# Exemplar-Based Direct Policy Search with Evolutionary Optimization

**Kokolo IKEDA**
Academic Center for Computing and Media Studies, Kyoto University
Yoshida-nihonmatsu-cho, Sakyo-ku, Kyoto-city, Japan
kokolo@media.kyoto-u.ac.jp

**Abstract- In this paper, an exemplar-based policy optimization framework for direct policy search is presented. In this exemplar-based approach, the policy to be optimized is composed of a set of exemplars and a case-based action selector. An implementation of this approach using a state-action-based policy representation and an evolutionary algorithm optimizer is shown to provide favorable search performance for two higher-dimensional problems.**

## 1 Introduction

Hard learning problems involving the acquisition of a sequence of actions through trial-and-error and evaluation have been studied extensively in the context of reinforcement learning as one of the most important problems in machine learning. Such problems are usually formulated as Markov Decision Processes (MDPs) [van der Wal, 1981], in which a learning agent develops a *policy*, that is, a mapping from a set of states to a set of actions.

The most common approach to such problems is the learning of a value function, typically by the temporal difference method [Sutton, 1988]. In this approach, a policy is derived from the value-function and the explicit representation is not required to maintain. The policy-space approach, which involves searching for policies that optimize an appropriate objective function, has also been widely studied [Moriarty *et al.*, 1999]. In *direct policy search (DPS)*, the core process of the policy-space approach, a policy is represented by a model with parameters, and the parameters are optimized so as to maximize the evaluation function by applying the parameterized policy to the problem.

The key point of DPS is the selection of a model to parameterize the policy. At a primitive level, the policy is represented by the complete table of concrete state-action pairs. However, in difficult problems with large state space, some generalization is required to reduce the size of the search space. In a DPS system such as SAMUEL [Grefenstette, *et al.*, 1990], a policy is represented by a set of condition-action rules or if-then rules. In simple terms, when the current state satisfies a condition, the associated action is employed. Alternatively, as is the case in GENITOR [Whitley and Kauth, 1988], a policy may be represented by an artificial neural network (ANN) that outputs an action given the current state as an input.

The approach adopted in the present research involves separating the set of referred information and the action decision function. In the weightlift problem [Rosenstein and Barto, 2001], where the task is to swing a weight up over the head, the via-point is the only search target, and the action is decided by a proportional-derivative controller from the current state to the via-point or the goal state. The present algorithm focuses on the difference between information such as the via-point, the condition-action rules, and parameters with no explicit meaning such as the weights of the ANN. The purpose is to make the condition for information referral vary adaptively depending on the relationships between other information, and to make it possible to utilize free-style representation of information such as "a state $s^1$ is better than $s^2$".

The framework of DPS algorithms for machine learning problems is introduced in this study considering a case-based policy representation with parameter optimization. The framework itself, a typical implementation, and application of the scheme to two problems are presented in this paper.

## 2 Exemplar-based policy optimization

Exemplar-based policy (EBP) optimization is a framework for direct policy search where the policy to be optimized is composed of a set of exemplars and a case-based action selector. The definition of EBP is considered here for a target problem limited to the domain of MDPs, where $S$ is the state space and $A$ is the action space. A policy is defined as a mapping from $S$ to $A$ such that the procedure selects an action $a \in A$ for a state $s \in S$.

For the purpose of selecting an action, the policy is defined as having two components: a set of *exemplars* and an *action selector*. An exemplar is specific information, while the action selector is a reasoner applied to the set of exemplars. Although a clear definition of exemplar is not given here, the notion is distinguished from explicit rules with prescribed invocation condition, and from cryptic parameters with no explicit meaning. The typical styles of EBP are as follows.

**Via-points style**
An exemplar is defined as a pair $(j, s_j)$ meaning "follow $s_j$ in order", and the action selector decides the action to achieve the next $s_j$ or the goal state.

**State-action style**
An exemplar is defined as a pair $(s_j, a_j)$ meaning "take action $a_j$ at $s_j$". The nearest-neighbor classification can be used as the action selector [Sheppard, 1997], that is, when the current state is $s$, the nearest state $s_{j*}$ is selected from the set of exemplars, and the associated action $a_{j*}$ is taken.

**State-value style**
An exemplar is defined as a pair $(s_j, v_j)$ with $v_j \in R$, meaning "the desirability of $s_j$ is $v_j$". If the state transition is deterministic and the agent at a state knows the following state $s^i$ after the action $a^i \in A$, the action selector estimates the desirability of each $s^i$ by a function generalization method, selects the best $s^{i*}$, and finally takes the action $a^{i*}$. State-state style where an exemplar is defined as a pair $(s_j^1, s_j^2)$, meaning "$s_j^1$ is better than $s_j^2$", is also considered.

**State-action-value style**
An exemplar is defined as a pair $(s_j, a_j, v_j)$ with $v_j \in R$, meaning "the desirability of $a_j$ at $s_j$ is $v_j$". The action selector estimates the desirability of each $a_j \in A$ by a function generalization, and then takes the best action $a_j*$. This style is analogous to the Q-value and the greedy selection.

