

ONE PIZZA

IS ALL YOU NEED

One Pizza Problem





A NOSSA ABORDAGEM DO PROBLEMA

A abordagem pela qual optamos consiste em criar um dicionário para cada cliente (com as suas preferências), uma lista de clientes (com esse conjunto de dicionários) e uma lista de todos os ingredientes. Para gerar vizinhos optamos por fazer de forma aleatória e avaliando a qualidade das seleções com base nas preferências dos clientes. Para algoritmos de otimização aplicamos **hill climbing**, **tabu search**, **algoritmos genéticos** e **simulated annealing**. Por fim, para avaliar a qualidade das soluções, contamos o número de clientes satisfeitos com a seleção de ingredientes e retornamos assim uma pontuação indicativa da qualidade da solução.



A NOSSA ABORDAGEM

Primeiro é
escolhido o
ficheiro a
ser utilizado

Depois é
escolhido o
algoritmo

No final retorna a
solução, a sua pontuação
e o tempo demorado



main.py



menu.py



parse.py



menu.py



algorithms.py



score.py

neste ficheiro são definidos
métodos para mutação,
crossover, seleção de pais e
seleção de filhos



genetic.py





ABORDAGEM

01 parse.py

Lê o ficheiro escolhido. Para cada cliente, cria um **dicionário** com as chaves 'likes' e 'dislikes', que armazenam os ingredientes que o cliente gosta e não gosta. No final, retorna a **lista de dicionários**, 'clients', e a **lista** de todos os ingredientes.

Representa um problema de otimização. Armazena uma lista de clientes e uma lista de ingredientes, fornece métodos para manipular uma lista booleana de ingredientes selecionados, **gera vizinhos aleatórios (adicionando ou removendo ingredientes)** e avalia a qualidade de uma seleção de ingredientes com base nas preferências dos clientes.

02 problem.py

03 algorithm.py

Define diferentes algoritmos de otimização, incluindo **hill climbing**, **tabu search**, **algoritmos genéticos com seleção por torneio e roleta**, e **simulated annealing**. Estes iteram sobre possíveis soluções, atualizando-as de acordo com os critérios de cada algoritmo, até que um critério de paragem seja atendido ou um número máximo de iterações seja alcançado. Os algoritmos visam encontrar a melhor solução possível para o problema, utilizando diferentes estratégias de exploração do espaço de solução.

04 score.py

Avalia a qualidade de uma **solução**. A função `evaluate` avalia uma solução representada como um conjunto de ingredientes, contando o número de clientes satisfeitos com essa solução. A função `evaluate_bool` avalia uma solução representada como uma lista booleana de ingredientes selecionados, utilizando um dicionário para mapear os ingredientes na lista booleana. Ambos os métodos retornam uma pontuação que indica o quão boa é a solução.



ALGORITMOS IMPLEMENTADOS

HILL CLIMBING


Heurística: A cada iteração, o algoritmo Hill Climbing realiza pequenas modificações na solução atual, movendo-se em direção a uma solução que maximize a função de avaliação.

Abordagem: Este algoritmo implementa uma busca local, onde seleciona sempre uma solução vizinha melhor do que a atual, sem considerar soluções que possam ser piores. Isso é feito iterativamente, com o algoritmo atualizando continuamente a solução para a melhor vizinha encontrada até que não haja mais melhorias ou o número máximo de iterações seja atingido.

TABU SEARCH

Heurística: Evita ciclos repetitivos na busca, mantendo uma lista tabu de movimentos recentemente explorados para evitar revisitar estados anteriores.

Abordagem: Explora o espaço de solução de forma mais diversificada, permitindo movimentos que não levam necessariamente a uma solução melhor.





ALGORITMOS IMPLEMENTADOS

SIMULATED ANNEALING


Heurística: Inspirado no processo físico de recozimento de metais, onde a estrutura cristalina é otimizada por meio de resfriamento lento, permitindo que o material atinja um estado de baixa energia. Também usamos reheating um processo que pontualmente aumenta a temperatura para poder escapar de local optima.

Abordagem: Começa com uma alta temperatura que permite a aceitação de soluções piores com uma certa probabilidade, que diminui ao longo do tempo, permitindo que o algoritmo escape de mínimos locais e explore o espaço de solução de forma mais ampla.

