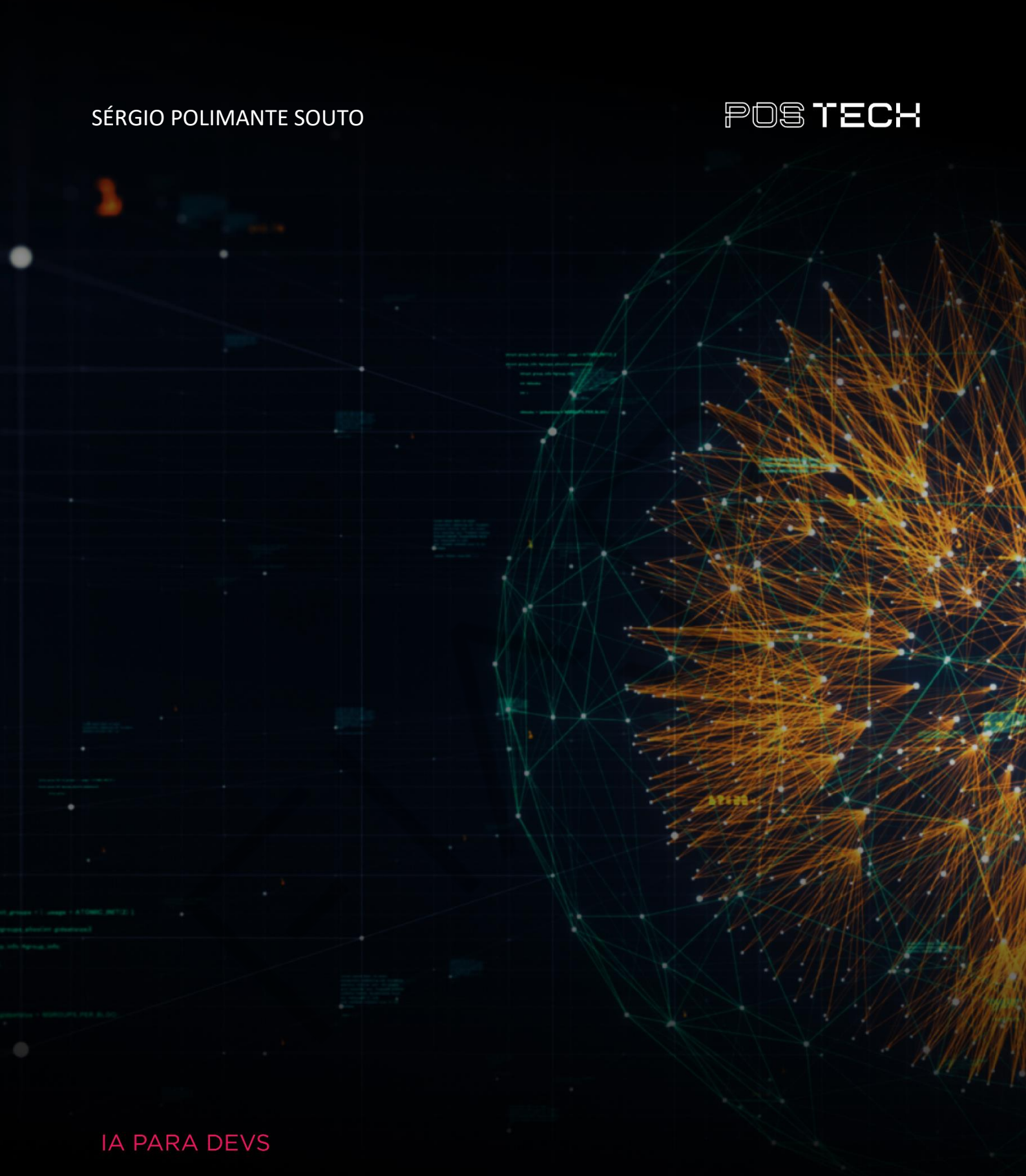
SÉRGIO POLIMANTE SOUTO

PÓS TECH



IA PARA DEVS

INTRODUÇÃO AO ALGORITMO GENÉTICO

AULA 01

SUMÁRIO

O QUE VEM POR AÍ?

