Solving Cryptarithmetic Puzzles by Logic Programming

Feng-Jen Yang
Department of Computer Science
Florida Polytechnic University
Lakeland, Florida, USA
fyang@floridapoly.edu

***Abstract*—As a personal interest of study, I tried a logic programming approach towards the problem solving of cryptarithmetic puzzles that are commonly discussed as a subcategory of constraint satisfaction problems in the literature of artificial intelligence. While there are possibly several methods capable of solving constraint satisfaction problems, I took into consideration the efficiency as well as the completeness that will identify all possible solutions under the specified constraints and exclude trivial and useless solutions from the perspective of real-life practice. In this paper, I demonstrated an approach that can be adapted to solve most of the constraint satisfaction problems especially within the context of cryptarithmatic puzzles. This method will also perform forward checking to have early backtracking and prevent searching the entire search tree exhaustively.**

***Index Terms*—Cryptarithmetic puzzle, Constraint Satisfaction Problem, Forward Checking, Early Backtracking**

I. Introduction

In the literature of Artificial Intelligence, cryptarithmetic puzzles are generally discussed as a kind of the Constraint Satisfaction Problems (CPSs) in which a solution to a given problem is represented by a problem state that meets of all the problem constraints. In the context of presenting a cryptarithmetic puzzle, the numerical values involved in an arithmetic computation are encrypted and represented not by numerical numbers, formed by digits from 0 to 9, but by encrypted alphabetical letters.

By concerning the real-life practice of data encryption and mathematical correctness, the solutions to a given puzzle have to comply with the following constraints:

- Each letter involved in the computation is representing a digit.
- No two letters are representing the same digit.
- After replacing each letter by its corresponding digit, the resultant value is mathematically correct.

As an instance of cryptographical problems, let's consider the following example:

```
  WAN
+ LAN
------
  BOB
```

A possible solution to this problem is replacing A by 7, replacing B by 8, replacing L by 3, replacing N by 9, replacing O by 5, and replacing W by 4. As a result, this solution is viewed and verified as:

```
A=7, B=8, L=3, N=9, O=5, W=4

  WAN              479
+ LAN            + 379
------    ===>   ------
  BOB              858
```

In this paper, I applied a logical programming approach to search for all possible solutions to the above instance of cryptarithmetic puzzles. While most of the constraint satisfaction problems can be solved by brute force methods, such as generate and test, this inefficient approach is usually not preferred in the society artificial intelligence. To prevent exhaustively searching the entire search tree, a forward checking method is incorporated to have early backtracking.

It is also worthy of mentioning that the problem constraints are purposefully specified to exclude the trivial solution that is simply replacing all letters by 0's such as:

```
  000
+ 000
------
  000
```

From the standing point of cryptography, this trivial encryption and decryption are useless and not of much practical interest in real-lie message security.

II. Some Preliminary Analysis of Possible Methodologies

Based on the general approaches of solving constraint satisfaction problems, the following analysis discussed two possible methods that are applicable of solving the aforementioned instance of puzzle.

A. The Brute Force Methodology

Let's start with solving a cryptarithmetic puzzle by using the brute force methods. This is by far the most intuitive but not so intelligent method. While it is not a heuristically informed method and often criticized by its inefficiency, the generate and test method is a very fundamental brute force method that is capable of solving problems in which solutions can be found within a given problem's search space. With no exception, this method is capable enough of solving any cryptarithmetic puzzle without concerning its inefficiency. The very core of a brute force problem-solving method is exhaustively searching through the entire search space that consists of all possible

978-1-7281-7624-6/20/$31.00 ©2020 IEEE
DOI 10.1109/CSCI51800.2020.00258

problem states and screen out those states that are not meeting all problem constraints.

Based on the instance that has been illustrated in this paper, the entire search space can be represented by the following search tree that is partially represented in Figure 1. Within this search tree, every path from the root to a leaf node is representing a possible problem state that consists of possible encryption consists of replacing every letter by the digit suggested along the path. Also, for the purpose of following the mathematical convention that starts the computation of addition from the 1's place and then goes on to the 10's place, 100's place, and so on. This search tree starts form possible encryption of the root N, then B, A, O, W, and finally ends at L.

