Evaluating the Performance of Equibel

Jesse Elliott

Abstract

This paper presents data collected from testing a software system called Equibel. Equibel is an implementation of a consistency based multi-agent belief change framework. A belief change problem is an undirected graph where nodes represent agents and edges represent communication links. A set of propositional formulas are assigned to each agent. Solving the problem involves each agent incorporating as much information as consistently possible from the others. Equibel solves belief change problems. This paper presents data collected from measuring Equibel's solving time for various belief change problems.

1 Introduction

A general framework for consistency based multi-agent belief change is presented in [1]. An implementation of this framework is presented in [2]. The implementation is called Equibel. It can be found on GitHub with the following link: https://github.com/asteroidhouse/equibel. This paper presents data collected from testing Equibel's performance in solving various belief change problems. The notation used in this paper follows the notation defined in [2].

A belief change problem is an undirected graph, where nodes represent agents and edges represent communication links. A set of propositions is assigned to each node. These propositions represent beliefs. Solving the problem involves each agent incorporating as much information as consistently possible from the others.

I investigate 3 types of graph configurations; complete graphs, path graphs, and star graphs.

Section 2 explains testing methodologies. Section 3 presents the data collected from testing Equibel. Section 4 is a brief description of some changes that have been made in Equibel since [2] was written. I discuss how these changes have resulted in faster solving time and more efficient memory allocation. In section 5, I discuss Equibel's software architecture. Equibel is made of a python component, and an asp component. I investigate and compare the proportion of time spent in these components during Equibel's solving process. Conclusions are presented in section 6.

2 Methodologies of Testing

For various belief change problems, I recorded Equibel's solving time using the python module 'time'.

Belief change problems can vary from one another in terms of the type of graph (i.e. the arrangement of edges), the number of nodes, the alphabet size**,** and the type of formulas assigned to each node. My testing consisted in varying these parameters

individually while recording the changes in solving time for the corresponding problems. I refer to a set of tests that focus on a specific parameter as an experiment. In section 3, I present data collected from a series of these experiments. I display the results of an experiment in a graph. On the graph, a changing parameter is displayed on the horizontal axis. This parameter is either the number of nodes in the graph (i.e. agents in the problem), or the alphabet size. The solving times for the corresponding problems are displayed on the vertical axis. For these experiments, I look at each type of graph configuration one at a time. I consider two types of propositional formulas; conjunctions of literals, and hard satisfiability formulas.

3 Experiments and Data

In this section I will explain the individual experiments and present the data.

All formulas in section 3 are randomly generated. Each problem with a randomly generated formula has a unique solving time. We are interested in the problems average solving time for a randomly generated formula. Thus, problems with random formulas are generated and solved 100 times. I record the mean solving time of the 100 repetitions. This time is recorded on the vertical axis of the experiments graph. However, occasionally a solving time will be much larger than the average solving time. In this case I consider the time an outlier. The mean solving time that is recorded does not include outliers. That is, if there are 10 outliers, then the time recorded is the mean of the remaining 90 repetitions that are not outliers. For each problem there is a cutoff time. Any solving time grater than or equal to the problems cutoff time is considered an outlier. Below the graph of each experiment will be a table displaying the number of outliers, and their cutoff times.

On the graph displaying the data, at each coordinate point there is a number. This number is the standard deviation of solving times for the corresponding problem.

3.1 Random Conjunctions of Literals

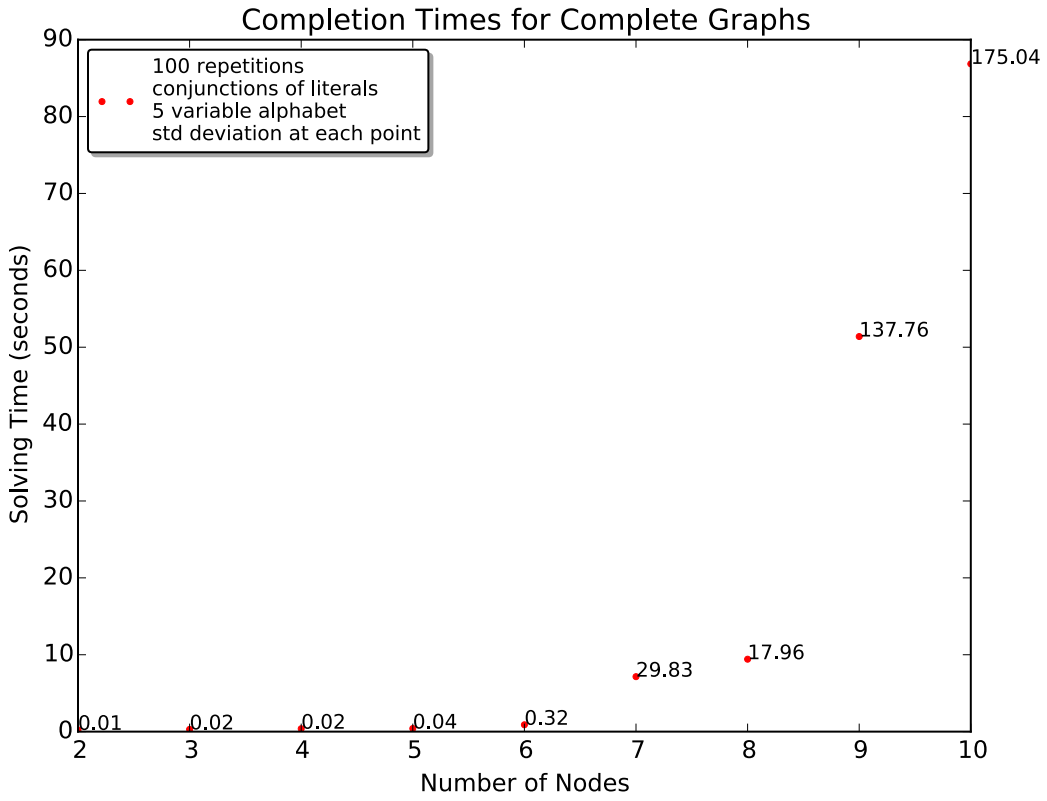
In section 3.1 we consider belief change problems where agents are assigned conjunctions of literals. The formulas are randomly generated. Approximately two thirds of the alphabet is assigned to each agent. That is, if the alphabet contains 10 variables, most likely agents will be assigned a conjunction of 7 literals. But, some agents will be assigned less and some will be assigned more. The alphabet size will be made clear with each experiment.

3.1.1 Complete Graphs

In section 3.1.1, problems involve agents networked into a complete graph.

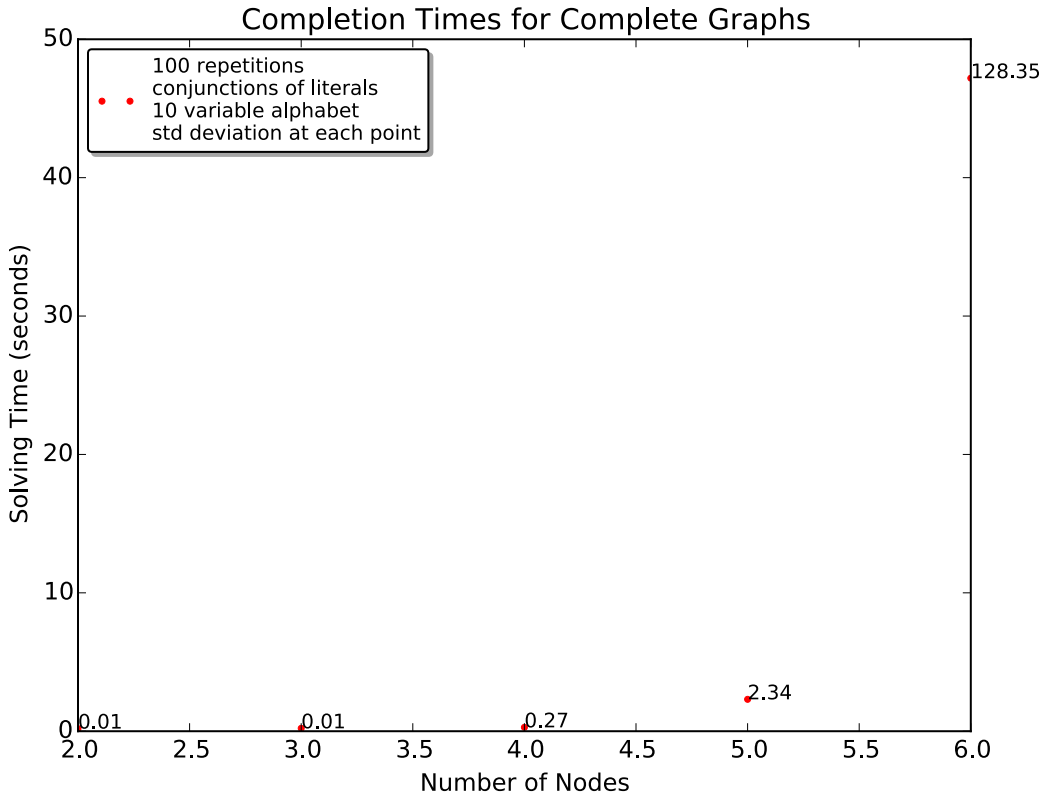
For experiments 1-3, the changing parameter on the horizontal axis is the number of nodes.

