Notas sobre TP1

SRCR

Pertende-se, partindo de um conjunto de paragens e carreiras, fornecido pelo professor, explorar a informação subjacente, nomeadamente, no âmbito do calcúlo de trajetos ótimos recorrendo a estratégias de pesquisa não-informada e informada. Porventura, podemos vir a estender predicados adicionais a este sistema.

1 Sobre os Datasets

É nos fornecido de partida dois conjuntos distintos de datasets que se completam entre si. O primeiro, diz respeito a todas as paragens de autocarro do concelho de Oeiras, enquanto que o segundo diz respeito a todas as paragens de autocarros organizadas por carreira e de forma sequencial.

Penso que só o segundo dataset é necessário para o que se pretende realizar, não temos interesse nas paragens individuais mas sim na sua adjacência para com outras. Porém, o primeiro dataset tem o objetivo máximo de nos fornecer com dados relativos ao povoamento das paragens. Que engloba os seguintes dados, com os seus potenciais usos:

- **Gid** Representa o identificador unívoco do ponto no espaço, este ponto pode ser utilizado para futuramente indicar os dois pontos sobre os quais pretendemos fazer a travessia. Por exemplo, indicar que pretendemos obter o melhor caminho do gid 10 até o gid 26. Omitindo assim toda a informação subjacente a esta paragens.
- **Latitude** em conjunto com a longitude permite conhecer a posição GPS da paragem em questão.
- **Longitude** em par com a latitude permite obter a posição, com a posição podemos utilizador predicados muito mais interessantes, nomeadamente, calcular o tempo que demora uma certa viagem entre pontos. Também, pode ser usado para que a distância entre pontos seja minima.
- **Estado de Conservação** O estado de Conservação varia de Bom, Razoavel e Mau. Secalhar seria interessante imbutir um predicado que

permitisse apenas passar por paragens num bom estado de conservação, porém não sei até que posto isto sera muito interessante.

- **Tipo de Abrigo** Indica se a paragem em questão é fechada dos lados, ou aberta, ou, sem abrigo. Aqui provavelmente já será uma melhor ideia imbutir funcionalidades adicionais, porque intuitivamente, é o que faz mais sentido. Por exemplo, um cliente indicar que quer o caminho mais curto entre duas paragens, sem incluir paragens sem abrigo, o que podia ser muito útil em dias de chuva. Claro que se conseguirmos fazer este, então tanto o predicado acima como abaixo serão extremamente de realizar, pois são só extensões destas, *nonetheless*, não deixa de ser uma funcionalidade interessante.
- **Abrigo com Publicidade** Indica se uma dada paragem possui publicidade ou não, pode não ser muito interessante, mas tendo o comportamento acima, seria imediato fazer isto.
- **Operadora** Define a entidade que faz usufruto dessa paragem, por exemplo, em Braga essa entidade será, maioritariamente a TUB, em quase todas as paragens. Este predicado é muito interessante, pois podemos permitir ao cliente viajar sempre na companhia da sua operadora favorita.
- **Carreira** O campo carreira é uma lista que de todos os ids de todos os autocarros (carreiras) que utilizam aquela paragem em questão, com isto podemos criar um fluxo do trajeto realizado por uma carreira, sendo que esta vai viajar sempre entre aqueles pontos.
- **Código de Rua** O código da rua é muito interessante do ponto de vista de funcionalidades, facilmente podemos tirar partido desta funcionalidade e estender este comportamento de forma a encapsular a viagem entre diferentes ruas. Com a rua temos a freguesia, e por isso indicando só a rua deverá ser suficiente para fazer uma viagem entre diferentes regiões, sem precisar de indicar o código da paragem, sendo que desta forma o sistema ficaria agnóstico à ideia de distância.
- **Nome da Rua** Logo de partida , é possivel prever que qualquer tentativa de tentar retirar diretamente proveito desta funcionalidade, será inutil e pode resultar em imensos casos, até tendo em conta que tira partido de unicode, e que em prolog isto aparece desformatado. Ademais, se já temos o código de rua, torna-se um bocado inútil tirar partido do nome de rua, potencialmente este campo será só útil do ponto de vista de apresentação do cliente.
- **Freguesia** Freguesia, apesar de também poder resultar em erro, pode ser útil do ponto de vista funcional, pois assim podemos incluir comportamentos adicionais de muito interesse.