For a given action selector, the performance of an EBP depends only on its exemplars. Regarding the exemplars as parameters, and assuming that the evaluated value of the parameters can be calculated by applying the policy to the given problem, optimization algorithms can be utilized to obtain better policies. This procedure, referred to as EBP optimization, does not necessarily require an initial set of positive exemplars because the exemplars are selected and evolved.

One advantage of EBP optimization is that both the introduction of prior knowledge and the extraction of knowledge after optimization are relatively straightforward. For example, if $a^*$ is known to be the best action at a particular state $s$, the state-action pair $(s, a^*)$ can be introduced directly as an exemplar. In contrast, if the condition-action rule is required, the condition including $s$ must be fixed properly. This is even more so for the type of prior knowledge "a state $s^1$ is better than $s^2$".

Another advantage is the power of policy expression. In contrast to neural networks or regular tile coding, EBP optimization can express a policy that with precise control for states and rough control in other parts naturally since the condition for exemplar referral is varied adaptively depending on the relationships between exemplars.

DPS is also advantageous over value-based reinforcement learning in that multi-objective optimization can be performed easily due to the ability of the DPS agent to act depending not on any objective values but on only its policy.

The proposal framework is more suitable for offline simulation-based optimization applications than for real-time solutions. And, one of the major challenge in DPS is that its optimization is higher-dimensional, nonlinear and no gradient method can be applied. Therefore powerful optimization algorithm is required for this scheme to obtain the satisfactory policy.

## 3 A practical implementation of EBP optimization

The implementation of EBP optimization requires the type of exemplar, the action selector and the optimization procedure to be fixed. In the present algorithm, the state-action style with nearest-neighbor action selector is employed as the most simple and general case, although other types may be required in some instances.

A genetic algorithm (GA) is employed for optimization. The GA is a powerful search technique that relies on parallels with nature [Nagata, 1997], maintaining and evolving a population of individuals. In the DPS, an individual is a policy consisting of a set of exemplars and an action selector. Although the initial set of exemplars is a dirty mixture of positive and negative exemplars, the evolutionary procedure leads to a refined favorable set of exemplars.

The implementation of EBP optimization with state-action policy representation and GA search is called the SAP-GA method for convenience. The notation employed is listed in Table 1.

Table 1: Notation

| | |
|---|---|
| $s, S$ | A state, and the state space. |
| $d(s_1, s_2)$ | A distance measure between two states $s_1$ and $s_2$. |
| $a, A$ | An action, and the action space. |
| $p^a(s, s')$ | The transition probability to state $s'$ when action $a$ is taken at state $s$. |
| $R^a(s, s')$ | The expected reward when action $a$ is taken at state $s$ and the agent transitions to state $s'$. |
| $N_{\text{step}}$ | The maximum number of steps in one episode. |
| $D$ | A case-based action selector. |
| $E^i$ | The set of exemplars of the $i$th policy. |
| $e_j^i$ | The $j$th exemplar of $E^i$, with the state-action pair $(s_j^i, a_j^i)$ |
| $\|E^i\|$ | The number of exemplars in $E^i$ |
| $L_{\min}, L_{\max}$ | The minimum/maximum number of exemplars in one policy. |
| $\pi^i(E^i, D)$ | The $i$th policy of a population, a mapping from $S$ to $A$, composed of exemplars $E^i$ and action selector $D$. |
| $f(\pi)$ | The evaluation function of policy $\pi$. |
| $N_{\text{pop}}$ | The number of solutions (policies) in the population. |
| $N_{\text{child}}$ | The number of children produced in the reproduction phase. |

The main procedure is defined as follows (see Fig. 1).

**Main procedure**

1. The problem, MDP environments $S$, $A$, $p$, and

$R$, and the criteria for policy evaluation $f, N_{\text{step}}$ are given.

2. The optimization operators, such as the action selector $D$ or the crossover operator, are fixed.
3. The parameters that specify the search space, such as $N_{\text{pop}}$, $N_{\text{child}}$, $L_{\min}$, and $L_{\max}$ are fixed.
4. The population (i.e., the set of solutions $\{E^i\}$) is initialized. The number of exemplars $L_{\min} \leq |E^i| \leq L_{\max}$ is usually fixed randomly, and one exemplar $e^i_j \in E^i$ (i.e., one state-action pair $(s^i_j, a^i_j) \in (S, A)$) is also generated **randomly**.
5. $N_{\text{pop}}$ solutions are randomly coupled into $N_{\text{pop}}/2$ pairs. All pairs are passed to the **alternation procedure**.
6. Repeat step (5) until the termination conditions are satisfied.
7. The final result is obtained and utilized in the next phase.

The alternation procedure is based on a standard family alternation model. However, other refined models may also be employed.

