Genetic based Algorithm for N – Puzzle Problem

Harsh Bhasin
Computergrad.com
Faridabad, Haryana

Neha Singla
Student, YMCAUST
Faridabad, Haryana

ABSTRACT
N – Puzzle problem is an important problem in mathematics and has implications in Artificial Intelligence especially in gaming. The work presented reviews the previous attempts to solve this problem. A formal definition of the problem has been presented. The reason why it is considered as NP hard problem and why Genetic Algorithms (GAs) is applied has been explained. The work here by presents a GAs based algorithm to solve N – Puzzle problem. The algorithm has been analyzed and it is a sturdy belief that the presented algorithm has complexity better than most of the works studied. The work is a part of larger endeavor to solve all NP Hard problems by GAs.

Keywords
N – Puzzle problem, Genetic Algorithms, NP Hard, Solvability, Iterative Deepening.

1. INTRODUCTION
The work presented here forth introduces a solution of N – Puzzle problem using Genetic Algorithms (GAs). N – Puzzle problem is significant both in terms of its computational complexity and non – availability of a definite algorithm to solve a problem. The work presented is a part of larger endeavor to solve all the NP Hard problems using GAs. As a part of a larger project Travelling Salesman Person problem [1], Vertex Cover Problem [2], Post correspondence Problem [3], and Subset Sum Problem [4] have been solved. The present work concentrates on a problem which is fundamentally different from all the above problem, as it seems to have more affinity to the game solving approaches as compared to the problems stated above. The following sections concentrate on the Literature Review followed by the formal definition of the problem. Many Papers have been studied and analyzed. The reason why it is considered as NP Hard problem has been explained. The motivation of use of GAs to the above problem has been explained in the section on GAs. There are very many instances of the problem that cannot be solved. Such class has been discussed in a separate section. The solution presented is robust as far as theoretical considerations are concerned. A new algorithm has been proposed and explained in the second last section. It is our sturdy belief that new algorithm opens the door of genetic approach to an extremely interesting problem, whose solution still eludes the fraternity. The last section dwells on the future scope of the work.

2. LITERATURE REVIEW
An extensive literature review was carried out and many papers were analyzed. The papers were divided into two categories, those pertaining to GAs and heuristic search processes and other section related to N – Puzzle problem.

A valuable insight to the problem was obtained by the work of Sam Loyd [5]. In the work proposed by Korf, a new heuristic function was proposed depending upon the sub codes. According to the author the net cost is lesser but the concept requires higher order heuristics as well as pruning duplicate nodes, so the net cost is seemingly not as good as presented [6]. Another work relied on Manhattan pair distance heuristics which is a combination of Manhattan distance and pair distance. The search used is Iterative Deepening A*. The work is an extension of previous work [7].

Another work by Calabro provides $O(n^2)$ algorithm to decide the solvability and $O(n^3)$ moves to solve the problem. The paper primarily dwells on solvability part and not on the solution part [8]. The work by Pizlo develops a cognitive model and analyzes the human behavior by solving the problem. It is a cognitive Science model and not a computational model [9].

Some of the works like that of Felner uses pattern databases which has not been used in presented approach [10][11]. A distributed approach was presented by Drogoul by using an eco problem solving model. The strategy is novel but it is meant more for theoretical analysis [12].

3. N – PUZZLE PROBLEM
N - Puzzle problem consist of a m x m board with N numbered tiles and a blank space such that, $N = m^2 - 1$. Values of N can be 8 (3 x 3), 15 (4 x 4), 24 (5 x 5), 35 (6 x 6) and so on. A tile adjacent to a blank space can slide to a blank space thus making way for the further arrangements. The objective is to reach one of the goal state, an instance of which is shown in Figure 1.

The description of the problem consists of states, initial state, goal state, path cost and successor function. Formally,

$P = (Q, q_0, F, f, C)$

Where,

Q = Set of states

q_0 = Initial state

F = Final state. This can be one of the goal states already defined.

f = Function called successor function which generates the next state. This state can be described a move left, right, up or down.

