Teknillinen korkeakoulu Tietotekniikan osasto

Tietojenkäsittelyopin laboratorio B

Helsinki University of Technology Department of Computer Science and Engineering

Laboratory of Information Processing Science B

Espoo 2003 TKO-B 152/03

GUIDING INITIAL STATE-SPACE EXPLORATION BY ACTION RANKING AND EPISODIC MEMORY

Kary Främling

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY

Distribution:

Helsinki University of Technology

Laboratory of Information Processing Science

URL: http://www.cs.hut.fi/english.html

This is an electronic version of the report. It can be found in Adobe Portable Document Format (PDF) at the following address:

http://www.cs.hut.fi/Publication/Reports/B152.pdf.

There is no separate paper version of this report.

ISBN: 951-22-6767-5

ISSN: 1239-6893

Espoo 2003

Abstract

Author: Kary Främling

Title: Guiding Initial State-space Exploration by Action Ranking and Episodic Memory

Publisher: Helsinki University of Technology
Laboratory of Information Processing Science

Report code: TKO-B 152/03 **Number of Pages**: 19

Date: October 21, 2003

Reinforcement learning is based on exploration of the environment and receiving reward that indicates which actions taken by the agent are good and which ones are bad. In many applications receiving even the first reward may require long exploration, during which the agent has no information about its progress.

This paper presents an approach that makes initial exploration advance more rapidly through the state space. The SLAP method presented reduces the probability of visiting same states or trying same actions that have been recently used in a given state. Concepts of short- and long-term memory combine this exploration strategy with existing reinforcement learning methods for value function estimation in order to make exploration converge towards an optimal solution.

Keywords: reinforcement learning, episodic memory, short-term memory, long-term memory, action ranking, grid world

Table of contents

1 Introduction

Trial and error seems to be one of the main ways that animals learn to solve problems. The success or failure of a trial should help to modify behavior in the "right" direction. The scientific research area called reinforcement learning (RL) (Barto et al., 1990) (Kaelbling et al, 1996) (Sutton&Barto, 1998) is one of those studying such behavior. RL methods have been successfully applied to many problems where the agent has to explore its environment and learn interactively. Depending on the current state of the environment and the agent, the agent has to take actions without a priori knowledge about how good or bad the action is, which may be known only much later when a goal is reached or when the task failed.

The agent explores the environment following a policy, usually referred to by the symbol π, and observing received reward. In tasks with delayed reward, the agent may have to do a random search through all possible states of the environment (the *state space*), which is long even with state spaces of moderate size. For some tasks, random exploration may never allow the agent to obtain any reward, which makes it impossible for the agent to learn (Tesauro, 1995).

The first goal of this paper is to present an alternative policy to random search for initial state-space exploration, especially in tasks where reward-giving states are sparse. It uses a general learning rule that shortens the traversal of the state space by reducing the number of times a state is visited and the number of times that the same action is used in a given state. The second goal is to study how the use of such a policy affects the capability of the agent to find an optimal solution to the task. This paper concentrates on episodic tasks, i.e. tasks with a pre-defined terminal state (Sutton&Barto, 1998), even though the methods described here are not limited to such tasks.

After this introduction, the most relevant RL methods to the scope of this paper are described in Section 2. Section 3 presents how to shorten initial exploration and how to combine it with methods presented in Section 2. Section 4 shows comparative test results for grid world tasks, followed by conclusions.

2 Reinforcement learning principles

One of the main domains treated by RL is solving Markov Decision Processes (MDPs). A (finite) MDP is a tuple $M=(S,A,T,R)$, where: S is a finite set of states; $A = \{a_1, \ldots, a_k\}$ is a set of $k \geq 2$ actions; $T = [P_{sa}(\cdot) \mid s \in S, a \in A\}$ are the next-state transition probabilities, with $P_{sa}(s')$ giving the probability of transitioning to state s' upon taking action a in state s; and R specifies the reward values given in different states $s \in S$. In this paper we will focus on the initial state space exploration in MDPs. The goal of this exploration is to allow an agent to interact with its environment in order to identify (learn) a value function that allows it to find a (preferably optimal) solution to the MDP.

Of the many different RL methods that exist, we will here concentrate only on the currently most used ones, i.e. Temporal Difference (TD) and Q-learning. Sutton and Barto (1998) give a good overview of other RL methods.

2.1 Value function

Current RL methods are based on the notion of *value functions*. Value functions are either *state-values* (i.e. value of a state) or *action-values* (i.e. value of taking an action in a given state). The value of a state $s \in S$ can be defined formally as

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \middle| s_t = s\right\}, \quad (1)$$

where $V^{\pi}(s)$ is the state value that corresponds to the expected return when starting in s and following policy π thereafter. The factor r_{t+k+1} is the reward obtained when arriving into states s_{t+1}, s_{t+2} etc. γ^{k} is a discounting factor that determines to what degree future rewards affect the value of state s. Action value functions are denoted $Q(s,a)$, where $a \in A$.

The policy used determines the balance between exploring the environment and exploiting already found, but possibly sub-optimal solutions. Random search achieves maximal exploration, while a greedy policy gives maximal exploitation by always taking the action that leads to the next state with the highest state value. A commonly used method for balancing exploration and exploitation is to use *ε-greedy exploration*, where actions are selected greedily with probability (1-ε) and randomly with probability ε.

