**Assignment 2**

# Relation to Course Learning Objectives

This assignment pertains to:

**CATEGORY 1 - General artificial intelligence objectives**

1. Explain, select and apply basic artificial intelligence techniques to solve simple or simplified problems.

**CATEGORY 2 - Topic-specific learning objectives**

1. ***Problem solving***: Use the general problem solving framework to describe a problem.
2. ***Search***:Select and apply different search strategies to find the solution to simple problems.

# The Task

In this assignment you will join forces with another student in the class to implement a “search shell” for solving one-player games and puzzles such as the 8-puzzle we used in class. What is a search shell? It is a general program that can be applied to different puzzles by plugging the type (or class) definitions and related functions that are puzzle specific. A search shell can use different search strategies on the same puzzle. This means that you must write your search shell program in a way that clearly separates the general search engine of the program from the parts that are puzzle-dependent. To do so, your search shell must make use of the general concepts we saw in class for problem formulation and for search strategy. The pseudo-code algorithms are in the slides and in the book.

**Problem Formulation**

* A puzzle goes through different **state**s. You need to have a computational representation of a state in the puzzle, which you use to represent the **initial state** and any other state the puzzle.
* One or more of the puzzle’s states is a **goal state**. You need to be able to test for when you have reached a goal state by applying the **goal test**.
* The puzzle changes state by applying **operator**s or **actions**. The **successor function** (or **transition model**) gives you the possible states that an input state can transition to based on the available actions that are applicable to that state. You need to have such a function that takes a state as input and returns a set of states that you may move to.
* The **path cost** function, which can be coded as the sum of the **step cost**s, i.e. the cost of each action. The **step cost** is associated with each action. It is a unit cost in many puzzles, but would not be a unit cost if applying search to find the shortest path in a network. You should allow for variable step costs.

Depending on the puzzle, the **initial state** and the **goal state** can change from one run of the program to the other, so you need to be able to give these as inputs to the program somehow. One option is to read them from a file; another is to type them in, though this may be tedious.

Note that, depending on the game, not all actions are performable in a given state. Your successor function should return states reachable by applying performable actions. Also, some actions may be performable but lead to an **illegal state** (a state where the game ends in a loss). Your program needs to check somewhere, whether in the successor function or elsewhere, for illegal states. Finally, actions may loop back to a state that has already been seen and expanded. To avoid potential infinite loops, you need to keep track of already expanded states by using a **closed list**.

**Search Strategy**

Your search shell should be able to apply different search strategies to the same puzzle upon request from the user. As discussed in class, in general the search strategy is really defined by two things:

1. In uninformed or blind search, how you manage the **frontier** (also called **fringe** or **open list**), i.e. how you insert in the frontier new states generated by expanding a node.
2. In informed or heuristic search, what evaluation function you use: do you use only heuristic distance from the state being evaluated to the goal or also the path cost?

Hence your search shell should be able to take as input the strategy to be used and, when relevant, a heuristic function defined in a fashion appropriate to the puzzle you are solving. E.g. for the 8 puzzle, there is a computation involved; for finding your path through a network, the function might just do table lookup. Again, it might make sense to consider how to store such information in a file, rather than hard-code it in the program (where it will be harder to change) or enter it each time you run the program (which has its set of associated issues).

You should implement the following strategies: 1) **depth-first**, 2) **breadth-first** and **uniform cost** (you can implement them as one, the difference is in the step cost), 3) **greedy-best first**, 4) **A\***.

**EXTRA CREDIT:** If you want to shine, implement depth-first with the option of setting a **depth cutoff**, which will allow using depth-limited search and provide support for implementing IDS. Similarly, implement A\* with a cutoff for the evaluation function, which provide support for implementing IDA\*. Getting the iteration in there is harder. How would you do it? Do it if you can.

**Search Progress Reporting**

The program should keep track of how many search states it has expanded.   
The program should also report when it has tried to expand a state and found that it is on the closed list (i.e., it has detected a loop) and keep track of the number of times this happens.

These values (number of nodes expanded, loops detected) should be periodically reported while the search is in progress and also reported at the end of the search (whether successful or failed).