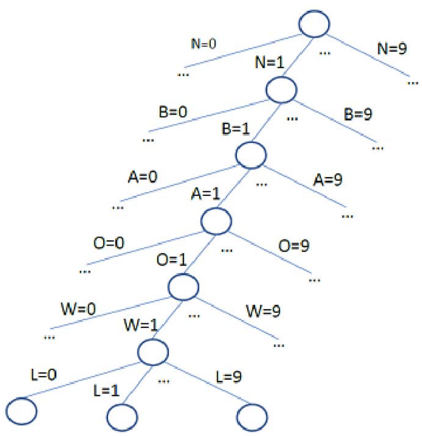

Fig. 1. Part of the Search Tree

The complete search tree comes with a height of 6 and a branching factor of 10. So, the entire search space has 1,000,000 paths. For the purpose of identifying all possible solutions, this brute force method can verify the correctness of each path by replacing letters with digits that are suggested along the path and screen out paths that are not meeting all the problem constraints.

Although this may appear to be a valid approach to solve the problem, it is simply not efficient and involves the examination of unnecessary replacements of letters by digits.

B. Applying Forward Checking Methods

While the brute force search is valid and will not miss any possible solution, it does involve many unnecessary overheads on the way of examining each path. This overhead can be avoided by applying the idea of forward checking to have early backtracking before processing the entire path. As soon as any of the letter replacement is found to be violating any problem constraint the rest of the path can be abandoned without any further examination. As an illustration of this forward checking, let's consider the path consists of N=1, B=1, A=2, O=3, W=4, and L=5. Right after seeing that B is repeating the digit 1, we notice the violation of "no two letters are representing the same digit." So, the rest of the path can be skipped from any further examination and the search will go on to the verification of the next path

A comprehensive verification of the conformance with problem constraints can be described in the following sequence:

By examining the computation of 1's place, we know that:

- N can be replaced by any digit.
- B can be replaced by a digit that is not used to replace N.
- The remainder of (N + N) ÷ 10 must be equal to B.

By examining the computation from 1's place to 10's place, we know that:

- N can be replaced by any digit.
- B can be replaced by a digit that is not used to replace N.
- A can be replaced by a digit that is not used to replace N and B.
- O can be replaced by a digit that is not used to replace N, B and A.
- The remainder of (N + N + (A × 10) + (A × 10)) ÷ 100 must be equal to (B + (O × 10)).

By examining the computation from 1's place to 100's place, we know that:

- N can be replaced by any digit.
- B can be replaced by a digit that is not used to replace N.
- A can be replaced by a digit that is not used to replace N and B.
- O can be replaced by a digit that is not used to replace N, B and A.
- W can be replaced by a digit that is not used to replace N, B, A and O.
- L can be replaced by a digit that is not used to replace N, B, A, O and W.
- Since there is no carrying from the 100's place to the 1000's place, (N + N + (A × 10) + (A × 10) + (W × 100) + (L × 100)) must be equal to (B + (O × 10) + (B × 100)).

III. The Implementation in CLIPS Programming Language

While there are many programming languages that can be adopted to implement the aforementioned problem-solving method, by concerning the advantage of having a built-in inference engine, I focused my selection on languages in the paradigm of logic programming and adopted the CLIPS programming language [1] [2] [3].

The initial fact-base comes with all letters and digits within the problem domain. In CLIPS syntax they are defined as a group of initial facts as follows:

```
(deffacts letters-and-digits
 (letter A)
 (letter B)
 (letter L)
 (letter N)
 (letter O)
 (letter W)
 (digit 0)
```

```
  (digit 1)
  (digit 2)
  (digit 3)
  (digit 4)
  (digit 5)
  (digit 6)
  (digit 7)
  (digit 8)
  (digit 9))
```

In addition to the initial contents, all possible replacements of letters by digits are added to the fact-base by the following knowledge rule:

```
(defrule all-replacements
  (letter ?letter)
  (digit ?digit)
  =>
  (assert (replace ?letter ?digit)))
```

