Universidade do Porto

Faculdade de Engenharia

FEUP

Pesquisa na implementação de um Guia Turístico

Relatório Final

Inteligência Artificial
3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:
João Pedro Milano Silva Cardoso – 200900579 – ee09063@fe.up.pt
Valter Emanuel Ribeiro da Silva – 201105632 – ee11043@fe.up.pt

31 de Maio de 2015

Conteúdo

Figuras

Objectivo

No seguimento da realização do projecto proposto na unidade curricular de Inteligência Artificial, o grupo constituído pelos elementos João Cardoso e Valter Silva, pretendem elaborar uma aplicação de guia turístico.

O presente relatório está dividido em: descrição do problema e análise do mesmo; cenário de exemplo; formato dos dados de entrada; a abordagem seguida, com descrição do algoritmo e heurística utilizada; detalhes extra de implementação; interface do projeto; experiências e resultados; conclusões e melhorias adicionais.

Descrição do Problema

O projeto a implementar visa no planeamento de um circuito turístico, com objectivo de gerar um plano de viagem. Este plano deverá minimizar o custo e maximizar o número de lugares de maior importância visitados.

O turista sai do seu hotel às 9h e volta às 19h, sendo então um dia limitado a 10 horas. O utilizador especifica:

- A importância que atribui aos locais que quer visitar;
- A localização do seu hotel.

O sistema tem a seguinte informação ao seu dispor:

- Mapa da cidade;
- O tempo de visita aconselhado para cada local de interesse.

Análise do Problema

Visto que o problema se refere a uma cidade, o grafo resultante será fortemente conexo.

Ao ser conhecido o mapa da cidade, é possível recorrer a latitude e longitude para distribuir os nós relativamente uns aos outros e calcular a distância e o tempo entre eles.

Como o objectivo é encontrar o caminho de maior importância em cada dia, a heurística irá depender desse valor. O tempo será utilizado para limitar as opções de cada nó em relação a vizinhos válidos. Um vizinho válido de um nó n é um outro nó v tal que:

$$Tt + Tnv + Tvh + Tvv <= Tlimite$$

Em que *Tt* é o tempo total gasto até ao nó n, *Tnv* tempo de viagem entre o nó n e o potencial vizinho v, *Tvh* o tempo de viagem do vizinho ao hotel e *Tvv* o tempo de visita do vizinho. Isto assegura que qualquer nó é capaz de fechar o caminho, voltando ao hotel, sem ultrapassar o tempo limite.

A importância atríbuida a cada nó será uma valor inteiro de 0 a 5.

Cenário

Para testes o ficheiro utilizado refere-se à cidade de Paris, França, composto por 16 nós.

