**Relatório INF1771 – Inteligência Artificial**

Fernando Homem, João Garcia, Júlia Aleixo, Rodrigo Leite

**Trabalho 1 - Problema do Caixeiro Viajante**

1. **Introdução**

O problema do caixeiro viajante consiste em encontrar o menor caminho para percorrer uma coleção de cidades e retornar à cidade inicial, visitando cada cidade somente uma vez.

O problema do caixeiro viajante é NP-complexo, e um dos problemas de otimização mais estudados.

Suas aplicações abrangem logística, astronomia, fabricação de microchips e sequenciamento de DNA (se modificado ligeiramente).

1. **Definição do Problema**
2. **Representação das Cidades**

Decidimos representar as cidades pelos seus números em um array, e as distâncias entre elas em uma matriz (que, em Python, é uma lista de listas). Todas as soluções possíveis tem a mesma cidade inicial, e, apesar de entrar no calculo da distancia do percurso, a volta à cidade inicial não é incluída no array da solução.

1. **Geração de vizinhança**

Função Swap: A partir de um vizinho, troca duas cidades adjacentes de lugar para gerar um novo vizinho. Para criar uma vizinhança inteira, faremos isso ao longo do vetor, trocando suas posições 1 e 2, 2 e 3, e assim por diante até n-1 até n. Não mudamos a posição inicial pois ele precisa ser sempre o mesmo.

1. **Avaliação dos vizinhos**

A função que avalia os vizinhos calcula a distância de seus percursos de forma circular (depois da ultima cidade ele volta para a cidade inicial para também incluir essa distância) e seleciona o vizinho com a menor distância de percurso.

1. **Busca**

Hill Climbing:

Simulated Annealing:

1. **Metodologia**

Decidimos optar por uma busca local, a hill climbing, e uma meta-heurística, simulated annealing. Assim, poderíamos comparar dois tipos de busca que seriam mais prováveis de nos darem resultados satisfatórios, uma vez que as buscas cegas não contém informação alguma sobre o problema, é feito na força bruta, e as buscas heurísticas não se adequam a esse tipo de problema.

* **Hill Climbing**

Essa busca local começa com uma possível solução, e a partir dela uma vizinhança é gerada. Os vizinhos gerados são então avaliados por uma função que calcula a distância do percurso de sua solução. O vizinho com a menor função de avaliação é escolhido e comparado com a solução inicial: caso tenha uma distância menor, ele substitui a solução inicial e o processo se repete, caso contrario, a busca termina. O problema com esse método são os máximos locais, dos quais o algoritmo não escapa.

* **Simulated Annealing**

Essa busca meta-heurística se baseia em probabilidade, e é fundamentada em uma analogia com a termodinâmica. Annealing é um processo térmico usado na metalurgia para obter estados de baixa energia em um sólido. Esse algoritmo funciona de forma análoga a esse processo: ele substitui a solução atual por uma solução de sua vizinhança, escolhida de acordo com uma função objetivo e de uma variável T (de temperatura, que no algoritmo simboliza o tempo). Quanto maior a T, maior a componente aleatória que será incluída na próxima solução escolhida. Conforme a progressão do algoritmo, o valor de T diminui, e mais perto ele está da solução ótima. A vantagem desse algoritmo é que, como ele permite algumas “pioras” na escolha de soluções, ótimos locais são evitados.

1. **Resultados**

* **Distância**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hill Climbing | Hill Climbing Alterado | Simulated Annealing |
| 17 cidades |  |  |  |
| 21 cidades |  |  |  |
| 24 cidades |  |  |  |
| 48 cidades |  |  |  |
| 175 cidades |  |  |  |

* **Tempo**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hill Climbing | Hill Climbing Alterado | Simulated Annealing |
| 17 cidades |  |  |  |
| 21 cidades |  |  |  |
| 24 cidades |  |  |  |
| 48 cidades |  |  |  |
| 175 cidades |  |  |  |

1. **Conclusão**