The real forward checking is implemented by the following knowledge rule:

```
(defrule forward-checking

  ;checking the 1's place
  (replace N ?n)
  (replace B ?b&~?n)
  (test (= (mod (+ ?n ?n) 10) ?b))

  ;checking up to 10's place
  (replace A ?a&~?n&~?b)
  (replace O ?o&~?n&~?b&~?a)
  (test (= (mod (+ ?n ?n (* ?a 10) (* ?a 10)) 100)
           (+ (* ?o 10) ?b)))

  ;checking up to 100's place
  (replace W ?w&~?n&~?b&~?a&~?o)
  (replace L ?l&~?n&~?b&~?a&~?o&~?w)
  (test (= (+ ?n ?n (* ?a 10) (* ?a 10)  (* ?w 100)
           (* ?l 100))
           (+ (* ?b 100) (* ?o 10) ?b)))

  =>

  ;display all solutions
  (printout t "A=" ?a ", B=" ?b ", L=" ?l ", N=" ?n
            ", O=" ?o ", W=" ?w crlf cefl)
  (printout t "     WAN               " ?w ?a ?n
            crlf)
  (printout t "   + LAN             + " ?l ?a ?n
            crlf)
  (printout t "   ------    ===>    ------" crlf)
  (printout t "     BOB               " ?b ?o ?b
            crlf crlf) )
```

