CSC384 Mini-Project

Advance Artificial Intelligence Application: CryptArithmetic

Project Type: CSP

MINGYE WANG

999573938

CDF: g3nicky

YIFEI YANG

999590526

CDF: g3yangyl

Date: Dec 8th 2015

Advance Artificial Intelligence Application: CryptArithmetic

1. Introduction

The problem which we choose for this project is known as "Cryptarithmetic", we believe that it is a very interesting problem. Crypt arithmetic is a kind of puzzle "which the digits were replaced by the letters of the alphabet or other symbols"[1]. For example, the most famous Crypt arithmetic puzzle in the world is:

```
  S E N D
+ M O R E
---------
M O N E Y
```

This case was created by H.E.Dudeney and first published in July 1924. The solution for this puzzle is "9567 + 1085 = 10652", every digit corresponded to a letter[2]. The addition is not the only mathematical operation for this puzzle, it can deal with any basic mathematical operation (such as subtraction, multiplication and division). When people were study in primary school and junior high school, Crypt arithmetic puzzle always exists in their math courses' exam. Although these problems were very simple, people usually spent lots of time on solving them, since the way they solve these kinds of problem was just guess and sometimes even cannot get the right answer. Crypt arithmetic puzzle became more and more popular among the public and people have found many methods to solve it efficiently. After taking CSC384 this semester, I realized that the Crypt arithmetic puzzles can be solved by artificial intelligence, the ideal of Constraint Satisfaction Problems(CSP) is a good way to solve it. If people apply CSP on Crypt arithmetic puzzles, they will get the solution very soon.

2. Describtion

The easiest way to solve Crypt arithmetic puzzle is using the enumeration. For each parameter, we can get it's all possible values:

```
Length = len(para1)
Para1_domain = []
For value in range(10*(Length - 1), 100*(Length - 1) - 1):
    Para1_domain.append(value)
```

The list Para1_domain contains all the possible values of para1 (i.e. if the parameter is "EAT", then it's domain should be the number from 100 to 999). The way to create Para2_domain, Para3_domain is the same as Para1_domain. After get all the parameters' domain of value, people must find all the combination which para1 + para2 = para3:

```
Combination_list = []
For para_1 in Para1_domain:
    For para_2 in Para1_domain:
        If para_1 + para2 in para3:
            Combination_list.append([para1, para2, para3])
```

The list Combination_list contains all the combination that para1 + para2 = para3. However, that was not the final answer, since different element should have different assign value and same element should have same value, people must loop the Combination_list. For every element in Combination_list, people need to check if

1) different element have same value
2) same element have different value.

If one of these two cases exist, that element should be removed. After doing that, if len(Combination_list) is not 0, then all the element in that list should be the solution. Obviously, people cannot use this method, since the last step will take lots of time, it should loop all the possible combinations, the time complexity of enumeration is so high, nearly $O(n^n)$. In this project, we cannot use this method, because of it's time complexity, we must find a way which has fewer time complexity.

To make the problem simpler and clear, in this project, the mathematical operation was only between two parameters (actually, the way to solve puzzle with two parameters is the similar with three parameters or more). When we first met the equation "SEND + MORE = MONEY", we have no idea that how to solve this equation without guessing it or using enumeration. After considering this problem for a long time, we found this puzzle was very similar to the Sudoku[3] after we modified it's structure a little bit. It was known to all that the structure of Sudoku is a 9 * 9 square, it can be regarded as a 9 * 9 matrix. According to the structure of Sudoku, the equation "SEND + MORE = MONEY" can be changed as:

```
  0 S E N D
+ 0 M O R E
-----------
  M O N E Y
```

For the addition operation, the sum of the two parameters should always larger than any parameters (assume two parameters should not be zero). The original structure may not in the form of n * m matrix. So, in order to use the CSP as Assignment2, people should first record the length of the result, mark it as N. Since the length of two parameters should always shorter than the result, people just need to add 0 at the front until their length equal to N. After finishing that, the structure of crypt arithmetic puzzle has become a 3 * N matrix, and this is the prototype of the CSP solution (this part's code is in Cryptarithmetic.py line 49 – 67). For subtraction, multiplication and division, the work are similar. Here is the results:

0 S E N D	M O N E Y	M O N E Y
* 0 M O R E	- 0 M O R E	÷ 0 0 A B C
M O R E S E N D	0 S E N D	0 0 0 M A

Then people need to define the variables and constraints of crypt arithmetic puzzle. This part is also similar to the Sudoku. After people changing the structure to a 3 * N matrix, then there exists 3N elements here, these elements are the variables of the puzzle. However, it does not mean that there are 3N variables, since there may exists duplicate elements (i.e. 0, E, N….), all the variables in CSP should be unique. The first thing people need to do is create the variable object, each variable corresponded to an element. If the element is 0, which means this element is created by people themselves, it does not actually exist, the domain of element 0 should be 0. If the element is not zero, which means that element was in the equation and people must get its value. There exist two cases:

1) If that element is the highest bit of that number (i.e. “S” in “SEND”), then the value of that element should not be 0, the domain of it should be [1, 2, 3, 4, 5, 6, 7, 8, 9]
2) If that element is not the highest bit of that number (i.e. “E” in “SEND”), then the value of that element can be any number, the domain of it should be [0,1, 2, 3, 4, 5, 6, 7, 8, 9]

After loop 3 words, all the variables were created and the duplicated variables were deleted (this part’s code is in Cryptarithmetic.py line 93 - 123).

