VQQL. Applying Vector Quantization to Reinforcement Learning 

Fernando Fernández and Daniel Borrajo

Universidad Carlos III de Madrid
Avda. de la Universidad, 30
28912 Leganés. Madrid (Spain)
ffernand@scalab.uc3m.es, dborrajo@ia.uc3m.es
http://scalab.uc3m.es/~ffernand, http://scalab.uc3m.es/~dborrajo

Abstract Reinforcement learning has proven to be a set of successful techniques for finding optimal policies on uncertain and/or dynamic domains, such as the RoboCup. One of the problems on using such techniques appears with large state and action spaces, as it is the case of input information coming from the Robosoccer simulator. In this paper, we describe a new mechanism for solving the states generalization problem in reinforcement learning algorithms. This clustering mechanism is based on the vector quantization technique for signal analog-to-digital conversion and compression, and on the Generalized Lloyd Algorithm for the design of vector quantizers. Furthermore, we present the VQQL model, that integrates Q-Learning as reinforcement learning technique and vector quantization as state generalization technique. We show some results on applying this model to learning the interception task skill for Robosoccer agents.

1 Introduction

Real world for autonomous robots is dynamic and unpredictable. Thus, for most robotic tasks, having a perfect domain theory (model) of how the actions of the robot affect the environment is usually an ideal. There are two ways of providing such models to robotic controllers: by careful and painstaking "ad-hoc" manual design of skills; or by automatically acquiring such skills. There have been already many different approaches for learning skills in robotic tasks, such as genetic algorithms [6], or neural networks and EBL [13].

Among them, reinforcement learning techniques have proven to be very useful when modeling the robot worlds as MDP or POMDP problems [1, 12, 17]. However, when using reinforcement learning techniques with large state and/or action spaces, two efficiency problems appear: the size of the state-action tables and the correct use of the experience. Current solutions to this problem rely on applying generalization techniques to the states and/or actions. Some systems have used decision trees [3], neural networks [9], or variable resolution dynamic programming [14].

In this paper, we present an approach to solve the generalization problem that uses a numerical clustering method: the generalized Lloyd algorithm for the design of vector quantizers [10]. This technique is extensively employed for signal analog-to-digital conversion and compression, which have common characteristics to MDP problems. We have used Q-Learning [18] as reinforcement learning algorithm, integrating it with vector quantization techniques in the VQQL model.

We have applied this model for compacting the set of states that a Robosoccer agent perceives, thus dramatically reducing the reinforcement table size. In particular, we have used the combination of vector quantization and reinforcement learning for acquiring the ball interception skill for agents playing in the Robosoccer simulator [16].

We introduce the reinforcement learning and the Q-learning algorithm in section 2. Then, the vector quantization technique and the generalized Lloyd algorithm are described in section 3. Section 4 describes how vector quantization is used to solve the generalization problem in the model VQQL, and in sections 5 and 6, the experiments performed to verify the utility of the model and the results are shown. Finally, the related work and conclusions are discussed.

2 Reinforcement Learning

The main objective of reinforcement learning is to automatically acquire knowledge to better decide what action an agent should perform at any moment to optimally achieve a goal. Among many different reinforcement learning techniques, Q-learning has been very widely used [18]. The Q-learning algorithm for non deterministic Markov decision processes is described in table 1 (execution of the same action from the same state by an agent arrives to different states, so different rewards could be obtained). It needs a definition of the possible states, $\mathcal{S}$, the actions that the agent can perform in the environment, $\mathcal{A}$, and the rewards that it receives at any moment for the states it arrives to after applying each action, r. It dynamically generates a reinforcement table $Q(s,a)$ (using equation 1) that allows it to follow a potentially optimal policy. Parameter γ controls the relative importance of future actions rewards with respect to immediate rewards. Parameter α refers to the probabilities involved, and it is computed using equation 2.

$$Q_n(s,a) \leftarrow (1-\alpha_n)Q_{n-1}(s,a) + \alpha_n\{r + \gamma \max_{a'} Q_{n-1}(s',a')\} \tag{1}$$

$$\alpha_n \leftarrow \frac{1}{1+\text{visits}_n(s,a)} \tag{2}$$

where $\text{visits}_n(s,a)$ is the total number of times that the state-action entry has been visited.

