GENETIC APPROACH TO SOLVE TOWER OF HANOI PROBLEM

Samran Naveed[1] and Muhammad Sajid Iqbal[2]
[1]Iinformation and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia
[2]Electrical Engineering Department, National University of Computer and Emerging Sciences, Karachi, Pakistan
Corresponding author email: samran.naveed@live.com

ABSTRACT: *Automated planning is a useful technique in finding the solution to complex problems. And it is usually helpful to have an algorithm to find the optimal solution for planning problems. During this work, we have proposed a genetic approach to planning, and presented a method to apply genetic algorithm to explore state space of planning problem. To justify the proposed approach, an artificial intelligence problem, "Tower of Hanoi" is considered. The mathematical formulation for the problem was made in planning domain by using indirect encoding and solution is sought by applying genetic algorithm. The proposed algorithm is flexible and to verify the flexibility, we have considered different cases of tower of Hanoi. Several experiments are performed by considering different initial states for Tower of Hanoi, and efficiency of algorithm is analyzed by varying the mutation and crossover rates. The obtained results are proving that our proposed algorithm is flexible and can be enhanced for more complex planning problems. The detailed results along with complete analysis are also included in results and discussion section of this paper.*

Key words – Genetic Algorithm, Automated Planning, Tower of Hanoi, Artificial Intelligence

INTRODUCTION

Many human activities require explicit planning for example when the problem is complex, the environment imposes risk/cost, or we are addressing a new situation. The main objectives of planning are Scientific Objectives and Engineering Objectives [1]. Scientific objectives involve, understanding of intelligent behavior and engineering objectives involves, building of intelligent entities. We can illustrate the conceptual model of planning as a state-transition system; in general, state-transition system consists of a set of states, set of actions and a state transition function and the objective is, "given an initial and goal state find a valid set of actions that lead the system from initial state to goal state". A plan is a structure that gives appropriate actions to apply in order to achieve goal state, when starting from a given initial state. State transition system is usually denoted as follows.

$$\Sigma = (S, A, \gamma) \qquad (1)$$

Where $S = \{s_1, s_{2\ldots} s_{n\}}$ is a finite set of states. $A = \{a_1, a_2\ldots, a_n\}$ is a finite set of actions γ is a state transition system and can be expressed as follows.

$$\gamma : S \text{ x } A \rightarrow S \qquad (2)$$

If a∈ A and γ (s, a) ≠ ø then "a" is applicable in "s". State transition system describes all ways in which a system may evolve. There could be different types of objectives of a planning problem. For example, some of the concrete objectives may involve finding a desired goal-state or a set of goal-states or optimize utility function attached to states [2]. In real world problems no single or agreed upon description is available and people care about the solution. To solve a planning problem different searching methods are used. In general, planning problems are more complex and difficult than searching problems. Planning problems involve large search spaces and we cannot always guarantee that the correct solution will be available every time. Before exploring the search space, we must assume that the environment is finite and discrete, fully observable, deterministic and static.

Most common searching algorithms for planning problems find a good solution rarely. Forward state space search and backward state space search are two most general and famous algorithms for searching plans. These algorithms are required to search entire search space to find a solution. In real world problems, the search spaces are very large, so these algorithms may not perform very well [3]. Moreover, in forward state-space and backward state-space search algorithms we use deterministic searching methods [1] that make them inefficient in large search spaces.

Genetic algorithm (GA) inspired by theory of evolution is a search heuristic algorithm. We often use GA to search the solutions for optimization problems. For large search spaces and NP-complete problems, genetic algorithms are suitable. To reach the final solution through GA, we generate multiple populations and the most-fit generation has higher chance to survive in next generation [1]. In the new population the generations undergo crossover and mutation processes. Each generation consists of a solution, and the most fit or most relevant generation becomes the final solution. The steps of GA are shown in Fig. 1.

