CS 4260/5260: Introduction to AI

Programming Assignment 3 [100 points]: (Planning)

Due: Wednesday, December 5, 2018, 11:59 pm Central Time on BrightSpace

General Instructions:

If anything is ambiguous or unclear.

1. Discuss possible interpretations with other students, your TA, and instructor.
2. Make use of web sources.

You are able to work with up to two others as a team to implement this project. If you are not socially connected in the class, or otherwise don't know of others who you might work with, please email Doug at douglas.h.fisher@vanderbilt.edu with a subject line of 'team pool' giving your desire to be added to a team pool (that other team members can see), and the instruction team can facilitate team formation.

If you work as a team, there is no point penalty. But if you work alone, there is a modest (5%) boost to your score, even if it puts you over 100%. Because the class is NOT graded on a curve, this is not a "trick of the eye".

Remember that this course's honor code allows you to get help from others, so that whether you submit alone or you work as a team, you can receive help from others (outside your team), but you must disclose the specific help you received in comments at the top of your submission files. Please refer to the Honor code for more.

Your name(s) should be at the top of your submission file, together with declarations of help received. If submitting a project with two or three members, each team member should submit the file, and include all team member names at the top of their file. Again, each member will make a submission, but only one will be graded, so the submissions should be identical.

Introduction

In this project, you will write code that plans course schedules and estimates the expected GPAs for each schedule using a belief network. The belief network is also be used to revise GPA estimates for a student using a schedule given grades in certain courses found in the schedule.

The code for this project is stored in files inside the main directory folder. The file you will be editing is called **course-planner.rkt**. Example course catalogs are stored in the preq-lists directory.

Once you have finished, upload **ONLY course-planner.rkt** to BrightSpace. Do not upload anything else.

How to Run Your Code

The code for this project consists of several Racket files, some of which you will need to read and understand in order to complete the assignment, and most of which you can ignore. You can download all the code and supporting files as a zip archive.

Functions for loading and interacting with the inputs:

The following functions are provided in read-in.rkt. You will need to use them to interact with the inputs to your functions.

For scheduling:

```
(preq-list-parser filepath)
  "returns a hash map from course names (which are symbols) to a
list of possible prerequisite sets, each prerequisite set also
contains information on whether the course is actually a course
(like CS4260) or an abstraction (like CSmajor). The values are
based on the rows of a file in the preq-lists directory."
(course-name course-entry)
  "returns the name of a course object in a list of possible
prerequisite sets. This name should match the hash key for the
list in the map returned by preq-list-parser."
(course-type course-entry)
  "returns the type of a course object in a list of possible
prerequisite sets. This will either by 'COURSE or 'ABSTRACTION.
Abstractions are satisfied if their prerequisites are satisfied.
To satisfy a course, a schedule must include that course in
addition to satisfying the prerequisites."
(course-preq course-entry)
  "returns the list of prerequisite names for a course object in
a list of possible prerequisite sets. This will be a list of
course/abstraction names. To satisfy this prerequisite set, a
schedule must satisfy all the prerequisites in the list.
However, a course may have multiple prerequisite sets (course-
entries) stored in the catalog, only one of which must be
satisfied."
(hash-ref catalog course-name-symbol)
  "returns a list of prerequisite sets for the course with name
course-name-symbol that are stored in the hash map catalog.
Catalog is the output of the preq-list-parser function."
(get-courses catalog)
  "returns a list of all the course names in the catalog where
catalog is the output of preq-list-parser."
```