IV. The List of All possible solutions

All possible solutions to the given cryptarithemetic problem are listed as follows:

```
A=7, B=8, L=3, N=9, O=5, W=4

    WAN               479
  + LAN             + 379
  ------    ===>    ------
    BOB               858

A=7, B=8, L=4, N=9, O=5, W=3

    WAN               379
  + LAN             + 479
  ------    ===>    ------
    BOB               858

A=3, B=6, L=2, N=8, O=7, W=4

    WAN               438
  + LAN             + 238
  ------    ===>    ------
    BOB               676

A=3, B=8, L=2, N=9, O=7, W=6

    WAN               639
  + LAN             + 239
  ------    ===>    ------
    BOB               878

A=6, B=8, L=2, N=9, O=3, W=5

    WAN               569
  + LAN             + 269
  ------    ===>    ------
    BOB               838

A=7, B=6, L=2, N=8, O=5, W=3

    WAN               378
  + LAN             + 278
  ------    ===>    ------
    BOB               656

A=7, B=6, L=3, N=8, O=5, W=2

    WAN               278
  + LAN             + 378
  ------    ===>    ------
    BOB               656

A=3, B=6, L=4, N=8, O=7, W=2

    WAN               238
  + LAN             + 438
  ------    ===>    ------
    BOB               676

A=3, B=8, L=6, N=9, O=7, W=2

    WAN               239
  + LAN             + 639
  ------    ===>    ------
    BOB               878

A=6, B=8, L=5, N=9, O=3, W=2

    WAN               269
  + LAN             + 569
  ------    ===>    ------
    BOB               838

A=1, B=8, L=6, N=9, O=3, W=2

    WAN               219
  + LAN             + 619
  ------    ===>    ------
    BOB               838

A=1, B=8, L=2, N=9, O=3, W=6

    WAN               619
  + LAN             + 219
  ------    ===>    ------
    BOB               838

A=1, B=6, L=4, N=8, O=3, W=2

    WAN               218
```

```
   + LAN            + 418
   ------   ===>    ------
     BOB              636

A=1, B=6, L=2, N=8, O=3, W=4

     WAN              418
   + LAN            + 218
   ------   ===>    ------
     BOB              636

A=1, B=8, L=5, N=4, O=2, W=3

     WAN              314
   + LAN            + 514
   ------   ===>    ------
     BOB              828

A=1, B=8, L=3, N=4, O=2, W=5

     WAN              514
   + LAN            + 314
   ------   ===>    ------
     BOB              828

A=7, B=8, L=1, N=9, O=5, W=6

     WAN              679
   + LAN            + 179
   ------   ===>    ------
     BOB              858

A=4, B=6, L=1, N=8, O=9, W=5

     WAN              548
   + LAN            + 148
   ------   ===>    ------
     BOB              696

A=7, B=6, L=1, N=8, O=5, W=4

     WAN              478
   + LAN            + 178
   ------   ===>    ------
     BOB              656

A=3, B=8, L=1, N=4, O=6, W=7

     WAN              734
   + LAN            + 134
   ------   ===>    ------
     BOB              868

A=3, B=6, L=1, N=8, O=7, W=5

     WAN              538
   + LAN            + 138
   ------   ===>    ------
     BOB              676

A=9, B=6, L=1, N=3, O=8, W=4

     WAN              493
   + LAN            + 193
   ------   ===>    ------
     BOB              686

A=4, B=6, L=1, N=3, O=8, W=5

     WAN              543
   + LAN            + 143
   ------   ===>    ------
     BOB              686

A=2, B=6, L=1, N=3, O=4, W=5

     WAN              523
   + LAN            + 123
   ------   ===>    ------
     BOB              646

A=2, B=4, L=1, N=7, O=5, W=3

     WAN              327
   + LAN            + 127
   ------   ===>    ------
     BOB              454

A=2, B=8, L=1, N=9, O=5, W=7

     WAN              729
   + LAN            + 129
   ------   ===>    ------
     BOB              858

A=6, B=4, L=1, N=7, O=3, W=2

     WAN              267
   + LAN            + 167
   ------   ===>    ------
     BOB              434

A=5, B=6, L=3, N=8, O=1, W=2

     WAN              258
   + LAN            + 358
   ------   ===>    ------
     BOB              616

A=5, B=6, L=2, N=8, O=1, W=3

     WAN              358
   + LAN            + 258
   ------   ===>    ------
     BOB              616

A=5, B=8, L=4, N=9, O=1, W=3

     WAN              359
   + LAN            + 459
   ------   ===>    ------
     BOB              818

A=5, B=8, L=3, N=9, O=1, W=4

     WAN              459
   + LAN            + 359
   ------   ===>    ------
     BOB              818

A=7, B=6, L=4, N=8, O=5, W=1

     WAN              178
   + LAN            + 478
   ------   ===>    ------
     BOB              656

A=7, B=8, L=6, N=9, O=5, W=1

     WAN              179
   + LAN            + 679
   ------   ===>    ------
     BOB              858

A=4, B=6, L=5, N=8, O=9, W=1

     WAN              148
   + LAN            + 548
```

```
   ------    ===>    ------
     BOB              696

A=3, B=8, L=7, N=4, O=6, W=1

     WAN              134
   + LAN            + 734
   ------    ===>    ------
     BOB              868

A=3, B=6, L=5, N=8, O=7, W=1

     WAN              138
   + LAN            + 538
   ------    ===>    ------
     BOB              676

A=9, B=6, L=4, N=3, O=8, W=1

     WAN              193
   + LAN            + 493
   ------    ===>    ------
     BOB              686

A=4, B=6, L=5, N=3, O=8, W=1

     WAN              143
   + LAN            + 543
   ------    ===>    ------
     BOB              686

A=6, B=4, L=2, N=7, O=3, W=1

     WAN              167
   + LAN            + 267
   ------    ===>    ------
