Intelligent Systems

Assignment 2. Problem-Solving Agents

Registration Number
A01746886
A01746887

This activity consists in solving a problem by formulating it as a simple state problem to be solved by a problem-solving agent (PSA), then designing and implementing several heuristic functions to solve it by using selected search methods.

Broken Calculator

Broken Calculator is a puzzle game that requires that you help Eric to calculate some numbers using his smashed calculator. Most of the keys of the calculator have fallen off, but some of it still works and you can use them to find a sequence of key presses to compute a group of numbers.

The game, that you can find and play in the link http://www.mathsisfun.com/games/broken-calculator.html, consist of several levels that show the calculator with different keys and different groups of numbers to compute within a time limit.

Examples of the levels are:

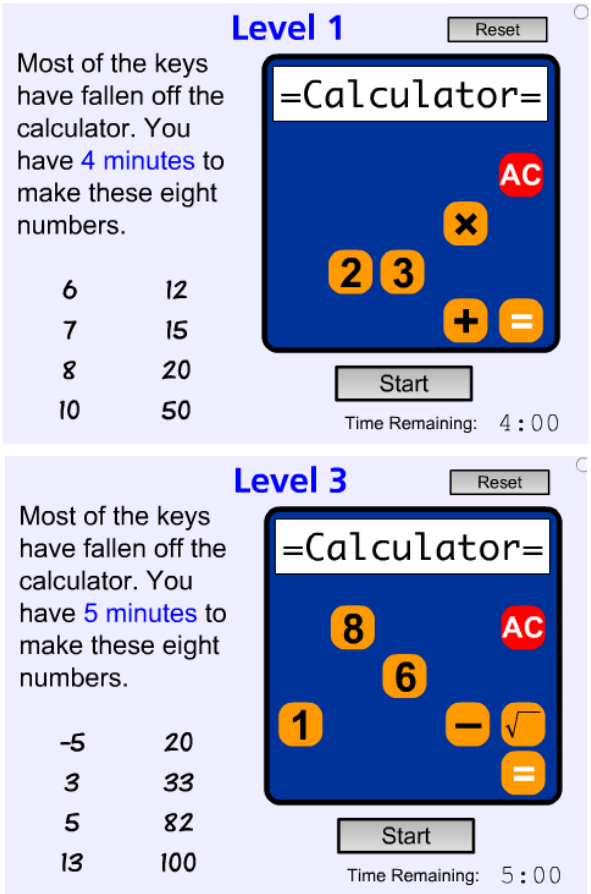


Image taken from Math is Fun (20016) Broken Calculator Game. Recovered from http://www.mathsisfun.com/games/broken -calculator.html

For this activity, you must do the following:

1. Analyze the problem to determine the information that is relevant to its formulation as a PSA. Choose a way to represent states in the programming language (**Python**). Use this form to describe the initial state for at least one level. The time limit of the original game is not relevant for this assignment.
2. Implement the required code in the programming language to be able to solve the selected level using the blind search methods already programmed and included in the Python AIMA code. You must properly display the states' information. Your code must work correctly when being used by the functions that implement the respective search methods.
3. Design two non-trivial heuristic functions, where at least one is admissible. Explain the intuition behind the heuristics and prove that the admissible one really is admissible.
4. Implement both heuristics in the programming language to be able to solve the challenges using the heuristic search methods also included in the Python AIMA code.
5. Execute the blind and heuristic search methods (with the 2 heuristics) to solve the selected level for 3 numbers that can be calculated and 3 numbers that cannot, and show the solutions found by each method.
6. Add observations and conclusions about your experience by programming and using each algorithm to find the solutions: Could they solve all or some problems? How efficient were they? Was it difficult to program the PSA? What cost you the most effort? Also, which algorithm seems to be the best for the selected problems? Which happened to be the best heuristic? And which methods found the best solutions?

PSA Formulation

Configuration State:

A initial value that represent the actual value of state about initial node, in this node can be use operators called actions, also the input provides by initial numbers, Initial numbers with which we can make different combinations to solve the problem and finally provides the goal number represent the number that the program must find.

Initial State:

Is represented by a number by default is zero (like a calculator), this value represents the initial value with which the next states can be calculated.

The initial numbers are those which the calculator is going to use for getting the goal state through the actions.

