A Genetic Search In Policy Space For Solving Markov Decision Processes

Danny Barash
Department of Applied Science
University of California, Davis and
Lawrence Livermore National Laboratories
Livermore, California 94550
email: barash@vsattui.llnl.gov

Abstract

Markov Decision Processes (MDPs) have been studied extensively in the context of decision making under uncertainty. This paper presents a new methodology for solving MDPs, based on genetic algorithms. In particular, the importance of discounting in the new framework is dealt with and applied to a model problem. Comparison with the policy iteration algorithm from dynamic programming reveals the advantages and disadvantages of the proposed method.

Introduction

Computational issues involved in making decisions under uncertainty are of interest in a variety of fields and application domains. From inventory control problems in operations research to robotic navigation in artificial intelligence, a wide range of problems can be tackled and solved using efficient search strategies.

A Markov Decision Process (MDP) [Howard 1960, Puterman 1994] is a model to solve sequential decision making under uncertainty. Search techniques which are used in this framework have been developed and investigated for over 40 years, since the invention of dynamic programming [Bellman 1957]. Recent reviews on the complexity and hierarchical solution of MDPs can be found in [Littman, Dean and Kaelbling 1995, Parr 1998].

The aim of this paper is to rigorously examine search algorithms based on natural selection and natural genetics [Holland 1975, Goldberg 1989] to solve MDPs. In an earlier paper [Chin and Jafari 1998] on using genetic algorithms (GAs) to solve a small sized finite MDP, no comparisons with existing techniques were reported. Therefore conclusions related to the success of the approach should be taken cautiously. Here an attempt is made to compare between an efficient GA implementation and policy iteration, which is the most closely related iterative method to GAs, on an extended yet small sized infinite-horizon MDP. Such a comparison enables a better understanding of the strengths and weaknesses of a GA approach. These lessons can prove useful when moving towards real-world applications with large state and action spaces, as well as more complicated model structures which are computationally intractable for existing techniques.

The paper is divided as follows. Section II presents the model problem used for the comparison and the dynamic programming approach with which the GA is compared. This section is further extended to include a discussion on discounting which will later reappear in the GA implementation. Section III initiates the GA approach by describing the binary representation of policies. In section IV, the issue of choosing the right fitness function evaluation for the GA is examined. Section V compares several GA evolution strategies and the most efficient one is then compared with the policy iteration algorithm. In section VI, conclusions are drawn with an eye towards the implementation of a GA in larger problems.

Model Problem Presentation

The following model problem is taken from [Russell and Norvig 1995]. An agent is situated in a 4x3 grid-world model environment. The start state is in (1,1) and the goal state is in (4,3). The goal is to find an optimal MDP policy to reach the goal state using a set of available actions. All details and parameters are taken to precisely match the problem as described in the reference mentioned above.

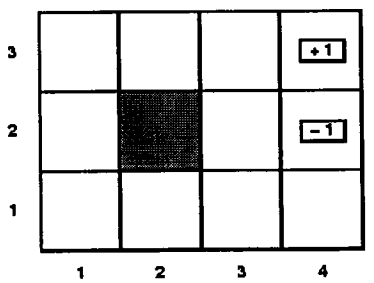

Figure 1: The 4x3 grid-world from [Russell and Norvig 1995]

The following notation will be used to describe the problem, its optimal policy solution and values of the states for the given optimal policy:

$$\begin{pmatrix} \star & \star & \star & +1 \\ \star & W & \star & -1 \\ \star & \star & \star & \star \end{pmatrix} \Rightarrow \begin{pmatrix} \rightarrow & \rightarrow & \rightarrow & +1 \\ \uparrow & W & \uparrow & -1 \\ \uparrow & \leftarrow & \leftarrow & \leftarrow \end{pmatrix}$$

$$\begin{pmatrix} 0.812 & 0.868 & 0.918 & +1 \\ 0.762 & W & 0.660 & -1 \\ 0.705 & 0.655 & 0.611 & 0.388 \end{pmatrix}$$

In this notation, W stands for wall, arrows $\rightarrow, \uparrow, \downarrow, \leftarrow$ are used to label all possible actions, while $\star$ denotes any of these actions.