Experiment 1
The alphabet contains 5 variables. Nodes range from 2 until 10 with a step size of 1.



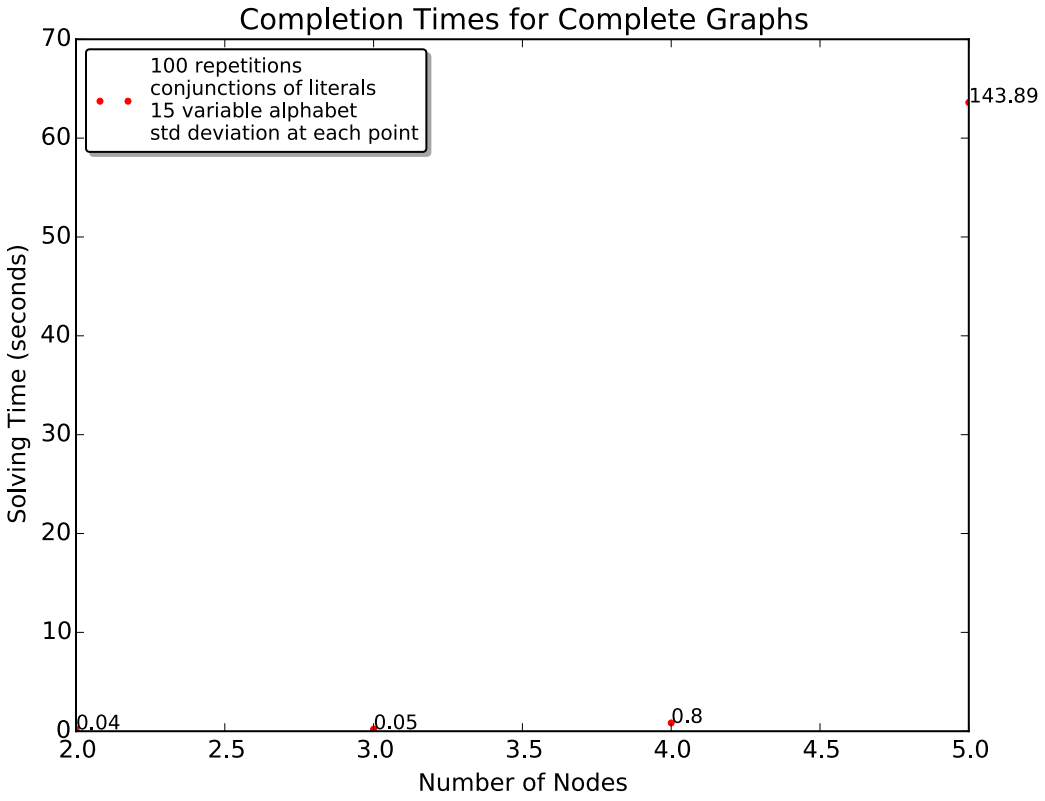
cut off time for outliers (seconds)	300	1000	1000
number of outliers	3	10	10
nodes	2-8	9	10

Experiment 2
The alphabet contains 10 variables. Nodes range from 2 until 6 with a step size of 1.



cut off time for outliers (seconds)	20	1000
number of outliers	8	27
nodes	2-5	6

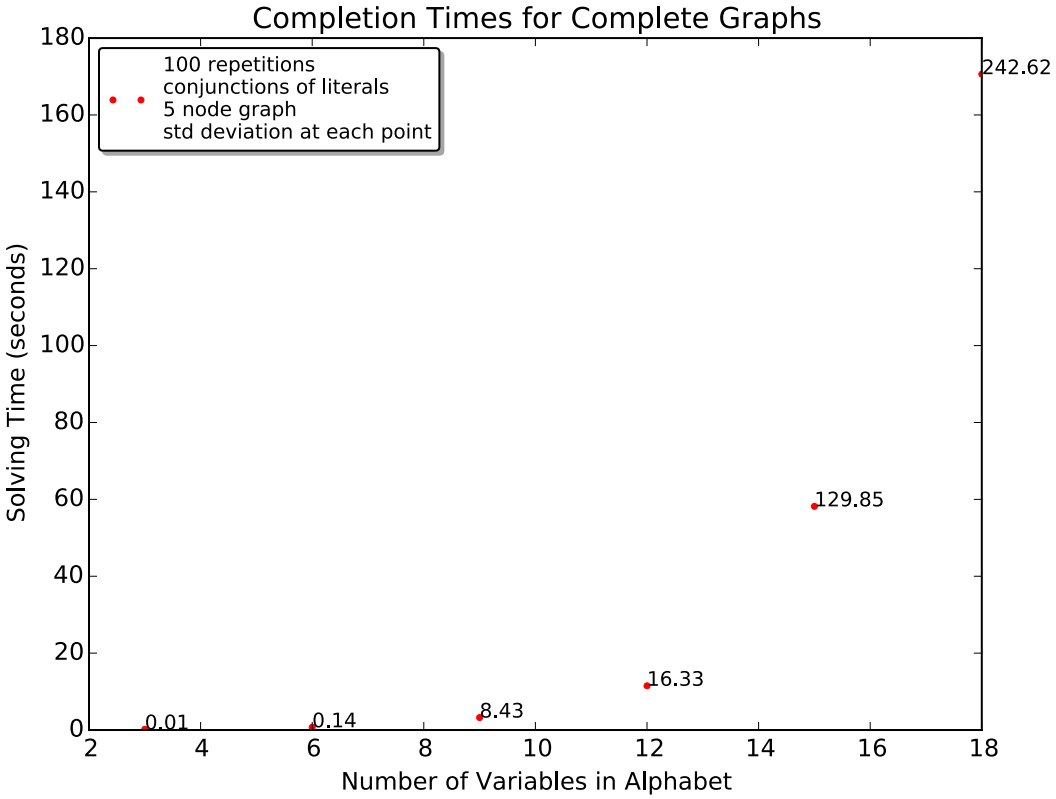
Experiment 3
The alphabet contains 15 variables. Nodes range from 2 until 5 with a step size of 1.