Q-learning algorithm (S, A)
For each pair ($s \in \mathcal{S}$, $a \in \mathcal{A}$), initialize the table entry $Q(s, a)$ to 0.
Observe the current state s
Do forever
– Select an action a and execute it
– Receive immediate reward r
– Observe the new state s'
– Update the table entry for $Q(s, a)$ using equation 1
– Set s to s'

Table1. Q-learning algorithm.

3 Vector Quantization (*VQ*)

Vector quantization appeared as an appropriate way of reducing the number of bits needed to represent and transmit information [7]. In the case of large state spaces in reinforcement learning, the problem is analogous: *how can we compactly represent a huge number of states with very few information?* In order to apply vector quantization to the reinforcement learning problem, we will provide first some definitions.

3.1 Definitions

Since our goal is to reduce the size of the reinforcement table, we have to find out a more compact representation of the states.[1] If we have K attributes describing the states, and each attribute a_i can have *values*(a_i) different values, where this number is usually big (in most cases infinite, since they are represented with real numbers), then the number of potential states can be computed as:

$$S = \prod_{i=1}^{K} values(a_i) \tag{3}$$

Since this can be a huge number, the goal is to reduce it to $N \ll S$ new states. These N states have to be able to approximately capture the same information as the S states; that is, all similar states in the previous representation, belong to the same new state in the new representation. The first definition takes this into account.

Definition 1. *A vector quantizer Q of dimension K and size N is a mapping from a vector (or a "point") in the K-dimensional Euclidean space, R^K, into a finite set C containing N output or reproduction points, called* code vectors*,* codewords*, or* codebook*. Thus,*

$$Q : R^K \longrightarrow C$$

where $C = (y_1, y_2, \ldots, y_N)$, $y_i \in R^K$.

[1] We will refer indistinctly to vectors and states.

Given C (computed by the generalized Lloyd algorithm, explained below), and a vector $x \in R^K$, $Q(x)$ assigns x to the closest state from C. In order to define the closeness of one vector x to a state in C, we need to define a measure of the *quantization error*, which needs a *distortion measure* (analog to the similarity metric of clustering techniques).

Definition 2. *A distortion measure d is an assignment of a nonnegative cost $d(x, q)$ associated with quantizing any input vector $x \in R^K$ with a reproduction vector $q = Q(x) \in C$.*

In digital communications, the most convenient and widely used measure of distortion between an input vector x and a quantizer vector $q = Q(x)$, is the squared error or squared Euclidean distance between two vectors defined by equation 4.

$$d(x, q) = \|x, q\|^2 = \sum_{i=1}^{K} (x[i] - q[i])^2 \tag{4}$$

However, sometimes differences in one attribute value are more important than in another. In those cases, the weighted squared error measure is more useful, because it allows a different emphasis to be given to different vector components, as in equation 5. In other cases, the values $x[i]$ and $q[i]$ are normalized by the range of values of the attribute. This is a special case of the equation 5 where weights would be computed as the inverse of the square of the range (maximum possible value minus minimum possible value).

$$d(x, q) = \sum_{i=1}^{K} w_i (x[i] - q[i])^2 \tag{5}$$

Once a distortion measure has been defined, we can define Q as in equation 6.

$$Q(x) = \arg \min_{y \in C} \{d(x, y)\} \tag{6}$$

In order to measure the average error produced by quantizing M training vectors x_j with Q, average distortion is defined as the expected distortion calculated among any input vector and the quantizer Q:

$$D = \frac{1}{M} \sum_{j=1}^{M} \min_{y \in C} d(x_j, y) \tag{7}$$

Finally, we define *partition* and *centroid*, concepts needed for presenting the Lloyd algorithm for computing C from M input vectors.