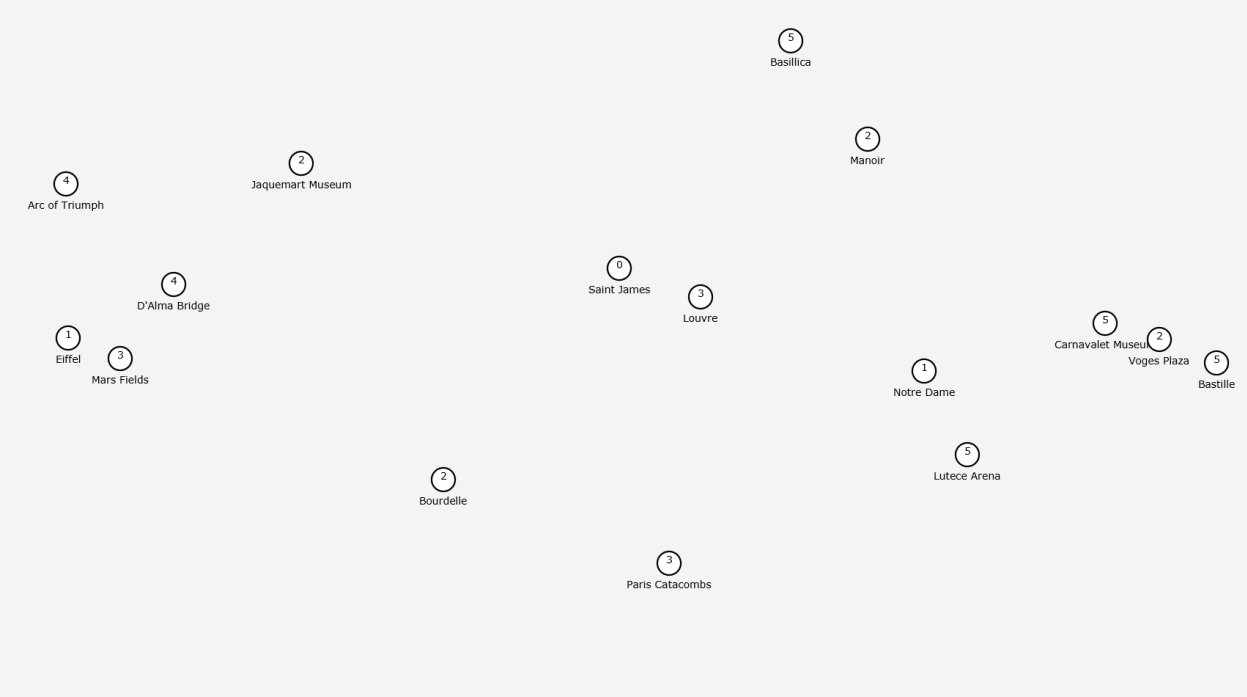

Figura 1 - Cenário Exemplo

Dados de Entrada

O ficheiro de entrada é um ficheiro de texto, em que cada linha contém informação referente a uma localidade, organizada do seguinte modo:

Nome | Nome Alternativo | Tempo de Visita | Importância

O projeto utiliza ainda um ficheiro de texto adicional, latlongDB, onde estão guardados os valores de latitude e longitude das localidades, no formato:

Nome Alternativo | Latitude | Longitude

Abordagem

Algoritmo A*

Para resolução do problema foi utilizado o algoritmo A*, ligeiramente modificado para servir o contexto do projeto. O algoritmo é normalmente utilizado para encontrar o caminho mais curto entre dois nós, sendo no entanto necessário encontrar neste problema um caminho circular, que comece e acabe no hotel, de maior importância possível dentro do tempo limite.

O algoritmo adaptado pode ser dividido nos seguintes passos:

1. Adicionar o nó inicial (hotel) à lista aberta.
2. Repetir:
    a. Procurar o nó com o menor valor F da lista aberta. Esse será o nó atual.
    b. Passar o nó atual para a lista fechada
    c. Para cada vizinho válido do nó:
        i. O seu valor G é calculado.
        ii. Se o vizinho não estiver na lista aberta é adicionado, os seus valores G, H e F são calculados.
        iii. Se o vizinho se encontrar na lista fechada verifica-se se o caminho atual é melhor comparando o valor de G. Um valor menor indica um caminho melhor.
        iv. Se o vizinho necessitar de ser atualizado é calculado um novo valor H.
3. O algoritmo pára e constrói o caminho quando:
    a. A lista aberta estiver vazia, indicando que não existem mais nós para explorar
    b. A lista de vizinhos válidos estiver vazia, o que indica o fecho do caminho.

Para fechar o caminho e assegurar que este é circular o hótel recebe como pai o último nó atual, tornando-se efetivamente o primeiro e último nó do caminho.

O caminho pode ser fechado quando um nó não encontrar vizinhos válidos visto que nesse momento nenhum outro nó na lista aberta se encontra na condição de ter como vizinho um nó com importância suficiente para formar um caminho melhor.

Heurística

No algoritmo A* é atribuído, para ordenação da lista aberta, um inteiro negativo F a cada nó, que resulta da soma do valor corrente G com o valor futuro H.

O valor corrente G de um nó vizinho resulta da soma do valor G do nó atual com a importância do próprio vizinho.

Em cada nó os valores de importância são negativos, de modo a que a um caminho de maior importância pertençam menores valores de G e F, de modo a poder ser mantida a lógica de A* de procurar e ordenar pelo menor valor, obtendo-se aqui um caminho de maior importância possível.

