Trabalho prático individual nº 2
Representação do conhecimento e resolução automática de problemas

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2015/2016

3 de Dezembro de 2015

I Observações importantes

1. Este trabalho deverá ser entregue no prazo de 48 horas após a publicação deste enunciado. Os trabalhos poderão ser entregues para além das 48 horas, mas serão penalizados em 5% por cada hora adicional.
2. Submeta as classes e funções pedidas num único ficheiro com o nome "tpi2.py"e inclua o seu nome e número mecanográfico; não deve modificar nenhum dos módulos fornecidos em anexo a este enunciado. Casos de teste, instruções de impressão e código não relevante devem ser comentados ou removidos.
3. Pode discutir o enunciado com colegas, mas não pode copiar programas, ou partes de programas, qualquer que seja a sua origem.
4. Se discutir o trabalho com colegas, inclua um comentário com o nome e número mecanográfico desses colegas. Se recorrer a outras fontes, identifique essas fontes também.
5. Todo o código submetido deverá ser original; embora confiando que a maioria dos alunos fará isso, serão usadas ferramentas de detecção de copianço. Alunos que participem em casos de copianço terão os seus trabalhos anulados.
6. Os programas serão avaliados tendo em conta: correcção e completude (70%); estilo (10%); e originalidade / evidência de trabalho independente (20%). A correcção e completude serão normalmente avaliadas através de teste automático. Se necessário, os módulos submetidos serão analisados pelos docentes para dar o devido crédito ao esforço feito.

II Exercícios

Em anexo a este enunciado, pode encontrar os módulos semnet e tree_search . Estes módulos são similares aos que usou nas aulas práticas, mas com pequenas alterações. Deverá resolver os

exercícios exclusivamente num novo módulo com o nome tpi2, deixando intactos os módulos dados. No módulo tpi2_tests existem alguns testes para as funcionalidades pedidas.

1. O pequeno módulo de redes semânticas usado nas aulas práticas foi concebido para facilitar a entrada dos alunos no tema. Por essa razão, foram deixados de fora muitos aspectos que, num sistema mais profissional, teriam que ser considerados. O módulo `semnet` que se encontra em anexo, foi concebido com base no das aulas, mas tem algumas diferenças relacionadas com a verificação de tipos. Por exemplo, no módulo das aulas não é possível saber se uma associação está estabelecida entre dois objectos ou entre dois tipos. Também não existe maneira de definir se uma associação admite apenas um valor ou vários. O construtor da classe `Association` foi então modificado, passando a ter os seguintes argumentos:

   - `entity1` - Primeiro argumento da associação.
   - `name` - Nome da associação.
   - `entity2` - Segundo argumento (ou valor) da associação.
   - `cardin` - Cardinalidade da associação, que pode ser:
     - `None` - A usar em associações entre objectos, ou seja, `entity1` e `entity2` são necessariamente nomes de objectos.
     - `"one"` - A usar em associações entre tipos. Especifica que, quando a associação for usada entre objectos, ela admitirá apenas um valor. Exemplo: uma pessoa tem apenas um pai.
     - `"many"` - A usar em associações entre tipos. Especifica que cada objecto pode ter essa associação com vários. Exemplo: uma pessoa pode ter vários filhos.
   - `default` - A usar em associações entre tipos. Especifica um valor por defeito numa associação em que a cardinalidade é `"one"`. Exemplo: por defeito, a altura de um homem é 1.75.

   Uma vez que não faz qualquer validação, o módulo das aulas permite que uma qualquer cadeia de caracteres `"xpto"` seja usada pelo mesmo utilizador simultaneamente como nome de um objecto e como nome de um tipo. No módulo `semnet` agora disponibilizado, a classe `SemanticNetwork` possui três novos métodos para criar relações, os quais retornam `True` caso seja possível criar a relação e `False` caso contrário.

   - `add_member(user,obj,type)`
   - `add_subtype(user,subtype,supertype)`
   - `add_association(user,e1,name,e2,cardin,default)`

   Note que a implementação destes métodos está muito incompleta por falta de métodos auxiliares a usar nas verificações de tipos. É nessas funcionalidades auxilares que deverá agora centrar a sua atenção:

   a) Desenvolva um método `object_exists(user,obj)` na classe `MySemNet` que verifica se o objecto `obj` existe nas declaracoes de `user`
   Exemplos:

```
>>> z.object_exists('descartes','socrates')
True
>>> z.object_exists('descartes','homem')
False
>>> z.object_exists('descartes','merkel')
False
```

b) Desenvolva um método `type_exists(user,type)` na classe `MySemNet` que verifica se o tipo `type` existe nas declaracoes de `user`

