Genetic Algorithms: Darwin solves some AI problems

M. Sasikumar

Artificial intelligence as a field has always been used to solve problems which are difficult to solve through formal framworks. Use of Genetic Algorithm as a solution model with neural network design to scheduling problems are been touched featuring GA as a valuable tool for solving a number of practical complex problems discussing the issues like local optima troubles AI, Design of Evolutionary Solution, forming a population, crossover/ mating with reference to the basic GA model.

Mr. M. Sasikumar is Head Department of Artificial Inteligence C-DAC, Mumbai.

Local Optima Troubles AI

Artificial Intelligence as a field has always been rich with problems difficult to solve through formal frameworks. These include natural language understanding, machine learning, game playing, planning and scheduling, pattern recognition, and so on. Among these are two major concerns which have attracted much attention, with relatively few effective solutions. One is the ability of a human being (and many other animals) to adapt to changes. We have certain types of behaviour which has become our nature. But if the environment changes overnight into something else (e.g. you move into a city from a village), we modify our behavioural patterns based on our observations of the current environment, till the new set of behaviour becomes natural. How does one model such an ability in a computer?

Another, perhaps more important concern has been optimisation. There are a number of problems which we face routinely where we would like an optimal solution. How to get to work in the shortest period of time? How to make a business/holiday trip with minimal expenses? How to schedule exams in the minimum time and maximising the convenience of teachers and students? How to identify best bus routes in a city? How to make a railway timetable? There are dozens of such examples we can think of. The field of operations research and related mathematics have been seized of this problem for many decades. Specialised techniques are available for use when the problem satisfies specific restrictions, for example, variable values are deterministic, constraints are linear, etc. A general purpose solution model applicable to a wide range of problems is yet to emerge.

Researchers in AI have been interested in these problems from the beginning. It was soon obvious that one has to search through all possible solutions in some manner to guarantee optimal solution. However, this is too time consuming. Even simple analysis will show that to get answer to non-trivial problems would require time of

the order of the age of the universe! Therefore, AI has actively pursued efficient ways to search through large spaces. This is quite similar to searching on a landscape for something like the tallest point. Your ability to locate the tallest point depends on visibility, obstruction due to your current neighbourhood, etc. Search spaces also posed similar problems, perhaps the most complex of which was what was called local optima. As you grope around for higher and higher regions, you reach a peak. Now every surrounding move takes you to a point of lower height. But you have no idea if you have reached the highest peak! You are stuck! This local optima problem is a tricky problem to address; most search algorithms give up on this issue.

Given these challenges, researchers have been exploring various search models. You can find ideas such as simulated annealing, iterative deepening A*, etc for example. In this article, we discuss an interesting idea inspired by the process of evolution!

Help from Mother Nature

When faced with difficult problems, it is not unusual for us to turn to mother (biological as well as nature!) for help. Similarly phenomena observed in nature has inspired many ideas and inventions. The idea of aircraft came from birds. Neural networks is inspired by whatever little is known of how human brain works. Observation of ants finding efficient path to food has led to a method known as 'ant colony optimisation'. The flocking behaviour of birds in flight has inspired much work in the area of Artificial Life.

In this case, the inspiration came from evolution. Going by the Darwinian model of evolution and Mendell's theories of genetics, nature has done a wonderful job of evolving sophisticated entities in evolution. Both the models are based on very simple ideas. In genetics, the chromosome provides a more compact representation of an individual than using the individual directly. It is a blueprint of an individual which determines all the critical aspects of an individual in its life time, and also what it will pass on to its children. The mating process enables nature to combine chromosomes from multiple individuals and create new combinations. In addition, nature also does random mutations in a chromosome from time to time. These are its tools to create variations in the chromosome structure, and thus producing individuals with different combinations of characteristics. Note that these changes are not induced by nature in response to some stimulus. If you move to a hot climate from cold, it is not likely that your children will have more characteristics for living in a hot climate. Nature experiments randomly. This is an important aspect of this model. It is darwin's model which tilts the balance to a dominance of hot-climate oriented people over a period.

Darwin's model suggests that those individuals with the most suitable characteristics for a given environment will survive better in that environment. Those who survive longer produces more children and hence creates more copies of individuals with that characteristic. Traits which are bad in a given environment leads to poor survival probability, leading to the extinction of individuals with that trait.

Genetic algorithm as a solution strategy borrows essentially these ideas and adapts them for a computer implementation. Do remember that though we take inspirations and ideas from nature, we do not mimic it blindly. Our aircrafts do not flap their wings when flying; they offer a better solution for our problem (longer distance, carry bigger weight, etc) than a bird does. The artificial neural networks we implement in a computer, do not resemble the biological neural network except on a few aspects. Similarly, we adopt the basic ideas outlined above and adapt them to suit our constraints and requirements.

Implementing an optimisation solution as an algorithm based on the evolution model requires us to formalise the notion of a chromosome, mechanism to map a chromosome to an individual to determine its behavioural characteristics, mimic the 'survival of fittest' notion, and mechanisms to make changes in the chromosome as we move from generation to

generation. We discuss these, in the context of a simple example, in the next section.