cut off time for outliers (seconds)	25	750
number of outliers	2	23
nodes	2-4	5

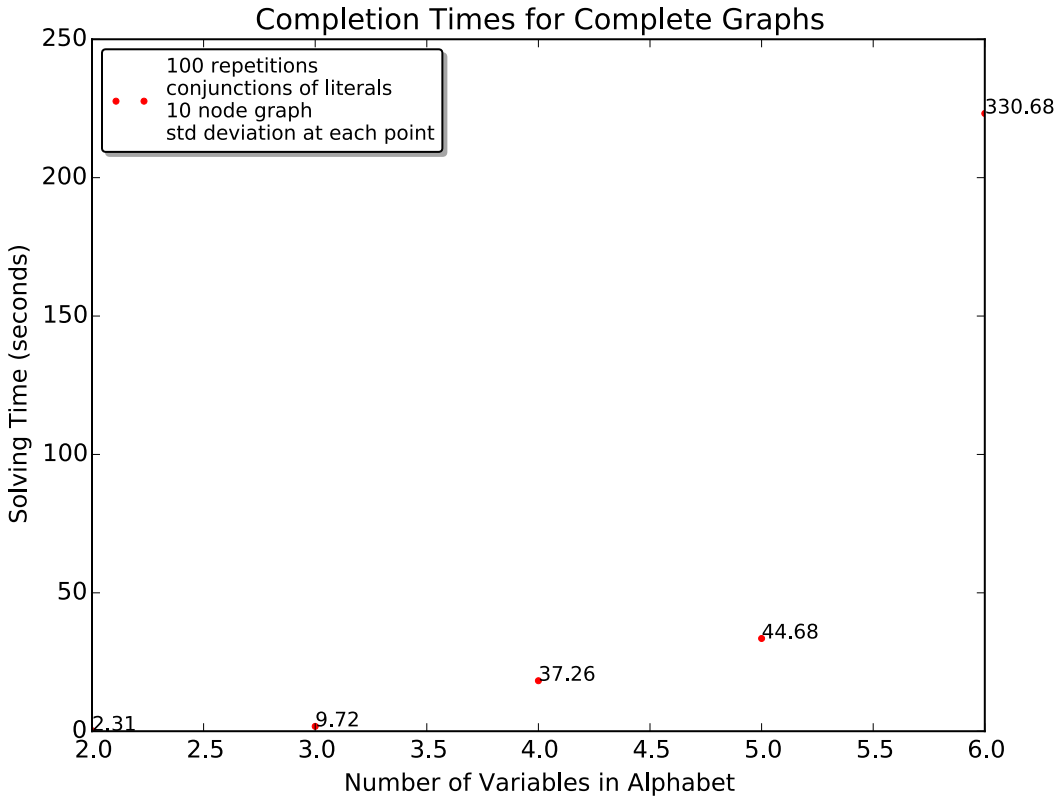
For experiments 4-5, the changing parameter on the horizontal axis is the alphabet size. A larger alphabet results in a conjunction of more literals assigned to each agent.

Experiment 4
Each problem involves a graph of 5 nodes. The number of variables in the alphabet range from 3 until 18 with a step size of 3.



cut off time for outliers (seconds)	200	1000
number of outliers	8	65
nodes	3-12	15-18

Experiment 5
Each problem involves a graph of 10 nodes. The number of variables in the alphabet range from 2 until 6 with a step size of 1.



cut off time for outliers (seconds)	200	1000
number of outliers	43	65
nodes	2-5	6

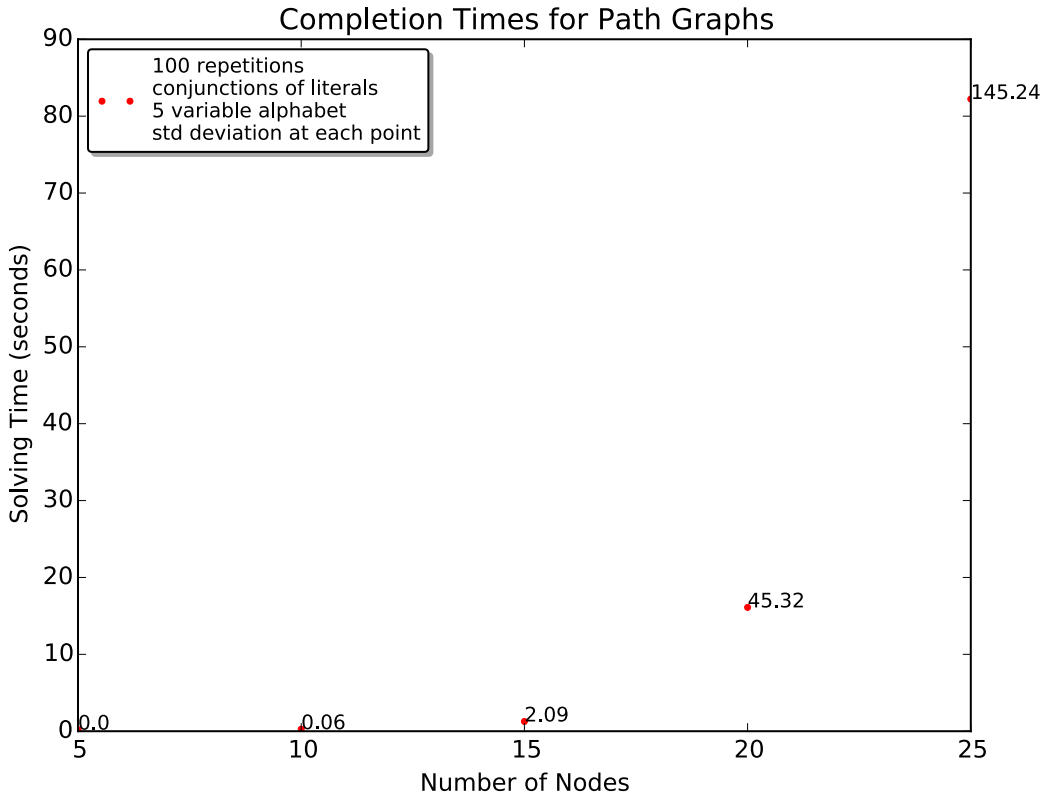
3.1.2 Path Graphs

In section 3.1.2, problems involve agents networked into a path graph.

For experiments 6-8, the changing parameter on the horizontal axis is the number of nodes.

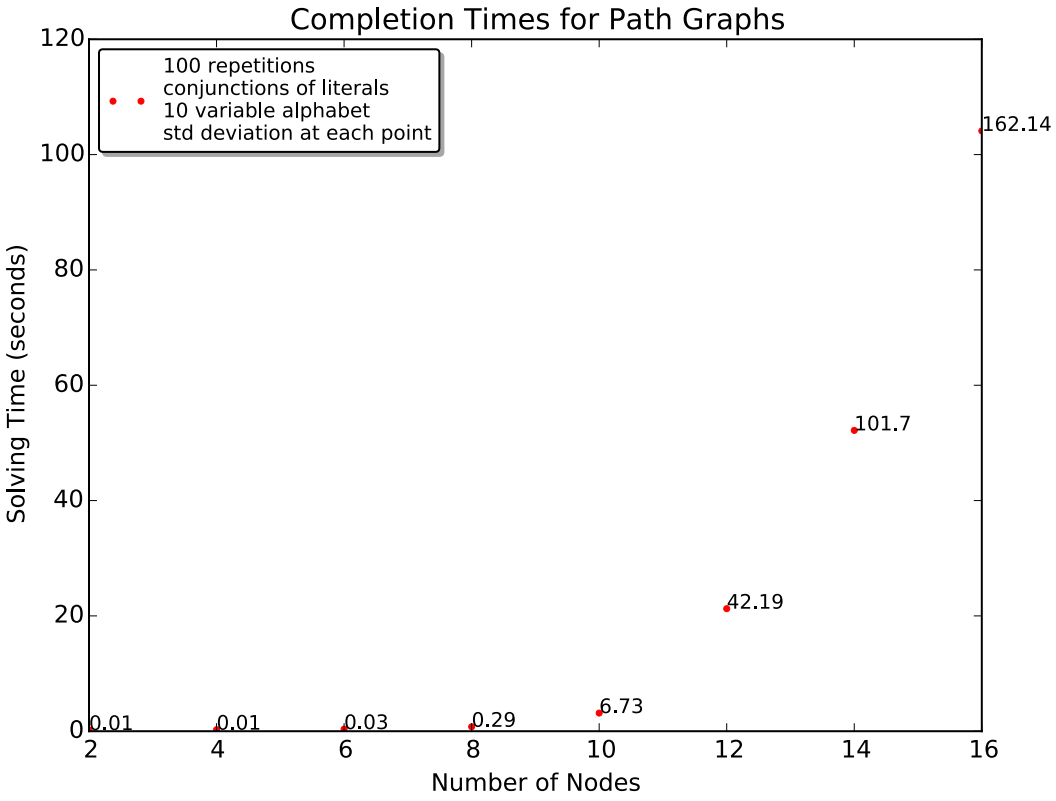
Experiment 6

The alphabet contains 5 variables. Nodes range from 5 until 25 with a step size of 5.



