# APRENDIZAGEM POR REFORÇO PARA TIMES DE ROBÔS

Carlos Henrique Costa Ribeiro – PQ
Lucas Heitzmann Gabrielli – IC

*Resumo: Times de robôs autônomos podem propiciar uma forma mais eficiente de solução de tarefas complexas através da sua divisão em subtarefas mais simples. Este artigo apresenta os resultados obtidos com uma técnica de representação compacta (CMAC) para representação de estados e aceleração de convergência de um algoritmo de aprendizagem por reforço para times de robôs em um domínio simplificado de futebol de robôs. Para uma redução do espaço de estados de uma ordem de magnitude, o algoritmo – através da generalização incorporada pela representação compacta – conseguiu aprender uma política de ações adequada.*

*Abstract: Autonomous robots teams can provide a more efficient way of solving complex tasks through simpler subtask splitting. This article presents results obtained with a compact representation technique (CMAC - Cerebellar Model Articulation Controller) for state representation and convergence acceleration of a robotic team reinforcement learning algorithm in a simplified robot soccer domain. For a tenfold state space reduction, the algorithm – via generalization incorporated by the compact representation – managed to learn an adequate action policy.*

## 1. INTRODUÇÃO

Times de robôs têm se mostrado promissores em tarefas que envolvem um elevado grau de complexidade caso sejam realizadas por um único agente. Nosso interesse é o estudo de estratégias de controle obtidas de forma autônoma (ou seja, sem modelagem prévia da dinâmica do ambiente), baseadas em aprendizado por reforço, para times de agentes robóticos atuando em conjunto.

Para tanto, escolhemos estudar o domínio do futebol de robôs, pois ele apresenta um alto grau de complexidade e a possibilidade de ações coordenadas entre os agentes com ou sem a existência de comunicação explícita. Mesmo nos casos mais simples este domínio envolve estratégias complexas e espaços de estado de grandes dimensões, inviabilizando a utilização de representações tabulares sobre as quais os algoritmos de aprendizado por reforço normalmente operam. Além disso, estes algoritmos tem convergência notoriamente lenta, pois dependem de uma modelagem implícita do ambiente (obtida de forma autônoma) a partir da qual as políticas de ações adequadas são obtidas.

De modo a acelerar a convergência e minimizar a dependência de uma representação tabular completa – inviável em problemas envolvendo times de robôs nos quais uma discretização fina do espaço de estados é necessária – propomos aqui o uso de uma representação tabular compacta aplicada a um algoritmo de aprendizado por reforço específico para jogos envolvendo pelo menos dois agentes. Os resultados obtidos mostram que, para uma redução do espaço de estados de uma ordem de magnitude, o algoritmo, através da generalização incorporada pela representação compacta, conseguiu aprender uma política de ações adequada.

## 2. APRENDIZAGEM POR REFORÇO

### 2.1 Princípios Básicos

A teoria de aprendizagem por reforço é teoricamente restrita a processos decisórios de Markov, apesar de as idéias e métodos poderem ser aplicados de forma mais genérica.

Um ambiente satisfaz a propriedade de Markov se o seu estado resume o passado de forma compacta, sem perder a habilidade de prever o futuro. Isto é, pode-se predizer qual será o próximo estado e próxima recompensa esperada dado o estado e ação atuais (Sutton, R. S. e Barto, A. G. (1998)). Um processo de Markov é uma seqüência de estados, com a propriedade de que qualquer predição de valor de estado futuro dependerá apenas do estado e ação tuais, e não da seqüência de estados passados.

Um processo de aprendizagem por reforço que satisfaz a propriedade de Markov é chamado de processo decisório de Markov (MDP - *Markov Decision Process*). Se o espaço de estados e ações for finito, então ele é chamado de processo decisório de Markov finito, base para a teoria de aprendizagem por reforço, que assume o ambiente como sendo deste tipo.

Formalmente, um processo decisório de Markov é definido por um conjunto $<S,A,P,R>$, onde temos: um conjunto finito de estados $S$ do sistema, um conjunto finito de ações $A$, um modelo de transição de estados $P$, que mapeia os pares estado-ação em uma distribuição de probabilidades sobre o conjunto de estados, e, finalmente, uma função de recompensa $R$, que especifica o reforço que o agente recebe por escolher uma determinada ação $a \in A$ no estado $s \in S$ . O estado $s$ e a ação $a$ atuais determinam (a) o próximo estado s' de acordo com a probabilidade $P(s'|s,a)$, e (b) a recompensa $r(s,a)$ associada.