Antes de iniciarmos nossa exploração neste renomado método de otimização bioinspirado, é crucial aprofundarmos nossos conhecimentos sobre o conceito de otimização. Nesta aula, abordaremos os fundamentos dos problemas de otimização, destacando diferentes tipos de algoritmos, como os exatos e heurísticos. Além disso, teremos a oportunidade de observar na prática a aplicação desses algoritmos na resolução de um problema clássico quando falamos sobre este tema: o “desafio do caixeiro viajante”. Este será um mergulho prático e esclarecedor, proporcionando uma compreensão mais sólida e aplicada dos conceitos discutidos, abrindo caminho para entrarmos nos Algoritmos Genéticos!

HANDS ON

Na primeira videoaula, mergulharemos no fascinante mundo dos algoritmos genéticos, uma classe especial de algoritmos de otimização. Para compreender a essência desses algoritmos, começamos definindo o que é um algoritmo de otimização. Exploraremos a ideia de melhorar soluções, delimitando as restrições que moldam os problemas de otimização. Utilizaremos o desafiador "Problema do Caixeiro Viajante" (do inglês, “Traveling Salesman Problem”, o que dá origem à sigla comum TSP) como exemplo, desdobrando sua função objetivo, revelando a codificação de possíveis soluções e elucidando suas restrições.

Aprofundaremos a compreensão dos tipos de problemas de otimização, distinguindo entre minimização e maximização, lineares e não lineares, contínuos e discretos, destacando o TSP como um exemplo emblemático. Em seguida, exploraremos as variedades de soluções, diferenciando métodos exatos, como a análise analítica e força bruta, que garantem a resposta ótima. Demonstraremos, de maneira prática, a aplicação da força bruta para o “problema do caixeiro viajante”, evidenciando que, em certos problemas, encontrar a solução ótima pode requerer a avaliação de todas as possíveis soluções.

Introduziremos os métodos heurísticos como soluções eficientes para problemas de larga escala e complexos. Demonstramos a técnica do "vizinho mais próximo" (do inglês “*nearest neighbour”*) aplicada ao TSP, enfatizando a velocidade e viabilidade dessas soluções. Além do “vizinho mais próximo”, vamos ver na prática outros algoritmos heurísticos determinísticos funcionando para resolver o “problema do caixeiro viajante na prática”, como o Convex Hull. Vamos discutir sobre a importância desse tipo de algoritmo em encontrar soluções factíveis (subótimas) com baixo custo computacional. E, por fim, discutiremos sobre outras abordagens heurísticas e apresentaremos os algoritmos genéticos como um algoritmo heurístico bioinspirado.

SAIBA MAIS

OUTROS PROBLEMAS DE OTIMIZAÇÃO

Problemas de otimização estão presentes em nosso cotidiano, abrangendo desde situações simples até desafios complexos. Para ilustrar a natureza desses problemas, podemos considerar casos simples, como o clássico problema da regressão linear. Encontrar a reta que se enquadra melhor em um conjunto de pontos é um problema de otimização onde queremos minimizar a métrica dos quadrados das diferenças entre os valores previstos e os valores reais.

Otimização também é algo que nós fazemos em nosso dia a dia sem talvez nos darmos conta. Por exemplo, quando queremos minimizar o tempo gasto em um trajeto diário para o trabalho. Nesse cenário, a otimização pode envolver a escolha da rota mais eficiente, levando em consideração variáveis como trânsito e distância, com o objetivo de minimizar o tempo total de deslocamento.

Por outro lado, em situações mais complexas, podemos citar o planejamento logístico de uma cadeia de suprimentos global. Aqui, a otimização busca coordenar eficientemente o transporte de mercadorias entre diferentes locais, considerando diversos fatores, como custos de transporte, tempos de trânsito, capacidade de armazenamento e demanda variável. O desafio reside em encontrar uma configuração logística que minimize os custos totais, garantindo ao mesmo tempo um fluxo eficiente de produtos.

Esses exemplos ilustram como problemas de otimização permeiam aspectos práticos de nossa vida diária, desde as escolhas mais simples até os desafios complexos enfrentados por empresas e indústrias. A aplicação de técnicas de otimização é fundamental para encontrar soluções eficazes em diversas situações, destacando a relevância desse campo em nosso dia a dia.