GENETIC ALGORITHMS

Heurística: Utiliza conceitos inspirados na evolução biológica, como seleção natural, crossover e mutação, para criar novas soluções a partir de soluções existentes.

Abordagem: Mantém uma população de soluções e evolui ao longo do tempo, selecionando as melhores soluções para reprodução e aplicando operadores genéticos para criar novas soluções.





ALGORITMOS GENÉTICOS

GENETIC ALGORITHM – TOURNAMENT

Seleciona pais através de torneios na população, onde os mais aptos são escolhidos. O processo de torneio é repetido para selecionar dois pais para o crossover. Após o crossover e mutação, os filhos substituem soluções mais fracas. Eficaz para diversidade e evita convergência prematura.

GENETIC ALGORITHM – ROULETTE

A seleção dos pais é proporcional à sua aptidão na população. Indivíduos mais aptos têm maior probabilidade de serem selecionados. Isso leva a uma frequência maior de reprodução dos indivíduos com melhores soluções.

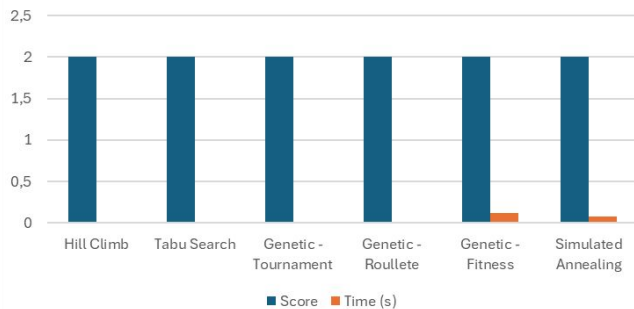
GENETIC ALGORITHM – FITNESS

Seleciona pais com base na aptidão, priorizando os melhores indivíduos. Eles são escolhidos de acordo com a ordem decrescente de sua aptidão. Isso assegura que os mais aptos tenham maior probabilidade de serem selecionados. Contribui para a próxima geração com maior chance de sucesso.