     BOB              434

A=2, B=6, L=5, N=3, O=4, W=1

     WAN              123
   + LAN            + 523
   ------    ===>    ------
     BOB              646

A=2, B=4, L=3, N=7, O=5, W=1

     WAN              127
   + LAN            + 327
   ------    ===>    ------
     BOB              454

A=2, B=8, L=7, N=9, O=5, W=1

     WAN              129
   + LAN            + 729
   ------    ===>    ------
     BOB              858

A=0, B=8, L=6, N=9, O=1, W=2

     WAN              209
   + LAN            + 609
   ------    ===>    ------
     BOB              818

A=0, B=8, L=5, N=9, O=1, W=3

     WAN              309
   + LAN            + 509
   ------    ===>    ------
     BOB              818

A=0, B=8, L=3, N=9, O=1, W=5

     WAN              509
   + LAN            + 309
   ------    ===>    ------
     BOB              818

A=0, B=8, L=2, N=9, O=1, W=6

     WAN              609
   + LAN            + 209
   ------    ===>    ------
     BOB              818

A=0, B=6, L=4, N=8, O=1, W=2

     WAN              208
   + LAN            + 408
   ------    ===>    ------
     BOB              616

A=0, B=6, L=2, N=8, O=1, W=4

     WAN              408
   + LAN            + 208
   ------    ===>    ------
     BOB              616

A=9, B=6, L=0, N=3, O=8, W=5

     WAN              593
   + LAN            + 093
   ------    ===>    ------
     BOB              686

A=7, B=6, L=0, N=3, O=4, W=5

     WAN              573
   + LAN            + 073
   ------    ===>    ------
     BOB              646

A=6, B=8, L=0, N=9, O=3, W=7

     WAN              769
   + LAN            + 069
   ------    ===>    ------
     BOB              838

A=9, B=4, L=0, N=2, O=8, W=3

     WAN              392
   + LAN            + 092
   ------    ===>    ------
     BOB              484

A=8, B=4, L=0, N=2, O=6, W=3

     WAN              382
   + LAN            + 082
   ------    ===>    ------
     BOB              464

A=6, B=8, L=0, N=4, O=2, W=7

     WAN              764
   + LAN            + 064
   ------    ===>    ------
     BOB              828

A=5, B=4, L=0, N=7, O=1, W=3

     WAN              357
   + LAN            + 057
   ------    ===>    ------
```

```
     BOB               414

A=5, B=8, L=0, N=9, O=1, W=7

     WAN               759
   + LAN             + 059
   ------    ===>    ------
     BOB               818

A=7, B=2, L=0, N=6, O=5, W=1

     WAN               176
   + LAN             + 076
   ------    ===>    ------
     BOB               252

A=8, B=2, L=0, N=6, O=7, W=1

     WAN               186
   + LAN             + 086
   ------    ===>    ------
     BOB               272

A=5, B=8, L=6, N=4, O=0, W=1

     WAN               154
   + LAN             + 654
   ------    ===>    ------
     BOB               808

A=5, B=8, L=1, N=4, O=0, W=6

     WAN               654
   + LAN             + 154
   ------    ===>    ------
     BOB               808

A=5, B=6, L=4, N=3, O=0, W=1

     WAN               153
   + LAN             + 453
   ------    ===>    ------
     BOB               606

A=5, B=6, L=1, N=3, O=0, W=4

     WAN               453
   + LAN             + 153
   ------    ===>    ------
     BOB               606

A=9, B=6, L=5, N=3, O=8, W=0

     WAN               093
   + LAN             + 593
   ------    ===>    ------
     BOB               686

A=7, B=6, L=5, N=3, O=4, W=0

     WAN               073
   + LAN             + 573
   ------    ===>    ------
     BOB               646

A=6, B=8, L=7, N=9, O=3, W=0

     WAN               069
   + LAN             + 769
   ------    ===>    ------
     BOB               838

A=8, B=2, L=1, N=6, O=7, W=0

     WAN               086
   + LAN             + 186
   ------    ===>    ------
     BOB               272

A=7, B=2, L=1, N=6, O=5, W=0

     WAN               076
   + LAN             + 176
   ------    ===>    ------
     BOB               252

A=9, B=4, L=3, N=2, O=8, W=0

     WAN               092
   + LAN             + 392
   ------    ===>    ------
     BOB               484

A=8, B=4, L=3, N=2, O=6, W=0

     WAN               082
   + LAN             + 382
   ------    ===>    ------
     BOB               464

A=6, B=8, L=7, N=4, O=2, W=0

     WAN               064
   + LAN             + 764
   ------    ===>    ------
     BOB               828

A=5, B=4, L=3, N=7, O=1, W=0

     WAN               057
   + LAN             + 357
   ------    ===>    ------
     BOB               414

A=5, B=8, L=7, N=9, O=1, W=0

     WAN               059
   + LAN             + 759
   ------    ===>    ------
     BOB               818
```

V. SUMMARY

Constraint satisfaction problems have been challenging problem-solvers for a long time for the sake of mental exercises. This paper demonstrated a logical programming approach that can be generalized to solve all cryptarithemetic puzzles. While this paper is not creating any new theory or methodology, it can be adopted as case study in artificial intelligence-related.

REFERENCES

[1] I. Bratko, "Prolog programming for artificial intelligence," 4th Ed., Addison Wesley, 2012.

[2] F. Yang, "Solving the Water Jug Puzzle in CLIPS," Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2018, San Francisco, CA, pp. 564-567, 2018.

[3] F. Yang, "A Logic Programming Solution to the Water Jug Puzzle," Transactions on Engineering Technologies: World Congress on Engineering and Computer Science 2018, pp. 66-73, 2020.