Actions:

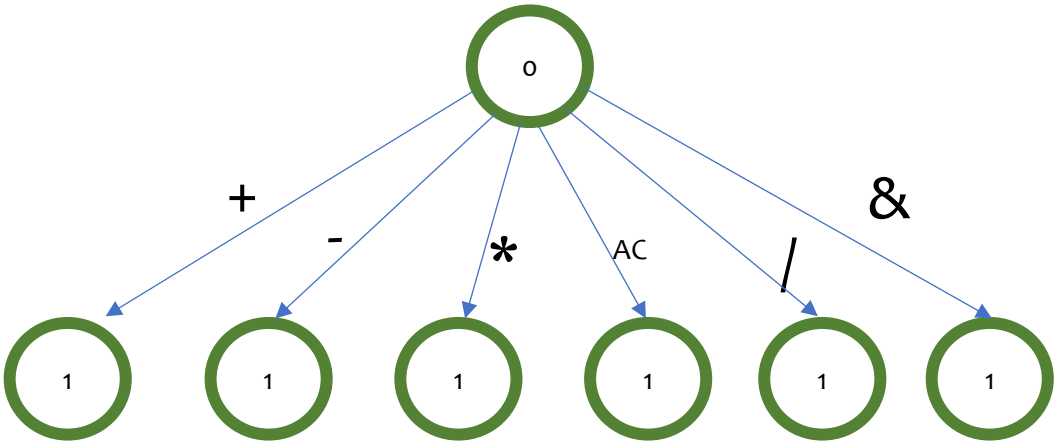
The actions that the problem accept are:

- MINUS (-): Subtract by a state a number to get a new state.
- DIVIDE (/): Divides to a state a number to get a new state.
- PRODUCT (*): Multiply a state with a number to get a new state.
- CONCATENATE (&): Convert the state to a string to concatenate a number to get a new state.
- PLUS (+): Add to state a number to get a new state.
- ALL CLEAR (AC): Reset a state with a value equal to zero.

Transition Model:

Each action can increase or decrease the cost depends the action that execute.

State- Space (Initial State and possible actions)



Represent the first state and the possible actions for state (initial numbers: 1).

Goal state: the goal number.
Path Cost = Sum of the cost from initial state to the actual state, from state j to state j+1 the cost is 1 for all j <= n where the n is the goal node.

Initial states: (Represented in Python)

The initial state is represented by a number (by default is zero). In the code, we can see something like that:

```
class BrokenCalculator(Problem):
        def __init__(self,initial=0, values=[2,3],goal=20,array_actions=['MUL']):
                        Problem.__init__(self,initial,goal)
                        self.using_values = values
                        self.count = len(self.using_values)
                        self.array_actions = array_actions
```

The way to calculate a new state is defined by the next code:

```
def get_new_state(state,new_value,operation):
        new_state = 0
        if operation == MULTIPLICATION:
                new_state = state*new_value
                return new_state
        elif operation == SUM:
                new_state = state+new_value
                return new_state
        elif operation == SUBTRACTION:
                new_state = state-new_value
                return new_state
        elif operation == DIVISION:
                new_state = state/new_value
                return new_state
        elif operation == CONCATENATION:
                new_state = float(str(state)+str(new_value))
                return new_state
        elif operation == RESET:
                new_state = 0
                return new_state
```

2. To get the whole project code, the link to access to the repositories is the following:
https://github.com/nikodtbVf/aima-si

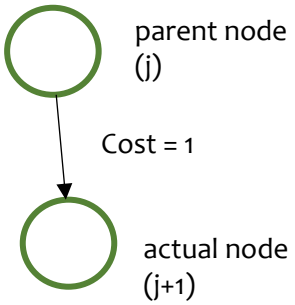
3. Design two non-trivial heuristic functions, where at least one is admissible. Explain the intuition behind the heuristics and prove that the admissible one really is admissible.

Heuristic 1:

The method implemented in the first heuristic consist in get the absolute value from an exponential function which has as variable the actual state value and as constant the goal state, so the heuristic function is:

$$h(n) = \left| e^{\frac{-(actual\ state - goal\ state)^2}{80}} - 1 \right|$$

this method was elected because makes sense that the actual state value is going to be less than 1 (if the limited is applied when actual state tend to +/- infinity). The heuristic is admissible because of h(n) <= 1 in all cases and the real cost defined by h(n*) is 1 for each path from node j to node j+1 (see the following figure). Therefore the total cost to go from initial state to the goal state is always greater than 1, then h(n)<=h(*).