Designing an Evolutionary Solution

Consider the popular, but difficult to solve, problem of the travelling sales person (TSP). There are N cities (C1, C2, ...) interconnected through a road network. Going from one city Ci to another city Cj incurs a certain cost which is a function of the source and destination (and nothing else), and is denoted as Pij. One has no formula to compute Pij, in general, and hence will be given a matrix of numbers corresponding to all i-j values. The person has to visit all the cities once, and would like to minimise the total expenses. The problem for us is to find the path for him. For example, your solution may look like: C5, C10, C3, C2, ..., with each city occurring once and only once. Its cost would be P(5,10)+P(10,3)+P(3,2)+....

The simplest way to solve this problem is to try every route. Considering 5 cities, we have the following routes to be explored.

1,2,3,4,5 1,2,3,5,4 1,2,4,3,5 1,2,4,5,3

1,3,2,4,5 1,3,2,5,4 1,3,4,2,5 1,3,4,5,2

and so on...

There will be 5! ways (approx 120). Find out the figure for 20 cities and see how quickly the number of options grow. It is infeasible to try even a fraction of this within reasonable time. Various heuristic solutions have been proposed to give good solutions; but no guarantee of optimal solutions. Heuristics generally have their weak points; they can give horrible solutions for certain data sets.

Let us try the genetic algorithm model for this.

The basic idea is to create a situation for evolution to occur, for which we first need a population. We also need to formulate a chromosome structure for an individual. (To keep the length of this article within limits, I will not follow the historical developments of ideas in GA; rather I will attempt to illustrate the key aspects of a GA through a realistic example.)

Gene Structure: There are various ways of formulating a chromosome structure for this problem. One simple way is to just use an array G of numbers, with G_i representing the i^{th} city to be visited. So the chromosome structure looks essentially like the solution. As an alternative, We can denote G_i to denote an offset from the previous city. If the (i-1)th city is C, then the next city is C+Gi. In this case, to get the solution, we need to process the chromosome structure. In some applications, the chromosome structure can be very different from the solution. We will adopt the array representation mentioned first.

Forming a population: Having decided the chromosome structure, it is easy to formulate a population of individuals. To start with we select a few randomly generated permutations of the sequence 1,2,3,...,N - each permutation representing one individual. Note that every permutation is a valid solution, and only a permutation is valid (each number once and only once in a solution). Let us create S such solutions to form our population. Obviously, the S solutions vary in their quality. Some may be very expensive, and some may be quite good. Here is an example for a population for a 5-city case, with S = 6.

- P1: 1,3,4,2,5
- P2: 1,4,2,5,3
- P3: 2,3,4,5,1
- P4: 1,3,5,2,4
- P5: 1,3,5,2,4
- P6: 2,4,1,3,5

Crossover/mating: We are now ready to play the game imitating nature. We allow the individuals to mate in pairs to produce children, which are in turn valid solutions. In the original model of genetic algorithm, the chromosome structure was a simple binary string like 1100101010. So mating could be implemented by taking alternate bits from each parent, taking

first 50% of the bits from the first parent and the rest from the second, and so on. These provide comparable contribution to the children from both the parents. In our case, such simple approaches can give us trouble. Consider two solutions: 1,2,3,4,5 and 5,4,3,2,1, and any interleaving of cities from these two. Unless we are careful, the resulting solution may not be a permutation of 1 to 5, and hence an invalid solution. In the GA parlance, one can find a plethora of such mating operators (called crossover) suitable for various representations. Here is one operator which works for us:

Cut one of the solutions at some random point, say, after the third number in our example. Take the first part as the start of the child. So the child starts as 1,2,3,x,x. The cities remaining to be included are 4 and 5. These remaining cities are listed in the order in which they occur in the second parent. In the example, this is 5,4. So the child chromosome will be: 1,2,3,5,4. Note that this borrows some aspect of the order from both parents. This is the basic concern in choosing a crossover operator: the child solution should borrow ideas from both parents.

So we pick a pair of solutions (chromosomes) from the population, mate them as above to get a child chromosome. Actually, the way most cross over operator is implemented, you get two children in each crossover: you get the second one by doing the same process after interchanging the two parents. The children may be better or worse than the parents.

One crucial concern is in selection of parent chromosomes. Remember our earlier observation of fitter individuals being able to reproduce more. This is very important to ensure that their chromosome structure is propagated more than the others. Keeping this in view, parent selection is normally done with a bias towards fitter solutions. Remember that getting a desirable change in behaviour may take multiple changes in the chromosome. As these changes occur one by one in a chromosome, the intermediate solutions may look bad. Imagine the growth of a hand on our body. Just a growth from the sides would possibly be bad for survival. Until the growth evolves into a full fledged hand, its impact on the fitness may be negative. But if you never give these individuals with a partially developed hand a chance to participate in evolution through mating, etc, the hand never gets a chance to evolve. Thus we do not completely rule out anyone from the mating. But the good ones get to do it more often, compared to the poor ones.

