1.Introdução

O segundo trabalho prático tem por objetivo o estudo e concretização de mecanismos raciocínio automático de procura em espaços de dados.

Desta forma, foi implementada uma biblioteca de uso geral sobre os mecanismos base de procuras em espaços de dados designado por pee.

Através da biblioteca pee será posta em prática a sua utilidade para casos reais através de um exemplo de um sistema de transporte e também da resolução de um puzzle de oito peças.

2.Desenvolvimento

2.1 Raciocício automático

A exploração de opções pode ser feita através de: Raciocínio prospetivo ou Simulação interna do mundo. Neste trabalho iremos focar na simulação interna do mundo, onde se possui uma representação interna do mundo. A avaliação de opções pode ter como base a utilidade ou custo e em alguns casos ambos.

A estrutura do trabalho é feita através de estados, onde cada estado irá representar uma situação entre muitas situações possiveis. A dinâmica é feita através de operadores, que no contexto representam transformações. Quando aplicado um operador a um estado este transita de estado para um novo estado.

2.2 Processo procura

Durante uma procura são realizados vários processos, entre eles a definição do objetivo do problema. Após definido o objetivo qual é o estado inicial ou nó raiz, de seguida é feito um varrimento pelo espaço de dados até se chegar ao objetivo pretendido.

Mais concontretamento para a concretização da biblioteca foram realizados os seguintes conceitos:

- Estado: corresponde a uma situação possível de um problema, isto é, define uma situação. Em termos de biblioteca a classe abstrata Estado que permite em termos funcionais saber se um estado é o objetivo do problema e qual a identificação desse mesmo estado.
- Operador: o operador permite definir transformações, essas transformações são aplicados aos estados para provocar uma transição no mesmo. Na implementação o Operador é uma interface que permite aplicar um operador a um estado e provocar um novo estado e também permite saber o custo entre dois estados.
- Problema: define as peças fundamentais do problema tal como o estado inicial, o objetivo e quais os operadores a aplicar. Em termos funcionais a classe Problema tal como foi dito possui um estado que é o estado inicial e um array de operadores que foram aplicados ao estado.
- Solução: a solução é constituida pelo percurso entre o estado inicial e o estado final. Em termos práticos a Solucao é um iterador que irá percorrer os diferentes passos solução para construir o caminho desde o objetivo até ao nó raiz.

Nota: Num espaço de estados, quando se está num nó que não o nó raiz através do operador é possível obter o nó que antecede o nó corrente, sendo a ideia por de trás da construção da solução.

2.3 Métodos de Procura

- Procura em profundidade: procura que explora primeiro os nós com maior profundidade, isto é, explora primeiros os nós mais recentes. Esta procura depende da ordem de aplicação dos operadores. A implementação da classe ProcuraProf permite iniciar a memoria do tipo LIFO(last in first out) onde os primeiros nós a ser procurados são os nós mais recentes.
- Procura em largura: procura que explora primeiramente os nós com menos profundidade, ou seja, são explorados primeiro os nós mais antigos. A expansão para o próximo nível de profundidade so é feito quando todos os nós do nível anterior já tiverem sido expandidos. A implentação da classe ProcuraLarg á semelhança da profundidade inicia a memória neste caso FIFO(firt in first out) onde os primeiros nós serão os primeiros a ser explorados.
- Procura profundidade iterativa: consiste numa procura em profundidade contudo com algumas limitações. Essas limitações são tais que ele realiza procura em profundidade até ao nível requerido, caso realize toda a procura até esse nível é aumentado o nível de profundidade e assim sucessivamente até ao limite de profundidade máxima. A classe permite definir o incremento sucessivo de passos e a definição da profundidade máxima da procura, possuindo portanto o seu próprio método resolver.

2.4 Procura em grafos com ciclos

Esta abordagem de procura visa á não expansão de nós anteriormente explorados, pois procurar um nó anteriormente explorado provoca desperdicio de tempo e memória. Este tipo de procura tem por base uma avaliação dos estados, sendo a função f a responsável por avaliar os nós gerados. No caso do trabalho prático f é responsável por estimar o custo desde um dado nó n até á solução, onde a fronteira de exploração é ordenada por ordem crescente da função f(n). Em termos funcionais a classe MemoriaProcura é a responsável por guardar os nós explorados e realizar as funções de inserir, limpar entre outras nessa mesma lista.

