An Implementation of Consistency-Based Multi-Agent Belief Change using ASP

Paul Vicol[1], James Delgrande[1], and Torsten Schaub[2]

[1] Simon Fraser University
Burnaby B.C, Canada V5A 1S6
pvicol@sfu.ca, jim@cs.sfu.ca
[2] Universität Potsdam
August-Bebel-Strasse 89
14482 Potsdam, Germany
torsten@cs.uni-potsdam.de

Abstract. This paper presents an implementation of a general framework for consistency-based belief change using Answer Set Programming (ASP). We describe Equibel, a software system for working with belief change operations on arbitrary graph topologies. The system has an ASP component that performs a core maximization procedure, and a Python component that performs additional processing on the output of the ASP solver. The Python component also provides an interactive interface that allows users to create a graph, set formulas at nodes, perform belief change operations, and query the resulting graph.

Keywords: belief change, belief merging, answer set programming, python

1 Introduction

We present an implementation of the consistency-based framework for multi-agent belief change discussed in [2]. In a network of connected agents, each with a set of beliefs, it is important to determine how the beliefs of the agents change as a result of incorporating information from other agents. We represent such a network by an undirected graph $G = \langle V, E \rangle$, where vertices represent agents and edges represent communication links by which agents share information. Associated with each agent is a *belief base* expressed as a propositional formula. Beliefs are shared among agents via a maximization procedure, wherein each agent incorporates as much information as consistently possible from other agents.

Before delving into the implementation, we introduce a motivating example:

Example 1. Consider a group of drones searching for missing people in a building. Each of the drones has some initial beliefs regarding where the missing people might be. Drone 1 believes that there is a person in the bookstore, as well as one in the atrium; drone 2 believes that there cannot be missing people in both the atrium *and* the bookstore; drone 3 just believes that there is a person in the cafeteria. The drones communicate with one another, and each is willing

to incorporate new information that does not conflict with its initial beliefs. Our goal is to determine what each drone will believe following the communication.

We have developed a software system called Equibel that can be used to simulate the above scenario, and determine where each drone would look for the missing people. More generally, Equibel allows for experimentation with belief sharing in arbitrary networks of agents. It uses Answer Set Programming (ASP) to perform the maximization step, and Python to manage the solving process and provide programmatic and interactive interfaces. The software is available online at `www.github.com/asteroidhouse/equibel`.

2 Related Work

Many methods have been proposed to deal with belief change involving multiple sources of information. Classical approaches to belief merging, such as [6] and [7], start with a set of belief bases and produce a single, merged belief base. Our approach differs in that we update multiple belief bases simultaneously.

The BReLS system [8] implements a framework for integrating information from multiple sources. In BReLS, pieces of information may have different degrees of reliability and may be believed at different discrete time points. Revision, update, and merging operations are each restrictions of the full semantics. The REV!GIS system [10] deals with belief revision in the context of geographic information systems, using information in a certain region to revise adjacent regions. There have been many approaches to iterative multi-agent belief sharing, including the iterated merging conciliation operators introduced in [4], and Belief Revision Games (BRGs) introduced in [9]. BRGs are ways to study the evolution of beliefs in a network of agents over time. While sharing a graph-based model, our framework differs from BRGs in two ways. First, we use a *consistency-based* approach, which is distinct from any of the revision policies in [9]. Second, we describe a "one-shot" method for belief sharing, rather than an iterated method.

The consistency-based framework we employ here has been developed in a series of papers, including [1], [2], and [3].

3 The Consistency-Based Belief Change Framework

We work with a propositional language $\mathcal{L}_{\mathcal{P}}$, defined over an alphabet $\mathcal{P} = \{p, q, r, \ldots\}$ of propositional atoms, using the connectives $\neg, \wedge, \vee, \rightarrow$ and $\equiv$ to construct formulas in the standard way. For $i \geq 0$, we define $\mathcal{P}^i = \left\{p^i \mid p \in \mathcal{P}\right\}$ containing superscripted versions of the atoms in $\mathcal{P}$, and define $\mathcal{L}^i$ to be the corresponding language. We denote the original, non-superscripted language by $\mathcal{L}^0$. We denote formulas by Greek letters α, β, etc. Given a formula $\alpha^i \in \mathcal{L}^i$, $\alpha^j \in \mathcal{L}^j$ is the formula obtained by replacing all occurrences of $p^i \in \mathcal{P}^i$ by $p^j \in \mathcal{P}^j$. For example, if $\alpha^1 = (p^1 \wedge \neg q^1) \rightarrow r^1$, then $\alpha^2 = (p^2 \wedge \neg q^2) \rightarrow r^2$.