In a more abstract mathematical formulation, an MDP is a 4-tuple, (S, A, T, R). S is a finite set of states s, A is a finite set of actions a. T is the transition model for the system, a mapping from $SxAxS$ into probabilities in $[0, 1]$, in particular $T(s, a, s') = P(s' \mid s, a)$. R is the reward function which maps from $SxAxS$ to real-valued rewards.

A policy π for an MDP is a mapping from states in S to actions in A. The value function V maps from elements of S to real values. The Bellman equation [Bellman 1957] from dynamic programming establishes a relationship between $V_\pi(s)$ and V_π for other states in the model:

$$V_\pi(s) = R(s, \pi(s)) + \beta \sum_{s'} T(s, a, s') V_\pi(s') \qquad (1)$$

Where $0 \leq \beta < 1$ is known as the discount factor. Discounting arises so that the sum in the equation above will converge to a finite amount in infinite-horizon models, as well as other reasons mentioned in [Puterman 1994, Russell and Norvig 1995]. Towards the end of this section, an additional reason will be examined.

The optimal policy will be labeled as π^* and optimal value function as V^*. Since the optimal policy assigns the best action to every state, the goal is to find an optimal policy π^*, characterized by actions a, such that:

$$V^*(s) = \max_a [R(s, a) + \beta \sum_{s'} T(s, a, s') V^*(s')] \qquad (2)$$

Two well known iterative procedures which aim at finding the optimal policy, based on the above equation, are value iteration and policy iteration. Since it is more logical for a GA implementation to search in policy space, the policy iteration algorithm is used for the comparison. Policy iteration first picks a policy, calculates the values of each state given that policy (value determination step), then updates the policy at each state using the values of the successor states (policy improvement), until the policy stabilizes.

The value determination step can be implemented by either a successive approximation, or directly by solving a set of linear equations. The latter can exploit the sparseness in the system, hence it is chosen for the model problem implementation. The Bellman equation for policy π can be written for each state in S, producing the following linear system of equations in a matrix form:

$$\left(\begin{array}{ccc|c} \beta T(s_1, \pi(s_1, s_1)) - 1 & \beta T(s_1, \pi(s_1, s_2)) & \dots & R(s_1) \\ \beta T(s_2, \pi(s_2, s_1)) & \beta T(s_2, \pi(s_2, s_2)) - 1 & \dots & R(s_2) \\ \vdots & \vdots & \ddots & \vdots \end{array}\right)$$

Attention is now turned to the discount factor β. In [Russell and Norvig 1995], for simplicity reasons, the Bellman equation does not contain discounting. While setting $\beta = 1$ does not cause a problem in the original model, consider the following slight modifications to the model:

$$\begin{pmatrix} \uparrow & W & \star & +1 \\ \star & \star & \star & -1 \\ \star & \star & \star & \star \end{pmatrix} or \begin{pmatrix} \uparrow & \uparrow & W & +1 \\ \star & W & \star & -1 \\ \star & \star & \star & \star \end{pmatrix}$$

In both cases the policy iteration, which without loss of generality starts in the model problem with an initial policy in which all arrows are lined upwards, will fail in the first value determination step. Without discounting, conservation of probability for each state s_i holds:

$$\sum_{k=1}^{N} T(s_i, \pi(s_i, s_k)) = 1 \qquad (3)$$

So that the sum of each row in the left-hand side of the system amounts to zero. In two cases, this can lead to no solution of the linear system and therefore $0 \leq \beta < 1$ is required to break the conservation. In the first case, corresponding to the left modification of the above model, one state interacts only with itself. The state (1,3) will produce a Bellman equation which has no solution. In the second case, shown in the right, only two states interact with each other. The states (1,3), (2,3) will create two Bellman equations that contradict each other, which again results in no solution. To avoid destructive two-states interactions as well as one state interacting with itself, a discount factor $\beta = 0.99$ is added from now on throughout the paper to the model problem. However, at the price of fixing cases of destructive interferences, the discounted model is different than the original model and the optimal policy, when calculated, will differ. For the $-1/25$ penalty in R, as in [Russell and Norvig 1995], the optimal policy for the state (3,1) will cease to be conservative: it will no longer recommend taking the long way around in order to avoid the risk of entering

(4,2). Therefore, along with introducing $\beta = 0.99$, the penalty is decreased to $-1/50$, resulting in the same optimal policy as in the original model.

