CSIS, BITS Pilani K. K. Birla Goa Campus

Artificial Intelligence (CS F407)

Programming Assignment 2

Aayush Mathur
2018A7PS0729G
Instructor Incharge: Sujith Thomas

Algorithm Used:

---

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*,*breeze*,*glitter*]
  static: *KB*, a knowledge base, initially containing the "physics" of the wumpus world
    x, *y*, *orientation*, the agent's position (initially [1,1]) and orientation (initially *right*)
    visited, an array indicating which squares have been visited, initially *false*
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

  update *x*,*y*,*orientation*, *visited* based on *action*
  if *stench* **then** TELL(*KB*, $S_{x,y}$) **else** TELL(*KB*, $\neg S_{x,y}$)
  if *breeze* **then** TELL(*KB*, $B_{x,y}$) **else** TELL(*KB*, $\neg B_{x,y}$)
  if *glitter* **then** *action* ← *grab*
  else if *plan* is nonempty then *action* ← POP(*plan*)
  else if for some fringe square [*i*,*j*], ASK(*KB*, $(\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
      for some fringe square [*i*,*j*], ASK(*KB*, $(P_{i,j} \vee W_{i,j})$) is *false* **then do**
    plan ← A*-GRAPH-SEARCH(ROUTE-PROBLEM([*x*,*y*], *orientation*, [*i*,*j*],*visited*))
    action ← POP(*plan*)
  else *action* ← a randomly chosen move
  return *action*

- An agent Object of class Agent is initialized. This object has a KnowledgeBase kb of class WumpusKB, the wumpusWorld map and other necessary variables.
- An object kb of class WumpusKB can be used to store the perceived information in the form of clauses. The Agent can tell the kb about its perception of the wumpusWorld. The Agent can also check if the kb entails a sentence using the WumpusKB.ask_if_true function.
- WumpusKB uses the DPLL algorithm to generate inferences.
- Sentences or Percepts are converted to objects of class Expr. The Expr class overrides various operators which helps us to generate Sentences which can further be broken down to clauses. It makes it easier to Resolve the Clauses.

- WumpusKB's __init__ method saves the initial state and the rules of the wumpusWorld as clauses inside the WumpusKB object.
- The Agent.PL_Wumpus_Agent method controls the agent object. It maintains and updates the visited list to store the cell locations which have been visited (safe cells). It calls the Agent.PerceiveCurrentLocation method and sends the perceived data to kb using WumpusKB.tell method.
- After informing the kb about the percept the agent will check if it has a plan. A plan is a sequence of actions. If it has a plan, it will pop the first action from the plan and execute it using Agent.TakeAction method. Else, it will loop over the non visited cells and see if it can move to a safe cell.
- The agent finds the safe cell by using the WumpusKB.ask_if_true method. Once it finds a safe cell it will call the findPath method to find a safe path to that cell. The path found is then pushed into the plan. And then the agent executes the first action in the plan.
- In the end Agent.PL_Wumpus_Agent calls itself and this process repeats until the agent has exited the wumpusWorld or if it dies.
- The DPLL Algorithm uses various heuristics like: Early Termination, Unit Clauses, Pure Symbols.
- The pl_true method returns True if the propositional logic expression is true in the model and False if it is false. It helps in early termination.
- The find_pure_symbol and find_unit_clause methods as their names suggest, help in finding pure symbols and unit clauses.
- The rest of the functions are helper functions which help the above mentioned functions.

Some Statistics-

1) Using all 3 heuristics (Early termination, Unit Clauses, Pure Symbols):
   DPLL Calls: 1857
   Time: 5.59secs
2) Using 2 heuristics (Early termination, Unit Clauses):
   DPLL Calls: 2919
   Time: 7.51secs
3) Using 2 heuristics (Early termination, Pure Symbols):
   DPLL Calls: -
   Time: Didn't terminate even after a minute

Unit Clauses heuristic performed much better than the Pure Symbols heuristic. The reason could be the fact that there are a lot of unit clauses in the knowledge base and pure symbols are rare. This rarity of pure symbols could be because any symbol for example (B22: Breeze at cell 2,2) is only directly related to its neighboring cells (which might have a pit) but we have a lot of clauses so B22 appearing in all of them has a very low probability. However, Unit Clauses will exist right from the beginning (for example, ~W11 and ~P11) and as we keep resolving the clauses we might generate new unit clauses.