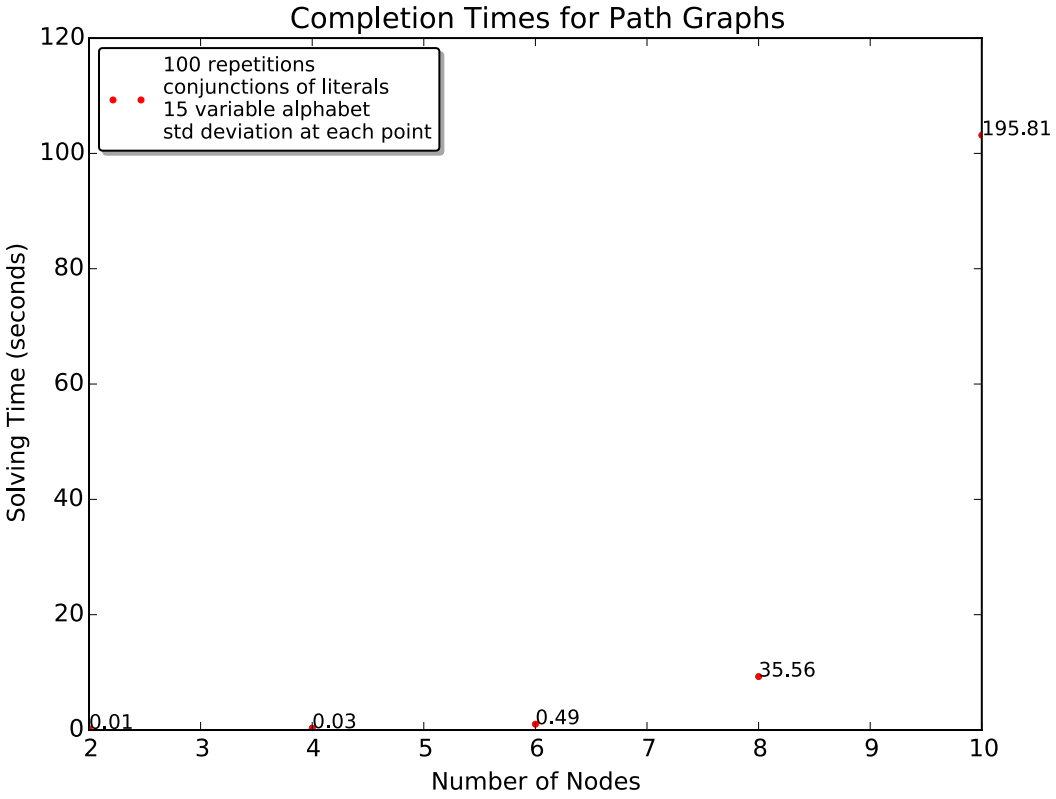
cut off time for outliers (seconds)	25	1000
number of outliers	2	27
Nodes	5-15	20-25

Experiment 7
The alphabet contains 10 variables. Nodes range from 2 until 16 with a step size of 2.



cut off time for outliers (seconds)	400	750	750
number of outliers	12	37	49
nodes	2-12	14	16

Experiment 8
The alphabet contains 15 variables. Nodes range from 2 until 10 with a step size of 2.

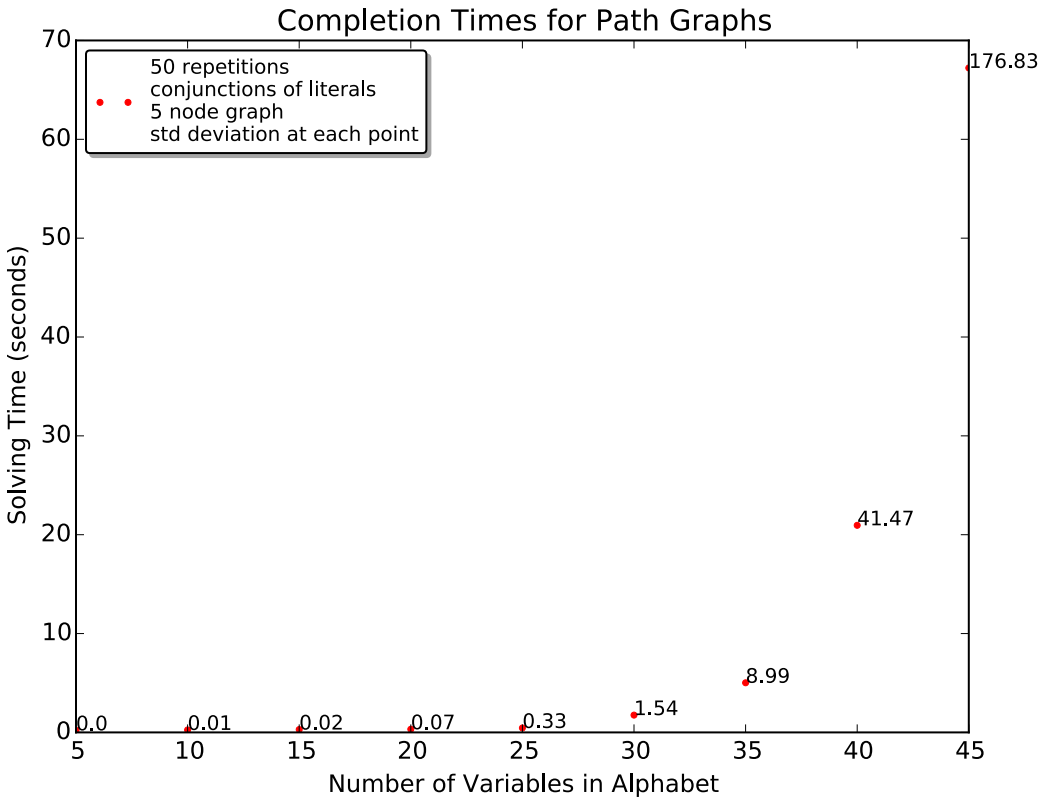


cut off time for outliers (seconds)	400	1000
number of outliers	2	27
nodes	2-8	10

For experiments 9-10, the changing parameter on the horizontal axis is the alphabet size. A larger alphabet results in a conjunction of more literals assigned to each agent.