In this study, we have considered "Tower of Hanoi" problem, and formalized it to solve by using genetic algorithm. Tower of Hanoi is a classic artificial intelligence problem in which we have three pegs and different number of disks and the goal is to move all disks from initial peg to goal peg without violating the rules. Some non-recursive algorithms [4] also proposed to solve the problem, but they do not perform well as the number of pegs and disks increased. But, in this research work we formulated the problem as a state transition system of planning; here we have set of states that contain all the possible states of pegs. In addition, set of actions consists of all possible moves. The initial state is the starting configuration of all the pegs, and the goal state is the desired configuration of the pegs. To solve this formulation with genetic algorithm we have used direct encoding of moves to represent the set of actions. The details of formulation can be found in section III and experiments carried out to validate this formulation along with their results are presented in section IV.

RELATED WORK

The enetic algorithm is an optimization algorithm; this fact makes genetic algorithm to be applied to solve numerous kind of problems including planning problems. R. N. Cardoso et. al. [5] described the ability of automated planning in solving real time problems and solved it with genetic algorithm by converting the problem into an optimization problem. The researchers considered the container loading problem (CLP) with the aim to reduce the logistical costs.

They designed a complete CLP system in automated planning domain and found the optimized set of actions to reduce the cost in container loading problem.

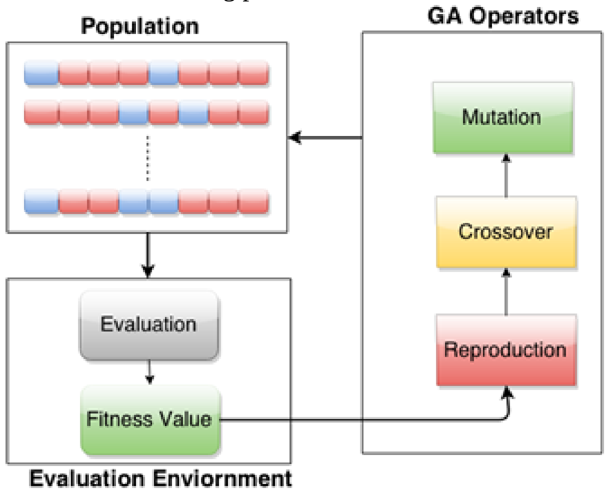

Fig.1. Steps for genetic algorithm

Despite of very good explanation and formulation provided by the authors they have not provided comprehensive details of fitness functions. Furthermore, the fitness evaluation details they provided, the lack of diversity and effectiveness with respect to guidance the algorithm towards a optimal solution.

Another real time application of automated planning can be found in [2] in which Y. Sulaiman *et. al*. presented an approach to automatically generating the sequence diagrams by considering them planning problem. The core idea was to identify the preconditions and post conditions and consider them the initial and goal states. They considered the methods as actions and formulated a plan to solve the sequence diagram generation problem. The authors have provided very limited empirical evaluation. And they have used traditional depth first and breadth first searching algorithms to search the state transition system which are not efficient if the search space is very large.

L. Machado *et. al*. [6] presented a planning based approach to solve task allocation problem for crowd-sourcing. Crowd-sourcing is comparatively new phenomenon in computer science and its primary function is to outsource a task to a crowd of participants who will solve that task but the challenge is the task allocation. Task allocation in crowd-sourcing is not that simple especially when there are many factors to tackle like, knowledge required, task's requirements, heterogeneity and size of the task. To efficiently handle the task allocation problem, they used automated planning as a test and to evaluate its performance. The shortcomings in this research are they have only tested the automated planning in the task allocation domain at a basic level, and did not perform extensive empirical evaluation.

S. Soltani *et. al*. [7] proposed an approach to the software product lines problem which is, feature model configuration based on function and non-functional requirements. The author formulated the problem using an automated planning domain and enabled the users to automatically select features' configuration which comply with both functions and non-functional requirements. In real time systems the functional and non-functional requirements are usually a large set, based on the formulation and details presented by the authors in [7] it will be very time consuming and resource demanding to compute suitable set of features.

The problem of tower of Hanoi is well described by T. Jones in [8] and our solution encoding is inspired from the formulation described in [8]. The author used the direct encoding to encode the solutions, and represented the moves as integers. The solution provided in [8] is only for three disks tower of Hanoi and is only providing the description of genetic algorithm steps, no information is given in the context of planning. Due to the lack of diversity in genetic algorithms, the solution converges prematurely and keeps on searching for a long time.