Entre os muitos tipos de desafios de otimização que enfrentamos, destacam-se problemas lineares, não lineares, inteiro-mistos, entre outros. Cada categoria apresenta suas próprias complexidades e características distintas. Neste contexto, exploraremos além das otimizações convencionais e discutiremos outras abordagens específicas: otimização convexa, combinatória, estocástica e multi objetivo. Essas variantes oferecem perspectivas únicas para lidar com problemas mais intricados e

adaptar-se a uma variedade de cenários aplicados. A compreensão dessas nuances é essencial para desenvolver soluções abrangentes e eficazes em diversos contextos de otimização.

Otimização Convexa

A otimização convexa refere-se a um ramo específico da matemática aplicada, que lida com um subconjunto de problemas de otimização nos quais a função objetivo e as restrições apresentam características convexas. Uma função é considerada convexa se o segmento de linha entre quaisquer dois pontos em seu domínio reside acima do gráfico da função, sendo esta propriedade estendida também às restrições do problema. Por exemplo, uma parábola é uma função convexa.

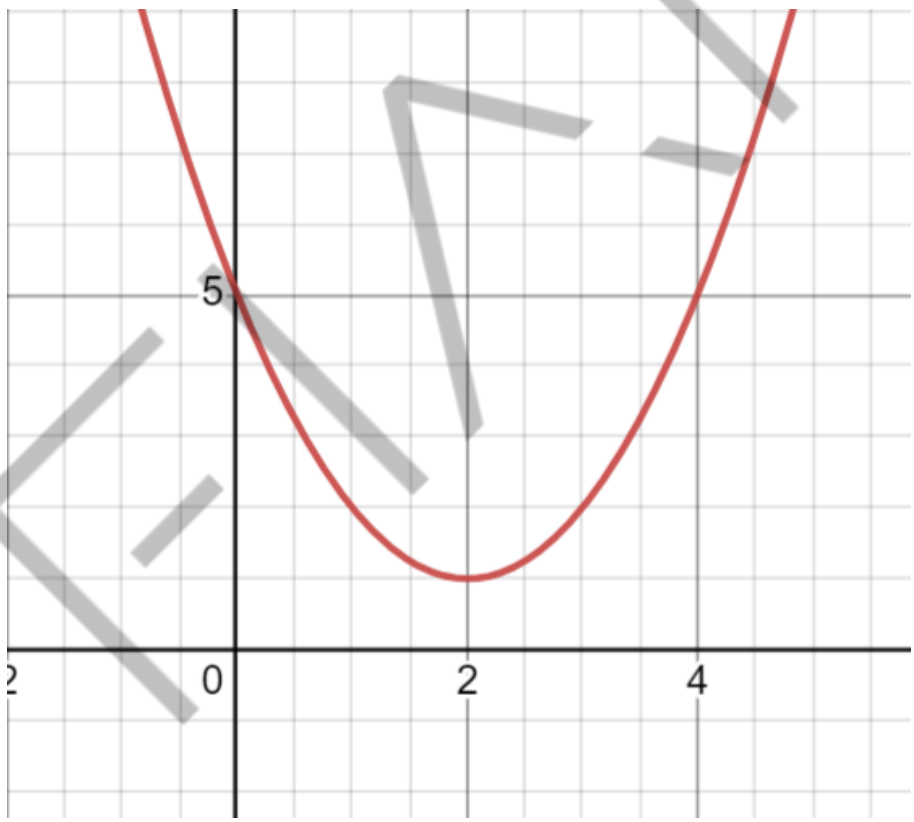


Figura 1 - Parábola é um exemplo de função convexa
Fonte: elaborado pelo autor (2024)