Se o modelo de transição de estados for conhecido, técnicas de Controle Ótimo podem – ao menos em tese – ser utilizadas para determinar uma política ótima de ações a ser seguida pelo agente. Alternativamente, aprendizagem por reforço é utilizada quando o modelo não está disponível (aprendizagem autônoma) ou quando existe apenas um modelo de simulação, que não permite a formulação analítica necessária para algoritmos de Programação Dinâmica.

Aprendizagem por reforço preocupa-se com o problema de um agente aprender, por tentativa e erro, a atingir um objetivo interagindo com o seu ambiente. O agente e o ambiente interagem em uma seqüência discreta de passos no tempo, $t=0,1,2,\ldots$. O estado e a ação em um dado instante ($s_t \in S$ e $a_t \in A$), determinam a distribuição de probabilidades para o estado $s_{t+1}$ e o reforço $r_t$. O objetivo do agente normalmente é escolher ações de modo a maximizar uma soma descontada dos reforços subseqüentes:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k} \qquad (1)$$

onde a taxa de desconto $0 < \gamma \leq 1$ determina o peso temporal relativo dos reforços. Existem formulações alternativas para esta função de otimização (Sutton, R. S. e Barto, A. G. (1998)).

As escolhas das ações do agente são feitas a partir de uma função do estado, chamada política, $\pi.S \rightarrow A$. O valor de utilidade de um estado, dada uma política, é o reforço esperado partindo do estado e seguindo a política:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \qquad (2)$$

e a política ótima de ações é aquela que maximiza o valor de estado:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \qquad (3)$$

Existe sempre ao menos uma política ótima $\pi^*$, que produz o valor de utilidade máximo em todos os estados $s \in S$.

Paralelamente a essas duas funções de valor de estado, existem duas funções de valor de ação,

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} \qquad (4)$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \qquad (5)$$

A partir de Q*, pode-se determinar uma política ótima apenas como $\pi^*(s) = \arg\max_a Q(s,a)$.

O método de aprendizagem por reforço mais popular é o Q-*learning* (Watkins, C. J. C. and Dayan, P. (1992)). Trata-se de um algoritmo que permite estabelecer autonomamente uma política de ações de maneira interativa. Pode-se demonstrar que o algoritmo *Q-learning* converge para um procedimento de controle ótimo, quando a hipótese de aprendizagem de pares estado-ação $Q$ for representada por uma tabela completa contendo a informação de valor de cada par. A convergência ocorre tanto em processos decisórios de Markov determinísticos quanto não-determinísticos.

A idéia básica por trás do *Q-learning* é que o algoritmo de aprendizagem aprende uma função de avaliação ótima sobre todo o espaço de pares estado-ação $S \times A$. A função $Q$ fornece um mapeamento da forma $Q:S \times A \rightarrow V$, onde $V$ é o valor de utilidade esperado ao se executar uma ação $a$ no estado $s$. Desde que o particionamento do espaço de estados do robô e o particionamento do espaço de ações não omitam informações relevantes nem introduzam novas, uma vez que a função ótima $Q$ seja aprendida o agente saberá precisamente que ação resultará na maior recompensa futura em uma situação particular $s$.

A função *Q(s,a)*, da recompensa futura esperada ao se escolher a ação *a* no estado *s*, é aprendida através de tentativa e erro segundo a equação a seguir:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_t + \gamma V_t(s_{t+1}) - Q_t(s_t, a_t)] \quad (6)$$

onde $\alpha$ é a taxa de aprendizagem, *r* é a recompensa, ou custo, resultante de tomar a ação *a* no estado *s*, $\gamma$ é o fator de desconto e o termo $V_t(s_{t+1}) = \max_a Q(s_{t+1}, a)$ é a utilidade do estado *s* resultante da ação *a*, obtida utilizando a função *Q* que foi aprendida até o presente.