We repeat this process N/2 times so that we have N more solutions in the population. Then the selection process is applied - the best N from the total of 2.N solutions present now, are retained and the others are discarded.

Now we are in the second generation with a set of N solutions in the population, and we are ready for the next cycle. Repeat the entire process. You will notice that the average solution quality of the population improves from generation to generation. This is the basic process of a Genetic algorithm based solution.

One more point before we close our design of a GA solution for TSP. Crossover only passes on existing information to a child chromosome; it does not try anything new. For example, if our initial population never contained the sequence 1,5,2 in that order, and if we just use crossover we will never get a solution with this sequence. May be the optimal solution has this sequence. So crossover by itself is not enough to explore a solution space. This is the role of mutation. Mutation introduces random changes in a solution. Normally these are small changes, so as not to drastically modify the solution. An example of mutation in our TSP example would be to swap two cities in a solution (1,2,3,4,5 becomes 1,2,4,3,5 for example). Crossover combined with mutation provides a balanced set of operators for a GA based approach.

Be Careful Though...

Based on the discussion above, a GA based approach may look very simple and adhoc. The basic GA model is indeed very simple. Note that the model is also quite general. You choose a chromosome representation, design crossover

and mutation procedures, and provide a fitness-evaluation function and the above model can be used. So why is it that not everyone is using GA for every problem?

The model is "deceptively" simple. On the one side, GA has no magical powers to convert an unsolvable problem into solvable class. It has some strengths which enable it to perform better than other approaches for many problems. Recall the adaptivity and local optima problems mentioned earlier. Since we have a population of solutions at any time, the chance of all of them getting stuck on a local peak is remote - just like a team of people exploring rather than a single person. This is one big strength.

Adaptivity is a relatively unexploited characteristic, though quite powerful. Suppose you run the program for a few generations, and got a few very good solutions. Suppose the environment now changes suddenly. In our case, this means the evaluation function changes. What was good a generation earlier is no longer good! But, GA has no big problem in adapting. Unlike other models, GA does not remember what changes the various individuals in the population has gone through. So it continues to give opportunity for the changes to be undone as much as for fresh changes to come in. And it starts rating solutions based on the new evaluation function. Over a period, the system will generate solutions which are good in the new environment. In dynamic enviroments this will be a major asset.

One major drawback is the blackbox nature of GA. It is difficult to see how it works. Though there are mathematical formulations using what is called "schema theorem" (we will not discuss it here), they are still not rich enough to be of use practically. Selection of a good representation, operator, etc are still an art. Normally there are a plethora of options available for these. There are few practically useful guidelines. On the other hand, the apparent simplicity of GA model encourages people to try it out without deep thought on all the relevant issues. For example, how big a change can mutation make? How do you select between a few available crossover operators? Is it good to have multiple crossover and mutation operator, or a single one of each? Are both types essential in all cases? What fraction of solutions should be mutated? What is a good population size for a given problem? What criteria should a good representation satisfy? There are quite a few such questions which are today answered largely empirically. But, the answers are important to get an algorithm that works. Blind application can result in random exploration of the huge search space, and frustrating wait!.

Also remember that GA can never guarantee optimal solution. The results so far only shows that GA does better than most commonly used algorithms and that it is a general purpose algorithm which can quickly adapt to particular problem characteristics. In general, one need to run GA multiple times and choose the best solution obtained.

Conclusion

GA as a solution model has been around for a few decades now. More and more problems covering variety of areas from natural language processing and neural network design to scheduling problems are being attempted through GA. There are even attempts to evolve computer programs through this approach - a field called evolutionary programming. The field has a large amount of adhocism - without adequate attempt to understand the system. There is some scattered work in developing mathematical abstractions and formal analysis. This is an active area of research. There is also substantial interest in exploiting parallelism on shared and distributed memory machines, cluster environments, etc. The inherently parallel nature of evolution makes it capable of exploiting large computing power; however, there are complex research issues regarding the effectiveness of large scale parallelism in GA.

In this article, we have touched upon only the major components and parameters. There are quite a few more as one gets closer to an implementation. They are not difficult to

understand, in purpose and significance, once the basics are understood. When properly understood and carefully applied, GA is a valuable tool for solving a number of practical complex problems.

REFERENCES

- An introudction to genetic algorithms. D Beasely, DR Bull, and RR Martin. Vivek Vol 7 (I), January 1994.
- Research topics in genetic algorithms. D Beasely, DR Bull, and RR Martin. Vivek Vol 7 (2), April 1994.
- Handbook of Genetic Algorithms. Lawrence Davis (ed). Van Nostrand Reinhold, 1991.
- Genetic Algorithms in search, optimisation and machine learning. DE Goldberg. Addison Wesley, 1989.
- Genetic algorithms + data structures = evolution programs. Zbigniew Michalewicz, Springer-verlag, 1992.
- Useful websites :- http://www.genetic-rogramming.org/ and http://www.illigal.ge.uiuc.edu/index/php3 (IlliGAL project) These websites provide pointers to a number of other relevant sites, current research work, applications and related areas.
- John Holland, "Genetic Algorithms", Scientific American, July 1992.