Com os dados acima é esperado que desenvolvamos um sistema completo, capaz de incluir todos os seguintes comportamentos base, porém, é recomendado que estes comportamentos sejam estendidos.

1. **Calcular um trajeto entre dois pontos**:

Predicado muito vago, isto pode ser praticamente qualquer coisa, por isso podemos usar simplesmente um algoritmo BFS ou DFS, penso que tendo qualquer outro ponto feito, este aqui será de grande facilidade.

2. **Selecionar apenas algumas operadoras para determinado percurso**:

Isto pode ser feito utilizando um função de *filter* sobre todos as paragens que forem consideradas, desta forma, usando a função acima como uma função de ordem superior, conseguimos com relativa facilidade indicar o domínio de paragens que pretendemos incluir. Esta função facilmente seria estendida para outros comportamentos, podiamos com grande facilidade estender isto para outras funcionalidade necessárias.

3. **Excluir um ou mais operadores de transporte para o percurso**:

Isto podia ser feito exatamente da forma que é feito acima, ou seja, ao termos o predicado de 2. feito, facilmente conseguimos estender este comportamente para o presente predicado.

4. **Identificar quais as paragens com o maior número de carreiras num determinado percurso**:

Este predicado envolve, primeiramente, obter um dado percurso, assumindo que um percurso corresponde a uma lista de tuplos (?), então para obter o resultado esperado será suficiente, mapear as paragens ao seu respectivo número de carreira, seguido de um sort por ordem inversa.

5. **Escolher o menor percurso, usando critério menor número de paragens** :

Para este efeito podemos utilizar diretamente um BFS, que assim que encontra o destino, devolve-o automaticamente, garantido assim que o percurso obtido é aquele com o menor número de paragens, identicamente, podemos utilizar um mecanismo semelhante, resolvendo esta questão utilizando pesquisa informada, nomeadamente `A*`, em que o custo de cada nó será correspondente a 1. (É preciso analisar a complexidade disto, e ver qual dos casos compensa mais do ponto de vista de complexida e/ou espaço).

6. **Escolher o percurso mais rápido, usando o critério de distância**:

Caso perfeito de pesquisa informada, é preciso que exista uma noção da distância que estamos a percorrer, para tal basta usar o algoritmo `A*`, sendo que este caso é completamente indiscutível. Podiamos também usar pesquisa gulosa, porém a solução não é ótima, pelo que o primeiro será muito mais útil.

7. **Escolher o percurso que passe apenas por abrigos com publicidade**:

Isto pode facilmente ser feito com uma função de filtro, como indicad nos casos acima. Penso que em qualquer tipo de algoritmo de pesquisa podem incluir a funcionalidade de pesquisa utilizando uma função de ordem superior.

8. **Escolher o percurso que passe apenas por paragens abrigadas**:

Igual ao cenário acima, usando um filtro é imediato.

9. **Escolher um ou mais pontos intermédios por onde o percurso deverá passar**:

Este aqui é que pode ser um bocado mais complexo, não sei muito bem como copiar este comportamento. Assume-se que pontos correspondem a gids, então provavelmente vamos ter de passar uma lista de gids.

E, de seguida, possivelmente calcular o melhor caminho entre os diferentes pontos, porém é preciso ter noção que isto pode não ser assim tão simples, por causa do problema de passar de fazer um trajeto mas que nao corresponde ao otimo passando por todos os pontos.

Deverá então, possivelmente, ser necessário garantir que os pontos intermédios passados como são interpolados de forma correta e que produza o menor caminho possivel entre pontos.

Assumindo um ponto de origem S e um ponto destino F, com um conjunto de pontos intermedios `T = {T_0, T_1, T_2, ..., T_N}`, então, é suficiente garantir que o cálculo do menor caminho deve ser feito da seguinte forma:

```
N_i <- menor_custo(S -> T)
```

Sendo que N_i representa o ponto com o qual S se deve ligar primeiro, de seguida bastava fazer:

```
N_i <- menor_custo(N_(i-1) -> T\(N_(i-1)) )
```

Provavelmente será adequado procurar algoritmos mais eficientes sobre este tema.