Investigating The Zebra Puzzle
and Finite State Methods

Ross McKinley
B.A.(Mod.) Computer Science
Final Year Project  April 2014
Supervisor: Tim Fernando

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Name

Date
Acknowledgements

Many thanks to Tim Fernando for continued guidance, advice and support throughout the course of the project. Thanks to Andrew Butterfield for insight into including interesting findings.
Abstract

This report looks at how Finite State Methods solve Constraint Satisfaction Problems, using the Xerox Finite State Toolkit and a comparable open source finite state toolkit developed by Mans Hulden, foma. The report then compares the performance with traditional Constraint Satisfaction Problem Methods, such as a primitive Backtracking method, and a more advanced Partial Instantiation method.

Specifically, this report looks at if Finite State Methods are a viable alternative for solving Constraint Satisfaction Problems and if not, why not. This is done by using the Zebra Puzzle, and a smaller version, the Mini-Zebra Puzzle, as example puzzles.

This report found that Traditional methods will perform well in a solution-rich environment, whereas Finite State Methods will perform well regardless of number of solutions. This tradeoff is best made on a per-problem basis, on the assumption that intrinsic knowledge is gained *a priori* about a rough number of solutions that exist to the problem. In the case that no prior knowledge exists, it then depends on the number of solutions required. Finite State Methods perform best when all solutions are desired, and Traditional Methods perform well when merely the first solution is required.

This report also found that Finite State Methods could be greatly optimised by developing entailments to see which constraints are necessary to gain a solution, before commencement of evaluation.
# Table of Contents

1. Introduction .................................................................1  
   1.1 Aims .................................................................1  
   1.2 Motivation ............................................................1  
   1.3 Reader’s Guide .........................................................2  

2. The Zebra Puzzle ..........................................................3  
   2.1 Background ...........................................................3  
   2.2 Previous Work .........................................................4  
   2.3 Solution .................................................................5  

3. The Mini-Zebra Puzzle ....................................................7  
   3.1 Background ...........................................................7  
   3.2 Previous Work .........................................................8  
   3.3 Solution .................................................................8  

4. Investigation ...............................................................9  
   4.1 Methods Used ........................................................9  
      4.1.1 Backtracking ......................................................10  
      4.1.2 Partial Instantiation ..........................................10  
      4.1.3 Finite State Methods .........................................10  
   4.2 Results .................................................................11  
      4.2.1 Unexpected Results ...........................................13  

5. Discussion ........................................................................16  
   5.1 Conclusions ............................................................20  
   5.2 Future Work .............................................................21  

6. Supporting Materials ......................................................23  
   6.1 Code ......................................................................23  
      6.1.1 Mini Zebra Puzzle .............................................23  
      6.1.2 Full Zebra Puzzle .............................................23  

7. Appendices ......................................................................24  
   7.1 Comparison Graphs ....................................................24  
   7.2 Mini-Zebra xfst / foma Code .......................................25  
   7.3 Mini-Zebra Backtracking Code .....................................26  
   7.4 Mini-Zebra Partial Instantiation Code ............................28  
   7.5 Zebra Puzzle xfst / foma Code ....................................30  
   7.6 Zebra Puzzle Backtracking Code ..................................32  
   7.7 Zebra Puzzle Partial Instantiation Code ..........................34  

8. References .......................................................................36
1. Introduction

1.1 Aims
The Aims of this project are to evaluate the viability of Finite State Methods (the method of building finite state machines or transducers to come to an agreeable solution for a particular problem) as a method of solving Constraint Satisfaction Problems. This project should be able to draw a conclusion on if Finite State Methods can be used to solve Constraint Satisfaction Problems, and if not, why not. Furthermore, the project should be able to detail if there are certain kinds of Constraint Satisfaction Problems which are more, or less, suited to Finite State Methods.

This report is not concerned with the most efficient methods of solving the Zebra Puzzle [1], but rather takes a simplistic comparative view of Finite State Methods contrasted with Traditional Methods for solving Constraint Satisfaction Problems. To achieve a comprehensive evaluation, the report looks at how Finite State Methods solve the Zebra Puzzle, and compares it with Traditional Methods such as a simple Generate and Test, and a more advanced Partial Instantiation. This report looks at the aspects of the Puzzle itself, as well as the solution, and the ramifications these have on using Finite State Methods to solve the puzzle. It is taken into account throughout the report that two languages which behave differently are being used during the comparison of different methods. The effect on the results is expected to be negated by the degree of difference between results.

1.2 Motivation
Finite State Methods offer an alternative method of solving problems, particularly Constraint Satisfaction Problems, it is therefore worthwhile to investigate what type of Constraint Sat-
isfaction Problems Finite State Methods are best suited to, or to discover which kind of problems they are not suited to. It is hoped that by investigating Finite State Methods, some guidance can be given as to whether they are viable alternatives to solve certain kinds of Constraint Satisfaction Problems.

1.3 Reader’s Guide
The remainder of this report is spread over seven chapters. The report covers the background of each method and details on the types of puzzles used, an implementation, evaluation and conclusion of the project, and finally a discussionary chapter.

The background contains two chapters, one chapter focuses on the Zebra Puzzle (Chapter 2) and the other on the Mini-Zebra Puzzle (Chapter 3). The different Zebra Puzzles are described, and the rules given. Each chapter then shows some of the work carried out investigating the Zebra Puzzle generally. Current papers and approaches are highlighted, and discussed.

Chapter 4 moves on to detail the overall approach of the investigation. The methods are explained and then evaluated. Results for each of the methods are shown.

Chapter 5 contains a discussion on evaluating Finite State Methods as a viable way to solve Constraint Satisfaction Problems. This chapter also contains details on further work which could be undertaken to improve the results.

Chapter 6 mentions supporting materials created during the course of this investigation.

Chapter 7 contains all the appendices.

