Assignment 1

CMSC 671 — Principles of Artificial Intelligence

Item	Summary
Assigned	Wednesday September 2
Due	Monday September 21
Topic	Search
Points	115

In this assignment you will gain experience with core AI search techniques. You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. You may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Discord discussions). You may use and reference code from `https://github.com/aimacode`, though acknowledge if you do so. Failure to do so, or the use of other external resources (including use of GenAI inconsistent with course policy) without prior written permission from the instructor, will be considered an academic integrity violation and result, in a minimum, in a 0 on this assignment.

The following table gives the overall point breakdown for this assignment.

Question	1	2	3	4
Points	30	15	30	40

How To Submit Submit the assignment on the submission site:

https://courses.cs.umbc.edu/graduate/671/fall26/submit.

Be sure to select "`Assignment 1`." You must turn in two items:

1. A writeup in PDF format that answer the questions.
2. A single `zip` or `tar.gz` file containing all code, environment files (if applicable) and execution instructions necessary to replicate your output.

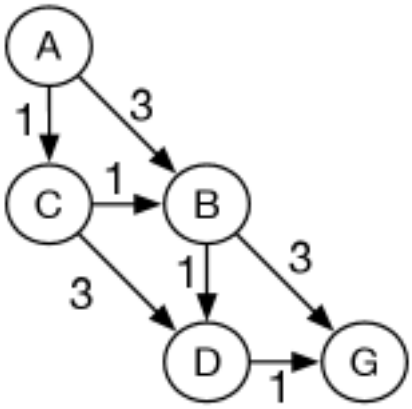
Questions

1. (**30 points**) From Ch 3.12 of Poole and Mackworth, answer the following two questions (mostly reproduced below for ease):

   (I) Question 1: Show the sequence of frontiers and path found for each of

   (a) Depth-first search (alphabetical tie-breaking)
   (b) Breadth-first search
   (c) Lowest-cost-first search
   (d) A* with the heuristic specified in the book.

   for the following graph.



   (II) Question 5: Draw two different graphs, indicating start and goal nodes, for which forward search is better in one and backward search is better in the other.

2. (**15 points**) This problem is set-up for the rest of the assignment. It involves optimizing $\mathbf{c}^\intercal\mathbf{x}$, subject to a system of linear equality constraints, $A\mathbf{x} = \mathbf{y}$, where $\mathbf{x}$ must be non-negative integers. If you need a refresher, see Appendix A.

Assignment Simplifying Assumption

Throughout this assignment, we will be concerned with the "satisfiability," rather than "optimality" problem. We will also only be concerned with the binary version, where each x_i is binary (0 or 1). However, especially when using a library like scipy or thinking about pruning strategies / path costs, it can be helpful to view it as a simplified "optimality" problem. So, to that end, you may assume $\mathbf{c} = \mathbf{1}$ when necessary or helpful).

   (a) Consider the following system of equations, with binary constraints on $\mathbf{x}$:

$$\begin{aligned} 5x_1 + 4x_2 &= 4 \\ 9x_1 + 3x_2 &= 0 \end{aligned} \tag{1}$$

   (i) Identify A and $\mathbf{y}$.

(ii) Solve a *relaxation* of this problem: that is, solve for $\mathbf{x}$ *without* binary constraints.

(iii) Use the SciPy simplex solver scipy.optimize.linprog to verify your above solution.

(iv) Now provide the solution for $\mathbf{x}$ with binary constraints, or show that it is not satisfiable. Discuss how this binary solution (or lack of one) relates to the solution for the relaxation above.

(b) Construct a system of equations with binary variables that has multiple possible solutions. That is, write out A and $\mathbf{y}$ such that there is more than one binary $\mathbf{x}$ that will make $A\mathbf{x} = \mathbf{y}$ true. Provide at least two of these values of $\mathbf{x}$.

3. (**30 points**) This question will help you think through possible agent and search design to solve integer (binary) systems of linear equations $A\mathbf{x} = \mathbf{y}$ as described in the previous question. Consider two different ways of representing the state $\mathbf{s}_t$ at time t: in option 1, $\mathbf{s}_t = \mathbf{x}$, that is, the state is simply the current values of our variables. Thus, with N variables, each state has dimension $O(N)$, and $\mathbf{s}_{t,i} = \mathbf{x}_i$. When dealing with non-unity optimization cost vectors $\mathbf{c}$ (i.e., $\mathbf{c} \neq \mathbf{1}$), additional state information related to cost may be needed.

   Option 2 is based off of the real-valued relaxation that you saw in question 2. Each variable $\mathbf{x}_i$ is instead represented by the current assigned value (if any) *and* lower and upper bounds, which specify an (increasingly narrow) range of feasible values for that particular variable. All of these may be *real-valued* (up to rounding/machine precision). Together, these assigned values (if any) and bounds constitute the state $\mathbf{s}_t$, where $\mathbf{x}_{t,i,v}$ is the current **v**alue (possibly None); $\mathbf{x}_{t,i,l}$ is the **l**ower bound for $\mathbf{x}_i$; and $\mathbf{x}_{t,i,u}$ is the **u**pper bound. Thus, $\mathbf{s}_t$ has dimension $O(3 \times N)$. As with option 1, when dealing with non-unity optimization cost vectors $\mathbf{c}$ (i.e., $\mathbf{c} \neq \mathbf{1}$), additional state information related to cost may be needed.

