Solução geométrica plana ao problema do caixeiro viajante utilizando divisão e conquista, curva de Hilbert e *branch and bound*

Sarah Klock Mauricio
Instituto Federal do Paraná
Cascavel, Paraná, Brasil
sarah.mauricio.academico@gmail.com

André Augusto Bortoli
Instituto Federal do Paraná
Cascavel, Paraná, Brasil
andreaugustobortoli@gmail.com

Darlon Vasata
Instituto Federal do Paraná
Cascavel, Paraná, Brasil
darlon.vasata@ifpr.edu.br

***Abstract* — The travelling salesman problem is a classic combinatorial analysis issue which consists in finding the route of minimum cost with the intention of coursing all the points in a space, which can be applied in many contexts in society. However, its solution by brute force doesn't have practical time of response, making it unassessable in various situations. That settled, there were created numerous ways of solving this problem with lower time of response, and this article has as purpose to suggest an approach that unites three of them: divide to conquer — through the application of the K-Means algorithm implemented in the scikit-learn library —, branch and bound, and Hilbert's space filling curve. The algorithm described in this work was developed using the Python programming language and its solution is a hamiltonian path, which refers to the geometric, plane and complete formation of the travelling salesman problem .**

***Key-words* — travelling salesman problem; divide to conquer; K-Means; branch and bound; Hilbert's space filling curve.**

***Resumo* — O problema do caixeiro viajante é uma questão de análise combinatória clássica que consiste em encontrar a rota de menor custo com fim de percorrer determinados pontos em um espaço, podendo ser aplicado em diversos contextos na sociedade. Contudo, sua solução por força bruta possui tempo de resposta incabível, tornando-a inacessível em diversas situações. Sendo assim, foram criadas ao longo do tempo diversas maneiras de resolver o problema do caixeiro viajante com tempo de resposta menor, e este artigo tem por objetivo propor uma abordagem que une três dessas maneiras: divisão e conquista — pela aplicação do algoritmo K-Means implementado na biblioteca scikit-learn —, *branch and bound* e curva de preenchimento de espaço de Hilbert. O algoritmo descrito neste trabalho foi desenvolvido utilizando a linguagem de programação Python e sua solução trata-se de um caminho hamiltoniano, referente a formação geométrica, plana e completa do problema do caixeiro viajante.**

***Palavras-chave* — problema do caixeiro viajante; divisão e conquista; K-Means; *branch and bound*; curva de preenchimento de espaço de Hilbert.**

I. Introdução

O problema do caixeiro viajante (PCV) consiste em encontrar o caminho de menor custo que um caixeiro poderia ter ao percorrer exatamente uma vez cada cidade de um determinado grupo de cidades. Tratando-se de uma questão de otimização de rotas, diversos contextos práticos poderiam ser acelerados por uma solução eficiente ao PCV, como fabricação de placas de circuito impresso [1], sequenciamento de genoma [2], roteamento de cabos [3], determinação rotas para empresas de transporte [4], entre outros [5] [6]. Dentre as possíveis formações do PCV, a abordada neste artigo é geométrica plana — todos os pontos a serem percorridos estão em um mesmo plano e os custos das arestas equivalem à distância euclidiana; simétrica — ir e voltar em uma aresta possuem o mesmo custo; completa — todos os pontos pode se conectar entre si; e cuja solução é um caminho hamiltoniano, ilustrado na Figura 1 [7].

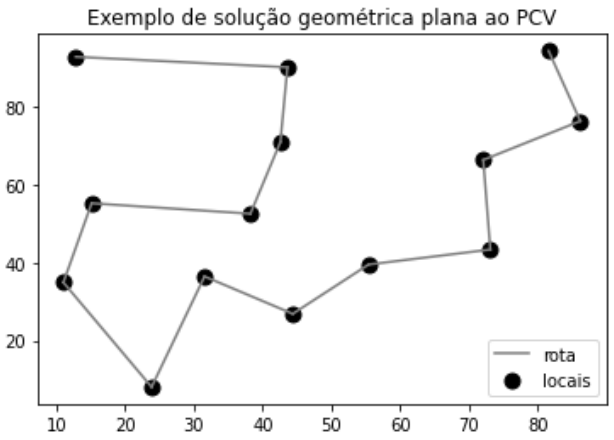

Fig. 1. Exemplo de malha contendo 14 pontos percorrida com uma rota solução euclidiana ao problema do caixeiro viajante (PCV).

Questões como o PCV são objetos de estudo recorrentes na área da complexidade computacional por possuírem enunciado conciso, mas de difícil resolução. Dentro dessa área de estudo, um problema polinomial (*P*) é aquele que pode ter seu tempo de execução descrito por um polinômio [8]; já um polinomial não determinístico (do ing. *non deterministic polynomial*, *NP*) pode ter a validade de sua resposta checada em um tempo que pode ser descrito por

um polinômio, mas não necessariamente é possível a escrita de um algoritmo solução que opere em tempo polinomial [9]. A pergunta $P = NP$ tem relação com esses conceitos, pois todo problema P é NP, mas ainda não é certo se todo dilema NP é P [10] [11].

O PCV foi provado NP-difícil por Richard Karp em 1972 e, sendo essa questão de classe NP que não possui solução com resposta e tempo de execução ótimos, estudos a respeito de aproximações à sua solução são pertinentes para que seja possível formar conclusões a respeito da possibilidade de resolvê-lo de maneira eficiente [6].

A. *Solução via força bruta*

A abordagem de força bruta, em que são computadas todas as rotas possíveis a fim de encontrar a menor, não é eficiente. Analisando-a, nota-se que, dado um grafo completo de n vértices como entrada, é possível usar o princípio multiplicativo para inferir a equação do número total de rotas possíveis, e, logo, de iterações realizadas pelo método de força bruta i_f:

$$i_f = n \times (n-1) \times \cdots \times 2 \times 1 = n! \qquad (1)$$

Sendo essa a definição de fatorial, este é um algoritmo de complexidade fatorial. Considerando que, para determinar a eficiência de um algoritmo, analisa-se quantas iterações, em função da quantidade de elementos na entrada, serão computadas caso a função seja completamente executada, o número de iterações computadas nesse cálculo cresce de maneira extremamente acelerada a cada adição de elemento a ser percorrido, tornando-o ineficiente [12][13].

Portanto, reconhecendo a importância desse problema e as limitações de sua solução via força bruta, cabe desenvolver outras maneiras de chegar à sua solução, ainda que isso implique em limitar sua precisão. A abordagem descrita no presente trabalho utiliza a linguagem de programação de código livre Python, e trata-se de um algoritmo Solução utilizando Divisão e conquista, Curva de Hilbert e *Branch and bound* (SDCB), com especificidade para casos geométricos, planos e completos.

