

Policy Search in Reinforcement Learning: A Survey

Ronald V. Bjarnason
Oregon State University
Department of Computer Science
ronny@cs.orst.edu

July 20, 2004

Abstract

We present a survey of policy search algorithms in reinforcement learning. The foundations of reinforcement learning and the historical development of policy search are discussed. Policy search algorithms are divided and examined along three axes. First, we examine the search methodology utilized by the algorithm. Second, we examine the representational structure of the policy. Finally, we examine the types of problems that the algorithms are designed to solve. We conclude by examining practical applications, future trends and other issues that pertain to current day policy search techniques.

1 Introduction

In recent years, a good deal of research focus has been placed on Reinforcement Learning (RL). One of the reasons it has received attention in recent years is its increased application to more general problems [4, 35, 43, 50, 52]. One of the methods that has enabled this success is the use of *policy search techniques*.

Policy search and the use of value functions constitute the two major approaches to solving RL problems. Value function methods determine a value for each state in the environment through satisfaction of some form of the Bellman equation. A one-step greedy search reveals the optimal path after this valuation is complete and reveals a mapping from states to optimal actions. This mapping of states to actions is called a *policy*. In contrast to value function methods, policy search methods explicitly represent a policy and search the policy space through manipulation of this representation.

Policy search methods are generally more applicable to problems with large and infinite state spaces, because they do not require policy representation through state enumeration, as value function methods typically do. The number of states in an environment typically increases exponentially in the number of dimensions representing the environment, and exponentially in the size of those dimensions. Application of value function methods to large problems becomes computationally impossible very quickly. Policy search methods, on the other hand, represent a policy through a set of parameters θ that can be expressed in a number of different ways (see Sec. 4.3) which typically increase linearly in both the number of dimensions and the size of those dimensions. For this reason, policy search methods are often more amenable to problems whose state space is large (or infinite).

The remainder of this paper will proceed as follows: In Section 2 we examine the Reinforcement Learning Problem and give it a proper definition and in Section 3 we examine the dynamic programming techniques and that serve as a foundation for much of the present-day research. In Section 4 we examine

| | | Representation of a State Value Function | |
|----------------------------|-----|--|-----------------------|
| | | No | Yes |
| Representation of a Policy | No | Simple DP | Value Function |
| | Yes | Policy Search | Hybrid Methods |

Table 1: Dividing RL techniques into major categories

policy search techniques based on three different criteria: first, the search methodology used by the specific algorithm, second, the manner in which the policy is represented and third the type of problem the algorithm is designed to solve. We also investigate some methods that are best described as hybrids between the value function and policy search methods in Section 5. In Section 6 we present some practical applications that have successfully utilized policy search methods. Section 7 briefly compares value function and policy search methods. Finally we offer some insight into future development of Policy Search methods in Section 8 and conclude in Section 9.

2 The Reinforcement Learning Problem

Reinforcement Learning (RL) [10, 21, 57] refers to a general class of problems in which an agent is situated in an environment with no knowledge of the transition function or the reward structure of the environment. Only through trial and error can the agent discover how to act. The goal is to discover a policy, π , that will maximize reward over the long run. This definition describes RL in terms of a set of problems rather than a set of algorithms to solve those problems. We think of RL algorithms as any methodology utilized to solve an RL problem.

We differentiate RL techniques based on many different aspects, but for the purposes of this survey, we will divide them according to their use of memory structures to explicitly represent value functions and policies. Table 1 illustrates this methodology. We divide the algorithms along two axes into four categories. We determine if an algorithm utilizes a memory structure to explicitly represent a state value function and/or a policy. This is very similar to other proposed approaches to dividing RL algorithms [26, 33].

We first consider the case in which neither a value function nor a policy is explicitly represented. In terms of a RL algorithm; such may not exist. In general, however, this accurately represents some simple dynamic programming (DP) techniques. Within RL, this would be similar to using DP to determine a scalar distance (manhattan distance) from each state to the goal. This may be considered a crude value function method, but in reality, the algorithm makes no attempt to represent the *true* value of any state based on transition functions and rewards of the environment.

Those algorithms that do not explicitly represent a policy, but do represent a value function are appropriately termed “Value Function” methods. These methods attempt to determine the *true* value of a state, or of a state-action pair based on some form of the Bellman equation, dependent on the transition and reward functions of the environment. When learning is halted, the policy is determined implicitly by the one step greedy lookahead from each state.

Algorithms that explicitly represent a policy, but do not explicitly represent a value function are labeled “Policy Search” methods. In the general case, it would be impossible to enumerate and evaluate the m^n total policies, where $m = |S|$ (number of states) and $n = |A|$ (number of actions). Rather, the policy space is parameterized by a vector $\theta \in \mathbb{R}^n$ that represents a restricted policy space. Even in these restricted spaces, finding the globally optimal policy requires time exponential in the size of the space, and theoretical guarantees offer policies that are only within ϵ of optimal. Other policy search techniques, such as policy gradient search, offer guarantees to converge to locally optimal policies.

Finally, we consider algorithms that explicitly represent a policy and utilize a value function to compute the “true” value of each state in the environment. These methods are hybrids between value function and policy search methods. It is to be noted that many algorithms that are classified as policy search al-

gorithms utilize a value function to determine different aspects of the space, such as the policy gradient. The specific purpose of the value function determines if the algorithm is a policy search method or a hybrid method. If the value function is used to determine the “true” value of a state *independent* of the policy, then these algorithms will be labeled as hybrids. If the value function is generated using the policy, and not used to determine some “true” state value, then the method will be labeled as a policy search method.

3 Dynamic Programming and Value Function Methods

Dynamic programming is a technique first proposed by Richard Bellman to solve the Bellman equation, which is

$$J^* = r + \beta P J^* \tag{1}$$

This equation computes the value of each state $J(s)$ by describing n_s equations with n_s unknown variables. This equation serves as the foundation for value function update equations in RL. Unfortunately, the computation required limits the applicability of DP techniques to relatively small environments.

DP serves as the basis for value function methods and many policy search methods as well. DP methods solve RL problems by estimating the solution to the Bellman equation through a series of “backups”. Each backup consists of adjusting the estimated value of the state according to the Bellman equation. A “full backup” consists of updating the value of every state in the environment. DP methods such as policy iteration, value iteration and temporal difference (TD) methods are value function methods because a value for each state is approximated through these updates.

