Capstone Project Abstract for CSCI 2951-O: Explorateur

Vignesh Pandiarajan, Cristina Hu
Advisor: Serdar Kadioglu

We implemented a python library called `Explorateur` [1] that allows one to solve problems using State-Space-Search (SSS). Our library is able to perform both tree search where it searches for a valid solution (validity defined by the user) as well as a graph search (where the search terminates upon reaching a specified state by the user). Both of these types of searches can be conducted through breadth first, depth first and best first search (using an objective function defined by the user). There are other heuristics to the search such as `max_iterations` or `max_runtime` if the user chooses to have a different metric to terminate the search.

The goal of this project is to allow users to easily implement SSS and understand how various problems can be modeled as simply transitioning between different states. Specifically, this is motivated by boolean satisfiability solving, one of the first problems we encountered in class. In the process of finding a valid truth assignment for a boolean formula, a solver will make 'guesses' or 'moves' about whether a specific variable should be set to true or false. In this instance, the concept of different states is simple, but it can extend to other problems such as solving a sudoku puzzle or even figuring out which medical tests to administer to a sick patient.

We provide the user with a basic framework, a `BaseState` and `BaseMove` for which there are functions they must implement in order for the search to work. It is up to them to decide what is a move in the context of their problem as well as what they want a state to be.

A user can even visualize their search, if they choose to provide a filename into which the program will write a graph in the DOT language. This can then be passed into an online graphviz visualizer. Example below:
The red states are the ones that have been explored and failed and the green state is the solution state.

[1] `https://github.com/skadio/explorateur`

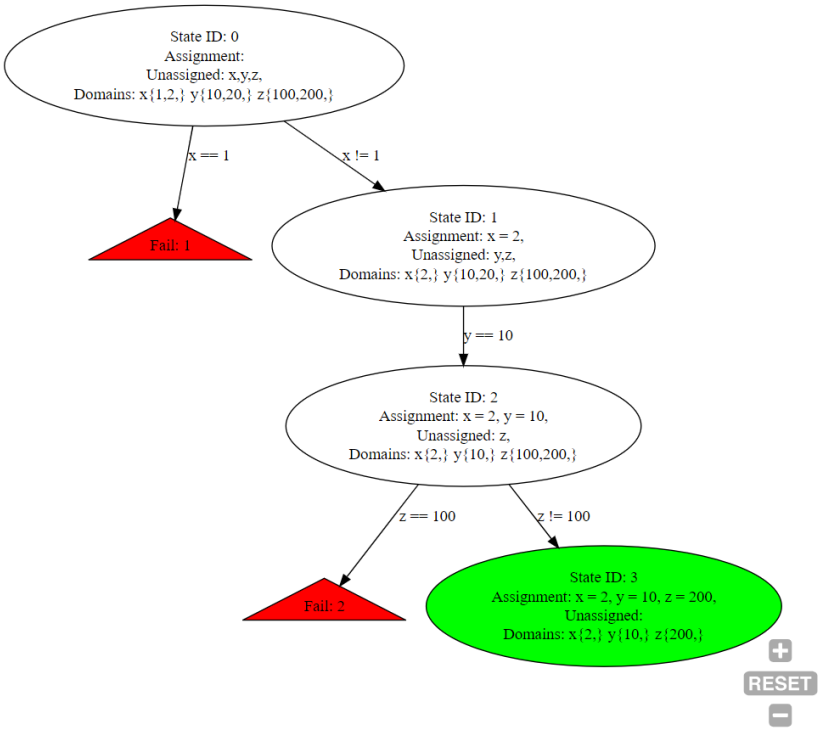

Figure 1: DFS on a simple problem

[2]If neither are true, we add child states to storage type

Algorithm 1 Search Function

```
1: procedure SEARCH(initial_state, goal_state, exploration_type, search_type,
   is_solution_path, max_depth, max_moves, max_runtime, dot_filename)
2:     Initialize the search parameters and validate them.
3:     Reset the search state (including open and closed lists).
4:     Create storage for open and closed states.
5:     Copy the initial state to a root node and mark it.
6:     Check if the initial state is a solution or if the search should terminate.
7:     if it is a solution then
8:         if returning the solution path is requested then
9:             Return the solution path.
10:        else
11:            return True
12:        end if
13:    else if the search should terminate then
14:        return False
15:    end if
16:    while there are still states to explore do
17:        Get the next state to explore from the open list.
18:        Mark the state as visited in the closed list if using graph search.
19:        Create a successor state by applying a move to the current state.
20:        if the move is successful then
21:            Check if the successor state is a solution or if the search should
   terminate. [2]
22:            if it is a solution then
23:                if returning the solution path is requested then
24:                    Return the solution path.
25:                else
26:                    return True
27:                end if
28:            else if the search should terminate then
29:                return False
30:            end if
31:        else
32:            Count the move as failed and continue.
33:        end if
34:        if any stopping condition is met then
35:            return False
36:        end if
37:    end while
38:    Return False if no solution is found and the search ends.
39: end procedure
```