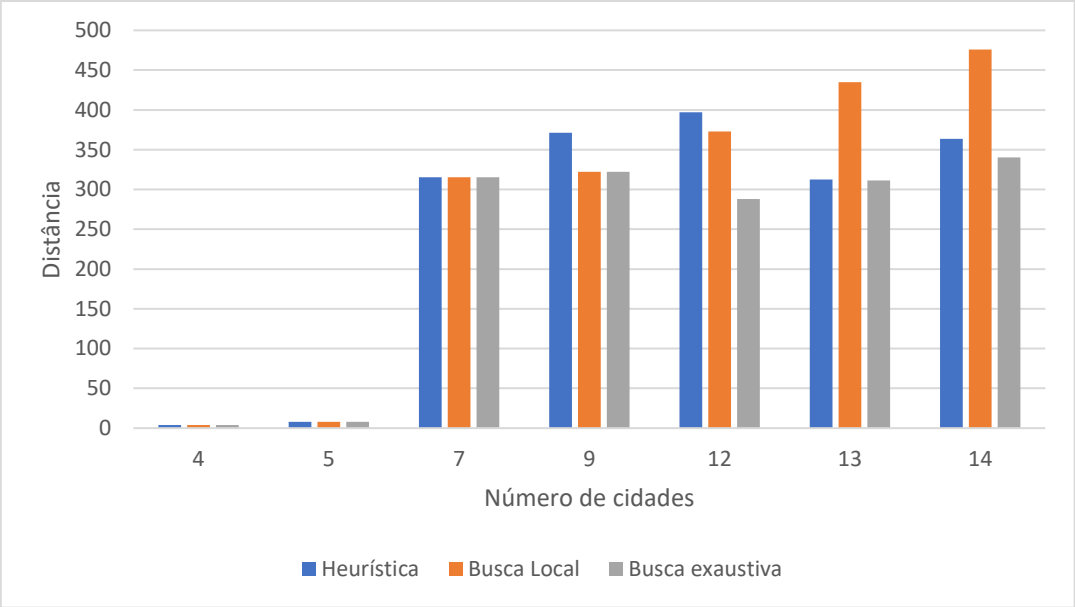

Gráfico 3.1: Diferença entre as distâncias obtidas, variando as entradas.

O gráfico e a tabela acima, demonstram a semelhança entre os resultados obtidos pelos três métodos. Porém, a grande diferença é o tempo em que cada um obteve esses resultados. Como podemos ver na tabela 1.1, para uma entrada de 15 cidades, a busca exaustiva conseguiu achar a resposta, somente depois de uma hora e quarenta e cinco minutos. Se aumentamos o número de cidades, esse tempo fica ainda maior, o que é um grande problema, pois este método não conseguiria lidar com uma entrada de cem cidades, por exemplo. Portanto, podemos afirmar que não vale a pena esperar pelo resultado da busca exaustiva, se os outros métodos são muito mais rápidos, e oferecem respostas similares.

Conclusão:

Depois de realizar todos os testes e programas, podemos tirar algumas conclusões interessantes sobre o projeto. Por exemplo, foi possível determinar que o melhor método para resolver esse problema é o da heurística, já que oferece soluções satisfatórias, em um tempo razoável. Também conseguimos verificar que, quando estamos expostos a uma grande quantidade de dados, é necessário investir em GPU, ao invés de mais cores na CPU. Finalmente, testamos a capacidade da busca exaustiva, à medida que aumentamos o número de cidades. Foi possível observar que não vale a pena utilizar este método, a menos que seja absolutamente necessário um resultado preciso. Essas conclusões são importantes, pois elas podem ser refletidas em outros problemas. Assim, no futuro, será mais fácil decidir a tecnologia e método necessários para cumprir os requisitos de algum projeto.