Inteligência Artificial - Projeto 1

Delivery Scheduling (3C)

João Guedes (up202108711)
João Sousa (up202106996)
Armando Martins (up201603566)

Specification

O objetivo do *Delivery Scheduling* é otimizar a entrega de três tipos de encomenda: frágil, normal, e urgente. Cada encomenda tem de ser transportada do ponto de partida (0, 0) e chegar ao destino considerando três critérios:

- A minimização de dano causado às encomendas frágeis;
- A minimização dos custos de viagem (cada quilômetro tem custo fixo);
- Aderir às restrições das encomendas urgentes - existe uma penalização por cada minuto de atraso com um custo fixo.

As encomendas frágeis tem a chance de se danificarem durante o transporte; as encomendas normais não existe risco de dano; e as encomendas urgentes recebem uma penalização se forem entregues com atraso.

Objetivo final: optimizar a ordem de entrega para minimizar o custo total usando diferentes algoritmos.

Hard **Constraints:**

1. You only have one vehicle available.
2. The delivery locations are specified by their coordinates.
3. Routes between all delivery coordinates are available.
4. The driver drives at 60km per hour and takes 0 seconds to deliver the goods.
5. The cost per km is C=0.3.

Optimization Problem (genetic algorithm)

Solution Representation:

Para representar a solução para este problema vamos usar uma "*Chromosome Structure*", sendo neste caso o cromossoma a ordem de entrega de todas as encomendas e cada gene representa uma encomenda (objeto *Package*).

Path = [Package], onde *Package = [(X,Y), type]*

Neighborhood/Mutation Functions:

Mutação por troca - consiste na seleção aleatória de dois genes dentro do cromossoma e na subsequente troca das suas posições (simula uma alteração na ordem de entrega que permite explorar outras opções que podem minimizar o custo total). Mutação por inversão - consiste em escolher aleatoriamente um subconjunto do cromossoma e inverter a ordem dos genes (operação semelhante à mutação por troca mas mais "drástica").

Crossover Functions:

Order Crossover (OX) - Um segmento de um *parent* é copiado para um *child* (descendente), e a informação restante é preenchida a partir do outro *parent*, respeitando a ordem das encomendas.

Partially Mapped Crossover (PMX) - Dois pontos de cruzamento são selecionados, e os segmentos entre eles nos *parents* são trocados para produzir *offspring* (descendência), resolvendo quaisquer conflitos para manter sequências válidas.

Evaluation Function:

A função de avaliação vai medir o quão boa uma solução é dependendo da: distância total percorrida, custo de danos de encomendas frágeis e penalização sobre encomendas urgentes tudo somado num custo total.

CustoTotal = DistanciaTotal*0.3 + ValorDanoTotal + TempoAtrasoTotal*0.3

Work Already Implemented

Programming Language: Python, com código implementado para a visualização de uma interface gráfica para observação das coordenadas (*matplotlib.pyplot*), cálculo do custo total e implementação de algoritmo greedy, hill climbing, simulated annealing, genetic algorithm e tabu search.

Development Environment: VSCode, GIT.

Data Structure: Package class, Path class (lista de packages que contém as coordenadas e outros atributos e um custo).

File Structure: Organização do projeto com o GitHub.

Approach

Começamos por criar uma população com o número de encomendas e tamanho do mapa, esta pode ser aleatória (o utilizador escolhe o número de encomendas e tamanho do mapa) ou pré-definida. Damos ainda opção ao utilizador de ver estatísticas referentes a cada um dos algoritmos, por exemplo, no *genetic algorithm* ver em que geração foi encontrada uma nova melhor solução.

Heurísticas: encontrar a encomenda mais próxima da nossa posição atual; explorar a melhor solução em soluções vizinhas; introduzir aleatoriedade; seleção natural e *crossover* genético; registrar soluções visitadas para evitar visitá-las outra vez.

Para avaliar as soluções criamos uma função na classe *Path*, de forma a conseguirmos calcular facilmente o custo de cada *Path*. Nessa função temos em conta: a distância entre todas as encomendas, o tempo de entrega (para entregas urgentes) e a chance de partir (para encomendas frágeis).

No final de cada algoritmo obtemos: uma interface gráfica representando o melhor caminho escolhido para as entregas das encomendas e uma tabela com os resultados da melhor solução com o menor custo total.

Implemented Algorithms

Hill Climbing:

O *Hill Climbing* é usado como uma etapa inicial para os outros algoritmos. Procura uma solução localmente ótima movendo-se para vizinhos com custo menor e começa com uma sequência gerada por um algoritmo *Greedy* que liga os pacotes minimizando a distância entre os mesmos.

Simulated Annealing:

Após o *Hill Climbing*, utilizamos o *Simulated Annealing* que é aplicado para refinar a solução. Este algoritmo introduz um elemento de aleatoriedade controlado por um parâmetro de temperatura e um *cooling rate*. A temperatura inicial controla a probabilidade de aceitar soluções melhores ou piores do que a atual, com a probabilidade de aceitar soluções piores diminuindo com o tempo de acordo com o *cooling rate*. Este algoritmo permite com que não fiquemos presos num *local best*, pesquisando outras soluções que, em primeira instância, sejam piores, mas que se tornem melhores. Fizemos também uma pequena alteração ao algoritmo base, guardando numa lista os melhores resultados encontrados ao longo do *Simulated Annealing*, para no final de toda a pesquisa ser escolhido o melhor *path*.

