Departamento de Ciência de Computadores FCUP

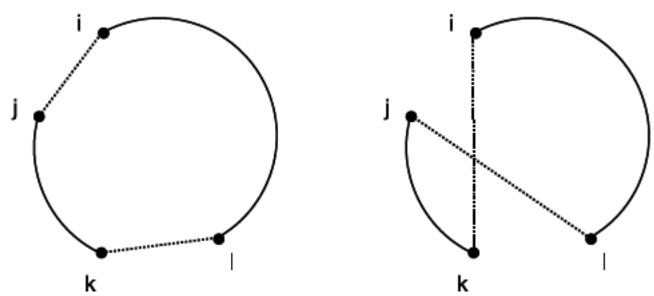
Inteligência Artificial CC2006 2020/2021

Trabalho 1

Recorde que no problema do caixeiro viajante (TSP, "**T**raveling **S**alesman **P**roblem") é dado um grafo não dirigido **completo** pesado e se procura determinar um percurso, que seja um ciclo de Hamilton com custo total mínimo. Um ciclo de Hamilton é um percurso fechado que não repete nós e passa em todos os nós. Admitimos que o grafo é completo pois, se não for, podemos introduzir ramos novos com peso ∞ (na prática, algum valor suficientemente grande).

Na procura de **candidatos a soluções** por **pesquisa local**, podemos ter **métodos construtivos** ou **métodos perturbativos**. Os métodos construtivos constroem candidatos a soluções, elemento a elemento, de acordo com alguma estratégia (heurística), que muitas vezes é *greedy*. Para o TSP, a heurística *"nearest neighbour first"* consiste em *inserir no percurso o nó ainda não visitado, que esteja mais póximo do último visitado*. O nó inicial pode ser escolhido aleatoriamente.

Os **métodos perturbativos** partem de um candidato completo e geram novos candidatos **numa vizinhança** desse, segundo alguma regra. No TSP, qualquer permutação dos nós define um candidato a solução (se tivermos xy nessa permutação, tal significa que estamos a usar o ramo não orientado $\{x, y\}$). A vizinhança de um candidato s pode ser definida, por exemplo, pela heurística **"2-exchange"** que retira dois ramos $\{i, j\}$ e $\{k, l\}$ de s e coloca os ramos $\{i, k\}$ e $\{j, l\}$, o que só é possível se estes não estiverem já em s.



O problema TSP tem aplicações na definição de rotas mas não só. Nos anos 1990s, Thomas Auer e Martin Held propuseram um **gerador polígonos aleatórios (RPG)**, que aplica ideias semelhantes para criar um polígono simples a partir de um conjunto de n pontos no plano, que serão os seus vértices. Um **polígono simples** não tem arestas que se intersectem a não ser em vértices.

Para saber mais sobre o RPG: `http://www.cosy.sbg.ac.at/~held/projects/rpd/rpd.html`.

Pretendemos **estudar métodos de pesquisa local e pesquisa local estocástica** usando este problema de geração de polígonos simples. Deverão implementar programas (em Python, C/C++ ou Java) para estudar experimentalmente o problema.

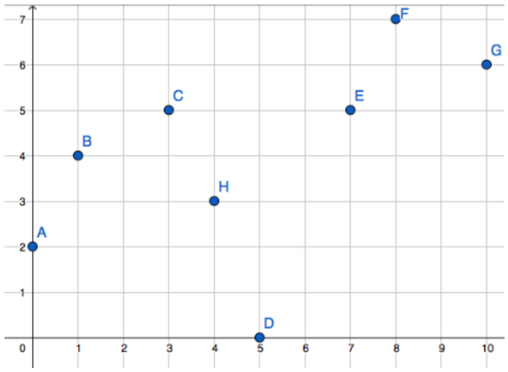
1. Gerar aleatoriamente n de pontos no plano com coordenadas inteiras, de $-m$ a m, para n e m dados.