The above discussion on discounting is highly relevant to the GA implementation and will further be discussed when formulating the fitness function.

Representation

The first task in a GA is to code the decision variables into binary strings. It is customary to work with a binary representation for which the schema theorem [Goldberg 1989] applies. Therefore, the search space which is the most natural to use in an MDP is policy space, rather than values of states which are a collection of real numbers.

In the model problem, there are four possible actions in A. These can be coded into $10, 11, 00, 01$ for $\rightarrow, \uparrow, \downarrow, \leftarrow$, respectively. A policy can then be constructed by concatenating the various actions at each state, to form a string of binary bits of length $2 * N$, where N is the number of states. Such a binary string to represent a policy is called a chromosome.

Out of eleven states in the 4x3 example, two are forced to choose a prescribed action. States $(4,2), (4,3)$ will always choose their action to be $\uparrow$, therefore each chromosome is of length $2 * 9 = 18$. Indexing is chosen to start at state (1,1), then move right to (4,1), then upwards to (1,2), then right to (3,2), repeatedly until termination at (3,3). For example, the optimal policy π^* is represented by the chromosome [110101011111101010]. Solely for investigative purposes (figure 2 in the next section), a real number representation of chromosomes was included in the simulation. For that reason, all possible binary combinations ($2^{18} = 262144$) were taken between -13.1072 and 13.1072, with a grid spacing of 0.0001. In that case, the optimal policy π^* amounts to 8.80 when converted to a real-number.

Fitness Function Evaluation

For each representation of a policy by a chromosome, a fitness value is assigned. Constructing the best fitness function, which maps from policies to fitness values, so that the GA evolves efficiently, is of central importance in the implementation.

The value determination step of policy iteration produces a vector V of N elements, each of which contains the value $V(s_i)$ at state s_i. Taking the separable value function to be additive, the fitness value of a policy π can then be formulated by summing over all elements of V:

$$f(\pi) = (1/N) \sum_{i=1}^{N} V(s_i) \qquad (4)$$

In an MDP, in general, an objective function has several choices. Among them are expected average reward (reward per step), expected total undiscounted reward and expected total discounted reward. For reasons already mentioned earlier, expected total discounted reward is mostly favored for the GA implementation when using value determination. In addition to the previous example in the presentation section, about two states interacting with each other to produce no solution in a slightly modified version of the original model when using policy iteration, all policy space is vulnerable in a GA search. Below is an example of a configuration in the original model, which does not arise in policy iteration, that needs to be avoided in the GA implementation by way of discounting:

$$\begin{pmatrix} \star & \star & \star & +1 \\ \downarrow & W & \star & -1 \\ \leftarrow & \star & \star & \star \end{pmatrix}$$

The two states which are interacting destructively in this example are (1,1) and (1,2). Therefore, $V(s_i)$ is chosen to be calculated by value determination using a discounted model.

Figure 2 illustrates how the fitness function, given in equation (4), is noisy and unpredictable. Policies are represented in the horizontal axis by converting binary strings into real numbers between -13.1072 and 13.1072 (although throughout the GA evolution, they obviously evolve as chromosomes in a binary form):

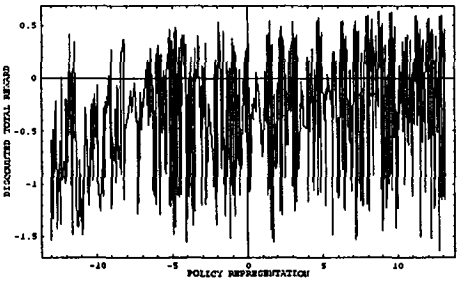

Figure 2: Fitness function used in the GA simulation

It is therefore evident that for searching the whole policy space, without favoring any certain subsections of the space initially, GA is perhaps the most efficient technique to locate the global maximum in order to find the optimal policy.