**Alternation procedure**

1. The parent solutions $\pi^{p1}$ and $\pi^{p2}$ are given.
2. $N_{\text{child}}$ solutions are reproduced by applying the **crossover operator** $N_{\text{child}}$ times.
3. The evaluated value of $f(\pi)$ for each member of the family (i.e., parents and children) is calculated by the evaluation procedure.
4. The best policy $\pi^*$ in the family is selected.
5. $\pi^*$ is cloned, and the **refresh operator** is applied to the clone.
6. $\pi^*$ and the refreshed clone are introduced into the population instead of the parents.

Although the performance of a policy is usually measured by the discounted expected reward, the accumulated reward for one episode is employed in the present scheme. If the state transition or the action selection is not deterministic, the performance gain is not static, and some averaging may be required.

**Evaluation procedure**

1. The policy $\pi^i(E^i, D)$ is given.
2. The state $s$ and the accumulated reward are initialized.
3. An action $a$ is selected by the action selector $D$.
4. The state is transitioned, and the reward is accumulated.
5. Repeat from step (3) until the goal condition is satisfied or the number of steps reaches $N_{\text{step}}$.
6. The accumulated reward of an episode is calculated and returned as $f(\pi^i)$.

The implementations of the remaining procedures, that is, the crossover operator, the refresh operator and the action selector, are shown below. Note that these implementations are examples and are exchangeable depending on the problem.

**Nearest-neighbor action selector**

1. The current state $s$ and exemplars $\{e_j\}$ (i.e., state-action pairs $\{(s_j, a_j)\}$) are given.
2. For each $s_j$, the distance $d(s, s_j)$ is measured.
3. The nearest state $s_{j*}$ is selected.
4. The action $a_{j*}$ is selected and taken.
5. The referred exemplar $e_{j*}$ is marked for the refresh operator.

**Uniform exemplar crossover operator**

1. Parent solutions $\pi^{p1}(E^{p1}, D)$ and $\pi^{p2}(E^{p2}, D)$ are given.
2. The selection rate $0.3 \leq r \leq 0.7$ is fixed (value not critical).
3. $E^c$ is initialized as an empty set.
4. Each element $e^{p1}_i \in E^{p1}$ is added to $E^c$ with probability $r$.
5. Each element $e^{p2}_i \in E^{p2}$ is added to $E^c$ with probability $1 - r$.
6. If any two elements of $E^c$ are identical, one is removed.
7. Return to step (2) if the condition $L_{\min} \leq |E^c| \leq L_{\max}$ is not satisfied.
8. The child policy $\pi^c(E^c, D)$ is returned.

**Refresh operator**

1. The policy $\pi$ and its set of exemplars $\{e_j\}$ are given (it is assumed that $\pi$ has been evaluated once).
2. The selection rate $0.0 \leq r \leq 0.5$ is fixed (value not critical).
3. $E^{lazy} \in \{e_j\}$ is defined as the set of exemplars that were not referred in the previous evaluation.
4. Every member of $E^{lazy}$ is re-initialized randomly with probability $r$.

This operator is specific for the EBP optimization domain. This is employed to remove "lazy" exemplars that are rarely referenced, and to provide new exemplars to the population.

