CIS 730 Artificial Intelligence
CIS 530 Principles of Artificial Intelligence

Fall 2018

Homework 2 of 10: Machine Problem (MP2)

Heuristic Search, Part 1 of 2

Assigned: Fri 31 Aug 2018
Due: Mon 10 Sep 2018 on-campus, Wed 12 Sep 2018 distance
(before midnight)

The purpose of this assignment is to exercise your basic understanding of uninformed and heuristic searches through implementation. **In the next machine problem, you will implement and experiment with variants of A* and IDA*.**

Unless specified, each problem is worth 15% for CIS 730 students and 30% for CIS 530 students. Upload to K-State Canvas assignment a compressed file named `MP2.{zip | tgz | tar.gz}`.

Use **Python, Java, or C#** **only** to solve the first three problems. Specify which you are using in a `README.txt` file and name the programs accordingly: each problem should have source code files such as `mp2_i.[LANGUAGE]`. Including compilation instructions in your `README.txt` file. File names should be given on the command line, e.g., "`mp2_i < inputfile`". The data format is specified in the example below.
Note: You may use the built-in implementations in any programming language, but your program's input and output must conform to the specification in this assignment.

Data format

Input will be taken from **standard input** in the following format, AIGraph v3:

```
| The vertical bar denotes comments.  Ignore all input on lines containing '|'.
|
| The first non-comment line contains N, the number of nodes in the graph.
4
| The second optional non-comment line contains the unique start node
| and is terminated with an 'S'.
| NB: The first node is 0, but this is not necessarily the start node.
0 S
| The third optional non-comment line contains a single goal node
| and is terminated with a 'G'.
3 G
| The fourth non-comment line specifies J, the number of heuristics given.
| (You may ignore this unless you are doing the extra credit problem.)
1
| The fifth non-comment line starts the edge-cost data structure.
| Starting in 2017 two data structure types will be used: an adjacency list
| and an adjacency matrix (the original default).
| The next optional non-comment line begins with either #list or #matrix.
#list
| 1. If the line begins with #list, scan in an adjacency list: an array
| of linked neighbor sublists. The format of each line is
| neighbor_j.edge_cost_ij, with – denoting a link to the next neighbor.
| The NULL terminator is denoted !. Rows are indexed 0 to N-1.
| 2. For all other values of the fifth non-comment line, scan in an adjacency
| matrix: a square matrix A_ij of edge costs for the edge (i, j).  Absent edges
| (with infinite cost) are denoted *. Rows are indexed 0 to N-1.
```

```
|
| Example adjacency list:
| 0 → (1) 1 → 2 (2) → NULL
| 1 → 3 (5) → NULL
| 2 → 3 (1) → NULL
| 3 → NULL
|
| Example adjacency matrix:
| * 1 2 *
| * * * 5
| * * * 1
| * * * *
1.1 - 2.2 - !
3.5 - !
3.1 - !
!
| After the adjacency matrix, the heuristic evaluation vectors are given,
| where each row containing the heuristic value h(n) for a node n
| The example below is not admissible. Why? Does it matter?
3
6
1
0
```

Notes

- Post questions to the Canvas discussion for MP2 or to channel `#ai` on `kstate-cis.courses.slack.com`.
- **Python is recommended for this assignment.** You may use Java or C#, but the instructor has not tested the provided code extensively since the new releases this year.
- **Javascript, Julia, and Scala implementations are not yet supported for this assignment.** You may look at the code for design ideas. (Cite your source in README files and comments, even if you did not port any of this code to another programming language.) However, if you wish to use any of these languages for a term project, specify this in your project proposal and discuss it in your proposal and interim interviews with the instructor and teaching assistant. **The Lisp implementation is supported by the textbook authors but you must request the instructor's permission to adapt it for machine problems. The C++ implementation is obsolete and deprecated for the 2nd, 3rd, and upcoming 4th editions of AIMA.**
- Turn in a `README.txt` file with compilation and runtime documentation for the entire assignment.
- You may download and use any of the code archives from the third-party Github repositories and consult the *Artificial Intelligence: A Modern Approach* code page (https://github.com/aimacode). However, this code is provided **without warranty** and the detail level of documentation varies. You may prefer to write your solution from scratch and use only code provided by the CIS 530/730 instructional staff for testing and validation. **Note:** this supersedes the old AIMA code web page (http://aima.cs.berkeley.edu/code.html).
- See the note to be posted (after Lecture 5) in the Canvas discussions and announcements for guidelines on good coding style.