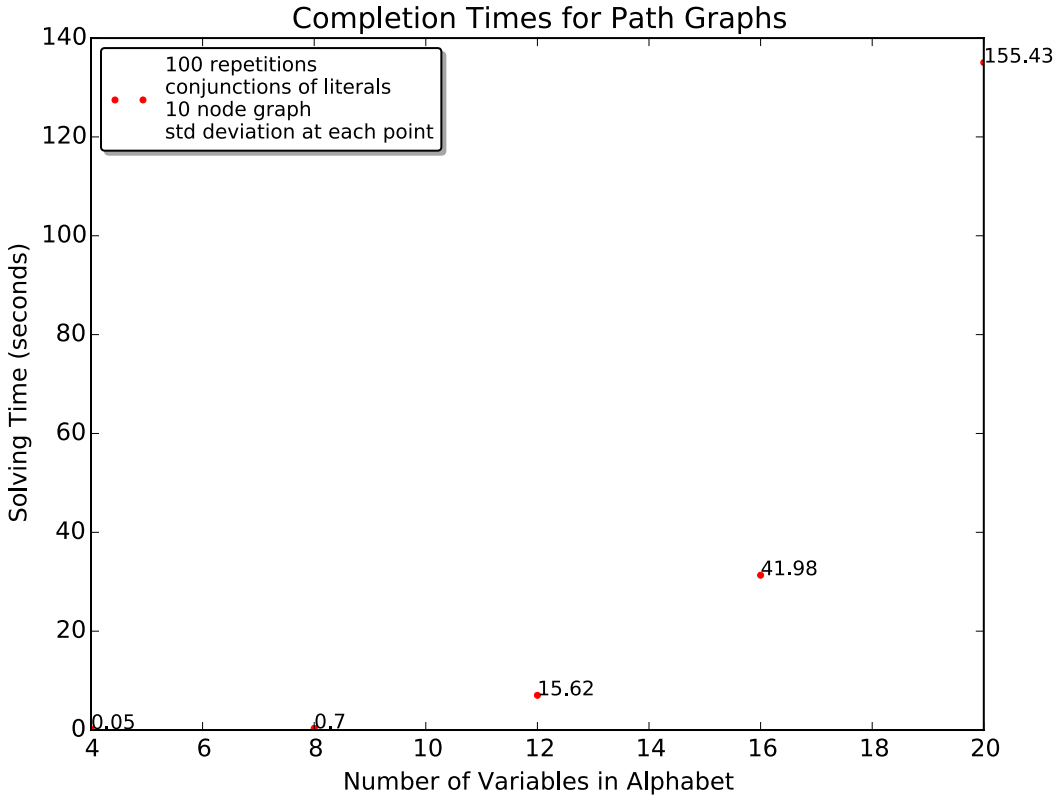
Experiment 9

Each problem involves a graph of 5 nodes. The number of variables in the alphabet range from 5 until 45 with a step size of 5. Unlike the experiments presented so far, problems are repeated only 50 times. The reason for this is that problems in experiment 9 require more memory to solve than those in previous experiments. In order to have enough memory to complete the experiment, we only repeat each problem 50 times.



cut off time for outliers (seconds)	400	1000
number of outliers	10	14
nodes	5-40	45

Experiment 10
Each problem involves a graph of 10 nodes. The number of variables in the alphabet range from 4 until 20 with a step size of 4.



cut off time for outliers (seconds)	400	1000
number of outliers	7	34
nodes	4-16	20

3.1.3 Star Graphs

In section 3.1.3 each problem involve agents networked into a star graph.

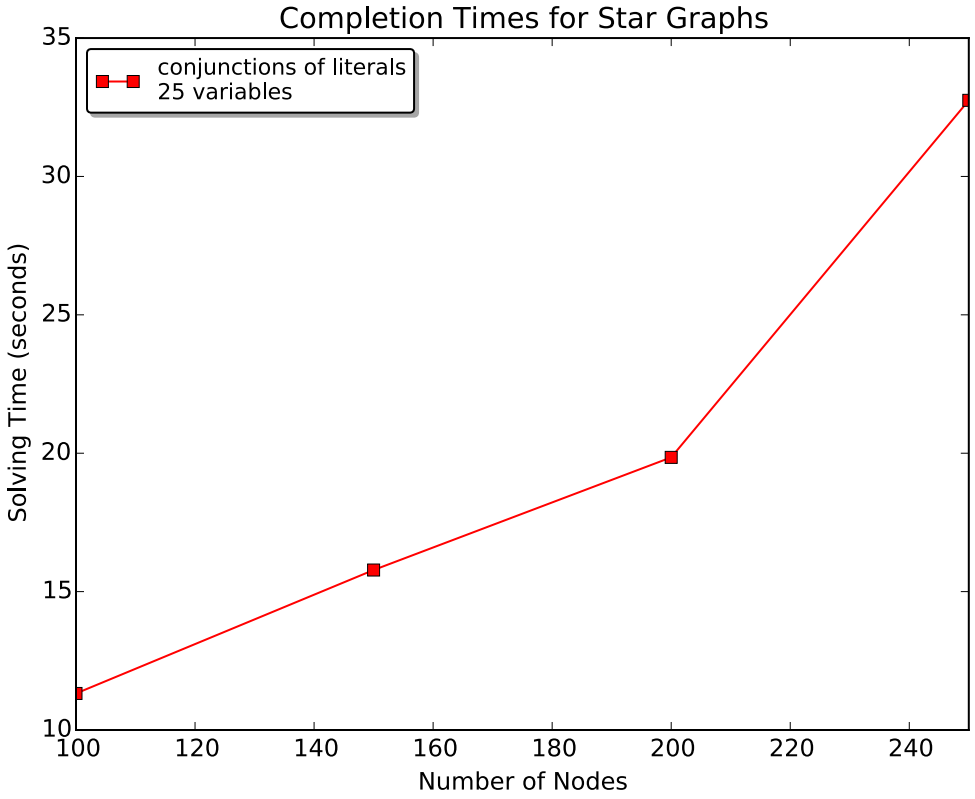
Belief change problems involving star graphs are faster for Equibel to solve than problems involving path or complete graphs. Thus, star graph problems can incorporate more agents, and formulas with more variables. However, problems involving more agents and larger alphabets require more memory. Thus, because problems with star

graph configurations require more memory, the following experiments involve fewer repetitions and fewer steps across the horizontal axis.

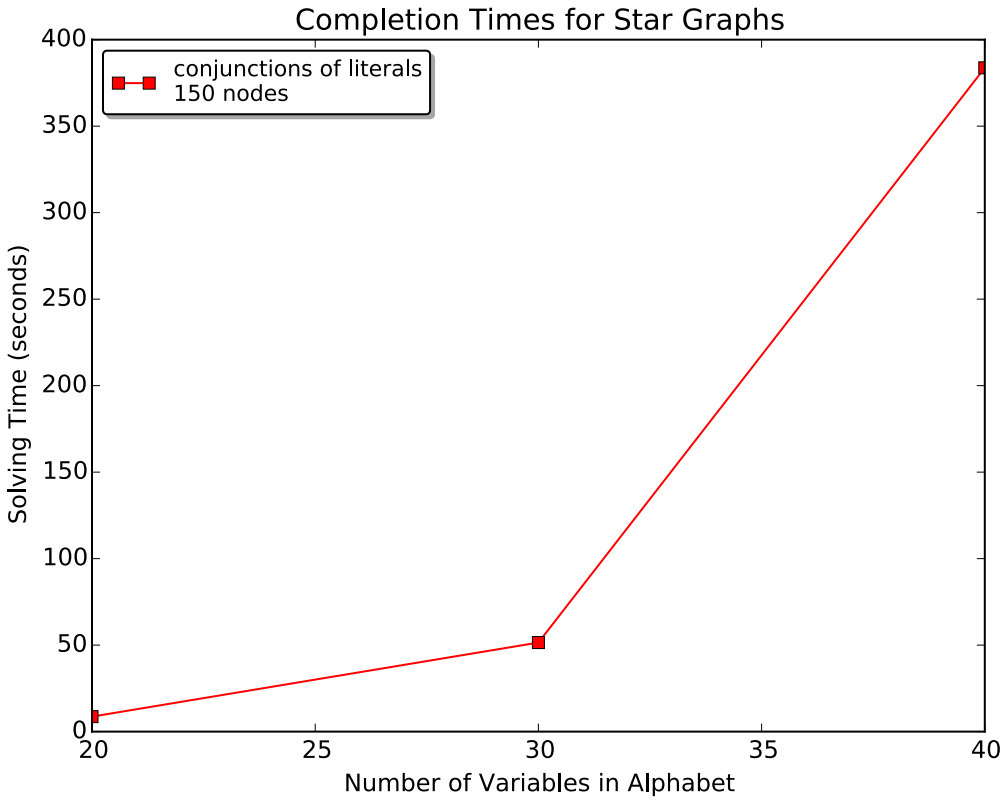
For experiments in section 3.1.3, we do not consider outliers. And, the standard deviation of solving times is not recorded.

Experiment 11

The alphabet contains 25 variables**.** The changing parameter on the horizontal axis is the number of nodes. Nodes range from 100 until 250 with a step size of 50. The problems with 100 and 150 nodes are repeated 25 times. With 200 nodes the problem is repeated 10 times. With 250 nodes, 4 times.