Como qualquer nó vizinho está em condições de ser adicionado ao caminho, e o objectivo é encontrar o caminho de maior importância, o potencial de um nó é decidido pela importância dos nós a que este tem acesso. Como tal, o valor futuro H de um nó é o menor valor de importância de todos os seus vizinhos.

Esta heurística é admissível, não irá sobrestimar a importância que ainda se pode ganhar até ao fecho do caminho porque a existência de vizinhos válidos indica que existe pelo menos mais um passo que pode ser dado, e esse passo irá ser o que adicionar maior importância disponível ao caminho, tendo em vista o objetivo.

Será também consistente,

Detalhes Adicionais de Implementação

A ordenação da lista aberta do algoritmo A* é feita utilizando um *comparator*, que compara dois nós do seguinte modo:

1. Se valores F diferentes, o nó menor será aquele com menor valor F.
2. Pela importância, se valores F iguais. O menor nó será aquele com menor importância.
3. Pela distância, se valores de importância iguais. O menor só será aquele mais perto do nó atual.

O projeto recorre ao ficheiro de texto latlongDB, localizado no directório *resources* para recolher para cada nó valores de latitude e longitude referentes à localidade. Estes valores são utilizados para:

1. Distribuir os nós no interface. As coordenadas cartesianas x e y são calculadas assumindo que a Terra é uma esfera:

$$x = R * cos(lat) * cos(lon)$$
$$y = R * cos(lat) * son(lon)$$

   Sendo R 6371 km, o raio da Terra.
   Estas coordenadas permitem a distribuição dos nós relativamente uns aos outros e são depois modificadas para serem apresentadas dentro da tela do interface. São encontrados os nós com maiores e menores valores de x e y, de modo a colocar o centro do grafo no centro da tela do interface.

2. Calcular a distância e tempo entre localidades. A distância entre localidades pode ser calculada recorrendo à formula de Haversine [1]. O tempo entre nós está dependente do método de transporte selecionado; a pé - 2 m/s ou de bicicleta - 4.2 m/s [2].

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$
$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$
$$d = R \cdot c$$

   Onde φ é latitude, λ longitude, R o raio da Terra.

Caso não se encontre na base de dados os valores do nó, o projeto recorre à API Geonames para procurar na sua base de dados os valores de latitude e longitude.

O nome alternativo do nó é a *string* utilizada para procura na base de dados exterior.

A API recolhe a latitude e longitude do primeiro resultado.

Convém assim uma pesquisa apriori da base de dados da Geoname para ter a certeza que as coordenadas se referem à localidade desejada.

O interface do programa providencia adicionalmente uma opção *Path,* que permite criar, para cada percurso diário, um circuito fechado em relação ao hotel, de modo a tentar diminuir o ruído visual.

Para este efeito, cada nó tem um ângulo, calculado tendo o hotel como âncora, em relação ao eixo x. Os caminhos resultantes da aplicação do algoritmo são então organizados segundo este ângulo, de modo a criar um circuito fechado em que os caminhos não se intersectem.

Interface

O interface está dividido em três secções: menu principal, tela de desenho e área de texto lateral.

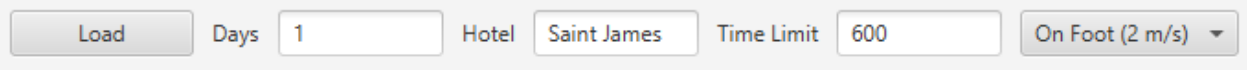

Figura 2 - Interface Parte 1

- Load: Permite o carregamento de um ficheiro de texto que descreva um conjunto de localidades. O directório é a pasta *graphs* dentro do directório de trabalho atual.
- Days: Campo de entrada para introduzir o número de dias.
- Hotel: Campo de entrada para selecionar o hotel.
- Time Limit: Campo de entrada para introduzir o tempo limite, em minutos.
- Método de Locomoção: Seleção do método de locomoção para cálculo de tempo de viagem entre localidades. Providecia dois métodos: a pé e de bicibleta.

