Genetic Programming as Policy Search in Markov Decision Processes

Chris Gearhart
P.O. Box 11491
Stanford, CA 94309
cmg33@stanford.edu

ABSTRACT

In this paper, we examine genetic programming as a policy search technique for planning problems representable as Markov Decision Processes. The planning task under consideration is derived from a real-time strategy war game. This problem presents unique challenges for standard genetic programming approaches; despite this, we show that genetic programming produces results competitive with standard techniques, albeit with certain trade-offs.

1. Introduction

The problem of planning under uncertainty is afflicted by what Bellman famously called the "curse of dimensionality" [1]. In a setting where a single agent attempts to make optimal decisions in a stochastic environment, the curse refers to the fact that the number of possible states of the environment, many or all of which must be considered as possible immediate or long-term outcomes of the agent's actions, grows exponentially in the number of variables used to describe the environment. Thus, naï ve representations and solutions of stochastic planning problems, such as the framework of general Markov Decision Processes (MDPs), quickly become intractable as the size of the problem description increases.

Recent work [2,4], however, has resulted in the Factored Markov Decision Process (FMDP) approach, which can efficiently solve problems with large but structured state and action spaces by decomposing a planning system into a set of small subsystems that have limited interactions with each other. The cost of planning in this way is often exponentially less than the cost of planning naively. Furthermore, while the assumption of decomposability rarely obtains in interesting domains, it is thought to often be "nearly" true, and therefore this approach is often an effective approximation.

However, FMDP techniques suffer from more subtle problems. In particular, quality approximations often require problem-specific knowledge from the user that encodes information about the form of the solution. The necessity of human expertise prevents the FMDP framework from being an automated planning solution.

As an alternative, we propose the use of genetic programming (GP) [5] as a method for directly obtaining MDP solutions. GP is a generic framework that does not require problem-specific information, unlike standard FMDP techniques. We show that policy search in MDPs is a challenging task for GP to address, but that competitive and substantially more efficient results can be obtained if the robustness of the solution is unimportant.

In the following section, we outline the MDP framework and address its shortcomings, along with those of its factored variant. In section 3, we discuss several issues that arise in employing GP as a method for policy search. In section 4, we present a challenging problem based on an aspect of a real-time strategy war game. Finally, in sections 5 and 6, we describe a series of progressively superior approaches to the problem and present their results.

2. Markov Decision Processes

Markov Decision Processes are a general framework for representing stochastic planning problems. An MDP is a 4-tuple (X, A, R, P) where X is a finite set of states; A is a finite set of actions; R is a reward function $R : X \times A \rightarrow \Re$; and P is a Markovian transition model such that $P(x' \mid x, a)$ is the probability of transitioning from state x to state x'

after taking action a. Thus, an agent (or set of agents) responds to the state of the environment by taking some action. Note that the full set of actions A is available regardless of the environment state. This action, in turn, has a stochastic influence on the state of the environment as specified by P. The Markovity of P implies that future states are independent of previous states given the current state and action. The agent receives a reward depending on the state it encounters (and, in general, on the action it takes from that state; for our purposes, however, we will assume that the reward depends only on the state). Our goal is to determine a mapping from states to actions (known as a policy) that will maximize the agent's expected long-term reward from any state. We assume that the MDP has an infinite horizon, meaning that the agent never stops acting in the environment, and thus we bound the expected long-term reward by discounting future rewards exponentially by some $\gamma \in [0, 1)$.

The optimal policy can be computed through any of a variety of methods. Many such procedures first compute a value function $V : X \rightarrow \Re$, which specifies the expected long-term value of acting optimally from a particular state. The optimal value function is the fixed point of the Bellman equations:

$$V(x) = \max_a R(x,a) + g \sum_{x'} P(x' | x,a) V(x')$$

Once the value function is computed, the optimal policy $\pi : X \rightarrow A$ can be computed in a greedy fashion:

$$p(x) = \arg\max_a R(x,a) + g \sum_{x'} P(x' | x,a) V(x')$$

Unfortunately, computing the value function is intractable in the general case. The total number of states under consideration is exponential in the number of variables that describe the environment, and the total number of actions that can be taken is exponential in the number of agents being modeled in a multi-agent setting. Thus, even storing the value function or the optimal policy may be infeasible.