Definition 3. *A partition or cell $R_i \subseteq R^K$ is the set of input vectors (old states) associated to the same (new) state in the codebook C.*

Definition 4. *We define the* centroid*, cent(R), of any set* $R \subseteq R^K$ *as that vector* $y \in R^K$ *that minimizes the distortion between any point* x *in* R *and* y*:*

$$cent(R) = \{y \in R^K \mid E[d(x,y)] \leq E[d(x,y')], \forall x \in R, y' \in R^K\} \tag{8}$$

where $E[z]$ *is the expected value of* z*.*

A common formula to calculate each component i of the centroid of a partition is given by equation 9.

$$cent(R)[i] = \frac{1}{\|R\|} \sum_{j=1}^{\|R\|} x_j[i] \tag{9}$$

where $x_j \in R$, $x_j[i]$ is the value of component (attribute) i of vector x_j, and $\|R\|$ is the cardinality of R.

3.2 Generalized Lloyd Algorithm (GLA)

The generalized Lloyd algorithm is a clustering technique, extension of the scalar case [11]. It consists of a number of iterations, each one recomputing the set of more appropriate partitions of the input states (vectors), and their centroids. The algorithm is shown in table 2. It takes as input a set T of M input states, and generates as output the set C of N new states (*quantization levels*).

Generalized Lloyd algorithm (T, N)
1. Begin with an initial codebook C_1.
2. Repeat
(a) Given a codebook (set of clusters defined by their centroids) $C_m = \{y_i; i = 1, \ldots, N\}$, redistribute each vector (state) $x \in T$ into one of the clusters in C_m by selecting the one whose centroid is closer to x.
(b) Recompute the centroids for each cluster just created, using the centroid definition in equation 9 to obtain the new codebook C_{m+1}.
(c) If an empty cell (cluster) was generated in the previous step, an alternative code vector assignment is made (instead of the centroid computation).
(d) Compute the average distortion for C_{m+1}, D_{m+1}
Until the distortion has only changed by a small enough amount since last iteration.

Table2. The generalized Lloyd algorithm.

There are three design decisions to be made when using such technique:

Stopping criterion Usually, average distortion of codebook at cycle m, D_m, is computed and compared to a threshold θ $(0 \leq \theta \leq 1)$ as in equation 10.

$$(D_m - D_{m+1})/D_m < \theta \tag{10}$$

Empty cells One of the most used mechanisms consists of splitting other partitions, and reassigning the new partition to the empty one. All empty cells generated by the GLA are changed in each iteration by another cell. To define the new one, another non-empty cell with big average distortion y, is splitted in two:

$$y_1 = \{y[1] - \epsilon, \ldots, y[K] - \epsilon\}, \text{and}$$
$$y_2 = \{y[1] + \epsilon, \ldots, y[K] + \epsilon\}$$

Initial codebook generation We have used a version of the GLA as explained in table 3, that requires a partition split mechanism as the one described above inserted into the GLA in table 2.

GLA with Splitting (T)

1. Begin with an initial codebook C_1 with N (number of levels of the codebook) set to 1. The only level of the codebook is the centroid of the input.
2. Repeat
   (a) Set N to $N * 2$
   (b) Generate a new codebook C_{m+1} with N levels that includes the codebook C_m. The rest N undefined levels can be initialized to 0
   (c) Execute the GLA algorithm in table 2 with the splitting mechanism with parameters (T, N) over the codebook obtained in previous step

   Until N is the desired level

Table3. A version of the generalized Lloyd algorithm that solves the initial codebook and empty cell problems.

4 Application of *VQ* to *Q-learning.* VQQL

The use of vector quantization and the generalized Lloyd algorithm to solve the generalization problem in reinforcement learning algorithms requires two consecutive phases:

Learn the quantizer. Or to design the N-levels vector quantizer from input data obtained from the environment.

Learn the Q function. Once the vector quantizer is designed (we have clustered the environment in N different states), it is needed to learn the Q function, generating the Q table, that will be composed of N rows, and a column for each action (one could also use the same algorithm for quantizing actions).

We have two ways of unifying both phases:

Off-line mode. We could obtain the information required to learn the quantizer and the Q function, and, later, learn both.

On-line mode. We could obtain data to generate only the vector quantizer, and, later, the Q function is learned by the interaction of the agent with the environment, using the previously designed quantizer.

The advantages of the first one are that it allows to use the same information several times, and the quantizer and the Q table are learned with the same data. The second one allows the agent to use greedy strategies in order to increase the learning rate (exploration versus exploitation).

