Belief Change Format

Paul Vicol

June 12, 2014

1 Introduction

This document describes a standard file format for specifying belief change problems. A belief change problem is a graph of connected agents, each with a set of beliefs, expressed as propositional formulas, and a certain degree of expertise in each atom in the alphabet. The goal of the problem is to determine how the beliefs of the agents change as a result of incorporating beliefs shared by other agents. A solution to a belief change problem is a new graph that has the same topology (i.e. agents and edges) as the original graph, but different formulas for agents, and possibly different levels of expertise for the atoms of each agent.

The format presented here is an extension of the DIMACS graph format for clique and colouring problems. It also incorporates ideas for specifying formulas from the DIMACS satisfiability format.

2 File Format

An input file is an ASCII file where information is contained in lines. The first character of a line signifies the type of the line. There are nine types of lines: comment lines, graph type lines, node lines, alphabet lines, formula lines, atom weight lines, edge lines, and entailment-based and consistency-based integrity constraint lines. These types are as follows:

- **Comment Lines**

  A comment line begins with a lowercase **c**, and contains human-readable notes about the file. Comment lines can appear anywhere in the file.

  Note: Blank lines (i.e. lines consisting of only whitespace) are ignored; it is not necessary to precede a blank line with a **c**. Blank lines can delineate the sections of a file, to increase readability.

  ```
  c This is a comment line.

  c The blank line above this line does not need the initial c.
  ```

- **Graph Type Lines**

  A graph type line begins with a lowercase **t** and indicates whether a graph is directed or undirected. There is at most one graph type line per file, and it must occur before any edges are defined. The graph type line has the following format:

  ```
  t TYPE
  ```

  where `TYPE` is one of the keywords "directed" or "undirected." The graph type line is optional; by default, graphs are directed.

  To make a graph undirected, use:

```
t undirected
```

- **Node Lines**

  A node line begins with a lowercase **n**, and gives the ID(s) of one or more nodes. Each node is identified by an ID that is a nonnegative integer.

  A node line has the following format:

  ```
  n NODE_ID_1 NODE_ID_2 ... NODE_ID_N
  ```

  A node line can give the ID of a single node, as:

  ```
  n 1
  ```

  or the IDs of multiple nodes, as:

  ```
  n 1 2 3
  ```

  or the IDs of a range of nodes, as:

  ```
  n 0..10
  ```

- **Alphabet Lines**

  An alphabet line begins with a lowercase **a** and specifies one or more atoms in the common alphabet. The atoms in the alphabet must be specified explicitly. The format of an alphabet line is:

  ```
  a ATOM_1 ATOM_2 ... ATOM_N
  ```

  After the initial **a**, each alphanumeric string delimited by whitespace is interpreted as an atom.

  The following example declares atoms p, q, and r:

  ```
  a p
  a q
  a r
  ```

  This can be written more succinctly as:

  ```
  a p q r
  ```

- **Atom Weight Lines**

  An atom weight line begins with a lowercase **w** and gives the weight of an atom for a specific agent. Weights on atoms represent degrees of expertise of an agent in some "field." An atom weight line has the following format:

  ```
  w NODE_ID ATOM WEIGHT
  ```

  Where `NODE_ID` and `ATOM` are declared in the file, and `WEIGHT` is a positive integer. Atom weights are optional; by default, every agent holds each atom with weight 1.

  To specify that node 2 holds atom q with weight 6, use:

```
w 2 q 6
```

- **Formula Lines**

  A formula line begins with a lowercase **f** and declares a formula belonging to a node. The format of a formula line is:

  ```
  f NODE_ID FORMULA
  ```

  The format of the formula is the DIMACS SAT format, but using letters rather than numbers to represent variables. The rules of the DIMACS SAT format are reproduced here:

  1. p and $-p$ are formulas for all p.
  2. If f is a valid formula, so is (f).
  3. If f is a valid formula, so is $-(f)$.
  4. If $f_1, f_2, \ldots, f_k$ are valid formulas, so is $*(f_1 f_2 \ldots f_k)$.
  5. If $f_1, f_2, \ldots, f_k$ are valid formulas, so is $+(f_1 f_2 \ldots f_k)$.

  Whitespace can be used anywhere between the parts of a formula. The "$*$" operator represents conjunction, the "$+$" operator represents disjunction, and "$-$" represents negation. The final formula must be of the form (f) for a valid formula f. This allows the formula to be split over multiple lines; formula lines are the only lines that are not necessarily terminated by a newline character.