Experiment 12
Each problem involves a graph with 150 nodes. The changing parameter on the horizontal axis is the alphabet size. The alphabet size ranges from 20 variables until 40 variables with a step size of 10. The problem with 20 variables is repeated 25 times, 30 variables is repeated 10 times, and 40 variables is repeated 5 times.



3.2 Hard Satisfiability Formulas

In section 3.2 we consider belief change problems where agents are assigned hard satisfiability formulas. However, each individual agent is not assigned its own hard formula. A single hard formula is divided up into pieces, and each piece is assigned to an agent. For example, in experiments 13-15, the formulas are divided so that each agent is assigned 5 clauses. Thus, if there are 5 nodes in the graph, then the formula has a total of 25 clauses and 5 clauses are assigned to each node. The hard formulas are in conjunctive normal form. Each clause is a disjunction of 3 literals. The hard formulas use all variables

in the alphabet. The source code for generating these hard formulas can be found at: https://toughsat.appspot.com/

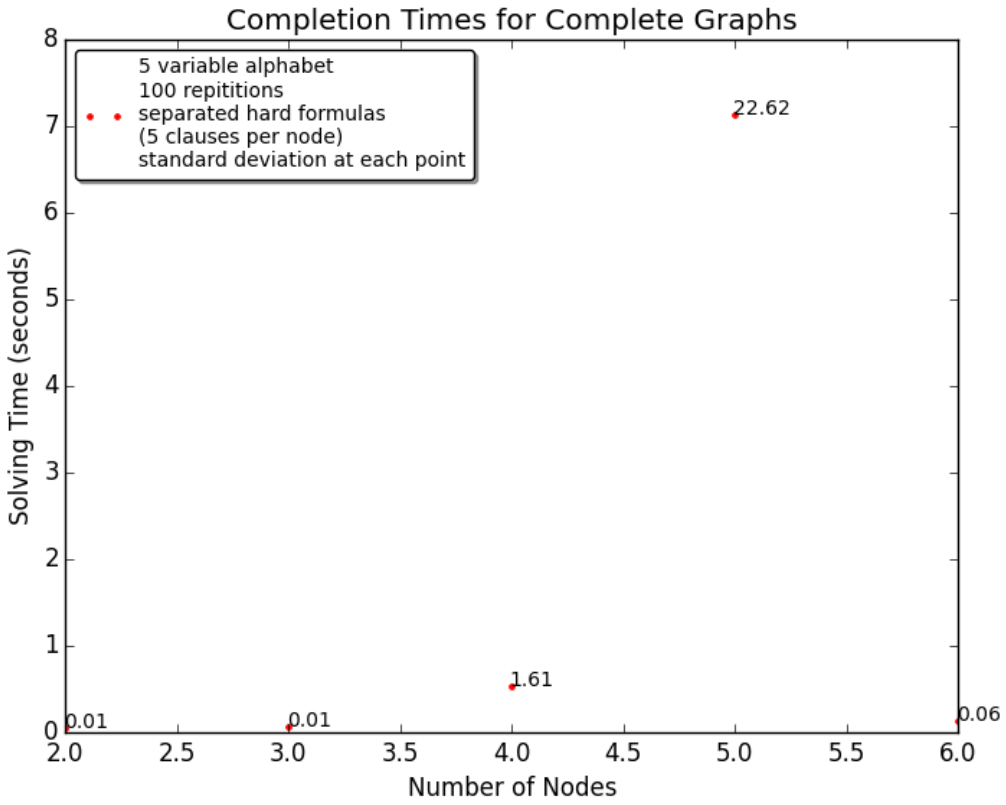
In experiments 13-15, the alphabet contains 5 variables. Each hard formula is made with all 5 variables. The changing parameter is the number of nodes in the graph.

3.2.1 Complete Graphs

The problems in experiment 13 involve agents networked into a complete graph.

Experiment 13

Nodes range from 2 until 6 with a step size of 1.



cut off time for outliers (seconds)	300	1000
number of outliers	13	50
nodes	2-5	6

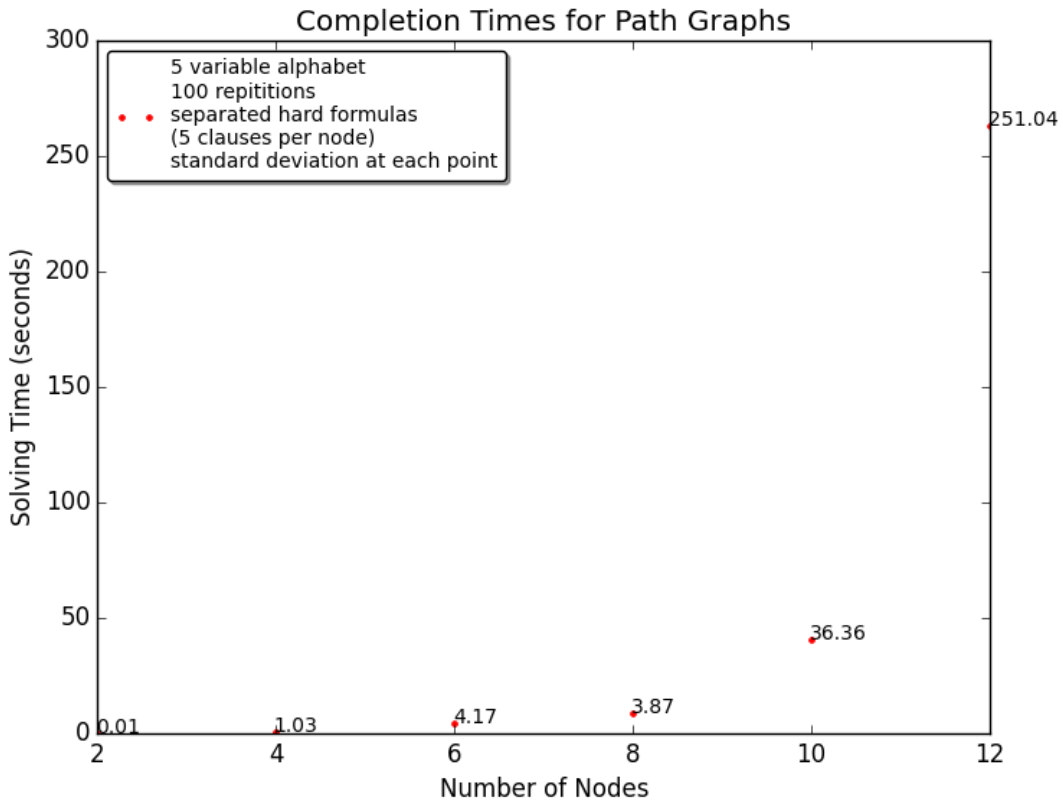
The solving time for half of the problems with 6 nodes went beyond 1000 seconds.

3.2.2 Path Graphs

The problems in experiment 14 are networked into a path graph.

Experiment 14

Nodes range from 2 until 12 with a step size of 2.