   This Option 2 representation is useful,[1] since it allows you to repeatedly solve the problem by iteratively constraining the bounds of each variable. That is, for a given state, you would solve the real-valued relaxation of the problem by fixing ("clamping") those variables that have current (non-None, *valid*) **v**alues and constraining the remaining free variables (those with None values) based on their respective lower and upper bounds. If a solution to this relaxation exists, then you can select an unconstrained variable x_i that has a fractional value (e.g., $\hat{x}_i = 0.7$) and "branch" (expand to the next level) based on resetting its lower value to rounding $\hat{x}_i$ *up* (and its upper value to rounding $\hat{x}_i$ *down*).

   (a) Discuss the advantages and disadvantages of each option, ***with examples***, specifically in terms of

   - the actions that your search may take upon these states;
   - how you would compute path costs under each option;
   - how you may be able to use these states to prune paths; and
   - how you would test for goal completion under each option.

   Remember that the goal completion predicate must not assume any "correct" values of $\mathbf{x}$: it must work for *any* binary $\mathbf{x}$.

[1] Option 2 is fundamentally how such "real world" solvers work, albeit with more aggressive pruning techniques than is considered here.

(b) For the system provided by Eq. (1), use Option 1 to provide the first two expansions of depth-first search. Use $\mathbf{x} = (0, 0)$ to help seed your initial state.

(c) For the system provided by Eq. (1), use Option 2 to provide the first two expansions of depth-first search.

4. (**40 points**) In this problem, implement "Option 2" to solve systems of integer linear equations.

   The specification for the system of equations will be provided by a JSON file. This JSON file is a dictionary with two keys: `A`, which maps to a 2-dimensional array of coefficients, and `y`, a 1-dimensional array of output values. See

   `https://courses.cs.umbc.edu/graduate/671/fall26/materials/a1/code/q2a.json`.

   for an example encoding of Eq. (1). (We *will* test your program on other systems with binary variables. For full credit, your program must work appropriately for these other files.)

   Specifically:

   (a) Implement a basic agent that is capable of solving a provided system of linear equations with binary variables. Demonstrate that your solver can find *a* correct solution by applying it to the system in Eq. (1), and the problem given by `https://courses.cs.umbc.edu/graduate/671/fall26/materials/a1/code/7x10.json`.

   (b) Discuss how you compute costs, both current and heuristic. Identify whether your "heuristic" is admissible or not, and provide proof of this. (Note: you can at least partially answer this even if your solution to part (a) is incomplete.)

   (c) Discuss how you select which variable to branch on. (Note: you can at least partially answer this even if your solution to part (a) is incomplete.)

   While not required, I **highly** suggest you also test your program on the examples you came up with in 2(b).

   For this assignment, you may use code provided by the `aimacode` github organization: `https://github.com/aimacode`. There is a `search` file, which contains the implementation of the different search algorithms. If you are coding in Python, this will be `search.py` in the `aima-python` repo. You are welcome to use these pre-implemented search functions, but you are not required to.

A Systems of Equations Overview

You may remember *systems of equations* from your math classes. They are M different equations of N variables. For example,

$$5x_1 + 4x_2 = 9$$
$$9x_2 = 9$$

is a system of $M = 2$ equations with $N = 2$ unknown variables (x_1 and x_2). In the *feasibility* problem, the goal is to find *some* values of the variables x_1 and x_2 that make the equations in the system true; in the *optimization* problem, the goal is to find values of $\mathbf{x}$ that achieve some optimal objective value. This objective is computed as a linear combination of the variables, $\mathbf{c}^\intercal\mathbf{x}$, where $\mathbf{c}$ is a pre-specified vector of variable-specific costs. For example, with $\mathbf{c} = (1, 1)$, the computed objective is simply 2.

Solving (either feasibility or optimality) systems of equations when the variables $\mathbf{x}$ are real-valued is relatively straight-forward: solving them when $\mathbf{x}$ are restricted to be binary values (that is, 0 or 1) is *hard*. That is, while it may not seem intuitive, allowing each x_n, for $n = 1 \dots N$, to be only either 0 or 1 makes the problem difficult to solve efficiently.

While it's convenient if $M = N$, it is not required. For example,

$$\begin{aligned} 5x_1 + 4x_2 &= 5 \\ 9x_3 &= 9 \end{aligned}$$

is a system of $M = 2$ equations with $N = 3$ unknown variables. In general, a system of equations can be written as

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,N}x_N &= y_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,N}x_N &= y_2 \\ &\cdots \\ a_{M,1}x_1 + a_{M,2}x_2 + \ldots + a_{M,N}x_N &= y_M. \end{aligned} \tag{2}$$

The $a_{m,n}$ are the coefficients, and y_m are the required output values. If we use matrix-vector notation, where the matrix (2-d array) A stores the coefficients, the one dimensional vector (array) $\mathbf{x}$ stores the variables and the one dimensional vector (array) $\mathbf{y}$ stores the output values

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,N} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,N} \\ & & \cdots & \\ a_{M,1} & a_{M,2} & \ldots & a_{M,N} \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_M \end{pmatrix}$$

then with matrix-vector multiplication we can write Eq. (2) compactly as $A\mathbf{x} = \mathbf{y}$. In the optimality problem, we optimize (generally minimize) $\mathbf{c}^\intercal\mathbf{x} = \sum_i c_i x_i$, subject to the constraints that $A\mathbf{x} = \mathbf{y}$.