C = Path cost. It is the number of steps in the path considering each move to be of unit cost.

This problem requires elicitation of transition rules and definite way of estimating the cost and making the heuristics. The moves in an N – puzzle problem generate a new configuration. The effect of move on the configuration is depicted in Figure 2.

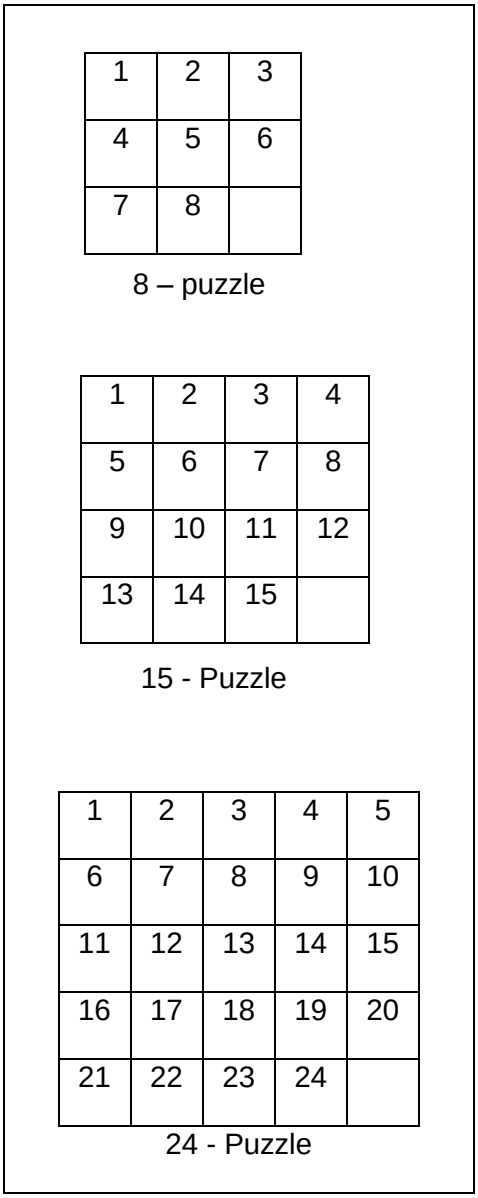

Fig 1. Goal State for 8 – puzzle, 15 – puzzle and 24 – puzzle problems.

