Software report

Nenișcă Maria

1.Introduction

In this software report, we solve the Cryptarithmetic puzzle and the Zebra puzzle using CSP. The implementation was written in Java, and a custom implementation for the CSP representation was written.

2. Architecture

For both problems we implemented a basic backtracking algorithm (BT), a forward-checking backtracking algorithm (FC-BT), and an AC3 + FC-BT algorithm (AC3-FC-BT).

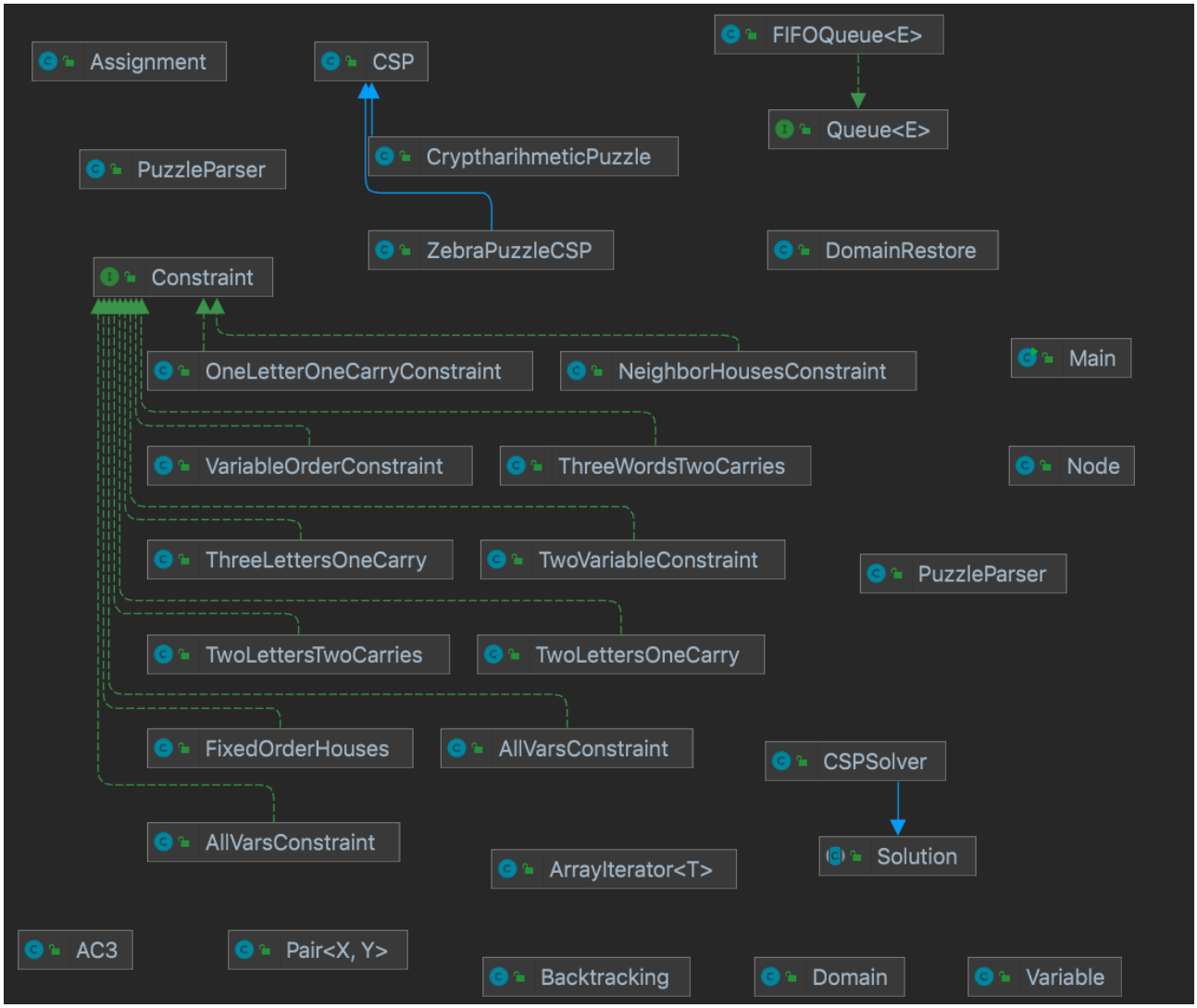

Figure 1 Architecture of the project

3.The cryptarithmetic puzzle

3.1. Problem description

Cryptarithmetic puzzles are arithmetic problems made of letters instead of numbers. The goal is to replace each letter with a single, unique digit in order to make the arithmetic work out correctly. These are the rules used for solving cryptarithmetic problems that were used in the implementation:

- Each letter represents only one digit throughout the problem.
- Numbers must not begin with zero i.e. 0567 (wrong) , 567 (correct).
- The aim is to find the value of each letter
- The numerical base is 10
- After replacing letters by their digits, the resulting arithmetic operations must be correct.