Evolution Strategies

Finally, an initial population of random policies and their respective fitness values are evolved for several generations using selection, reproduction, crossover and mutation, until convergence to the optimal policy. Figure 3 illustrates some of the GA terminology used so far.

The genetic operators which are used to solve the model problem, as well as the chosen GA parameter

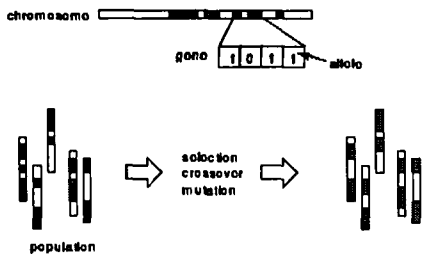

Figure 3: GA basic terminology

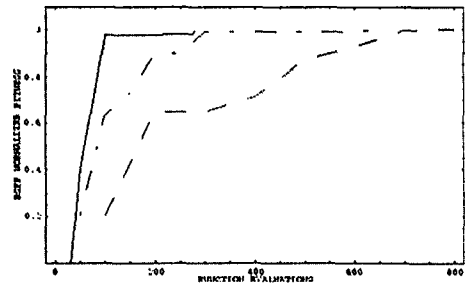

Figure 4: Best-So-Far-Fitness GA performance evaluation: $N_p = 100$ (dashed), $N_p = 50$ (dot-dashed), $N_p = 5$ (solid).

values, are the ones which are commonly used [Goldberg 1989] to search for a global maximum in a noisy function similar to the one in figure 2. A modern GA implementation by [Carroll 1996], which is freely distributed, was modified to handle the 4x3 model-environment, as well as representation of policies and value determination by means of gaussian elimination for the fitness function evaluation.

The above implementation includes a variant of GAs which allows a small population size to be evolved. Typically the population size is set to be at least $N_p = 50$. However, an approach called Micro-GAs (μGAs) [Krishnakumar 1989] works with a small population size of $N_p = 5$. It is known in practice to reach a near-optimal region faster than the simple GA (SGA), in terms of the number of function evaluations required, in certain problems where average effects are of secondary importance. It aims at finding the optimum as quickly as possible through the best-so-far string without improving any average performance.

In both SGA and μGA, selection is done based on tournament selection and uniform crossover is used. However, no mutations are necessary in μGA since there is a constant infusion of new schema at regular intervals. It is also worthwhile mentioning that sensitivity experiments on this problem, by slightly modifying basic GA parameters, yielded no better results.

In figure 4, SGA for two typical population sizes ($N_p = 50$, $N_p = 100$) and μGA ($N_p = 5$) are compared in terms of function evaluations (the number of generations required for convergence, multiplied by population size). It turns out that for searching policy space in the 4x3 model problem, μGA works best performancewise. Convergence to the optimal policy was achieved after 8 generations for $N_p = 100$, 13 generations for $N_p = 50$ and 56 generations for $N_p = 5$, resulting in the least amount of function evaluations ($56*5 = 280$) when μGA was used.

Using the policy iteration algorithm, in which the same value determination as in the GA is performed in each iteration, only 5 iterations were needed to reach the optimal policy. Policy improvement is therefore surprisingly more efficient than a GA for a problem with a small state space.

Finally, it is instructive to examine the intermediate steps before reaching the optimal policy in both methods. In policy iteration, after the first iteration (20% of total calculation), the policy at hand is quite close to the optimal: only three actions in three states are needed to be adapted in consequent steps. No 'illogical' policies are considered, such as an action somewhere pointing to a wall with no apparent reason. In addition, policies are consistently improving at each step. In μGA, after 11 generations (roughly 20% of total calculation), the best-so-far policy differs from the optimal one by four actions (three of which point to a wall), clearly not as good as policy iteration in the early stage. Also, at a later generation it goes downhill, from what seems to be a good policy to a lesser one, in order to emerge with an improved policy after one more generation. This behavior is attributed to the different nature of the two methods. With respect to certain 'illogical' actions, perhaps restricting the search space in the GA implementation rather than leaving it flexible can yield a better performance, if such a strategy proves to be cost effective.

