Artificial Intelligence DT2016

Task 2 - Constraint Satisfaction Problem

By *Özgun Mirtchev*

Professor *Franziska Klügl*

Örebro University

April 20, 2017

1 Introduction

Constraint Problem Solving has been an important area to teach computers, after all that's what the computers are for. Solving problems that we humans cannot. Ever since the beginning researchers has come up with different algorithms and different approaches to take on different problems.

1.1 Objective

In this lab we will explore two different problem solving algorithms namely Backtracking Search and Arc Consistency. The problem is a 3x3 cross word puzzle (see Figure 1), for which we will find out which words can fit together according to custom constraints. The problem will be solved as a constraint satisfaction problem at first with backtracking search algorithm. Secondly, the problem will be made network consistent using the Arc Consistency 3 algorithm. At last the problem will be solved by using the two algorithms together. The variables are of A1, A2, A3, D1, D2 and D3 - representing three words across and three words down.

A1, D1	D2	D3
A2		
A3		

Figure 1: The provided 3x3 Crossword Puzzle

1.2 Constraint Satisfaction Problem (CSP)

The backtracking search is in itself a recursive backtracking procedure. The problem consists of a set of variables, a domain for each of the variables and a set of constraint rules. The goal is to choose a value for each variables so that the resulting possible word satisfies the constraints. For every word that passes the constraints, a new level of the algorithm is entered through recursive action. At that level, the same procedure is then done as before, checking the next variable instead. If the constraints are not met for a certain variable, the recursive function returns a false statement, which rolls back to the variable where the constraints were satisfied. It keeps going like this, until all the domains for each variable has been reduced to one value which satisfies all the constraints for each variable or no values, which leaves it with no solution.

1.3 Arc Consistency (AC)

A variable Xi is arc-consistent with respect to variable Xj, if for every value in the current domain Di, there is some value in the domain Dj that satisfies the (binary) constraint on the arc (Xi,Xj). A network is arc-consistent if every variable is arc-consistent with every other variable.

This is the general description for Arc-Consistency. The problem setup is similar to CSP. Since it's a network, one can see it as a web, with each variable having a connection to every variable with common constraints. There are three possible outcomes when it comes to AC. Either (i) one domain is empty, in which case there is no solution to the problem, or (ii) each domain has a single value, which means the problem has a unique solution, or (iii) some domains have more than one value. This gives an indication of two things; either the problem has a solution, or it does not. The answer will, in that case, be found out by performing search.

2 Results

2.1 Scenario Preparation

With guidance from the lab instructions it was suggested to have an object for each variable and domains. The way this was solved instead was creating a class which holds a list of variables and a class which holds a list of domains, to keep them separate from each other when working with them. There was a file with 40 words to use for the problem. The words were read into the program by adding them to the domain class as strings and the variables were done the same way. Adding them as strings is probably a controversial way of doing it, but it proved to be easier after having tried both approaches; of storing them in a specific Word classes and just manipulating the strings directly. The constraints are in their own class so that it may be easier for both the CSP and AC algorithms to access. However, even if they have similar constraints to check, they do it in similar but still different ways, because of some bad choices, which required the constraints to be custom-made for each sub-task.

2.2 Program procedure

In this section, info about each sub-task will be described through how the program was designed. Upon starting the program in the console, a menu is displayed listing the available problem solving options:

1. Backtracking
2. Arc-Consistency(3)
3. Backtracking with AC

2.2.1 Backtracking

When choosing backtracking the program loads the corresponding data and instantly starts solving the problem. Depending on how big the domain is for each variable, it will take a few seconds for the problem to finish solving, at which point it will tell the user if there is a solution or not. If the program was able to solve the problem, the solution will be presented with the words for each variable in a crossword print, some detailed information on how many iterations were done on each recursive level and the time it took to solve the problem:

```
Solving with backtracking. Please wait...

Problem was solved. Solution:
                        D1   D2   D3
    +-------------+    +-------------+
A1  | B | E | E |    | B | E | E |
    +-------------+    +-------------+
A2  | O | A | F |    | O | A | F |
    +-------------+    +-------------+
A3  | A | R | T |    | A | R | T |
    +-------------+    +-------------+

Number of domains: 40
Number of variables: 6

Total assignments: 42911 ({1: 28, 2: 1089, 3: 41361, 4: 427, 5: 5, 6: 1})
Total backtracks:  42905
Approximate time:  2.901568769688661 s
```