A função *Q* representa a recompensa descontada esperada ao se tomar uma ação *a* quando visitando o estado *s*, e seguindo-se uma política ótima desde então.

A forma procedimental do algoritmo Q-*learning* é:

Para cada *s,a* inicialize *Q(s,a)=0*
Observe *s*
Repita
- Selecione ação *a* usando a política de ações atual
- Execute a ação *a*
- Receba a recompensa imediata *r(s,a)*
- Observe o novo estado *s'*
- Atualize o item *Q(s,a)* de acordo com a equação (6)
- *s* ← *s'*

Até que critério de parada seja satisfeito

Uma vez que todos os pares estado-ação tenham sido visitados um número infinito de vezes, garante-se que o método gerará estimativas $Q_t$ que convergem para o valor de *Q* (Watkins, C. J. C. and Dayan, P. (1992)). Na prática, a política de ações converge para a política ótima em tempo finito, embora de forma lenta.

### 2.2 Jogos de Markov

Considere n agentes interagindo com um ambiente através de percepções e ações. Em cada interação, o agente i observa o estado $s_t$ e escolhe uma ação $a_i$ a executar. O conjunto de ações $a_1$, ..., $a_n$ altera o estado $s_t$ do ambiente, e um reforço escalar $r_i$ é produzido para cada agente, indicando e desejabilidade do estado resultante.

Formalmente, um jogo de Markov é definido por um conjunto $\langle n, S, A_1, ..., A_n, r_1, ..., r_n, P \rangle$, onde temos: um número n de agentes, um conjunto finito de estados *S* do sistema, conjuntos finito de ações $A_i$ para cada agente i, reforços $r_i$ recebidos por cada agente, e um modelo de transição de estados P, que mapeia as n-uplas (s, $a_1$, $a_2$, ..., $a_n$) em uma distribuição de probabilidades sobre o conjunto de estados.

### 2.3 Algoritmo Minimax-Q

O algoritmo Minimax-Q (Littman, M., (1994)) é uma generalização de Q-learning para um caso particular dos jogos de Markov, para casos de dois agentes executando ações em passos alternados, em um jogo de soma zero.

Sejam A e B os jogadores envolvidos. O objetivo de A é aprender uma política ótima de ações que maximize o valor esperado descontado de reforços. O aprendizado desta política é em geral muito difícil, pois depende das ações executadas pelo adversário. Uma solução para este problema é avaliar cada política com relação à estratégia do adversário que a torna pior, no espírito da técnica Minimax frequentemente utilizada em jogos modelados. Para políticas de ação determinísticas, o valor ótimo de um estado s em um jogo de Markov é:

$$V^*(s) = \max_\pi \min_b Q(s, a, b) \quad (7)$$

e a regra de atualização Minimax-Q é:

$$\Delta Q_t(s_t, a_t, b_t) = \alpha[r_t + \gamma V_t(s_{t+1}) - Q_t(s_t, a_t, b_t)] \quad (8)$$

onde $\Delta Q_t(s_t, a_t, b_t) = Q_{t+1}(s_t, a_t, b_t) - Q_t(s_t, a_t, b_t)$ $\alpha$ é a taxa de aprendizagem, *r* é a recompensa, ou custo, resultante de A tomar a ação *a* e B tomar a ação b no estado *s*, $\gamma$ é o fator de desconto.

## 3. REPRESENTAÇÃO CMAC

O princípio do CMAC (Albus, J. (1975)) é realizar várias discretizações independentes do ambiente (tilings) e mapeá-las aleatoriamente em um único vetor unidimensional de dimensão bem inferior ao espaço de estados original, definindo desta forma uma representação compacta sobre a qual o algoritmo de aprendizado operará.

Associamos a cada membro do vetor unidimensional um conjunto de posições (discretas) de cada tiling aleatoriamente, de forma que, a cada ponto do ambiente, corresponde uma posição de cada tiling e, portanto, um conjunto de valores do vetor unidimensional (figura 1). O valor então associado a cada ponto (ou seja, Q(s,a,b)) é calculado como a média dos valores do vetor que representam tal ponto, e a atualização (8) é feita em todos esses membros do vetor (figura 2).