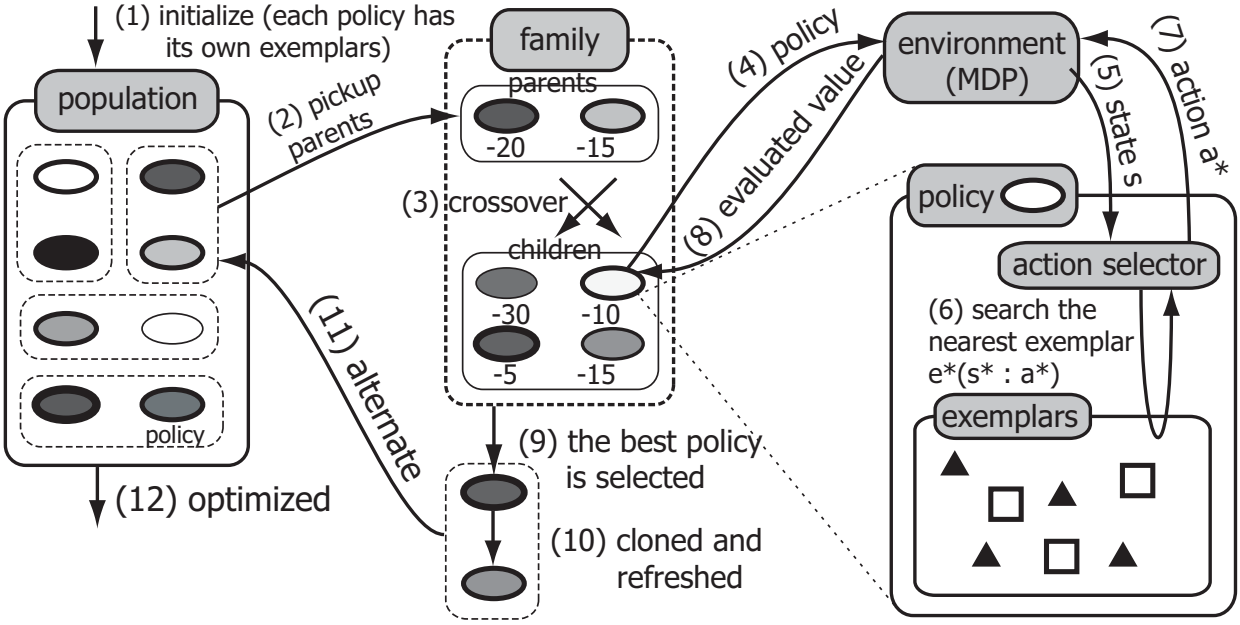


Figure 1: Basic procedure of the SAP-GA method. (1) Given a problem, the population is initialized. (2) In a generation, two solutions (parents) are picked for alternation. (3) Several children are produced by the crossover operator. (4) A policy, a solution in the family, is evaluated in the environment of the target problem. (5) For each step, the observation of state $s$ is sent to the action selector of the policy. (6) The nearest exemplar $(s^*, a^*)$ to $s$ is selected. (7) The action $a^*$ is taken. (8) At the end of the MDP, the evaluated value of the policy is calculated and returned. (9) The best policy in the family is selected. (10) The best policy is cloned and the refresh operator is applied. (11) Two solutions, the best solution and the refreshed solution, are sent back to the population instead of the parents. (12) The policy is optimized.

## 4 Experiments

The performance of the SAP-GA was evaluated in two problem domains; the Acrobot [Spong, 1994] and the parallel-type double inverted pendulum (PDIP). Methods based on control theories and physical models are often used to solve such problem. In this paper, the physical model is used only for simulation; the agent makes its decision depending only on the state observation and its policy.

### 4.1 Acrobot problem

The Acrobot task involves swinging a tandem double pendulum. Two links are connected by a hinge joint, and a single motor exerts torque between the links (see Fig. 2).

- The state is defined by the link angles $\theta_1$, $\theta_2$ and their velocities $\dot{\theta}_1$ and $\dot{\theta}_2$, which are normalized to $[0, 2\pi]$. $\dot{\theta}_1$ is bounded by $[-4\pi, 4\pi]$, and $\dot{\theta}_2$ is bounded by $[-9\pi, 9\pi]$.
- The initial state is fixed at $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (3\pi/2, 0, 0, 0)$.
- The action (torque) space is defined as $A = \{-2, 0, 2\}$. (Note: SI units are used in this paper)
- The state transition follows the physical law (the code in C is available at http://www.cmap.polytechnique.fr/~munos/variable/acrobot.html). In this paper, a single timestep for the MDP is defined as 0.05 s, and the simulation timestep $dt$ is set to 0.01 s.
- The reward is $\sin(\theta_1) + \sin(\theta_1 + \theta_2) - 3$, defined in relation to the altitude of the tip of the pendulum.
- The best state is $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (\pi/2, 0, 0, 0)$. The goal area $S_\delta$ is defined by $(\pi/2 \pm 2\pi\delta,\ \pm 2\pi\delta,\ \pm 8\pi\delta,\ \pm 18\pi\delta)$, where $\delta$ is a difficulty parameter. When $\delta = 0.05$, the goal area is 0.01% of the entire area $S$. The episode is terminated if $s \in S_\delta$.
- $N_{\text{step}}$ is set to 400.

### 4.2 Parallel-type double inverted pendulum problem

The PDIP task involves swinging two inverted pendulums linked to a car on a rail. The torque is applied not to the links but to the car (see Fig. 2).

- The state is defined by the position of the car $x$, its velocity $v$, the link angles $\theta_1$, $\theta_2$, and their velocities $\dot{\theta}_1, \dot{\theta}_2$.