Using "optimism in the face of uncertainty" (Kaelbling et al., 1996) or "optimistic initial values" (Kakade&Dayan, 2001) is an alternative to random search. This technique has become standard practice in reinforcement learning to encourage systems to get to novel or unfamiliar states (Kakade&Dayan, 2001). It can be implemented by "optimistic" initial value function estimates (zero or greater) and giving negative reward at every step. This technique indeed shortens initial exploration by avoiding previously visited states. However, this technique also encourages the agent to explore the entire state space.

Exploration of the entire state space is impossible in many practical applications reported like backgammon, which has approximately 10^{20} states. Reward shaping (Gullapalli, 1992) (Mataric, 1994) (Randløv&Alstrøm, 1998) (Ng at al., 1999) (Randløv, 2000) is one method that can make exploration significantly faster by modifying the reward given to the agent in a way that guides the agent directly towards the optimal policy. However, reward shaping requires a priori knowledge about the environment (Randløv&Alstrøm, 1998), such as where the goal is located, the stochastic level of the environment or the number of sub goals to reach (Ng at al., 1999).

Another approach is to use state generalization, where it is assumed that similar states have similar value functions. Tesauro (1995) successfully used a multi-layer artificial neural network (ANN) with back-propagation learning to solve the backgammon task. Many others have also used ANNs in various tasks (Sutton&Barto, 1998). Unfortunately the use of state generalization generally makes the task non-Markovian, which may give great problems in making the learning converge even for simple tasks (Boyan&Moore, 1995).

2.2 Temporal difference and Q-learning

Temporal difference (TD) learning is an example of so-called *bootstrapping* methods (Sutton, 1988). Bootstrapping signifies that value function updates occur at every state transition based not only on received reward, but also on the difference between the current state value and the state value of the next state. State values are updated according to

$$V(s_t) \leftarrow V(s_t) + \beta[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \quad (2)$$

where β is a learning rate. This version is known as the *TD(0)* method. The *TD(λ)* algorithm, of which TD(0) is an instance, uses a notion of *eligibility trace*, which λ refers to (Barto et al., 1983) (Singh&Sutton, 1996). An eligibility trace memorizes at least a part of the state history of the current episode. λ is a *trace decay* parameter, which together with γ determines how quickly rewards are propagated backwards.

Q-learning (Watkins, 1989) is a TD control algorithm, which calculates action values rather than state values. In its simplest form, one-step Q-learning, action values are updated according to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (3)$$

3 SLAP reinforcement and notions of short- and long-term memory

This section starts by describing linear function approximation by the one-layer Adaline ANN and Widrow-Hoff learning (Widrow&Hoff, 1960), followed by a description of how these are used to implement the ***SLAP*** (Set Lower Action Priority) principle for guiding state-space exploration. Finally, the use of a short- and long-term memory model is explained for combining SLAP with action-value learning.

A linear function approximator calculates action values as the weighted sum of action neuron input values (Barto et al., 1983, 1990)

$$a_j(s) = \sum_{i=1}^{N} s_i w_{i,j} , \quad (4)$$

where s_i is the value of state variable i, $w_{i,j}$ is the weight of action neuron j for input i, a_j is the output value of action neuron j and N is the number of state variables. Weights are typically stored in a two-dimensional matrix of size M×N, where M is the number of actions. This representation is identical to the lookup-table representation usually used in discrete RL tasks. However, one major advantage of linear function approximators over lookup-tables is that they can handle continuous-valued state variables, where the state-space is infinite. Adalines can be trained using the Widrow-Hoff training rule

$$w_{i,j}^{new} = w_{i,j} + \alpha(a_j' - a_j)s_i \quad (5)$$

, where a_j' is the "correct" value used in supervised learning. α is a learning rate parameter that determines the step size of weight modifications. Widrow-Hoff learning is a gradient descent method that minimizes the root mean square error (RMSE) between all a_j' and a_j values. The RMSE error function has only one minimal solution for Adalines, so gradient descent is guaranteed to find optimal weights if α is small enough. The generalized Widrow-Hoff rule allows for gradient descent also in multi-layer ANNs, but then training is usually much longer and the risk of local minima in the error function means that finding optimal weights is not guaranteed. The use of multi-layer ANNs in RL tasks is covered by Barto et al. (1990), where s_i is replaced by the feature value ϕ_i that may be calculated by the hidden layer of a multi-layer ANN, be values from measuring instruments or some other mechanism that estimates the system's state.

3.1 Set Lower Action Priority (SLAP)

SLAP is based on the common-sense idea that it is not useful to continue using the same action infinitely if it does not seem to make the agent progress in the task. For a state visited previously during the same episode, for instance, taking a different action from the previous time could make exploration quicker. Also, going straight back to the previous state visited usually does not make exploration quicker, so avoiding to go back is probably useful in tasks with known invertible actions. SLAP uses the Widrow-Hoff weight update rule for decreasing the weights of these "undesired" actions, but the question is what value to use as "correct" value a_j'? The choice made is

$$a_j'(s) = a_{min}(s) - margin\ , \tag{6}$$

where $a_{min}(s)$ is the smallest $a_j(s)$ value in state s[1]. Using the "correct" value given by Equation (6) is the idea of SLAP[2]. A general algorithm for using SLAP in a learning task is given in Figure 1.

```
Initialize parameters
REPEAT (for each episode)
  s ← initial state of episode
  REPEAT (for each step in episode)
    a  ←  action given by π for s
    Take action a, observe next state s'
    SLAP "undesired" actions
    s ← s'
```

Figure 1. General algorithm for using SLAP in a learning task.