Desse modo, com a escolha adequada do número de tilings, refinamento de cada um e tamanho do vetor unidimensional, podemos ver que a aprendizagem pode ser acelerada, pois ocorrerá tanto no ponto escolhido como na sua vizinhança, já que pontos próximos acabarão tendo representação mais semelhante no vetor unidimensional e pontos mais distantes, terão menor semelhança. Além disso, sobrepondo-se os vários tilings, obtemos um ambiente mais refinado que o inicialmente utilizado.

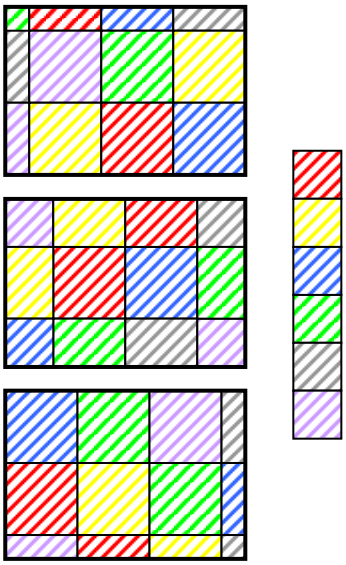

Figura 1 – Associações das partições dos tilings com o vetor unidimensional. No caso, o vetor (à esquerda) possui dimensão 6, enquanto cada um dos 3 tilings (à direita) tem dimensão 12, obrigando-nos a associar dois estados de cada tiling a cada membro do vetor (mesma cor)

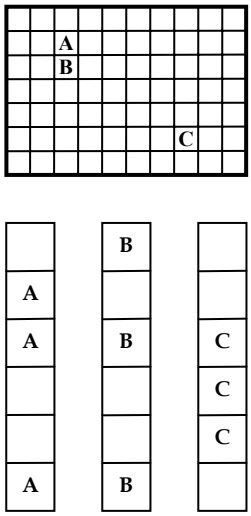


Figura 2 – Correspondência de alguns pontos com os membros do vetor. O ambiente está representado como superposição dos tilings, mostrando-se mais refinado. Cada uma das 3 alocações de valores do vetor à direita representa um dos 3 pontos. Observamos que pontos próximos têm maior semelhança de representação, enquanto pontos mais distantes são representados com menos membros em comum.

## 4. ARRANJO EXPERIMENTAL E RESULTADOS

Para a obtenção dos resultados foi utilizado um simulador escrito em Java chamado TeamBots (http://www.teambots.org) por sua facilidade de uso e recursos apresentados. Ele propicia a criação do sistema de controle do agente e uma grande variedade de ambientes. Especificamente para o domínio do futebol de robôs, é possível determinar o número de jogadores de cada time (no nosso caso, apenas dois jogadores) e selecionar o sistema de controle de cada agente independentemente.

O campo foi particionado como uma grade de 20 células (5 x 4) e os movimentos dos jogadores foram limitados a apenas 9: frente (norte), trás (sul), laterais (esquerda, direita), diagonais (nordeste, sudeste, sudoeste, noroeste) e permanecer parado. Sempre que o jogador estiver na mesma casa que a bola, ele automaticamente se posiciona de forma a chutar a bola em direção ao gol adversário, caso contrário, ele fica no centro da casa. O jogador chuta a bola apenas quando estiver corretamente posicionado e selecionar a ação de permanecer parado.

Considerando que os resultados envolvem apenas dois jogadores (um jogador A e seu adversário B), o número de estados é definido pela multiplicação dos seguintes valores: número de

possíveis posições para A, para B e para a bola e número de possíveis movimentos para A e para B, o que resulta em um total de 20 x 20 x 20 x 9 x 9 = 648.000 estados. O vetor unidimensional utilizado para a codificação CMAC é de 65.000 posições, representando uma redução de aproximadamente 10 vezes. Note que a generalização é uma consequência direta da representação compacta CMAC.

Os parâmetros utilizados para o algoritmo Minimax-Q foram $\gamma = 0.9$ e $\alpha = 1.0$, decaindo linearmente a 0.0 ao final de 150 jogos. Cada jogo corresponde a uma partida de aproximadamente 1 minuto de duração.