Existem diversos métodos numéricos empregados para resolver problemas de otimização convexa, sendo dois notáveis o algoritmo simplex e o método do gradiente. O algoritmo simplex destaca-se como uma abordagem clássica para a solução de

problemas de programação linear. Funcionando iterativamente, ele começa com uma solução viável e se move de forma ascendente no espaço de soluções, buscando iterativamente a melhoria. O algoritmo simplex é eficaz, especialmente em problemas de dimensões moderadas, mas pode enfrentar desafios em casos de alta dimensionalidade. Por outro lado, o método do gradiente é amplamente utilizado para problemas de otimização convexa mais gerais. Baseando-se no conceito de derivadas parciais, o método do gradiente procura encontrar a direção de máximo crescimento da função objetivo e, a partir disso, ajusta iterativamente os parâmetros para otimizar a solução. Apesar de sua aplicabilidade abrangente, o método do gradiente pode ser sensível à escolha da taxa de aprendizado, exigindo ajustes cuidadosos para garantir convergência eficiente. Ambos os métodos desempenham um papel crucial na resolução de problemas de otimização convexa, oferecendo abordagens distintas para atender a diferentes contextos e complexidades.

Otimização Combinatória

A otimização combinatória concentra-se na busca de soluções para problemas discretos, nos quais as decisões a serem tomadas envolvem a seleção de elementos de um conjunto finito. Este campo abrange uma vasta gama de desafios, como “o problema do caixeiro viajante”, alocação de recursos, e programação de horários. A complexidade desses problemas decorre da natureza discreta das variáveis envolvidas, exigindo estratégias específicas para encontrar soluções eficientes.

Na indústria, a otimização combinatória desempenha um papel crucial em diversas áreas. Por exemplo, na logística, as otimizações são frequentemente aplicadas para agilizar rotas de entrega, minimizando custos e tempo. Na indústria de manufatura, a programação de produção é um desafio combinatório comum, buscando maximizar a eficiência da linha de produção. A maior parte dos problemas de agendamento de todos os tipos são modelados como problemas de otimização combinatória, como escala de funcionários e máquinas, atendimento a clientes, horários de saída e chegada, todos os tipos de transportes como ônibus, avião, barco etc.

Bibliotecas em Python, como NetworkX para problemas em grafos e PuLP para programação linear inteira, são ferramentas valiosas para implementar algoritmos de otimização combinatória de maneira eficiente e escalável.

Otimização Estocástica

A otimização estocástica lida com problemas em que a incerteza desempenha um papel significativo, incorporando elementos de aleatoriedade nas variáveis do modelo. Diferentemente dos problemas determinísticos, nos quais as condições são totalmente conhecidas, a otimização estocástica busca soluções robustas que considerem a variabilidade inerente ao ambiente. Essa abordagem é especialmente relevante em situações em que fatores imprevisíveis, como flutuações de demanda, variações de preços ou eventos inesperados, podem impactar as decisões ótimas.

Na indústria, a otimização estocástica encontra aplicação em diversos cenários. Por exemplo, na gestão de cadeias de suprimentos, onde as condições do mercado e as interrupções logísticas podem ser incertas, a otimização estocástica é crucial para desenvolver estratégias de estoque e distribuição resilientes. Bibliotecas de Python como a “Pyomo” oferecem recursos para modelagem de otimização estocástica, permitindo a implementação de algoritmos que consideram a incerteza e fornecem soluções robustas em ambientes dinâmicos e imprevisíveis. Por exemplo, este código é um exemplo de como resolver o problema clássico de otimização chamado knapsack usando a biblioteca Pyomo. O código foi extraído da documentação do Pyomo.

```
# knapsack.py

from pyomo_simplemodel import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
limit = 14
items = list(sorted(v.keys()))

# Create model
m = SimpleModel(maximize=True)

# Variables
x = m.var('m', items, within=Binary)

# Objective
m += sum(v[i]*x[i] for i in items)

# Constraint
m += sum(w[i]*x[i] for i in items) <= limit



# Optimize
status = m.solve('glpk')
```

```
# Print the status of the solved LP
print("Status = %s" % status.solver.termination_condition)

# Print the value of the variables at the optimum
for i in items:
    print("%s = %f" % (x[i], value(x[i])))

# Print the value of the objective
print("Objective = %f" % value(m.objective()))
```

Otimização Multi-Objetivo

A otimização multi objetivo envolve a busca simultânea por soluções que otimizem mais de um critério, muitas vezes conflitantes. Diferentemente da otimização tradicional, que busca uma solução única e ótima, a otimização multi objetivo lida com a complexidade inerente às situações em que diferentes objetivos competem entre si. Esse campo visa encontrar um conjunto de soluções, conhecido como “fronteira de Pareto”, que representam compromissos ideais entre os objetivos conflitantes.