Chapter 8 is a bibliography of works read and referred to throughout the course of this report.
2. The Zebra Puzzle

2.1 Background

The Zebra Puzzle, sometimes referred to as Einsteins Puzzle[2] or simply the Zebra Problem[3], is an often explored example of a Constraint Satisfaction Problem. The premise is that there are five houses in a row, and there are fifteen rules, or constraints, about the aforementioned houses. In order to solve the puzzle, one must navigate the available knowledge to arrive at a satisfactory conclusion, answering the question “Who owns the Zebra?”.

Due to the ubiquity of the puzzle, it appears as though nearly every reference to it uses a different set of variables, apparently to provide social relevance to people in their vernacular. This report uses a homogenised version of the constraints which appeared in the Life International Magazine, 1962 [3], which are reportedly the earliest published rules. By using the same rule-set, comparisons are more easily drawn between differing programs.

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in the house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
15. The Norwegian lives next to the blue house.
As it stands, the puzzle is fully constrained; no more constraints need be made in order to arrive at a solution. However, there are additional constraints which make the puzzle over-constrained.

Note: “It must be added that each of the five houses is painted a different colour, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarettes”

In the interest of performing a robust evaluation of Finite State Methods generally, this report has modified known good solutions to include these often omitted constraints. The reasoning in including these additional constraints is that their superfluous nature cannot be know a priori.

2.2 Previous Work

There are many versions of the Zebra Puzzle. Most versions change the names of the variables to be more suited to the intended audience. This report looks closely at Lauri Karttunen’s version[2], which is written for the Xerox Finite State Toolkit[4]. It also runs in foma, which is reported to have comparable speeds for Constraint Satisfaction Problems[5]. This version seems to be the accepted best solution for Finite State Methods, and therefore warrants its inclusion in this report. This version, however, does not include the uniqueness constraints, so it has been modified to contain them.

Backtracking is a common solution to search problems such as Constraint Satisfaction, this report investigates a version by Jonathan Mohr[6], which he himself highlights as being an un-optimised approach[7]. This example of Backtracking could be improved with the aide of heuristics; however, to represent only
optimised alternatives to Finite State Methods would be an unfair comparison.

Jonathan Mohr, instead suggests a Partial Instantiation version as an alternative to Backtracking[8]. This version has significantly better performance, and is still general enough to apply to other Constraint Satisfaction Problems.

Others have come up with much more advanced methods of solving the Zebra Puzzle, one such method[1] claims to have found an optimal solution, which beats previous methods such as Backtracking, Minimum Remaining Values, Forward Chaining, and Minimum Conflicts. Other methods are more general, but still claim better performance over simplistic methods such as Backtracking[9]. Marc Van Dongen has a Constraint Propagation method which looks to be highly optimised[10].

A group in Cork claim to have found good results regarding the Zebra Puzzle[11] by utilising what they call Case-Based Reasoning for Constraint Programming. This is similar to a prototype development process, where the system refines over time. All of these alternative could be explored in the scope of a more comprehensive evaluation of methods to solve the Zebra Puzzle. For the sake of this report, a limited number of alternatives can be investigated thoroughly.

2.3 Solution

One can see from Figure 1 the solution to the problem is unique. The Japanese person keeps Zebra, and the Norwegian drinks Water. Interestingly, as Lauri Karttunen has highlighted [2], in encoding some of the rules we make assumptions about the direction the viewer is facing.

The green house is just to the left of the white one.
The encoding of this could depend on if you are facing towards or away from the house. This decision changes the outcome of the houses, as it creates an ambiguity. In that case you might encode both perspectives, and come out with two logical solutions, each one satisfying one perspective.

This observation does little more than to offer insight to the truths surrounding Constraint Satisfaction Problems, namely that if you reduce the rigidity of the constraints, it results in more solutions being satisfactory. Further to this, after a certain point, additional constraints will have no effect other than to complicate the computation.

In the case of the Zebra Puzzle, the initial search space is really big. There are $5^{25}$ possibilities, or as many arcs as $298,023,223,876,953,000$. This represents a significant solution search space. However, most of these are not evaluated, as they break the constraints. By including the uniqueness constraints, this search space can be vastly reduced. Now, there will only be $(5!)^5$ possibilities, or 4,424 arcs in the graph. Although this reduces the search space, the cost of performing this reduction is significant, as seen in the results for Finite State Methods.
3. The Mini-Zebra Puzzle

3.1 Background
To gain more robust information, this report looks at both the full Zebra Puzzle and a mini-Zebra Puzzle [12]. The Mini-Zebra Puzzle has a reduced problem domain. Consequently, it is an example of a more simplistic Constraint Satisfaction Problem. The rules are given below.

1. The Spaniard owns a dog.
2. The Ukrainian drinks tea.
3. The man in the third house drinks milk.
4. The Norwegian lives next to the tea drinker.
5. The juice drinker owns a fox.
6. The fox is next door to the dog.

In this version of the Zebra Puzzle one still must answer the question, “Who owns the Zebra?”.

As it stands, the puzzle is fully constrained; no more constraints need be made in order to arrive at a solution. There are additional constraints, however, which make the puzzle over-constrained, much like the full Zebra Puzzle.

Note: “It must be added that each of the three houses’ inhabitants are of different national extractions, own different pets, and drink different beverages.”

The above uniqueness constraints are often left out, since the puzzle is already fully constrained. However, this can only be known after solving the puzzle, or with knowledge a priori. Therefore, in the interest of testing the feasibility of Finite State Methods, the uniqueness constraints must also be satisfied.
3.2 Previous Work
At the time of writing, there are very little resources available regarding a smaller version of the Zebra Puzzle. The version this report uses is that of Henri Kautz[12], which appears to be a contrived example of Constraint Satisfaction Problems.

