Eric Komperud – u0844210

A3 – CS4300 Lab Report – 9/28/17

Logical Agent using Resolution Theorem Proving

1. Introduction

A logical agent is one that makes use of *Knowledge Base* in order to not only draw information about potential actions, but also to store new information in about its environment. Our agent's knowledge base is one that stores all the rules of Wumpus world as well as the learned facts of Wumpus world using propositional logic. In order to make full use of this knowledge though, it also has to be able to infer new tidbits of knowledge from previously gathered information and rules. For instance, detecting a breeze in (2,1) lets the agent know there could be a breeze in (1,1), (2,2), and/or (3,1). This, combined with the knowledge that there is definitely **not** a pit in (1,1) allows the agent to infer that there must be a pit in either (2,2) and/or (3,1). Furthermore, if the agent doesn't detect a breeze in (1,2), they should be able to infer that not only is there not a pit in (2,2), but also combine that knowledge with previously gained knowledge to infer that there **must** be a pit in (3,1). To give the agent this power, we will be utilizing a Resolution Theorem Prover (or RTP for short). The RTP uses proof by contradiction to show that, given an existing knowledge base, one state of a Wumpus world square **cannot** exist, and therefore the 'not' of that state **must** exist instead. The problem with our RTP is that it will (without many optimizations) look through our entire knowledge base to find some pretty localized piece of knowledge. Furthermore, when attempting to combine knowledge, it will create even more sentences which must then be combined with previously known knowledge. The cherry on the cake is that many of these newly generated sentences which took exponential effort to create will lead nowhere useful because the algorithm isn't looking in the right place in the first place. With these limitations in mind, I will test our agent on a Knowledge Base size variable. That is, I will increase the number of sentences in the knowledge base, expecting to see an exponential change in time elapsed per resolution. If the complexity of our RTP is indeed exponential like I anticipate it to be, the question that should be asked is: is it worth pursuing RTP further as a legitimate way to give our agent the power of inference, and if it is, then what optimizations might be made to improve its performance?

2. Method

My RTP will run as explained through pseudocode below:

```
CS4300_RTP(sentences,thm,vars) returns true or false
     Inputs:    sentences: A list of all sentences in the KB
                thm: Theorem to be proved true or false
                vars: A list of all states in Wumpus World
     neg_thm = -thm
     sentences += neg_thm
     new_clauses = {}
     loop ∞
          Length = sentences.length
          for each sentence pair that hasn't been resolved before:
               resolvents = Resolve(sentence pair)
               if resolvents contains the empty pair, return true
               add unique resolvents to new_clauses
```

```
        add new_clauses to sentences without adding duplicates
        if sentences.length == Length, return false
```

Fig 1.

The one optimization I made to my code for this assignment was not allowing repeat sentence resolutions to be made. In order to test the effects of the various variables, I will be creating random knowledge bases and theorems of various lengths to test. For each variable size (trial), I will be doing 500 runs to obtain a mean, variance, and 95% confidence interval. The variable will be tested as follows:

- Number of statements in knowledge base (S)
  - Sentences in KB: [5,6,7,8,9,10,11,12,13,14,15]
  - Clauses per sentence: 2
  - Theorem Length: 1

3. Verification of Program

Given a Knowledge base *KB* of:

1v2	-2v3	3v4	1	-2	-3

Fig 2.

What theorem *thm* would allow "*KB* **entails** *thm*" to be true? I propose that *thm* = 4 will suffice and *thm* = -4 will not. Substituting these values into truth tables yields the following results:

1	2		1 v 2
1	0		1

Fig 3.

2	3		-2 v 3
0	0		1

Fig 4.

3	4		3 v 4
0	0		0
0	1		1

Thus, we know that 4 must be true for this to be a valid model. When I run my program, it produces an empty array (true) for *thm = 4* and a CNF of 8 sentences (false) for *thm = -4*.

Fig 5.

4. Data

# of sentences	5	6	7	8	9	10	11	12	13	14	15
Time (s)	0.0011	0.0012	0.0013	0.0015	0.0023	0.0029	0.0034	0.0039	0.0046	0.0052	0.0068

Fig 6.

5. Analysis

As expected, the amount of time passed increased exponentially as the number of sentences in the knowledge base increased. What I did not record here is that I also tested for the number of clauses per sentence in the knowledge base and the time increase was even more dramatic. So much in fact that I waited several minutes at 4 clauses per sentence with no resolution. I did not record this data as I did not think time of that length would be consistent or reliable data.

6. Interpretation

So while even though a large amount of sentences does not make the waiting time dramatically long for small knowledge bases, it has the potential to do so as the knowledge base gets big. Furthermore, I saw that increasing the number of clauses per sentence had the potential to completely destroy the viability of my RTP in terms of complexity. It's hard to say with certainty whether or not optimizations could make the program run faster, but I would postulate that is possible. Being able to prioritize sentences with literals that are also in the theorem likely would have made the RTP quicker. I also think this algorithm's glaring weakness is a sign that looking into other more efficient algorithms is probably a more worthwhile effort.

7. Critique

In this assignment, I again learned that random searching is not particularly effective in large-scale models. Always paying attention to complexity is a very difficult task, especially when there' loops within loops that reference other functions that also have loops. If one wants to keep track of their complexity, it's best to note how many loops are being used and where from the very beginning and take care to eliminate as many as possible.

8. Log

I spent approximately 6 hours building the KB generator. I spent approximately 6 hours researching, building, and tweaking the RTP. I spent approximately 3 hours testing, collecting data, and writing this lab report.