Na indústria, a otimização multi objetivo é aplicada em diversas áreas. Por exemplo, na engenharia de design, onde é crucial considerar simultaneamente fatores como custo, desempenho e sustentabilidade. Em sistemas de energia, a otimização multi objetivo é utilizada para equilibrar a eficiência operacional e a redução das emissões de carbono. Em veículos aéreos autônomos, a otimização multiobjetivo busca maximizar ou minimizar as distâncias percorridas enquanto minimiza os ângulos de manobra. Bibliotecas em Python, como a DEAP (Distributed Evolutionary Algorithms in Python) usando algoritmos como NSGAII, oferecem ferramentas para a implementação de algoritmos evolutivos e outras técnicas eficazes na busca de soluções de otimização multi objetivo. Veja mais detalhes sobre o DEAP com NSGAII.

Tipos de Problemas de Otimização e Complexidade Computacional

Os problemas de otimização podem ser categorizados com base em sua complexidade computacional. Um termo comum é "NP-hard", que se refere a problemas para os quais não se conhece um algoritmo eficiente para encontrar uma solução em tempo polinomial. Em outras palavras, à medida que o tamanho do problema aumenta, a quantidade de tempo necessária para encontrar uma solução cresce de maneira substancial. Exemplos de problemas NP-hard incluem o “problema do caixeiro viajante”, o “problema da mochila” e muitos problemas de programação linear inteira.

Na indústria, esses problemas desafiadores têm aplicação em várias áreas. Por exemplo, na otimização de rotas de entrega, onde encontrar a rota mais eficiente pode ser um problema NP-hard devido às muitas variáveis envolvidas. Em finanças, a alocação otimizada de portfólio pode ser um desafio NP-hard, dada a complexidade de considerar múltiplos ativos e restrições.

A complexidade computacional em otimização é crucial, pois determina a viabilidade prática de resolver problemas em tempo útil. Algoritmos de otimização eficientes são fundamentais para lidar com problemas NP-hard. Métodos heurísticos e algoritmos aproximados, como algoritmos genéticos, algoritmos de enxame de partículas (do inglês, “Particle Swarm Optimization” - PSO) e algoritmos de otimização por colônia de formigas (do inglês, “Ant Colony Optimization” - ACO), são frequentemente empregados para encontrar soluções próximas ao que seria considerada ótima em um tempo razoável. Além disso, métodos exatos, como programação linear inteira, podem ser aplicados para problemas específicos, embora a escalabilidade possa ser limitada em problemas de grande porte. A escolha do algoritmo depende da natureza do problema e dos recursos disponíveis para a sua resolução.

OTIMIZAÇÃO COMBINATÓRIA, PCV e CUSTO COMPUTACIONAL

O “Problema do Caixeiro Viajante” – ou PCV, é um dos problemas mais conhecidos e estudados na teoria da complexidade computacional. Ele pertence à classe de problemas NP-completos, o que significa que não se conhece nenhum algoritmo eficiente (polinomial) para resolvê-lo em todos os casos, e que, se alguém encontrasse uma solução eficiente para um caso específico, essa solução poderia ser estendida para resolver todos os casos da classe NP em tempo polinomial. Veja abaixo uma lista de alguns problemas NP-completos:

- Boolean satisfiability problem (SAT)
- Knapsack problem
- Hamiltonian path problem
- Subgraph isomorphism problem
- Clique problem

Veja uma lista completa aqui.

Formulação clássica do Problema do Caixeiro Viajante

Dada uma lista das distâncias entre cada par de cidades, o objetivo é encontrar o menor caminho, para que se visite cada cidade exatamente uma vez e que seja possível retornar à cidade de origem.

Held-Karp: o algoritmo mais eficiente que resolve o PCV

A complexidade computacional do PCV é frequentemente expressa em termos da notação de Landau. O algoritmo mais eficiente conhecido para resolver o PCV em sua forma geral é o algoritmo de Held-Karp, feito em programação dinâmica, que tem uma complexidade de tempo de aproximadamente O(n^2 * 2^n), onde 'n' é o número de cidades. Esta é uma complexidade exponencial, o que significa que o tempo de execução aumenta de maneira exponencial com o aumento do número de cidades.

