Trabalho prático individual nº 2

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2020/2021

08 de Janeiro de 2021

I Observações importantes

1. This assignment should be submitted via *GitHub* within 32 hours after the publication of this description. The assignment can be submitted after 32 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested functions in module `"tpi1.py"`, provided together with this description. Keep in mind that the language adopted in this course is Python3.
3. Include your name and number and comment or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi1.py"`.
4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part. Limit these discussions to the general understanding of the problem and avoid detailed discussions about implementation.
5. Include a comment with the names and numbers of the colleagues with whom you discussed this assigment. If you turn to other sources, identify those sources as well.
6. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved in plagiarism will have their submissions canceled.
7. The submitted programs will be evaluated taking into account: performance; style; and originality / evidence of independent work. Performance is mainly evaluated concerning correctness and completeness, although efficiency may also be taken into acount. Performance is evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teacher in order to appropriately credit the student's work.

II Exercícios

Together with this description, you can find modules semantic_network, bayes_net and constraintsearch . They are similar to the ones used in practical classes, but small changes and additions were introduced. In particular, there are new attributes in the class `Association` (module semantic_network):

- `cardinality` - can be `'single'` or `'multiple'`, that is, the association can take a single value or multiple values. For instance, a person has a single father but can have multiple children.
- `inverse` - specifies the inverse association.
- `invcard` - specifies the cardinality of the inverse association.

In constraintsearch , the search algorithm conducts constraint propagation as we implemented in practical classes.

Module `tpi2` contains some derived classes. In the following exercices, you are asked to complete certain methods in these classes. Any other code that you develop and integrate in other modules will be ignored.

The module `tpi1_tests` contains some test code, including a semantic network, a Bayesian network and constraint satisfaction problem. You can add other test code in this module. Don't change the semantic_network, bayes_net and constraintsearch modules.

You can find the intended results of `tpi2_tests` in the file `tpi2_tests.txt`

The responses to the main questions asked by students during this TPI will be collected in section III below.

1. Develop a method `individual_probabilities()` in class `MyBN` to compute the individual probabilities of all variables of the network. The result is given in the form of a dictionary.

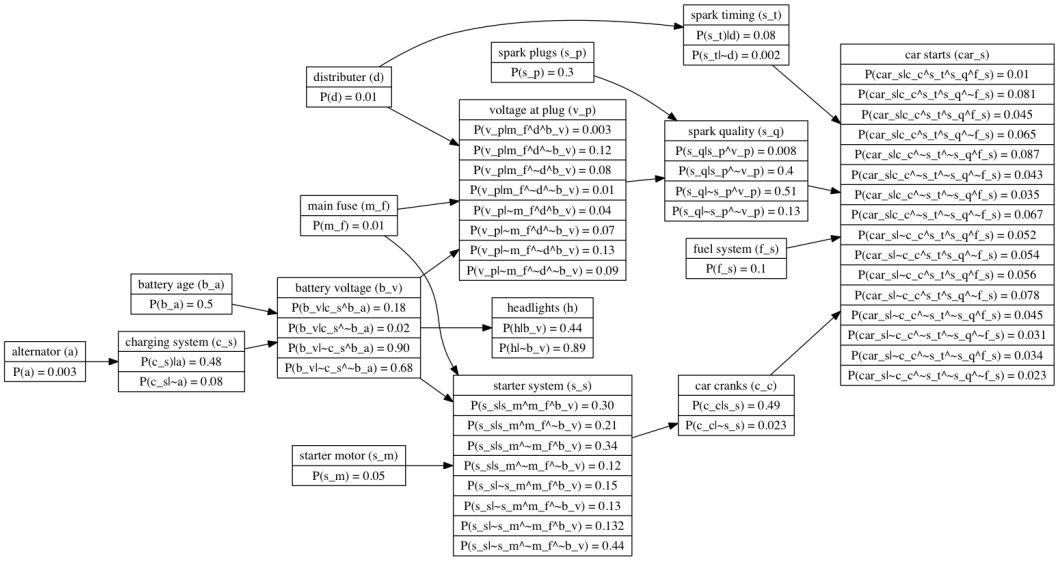

Figura 1: Bayesian network in the `tpi2_tests` module

2. Develop a method `translate_ontology()` in class `MySemNet` to translate the set of `Subtype` relations in the semantic network to first-order logic. The result will be a list of strings representing logical formulas. See the intended format in the results file. Note: The output of your method will be evaluated by comparing strings, so make sure your returned strings are exactly equal.

3. Develop a method `query_inherit(entity,assocname)`, in class `MySemNet`, that returns a list of declarations of associations that exist locally in `entity` or are inherited by `entity`. The returned associations should have the name `assocname` or the name of the inverse association, in case it is declared, as applicable.

4. Develop a method `query(entity,relname)`, in class `MySemNet`, that returns a list of values for a given relation in a given entity, including inherited values where applicable. The different types of relations are handled in different ways. The `Member` and `Subtype` relations should be handled in the usual way. Inheritance of associations with cardinality `'single'` is processed with cancelling, i.e. the existence of one of these associations in a given entity cancels the inheritance of a similar association with the same name in a predecessor entity. When there are several declarations with the same association in a given entity and this association has cardinality `'single'`, only the most common value should be returned. In the case of associations with cardinality `'multiple'`, inheritance is processed without cancelling. Therefore, all such associations found in predecessors are relevant and should be collected. Finally, because different users may use a given association with different properties (see method `assoc_properties()` in class `Association`), the properties most frequently used with a given association should be considered the correct properties of the association. Declarations of this association with different properties should be ignored. For example, `professorOf` was used four times with properties `('multiple', None, None)` and only once with properties `('single', None, None)`. Similarly, `eats` was used twice times with properties `('multiple', 'eatenBy', 'multiple')` and only once with properties `('multiple', None, None)`.

5. The `constraintsearch module`, provided together with this document, already has some improvements, including the powerful constraint propagation. There are, however, two limitations for which your help is sought:

   - The `search()` method of the `ConstraintSearch` class returns only one solution. Develop in the derived class `MyCS`, a similar `search_all()` method modified to return a list with all solutions.
   - Due to the number of original variables and the number of higher order constraints, the `TTWO+TWO=FOUR` problem (implemented in the test module) presents explosive combinatorics. The naive implementation of `search()` makes things even worse since, due to considering all possible sequences for selecting values for the different variables. Modify your `search_all` method to ensure that each solution is found only once. Note: This exercice will be evaluated taking into account correctness, completeness and efficiency.

III Clarification of doubts

This work will be followed through `http://detiuaveiro.slack.com`. The clarification of the main doubts will be placed here.