Note that you might not always be able to find the solution in a decent amount of time, so prompt the user to see if they want to keep going.

**Search Tree**

Keep in mind that the **search tree** that is developed in the process of the search is not the same as the **search space** (or **state space**). The search tree is developed as you explore the search space using a given strategy. You should have a clear representation for the **search nodes**, which refer to states in the search space, but also keep information that allow you to reconstruct the solution and know its cost. See the slides and/or the textbook. The **search node** is one of those elements of a search shell that is not puzzle-dependent.

**Puzzles**

You will be asked to try out your search shell on 2 different puzzles of a rather different nature.

**Puzzle 1**: This is the Pegs logic game you were asked to think about in Assignment 1, described again below. The URL: <https://en.wikipedia.org/wiki/File:Peg_Solitaire_game_board_shapes.svg> gives you some of the board options. The game board displayed below (4, the number refers to the one used in the Wikipedia page) is easier than one in Assignment 1 (2). You should try it with more than one game board, for example (1), (4), and (5), but you can restrict yourself to game boards that use a square grid (i.e. you can avoid (6)).

|  |  |
| --- | --- |
|  | The board at the left shows an initial configuration of the game, with pegs (brown circles) arranged in the given configuration. The gray square in the middle is a hole with no peg in it.The goal of this game is to remove all pegs but one from the board. The perfect game leaves only one peg in the empty position. Less perfect games leave one peg in other positions on the board.Pegs are removed by jumping over each peg with another peg. You can only jump over a peg if there is an empty space on the other side of it and you are right before it. E.g., in the picture to the left, only the pegs with circles around them can move. The peg they jump over is removed from the board. Only horizontal and vertical jumps are allowed, not diagonal ones.The game is over when no more jumps are possible. If more than one peg remains, the game is lost. |

**EXTRA CREDIT:** Develop problem-specific information for the triangular board (6). Does the same heuristic work? Is there a better one?

**Puzzle 2**: The other puzzle you should try your search shell with is Missionaries and Cannibals, described here: <https://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem>. Thebasic version to aim for is the one that uses a boat for 2 people, 3 missionaries and 3 cannibals.

**EXTRA CREDIT:** In developing your representation of state and successor function, allow for a boat able to carry more than 2 people and run your program with that and corresponding numbers of couples:

* With a 2-person boat: 2 couples require 5 trips; 3 couples require 11 trips (basic problem); 4 couples or more has no solution.
* With a 3-person boat, how many trips are required for 2, 3, 4, 5 couples? (More than 5 couples has no solution.)
* With a 4-person boat, why can any number of couples cross?

**Heuristics**

For each of the two puzzles, look for or come up with one or two heuristics that help you reach the solution faster and implement them. Then answer the following questions.

1. Are they admissible/consistent? Why or why not?
2. Does it help or not? If there is a gain, is it in the cost of the search, in the cost of the solution found, both , neither? Is there a tradeoff? Do a little analysis and discuss your results.

# Submission

You are expected to turn in:

1. Working code with a clear separation between what belongs to the search shell and what is puzzle specific and instructions for how to put the search shell and the puzzle specific code together, how to compile and run. Make sure you tell me what language and version of the language/environment you used. You might also try to turn in an executable (though it may not work on my machine. I might make an appointment with you to discuss your program if I have trouble running it or using it.
2. Documentation in Word or Open Office file describing how your search shell knows (is told) which problem, strategy, evaluation function and heuristic to use (where relevant). Also, which initial state and goal test. Note that the two puzzles above have a standard initial state and final state, but your search shell should allow more flexibility than that.
3. A Word or Open Office describing/discussing your heuristic(s), how well they worked or didn’t, any problems you encountered, including not reaching a solution because of space or time complexity, and anything else you would like to bring to my attention.

**REMINDER:** The goal of this assignment is for you to show that you understand the basic concepts behind different types of search, how to apply them to the problems, and the issues that arise.

Turn in the assignment to Jenzabar, under Coursework >> Assignment 2.