Some examples of cryptarithmetic puzzles:

- TO+TO+FOR
- TWO+TWO+FOUR
- ODD+ODD=EVEN
- USA+USSR=PEACE

3.2. Solution model

We will take as an example the puzzle: TWO+TWO=FOUR.

Each letter in a cryptarithmetic puzzle represents a different digit: this would be represented as the global constraint AllVarsConstraint(F,T,U,W,R,O). The addition constraints on the four columns of the puzzle can be written as the following n-ary constraints:

- ThreeLettersOneCarry: $O + O = R + 10*C_{10}$
- ThreeWordsTwoCarries: $C_{10} + W + W = U + 10*C_{100}$
- ThreeWordsTwoCarries: $C_{100} + T + T = O + 10*C_{1000}$
- OneLetterOneCary: $C_{1000} = F$

where C_{10}, C_{100}, and C_{1000} are auxiliary variables representing the digit carried over into the tens, hundreds, or thousands column. These constraints can be represented in a constraint hypergraph (see Figure 2), which consists of ordinary nodes (the circles in the figure) and hypernodes (the squares), which represent n-ary constraints.

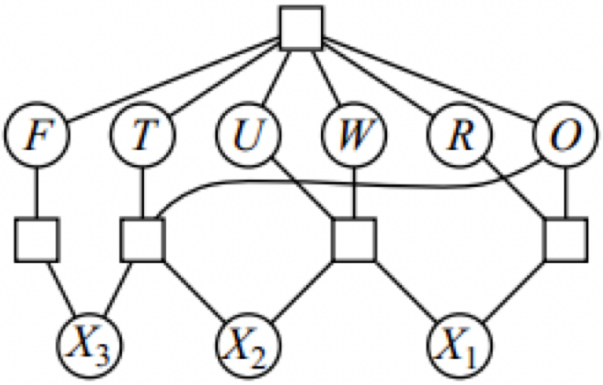

Figure 2 Constraint graph for TWO+TWO=FOUR

We can generalize this model for any cryptarithmetic puzzle. The number of carries in an addition is equal to the length of the result – 1. The values that a carry can take are {0,1}. We parse a given puzzle from right-to-left, on columns. Ex:

	T	W	O
	T	W	O
F	O	U	R
C3	C3	C2	C1

During parsing, apply the following constraints:

- ThreeLettersOneCarry for last column in the table (ex: O, O, R + first carry)
- ThreeWordsTwoCarries for the middle columns of letters (ex: W, W, U + the appropriate carry pair).
- ThreeLettersOneCarry, when we get to the first column, and we have 3 letters in it + the last carry
- TwoLettersOneCarry, when we get to the first column of the puzzle, and we have 2 letters in the column + the last carry
- OneLetterOneCary, when we get to the first column, and we have only one letter in the column (the one from the sum) + the last carry
- AllVarsConstraint, applied at the end for all the variables in the puzzle. It says that all the variables (besides the carries) need to have unique digits assigned to them.

The variables of the CSP are the unique letters of the puzzle + one variable for each carry in the sum. In our case, we would have for carries: {C1, C2, C3}.

3.3. Comparison between algorithms

We ran the algorithms on our example case, TWO+TWO=FOUR, and the statistics are in Table 1.

Table 1 Statistics of the 3 algorithms for the cryptarithmetic puzzle TWO+TWO=FOUR

Algorithm	BT	AC3-FC-BT	FC-BT
No. of solutions	27	27	27
Used memory (Mb)	152.171	8.176	142.183
Elapsed time (s)	0.608	0.163	0.180

The size of the constraint graph is for this problem is 19.

From Table 1, it can be observed that the best performing algorithm was AC3-FC-BT, followed by FC-BT, and lastly, BT. All the algorithms found all the solutions to our cryptarithmetic puzzle. It was example that the BT algorithm would perform the worst, as it compares all the possible combinations of variable domains, with no heuristic. AC3-FC-BT was expected to have the best performance, because in this algorithm we minimize the domains of each variable to have an arc consistent graph, and we also, during FC-BT, remove from the domain of the variables the values that are not compatible with the current solution during computation. Doing these 2 things, we reduce the search space, resulting in less memory used and in a faster run.

We ran it on the USA+USSR=PEACE puzzle as well, and the results are in Table 2.

Algorithm	BT	AC3-FC-BT	FC-BT
No. of solutions	10	10	10
Used memory (Mb)	172.104	23.213	139.221
Elapsed time (s)	5.182	1.423	1.449

Table 2 Results of the 3 algorithm run on USA+USSR=PEACE

The size of the constraint graph for this puzzle is 26.

