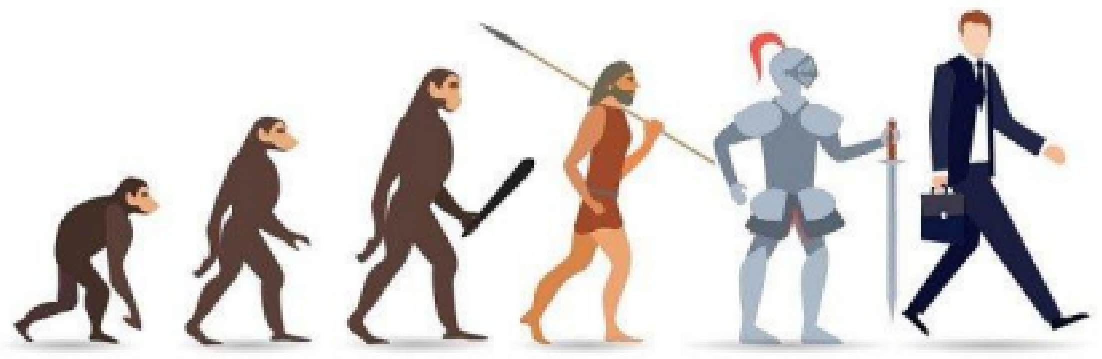
CSC14003 – Introduction to Artificial Intelligence

Cryptarithmetic Problem

MSSV	Name	Role	Contribution
19127346	Nguyễn Nhật Cường	Leader	40%
19127382	Đinh Hải Giang	Member	20%
19127567	Võ Trần Quang Thông	Member	25%
19127517	Hồ Thiên Phước	Member	15%

The degree of completion level for project: *100%*
Maximum runtime of the algorithm: *5 minutes*

An Evolutionary Algorithm to Solve Cryptarithmetic Problem



I. CRYPTARITHMETIC PROBLEM:

- Cryptarithmetic problem is an interesting constraint satisfaction problem for which different algorithms have been given. But none of them can solve the problem intelligently using small space and limited time. The paper illustrates the design and implementation of an efficient evolutionary algorithm that can find the solution more quickly taking less memory.
- Cryptarithm is a genre of mathematical puzzle in which the digits are replaced by letters of the alphabet or other symbols. Cryptarithmetic is the science and art of creating and solving cryptarithms.

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

II. STUDY OF THE PROBLEM:

- The constraints of defining a Cryptarithmetic problem are as follows:
  + Each letter or symbol represents only one and a unique digit throughout the problem.
  + When the digits replace letters or symbols, the resultant arithmetical operation must be correct.

- These two constraints lead to some other restrictions in the problem. Consider that, the base of the numbers is 10. Then there must be at most 10 unique symbols or letters in the problem. Otherwise, it would not be possible to assign a unique digit to each unique letter or symbol in the problem. To be semantically meaningful, a number must not begin with a zero. So, the letters at the beginning of each number should not correspond to zero.

III. FINDING SOLUTION BY SEARCHING:

- A blind search can eventually find the solutions, if there is any, in a bounded time. Given that the base of the number is 10, there may be ${}^{10}p_n$ solutions to be checked in the problem space; where n is the number of unique letters or symbols in the problem. Each distinctive letter is given in the SEND+MORE=MONEY, blind search might have to search at most ${}^{10}p_8$=1814400 solutions. If the searching algorithm can check 1000 solutions per second, yet it will take about 30 minutes to get the solution in the worst case.
- The toughness of the problem can be interpreted as less number of valid solutions in relatively huge problem space. The precedence of mathematical operation in the problem indicates a basic estimation of toughness. Again, if there are few letters in the problem, the problem space becomes smaller which makes the problem easy. Given the number of letters in the problem, a problem can be very tough yet. This is because some of the letters or symbols can occur more than once in the problem. These critical letters, the letters that occur more than once in the problem can introduce more restrictions in the problem and reduce the number of valid solutions.

⇒ An Evolutionary Algorithm will search for solutions in the shortest time but the performance will also reflect the toughness of the problem.