For estimating GPA:

```
(bayesian-network-parser filepath)
```

```
  “returns a Bayesian network based on a file in the bayesian-
networks directory. The network describes beliefs about the
grade a student will get in a course given their grades in other
courses. The grades are numbers between 0 and 4.”
(get-condition-names variable network)
  “returns the names of the courses that the probability of
variable depends on in the network. In this case variable is a
symbol name like ‘CS2301 and network is the output of bayesian-
network-parser.”
(to-condition-pairs conditions)
  “Turns a list of tuples into condition-pairs. For example:
(to-condition-pairs '((a 0) (c 4) (b 3)))
returns
(list (condition-pair 'a 0) (condition-pair 'c 4) (condition-
pair 'b 3))”
(get-beliefs variable network conditions)
  “returns the beliefs in the network about the course with name
variable given the condition assignments in conditions where
conditions is a list of condition pairs. It is easiest to make
this list with the to-condition-pairs function.”
(print-beliefs variable network conditions)
  “Prints the beliefs about a variable given the conditions in a
pretty easy to read way. For example:
(define network (bayesian-network-parser "bayesian-
networks/simple.txt"))
(define conditions (to-condition-pairs '((a 0) (c 4) (b 3))))
(print-beliefs 'd network conditions)
returns
P(d=0|a=0, b=3, c=4, ) = 0.12831981841656745
P(d=1|a=0, b=3, c=4, ) = 0.2029295881293878
P(d=2|a=0, b=3, c=4, ) = 0.24993287847190324
P(d=3|a=0, b=3, c=4, ) = 0.23973298201518906
P(d=4|a=0, b=3, c=4, ) = 0.1790847329669524”
(get-variables network)
  “returns a list of all the courses names in the network”
(get-child-names variable network)
  “returns a list of all the courses in the network that are
conditioned by variable.”
(belief-prob belief)
  “returns the probability of a belief where a belief is an
object in the list returned by get-beliefs. Each belief is of
the form P(<course>=<grade>|<conditioned-on>)=<prob>”
(belief-course belief)
  “returns the course name of a belief where a belief is an
object in the list returned by get-beliefs. Each belief is of
the form P(<course>=<grade>|<conditioned-on>)=<prob>”
(belief-grade belief)
```

```
  "returns the grade (course assignment) of a belief where a
belief is an object in the list returned by get-beliefs. Each
belief is of the form P(<course>=<grade>|<conditioned-
on>)=<prob>"
(belief-conditioned-on belief)
  "returns a list of the conditions of a belief where a belief
is an object in the list returned by get-beliefs. If a belief
has no conditions this list will be empty. Each belief is of the
form P(<course>=<grade>|<conditioned-on>)=<prob>"
(condition-pair-course condition)
  "returns the course that a condition pair is referring to.
Condition pairs are the objects in the conditioned-on list in a
belief and the objects in the list returned by to-condition-
pairs"
(condition-pair-grade condition)
  "returns the grade assignment of a condition pair. Condition
pairs are the objects in the conditioned-on list in a belief and
the objects in the list returned by to-condition-pairs"
```

The Course Catalog

The project will use data about courses and prerequisites. Course names and their prerequisites are stored in a file and are translated using `(preq-list-parser filepath)`, which is a function that you are given (as indicated above). This section is for conceptual background, but it will also let you modify course catalogs and generate new test cases. Here are some sample courses as found in a file that we will provide (without the comments).

```
COURSE CS1101                   " CS1101 is a course with no prerequisites "
COURSE CS2201 CS1101            " CS2101 is a course with CS1101 as prerequisite "
COURSE CS2212                   " CS2212 is a course with no prerequisites "
COURSE CS3250 CS2201 CS2212     " CS3250 has CS2201 and CS2212 as prerequisites "
COURSE CS3251 CS2201
      " CS3259 has 4 ways of satisfying prerequisites "
COURSE CS3259 CS2201 MATH2410 " or "
COURSE CS3259 CS2201 MATH2400 " or "
COURSE CS3259 CS2201 MATH2600 " or "
COURSE CS3259 CS2201 MATH2501 " prereqs for same course across lines is a disjunction "
      " thus each course has a disjunctive normal form description of prerequisites "
```

Note that each course is preceded by the COURSE keyword. This not only demarcates a line, but is added because there is another type of entry called an ABSTRACTION, which corresponds to the higher level requirements you find in the catalog or degree audit component of YES.