3.1 Policy Iteration

Policy iteration is a two step procedure that consists of *policy evaluation* and *policy improvement*. Through policy evaluation, the value of each state is determined. States that are more likely to lead to a reward receive higher value. This can be done by performing full backups (according to the current policy) until the estimated state values converge sufficiently. In policy improvement, each action choice is evaluated for each state. If an off-policy action choice will lead to a more valuable state than the current policy choice, the policy is changed at that state from the previous action choice to the more valuable action. Policy iteration consists of repeating the steps of policy evaluation and policy improvement until the policy does not change during an instance of policy improvement.

As a DP technique, policy iteration is guaranteed to find the optimal policy in the limit, but it is a time and space intensive process. For large environments, it may not be possible to even store the values of every state. Even if it is, each policy evaluation step is a time consuming process that requires multiple backups for each state in the environment.

The steps of policy iteration can be generalized to fit different incarnations of policy evaluation and policy improvement, and is the prototype and template for many RL algorithms, including many policy search methods where a policy is explicitly represented [12, 13, 17, 29, 58, 60].

3.2 Value Iteration

One of the main drawbacks of policy iteration is the long policy evaluation step that must be performed during each iteration of the algorithm. The policy evaluation step itself can require several updates to *each* state value. Value iteration is a method that attempts to cut back on this process, while maintaining the convergence guarantee. Rather than perform a full update during the policy evaluation step, the backup process for value iteration involves updating each state only once. This significantly cuts back on the processing time required for each iteration of the algorithm.

While value iteration performs each step in the process at a faster rate, this comes at a cost. At each step the policy does not typically improve as much as it would have, had a full backup been performed. Because of this, value iteration typically must be allowed to run many more steps than policy iteration to see the same improvement.

3.3 Monte Carlo and Temporal Difference Learning

Rather than update every action in the space after each of an agent's trials in the space, Monte Carlo techniques randomly (or by some other policy) choose trajectories through the space and update the affected states when a reward is received. In this manner, certain states are updated after an episode, but not all states, as in value or policy iteration. Temporal Difference (TD) learning techniques are very similar to Monte Carlo methods, but update states in a trajectory *after each step*, based on the current estimated values of the states, rather than waiting for a reward to perform the update. TD methods have played a significant role in the development of RL algorithms, most notably Q-learning by Watkins. [62] Q-learning allows simulation of an environment, without actual knowledge of the underlying model, to compute a provably correct policy.

4 Learning Through Policy Search

4.1 What is Policy Search?

There continues to be some discrepancy regarding the definition of "Policy Search" methods. It appears this discrepancy evolves from the apparent dichotomy of the terms used to define the two classes. "Value function" seems to describe the presence or absence of a value function - independent of its uses in exploration. "Policy search" seems to describe the manner in which exploration leads to a final policy - independent of any sort of representational issues. These two terms seem to be referring to orthogonal attributes of learning algorithms - one regarding memory structures and one regarding search protocol. This is not the case. Both classes can be defined by the methods used to improve a policy. Value function methods improve a policy by evaluating a greedy lookahead based on state values. Policy search methods improve a policy by way of a policy representation, either directly or indirectly. Direct policy search methods [30, 38, 50] clearly belong to the class of policy search methods. Indirect policy search methods, such as those that calculate a gradient [6, 8, 64], or those that utilize a value function in addition to an explicit policy [26, 58] seem to be the cause of this confusion.

The criteria we have adopted rises from two papers co-authored [26, 33] by Tsitsiklis, and bases the split on the use of memory structures that explicitly represent a policy and/or a value function. In [26], Konda and Tsitsiklis present an actor-critic algorithm that utilizes both structures and is classified as a hybrid method. Mannor, on the other hand, [32] splits RL into three categories - model based, model free and policy search, placing the actor-critic algorithms squarely within policy search methods (despite the use of a model). Moriarty [38] includes "dynamic programming, value iteration, simulated annealing and evolutionary algorithms" in his list of "Policy-space search methods". In the end, Moriarty seems to split the field based on the existence of an explicit policy (which would again place actor-critic algorithms squarely in policy search space). Despite this confusion, others [2] have recently begun to adopt Tsitsiklis's interpretation, and it appears this interpretation will soon become a standard.

Recent surveys of policy search methods can be found in [2, 40] and [43].

4.2 Policy Search Techniques

Policy search methods can be categorized in a number of different ways. The most important thing separating policy search methods from other methods in RL is the manner in which the search for a best policy is performed. In this section, we analyze the different methods used to search through the space of policies. Some of these methods are undoubtedly more useful in certain types of environments and with certain structural representation, but in this section, we analyze search method independent of these other factors, which will each be analyzed in later sections.

Policy search techniques are not completely independent of each other. Some would best be described as hybrids between different search methodologies. This section attempts to present a base set of methods that will cover the major approaches used to search the policy space.

Policy Gradient vs. Direct Policy Search There have been two major approaches to constructing policy search techniques. Policy gradient techniques [43] generate some model of the surface of the

policy in an attempt to measure directly or approximate a gradient. Following this gradient, in many cases guarantees convergence to a local minimum. Borrowing a definition from the statistics literature, direct policy search methods [30] do not gather information to construct a model of the system. Specifically, direct methods do not attempt to directly measure or approximate the policy gradient. Instead, they explore policy through direct manipulation of the policy parameters, rather than indirectly by way of changes suggested by a model or gradient. These algorithms include genetic and evolutionary algorithms as well as most of the Monte Carlo algorithms discussed in this section.

Rosenstein [50] characterizes direct policy search techniques as taking advantage of structure in the solution space, while simultaneously characterizing value function methods as those that take advantage of structure in the problem space.

4.2.1 Search Techniques: Policy Gradient Search

Of all the policy search methods, those utilizing a policy gradient have received the largest share of attention in the past years. In a recent survey of policy search methods [43], the author seems to exclude other methods in favor of the gradient methods, illustrating the dominance gradient methods have enjoyed over other techniques in recent years. Another survey of approximate gradient methods can be found in [34].

Grudic and Ungar [15] point out three major reasons policy gradient methods have received attention in recent research. First, the function approximation methods used to represent the policy address the need for generalization in large state spaces [21]. Second, policy gradient methods learn policies with respect to the parameters used to represent the policies. Thus, policy representation grows linearly in the number of parameters used. Third, the great majority of these algorithms are provably convergent, with most of these converging to at least locally optimal policies.

REINFORCE The foundation of policy gradient methods lies with Williams’ REINFORCE algorithm [64]. The class of REINFORCE algorithms are based on adjusting the weights of a neural net (parameters of the policy) according to the rule

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij} \tag{2}$$

where α_{ij} is a learning rate factor, b_{ij} is a reinforcement baseline and $e_{ij} = \partial \ln g_i / \partial w_{ij}$ is the characteristic eligibility of w_{ij} . g_i is the probability mass function that determines the value of the output with respect to the input and the parameters, so that e_{ij} is a measure of how the natural log of this function (g_i) changes with respect to changes in the weights [64].