Our implementation uses the maximization approach to belief sharing described in [2]; here we recall the terminology and notation for maximal equivalence sets, and we refer the reader to [2] for more details.

Definition 1 (G-scenario). *Let $G = \langle V, E\rangle$ be a graph with $|V| = \{1, 2, \ldots, n\}$. A G-scenario Σ_G is a vector of formulas $\langle \varphi_1, \ldots, \varphi_n\rangle$. The notation $\Sigma_G[i]$ denotes the i^{th} component, φ_i.*

An agent starts with some initial beliefs that she does not want to give up, and then "includes" as much information as consistently possible from other agents. This is done as follows. Each agent expresses her beliefs in a distinct language, such that the languages used by any two agents are isomorphic. Specifically, agent i expresses her beliefs as formulas of the language $\mathcal{L}^i$. Because the agents' languages are disjoint, $\bigcup_{1\leq i\leq n} \varphi_i^i$ is trivially consistent. For each agent, we want to find out what "pieces" of beliefs from other agents she can incorporate. To do this, we assert that the languages of adjacent agents agree on the truth values of corresponding atoms as much as consistently possible. The equivalences between the atoms of agents i and j tell us what those agents can and cannot agree on, and provide a means to "translate" formulas between those agents' languages. If agents i and j cannot agree on the truth value of p, then we can translate formulas from $\mathcal{L}^i$ to $\mathcal{L}^j$ by replacing p^i by $\neg p^j$. This process is formalized below.

Definition 2 (Equivalence sets, Fits, Maximal fits). *Let $G = \langle V, E\rangle$ be a graph and $\mathcal{P}$ be an alphabet.*

- *An equivalence set EQ is a subset of $\{p^i \equiv p^j \mid \langle\{i, j\}, p\rangle \in E \times \mathcal{P}\}$.*
- *Given a G-scenario $\Sigma_G = \langle \varphi_1, \ldots, \varphi_n\rangle$, a fit for Σ_G is an equivalence set EQ such that $EQ \cup \bigcup_{i=1}^{n} \varphi_i^i$ is consistent.*
- *A maximal fit for Σ_G is a fit EQ such that for all fits $EQ' \supset EQ$, we have that $EQ' \cup \bigcup_{i=1}^{n} \varphi_i^i$ is inconsistent.*

Let $\Sigma_G = \langle \varphi_1, \ldots, \varphi_n\rangle$ be a G-scenario and $\mathbf{F}$ be the set of maximal fits for Σ_G. Informally, the *completion* of Σ_G, denoted $\Theta(\Sigma_G)$, is a G-scenario $\Sigma'_G = \langle \varphi'_1, \ldots, \varphi'_n\rangle$ consisting of updated formulas for each agent following a belief sharing procedure. [2] gives both semantic and syntactic characterizations of the completion. Here we state the syntactic characterization, based on translation.

Definition 3 (Substitution function). *Let $G = \langle V, E\rangle$ be a graph, and EQ be an equivalence set. Let R^* denote the transitive closure of a binary relation R. Then, for $i, j \in V$, we define a substitution function $s_{i,j}^{EQ} : \mathcal{P}^i \to \{l(p^j) \mid p^j \in \mathcal{P}^j\}$, where $l(p^j)$ is either p^j or $\neg p^j$, as follows:*

$$s_{i,j}^{EQ}(p^i) = \begin{cases} p^j : (p^i \equiv p^j) \in EQ \\ \neg p^j : \{i, j\} \in E^*, (p^i \equiv p^j) \notin EQ \end{cases}$$

Given a formula α^i, $s_{i,j}^{EQ}(\alpha^i)$ is the formula that results from replacing each atom p^i in α^i by its unique counterpart $s_{i,j}^{EQ}(p^i)$. Thus, $s_{i,j}^{EQ}(\alpha^i)$ is a *translation* of α^i into the language of agent j, that is consistent with agent j's initial beliefs.

