Placeholder

Sam Tadey

2021

Table of Contents

[Summary 2](#_Toc76121688)

[Tool Layout 2](#_Toc76121689)

[User Guide 3](#_Toc76121690)

[Define a Vocabulary 3](#_Toc76121691)

[Generate a Trust Graph 3](#_Toc76121692)

[Update the Trust Graph 4](#_Toc76121693)

[Manual Update 4](#_Toc76121694)

[Add Reports 4](#_Toc76121695)

[Weights 5](#_Toc76121696)

[Trust Graph Constraints 7](#_Toc76121697)

[Defining Triangle Inequality 8](#_Toc76121698)

[Value Unchanged 9](#_Toc76121699)

[Next Available Value 9](#_Toc76121700)

[MinMax/MaxMin Distance 9](#_Toc76121701)

[Specify Initial Beliefs 10](#_Toc76121702)

[Hamming Ranking 10](#_Toc76121703)

[File Ranking 10](#_Toc76121704)

[Specify Sentence to Revise By 11](#_Toc76121705)

[Revise 11](#_Toc76121706)

[Results 12](#_Toc76121707)

# Summary

The Distance Visualizer is a tool that can be used to visualize distances between states. This document is a user guide illustrating how to use the belief revision tool. The tool actions must be used in a specific order to yield results. The general order of actions is:

1. Define a vocabulary.
2. Generate a Trust Graph.
   1. Modify the trust graph with reports.
3. Specify beliefs.
4. Specify a sentence to revise by.
5. Revision.

## Tool Layout

DO TOOL LAYOUT

Graphical user interface

Description automatically generated

# User Guide

The Distance Visualizer is a simple tool to use to visualize belief revision with trust. There are several steps needed to execute the software effectively. Each step will be discussed in detail with examples and technical description.

## Define a Vocabulary

A propositional vocabulary must be defined for many of the following components to work. Defining a vocabulary must be done in the **Propositional Vocabulary** text field and follow a specific format. Each vocabulary member must be a character letter. To separate each character, use a comma. Use the image below for propositional vocabulary input guidance.

Graphical user interface, text, application

Description automatically generated

## Generate a Trust Graph

Generating a basic Trust Graph requires only a vocabulary as a dependency. Once that has been defined, selecting the **Generate Trust Graph** action will generate a graph in the **Trust Graph** pane. The **Generate Trust Graph** action can be found in the **Actions** pane.

Graphical user interface, text, application

Description automatically generated

After the **Generate Trust Graph** action has been executed the trust graph will be generated.

Table

Description automatically generated

## Update the Trust Graph

There are two ways to modify trust values in the **Trust Graph**. Values can be changed manually one-by-one or automated through the **Add Reports** action.

### Manual Update

Each white **Trust Graph** grid item can be modified manually by selecting that item and changing the value through keyboard input. If an invalid value is inputted, the grid item value will be reverted to the value before the change, and an error message will appear in the **Errors** pane.

### Add Reports

Adding Reports is another method of updating trust values between a set of states. The **Formula** field will be filled in by a valid propositional formula. The variables used in the formula must match those defined in the **Propositional Vocabulary** field. The **Result** field must be a one-character input of either a **0** or **1**.

Graphical user interface, application

Description automatically generated

When satisfied with report input, the **Add Reports** action will apply the reports to the **Trust Graph**, updating values to account for the new information.

Table

Description automatically generated

### Weights

Weights can be set to modify values added/subtracted during the **Add Report** process. These weights allow specific reports to be weighted more highly, or lower than others. Additionally, should a user find a reason for specific variables to be more impactful in the trust graph, variable weights can be tweaked to allow the increment/decrement values within reports to change.

1. Report Weights change the value difference for ALL edges affected by the report.
2. Variable Weights change the value difference on an edge-by-edge basis, based on the differences between states and variable weights.

#### Report Weights

The user would like to add these reports to the trust graph.

