Primeiro Trabalho de Inteligência Artificial CC2006

Pedro Miguel Ribeiro Carvalho, up201805068

João Paulo Macedo Sampaio, up201805112

Resumo

Este trabalho foi proposto na cadeira de Inteligência Artificial CC2006 com objetivo de estudar métodos de pesquisa local e pesquisa local estocástica usando um problema de geração de polígonos simples. Este problema tem como base o gerador de polígonos aleatórios (RPG) de Thomas Auer e Martin Held[1], que consiste na geração aleatória de polígonos simples a partir de um conjunto também aleatorio de pontos usando diferentes heuristicas. De referir que polígonos simples não tem arestas que se intersectam a não ser em vértices e importante de elucidar que as arestas dos polígonos devem formar um ciclo de Hamilton, isto é, um ciclo de Hamilton é um percurso fechado que não repete nós e passa em todos os nós. Assim sendo soluções otimas são apenas poligonos que sejam simples. Acima de tudo, neste trabalho para a implementação dos diferentes algoritmos que nos eram pedidos usamos a linguagem Java e a ferramenta Geogebra[2] para os gráficos dos polígonos aqui referidos.

Introdução

O trabalho está dividido principalmente em seis questões. A primeira tem como objetivo a criação aleatória de *n* pontos num plano com coordenadas compreendidas entre *-m e m*, estes parâmetros dados. De seguida, é solicitado que seja determinado um candidato a solução considerando duas alternativas, por um lado, um permutação qualquer dos pontos, ou seja, um polígono formado com um n número de pontos de forma aleatória. Por outro lado, é pedido que se use a heurística “nearest-neighbour first” a partir de um ponto inicial, ponto esse que pode ser aleatoriamente escolhido. Neste contexto em especifico a heurística “nearest-neighbour first” refere-se a visitar o próximo ponto o que estiver mais próximo do atual, mais concretamente esse ponto é obtido calculando a distância euclideana, de ressalvar a importância de no cálculo ser usado o quadrado da distância para evitar erros numéricos. Em seguida, na terceira questão é requerido que se determine a vizinhança para um candidato *s* pela regra “2-exchange”, muito sucintamente esta regra é a troca de dois ramos que se intersectarem, ou seja, se os ramos *{i,j} e {k,l}* se intersectarem eles são removidos e adicionados os seguintes ramos *{i,k}* e *{j,l}*. Assim sendo para descobrir se dois segmentos se intersectavam ou usaríamos um algoritmo de deteçao do seu ponto de interseção, caso exista, ou então um algoritmo para determinar se dois segmentos se intersectam[3].

Consequentemente, é necessário um outro algoritmo para determinar todos os segmentos que intersectam neste caso existe uma variedade de algoritmos desde o mais simples de força-bruta, Shamos e Hoey[4], Bentley-Ottmanm[5], etc. Após obtermos a vizinhança na quarta pergunta é pedido que se aplique melhoramente iterativo mais concretamente o algoritmo Hill Climbing, algoritmo este basicamente parte de uma possível solução e passo a passo calcula um melhor candidato até não encontrar um melhor, tem as vantagens de não precisar de guardar toda a árvore de pesquisa, ou seja, apenas grava o estado atual. Obviamente esta técnica é muito eficiente em termos de mémoria. Porém pode ficar preso num máximo local uma vez que é um algoritmo greedy e apenas analisa o estado seguinte. Nesta questão usamos o Hill Climbing através da análise da vizinhança já calculada. Porém é pedido que se usem quatro diferentes alternativas, primeiramente que se opte por uma heuristica “best-improvement first”, ou seja, que seja escolhido o candidato que reduza mais o perímetro. Em segundo lugar que se opte por uma heurística de “first-improvement”, isto é, que se escolhe o primeiro candidato nessa vizinhança. Em terceiro lugar, optar pelo candidato que tiver menos conflitos de arestas. Por último, que se escolha um qualquer convidado nessa vizinhança. De ressalvar que qualquer que seja o candidato na vizinhança é sempre melhor que o candidato original, isto é, o candidato terá sempre maior perímetro que qualquer candidato da sua vizinhança, isto é garantido pela regra “2-exchange”. Na quinta questão é pedido também que se calcule um candidato a partir de uma vizinhança porém, desta vez será usado Simulated Annealing, este que se assemelha muito ao Hill Climbing mas difere no facto de permtir retrocessos, ou seja, permite que maior grau de probabilidade no inicio da execução do algoritmo se obtenha uma solução pior que a inicial, como forma de tentar evitar máximos locais e ao longo da execução essa probabilidade é cada vez menor. Por questões de tempo não nos foi possível executar a última tarefa assim sendo não estará presente neste relatório nem no ficheiro de código fonte deste trabalho.

Explicação das implementações

Estrutras de dados

Inicialmente é importante explicar as estruturas de dados usadas ao longo do trabalho. O trabalho foi dividido em cinco classes. Point, Map, Intersection, Candidate, Ai.

Point

A classe Point representa um objecto do tipo Point, com três campos: int id, int x e int y. O atributo id expressa o identificador do ponto e os campos x e y foram adicionados para retratar a abcissa e a ordenada respetivamente. Esta classe possui ainda dois construtores Point(int id, int x, int y) e Point(int x, int y), muito semelhantes porém, o primeiro é usado para situações em que é util guardar o identificador do ponto enquanto o segundo construtor não guarda essa informação. Adicionalmente existem três getters para obter os atributos da classe, int getId(), int getX() e int getY(). E por fim, dois métodos Point subPoint(Point p) e int mulPoint(Point p), o primeiro simboliza a subtração de pontos obtendo assim um novo ponto, por outro lado, o segundo método é

nada mais nada mesmo que o produto vetorial em que neste contexto apenas nos interessa o sinal da operação para concluir se uma determinada aresta está ou no sentido do rélogio ou no sentido contrario do rélogio em referencia a outra aresta. Esta propriedade será muito importante posteriormente para determinar as interseções existentes.