VAPS Baird and Moore’s VAPS algorithm [6] unifies the value function and the policy search algorithms. The general VAPS algorithm is defined by the following equations:

$$\Delta w_t = -\alpha \left[\frac{\partial}{\partial w} e(s_t) + e(s_t)T_t \right] \tag{3}$$

$$\Delta T_t = \frac{\partial}{\partial w} \ln(P(u_{t-1}|s_{t-1})) \tag{4}$$

in which e is some error function that must be smooth with respect to the weights \vec{w} and T is a trace for the individual weights of the network.

If e is defined as a residual error of a Bellman equation, then VAPS behaves like a value function algorithm, such as Q-learning [62] or SARSA [57]. e can also be defined so that VAPS will attempt to directly maximize the value of the policy by following the gradient, essentially reducing to REINFORCE. The definition of e can also be defined so that VAPS learns as a linear combination of a value function and a policy gradient method.

Actor-Critic Algorithms and PIFA Konda and Tsitsiklis [26] and Sutton *et al.* [58] independently developed other hybrid method designed to reduce the variance typically associated with stochastic gradient ascent methods. In their Actor-Critic (AC) algorithm, Konda and Tsitsiklis utilized a function approximator to sample state values and calculate a gradient rather than sampling the state values directly.

Sutton *et al.* [58] developed a similar method in their Policy Iteration with Function Approximation (PIFA) algorithm. PIFA utilizes a value function in a similar role, but in the context of policy iteration. The *policy improvement* step of PIFA involves adjusting the policy in the direction of a gradient calculated by the approximation of the state values. Both AC and PIFA were shown to speed learning through the reduction of the variance. Unlike VAPS, AC and PIFA are guaranteed to converge to a local minimum. An examination of the need for function approximation to reduce variance, with direct reference to AC and PIFA, is discussed in Section 5.1.

GPOMDP Baxter and Bartlett [8] introduced MCG, an algorithm for approximating arbitrarily accurate estimates of the gradient of a Markov chain. They also demonstrate how this can be applied to *partially observable Markov decision process* (POMDP) in the form of their GPOMDP algorithm. GPOMDP is a simulation based algorithm for generating a biased estimate of the average reward gradient that does not require an identifiable recurrent state. GPOMDP is based on an approximation of the gradient of the average reward, $\eta(\theta)$:

$$\hat{\nabla}_{\beta} \eta(\theta) := \frac{1}{T} \sum_{t=0}^{T-1} r(X_t) z_t(\beta) \quad (5)$$

The variable β represents a discount factor on the eligibility trace z_t and has as a natural interpretation of the bias/variance trade-off. In a companion paper to [8], Baxter and Bartlett [9] present on-line and off-line algorithms based on the gradient estimates generated by GPOMDP. The off-line algorithm utilizes GSEARCH, a line-search algorithm that uses gradient estimates rather than value estimates to bracket the function maximum.

Using a Model to Improve Gradient Search Recent papers comparing results to GPOMDP [8, 9] include Aberdeen and Baxter [1] and Wang and Dietterich [61]. Both rely on models of the environment to reduce variance and speed up the learning process. Aberdeen and Baxter rely on a known model or a simulated model in presenting three new policy gradient algorithms. Wang and Dietterich present Model Based Policy Gradient (MBPG). By calculating a (partial and local) model of the MDP, MBPG can determine the number of times each state will be visited under the current policy. This increases the accuracy of the calculation of the value function and its integration with respect to the policy parameters θ . MBPG is shown to be faster than GPOMDP in MDP environments.

Calculating Gradients Bagnell and Schneider [5] introduced the calculation of a covariant gradient to policy search. Covariant search had previously only been applied to value function methods. Meuleau *et al.* [37] demonstrated the ability to calculate an exact policy gradient (unlike VAPS, REINFORCE, and GPOMDP) by working in the space of the cross product of a POMDP and the policy graph encoding the current policy. In doing so, they found guaranteed locally optimal policies from among a restricted set of policies with finite memory. In a companion paper [36], Meuleau demonstrates that there are some environments in which an approximated gradient method outperforms a method using exact gradient calculations.

4.2.2 Search Techniques: Evolutionary Algorithms

As a direct policy search method, evolutionary algorithms manipulate the parameters of the policy directly without the use of a model of the environment or calculation of a policy gradient. They are the only set of algorithms specifically singled out in the foundational RL survey by Kaelbling *et al.* [21] as searching the behavior space (as opposed to the value space). Evolutionary algorithms (EA) directly

manipulate the representation of the parameterized policy θ through mutation, crossover, inheritance and similar methods to generate agents with policies based on successful policies already present in a given population.

Schmidhuber [53] directly compares evolutionary computation (EC) algorithms to dynamic programming based reinforcement learning algorithms, and cites five advantages that the EC algorithms have, most of them having to do with partial observability. In addition, a number of disadvantages are presented along with possible solutions in the form of Schmidhuber’s *Success Story Algorithm*. Moriarty [38] provides a survey of EC algorithms, including SANE, which evolves a diverse set of a neurons in a network to provide robust solutions to large scale problems, such as Othello and obstacle avoidance. Polani and Miikkulainen [48] added combinatorial search in their EuSANE adaption of the SANE algorithm, to address specific needs of the current population.

Kwee *et al.* added memory to the Hayak4 [7] learning system. Hayak4 is a market-based algorithm in which a collection of agents are allocated resources which they are allowed to use to bid on tasks. If the task is performed well, they are given additional resources and are permitted to generate adapted agents based on their successful skill set. The addition of memory allows Hayak4 to perform well in POMDPs.

4.2.3 Search Techniques: Monte Carlo Techniques

In the absence of a known model of the environment, Monte Carlo methods provide the ability to gain experience by way of random sampling and simulation. Excluding the most remedial dynamic programming methods, such as value iteration and policy iteration, almost all RL methods utilize Monte Carlo techniques to some degree. These methods can be used to directly approximate the value of a state, such as in sparse sampling [23] or policy rollout [12] techniques. Monte Carlo methods also play an important role in many of the gradient search algorithms in approximating the true gradient, such as REINFORCE [64] and VAPS [6]. If an algorithm utilizes simulated trials or sampled values rather than exact calculations to gain information, that algorithm is a Monte Carlo algorithm.

In this section, we will examine some of the Monte Carlo algorithms that directly use simulations as their main source of information in calculating modifications for the parameterized policy.