Proposition 1. *Let $G = \langle V, E\rangle$ be a graph and $\Sigma_G = \langle \varphi_1, \ldots, \varphi_n\rangle$ be a G-scenario. Let $\Theta(\Sigma_G) = \langle \varphi'_1, \ldots, \varphi'_n\rangle$ be the completion of Σ_G , and let $\boldsymbol{F}$ be the set of maximal fits of Σ_G. Then, we find φ'_j, for $j \in \{1, \ldots, n\}$, as follows:*

$$\varphi'_j \equiv \bigvee_{EQ\in F} \left(\bigwedge_{\{i,j\}\in E^*} (s_{i,j}^{EQ}(\varphi_i^i))^0 \right)$$

4 System Design

Equibel is split into two architectural layers: an ASP layer, which performs the core maximization procedure, and a Python layer, which performs post-processing of answer sets and provides programmatic and interactive user interfaces to experiment with belief sharing on custom graphs. Equibel provides a Python package (`equibel`) that allows users to perform belief change operations in programs, and a user-friendly command-line interface (CLI) that allows for real-time experimentation. The CLI allows users to enter commands to create agents, edges, and formulas, execute belief change operations and query the resulting graph. A query might ask what a particular agent believes, or what the common knowledge is (the disjunction of all agents' beliefs).

There are three major stages to computing the completion of a G-scenario: 1) finding maximal sets of equivalences between atoms of adjacent agents; 2) translating beliefs between the languages of adjacent agents; and 3) combining beliefs that result from different maximal equivalence sets. The first two steps are done in ASP, while the third is done in Python.

The ASP layer consists of a set of logic programs that can be combined in different ways to achieve different functionality. The core of Equibel is the `eq_sets.lp` logic program that finds maximal sets of equivalences of the form $p^i \equiv p^j$ between atoms at neighbouring agents i and j. We use the logic program `translate.lp` to translate formulas between the languages of connected agents based on the equivalence sets. Each optimal answer set gives the new information incorporated by each agent, based on a specific maximal EQ set. We use the ASP grounder/solver `clingo`, from the Potsdam Answer Set Solving Collection [5].

The Python component combines formulas that occur in different answer sets. The Python `clingo` interface also manages the solving state, by loading specific combinations of logic modules. This allows the system to find either cardinality- or containment-maximal EQ sets, and potentially perform iterated belief sharing. We also designed a file format for specifying belief change problems, called the Belief Change Format (BCF). This is an extension of the DIMACS graph format, and is a standard for communication within our system.

5 ASP Implementation

Encoding Graphs Encoding a graph involves creating agents, assigning formulas to the agents, and setting up connections between the agents. We declare agents using the `node/1` predicate, and declare edges using `edge/2`. We assign formulas to agents using `formula/2`, where the first argument is a formula built using the function symbols `and/2`, `or/2`, `implies/2`, `iff/2`, and `neg/1`, and the second argument is an integer identifying an agent. For example, we can assign the formula $(p \wedge q) \vee \neg r$ to agent 1 with `formula(or(and(p,q),neg(r)),1)`.

Finding Maximal EQ Sets Maximal equivalence sets are found by `eq_sets.lp`, which: 1) *generates* candidate equivalence sets; 2) *tests* the equivalence sets by

attempting to find a truth assignment, constrained by the equivalences, that satisfies all agents' initial beliefs; and 3) *optimizes* the results to find containment- or cardinality-maximal sets.

