Trabalho Prático 2
Soluções para Problemas Difíceis

Carolina Brandão Farinha Baeta[1]

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

carolinabrandao@ufmg.br

Abstract. *This article explores the implementation and evaluation of algorithms to solve the geometric traveling salesman problem, centering on three approaches: one featuring an exact solution using the Branch and Bound algorithm, and two employing approximate solutions—the Twice Around the Tree and Christofides algorithms. The evaluation encompasses three primary metrics: execution time, memory usage, and solution quality.*

Resumo. *Este artigo aborda a implementação e avaliação de algoritmos para a solução do problema do caixeiro viajante geométrico, com foco em três abordagens: uma com solução exata, com o algoritmo Branch and Bound, e duas com soluções aproximadas, os algoritmos Twice Around the Tree e Christofides. A avaliação conta com três métricas principais: tempo de execução, consumo de espaço e qualidade da solução.*

1. Introdução

O trabalho proposto tem como objetivo explorar os desafios práticos associados à implementação de algoritmos para o problema do caixeiro viajante. O caixeiro viajante é um problema NP-difícil, desafiando as fronteiras da computação, já que não se conhece um algoritmo eficiente para resolvê-lo em tempo polinomial. Com aplicações em logística, roteamento e otimização, encontrar a rota mais eficiente que visita cada cidade exatamente uma vez e retorna ao ponto de origem é uma tarefa que requer estratégias algorítmicas avançadas. A natureza NP-difícil do problema impõe a busca por soluções que equilibrem a eficiência computacional com a necessidade de encontrar rotas ótimas, tornando-o um campo fértil para a exploração de técnicas exatas e aproximadas.

Definindo formalmente, o problema consiste em, dada uma coleção finita de pontos no plano cartesiano, cada ponto representando uma cidade, o objetivo é encontrar o menor caminho fechado que visite cada cidade exatamente uma vez e retorne à cidade de origem. A métrica de distância entre as cidades, neste trabalho, foi a distância euclidiana, embora possam ser consideradas outras métricas geométricas. O desafio reside na minimização da soma total das distâncias percorridas, o que se torna computacionalmente difícil à medida que o número de cidades aumenta.

Na seção 2, exploraremos as implementações dos algoritmos propostos, abordando as decisões cruciais relacionadas à representação de dados, escolhas de estruturas algorítmicas e estratégias de otimização. A seção 3 se dedicará à análise e avaliação dessas implementações, considerando métricas cruciais como tempo de execução, consumo de espaço e qualidade das soluções obtidas. Na 4 seção, procederemos à comparação entre as abordagens utilizadas, destacando os pontos fortes e limitações de cada algoritmo. Por fim, na seção 5, apresentaremos as conclusões derivadas dos experimentos.

2. Implementação

2.1. Branch and Bound

A implementação do algoritmo de Branch-and-Bound para resolver o problema utiliza uma classe chamada *Node* para representar cada nó na árvore de busca. Cada instância dessa classe contéma estimativa de custo mínimo , as arestas contribuintes para a estimativa, o custo acumulado, e o caminho parcial até o nó. A estimativa de custo é inicialmente calculada considerando as duas arestas de menor peso conectadas a cada nó. A estratégia visa fornecer uma estimativa inicial razoável para o custo mínimo do subproblema associado ao nó.

O algoritmo de Branch-and-Bound adota uma abordagem best-first, priorizando a exploração de nós promissores com menor estimativa de custo. Isso é implementado através de uma fila de prioridade (heap). A função principal inicia o processo com o nó raiz e itera pela árvore de busca, atualizando a melhor solução encontrada. O loop principal considera diferentes casos com base no nível atual na árvore de busca. Para níveis intermediários, o algoritmo gera novos nós candidatos exploráveis. Para o último nível, o código verifica se uma solução melhor foi encontrada.