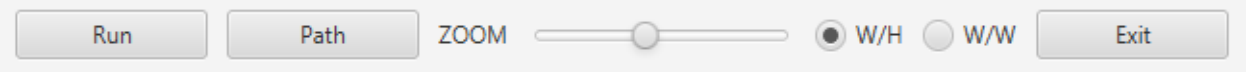

Figura 3 - Interface Parte 2

- Run: Corre o algoritmo A*, uma vez para cada dia.
- Path: Permite a criação de um caminho fechado em que as arestas não se intersectem.
- Zoom: Slider que permite ajustar o nível de zoom da aplicação

- Butões Rádio W/H e WW: Permite mudar o rácio de desenho dos nós.
- Exit: Sai da aplicação.

Na área de texto lateral são impressos os percursos para cada dia.

Estrutura

Módulos

O projeto foi dividido nas seguintes packages:

- AStar: Aplicação do algoritmo
- Comparators: Comparadores de nós e ângulos
- Database: Recolha e adição de dados à base de dados
- Geonames: Acesso à API Geonames para recolha de latitude e longitude
- Graph: Objectos Graph e Node
- GUI: Classe principal que invoca o interface - Tourist - e classes responsáveis pelo menu, tela de área de texto.
- Utilities: Classes utilitárias.

Diagrama de classes

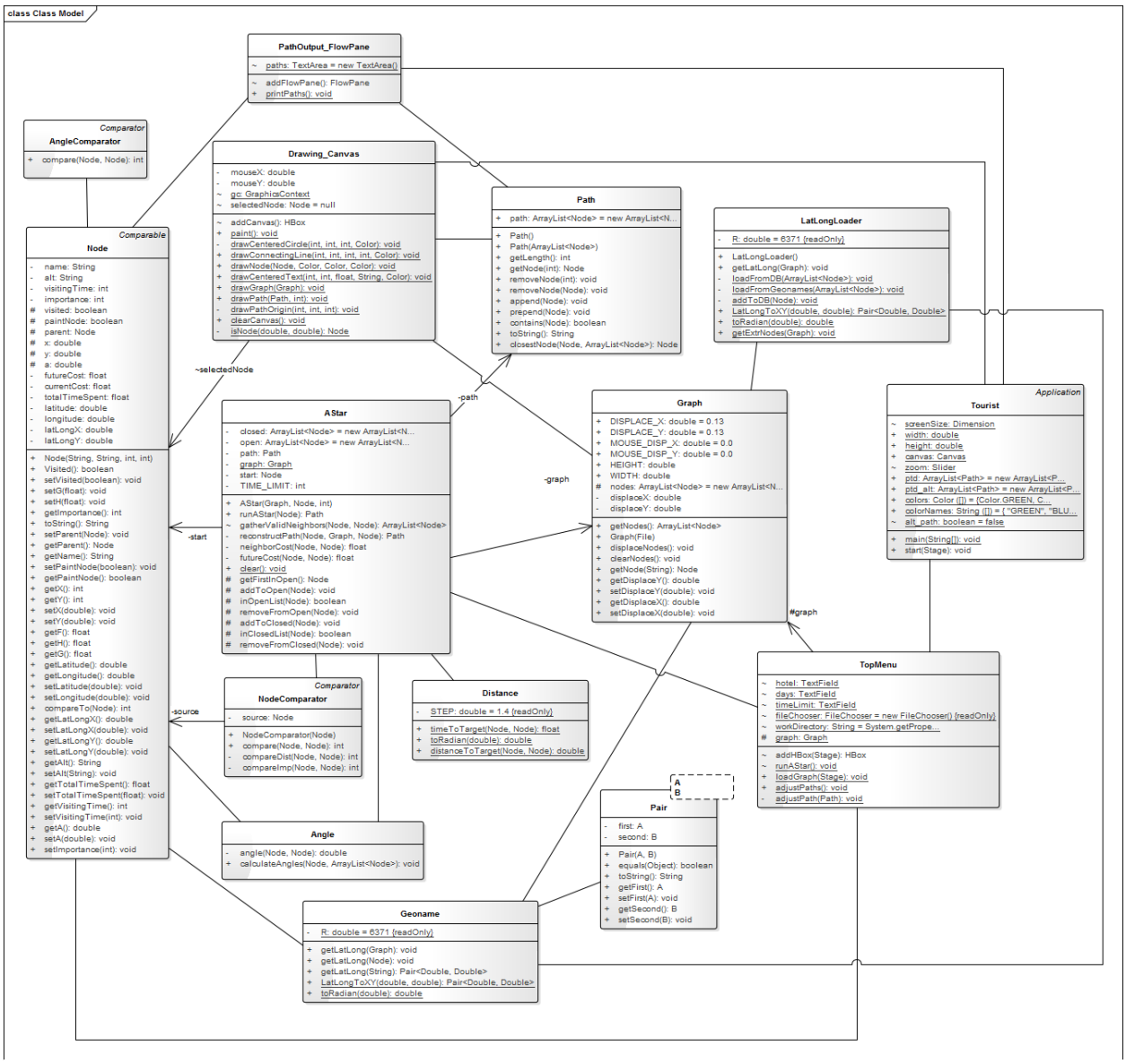

Figura 4 - Diagrama de Classes

Experiências

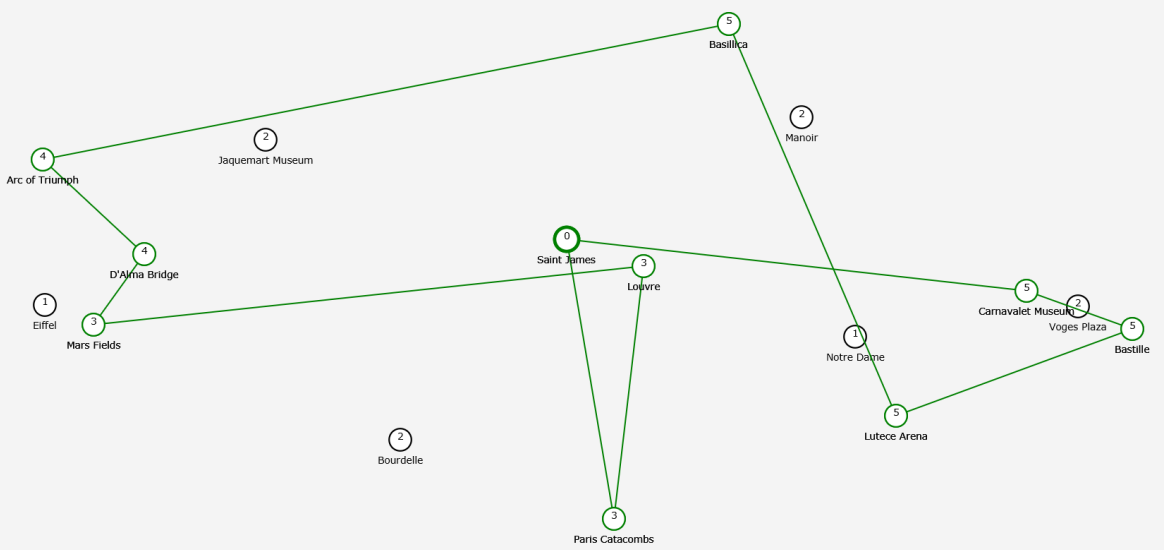

Figura 5 - Teste 1

Com um tempo limite de 300 minutos, tempo de percurso de 294 minutos, passo de 2 m/s e 1 dia, não é possível percorrer todos os nós. É possível verificar que o percurso tem como início e fim o hotel, Saint James, começando por visitar nós de maior importância disponível, levando em conta a sua proximidade, e que a importância dos nós decresce à medida que o caminho é construído.