Policy Rollout First introduced by Tesauro and Galperin [59], policy rollout is a technique that approximates the value of a state by taking an action from that state, and following the current policy thereafter for a fixed number of iterations. It does this for each possible action choice in the state, and determines a value based on the rewards it receives from the aggregation of all actions. In this manner, it “rolls out” the current policy from each sampled state. Fern *et al.*[12] utilize policy rollouts in their Approximate Policy Iteration (API) algorithm to estimate the value of states in very large MDPs, such as those designed for planning competitions. In large MDPs, it is not computationally possible to compute the value for each possible state. In API, the values determined by the rollouts are used to construct decision lists, from which relational rules are formed that define the policy. McGovern *et al.*[35] use rollouts to successfully build instruction schedulers for straight line code. Bagnell and Schneider [4] use rollouts to develop a controller for an autonomous helicopter.

Sparse Sampling Sparse sampling is a method used in very large or infinite state spaces. It is a method that assumes at least a somewhat smooth value function over the state space. Kearns, Mansour and Ng [23] present a sparse sampling algorithm that determines the best action (within a bound of ϵ) of a given state by sampling a vanishing fraction of the lookahead tree in time constant with the number of states. The algorithm has exponential performance in horizon time, but has no dependence on the actual size of the MDP.

Selective Sampling Grudic and Ungar [15] present a *selective* sampling algorithm in Action Transition Policy Gradient (ATPG). ATPG assumes that states that are close in the state space and have similar action choices will have similar values, sampling the state-action value function only when the policy changes actions. These selective samples are used to calculate a low variance gradient of the policy.

In [16], ATPG is applied to modal environments. Samples are taken on the mode boundaries, where policy changes are likely to occur. Grudic and Ungar also demonstrate that any stochastic policy can be transformed into a mode switching policy, increasing the applicability of these types of algorithms.

Cross Entropy The Cross Entropy method [11] is a well studied Monte Carlo estimation tool that has recently been applied to policy search in RL [32]. Cross Entropy is a method that generates improved trajectories based on information gathered from prior trajectories. Mannor *et al.* [32] claim the CE method to be faster and more robust than contemporary policy gradient methods.

4.2.4 Search Techniques: Heuristic Search

Littman introduced branch and bound techniques to RL [31] as part of a heuristic search algorithm. Branch and bound techniques systematically search the state space for partial policies that can be excluded based on an upper or lower bound calculation. Hansen [17] utilizes a heuristic branch and bound search to explore the space of policy graphs that is experimentally faster than similar techniques utilizing policy iteration and value iteration. Poupart [49] adapted Hansen’s search to stochastic controllers. His algorithm is able to prune nodes that are dominated by a combination of states, rather than being limited to pruning nodes that are point-wise dominated by a single other state as in [17]. Meuleau [37] implemented a branch and bound algorithm in exploring policy graphs to find optimal policies among a set of policies with a finite memory.

4.3 Representing the Policy Space

Independent of the manner in which the policy space is searched is the structure used to represent the policy. Some of the representational structures partially bias the policy toward certain learning methods. Some of the algorithms presented in this survey do not specify a representational model for the policy. Rather, they state assumptions made regarding its structure, such as differentiability. REINFORCE [64] is one of these algorithms, that specifically states that while it represents the policy as weights in a feed forward network, the REINFORCE algorithm itself is adaptable to any representation that meets the stated criteria.

4.3.1 Policy Representation: Weights in a Connectionist Network

One of the most familiar function approximation techniques is the use of a neural network. The values of the weights combine with the input to determine the network output. The weights between the layers of the network represent the parameters θ of the policy.

Algorithms that rely on gradient ascent of the policy typically rely on differentiability restrictions based on the change of the parameters of the policy with respect to the output of the function. Feed forward networks satisfy these restrictions, and many of the policy gradient algorithms in this survey are presented in the context of neural networks. Williams’ REINFORCE [64] algorithm was first presented in the context of a neural network. Algorithms with similar gradient assumptions include VAPS [6], GPOMDP [8], in addition to [1, 9, 14, 20, 25, 26, 33, 45, 44, 51, 52, 58, 61]

In addition to the gradient methods, Moriarty presents two algorithms, GENITOR and SANE [38] that learn the weights of a neural network through evolutionary algorithms. Polani and Miikkulainen [48] present a modified version of SANE that uses a global search algorithm to select the hidden layer of neurons in a multi-layer feed forward network.

4.3.2 Policy Representation: Finite State-Machine Controllers

Finite state-machine controllers (FSC) and policy graphs are finite automaton wherein nodes (states in the the finite state machine) represent linear functions that are associated with actions [18]. FSCs are a powerful tool for representing policies because they can compactly represent historical context. Two major approaches to learning FSCs have been gradient search and policy iteration.

Policy Iteration for FSCs Hansen [17] introduced an algorithm based on policy iteration, in which the FSC representing the current policy is converted to a value function, on which is performed a value function update. During this policy improvement phase, machine states can be added, pruned or changed. Poupart [49] adapts Hansen’s algorithm to stochastic controllers. In doing so, he improves the node elimination functionality, allowing for elimination of nodes that are dominated by a convex combination of nodes. In Hansen’s version, only nodes that are pointwise dominated by a single node can be eliminated. Poupart’s algorithm also introduces an escape technique designed to aid the controller in escaping from local optima. Both Hansen’s and Poupart’s algorithms are designed to find policies with limited memory and limited number of FSC nodes.

Heuristic Search for FSCs In addition to his policy iteration algorithm, Hansen [17] also presented a branch-and-bound algorithm for learning FSCs. This algorithm outperforms his policy iteration algorithm because it searches the space without the need to perform dynamic programming updates. Meuleau [37] designed a branch-and-bound algorithm for searching the space of the cross product of a POMDP and a FSC representation of a policy, designed to find the globally optimal policy within a FSC of a fixed size, which Meuleau demonstrated to be an NP-Hard problem.

Gradient Search for FSCs Meuleau also presented a policy gradient algorithm [37] designed to find local optima in the same cross-product space, which allowed the algorithm to compute an exact gradient. In a companion paper, Meuleau [36] presented another FSC gradient descent algorithm based on the approximate gradient calculations of VAPS. He demonstrated that in cases of large state spaces and large discount factors, stochastic gradient descent can outperform the exact gradient method. Kim and Meuleau [25] later extended this work to factored MDPs. Aberdeen [1] demonstrated a similar algorithm that is able to solve problems an order of magnitude larger than [37] by utilizing iterative approximations and sparse FSCs.