Map

Seguidamente, temos a classe Map que reflete um objecto do tipo Map, isto é, cada instância expressa um mapa no sentido em que guarda os diferentes pontos desse mapa. Contém dois atributos: int numberPoints e Point[] points, o inteiro numberPoints armazena o número de pontos para cada instância do problema. Enquanto que, o array de pontos tem como objetivo guardar os diferentes pontos criados de forma aleatória ou manualmente de acordo com o utilizador" e muda o construtor para incluir a flag, diz também que se manual, não os atributos do mapa são irrelevantes Esta classe possui um construtor Map(int n, int maxX, int minX, int maxY, int minY) em que os maxX, minX, MaxY, minY neste contexto são iguais em termos absolutos uma vez que que o plano de coordenadas inteiras estará sempre compreendido entre -m e m quer no eixo das abcissas quer no eixo das ordenadas. De uma forma geral este construtor cria instâncias de pontos com abcissas e ordenadas aleatórias que são guardas no atributo points da classe e verifica se para cada ponto criado se já existe ou não com uma função que será explicada de seguida. Por último, possui dois métodos, o void printCoordMap(int numberPoints) serve para imprimir o mapa de coordenadas, enquanto que, o método int searchPoint(int x1, int y1, int size) foi criado para verificar se um ponto já existe ou não, retornando -1 caso não existe e caso exista retorna a posição do ponto já existente.

Intersection

Contemos também a classe Intersection que simboliza um objecto do tipo Intersection, ou seja, expressa aquilo que se entende por uma interseção entre duas arestas (p1p2 e p3p4) neste problema. Assim sendo possui quatro atributos, todos do mesmo tipo Point, p1, p2, p3, p4. Estes pontos representam os quatro vértices das arestas que se intersectam. Assim sendo a classe contém um construtor Intersection(Point p1, Point p2, Point p3, Point p4) cujo propósito é de inicializar os atributos inerentes à classe. Por fim, contém quatro getters dos respetivos atributos, Point getP1(), Point getP2(), Point getP3(), Point getP4().

Candidate

De referir que a classe Candidate que exprime um objecto do tipo Candidate, quer isto dizer, tem como objetivo representar os candidatos a solução deste problema quer sejam ótimas ou não. Contém dois atributos LinkedList<Point> l e LinkedList<Intersection> in o primeiro é uma lista ligada de pontos que serve para guardar os pontos e a sua ordem para esse candidato, por outro lado, o segundo campo serve para armazenar a lista de interseções para esse mesmo candidato. Como é hábito contém o seu respetivo construtor que inicializa os atributos intrínsecos da classe. Para além de conter os habituais getters, LinkedList<Point> getL() e LinkedList<Intersection> getIn(), contém ainda um método para obter o número de interseções desse candidato, que nesta implementação é precisamente o tamanho da lista de interseções in.

Ai

Por último e não menos importante a classe Ai representa uma instância deste problema bem como a resolução de todas as questões que foram propostas. Os atributos desta classe são (ALTERAR AQUI) Scanner sc, int nPoints, int m. O atributo sc serve para a leitura da entrada padrão dos atributos nPoints e m que são respetivamente o número de pontos a considerar para a instância do problema e o tamanho do mapa. Contém claro o seu construtor Ai() que simplesmente chama o método void readInput() que faz a leitura da entrada padrão dos parâmetros do problema e as suas respetivas inicializações.

De seguida vamos explicar para cada questão os métodos usados, que por sua vez, estão inseridos na classe Ai bem como os seus aspectos mais relevantes e mais importante de tudo os resultados obtidos para cada questão do mesmo modo que, uma breve explicação.

Primeiramente é importante revelar as instâncias que foram usadas para a obtenção dos resultados que mais tarde serão introduzidos.

Instância n.º	N	M	Pontos
1	10	10	(-3,0), (-10,-3), (0,-4), (7,2), (-3,-5), (1,5), (5,-6), (1,3), (1,2), (-7,9)
2	15	10	(8,5), (-5,4), (-1,-2), (-8,-9), (8,-1), (0,3), (-3,3), (5,3), (0,4), (-1,-1), (1,0), (-7,3), (-3,-5), (-7,-8), (5,-9)
3	50	10	(3,5), (4,-3), (-7,0), (4,8), (9,3), (0,-3), (-2,-5), (7,0), (-5,-9), (2,-3), (-6,-9), (-9,2), (-7,-3), (0,8), (-8,9), (-4,6), (6,8), (-6,0), (4,4), (-6,-6), (-4,7), (0,-5), (-1,-3), (8,1), (1,7), (-9,-6), (-1,-10), (-3,0), (-5,1), (-8,-6), (8,9), (6,-9), (1,-9), (-8,0), (3,-9), (-3,6), (-6,-2), (-1,7), (-9,7), (5,-6), (3,6), (-10,-10), (-10,-9), (4,-8), (9,-9), (7,-6), (7,6), (-9,-7), (-1,5), (2,-6)

Figura 1: Tabela das instâncias, com o n.º da instância, o n.º de pontos gerados, o tamanho do mapa e a sequência dos pontos gerados.

Ao longo do problema usamos como forma de representação dos candidatos a soluções arrays com os índices na ordem pontos originais (de 0 a n-1). Importante também referir que a última posição do array é sempre igual a primeira uma vez que simboliza um ciclo e como tal o último ponto está ligado ao primeiro ponto. Assim sendo a fim de exemplo o candidato a solução da instância n.º 1 seria desta forma, [0,1,2,3,4,5,6,7,8,9,0]. Em que o 0 corresponde ao ponto (-3,0), o 1 ao ponto (-10,-3) e da mesma forma para os restantes.

Agora iremos mostrar graficamente como ficam dispostos os pontos no mapa. É importante mencionar que no uso da ferramenta Geogebra que por incapacidade da ferramenta a nomeação dos pontos apenas pode ser feita por letras ou conjunto de letras e números (A ou A1), como tal o ponto A refere-se ao ponto 0 etc. Por outro lado, os pontos X e Y não são criados, por razões

desconhecidas, assim sendo, após o ponto V tem lugar o ponto W e não o X ou o Y, e por último após a nomeação do Z tem lugar o ponto A1 e assim consecutivamente.