After the weight update, we get the following value for $a_j^{new}(s)$

$$\begin{aligned} a_j^{new}(s) &= \sum_{i=1}^{N} s_i w_{i,j}^{new} = \sum_{i=1}^{N} s_i \left(w_{i,j} + \alpha(a_j' - a_j)s_i\right) = \\ &\sum_{i=1}^{N} s_i w_{i,j} + \sum_{i=1}^{N} \alpha s_i^2 (a_j' - a_j) = a_j + \alpha \sum_{i=1}^{N} s_i^2 (a_j' - a_j) \end{aligned} \tag{7}$$

, where we can see that setting α to $1/\Sigma s_i^2$ guarantees that a_j^{new} will become a_j', i.e. the lowest action-value, in state s after SLAPping action j. Therefore the same action will not be used again until all other possible actions have been tested in the same state, which is especially useful in deterministic tasks. In stochastic tasks, α should be inferior to $1/\Sigma s_i^2$ because even the optimal action may not always be successful, so immediately making its action-value the lowest would not be a good idea.

As long as the value of $a_{min}(s)$ doesn't change, the value $a_j'(s)$ remains the same for all j. This is true until $a_j^{new}(s)$ becomes lower than the current $a_{min}(s)$ for some j. Therefore, action

[1] The *margin* is a constant that has been set to 0.1 in all tests performed. This value for *margin* is mainly motivated by the weight initialisation interval [0,1) used for short-term memory weights explained in the next section. Using the value 0.1 still remains intuitive and its potential influence on the results remains to be tested.

[2] In earlier work (Främling, 2002) SLAP used a slightly different weight modification rule.

values (and weight values) slowly go towards minus infinity when using SLAP an infinite number of times[3].

3.2 Short- and long-term memory

SLAP is used to guide exploration and make initial learning faster, but it does not try to maximize the amount of reward, which is not even included in SLAP calculations. This is why a dual memory model with *short-term memory* (STM) and *long-term memory* (LTM) is used. The memory model has the working name ***BIMM***, which comes from the words brain-inspired memory model[4]. When using BIMM, Adaline outputs are calculated according to

$$a_j(s) = \sum_{i=1}^{N} s_i \left(\eta_{stm} stw_{i,j} + \eta_{ltm} ltw_{i,j} \right) \tag{8}$$

, where η_{stm} is a parameter that determines the influence of STM and η_{ltm} determines the influence of LTM. $stw_{i,j}$ is the STM weight for action neuron j and input i. $ltw_{i,j}$ is the corresponding LTM weight. STM weights have been initialized to random values in the interval [0,1) and LTM weights have been initialized to zero in all tests reported here. STM is re-initialized at the end of each episode. Only STM weights are modified by the Widrow-Hoff rule, which becomes

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \tag{9}$$

The new activation value is then

$$\begin{aligned} a_j^{new}(s) &= \sum_{i=1}^{N} s_i \left(\eta_{stm} stw_{i,j}^{new} + \eta_{ltm} ltw_{i,j} \right) = \sum_{i=1}^{N} s_i \left(\eta_{stm} \left(stw_{i,j} + \alpha(a_j' - a_j)s_i \right) + \eta_{ltm} ltw_{i,j} \right) = \\ &\sum_{i=1}^{N} s_i \left(\eta_{stm} stw_{i,j} + \eta_{ltm} ltw_{i,j} \right) + \sum_{i=1}^{N} \alpha s_i^2 \eta_{stm} (a_j' - a_j) = a_j + \alpha \sum_{i=1}^{N} s_i^2 \eta_{stm} (a_j' - a_j) \end{aligned} \tag{10}$$

, where we can see that setting α to $1/(\Sigma s_i^2 \eta_{stm})$ guarantees that a_j^{new} will become a_j' in state s after SLAPping action j. Since STM connection weights are initialized to random values, adjusting the parameters η_{stm} and η_{ltm} makes it possible to balance between a stochastic policy by STM and a deterministic policy by LTM. The purpose of LTM is to be able to solve a previously encountered task more efficiently in the future.

3.2.1 Using Q-learning for long-term memory updates

Q-learning is an off-policy method, so it can be used to update LTM weights while taking greedy actions determined by Equation (8). Initial exploration is guided by SLAP and STM, while LTM is updated according to

$$ltw_{a,s} \leftarrow ltw_{a,s} + \beta \left[r_{t+1} + \gamma \max_a ltw_{a,s(t+1)} - ltw_{a,s} \right], \tag{11}$$

which is identical to Equation (3). When initializing STM weights to random values from the interval [0,1) and LTM weights to zero, η_{stm} must be lower than η_{ltm} so that Q-values in

[3] Setting $a_j'(s) = a_{max}(s) + margin$ would increase weights in the same way that SLAP decreases them, but this has not been used here.

[4] BIMM is purely the author's subjective working name for the net, for which Bi-Memory-Model could be used as well.

LTM dominate as the number of episodes increases. The η_{stm} value to use depends on reward values and discount rate. Q-learning or Sarsa with eligibility traces (Sutton&Barto, 1998) could also be used for LTM updates. Then the value function would probably converge faster, but it would not affect initial exploration.

4 Experimental results

Grid worlds are often used for comparing how quickly different RL methods converge towards a stable solution (Sutton, 1990, 1991), (Moore&Atkeson, 1993), (Kaelbling et al, 1996), (Ng et al, 1999). Grid worlds are interesting because they do not involve any "game rules" which guarantee that random search would eventually find a terminal state, as in TD-Gammon for instance (Tesauro, 1995). In real-world tasks some state transitions may be impossible, which corresponds to the situation where there are obstacles or "walls" in the grid world. Such tasks are here treated under the title of "maze problems", which may also involve transition time between states due to "corridors" in the maze.