4.3.3 Policy Representation: Rules in a Language

One of the most straightforward ways to represent a policy is by defining a set of rules over the sets of possible observations and actions. Fern’s Approximate Policy Iteration (API) algorithm [12] represents a policy as rules in a relational language. This set of rules is learned inductively by way of a version of Rivest’s learning algorithm from a decision list generated through policy rollout.

Moriarty discusses two genetic algorithms SAMUEL and ALECSYS [38] that both represent their policies as rules in a simple language. Kwee *et al.* [28] also present a market based evolutionary reimplementation of Hayak4 wherein each agent is defined by only a single rule.

4.3.4 Policy Representation: Programs in a Language

Wiering and Schmidhuber [63] and Schmidhuber *et al.* [54] demonstrate the use of Levin Search (LS) in solving partially observable domains. LS searches the space of programs in a language, finding solutions to search problems in the optimal order of computational complexity. In [54], LS is used in conjunction with *success story algorithm* and in [63], it is used with *environment-independent reinforcement acceleration*.

4.4 Representing the Environment

We have already classified algorithms based on the search methodology and on the representation structure of the policy. One of the most common ways to classify algorithms is based on the types of problems they can solve. Many surveys [2, 18, 40] focus on gathering algorithms designed to solve a similar problem. The standard representation of environments for RL problems is the Markov Decision Process (MDP).

Formally, a MDP consists of a set of states, \mathcal{S} with actions $\{a_1, \dots, a_k\}$ and a next-state distribution, $P(s'|s, a)$, that defines the probability of transitioning to each state s' upon execution of action a from

state s . Also defined is a reward function, $R(s, a) \in \mathbb{R}$ that defines a real-valued reward as feedback for execution of action a from state s .

The definition of the MDP is important in that the reward and the next-state distribution are conditioned only on the current state. This condition is known as the *Markov property*. An agent tracking a history will have no additional information regarding reward or next-state distribution compared to an agent with no memory at all.

Traditionally, RL algorithms are defined in the context of some variant of an MDP. Simple MDPs have very few practical applications, and a multitude of solution strategies available to them. More research has been done in complex variants of the MDP, such as large MDPs, infinite MDPs or Partially Observable MDPs (POMDPs). These variants are more applicable to real-world problems, but have relatively few techniques capable of handling their complexity.

4.4.1 Environment Representation: POMDPs

A POMDP is defined by an underlying MDP, along with an observation distribution, $Q(o|s)$, for each state s , where o is a random variable called the observation made at state s . Unlike typical MDPs, an agent within a POMDP may not be able to discern the true state of the environment from the observations available to it at that state. This is known as *state aliasing*, in which different states will appear to be the same to the agent. Unlike MDPs, agents who accumulate historical information may have an advantage over agents with no memory.

Because of state aliasing, the optimal (Bayesian) policy for a POMDP could be as long as the history of the agent. At some point, the policy must be truncated, if it is to be represented in a finite structure. It is for this reason that policy search techniques seem to be especially well suited for learning in POMDPs. Most policy search algorithms learn in a policy space defined by a set of parameters θ that represent a restricted policy set. The optimal POMDP cannot be represented finitely, and the policy search algorithm cannot typically represent an infinite policy. Both are restricted to finding an approximation of the optimal policy.

Many of the algorithms examined in this survey are suitable for finding solutions in POMDPs. Two of the major approaches used specifically for POMDPs are calculating the Internal Belief State and utilizing a Generative Model. These two methods are *not* mutually exclusive, but rather, represent two independent approaches to solving the same problem. There are many other methods [28, 37, 47, 54, 63] that do not fall directly into these two approaches. Each of these methods have been covered in other areas of this survey. Recent surveys devoted to methods for solving POMDPs can be found in [2, 18, 40].

Internal Belief States Although the state in a POMDP cannot be directly observed, the probability of being in a certain state can be calculated. Let $b(s)$ represent the probability that the system is in state s . b is a vector of state probabilities that is called the *belief state*. When an action a is taken and observation z follows, the successor belief state b_z^a is determined by revising each state probability with the following,

$$b_z^a(s') = \frac{Pr(z|s', a) \sum_{s \in S} Pr(s'|s, a)b(s)}{Pr(z|b, a)} \quad (6)$$

where the denominator is a normalizing factor $P(z|b, a) = \sum_{s' \in S} Pr(z|s', a) \sum_{s \in S} Pr(s'|s, a)b(s)$. The POMDP can be recast as an MDP with a continuous $|S|$ -dimensional state space representing all possible belief states. By recasting the POMDP as a MDP of vectors, dynamic programming methods including value iteration and policy iteration can be utilized to solve the MDP for the optimal policy. Hansen [17] presents a policy iteration and a heuristic search algorithm based on this MDP representation of a POMDP. Extensions of Hansen’s algorithms are discussed in Section 4.3.2.

Aberdeen and Baxter [1] present three policy gradient algorithms that rely on internal state representation of POMDPs. The first relies on a known model to compute to compute the policy gradient. The second two compute internal-state trajectories to stochastically compute the policy gradient and reduce the gradient, respectively.

Generative Models A generative model for a POMDP is a randomized algorithm that given a state action pair (s, a) returns an observation o based on $Q(\cdot|s)$, a reward r based on $R(s, a)$ and a new state s' based on $P(\cdot|s, a)$. Kearns *et al.* [22] address the apparent paradox of having a *fully* observable model of a *partially* observable environment. While it would be possible to generate a (observable state to action) policy based on the model (and underlying MDP), it would be worthless in the actual POMDP setting, because the agent wouldn't know which state it was in. The agent must learn a policy for the partially observable world, as a mapping from past observations to actions, rather than states to actions.

The solutions Kearns *et al.* suggest are based on the generative model. The first uses a strong version of a generative model to construct a *trajectory tree*. This tree is constructed based on a fixed horizon beyond which all rewards are (because of discounting) worth less than some value ϵ . The second algorithm does not construct a tree, but rather calculates returns based on random runs through the environment.

Ng *et al.* [42] utilize a generative model in an algorithm that performs gradient ascent in a Bayes net induced by a "time slice density". Ng and Jordan [41] also present the PEGASUS algorithm which utilizes an idea of a strong generative model similar to that of Kearns [22]. Their *deterministic simulative model* is a generative model of the POMDP without stochasticity. The random number generator must be given to the algorithm. Use of this model can transform any stochastic POMDP into a deterministic POMDP. PEGASUS learns in this transformed space.