For many problems, however, this framework is overly generic. It may be the case, for instance, that the reward function has a compact description, or that some of the state variables may transition in a manner that depends only on a subset of the state variables, or a subset of the agents. Factored MDPs make exactly these assumptions in the hopes of achieving tractability by exploiting structure. Unfortunately, compactness in the reward and transition functions does not entail compactness in the value function; variables that are independent across a single step may become correlated over time. Thus, FMDPs can usually only approximate the value function, generally by restricting attention to a class of compact value functions. Automatic methods of specifying this class so as to ensure a quality approximation given tractable computation are presently unknown. In practice, the approximating class is determined by human expertise or trial-and-error. The quality of the policy obtained depends heavily on the choice of approximating class (usually in a problem-specific way), and this is a significant disadvantage of the approach.

3. GP as Policy Search

Of course, approximating the value function is unnecessary if we can determine a good policy directly. Our purpose in this paper is to apply genetic programming to the problem of directly computing an optimal or near-optimal policy for a Markov Decision Process (which will be factored in our example). In this context, GP becomes a technique for policy search. We are interested in the successful application of GP to this task because GP tends to be more problem-independent than the FMDP solution methods mentioned above. In particular, although GP requires user specification of the function and terminal sets being used, in addition to various run parameters such as the population size, it does not generally require significant information about the structure of the solution. By contrast, specifying an appropriate approximation for a value function often requires user expertise in the problem domain, and the approximation often encodes nontrivial information about the form of the solution. In effect, FMDP solution algorithms often require that the user supply them with information about the solution (such as whether two variables are independent or correlated in their effect on the value function), which is undesirable. GP, on the other hand, could be capable of evolving compact approximate policies for many problems with relatively generic function and terminal sets.

However, MDP policy search also presents several difficulties for GP, which we address but do not fully resolve in the course of this paper. The most significant problem is that of evaluation. In general, the quality of a policy is specified by its corresponding value function. Computing the value of a particular policy is easier than computing the optimal value function (in the former case, the value function is the fixed point of a set of linear equations), but is still intractable if the number of states is sufficiently large. Thus, performing an exact evaluation of an individual in a nontrivial MDP policy search problem is effectively as difficult as solving the MDP in the first place.

For most problems, we must therefore approximate the value function. Using a user-specified approximation class would effectively negate the advantage of using GP, so we must turn to other options. One such method is Monte Carlo simulation. We can place the agent at some initial state and simulate the policy's interaction with the stochastic environment for some fixed number of steps while tracking the total discounted reward received. If we perform several such runs, the mean value (per run) obtained will approach the value of the initial state. Unlike general value function determination, Monte Carlo focuses the computation on probable states at the expense of less probable states. This is generally a mixed blessing. On one hand, it is desirable to take knowledge of an initial state into account, and we can feel reasonably sure about our assessment of the value at that state. On the other, states that are rarely reached receive little evaluation. Thus, Monte Carlo may not be able to distinguish the optimal policy from a policy with a similar value at the initial state but highly suboptimal values at less probable states.

In addition, Monte Carlo typically requires a very large number of runs for reasonable accuracy. The number of runs depends heavily on the overall distribution over states. Clearly, performing a thousand simulations (which is not an unreasonable number for Monte Carlo, depending on the problem) of a possibly complex stochastic environment for each unique individual in a large population over many generations will be infeasible. However, inaccurate evaluation is also hardly an option, unless we want policies to succeed through chance as much as through fitness. Monte Carlo evaluation is in fact used in our approach, and we show a technique for dealing with this problem in subsequent sections.