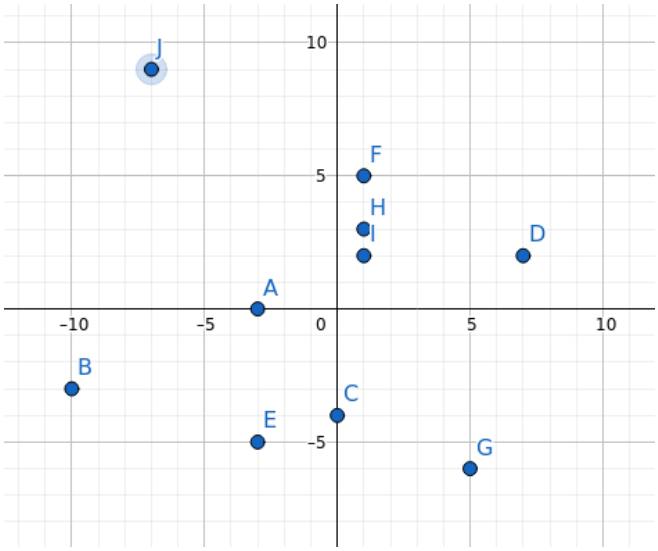

Figura 2: Gráfico inicial para a instância n.º 1.

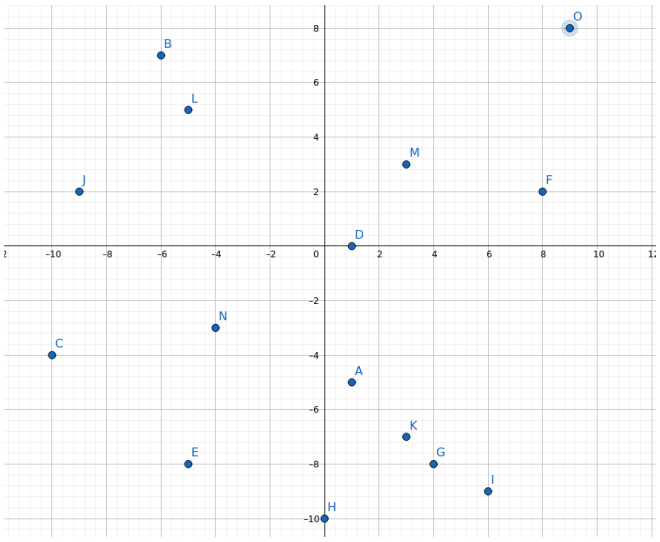

Figura 3: Gráfico inicial para a instância n.º 2.

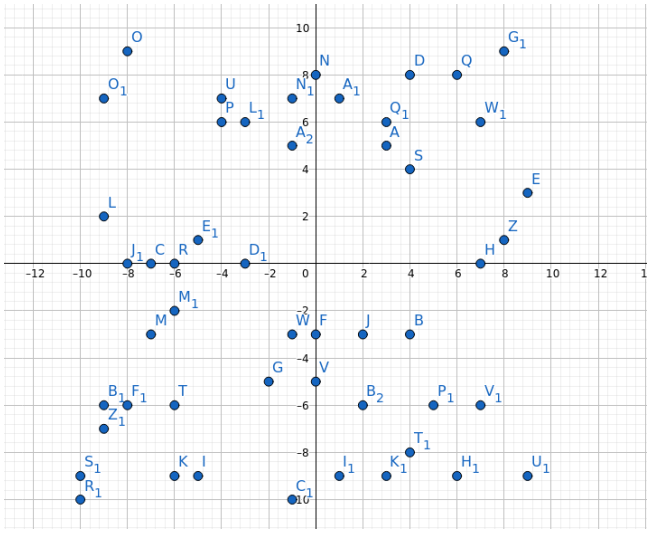

Figura 4: Gráfico inicial para a instância n.º 3.

Métodos

Neste momento iremos dar inicio à explicação das implementações dos métodos usados.

1ª questão

Na primeira questão para gerar aleatoriamente n pontos num plano com coordenadas inteiras apenas são precisas as seguintes instruções, a primeira para criar um objecto do tipo Map que tratará de gerar os n pontos de forma aleatória sempre compreendidos entre -m e m (a forma como é feita a criação já foi referida anteriormente na parte da descrição das classes usadas neste trabalho). Map map = new Map(ia.getN(), ia.getM(), -(ia.getM()), ia.getM(), -(ia.getM())); , e por fim a instrução que imprime os pontos previamente gerados. map.coordMap(ia.getN()); (função que já foi explicada na mesma secção deste relatório).

Resultados parciais:

Para cada instância obtemos os resultados anteriormente apresentados na figura n.º 1, mais concretamente, na coluna dos pontos. Não se consegue analisar muito sobre estes resultados obtidos, uma vez que, são os resultados esperados tendo em conta a aleatoriedade implícita.

2ª questão

Seguidamente, na segunda questão é pedido que na primeira alternativa seja gerada uma permutação qualquer dos pontos e numa segunda alternativa, a formulação do candidato a solução deve seguir uma heurística de “nearest-neighbour first”. Uma vez que a criação dos pontos é feita de forma aleatória gerar uma permutação qualquer dos pontos é bastante fácil, portanto, foi criado o método LinkedList>Point> permutation(Map map), que muito sucintamente percorre o array points da classe Map e adicionada cada ponto a uma lista ligada, obtendo assim uma permutação qualquer de pontos. Por outro lado, a elaboração deste candidato segundo a heurística de “nearest-neighbour first” é feita utilizando o método LinkedList<Point> neighbour(LinkedList<Point> l), que basicamente percorre a lista, que é a dada como argumento na função (lista obtida na alternativa anterior), usamos esta lista uma vez que, em nada altera a forma como o método funciona pois iremos percorrer toda a lista à procura do ponto mais próximo, segundo a distância euclideana, para cada ponto da lista. Facilitando assim a implementação do método. Como é de esperar assim que se obtenha o ponto mais próximo este é adicionado a uma nova lista e removido da atual. Por fim é retornada a nova lista que segue a heurística pretendida.

Resultados parciais

Para cada instância do problema foram obtidos os seguintes resultados presentes nesta tabela bem como os gráficos que comprovam os resultados.