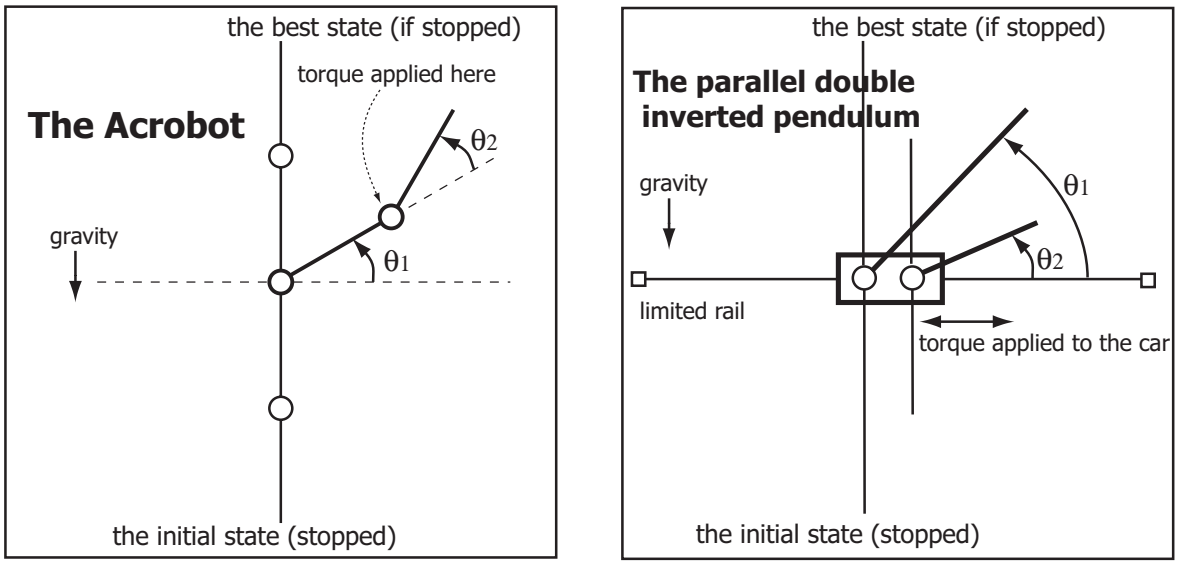


Figure 2: The Acrobot (left) and the PDIP (right)

- $x$ is bounded by $[-2.5, 2.5]$ and $v$ is bounded by $[-16, 16]$. $\theta_1$ and $\theta_2$ is normalized to $[0, 2\pi]$, $\dot{\theta}_1$ and $\dot{\theta}_2$ are bounded by $[-4\pi, 4\pi]$.
- The initial state is fixed at $(x, v, \theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 3\pi/2, 3\pi/2, 0, 0)$.
- The action (torque) space is defined as $A = \{-40, 0, 40\}$.
- The state transition follows the physical law. Let $g = 9.8$, $m_0$ be the weight of the car, $m_i$ be the weight of the $i$th pole, $l_i$ be the length, $s_i = \sin(\theta_i)$, and $c_i = \cos(\theta_i)$. When a torque $T$ is applied, the acceleration of the car $a$ and the update procedures are as follows.

$$a = \frac{g(m_1 s_1 c_1 + m_2 s_2 c_2) - \frac{4}{3}(T + m_1 l_1 \dot{\theta}_1^2 c_1 + m_2 l_2 \dot{\theta}_2^2 c_2)}{m_1 s_1^2 + m_2 s_2^2 - \frac{4}{3}(m_0 + m_1 + m_2)}$$

$$\dot{\theta}_i \mathrel{+}= \frac{a s_i - g c_i}{\frac{4}{3} l_i} dt, \quad \theta_i \mathrel{+}= \dot{\theta}_i dt$$

$$v \mathrel{+}= a dt, \quad x \mathrel{+}= v dt$$

- the simulation timestep $dt$ is set to $0.01s$.
- The reward is $\sin(\theta_1) + \sin(\theta_1 + \theta_2) - 3$. When the position restriction is broken, $-100$ reward is added.
- The best state is $(x, v, \theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, \pi/2, \pi/2, 0, 0)$. The goal area $S_\delta$ is defined by $(\pm 2, \ \pm 16\delta, \ \pi/2 \pm 2\pi\delta, \ \pi/2 \pm 2\pi\delta, \ \pm 8\pi\delta, \ \pm 8\pi\delta)$, where $\delta$ is a difficulty parameter. When $\delta = 0.05$, the goal area is 0.0008% of the entire area $S$. The episode is terminated if $s \in S_\delta$.
- $N_{\text{step}}$ is set to 400.
- $(m_0, m_1, m_2, l_1, l_2) = (1.0, \ 0.2, \ 0.1, \ 1.0, \ 0.5)$ by default. When $l_2 = l_1$, this problem is equivalent to the single inverted pendulum problem.

### 4.3 Learner settings

The following parameters were used in the experiments:

- The number of solutions $N_{\text{pop}} = 100$, and the number of children $N_{\text{child}} = 10$.
- The range of the number of exemplars $L_{\min} = 50$, and $L_{\max} = 100$.
- The distance $d(s_1, s_2)$ is defined by the normalized Euclidian distance.

$Q$-learning [Watkins, 1992] using an $\epsilon$-greedy search was also tested for comparison using the following parameters: exploration rate $\epsilon = 0.02, 0.01$ (for Acrobot,PDIP respectively), learning rate $\eta = 0.02, 0.01$, discount factor $\gamma = 1, 1$. Square tiling is employed to quantize the continuous space into discrete states, where $N_{\text{tile}}$ is the number of tiles.

Although the state of PDIP is six-dimensional by default, $l_2 = 1.0$ is used for $Q$-learning, resulting in four dimensions. $N_{\text{tile}} = 20^4$ is used for both problems.

### 4.4 Experimental results

The method and parameters for the four problems are summarized in Table 2. Each experiment was performed 20 times using different random seeds, and the average performance and standard deviation were recorded. The performance measures are as follows: the best evaluated values in a period, the minimum $\delta^*$ for $\exists s$ in all achieved states through the period such that $s \in S_{\delta^*}$ (i.e., how a closer state to the best goal is obtained), and whether $\delta^*$ is under the given $\delta$ (i.e., whether any agent achieved the goal).