Graphical user interface, application

Description automatically generated

The weight input for each report modifies the value to increment/decrement for all affected edges in that specific report. The default value to inc/dec is 1. Here is how each report will affect the trust graph.

1. **‘a’** will increase all affected edges by 2.
2. **‘b & c’** will increase all affected edges by 1.
3. **‘a | b’** will decrease all affected edges by 0.5.

The weight option for each report allows the user to uniquely weight each report, and to uniformly modify trust graph values across the entire report.

#### Variable Weights

Variable weighting allows the user the option to change edge values on a variable-by-variable basis. Here is a quick example of how the weights are integrated into a report.

Graphical user interface, application

Description automatically generated

Take the example weights above, with a default weight for a report (1). Variable weights must be combined into one value before they can be applied to the existing value in the graph. This is done by comparing the differences between the two states being modified.

Eg. (000,100)

The only differing variable between these two states is the variable ‘a’. In terms of the weighting algorithm, this is the only weight being considered when converting variable weights into the inc/dec weight. The value is calculated using this formula.

(Variable weight + variable weight) / number of differing variables

0.5 / 1 = 0.5

Therefore, the report would increment the (000,100) edge by 0.5 with this variable weighting.

This value would differ from others in the same report, take (000, 110).

The variable differences in this state are now ‘a’ and ‘b’, meaning both variable weights must be considered. Using the formula:

0.5 + 1 / 2 = 0.75

Therefore, the existing trust graph value would be modified by 0.75.

#### Combining Weights

Report weights and variable weights are combined when both are set to values other than default (1). They are combined as the multiplication of report and resulting variable weight.

Report Weight \* Variable Weight

To use the example from the section above, with resulting variable weights of 0.5 for (000,100) and 0.75 (000,110), say the report weight has been changed from 1 to 2.

The resulting weights would be:

(000, 100) Report Weight (2) \* Variable Weight (0.5) = 1

(000, 110) Report Weight (2) \* Variable Weight (0.75) = 1.5

## Trust Graph Constraints

There are two main constraints that contribute to value change from one Trust Graph state to another.

1. Values > 0.
2. Values must satisfy Triangle inequality.

There are specific response actions the user can take to handle situations of Triangle Inequality when they arise. An option can be selected in the **Triangle Inequality Response** pane below the **Reports** pane.

Graphical user interface, text, application

Description automatically generated

### Defining Triangle Inequality

Basic Triangle Inequality is defined as:

**z ≤ x + y**

There are multiple ways that Triangle Inequality can be violated when changing values throughout the Trust Graph. Let us go through a couple examples using the triangle below.

000

z = 3

x = 2

111

y = 2

010

Problems arise in the Trust Graph when any kind of value is changed in the graph. Say the user has provided a positive report to the system that would modify all affected graph values by 2. This would change the value of **z** from 3 to 5. But **z ≤ x + y** must be maintained, and the value of 5 violates this formula. Therefore, this value is invalid.

Say the user would like to add a negative report to the system, which would decrease some values in the trust graph. The report a user has entered would decrease values in the graph by 1.5. One value affected is the value y, which would modify the value from 2 to 0.5. This value is also invalid due to Triangle Inequality. **z ≤ x + y** or 3 **≤** 2 + 0.5.

There are 3 implementations that handle Triangle Inequality Events when they occur. The **Value Unchanged**, **Next Available Value**, and **MinMax/MaxMin Distance** options all handle Triangle Inequality.

### Value Unchanged

The Value Unchanged option handles triangle inequality resetting the affected graph edge back to its original value. Say the value of (000/111) would be changed from 3 to 4, but the value 4 is invalid. The value would simply be set back to the original value of 3.

### Next Available Value

The Next Available Value handler chooses a different value for the edge, a value that does not violate Triangle Inequality. It sets the value to the max available value for that edge, given values to intermediate states. I will use this example to demonstrate.