Instância n.º	2º A	2 º B
1	[0,1,2,3,4,5,6,7,8,9,0]	[0,8,7,5,3,6,2,4,1,9,0]
2	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0]	[0,4,7,10,9,2,12,13,3,11,1,6,5,8,14,0]
3	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, 16,17,18,19,20,21,22,23,24,25,26,27,28, 29,30,31,32,33,34,35,36,37,38,39,40,41, 42,43,44,45,46,47,48,49,0]	[0,40,3,16,30,46,4,23,7,1,9,5,22,6,21,49,43, 34,32,26,8,10,19,29,25,47,42,41,12,36,17,2, 33,11,28,27,48,37,13,24,35,15,20,14,38,18, 39,45,31,44,0]

Figura 5: Tabela de resultados para a questão n.º 2.

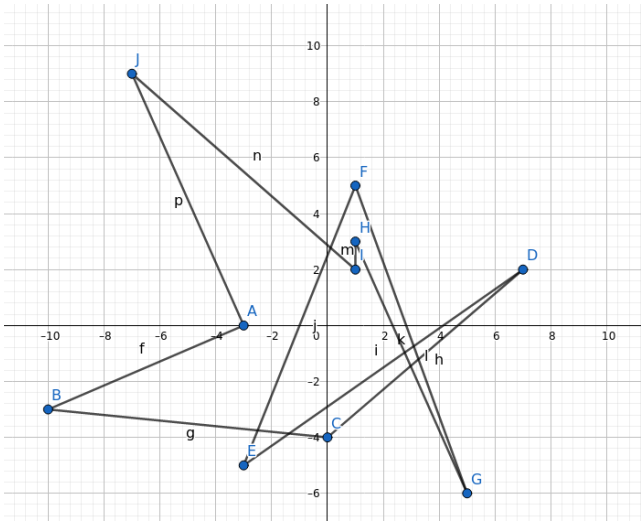

Figura 6: Gráfico da questão 2ºA para a instância n.º 1.

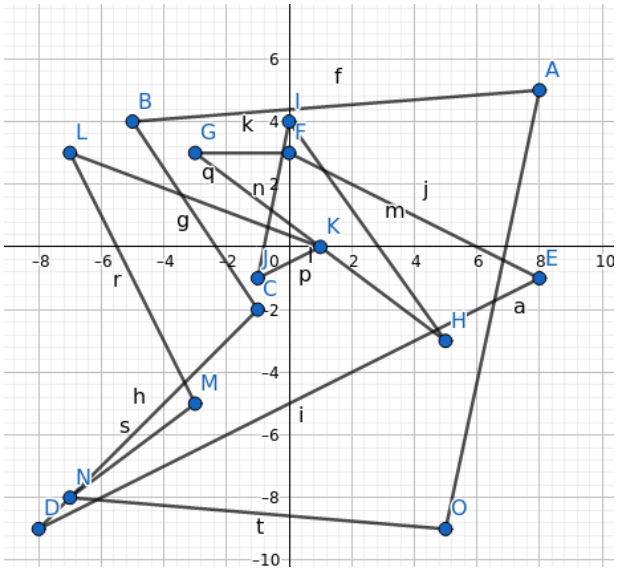

Figura 7: Gráfico da questão 2ºA para a instância n.º 2.

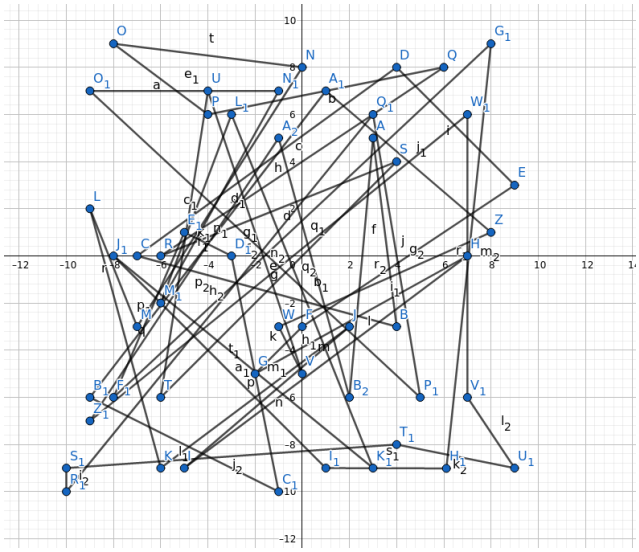

Figura 8: Gráfico da questão 2ºA para a instância n.º 3.

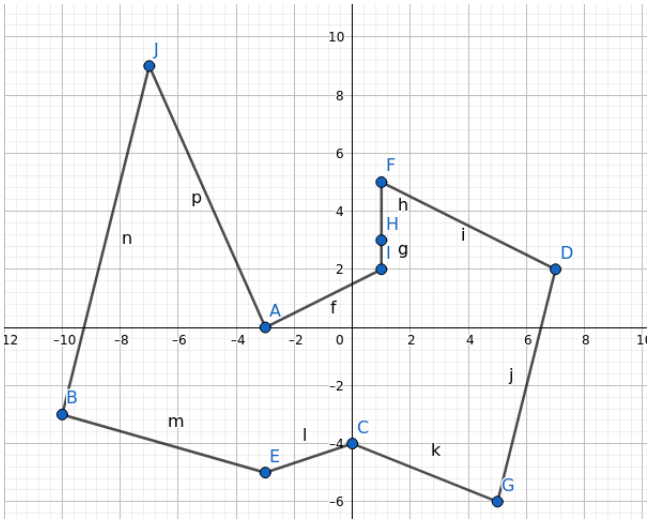

Figura 9: Gráfico da questão 2ºB para a instância n.º 1.