There are also some limitations to grid worlds. States are uniquely and deterministically identified and there is a known "inverse" action to every possible action, i.e. an action that normally takes the agent back to the previous state. These limitations are not necessary for using SLAP or BIMM. Work in progress on a real-world light-seeking robot addresses these aspects, where SLAP solves the problem efficiently. However, the light-seeking robot task involves hidden state and partial observability due to noise in sensors and other sources of uncertainty in the environment. This task is therefore involves out of the scope of this paper.

Two different methods are compared: Q-learning (**Q**) and BIMM using Q-learning for LTM weights (**QBIMM**). In all tests performed, there are four possible actions that correspond to the compass directions east, west, north and south. Taking forbidden actions, i.e. actions making the agent hit a wall, was not counted. Q-learning agents avoided selecting the same forbidden action more than once by directly setting the corresponding Q-value to a small value, which corresponds to giving negative reward for hitting a wall. QBIMM agents were SLAPped when hitting a wall, which eventually leads to a valid action being selected. However, since attempting forbidden actions are not included in the step count of neither Q- nor QBIMM agents, neither method is advantaged by the selected implementation.

Optimistic initial values and reward shaping are not used here because they both violate the initial assumption of tasks where reward-giving states are sparse. Optimistic initial values are also difficult to use when reward values given by the environment are also negative, as in the classical cart-pole task (Barto et al., 1983) (Gullapalli, 1992) (Kimura&Kobayashi, 1998, 1999). Reward shaping again requires a priori knowledge about the optimal policy to learn, e.g. the direction of the "goal" in grid worlds. Finally, modifying the actual reward function often leads to sub-optimal policies (Randløv&Alstrøm, 1998) (Ng et al., 1999).

Reward +1 was given for reaching the terminal state ("goal"), while zero reward was given for all other state transitions. All tasks are performed for both deterministic and stochastic state transitions. In the deterministic case, the agent always moves in the intended direction. In the stochastic case, two stochastic levels were tested, i.e. 0.2 and 0.5. For the 0.2 level, the action selected by the agent had a 80% probability to be the one actually performed and 20% probability to be one of the three others. For the 0.5 level, the action selected by the agent had a 50% probability to be the one actually performed and 50% probability to be one of the three others.

For QBIMM agents, SLAP was used according to the algorithm in Figure 1. "Undesired" actions were SLAPped according to the following rules when entering a new state and before performing the next action: 1) SLAP the "inverse" action, 2) if the new state is already visited during the episode, SLAP action with the biggest value for Equation (8) in new state and 3) if the selected action in the new state is forbidden, SLAP action and leave agent in the same state (without counting the step as explained previously). The three cases are tested in the order indicated.

4.1 Grid world tests

Three different sizes of grid worlds are used here, where the start state and the goal state are in opposite corners of the grid world. There are no forbidden state transitions, except for those that would make the agent exit the grid world. The three different grid world sizes are 20x20 (400 states), 40x40 (1600 states) and 60x60 (3600 states).

Learning parameters for Q agents are indicated in Table 1 and for QBIMM agents in Table 2. Parameter values were determined by testing and those used here are the best ones found. In deterministic tasks β=1 is always the best learning rate for Q-learning (Kaelbling et al., 1996). In the stochastic case β=0.1 gave good results that did not change much as a function of the learning rate used.

Table 1. Q agent parameter values in grid world tests. Discount rate was γ=0.95 in all tests.

Grid world	β	ε
Deterministic (all grids)	1	0.1
Stochastic, 0.2 (all grids)	0.1	0.1
Stochastic, 0.5 (all grids)	0.1	0.1

Table 2. QBIMM parameter values in grid world tests. Discount rate was γ=0.95 in all tests.

Grid world	α	η_{stm}	η_{ltm}	β
Deterministic 20x20	10	0.1	1	1
Deterministic 40x40	200	0.005	1	1
Deterministic 60x60	1000	0.0001	1	1
Stochastic, 0.2 (20x20)	10	0.01	1	0.9
Stochastic, 0.2 (40x40)	20	0.005	1	0.9
Stochastic, 0.2 (60x60)	100	0.001	1	0.9
Stochastic, 0.5 (20x20)	10	0.005	1	0.9
Stochastic, 0.5 (40x40)	500	0.0001	1	0.9
Stochastic, 0.5 (60x60)	5000	0.00001	1	0.9

QBIMM has more parameters to adjust. However, the η_{ltm} parameter does not need to be adjusted if η_{stm} is adjusted instead. The Q learning rate β does not need much adjusting

neither. In the deterministic case, β=1 was used for all grid sizes, while β=0.9 seemed to give the best results in all stochastic worlds.

