Trabalho prático individual nº 2

Inteligência Artificial + Sistemas Inteligentes I
Ano Lectivo de 2025/2026

12-13 December 2025

I Observações importantes

1. This assignment should be submitted via *GitHub* within 28 hours after the publication of this description. The assignment can be submitted after 28 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested functions in module `"tpi2.py"`, provided together with this description.
3. Include your name and number and comment out or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi2.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 assignment. If you turn to other sources, identify them 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 the module semanticnetwork. It is similar to the one used in practical classes, but small changes and additions were introduced. In particular, the original class `Association` was replaced by the following derived classes:

- `AssocOne` - an association that accepts a single symbolic value as second argument. Example: `hasFather` (a person has only one father)
- `AssocSome` - an association that accepts multiple symbolic values. Example: `likes` (a person may like meat, fish and vegetables)

The following methods were added to the `SemanticNetwork` class:

- `addInverse(assocname1,assocname2)` - declares that the two given associations are the inverse of each other. Example: the associations `fatherOf` and `hasFather` are inverse of each other.
- `addOpposite(assoc1,assoc1)` - declares that the two given associations are the opposite of each other. Example: the associations `likes` and `dislikes` are the opposites.

Both methods store the given information in dictionaries.

The `constraintsearch` module, also provided together with this document, already has some improvements, including the powerful constraint propagation. The `bayes_net` module is provided here without changes.

The module `tpi2_tests` contains example problems for testing. You can add other test code in this module. Don't change the `semanticnetwork`, `constraintsearch` and `bayes_net` modules. Module `tpi2` contains classes `MySN`, `MyCS` and `MyBN`. In the following exercices, you are asked to complete certain methods in these classes. All code that you may need to develop should me integrated in the same module.

1. In class `MySN`, method `new_query_local(e1,relname=None,e2=None)` - Returns a list of pairs `(relname,e2)` such that a relation `relname` exists between entities `e1` and `e2`. Note that arguments `relname` and `e2` are optional. This query does not involve inheritance, i.e. it is based on local declarations only. However, it should take into account information about inverse relations.
2. In class `MySN`, method `new_query(entity,relname)` - Returns a list of values for a given relation in a given entity, including inherited values. The different types of relations are handled in different ways. In which concerns `Member` and `Subtype` relations, only the local ones are relevant. Inheritance of `AssocOne` is processed with cancelling, i.e. the existance of an association of this type and with the same name in a given entity cancels the effects of a similar association in a predecessor entity. When there are several declarations of an `AssocOne` association, the most common value should be returned. In the case of `AssocSome`, inheritance is processed without cancelling. Therefore, all such associations found in predecessors are relevant. In addition, for both types of associations, there is inheritance cancelling when an opposite association exists. For example, `man likes meat`, but `socrates dislikes meat`, therefore that property of `man` will not be inherited by `socrates`. Finally, because different users may use a given association name with different associations types, the type of association most frequently used with a given name should be considered the correct type of the association. For example, `likes` was used several times as `AssocSome` and only once as `AssocOne`, therefore the type of this association is considered to be `AssocSome` and the only declaration in which it was used as `AssocOne` is ignored.
3. In class `MyCS(ConstraintSearch)`, method `search_all()` - It is similar to the original search method, but modified to return a list with all solutions. In addition, modify

`search_all()` so that, in each recursive call, only the most constrained variable is used for expansion. The most constrained variable is the one with the smallest domain. In case there are two or more variables with domains of equal sizes, the one that comes first in the alphabetical order is prefered. Note: This exercice will be evaluated taking into account correctness, completeness and efficiency.

4. In class `MyBN(BayesNet)`, method `independence_bag(v1,v2)` - When we need to compute individual probabilities in problems with a large number of variables, or joint probabilities involving a subset of all variables, the method of summing up all joint probabilities for all possible conjunctions of all variables, as presented in classes, becomes inefficient. To make such computations more efficient, one approach starts by computing an independence bag. Here, we focus on subsets of only two variables, i.e. v_1 and v_2. In this case, to compute the independence bag, we collect all variables along the paths connecting v_1 and v_2 to their common ancestors, excluding ancestors of ancestors. In a second step, we add to the independence bag the mothers of the variables collected in the previous step. The figure below displays the Bayesian network used for testing. Note: This exercice will be evaluated taking into account correctness, completeness and efficiency.

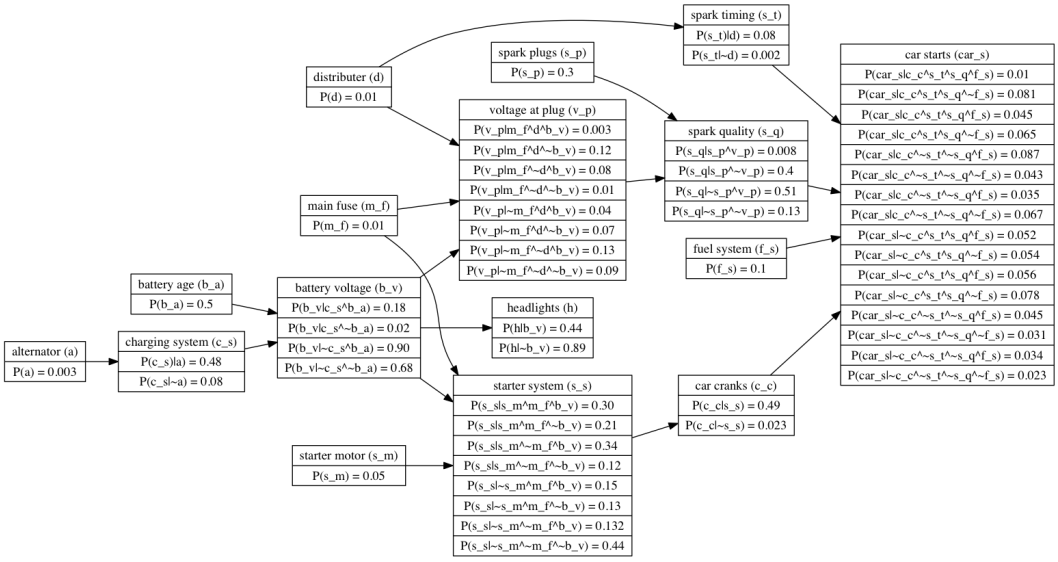

Figura 1: Bayesian network in the `tpi2_tests` module

III Clarification of doubts

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

1. ...

   Response: ...