II. Trabalhos relacionados

Sendo o PCV um problema clássico da análise combinatória, são diversos os trabalhos e as maneiras pensadas para resolvê-lo. O estudo realizado em [14] emprega o método de *branch and bound* (B&B) como alternativa à força bruta para solucionar diversos problemas matemáticos, incluindo o PCV, visto ser mais eficiente que esta devido a não realizar todas as comparações possíveis, mas ainda garantir o resultado ótimo. Em [15], são exploradas as ferramentas de agrupamento K-Means e modelo de mistura gaussiano para solução do PCV com abordagem de divisão e conquista e algoritmos genéticos, e sugere uma maneira de conectar os agrupamentos utilizando a proximidade dos centroides. A aplicação da curva de preenchimento de espaço de Hilbert, presente em [16], é uma solução ao PCV aplicável a casos de pequena e larga escala, sendo extremamente rápida e simples.

Os trabalhos apresentados aplicam de maneira individual algum dos métodos de *branch and bound*, divisão e conquista ou curva de preenchimento de espaço de Hilbert, a fim de prover uma resolução ao PCV; contudo, diferem do presente estudo pelo fato deste os abordar em conjunto para uma única solução.

TABELA I
MÉTODOS ABORDADOS EM CADA TRABALHO

Método	[14]	[15]	[16]	Atual
B&B	x			x
Curva de Hilbert			x	x
Divisão e conquista		x		x
Algoritmo genético		x		

III. Materiais e métodos

Conforme descrito na seção anterior, existem diversas maneiras de otimizar a solução ao PCV, cada uma com suas particularidades. A maneira descrita no presente trabalho utiliza-se de três conceitos: divisão e conquista, curva de preenchimento de espaço de Hilbert e *branch and bound*, detalhados a seguir, juntamente dos seus complementos.

A. *Divisão e conquista: K-Means*

Uma medida capaz de contornar o problema da grande quantidade de caminhos possíveis é a divisão e conquista. Ao distribuir em grupos os pontos a serem percorridos e aplicar uma função que retorne a menor rota internamente nos aglomerados, ao invés de no todo, o tempo de execução é significativamente reduzido, apesar de parte da acurácia ser perdida. Nesse contexto, faz-se necessária a seleção de uma maneira de agrupar pontos; apesar de existirem diversos outros métodos, o algoritmo de agrupamento K-Means foi escolhido devido a sua velocidade de execução. Ele opera a partir de diversas iterações nas quais os pontos de um conjunto são divididos de acordo com sua proximidade geométrica, tendo como parâmetros um conjunto de pontos e o número de grupos em que os mesmos serão divididos [15] [17]. Sua versão implementada na biblioteca *scikit-learn* possui um número máximo de iterações, portanto não há preocupação quanto ao seu pior caso original de execução, que leva seu tempo consumido a casos extremos, no entanto a impossibilidade de definir previamente o número de pontos por grupo abre um novo pior caso, onde diversas re-execuções podem ocorrer [18]. A aplicação do K-Means está exemplificada e ilustrada na Figura 2.

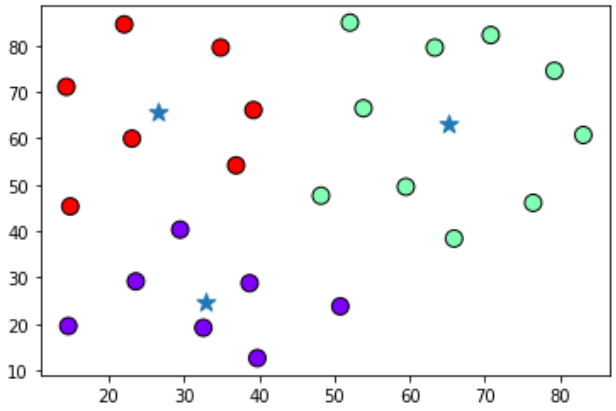

Fig. 2. Exemplo da aplicação do K-Means em uma malha, que dividiu 24 pontos (círculos) em 3 grupos e determinou seus centroides (estrelas).

B. *Branch and Bound*

É possível otimizar a função de força bruta com um algoritmo de *branch and bound* (B&B), a fim de minimizar o tempo de processamento, mantendo o resultado ótimo. A diferença está na operação de formação das rotas a serem computadas: caso os pesos sejam somados sob demanda, pode-se guardar em memória o menor peso já calculado e interromper a computação de qualquer caminho que ultrapasse esse valor [13].

É possível estimar o tempo máximo a ser consumido na execução do B&B usando resultados prévios e a complexidade da força bruta, explicada na subseção A da Introdução. No entanto, devido à impossibilidade de determinar previamente quantos ciclos de execução serão poupados, o tempo de execução esperado pode diferir aleatoriamente em relação ao obtido. Por esse motivo, foram registrados na Tabela II os cálculos da média de tempo de execução em função do número de vértices computados, até o limite de 12 vértices, pois nesse ponto fica notável a diferença de recursos computacionais utilizados. Vale evidenciar que os resultados dos testes executados não são universais, mas relativos especificamente à máquina utilizada para sua realização, devido à diferença de processamento em cada aparelho. Entretanto, ainda têm utilidade para o fim de comparar o desempenho dos métodos mencionados e direcionar a capacidade máxima dos grupos.

Observando a Tabela II, nota-se que, até 5 vértices, a força bruta tem retorno mais rápido, devido que, na B&B, o tempo gasto na otimização é maior que o tempo computado pela força bruta. Porém, a partir de 6 vértices, os tempos médios de execução da função otimizada tornam-se, de maneira contínua e expressa, menores que o tempo de execução da função via força bruta.

TABELA II
TEMPO MÉDIO DE EXECUÇÃO EM SEGUNDOS

N° de vértices	B&B	Força bruta
4	0.0005	0.0002
5	0.0018	0.0012
6	0.0060	0.0086
7	0.0226	0.0685
8	0.0705	0.6289
9	0.2724	6.4809
10	0.7478	72.5371
11	3.2799	891.0082
12	10.2365	9484.5485

Visto que o objetivo é diminuir o tempo de execução, convém determinar o intervalo ideal de números de elementos por grupo, em que o B&B será aplicado internamente. Esse intervalo, como exemplificado na Figura 2, foi determinado empiricamente como valores próximos a 8, com tolerância a casos máximos de 10 pontos, considerando que em casos menores que 8 o tempo máximo — semelhante ao da força bruta — está na ordem de segundos, e a partir de 10 passam a ser minutos (Tabela II). Cabe considerar que valores demasiadamente altos implicam na inviabilidade da aplicação do B&B, posta sua natureza fatorial, e números baixos representam a minimização da precisão.