IV. EVOLUTION APPROACH:

- An Evolutionary Algorithm (EA) is a common term for algorithms that utilize the adaptive behavior modeled after principles of nature.
- Although the definition of Evolutionary Algorithm differs, the more common properties of EAs are that collections of potential solutions to the problem at hand are maintained. These solutions are referred to as the

population of a current generation. Each potential solution is called a chromosome. Operations are applied to the current population to produce a new generation that will hopefully contain chromosomes that are better solutions of the problem. This process continues until some threshold value or stopping criterion is met.

- The new population is produced through the operators on selected chromosomes of current generation. Typically, the chromosomes of the current generation, to whom the operators are applied, are chosen based on their quality. In this way, it is more likely that the offspring chromosomes inherit desirable characteristics of its parents. Some heuristics or fitness functions are used to choose parent chromosomes in a generation.
- Typically operators are developed from the genetic notation of mutation, recombination (or crossover) and inversion. As an example, a mutation operator can choose two random numbers that may be applied to randomly change some inherited characteristics of the chromosome. Crossover operations involve combining characteristics from two or more parents and placing them in the resulting offspring.

V. FORMULATION OF ALGORITHM:

A. Definition of objects:

[1]. string ⇒ The problem given

Example: TWO+TWO=FOUR
SEND+MORE=MONEY
SIX+SEVEN+SEVEN=TWENTY
SEND+(MORE+MONEY)-OR+DIE=NUOYI

[2]. stringList ⇒ A list of strings is separated from [1]

Rules:

+ The string that acts as a positive number will be separated into an independent element in the list.
+ The string acts as the exception or final factor in multiplication, meaning that the previous character is '-' or '*', will be separated with the character corresponding to it.
+ If the string is located with the character '(' or character ')' will be separated with that character.

Example: A+(B-C)*D*(H+M-F)*HY=RE
⇒ stringList: ['A', '(B', '-C)', '*D', '*(H', 'M', '-F)', '*HY', 'RE']

[3]. *clas*s node ⇒ Used to save independent characters in the problem

Include:

+ _char: an upper character in the alphabet.
+ _leader: if the character is the beginning of the string, _leader will be True and vice versa.

B. Encoding solution to chromosome:

- The initial part of solving a problem in Evolutionary Algorithm is the encoding of solutions into chromosomes. The chromosomes should be simple enough so that operations can be executed easily and descriptive enough to characterize a unique solution. In this problem, letters should be assigned unique digits to characterize a solution and they should be shown as a list with elements as a class node (nodeList) to easily check the constraints of the problem. The elements in nodelist will be randomly assigned with a value from 0 to 9 and must satisfy constraints.

$$\begin{array}{c}\begin{pmatrix} D & E & M & N & O & S & Y & \_ & \_ & \_ \\ F & F & T & F & F & T & F & F & F & F \end{pmatrix} \\ \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix}\end{array}$$

C. Mutation Operation:

- A mutation operator will randomly generate two different numbers from 0 to 9 and will exchange the positions of the letters in these two indexes. This will correspond to a completely different solution.

$$\begin{array}{c}\begin{pmatrix} D & E & M & N & O & S & Y & \_ & \_ & \_ \\ F & F & T & F & F & T & F & F & F & F \end{pmatrix} \\ \begin{matrix} 0 & 1 & 2 & 6 & 4 & 5 & 3 & 7 & 8 & 9 \end{matrix}\end{array}$$

(a)

$$\begin{array}{c}\begin{pmatrix} D & E & M & N & O & S & Y & \_ & \_ & \_ \\ F & F & T & F & F & T & F & F & F & F \end{pmatrix} \\ \begin{matrix} 4 & 1 & 2 & 3 & 0 & 5 & 6 & 7 & 8 & 9 \end{matrix}\end{array}$$

(b)

...

- Here the only restriction is that the letter that comes at the starting of a number in the problem should not be the starting letter of the chromosome. In the above problem, S and M cannot be 0. This is because they are the starting numbers in the problem. A more productive mutation operator ensures that it will not exchange the position of two don't care symbols with each other.
- Similar cross-over operation can be designed where two chromosomes exchange their attributes. This can be useful in moving faster towards the solution especially at the middle of searching when selected chromosomes adopt some of the good attributes. But for some tough problems it makes the algorithm more susceptible to local minima deadlock. So, ***cross-over operation may not be included in a general algorithm***.