3.3 Solution
The Mini-Zebra Puzzle is just a smaller version of the Zebra Puzzle[12], and the solution is much the same. In Figure 2, one can see that the Ukrainian person keeps Zebra. In this particular variation on the Zebra Puzzle, there is only one unknown variable, the owner of the Zebra. All other variables are explicitly mentioned in the rules. Again, this vastly reduces the search space for the puzzle, so one would expect much faster results than the more complex full Zebra Puzzle.

![Figure 2. Solution of Mini-Zebra Puzzle (One house per row)](image)

With only three houses, with three variables each, it is clear to see that this problem is significantly easier to solve. The inclusion of this particular puzzle in this report is to give an idea of performance on both large and small problems. Efficiency in a small problem which could be executed millions of times, has good reason to be considered important, too.
4. Investigation

This Report looked at three differing approaches to two problems; a big logic puzzle, and a more manageable puzzle. By investigating the consequences of problem size on each of the methods, a holistic view into the performance of each approach could be achieved.

Using the time function on the command-line, rough timings were gathered for each of the different methods, for both the Mini-Zebra Puzzle and the Zebra Puzzle. Although this is a crude benchmarking method, the problems and the methods being evaluated are sufficiently complex that a large discrepancy in results is expected. Rather than evaluate performance at a fine grain, which is not necessarily well suited to generalisation, this coarse timing method will allow a particular method to exceed in the case that it is obviously better suited to the task at hand. This means that the findings in the report are restricted to generalisations, rather than explicitly stating that one method is better or worse than another at one specific puzzle. The aim here is to explore Finite State Methods generally.

4.1 Methods Used

This report uses a small variety of Traditional Methods, as well as a known good example of Finite State Methods for comparison. It is worth noting that both Traditional Methods and Finite State Methods come with their own bag of tricks with regards optimisation techniques and even tools for optimisation. In both cases not all optimisations which are possible are implemented.

Extensive work has been carried out in Finite State Methods[3], as well as Traditional Methods at solving the Zebra Puzzle.[1]
4.1.1 Backtracking
Backtracking is a well known Traditional Method to tackle Constraint Satisfaction Problems[13]. It serves as a refinement of the Brute Force method, and offers greater performance with the use of heuristics to eliminate paths as early as possible[14]. The backtracking method used in this report closely resembles that of Jonathan Mohr[6].

4.1.2 Partial Instantiation
Some of the biggest problems with Backtracking are that you must commit to a particular assumption before finding out if it is correct or not. There exist efforts to mitigate this problem[15], and alternative methods exist, such as using a breadth-first search[16].

Partial Instantiation is one such alternative[17], and we evaluate its effectiveness in this report. Partial Instantiation avoids many of the issues relating to search space explosion by binding only the variables it already knows, and keeping the unknown variables unfilled as long as possible. This way, Partial Instantiation can focus on searching for the unknowns, rather than all the variables already known to be valid, until a total instantiation can be reached.[18] The biggest benefit here is that if one instantiation violates one of the constraints, then the whole instantiation is inconsistent, and those can be ruled out.

4.1.3 Finite State Methods
Finite State Methods offer an alternative method of evaluating problems[19]. Finite State Methods encompass using finite state machines and finite state transducers to build a network of solutions. These solutions come in the form of finite state machines, and thus are easily comparable with strings of solutions, since each node in the network can be seen as being a token (or variable). Lauri Karttunen has produced a Finite State Methods based approach to tackling the problems in this pa-
per[2], however, the method used leaves out the often forgotten constrains of uniqueness. These must be included since one cannot know for certain that the problem is over specified until the problem is evaluated, or knowledge is known a priori.

It is hoped that by the end of this report, more knowledge as to the suitability of Finite State Methods to be applied generally to Constraint Satisfaction Problems, will be made available.

4.2 Results

The results demonstrated that for small search spaces, Finite State Methods proved to be a viable contender. The less complex nature of the problem made little impact to each of the Traditional Methods used, as can be seen in Figure 3, however Finite State Method delivered all solutions much faster. This is a positive result for simple problems, but may only represent...
the size of the puzzle at hand, rather than the effectiveness of the method.

The benefits and caveats of using Finite State Methods became apparent when looking at the more complex full Zebra Problem, as can be seen in Figure 4. With 25 variables in play, excluding the positioning of each house, a complexity explosion was encountered when uniqueness constraints were added. This is because the Finite State Toolkits create a huge state machine (over 2GB in size) for all possible combinations of the variables, while applying the constraints. This is misleading, because each constraint should be reducing the search space of the problem, so you would not expect that to have an adverse affect on compilation time. In fact, the time taken is due to the reduction in search space. This is counter-intuitive, but happens because of the way the Finite State Methods work. They apply each constraint one by one, until there are none left. In this way, the entire search graph of uniqueness constrained

![Figure 4. Calculating Zebra Puzzle including Uniqueness Constraints (lower is better, Logarithmic Scale)](image-url)
variables is built, and then the other constraints are applied against it until there are no more constraints left to apply. At that point, all that remains are the valid solutions to the problem at hand. If a variable can be any number of items, the finite state machine just says it can be one of these, when including the uniqueness constraints, the finite state machine has precise data on what can and cannot be chosen as the variable. Due to this, it creates a graph of all possibilities (as seen in the case of the Mini-Zebra puzzle in Figure 9), and this is where the computational complexity enters into the method generally.

The massive benefit of this approach is that once the graph is compiled, the testing phase is straightforward, and the cost is very little indeed, since all paths which lead to the solution are valid. This valid solution graph could then be handed off to another program for further testing, or to find an optimal path via some other method.

It so happens that the Zebra Puzzle is fully constrained and results in only one solution, but if we change one of the rules to allow for ambiguity, we can very easily find all possible paths with Finite State Methods.

Comparing this with Traditional Methods, it is clear to see that Finite State Methods offer a better solution if the task at hand is to provide all possible solutions to the puzzle or Constraint Satisfaction Problem, whereas they offer a significantly costly alternative if the goal is to simply find the first path which satisfies the constraints.