A complexidade do algoritmo permanece no mesmo nível teórico que a abordagem de força bruta no pior caso, ou seja, $O(n!)$, o que implica que todas as combinações de vértices são testadas. No entanto, essa implementação destaca-se pela aplicação de otimizações estratégicas, resultando em desempenho significativamente mais eficiente em comparação com a solução tradicional. A introdução de um limite inferior calculado de forma constante para cada nó permite a poda de ramos não promissores na árvore de possibilidades, acelerando o processo de busca. Essa estratégia é aprimorada pelo uso de um heap para ordenar os nós, priorizando aqueles com menor tamanho de solução e um limite inferior mais promissor. Além disso, a implementação incorpora otimizações inteligentes, como a eliminação de caminhos repetidos em sentido contrário e a previsão do resto quando restam apenas 2 vértices, evitando processamentos desnecessários. Essas melhorias permitiram que o algoritmo, embora mantendo a mesma complexidade teórica de força bruta, se destacasse na resolução eficiente de instâncias grandes do Problema do Caixeiro Viajante.

2.2. Twice Around the Tree

O algoritmo "Twice Around the Tree" é uma heurística eficaz para abordar o desafiador Problema do Caixeiro Viajante (TSP). O algoritmo em questão busca fornecer uma solução aproximada para este problema em tempo polinomial. Ele consiste em gerar uma Árvore Geradora Mínima (AGM), uma subestrutura que conecta todos os nós com o menor custo total possível, e serve como a espinha dorsal para a construção do caminho desejado. Por meio de uma busca em profundidade (DFS) na AGM, o algoritmo gera um caminho Hamiltoniano, que percorre cada nó uma vez e retorna ao ponto de partida. Esse caminho é crucial para a determinação da solução aproximada.O peso do caminho Hamiltoniano é então calculado, somando os pesos das arestas ao longo do caminho. Isso proporciona uma medida do custo total da solução encontrada pelo algoritmo.

A complexidade temporal é dominada pela construção da árvore, que é feita utilizando o algoritmo de Kruskal, que tem complexidade assintótica $O(|E|log|V|)$ onde E é o número de arestas e V é o número de vértices no grafo. Já a complexidade de espaço

é linear em relação a entrada. Levitin, em seu livro "Introduction to The Design and Analysis of Algorithms", prova que a solução gerada por esse algoritmo é 2-aproximada.

2.3. Christofides

O algoritmo de Christofides também é uma heurística para resolver o Problema do Caixeiro Viajante (TSP), sendo baseado no Twice Around the Tree, porém com melhorias. Ele começa calculando uma Árvore Geradora Mínima (AGM) usando o algoritmo de Kruskal. Identificando nós com grau ímpar na AGM, o algoritmo encontra um emparelhamento perfeito mínimo nesse subgrafo ímpar. A AGM é expandida adicionando as arestas do emparelhamento, e um caminho euleriano é obtido. Esse caminho é transformado em um circuito hamiltoniano, resultando em uma solução aproximada para o TSP. O emparelhamento mínimo é uma parte crucial do algoritmo, já que minimiza o peso das arestas.

No livro "Introduction to The Design and Analysis of Algorithms", se encontra a prova de que a solução é no máximo 1.5 vezes pior que a ótima, sendo considerada uma heurística de aproximação com uma boa garantia de desempenho. Em relação ao tempo, sua complexidade é dominada pelo emparelhamento, que é feito utilizando o algoritmo de Edmonds, que tem complexidade $O(|V|^2|E|)$. A complexidade de espaço é linear em relação a entrada. Podemos concluir que algoritmo de Christofides oferece uma boa relação entre eficiência e precisão.

3. Análise dos Experimentos

A avaliação dos algoritmos teve como foco primário o não ultrapassar o limite de tempo estabelecido em 30 minutos. Este critério é importante para assegurar que a execução dos algoritmos seja controlada e gerenciável. Depois, os algoritmos seram submetido a uma análise mais abrangente, considerando não apenas o tempo de execução, mas também aspectos como o consumo de memória e a qualidade das soluções obtidas.