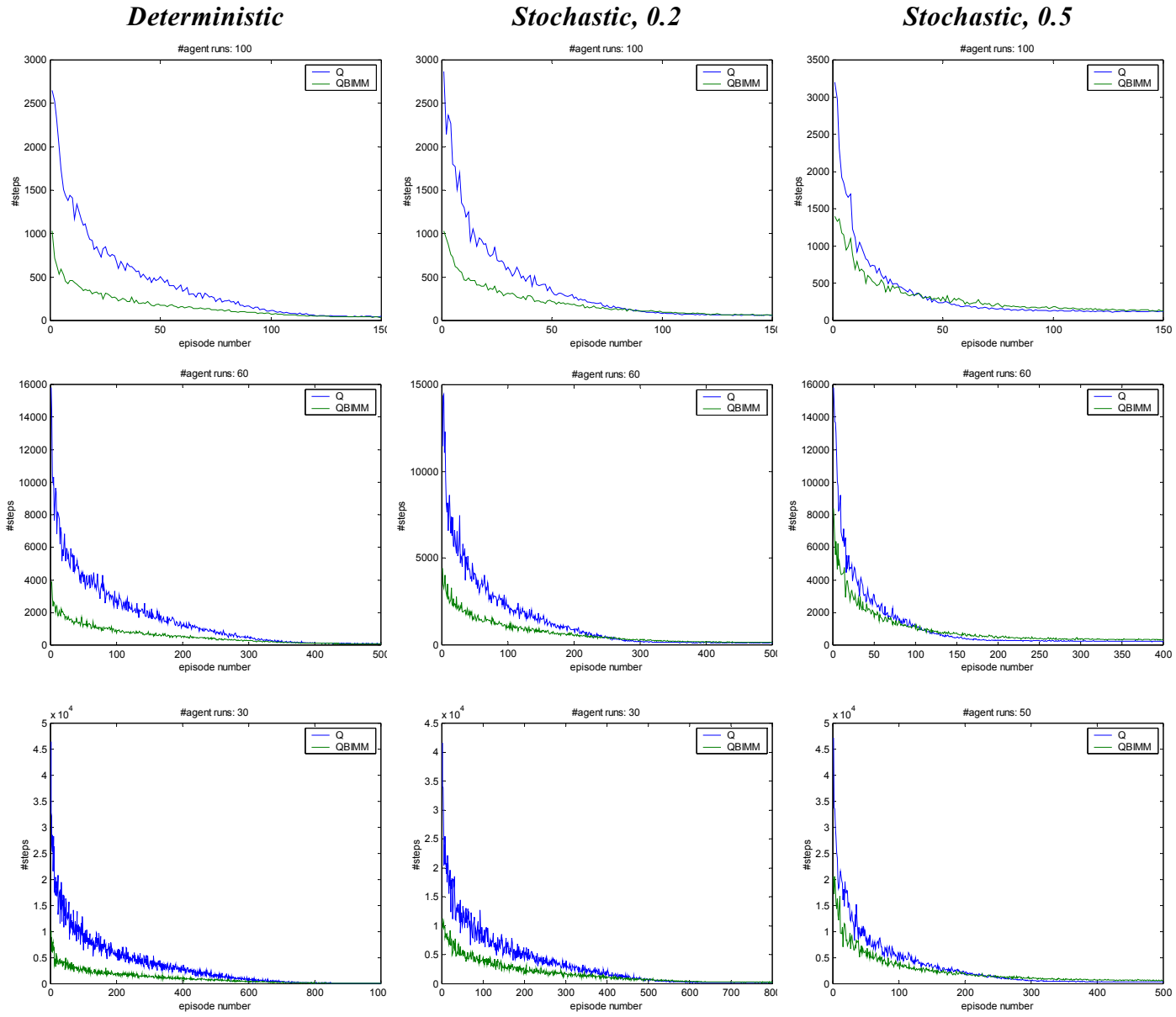

Figure 2. Grid world results as average number of steps per episode. Grid sizes are 20x20 (first row), 40x40 (second row) and 60x60 (third row). First column is deterministic grid, second and third columns are stochastic grids at levels 0.2 and 0.5. The number of agent runs used for calculating the average numbers is indicated at the top of each graph.

The choice of η_{stm} is very important for obtaining a good stable policy, but it has little impact on initial exploration. α has an impact on initial exploration, but results are not very sensitive to this parameter. Therefore α was set to $1/(\Sigma s_i^2 \eta_{stm})$[5] as explained in Section 3.2 for the deterministic grid world. Rules of thumb were used for the stochastic grid worlds,

[5] In the grid world case with uniquely identified states Σs_i^2=1 for all states.

where α was set to 10% of the deterministic value for stochastic level 0.2 and to 5% of the deterministic value for stochastic level 0.5. The small values used for η_{stm} in the bigger grid worlds are mainly for making the stable policy as good as possible, it does not have much impact on initial exploration.

The graphs in Figure 2 show the average number of steps as a function of the episode. In general it can be said that the graphs have quite a similar shape for all grid sizes at the same stochastic level. From the figures, it also seems clear that QBIMM gives the greatest advantage in the deterministic case. The more stochastic the task becomes, the smaller is the advantage of using SLAP for guiding exploration. This seems logical; in a 100% stochastic environment, random exploration will be as good as any other policy. Still, in most real-life applications state transitions are probably not as stochastic as the 20% and 50% used here.

Table 3 gives numeric comparisons for the performance of plain Q-learning and QBIMM. In the deterministic case, the advantage of QBIMM clearly increases as the size of the state space grows bigger, as indicated by the ratios in the "first episode length" and "total" columns[6]. This also seems to be the case for the stochastic level 0.2. At the stochastic level 0.5 there is still a clear advantage to QBIMM concerning initial exploration, but it is difficult to tell if the advantage increases with the size of the state space. Especially the "total" value is penalized for QBIMM agents by the fact that their stable policy episode length remains higher than for Q agents.

Table 3. Grid world results as average number of steps for the episode intervals indicated. Numbers are indicated as QBIMM/Q, that is result for QBIMM first, plain Q-learning after. First table shows results for deterministic case, second for stochasticity level 0.2 and last for stochasticity level 0.5.