4.2.1 Unexpected Results

During this investigation, it was found that Finite State Methods have an interesting relationship with memory requirements and constraint complexity. From the graph in Figure 5, it can be
seen that the memory requirements increase for each additional uniqueness constraint, reaching RAM requirements of 65 GB for a fully unique solution to the Zebra Puzzle. This is contrary to what occurs in both Backtracking and Partial Instantiation (whose lines are so close together on the graph that they are nearly as one), as they only ever have a fixed amount of items in memory at a given time, and evaluate each constraint multiple times as they search for a solution. This allows both Backtracking and Partial Instantiation to remain at 3.9 MB and 4.1 MB respectively. This unexpected result confirms that Finite State Methods take a holistic approach, by evaluating the entire puzzle at once, and by applying constraints one by one rather than instantiating the variables one by one.

Figure 5. System Memory for Each Method (Logarithmic Scale)
It is possible that this memory explosion could be countered with some optimisation techniques not explored as part of this paper. By researching and developing entailments to check if constraints are necessary, it may be possible to reduce this explosion effect, or even eliminate it entirely.

We can see in Figure 6, that the memory requirements are significantly less when the uniqueness constraints are not included. In this case, the RAM requirements remained close to the same values as seen in the earlier stages of including low numbers of uniqueness constraints. This is a strong case for developing entailments to check if constraints are necessary, or if certain constraints can be avoided completely, since it has such a negative impact on performance.
5. Discussion

This report has looked at two discrete cases where Finite State Methods can be applied. It is important to note that the report is not concerned with the most efficient methods of solving the Zebra Puzzle specifically, but rather takes a comparative view of Finite State Methods contrasted with Traditional Methods for solving Constraint Satisfaction Problems.

The times reported in Figure 3 demonstrate how Finite State Methods are very quick at solving the Mini-Zebra Puzzle, particularly in comparison with the Traditional Methods. In this test, the uniqueness constraints were included. It turns out that this provides a significant computational challenge on larger problems, however, this is not a concern for the Mini-Zebra Problem. Without the inclusion of the uniqueness constraints the problem proves to have a reduced complexity, and solutions are quicker across all methods. For the Mini-Zebra

![Figure 7. Calculating Mini-Zebra Puzzle Excluding Uniqueness Constraints (lower is better)](image)
Puzzle, this doesn’t have much of an impact, as seen in Figure 7, since at a millisecond level the differences are hard to see.

The most interesting of results come in the form of the Zebra Puzzle results. In Figure 4 it is clear to see that some form of complexity explosion occurred. Figure 4 compares the times in calculating all possible solutions to the Zebra Puzzle including Uniqueness Constraints. One can see that the Finite State Methods (marked as FSM) took approximately 20 minutes to arrive at all the solutions, whereas the Traditional Methods had more varied results: Backtracking took over 4 hours to complete, and Partial Instantiation took only 1 minute. This is a large discrepancy, and highlights that the methods used vary in their efficiencies and optimisations. It is clear to see that Partial Instantiation offers an optimised alternative over the primitive Backtracking Method. Backtracking could be improved with the use of some heuristics, however.

The most interesting result is in comparing the full Zebra Puzzle with and without Uniqueness Constraints as seen in

![Figure 8. Calculating Zebra Puzzle excluding Uniqueness Constraints (lower is better)](image)
Figure 8. It is when this happens that we can see the jump in complexity.

The Difference between the methods here, is that Traditional Methods take an iterative approach. That is to say that they are constantly checking to see if the next node, or the next variable, satisfies the constraints. Finite State Methods take an alternative approach. Finite State Methods take a holistic view at the problem, and apply each constraint once. Each constraint acts as its own finite state machine, and thus, is applied once on compilation with the other constraints (ranging in size from being miniature to representing a large network of uniquely ordered variables as seen in Figure 9). Interestingly, when adding uniqueness constraints, the size of the finite state machine which represents it has a relationship with the number of variables and the number of possibilities available to them. In the case of the Zebra Puzzle, this works out at being the number of values per variable to the power of the number of variables. Figure 9 shows the number of arcs on the graph with uniqueness constraints applied. Table 1 shows the constraint to size relationship, here it is easy to see that by having many free variables, the search space is greatly increased.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zebra</td>
<td></td>
</tr>
<tr>
<td>without uniqueness (values)</td>
<td>298,023,224,000,000,000</td>
</tr>
<tr>
<td>with uniqueness (values!)</td>
<td>4,424</td>
</tr>
<tr>
<td>Mini-Zebra</td>
<td></td>
</tr>
<tr>
<td>without uniqueness (values)</td>
<td>19,683</td>
</tr>
<tr>
<td>with uniqueness (values!)</td>
<td>216</td>
</tr>
</tbody>
</table>

Once all constraints are applied, all that is left are the solutions. In Traditional Methods this is not the case; normally, the
method looks individually at the next node in a search graph which could contain any number of invalid options.

Figure 9. Mini-Zebra Network of Uniquely Constrained Variables (216 arcs between 133 nodes)
Finite State Methods eliminate the need to search over a partially correct graph in favour of a graph which only contains valid paths, at the cost of both compilation time and RAM requirements during the process of compilation. Figures 5 and 6 show the RAM requirements for each method, both with and without Uniqueness Constraints. It is clear that should a method of determining entailments of the constraints (uniqueness or otherwise) be developed, it would result in a vastly superior improvement over the Finite State Method we see in this report.

Other interesting points to mention involve the case where there are no solutions. For Finite State Methods, having zero solutions is much the same as having millions of solutions, since all solutions are obtained at once. For Traditional Methods, this is not the case. Traditional methods must evaluate all paths and see if they satisfy the constraints. This leads to bad performance if you want to get all possible solutions. There is a trade off to be had here, consistent performance regardless of the number of solutions (Finite State Methods), or a degradation of performance based on an inverse of the number of solutions (Traditional Methods).