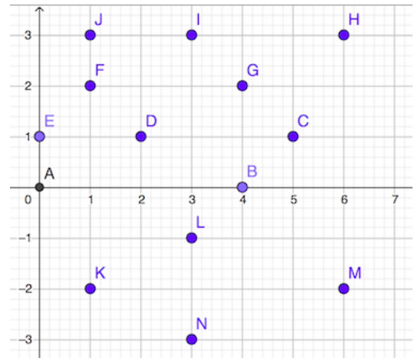
2. Determinar um candidato a solução, considerando as alternativas seguintes:

a) Gerar uma **permutação** qualquer dos pontos;

b) Aplicar a heurística **"nearest-neighbour first"** a partir de um ponto inicial, que pode ser escolhido aleatoriamente. O próximo ponto a visitar será um dos mais próximos do atual, segundo a distância euclideana (na comparação de distâncias, usam o **quadrado da distância** para evitar erros numéricos).

Para ganhar intuição, podem analisar como seria nas instâncias desenhadas:





3. Para um candidato s, determinar a vizinhança obtida por “2-exchange”. As duas arestas selecionadas para remoção devem intersectar-se no interior. Existirão sempre arestas nessas condições, a menos que o polígono seja simples (o que significaria que tinhamos chegado ao objetivo).

- Para verificar se dois segmentos se intersectam, podem tentar calcular o seu ponto de interseção, mas é menos robusto numericamente do que a aplicação de **testes baseados no produto vetorial**.
  - Capítulo 33 do livro CLRS, i.e, *Introduction to Algorithms, 3rd Edition, MIT, 2009* de T.H.Cormen, C.E.Leiserson, R.L.Rivest e C.Stein.
  - Algoritmo para interseção de um par de segmentos em pseudocódigo (cf, CLRS, cap 33): `http://www.inf.ed.ac.uk/teaching/courses/ads/Lects/lecture1516.pdf`
  - Explicação em vídeo: `https://www.youtube.com/watch?v=3YFUQDRL1s4` `https://www.youtube.com/watch?v=R08OY6yDNy0`

- Para determinar **um par de segmentos que se intersecte** ou **todos os pares de segmentos que se intersectam** podem aplicar um algoritmo *força-bruta* $O(n^2)$.

  Mas, se tiverem interesse, podem analisar a possibilidade de adaptar o **algoritmo Shamos-Hoey**, para determinar um par que se intersecte, e o **algoritmo de Bentley–Ottmann**, para determinar todos os pares que se intersectam.

  - Algoritmo de Shamos e Hoey `http://euro.ecom.cmu.edu/people/faculty/mshamos/1976GeometricIntersection.pdf`
  - Algoritmo de Bentley–Ottmann
    Pseudocódigo: FINDINTERSECTIONS(S)
    `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`

  - M. de Berg et al, Computational Geometry : Algorithms and Applications (3rd Ed), Chapter 2 ”Line Segment Intersection”(2008) `https://people.inf.elte.hu/fekete/algoritmusok_msc/terinfo_geom/konyvek/`
  - Computational Geometry-GEI, Vera Sacristán, UPC `https://dccg.upc.edu/people/vera/teaching/courses/computational-geometry/#material`

4. Aplicar melhoramento iterativo (*hill climbing*). Analisar várias alternativas para escolha do candidato na vizinhança "2-exchange" do atual:

a) optar pelo candidato que reduziria mais o perímetro (i.e., cuja soma dos comprimentos das arestas do polígono é mínima), o que corresponde a uma heurística "best-improvement first";

b) optar pelo primeiro candidato que encontrar nessa vizinhança ("first-improvement");

c) optar pelo candidato que tiver menos conflitos de arestas (menos cruzamentos de arestas);

d) optar por um qualquer candidato nessa vizinhança.

NB (*propriedade geométrica*): Como arestas $\{i, j\}$ e $\{k, l\}$ escolhidas para serem removidas se intersectam no interior, a substituição dessas arestas por $\{i, k\}$ e $\{j, l\}$, faz com que o novo polígono tenha sempre um perímetro menor.