If however the program could not solve the problem, it will not display a solution and instead just print out the detailed information and the time it took to run the problem solving algorithm. As an example, the domain "BOA" was removed:

```
Solving with backtracking. Please wait...

Unable to solve problem. No solution.

Number of domains: 39
Number of variables: 6

Total assignments: 56894 ({1: 39, 2: 1482, 3: 54834, 4: 539})
Total backtracks:  56894
Approximate time:  3.79085442019866 s
```

2.2.2 Arc-Consistency(3)

When choosing the Arc-Consistency option, the program runs the problem on the Arc-Consistency 3 algorithm. This algorithm, is more efficient than earlier implementations and a bit easier to implement that the later ones, such as AC-4 and AC-5.

Similar to the CSP option, the AC choice, loads the data into the corresponding data structures and starts solving the problem immediately. During

execution it also prints out if the arc network, between two variables, are consistent or not. If it's not consistent, the algorithm removes the domains that are violating the constraints of the arc and prints it out.

```
...
Arc: D2->A3 - Consistent
Arc: D3->A3 - Consistent
Arc: A1->D1 - Removed ['ARE', 'ART']
Arc: A2->D1 - Consistent
Arc: A3->D1 - Removed ['OAF']
Arc: A1->D1 - Consistent
Arc: A2->D1 - Consistent
...
```

When a solution is found it prints out the entire arc consistent network for the user:

```
Network is Arc-consistent:
A1 ['BEE', 'BOA', 'LEE']
A2 ['EAR', 'OAF']
A3 ['ARC', 'ARE', 'ARK', 'ARM', 'ART', 'EFT']
D1 ['BEE', 'BOA', 'LEE']
D2 ['EAR', 'OAF']
D3 ['ARC', 'ARE', 'ARK', 'ARM', 'ART', 'EFT']
```

Otherwise it prints out that an empty domain was detected and there is no solution:

```
Empty domain detected. No solution.
```

Illustrating the arc-consistency network would make it easier to understand how it looks like before and after running the AC3 algorithm (see Figure 2 & 3)