Exemplos:

```
>>> z.type_exists('descartes','socrates')
False
>>> z.type_exists('descartes','homem')
True
>>> z.type_exists('descartes','number')
True
>>> z.type_exists('descartes','reptil')
False
```

c) Com objectivo de manter a consistência, é necessário verificar, para cada nova relação a acrescentar, se os tipos das entidades nessa relação são consistentes com os tipos já existentes na rede. Uma vez que, nesta rede semântica, é possível declarar uma associação concreta entre dois objectos sem declarar as características gerais dessa associação ao nível dos tipos, torna-se necessário desenvolver mecanimos de inferência de tipos. Assim, desenvolva os seguintes métodos na classe `MySemNet`:

- `infer_object_type(user,obj)` - Infere o tipo de um objecto tendo por base as declarações de `user`. O resultado deverá ser:
  - `<tipo>` - o tipo caso seja possível inferir
  - `"__unknown__"` - se `obj` existe, mas nao se sabe o tipo
  - `None` - se `obj` nao existe
- `infer_assoc_type(user,assoc)` - Infere o tipo de uma associação tendo por base as declarações de `user`. O resultado deverá ser:
  - `(t1,t2)` - em que t1 e t2 sao os tipos das entidades envolvidas na associação
  - `None` - caso a associacao nao exista

Estas duas funções dependem uma da outra, pelo que, para ficarem completas, terão que ser implementadas como funções mutuamente recursivas. No entanto, poderá obter parte da cotação resolvendo os casos mais simples.

Exemplo:

```
>>> z.infer_object_type('descartes','platao')
homem
>>> z.infer_object_type('descartes','marx')
None
>>> z.infer_object_type('descartes','filosofia')
__unknown__
>>> z.infer_object_type('descartes',1.85)
number
```

2. Nas restantes alíneas, deverá extender as classes SearchDomain e SearchTree do módulo tree_search fornecido em anexo. Este módulo é similar ao das aulas, com pequenas alterações:

   i) O SearchProblem foi alterado no sentido de tanto o estado inicial (initial) como o objectivo (goal) passarem ser listas. Passa pois a haver a possibilidade se atingir múltiplos objectivos a partir de múltiplas raízes.

   ii) O método search() já faz prevenção de ciclos.

As extensões devem ser feitas nas classes Wikipedia e MySearchTree do módulo tpi2.

Se seguir sempre o primeiro link de uma página na wikipedia, eventualmente irá chegar à pagina sobre Filosofia (Philosophy) [1].

O desafio que se coloca neste exercício é exactamente explorar este facto.

O número de páginas da wikipedia supera as 114.000.000, o que tornaria este desafio num problema que requere grandes capacidades de memória e processamento. Assim sendo, e como o tempo desta prova não permite mais, são fornecidos dois datasets que representam $< 5\%$ da wikipedia. Consequência deste facto é que é natural que muitos dos identificadores de páginas referidas nos datasets fornecidos não tenham correspondência, isto é, apontam para páginas que não existem no dataset fornecido. É pois importante salientar que o seu código TEM que lidar com essas falhas.

Os datasets fornecidos estão em formato JSON e podem ser facilmente importados em Python recorrendo ao modulo json[2]. Consulte a documentação do modulo para encontrar exemplos que lhe permitem ler um ficheiro JSON para um dicionário Python.

O ficheiro links−small.json contém um dicionário em que a chave é o identificador de uma página (número em String) e o valor é uma lista de identificadores das páginas correspondentes aos links existentes na página.

O ficheiro titles −small.json por sua vez contém um dicionário que faz o mapeamento entre o identificador de uma página e o nome da página.

a) Implemente a classe Wikipedia por forma a poder fazer pesquisas utilizando a classe SearchTree fornecida.

   Além dos métodos actions e result, a classe Wikipedia deverá ter um construtor que receba os dois ficheiros contendo o dataset:

```
def __init__(self, links_filename, titles_filename):
```

   Criando um `SearchDomain` chamado `wikipedia` deverá poder obter o seguinte resultado:

```
>>> p = SearchProblem(wikipedia, 'Social_thought', 'Philosophy')
>>> t = SearchTree(p, 'breadth')
>>> t.search()
['Social_thought', u'Philosophy']
>>>
```

b) Implemente a função all_paths_to_philosophy () que determina de entre todas as páginas da wikipedia que obedeçam a um critério definido através de uma função lambda todos os caminhos possíveis entre essas páginas e a página 'Philosophy'.

   Para realizar esta alínea aconselha-se a implementação da classe MySearchTree acrescentando-lhe uma nova função de pesquisa, não alterando o `search()` já existente.