A genetic algorithm based approach proposed by J. Li et. al. in [9], the authors formulated the tower of Hanoi problem in searching domain and used a genetic algorithm to find the solution. For genetic algorithm the direct encoding of chromosomes is used. Although the presented approach successfully finds the solution, but the algorithm performs very slow as the number of disks increases also the algorithm finds the solution rarely. Even with a small number of disks the algorithm takes reasonable time to find the optimal solution. Finally, only mutation is applied during the design of algorithm no crossover has been used which may have anegative impact on the results.

PROBLEM FORMULATION

Before describing the actual problem formulation in detail, it is worth describing here the problem rules. To solve tower of Hanoi problem one must follow these rules.

- At a time one disk can be moved
- A smaller disk cannot be placed under the larger disk
- All disks, except the one being moved must be on a peg

Above mentioned rules will decide the applicability of a certain action to some state. There are different steps of genetic algorithm to find a solution. Fig. 1 is showing the flow of these steps and the following lines are discussing it with respect to tower of Hanoi. Before going into the details of genetic algorithm steps lets elaborate the states, actions and state transition.

$$S = \{different\ configurations\ of\ pegs\} \quad (3)$$

$$A = \{all\ possible\ moves\} \quad (4)$$

Therefore, the plan for our problem becomes:

$$P = (\Sigma,\ s_0, s_g) \quad (5)$$

Where $s_0$is initial state, we have solved this problem for different initial states. For example, one can provide different configuration as initial state (for 3 disks problem, let say we provide 1 disk on peg A, 1 disk on peg B and 1 disk on peg C) and s_g is goal state.

Population: Each population consists of different generations called chromosomes and each chromosome consists of genes. In chromosomes, the genes are problem specific. In some problems, genes may be binary and in some other problems, genes may be characters. In our case, each gene will represent a move. We have used a direct encoding to encode the moves. Three pegs A, B and C are represented as integers 0, 1 and 2 respectively and the moves are encoded similarly, for example move from peg A to B is represented as 01 and move from A to C is represented as 02 and so on. Each population will consist of a reasonable length of chromosomes and we have set the length of individual chromosome equal to the optimum number of moves. The optimum number of moves to solve towers of Hanoi problem is given as $2^n - 1$. Where n is the number of disks. When algorithm runs for the first time, it will generate a population randomly.

Evaluation: Once the population is generated and initialized, it is evaluated to calculate its chromosome's fitness. To evaluate the fitness of a chromosome it's all genes are applied to the initial state one by one and all the illegal moves are counted. Every move that violates the rules described above is considered as illegal. For example, let say the current move is "02" that is applied to the initial state (all disks are on peg A) and after application of this move one disk will be moved to peg C, this is a legal move, as it did not violate any rule. Now, let say the next move is again 02 this move cannot be accomplished because it is an illegal move due to th second rule. The fitness is calculated by first calculating the Match Fitness (MF) and multiplying it with Goal Fitness (GF) of the chromosome. Match fitness and goalfitness is calculated as follows.

$$MF = \frac{Length\ of\ chromosome - illegal\ moves}{Length\ of\ chromosome} \quad (6)$$

$$GF = \frac{number\ of\ disks\ on\ goal\ peg}{total\ number\ of\ disks} \quad (7)$$

$$fitness = MF \times GF \quad (8)$$

The fitness is calculated for a chromosome and the calculated value is assigned to the same chromosome. This process will continue for all chromosomes in current population. The calculated fitness is compared with the max fitness (i.e. 1), if it is equal to max fitness it means this is the best-fit chromosome and it contains the solution as well. If it is not equal to max fitness, then the algorithm will check next chromosome until it finds the solution of most fit chromosome in the current population. If algorithm is unable to find the solution in current population, it will reproduce te new population and check the fitness of all chromosomes again.

Selection (Reproduction): We want to improve the populations overall fitness. Selection helps us to improve the fitness by rejecting the bad chromosomes and keeping the best chromosomes in the population. There are different selection methods but the basic idea is same, that the fitter chromosomes will be selected for te next generation. In our solution, we have used Tournament Selection approach to select chromosomes for te next generation. To avoid the algorithm not to stuck in local optimum it is important to not to choose the top fittest chromosomes.