4.4.2 Environment Representation: Large MDPs

The application of RL algorithms to large MDP domains is a breakthrough of recent years. In general, RL solution techniques are extremely limited in the size of the domain that they can be applied to. Because of this, experiments are often limited to toy domain problems, and fall far short of real-world application. The 'large' term begins to include domains that are encountered in the real world. Policies for domains in these categories are typically too large to even be enumerated, much less stored and explored individually. Because of this, algorithms exploring these domains are either limited to exhaustively exploring a restricted set of policies, or are limited to finding locally optimal policies rather than globally optimal policies.

Policy search methods do not have the same advantage in Large MDPs over value function methods like they have in POMDP methods. The optimization methods that make policy search method suitable for searching in large state spaces, such as state space factorization, information reuse and sampling methods are also useful to value function methods. Some recent examples of successful value function algorithms that perform well in large states includes [39] and [65], the latter solving a 2-armed pendulum control problem with 75,000,000 states.

Reusing Information Gathering information is costly in any domain. In large domains, it may not be computationally possible to gather information from every state. Many algorithms simulate data for each new policy, discarding previous trajectories and wasting information. Kearns *et al.* [22] pioneer the reuse of old trajectories. Ng developed the PEGASUS [41] algorithm around reusing trajectory information. Peshkin uses likelihood ratio estimation [46, 47] to reuse trajectories from similar policies. Strens and Moore [56] evaluate the PEGASUS method and other paired comparison methods in the mountain car problem and a 12 dimension version of a pursuer/evader problem. The methods are shown to be very effective in the high-dimensional space.

Monte Carlo Sampling The sampling procedures presented in Section 4.2.3 are specifically designed for large and infinite MDPs. Fern's [12] Approximate Policy Iteration (API) described in Section 4.3.3 utilizes rollout to sample states and generate rules based on broad samples across a large domain. Grudic and Ungar's selective sampling algorithm [16, 15] explores only on the boundaries of decisions, leaving most of the state space untouched.

4.4.3 Environment Representation: Infinite MDPs

Kearns *et al.* [23], present a sparse sampling algorithm whose space and time complexity is not bounded by the size of the MDP, but is bounded exponentially by the length of the horizon. Typical algorithms are bounded linearly in both the size of the state space and the horizon. The algorithm determines the best action (within a bound of ϵ) of a given state by sampling a vanishing fraction of the lookahead tree in time constant with the number of states by way of a generative model (see Section 4.4.1).

4.4.4 Environment Classifications: Factored MDPs

Factored MDPs (also known as Dynamic Bayes Net-MDPs) exploit the structure of the MDP so that it can be represented compactly without loss of information. Unfortunately, most MDPs cannot be factored. For those algorithm that are factorable, policies can be represented with a Finite State-machine Controller. These algorithms are presented in Section 4.3.2.

4.4.5 Environment Representation: Semi MDPs

Unlike typical MDPs, Semi MDPs (SMDPs) take into account the time spent at each time step. This makes the SMDP representation much more applicable to some RL problems where average reward is important, but based on a variable time step. Gosavi [13] presents an average reward reinforcement learning algorithm based closely on policy iteration for solving Semi-Markov Decision Problems (SMDPs), which take into account the time spent in each state transition.

5 Combining Value Functions and Policy Search

In Table 1, we illustrated a methodology for dividing RL algorithms into four categories along two axes. Two of these groups comprise the majority of RL algorithms: methods that use value function and methods that use policy search. There are a small group of algorithms that comfortably span these two groups. In this section we examine these groups and the manner in which they can act both as a value function and a policy search algorithm

5.1 Actor-Critic Methods

Actor-Critic (AC) methods are simply value function methods with an additional structure to store an explicit policy - the actor. The critic typically takes the form of some state value function that evaluates the actions taken by the actor. The critic does not have the power to adjust the policy of the actor, only to give feedback in the form of a differential evaluation. The actor receives the feedback and adjusts the policy accordingly. [26, 27]

Konda and Tsitsiklis [26, 27] and Sutton *et al.* independently developed the AC algorithm and Policy Iteration with Function Approximation (PIFA), respectively. Each of these algorithms utilizes a function approximator of the state space (value function) and a policy independently in searching for a good policy. In the AC algorithm, a TD learner with linear approximation updates a policy in the direction of the gradient (calculated by the TD learner). PIFA takes the form of policy iteration in which *policy improvement* takes a step in the direction of the gradient rather than a greedy step. This gradient is provided by a function approximator independent of the mechanism generating the parameterized policy. Both Actor-Critic algorithms and PIFA are proved to be convergent to local maxima and are designed to reduce the high variance commonly associated with policy search algorithms.

Grudic and Ungar [15] critiqued the notion (addressed by AC and PIFA) that a function approximation of the state values is needed to reduce variance in the learning process. Rather than utilizing a function approximator, their ATPG algorithm selectively samples the state space on the boundaries of policy choices. The demonstrate that both ATPG and PIFA outperform REINFORCE by a factor of no less than 10 on each trial, and that ATPG outperforms PIFA on each trial, demonstrating that variance can be reduced and learning can be improved without a function approximation of the state values.

5.2 VAPS

Baird and Moore’s Value and Policy Search (VAPS) algorithm [6] is a hybrid algorithm in a different sense than either AC [26] or PIFA [58]. Both AC and PIFA have two separate mechanisms, one for the ‘actor’ or policy and another for the ‘critic’, recommending adjustments to the actor. VAPS has but a single mechanism that can simultaneously behave as both an actor and a critic.

VAPS uses a connectionist framework in which the weighted connections are updated to reduce the error function shown in Equation 3. Depending on the value of e (from the equation), VAPS can act like a value function or like a policy search algorithm. If e is designed to represent a form of Bellman residual, such as

$$e_s = e_{SARSA}(s_t) = \frac{1}{2}E^2[R_{t-1} + \gamma Q(x_t, u_t) - Q(x_{t-1}, u_{t-1})] \quad (7)$$

VAPS will emulate a value function algorithm in which the network will adjust to satisfy the Bellman equation. If e represents the reinforcement received, such as

$$e_r = e_{REINFORCE}(s_t) = b - \gamma^t R_t \quad (8)$$

then VAPS will follow the gradient, essentially reduced to the REINFORCE [64] algorithm. VAPS can also be designed to satisfy a linear combination of the two such as

$$e_{sr} = (1 - \beta)e_s(s_t) + (\beta)e_r \quad (9)$$

in which VAPS satisfies a portion of the value function and a portion of the policy search methods *at the same time*. The value β adjusts the tradeoff between the value function and policy search aspects of the function. Like AC and PIFA, VAPS works to reduce the variance. In some environments, the hybrid version of VAPS has been shown to outperform both of the strict methods at the extremes of its linear-combination hybrid approach.