Uma vez que o K-Means não permite determinar a quantidade de elementos por agrupamento, apenas quantos grupos serão gerados, para evitar que o número de componentes por aglomerado ultrapasse o limite de 10, é imperioso verificar após sua execução se há algum grupo nessa condição e, havendo, aumentar o número de grupos para extingui-lo.

Mas, antes disso, é preciso definir um número inicial mínimo de grupos (g) que os direcione a ter em torno de 8 elementos. Este valor é calculado considerando quantas aglomerações são necessárias para que cada uma consiga comportar aproximadamente 8 pontos, de acordo com a quantidade total de vértices (n).

Entretanto, considerando a possibilidade de n não ser múltiplo de 8, g não será inteiro. Nesse cenário, a função teto é aplicada para que g assuma o menor número inteiro maior que ou igual a si, como representado em (2). Visando antepor tempo de execução à acuracidade, essa opção é preferível em relação à função piso — que tornaria g o maior número inteiro menor que ele —, pois, quanto mais

grupos, menores tendem a ser suas quantidades de pontos internos, e menor será o tempo de execução do algoritmo.

$$g = \lceil n \div 8 \rceil \quad (2)$$

O número máximo de iterações realizadas nas re-execuções do K-Means (i_k) a fim de evitar grupos com mais de 10 pontos é equivalente à diferença entre o número máximo de grupos que podem ser gerados e o valor inicial definido para o número de grupos. Esse número máximo de grupos se dá no caso em que a maioria dos pontos são os únicos vértices em seu grupo, exceto por 10, que estão no mesmo aglomerado. Portanto, esse valor limite pode ser descrito por:

$$i_k = n - 10 + 1 \Rightarrow r = n - 9 \quad (3)$$

C. *Grafos e malhas*

Considerando o aspecto geométrico presente no K-Means e, consequentemente, no SDCB, o formato de entrada mais adequado para o mesmo é o de malhas. Apesar dos grafos serem estruturas úteis para representação de dados dentro de um algoritmo, eles se limitam às relações entre seus vértices e arestas — os elementos nele contidos e as ligações entre eles, respectivamente. São eles os geralmente considerados para o cálculo de força bruta, visto que ela não se retém a casos euclidianos [13].

Por outro lado, as malhas possuem, além das características de um grafo, as posições cartesianas de cada ponto, o que as leva a manter o aspecto geométrico da entrada [19]. Visto que dois dos métodos empregados no SDCB — curva de Hilbert e K-Means — baseiam sua solução em dados geométricos [15] [16], malhas são adequadas para assumirem o papel de entrada do algoritmo.

D. *Curva de preenchimento de espaço de Hilbert*

Apesar de implementar *branch and bound* internamente nos agrupamentos, foi eleita outra maneira de definir a ordem de passagem entre os aglomerados, visto que em casos de larga escala haveria muitos grupos e retornaria-se ao problema da natureza fatorial presente na solução via força bruta ou B&B. Para esse propósito, a escolha foi a aplicação da curva de Hilbert.

A curva de preenchimento de espaço de Hilbert (livre trad. do ing. *Hilbert space filling curve*, HSFC) foi proposta no século XIX pelo matemático David Hilbert e é o resultado da conexão de infinitas divisões de uma malha quadrada [20]. As pseudo-curvas de Hilbert, exemplificadas na Figura 3, são recortes finitos dessas conexões computados em uma ordem m. A ordem de uma pseudo-curva de Hilbert é o resultado da conexão de quatro cópias da ordem anterior dispostas em um quadrado, sendo os quadrantes — um quarto da pseudo-curva analisada — superiores cópias exatas e os inferiores cópias rotacionadas em 90 e 270 graus, respectivamente.

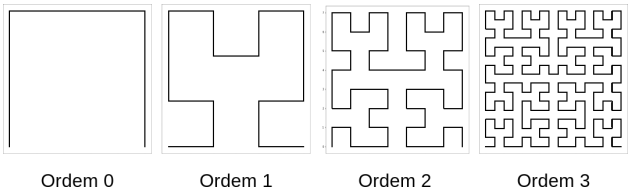

Fig. 3. Demonstração visual de diferentes ordens de pseudo-curvas de Hilbert.

Mapeando os elementos a partir desse fractal [21], nota-se que pontos próximos na extensão da curva de Hilbert tendem a também ter posições próximas no plano cartesiano [4]. Diante dessa propriedade, foi adotada como estratégia para sequenciamento dos grupos a projeção de uma pseudo-curva de Hilbert sobre a malha analisada, com objetivo de minimizar o uso de abordagens ineficientes.

O algoritmo iterativo para geração das pseudo-curvas de Hilbert descrito em [22] aproveita-se da auto-semelhança presente na curva de Hilbert para gerar a pseudo-curva. O processo consiste em aplicar continuamente cópias da ordem 0, rotacionando-a de acordo com o quadrante em que ela será anexada. Para determinar qual a rotação aplicada, considera-se recursivamente a posição do quadrante, em outras palavras, calcula-se a rotação aplicada em um quadrante, baseando-se no processo de formação (90° para o 1°; 0° para o 2° e o 3°; 270° para o 4°), e repete-se o processo, considerando o quadrante calculado ao invés de toda pseudo-curva. Sabendo que cada quadrante será uma ordem imediatamente anterior à analisada, esse processo será repetido m vezes, em que m corresponde à ordem na pseudo-curva.

Uma função análoga a esta foi aplicada no SDCB para ordenar os grupos a partir da sua posição na pseudo-curva de Hilbert, porém com algumas adaptações, visto que o contexto não exige a formação da curva completa, mas apenas a sequência em que ela percorre os quadrantes contendo os centróides. Primeiro, a malha é quadriculada de modo que cada centróide esteja dentro de um quadrado diferente; a ordem m da pseudo-curva é definida em função do número de quadrados formados. Então, ocorre uma iteração em cada centróide para verificar em que quadrado ele se encontra, e em que quadrante de cada ordem ele está: sabendo o quadrante, é possível saber quantos quadrantes seriam percorridos antes pela curva, sem necessariamente saber a sequência em que isso acontece. A única determinação necessária para isso é saber a rotação de cada ordem — da maior para a menor — em que o quadrado em questão está, ou seja, se o quadrante do centróide é o 1°, 2°, 3° ou 4°, partindo dos quadrantes da ordem m (malha dividida em 4), para os da ordem $m - 1$ (malha divida em 16), para os da ordem $m - 2$ (malha dividida em 64), e assim por diante até chegar na ordem 0. Essa etapa se repete por $m + 1$ vezes. Ao obter o número de quadrantes e, por fim, quadrados percorridos anteriormente pela curva

relativos a cada centróide, basta ordenar estes do menor para o maior para descobrir a sequência final dos grupos.