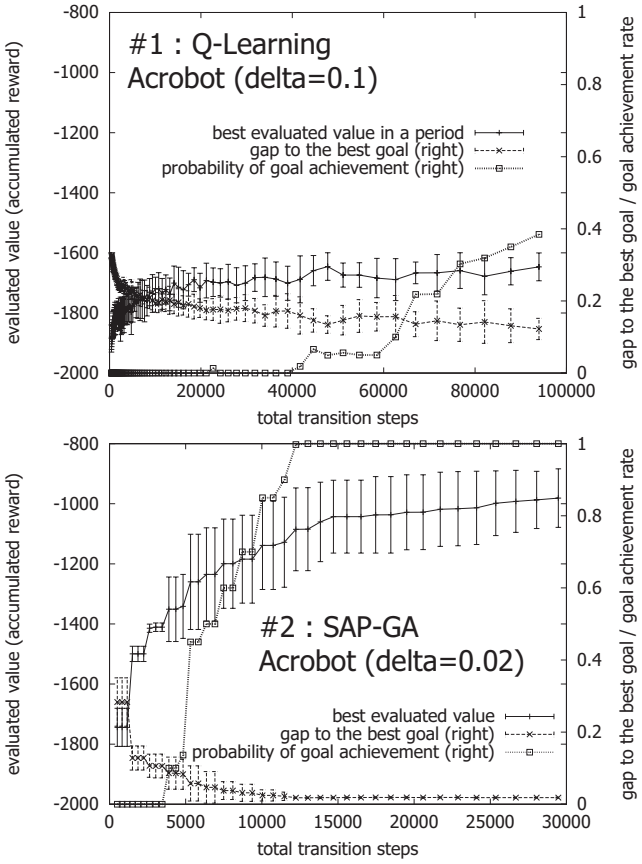


Figure 3: Results for Acrobot

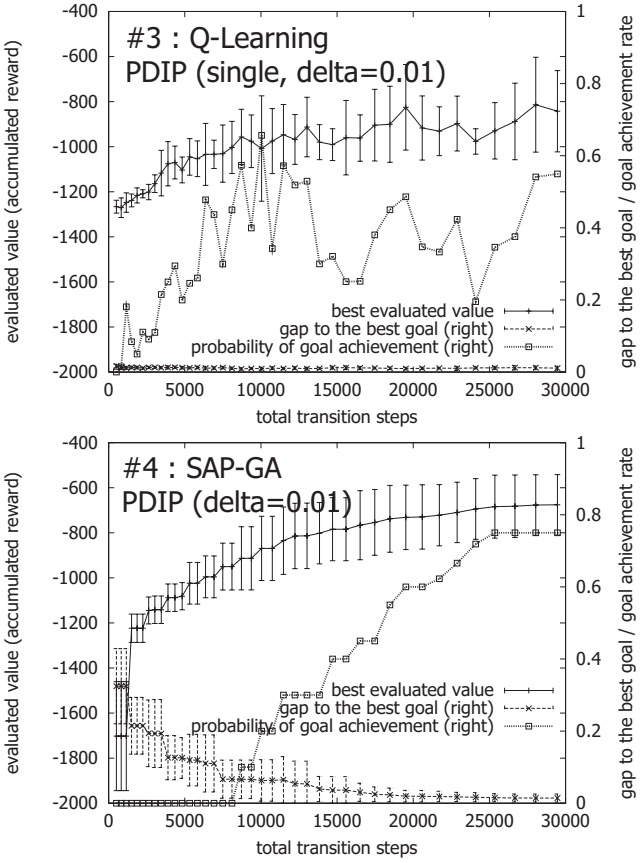


Figure 4: Results for PDIP:).

Table 2: Experimental settings

| No. | Problem | Method | Total steps |
|---|---|---|---|
| #1 | Acrobot, $\delta = 0.1$ | QL | $100 \times 10^6$ |
| #2 | Acrobot, $\delta = 0.02$ | SAP-GA | $30 \times 10^6$ |
| #3 | PDIP, $l_2 = 1.0$, $\delta = 0.01$ | QL | $30 \times 10^6$ |
| #4 | PDIP, $\delta = 0.01$ | SAP-GA | $30 \times 10^6$ |

Figures 3 and 4 show the experimental results for these two problems. In both problems, the performance of SAP-GA is better than that of QL, even though the goal of QL ($\delta = 0.1$) is not strict and a longer learning time is provided (#1, #2), and though the single inverted pendulum problem is quite easier to the double one (#3, #4). The superiority of SAP-GA for these two problems, and the fact that six-dimensional problem was solved as well as four-dimensional one, suggests that the case-based representation is more powerful than tile-based coding in cases of high dimensionality. The power of policy expression in EBP is also richer than that for tile-coding QL (see Fig. 5).

The Acrobot control trajectory for a policy derived by SAP-GA is shown in Fig. 6, and the two PDIP control trajectories for two policies determined by SAP-GA are shown in Fig. 7. One of the desirable properties of the SAP-GA method is that various policies with comparable qualities are obtained.