   Exemplo em que se procuram todos os caminhos a partir de páginas que contenham a palavra 'Taekwondo':

```
>>> all_paths_to_philosophy(lambda p: "Taekwondo" in p)
[[u'Taekwondo', u'Philosophy'], [u'Taekwondo', u'2007', u'1970s',
u'Steve_Reich', u'Philosophy'], [u'Taekwondo', u'2007', u'Grenoble',
u'Physics', u'Philosophy'], [u'Taekwondo', u'2007', u'July_20',
```

[1] `https://en.wikipedia.org/wiki/Wikipedia:Getting_to_Philosophy`
[2] `https://docs.python.org/2/library/json.html`

```
u'Physics', u'Philosophy'], [u'Taekwondo', u'2007', u'May_10',
u'Olaf_Stapledon', u'Philosophy'], [u'Taekwondo', u'2007',
u'Pierre-Gilles_de_Gennes', u'Physics', u'Philosophy'],
[u'Taekwondo', u'2007', u'Republic_of_China_presidential_election,_2008',
u'Three_Principles_of_the_People', u'Philosophy'],
[u'Taekwondo', u'Gatka', u'Kshatriya', u'Glossary_of_terms_in_Hinduism',
u'Philosophy'], [u'Taekwondo', u'Pehlwani', u'Hanuman',
u'Glossary_of_terms_in_Hinduism', u'Philosophy'], [u'Taekwondo',
u'Pehlwani', u'Kshatriya', u'Glossary_of_terms_in_Hinduism',
u'Philosophy']]
>>>
```

III Esclarecimento de dúvidas

Pedidos de esclarecimentos de dúvidas enviados aos docentes (lsl@ua.pt, dgomes@ua.pt) serão respondidos aqui. Estamos também disponíveis em `http://detiuaveiro.slack.com`. Bom trabalho!

1. Surgiram-me algumas (pequenas) dúvidas sobre o exercício 1: Em 1 a) podemos considerar os objetos como o `d.relation.entity1` ? Em 1 b) podemos considerar os types como o `d.relation.entity2` ? Os dois exercícios estão um pouco semelhantes, mudando apenas alguns filtros. Este será o método correto a fazer?

   Resposta: Sim, tanto objectos como tipos podem aparecer como `d.relation.entity1` ou `d.relation.entity2`. Varia no entanto conforme o tipo de relação. As funções pedidas em 1a) e 1b) são parecidas, mas não iguais ...

2. No exercicio 1, o que é quer dizer com `obj` e `type`? É que nos exemplos existem funçoes que dão verdadeiro para relações que não são da classe `Member`:

   Resposta: São os significados habituais: objecto ou instância; tipo ou classe. Os objectos não aparecem apenas em `Member`.

3. Na questão 1 a), `object_exists(user,obj)`, se a cardinalidade de `Association` for `None`, então a `entity1` e `entity2` são automaticamente objectos como especificado no enunciado certo?

   Resposta: Correcto! Mais um exemplo:

   ```
   >>> z.object_exists('descartes','filosofia')
   True
   ```

4. No execicio 1c, não estou a perceber porque existe necessidade de se usarem mutuamente. E por isso não percebo porque o último teste dá `number`.

   Resposta: O tipo inferido de `1.85` é `number` porque ambos aparecem como segundo argumento em duas ocorrências da associação `altura`. A primeira ocorrência especifica que os mamiferos tem altura do tipo `number`. A segunda ocorrência especifica que a altura do `socrates` é `1.85`.

   A recursividade pode ser necessária quanto o tipo de um objecto é inferido a partir do tipo de uma associação com várias ocorrências e este por sua vez é inferido de outros objectos, etc.

   Mais alguns exemplos para a 1c:

```
>>> z.infer_assoc_type('descartes','professor')
('homem', '__unknown__')
>>> z.infer_assoc_type('descartes','amigo')
None
>>> z.infer_assoc_type('descartes','altura')
('mamifero', 'number')
```

5. No exemplo do exercício 2a, o search retorna um tuplo com o caminho e um número. É mesmo assim?

   Resposta: Não, basta retornar o caminho. (exemplo corrigido acima)

6. Não consigo perceber onde será preciso recursividade no exercício 1c, não vejo nenhum exemplo na rede semântica que me faça lembrar de um caso...

   Resposta: Veja este caso, com novas associações acrescentadas:

```
>>> z.add_association('descartes','bacon','professor','filosofia')
True
>>> z.add_association('descartes','platao','amigo','bacon')
True
>>> z.infer_assoc_type('descartes','amigo')
('homem', 'homem')
```