Dado o método de formação da pseudo-curva, o número de quadrados na malha (q) está relacionado com a ordem m por uma progressão geométrica, já que o número de quadrados é multiplicado por 4 a cada nova ordem. Para cada grupo g, como discorrido anteriormente, serão computadas $m + 1$ operações para definir sua posição. Isto posto, define-se que $q=4^{m+1} \Rightarrow log_4 q = m + 1 \Rightarrow m = log_4 q - 1$. O número de iterações realizadas i_h será $g \times (m + 1)$; utilizando a decomposição descrita de m por q, é possível afirmar que o número de iterações i_h realizadas no algoritmo baseado na curva de Hilbert, em função do número de grupos g e de quadrados $q$em que a malha foi dividida é:

$$i_h = g \times (log_4 q - 1 + 1) \Rightarrow i_h = g \times log_4 q \quad (4)$$

E. Pontos de conexão

Tendo estabelecida a sucessão dos grupos, o próximo passo é designar os pontos de conexão — iniciais e finais — entre grupos consecutivos, a fim de otimizar a passagem pelos mesmos. Em outras palavras, encontrar os pontos em que a rota sairá de um grupo e entrará no outro de modo que essa conexão seja a menor possível, como ilustrado na Figura 7.

A função mais intuitiva e precisa para determinar os pontos de conexão certamente seria a que utiliza o cálculo da força bruta, comparando todos os pontos de um grupo com os do grupo seguinte, a fim de encontrar os com menor distância entre si. Contudo, esta função ocupa significativamente parte do tempo de execução e processamento, visto que, definindo P como o conjunto contendo os números de pontos por grupo, e g como a quantidade de grupos, o número de comparações $c$realizadas definindo pontos de conexão para cada par sucessivo de grupos e a quantidade total i_p dessas comparações serão:

$$c = P_k \times P_{k+1} \quad (5)$$

$$i_b = \sum_{k=0}^{g-1} (P_k \times P_{k+1}) \quad (6)$$

Em que $0 \leq k < g, k \in \mathbb{N}$. Sendo 10 o valor máximo de pontos por agrupamento, o maior número de comparações por cada par sucessivo de grupos seria $10 \times 10 = 100$.

Portanto, a solução empregada utiliza outra lógica, baseada nos centroides a fim prevenir a delonga do tempo de processamento. Tendo em mente que os centróides representam seus grupos de maneira centralizada, é justificado considerá-los no momento de definir os pontos de conexão entre estes. Preliminarmente, compara-se a distância entre o centróide de um grupo e todos os pontos do seguinte, encontrando o ponto com o menor afastamento. Então, esse ponto encontrado é comparado com todos os pontos do grupo anterior para determinar o mais próximo a si. Dessa forma:

$$c = P_k + P_{k+1} \quad (7)$$

$$i_p = \sum_{k=0}^{g-1} (P_k + P_{k+1}) \quad (8)$$

Novamente sendo 10 o pior caso de vértices por grupo, o número máximo de comparações por par sucessivo de grupos seria $10 + 10 = 20$, cinco vezes menor que o resultado da função que emprega força bruta.

Em [15], foi proposta para conectar os agrupamentos uma função que toma como os dois pontos de conexão os vértices mais próximos aos centróides do grupo anterior/sucessor. Contudo, comparando-a com a função empregada, nota-se que sua precisão tende a ser menor, visto que não considera as distâncias entre os pontos a serem conectados. Atentando-se a isso e ao fato de o número de comparações de ambas ser o mesmo — representado por (7) e (8) —, a função adotada foi a que considera apenas um centróide na decisão, descrita nos dois parágrafos precedentes.

Por fim, aplica-se o algoritmo de *branch and bound* internamente aos grupos determinados pelo K-Means, restringindo os seus respectivos pontos iniciais e finais. As exceções são o primeiro e o último grupo da sequência, que não possuem restrição de ponto inicial e final, nessa ordem. As rotas internas de cada grupo encadeadas pelos pontos de conexão resultarão na ordem final: um caminho hamiltoniano solução para o problema do caixeiro viajante de formação geométrica, plana e completa. O processo descrito é referente ao algoritmo do SDCB, representado no fluxograma da Figura 4, e está ilustrado, conforme a sequência de execução, nas Figuras de 5 a 8.

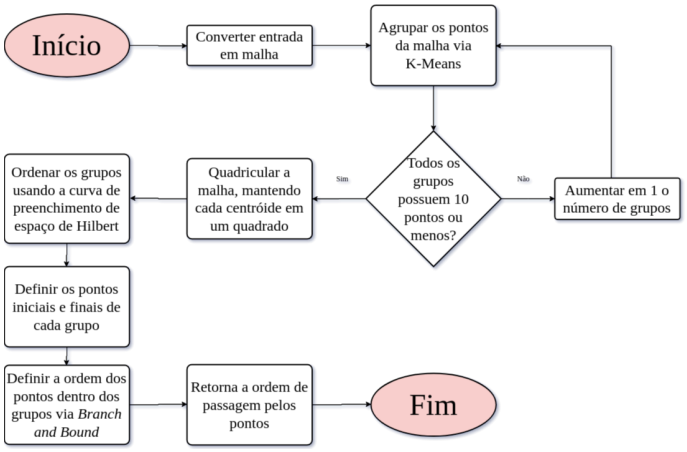

Fig. 4. Fluxograma do SDCB.

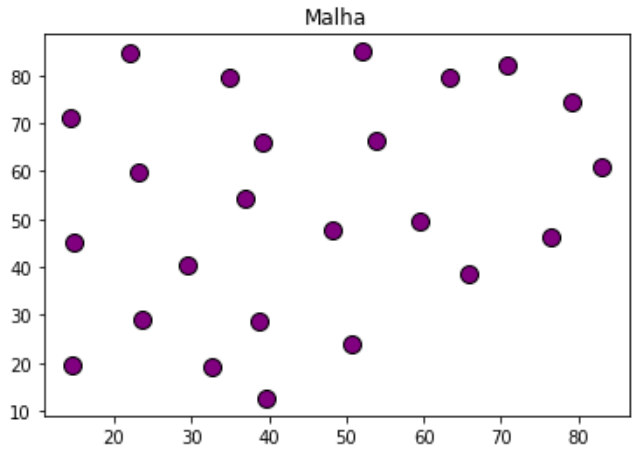

Fig. 5. Malha contendo 24 pontos.