Com as 78 instâncias do conjunto de teste provenientes da biblioteca TSPLIB organizadas em ordem crescente do número de nós, foi adotada uma abordagem prática durante a execução dos algoritmos. Caso um algoritmo ultrapassasse o limite de 30 minutos para uma instância com x nós, a avaliação desse algoritmo era interrompida para as instâncias subsequentes. Essa decisão foi tomada considerando a expectativa de que o tempo de execução, uma vez excedido o limite inicial, provavelmente seria ainda mais extenso para instâncias com um número maior de nós.

3.1. Branch and Bound

A menor instância, contendo 51 nós, foi interrompida devido à ultrapassagem do limite de tempo estabelecido, resultando na não execução do algoritmo para qualquer instância do conjunto de teste. A expectativa de um tempo de execução elevado era previsível, devido à abordagem exaustiva desse algoritmo. O Branch and Bound, ao analisar de maneira sistemática as diferentes soluções candidatas, enfrenta um crescimento exponencial na complexidade de tempo à medida que o número de nós na instância do problema aumenta.

Afim de analisar o algoritmo, ele foi testado com 4 instancias, de 10, 15, 20 e 30 nós, no qual o último ultrapassou os 30 minutos. A tabela da Figura 1 mostra os resultados obtidos, que revelam uma tendência clara de um enorme aumento no tempo de execução à medida que o número de nós no problema aumenta.

Numero de Nós	Tempo em segundos	Memória gasta em bytes
10	0.24533987045288086	19614
15	5.657937049865723	34990
20	366.05015206336975	58862

Figure 1. Resultados do Algoritmo Branch and Bound para instâncias pequenas

3.2. Twice Around the Tree

O desempenho do algoritmo Twice Around the Tree foi notável, proporcionando soluções aproximadas em menos de trinta minutos para 73 das 78 instâncias testadas. A instância 74, composta por 11.849 nós, ultrapassou o limite de tempo estabelecido. Em um contraste interessante, a instância anterior a ela, com 5.934 nós, foi resolvida em apenas 158 segundos, equivalente a menos de 3 minutos, e, a instância contendo 2.392 nós foi resolvida em meros 18 segundos, indicando uma relação não linear entre o tamanho da instância e o tempo de execução. Essa análise destaca a eficácia, mas também a sensibilidade do algoritmo à medida que o problema se torna mais complexo. A habilidade de resolver rapidamente instâncias menores e o aumento notável no tempo para instâncias substancialmente maiores ressalta a importância de considerar a escalabilidade do Twice Around the Tree.

O Twice Around the Tree apresentou uma gestão eficiente de recursos de memória, com um gasto que se manteve linear em relação ao tamanho da entrada, ela está intrinsecamente ligada à sua abordagem de construção da Árvore Geradora Mínima (AGM). Essa característica é particularmente significativa, pois destaca a capacidade do algoritmo em lidar com instâncias de grande porte do Problema do Caixeiro Viajante sem comprometer excessivamente os recursos do sistema.

Em relação à qualidade da solução, a métrica adotada consistiu na divisão do peso encontrado pelo algoritmo pelo peso ótimo. A instância que apresentou o resultado mais próximo ao ótimo foi a ”kroB100” com 100 nós, alcançando um índice de 1.168926. Em contraste, a pior performance foi observada na instância ”pr136” com 136 nós, registrando um índice de 1.569811. Essa análise reflete a falta de um padrão consistente na qualidade do algoritmo, uma vez que instâncias tão próximas em termos de número de nós apresentaram resultados extremos, tanto no melhor quanto no pior desempenho. Essa variabilidade ressalta mais uma sensibilidade do Twice Around the Tree, agora às características específicas de cada instância.