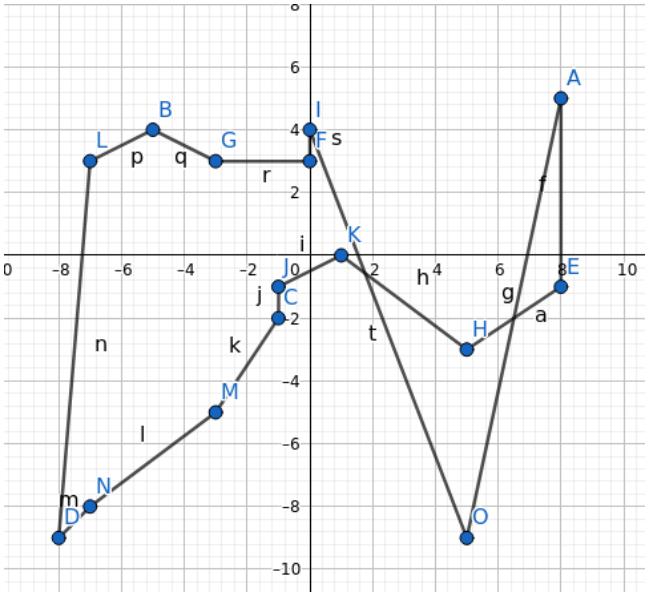

Figura 10: Gráfico da questão 2ºB para a instância n.º 2.

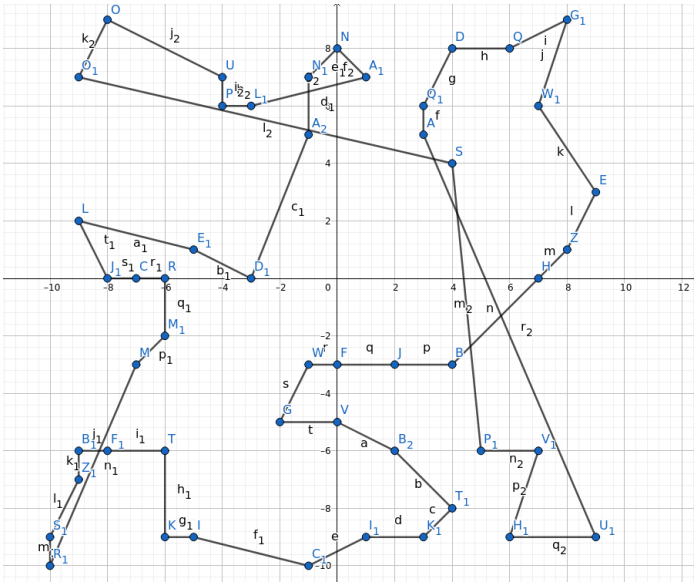

Figura 11: Gráfico da questão 2ºB para a instância n.º 3.

Referente à segunda questão, os resultados para a primeira alternativa demonstra que gerando uma permutação de forma aleatória nunca se obtém um clico de hamilton. Enquanto que, na segunda alternativa o candidato a solução também é raro obter um ciclo de hamilton porém em algumas situações está muito perto de isso acontecer e com métodos um pouco mais eficazes é provável que se consigam polígonos simples.

3ª questão

Continuamente, daremos ênfase à terceira questão deste problema. Nesta situação é pedido que seja gerada a vizinhança de um candidato s usando a regra "2-exchange". Para tal, calculamos as interseções existentes no candidato s, usando a classe Candidate para gerar um candidato com os seus argumentos a serem a lista de pontos do candidato já calculada e ainda um outro argumento, a mesma lista aplicada ao método brute(). (Candidate original = new Candidate(l,ia.brute(n));). O método LinkedList<Intersection> brute(LinkedList<Point> l) tal como foi dito recebe uma lista de pontos e calcula de forma bruta $(O(n^2))$ as interseções existentes num determinado candidato. Primeiramente temos um caso de verficação para situações em que a lista de pontos tem exatamente três ou menos pontos, isto é claramente trivial pois com 3 ou menos pontos é impossível haver interseções. Posteriormente, temos de percorrer toda a lista de pontos, porém temos de a percorrer de uma forma especial, ou seja, percorremos a lista de dois em dois pontos, o que simboliza verificar aresta por aresta. Caso as arestas se intersectem a quando da chamada da função segmentsIntersect(p1, p2, p3, p4) é criada uma nova interseção entre esses quatro pontos e adiciona à lista de interseções. A função boolean segmentsIntersect(Point p1, Point p2, Point p3, Point p4) é nada mais nada menos que a implementação do algoritmo para interseção de um par de segmentos[3]. Por último, falta verificar o último segmento da lista. A construção da lista de

candidatos é feita com o método newCandidates (LinkedList<Candidate> c = ia.newCandidates(in, n);), que tem como argumentos a lista de interseções e a lista de pontos. Este método percorre a lista de interseçoes dada e para cada interseção adiciona o candidato com a troca das arestas feitas pela regra "2-exchange" através do método (listExchanged(intersection, l, intersectionNumber));) . A função listExchanged primeiramente encontra a posição dos pontos p2 e p3 e consequemente chama a função a exchange (exchange(l2, j, k,intersectionNumber);). A função exchange faz a troca das arestas. Terminado assim por retornar o candidato.

Resultados parciais

Para cada instância do problema foram obtidos os candidatos representados na segundo coluna da tabela para cada instância.

Instância n.º	3º
1	[]
2	[4,7,8,6,10,0,13,3,12,5,14,11,1,9,2,4]
	[0,40,3,16,30,46,4,23,7,18,38,14,20,15,35,24,13,37,48,27,28,11,33,2,17,36,12,41,42, 47,25,29,19,10,8,26,32,34,43,49,21,6,22,5,9,1,39,45,31,44,0]
	[0,40,3,16,30,46,4,23,7,1,9,5,22,6,21,49,43,34,32,26,8,10,19,29,41,42,47,25,12,36,17, 2,33,11,28,27,48,37,13,24,35,15,20,14,38,18,39,45,31,44,0]
	[0,40,3,16,30,46,4,23,7,1,9,5,22,6,21,49,43,34,32,26,8,10,19,29,25,47,42,41,12,36,17, 2,33,11,28,27,48,24,13,37,35,15,20,14,38,18,39,45,31,44,0]
3	[0,40,3,16,30,46,4,23,7,1,9,5,22,6,21,49,43,34,32,26,8,10,19,29,25,47,42,41,12,36,17, 2,33,11,28,27,48,38,14,20,15,35,24,13,37,18,39,45,31,44,0]
	[7,23,4,46,30,16,3,40,0,1,9,5,22,6,21,49,43,34,32,26,8,10,19,29,25,47,42,41,12,36,17, 2,33,11,28,27,48,37,13,24,35,15,20,14,38,18,39,45,31,44,0]
	[38,14,20,15,35,24,13,37,48,27,28,11,33,2,17,36,12,41,42,47,25,29,19,10,8,26,32,34, 43,49,21,6,22,5,9,1,7,23,4,46,30,16,3,40,0,18,39,45,31,44,0]
	[18,38,14,20,15,35,24,13,37,48,27,28,11,33,2,17,36,12,41,42,47,25,29,19,10,8,26,32, 34,43,49,21,6,22,5,9,1,7,23,4,46,30,16,3,40,0,39,45,31,44,0]