Another problem is the potential sensitivity of the environment defined by an MDP to subtle changes in a policy. For example, changing a single agent's action in a multi-agent domain requiring coordination between agents may have disastrous effects on the long-term collective reward received. Furthermore, subtle state changes may require dramatic changes in actions. Thus, the value surface over the (very large) space of policies tends to be spiky and discontinuous. As we will show, using a simple policy construction which accepts as input an integer indexing the set of possible states and returns as output an integer indexing the set of possible actions (even assuming that the problem is sufficiently tractable to do so) can easily lead to poor results. However, we also present an improvement that allows for coordination between agents in our multi-agent setting in a way that minimizes the effect of small policy changes.

4. Freecraft Tactical Problem

For our experiments, we used an instance of the Freecraft domain outlined in [3]. Freecraft is a freeware real-time strategy war game. The goal of playing is to control a set of agents and manage resources so as to construct an army and defeat an opponent. The problems introduced in [3] are simplified variants of different aspects of Freecraft, but are sufficiently interesting that the policies obtained from the simple models can be successfully employed as part of an in-game strategy. Furthermore, these simplified problems are difficult to solve by themselves, even for FMDP techniques. Generally speaking, Freecraft problems tend to involve dense, rather than isolated, interactions between variables, so they are not readily factored. Coordination between agents is essential, but no natural coordination structure is implied by the problem. In summary, Freecraft is an excellent challenge problem for MDPs, and we can demonstrate the value of GP as a policy search technique by solving such a problem.

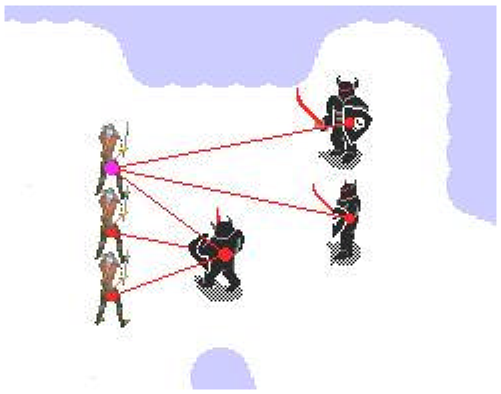

Figure 1: Freecraft Tactical Problem

The tactical problem is a test of agent control; an in-game screenshot of the tactical scenario is shown in Figure 1. The player (or agent, in our example) starts with *n* footmen, and is confronted by an enemy with an equal number of equivalent footmen. Each footman is capable of attacking any individual enemy, and may switch targets at any time. In our model, enemies are assigned distinct and static targets. This is done both for simplicity and to provide structure for a smart policy to exploit. Rewards are achieved for each enemy killed, and the overall goal is to defeat the opposing force as quickly as possible.

More formally, each footman and enemy are associated with state variables representing their health states. Each such variable can take one of five values: *Unhurt, Barely Wounded, Wounded, Badly*

Wounded, and *Dead*. Additionally, each footman's action at each time step is an integer ranging from 1 to n, and specifies which enemy it targets at that time step. Note that any valid policy will automatically target and attack some enemies. This increases the difficulty involved in finding the optimal policy, because the difference in value between the optimal policy and other policies is diminished.

Each living footman and enemy has an 80% chance of striking his opponent in a given time step. If it strikes successfully, it lowers the target's health status by one (unless the target is already dead). If multiple footmen target the same enemy, they each have an independent chance to strike, and each successful strike will lower its health status by one. The reward collectively received by the footmen at each state is equal to the number of dead enemies in that state.

For our experiments, we use $n = 3$ and $\gamma = 0.98$ as a discount factor. Three footmen and enemies do not constitute particularly large state and action spaces, but nonetheless require considerable computational effort for FMDP methods. Indeed, instances with $n = 4$ prove infeasible. Fortunately, the optimal policy with $n = 3$ is sufficiently interesting for our purposes.

