A new belief announcement solver

Aaron Hunter, John Agapeyev

August 31, 2018

Abstract

We present a new tool for solving the problem of belief revision across multiple agents using a shared announcement formula. We model the problem as a deepest-node search inside a multitree, where the depth represents the number of agents satisfied by a given formula. We hope modeling the problem in this way will allow future improvements to be made.

1 Introduction

Belief revision refers to the process where an agent incorporates new information to an existing set of beliefs. For the sake of efficiency, it may be desirable to perform belief revision in bulk across multiple agents to accomplish multiple goals. The key problem behind this idea of shared bulk revision is finding a suitable announcement. Belief Revision is already a known hard problem, so the goal is to minimize overhead in solving the problem across multiple agents. We present a new tool to solve this problem in an innovative and efficient way that lays the groundwork for future improvements that have been applied to similar problem modelling schemes.

2 Background

I don't understand the problem enough to write background on it.

3 Design

We model the problem of finding a suitable announcement for N agents as exploring a multitree, where each node is a possible announcement formula, and its depth represents the number of agents whose goals are satisfied by the formula. In this way, the problem can be modeled as a deepest-node search inside the multitree, where we want to find the node farthest from the roots, as it would represent either a solution for all N agents, or it would fail to reach the maxmium depth, indicating the problem has no solution. The idea behind modeling the problem in this fashion is to potentially benefit from algorithmic improvements in future work such as heuristics and backtracking that can be found in algorithms such as Conflict Driven Clause Learning and Alpha-Beta pruning.

Algorithm 1 Belief Announcement

```
function FindAnnouncement($\beta, \gamma$)
  1. if SAT($\{\gamma_1 \cup \gamma_2 \cup \ldots \gamma_N\}$) then return $\{\gamma_1 \cup \gamma_2 \cup \ldots \gamma_N\}$
  2. while $\phi = GetNextFormula()$ do
     (a) For $i = 1$ up to $N$
         i. $R = BeliefRevise(\beta_i, \phi)$
         ii. if $R \subseteq \gamma_i$ then goto 2
     (b) end for
     (c) return $\phi$
  3. end while
  4. return NO_SOLUTION
```

To find trivial solutions, we first check if all the agents goals are satisfiable using a SAT solver. If all goals are satisfiable together, then the solution is simply the concatenation of all the goals into a single formula. If the goals are unsatisfiable, then we enumerate the list of possible revision formulas. Each formula, we revise the first agent's beliefs, and see if its goal is satisfied by the result. If it is, repeat for the next agent. If the goal is not satisfied, then the formula is not a valid solution, so exit that iteration and get a new formula to test. If all agents are satisfied, that formula is the announcement solution. If no possible formula satisfies all agents, then the problem has no solution. *GetNextFormula()* returns the next formula in an $O(2^{3M-1})$ brute force where M is the total number of variables in the set of agent beliefs and goals. Belief Revision is performed through an external tool known as GenC.

4 Implementation

4.1 Input/Output

Due to the complexity of the input data, relying on file I/O is difficult from both an implementation and usage standpoint. To remedy this, all I/O is performed through a GUI, implemented using Qt. All inputs are human readable formula strings such as "(1 and 2 or (3 and not 4))". Variables are represented as numbers , and the logical operations are limited to AND, OR, and NOT. This allows efficient representation and input of fairly complex formulae from a user standpoint. Output is displayed through pop-up message boxes displaying the relevant information, be it an error, or the solution to the problem. For a full list of constraints required of the input formulas, see GenC. The tool also supports some benchmarking options that bypass the GUI. It supports both a worst-case n-variable solve, which trivially has no solution, and a full enumeration of the 2-variable input space for 2 agents. These options allow one to test the scaling of the tool without having to manually enter large complex equations into a user interface.

4.2 Data Representation

Internally, all formulas are represented as 2D arrays of integers, where the integer is the variable number, and its sign represents logical negation. All manipulation and I/O is based around this construct, and is converted to and from other forms transparently by the application. For enumerating the list of possible revision formulas, this is performed as a brute force, split into multiple sections. The first section enumerates the list of all possible combinations of varying lengths. For each combination, the data is negated through an N-bit brute force to obtain all possible logical negation states. For each possible set of negated inputs, conjunctions between variables are generated through an N-1 bit brute force. The total operation has complexity $O(2^N * 2^N * 2^{N-1}) = O(2^{3N-1})$. This is how the *GetNextFormula()* function is implemented in section 3.

5 Conclusion

We present a tool for efficiently solving the problem of determining shared belief announcement among multiple agents. We model the problem as a deepest-node search inside a multitree, where the depth of a node is the number of agents a given formula satisfies. Our hope is to lay the groundwork for future algorithmic improvements based on this initial problem model.