D. Fitness Function:

- A Fitness function usually indicates the degree of correctness of a chromosome. An evaluation function can be easily formulated which will calculate the error of the mathematical result in the problem.

```
      S E N D
      9 7 1 6
    + M O R E
      3 5 2 7
  -------------
    M O N E Y
    3 5 1 7 0
ERROR = ABS (35170-(9716+3527)) = 21927
```

- Now this can be thought of as negative fitness. Chromosome with minimum error is the fittest chromosome in a generation. This algorithm aims to minimize this value. When the value is zero, the solution is found.
- Assuming the left of '=' is LEFT and the right is RIGHT. General formula will be

```
ERROR = ABS (RIGHT-LEFT)
```

E. Improving Generation through Iteration:

- Initially we have only one chromosome that is shown in B. We will first use this chromosome to generate the next generation. By mutation operation this chromosome generates a new generation, which represents the current population. Then we will take some fittest chromosomes from the current generation to produce the next generation (Generation with smaller fitness). In an iterative process, the fittest chromosomes of a generation get the chance to generate the offspring generation.

VI. ALGORITHM:

Step 1: Scan the input strings.

Step 2: Put the letters or symbols in ARRAY[10]

Step 3: If the number of distinct letters is less than 10 then fill the rest of the indices of ARRAY by '_' and create nodeList[10].

Step 4: Randomly generated current generation (ancestor).

Step 5: Generate two random numbers m, n and swap the contents of index m and n of any one chromosome of current generation.

Step 6: Evaluate the fitness of the generation in step 4, if smaller than the fitness of the current generation, that becomes our current generation . If there is no chromosome with error 0 in the current generation then go to step 4. If the chromosome is found with error 0 then report the solution and exit.

VII. PERFORMANCE:

A. Level 1:

1. SAT+MHI=LAMP:

Output: + first time: 43718265

+ second time: 50816294

+ third time: 26819537

Average: 34483 times

Average time: 745ms

Blind search requires about ${}^{10}p_8$ / 2 = 907200 times

2. SENDE-DIA=NUOYI:

Output: + first time: 250837496

+ second time: 360714298

+ third time: 250837496

Average: 346376 times

Average time: 9.1s

Blind search requires about ${}^{10}p_9$ / 2 = 1814400 times

3. SEND+NULL=MONRE:

Output: + first time: 967150348

+ second time: 374152968
+ third time: 572160839

Average: 173424 times

Average time: 3.1s

Blind search requires about ${}^{10}p_9 / 2 = 1814400$ times

4. SEND+MORE=MONEY:

Output: + first time: 75160892
+ second time: 75160892
+ third time: 75160892

Average: 85646 times

Average time: 1.9s

Blind search requires about ${}^{10}p_8 / 2 = 907200$ times

5. TWO+TWO=FOUR:

Output: + first time: 150736
+ second time: 148763
+ third time: 162873

Average: 10439 times

Average time: 321ms

Blind search requires about ${}^{10}p_6 / 2 = 75600$ times

B. Level 2:

1. AFKE-HAKA-FAKE-EFD-GAF-FQ-AT=ATH:

Output: + first time: 653281497
+ second time: 957362410
+ third time: 648231507

Average: 50752 times

Average time: 6.7s

Blind search requires about ${}^{10}p_9 / 2 = 1814400$ times

2. HAND+SOME+KAKA+SAND+HOME+KOME +SOKA-HEMO+DOHA+MEDO=KEDOS:

Output: + first time: 590247683
+ second time:734851069
+ third time: 958426371

Average: 119447

Average time: 17.5s

Blind search requires about ${}^{10}p_9$ / 2 = 1814400 times

3. **SO+MANY+MORE+MEN+SEEM+TO+SAY+THAT+THEY+MAY +SOON+TRY+TO+STAY+AT+HOME+SO+AS+TO+SEE+OR+ HEAR+THE+SAME+ONE+MAN+TRY+TO+MEET+THE+TEAM +ON+THE+MOON+AT+HE+HAS+AT+THE+OTHER+TEN=TESTS:**