2.4.1 Procura custo uniforme

A procura de custo uniforme é uma procura não informada tal como as dispostas anteriormente, isto, não dispõe de informação do domínio do problema. A estratégia de procura é explorar primeiro os caminhos com menor custo. A classe ProcuraCustoUnif possui somente o método f que retorna o custo do nó, a partir deste é possível construir o melhor caminho tendo o custo por base.

2.5 Escolha método procura

Aspetos para ter em conta num método de procura:

- Completo: existe a garantia de no caso de existir solução durante a procura esta será encontrada.
- Ótimo: existe a garantia de durante uma procura com várias soluções possiveis a solução encontrada será a melhor solução.
- Complexidade temporal: tempo necessário para encontrar solução.
- Complexidade espacial: memória necessária para encontrar solução.

Parâmetros caracterização dos métodos procura:

- f(x) é de ordem O(g(x)) se existem duas constante positivas x_0 e c
- Fator de ramicação (b) – número máximo de sucessor para um qualquer estado.
- Fator de procura (d) – dimensão do percurso entre o estado inicial e o estado objetivo.

Procura:

- Profundidade:
  - Completo: Não
  - Ótimo: Não
  - Complexidade temporal: $O(b^m)$, onde m representa profundidade da árvore de procura
  - Complexidade espacial: $O(bm)$
- Largura:
  - Completo: Sim
  - Ótimo: Sim
  - Complexidade temporal: $O(b^d)$
  - Complexidade espacial: $O(b^d)$
- Profundidade iterativa:
  - Completo: Não
  - Ótimo: Não
  - Complexidade temporal: $O(b^d)$
  - Complexidade espacial: $O(bd)$
- Custo uniforme:
  - Completo: Sim
  - Ótimo: Sim
  - Complexidade temporal: $O(b^{\lceil c^*/\varepsilon})$
  - Complexidade espacial: $O(b^{\lceil c^*/\varepsilon})$

2.6 Procura em espaços de dados

Este tipo de procura é um tipo de procura informado, isto é, a estratégia de exploração do espaço de estados tiram partido do conhecimento do domínio do problema para ordenar a fronteira exploração. Quer isto dizer que é uma procura guiada.

Função heurística representa uma estimativa do custo do percurso desde o nó n até ao nó objetivo. Esta função reflete conhecimento acerca do domínio do problema, pois será através dela que a procura se irá guiar. O seu cálculo depende somente do estado associado ao nó e do nó objetivo. Este comportamento é concretizado pela interface ProcuraHeur similar á interface Procura mas utiliza a classe ProblemaHeur. A classe ProblemaHeur obriga a implementação do método heuristica que será o método utilizado para calcular a heuristica tendo o conhecimento do dominio do problema.

Comparativamente entre g(n) e h(n), g(n) representa o custo real do percurso até n, enquanto h(n) representa uma estimativa do percurso de n até ao objetivo.

2.6.1 Procura Melhor-Primeiro

Esta procura utiliza uma função f(n) para avaliar cada nó gerado. f(n) faz uma avaliação do custo da solução através de um nó n, quanto menor o f(n) para o nó mais esperançoso é o nó. A fronteira de exploração será ordenada por ordem crescente de f(n). A classe ProcuraMelhorPrim permite criar uma Memória de Procura com uma prioridade, o método compare que compara o custo entre dois nós e a função f que será a dita estimativa até ao destino.

Esta procura possui três variantes:

- Procura custo uniforme (f(n) = g(n)) -> procura que não tira conhecimento a partir do domínio do problema pela função heurística.
- Procura sôfrega (f(n) = h(n)) -> procura que não tem em concideração o custo dos percursos explorados.
- Procura A* (g(n) + h(n)) -> procura faz a minimização do custo global da procura.

Na implementação da procura sôfrega graças à grande modularidade do código esta classe apenas implementa a função f que tal como foi dito é igual á heuristica do estado.

A implementação da procura A* é um comprimisso entre o custo uniforme e a procura sofrega pelo que a função f é a soma da função g(n) e h(n).