"The prerequisites of a requirement like CSMajor can be a mix of other ABSTRACTIONs (e.g., CScore) and COURSEs (e.g., CS4959 and CS1151) "

ABSTRACTION CSMajor CSmathematics CSsciencelab CSscienceb CSsciencec CSesintro CSliberalarts CScore CSdepth CS4959 CStechelectives CS1151 CSopenelectives CSwritingrequirement

ABSTRACTION CSMathematics CScalculus CSstatsprobability CSmathelective

"There are 5 alternative was of satisfying CScalculus "
ABSTRACTION CScalculus MATH1200 MATH1201 MATH1301 MATH2300 MATH2410
ABSTRACTION CScalculus MATH1200 MATH1201 MATH1301 MATH2300 MATH2600
ABSTRACTION CScalculus MATH1300 MATH1301 MATH2300 MATH2410
ABSTRACTION CScalculus MATH1300 MATH1301 MATH2300 MATH2600
ABSTRACTION CScalculus MATH1300 MATH1301 MATH2500 MATH2501

ABSTRACTION CSstatsprobability MATH2810
ABSTRACTION CSstatsprobability MATH2820
ABSTRACTION CSstatsprobability MATH3640

…

"The way that a higher level requirement like CSdepth is handled is to introduce four distinct high-level requirements, one for each of the three normally required CS courses and one for the project course "
ABSTRACTION CSdepth CSdepthproject CSdepthothera CSdepthotherb CSdepthotherc

"The way that a higher level requirement like CSdepth is handled is to introduce four distinct high-level requirements, one for each of the three normally required CS courses (otherA through otherC) and one for the project course. Note that these four prerequisite abstractions can be satisfied by the same courses so far as the syntax of a legal schedule, but this would not be legal in the real world, and ensuring that CSdeptha and CSdepthb, for example, are not bound to the same course would result in improved performance. "
ABSTRACTION CSdepthproject CS3259, CS3892, CS4269, CS4279, CS4287
ABSTRACTION CSdeptha CS3259, CS3282, CS3860, CS3861, …
ABSTRACTION CSdepthb CS3259, CS3282, CS3860, CS3861, …
ABSTRACTION CSdepthc CS3259, CS3282, CS3860, CS3861, …
….

There are many more courses and higher-level requirements. Again, the code for translating from a file into a data structure you can use for the catalog is provided to you. You can create your own test cases by creating new files, perhaps by modifying old ones, and rerunning the parser we provide.

Schedule-Advisor

For this project, you will implement a top-level function `schedule-advisor`, and subordinate functions `get-expected-values`, and `get-expected-gpa`. You can implement other helper functions as desired.

Write the top-level function called schedule-advisor, which is an interactive algorithm that finds two values: a schedule and the expected GPA of that schedule prior to any courses being taken. After returning and listing the first (schedule, GPA) pair, the user is queried as to whether they want to see another (*different*) schedule for the same goal conditions, and its estimated *a priori* GPA; if 'yes' then the next (schedule, GPA) pair is listed for the original goals, followed by the same query of whether to continue or not; this can continue indefinitely.

Schedule advisor's interface is

```
(define (schedule-advisor catalog goal-conditions network interact)
  (values <schedule> <expected GPA>))
```

where

· `catalog` is the return result of `preq-list-parser;`

· `goal-conditions` is a list of course and abstraction names;

· `network` is the result of `bayesian-network-parser;`

· `interact` is a Boolean that indicates that the first solution should be returned (false, indicating no prompting of the user) or that the user should be prompted to continue search for alternative schedules (true). If the user wants to continue to see alternative schedules in interactive mode, then the user type 'Y', else 'N' to exit the function (and receive the last solution as output).

When `interact` is true, the (schedule, GPA) pair that is last seen by the user (i.e., when the user enters 'N') is the values returned by the function. To say again, when `interact` is false, the first solution that satisfies the goals is returned.

In a (schedule, GPA) pair, the schedule is a list of "semester" lists. Each "semester" list must not contain more than 5 courses. It should not include abstractions, though as noted below, the abstractions that are goals should still be satisfied by the final schedule.

At the end of the schedule, all the goals in goal-conditions must be satisfied. This means prerequisites for the goal courses and abstractions must be satisfied for the goals as well. Each goal is represented using its name, which is a symbol, from the catalog.