3.3. Christofides

O Algoritmo de Christofides foi capaz de gerar soluções para 70 instâncias, sendo a última com 3.795 nós, demandando 1.375 segundos, equivalentes a 22 minutos. Ao analisar o gráfico apresentado na Figura 2, torna-se evidente que o aumento no tempo de execução exibe uma tendência exponencial, principalmente a partir de instâncias com cerca de 2.000 nós. Além disso, é possível observar a sensibilidade do algoritmo às características específicas de cada instância, evidenciada pelo pico de tempo em instâncias próximas a 1.500 nós. A instância com 2.103 nós destacou-se como a solução mais próxima da ótima

pelo Algoritmo de Christofides, alcançando um impressionante índice de 1.053725. Em contrapartida, a pior performance foi observada na instância ”pr144” com um índice de 1.202748.

Essa análise sugere que o desempenho temporal do Algoritmo de Christofides está fortemente relacionado ao tamanho da instância do problema, com um aumento significativo na complexidade computacional para instâncias mais extensas. revela uma variabilidade considerável na qualidade das soluções obtidas pelo Algoritmo de Christofides, e essa variação não parece ter uma relação linear ou consistente com o tamanho da instância do problema. Isso ressalta a importância de uma avaliação completa, levando em consideração tanto o desempenho temporal quanto a qualidade das soluções, ao analisar algoritmos para o Problema do Caixeiro Viajante.

Assim como o Twice Around the Tree, o algoritmo de Christofides mantém uma eficiência notável na utilização de recursos de memória, ambos os algoritmos baseiam sua abordagem na construção de uma Árvore Geradora Mínima (AGM), o que contribui para um gasto de memória comparável entre eles.

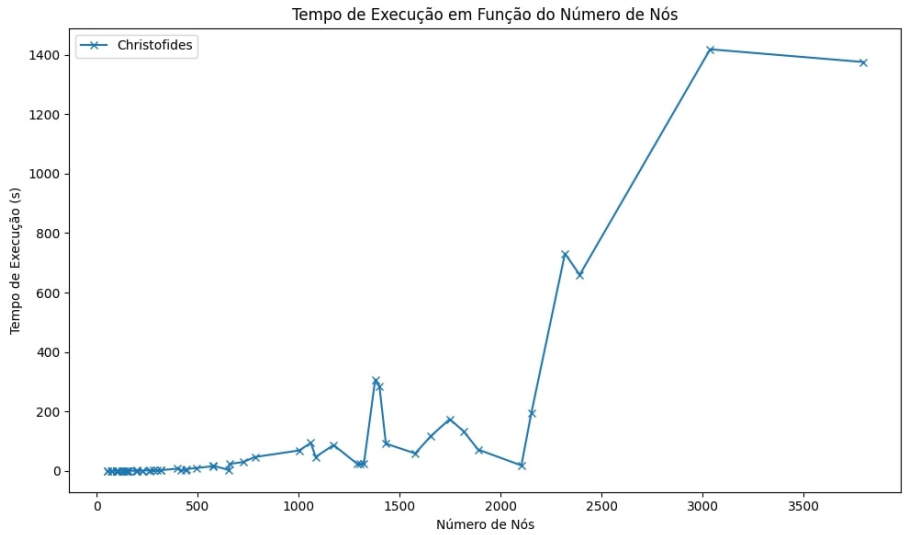

Figure 2. Gráfico do tempo de execução do algoritmo de Christofides

4. Comparação

Dado que o algoritmo Branch and Bound não conseguiu fornecer soluções para as instâncias avaliadas, a comparação será entre os algoritmos Twice Around the Tree e Christofides. Em relação ao tempo de execução, observamos na Figura 3 que o Christofides tende a demorar significativamente mais à medida que o número de nós aumenta. Portanto, é interessante considerar o uso do Twice Around the Tree, especialmente em cenários com um grande número de nós, onde sua eficiência temporal se destaca em comparação com o Christofides.

Em relação à qualidade das soluções, é evidente, na Figura 4, que o Christofides apresenta soluções mais próximas da ótima em comparação com o Twice Around the Tree. O pior desempenho do Christofides ainda se mantém próximo ao melhor resultado obtido pelo Twice Around the Tree, e há apenas uma instância em que a solução do

Christofides é pior que a do Twice Around the Tree, mas ainda assim, são soluções bastante similares. Esse padrão sugere que, embora o Twice Around the Tree ofereça uma abordagem mais eficiente em termos de tempo para instâncias com um grande número de nós, o Christofides destaca-se pela qualidade de suas soluções. A escolha entre esses algoritmos dependerá da prioridade atribuída à precisão da solução versus a eficiência temporal em contextos específicos.