Output: + first time: 7052618394
+ second time: 7052618394
+ third time: 7052618394

Average: 236831 times

Average time: 37.81s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

4. **YOU+HAVE+A+HAND+AND+DNAH+A+EVAH+UOY+HAHA +HEHE+HOHO+HUHU+YOYO+YEYE+YAYA+YOYO=UDHIO:**

Output: + first time: 2534186709
+ second time: 6183290574
+ third time: 6234801759

Average: 45025 times

Average time: 16.75s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

5. **FALL+FAIL+FINN+FATHER+FAR+FIR+TAIL+TER+FAIR +NAIL+RAR+RAIL+TALL+FILL+FELL+LETTER+THEIR+ TILL+TELL+TELLER+TALLER+LAL+LIL+LEE+NEAL+NAIL +LALALA+LELELE+TELE+TILI+FA+FARLER=HIINREE:**

Output: + first time: 123456789
+ second time: 123456789
+ third time: 123456789

Average: 89638 times

Average time: 22.5s

Blind search requires about ${}^{10}p_9$ / 2 = 1814400 times

C. Level 3:

1. **(SEND+NULL)-(KE+NU)+SE=MONRE:**

Output: + first time: 5374180926
+ second time: 2346197085
+ third time: 8364120579

Average: 78274 times

Average time: 2.85s

Blind search requires about $^{10}p_{10}$ / 2 = 1814400 times

2. **BANANA+PYJAMAS+(PEN-PAN+PJN)-(JEN+JAN+JANNA)+BENANBAN+BANES+NANA-(YAN+YEN-YJN+YES+YAS)+(PES-PASS+PANAPYN)+SAM+NANA-SEMANE-MYMY+MAMA+PANAMA-PENEME+PYNY+MESSE=MAYYSMBE:**

Output: + first time: 129345678
+ second time: 129345678
+ third time: 129345678

Average: 48858 times

Average time: 17.56s

Blind search requires about $^{10}p_{9}$ / 2 = 1814400 times

3. **KHAN+KHIN-(HIN+HAN-KAN)+(KIN+NEKE-NAKA)-KAKI+HAHI-NIKI=KAKI:**

Output: + first time: 592063
+ second time: 592063
+ third time: 592063

Average: 78020 times

Average time: 11.6s

Blind search requires about $^{10}p_{6}$ / 2 = 75600 times

4. **(JAN-JEN+QON)-(QAN+GEN-JON)+GEM-GIM+(GON-GOM+GAM-JAM)=JOM:**

Output: + first time: 925048631
+ second time: 736815024
+ third time: 187046293

Average: 726 times

Average time: 282 ms

Blind search requires about $^{10}p_{9}$ / 2 = 1814400 times

5. **MARA+DONA-(MESSI-RANDOM-MEME)+RONAL+DO-MAI+DORA+(MOMO-MISI+SIM-SIN)-(MAN-DESIN+LOL-MONAL)=SDODIM:**

Output: + first time: 5609871423
+ second time: 5879612034
+ third time: 1396240578

Average: 92030 times

Average time: 16.7s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

D. Level 4:

1. **JK+AH*(JQY+FHW)*MQ+FY-AJ*(HF+KW*QM)-AKW*FJ*(MW+FK*F)+MFQ-KY*WA-AH*FJF=HAHWFW:**

Output: + first time: 928356174
+ second time: 928356174
+ third time: 928356174

Average: 1028768 times

Average time: 11.6s

Blind search requires about ${}^{10}p_{9}$ / 2 = 1814400 times

2. **BEST-YASUO+LEE+AYA*(LEABLA-SE+BTU*UY)-LEY*(BUET+USU*ES)-ES*LSY*(BUT+US*OSA*B)-AYS*BT*(TSA+EB*YS)-BTS*BET*(BU+EB*L)-LY*LTSY*O-UTTSLAS=OASIS:**