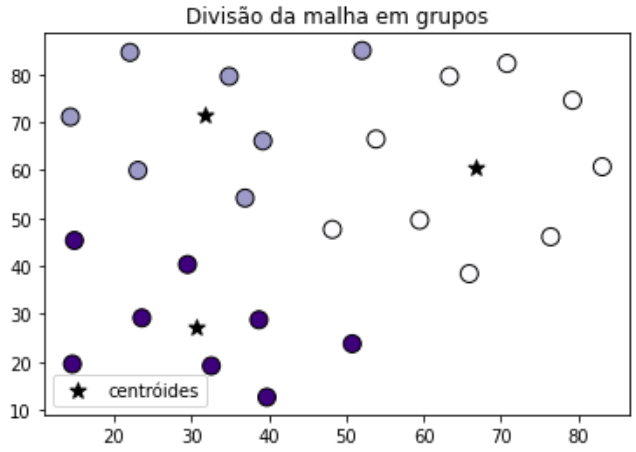

Fig. 6. Malha da Figura 5 dividida em 3 grupos gerados pelo K-Means, contendo 8, 8 e 7 pontos.

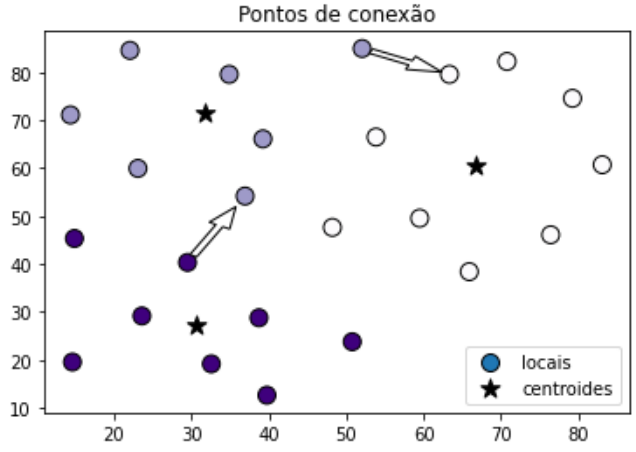

Fig. 7. Malha da Figura 5 com a divisão de grupos da Figura 6 com seus centróides e conexões representados por estrelas e setas, respectivamente.

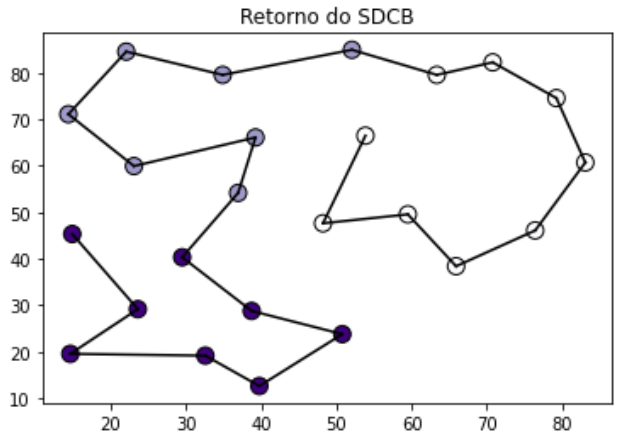

Fig. 8. Malha da Figura 5 com a divisão de grupos da Figura 6 e com seus pontos conectados pela rota resultante da implementação do algoritmo Solução Utilizando Divisão e Conquista e Curva de Hilbert.

Ao determinar um limite de pontos que serão computados pelo B&B e definir o ponto inicial e/ou final dos grupos, é possível prever um pior caso e, assim, calcular o número máximo de iterações que ocorrerão: no pior cenário o B&B computará tantas operações quanto a força bruta:$P_k!$, exceto pelos casos que não contemplem as limitações estabelecidas, visto que, ao ter pontos fixos, os mesmos são retirados da conta do fatorial. Sendo 10 o maior número que P_k pode assumir, o pior caso de execução do B&B por grupo torna-se $(10-1)! = 9!$ para os grupos inicial e final, e $(10-2)! = 8!$ para os grupos intermediários, visto que estes possuem definidos pontos iniciais e finais, e o primeiro e último grupo, apenas o inicial ou final. Isto posto, o número total de iterações no pior caso do B&B no SDCB (i_b) é:

$$i_b = 2 \times 9! + (g-2) \times 8! \quad (7)$$

IV. Resultados e discussão

Para verificar o desempenho prático do SDCB, este foi submetido a testes utilizando malhas com pontos definidos de maneira aleatória para comparar seus resultados aos de outros métodos citados. Estes testes direcionarão a análise do desempenho por meio da acuracidade e do tempo de execução do retorno do SDCB.

A. Acuracidade

O critério usado para determinar a acuracidade do algoritmo foi a porcentagem média aproximada de diferença entre o peso dos resultados computados pelo SDCB e pelo B&B, ou seja, o quão maior é o caminho retornado pelo programa desenvolvido em relação ao menor caminho possível. Essa média foi retirada de 100 execuções para malhas com até 11 pontos e de 10 execuções para malhas de 12 a 17 pontos, visando obtê-la em tempo de resposta

viável, e parando neste número, pois ele é suficiente para interpretar o algoritmo.

Em entradas com até 8 vértices, esse número se manteve constante como 0%; até 16, este valor variou majoritariamente em torno de 2%, mas tendo exceções de 0, 4 e 17%; e, ao alcançar as malhas de 17 pontos, a porcentagem assumiu irregularmente valores entre 0 e 20.

Visto que o número desejado de pontos por aglomeração foi estabelecido como 8, em malhas com até 8 vértices há apenas um grupo; por conseguinte, o seu resultado sempre será semelhante ao do B&B, gerando diferença de 0%. De 9 a 16 pontos, a tendência é a formação de dois grupos, e de 17 a 24 elementos, de ocorrerem três agrupamentos.

A fim de ilustrar a discussão sobre os dados expostos, serão exemplificados nas Figuras 9 a 16 retornos do SDCB seguidos das menores rotas possíveis, relativos a 4 malhas distintas. Vale ressaltar que os exemplos são casos selecionados para evidenciar as possíveis diferenças entre as soluções em questão, e, portanto, não necessariamente contemplam a maioria das ocorrências.

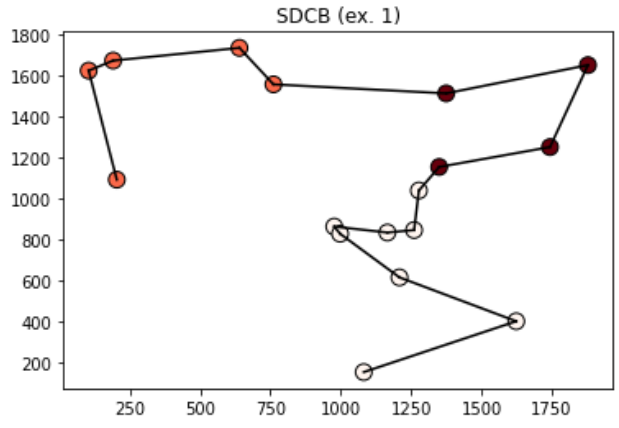

Fig. 9. Malha do Exemplo 1 percorrida pelo retorno do SDCB.