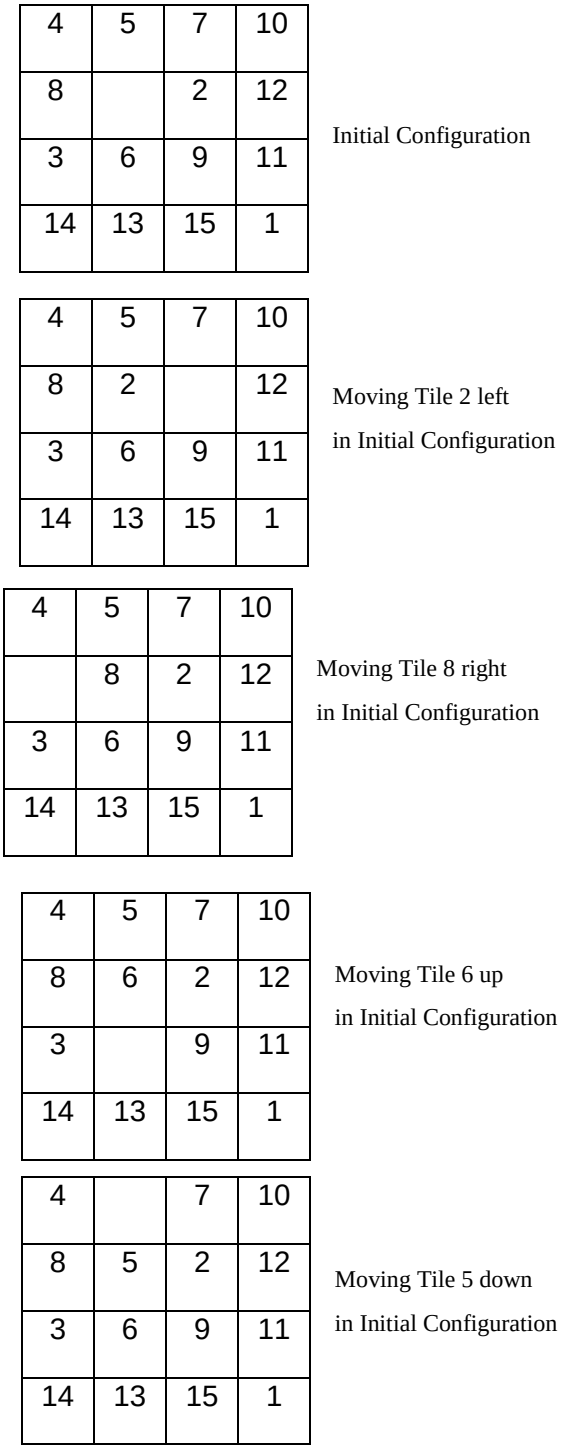

Fig 2. Depiction of moves in an N - puzzle problems.

3.1 8 – Puzzle Problem

The 8 puzzle problem consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. The problem is to change the initial state to goal state by sliding the tiles, one at a time, in minimum moves. One of the instances of the initial and the final state are depicted in Figure 3.

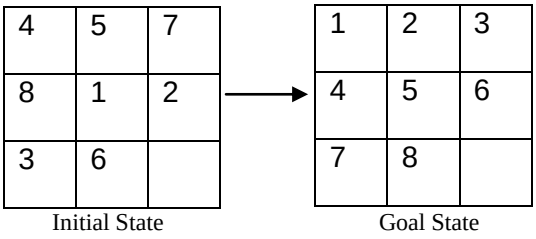

Fig 3: Instance of 8 – Puzzle Problem

3.2 15 – Puzzle Problem

The 15 puzzle problem consists of 15 squares numbered from 1 to 15 that are placed in a box leaving one position out of the 16 empty. The goal is to reposition the squares from a given arbitrary starting arrangement by sliding them one at a time into the final configuration. One of the instances of the initial and the final state are depicted in Figure 4.

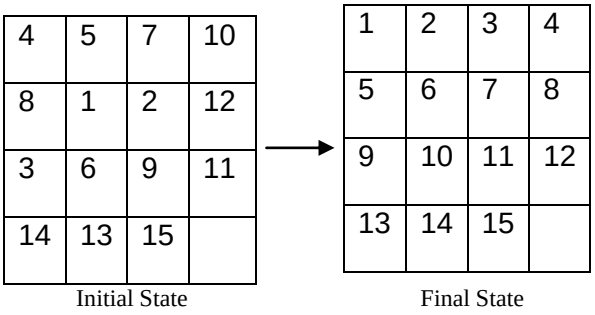

Fig 4: Instance of 15 – Puzzle Problem

3.3 Conventional Solution

There are various methods of solution of the above problem. Most easy to understand is brute force algorithm. Considering the fact that, there can be at maximum 3 slides, if the empty space is somewhere in between the board and minimum of 1, if the empty space is at corner of the board. If we construct a state space tree of the problem and use the concept of depth limited search, stopping at the n^{th} level, then the maximum complexity can be $3^{n+1} - 1$. Assume that the value of n at which we intend to stop is 18, then, the value of complexity factor comes out to be 1162261466. These many instructions if performed at the speed of 10^3 instructions per second, evaluates to 1162261 seconds, which is equivalent to 13.4 days in a normal desktop computer. In order to find out a better solution, suppose we decide to have 25 levels in the state space tree, then, by the above calculations it will take 26.8 years for a normal desktop computer to solve the problem.