Output: + first time: 732964518
+ second time: 732964518
+ third time: 732964518

Average: 222090 times

Average time: 64.13s

Blind search requires about ${}^{10}p_{9}$ / 2 = 1814400 times

3. **(JAN-JEN+QON)*QUIN+GON*(QAN-JON)+GEM*GIM+(GON*GOM-GAM*JAM)=JANQUE:**

Output: + first time: 0421936758
+ second time: 0421936758
+ third time: 0421936758

Average: 870893 times

Average time: 182.86s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

4. **(BCD+CED*JI-GDC)*EF+HI*(GCD-BA)-(HIJ+FE)+DC*(CA+BCD)*(HC+FJ*GD*C)-(BJ+C*BD)*BAA=DFEDIAFC:**

Output: + first time: 0123456789

+ second time: 0123456789
+ third time: 0123456789

Average: 540226 times

Average time: 133.56s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

5. CB*(CA+BJ*BK-JB*K)-BF*GA+BW+(FKA*CB-BJ+AWK)+JBK-WK*BA+(GJK*ABC-FAB+L)=AAGFJG:

Output: + first time: 624397185
+ second time: 624397185
+ third time: 624397185

Average: 232731 times

Average time: 23.9s

Blind search requires about ${}^{10}p_{9}$ / 2 = 1814400 times

6. SEND*TA+(A-B*E)*(AD+MN)=MONEY:

Output: + first time: 9736405218
+ second time: 9567438210
+ third time: 8702963514

Average: 288086 times

Average time: 12.06s

Blind search requires about ${}^{10}p_{10}$ / 2 = 1814400 times

VIII. Statistics:

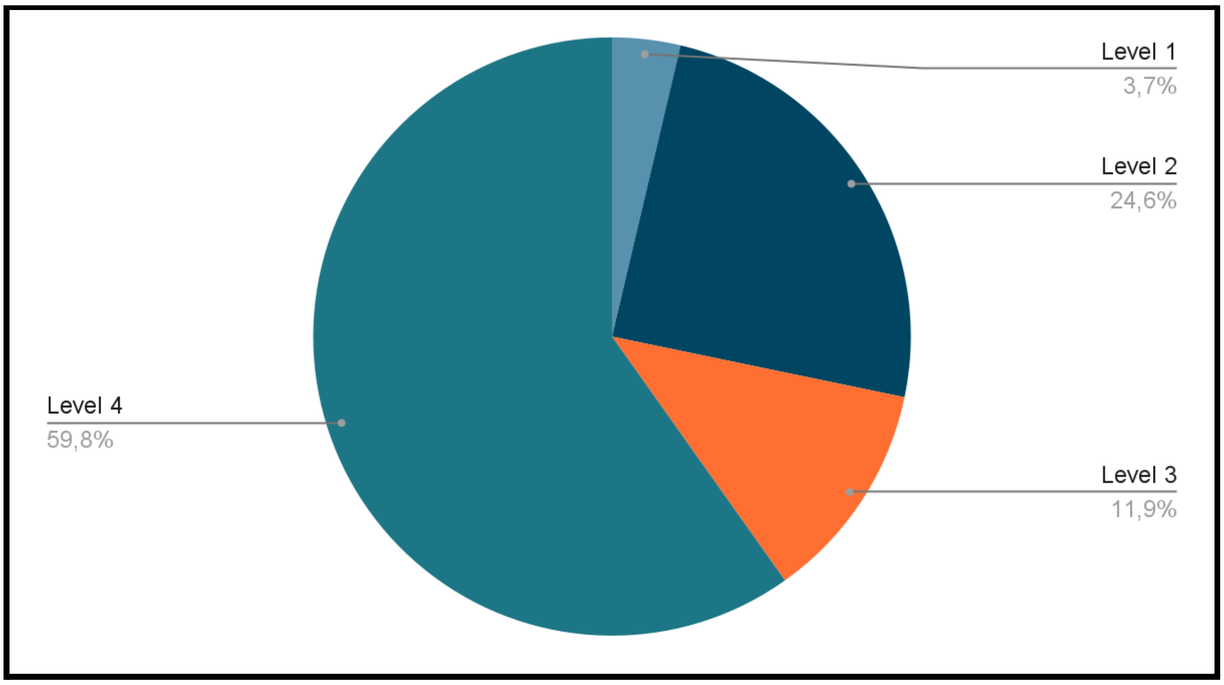


Figure 1. Runtime between levels.