Figura n.º 12: Tabela de resultados para a questão n.º 3.

Podemos analisar os resultados da seguinte maneira, como na primeira instância não existem interseções é de se esperar que não existam candidatos. Por outro lado, na segunda instância temos uma interseção nas arestas CA e EN e segundo a regra "2-exchange" troca-se o ponto A pelo E e ficando assim com as arestas CE e AN, consequente na lista dos resultados para além de trocar as letras A e E (neste caso os números 0 e 4) entre si também é obrigatório trocar a ordem dos pontos

entre A e E. Mais concretamente passaríamos de [0,10,6,8,7,4,…,0] para [4,7,8,6,10,0,…,4]. Finalmente, na última instância temos sete interseções e como é esperado temos sete candidatos e de recordar que as trocas dos pontos entre os vértices das arestas que se intersectam também se trocam. Em suma, nesta questão conseguimos obter os resultados esperados para as instâncias.

4ª questão

Nesta questão é pedido que se aplique melhoramento iterativo (Hill Climbing) usando quatro diferentes alternativas. Portanto, antes da execução do Hill Climbing verificamos se existem candidatos porque caso não existem a solução já é otima. Existindo candidatos para cada uma das alternativas é feita a chamada da função hc (ia.hc(c,i,l,map);), recebe como argumentos a lista de candidatos da lista "nearest-neighbour first" criado pelo método neighbour, a lista de pontos, o mapa que contem os pontos e uma flag para passar diretamente ao hillClimbing. em que este é o método mãe do int hillClimbing(LinkedList<Candidate> candidateList,int alternative). O objetivo do método é determinar se a solução dada pelo hillClimbing é melhor que a lista de pontos com o cálculo do número de interseções, caso a solução tenha menos interseções é descoberto os novos candidatos para essa solução. Só é terminado quando a solução nao apresentar melhorias á instançia anterior ou quando não houver nenhuma interseção. No final apresenta se a alternativa foi ótima ou não e a lista de pontos na última instância em que houve melhorias.

Resultados parciais

Para cada instância do problema foram obtidos as seguintes soluções representados nas tabelas.

Instância n.º	4º A	4º B
1	[]	[]
2	[4,7,8,6,10,0,13,3,12,5,14, 11,1,9,2,4]	[4,7,8,6,10,0,13,3,12,5,14, 11,1,9,2,4]
3	[0,40,3,16,30,46,4,23,7,18,38,14,20,15,35 ,24,13,37,48,27,28,11,33,2,17,36,12, 41,42,47,25,29,19,10,8,26,32,34,43,49, 21,6,22, 5,9,1,39,45,31,44,0]	[0,40,3,16,30,46,4,23,7,18,37,13,24,35,15, 2,14,38,48,27,28,11,33,2,17,36,12,41,42,47, 25,29,19,10,8,26,32,34,43,49,21,6,22,5,9,1, 39,45,31,44,0]

Instância n.º	4º C	4º D
1	[]	[]
2	[4,7,8,6,10,0,13,3,12,5,14, 11,1,9,2]	[4,7,8,6,10,0,13,3,12,5,14, 11,1,9,2]
3	[18,7,23,4,46,30,16,3,40,0,24,13,37,35, 15,20,14,38,48,27,28,11,33,2,17,36,12,25, 4742,41,29,19,10,8,26,32,34,43,49,21,6, 22,5,9,1,39,45,31,44,18]	[0,40,3,16,30,46,4,23,7,18,38,14,20,15,35, 37,13,24,48,27,28,11,33,2,17,36,12,25,47, 42,41,29,19,10,8,26,32,34,43,49,21,6,22,5,9, 1,39,45,31,44,0]

Figura n.º 13: Tabela de resultados para a questão n.º 4.

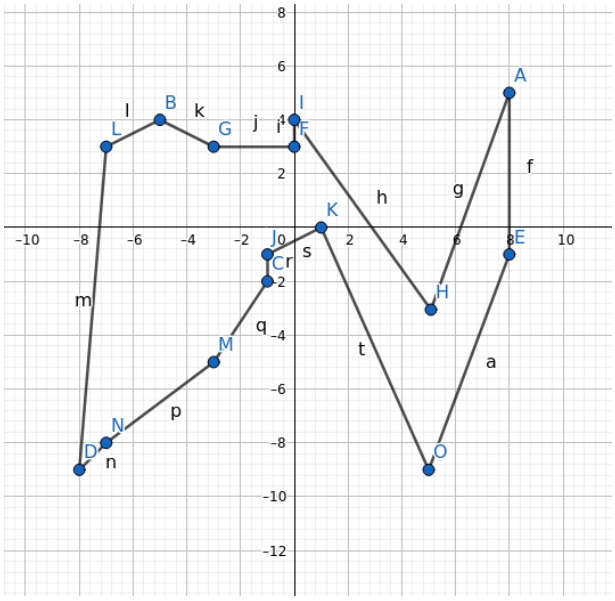

Figura 14: Gráfico da questão 4 para a instância n.º 2.