Grid size	**First eps.**	**Eps. 1-10**	**Eps. 1-100**	**Last 50/100 eps.**	**Total**
20x20	1035/2646(0.39)	585/1841	229/609	52/64(50eps.)	25441/64065(0.40))
40x40	3902/15805(0.25)	2749/10321	1460/4928	93/103	279710/798430(0.35)
60x60	10255/46461(0.22)	7665/29063	3989/14809	131/144	1139600/3491300(0.33)
20x20	1032/2869(0.36)	717/1908	272/543	75/68(50eps.)	30925/57734(0.54)
40x40	4441/11459(0.39)	3386/10367	1835/4649	157/127	342350/683580(0.50)
60x60	11322/41642(0.27)	9558/26597	5783/13562	289/195	1437100/2837000(0.51)
20x20	1397/3204(0.44)	1104/1967	389/483	150/125(50eps.)	46358/54542(0.85)
40x40	8389/15796(0.53)	5651/10864	2402/3610	350/235	386950/463870(0.83)
60x60	20567/47155(0.44)	16314/27856	7146/11606	696/380	1269000/1728500(0.73)

In the deterministic case, QBIMM agents achieve a better stable policy than ε–greedy Q-learning as shown by the average number of steps per episode for the last 50 (in 20x20 grid) or 100 episodes. The difference is mainly due to the fact that ε–greedy exploration regularly puts the agent off the stable path. In both stochastic cases, however, the stable solution of QBIMM is higher than that of ε–greedy Q-learning. This is because it is impossible for

[6] For the other columns such a ratio could give misleading values due to the scale difference on the episode axis for different grid sizes.

QBIMM agents to know if they come back to a state previously encountered due to a change in the environment or to its stochastic nature, so even optimal actions only get selected as long as they do not get SLAPped too many times. The ε–greedy agent again keeps on trying the same action infinitely unless a random action is selected according to ε. If similar behavior is desired for QBIMM agents, then a decreasing η_{stm} should be used that would asymptotically approach zero. Another possibility would be to switch to ε–greedy exploration at some moment during training.

4.2 Maze tasks

Test tasks have been divided into a "grid" section and a "maze" section mainly for the following reasons:

1. In pure grid worlds like those shown in the previous section, there is a great number of optimal solutions, which are very similar to each other.
2. In real-world tasks, like a robot learning to navigate in a building, the robot only needs to take decisions when there is more than one possible (or useful) action. If a robot is going forward in a corridor, there is normally no need to decide turning back.

The grid/maze world implementation used does not treat "corridors" as decision-making states. Instead the agent continues forward until it reaches a state where it can (or has to) go in another direction than straight forward. Therefore, corridors introduce a transition time between states, where taking a bad action in some state may greatly increase the number of steps required to reach the goal.

Two test tasks are used. The first one has very few corridors, but it contains obstacles that make it more probable to find a sub-optimal policy than an optimal one. The second task is a true maze, where the number of possible solutions remains small, but where the length of different solutions greatly differs.

4.2.1 Grid world with obstacles

Sutton (1990, 1991) used the grid world shown in Figure 3 to illustrate the advantages of using a model for guiding exploration in what he called Dyna agents. This grid world is challenging because the 14-step optimal solutions have a much smaller probability of being discovered than the 16-step sub-optimal solutions. Parameter values of the Q agent are the same as in Table 1, while QBIMM parameter values are shown in Table 4.

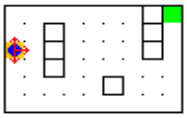

Figure 3. Grid world. Agent in start position, goal position in upper right corner.

Table 4. QBIMM parameter values in grid world with obstacle tests. γ=0.95 in all tests.

Grid world	α	η_{stm}	η_{ltm}	β
Deterministic	10	0.1	1	1
Stochastic, 0.2	1	0.1	1	0.9
Stochastic, 0.5	1	0.1	1	0.5

The results are similar to those obtained for the grid worlds in Section 4.1. Initial exploration is clearly faster especially in the deterministic world. In the stochastic worlds the difference is smaller but significant. It is interesting to notice that for stochastic level 0.5, QBIMM finds a much better stable policy than the Q agent, which is a significant difference compared to the grid world results in Section 4.1. This was achieved by setting β=0.5 instead of using β=0.9. With β=0.9 the stable policy of QBIMM was about the same as for the Q agent. Using β=0.5 was also tested for the grid worlds in Section 4.1 but without observing any similar phenomenon.

Deterministic ***Stochastic, 0.2*** ***Stochastic, 0.5***

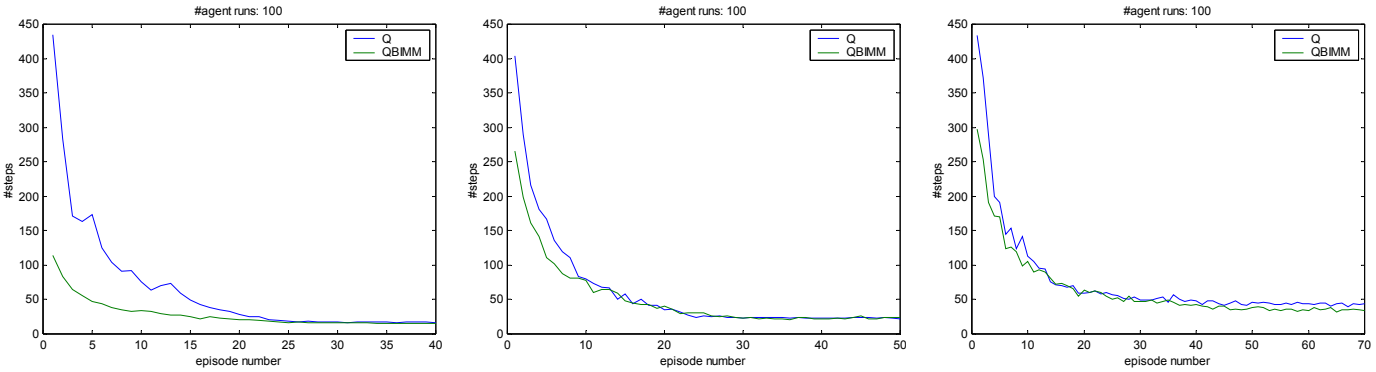

Figure 4. Grid world with obstacles results, 100-agent average number of steps per episode.