## 5 Conclusion

The framework for exemplar-based policy optimization was presented as a novel approach to reinforcement learning. This framework defines a policy as being composed of a set of exemplars and an action selector, with parameter optimized directly by the evaluated value. A simple implementation of this framework using a state-action based representation and a GA was also presented, and the excellent performance of the scheme was demonstrated in two problem domains. The accurate control and robustness of the exemplar-based method for higher-dimensional problems originate from the adaptive definition of the conditions for exemplar referral, which vary depending on the relationships between exemplars.

In the context of direct policy search, many common difficulties such as the fluctuation of evaluations or lack of optimality has been widely discussed. Although they are still problems which to be solved with the present approach also, the new possibility this study suggested should be discovered. It is expected that other types of exemplars, such as a state-state style are also quite promising for certain problem domains.

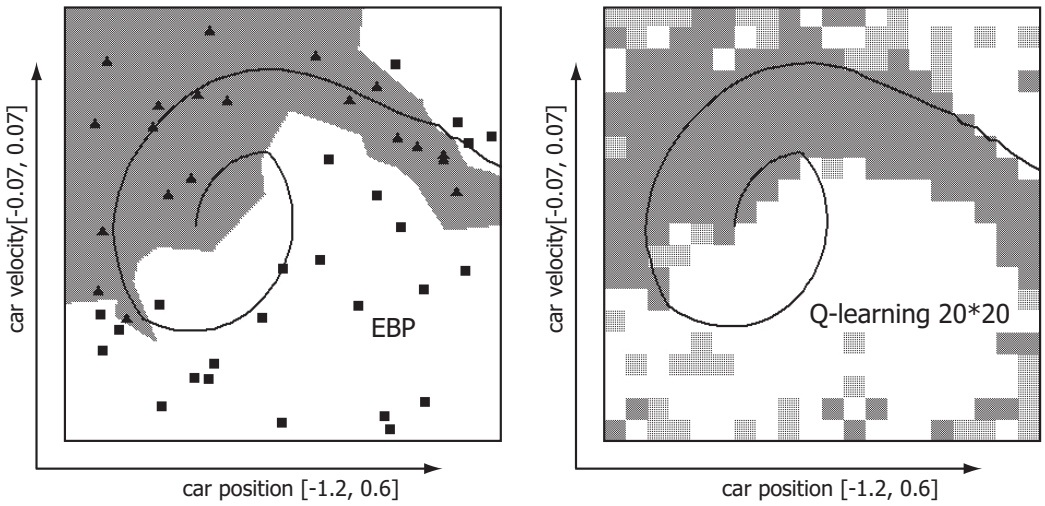


Figure 5: EBP (left) and QL policy (right) for a two-dimensional problem (modified mountain car problem). The action 1 (to move right) is taken in the dark gray area, and the action $-1$ is taken in the white area. Triangles and squares denote exemplars, the solid line denotes a trajectory.