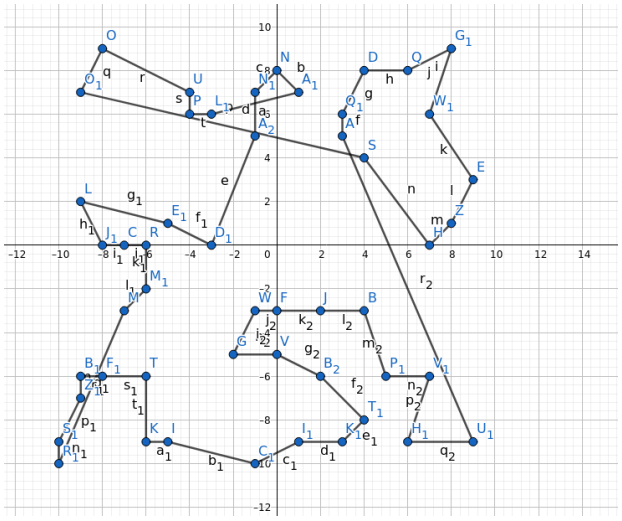

Figura 15: Gráfico da questão 4A para a instância n.º 3.

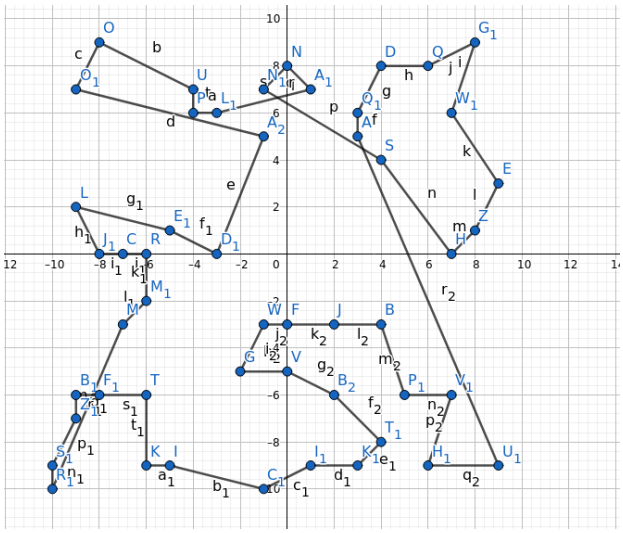

Figura 16: Gráfico da questão 4B para a instância n.º 3.

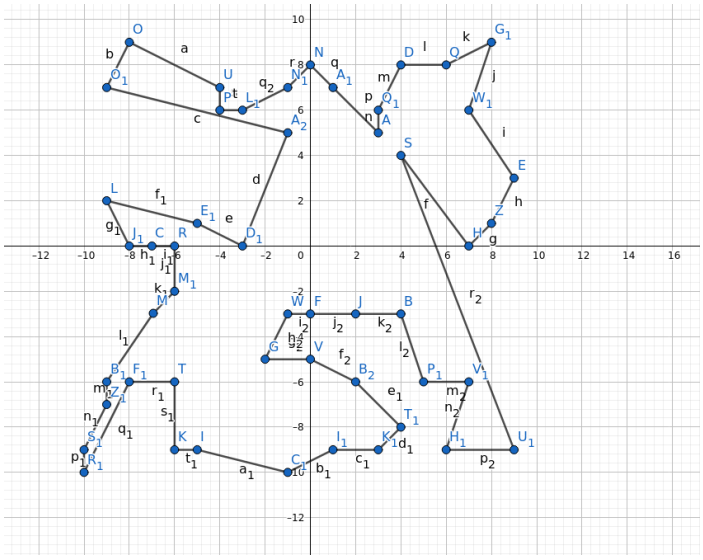

Figura 17: Gráfico da questão 4C para a instância n.º 3.

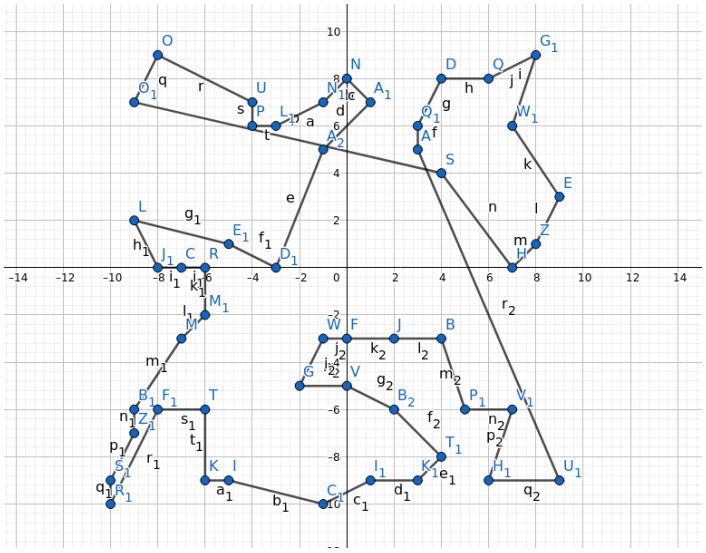

Figura 18: Gráfico da questão 4D para a instância n.º 3.

De uma forma sucinta podemos verificar que para a primeira instância uma vez que já foi encontrada um polígono simples o método não é executado. Para a segunda instância uma vez que só existe uma intersecção rapidamente atinge uma solução ótima, independentemente da heurística utilizada. Enquanto que, na terceira instância é possível observar que quer usando a heurística da 4A ou da 4D os resultados tendem a ser semelhantes. Sendo pior a segunda alternativa e obviamente a melhor a terceira alternativa. Estes resultados serão apresentados nos resultados finais do problema.

Conclui-se assim que o método é rápido para instâncias com poucos pontos isto porque, quando temos uma situação com muitos pontos é raro dar uma solução ótima.

5ª questão

Finalmente, na quinta questão é aplicado o método Simulated annealing usando como medida de custo o número de cruzamentos de arestas. A alternativa do Simulated Annealing é executada pelo método sa que recebe como argumento a lista original criada pelo utilizador ou aleatória, o pseudocódigo pode ser encontrado na wikipedia[6]. O kmax é uma variavel da classe ia que é alterada caso seje o valor seje -1 para um valor default, que é o quintuplo do numero de pontos. O simulated annealing é uma alternativa de procura aleatória em que as soluções são baseadas na função P baseado no artigo de Kirkpatrick[6] em simulated annealing. No fim da função o programa imprime a ordem da lista na instancia final e caso a escolha foi ótima de acordo com o valor de kmax.