From Table 2, we can see that the classification of the algorithm is the same as for the other example when it comes to performance. However, in this case, the constraint graph size is larger, which has resulted in more memory used and in a higher elapsed time for all the algorithms. Therefore, we can see that the performance of the algorithms is not determined only the by size of the search space, but also by the size of the constraint graph.

4. The zebra puzzle

4.1. Problem description

The zebra puzzle we try to solve in our implementation is the following:

1. There are five houses.
2. The English man lives in the red house.
3. The Swede has a dog.
4. The Dane drinks tea.
5. The green house is immediate to the left of the white house.
6. They drink coffee in the green house.
7. The man who smokes Pall Mall has birds.
8. In the yellow house they smoke Dunhill.
9. In the middle house they drink milk.
10. The Norwegian lives in the first house.
11. The man who smokes Blend lives in the house next to the house with cats.
12. In a house next to the house where they have a horse, they smoke Dunhill.
13. The man who smokes Blue Master drinks beer.
14. The German smokes Prince.
15. The Norwegian lives next to the blue house.
16. They drink water in a house next to the house where they smoke Blend.

The goal is to complete the following table, based on the above constraints:

House	Nation	Animal	Drink	Cigarettes	Color
1	Norwegian				
2					Blue
3			Milk		
4					
5					

4.2. Solution model

A zebra puzzle contains multiple entity types: house, nation, animal, drink, cigarettes, and colors. Each entity type has 5 entities under it, all of them specified in the problem definition:

- Nations: English, Swedish, Danish, Norwegian, German
- Animals: Dog, Birds, Cats, Horse, Zebra
- Drinks: Tea, Coffee, Milk, Beer, Water
- Cigarettes: Pall Mall, Dunhill, Blend, Blue Master, Prince
- Color: Red, Green, White, Yellow, Blue
- House: 1, 2, 3, 4, 5

In our CSP model, we have as variables each entity, besides the ones of type House. We notate each as follows:

- Nations: N{number}
- Animals: A{number}
- Drinks: D{number}
- Cigarettes: T{number}
- Color: C{number}

The variable domain is the set of house numbers: Domain = {1, 2, 3, 4, 5}.
So, the variable set is:

$$\{\{letter\}\{number\} \mid number \in Domain, letter \in N, A, D, T, C\}$$

The all-variables constraint is that each house needs to have only 5 entities assigned to it, i.e., if X is a letter: $X_i! = X_j \leftrightarrow i! = j$, where $i, j \in Domain$.

The set of constraints is the following:

1. The English man lives in the red house. => $N_1=C_1$
2. The Swede has a dog. => $N_2=A_1$
3. The Dane drinks tea. => $N_3=D_1$
4. The green house is immediate to the left of the white house. => $C_2=C_3-1$
5. They drink coffee in the green house. => $D_2=C_2$
6. The man who smokes Pall Mall has birds. => $T_1=A_2$
7. In the yellow house they smoke Dunhill. => $C_4=T_2$
8. In the middle house they drink milk. => $D_3=3$
9. The Norwegian lives in the first house. => $N_4=1$
10. The man who smokes Blend lives in the house next to the house with cats. => $|T_3-A_3|=1$
11. In a house next to the house where they have a horse, they smoke Dunhill. => $|T_2-A_4|=1$
12. The man who smokes Blue Master drinks beer. => $T_4=D_4$
13. The German smokes Prince. => $N_5=T_5$
14. The Norwegian lives next to the blue house. => $|N_4-C_5|=1$

15. They drink water in a house next to the house where they smoke Blend. => $|D_5-T_3|=1$

4.3. Comparison between algorithms

We ran the same algorithms as for the cryptarithmetic puzzle, with the current problem model in place. The performance results are in Table 3.

Algorithm	BT	AC3-FC-BT	FC-BT
No. of solutions	1	1	1
Used memory (Mb)	17.220	11.298	7.287
Elapsed time (s)	0.195	0.131	0.079

Table 3 Performance results of the zebra puzzle

The solution is:

House	Nation	Animal	Drink	Cigarettes	Color
1	Norwegian	Cats	Water	Dunhill	Yellow
2	Danish	Horse	Tea	Blend	Blue
3	English	Birds	Milk	Pall Mall	Red
4	German	Zebra	Prince	Coffee	Green
5	Swedish	Dog	Beer	Blue Master	White

And the size of the constraint graph is 41.

Analyzing the performance metrics, we notice that FC-BT performed the best, and BT the worst. Again, it was expected for the BT algorithm to have the worst performance. However, AC3-FC-BT did not have the best performance here as it did in the previous puzzle. This indicates that trying to make the graph arc-consistent does not help in this problem, it only adds additional complexity to the solution, with no extra benefit.

5. Future work

As future work, more search strategies can be implemented, as well as a generalization for the zebra puzzle.