5.3 Two Step Methods

Roy and Thrun [51, 52] present a different type of hybrid algorithm that takes advantage of some of the strengths of policy search and value function methods. They developed a two-step algorithm in which a value function algorithm is used to learn a low dimensional problem. The solution to this problem is used as a seed to solve a similar higher dimensional problem with policy search. The value function method is efficient in the lower-dimension space. The solution is projected into the higher dimension space where a policy gradient method finds a local maximum. This is shown to succeed in a robot navigation domain where other conventional methods fail.

6 Practical Applications

One of the major reasons policy search techniques have received the attention they have is their recent demonstrated ability to learn good policies in practical applications. Unlike the synthetic applications, practical applications generally have high dimensionality and large state spaces. Some synthetic problems, such as those investigated in [12] and [56] do exhibit these properties. Despite this, there is something to be said for algorithms that really do work in real life on real machines acting on real experience. In this section, we highlight some of the algorithms that have demonstrated success in real world applications.

Robotic Control Within the past few years several policy search application for robotic control have surfaced. Bagnell and Schneider [4] utilized a policy rollout technique to learn autonomous control of a radio controlled model helicopter. After learning on a simulator with 14 dimensions, the controller was tested on Carnegie Mellon’s autonomous R-50 helicopter. Results were very encouraging, demonstrating an ability to track moving set points and resist strong wind gusts, demonstrating a superior policy compared to an expert human controller.

Rosenstein and Barto [50] developed a *simple random search* (SRS) utilizing biologically motivated constraints to learn a controller for a robot arm tasked to lift a weight from the hanging stable equilibrium point to the balanced unstable equilibrium point. SRS requires no gradient estimate or any sort of model and can make immediate progress in high dimension space where other algorithms struggle building a model or establishing a search direction.

Roy and Thrun [52] demonstrated the abilities of their two-step hybrid algorithm on mobile robot navigation. A low dimensional discrete plan is first learned by way of dynamic programming, and represented as a series of way points. This ‘seed’ plan is projected into a higher dimension where policy search is performed, increasing the expected reward of the plan. The key to the learning algorithm is the assumption that the initial lower-order DP result biases the policy search algorithm toward a successful plan, and will not lead to an arbitrarily bad controller.

Instruction Scheduling While straight-line instruction scheduling is most definitely a simulated task (as opposed to learning a controller of an object in space), the scheduler is tested on real code against a commercial scheduler. In our view, this is a significant accomplishment of a valuable everyday task, and is included in this section.

McGovern *et al.* [35] present three methods, one using rollouts, one using reinforcement learning and a combination of the two methods to address instruction block scheduling. The rollouts method worked the best, but performed rather slowly. The reinforcement learning method was quicker, but did not perform as well. The combination method maintained the speed of the RL method with a significant improvement (although not as good as the rollouts method). These all compared favorably to the currently used heuristic method, especially the rollouts method, which outperformed the commercial method on all benchmarks.

7 Comparisons Among Various Learning Methods

Without standard benchmarks and a general lack of direct comparison between algorithms on similar tasks, it is difficult to declare certain classes of algorithms more favorable than others. At the same time, certain structures and methods tend to lend themselves to certain environments. The former (direct) method of comparison is always preferable, but typically the latter method is the only one available. For an isolated direct comparison between value function and policy search methods, see [3].

Large MDPs Three of the four applications reviewed Section 6 were represented by large MDPs. In particular, the robot weightlifting [50] and the robot navigation [52] were impressive. Also impressive was the the 75,000,000 state two-armed pendulum solution by Wingate [65]. Kearns [24] recently demonstrated a polynomial time value iteration solution to large MDP spaces, but yet lacks a practical implementation. While the polynomial time solution is still high, typical policy space solutions are only guaranteed in the limit.

POMDPs As stated in Sec 4.4.1, the policy search approach to learning in partially observable domains seems a natural fit. The optimal policy may not be representable in the restricted parameterization of the policy, but neither is it representable in a finite amount of space. An approximation of the optimal policy is the best that can be expected of the algorithm and from the environment. It is generally held that POMDP representations are best suited for representing the noisy and unpredictable environments of everyday problems. It would be expected that, as learning algorithms become more powerful, we would see more practical applications represented as POMDPs and solved with policy search techniques. Only Bagnell’s helicopter control [4] was found as an example of a policy search method solving a real-world problem represented as a POMDP. The literature [2, 40] seems to suggest (by magnitude of references alone) a bias toward approaches covered in this survey.

Conclusion In simple MDP problems with a known model, perhaps simple dynamic programming will always be favored. No method has displayed clear dominance in complex domains. In all likelihood,

researchers will continue to attack problems with their own favorite method, ensuring continued research in all fields from all points of view.

8 Future Work

In his introduction to the 2002 special issue of *Machine Learning*, Singh [55] contends that the current problems with Reinforcement Learning remain intractability and inapplicability which revolve around the current representations (MDPs, Semi-MDPs, and POMDPs). Future progress in RL will involve new representations, including history-based representations, deictic representations and test-based representations, along with innovative representations yet to be developed.

Recent progress in the theoretical underpinnings of policy search algorithms are encouraging. Many of these algorithms have yet to be tested in real world applications. One of the major obstacles AI faces in general is the application of learning ideas to real world domains. As algorithms improve and as computational power continues to grow, there will be an increased opportunity to apply these algorithms to financially viable commercial systems, such as ROOMBA, the automated vacuum [19]. Successes such as these will drive the next wave of research in artificial intelligence, as an entire generation becomes more comfortable with allowing intelligent automation to assist them in their everyday lives.

9 Conclusion

We have presented a survey of policy search algorithms applied to the Reinforcement Learning problem. Much recent work has been done in this area and the most recent works have received prominent places in this survey. We presented a brief history of algorithms applied to reinforcement learning, beginning with dynamic programming methods based on the Bellman equation. These methods serve as the foundation and template for most of the current day RL techniques.

We noticed that RL algorithms can be logically split along two axes, into four groups based on the presence or absence of an explicit policy and of a approximation of the state values, in the form of a value function. We noticed that there are two major approaches to RL based on this division. Value function approaches have a function approximation, but no explicit representation of a policy. Policy search algorithms explicitly represent a policy, but do not have a value function. It was also noticed that there are a group of hybrid algorithms that have both a value function and represent an explicit policy.

We analyzed policy search algorithms based on three criteria. First, we analyzed policy search methods based on the manner in which they search the policy space. Secod, we analyzed the methods based on the manner in which they represent their explicit policy. Third, we analyzed them according to the types of problems they are able to solve.

We analyzed the hybrid methods and examined the specific reasons they were determined to be hybrids. We also discussed some practical applications that have employed policy search methods in recent years. We briefly discussed the applicability of other learning methods to these same problems. Finally we discussed some future problems that need to be addressed and concluded.