5.1 Conclusions
Traditional methods will perform well in a solution-rich environment, whereas Finite State Methods will perform equally well regardless of number of solutions. This tradeoff is best made on a per-problem basis, on the assumption that intrinsic knowledge is gained a priori about a rough number of solutions that exist to the problem. In the case that no prior knowledge exists, it then depends on the number of solutions required. Finite State Methods perform best when all solutions are desired, and Traditional Methods perform well when merely the first solution is required.
This report saw that Finite State Methods are well suited to problems where you need to retrieve all possible solutions. Finite State Methods are also suitable when all that is wanted is to compile all possible solutions and then hand them off to another program. The results themselves could become input for some program to find the optimal solution, rather than any solution. This would make querying a hypothetical knowledge-base relatively inexpensive, since there would be no concern on whether a particular path no longer satisfies the constraints.

Traditional methods are preferable for problems where it is known a priori that the search space is solution-rich, and only one solution is required. This can result in vast speed ups, as the method needs not evaluate all possible paths, and rather, merely needs to find one path which is satisfactory. The benefits here are particularly noticeable in larger search spaces. The worst case scenario is that the method will have to search all possible paths, which can be mitigated by using some heuristics.

Regarding logic problems, a solution remains a solution regardless of how it was solved; however, consider the scenario when certain paths become more or less expensive. Should weights or costs be assigned to each of the paths, Finite State Methods will be able to pass all valid solutions to another evaluator to determine the best solution to the given task. It is in this scenario that Finite State Methods are the preferred option.

5.2 Future Work

The testing methods used in this report are sub-optimal. More meaningful results could be achieved by utilising a timing framework. As it stands, this report uses crude methods to get ball park approximate timings for each method.
The nature of using different languages for different approaches does not lend itself well to fair testing. Some mitigations for this may be to run the tests many times per method, so that the set-up of each languages environment is alleviated. It is thought that this is not a large concern for the tests conducted as part of this report, since the calculations for the Zebra Puzzle were substantial enough to have language/environment costs not impact so much on the overall time taken. In this case, the differences in data gathered are substantial enough to be able to determine that each method is good at a particular task, without getting into details at a fine grain on which language/method combination is preferable.

This report focuses on only two “tests”; namely the Mini-Zebra Puzzle and the Zebra Puzzle. There are many other well known logic puzzles available which would be well worth investigating to see if the results still hold. Other puzzles which have solutions readily available in Finite State Methods and Traditional Methods, which could be explored as part of a more comprehensive investigation include Map Colouring[20] and the 8-Queens Problem[21], which could then be generalised to the n-Queens Problem[22]. This would demonstrate if the state explosion is truly at fault for the degradation in speed, or if that was pure happenstance.

The report experienced a “state explosion” while evaluating the uniqueness constraints for between about 20 variables and 25. It may hold value to investigate the tipping point for this explosion in more depth. Developing entailments to determine which constraints to include in evaluation, to avoid encountering this issue, would hold significant merit.
6. Supporting Materials

6.1 Code

All code written as part of this report is available in the appendices; electronic versions are available online and should accompany this document. In total, six programs were written and modified. In the interest of clarity, all versions were altered to fit the constraint set which appeared in Life International Magazine[3], to make results more easily comparable. In the case of the Mini Zebra Puzzle, a smaller homogenised set of variables was produced, which allowed comparison with the larger full Zebra Problem.

6.1.1 Mini Zebra Puzzle

I use a modified xfst model originally designed by Lauri Karttunen[23]. This program was altered to accommodate the mini-zebra puzzle by Henri Kautz[12].

I use a modified Backtracking prolog code and a modified Partial Instantiation prolog code, both based off Jonathan Mohr's solution[8].

6.1.2 Full Zebra Puzzle

The prolog code is based off Jonathan Mohr's solution for the full Zebra Puzzle[8]. I use a Backtracking prolog code and a Partial Instantiation prolog code. I use a modified xfst model originally designed by Lauri Karttunen[23] which was changed to include uniqueness constraints on each variable.
7. Appendices

7.1 Comparison Graphs

Figure 10. Calculating Mini-Zebra Puzzle
Uniqueness Constraints difference (lower is better)

Figure 11. Calculating Zebra Puzzle (Logarithmic Scale)
Uniqueness Constraints difference (lower is better)
7.2 Mini-Zebra xfst / foma Code

# Adapted by Ross McKinley (mckinlrw@tcd.ie) to include Mini-Zebra Constraints and Uniqueness Constraints
# Trinity College Dublin, Ireland

# Original Mini-Zebra Puzzle by
# Henry Kautz (kautz@cs.rochester.edu)
# University of Rochester, Rochester, NY 14627.

# Original full zebra puzzle solution by
# Lauri Karttunen <laurik@stanford.edu>
# Centre for the Study of Language and Information
# 210 Panama St, Stanford, CA 94305, USA
# Available at: http://www.stanford.edu/~laurik/fsmbook/examples/Einstein%27sPuzzle.html

define Nationality [Spaniard | Ukrainian | Norwegian];
define Drink [Milk | Tea | Juice];
define Pet [Fox | Dog | Zebra];
# What are the variables?
define House [Nationality Drink Pet];
# What does a house look like?