Your file should be called: “LastNameInitial-LastNameInitial2\_Asg2.zip”. For example, “LastF1-LastF2\_Asg2.zip” or some other common archive (.rar, .7z).

I prefer .docx files to .pdf files for documents, because it’s either to insert comments in them, but you can turn in both if you like to submit .pdfs. **Don’t submit a hard copy**, just the soft copy!

**Due date:** Specified in the Jenzabar portal.

**Late penalty: 1% per hour late, rounded up to the next hour.**

# Grading

Scores will be assigned as shown in the table. This is group work.

|  |  |
| --- | --- |
| **ASPECT** | **POINTS** |
| **Search Shell/Problem Formulation** | **14** |
| > Problem/strategy independent search algorithm? | 2 |
| > Use of general **goal test**? | 2 |
| > Use of general **successor function**? | 2 |
| > Use of general **path cost function**? | 2 |
| > Clear representation of closed list? | 2 |
| > Clear representation of frontier (open list)? | 2 |
| > Good search node representation? | 2 |
| **Search Progress Reporting and Control** | **7** |
| > Good solution reporting (or graceful failure)? | 2 |
| > Reports number of expanded nodes? | 2 |
| > Reports loop detection and total number of loops detected? | 2 |
| > Can stop search gracefully? | 1 |
| **Interface to Specific Problems/Puzzles** | **6** |
| > Good way of inputting problem-specific state information? | 3 |
| > Good way of providing problem-specific function information? | 3 |
| **Search Strategies** | **16** |
| > Good way of specifying search strategies to use? | 2 |
| > Search strategies act by managing the frontier? | 2 |
| > DF | 3 |
| > BF | 3 |
| > GBeF | 3 |
| > A\* | 3 |
| **Pegs Puzzle** | **25** |
| State representation | 1 |
| Goal test | 1 |
| Successor function | 1 |
| Path cost function | 1 |
| Heuristics (at least one tried) | 3 |
| Strategies tried (DF, BF, GBeF, A\* even if they fail) | 8 |
| Game board (4) | 2 |
| Other game board | 2 |
| Heuristics discussion (a): admissible/consistent? why? | 3 |
| Heuristics discussion (b): gain? where? Why? | 3 |
| **Missionaries and Cannibals** | **22** |
| State representation | 1 |
| Goal test | 1 |
| Successor function (2-person boat) | 2 |
| Path cost function | 1 |
| Heuristics (at least one tried) | 3 |
| Strategies tried (DF, BF, GBeF, A\* even if they fail) | 8 |
| Heuristics discussion (a): admissible/consistent? why? | 3 |
| Heuristics discussion (b): gain? where? Why? | 3 |
| **Extra Credit Search Strategy** | **10** |
| Depth-limited search? | 2 |
| A\* with cutoff? | 2 |
| Iteration thoughts? | 2 |
| IDS implemented? | 2 |
| IDA\* implemented? | 2 |
| **Extra Credit Pegs Puzzle** | **8** |
| State representation | 2 |
| Goal state | 2 |
| Successor function | 2 |
| Path cost function | 2 |
| Heuristic | 2 |
| **Extra Credit M&C Puzzle** | **12** |
| Flexible representation for 2-, 3-, 4-person boat? | 1 |
| Successor function for 3 person boat? | 3 |
| Input provided for 2-person boat, 2 couples? | 0.5 |
| Input provided for 2-person boat, 4 couples? (no solution) | 0.5 |
| Input provided for 3-person boat 2 couples? | 0.5 |
| Input provided for 3-person boat 3 couples? | 0.5 |
| Input provided for 3-person boat 4 couples? | 0.5 |
| Input provided for 3-person boat 5 couples? | 0.5 |
| Input provided for 3-person boat 6 couples? | 0.5 |
| Successor function for 4 person boat? | 3 |
| Input provided for 4-person boat different numbers of couples? (2 is trivial; show 3, 4, 5, 6) | 1.5 |
| **TOTAL BASIC** | **90** |
| **TOTAL EXTRA CREDIT** | **30** |
| **TOTAL BASIC + EXTRA CREDIT** | **120** |