The first semester in a schedule must exclusively contain courses with no prerequisites. For each following semester, any prerequisites must be satisfied by courses in previous semesters. (Aside: a more sophisticated planner would allow a non-empty initial state, since students can come in with AP and transfer credit, but your implementation should not do this).

A course should appear only once in a schedule.

You have flexibility in how to implement the scheduler. It could, for example, be a (heuristic) depth-first, regression scheduler that achieves a goal set of requirements. Or it could be a heuristic forward scheduler or some other implementation, but whatever the implementation, your header comments should clearly identify the AI constructs used in the implementation.

Also, whatever the implementation, it must implement an interactive functionality that allows a user to continue searching for alternative schedules. And each time a schedule is listed for the user, its expected gpa score will be paired with it and provided (printed) to the user too.

Think about adopting an approach for search, like the generic search algorithm, and instead of simply returning the first solution found, look to see the value of `interact`, ask the user whether to continue if interact is true, and continue search from the current frontier if that is indicated.

If no solution exists that is distinct from previously found solutions, then return **False, False**. **Schedules are distinct** from another if different courses appear anywhere in the schedule or the identical set of courses exist, but at least one of them is in different semesters.

Get Expected Grades

You will be implementing a function called `get-expected-grade-values` that calculates the expected grade for each course in a schedule given a list of known course grades from the schedule and a belief network. The output should be a list of condition pairs (see the `to-condition-pairs` function) in which each condition is a course and the expected grade of that course.

The interface for this function is:

```
(define (get-expected-values schedule network conditions)
  <list of condition pairs where each pair is a course name and
an expected grade>)
```

Hint: This function should first find probabilities for all courses conditioned on the known grades (integer of 0 through 4) for courses. Some of course variables will be found directly in probability tables, like those found by `get-beliefs,` but probabilities for other nodes will have to be inferred. From the probabilities for the various grades of the various courses, compute the expected grade of each course (real of 0.0 through 4.0)

Get Expected GPA

You will be implementing a function called `get-expected-gpa` that calculates the expected GPA of a student given a schedule, a belief network, and a list of grades the student has already received for the courses in the schedule. The expectation is that this function is called before a complete schedule is listed for the user, but an advanced (but not recommended implementation given class constraints) is that it could be called for partial schedules to guide search).

`Get-expected-gpa` will call `get-expected-grade-values`.

Functions for testing your code:

The following functions are provided in schedule-tester.rkt. We recommend using them to test your code before submitting:

```
(passes-test? schedule goal-list catalog)
  "returns true if the schedule is valid and fulfills the goals
in the goal-list"
(is-valid? schedule catalog)
  "returns #t if all semesters in the schedule have 6 or less
courses and all the prerequisites for each course are taken
before the semester that the course is taken in"
(goal-achieved? schedule goal-list catalog)
  "returns #t if all the goals named in the goal list are
fulfilled by the schedule"
```

Note that `passes-test?` is sufficient to test a solution, but you may wish to use `is-valid?` and `goal-achieved?`

Grading

There is a correctness/efficiency and style grade. Style, including commenting, will count 10%. What we are looking for in terms of style has been promulgated previously.

Of the 90% that remains for correctness and efficiency, 80% of that (i.e., 0.8 * 0.9 = 72%) is on the correctness/efficiency of the scheduler (without consideration of the estimated GPA estimation). Thus, you could obtain 72% + 10% = 82% if `schedule-advisor` simply called a stub `get-expected-gpa` function and your code was otherwise nice. Efficiency of the scheduler has a great deal to do with how interactivity is handled – do you implement a big-hack attack that restarts search on each continuance, accompanied by some checking that a previous solution is not "skipped over", or do you continue search more elegantly.

This leaves 18% for the correctness/efficiency of the belief network computing of expected gpa. Very little of this will be focused on efficiency, but rather it will be on the correctness of estimated gpa. One way for obtaining partial credit in the correctness is to make sure that your forward inferencing functions are working correctly (perhaps 50% * 18% = 9%), for a total of 91% when taking into account the scheduler correctness/efficiency and style. Backward reasoning capabilities, if done correctly, would get full credit.