The above statistics point to the fact that it is a NP Hard problem. The data collected proves the above fact. The above fact has also been proved by Kendall [13]. To solve a 3 x 3 problem, which is solvable, it requires .01 seconds to run an exhaustive search algorithm whereas for a 24 puzzle problem, the time required is 12 billion years.

One of the ways of handling the above problem is A* algorithm which takes into account, the cost travelled so far and the heuristic to reach the final destination. In most of the papers studied, Manhattan heuristic is taken as a heuristic function. The work proposed intense to solve the above problem by applying genetic algorithms as they are best suited for finding out the solution from amongst the large set of solution.

In the above statements, for every move, maximum 3 possible moves have been considered and not 4. The reason being, we will not move a tile which has already being moved in a previous step. So for every move, there can be at max 3 possible moves. Therefore, in the next level there will be 3^2 moves and so on. The state space tree has been shown in Figure 5. It is evident that if we move to n^{th} level then the upper bound of the complexity becomes $3^{n+1} - 1$, which is equivalent to 3^{n+1} thus giving $O(3^n)$ complexity. The above discussion proves that the complexity of brute force search mechanism is exponential.

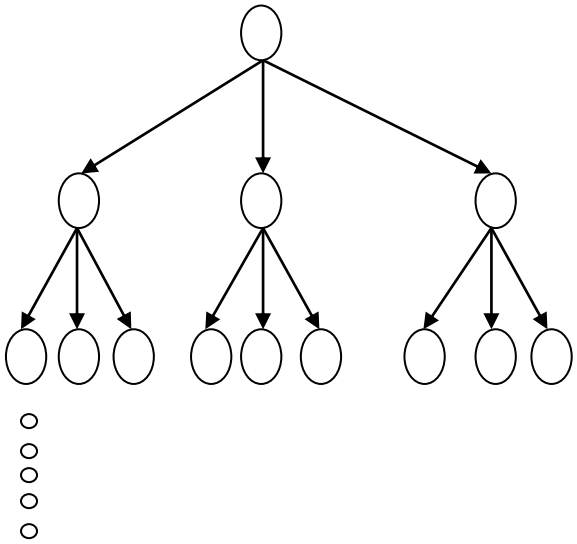

Fig 5: State Space Tree

4. SOLVABILITY

In N – Puzzle problem, there are total of N +1 tiles which contains distinct numbers and a blank space. These N+1 tiles can result in (N+1)! initial configurations. Out of these much configurations only half of the configurations are solvable and others are not. Thus only (N+1)! / 2 initial configurations can lead to goal configuration using limited number of moves.

Given an initial configuration, it can be checked whether the configuration is solvable or not. The steps for determining solvability are as follows:

Step 1: Shift the blank tile at the bottom right corner of the grid. This can be easily done.

Step 2: Calculate Permutation Inversion for each tile. An inversion is when a tile precedes another tile with a lower number on it [14].

For example consider a configuration shown in Figure 6.

Consider tile 12, there are 11 tiles numbered 1 – 11, with smaller number than 12, that appear after 12. Thus inversion number for tile 12 is 11.

Consider tile 14, there are 6 tiles numbered 5, 9, 3, 8, 13 and 6, with smaller number than 14, that appear after 14. Thus, inversion number for tile 14 is 6.

12	1	10	2
7	11	4	14
5	9	15	3
8	13	6	

Fig 6: Configuration

Inversion number of the tiles is as follows:

Tile 12 → 11 inversions
Tile 1 → 0 inversions
Tile 10 → 8 inversions
Tile 2 → 0 inversions
Tile 7 → 4 inversions
Tile 11 → 6 inversions
Tile 4 → 1 inversions
Tile 14 → 6 inversions
Tile 5 → 1 inversions
Tile 9 → 3 inversions
Tile 15 → 4 inversions
Tile 3 → 0 inversions
Tile 8 → 1 inversions
Tile 13 → 1 inversions
Tile 6 → 0 inversions

Step 3: Calculate the sum of inversions for all the tiles.

In the above example, sum = 46.