5. Aplicar *simulated annealing*. Usar como medida de custo o número de cruzamentos de arestas.

6. Aplicar *metaheurística* ACO (*ant colony optimization*), com várias formigas (Ant System), que podem partir de pontos distintos. Esta metaheurística não foi dada nas aulas, mas é interessante avaliar a sua aplicação a este problema. Existem diversas variantes, mas poderão restringir-se à versão clássica (possivelmente, integrando-a com um dos métodos de pesquisa local anteriores). Na aplicação a TSP, em cada iteração, cada uma das formigas determina um percurso que passará por todos os nós, a partir de uma origem (fixa ou aleatória). Na construção desse percurso, a probabilidade de a formiga k usar o ramo (i, j) se estiver no nó i é dada por

$$p_{ij}^{(k)} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^{(k)}} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ para } j \in N_i^{(k)}$$

sendo $N_i^{(k)}$ os vizinhos do nó i aceitáveis para a formiga k (i.e., os que ainda não estão no percurso até i).

O valor τ_{ij} é a quantidade de feromona no ramo (i, j) e η_{ij} o inverso do peso de (i, j), isto é $1/d_{ij}$, sendo d_{ij} a distância entre i e j e representa *informação heurística*. Os parâmetros α e β são constantes positivas e refletem a importância relativa que se dá a essa heurística e à feromona, A **feromona** depositada nos ramos constituiu a forma de comunicação entre as formigas. Ao marcarem os trajetos que encontraram, aumentam a probabilidade de outras formigas usarem os mesmos ramos numa iteração seguinte. Para diversificar a procura, há depósito de feromona e também evaporação. Podem admitir que, no final de uma iteração, a atualização de feromona em (i, j) é dada por

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_k \Delta\tau_{ij}^k,$$

onde $\Delta\tau_{ij}^k = 0$ é zero se (i, j) não pertence ao percurso que a formiga k encontrou nessa iteração e $\Delta\tau_{ij}^k = Q/L_k$, caso contrário, sendo L_k igual ao comprimento do percurso que a formiga k encontrou, e Q e ρ constantes. Q e ρ são parâmetros a definir assim como o número de formigas da colónia, α e β. O valor de $\rho \in]0, 1]$ representa a taxa de evaporação da feromona. Na atualização de τ_{ij}, podem também selecionar alguns dos trajetos construídos na iteração e, possivelmente, a melhor solução encontrada até ao momento e usar $\rho\Delta\tau_{ij}^k$ em vez de $\Delta\tau_{ij}^k$. **Na aplicação à geração de polígonos, poderá fazer sentido alterar esta definição de $\Delta\tau_{ij}^k$, para penalizar rotas que têm mais cruzamentos?**

Referências para Ant Colony Optimization

- M.Dorigo, C.Blum (2005): Ant colony optimization theory: A survey. TCS 244(2-3), pp 243-278

  `https://www.sciencedirect.com/science/article/pii/S0304397505003798`

  disponível também em: `https://staff.fmi.uvt.ro/~daniela.zaharie/am2016/proiecte/tehnici/ACO/aco_survey.pdf`

- Bio-inspired metaheuristics, by Daniela Zaharie, West University of Timisoara, 2019.
  `https://staff.fmi.uvt.ro/~daniela.zaharie/ma2019/lectures/metaheuristics2019_slides7.pdf`

Desenvolvimento do trabalho e prazos:

- O trabalho deverá ser desenvolvido em grupos de **dois elementos**, preferencialmente da mesma turma prática. Os grupos devem manter a mesma constituição para o segundo trabalho prático. Qualquer dificuldade na constituição dos grupos deve ser comunicada aos docentes.
- Os grupos devem ser registados no Piazza até **22 de Março**.
- O prazo limite para submissão do trabalho é **5 de Abril**.
- Até **24 de Março**, deverão resolver pelo menos as questões 1. a 4. A utilização/adaptação dos algoritmos de Bentley-Ottmann e de Shamos-Hoey é optativa.
- No fim, devem **submeter o código** e **um relatório**, com a descrição dos métodos, da implementação e da análise experimental (discussão de resultados e casos de teste).
- Os grupos podem discutir abordagens mas **não podem partilhar código**. Caso os programas incluam excertos que o grupo reutilizou, devem identificar claramente onde começa e onde termina o código que o grupo não implementou. Devem procurar implementar programas bem estruturados.
- Os elementos do grupo devem dividir o trabalho entre si. Espera-se que qualquer um dos elementos **conheça e possa descrever** o trabalho efetuado por si e pelo outro elemento do grupo.
- Os exames da UC poderão ter questões relacionadas com os trabalhos práticos.