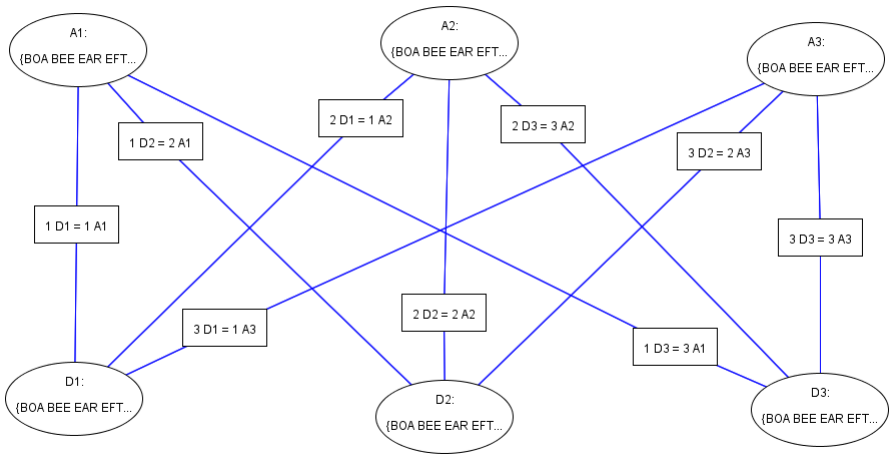

Figure 2: Network before AC3

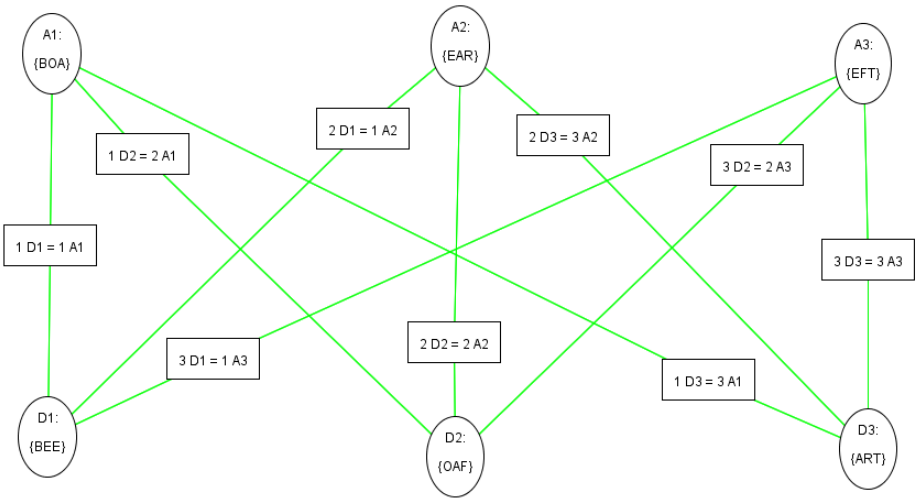

Figure 3: Network after AC3

2.2.3 Backtracking with AC

Selecting the third option, the program will first make the network arc-consistent by running the AC3 and then perform the search to solve the problem.

```
Solving with arc-consistency backtracking. Please wait...
...
Arc: D1->A2 - Consistent
Arc: D2->A2 - Consistent
Arc: D3->A2 - Removed ['ADD', 'ADO', 'AYE', 'EEL', 'LEE']
Arc: D1->A3 - Removed ['EEL']
Arc: D2->A3 - Consistent
Arc: D3->A3 - Consistent
...
Problem was solved. Solution:
                        D1   D2   D3
   +-------------+   +-------------+
A1 | B | E | E |   | B | E | E |
   +-------------+   +-------------+
A2 | O | A | F |   | O | A | F |
   +-------------+   +-------------+
A3 | A | R | T |   | A | R | T |
   +-------------+   +-------------+

Number of domains: 22
Number of variables: 6

Total assignments: 94 ({1: 1, 2: 4, 3: 62, 4: 9, 5: 17, 6: 1})
Total backtracks:  88
Approximate time:  0.04246530067866144 s
```

2.3 Profiling tests

The average time taken and the amount of constraint rule calls were measured with a profiler. The measurements may vary depending on which system is used and what kind of CPU is used because of different cycles. The data presented here is more of a guideline to see how much faster it is between two methods of solving a problem.

	Backtracking	Arc Consistency with Backtracking
Time to solve	~3s	~0.05s
Total constraint calls	1.5M	25k

3 Conclusion

Running the backtracking search with a brute force tactic, without any means of reducing unnecessary values in the domains, will take a long time to produce a solution. The average time to solve the presented problem with the crossword puzzle, takes about 3 seconds with an immense amount of calls to the constraint rules, because it checks every possible combination for a valid candidate, even if

it has been proven before that the value of the domain is not consistent. Adding more values to the domain will increase the time it takes to solve, because of the nature of the algorithm.

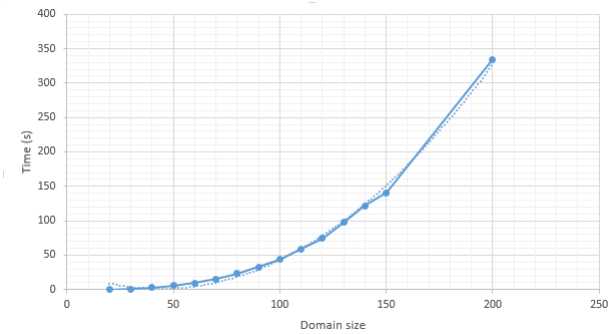

Figure 4: Time to solve with recursive backtracking

Running arc-consistency on a problem before doing search, may as one can see, reduce the time it takes to solve the problem from a few seconds to almost instantly. Since arc-consistency removes the domains that are not consistent with the constraints, there becomes fewer to iterate through when solving the problem with backtracking search. Total assignments when doing only backtracking was up to 42k, while doing it with an arc-consistent network reduced it down to barely 100 assignments, making it an invaluable method to solve problem, to reduce time.

Improvements that could've been made would be the time it takes to solve a problem. It might have been able to be reduced even more if the constraint rules would be more optimized so that unnecessary checks are not done.

This task took me the longest time to finish, because of my unfamiliarity with the constraint satisfaction problems and because I'm not that fast at learning something just through text (mostly supplied from the course material) but had to complement with pictures and animations from the internet to learn about it. To learn about Arc-Consistency and more about Constraint Satisfaction Problems, I would like to suggest a web-page (http://www.aispace.org/mainTools.shtml), that you hopefully may add as a suggestion in the course material for learning more. It offers several tools for learning about these problems through animations and illustrations. It helped me a lot in understanding certain subjects, which were difficult to get a visual picture of just through text.

This task was very good for learning about CSP and the setup of the lab was good. It was easy because of the structure, so that one focuses on solving one problem first before moving on to the next on and then at last combining the two together.