Rule: Odd permutation inversions of the puzzle are impossible to solve [15], all even permutations are solvable [16]. Archer also presented a simple proof of above rules [17].

Thus, the above configuration is solvable as the permutation inversion is even.

Consider another configuration as shown in Figure 7.

13	10	11	6
5	7	4	8
1	12	14	9
3	15	2	

Fig 7: Configuration

Inversion number of the tiles is as follows:

Tile 13 → 12 inversions
Tile 10 → 9 inversions
Tile 11 → 9 inversions
Tile 6 → 5 inversions
Tile 5 → 4 inversions
Tile 7 → 4 inversions
Tile 4 → 3 inversions
Tile 8 → 3 inversions
Tile 1 → 0 inversions
Tile 12 → 3 inversions
Tile 14 → 3 inversions
Tile 9 → 3 inversions
Tile 3 → 1 inversions
Tile 15 → 1 inversions
Tile 2 → 0 inversions

Sum = 59

Since, the number is odd, the above arrangement of the puzzle cannot be solved.

5. GENETIC ALGORITHMS

GAs are adaptive systems capable of solving problem by mimicking Nature. They are computational analogy of the theory of natural selection employing variation. To implement variation, variation-inducing operators are used such as mutation and crossover. Reproduction is done by evaluating the fitness of a chromosome using fitness function.

5.1 Algorithm

Step 1: Randomly generate an initial population P(0).

Step 2: Compute and save the fitness score f(p) for each individual p in the current population P(t). The fitness score is a measure of how good that chromosome is at solving the problem to hand.

Step 3: Select two members from the current population. The chance of being selected is proportional to the chromosomes fitness value. Roulette wheel selection is a commonly used method.

Step 4: Generate new chromosome for P(t+1) from P(t) to produce offspring via genetic operators.

Step 5: Repeat step 3 until a new generation P(t+1) is completed.

Step 6: Repeat step 2 until satisfying solution is obtained or futile value is reached.

The process has been explained in the Figure 8.