Crossover: Crossover is the process that creates a new chromosome by partially inheriting genes from two different chromosomes. Once the chromosomes are selected for new generation, they undergo the process of crossover. We performed different rate of crossover.

*Mutation: M*utation is the process in which a gene of a chromosome is changed randomly. In under discussion problem a move (represented as gene) of a chromosome will change randomly.

Repeat: The process of selection, crossover and mutation will repeat until the algorithm found the solution or the population fitness is not improving for a certain number of generations. For example, if the fitness of the generation is stuck to a specific number and it is not improving since 100 generations then the algorithm should terminate.

We implemented the above formulation to solve the tower of Hanoi problem. Our implementation can solve not only the traditional tower of Hanoi (initially all disks are on peg A) problem, but also the variations of the problem (user can define te number of disks on all three pegs in initial state). Fig. 2 is showing one of the possible initial states.

EXPERIMENTS AND RESULTS

Different experiments with different initial states are conducted with the implemented solution. Before describing the experimental results, the details about experimental setup are provided below.

The solution is implemented in PHP version 5.5.15 and to draw the final states graphically, a jQuery library is used. The experiments are carried out on core i5 machine with 4 GB RAM. As PHP is a scripting language and it runs in browsers so the experiments were performed in google chrome browser. The implemented solution executed for multiple times by varying the genetic parameters to analyze the effect of genetic parameters to find the solution for the give initial state. Table I is shows the details of experiments that were conducted by varying the crossover rate. First row in Table I shows the number of disks used to perform the experiment, second row is showing the initial state A: 3 means that there are three disks on peg A in initial state and zero disk on other pegs. In third row the mutation rate is given that is fixed to 0.01 for the experiments whose results are given in the table. The fourth row is showing the different crossover rates that we used in different iterations of the experiments and fifth row is showing the average execution time. The last row is showing that after how much iteration the implemented solution finds the solution for tower of Hanoi problem by using given settings. We executed thirty runs for each experiment so the average execution time and average number of iterations for solutions are for thirty iterations.

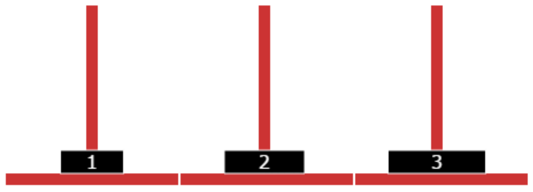

Fig. 2. Initial state

Table I
Effect of Crossover on Experimental Results

No. of Disks	3			
Initial State	A: 3 B: 0 C: 0			
Mutation	0.01			
No. of Generations	100			
Crossover	0.1	0.2	0.3	0.4

Avg. Execution Time	0.364s	0.371s	0.369s	0.361s
Avg. no. of Iterations for Solution	23	19	16	15

Table II
Effect of Mutation on Experimental Results

No. of Disks	3			
Initial State	A: 3 B: 0 C: 0			
Crossover	0.4			
No. of Generations	100			
Mutation	0.02	0.03	0.04	0.05
Avg. Execution Time	0.360s	0.381s	0.373s	0.368s
Avg. no. of Iterations for Solution	15	13	9	9