In order to check whether a truth assignment is satisfying, we first break each formula down into its subformulas. After truth values have been assigned to the atoms, an agent's beliefs are built back up from its subformulas; this allows us to determine whether an assignment models the original beliefs of each agent. We classify each subformula as either a compound or an atomic proposition:

```
atom(P,X) :- subform(P,X), not compound_prop(P,X).
atom(P)   :- atom(P,_).
```

Candidate EQ sets are generated by:

```
{ eq(P,X,Y) : atom(P), edge(X,Y), X < Y }.
```

The predicate `eq(P,X,Y)` expresses that $P^X \equiv P^Y$. The condition $X < Y$ halves the search space over edges; this is justified because edges are undirected. After we generate a candidate EQ set, we check whether it is possible to assign truth values to all atoms, restricted by the equivalences, such that the agents' original formulas are satisfied. We assign a truth value to each atom, with the constraint that atoms linked by an equivalence must have the same truth value:

```
1 { truth_value(P,X,true), truth_value(P,X,false) } 1 :-
       atom(P), node(X).
:- eq(P,X,Y), truth_value(P,X,V), truth_value(P,Y,W), V != W.
```

Now we build up the original formulas from their subformulas, to see if the assignment is satisfying. A sample of the code used to build up the original formulas starting from the atoms is shown below:

```
sat(F,X) :- F = and(A,B), sat(A,X), sat(B,X),
            subform(F,X), subform(A,X), subform(B,X).
```

For an EQ set to be acceptable, it must be possible to find a truth assignment that satisfies all the original formulas. Thus, we introduce the constraint:

```
:- formula(F,X), not sat(F,X).
```

There are two types of maximality, each requiring a different program statement and solving configuration. The standard `#maximize` statement in `clingo` finds EQ sets that are maximal with respect to cardinality. To find *containment-maximal* EQ sets, however, we use a domain-specific heuristic:

```
_heuristic(eq(P,X,Y), true, 1) :- atom(P), edge(X,Y), X < Y.
```

The `true` heuristic modifier tells the solver to decide first on `eq` atoms, and to set them to `true`. The solver initially makes all `eq` atoms true, and then "whittles down" the set, producing containment-maximal sets.

For our one-shot approach to belief sharing, we take the transitive closure of the `eq/3` predicates. This allows for an agent to learn from other agents throughout the graph, not just from its immediate neighbours. The module `translate.lp` translates formulas between the languages of connected agents,

and outputs `new_formula/2` predicates that indicate the new information obtained by an agent from its neighbours.

Consider an equivalence set EQ and an agent i. The new belief of i, based on EQ, is the conjunction of translated beliefs from all agents connected to i. But we may have multiple maximal equivalence sets, each of which represents an equally plausible way to share information. Thus, we combine beliefs that result from different equivalence sets by taking their disjunction. Both of these steps are performed in Python, using the output of `translate.lp`.

We now look at how the system works on our opening example. Let the atomic propositions `a`, `b`, and `c` denote the facts that there are missing people in the atrium, bookstore, and cafeteria, respectively. The network of drones is represented by a complete graph on three nodes, numbered 1 to 3, and the associated G-scenario is $\Sigma_G = \langle a \wedge b, \neg a \vee \neg b, c \rangle$. Solving with `eq_sets.lp`, we find four maximal EQ sets. Based on the first set, $\{a_1 \equiv a_2, a_1 \equiv a_3, a_2 \equiv a_3, b_1 \equiv b_3, c_1 \equiv c_2, c_1 \equiv c_3, c_2 \equiv c_3\}$, the new beliefs of the drones would be $\langle a \wedge b \wedge c, a \wedge \neg b \wedge c, a \wedge b \wedge c \rangle$. Taking the disjunction of formulas obtained from different EQ sets, the final beliefs of the drones are $\langle a \wedge b \wedge c, (a \equiv \neg b) \wedge c, (a \vee b) \wedge c \rangle$.

6 Expressing Revision and Merging

The `equibel` Python module allows users to perform belief change operations such as revision and merging, without explicitly creating graph topologies. For these operations, the user only needs to specify formulas representing belief bases to be operated on, and the system constructs an *implicit* graph topology, finds the completion, and returns either a formula of a specific agent in the completion, or the invariant knowledge. In this section, we show how belief revision and two types of merging are expressed in our framework.