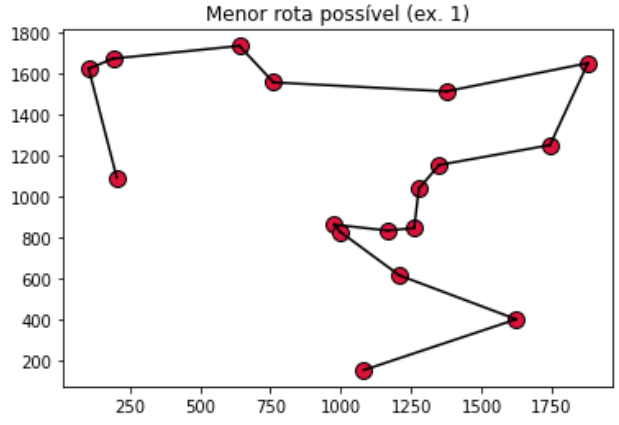

Fig. 10. Malha do Exemplo 1 percorrida pela menor rota possível.

Ao observar a menor rota possível e a solução do SDCB para a malha do Exemplo 1 (Figuras 10 e 11), percebe-se três fatores que podem justificar os dados obtidos nos testes: I. todos os vértices agrupados na solução do SDCB são percorridos na rota ótima sem que pontos de outros grupos sejam percorridos antes; II. os dois pontos que conectam os grupos na rota do SDCB estão igualmente conectados na rota de menor custo; e III. a sequência dos grupos determinada pela HSFC é contemplada pela menor rota. A união desses fatores resultou em 0% de diferença entre ambas soluções.

Por outro lado, no Exemplo 2, a menor rota (Figura 11) não se detém aos grupos determinados no percurso retornado pelo SDCB (Figura 12), o que resultou em diferença de 17% entre as rotas, por mais que os pontos de conexão estivessem conectados na rota ótima. E, no caso do Exemplo 3, apesar de os vértices agrupados na solução do SDCB (Figura 13) serem percorridos sem interrupção na menor rota possível (Figura 14), a rota desta não contempla a conexão entre os grupos, fator determinante para a diferença de peso entre as duas soluções ser 4%.

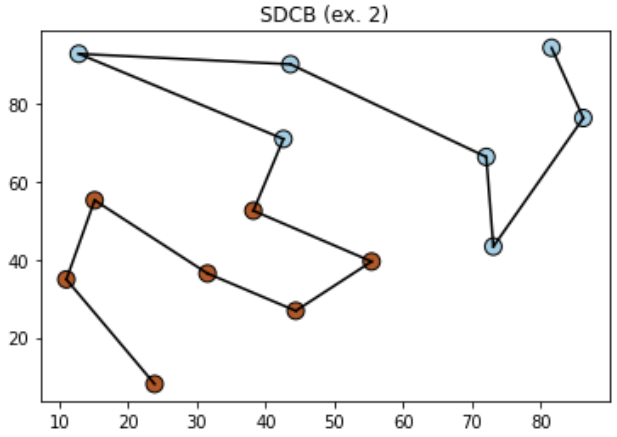

Fig. 11. Malha do Exemplo 2 percorrida pelo retorno do SDCB.

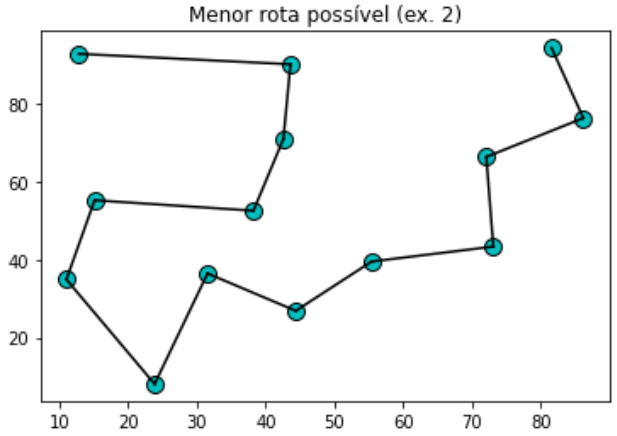

Fig. 12. Malha do Exemplo 2 percorrida pela menor rota possível.

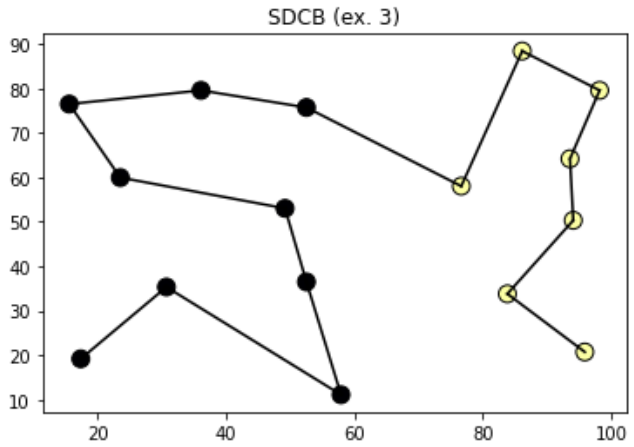

Fig. 13. Malha do Exemplo 3 percorrida pelo retorno do SDCB.

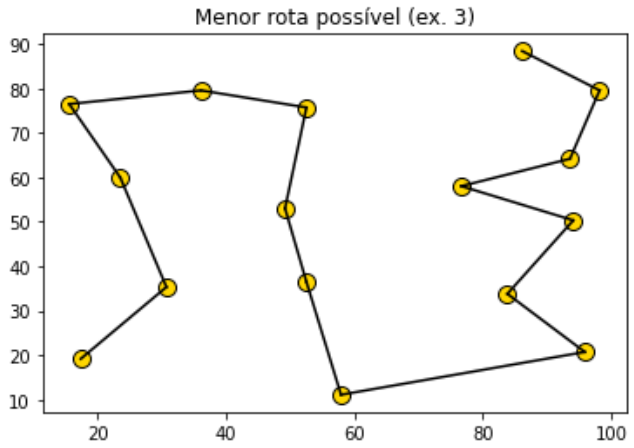

Fig. 14. Malha do Exemplo 3 percorrida pela menor rota possível.

