Trabalho Prático 2 - Soluções para Problemas Difíceis

Frederico D. S. Baker[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

fredericodolhersb@dcc.ufmg.br

Resumo. *Este estudo aborda a resolução do Problema do Caixeiro Viajante (TSP) utilizando o algoritmo de Branch and Bound e métodos aproximativos. Observou-se que o Branch and Bound não alcançou soluções ótimas dentro do limite de 30 minutos, enquanto os algoritmos aproximativos Christofides e Twice Around the Tree forneceram soluções em tempo adequado. Christofides demonstrou maior precisão, mas maior tempo de execução, ao passo que o Twice Around the Tree foi mais rápido, porém menos preciso. A escolha do algoritmo adequado depende da prioridade entre precisão e rapidez, com o Branch and Bound sendo recomendado para casos onde a solução exata é essencial.*

1. Introdução

Os problemas NP-Difíceis representam uma classe de problemas computacionais conhecida por sua complexidade intrínseca. Estes problemas são notáveis porque, até o momento, não se conhece uma solução que os resolva em tempo polinomial. Isso significa que, à medida que o tamanho da entrada aumenta, o tempo necessário para resolver o problema cresce exponencialmente, tornando-os computacionalmente custosos, especialmente para grandes entradas.

Dentro do universo dos algoritmos para solucionar problemas NP-Difíceis, existem abordagens exatas que tentam explorar sistematicamente o espaço de busca. Duas dessas abordagens são o backtracking e o branch and bound. O backtracking é uma técnica que constrói candidatos a soluções incrementalmente e descarta uma candidata assim que determina que esta candidata não pode resultar em uma solução válida. Já o branch and bound é um método que divide o problema em subproblemas menores (branching) e usa limites superiores ou inferiores para descartar certas escolhas sem explorá-las completamente (bounding).

Apesar de as abordagens exatas como o backtracking e o branch and bound explorarem o espaço de busca de uma forma inteligente, elas ainda apresentam complexidade exponencial no pior caso, o que as torna impraticáveis para instâncias grandes. Por isso, algoritmos aproximativos se tornam atraentes, pois buscam entregar soluções de qualidade satisfatória em um tempo computacional mais eficiente. Esses algoritmos não garantem a solução ótima, mas fornecem uma garantia sobre a proximidade da solução ideal.

Neste contexto, este trabalho se concentra no Problema do Caixeiro Viajante (TSP), um exemplo clássico de um problema NP-Difícil. Propomos implementar uma solução exata baseada no algoritmo de branch and bound e duas soluções aproximativas: o algoritmo Twice Around the Tree e o algoritmo de Christofides. O objetivo é comparar a eficácia dessas abordagens, tanto em termos de precisão quanto de eficiência computacional, para resolver o TSP.

2. Algoritmos Aproximativos

O Problema do Caixeiro Viajante (TSP) é um desafio clássico em otimização e teoria dos grafos. Devido à sua natureza NP-difícil, algoritmos aproximativos são frequentemente empregados. Neste trabalho, implementamos dois desses algoritmos utilizando a biblioteca NetworkX em Python: o algoritmo Twice Around the Tree (TAT) e o algoritmo de Christofides.

2.1. Algoritmo Twice Around the Tree (TAT)

O algoritmo TAT segue os seguintes passos:

1. **Construção da Árvore Geradora Mínima (MST):** O TAT inicia com a construção de uma MST do grafo do TSP. A complexidade desse processo é O(E log V), onde E representa o número de arestas e V o número de vértices.
2. **Percurso em Profundidade (DFS) e Formação do Caminho:** Após a construção da MST, o algoritmo realiza um percurso em profundidade (DFS) em preordem para visitar todos os vértices, formando assim um circuito Hamiltoniano.
3. **Fator de Aproximação:** O TAT oferece um fator de aproximação de 2, o que significa que o custo da solução encontrada será, no máximo, o dobro do custo da solução ótima.

2.2. Algoritmo de Christofides

O algoritmo de Christofides é estruturado da seguinte forma:

1. **Construção da MST e Identificação de Vértices de Grau Ímpar:** Similarmente, começa com a construção de uma MST, seguida pela identificação de todos os vértices de grau ímpar.
2. **Emparelhamento de Peso Mínimo:** O passo seguinte é encontrar um emparelhamento de peso mínimo entre os vértices de grau ímpar, adicionando estas arestas à MST. A complexidade desse passo é dominada por O(n^3).
3. **Formação do Circuito Euleriano e Conversão em Hamiltoniano:** O grafo resultante é então utilizado para formar um circuito Euleriano, que é convertido em um circuito Hamiltoniano.
4. **Fator de Aproximação:** O algoritmo de Christofides tem um fator de aproximação de 3/2, o que significa que o custo da solução encontrada será, no máximo, 1,5 vezes o custo da solução ótima.