# Here we have the uniqueness constraint. This says all of the variables are shuffled.
# It turns out that doing a shuffle is more efficient than
# a full lookahead/behind for each variable, and then combining them
#define C0 [Spaniard <> Ukrainian <> Norwegian
  <> Milk <> Tea <> Juice
  <> Fox <> Dog <> Zebra ];

define C1 ![Spaniard ~Pet Dog];
# 1. The Spaniard own a dog.
define C2 ![Ukrainian Tea];
# 2. The Ukrainian drinks tea.
define C3 ![House^2 ? ~Drink Milk];
# 3. The man in the third house drinks milk.
define C4 ![Norwegian ? ~Drink Tea | Tea ~Nationality Norwegian];
# 4. The Norwegian lives next to the tea drinker.
define C5 ![Juice Fox];
# 5. The juice drinker owns a fox.
define C6 ![Fox ~Pet Dog | Dog ? ~Pet Fox];
# 6. The fox is next door to the dog.
define Solution [House^3 & C0 & C1 & C2 & C3 & C4 & C5 & C6 ];

regex Solution;
7.3 Mini-Zebra Backtracking Code

%% Adapted by Ross McKinley (mckinlimw@tcd.ie) to include Mini-Zebra Constraints and Uniqueness Constraints
%% Trinity College Dublin, Ireland

%% Original Mini-Zebra Puzzle by
%% Henry Kautz (kautz@cs.rochester.edu)
%% University of Rochester, Rochester, NY 14627.

% Original full zebra puzzle solution by
% Jonathan Mohr (mohrj@augustana.ab.ca)
% Augustana University College, Camrose, AB, Canada T4V 2R3
% Available at: http://www.augustana.ca/~mohrj/courses/2000.fall/csc370/lecture_notes/prolog_examples/zebra_not.pro

solve(Who, [N1, D1, P1]) :-
% There are five houses.
% All house owners are of different nationalities.
N = [spanish, ukrainian, norwegian],
permutation(N, N1),
%write(N1),nl,
% They all have different pets.
P = [fox, dog, zebra],
permutation(P, P1),
%write(P1),nl,
% They all drink different drinks.
D = [milk, tea, juice],
permutation(D, D1),
%write(D1), nl,
% The Spaniard owns a dog.
samepos(spanish, N1, dog, P1),
% The ukrainian drinks tea.
samepos(ukrainian, N1, tea, D1),
% The man in the third house drinks milk.
D1 = [_, _, milk],
% The Norwegian lives next to the tea drinker.
next_to(norwegian, N1, tea, D1),
% The juice drinker owns a fox.
samepos(juice, D1, fox, P1),
% The fox is next door to the dog.
next_to(fox, P1, dog, P1),
% Who owns the zebra?
samepos(zebra, P1, Who, N1).

permutation(L, [H | T]) :-
append(U, [H | V], L),
append(U, V, W),
permutation(W, T).
permutation([], []).

samepos(A, [A|_], B, [B|_]).
samepos(A, [T1|_], B, [T2|_]) :-
samepos(A, T1, B, T2).

inorder(L1, L2, [L1, L2|_]).
inorder(L1, L2, [_|Rest|_]) :-
inorder(L1, L2, Rest).

next_to(A, [A|_], B, [_,B|_]).
next_to(A, [_,A|_], B, [B|_]).
next_to(A, [T1|_], B, [T2|_]) :-
next_to(A, T1, B, T2).

:- time((
solve(Who, S), maplist(writeln,S), nl, write(Who), nl, nl, fail
; write(‘No more solutions.’)))).

%% this times how long it takes to get ALL possible solutions.
%% even though the Zebra Puzzle has a unique solution, that can’t
%% be known before computation begins.
7.4 Mini-Zebra Partial Instantiation Code

%%% Adapted by Ross McKinley (mckinlrw@tcd.ie) to include Mini-Zebra Constraints and Uniqueness Constraints
%%% Trinity College Dublin, Ireland

%%% Original Mini-Zebra Puzzle by
%%% Henry Kautz (kautz@cs.rochester.edu)
%%% University of Rochester, Rochester, NY 14627.

%%% Original full zebra puzzle solution by
%%% Jonathan Mohr (mohrj@augustana.ab.ca)
%%% Augustana University College, Camrose, AB, Canada T4V 2R3
%%% Available at: http://www.augustana.ca/~mohrj/courses/2000.fall/csc370/lecture_notes/prolog_examples/zebra_puzzle.pro

solve(Who, S):-
%%% There are three houses in a row on a street. Each house is inhabited by a man of a different nationality, who has a different pet, and drinks a different beverage.

%%% 1. The Spaniard owns a dog.
%%% 2. The ukrainian drinks tea.
%%% 3. The man in the third house drinks milk.
%%% 4. The Norwegian lives next to the tea drinker.
%%% 5. The juice drinker owns a fox.
%%% 6. The fox is next door to the dog.

%%% Question: Who owns the zebra? More generally, for each house: who lives there, what pet does he have, and what does he drink?
%%% define House       [Nationality Drink Pet];

% (The constraints that all colours, etc., are different can only be
% applied after all or most of the variables have been instantiated.
% See below.)

% S = [[Nationality1, Drink1, Pet1] [,]]
% The order of the sublists is the order of the houses, left to right.
% S = [[N1,D1,P1],
%     [N2,D2,P2],
%     [N3,D3,P3]],

%%% The Spaniard own a dog.
  member(["spaniard", _, dog], S),
%%% The ukrainian drinks tea.
  member(["ukrainian", tea, _], S),
%%% The man in the third house drinks milk.
  D3 = milk,
%%% The Norwegian lives next to the tea drinker.
next_to(["norwegian", _, _], [_, tea, _ []], S),
% The juice drinker owns a fox.
member([_, juice, fox], S),
% The fox is next door to the dog.
next_to([_, _, fox], [_, _, dog []], S),

%
% The puzzle is so constrained that the following checks are not really
% required, but I include them for completeness (since one would not
% know in advance of solving the puzzle if it were fully constrained
% or not).
%
% All house owners are of different nationalities.
N1 \== N2, N1 \== N3, N2 \== N3,
% They all have different pets.
P1 \== P2, P1 \== P3, P2 \== P3,
% They all drink different drinks.
D1 \== D2, D1 \== D3, D2 \== D3,

% Who owns the zebra?
member([Who, _, zebra], S).

left_of(L1, L2, [L1, L2 []]).
left_of(L1, L2, [_ Rest ]):-
    left_of(L1, L2, Rest).

next_to(L1, L2, S):-
    left_of(L1, L2, S).
next_to(L1, L2, S):-
    left_of(L2, L1, S).