In both cases, the behavior of the agent, once the quantizer and the Q function are learned, is the same; a loop that:

- Receives the current state, s, from the environment.
- Obtains the quantization level, s', or state to which the current state belongs.
- Obtains the action, a, from the Q table with bigger Q value for s'.
- Executes action a.

5 The Robosoccer domain

In order to verify the usefulness of the vector quantization technique to solve the generalization problem in reinforcement learning algorithms, we have selected a robotic soccer domain that presents us all the problems that we have defined in previous sections. The RoboCup, and its Soccer Server Simulator, gives us the needed support [16].

The Soccer Server provides an environment to confront two teams of players (agents). Each agent perceives at any moment two types of information: visual and auditorial [16]. Visual information describes a player what it sees in the field. For example, an agent sees other agents, field marks such as the center of the field or the goals, and the ball. Auditorial information describes a player what it hears in the field. A player can hear messages from the referee, from its coach, or from other players. Any agent (player) can execute actions such as run (dash), turn (turn), send messages (say), kick the ball (kick), catch the ball (catch), etc.

One of the more basic skills a soccer player must have is ball interception. The importance of this skill comes from the dependency that other basic skills, such as kick or catch the ball, have with this one. Furthermore, ball interception is presented as one of the more difficult tasks to solve in the Robosoccer simulator, and it has been studied in depth by other authors [17]. In the case of Stone's work, neural networks were used to solve the ball interception problem posed as a supervised learning task.

The essential difficulties of this skill come from the visual limitations of the agent, as well as from the noise that the simulator includes in movements of objects. In order to intercept the ball, our agents parse the visual information that they receive from the simulator, and obtain the following information:[2]

[2] The Robosoccer simulator protocol version 4.21 has been used for training. In other versions of the simulator, other information could be obtained.

– Relative Distance from the ball to the player.
– Relative Direction from the ball to the player.
– Distance Change, gives an idea of how Distance is changing.
– Direction Change, gives an idea of how Direction is changing.

In order to intercept the ball, after knowing the values of these parameters, each player can execute several actions:

Turn changing the direction of the player according to a moment between -180 and 180 degrees.
Dash increasing the velocity of the player in the direction it is facing with a power between -30 and 100.

To reduce the number of possible actions that an agent can perform (generalization over actions problem), we have used macro-actions defined as follows. Macro-actions are composed of two consecutive actions: turn(T), and dash(D), resulting in turn-dash(T, D). We have selected $D = 100$, and T is computed according to $A + \Delta_A$, where A is the angle between the agent and the ball, and Δ_A can be: +45,+10,0,-10,-45. Therefore, we have reduced the set of actions to five actions.

6 Results

In this section, the results of using the VQQL model for learning the ball interception skill in the Robosoccer domain are shown. In order to test the performance of the Lloyd algorithm, we generated a training set of 94.852 states with the following iterative process, similar to the one used in [17]:

– The goalie starts at a distance of four meters in front of the center of the goal, facing directly away from the goal.
– The ball and the shooter are placed randomly at a distance between 15 and 25 from the defender.
– For each training example, the shooter kicks the ball towards the center of the goal with a maximum power (100), and an angle in the range $(-20, 20)$.
– The defender goal is to catch the ball. It waits until the ball is at a distance less or equal than 14, and starts to execute actions defined in section 5, while the goal is not in the catchable area [16]. Currently, we are only giving positive rewards. Therefore, if the ball is in the catchable area, the goalie tries to catch the ball, and if it succeeds, a positive reward is given to the last decision. If the goalie does not catch the ball, it can execute new actions. Finally, if the shooter goals, or the ball goes out of the field, it receives a reward of 0.

Then, we used the algorithm described in Section 3 with different number of quantization levels (new states). Figure 1 shows the evolution of the average distortion of the training sequence. The x-axis shows the logarithm of the

number of quantization levels, i.e. the number of different states what will be used afterwards by the reinforcement learning algorithm and the y-axis shows the average distortion obtained by GLA. The distortion measure used has been the quadratic error, as shown in equation 4. As it can be seen, when using 2^6 to 2^8 quantization levels, the distortion becomes practically 0.