Expectativas dos Resultados Experimentais: Espera-se que o algoritmo de Christofides, apesar de ter um tempo de execução maior devido à sua complexidade mais elevada, forneça soluções de melhor qualidade em comparação com o TAT.

3. Algoritmo exato

Nesta seção, é explorada a aplicação do algoritmo de Branch and Bound para resolver o Problema do Caixeiro Viajante (TSP) de maneira exata. O Branch and Bound é uma técnica eficiente para problemas de otimização, como o TSP, que busca reduzir o espaço de busca por meio de uma combinação de exploração sistemática e poda baseada em estimativas de custo.

3.1. Estimativa de Custo Utilizando Árvore Geradora Mínima (MST)

Inicialmente, foi tentada uma abordagem avançada para a estimativa de custo no Branch and Bound, inspirada pelos métodos adotados em [1]. Essa abordagem envolvia a utilização da Árvore Geradora Mínima (MST) do subgrafo formado pelos vértices ainda não visitados. A lógica por trás dessa estratégia era que a MST poderia fornecer uma estimativa de custo mais próxima ao custo real da solução ótima. Isso, por sua vez, teria o potencial de permitir uma poda mais eficaz do espaço de busca, excluindo ramos com estimativas de custo já superiores à melhor solução encontrada até então.

Entretanto, na prática, essa abordagem não se mostrou eficiente por dois motivos principais. Primeiro, o cálculo da MST para cada subgrafo, repetidas vezes durante a execução do algoritmo, resultou em um alto custo computacional. A necessidade de recalcular a MST a cada nova iteração do algoritmo, especialmente para subgrafos maiores, aumentava significativamente o tempo de execução. E por conseguinte, apesar da precisão teórica da abordagem, na prática, o ganho obtido com a estimativa mais precisa não foi suficiente para compensar o aumento no tempo de execução causado pelo cálculo repetido das MSTs.

Dessa forma, apesar do apelo teórico de uma estimativa de custo mais precisa usando a MST, a implementação resultou em um desempenho subótimo quando comparada com abordagens mais simples para a estimativa de custo no Branch and Bound. O alto custo computacional, possivelmente devido à maneira como foi implementado, e o ganho limitado na poda fez com que a abordagem fosse reconsiderada, buscando um equilíbrio entre a precisão da estimativa e a eficiência computacional.

3.2. Estimativa de Custo Baseada nas Duas Menores Arestas de Cada Vértice

Em vista dos desafios encontrados com a abordagem da MST, foi implementada uma estratégia mais tradicional e computacionalmente eficiente para a estimativa de custo no Branch and Bound. Esta abordagem envolve a soma das duas menores arestas de cada vértice ainda não incluído no caminho parcial. O processo é descrito a seguir:

1. Para cada vértice não presente no caminho atual, identifica-se as duas arestas de menor peso conectadas a ele.
2. Somam-se os pesos dessas duas arestas para cada um desses vértices.
3. Para os vértices já incluídos no caminho, considera-se o peso das arestas que fazem parte do caminho.
4. O limite inferior estimado para o custo total do caminho é obtido somando todos esses valores e dividindo por dois, o que reflete o custo mínimo necessário para completar o caminho a partir do estado atual.

Essa abordagem, embora menos precisa do que o cálculo da MST, é significativamente mais rápida de computar e ainda fornece uma boa estimativa para a poda efetiva do espaço de busca. A implementação desta estratégia foi otimizada através do uso de operações vetorizadas, permitindo cálculos mais rápidos e eficientes. Adicionalmente, a biblioteca Numba foi utilizada para acelerar ainda mais o processo, compilando partes do código Python para código de máquina, o que resulta em um aumento significativo no desempenho. Assim, ao equilibrar a precisão e eficiência, foi possível melhorar a performance geral do algoritmo de Branch and Bound para o TSP.

3.3. Estrutura de Dados e Abordagem de Busca

Foi a estrutura estrutura de dados Heap, especificamente um heap mínimo, para gerenciar e explorar o espaço de busca. O Heap é ideal para este propósito, pois permite recuperar e remover eficientemente o elemento com a menor prioridade, o que é crucial para a estratégia de busca.

Foi adotada uma abordagem de *best first search*, na qual inicialmente são explorados os nós do espaço de busca que possuem a menor estimativa de custo para o caminho. Essa estratégia ajuda a direcionar a busca para as soluções mais promissoras mais rapidamente.