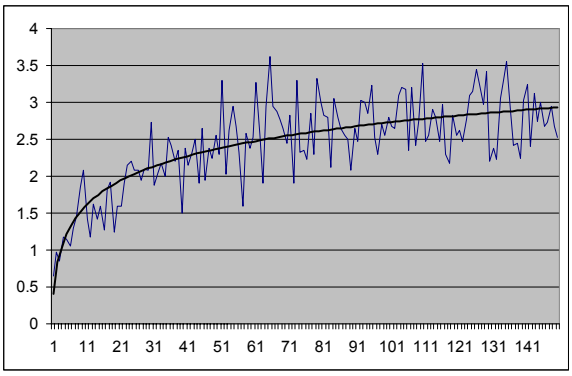


Gráfico 1 – Curva de aprendizagem média, 8 tilings, oponente aleatório.

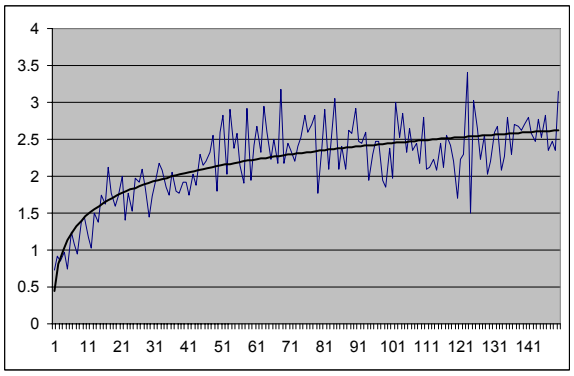


Gráfico 3 – Curva de aprendizagem média, 12 tilings, oponente aleatório.

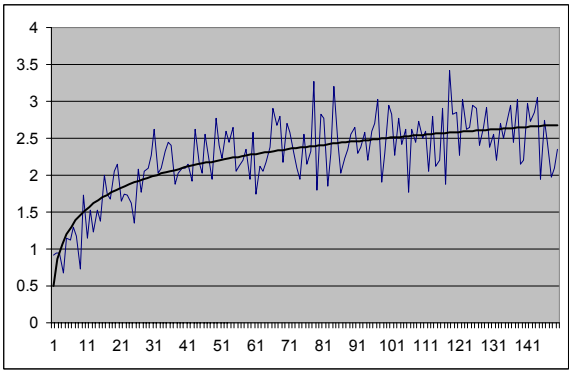


Gráfico 2 – Curva de aprendizagem média, 10 tilings, oponente aleatório.

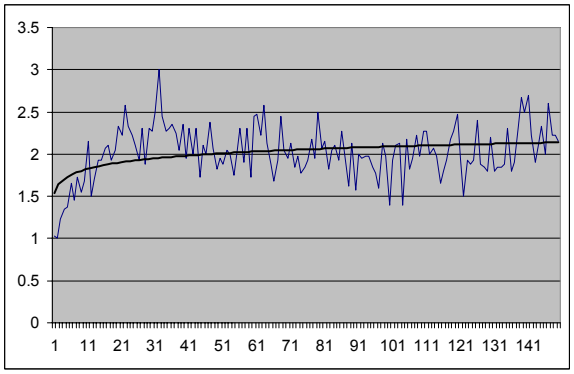


Gráfico 4 – Curva de aprendizagem média, 8 tilings, oponente autônomo.

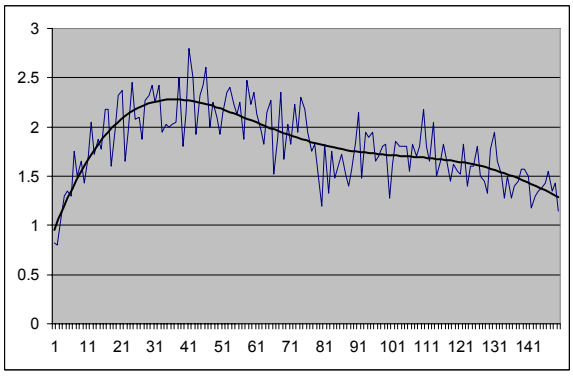


Gráfico 5 – Curva de aprendizagem média, 10 tilings, oponente autônomo.