5. Methods

In this section, we present a series of GP approaches to the problem under consideration. We begin with relatively simple attempts, analyze their limitations, and refine them to yield progressively better policies.

5.1. Simple Integer Mapping

Our initial attempt was the most general and scalable, and also the least successful. The corresponding tableau is shown in Table 1 below. All terminal and function values are typed as integers. The terminal set consists of two integer inputs to the program, P and E. P is an integer index into the set of all possible player (agent) health states, and E is an index into the set of all enemy health states. Thus, P and E completely describe the Markov state, although in too general a manner to be particularly useful. The function set is a smattering of arithmetic, logical, and conditional functions. Most have direct equivalents in C, with the exception of the protected division operator %. For any integers a and b, $\%(a,b) = a / b$ unless $a = b = 0$, in which case $\%(a,b) = 1$. The ternary operator ? is shorthand for the ?: operators in C; $?(a,b,c)$ is equivalent to the C statement $a ? b : c$. All integers are interpreted as Boolean values according to C conventions (namely, an integer is true if and only if it is nonzero).

Objective:	Find a policy for the Freecraft tactical problem that has the maximum expected value
Terminal Set:	P, E
Function Set:	+, -, *, %, &&, ||, !, ?
Raw Fitness:	Average of 10 Monte Carlo runs of 20 steps each
Standardized Fitness:	150 – Raw Fitness
Adjusted Fitness:	1 / (1 + Std. Fitness)
Wrapper:	Return value interpreted as index into set of all possible actions
Parameters:	Population size = 10000 # of generations = 100 Internal crossover = 0.8 External crossover = 0.09 Reproduction = 0.1 Mutation = 0.01
Depth restrictions:	Initial: 5-10
Success Predicate:	The best-of-run individual

Table 1: Simple Integer Map Tableau

The return value of an individual is interpreted as an index into the set of all possible actions. If the value is lower than the smallest possible index, it is simply set to that index, and likewise if larger than the largest possible index. The raw fitness measure is the average of a small number (in this case, 10) of Monte Carlo evaluations. A loose maximum on the value obtained can be determined mathematically as $(R_{max} / (1 - \gamma))$, which evaluates to 150 in this case. A measure of standardized fitness can therefore be easily obtained. The run parameters were selected to be representative of standard GP parameter values; no special effort went into selecting these values. All initial populations were created with a depth ramp ranging from 5 to 10, and subsequent populations can have individuals of any depth.

The results of this approach (shown, along with all subsequent results, in Table 4 in section 6) are quite poor when compared to the policy computed by FMDP techniques. We also tested whether doubling the population size, number of generations or number of Monte Carlo runs per fitness evaluation had any effect on the value. These changes were ineffectual; the problem with simple integer mapping is more systemic.

5.2. Coordinated ADFs

Simple integer mapping is not robust to small changes in its input or output, and generally does not map well onto the sort of problem that we are trying to solve. Small changes to an input terminal can map onto very different Markov states, and small changes to the output value can map onto very different actions, regardless of the orderings associated with the state and action sets. Furthermore, an attempt to find a single, pure mapping from states to actions is very general, but does not take advantage of any features of the problem under consideration.

Correspondingly, we make two revisions to the simple integer map model. First, we replace the terminals P and E by the full set of state variables in the problem. It is thereby easier to evolve a program that examines particular nuances of the state because they are readily available as separate terminals; they don't need to be picked out of the integer encoding the state. Furthermore, we can view our problem more appropriately as a coordinated multi-agent task. Rather than determining a single joint action for all agents, we can each agent determine its own action separately. This effectively decomposes the problem into a set of much simpler sub-problems, where each requires a considerably smaller program to be solved. However, we must have coordination between our agents (they will often want to ensure that they attack the same target, for instance). We do so by imposing a general coordination hierarchy on the problem: we specify an order in which actions will be determined, and allow each agent to see all actions that were previously determined. Thus, we are interested in evolving a set of separate programs that make use of each other; in other words, we want a set of automatically defined functions (ADFs), one for each agent, without a central result-producing branch.