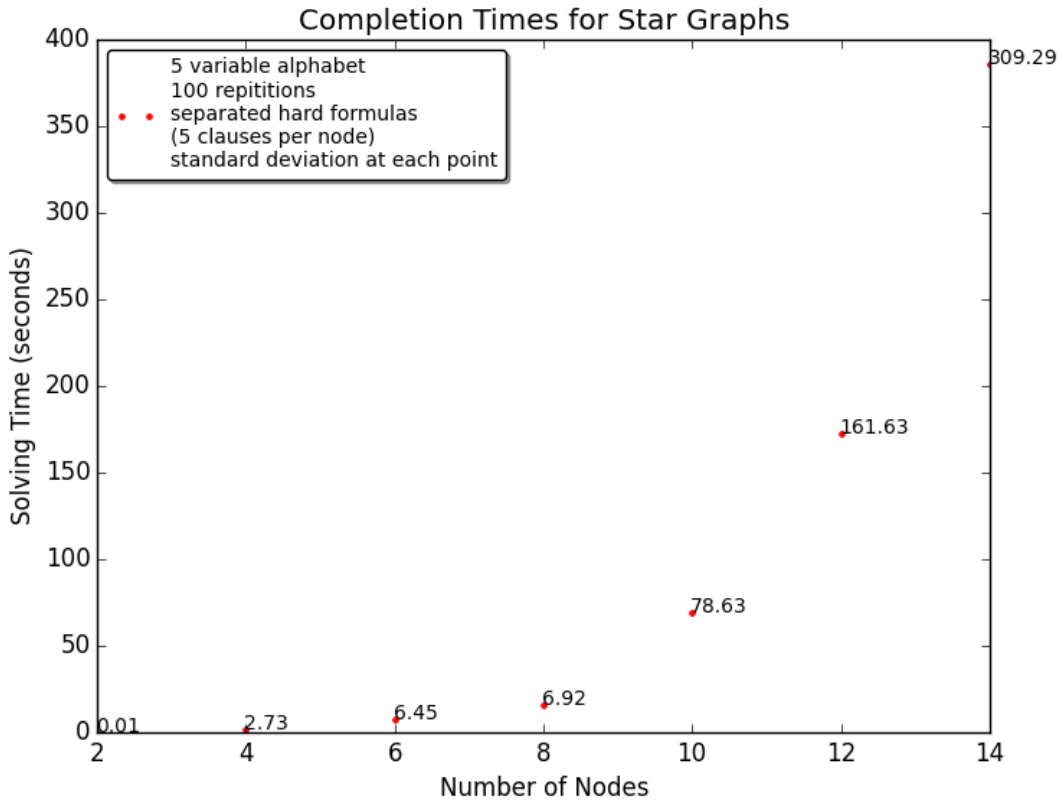
cut off time for outliers (seconds)	200	1000
number of outliers	12	85
nodes	2-10	12

3.2.3 Star Graphs

The problems in experiments 15-16 are networked into a star graph.

Experiment 15

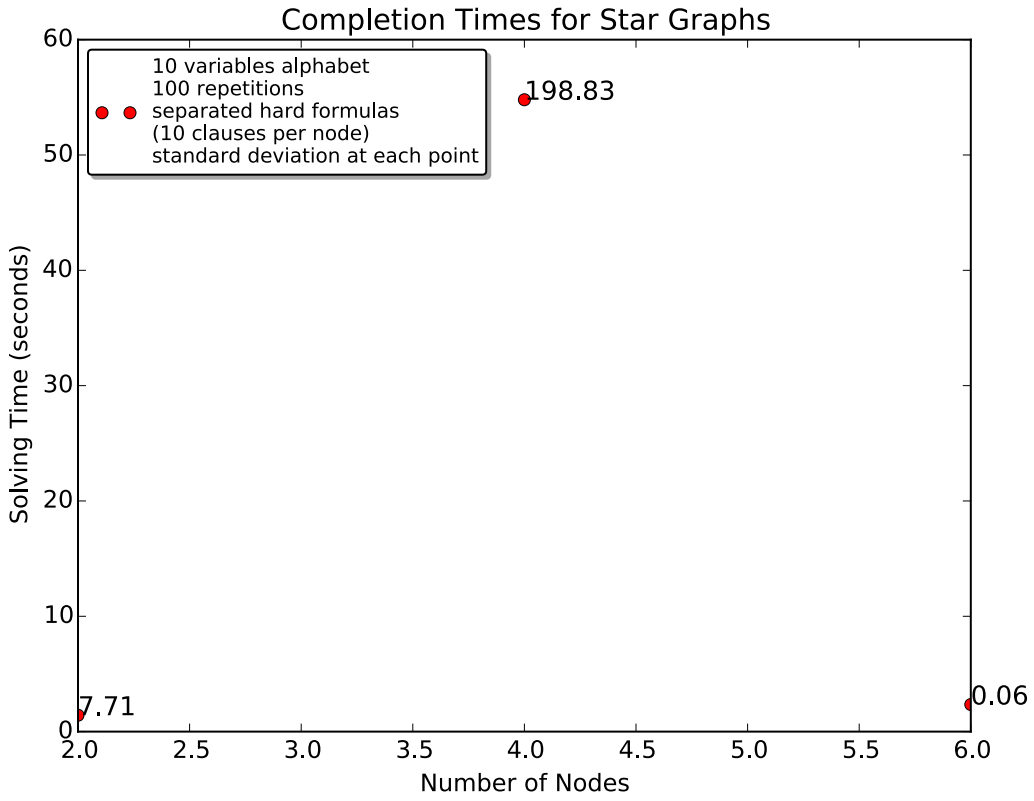
Nodes range from 2 until 14 with a step size of 2.



cut off time for outliers (seconds)	500	1000
number of outliers	49	92
nodes	2-12	14

In experiments 16, each node in the graph is assigned 10 clauses. The alphabet contains 10 variables. Each hard formula is made with all 10 variables. The changing parameter is again the number of nodes in the graph.

Experiment 16
Nodes range from 2 until 6 with a step size of 2.



cut off time for outliers (seconds)	500	1000
number of outliers	20	95
nodes	2-4	6

The solving time for 95 percent of the problems with 6 nodes went beyond 1000 seconds.

4 Improvements in Memory and Time Performance

Since the time that [2] was written, Equibel has been modified. These modifications have resulted in faster solving time and more efficient memory allocation. All the data presented in section 3 of this paper is collected from testing the modified version of Equibel. Here, I briefly discuss some of the changes that have been made in Equibel since [2] was written.

4.1 Gringo Reference Counting Issue

As discussed in [2], Equibel has both an asp component and a python component. Equibel uses the ASP grounder Gringo from the Postdam Answer Set Solving Collection. The author of Equibel discovered a reference counting issue in Gringo. This issue has been solved. Now, Equibel uses less memory.

4.2 Semantic Characterization

The framework for consistency based belief change presented in [1] includes two approaches. One approach is a maximization process based on a syntactic characterization. The other is a minimization process based on a semantic characterization. [2] presents Equibel as an implementation of the syntactic characterization. Since the time that [2] was written, Equibel has been modified so that now it is an implementation of the semantic characterization. In [2], Vicol explains that the asp component of Equibel is made of three logic programs; eq_sets.lp, translate.lp, and transitive.lp. In the modified version of Equibel, translate.lp and transitive.lp are no longer used. The only asp program used is eq_sets.lp. It is used to generate maximal equivalence sets. Since maximal equivalence sets are the duals of minimal change sets, and since preferred models are found from minimal change sets, eq_sets.lp is used to produce the set of preferred models. These preferred models are stored in python as binary representations of integers. From these preferred models, python finds the completion of the graph scenario. This change has resulted in a faster solving time.