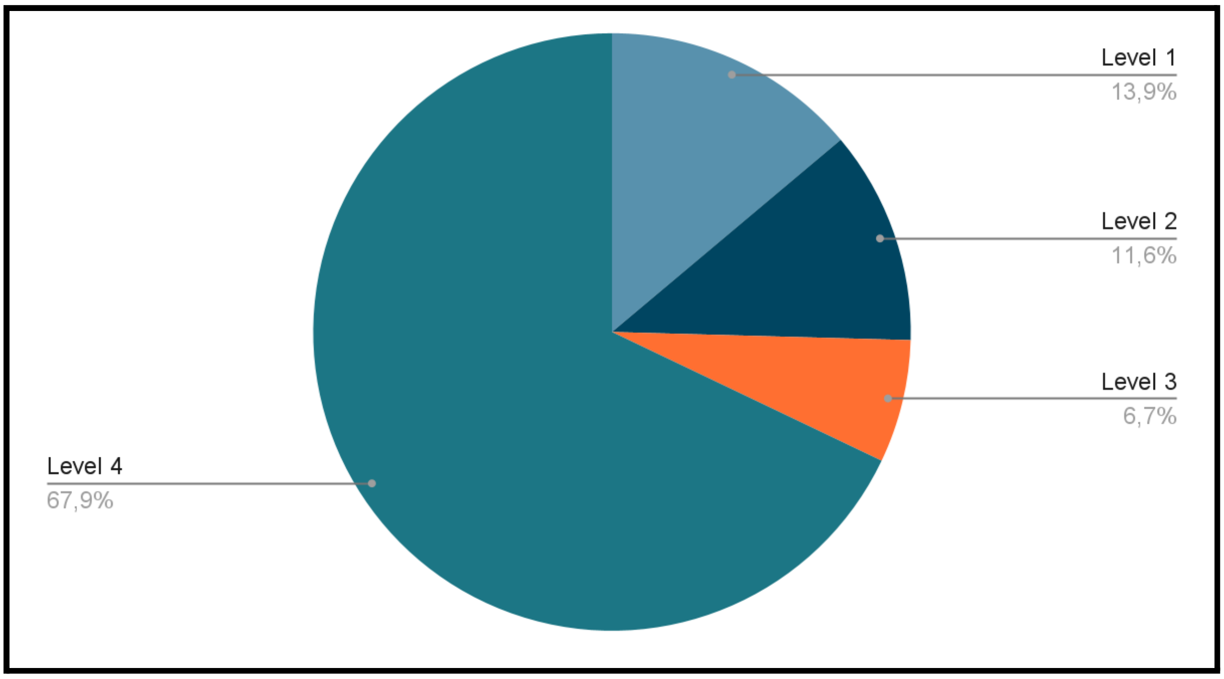


Figure 2. The generated generations between levels.

Conclusion:

- Because it depends on the complexity of the problem, the runtime may differ when arising with the same number of individuals.
- In simple problems, the number of generated generations can be much but time to solve is quite fast. On the contrary, in more complex problems, the runtime may be slower but the generated generations are less. Specific examples such as Level 1 and Level 2.

IX. Limitation:

- This is a greedy algorithm as it is taking the best chromosomes at each iteration and tries to converge to the nearest solution. Like other greedy approaches, it may stick at some local minima. To overcome these local minima we have used a random chromosome at each generation. This will guarantee us that we will be able to overcome the local minima and reach our global minima.

Environment

- ∇ Programming language: Python
- ∇ Version: 3.9.4
- ∇ IDE: Sublime Text

References

⚥ https://www.kdnuggets.com/2017/11/rapidminer-evolutionary-algorithms-feature-selection.html?fbclid=IwAR1tc0whvhFcy819U3DA9-R1A47EF5iBslCh-zEDqLTiBR0nvenbyC0i8OY

⚥ https://www.scribd.com/doc/73194984/An-Evolutionary-Algorithm-to-Solve-Crypt-Arithmetic-Problem

⚥ https://www.tutorialspoint.com/Solving-Cryptarithmetic-Puzzles