Os gráficos 1 a 5 mostram os resultados obtidos (curvas de aprendizagem média sobre 40 conjuntos de 150 jogos em cada caso) com o uso do algoritmo Minimax-Q para codificações CMAC sobre a grade de 20 partições, para 8, 10 e 12 tilings retangulares, definidos sobre a grade original e arranjados com deslocamento uniforme sobre esta última. O eixo *x* indica o jogo realizado, e o eixo *y* indica o saldo de gols obtido. A linha contínua mais escura é uma interpolação logaritmica das curvas obtidas, para uma indicação mais clara da tendência observada ao longo do processo de aprendizagem,

com excessão do gráfico 5 (interpolação polinomial). O jogador B executa uma política aleatória de ações nos 3 primeiros gráficos e o algoritmo CMAC com representação tabular completa nos demais.

Nota-se que nos 3 primeiros casos ocorreu aprendizagem rápida de uma política de ações vitoriosa, com uma tendência de aumento do saldo de gols permanecendo ao final dos 150 jogos. A convergência mais rápida ocorreu para 8 tilings, o que indica que um grau maior de generalização (menos tilings) mostrou-se conveniente neste problema.

Contra o adversário autônomo observam-se dois comportamentos bastante distintos. No primeiro caso (gráfico 4) o agente demora um pouco mais para obter uma grade vantagem de gols, mas consegue mantê-la com o passar do tempo, quando o oponente também vai aprendedo. Já no segundo caso (gráfico 5) o jogador consegue uma política muito boa rapidamente, mas que não leva a uma generalização satisfatória conforme o adversário se desenvolve. Deve-se optar então entre a rapidez do aprendizado ou a robustez da política de ações obtida.

## 5. CONCLUSÕES

Os resultados mostram que a utilização de representações compactas do tipo CMAC, aliadas ao uso de um algoritmo de aprendizagem por reforço pode produzir convergência rápida em problemas envolvendo times de robôs atuando em domínios não-modelados previamente.

Como trabalhos futuros, indica-se a avaliação de desempenho para um aumento da dimensão do espaço de estados, através da inclusão de mais jogadores, e uma comparação do método CMAC com outras técnicas de generalizações, sejam basedas em representações compactas (Tsitsiklis, J. e Bertsekas, D. (1996)) ou não, tais como aquelas discutidas e avaliadas em (Ribeiro, C., Pegoraro, R. e Reali Costa, A. (2002)).

## AGRADECIMENTOS

Autor e coordenador agradecem o apoio financeiro do CNPq através do Programa Institucional de Bolsas de Iniciação Científica – PIBIC.

## REFERÊNCIAS BIBLIOGRÁFICAS

1. Albus, J.; *A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)*; Journal of Dynamic Systems, Measurement and Control. American Soc. of Mechanical Engineers, **1975**.
2. Littman, M.; *Markov games as a framework for multi-agent reinforcement learning*; Proceedings of the Eleventh International Conference on Machine Learning, San Francisco, CA. Morgan Kaufmann, **1994**; p. 157.
3. Ribeiro, C., Pegoraro, R. and Reali Costa, A.; *Experience Generalization for Concurrent Reinforcement Learners: the Minimax-QS Algorithm*; Procs. of the First Int. Joint Conf. on Autonomous Agents and Multiagent Systems - AAMAS2002, Bologna, ACM Press, **2002**; v. 3, p. 1239.
4. Sutton, R. S. e Barto, A. G.; *Reinforcement Learning: An Introduction*; Massachusetts: MIT Press, **1998**.
5. Watkins, C. J. C. H. e Dayan, P.; *Q-learning*; Machine Learning, **1992**; n. 8, 279.
6. Tsitsiklis, J. e Bertsekas, D. (1996). Neuro-Dynamic Programming. Athena Scientific, **1996**.
7. Ribeiro, C., Reali Costa, A. and Romero, R.; *Robôs Móveis Inteligentes: Princípios e Técnica*; A.T. Martins e D.L. Borges, editors, Anais do XXI Congresso da Sociedade Brasileira de Computação, **2001**; vol. 3, p. 257.
8. Ribeiro, C.; *A Tutorial on Reinforcement Learning Techniques*, in A. P. Braga, editor, CD-ROM textbook version of the Supervised Learning track tutorials of the 1999 International Joint Conference on Neural Networks, INNS Press, Washington DC, **July 1999**.