4.3 Maximal Equivalence Sets No Longer Stored in Dictionaries

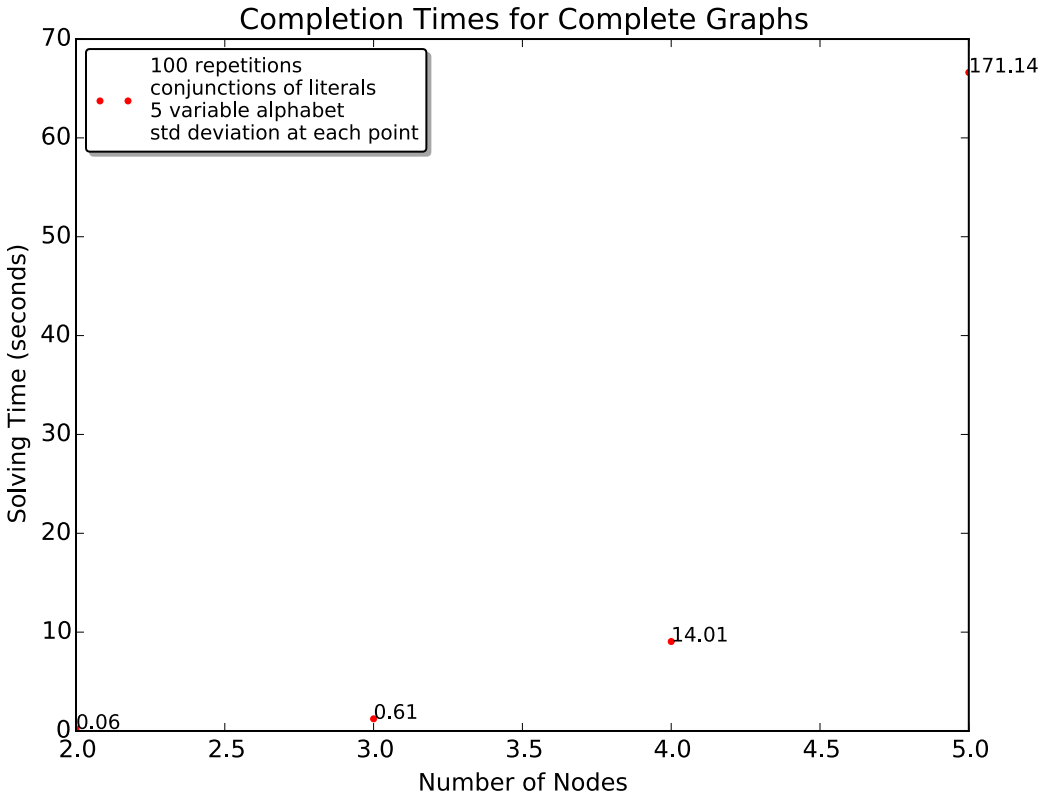
Originally, Equibel stored maximal equivalence sets in pythons dictionary data structure. The modified version of Equibel does not. Instead, maximal equivalence sets are only generated in asp. Since memory no longer needs to be allocated to dictionaries, the modified Equibel uses less memory.

4.4 Demonstrating an Improvement in Solving Time

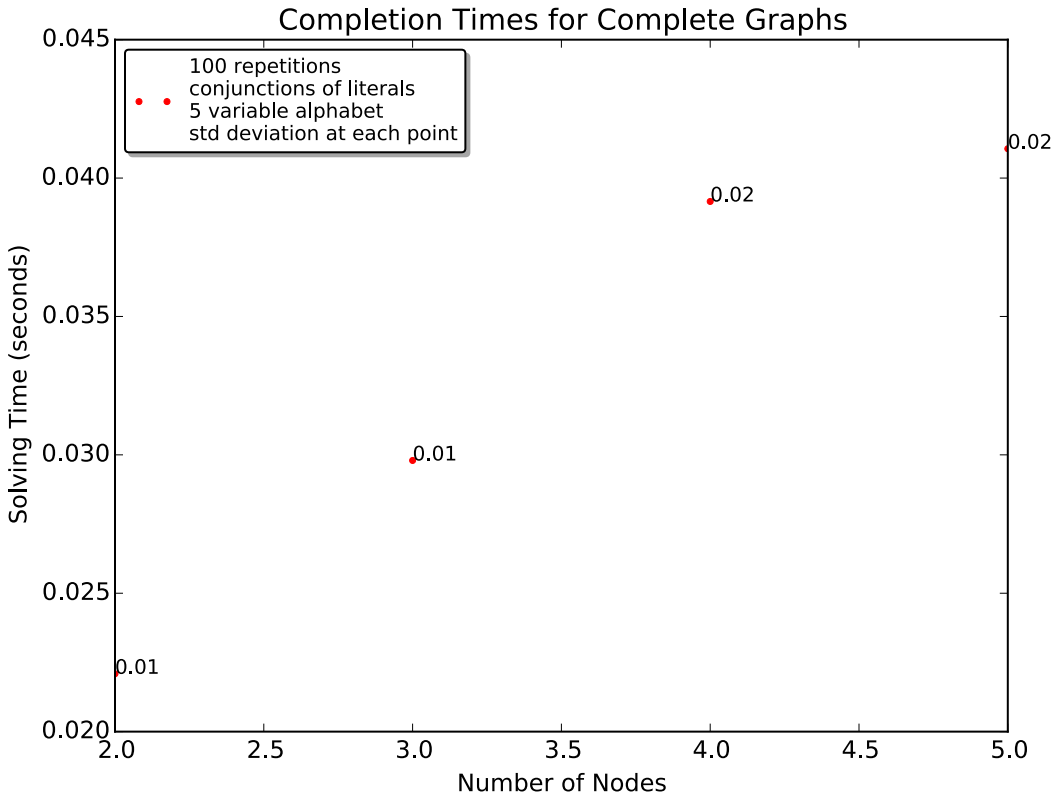
Here I demonstrate that the modified version of Equibel has a faster solving time. I present two experiments. The first (experiment 4.1) was done with the original version of Equibel presented in [2]. The second (experiment 4.2) was done with the modified version of Equibel. Other than being run on different version of Equibel, experiments 4.1 and 4.2 are identical.

In experiments 4.1 and 4.2, agents are networked into a complete graph. The alphabet contains 5 variables. Formula assignments are random conjunctions of literals making up approximately two thirds of the alphabet. The changing parameter on the horizontal axis is the number of nodes, ranging from 2 until 5 with a step size of 1. Problems are repeated 100 times. Outliers are not considered.

Experiment 4.1 (Original Version of Equibel)



Experiment 4.2 (Modified Version of Equibel)



We can see a definite improvement in solving time.

5 Comparing Equibel's Solving Time in Python and ASP

As was discussed in section 4.2, Equibel has an asp component and a python component. Here we compare the time that Equibel spends in each component while solving a particular problem.

As was also discussed in section 4.2, Equibel has been modified since [2] was written. In the modified version of Equibel, eq_sets.lp is the only asp program used in the solving process. Thus, to measure the time spent in asp while solving a particular problem we only need to measure the time required to compute eq_sets.lp for that problem. To measure the time spent solving in python we can measure the time required to do the entire completion process with both components, and subtract the time required to compute eq_sets.lp.

We now consider 2 specific problems, and compare the time that Equibel spends solving in python and asp.

Problem 1
Consider a 10 node path graph with nodes labeled 0 through 9. Assigned to node 0 is the formula ‘p & q & r & s & t’. Assigned to node 9 is the formula ‘~p & ~q & ~r & ~s & ~t’. No other nodes are assigned formulas. I solve this problem and measure the time spent in each component. Even though this problem does not involve a randomly generated formula, each time it is solved there is a small variation in time. I solve the problem 10 times and record the average solving time.

	Python	ASP
Time (seconds)	9.34	2.11
Percentage of Time (%)	82	18

Problem 2
Consider a 7 node complete graph with nodes labeled 0 through 6. We make the following assignments of formulas to nodes:

Node	0	1	2	3	4	5	6
Formula	p & q & r	~s	~q & ~r	p	r	s	~p & ~q”.

We solve this problem and compare the time spent solving in each component. We, again, solve the problem 10 times and record the average solving time.

	Python	ASP
Time (seconds)	241.06	28.76
Percentage of Time (%)	89	11

Judging only from these two problems, between 80 and 90 percent of solving time is spent in python, between 10 and 20 percent is spent in asp.

7 Conclusions

In this paper, I presented data collected from testing the performance of the software system Equibel. Equibel is an implementation of a consistency based multi-agent belief change framework. I tested three types of graph configurations; complete graphs, path graphs, and star graphs. In terms of solving time, the most difficult graph configuration is the complete graph. The easiest is the star graph. With each graph configuration, whether increasing the number of nodes or variables, at some point the solving time steeply increases.

I briefly discussed some modifications and improvements that have been made in Equibel since [2] was written. I discussed how Equibel’s software architecture is made of both a python and an asp component, and investigated the proportion of time Equibel

spends solving in each component. Judging only from the two problems investigated, between 80 and 90 percent of solving time is spent in python, between 10 and 20 percent is spent in asp.

References

1. J.P. Delgrande, J. Lang, and T. Schaub, “Belief Change Based on Global Minimization”, *Proceedings of the International Joint Conference on Artificial Inteligence,* Hyderabad, India, 2007.
2. P. Vicol, J.P. Delgrande, and T. Schaub, “An Implementation of Consistency-Based Multi-Agent Belief Change using ASP”, *Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning,* Lexington, KY, 2015.