Essa complexidade exponencial é o que torna o PCV um problema desafiador, especialmente à medida que o número de cidades cresce. Atualmente, não existe um algoritmo conhecido que resolva o PCV em tempo polinomial, e a busca por soluções

eficientes é um tópico ativo de pesquisa na teoria da complexidade computacional e na otimização combinatória.

Custo computacional usando força bruta

A complexidade computacional do método de força bruta para resolver o “Problema do Caixeiro Viajante” (PCV) é O(n!), onde n é o número de cidades. Isso significa que o tempo de execução do algoritmo cresce fatorialmente com o número de cidades, tornando-se rapidamente impraticável à medida que n aumenta.

O método de força bruta aborda o PCV considerando todas as possíveis permutações dos caminhos que o caixeiro viajante pode tomar e calculando o comprimento total de cada caminho. Em seguida, seleciona o caminho com o comprimento mínimo como a solução. O número de permutações de n elementos é n! (n fatorial), que é a razão pela qual a complexidade é O(n!).

Veja na imagem a seguir, a linha vermelha ilustra a complexidade computacional O(n!), enquanto a azul representa a complexidade do algoritmo mais eficiente, Held-Karp, com complexidade computacional O(n^2 * 2^n). Note que para valores de cidades menor do que 9, o método por força bruta é mais eficiente, mas para problemas com o número de cidades maior do que 9, o método Held-Karp torna-se mais eficiente na medida que o número de cidades aumenta.

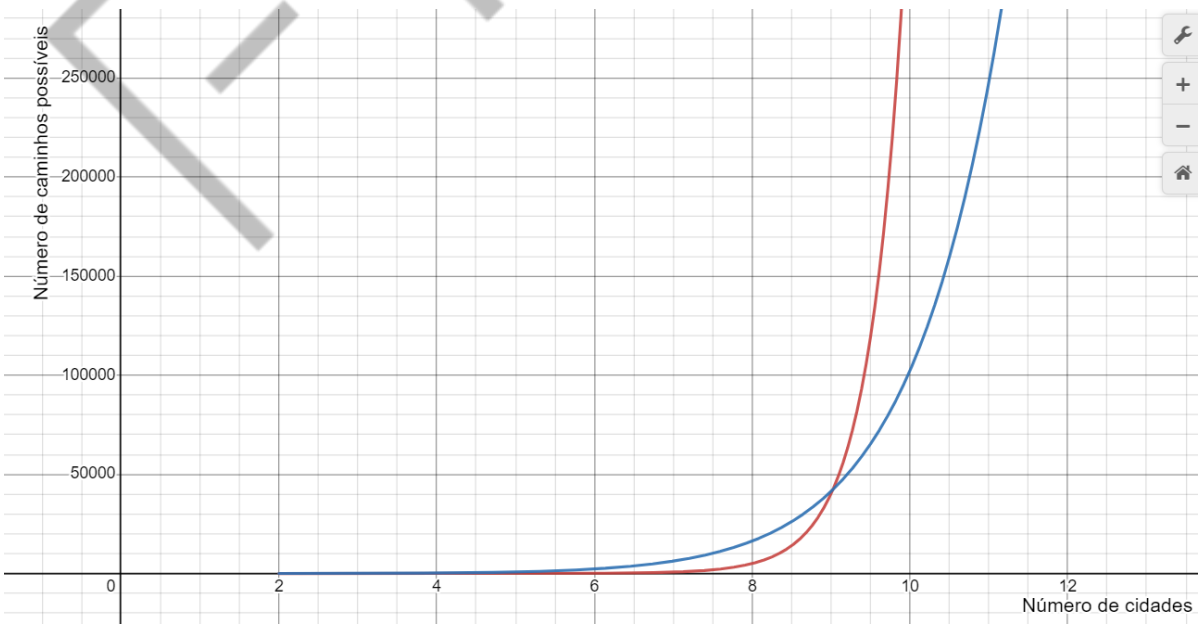


Figura 2 - Complexidade computacional. Vermelho: O(n!), Azul: O(n^2 * 2^n)
Fonte: elaborado pelo autor (2024)

Veja o gráfico a seguir, que foi feito experimentalmente usando o código fornecido mais adiante. Note como o tempo decorrido para encontrar o trajeto ótimo cresce de maneira muito rápida com o maior número de cidades.