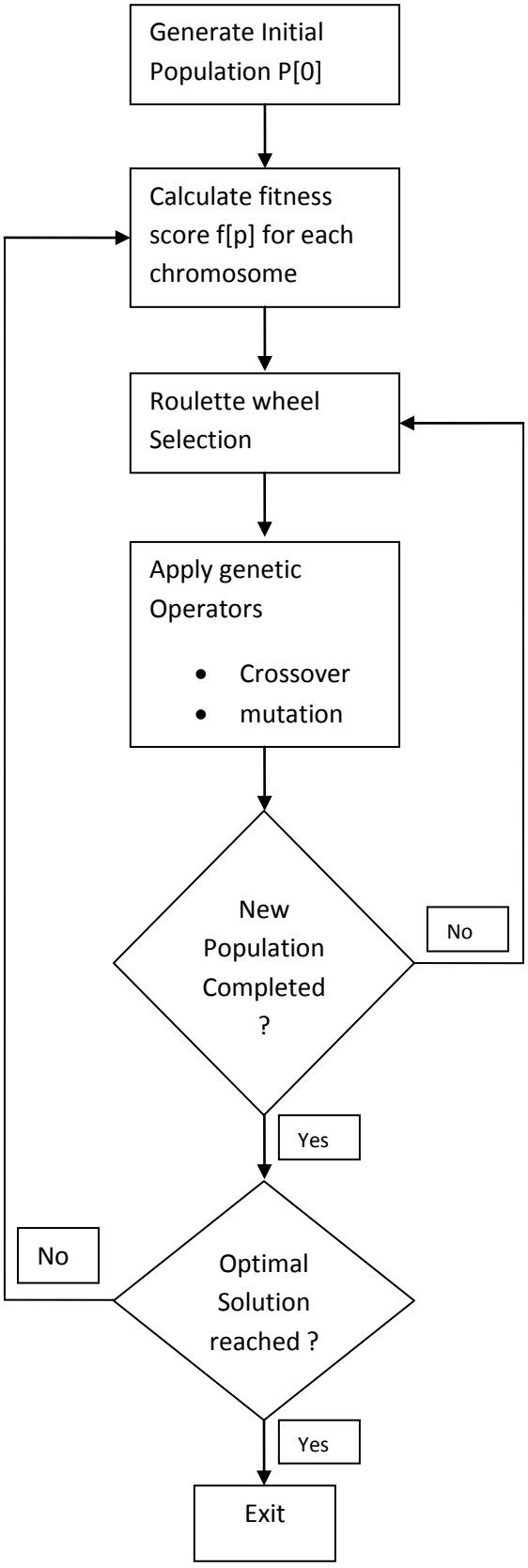

Fig 8: Process of Genetic Algorithms

5.2 Crossover

Crossover operator has the significance as that of crossover in natural genetic process. In this operation two chromosomes are taken and a new chromosome is generated by taking some attributes of first chromosome and the rest from second chromosome. In GAs a crossover can be following types:

5.2.1 Single Point Crossover

Single Point Crossover is performed by selecting a random gene along the length of the chromosomes and swapping all the genes after that point.

Eg. Given two chromosomes, choose a random bit along the length, say at position 9, and swap all the bits after that point. The Single Point crossover operator is shown in Figure 9.

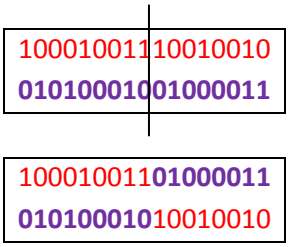

Fig 9: Single Point crossover

5.2.2 Two Point crossover

In this crossover, two crossover points are selected. The Two Point crossover operator is shown in Figure 10.

Fig 10: Two Point crossover

5.2.3 Uniform Crossover

In this crossover bits are uniformly copied from both the chromosomes. The Uniform crossover operator is shown in Figure 11.

Fig 11: Uniform crossover

5.2.4 Cut and Splice Crossover

The Cut and Splice approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point. The Cut and Splice crossover operator is shown in Figure 12.

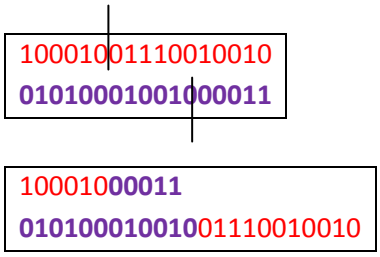

Fig 12: Cut and Splice crossover

5.2.5 Crossover Rate

Crossover Rate is the chance that two chromosomes will swap their bits. A good value for this is around 2% - 5%.

5.3 Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of population to the next. It is similar to biological mutation [18]. Mutation allows the algorithm to avoid local minima by preventing the population chromosomes from becoming too similar to each other [19]. The Mutation operator is shown in Figure 13.

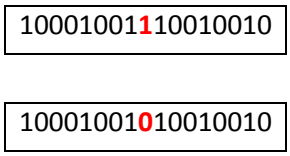

Fig 13: Mutation operator

5.3.1 Mutation Rate

Mutation Rate is the chance that a bit within a chromosome will be flipped (0 becomes 1, 1 becomes 0). This is usually a very low value for binary encoded genes, say 0.2% - 0.5%.

5.4 Why Genetic Algorithms are being used

N – Puzzle being NP Hard problem, if solved using brute force technique requires a lot of time which is exponential in nature. Facts regarding this have already been explained in the previous section. The problem has many optimal solutions and to reach one of the optimal solution GAs have been proved robust and best. The puzzle is not a graph based problem, therefore, is suitable to be represented as a state space tree. A chromosome of a genetic population can serve as a path from initial state to goal state as explained in further section. It eliminates the requirement for evaluating all the paths of the tree that can be generated. This gives the optimal solution in lesser time and with less number of moves. GAs are known to be better than randomized algorithm as they are more robust.