Table 5 compares the number of steps per episode during initial exploration for all methods as well as the average number of steps per episode for the stable policy. The results confirm that the more deterministic the environment is, the bigger is the advantage of using SLAP to guide initial exploration.

Table 5. Grid world with obstacles results. First table shows results for deterministic case, second for stochastic level 0.2 and third for stochastic level 0.5.

Agent	**Steps first eps.**	**Average eps. 1-10**	**Average last 10 eps.**	**Total**
Q	435	172	17.2	2580
QBIMM	114	55	15.9	1141

Q	404	179	23.2	3058
QBIMM	265	131	23.4	2559

Q	434	216	43.3	5320
QBIMM	297	166	35.3	4492

4.2.2 Maze with transition delays

Bradtke&Duff (1995) have studied the use of RL methods in Semi-Markov Decision Processes (SMDP's). The difference between SMDP's and MDP's is that SMDP's may include continuous time. This signifies that the transition time from one state to another

depends on a probability distribution F_{xy}. Packet routing in dynamically changing networks (Boyan&Littman, 1994) is an example of an SMDP.

The test task used here introduces state transition time by the notion of "corridors", i.e, states with only two possible actions that represent opposite directions. Since state transition times are deterministic, the notion of transition time here mainly introduces an "extra penalty" for bad decisions like walking around in circles or running into dead-ends.

The maze world is of size 20x20 (Figure 5), where 10 supplementary doors have been opened in addition to the initial unique solution. This signifies that there are many possible routes and it also makes loops possible.

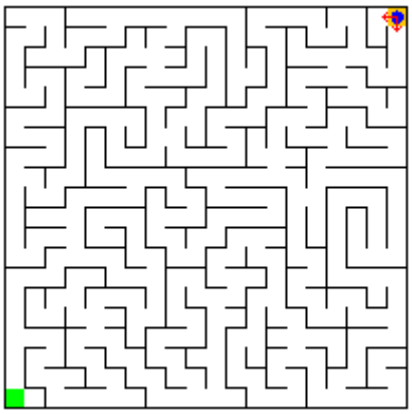

Figure 5. Maze with 10 supplementary "doors" opened in the walls in addition to the initial unique route. Agent in start position, goal position in lower left corner

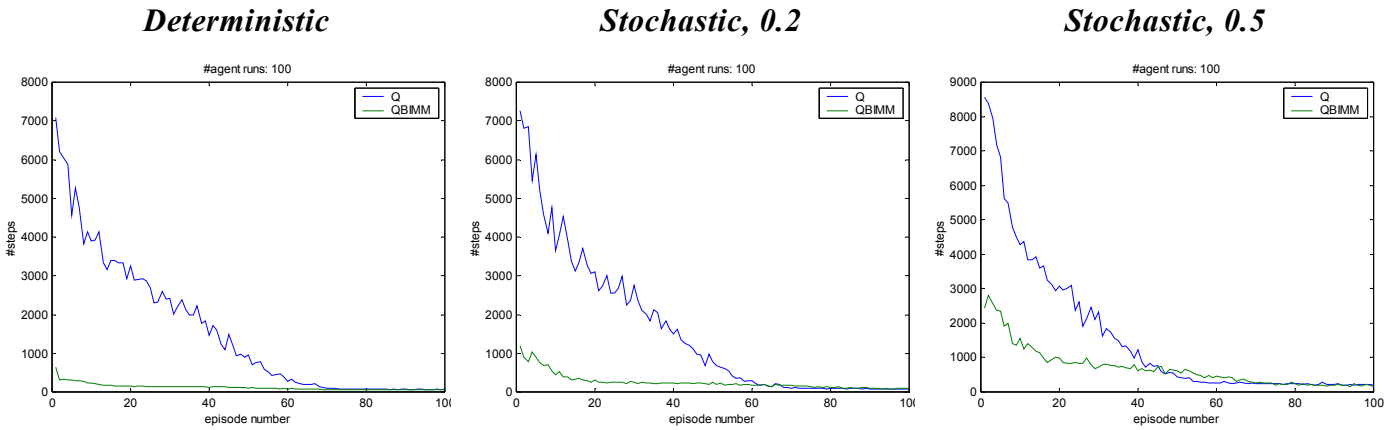

Figure 6. 20x20 maze results as average number of steps per episode, calculated for 100 agent runs.

Figure 6 shows the average number of steps per episode for Q-agents and QBIMM agents. In this task, the advantage of QBIMM is greater than in the grid world tasks. The advantage remains great for the stochastic worlds. QBIMM agents also find better stable policies than Q agents as shown by the results in the fourth column of Table 6.

Table 6. 20x20 maze results. First table shows results for deterministic case, second for stochastic level 0.2 and third for stochastic level 0.5. The numbers in parenthesis in the fourth column indicate average number of steps for the stable policy, calculated for episodes 200-250.

