Trabalho de Inteligência Artificial parte 1

Relatório

Elementos do grupo:

- Nuno Filipe Araújo Gonçalves nº mecanográfico:201402720

Índice:

1. Gerador de testes

O algoritmo utilizado para gerar universos e pontos usamos nos testes dos algoritmos seguidamente apresentados, situa-se na classe Generator. Ele utiliza um valor n e um valor m, introduzidos pelo utilizador, que representam respetivamente o numero de pontos a serem gerados e o limite das suas coordenadas. O algoritmo utilizada uma classe do java.util (a classe Random) para gerar n*2 inteiros, através da condição random(high-low) + low, assegura-se que as coordenadas obedecem aos limites impostos e serve-se de um ciclo que se assegura que cada geração de pontos não é repetida. Assim sendo não se trata de um algoritmo muito eficiente sendo a sua complexidade da ordem $O(n^2)$.

2. Classes auxiliares

2.1 Classe Points

Para alem da classe Generator o projeto serve-se de algumas estruturas de dados auxiliares para a criação dos algoritmos de solução. A primeira aqui descrita é a classe Points que, tal como o nome indica, é utilizada para armazenar as coordenadas dos nossos pontos. Para além do construtor base e dos básicos métodos getters, setters e toString, tem também um método que calcula o quadrado da distancia euclidiana chamado edistance e um método crossProduct que calcula o produto vetorial entre 2 vetores com 3 pontos onde é feita uma deslocação da origem no ponto base.

2.2 Classe Ramos

A classe ramos guarda tuplos de duas estruturas de dados do tipo Points, que representam um segmento de reta com esses 2 pontos como vértices. Para alem dos métodos getters, setters e toString básicos, contem um método chamado internProduct que calcula o produto interno entre dois ramos, um método belongs que retorna um valor de verdade sobre a condição de um ponto pertencer ramo/segmento de reta e um método interset que retorna um valor de verdade sobre a condição de dois segmentos de reta se intersetarem. Este ultimo método utiliza caraterísticas geométricas sobre produto vetorial entre segmentos para verificar se colidem e no caso de um estar contido no outro, através do calculo do produto interno apenas aceita casos em que os vetores tenham direções opostas. A razão pela qual esta condição existe será explicada mais á frente neste relatório.

2.3 Classe Grafo

A classe grafo é a estrutura de dados utilizada para guardar o ciclo de pontos que constrói, na pratica um ciclo de Hamilton. A classe contem apenas 2 atributos, um inteiro size responsável pelo tamanho do grafo e uma linkedList, que é uma lista ligada utilizada para guardar os pontos pertencentes ao grafo. A classe contem também um método getsize, que retorna o tamanho do grafo, um método addPoint, que adiciona um ponto no fim da lista, e um método getPoint, que dado um inteiro retorna o ponto com esse índice na lista.

3. Algoritmos

A função main da classe MainIa é a funçao responsável pela leitura dos dados criados na classe Generator e pela chamada de todas as função que utilizam os algoritmos de pesquisa que formam ciclos de Hamilton.

3.1 Algoritmo All Random

A função all_random é a função responsável pelo metodo all random na nossa classe MainIa. Ela é chamada com um array de Points que contem todos os pontos que o ciclo de Hamilton deve de incluir e retorna um grafo. Através da classe Random do java, ela previamente cria um grafo vazio e escolhe um ponto do array ao calhas para servir de cabeça adicionando-o ao grafo. Depois troca a posição do ponto escolhido com a posição do ultimo ponto no array e avança para um ciclo for que aplica o mesmo método feito para escolher o primeiro ponto, ou seja escolhe um ponto ao calhas e troca com o ponto da posição n-i (com n = tamanho do array e i = à iteração no ciclo for) do array. A condição r.nextInt(n-1) certifica-se que o inteiro gerado é majorado pelo menor índice dum ponto já adicionado. Este algortimo só percorre o array de pontos uma vez para gerar o grafo portanto a sua complexidade é de O(n).O grafo formado por este algoritmo não é solução para o problema do caixeiro viajante.

3.2 Algoritmo Nearest Neighbour

A função nearest_neighbour da classe MainIa é a responsável pela heuristica do vizinho mais proximo primeiro. Ela é chamada com um array de Points que contem todos os pontos que o ciclo de Hamilton deve incluir e retorna um grafo. O primeiro ponto a ser inserido no grafo, tal como no algoritmo anterior é escolhido ao calhas. Para a escolha dos pontos seguintes o algoritmo serve-se de auxilio de um array visit de inteiros, com o mesmo tamanho que o array de

pontos
e os indices correspondem aos indices dos pontos no array de pontos e que guarda a informação de que um ponto já foi ou não inserido no grafo, sendo o seu valor 0 se ainda não foi inserido ou 1 se já o tiver sido. Inicia-se entao um ciclo for que percorre todos os pontos ainda não inseridos, calcula a distancia euclidiana desses pontos com a do ultimo inserido e escolhe o com menor distancia para ser o proximo a ser inserido. De notar que para a escolha de um valor minimo para servir de comparação no ciclo, o algoritmos percorre o array visit uma vez até encontrar o primeiro ponto ainda não inserido e utiliza a distancia euclidiana entre esses dois pontos como valor minimo das distancias para comparação. O algoritmo apresenta uma complexidade O(n^2) e não assegura que o grafo formado seja solução para o problema do caixeiro viajante embora seja um grafo muito mais proximo de ser solução do que o grafo formado pelo algoritmo all random.