Essas observações ressaltam a importância de escolher o algoritmo mais adequado com base nas características específicas do problema a ser resolvido.

A Tabela 1 exibe, para as instâncias que obtiveram solução, as métricas para cada algoritmo.

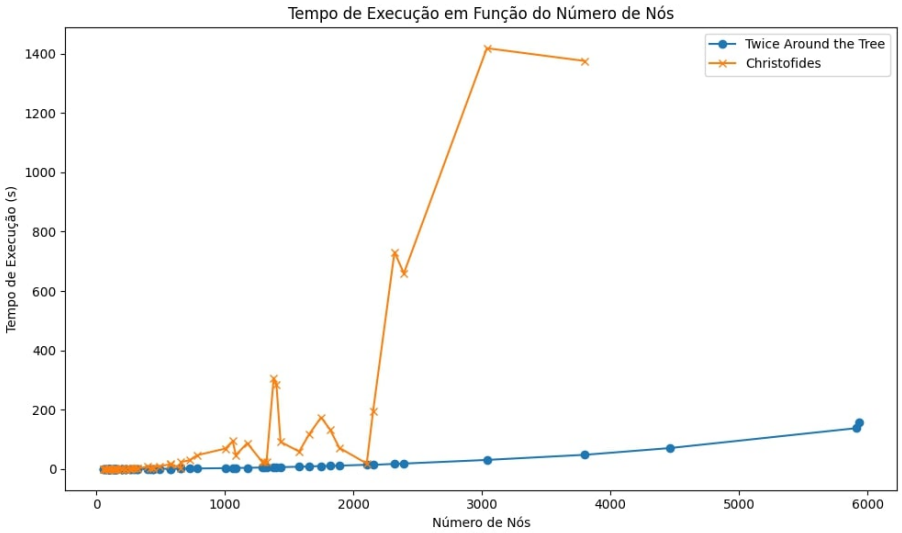

Figure 3. Gráfico do tempo de execução dos dois algoritmos

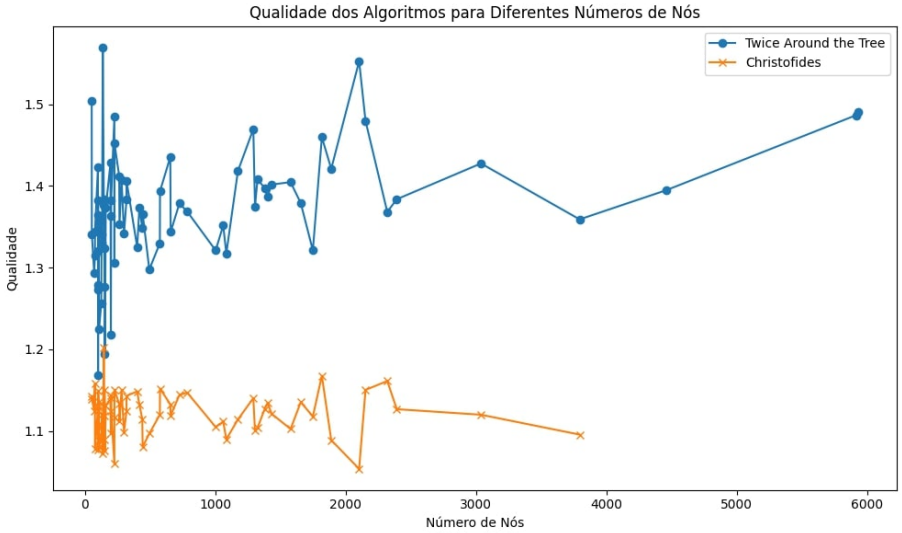

Figure 4. Gráfico da qualidade dos dois algoritmos

5. Conclusão

Em conclusão, esta análise comparativa destaca a complexidade inerente ao Problema do Caixeiro Viajante e a importância de escolher abordagens algorítmicas adaptadas a diferentes contextos. Embora o Twice Around the Tree tenha se destacado pela eficiência temporal em instâncias extensas, sua sensibilidade e variabilidade nas soluções apontam para desafios significativos. O Christofides, por sua vez, ofereceu soluções de alta qualidade, ressaltando a necessidade de um compromisso entre precisão e eficiência temporal. A inabilidade do Branch and Bound em fornecer soluções para as instâncias enfatiza a dificuldade intrínseca do problema. Esta análise evidencia que, mesmo com algoritmos otimizados, encontrar a solução ótima para o Problema do Caixeiro Viajante pode ser um desafio formidável. Assim, a pesquisa e a seleção criteriosa de abordagens algorítmicas continuam sendo aspectos fundamentais na resolução de problemas complexos.

Instância	Algoritmo	Limiar	Resultado	Qualidade	Tempo(s)	Bytes
eil51	TATT	426	640.90	1.50	0.01	123206
eil51	CHRI	426	486.82	1.14	0.02	709664
berlin52	TATT	7542	10116.01	1.34	0.01	34080
berlin52	CHRI	7542	8594.03	1.14	0.02	509720
st70	TATT	675	873.35	1.29	0.01	104968
st70	CHRI	675	759.16	1.12	0.05	1058334