We have plotted the population fitness that illustrates how population evolved to find a solution. Fig. 3 is showing the fitness plot for crossover 0.4 with mutation rate of 0.04, it is clear from the plot that when mentioned crossover and mutation rate is used then the fitness converges quickly and algorithm finds solution quickly. Fig. 4 is showing the plot for crossover rate of 0.3 with mutation rate of 0.04 it is clear from the plot that the algorithm find solution approximately at generation 37. Fig. 5 and Fig. 6 are showing the different fitness plot that is plotted when the crossover rate is 0.2 and 0.1 respectively with mutation rate of 0.04. Sometimes it happens that the algorithm stuck in the local maximum and unable to converge. Fig. 7 is showing one of the stuck condition where the algorithm was stuck at 0.666 fitness and did not converge. We have applied termination condition that if the algorithm remains stuck on some fitness for predefined number of generations, and then stop the execution. Finally, when the algorithm finds a solution it will display it with the final state as shown in Fig. 8.
Along with traditional initial states we also performed experiments on nontraditional initial state for example in Table III one nontraditional state is mentioned where all three pegs have one disks. In order to define these states, we have to provide the disks weights, so the algorithm can differentiate on which peg, what disk is placed, higher weight means bigger disk. In the Table III the initial state is, "A: 1", "B: 1" and "C: 1". Which means that on all three pegs there is one disk, and we defined the weights of the disks in a way that, A has the smallest disk placed on it B has the medium disk placed on it and C has the largest disk placed on it. From the previous experiments results we can infer that, the suitable values for crossover and mutation rate are 0.4 and 0.04 respectively. So to perform the experiments same thirty iterations are used for nontraditional initial state experiment, and we found that the algorithm was able to find the solution early as compared to traditional initial state.
As we described before that we defined the initial state in such a way that last peg contains the biggest disk so to solve this problem we only need two actions. First action will put the medium disk on last peg on top of largest disk and the second action will put the smallest disk on top of other disks and once we found these two more suitable actions we will stop the execution of the algorithm.

Table III
Effect of Crossover on Experimental Results

No. of Disks	3
Initial State	A: 1 B: 1 C: 1
Crossover	0.4
No. of Generations	100
Mutation	0.04
Avg. Execution Time	0.11s
Avg. no. of Iterations for Solution	3