Implemented Algorithms

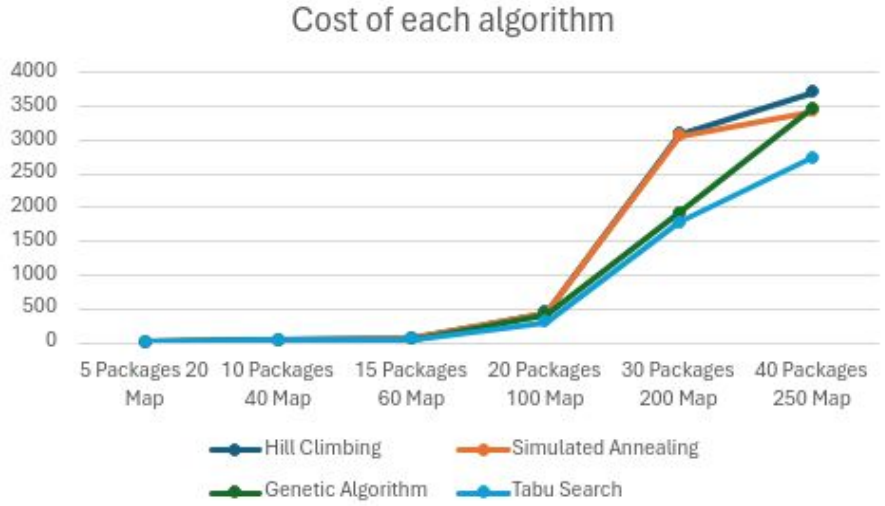
Genetic Algorithm:

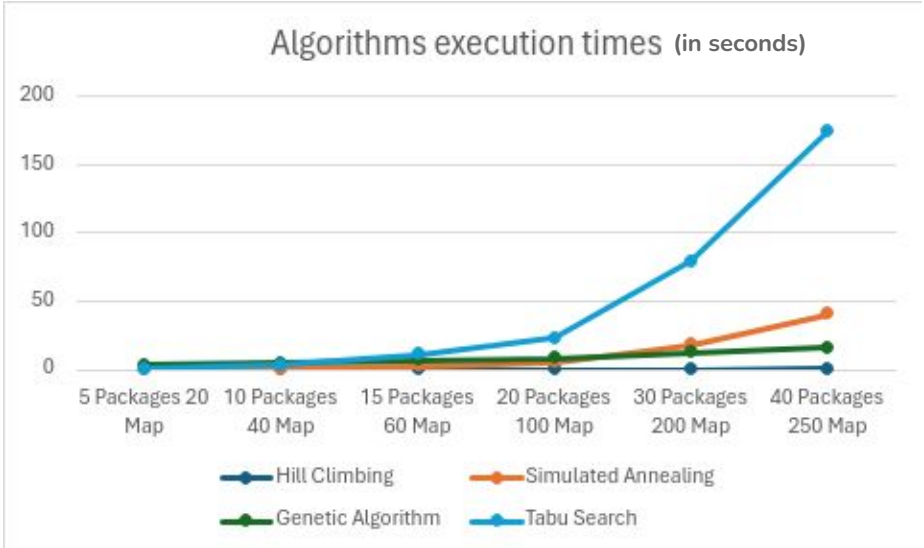
O *Genetic Algorithm* é usado para gerar uma nova população de soluções candidatas. O algoritmo usa seleção (com base no *fitness*/custo total), *crossover* (*ordered*) e mutação para criar novas soluções a partir de uma população base "*random*" onde colocamos a solução do *Hill Climbing* de modo a encontrar melhores resultados.

Tabu Search:

A *Tabu Search* é uma meta-heurística que mantém uma lista de movimentos tabu para evitar uma nova pesquisa em soluções recentes. Essa lista é ajustada dinamicamente para evitar estagnação. A diversificação é utilizada para introduzir movimentos aleatórios em impasses. O *tenure* da *tabu list* é ajustado conforme o progresso da pesquisa. Usamos um *aspiration criteria* de modo a permitir aceitar soluções *tabu* se estas forem melhores que a melhor solução atual.

Experimental Results





Conclusions

Os resultados demonstram que todos os algoritmos apresentam um aumento no custo à medida que o problema se torna mais complexo, com o *Tabu Search* mostrando-se consistentemente mais eficiente em termos de custo, especialmente em cenários de maior escala. Em contrapartida, o *Hill Climbing* destaca-se pelo seu menor tempo de execução, embora com custos mais elevados, indicando uma rápida convergência para soluções subótimas. O *Simulated Annealing* e o *Genetic Algorithm* apresentam um equilíbrio entre custo e tempo de execução, com o *Simulated Annealing* a mostrar ligeira vantagem em eficiência de custo para problemas de maior dimensão. O tempo de execução do *Tabu Search*, apesar de ser o mais elevado, justifica-se pelos resultados substancialmente melhores em termos de custo, particularmente em instâncias maiores.

Estas observações sugerem que a escolha do algoritmo deve ser baseada numa avaliação cuidadosa entre a qualidade da solução desejada e os recursos computacionais disponíveis, com o *Tabu Search* a emergir como uma opção robusta para otimizações de larga escala, apesar do seu maior requisito de tempo.

Related Work

Zhuagenborn, 2021, Huawei Delivery Optimization,
https://github.com/Zhuagenborn/Huawei-Delivery-Optimization/blob/main/Huawei%20Delivery%20Optimization%20Competition.pdf

Baeldung, 2023, Partially Mapped Crossover in Genetic Algorithms,
https://www.baeldung.com/cs/ga-pmx-operator

What tools and techniques optimize delivery routing and scheduling?,
https://www.linkedin.com/advice/3/what-tools-techniques-optimize-delivery-routing#:~:text=Tools%20and%20techniques%20that%20optimize,efficiency%2C%20and%20enhance%20customer%20satisfaction.Message%20@despair

Subham Datta, 2024, Genetic Algorithms: Order One Crossover,
https://www.baeldung.com/cs/ga-order-one-crossover