eil76	TATT	538	707.15	1.31	0.01	86544
eil76	CHRI	538	623.06	1.16	0.06	1220584
pr76	TATT	108159	145338.11	1.34	0.01	80296
pr76	CHRI	108159	116683.53	1.08	0.03	982008
rat99	TATT	1211	1723.23	1.42	0.02	117208
rat99	CHRI	1211	1367.43	1.13	0.07	1991536
kroA100	TATT	21282	27211.68	1.28	0.04	144688
kroA100	CHRI	21282	23292.98	1.09	0.16	2188550
kroB100	TATT	22141	25881.20	1.17	0.02	140736
kroB100	CHRI	22141	24009.19	1.08	0.07	2016136
kroC100	TATT	20749	27966.54	1.35	0.02	145848
kroC100	CHRI	20749	23469.13	1.13	0.09	2137968
kroD100	TATT	21294	27113.30	1.27	0.02	142824
kroD100	CHRI	21294	23587.75	1.11	0.08	2063920
kroE100	TATT	22068	30507.42	1.38	0.02	142752
kroE100	CHRI	22068	23782.61	1.08	0.11	2222134
rd100	TATT	7910	10791.66	1.36	0.02	145216
rd100	CHRI	7910	8905.49	1.13	0.13	2235384
eil101	TATT	629	830.54	1.32	0.02	148696
eil101	CHRI	629	723.00	1.15	0.11	2317944
lin105	TATT	14379	19498.41	1.36	0.03	130424
lin105	CHRI	14379	16323.68	1.14	0.08	2187248
pr107	TATT	44303	54238.04	1.22	0.03	149944
pr107	CHRI	44303	47894.18	1.08	0.13	2406142
pr124	TATT	59030	74140.96	1.26	0.04	164104
pr124	CHRI	59030	64437.70	1.09	0.06	2542776
bier127	TATT	118282	158637.61	1.34	0.04	198424
bier127	CHRI	118282	132456.25	1.12	0.21	3427390
ch130	TATT	6110	8128.76	1.33	0.04	181688
ch130	CHRI	6110	6752.28	1.11	0.13	2748440
pr136	TATT	96772	151913.74	1.57	0.05	176840
pr136	CHRI	96772	103771.91	1.07	0.08	3167752
pr144	TATT	58537	80596.32	1.38	0.05	174384
pr144	CHRI	58537	70405.28	1.20	0.07	3223896
ch150	TATT	6528	8333.47	1.28	0.05	194128
ch150	CHRI	6528	7112.04	1.09	0.27	3890448
kroA150	TATT	26524	35122.57	1.32	0.05	205672
kroA150	CHRI	26524	29688.32	1.12	0.28	4207334
kroB150	TATT	26130	36154.73	1.38	0.06	205600