The resulting revisions to the simple integer map tableau are shown in Table 2. Note that we call each ADF separately when evaluating the overall policy. Agent 1 determines its action and submits it to Agent 2, who takes it into action, determines its action, and submits both actions to Agent 3, who takes both into account in determining its own action. Each individual policy is relatively simple, and we therefore cap the maximum depth of a program at a relatively shallow level.

The policies yielded by this approach are a significant improvement over those computed by simple integer mapping, although they still fall short of the FMDP results.

Terminal Set (for all ADFs):	P1, P2, P3, E1, E2, E3
ADF1 Function Set:	+, -, *, %, &&, ||, !, ?
ADF2 Function Set:	+, -, *, %, &&, ||, !, ?, ADF1
ADF3 Function Set:	+, -, *, %, &&, ||, !, ?, ADF1, ADF2
Wrapper:	Each ADF is interpreted as the value for a particular agent.
Depth restrictions:	Initial: 2-5 Maximum depth: 5

Table 2: Coordi nated ADF Tableau Revisions

5.3. Best-of-last-generation Selection

Up to this point, we have not considered any methods of dealing with the evaluation problem described in section 3. One technique implicitly used in the above approaches is to recompute fitness for an individual each time it is encountered, rather than computing once and caching the result. This is effectively a way of extending Monte Carlo evaluation for individuals that appear often in a population. More precisely, if a given individual appears *m* times in a given population, and each occurrence is evaluated by a distinct Monte Carlo simulation of *n* runs, then the average fitness for that individual in the population (which fundamentally determines its success) is equivalent to the result of a Monte Carlo simulation with *mn* runs. Individuals that benefit (or suffer) from lucky (or unlucky) simulations will be evaluated again in later rounds as a hedge against chance. Throughout the entire process, individuals that succeed will receive increasingly careful scrutiny, which can increase or diminish their success as appropriate.

Given the above, however, choosing the best-of-run individual is a very poor way of selecting the fittest policy. Generally speaking, it selects among decent candidates by picking the one with the most favorable one-time limited-accuracy evaluation. It is much more reasonable to restrict our attention to individuals that survive until the final generation, given the above results. Furthermore, we can perform more accurate evaluations on this limited population, and select the fittest candidate accordingly.

This iteration's tableau revisions are shown in Table 3. Note that we have also doubled both the number of generations (which allows greater confidence in the success of the last generation) and the number of Monte Carlo runs. These changes are allowed by the considerably improved running time of the coordinated ADF approach.

Raw Fitness:	Average of 20 Monte Carlo runs of 20 steps each
Parameters:	# of generations = 200
Success Predicate:	The best of last generation individual, according to the average of 100 Monte Carlo runs of 20 steps each

Table 3: Best-of-last-generation Tableau Revisions

Once again, the resulting policies are an improvement over those of the previous approach, but do not yet match FMDP policies.

5.4. Revised Standardized Fitness

We can achieve better results still by tightening the bound on raw fitness used in computing standardized fitness, thereby increasing the difference in fitness for policies with similar values. In particular, we note that the upper bound of 150 is utopian and cannot be obtained by any policy, since a minimum of 5 steps are necessary before maximum reward can be received. Furthermore, none of our Monte Carlo evaluations extend past 20 steps, where the upper bound of 150 is the infinite limit of reward received. Observing that raw fitness values of 35 and 37 are rather small when compared to this upper bound, and noting that they correspond to adjusted fitness values of 0.00862 and 0.00877, respectively, we can see that they will be considered as almost equally fit in our GP runs. Given that only 20 steps of reward will be considered in each Monte Carlo evaluation, and assuming an ideal case where every footman always hits and every enemy always misses, we can tighten our upper bound to 43, which is much more in line with the received fitness values. Given this new upper bound, raw fitness values of 35 and 37 correspond to adjusted fitness values of 0.11111 and 0.14286, which are far more readily differentiated (the percentage increase between the values has changed from 2% to 29%).