3.4. Priorização Baseada no Aumento Médio do Limite Inferior

Conforme sugerido em [1], a ordem de busca no espaço de decisão é crucial para a eficácia do algoritmo. A abordagem definida no artigo foca em promover a exploração de subconjuntos com o menor aumento do limite inferior. A fórmula para calcular a prioridade de um subconjunto no heap é dada por:

$$P = \frac{1}{AI}, AI = \frac{LB}{k} \tag{1}$$

onde P é a prioridade de um nó, AI é o aumento médio do limite inferior, LB é o limite inferior, e k é o número de cidades visitadas.

Essa abordagem de priorização leva a uma tendência do algoritmo em direcionar a busca para as folhas do espaço de decisão, facilitando a descoberta de novos limites superiores e a poda eficiente do espaço de busca. Nessa implementação, foi considerada essa priorização sem a necessidade de inversão do AI, devido ao uso de um heap mínimo, onde um valor menor de P indica maior prioridade.

4. Resultados

4.1. Comparação entre Métodos de Priorização

Inicialmente foram comparados os métodos de priorização tradicional e otimizado na resolução do Problema do Caixeiro Viajante (TSP) através do algoritmo de Branch and Bound. Para isso, ambos os métodos foram aplicados em uma instância do TSP de tamanho 20, sendo cada um executado 100 vezes.

No boxplot da Figura 1 está plotada a dispersão do tempo de execução do algoritmo com cada um desses métodos. Fica evidente que o método otimizado apresenta tanto uma mediana quanto um quartil superior menores em comparação com o método regular. Esta observação indica que o método otimizado tende a alcançar a solução final mais rapidamente.

Além do desempenho em tempo, um aspecto crucial observado foi o número de soluções parciais (folhas do espaço de busca) alcançadas durante a execução. Em média, o método otimizado explorou cerca de 6 folhas, enquanto o método regular tendeu a explorar apenas uma. Esta diferença é particularmente relevante, pois o Branch and Bound pode ser um processo demorado para alcançar a solução ótima, assim a capacidade do método otimizado de alcançar soluções parciais durante o processo de busca é um fator

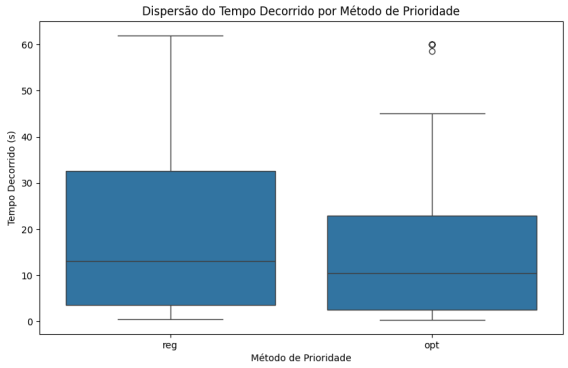

Figure 1. Dispersão dos tempos de execução de cada método de priorização

importante, especialmente em instâncias maiores do TSP, onde encontrar a solução ótima pode ser inviável dentro de um tempo razoável. Com o método otimizado, mesmo se a execução do algoritmo for interrompida antes de alcançar a solução final, as soluções parciais encontradas podem ser próximas da ótima, oferecendo uma alternativa viável. Em contraste, o método de priorização tradicional geralmente não oferece essa vantagem.

Esses resultados demonstram a superioridade do método de priorização otimizado em termos de eficiência e praticidade, especialmente em contextos onde o tempo de execução é um fator crítico.

4.2. Comparação de Tempo de execução

Na análise do desempenho de tempo dos algoritmos para resolver o Problema do Caixeiro Viajante, serão abordados os métodos aproximativos, uma vez que o algoritmo de Branch and Bound não foi capaz de retornar a solução exata dentro do limite de tempo estabelecido de 30 minutos para todos os datasets testados. Esta limitação levou à interrupção necessária da execução do Branch and Bound.

O gráfico da Figura 2 apresenta o tempo de execução do algoritmo Twice Around the Tree (TAT) e do algoritmo de Christofides para diferentes tamanhos de instâncias do TSP. Para uma compreensão mais clara e detalhada dos resultados, os eixos do gráfico estão em escala logarítmica.