:- time( solve(Who, S), maplist(write, S), nl, write(Who), nl, nl, fail
        ; write("No more solutions.")).
%% this times how long it takes to get ALL possible solutions.
%% even though the Zebra Puzzle has a unique solution, that can't
%% be known before computation begins.
7.5 Zebra Puzzle xfst / foma Code

# Adapted by Ross McKinley (mckinlrw@tcd.ie) to the Life International Magazine Rule-set and to include Uniqueness Constraints
# Trinity College Dublin, Ireland

# Original full zebra puzzle solution by
# Lauri Karttunen <laurik@stanford.edu>
# Centre for the Study of Language and Information
# 210 Panama St, Stanford, CA 94305, USA
# Available at: http://www.stanford.edu/~laurik/fsmbook/examples/Einstein%27sPuzzle.html

define Color [Red | Green | Ivory | Yellow | Blue];
define Nationality [Englishman | Spaniard | Ukrainian | Norwegian | Japanese];
define Drink [OrangeJuice | Coffee | Milk | Tea | Water];
define Cigarette [OldGold | Kools | Chesterfields | LuckyStrike | Parliaments];
define Pet [Dogs | Snails | Fox | Horses | Zebra];

# What are the variables?
define House [Color Nationality Drink Cigarette Pet];

# What does a house look like??
define C1 [House^5];
#1 There are five houses
define C2 [Red Englishman];
#2 The Englishman lives in the red house.
define C3 [Spaniard ~Pet Dogs];
#3 The Spaniard owns the Dogs.
define C4 [Green ~Drink Coffee];
#4 Coffee is drunk in the green house.
define C5 [Ukrainian Tea];
#5 The Ukrainian drinks tea.
define C6 [Ivory ~Color Green];
#6 The green house is immediately to the right of the ivory house.
define C7 [OldGold Snails];
#7 The Old Gold smoker owns snails.
define C8 [Yellow ~Cigarette Kools];
#8 Kools are smoked in the yellow house.
define C9 [House^2 ~Drink Milk ~Drink House^2];
#9 Milk is drunk in the middle house.
define C10 [? Norwegian ?*];
#10 The Norwegian lives in the first house.
define C11 [Chesterfields ? ~Pet Fox | Fox ~Cigarette Chesterfields];
#11 The man who smokes Chesterfields lives in the house next to the man with the fox.
define C12 [Horses ~Cigarette Kools | Kools ? ~Pet Horses];
#12 Kools are smoked in the house next to the house where the horse is kept.
define C13 [OrangeJuice LuckyStrike];
#13 The Lucky Strike smoker drinks orange juice.
define C14 [Japanese ~Cigarette Parliaments];
#14 The Japanese smokes Parliaments.
define C15 $[Norwegian ~$Color Blue | Blue ? ~$Nationality Norwegian]; #15 The Norwegian lives next to the blue house.

# Here we have the uniqueness constraint. This says all of the variables are shuffled.
# It turns out that doing a shuffle is more efficient than
# a full lookahead/behind for each variable, and then combining them

define CX [Red <> Green <> Ivory <> Yellow <> Blue
  <> Englishman <> Spaniard <> Ukrainian <> Norwegian <>
  Japanese
  <> OrangeJuice <> Coffee <> Milk <> Tea <> Water
  <> OldGold <> Kools <> Chesterfields <> LuckyStrike <>
Parliaments
  <> Dogs <> Snails <> Fox <> Horses <> Zebra
]; # there is only one of each variable

define Solution [ C1 & C2 & C3 & C4 &
  C5 & C6 & C7 & C8 &
  C9 & C10 & C11 & C12 &
  C13 & C14 & C15 & CX
  & $Water & $Zebra
 ];

regex Solution;
solve(Who, [C1, N1, P1, D1, S1]) :-
% 1 There are five houses.
% X Each house has its own unique color.
  C = [red, green, ivory, yellow, blue],
  permutation(C,C1),
% X All house owners are of different nationalities.
  N = [englishman, spaniard, ukrainian, norwegian, japanese],
  permutation(N,N1),
% X They all have different pets.
  P = [dogs, snails, horse, zebra],
  permutation(P,P1),
% X They all drink different drinks.
  D = [orangejuice, coffee, milk, tea, water],
  permutation(D,D1),
% X They all smoke different cigarettes.
  S = [oldgold, kools, chesterfields, luckystrike, parliaments],
  permutation(S,S1),
% 2 The English man lives in the red house.
  samepos(red, C1, english, N1),
% 3 The spaniard has a dog.
  samepos(spaniard, N1, dogs, P1),
% 4 In the green house, they drink coffee.
  samepos(green, C1, coffee, D1),
% 5 The ukrainian drinks tea.
  samepos(ukrainian, N1, tea, D1),
% 6 The green house is on the right side of the ivory house.
  inorder(ivory, green, C1),
% 7 The man who smokes oldgold has snails.
  samepos(snails, P1, oldgold, S1),
% 8 In the yellow house, they smoke kools.
  samepos(yellow, C1, kools, S1),
% 9 In the middle house, they drink milk.
  D1 = [, , milk, , ],
% 10 The Norwegian lives in the first house.
  N1 = [norwegian | ],
% 11 The man who smokes Chesterfields lives in the house next to the man with the fox.
  next_to(chesterfields, S1, fox, P1),
% In the house next to the house with the horse, they smoke kools.
  next_to(dunhill, S1, horse, P1),
% The man who smokes luckystrike drinks orange juice.
samepos(orange juice, D1, luckystrike, S1),
% The Japanese smokes parliaments.
samepos(japanese, N1, parliaments, S1),
% The Norwegian lives next to the blue house.
next_to(norwegian, N1, blue, C1),
% Who owns the zebra?
samepos(zebra, P1, Who, N1).

permutation( L, [ H | T ] ) :-
  append( U, [ H | V ], L ),
  append( U, V, W ),
  permutation( W, T ).
permutation( [], [] ).
samepos(A, [A|], B, [B|]).
samepos(A, [ |T1], B, [ |T2]) :-
  samepos(A, T1, B, T2).
inorder(L1, L2, [L1, L2 | ]).
inorder(L1, L2, [ | Rest ]) :-
  inorder(L1, L2, Rest).
next_to(A, [A|], B, [ |B|]).
next_to(A, [ |A|], B, [B|]).
next_to(A, [ |T1], B, [ |T2]) :-
  next_to(A, T1, B, T2).