Given this revised standardized fitness, the value of the policies improves to match that of the FMDP policy. Even so, the policies are not equal: the GP policy behaves poorly, as predicted, in states that are rarely reached, where the FMDP policy does not. We discuss this further in the following section.

6. Results

All GP experiments were implemented in C using LIL-GP [6] and were run on a 900 MHz Sun UltraSPARC-III+ workstation. All runs were initialized with the same random seed, implying that two runs with identical population sizes and restrictions have identical initial populations. All values were obtained by 1000 Monte Carlo runs of 20 steps each on the selected candidate. All times shown are in seconds. The FMDP solution is based on a technique presented in [3], and was implemented in C++ and executed on a 700 MHz Pentium III. Our results are reported in Table 4 below.

	Integer Mapping	Integer Mapping Gen. = 200
Value	27.0988	27.9260
Time	10894	36244
	Integer Mapping Pop. = 20000	Integer Mapping Runs = 20
Value	28.3471	27.7703
Time	21305	47147
	Coordinated ADFs	Best-of-last Selection
Value	32.1184	35.8467
Time	1893	3735
	Revised Standardized Fitness	FMDP Solution
Value	37.8953	37.3289
Time	4906	36031

Table 4: Experimental Results

We note, initially, that changing the number of generations, population size, or number of Monte Carlo runs per evaluation has little effect on the values of the policies obtained via the simple integer mapping method. By contrast, working to take advantage of problem structure in the coordinated ADF method yields a noticeable improvement in policy value and an order-of-magnitude improvement in running time. Eventually, the fourth iteration of our approach yields a policy value which is essentially identical to that of the FMDP solution, in roughly 1/7 the time.

This policy, however, is not optimal; unlike the FMDP policy, it selects poor actions in states that are seldom reached. For example, in one Monte Carlo run, the agents initially focus their attacks on Enemy

3. Once Enemy 3 is badly wounded, two of the agents attack another target and leave the third to finish off the nearly dead enemy. However, over the course of these steps, Agent 3 has never been struck by Enemy 3, which has a 0.008 probability of occurrence. After killing Enemy 3, the attacking agent continues to attack his corpse, which is clearly suboptimal. However, in another, more likely run, all agents again initially focus their attacks on Enemy 3. Again, two of the agents attack another target and leave the third to finish off Enemy 3. This time, however, Agent 3 has been struck, and when the attacking agent kills Enemy 3, he moves on to another target. This trend pervades the GP policy, as expected. It is a fundamental flaw of using Monte Carlo for evaluation in this way.

7. Conclusions

Clearly, more remains to be said about the use of GP as a policy search technique for MDPs. Here, however, we have shown that GP using Monte Carlo evaluation and a coordinated ADF structure can effectively and very efficiently match FMDP solutions for difficult coordinated multi-agent problems. GP combined with Monte Carlo exploits knowledge of an initial state in order to produce policies with high overall value in a minimum of time, while MDP solution algorithms expend extra time to produce more robust policies that have high value from any state, however improbable. However, unlike approximate MDP approaches, GP does not require problem-specific knowledge about the value structure, and may be superior when truly automated planning is desired.

Bibliography

[1] R. E. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, Princeton, New Jersey, 1961.

[2] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. IJCAI*, pages 1104-1111, 1995.

[3] C. E. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, Aug. 2003. Morgan Kaufmann.

[4] C. E. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. Accepted in *Journal of Artificial Intelligence Research (JAIR)*, 2002.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, 1992.

[6] B. Punch and E. Goodman. Available at http://garage.cps.msu.edu/software/lil-gp/.