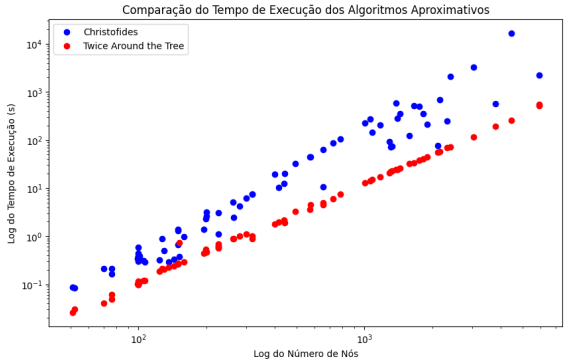

Figure 2. Tempos de execução para cada tamanho de entrada.

Esse gráfico indica que, consistentemente, o algoritmo TAT requer menos tempo para a execução do que o algoritmo de Christofides. Esta observação está alinhada

com a nossa expectativa baseada na análise preliminar, considerando as complexidades computacionais inerentes a cada um dos algoritmos. O algoritmo de Christofides, com uma complexidade teórica mais alta, tende a consumir mais tempo de processamento em comparação com o TAT.

Importante ressaltar que os testes com os algoritmos aproximativos foram realizados apenas até instâncias de tamanho 5934. Neste ponto, o tempo de execução do algoritmo de Christofides já ultrapassava significativamente os 30 minutos, e ambos os algoritmos (TAT e Christofides) demandavam uma quantidade de memória computacional considerada inviável. Essas limitações impediram a continuação dos testes em instâncias maiores. Entretanto ainda seria possível continuar executando o TAT para instâncias maiores a não ser pela limitação de memória.

4.3. Comparação do Uso de Memória

Na análise do consumo de memória pelos algoritmos destinados à resolução do Problema do Caixeiro Viajante, observamos comportamentos distintos entre os métodos aproximativos e o algoritmo de Branch and Bound. O gráfico da Figura 3 apresenta o uso de memória pelos algoritmos aproximativos e mostra que ambos, Christofides e Twice Around the Tree, mantêm um consumo de memória parecido, independentemente do tamanho da instância do problema. Esse padrão era esperado, considerando que ambos os algoritmos armazenam em memória a mesma estrutura fundamental: um grafo da biblioteca NetworkX. Portanto, não há uma diferença significativa no consumo de memória entre esses algoritmos.

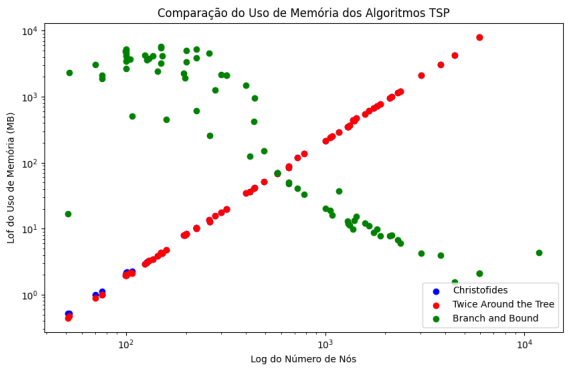

Figure 3. Consumo de memória para cada instância.

Por outro lado, o algoritmo de Branch and Bound apresenta um perfil de uso de memória distinto. Para instâncias menores do problema, o consumo de memória é mais elevado, diminuindo para instâncias maiores. Esta tendência contraintuitiva pode ser explicada pela natureza da árvore de espaço de busca, que é o principal consumidor de memória no Branch and Bound.

Em instâncias menores, a árvore de espaço de busca tem potencial para crescer mais rapidamente, devido à menor complexidade em cada nó. No entanto, à medida que o tamanho da instância aumenta, o tempo necessário para computar a função de bound em cada nó do espaço de busca se torna substancialmente maior. Isso resulta em uma progressão mais lenta através da árvore de espaço de busca e, consequentemente, em um crescimento mais lento do heap, o que limita o uso de memória. Porém, essa limitação

também implica que a solução ótima está longe de ser alcançada dentro do tempo limite estabelecido.

Ademais, também observa-se que já para instâncias pequensa o consumo de memória do branch and bound é elevado, o que deve ser levado em consideração para escolher o algoritmo mais adequado em cada caso.

4.4. Comparação do Erro em Relação à Solução Òtima

Na análise dos algoritmos para o Problema do Caixeiro Viajante, uma consideração importante foi a precisão das soluções geradas em comparação com a solução ótima conhecida. No caso do algoritmo de Branch and Bound, foi necessário interromper a execução antes de atingir a solução ótima em todas as instâncias, pois o tempo de processamento ultrapassava o limite estabelecido de trinta minutos. No entanto, nas instâncias menores, a aplicação do método de priorização otimizado permitiu que o algoritmo alcançasse soluções parciais. Essas soluções parciais foram utilizadas para comparar o desempenho do Branch and Bound com os algoritmos aproximativos.