Table 1 – Continuação

Instância	Algoritmo	Limiar	Resultado	Qualidade	Tempo(s)	Bytes
kroB150	CHRI	26130	30054.39	1.15	0.61	4252886
pr152	TATT	73682	87998.69	1.19	0.05	188912
pr152	CHRI	73682	79314.35	1.08	0.07	3524912
u159	TATT	42080	57787.97	1.37	0.06	190024
u159	CHRI	42080	47581.72	1.13	0.35	4490382
rat195	TATT	2323	3317.72	1.43	0.08	220048
rat195	CHRI	2323	2648.10	1.14	0.32	6289136
d198	TATT	15780	19218.43	1.22	0.09	262608
d198	CHRI	15780	17316.37	1.10	0.70	6591744
kroA200	TATT	29368	40030.87	1.36	0.09	263384
kroA200	CHRI	29368	33602.86	1.14	1.12	7983870
kroB200	TATT	29437	40710.96	1.38	0.10	263000
kroB200	CHRI	29437	33118.12	1.13	0.84	7625582
ts225	TATT	126643	188009.70	1.48	0.12	274608
ts225	CHRI	126643	134282.17	1.06	0.16	8268584
tsp225	TATT	3919	5115.93	1.31	0.12	275608
tsp225	CHRI	3919	4376.00	1.12	0.81	8877246
pr226	TATT	80369	116692.15	1.45	0.12	276568
pr226	CHRI	80369	92425.75	1.15	0.65	9383022
gil262	TATT	2378	3358.15	1.41	0.15	284600
gil262	CHRI	2378	2695.15	1.13	1.82	12276070
pr264	TATT	49135	66466.84	1.35	0.18	294200
pr264	CHRI	49135	54663.60	1.11	0.65	10412216
a280	TATT	2579	3629.72	1.41	0.21	278760
a280	CHRI	2579	2966.28	1.15	1.44	13528310
pr299	TATT	48191	64646.03	1.34	0.23	280984
pr299	CHRI	48191	52968.22	1.10	1.98	14985720
lin318	TATT	42029	58145.44	1.38	0.27	303592
lin318	CHRI	42029	47259.76	1.12	2.52	17009286
linhp318	TATT	41345	58145.44	1.41	0.28	303592
linhp318	CHRI	41345	47259.76	1.14	2.53	17009286
rd400	TATT	15281	20250.38	1.33	0.42	377528
rd400	CHRI	15281	17551.75	1.15	7.63	30309686
fl417	TATT	11861	16297.17	1.37	0.47	387160
fl417	CHRI	11861	13429.64	1.13	2.98	29515824
pr439	TATT	107217	144624.16	1.35	0.51	404128
pr439	CHRI	107217	119532.04	1.11	3.17	31912104
pcb442	TATT	50778	69364.90	1.37	0.52	398120
pcb442	CHRI	50778	54876.77	1.08	5.93	35159710
d493	TATT	35002	45427.09	1.30	0.69	426616
d493	CHRI	35002	38393.64	1.10	9.75	41568646
u574	TATT	36905	49083.15	1.33	0.93	466512
u574	CHRI	36905	41337.61	1.12	15.63	54511758
rat575	TATT	6773	9438.93	1.39	0.94	451600
rat575	CHRI	6773	7802.75	1.15	17.84	55750806