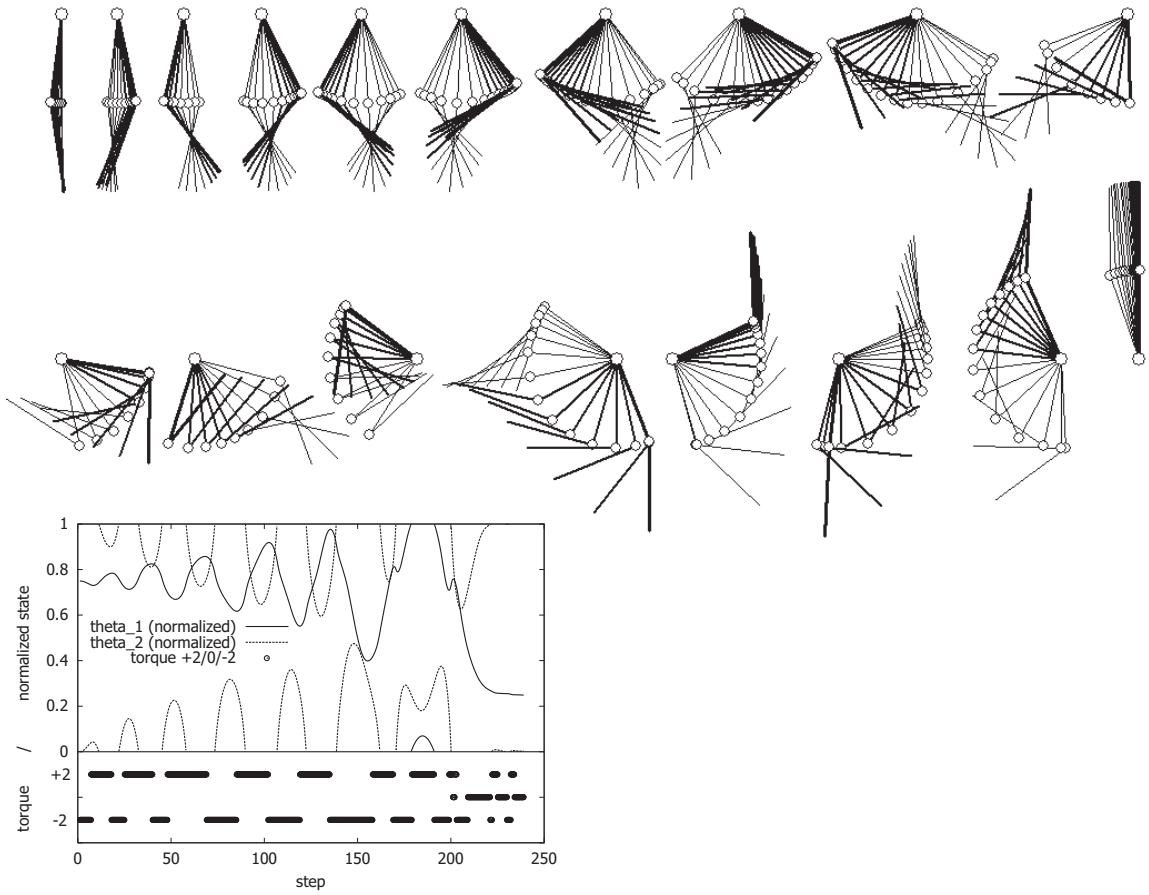


Figure 6: Control trajectory of a policy derived by the SAP-GA method for the Acrobot problem. The double pendulum swings 12 times and stands upright at step 240.

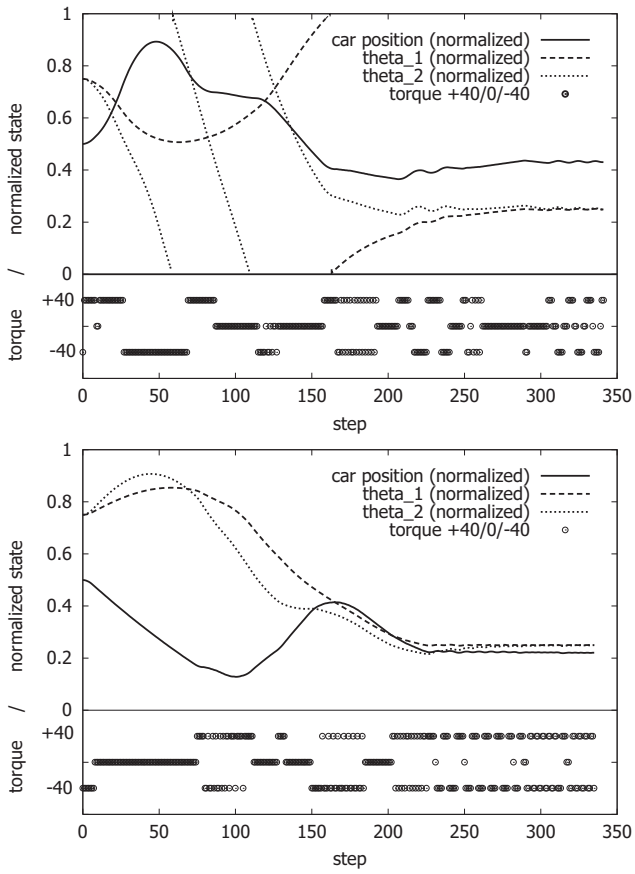


Figure 7: Control trajectories for two policies determined for the PDIP problem by the SAP-GA method. Both solutions provide good control but differ remarkably.

## Bibliography


[Grefenstette, *et al.*, 1990] Grefenstette, J.J., Ramsey, C.L. and Shultz, A.C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning, 5*, pp. 355-381.

[Moriarty *et al.*, 1999] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research 11*, pp. 241-276.

[Nagata, 1997] Nagata, Y. and Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 450-457.

[Rosenstein and Barto, 2001] Rosenstein, M.T. and Barto, A.G. (2001). Robot weightlifting by direct policy search. *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, vol. 2, pp. 839-844.

[Sheppard, 1997] Sheppard, J.W. and Salzberg, S.L. (1997). A teaching strategy for memory-based control. *Artificial Intelligence Review 11*, pp. 343-370.

[Spong, 1994] Spong, M.W. (1994). Swing up control of the acrobot. *Proceedings of the 1994 IEEE Conference on Robotics and Automation*, San Diego, CA.

[Sutton, 1988] Sutton, R.S. (1988). Learning to predict my methods of temporal differences. *Machine Learning 3*. pp. 9-44.

[van der Wal, 1981] van der Wal, J. (1981). Stochastic dynamic programming. Amsterdam: Morgan Kaufmann.

[Watkins, 1992] Watkins, C.J.C.H., Dayan, P. (1992). Q-learning. *Machine Learning, 8*, pp. 179-292.

[Whitley and Kauth, 1988] Whitley, D. and Kauth, J. (1988). GENITOR: A different genetic algorithm. *Proceedings of the Rocky Mountain Conference on Artificial Intelligence* pp 116-121.