Conclusions

A GA approach was implemented to solve an MDP model problem with small state and action spaces. It was found that a discounted model is required in order to search policy space using value determination steps. The approach was optimized to perform in the most efficient manner, using a GA variant called μGA.

Comparison with the policy iteration algorithm reveals that for relatively small problem sizes, policy iteration outperforms the GA. Larger examples reveal that the inefficiency of GAs become even more pronounced, since policy iteration converges in a surprisingly small number of iterations. Policy iteration only examines a small portion of policy space right from the beginning whereas the GA starts initially with

a random distribution covering all space. Therefore, the workload the GA is facing is by far heavier than that of policy iteration.

On the other hand, it should be noted that in very large state spaces, it is no longer possible to represent a policy as a simple vector mapping states to actions. The length of such a vector becomes too large, value determination steps become tremendously expensive, therefore building an accurate model becomes impractical. In such problems, policy iteration in its current form can not be used. One approach to try and tackle huge state spaces is to use a compact representation of a policy, by mapping a partial description of the state to actions. The fitness of the policy representation can then be evaluated by measuring the actual (or simulated) performance from some initial condition. Genetic operators on the compact representation of the policy were used [Moriarty and Langley 1998] to improve performance using the SANE (Symbiotic, Adaptive Neuro-Evolution) reinforcement learning method. No comparison to dynamic programming methods is possible in simulations of this kind.

To conclude, based on the comparison in this paper, it is unlikely that searching policy space by using GAs can offer a competitive approach in cases where the policy iteration algorithm can be implemented. Evolutionary algorithms offer an approach, combined with neural networks, to search very large state spaces in cases where dynamic programming methods are no longer valid. While further advancement on the evolutionary track is to be expected, it remains a challenge to develop methods which are based on policy iteration for making decisions in large state spaces.

Acknowledgements

Thanks to Ronald E. Parr for the helpful clarifications and feedback throughout the paper, as well as commenting on an early draft before submission. His MDP code distribution in Common Lisp is used for the comparison with dynamic programming. Work in Livermore was performed under the auspices of the University of California at Lawrence Livermore National Laboratory through Department of Energy Contract W-7405-Eng-48. Computing resources were supported by NPACI (National Partnership for Advanced Computational Infrastructure) using the NOW (Network of Workstations) at UC Berkeley under the fine hospitality of Ann E. Orel and W. Hugh Woodin.

References

Bellman, R. 1957. *Dynamic Programming.* Princeton University Press.

Carroll, D.L. 1996. Chemical Laser Modeling with Genetic Algorithms. *American Institute of Aeronautics and Astronautics* 34(2):338.

Chin, H.H., Jafari, A.A. 1998. Genetic Algorithm Methods for Solving the Best Stationary Policy of Finite Markov Decision Processes. In *Proceedings of the Thirtieth Southeastern Symposium on System Theory.* Morgantown, West Virginia.: IEEE Press.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems.* Cambridge, Mass.: MIT Press.

Howard, R.A. 1960. *Dynamic Programming and Markov Processes.* Cambridge, Mass.: MIT Press.

Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, Mass.: Addison-Wesley.

Krishnakumar, K. 1989. Micro-Genetic Algorithms for Stationary and Non- Stationary Function Optimization. *SPIE: Intelligent Control and Adaptive Systems* 1196:289.

Littman, M., Dean, T.L., Kaelbling, L.P. 1995. On the Complexity of Solving Markov Decision Problems. In *Proceedings of the Eleventh Conference on Uncertainty In Artificial Intelligence.* Montreal, Canada.: Morgan Kaufmann.

Moriarty, D., Langley, P. 1998. Learning Cooperative Lane Selection Strategies for Highways. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* Madison, WI.: AAAI Press.

Parr, R. E. 1998. Hierarchical Control and Learning for Markov Decision Processes. Ph.D. diss., Dept. of Computer Science, University of California at Berkeley.

Puterman, M.L. 1994. *Markov Decision Processes.* New York, NY.: John Wiley and Sons.

Russell, S.J., Norvig, P. 1995. *Artificial Intelligence: A Modern Approach.* Upper Saddle River, N.J.: Prentice Hall.