Table 1 – Continuação

Instância	Algoritmo	Limiar	Resultado	Qualidade	Tempo(s)	Bytes
p654	TATT	34643	49731.90	1.44	1.23	499608
p654	CHRI	34643	39244.71	1.13	2.84	60313912
d657	TATT	48912	65730.19	1.34	1.29	503504
d657	CHRI	48912	54738.30	1.12	23.54	68573942
u724	TATT	41910	57779.67	1.38	1.54	557256
u724	CHRI	41910	47954.68	1.14	29.52	95666006
rat783	TATT	8806	12054.45	1.37	1.81	596128
rat783	CHRI	8806	10102.87	1.15	46.69	114428022
pr1002	TATT	259045	342244.46	1.32	2.91	688728
pr1002	CHRI	259045	286233.10	1.10	68.70	173886110
u1060	TATT	224094	302914.87	1.35	3.29	738984
u1060	CHRI	224094	249108.58	1.11	94.12	188041750
vm1084	TATT	239297	315268.35	1.32	3.68	750416
vm1084	CHRI	239297	260833.06	1.09	47.03	184252278
pcb1173	TATT	56892	80694.89	1.42	4.02	798752
pcb1173	CHRI	56892	63392.44	1.11	87.07	215393102
d1291	TATT	50801	74658.30	1.47	5.05	849080
d1291	CHRI	50801	57930.37	1.14	23.09	223378200
rl1304	TATT	252948	347782.57	1.37	5.27	860912
rl1304	CHRI	252948	278319.19	1.10	22.95	236985054
rl1323	TATT	270199	380421.59	1.41	5.51	873664
rl1323	CHRI	270199	298261.72	1.10	23.65	243283998
nrw1379	TATT	56638	79156.27	1.40	5.92	966592
nrw1379	CHRI	56638	63845.07	1.13	307.03	357901814
fl1400	TATT	20127	27915.52	1.39	5.87	956144
fl1400	CHRI	20127	22834.35	1.13	284.79	390370478
u1432	TATT	152970	214430.78	1.40	6.33	990592
u1432	CHRI	152970	171495.23	1.12	91.68	365354934
fl1577	TATT	22226.5	31223.10	1.40	7.65	1074328
fl1577	CHRI	22226.5	24505.55	1.10	58.77	404255470
d1655	TATT	62128	85681.08	1.38	8.62	1034424
d1655	CHRI	62128	70542.34	1.14	117.26	449945958
vm1748	TATT	336556	444658.63	1.32	10.11	1176960
vm1748	CHRI	336556	376240.90	1.12	174.22	493294598
u1817	TATT	57201	83501.56	1.46	10.35	1177320
u1817	CHRI	57201	66796.26	1.17	133.20	525434526
rl1889	TATT	316536	449811.49	1.42	11.41	1222672
rl1889	CHRI	316536	344691.95	1.09	71.65	538068390
d2103	TATT	80201.0	124533.87	1.55	14.31	1317392
d2103	CHRI	80201.0	84509.81	1.05	18.37	622765318
u2152	TATT	64253	95076.22	1.48	14.09	1338936
u2152	CHRI	64253	73917.83	1.15	193.13	702070990
u2319	TATT	234256	320532.84	1.37	17.28	1417176
u2319	CHRI	234256	272035.61	1.16	731.07	972415262
pr2392	TATT	378032	523132.91	1.38	18.41	1463936

Table 1 – Continuação

Instância	Algoritmo	Limiar	Resultado	Qualidade	Tempo(s)	Bytes
pr2392	CHRI	378032	425979.59	1.13	659.54	919999766
pcb3038	TATT	137694	196573.90	1.43	30.63	1925792
pcb3038	CHRI	137694	154218.19	1.12	1418.17	1590859302
fl3795	TATT	28747.5	39072.83	1.36	47.79	2344496
fl3795	CHRI	28747.5	31498.97	1.10	1375.50	2300591318
fnl4461	TATT	182566	254671.63	1.39	70.55	2605160
rl5915	TATT	565285.0	840348.36	1.49	137.76	3632104
rl5934	TATT	555057.5	827482.34	1.49	158.23	5417719137

Table 1: Comparação entre Twice Around the Three e Christofides

References

Levitin, A. (2011). Introduction to the design & analysis of algorithms. Addison-Wesley, 3rd edition.

Renato Vimieiro - Slides da disciplina de Algoritmos 2.