6. PROPOSED ALGORITHM

Step 1: POPULATION GENERATION

Generate a population of n chromosomes. Each chromosome has 2^m cells, where m is an integer indicating the proportionate size of the chromosome. At the end of the algorithm the significance of m will become clearer as it also gives an idea of the level of iterative deepening. The algorithm for the generation of population is given below:

```
for each chromosome
        for each cell
                if(random()%100 > 50)
                        cell = 1
                else
                        cell = 0
```

Step 2: ENCODING

For a particular chromosome, make pairs of two cells. Each pair is converted into a binary number and remainder is evaluated when the number is divided by 3. The output can be 0, 1 or 2.

0 stands for the first child of the present node in the state space tree, 1 stands for the second child and 2 stands for the third child. Here the numbering is taken in anticlockwise order.

If it is the case of a blank tile at the edge, then there will be two cases and not three and in that case modulo 2 is taken instead of modulo 3.

In case of a blank tile at the corner, there will be one case and modulo 1 will be done.

Step 3: CHROMOSOME ACCEPTANCE TEST

The above step gives us ~m levels, since each chromosome has 2^m cells. These levels are traversed and while traversing, if the configuration become unsolvable as defined earlier, then that chromosome is rejected.

Step 4: PATH TESTING AND CROSSOVER

From amongst, the remaining chromosomes, if any path gives us the solution then stop. It is to be noted that there can be 1 solution for particular instance of a problem. If it is not the case, then from all the chromosomes evaluated, we take two configurations, as in, 2 chromosomes and apply crossover operator to them. This gives us an all together new path which is to be traversed to see its effect.

Step 5: MUTATION

If by crossover, a solution is not obtained, then a mutant configuration is generated by randomly making a move. This can be implemented by using Pseudo Random Number Generator and changing the path at i^{th} level.

Step 6: ITERATIVE DEEPENING

The value of m is incremented in order starting from m = 2. This will provide the strength of Iterative Deepening along with a blend of heuristic search.

If the allotted resources or time exceeds a particular threshold as defined, then futility is said to be reached and the algorithm stops.

The process has been depicted in the Figure 14.

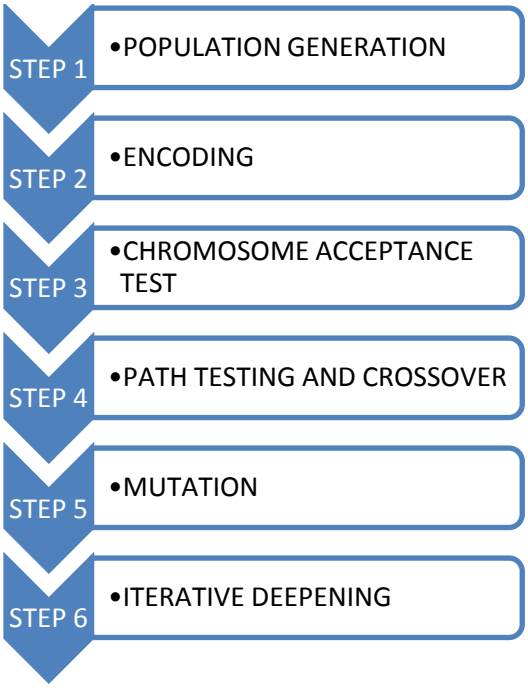

Fig 14: Genetic Based Solution to N puzzle Problem

6.1 Complexity Analysis

The above algorithm is better than the existing ones in terms of time complexity. The calculation of the complexity has been explained in this section. If the number of bits in a particular chromosome is taken as the power of 2, then at the n^{th} level the total traversals become O(log N). Where $N = 2^{m-1}$.Thus, the complexity will be $O(\log_2 2^{m-1}) \sim O(m-1)$.

This, results in the simple calculations for analysis of the results in term of complexity.

7. CONCUSION AND FUTURE SCOPE

A new algorithm has been proposed. It gives a sequential approach in solving the problem. Moreover, it is intended to establish precise theoretical study of the above problem which will also help in other peripheral problems. It is intended to implement the proposed algorithm in C# language and analyze the results obtained by deriving regression equations and performing the requisite statistical tests. The tests will instill the confidence in the proposed algorithm.

