Relatório Atividade 1: Subida da Encosta Para Caixeiro Viajante

Nome: Lucas de Araújo Torres e Francisco Pedro Mota Braga Carneiro
Matrícula: 557156, 501039
Professor: Samy Soares Passos de Sá

Resumo

Este relatório tem o objetivo de explicar e especificar o trabalho 1 da cadeira de Inteligência Artificial. O trabalho consiste em escrever o algoritmo da subida da encosta para o problema do caixeiro viajante. O problema consiste em, dado vários pontos em um plano, sendo esses pontos cidades, criar um circuito que sai de uma cidade inicial, passa por todas as cidades e volta para a primeira cidade. A distância de cada par de cidades é dada pela distância Euclidiana. A ideia é sempre procurar um valor ótimo para cada heurística.

No trabalho, foram feitas 8 variações do algoritmo para a resolução do problema, sendo cada uma delas executadas 30 vezes no programa e tendo seus resultados analisados. A implementação dos códigos foi feita pela linguagem Python, sendo necessário ter ela instalada na máquina para poder executar os códigos. Ademais, neste documento será mostrado um pouco do funcionamento das variações e de seus respectivos resultados.

Descrição da Atividade

Explicando a estrutura do projeto, há um arquivo help.txt que especifica bem como rodar o código. O arquivo entries.txt serve como entrada do problema, ou seja, as posições das cidades. O arquivo hill_climb_functions.py possui todas as funções necessárias dos algoritmos, com todas as variações descritas da atividade. O arquivo main.py é o código principal que deve ser rodado, na qual reproduz todos os custos das 8 variações, imprimindo no terminal e salvando em forma de tabela em um arquivo chamado table.txt. Toda a descrição do trabalho está no PDF original do professor.

Todos os algoritmos sem randomização, ou seja, sem geração aleatória, tiveram permutações feitas da esquerda para a direita, para ambos operadores. Com a randomização, foram salvos estados em cada loop interno que já foram verificados. Isso vai servir para o algoritmo não ficar preso dentro de um loop cometendo sempre as mesmas repetições. Foram deixados alguns comentários pelo código para facilitar a leitura do mesmo. A ideia é preservar o algoritmo de subida da encosta, que é bem semelhante ao algoritmo guloso, porém, não é a mesma coisa. Tem a mesma ideia de atacar mínimos ou máximos locais para se chegar a um global. Por isso a necessidade de vários testes e diversas formas de implementar, pois não é garantido de chegar a um estado ótimo global. Sobre o estado inicial 2, na qual algumas variações usaram ele, ele foi aplicado no valor de entrada do arquivo txt, ou seja, foi feito uma permutação aleatória desse primeiro estado.

Deixamos o código com leitura em inglês para futuramente publicarmos como um repositório aberto à comunidade.

Resultados

Como dito anteriormente, os resultados serão impressos tanto no terminal, como no arquivo table.txt. Cada coluna mostra uma variação e cada linha uma execução de sua variação. São no total 30 linhas, para cada execução e 8 colunas para cada variação. Segue um exemplo de um resultado obtido para a seguinte entrada:

Entrada
0.0 1.0 2.0 0.0 1.0 1.0 3.0 4.0 1.0
0.0 2.0 0.0 2.0 1.0 3.0 2.0 5.0 0.0

	variation 1	variation 2	variation 3	variation 4	variation 5	variation 6	variation 7	variation 8
0	19.0578	17.6543	15.8324	16.4182	15.8324	16.4182	15.8324	16.4182
1	19.0578	18.0105	15.8324	16.8836	15.8324	17.2268	15.8324	16.4182
2	19.0578	18.0552	15.8324	17.2466	15.8324	17.4247	15.8324	16.4182
3	19.0578	18.1197	15.8324	17.6120	16.5963	17.8126	15.8324	16.5963
4	19.0578	18.3046	15.8324	17.6543	17.0040	18.0105	15.8324	16.5963
5	19.0578	18.4182	15.8324	17.8126	17.4694	18.0105	15.8324	17.1821
6	19.0578	18.4182	15.8324	17.8324	17.6120	18.2401	15.8324	17.4247
7	19.0578	18.4629	15.8324	18.1978	17.7586	18.2401	15.8324	17.6543
8	19.0578	18.4827	15.8324	18.4827	17.7901	18.3444	15.8324	17.8126
9	19.0578	18.6608	15.8324	18.5963	17.8324	18.4182	15.8324	17.8324
10	19.0578	18.8904	15.8324	18.6299	17.8324	18.4759	15.8324	17.8324
11	19.0578	18.8904	15.8324	18.6608	17.8324	18.8191	15.8324	17.8324
12	19.0578	18.9617	15.8324	18.6608	18.1197	18.8389	15.8324	18.0552
13	19.0578	19.0487	15.8324	18.7055	18.1978	18.9127	15.8324	18.1488
14	19.0578	19.2268	15.8324	18.8481	18.2978	19.0487	15.8324	18.3269
15	19.0578	19.2559	15.8324	18.9302	18.3046	19.0487	15.8324	18.4182
16	19.0578	19.7281	15.8324	19.0487	18.4629	19.1530	15.8324	18.4759
17	19.0578	19.8126	15.8324	19.0487	18.5963	19.2175	15.8324	18.6317
18	19.0578	19.8324	15.8324	19.0578	18.5963	19.7281	15.8324	18.7346
19	19.0578	20.1978	15.8324	19.0578	18.6317	19.9707	15.8324	18.8191
20	19.0578	20.2333	15.8324	19.0578	18.7055	20.0842	15.8324	18.9749
21	19.0578	20.3798	15.8324	19.0578	18.7055	20.1197	15.8324	19.0487
22	19.0578	20.3891	15.8324	19.2466	18.9617	20.1488	15.8324	19.0487
23	19.0578	20.4536	15.8324	19.3111	19.0685	20.1978	15.8324	19.4630
24	19.0578	20.4759	15.8324	20.0842	19.2268	20.2848	15.8324	19.8678
25	19.0578	20.6590	15.8324	20.2110	19.2359	20.4720	15.8324	19.8951
26	19.0578	20.7346	15.8324	20.2294	19.4247	20.6700	16.5963	20.6210
27	19.0578	20.7743	15.8324	20.6700	19.4694	20.7991	16.5963	20.8482
28	19.0578	20.8482	15.8324	21.2820	20.3230	21.0883	16.5963	20.9772
29	19.0578	21.3093	15.8324	21.3558	20.4920	21.2688	16.5963	21.1821

Arredondamos o ponto flutuante para apenas 4 casas decimais para uma maior facilidade de leitura. Veja que nas variações 1 e 3, os resultados obtidos são os mesmos, pois não há uma geração aleatória entre os estados. Ou seja, o algoritmo permuta da esquerda para a direita. Entretanto, nas outras variações, há pelo menos uma geração aleatória de estado. Seja pelo estado inicial 2 ou com a inclusão da randomização. Note que

o possível resultado ótimo desse experimento foi 15.8324, pois foi o menor encontrado. A variação 3, mesmo sem ter aleatoriedade, conseguiu atingir esse menor custo que as outras variações. A variação 7 também se manteve bem estável, pois só existem randomização em seu estado inicial, e é usado o operador 2. Podemos observar também que, o operador 2 se torna mais eficiente que o operador 1, em todas as variações semelhantes.

Outro ponto importante é que ordenamos os custos em ordem crescente para melhorar a visualização. Isso indica que esses resultados não foram obtidos nessa ordem. Para cada função, foi criado um output para mostrar o funcionamento de cada estado sendo encontrado.

Conclusão

Com isso, concluímos nosso relatório do trabalho. É bom fazer mais testes para verificar melhor o funcionamento das variações para diferentes tipos de entradas.

Alteramos a entrega desse trabalho. Melhoramos apenas nossa forma de fazer os casos com randomização, onde geramos uma lista de permutações dos índices, embaralhada, e percorremos para ver se existe algum estado melhor que o atual. Dessa forma, não ocorrerá problemas do algoritmo aleatório sempre estar escolhendo os mesmo índices e ficar demorando no loop.