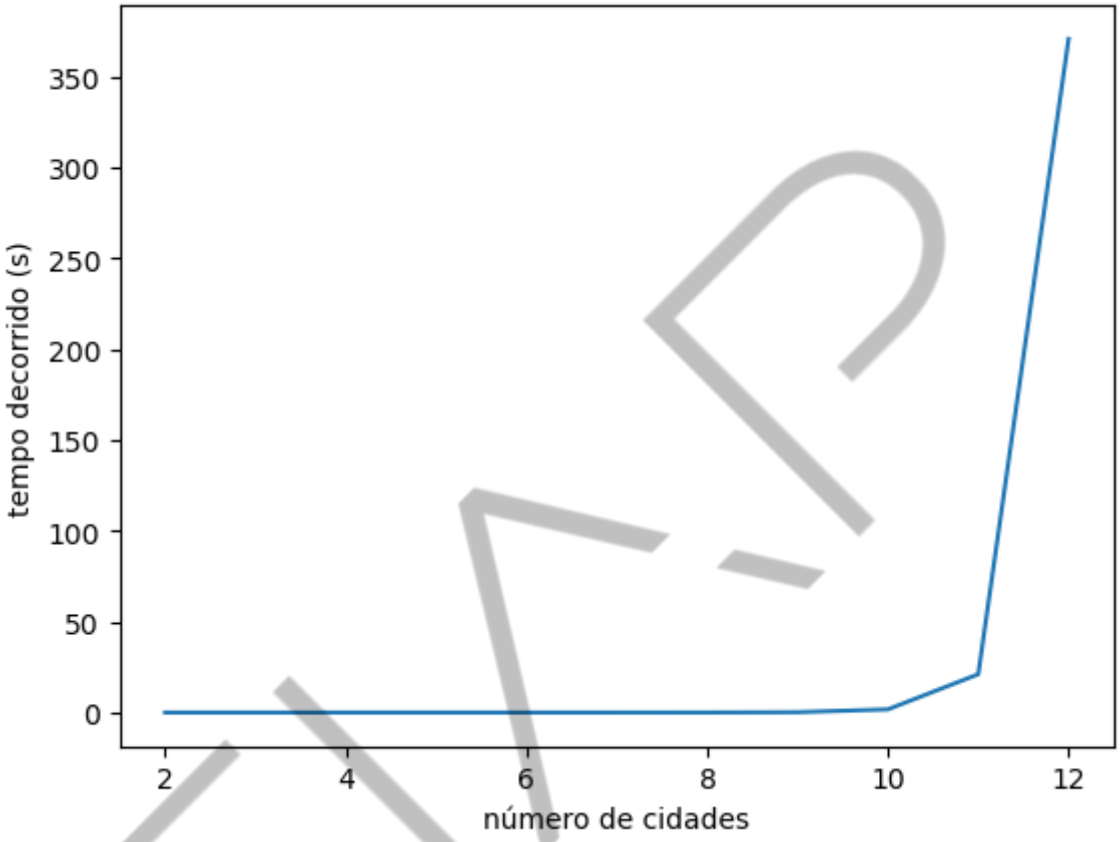


Figura 3 - Tempo de processamento decorrido para diferentes quantidades de cidades na solução do TSP por força bruta
Fonte: elaborado pelo autor (2024)

Como n! cresce muito rapidamente, o método de força bruta torna-se impraticável para instâncias do PCV com um número relativamente pequeno de cidades. Portanto, embora o método de força bruta seja conceitualmente simples, não é uma abordagem eficiente para resolver o PCV em casos gerais. Em vez disso, métodos mais sofisticados, como algoritmos de programação dinâmica (por exemplo, o algoritmo de Held-Karp) ou heurísticas (por exemplo, algoritmos genéticos, busca tabu) são comumente utilizados na prática para lidar com instâncias realistas do problema.