While analyzing the complexity of the solution proposed in terms of time, by taking the length of chromosome as power of 2, the calculations for complexity become simpler. Due to simplicity, unknown constants of regression equation can be easily computed.

The use of single point crossover has been proposed above. It is intended to analyze the effect of two point crossover, multi point crossover, cut and splice crossover on the problem and complexity obtained.

After the above things, it is intended to apply Diploid Genetic Algorithms to solve the problem, to have lesser complexity and better results.

8. REFERENCES

[1] Harsh Bhasin and Neha Singla, (2012), "Harnessing Cellular Automata and Genetic Algorithms To Solve Travelling Salesman Problem', International Conference on Information, Computing and Telecommunications, (ICICT -2012), pp. 72 – 77.

[2] Harsh Bhasin and Gitanjali, (2012), "Harnessing Genetic Algorithm for Vertex Cover Problem", International Journal on Computer Science and Engineering (IJCSE), Vol. 1, Issue 2, pp. 218 - 223.

[3] Harsh Bhasin and Nishant Gupta, (2012), Randomized algorithm approach for solving PCP, IJCSE 2012; 4(1):106-113. ICID: 976303

[4] Harsh Bhasin and Neha Singla, (2012), Modified Genetic Algorithms Based Solution to Subset Sum Problem, IJARAI Vol. 1 (1).

[5] Hayes, Richard. (2001), 'The Sam Loyd 15-Puzzle'. - Dublin, Trinity College Dublin, Department of Computer Science, TCD-CS-2001-24, pp28.

[6] Richard E. Korf and Larry A. Taylor, (1996), 'Finding optimal solutions to the twenty-four puzzle', in Proceedings AAAI 1996, pp. 1202–1207

[7] Bauer, Bernard, (1994), The Manhattan Pair Distance Heuristic for the 15 Puzzle, Paderborn, Germany.

[8] Chris Calabro, (2005), Solving the 15-Puzzle.

[9] Zygmunt Pizlo; Zheng Li, (2005), Solving combinatorial problems: The 15-puzzle. Memory & Cognition, Vol. 33 Issue 6, p1069.

[10] Ariel Felner and Amir Adler, (2005), "Solving the 24 Puzzle with Instance Dependent Pattern Databases". Proceedings of the Sixth International Symposium on Abstraction, Reformulation and Approximation (SARA-05), pages 248-260.

[11] Ariel Felner, Richard E. Korf and Sarit Hanan, (2004), "Additive Pattern Database Heuristics", Journal of Artificial Intelligence Research (JAIR), 22:279-318.

[12] Alexis Drogoul and Christophe Dubreuil, (1993), 'A Distributed Approach to. N-Puzzle Solving', Proceedings of the Distributed Artificial Intelligence, pp. 95 – 108.

[13] Graham Kendall, Andrew J. Parkes, Kristian Spoerer, (2008), A Survey of NP-Complete Puzzles.ICGA Journal 31(1), pp. 13-34.

[14] http://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html

[15] Johnson, W. W. (1879). Notes on the "15" Puzzle. American Journal of Mathematics 2(4):397–404.

[16] Story, W. E. (1879) "Notes on the '15 Puzzle. II.' ", American Journal of Mathematics 2, 399-404.

[17] Archer, Aaron F. (1999), "A modern treatment of the 15 puzzle", The American Mathematical Monthly 106 (9): 793–799.

[18] S. Thrun, (1995) "Learning to Play the Game of Chess", In G. Tesauro, D. Touretzky, and T. Leen, editors, Advances in Neural Information Processing Systems(NIPS) 7, Cambridge, MA, MIT Press.

[19] H. Bhasin and S. Bhatia, (2011), "Application of Genetic Algorithms in Machine learning", IJCSIT, Vol. 2 (5).