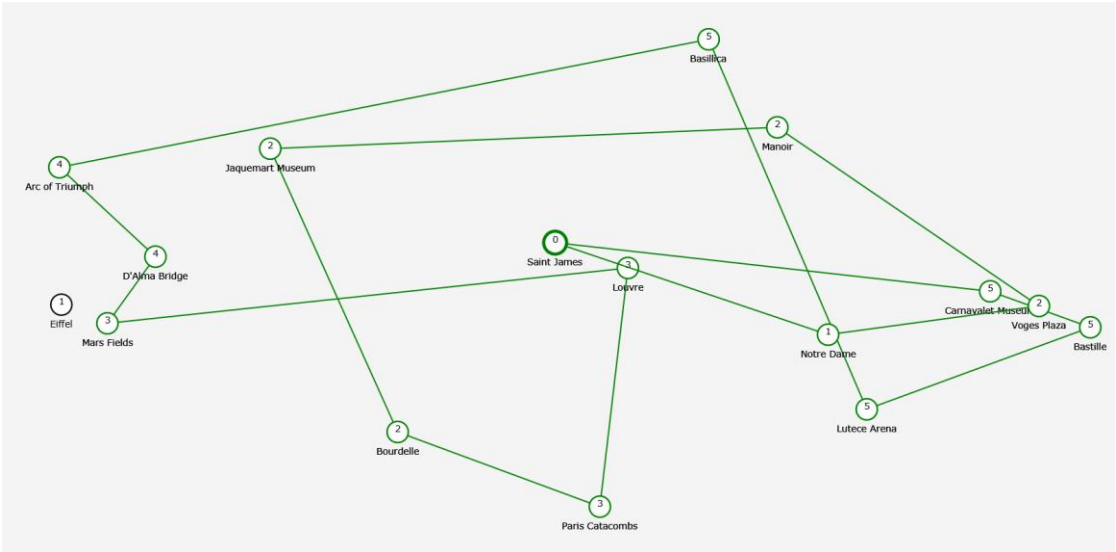

Figura 6 - Teste 2

Com o mesmo tempo limite de 300 minutos mas com passo de 4.2 m/s, o caminho pode agora ser feito em 277 minutos, conseguindo incluir mais nós mas não sendo suficiente para cobrir todos.

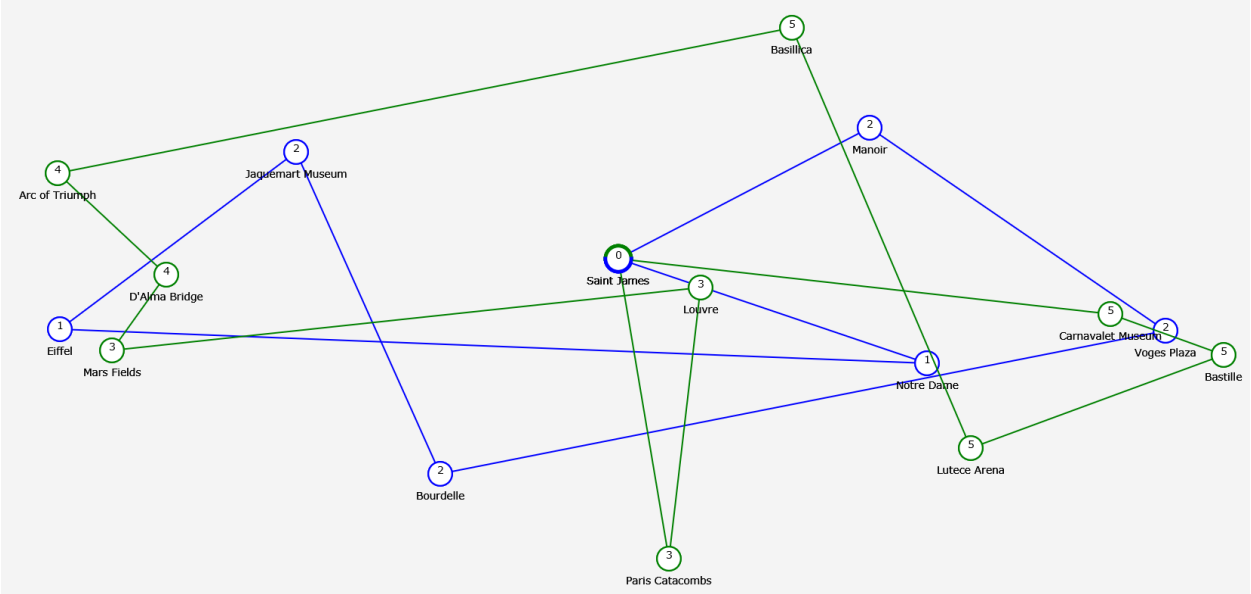

Figura 7 - Teste 3

Com tempo limite de 300 minutos, passo de 2 m/s e 2 dias, é possível verificar a construção de vários caminhos, que correspondem a diferentes dias. O caminho do primeiro dia, a verde, será o mesmo do 1ª teste. O segundo caminho ,a azul, irá cobrir os restantes nós, começando por visitar nós de maior importância disponível, levando em conta a sua proximidade.