Além das duas possíveis divergências entre as soluções citadas, há outro fator que pode diferir assim que o número de grupos superar dois: a sequência dos grupos. Nota-se no Exemplo 4 que todos os elementos de cada grupo da SDCB são percorridos antes de a rota abranger pontos de outros grupos, mas a sequência em que esses pontos agrupados são percorridos na menor rota é diferente, levando a diferença entre as duas rotas a ser de 16%.

Isto posto, repara-se que a probabilidade do retorno do SDCB ser exatamente igual ao da força bruta depende de se os pontos agrupados não são interrompidos, se as conexões entre os grupos estão presentes na menor rota e se a sequência dos agrupamentos é a mesma. Logo, é notável que, quanto maior a quantidade de grupos em que pontos da malha são divididos, menor a probabilidade de o retorno ser ótimo, e maior a tendência da porcentagem da sua diferença com a menor rota crescer.

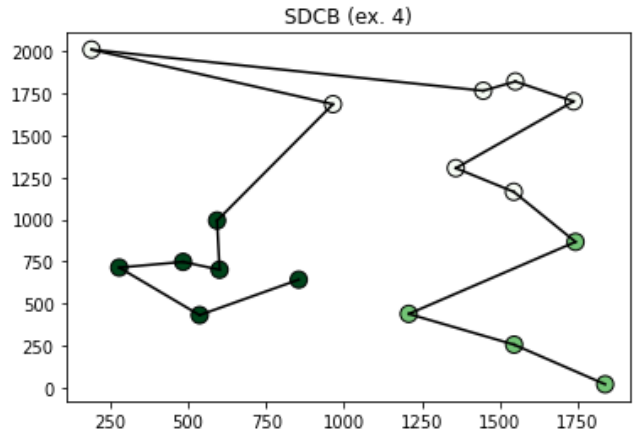

Fig. 15. Malha do Exemplo 4 percorrida pelo retorno do SDCB.

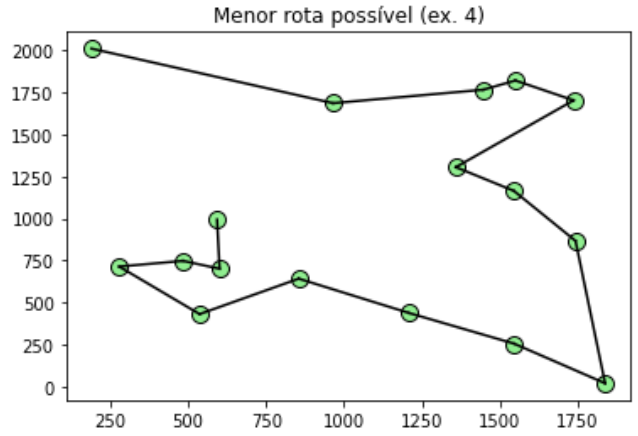

Fig. 16. Malha do Exemplo 4 percorrida pela menor rota possível.

B. *Tempo de execução*

Apesar da acuracidade obtida pelo SDCB não ser satisfatória em alguns casos, seu tempo de execução o torna uma ferramenta viável para casos que priorizam obter a resposta com mais agilidade. No intuito de comparar o tempo de resposta do SDCB com os de outras soluções, foram realizados testes em um ambiente de execução fixo relativos aos tempos de execução dos algoritmos de força bruta, B&B e SDCB, em função do número de vértices da malha de entrada. A comparação é representada no recorte de gráfico da Figura 17

Observando a Figura 17, nota-se o problema da complexidade fatorial presente nos algoritmos B&B e K-Means tornando-se mais visível. Para entradas com números limitados de pontos ele mostra-se uma solução viável, no entanto, conforme a quantidade de vértices se torna mais expressiva, o tempo necessário para obter um retorno acaba tornando-o impraticável.

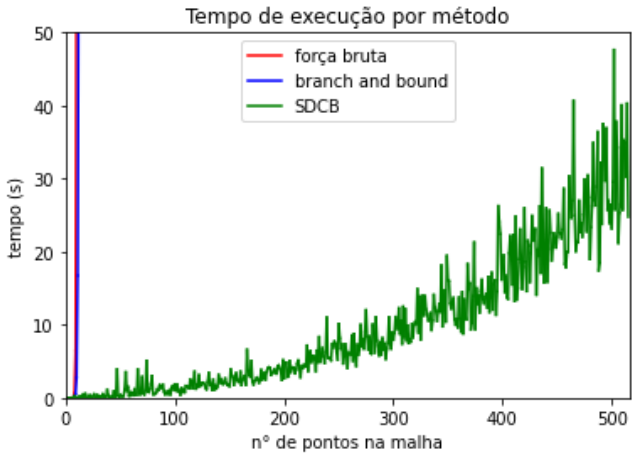

Fig. 17. Gráfico de linhas apresentando a comparação entre os tempos execução dos algoritmos: força bruta, *branch and bound* e SDCB

O tempo consumido pela execução do SDCB pode ser estimado a partir do tempo gasto por cada uma de suas etapas, que foi explicado na seção II: durante a divisão de grupos pelo K-Means, o número máximo de iterações pode ser calculado a partir do número de vértices e é uma função linear, mas sua complexidade e re-execuções o tornam exponencial. O impacto causado pelo K-Means no tempo de execução está ilustrado no gráfico da Figura 18. As computações realizadas pelo B&B também seguem um padrão exponencial, não mais fatorial devido ao processo de divisão e conquista. A ordenação pela pseudo-curva de Hilbert tem seu tempo de execução calculado por uma função logarítmica. Dado que o aumento de tempo mais significativo de suas etapas é exponencial, é seguro afirmar que o tempo consumido pelo SDCB em função do número de vértices é exponencial.

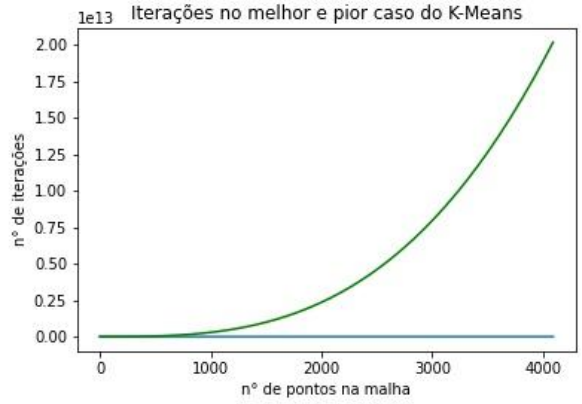

Fig. 18. Gráfico com o pior número de iterações: do pior caso de re-execuções do K-Means (em verde); e do caso em que o K-Means executa apenas uma vez (em azul)

Entretanto, devido à irregularidade do tempo de execução do B&B, das divisões dos grupos e das possíveis re-execuções do K-Means para evitar grupos com mais de 10 pontos, nota-se oscilação nos tempos do SDCB. Mas, considerando que seus casos máximos crescem de maneira exponencial, pode-se afirmar que ele é uma alternativa mais ágil que a força bruta e o *branch and bound,* cujos casos máximos crescem de maneira fatorial.