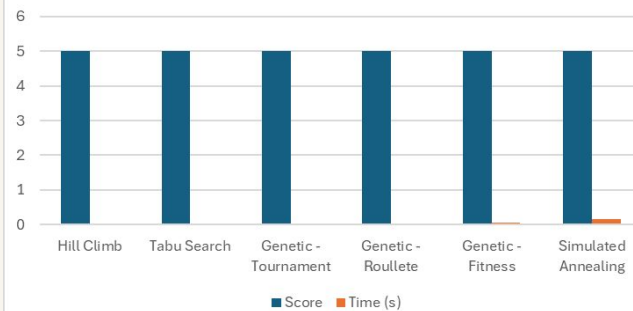


RESULTADOS OBTIDOS

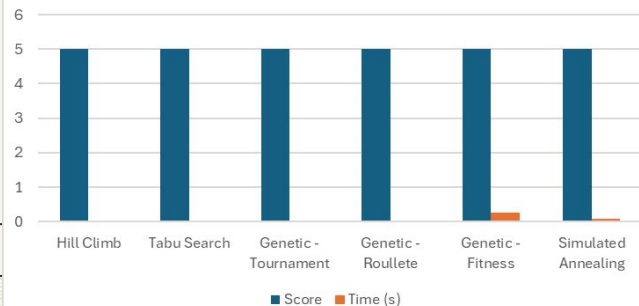
An example



Basic



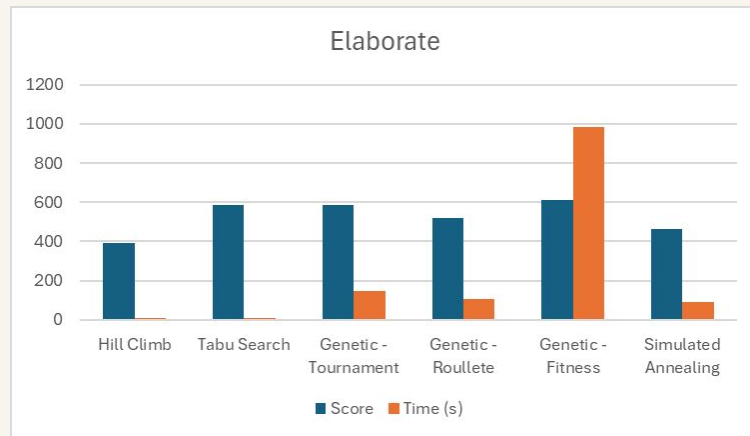
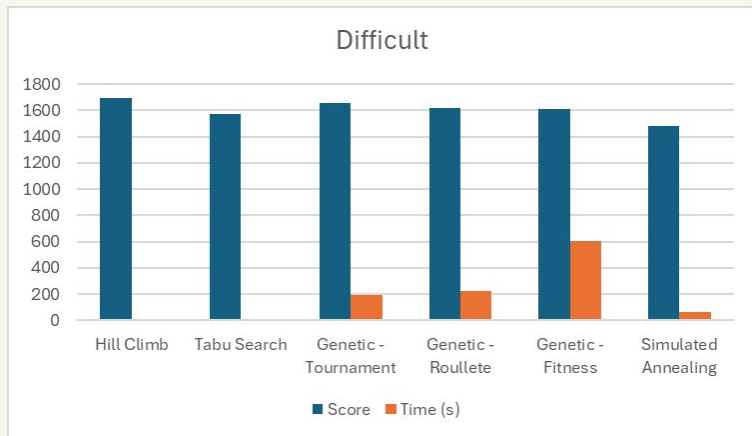
Coarse



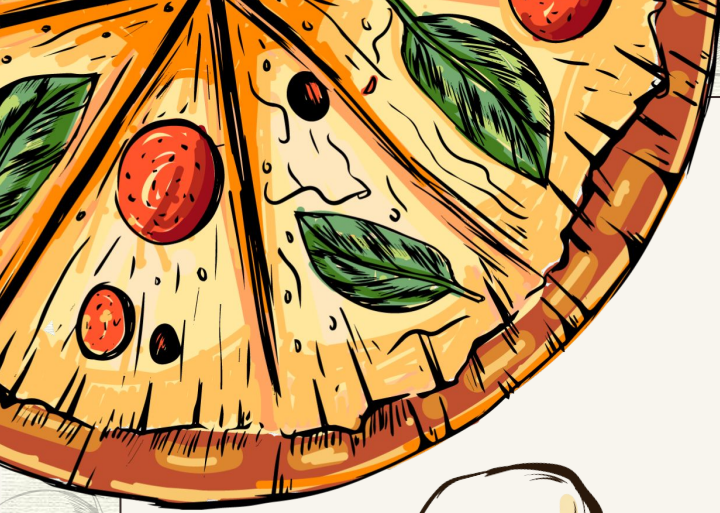
Como é observável pelos dados obtidos, em ambos os 3 menores ficheiros, a **pontuação** obtida pela solução de cada algoritmo foi **constante**. No entanto, verificou-se que para todos estes ficheiros, os algoritmos genético com seleção por aptidão e o simulated annealing foram mais demorados.



RESULTADOS OBTIDOS



Em ambos os dois maiores ficheiros, a mesma coisa não se verificou, tendo já havido **diferenças na qualidade das soluções** dependendo do algoritmo. Quanto ao tempo de execução, os algoritmos genéticos e simulated annealing foram consideravelmente mais demorados.



CONCLUSÃO

Com este trabalho, aplicamos e aprofundamos o nosso conhecimento sobre algoritmos de otimização, tendo assim adquirido uma compreensão mais profunda sobre como resolver problemas complexos de forma eficiente.

Ao abordar o problema 'One Pizza', tentamos solucionar da melhor forma o desafio, utilizando uma variedade de técnicas de otimização, incluindo pesquisa local, metaheurísticas e algoritmos genéticos, explorando as suas vantagens e limitações em diferentes contextos.



REFERÊNCIAS E MATERIAIS CONSULTADOS



Solving-One-Pizza-Practice-Problem-Hashcode-2022-Using-Java

<https://github.com/GasbaouiMohammedAlAmin/Solving-One-Pizza-Practice-Problem-Hashcode-2022-Using-Java>



"One Pizza" Practice Problem Solution | Google Hash Code Competition 2022

https://www.youtube.com/watch?v=A_F4xmLQiFjo



Google Hashcode 2022 Practice Round "One pizza" Discussion

<https://codeforces.com/blog/entry/99020>



Solving One Pizza Practice Problem Google Competition Hash Code 2022 JAVA

https://www.youtube.com/watch?v=V_8IDV97-wuo