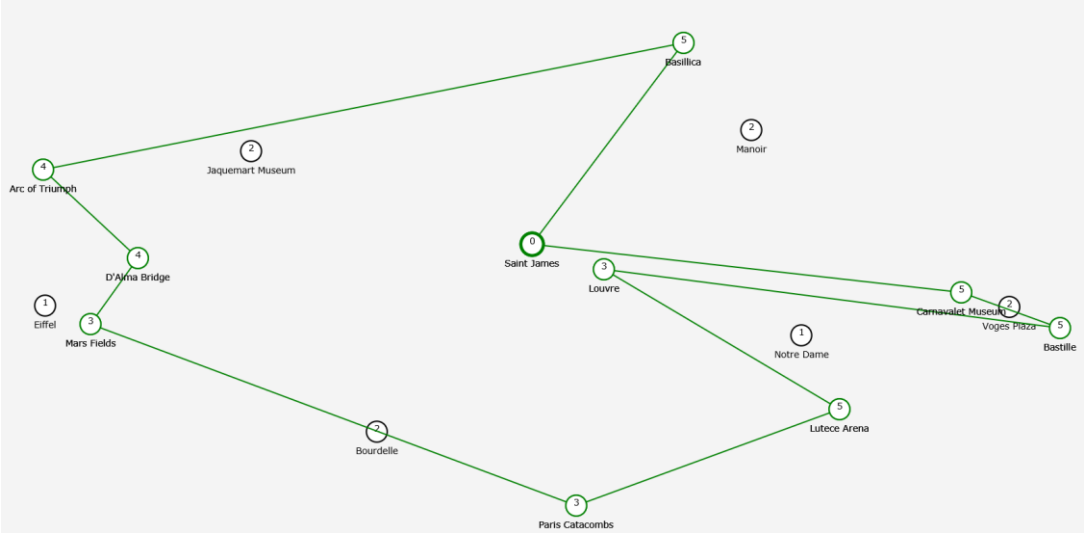

Figura 8 - Caminho fechado

Criação de um caminho fechado sem arestas que se intersectem, baseado no 1ª teste.

Conclusões

O desenvolvimento do projeto alcançou o objectivo, sendo assim capaz de, num grafo representando uma cidade e as suas localidades, encontrar um caminho de maior importância, iniciando e terminando o percurso na mesma localidade, podendo extender-se por vários dias.

Pode-se concluir adicionalmente que o problema apresenta semelhanças aos problemas Knapsack e Traveling Salesman, pelo que métodos de resolução focados em optimização poderão ser capazes de resolver o problema de um modo mais eficaz, visto que A* é normalmente utilizado para encontrar o percurso mais curto entre dois pontos.

Melhorias

- Ter mais escolhas para o utilizador, de modo a melhorar a personalização do percurso, como métodos de transporte, custos monetários, e classificação das localidades por tipo (e.g. igreja, museu).
- Considerar o horário de funcionamento de cada uma das localidades no planeamento do percurso e rotas de transportes públicos.

Desenvolvimento

Sistema Operativo: Windows 7 / 8.1

IDE : Eclipse Luna com plugin e(fx)clipse

Projecto desenvolvido em Java

Interface desenvolvido em JavaFx

Bibliotecas adicionais

Cliente Java para GeoNames Webservices (http://www.geonames.org/source-code/)

JDOM, utilizado para interpretação de XML (http://www.jdom.org/)

Elementos do Grupo

João Cardoso – Algoritmo, Interface – Total: 75%

Valter Silva – Algoritmo – Total: 25%

Bibliografia

[1] Movable Type Scripts (Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript)

http://www.movable-type.co.uk/scripts/latlong.html

[2] Jensen, Pablo, Jean-Baptiste Rouquier, Nicolas Ovtracht, and Céline Robardet. "Characterizing the Speed and Paths of Shared Bicycle Use in Lyon." Transportation Research Part D: Transport and Environment, 2010

http://arxiv.org/ftp/arxiv/papers/1011/1011.6266.pdf