  An example formula line representing $p \wedge (q \vee r \vee \neg w)$ for node 1 is:

  ```
  f 1 (*(p +(q r -w)))
  ```

- **Edge Lines**

  An edge line begins with a lowercase **e** and declares an edge by specifying its initial and terminal nodes. By default, edges are directed; to obtain undirected behavior, either encode both directions explicitly, or declare the graph to be undirected in the graph type line. The format of an edge line is:

  ```
  e FROM_NODE_ID TO_NODE_ID
  ```

  For example,

  ```
  e 1 2
  e 2 3
  ```

- **Entailment-Based Integrity Constraints**

  An entailment-based integrity constraint line begins with a lowercase **m** and specifies a formula that must be entailed by the knowledge bases at all the nodes in the graph. The format is as follows:

  ```
  m FORMULA
  ```

  where the formula is given in the DIMACS SAT format.

  For example, to specify that $p \wedge q \wedge \neg r$ is an entailment-based integrity constraint, use:

```
m (*(p q -r))
```

- **Consistency-Based Integrity Constraints**

  A consistency-based integrity constraint line begins with a lowercase **s** and has the following format:

  ```
  s FORMULA
  ```

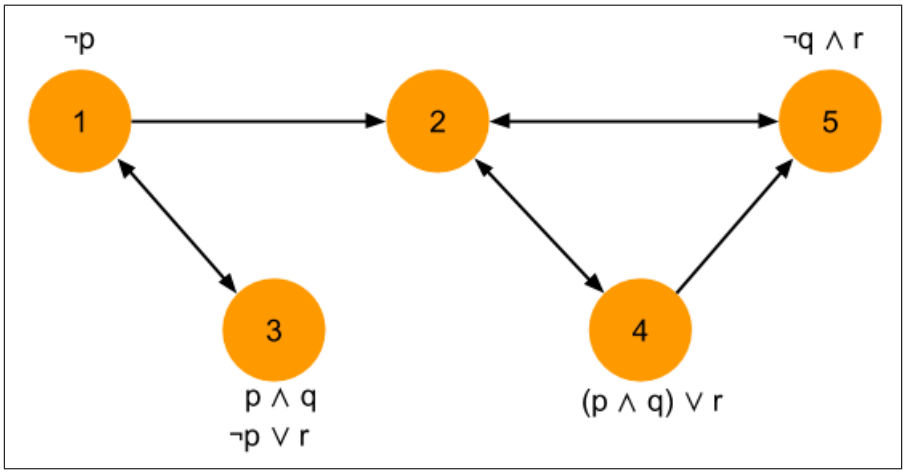
  where the formula is in the DIMACS SAT format.

  To specify that $\neg p \vee \neg r$ is a consistency-based integrity constraint, use:

  ```
  s (+(-p -r))
  ```

3 An Example

The following example shows how a whole graph is encoded.



The weights on atoms are as follows (atoms not mentioned have default weights):
$w_1(p) = 5, w_3(p) = 2, w_3(q) = 3, w_3(r) = 1, w_4(p) = 4, w_4(q) = 3, w_5(q) = 4, w_5(r) = 3$.

The encoding of this graph in the belief change format is:

```
t directed

n 1..5

a p q r

w 1 p 5
w 3 p 2
w 3 q 3
w 3 r 1
w 4 p 4
w 4 q 3
w 4 r 1
w 5 q 4
w 5 r 3
```

```
f 1 (-p)
f 3 (*(p q))
f 3 (+(-p r))
f 4 (+(*(p q) r))
f 5 (*(-q r))

e 1 2
e 1 3
e 3 1
e 2 4
e 4 2
e 2 5
e 5 2
e 4 5
```

4 Output Files

Output files have the same format as input files. Belief change operations take a graph as input, and produce a new graph as output, representing a modification of the state of the original graph. Since the output format is the same as the input format, the output of one operation can be used as the input to another. Chaining outputs to inputs is one way to achieve iterated belief change.

5 Purpose of the Format

This format is meant to serve as a standard for communication between programs. It is designed to be easy to parse and convert to other formats. For example, it is easy to translate this format to ASP code, creating a usable problem instance.

However, this format is not a particularly intuitive way to specify problems directly. For this, it is useful to have a high-level interface that provides a domain-specific language for creating graphs, performing various belief change operations, and querying graphs.