Policy search methods will continue to be popular in solving large and complex problems. The further development of solutions to these complex problems (by whatever means) will drive the application of artificial intelligence and machine learning techniques to every day uses by the public masses. In the near future, public applications of machine learning and AI will become commonplace and commercially successful. Policy search techniques will play a major role in these coming applications.

References

- [1] Douglas Aberdeen and Jonathan Baxter. Scaling internal-state policy-gradient methods for pomdps. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.
- [2] Douglas Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, Research School of Information Science and Engineering, Australian National University, December 2003.

- [3] Charles W. Anderson. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Computer Science Department, Colorado State University, February 2000.
- [4] Drew Bagnell and Jeff Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation*. IEEE, May 2001.
- [5] Drew Bagnell and Jeff Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, July 2003.
- [6] Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press.
- [7] E. B. Baum and I. Durdanovic. Toward a model of mind as an economy of agents. *Machine Learning*, 35(2):155–189, 1999.
- [8] Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.
- [9] Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.
- [10] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [11] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Rueven Y. Rubinstein. A tutorial on the cross-entropy method, 2003.
- [12] Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.
- [13] Abhijit Gosavi. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, 55(1):5–29, April 2004.
- [14] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems 14*, pages 1507–1514, Cambridge, MA, 2002. MIT Press.
- [15] Gregory Z. Grudic and Lyle H. Ungar. Localizing policy gradient estimates to action transitions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.
- [16] Gregory Z. Grudic and Lyle H. Ungar. Localizing search in reinforcement learning. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 590–595, 2000.
- [17] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 1998.
- [18] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.
- [19] iRobot. www.roombavac.com.
- [20] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.
- [21] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [22] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural information Processing Systems 12*. MIT Press, 2000.
- [23] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2/3):193–208, 2002.

- [24] Michael Kearns and Satinder Singh. Near optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2/3):209–232, November 2002.
- [25] Kee-Eung Kim, Thomas Dean, and Nicolas Meuleau. Approximate solutions to factored markov decision processes via greedy search in the space of finite state controllers. In *Artificial Intelligence Planning Systems*, pages 323–330, 2000.
- [26] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.
- [27] Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *Society for Industrial and Applied Mathematics Journal on Control and Optimization*, 42(4):1143–1166, 2003.
- [28] Ivo Kwee, Marcus Hutter, and Jurgen Schmidhuber. Market-based reinforcement learning in partially observable worlds. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-2001)*, pages 865–873, 2001.
- [29] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.
- [30] Robert M. Lewis, Virginia Torczon, and Michael W. Trosset. Direct search methods : Then and now. Technical Report NASA/CR-2000-210125 ICASE Report No. 2000-26, Institute for Computer Applications in Science and Engineering, May 2000.
- [31] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animals 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994.
- [32] Shie Mannor, Ruvien Rubinstein, and Yoichi Gat. The cross entropy method for fast policy search. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 512–519, 2003.
- [33] Peter Marbach and John N. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, February 2001.
- [34] Peter Marbach and John N. Tsitsiklis. Approximate gradient methods in policy-space optimization of markov reward processes. *Journal of Discrete Event Dynamical Systems*, 13:111–148, 2003.
- [35] Amy McGovern, Eliot Moss, and Andrew S. Barto. Building a basic block instruction scheduler with reinforcement learning and rollouts. *Machine Learning*, 49(2/3):141–160, November 2002.
- [36] Nicolas Meleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 427–436, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
- [37] Nicolas Meuleau, Kee-Eung Kim, Leslie Kaelbling, and Anthony Cassandra. Solving pomdps by searching the space of finite policies. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 417–426, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
- [38] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.
- [39] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2/3):291–323, November 2002.
- [40] Kevin P. Murphy. A survey of pomdp solution techniques. url = citeseer.ist.psu.edu/murphy00survey.html.
- [41] Andrew Y. Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth Conference of Uncertainty in Artificial Intelligence (UAI)*, 2000.
- [42] Andrew Y. Ng, Ronald Parr, and Daphne Koller. Policy search via density estimation. In *Proceedings of the Twelfth Annual Conference on Neural Information Processing Systems*, December 1999.

- [43] Leonid Peshkin. *Reinforcement Learning by Policy Search*. PhD thesis, Brown University, April 2002.
- [44] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, page 489, 2000.
- [45] Leonid Peshkin, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.
- [46] Leonid Peshkin and Sayan Mukherjee. Bounds on sample size for policy evaluation in markov environments. In *proceedings of the 14th annual conference on computational learning theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 616–630. Springer, 2001.
- [47] Leonid Peshkin and Christian R. Shelton. Learning from scarce experience. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 498–505, 2002.
- [48] Daniel Polani and Risto Miikkulainen. Fast reinforcement learning through eugenic neuro-evolution. Technical Report AI99-277, University of Texas at Austin, Austin, TX, January 1999.
- [49] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16*, 2004.
- [50] Michael T. Rosenstein and Andrew G. Barto. Robot weightlifting by direct policy search. In *Proceedings of the Seventeenth international Joint Conference on Artificial Intelligence*, pages 839–846, 2001.
- [51] Nicholas Roy and Sebastian Thrun. Integrating value functions and policy search for continuous markov decision processes. citeseer.ist.psu.edu/454082.html.
- [52] Nicolas Roy and Sebastian Thrun. Motion planning through policy search. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, 2002.
- [53] Jurgen Schmidhuber. Evolutionary computation versus reinforcement learning, [cite-seer.ist.psu.edu/schmidhuber00evolutionary.html](http://citeseer.ist.psu.edu/schmidhuber00evolutionary.html).
- [54] Jurgen Schmidhuber, Jieyu Zhao, and Marco Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997.
- [55] Satinder Singh. Introduction. *Machine Learning*, 49(2/3):107–109, November 2002.
- [56] Malcolm Strens and Andrew Moore. Policy search using paired comparisons. *Journal of Machine Learning Research*, 3:921–950, 2002.
- [57] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.
- [58] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
- [59] Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, 1996.
- [60] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72, 2002.
- [61] Xin Wang and Thomas G. Dietterich. Model-based policy gradient reinforcement learning. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, 2003.
- [62] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
- [63] Marco Wiering and Juergen Schmidhuber. Solving pomdps with levin search and eira. In *Proceedings of thirteenth International Conference on Machine Learning (ICML)*, Bari, Italy, 1996.
- [64] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.
- [65] David Wingate and Kevin Seppi. Solving large mdps quickly with partitioned value iteration, 2004 (in submission).