Distance Visualizer

Sam Tadey

2021

Table of Contents

[Summary 2](#_Toc73450971)

[User Guide 2](#_Toc73450972)

[Grid Generation 2](#_Toc73450973)

[Grid Modification: Reports 3](#_Toc73450974)

[Grid Modification: Manual 4](#_Toc73450975)

[Constraints 6](#_Toc73450976)

[Implementation Notes 7](#_Toc73450977)

[Report 7](#_Toc73450978)

[DistanceState 8](#_Toc73450979)

# 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 software, as well as a companion document describing the back-end implementation of the Visualizer.

# User Guide

The Distance Visualizer is a simple tool to use to visualize distances between states. The tool allows users to manually change distance values, or by adding reports to automate the process.

## Grid Generation

The first step to using the visualizer grid generation. The grid uses propositional variables to generate a default grid. The **Generate Default Grid** action assigns default values to all state combinations given the inputted propositional variables.

Graphical user interface, application, Word

Description automatically generated

Propositional variable input is case-sensitive. Variables must be separated by commas, with no spaces. Variables must also be one character. Eg (no variables such as AA).

Successful propositional variable input will generate a grid such as the one below. It is also important to note that state ordering will correspond to the order in which the variables were inputted. If variables are inputted such as eg. (b,c,a), states must now be read in that order.

Table

Description automatically generated

## Grid Modification: Reports

One way to modify grid values is through reports. The reporting function takes two inputs. The first is a propositional formula, the second is a Boolean value. A report can be found to be true or false. A true report is used to decrease distances between states, while false a false report increases distances between states. Below is an example of the process of adding a false report to the distance function.

Table

Description automatically generated

2.

1.

To add a report to the distance function, follow these steps:

1. Input a propositional formula into the **Formula** text box. Use only propositional variables used in the grid generation step, as well as the following relational symbols (|, &, ~, =>, <=>). Parenthesis are allowed.
2. Input a 0 or 1 into the **Result** text box. (1: True, 0: False).
3. Press the **Add Report** action

The report will then be added to the current grid state. The grid will update to show the new values calculated by adding this new report. The grid above has had its values updated by the **Add Report** function. Since this is a false report, modified values have increased in value.

## Grid Modification: Manual

Distances between states can also be changed manually by interacting with the grid directly. Input will be accepted if it is valid. The input must be numerical, and this is the first condition for acceptance. Next the input must be valid regarding distance constraints. Current constraints require values to be greater or equal to zero or satisfy the triangle inequality condition.

The program handles invalid input by resetting the form to the original input and then displaying an error message indicating the constraint that was violated.

Table

Description automatically generated

Table

Description automatically generated

## Constraints

The distance function constraints are the following:

1. Distance values must be greater than or equal to 0.
2. Distance values must not violate triangle inequality.

Below is an example of some values that would violate the triangle inequality constraint.

Error:

The value being modified is the 111/011 value to a value of 6.0. Triangle inequality disallows this value because of the distances between the two states in question (111 and 011) to all intermediate states. The intermediate state 000 violates triangle inequality causing an error. 011/111 < 000/011 + 011/111.

111

6

1

3

011

000

Table

Description automatically generated

# Implementation Notes

There are four classes that are used to manage the distance visualizer:

* **State**: Represented as a string of 0’s and 1’s, with length equal to the propositional vocabulary
* **BeliefState**: A collection of State
* **DistanceState**: Manages the distances between all possible state combinations.
* **Report**: A formula and reported value which indicates the accuracy of the formula

## Report

A Report object is used by a DistanceState object to modify distances between States. The implementation of the Report class is simple. This object is intended to store a String formula and an Integer ‘boolean’ value. The only function it has, other than getters and setters, is to convert its formula member variable into a BeliefState object.

**Member Variables:**

1. String formula
2. Int report value (0 or 1)

Ex. Report object

**String** = “a&b”

**int** = 0

**Functions:**

**convertFormToStates(Set characters vocab)**

return a BeliefState representation of the formula member variable.

## DistanceState

A DistanceState object stores the distance between two states. When an object is created, all possible states for a given vocabulary are generated and stored as a BeliefState. Next, all possible combinations of these states are stored in the Hashmap data member with a default distance value.

**Member Variables:**

1. Set of characters as vocab
2. BeliefState containing all possible states, given the vocabulary
3. Hashmap of state combinations storing the distance between them.

Ex. DistanceState object:

**Set** = {A,B}

**BeliefState** = {00,01,10,11}

**HashMap** = <state1, HashMap<state2, distance>>

[<00 , <01, 2>, <10, 1>, <11, 1>>

<01 -> <10, 3>, <11, 1>>

<10 -> <11, 4>>]

**Functions:**

**getDistance(State s1, State s2)**

returns the distance based on both states

**setDistance(State s1, State s2, double distance)**

sets the distance based on both states

**checkTriangleInequality(BeliefState possiblestates, State s, State u, double val, List errors)**

for each intermediate state t

if val violates triangle inequality between (s,t) and (t,u)

add to errors

return true

return false

**modByReport(BeliefState s1, BeliefState s2, double val)**

for each combination of states in s1 and s2

add val to existing val

**modByReport(BeliefState s1, double val)**

for each combination of states in s1

add val to existing val

**addReport(Report r)**

convert report to states

separate states into two categories

* + States that satisfy formula
  + States that do not satisfy formula

If r val == 0

modByReport(satset, unsatset, 1)

if r val == 1

modByReport(satset, -1)

modByReport(unsatset, -1)

return errors if any