A análise do gráfico da Figura 4, que compara o erro relativo à solução ótima, revela diferenças significativas entre os algoritmos aproximativos e o Branch and Bound. O algoritmo de Christofides mostrou-se o mais preciso, aproximando-se do valor ótimo com um erro médio em torno de 15%. Em contraste, o algoritmo Twice Around the Tree apresentou um desempenho inferior, com um erro médio aproximado de 40%.

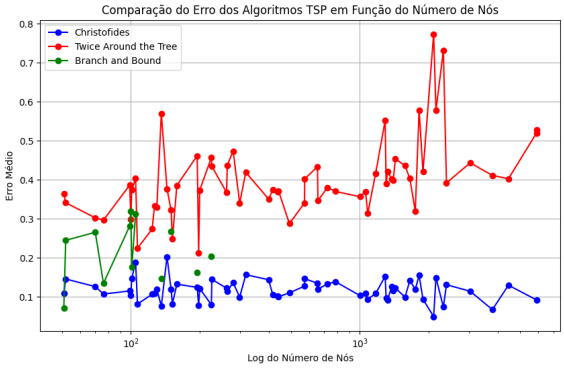

Figure 4. Erro de cada algoritmo em cada instância.

Nas instâncias onde foi possível obter soluções parciais, o algoritmo de Branch and Bound exibiu um erro que se situava entre os dois métodos aproximativos, com um valor médio em torno de 20%. Esta observação é relevante, pois indica que, mesmo não alcançando a solução ótima dentro do tempo estipulado, o Branch and Bound foi capaz de gerar soluções de qualidade razoável.

Esses resultados evidenciam a eficácia do algoritmo de Christofides em termos de precisão, apesar do seu maior tempo de execução. O algoritmo Twice Around the Tree, embora mais rápido, compromete a precisão. O Branch and Bound, com seu desempenho intermediário em termos de erro, destaca-se como uma abordagem viável para instâncias menores, onde é capaz de fornecer soluções parciais úteis dentro do limite de tempo, mas é expressamente inviável para instâncias maiores.

5. Conclusão

Ao longo deste trabalho, foi investigada a aplicação de diferentes algoritmos para resolver o Problema do Caixeiro Viajante (TSP), um desafio clássico em problemas NP-difíceis. A implementação do algoritmo de Branch and Bound, apesar de teoricamente capaz de encontrar a solução exata, revelou-se ineficiente na implementação realizada, não conseguindo alcançar a solução ótima dentro do limite de tempo de 30 minutos para nenhuma das instâncias testadas.

A natureza NP-difícil do TSP implica que algoritmos como o Branch and Bound enfrentam desafios significativos de eficiência. Embora seja possível que uma implementação mais otimizada e uma estimativa de custo melhor balanceada entre precisão e tempo de execução pudessem melhorar o desempenho em instâncias menores, os resultados obtidos destacam as limitações intrínsecas dessa abordagem para o problema.

Por outro lado, os algoritmos aproximativos, Christofides e Twice Around the Tree, demonstraram ser capazes de alcançar soluções aproximadas em um tempo adequado. O algoritmo de Christofides, embora mais lento, consistentemente alcançou uma precisão superior, com um erro médio em torno de 15%. Já o Twice Around the Tree, mais rápido, apresentou um erro médio de aproximadamente 40%.

Neste cenário, a escolha do algoritmo mais apropriado para o TSP depende do contexto específico e das prioridades do problema. Para situações onde uma resposta rápida é mais valorizada do que a precisão absoluta, os algoritmos aproximativos são alternativas viáveis, sendo o Christofides mais adequado para instâncias de tamanho médio, oferecendo um bom equilíbrio entre tempo e precisão. Em contraste, para instâncias maiores, o tempo de execução do Christofides se torna inviável, tornando o Twice Around the Tree a opção preferível.

O algoritmo de Branch and Bound, embora não eficiente nesse estudo, permanece relevante em situações onde a obtenção da solução exata é imprescindível. No entanto, é importante estar ciente de que, mesmo com otimizações, o tempo necessário para alcançar a solução ótima pode ser consideravelmente longo, especialmente para instâncias de grande escala.

6. Referências

[1] Radoslaw Grymin, Szymon Jagiello. Fast Branch and Bound Algorithm for the Travelling Salesman Problem. 15th IFIP International Conference on Computer Information Systems and Industrial Management (CISIM), Sep 2016, Vilnius, Lithuania. pp.206-217, 10.1007/978-3-319-45378-119