3.3 Two Exchange

O algoritmo two exchange é chamado na função two_exchange, que contem como argumento um array de ramos que constituem um grafo e retorna uma lista ligada de points que contem os índices de todos os ramos que se intersetam. O algoritmo tem um ciclo que percorre todos os ramos do array e, em cada um deles, calcula, através do método interset contido na classe Ramos, quais dos outros ramos este interseta. Sempre que a condição acontece, o tuplo constituído pelos índices dos dois ramos é guardado na cauda da lista ligada. O algoritmo guarda também a posição do ramo que esta atualmente a ser pesquisado e certifica-se que não é calculada a interseção de um ramo com ele próprio. Como o algoritmo percorre, em cada iteração do seu ciclo o array de ramos, apresenta uma complexidade da ordem O(n^2).

3.3.1 Exchange

A função exchange é uma função que, embora não faça parte do algoritmo two exchange, complementa a utilização do mesmo. A função recebe como argumentos um array de ramos e a posição dos dois ramos, nesse array, que deveram ter os pontos de extremos trocados. Trata-se, portanto, uma função que não retorna nada, ou seja, do tipo void. Para além de trocar o segundo ponto do ramo com índice mais pequeno, com o primeiro ponto do ramo com índice maior, a função inverte também a ordem e direção de todos os ramos que se encontram entre os dois ramos em questão. Este função apresenta um tempo de execução da ordem do O(n).

3.4 Hill Climbing

3.4.1 Best Perimeter

A função que utiliza o hill climbimg como melhoramento iterativo e o melhor perímetro como heurística é a função hsBest da classe MainIa. A função recebe como argumento um grafo que já constitui um ciclo de Hamiltom e retorna um grafo que é solução para o problema do caixeiro viajante. A função cria um array de ramos auxiliar, formado pelos ramos constituintes do grafo recebido, e com esse array inicia um ciclo do while que em cada iteração chama a função two exchange, recebe uma lista dos ramos que se intersetam da mesma e calcula a diferença do perimetro de cada um dos ramos intersetados caso sofram uma exchange. O ciclo chama a função exchange sobre o tuplo de ramos que originem a maior diferença no perímetro do grafo e, depois começa uma nova iteração do seu ciclo. De notar que a condição de saída do ciclo é a de o min<0 isto porque para a condição ser verdadeira o valor de min tem de ser zero, algo que só se verifica quando não houve nenhum calculo da diferença do perímetro (por métodos geométricos é possível demonstrar que a remoção de ramos que se intersetam seguindo as condições do problema reduzem sempre o perímetro do polígono) ou seja, quando a lista retornada por two exchange é vazia (não existem mais ramos que se intersetam).

3.4.2 First neighbour

A função que utiliza o hill climbimg como melhoramento iterativo e o primeiro conjunto de ramos que se intersetam como heurística é a função hsFirst da classe MainIa. A funçao cria, tal como no algoritmo anterior, um array auxiliar constituído pelos ramos do grafo e usa-o para chamar a funçao exchange num ciclo. Como, pela como a funçao two exchange foi construida, retorna uma lista em que a cabeça é o tuplo dos primeiros ramos que se intersetam, o nosso ciclo retira a cabeça da lista retornada por two exchange chama a funçao exchange com argumentos a cabeça da lista e volta a calcular a lista dos ramos intersetados agora com a mudança já presente. O ciclo encontra o seu termio quando a lista retornada por two exchange é vazia. Esta função recebe como argumento um grafo que já constitui um ciclo de Hamiltom e retorna um grafo que é solução para o problema do caixeiro viajante. Este algoritmo para nº de pontos mais elevados é bastante mais lento que o algoritmo que usa como heurística o valor do perímetro.

3.4.3 Random neighbour

A função que utiliza o hill climbimg como melhoramento iterativo e um conjunto de ramos ao calhas que se interseta como heurística é a função hsRandom da classe MainIa. È uma função muito idêntica á funçao hill climbing first neighbour, sendo a única diferenço o facto de que em vez de escolher sempre a cabeça da lista retorna como tuplo a ser usado para sofre exchange, utilizada a classe Random do java para escolher um tuplo ao calhas de todos os tuplos de índices de ramos que se intersetam. Utiliza-se o tamanho da lista tanto como condição de saida do ciclo, como limite majorante dos inteiros gerados ao calhas.

4. Desenho Gráfico

O desenho dos modelos obtidos num gráfico de duas coordenadas é efetuado com o auxilio de uma pagina html e de um script escrito em javascript. No script é utilizada uma ferramenta chamada de d3 que é uma biblioteca de javascript para manipulação de documentos baseados em dados. Para utilizar esta funcionalidade deve se abrir o ficheito Graph.html no browser, inserir um ficheiro .txt com o output da execução da classe MainIa e indicar o limite dos valores dos pontos criados na respetiva caixa de informação. De seguida é só necessário selecionar qual o grafo do algoritmo gerado quer ver. Pode trocar entre vários grafos sem ter de reuniciar a pagina.