1. **(530 / 730) Adjacency lists and matrices.** Write a program to read in files in AIGraph v3 format and print out adjacency lists and matrices as shown in Lecture 3. A standard format and sample files will be posted after Lecture 5.

   Turn in: `mp2-1.py`, which takes an AIGraph v3 file as a command line argument (e.g, `mp2-1.py mp2-input-good1.aig`).

2. **(530 / 730) Calculating path costs: g function.** Write a program that reads in a space-delimited node sequence on one line, such as:

   ```
   0 1 3
   ```

   and calculates the total path cost for each node given a graph that has been read in. You should implement a function that actually calculates g(n) for each g given, as long as all nodes in the sequence are on a contiguous path. For the above sequence, the output should be

   ```
   0 1 6
   ```

   You should use an adjacency list for this.

   Turn in: `mp2-2.py`, which takes the same command line argument as MP2-1 above, plus input from the console (standard input), e.g., `mp2-2.py mp2-input-good1.aig < mp2-input-good1-path.txt`

3. **(530 / 730) Implementing Branch-and-Bound (B&B) Search.** Using your solutions to MP2-1 and MP2-2, implement Branch-and-Bound Search.

   Turn in: `mp2-3.py`, which takes the same command line argument as MP2-1 above, plus input from the console (standard input).

   For example:
   `mp2-3.py mp2-input-good1.aig`

   The end result should include the path and stepwise costs. Your program must print out the actual path and total cost in this format, when run from the command line:

   ```
   0 -[2]-> 2 -[1]-> 3
   Total path cost: 3
   ```

4. **(730 only). Testing B&B and Greedy search.** Run Branch-and-Bound Search, which you implemented in MP2-3. Compare your results to Greedy Search. You **may** use any provided code, but cite your sources in comments and your `README.txt` file.

   Turn in: `mp2-4.py`, which takes the same command line argument as MP2-1 above plus a command-line switch. e.g.,

   ```
   mp2-4.py -bnb mp2-input-good1.aig
   mp2-4.py -gbfs mp2-input-good1.aig
   ```

5. **(730 only: 40%) A/A*.** Implement A* as a synthesis of your two-part solution to MP2-4.

   Your program should be invoked as MP2-4 is:

   ```
   mp2-5.py -a mp2-input-good1.aig
   ```

6. **(530 only: 10%) Analysis.** Show that it is possible for a heuristic to be admissible but not consistent, but not the other way around. (The first proof is by counterexample; the second is by contradiction.)

Class Participation (required)

Post your draft project proposal by the due date specified in the syllabus.

Also, please post any unclear points you may have on search to the class mailing list. That is, ask questions about any search-related topic for which your understanding is unclear.

Extra Credit (10% each)

1. Attend the SIGAI fall kickoff meeting in the lower atrium of Engineering Hall at 1730 CDT Thu 06 Sep 2018. A sign-in sheet will be circulated.
2. Modify your program to be downward compatible with **AIgraph v1**, which does not use `#list` or `#matrix` and defaults to the adjacency matrix format, or **AIgraph v2**, which does not use 'S' after the start node or 'G' after the goal node. A sample `.aig` file (whose header indicates that it is v1) will be provided. Specify in your source file and `README.txt` which format(s) your program is compatible with.
3. Modify your search to handle `heuristic-number`, which selects one of the column vectors $\boldsymbol{h}j$ (where $0 \leq j \leq J$ and $0 \leq n \leq N$ as specified in the example comments above).