V. Conclusão

Sendo um tema pesquisado desde o século XIX [16], existem diversas maneiras de solucionar o problema do caixeiro viajante a fim de diminuir seu tempo de execução. A do presente artigo, SDCB, aborda três outras: divisão e conquista, *branch and bound* e curva de preenchimento de espaço de Hilbert, tendo enfoque em casos euclidianos. Ela contempla quatro passos: dividir a malha em grupos — K-Means; determinar a sequência de grupos — curva de Hilbert; determinar os pontos de conexão entre os grupos; e percorrer os grupos internamente com as limitações dos pontos de conexão.

Os resultados experimentais demonstraram que, apesar de a probabilidade da rota retornada pelo SDCB ser a menor possível diminuir a cada agrupamento realizado, a sua acuracidade se mantém imprevisível. E o tempo de execução do SDCB cresce irregularmente de maneira exponencial, em função do número de pontos da entrada, fazendo sua aplicação viável em contextos reais de aplicação do PCV.

Adicionalmente, o desenvolvimento deste trabalho, pelo seu caráter formativo dos autores, possibilitou aos mesmos o entendimento mais aprofundado a respeito das possibilidades e limitações de algoritmos computacionais, assim como expandiu o vislumbre de maneiras como contorná-las.

Para trabalhos futuros, propõe-se a aplicação do SDCB em diferentes formações do PCV além da geométrica plana, como a clássica ou a tridimensional; sugere-se também a alternação do K-Means por outros algoritmos de agrupamento, assim como estudar outras maneiras de implementar a curva de Hilbert e aplicação de algoritmos preliminares ao B&B, visando testar qual aplicação retorna melhor tempo de execução e confiabilidade.

Agradecimentos

Os autores deste artigo desejam agradecer aos seus familiares pelo apoio e ao Instituto Federal do Paraná que ampliou seus conhecimentos possibilitando o desenvolvimento desse projeto, incluindo seus professores e servidores, especialmente ao orientador Darlon Vasata, à professora Poliana Sella e ao técnico de laboratório de informática Lucas Ruchel.

Referências

[1] J. Y. Liu, R. Linn, P. S. H. Kowe, “A study on heuristic methods for PCB drilling route optimization”, Source International Journal of Industrial Engineering: Theory, Applications and

Practice", v. 6, p. 289-296, 1999.

[2] T. C. Holanda, "O Relacionamento do problema de sequenciamento clássico com o problema do caixeiro viajante e sua resolução numa abordagem evolutiva", 2015

[3] C. E. Mokhi, A. Addaim; "Optimization of Wind Turbine Interconnections in an Offshore Wind Farm Using Metaheuristic Algorithms", National School of Applied Sciences, Ibn Tofail University, 14000 Kenitra, Morocco, 2020

[4] W. R. Mendes, "Modelo híbrido constituído por algoritmo genético e Space Filling Curve aplicado à otimização de rotas veiculares.", 2017. 139 f. Dissertação (Mestrado em Engenharia de Controle e Automação) - Instituto Federal do Espírito Santo, Serra, 2017.

[5] K. L. Hoffman, M. Padberg, "Traveling salesman problem." In: Gass S.I., Harris C.M. (eds) Encyclopedia of Operations Research and Management Science. Springer, New York, NY, 2001. https://doi.org/10.1007/1-4020-0611-X_1068

[6] M. Jünger, R. Gerhard, R. Giovanni, "Chapter 4 The traveling salesman problem." in Science Direct, 1995.

[7] I. Tomescu. "An upper bound for the shortest hamiltonian path in the symmetric euclidean case", RAIRO-Operations Research, 1983.

[8] J. Marion, R. Péchoux, "Characterizations of polynomial complexity classes with a better intensionality." in Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming (PPDP '08), Association for Computing Machinery, New York, NY, USA, 79–88, 2008.

[9] A. Paz, S. Moran, "Non Deterministic Polynomial Optimizations Problems and Their Aproximations" in Theoretical Computer Science 15, pp. 51-277, North-Holland Publishing Company, 1979.

[10] J. Carlson, A. M. Jaffe, and A. Wiles, The millennium prize problems. Cambridge, MA: Clay Mathematics Inst., p 87-88, 2006.

[11] A. Lusem "P = NP ou as Sutilezas da Complexidade Computacional" in Matemática Universitária, Instituto de Matemática Pura e Aplicada - CNPq, N° 5, pp. 33-60, 1987.

[12] S. Smale, "On the efficiency of algorithms of analysis", Bulletin Of The American Mathematical Society, 1985.

[13] S. Violina, "Analysis of Brute Force and Branch & Bound Algorithms to solve the Traveling Salesperson Problem (TSP)", in Turkish Journal of Computer and Mathematics Education, vol.12, no.8, pp. 1226-1229, 2021.

[14] J. Clausen, "branch and bound algorithms-principles and examples", Department of Computer Science, University of Copenhagen, p. 1-30, 1999.

[15] T. Phienthrakul, "Clustering Evolutionary Computation for Solving Travelling Salesman Problems", International Journal of Advanced Computer Science and Information Technology (IJACSIT), Vol. 3, No. 3, pp. 243-262, 2014.

[16] L. K. Platzman; J. J. Batholdi III, "An O(NlogN) Planar Travelling Salesman Heuristic Based on Spacefilling Curves", School of Industrial and Systems Engineering Georgia Institute of Technology Atlanta, Georgia 30332, PDRC Report Series 82-07, April 1982.

[17] D. Arthur, S. Vassilvitskii, "How slow is the k-means method?" in SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry, 2006.

[18] scikit-learn.org, 'sklearn.cluster.KMeans — scikit-learn 1.0 documentation', 2007. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans. [Accessed: 27- Sep- 2021].

[19] F. R. Rafaeli et. al. , "Introdução à Geração de Malhas Triangulares", Sociedade Brasileira de Matemática Aplicada, Vol 79, pp. 2-3, 2015.

[20] H. Haverkort, "An inventory of three-dimensional Hilbert space-filling curves", 2011.

[21] G. P. Da Cruz. "Fractais: Padrões complexos de incrível beleza", Universidade Federal do Pará, 2014.

[22] H. S. Warren, "Hacker's Delight", Pearson Education, 2013.