:- time(
  solve(Who, S),
  maplist(writeln,S), nl,
  write(Who), nl, nl, fail
; write('No more solutions.' ) ).

%%% this times how long it takes to get ALL possible solutions.
%%% even though the Zebra Puzzle has a unique solution, that can’t
%%% be known before computation.
% 7.7 Zebra Puzzle Partial Instantiation Code

%% Adapted by Ross McKinley (mckinlrw@tcd.ie) to the Life International Magazine Rule-set
%% Trinity College Dublin, Ireland

% Original zebra puzzle solution by
% Jonathan Mohr (mohrj@augustana.ab.ca) Augustana University College, Camrose, AB, Canada T4V 2R3
% Available at: http://www.augustana.ca/~mohrj/courses/2000.fall/csc370/lecture_notes/prolog_examples/zebra_puzzle.pro

solve(Who, S) :-
  % There are five houses.
  S = [ [C1,N1,P1,D1,S1],
        [C2,N2,P2,D2,S2],
        [C3,N3,P3,D3,S3],
        [C4,N4,P4,D4,S4],
        [C5,N5,P5,D5,S5] ],
  % 1 There are five houses

%2 The Englishman lives in the red house.
  member([red, englishman, _, _, _], S),

% The Spaniard has a dog.
  member([_, spaniard, dog, _, _], S),

% In the green house, they drink coffee.
  member([green, _, _, coffee, _], S),

% The ukrainian drinks tea.
  member([_, ukrainian, _, tea, _], S),

% The green house is immediately to the right of the ivory house.
  left_of([ivory _], [green _], S),

% 7 The Old Gold smoker owns snails.
  member([_, _, snails, _, oldgold], S),

% 8 In the yellow house, they smoke Dunhill.
  member([yellow, _, _, _, kools], S),

% 9 In the middle house, they drink milk.
  D3 = milk,

% 10 The Norwegian lives in the first house.
  N1 = norwegian,

% 11 The man who smokes Chesterfields lives in the house next to the man with the fox.
  next_to([_, _, _, chesterfields], [_, _, fox _], S),

% 12 Kools are smoked in the house next to the house where the horse is kept.
  next_to([_, _, _, kools], [_, _, horse _], S),

% 13 The Lucky Strike smoker drinks orange juice.
  member([_, _, orangejuice, luckystrike], S),

% 14 The Japanese smokes Parliaments.
  member([_, japanese, _,_, parliaments], S),

% The Norwegian lives next to the blue house.
% The puzzle is so constrained that the following checks are not really required, but I include them for completeness (since one would not know in advance of solving the puzzle if it were fully constrained or not).

% Each house has its own unique color.
C1 \(\equiv\) C2, C1 \(\equiv\) C3, C1 \(\equiv\) C4, C1 \(\equiv\) C5,  
C2 \(\equiv\) C3, C2 \(\equiv\) C4, C2 \(\equiv\) C5,  
C3 \(\equiv\) C4, C3 \(\equiv\) C5, C4 \(\equiv\) C5,

% All house owners are of different nationalities.
N1 \(\equiv\) N2, N1 \(\equiv\) N3, N1 \(\equiv\) N4, N1 \(\equiv\) N5,  
N2 \(\equiv\) N3, N2 \(\equiv\) N4, N2 \(\equiv\) N5,  
N3 \(\equiv\) N4, N3 \(\equiv\) N5, N4 \(\equiv\) N5,

% They all have different pets.
P1 \(\equiv\) P2, P1 \(\equiv\) P3, P1 \(\equiv\) P4, P1 \(\equiv\) P5,  
P2 \(\equiv\) P3, P2 \(\equiv\) P4, P2 \(\equiv\) P5,  
P3 \(\equiv\) P4, P3 \(\equiv\) P5, P4 \(\equiv\) P5,

% They all drink different drinks.
D1 \(\equiv\) D2, D1 \(\equiv\) D3, D1 \(\equiv\) D4, D1 \(\equiv\) D5,  
D2 \(\equiv\) D3, D2 \(\equiv\) D4, D2 \(\equiv\) D5,  
D3 \(\equiv\) D4, D3 \(\equiv\) D5, D4 \(\equiv\) D5,

% They all smoke different cigarettes.
S1 \(\equiv\) S2, S1 \(\equiv\) S3, S1 \(\equiv\) S4, S1 \(\equiv\) S5,  
S2 \(\equiv\) S3, S2 \(\equiv\) S4, S2 \(\equiv\) S5,  
S3 \(\equiv\) S4, S3 \(\equiv\) S5, S4 \(\equiv\) S5,

% Who owns the zebra?
member([_, Who, zebra, _, _], S).

% Or, replace the last line by:
%   format("The ~w owns the zebra.", Who).

left_of(L1, L2, [L1, L2 | _]).
left_of(L1, L2, [_ Rest ]) :-
    left_of(L1, L2, Rest).

next_to(L1, L2, S) :-
    left_of(L1, L2, S).
next_to(L1, L2, S) :-
    left_of(L2, L1, S).

:- time( ( solve(Who, S), maplist(writeln,S), nl, write(Who), nl, nl, fail
          ; write("No more solutions.") ) ).
% this times how long it takes to get ALL possible solutions.
% even though the Zebra Puzzle has a unique solution, that can't be known before computation.
8. References