Demonstração de solução do PCV por força bruta

Use o código fornecido neste repositório do Github para explorar a solução por força bruta dos problema do caixeiro viajante.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, exploramos os princípios fundamentais dos problemas e algoritmos de otimização. Discutimos tanto métodos exatos quanto métodos aproximados, conhecidos como heurísticas, e abordamos conceitos como soluções ótimas e sub ótimas, ou seja, soluções otimizadas.

Reconhecemos que, em cenários práticos, nem sempre buscamos a solução ótima devido ao custo associado à sua obtenção, o qual frequentemente é proibitivo. Em muitos casos, uma solução sub ótima, capaz de atender aos requisitos desejados do problema e alcançada a um custo razoável, revela-se a abordagem mais adequada. Algoritmos genéticos, nesse contexto, destacam-se como escolhas excelentes.

Dessa forma, estamos equipados e equipadas com o conhecimento necessário para iniciar nossa jornada de estudos sobre algoritmos genéticos, compreendendo sua natureza e funcionamento. Agora, gostaríamos de saber a sua opinião sobre o conteúdo: compartilhe seus comentários e dúvidas no Discord, onde estamos disponíveis na comunidade para responder a perguntas, promover networking, fornecer avisos e muito mais. Junte-se a nós!

REFERÊNCIAS

ANDRÉASSON, N.; EVGRAFOV, A.; PATRIKSSON, M. **An Introduction to Optimization: Foundations and Fundamental Algorithms**. [s.l.]. 2005. Disponível em: https://www.math.chalmers.se/Math/Grundutb/CTH/tma947/0405/kompendium_sub.pdf. Acesso em: 25 mar. 2024.

BOYD, S.; VANDENBERGHE, L. **Convex Optimization**. Nova York: Cambridge University Press, 2009. Disponível em: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf. Acesso em: 25 mar. 2024.

LUU, Q.; Traveling Salesman Problem: Exact Solutions vs. Heuristic vs. Approximation Algorithms. **Baeldung**. [s.l]. 2024. Disponível em: https://www.baeldung.com/cs/tsp-exact-solutions-vs-heuristic-vs-approximation-algorithms#:~:text=Lin%2DKernighan%20is%20a%20local,a%20local%20minimum%20is%20reached. Acesso em: 25 mar. 2024. SÉRGIO POLIMANTE

POLIAK, B. **Introduction to Optimization**. Nova York: Optimization Software, INC. Publications Division, New York, 2010. Disponível em: https://www.researchgate.net/publication/376134695_Evolucao_multiobjetivo_de_trajetorias_como_multiplas_curvas_de_Bezier_para_VANTs?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoicHJvZmlsZSIsInByZXZpb3VzUGFnZSI6ImhvbWUiLCJwb3NpdGlvbiI6InBhZ2VDb250ZW50In19. Acesso em: 25 mar. 2024.

POLIMANTE, S.; PRATI, R.; KLEINSCHMIDT, J.H. **Otimização Multiobjetivo de Trajetórias de VANTs Utilizando Curvas de Bézier e Algoritmos Genéticos**. XIV Brazilian Congress of Computational Intelligence. Belém, 2019. Disponível em: https://www.researchgate.net/publication/337051141_Otimizacao_Multiobjetivo_de_Trajetorias_de_VANTs_Utilizando_Curvas_de_Bezier_e_Algoritmos_Geneticos?_sg%5B0%5D=lOnIBaWaTyXu8LlN_nyhu_lRu8blVBoFeIg7r0WVCl7S249KDxDz5Jqrzwkb5ZanboSBzzOjGkURebtZD1uQtAEu0SyCSzncMvp5Zc8j.-bZRlMMHDrXhoXsA-ONIZhU4WjDe99uSD_C_scN2bBsbwfUMBqx0CEjRoXR6wOsVLc07vbSatcdZBqw0jnfXvg&_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoicHJvZmlsZSIsInBvc2l0aW9uIjoicGFnZUNvbnRlbnQifX0. Acesso em: 25 mar. 2024.

SÉRGIO POLIMANTE. sergiopolimante / tsp. **GitHub**. [s.l], [s.d]. Disponível em: https://github.com/sergiopolimante/tsp/blob/main/TSP%20brute%20force%20demonstration.ipynb. Acesso em: 25 mar. 2024.

PALAVRAS-CHAVE

Algoritmos Genéticos. Otimização. Programação Linear.