After created variables, people need to construct constraints. The way to construct constraints is similar to the Sudoku. As described above, the structure of puzzle had become a 3*N matrix. For each row of the matrix, we create a constraint, the variables of the constraint are all the elements of that row. Then we need to find the satisfied tuples. Since different elements cannot have same value. People can use itertools.product() to get all the possible values and remove the invalid values, use the function add_satisfying_tuples() add the values into the constraints. After created 3 constraints above, for each column of the matrix, people also need to construct the constraints. How to get the variables of the constraint are same as before, but we need to consider more on satisfied tuples. Since for each column (defined as col), col[2] should equal to col[0] + col[1] or col[0] + col[1] – 10. For example, in the “SEND + MORE = MONEY”:

D	invalid satisfied tuple:	7	valid satisfied tuple:	7
E		5		5
Y		3		2

After checking the invalid value combinations, people need to consider the condition above, if one value combination cannot satisfy the condition above, it should be removed from the satisfied tuple. The last constraint is the whole matrix, the value of the constraint are all the variables, the satisfied tuple are all the combinations that first row + second row = third row. (other mathematical operation are similar with addition, and this part's code is in Cryptarithmetic.py line 124 – 289)

After creating variables and constraints, people need to create a CSP object (define as simpleCSP), The variable of simpleCSP is the variables crested above, then add all the constraints into simpleCSP. After finishing all the steps, people need to use prop_GAC() to find the final solutions.

3. Reason why we choose CSP

The reason why we choose the idea of CSP to solve Crypt arithmetic puzzle is that it can save lots of time when it searches for the final solution. In Assignment 2, after people restricted the variables and constraints, the domain of value which program need to check become smaller, so that the program can find the solution more quickly. In our project, as described above, after modified puzzle's structure, the puzzle become similar to the Sudoku. Each time, when a variable was created, we must check whether it can be 0 or not. When a row constraint was created, people should consider about the all-different condition, then check and delete the invalid satisfied tuples. When a column constraint was created, people not only need to consider about the all-different condition, but also the sum condition as mentioned above. Although these work will take people some time to implement, however, after doing these, each variable's domain has been restricted, many possible satisfied tuples were deleted. When people use prop_GAC() to solve the puzzle, the

number of values which program should loop and check become small, it will run faster. The running time will obviously decrease contrast to the enumeration.

Here is the result of some examples:

The input is a list of list, the first two lists are the parameters that we need to deal with, the third list is the result of the first two lists after operation, and the last element is the operation type.

```
>>> puzzle([['a'],['b','b'],['a','c','c'], "+"])
a: 1
b: 9
c: 0

>>> puzzle([['a','b'],['c','b','d'],['c','c','b'], "+"])
a: 1
b: 2
c: 3
d: 0

>>> puzzle([['a','b','c'],['c','a','b'],['d','a','c'], "+"])
a: 1
b: 0
c: 2
d: 3

>>> puzzle([['a','b'],['c','c','b'],['a','d','d','e'], "+"])
a: 1
b: 2
c: 9
d: 0
e: 4

>>> puzzle([['a','b'],['b','c'],['d','c','b'],'*'])
a: 2
b: 3
c: 1
d: 7

>>> puzzle([['a','b','c'],['c','c'],['a','d','a','c'],'*'])
a: 3
b: 2
c: 1
d: 5
```

```
>>> puzzle([['a','b','b'],['a'],['c','c'],'-'])
a: 1
b: 0
c: 9

>>> puzzle([['a','b','c','d'],['c','d','b'],['e','f','b'],'-'])
a: 1
b: 2
c: 3
d: 4
e: 8
f: 9

>>> puzzle([['a','b','a'],['a','a'],['a','a'],'/'])
a: 1
b: 2

>>> puzzle([['a','b','c','d'],['e','a','e'],['a','d'],'/'])
a: 2
b: 7
c: 8
d: 3
e: 1
```

4. Conclusion

After trying many examples, it shows that our program based on CSP can always get the right answer. The running time of the program is very small, the results come out quickly. From the information above, people can conclude that CSP is a good method to solve the problem which need to use the enumeration to solve. Constraints can divide the problem into many small parts, people can deal with constraints based on it's own condition to optimize it. After optimizing each part of the problem, just combine them and do search, and the running time would sharply decrease. Based on the experiments' result, we found that CSP is really a perfect method in artificial intelligence, not only useful but also interesting.

Reference

1. What is Cryptarithmetic?

   Retrieved from http://www.elitmuszone.com/elitmus/what-is-cryptarithmetic/

   Line 2-3

2. What is Cryptarthimetic?

   Retrieved from http://www.elitmuszone.com/elitmus/what-is-cryptarithmetic/

   Line 8-10

3. Assignment2 and lecture slides