The edge (000,111) has been set to 5.0, but this value is invalid because the triangle inequality would be violated by these intermediate states (001,100). I will set some example values for the other edges.

1st Triangle: (000,111) <= (000,001) + (001, 111) **or 5 <= 2 + 2**

2nd Triangle (000,111) <= (000,100) + (100, 111) **or 5 <= 1 + 2**

To set a valid value for the edge (000,111) this handler considers both invalid triangles and chooses minimum valid value out of these states. In this example the algorithm would only consider the intermediate states 001 and 100. The valid values for each triangle are different and indicated below.

1st Triangle Setting (000,111) to **4 <= 2 + 2**

2nd Triangle Setting (000,111) to **3 <= 1 + 2**

The reason we set the new value to the min of the values is because this value is guaranteed to satisfy Triangle Inequality for **ALL** the intermediate triangles. Eg. Setting the value to 4 would satisfy the 1st triangle, but not the 2nd.

In conclusion, the new value is set to the min value of the adjacent edges for all the triangles being considered.

**2 + 1 < 2 + 2**, therefore we choose **2 + 1** or **3** for the handled value.

A very similar process is used when **decreasing** values in the trust graph and triangle inequality arises. The only difference, in this case, is that the system looks at different edges to ensure triangle inequality, and the value is set to the **MAX** of the available values.

### MinMax/MaxMin Distance

Minimax and Maximin distances are used to handle triangle inequality when this option is selected. The Process for selecting a value is identical to the **Next Available Value** method, but now **ALL** intermediate states are considered during the process, not just the state/triangles that would be invalid.

To find a valid value between (000,111) if the specified value is invalid and **increased** from the initial value, consider all intermediate states.

(001,010,011,100,101,110).

So, we are taking the **Min** of Eg. (000, 001) + (001,111) or (000,010) + (010, 111) ….

When the value is invalid and being **decreased** from the original value the value selected is the **Max** of all intermediates.

So, **Max** of (000,001) + (001,111) or (000,010) + (010, 111) …

## Specify Initial Beliefs

Initial Beliefs of an agent are represented by a Ranking Function. There are two ways to specify an initial belief state. The first is by inputting a propositional formula. The second is by loading a file with specific state/ranking values.

### Hamming Ranking

To define a belief state manually through the tool, ensure the **Hamming Ranking** option is selected in the drop-down menu. This selection will allow text input under the **Belief** header. The possible states that satisfy the propositional formula are given a rank of 0, the lowest and most likely belief value. All other states not defined by the formula are then given a rank based on the hamming distance to any one satisfied state.

Shape

Description automatically generated with medium confidence

For example, the ranking function defined by the input (**a | b)** would be:

010,011,100,101,110,111 = 0

000,001 = 1

000 and 001 are given a rank 1 because the minimum hamming distance to any of the satisfied states is 1.

### File Ranking

An initial belief state can be easily defined through file input.

DEFINE INPUT FILE FORMAT

Graphical user interface, application

Description automatically generated

## Specify Sentence to Revise By

The last step before revision is to specify a sentence to revise by. The **Sentences** text field takes a propositional formula as an input. Just as with other fields, ensure that you are using variables defined in the **Propositional Vocabulary** field.

Shape, square

Description automatically generated

## Revise

When all the steps have been taken, the last thing to do is execute the **Revise** action.

Graphical user interface, text, application

Description automatically generated

After the **Revise** action has been run, the resulting states to revise by will be displayed in the **Results** text field. They have been converted into propositional variables for readability.

A picture containing shape

Description automatically generated

Each line corresponds to an output state. So, the first two lines could be read like this:

Eg. 100 or 111

## Results

After following these steps, the tool should look something like the next image. The **Revise** action does not signify the end of belief revision, and new beliefs, sentences, and trust values can be introduced to generate differing outputs.

Graphical user interface

Description automatically generated with medium confidence