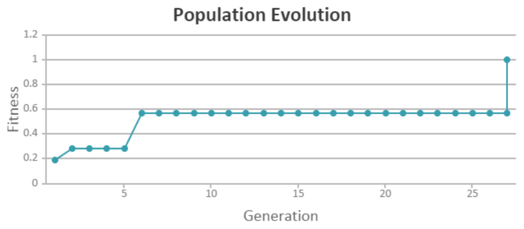

Fig. 3. Population Evolution for crossover 0.4 and mutation 0.04

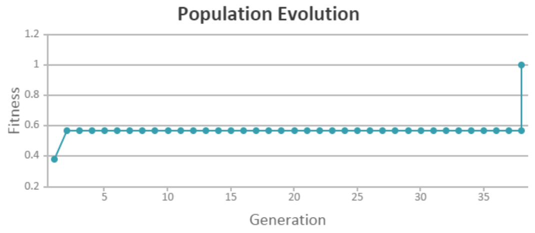

Fig. 4. Population Evolution for crossover 0.3 and mutation 0.04

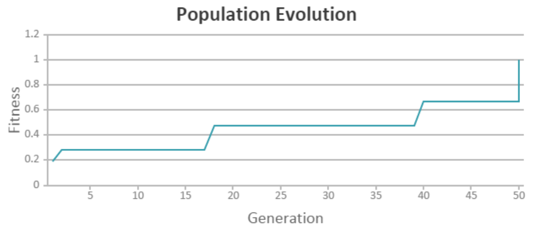

Fig. 5. Population for crossover 0.2 and mutation 0.04

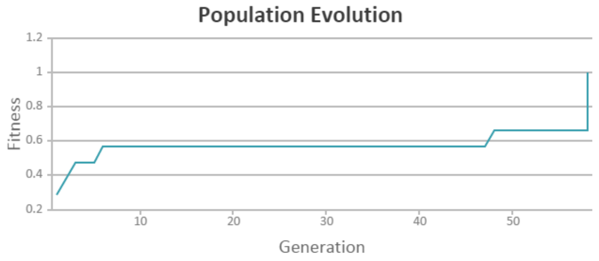

Fig. 6. Population for crossover 0.1 and mutation 0.04

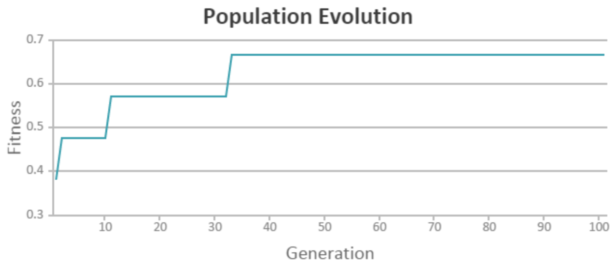

Fig. 7. Stuck condition

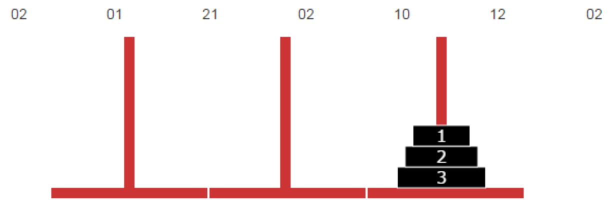

Fig. 8. Final state with the optimal solution

THREATS TO VALIDITY

Before approaching the conclusion, it is worthwhile to state some of the threats to validity of the experiments. To implement the solution, we followed the OOP paradigm and we used arrays as data structure to handle populations and genes. The performance may be improved if some other data structures are used like heaps or binary trees, it may also improve the execution time overall.

Another possible threat to validity is the lack of diversity in genetic algorithm and its non-deterministic behavior. Sometimes if we are lucky, we can find the solution quickly and sometime we were unable to find the solution after several runs, instead of finding the optimal solution the algorithm stuck in local optimum.

CONCLUSION

In this paper, we have presented a planning based formulation of tower of Hanoi problem and solved it by using genetic algorithm. The problem formulation was made by using the indirect encoding and successfully implemented in PHP. Results of the experiments are showing that the genetic algorithm is capable of solving tower of Hanoi problem. The genetic parameters like crossover rate and mutation rate play an important role in finding the solution. It can be seen from the results of the conducted experiments that in three disks case the implemented solution was able to efficiently find the optimal solution.

Some threats to validity of this solution are also discussed, which may be dealt by enhancing the problem formulation and implementation. It can be noticed from the literature that very limited number of studies has been conducted on automated planning by using genetic algorithm. Automated planning and genetic algorithm are being used in various domains, but they are used separately and for different purposes. So we can say that our work is one of the earliest works of applying genetic algorithms to planning problems. We can perform more experiments by varying other genetic factors like population size, generation size in order to enhance the justification of our approach. We are also planning to improve our approach further and apply it to different domains and complex problems. For example the automatic generation of sequence diagrams can be thought as a planning problem under certain conditions, so we can apply the genetic algorithm approach to automatically generate the sequence diagrams.

REFERENCES

[1] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach: Pearson Education, 2003.

[2] Y. Sulaiman and M. Ahmed, "Automating UML Sequence Diagram Generation by treating it as a Planning Problem", presented at the International Conference on Distributed Multimedia Systems, 2012.

[3] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] F. Ren, Q. Yang, J. Zheng, and H. Yan, "Non-recursive Algorithm of Tower of Hanoi Problem," presented at the Proceedings of the 10th IEEE International Conference on Computer and Information Technology, 2010.

[5] R. N. Cardoso, M. V. M. Ferreira, A. R. de Sousa, and J. J.-P. Z. S. Tavares, "A Genetic Algorithm Approach to the Automated System for Solving the Container Loading Problem," in Robotics: 12th Latin American Robotics Symposium and Third Brazilian Symposium on Robotics, LARS 2015/SBR 2015, Uberlândia, Brazil, October 28 - November 1, 2015, Revised Selected Papers, F. Santos Osório and R. Sales Gonçalves, Eds., ed Cham: Springer International Publishing, 2016, pp. 267-280.

[6] L. Machado, R. Prikladnicki, F. Meneguzzi, C. R. B. d. Souza, and E. Carmel, "Task allocation for crowdsourcing using AI planning," presented at the Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering, Austin, Texas, 2016.

[7] S. Soltani, M. Asadi, D. Ga, #353, evi, #263, M. Hatala, and E. Bagheri, "Automated planning for feature model configuration based on functional and non-functional requirements," presented at the Proceedings of the 16th International Software Product Line Conference - Volume 1, Salvador, Brazil, 2012.

[8] T. Jones, Artificial Intelligence: A Systems Approach with CD: Jones and Bartlett Publishers, Inc., 2008.

[9] J. Li and R. Shen, "An Evolutionary Approach to Tower of Hanoi Problem," in Genetic and Evolutionary Computing: Proceeding of the Eighth International Conference on Genetic and Evolutionary Computing, October 18-20, 2014.