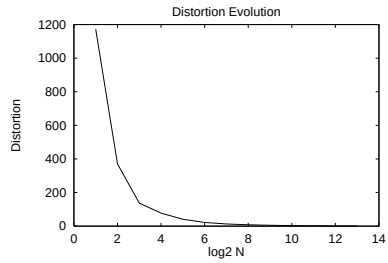


Figure1. Average distortion evolution depending on the number of states in the codebook.

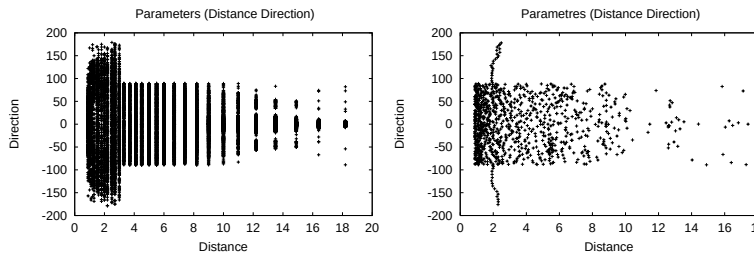
To solve the state generalization problem, a single mechanism could be used, such as a typical scalar quantization on each parameter. In this case, the average quantization error, following the error quadratic distortion measure defined in equation 4, could be calculated as follows. The Distance parameter range is usually in (0.9,17.3). Thus, if we allow 0.5 as the maximum quantization error, we need around 17 levels. Direction is in the (-179,179) range, so, if we allow a quantization error of 2, 90 levels will be needed. Distance Change parameter is usually in (-1.9,1), so we need close to 15 levels, allowing an error of 0.1, and Direction Change is usually in (-170,170), so we need 85 levels, allowing an error of 2. Then, following equation 3 we need $17*90*15*85 = 1,950,750$ states!!! This is a huge size for a reinforcement learning approach. Also, the average distortion that is obtained according to equation 4 is:

$$(\frac{0.5}{2})^2 + (\frac{2}{2})^2 + (\frac{0.1}{2})^2 + (\frac{2}{2})^2 = 2.7$$

given that the quantization error on each quantization is half of the maximum possible error. Instead, using the generalized Lloyd algorithm, with many less states, 2048, the average distortion goes under 2.0. So, it reduces both the number of states to be represented, and the average quantization error.

Why is this reduction possible on the quantization error? The answer is given by the statistical advantages that the vector quantization provides over the scalar quantization. These advantages can be seen in Figure 2. In Figure 2(a), only the pairs of Distance and Direction that appeared in the training vectors have been plotted. As we can see, only some regions of the bidimensional space have values, showing that not all combinations of the possible values of the Distance and Direction parameters exist in the training set of input states. Therefore, the

reinforcement tables do not have to consider all possible combinations of these two parameters. Precisely, this is what vector quantization does. Figure 2(b) shows the points considered by 1024 states quantization. As it can be seen, it only generates states that represent minimally the states in the training set. The fact that there are parts of the space that are not covered by the quantization is due to the importance of the other two factors not considered in the figure (change in distance and direction).



(a) (b)

Figure2. Distance and Direction parameters from (a) original data, and (b) a codebook obtained by the GLA.

In order to test what the best number of quantization levels is, we have varied that number, and learned a Q table per obtained quantizer. We measured successful performance as the percentage of kicks of a new set of 100 testing problems that go towards the goal, and are catched by the goalie. The results of these experiments are shown in Figure 3. In that figure the performance of the goalie is shown, depending of the size of the Q table. As a reference, a random goalie would only achieve a 20% of success, and a goalie with the most used heuristic of *always go towards the ball* achieves only a 25% of successful behavior. As it can be seen, Q table sizes less than 128 obtain a quasi-random behavior. From sizes of the Q table from 128 to 1024, the performance increases until the maximum performance obtained, which is close to 60%. From 4096 states and up, the performance decreases. That might be the effect of obtaining again a very large domain (huge number of state).