For belief revision, we need to consistently introduce a new belief α into a belief base K, while retaining as much of K as possible. Belief revision can be modeled as a two-agent graph $G = \langle V, E \rangle$, with $V = \{1, 2\}$, $E = \{\{1, 2\}\}$, and $\Sigma_G = \langle K, \alpha \rangle$. Through belief sharing, agent 2 will incorporate as much information from agent 1 as possible, while maintaining consistency with α. The belief of agent 2 in the completion, $\Theta(\Sigma_G)[2]$, is the result of the revision, $K * \alpha$. This corresponds to consistency-based revision, as defined in [1].

Turning to belief merging, multiple, potentially mutually inconsistent, bodies of knowledge need to be combined into a coherent whole. Two approaches to merging are described in [3]. The first approach is a generalization of belief revision called *projection*. Given a multiset $\mathcal{K} = \langle K_1, \ldots, K_n \rangle$ and a constraint μ, the contents of each belief base K_i are projected onto a distinguished belief base which initially contains just μ. This is expressed in our framework using a star graph, such that the central agent initially believes μ, and incorporates as much information as possible from each of its neighbours. Formally, $G = \langle V, E \rangle$, where $V = \{0, 1, \ldots, n\}$, $E = \{\{0, i\} \mid i \in V \setminus \{0\}\}$, and $\Sigma = \langle \mu, K_1, \ldots, K_n \rangle$. The merged belief base $\Delta_\mu(\mathcal{K})$ is the belief of the central agent in the completion, $\Theta(\Sigma)[0]$. The second approach to merging, called *consensus merging*, involves

"pooling" together information from the belief bases. Let G be a complete graph and let Σ be a G-scenario. Information is pooled by taking the invariant of the completion $\Theta(\Sigma) = \langle \varphi'_1, \ldots, \varphi'_n \rangle$, so that $\Delta(\mathcal{K}) = \bigvee_{i=1}^{n} \varphi'_i$.

7 Conclusion

In this paper, we introduce Equibel, a software system for experimenting with consistency-based multi-agent belief sharing. We model networks of communicating agents using arbitrary undirected graphs, where each node is associated with a belief base represented by a propositional formula. Each agent shares information with its neighbours, and learns as much as possible from connected agents, while not giving up her initial beliefs. Belief sharing is carried out via a global procedure that maximizes similarities between belief bases of adjacent agents. We describe Equibel's architecture, examine how maximal equivalence sets are found using ASP, and look at how Equibel handles belief revision and merging by constructing implicit graph topologies. Other operations, such as belief extrapolation, can also be expressed within this framework. We are working on expanding the system to support iterated change and agent expertise.

Acknowledgements

Financial support was gratefully received from the Natural Sciences and Engineering Research Council of Canada.

References

1. J. Delgrande and T. Schaub. A consistency-based approach for belief change. *Artificial Intelligence*, 151(1-2):1–41, 2003.
2. J.P. Delgrande, J. Lang, and T. Schaub. Belief change based on global minimisation. In *IJCAI*, Hyderabad, India, 2007.
3. J.P. Delgrande and T. Schaub. A consistency-based framework for merging knowledge bases. *Journal of Applied Logic*, 5(3):459–477, 2006.
4. O. Gauwin, S. Konieczny, and P. Marquis. Iterated belief merging as conciliation operators. In *7th Int Symp on Logical Formalizations of Commonsense Reasoning*, pages 85–92, Corfu, Greece, 2005.
5. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Clingo* = ASP + control: Preliminary report. volume arXiv:1405.3694v1.
6. S. Konieczny and R. Pino Pérez. Merging information under constraints: A logical framework. *Journal of Logic and Computation*, 12(5):773–808, 2002.
7. P. Liberatore and M. Schaerf. Arbitration: A commutative operator for belief revision. In *Proc of 2nd WOCFAI-95*, pages 217–228, 1995.
8. P. Liberatore and M. Schaerf. Brels: A system for the integration of knowledge bases. In *Proc 7th Int Conf on Principles of KR&R*, pages 145–152, 2000.
9. Nicolas Schwind, Katsumi Inoue, Gauvain Bourgne, Sébastien Konieczny, and Pierre Marquis. Belief revision games. *AAAI*, 2015.
10. E. Würbel, R. Jeansoulin, and O. Papini. Revision: An application in the framework of GIS. In *Proc 7th Int Conf on Principles of KR&R*, pages 505–515, 2000.