Heuristic 2:

In the heuristic 2, the method implemented is defined as follow:

$$h(n) = \left| \frac{\sqrt{2}}{2} - \frac{actual\ state}{\sqrt{goal\ state^2 + actual\ state^2}} \right|$$

This function was elected because the heuristic evaluation function's behavior tends to decrease faster allowing reach the goal state value. Although is not an admissible heuristic because is greater than the real cost in the most cases, the function is able to reach the value requested.

4. Implement both heuristics in the programming language to be able to solve the challenges using the heuristic search methods also included in the Python AIMA code.

As it was mention before, the whole project code is in the link: https://github.com/nikodtbVf/aima-si

5. Execute the blind and heuristic search methods (with the 2 heuristics) to solve the selected level for 3 numbers that can be calculated and 3 numbers that cannot, and show the solutions found by each method.
   - Three numbers with solutions

Initial numbers: [2,3,10]

Numbers: [123,147,-102]

Operators: [SUM,MUL,DIV,MINUS]

```
---------------A* method-------------Admissible heuristic
Number:  123  time:  1.4719116687774658
Initial: 0
SUM 2: 2
SUM 10: 12
MUL 10: 120
SUM 3: 123
Number:  147  time:  37.75523018836975
Initial: 0
SUM 3: 3
DIV 2: 1.5
MUL 10: 15.0
MUL 10: 150.0
MINUS 3: 147.0
Number:  -102  time:  0.004659175872802734
Initial: 0
MINUS 10: -10
MUL 10: -100
MINUS 2: -102
---------------Breadth First-------------
Number:  123  time:  0.0985562801361084
Initial: 0
SUM 10: 10
SUM 2: 12
MUL 10: 120
SUM 3: 123
Number:  147  time:  0.7190091609954834
Initial..0: 0
SUM 10: 10
MUL 10: 100
```

```
MUL 3: 300
DIV 2: 150.0
MINUS 3: 147.0
Number:  -102  time:  0.0051195621490478516
Initial: 0
MINUS 10: -10
MUL 10: -100
MINUS 2: -102
---------------A* method-------------Heuristic no admissible
Number:  123  time:  1.1228628158569336
Initial: 0
SUM 10: 10
SUM 2: 12
MUL 10: 120
SUM 3: 123
Number:  147  time:  31.263609886169434
Initial: 0
SUM 10: 10
MUL 10: 100
MUL 3: 300
DIV 2: 150.0
MINUS 3: 147.0
Number:  -102  time:  3.7120091915130615
Initial: 0
MINUS 10: -10
MUL 10: -100
MINUS 2: -102
```

- Three numbers without solutions

Initial numbers: [2,6]

Numbers: [101,47,19]

Operators: [SUM,MUL]

No solutions for A* with two heuristics and the `Breadth First`

6. Add observations and conclusions about your experience by programming and using each algorithm to find the solutions:

   Could they solve all or some problems?
   Some problems, in others the algorithms have difficult to find a solution (so much time) or not found any solutions.
   How efficient were they?
   In some cases, for example to A* is more efficient than others, in other cases was too difficult to find a solution because the limit was reached. The solutions depended on the

operators (actions) occupied in the algorithms. The same for BFS

Was it difficult to program the PSA?

At first it was very complicated, because it was not clear how the code work, and the functions were confusing, but after this understanding the PSA logic to create a new program was not difficult.

What cost you the most effort?

Understand the base code to know how the program base works to coding the solution, maybe the most difficult part was find the sequence of the methods to construct a solution.

Also, which algorithm seems to be the best for the selected problems?

Depend on the numbers selected and operators, but the most efficient or at least the algorithm with the fastest solution was de BFS and in relation with the number of actions used both BFS and A* had the same number of actions.

Which happened to be the best heuristic?

the not admissible heuristic function, from the three numbers used 2 of them have less time than the admissible heuristic and both with the same number of actions.

And which methods found the best solutions?

A* Search was the best solution because the path founded is the lowest meanwhile the blind method is the fasted in others words it needs less time to compute.