7 Related Work

Other models to solve the generalization problem in reinforcement learning use decision trees as in the G-learning algorithm [3], and kd-trees (similar to a decision tree) in the VRDP algorithm [14]. Another solution is Moore's PartiGame algorithm [15] or neural networks [9]. One advantage of vector quantization is that it allows to easily define control parameters for obtaining different behaviors of the reinforcement learning technique. The main two parameters that have to be defined are number of quantization levels, and average distortion (similarity

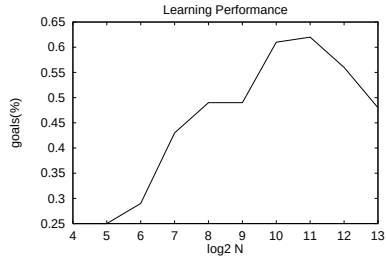


Figure3. Learning performance depending on the number of states in the codebook.

metric). Other approaches to this problem were proposed in [5] and [2] where bayesian networks are used. Similar ideas to our approach to Vector Quantization have been used by other researchers, as in [4] where Q-Learning is used as in this paper, using LVQ [8]. Again, it is easier for VQ to define its learning parameters than it is for neural networks based systems.

8 Conclusions and Future Work

In this paper, we have shown that the use of vector quantization for the generalization problem of reinforcement learning techniques provides a solution to how to partition a continuous environment into regions of states that can be considered the same for the purposes of learning and generating actions. It also solves the problem of knowing what granularity or placement of partitions is more appropriate.

However, this mechanism introduces a set of open questions that we expect to tackle next. As we explained above, the GL algorithm allows us to generate codebooks or sets of states of different sizes, each of them giving us different quantization errors. So, an important question is the relation between the number of quantization levels and the performance of the reinforcement learning algorithm. Another important issue relates to whether this technique can be applied not only to the state generalization problem, but also to actions generalization. We are also currently exploring the influence of providing negative rewards to the reinforcement learning technique. Finally, in the short term we intend to compare it against using decision trees and LVQ.

References

1. Jacky Baltes and Yuming Lin. Path-tracking control of non-holonomic car-like robot with reinforcement learning. In Manuela Veloso, editor, *Working notes of the IJCAI'99 Third International Workshop on Robocup*, pages 17–21, Stockholm, Sweden, July-August 1999. IJCAI Press.

2. Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1111, Montreal, Quebec, Canada, August 1995. Morgan Kaufmann.
3. David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.
4. C. Claussen, S. Gutta, and H. Wechsler. Reinforcement learning using funtional approximation for generalization and their application to cart centering and fractal compression. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1362–1367, Stockholm, Sweden, August 1999.
5. Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *Proceedings of the American Association of Artificial Intelligence (AAAI-97)*. AAAI Press, 1997.
6. Marco Dorigo. Message-based bucket brigade: An algorithm for the appointment of credit problem. In Yves Kodratoff, editor, *Machine Learning. European Workshop on Machine Learning*, LNAI 482, pages 235–244. Springer-Verlag, 1991.
7. Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
8. T. Kohonen. The self-organizing map. In *Proceedings of IEEE*, volume 2, pages 1464–1480, 1990.
9. Long-Ji Lin. Scaling-up reinforcement learning for robot control. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 182–189, Amherst, MA, June 1993. Morgan Kaufman.
10. Yoseph Linde, André Buzo, and Robet M. Gray. An algorithm for vector quantizer design. In *IEEE Transactions on Communications, Vol1. Com-28, Nº 1*, pages 84–95, 1980.
11. S. P. Lloyd. Least squares quantization in pcm. In *IEEE Transactions on Information Theory*, number 28 in IT, pages 127–135, March 1982.
12. S. Mahavedan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.
13. Tom M. Mitchell and Sebastian B. Thrun. Explanation based learning: A comparison of symbolic and neural network approaches. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 197–204, University of Massachusetts, Amherts, MA, USA, 1993. Morgan Kaufmann.
14. Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. *Proceedings in Eighth International Machine Learning Workshop*, 1991.
15. Andrew W. Moore. The party-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, pages 711–718, San Mateo, CA, 1994. Morgan Kaufmann.
16. Itsuki Noda. *Soccer Server Manual*, version 4.02 edition, January 1999.
17. Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Berlin, 1999. Springer Verlag.
18. C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, May 1992.