Resultados parciais

Para cada instância do problema foram obtidos as seguintes soluções representados na tabela.

Instância n.º	5
1	[]
2	[5,8,7,6,4,2,9,1,11,12,13,10,0,3,14,5]
3	[32,26,41,42,25,29,12,2,17,36,28,15,33,11,38,20,35,48,0,37,14,13,3,16,30,4,44,45,43, 31,34,39,1,7,23,46,24,40,18,5,6,22,27,19,47,10,8,21,9,49,32]

Figura 19: Tabela de resultados para a questão n.º 4.

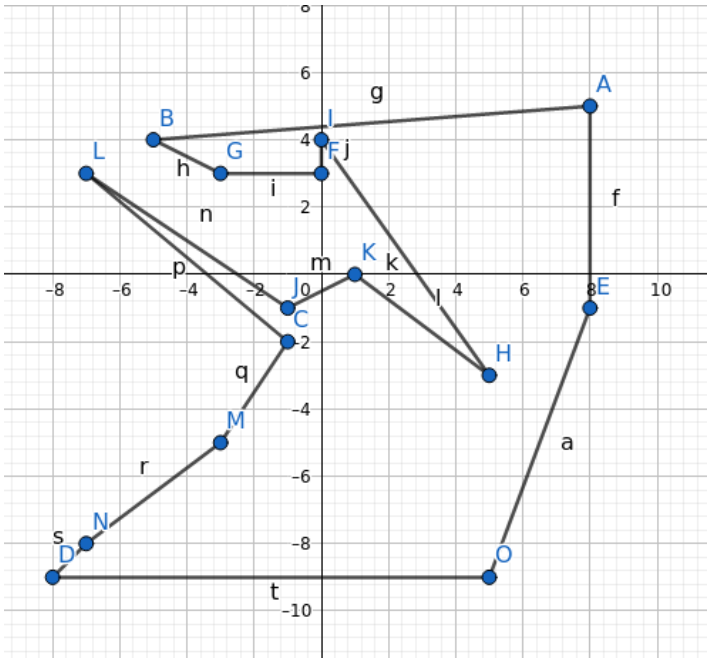

Figura 20: Gráfico da questão 5 para a instância n.º 2.

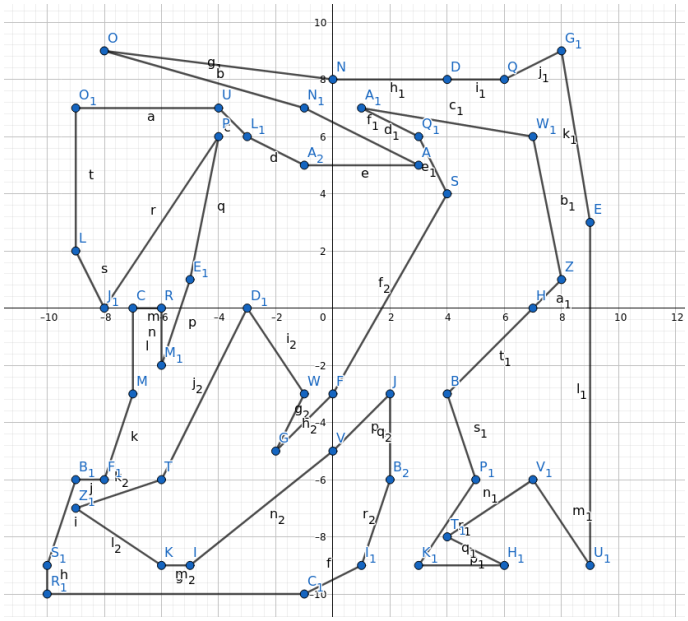

Figura 21: Gráfico da questão 5 para a instância n.º 3.

Com resultados podemos verificar que para a primeira instância pela mesma razão que na questão anterior a lista é vazia. Para as instâncias dois e três as soluções foram otimas o que demonstra a eficácia do algoritmo.

Resultados Finais

Nesta secção apenas nos parece pertinente abordar as questões quatro e cinco. Assim sendo, realizamos 20 testes com 100 pontos para ambas as questões e obtivemos os seguintes resultados.

Resultados	4A	4B	4C	4D	5
	5/20 = 25%	3/20 = 15%	11/20=55%	5/20 = 25%	20/20=100%

Figura 22: Tabela dos testes 20 realizados para as questões 4 e 5.

Observou-se que para as questões A,B,D raramente foi possível obter soluções otimas enquanto que na C pouco mais de metade das vezes a solução era otima. E claramente na 5 obtivemos sempre uma solução. Importante referir que o Simulated Annealing funciona sempre para um kmax suficientemente grande, apesar de ser lento, o que nos leva a concluir que para usar o Simulated Annealing tem de existir sempre uma troca entre otimização e tempo de execução.

Referências

[1] Gerador de polígonos aleatórios (RPG) de Thomas Auer e Martin Held - https://www.cosy.sbg.ac.at/~held/projects/rpd/rpd.html.

[2] Geogebra website – https://www.geogebra.org/graphing?lang=en.

[3] Pseudocódigo do algoritmo de interseção de um par de segmentos - http://www.inf.ed.ac.uk/teaching/courses/ads/Lects/lecture1516.pdf.

[4] Pseudocódigo do algoritmo de Shamos e Hoey- http://euro.ecom.cmu.edu/people/faculty/mshamos/1976GeometricIntersection.pdf .

[5] Pseudocódigo do algoritmo de Bentley-Ottmann- http://www.cs.uu.nl/geobook/pseudo.pdf, http://www.cs.uu.nl/geobook/, http://www.cs.uu.nl/docs/vakken/ga/2020/slides/slides2a.pdf.

[6] Pseudocódigo do Simulated Annealing - https://en.wikipedia.org/wiki/Simulated_annealing.

Distribuição do trabalho pelos elementos

Código realizado pelos dois membros bem como o relatório. Não houve colabaração de qualquer outro grupo.