Agent	**First episode**	**Average eps. 1-10**	**Average eps. 91-100**	**Total**
Q	7078	5163	69.4 (66.7)	153800
QBIMM	650	326	60.3 (60.3)	12498

Q	7251	5480	84.0 (83.3)	155530
QBIMM	1182	792	92.3 (82.3)	25520

Q	8557	6353	211.5 (178.2)	157750
QBIMM	2426	2076	196.2 (167.7)	68970

5 Conclusions

The results show that SLAP and BIMM can reduce the length of initial exploration both in deterministic and stochastic environments. It is also shown that benefits in initial exploration do not reduce the probability of finding optimal solutions. Therefore, the two goals set out in the introduction are achieved.

One big difference between BIMM and existing methods for making initial exploration faster is that BIMM agents only use their own internal information and experience. This makes them interesting compared with methods like reward shaping, which usually require some a priori knowledge about the environment, such as where the goal is located, the stochastic level of the environment or the number of sub goals to reach.

Even though only grid world and maze problems were used here, it should be possible to generalize the results to many other RL tasks. This should be the case especially for tasks where the environment does not guarantee that random search would find an initial solution in reasonable time. Since SLAP and BIMM use standard ANN structures and learning rules, they are also applicable to tasks involving continuous-valued state variables. Such tasks are a subject of current and future research, where explosion of the state-space size due to state variable discretization is a problem.

References

Barto, A.G., Sutton, R.S., Anderson, C.W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 13, No. 5. 835-846.

Barto, Andrew G., Sutton, R.S., Watkins C.J.C.H (1990). Learning and Sequential Decision Making. In: M. Gabriel and J. Moore (eds.), *Learning and computational neuroscience : foundations of adaptive networks*, M.I.T. Press, Cambridge, Mass.

Boyan, J. A., Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In: Cowan, J. D., Tesauro, G., and Alspector, J. (eds.), *Advances in Neural Information Processing Systems 6* (NIPS), Morgan Kaufmann. 671-678.

Boyan, J. A., Moore, A. W. (1995). Generalization in Reinforcement Learning: Safely Approximating the Value Function. In: Tesauro, G., Touretzky, D., Leen, T., (eds.), *Advances in Neural Information Processing Systems 7*, Morgan-Kaufmann. 369-376.

Bradtke, S.J., Duff, M.O. (1995). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In: Tesauro, G., Touretzky, D., Leen, T., (eds.), *Advances in Neural Information Processing Systems 7*, Morgan-Kaufmann.

Främling, K. (2002). Reducing state space exploration in reinforcement learning problems by rapid identification of initial solutions and progressive improvement of them. In: *Grmela, Ales, Mastorakis, Nikos E. (eds): Advances in Neural Networks World, Proc. of 3rd WSES Int. Conf. NNA'02, Interlaken, Switzerland*. 83-88.

Gullapalli, Vijaykumar (1992). *Reinforcement Learning and its Application to Control*. (Ph.D. Thesis) COINS Technical Report 92-10, Univ. of Massachusetts, Amherst. 171 p.

Kaelbling, L.P., Littman, M.L., Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, Vol. 4, 237-285.

Kakade, Sham, Dayan, Peter (2001). Dopamine Bonuses. In: Todd Leen, Tom Dietterich, Volker Tresp (eds.) *Advances in Neural Information Processing Systems 13*, MIT Press, Cambridge, MA.

Kimura, Hajime, Kobayashi, Shigenobu (1998). An Analysis of Actor/Critic Algorithms using Eligibility Traces: Reinforcement Learning with Imperfect Value Functions. *Proc. of 15th Int. Conf. on Machine Learning*. 278-286.

Kimura, Hajime, Kobayashi, Shigenobu (1999). Efficient Non-Linear Control by Combining Q-learning with Local Linear Controllers. *Proc. of 16th Int. Conf. on Machine Learning*. 210-219.

Mataric, Maja J. (1994). Reward Functions for Accelerated Learning. In: Cohen, W. W., Hirsch, H. (eds.) *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan-Kaufmann, CA.

Moore, Andrew W., Atkeson, Christopher G (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, Vol. 13. 103-130.

Ng, Andrew Y., Harada, Daishi, Russell, Stuart (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Proc. of 16th Int. Conf. on Machine Learning*.

Randløv, J., Alstrøm, P. (1998). Learning to Drive a Bicycle using Reinforcement Learning and Shaping. *Proc. of ICML-98 conference*. 463-471.

Randløv, Jette (2000). Shaping in Reinforcement Learning by Changing the Physics of the Problem. *Proc. of ICML-2000 conference*. 767-774.

Singh, S.P., Sutton, R.S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, Vol. 22. 123-158.

Sutton, R.S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, Vol. 3. 9-44.

Sutton, R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh International Conference on Machine Learning*, Morgan Kaufmann Publishers. 216-224.

Sutton, Richard S. (1991). Integrated Modeling and Control Based on Reinforcement Learning and Dynamic Programming. In: Richard Lippmann, John Moody, David Touretzky (eds) *Advances in Neural Information Processing Systems 3*, Morgan-Kaufmann. 471-478.

Sutton, R.S., Barto, A.G. (1998). *Reinforcement Learning*. A Bradford Book, MIT Press, Cambridge, MA. 322 p.

Tesauro, G.J. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, Vol. 38, No. 3. 58-68.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.

Widrow, B., Hoff, M.E. (1960). Adaptive switching circuits. *1960 WESCON Convention record Part IV*, Institute of Radio Engineers, New York. 96-104.