CS5820 – Spring 2015 - Kaminski

Asgn 3 – Michigan route-finder
A comparison of search techniques

SUMMARY: This program compares the results of various INTELLIGENT SEARCHES with the OPTIMAL (informed) SEARCH (Dijkstra's Alg.) in terms of:

1) the cost of the SOLUTION
and 2) the cost of FINDING the solution.

It uses "my" GENERIC Prolog search code which ***NEEDS TO BE MODIFIED***

1) to make it work with your Prolog environment (probably no changes),
2) to make it work for this particular project,
3) to modularize it as described below.

Also, ***ADDITIONAL CODE MUST BE WRITTEN***

1) for this specific problem, ("the 4 rules")
2) to handle doing all the different searches and
3) for handling the batch processing & the report writing.

The search space is Michigan's main cities/roads (an explicit search space).

PROJECT "INPUT" TRANSACTIONS FILE: `CityPairs.pl`

Batch file of StartCity & EndCity pairs (1 pair per line) in Prolog-friendly format:
e.g., `startstop(kalamazoo, detroit).`

PROJECT "OUTPUT" REPORT FILE: `Report.txt`

For EACH CITY PAIR print out:
1) 1 line of #'s
2) 1 line containing the StartCity name & EndCity name
3) for EACH SEARCH print out:
   line 1: search name, Distance (from StartCity to EndCity),
   line 2: # Cities in the solution path, a space,
   then the list of cities from the StartCity to the EndCity
   (all in one line, but with line-wrap-around if needed when printing in NotePad)
   line 3: # Cities Closed, a space, then the list of the closed cities
   (using bagof's List) (all in one line, but line-wrap-around if needed…)
   [This includes both StartCity and EndCity – fix List if needed] !!!
   line 4: # Cities still Open, a space, then the list of the open cities
   (using bagof's List) (all in one line, but with line-wrap-around if needed…)

THE 6 MODULES: (all 6 are physically separate files):

1) **SearchSpace: `MichiganRoads.pl` `file`** *[MY file - don't change it]*
   Prolog-friendly format.
   Graph edges as: `dist(City1, City2, Distance).`
   Because of symmetry (undirected graph), YOUR code only accesses the `connected` rules, not the `dist` rules. This rule is already in this data file.

2) **EstimatorData: `MichiganLocations.pl` `file`** *[MY file - don't change it]*
   Prolog-friendly format.
   Graph nodes (city locations) as:
   `latlong(42.261596, -85.510095, kalamazoo).`
   NOTE: Latitude (1st) ~= YCoordinate, Longitude (2nd) ~= XCoordinate
   These are used for calculating HHat's.
   SEE NOTES BELOW ON AIR DISTANCE calculations.

3) **UserInterface**
   Contains ALL rules for doing END-USER (batch) I/O
   & related support rules like:
   - "reading in" a CityPair data (or…)
   - writing out the report data
   This does NOT include other consulting, like getting the EstimatorData. . .
   the Controller does that
   REMOVE any I/O code from generic Search file (& put it in here) !!!

4) **ProblemInterface**
   The 4 problem-specific rules to interface the Search with the SearchSpace
   including doing HHat calculations
   See comments inside the Prolog search code (especially the TopComment).

5) **Search** *[MY file - needs modifying]*
   Move appropriate code from this file to:
   UserInterface and/or ProblemInterface and/or Controller, like:
   - the problem-specific comments
   - any user-interface things (like writing the answer stuff & costs)
   - any cleanup that belongs in the controller
   Remove unnecessary code which doesn't apply for this project.
   Make appropriate changes for the various searches, etc.
   (However, try to make changes to the problem interface instead,
   where possible – like in calculating HHat)

6) **Controller** (does the following things - NOT necessarily in this sequence):
   - consults the necessary files (except the project "input" data)
   - contains the looping mechanism for processing the `CityPairs` data
   - handles the execution of the 7 searches including
     - setting/unsetting switches & values
     - cleaning up the temporary facts at appropriate times
   - calls rules in the UserInterface rules to:
     - get a StartCity and EndCity
     - print the report for that pair
   - includes self-starting & self-stopping code (once the program is working)
   - includes other management tasks, as needed

The 7 searches (done in this order)
The optimum (guaranteed minimum cost path solution) – an informed search:
1) Dijkstra's Shortest Path (Branch & Bound)

The intelligent searches:
2) A* (direct) (using straight air distance for HHat)
3) A* (via the Bridge) (using air distance via the Bridge for HHat, but see NOTE 4)
4) A* (via the Bridge + 10%) (ditto & see NOTE 4)
5) A* (via the Bridge + 50%) (ditto & see NOTE 4)
6) Best-first (via the Bridge) (ditto & see NOTE 4)
7) Hill-climbing (via the Bridge) (ditto & see NOTE 4)

NOTE: All the searches should select based on FHat (as the code currently does)
but: Dijkstra's uses: FHat = G + 0 (i.e., code sets HHat to 0)
The 4 A*'s use: FHat = G + HHat (code currently does this)
(the 4 different A* searches do different HHat calculations)
Best-first FHat = 0 + HHat
Hill-climbing FHat = 0 + HHat & it clears out opened each time

---

NOTE on CITY NAMES
All city names in various files are always 1-word (e.g., `grandRapids`) and always start with a small letter.

DEMO – What to hand in:
The Report file printed in NotePad using landscape, wrap-around, and small font to save paper.
Program files (in this order)
UserInterface
ProblemInterface
Search *[since you changed MY file]*
Controller

Some NOTES on Air distance calculations (used for HHat)

NOTE 1
The notes below are sometimes in Math/English/pseudocode/pseudoProlog.
Make sure to translate things into proper Prolog. For example,
- use `is` not = or ==
- Prolog has a built-in square root function
- calling functions is done differently (usually in several steps)

NOTE 2
(Reminder) Euclidean distance (AirDistance) calculations:

```
        A^2 + B^2 = C^2
so      C = sqrt (A^2 + B^2)
so      DistanceBetween2Points =
              sqrt( (X2 - X1)^2  +  (Y2 - Y1)^2 )
```

NOTE 3
EstimatorData is in DEGREES (for latitude & longitude) (used to calculate HHat).
But the StateSpace road data is in MILES (for determining G).
So, to convert degrees to approximate miles use:

```
airDistance(A,B,Distance) :-            % "Ben's correction"
      % http://mathforum.org/library/drmath/view/51717.html
      latlong(Lat1,Long1,A),
      latlong(Lat2,Long2,B),
      EarthRadius is 3963.1676,
      Pi is 3.14159265,
      D2R is Pi / 180,     % degrees to radians conversion factor
      AOB is acos(cos(Lat1*D2R) * cos(Lat2*D2R) *
            cos((Long1-Long2)*D2R) + sin(Lat1*D2R) *
            sin(Lat2*D2R)),
      Distance is EarthRadius * AOB.
```

This conversion is done in calculating HHat (find_hhat rule returns MILES),
NOT when the search calculates FHat.

NOTE 4 *
For calculating "via the Bridge" in searches #3 through #7, you can NOT just automatically use: Distance from the Node to the Bridge
PLUS Distance from the Bridge to the EndCity
You'll need to check if `StateIn` & `Goal` are:
1) both in lower peninsula (LP) or both in the upper peninsula (UP)
then just use:

```
      airDistance(StateIn, Goal, Estimate),
```

2) one is in the UP, and the other is in the LP – then you have to use:

```
      Actual Estimate =
            airDistance(StateIn, theBridge, Est1),
          + airDistance(theBridge, Goal, Est2),
```