Modular Value Iteration through Regional Decomposition

Linus Gisslen, Mark Ring, Matthew Luciw, and Jürgen Schmidhuber

IDSIA
Manno-Lugano, 6928, Switzerland
{linus,mark,matthew,juergen}@idsia.com

Abstract. Future AGIs will need to solve large reinforcement-learning problems involving complex reward functions having multiple reward sources. One way to make progress on such problems is to decompose them into smaller regions that can be solved efficiently. We introduce a novel modular version of Least Squares Policy Iteration (LSPI), called M-LSPI, which 1. breaks up Markov decision problems (MDPs) into a set of mutually exclusive regions; 2. iteratively solves each region by a single matrix inversion and then combines the solutions by value iteration. The resulting algorithm leverages regional decomposition to efficiently solve the MDP. As the number of states increases, on both structured and unstructured MDPs, M-LSPI yields substantial improvements over traditional algorithms in terms of time to convergence to the value function of the optimal policy, especially as the discount factor approaches one.

1 Introduction

Reinforcement learning is one of the most promising approaches for achieving artificial general intelligence, yet current methods tend to scale poorly to large problems. Effective methods exist for finding the optimal policy and value function of a Markov decision process (MDP; [1]) given a model of transition probabilities and expected rewards [1–7], but these methods run into computational bottlenecks as the number of states and their degree of connectivity increases or when the reward function is complex (*i.e.,* when there are many sources of reward). Yet it is quite common for real-world, applied MDPs to have complex reward functions with positive and negative rewards distributed throughout the task space. In robotics, for example, all near-collision states might generate negative reward.

In this work we assume access to an accurate model of the MDP and focus on new methods for solving the MDP from the model. Any existing approach for building the model from interaction with the environment can be used, *e.g.*, that of Gisslen *et al.* (2011) [8]. Value Iteration (VI; [2, 9]) is the most basic MDP solver. It does state-value backups [1] on all states, equally often, independent of MDP structure. VI handles complex reward functions well, but scales poorly as the discount factor approaches one and as the degree of connectivity in the MDP increases. Thus, there have been previous attempts to improve its efficiency. Prioritized Sweeping (PS; [4]), for example, uses a priority queue to

J. Bach, B. Goertzel, and M. Iklé (Eds.): AGI 2012, LNAI 7716, pp. 69–78, 2012.

order partial state-value backups prioritized on (generally) the Bellman error. PS gets a better *approximate* result than VI with less data and less time, but final convergence (to the same result as VI) often takes just as long or longer due to queue maintenance. Dai *et al.* [5] tried to avoid the overhead of priority queue maintenance, suggesting heuristics to prioritize states, such as reachability of a state from the goal when following the current policy. Wingate [6] organized the bookkeeping overhead, prioritizing via an "information frontier" starting with the "most informative state" (*e.g.*, the absorbing goal), then propagated the frontier throughout the system. These approaches are very effective for single-goal situations, but do not efficiently handle complex reward functions. Least Squares Policy Iteration (LSPI; [7]), based on policy iteration (PI; [3]), finds the optimal linear-function approximation of an MDP's value function via a series of matrix inversions.[1] Unfortunately, matrix inversions can be very computationally demanding, scaling nearly cubically with the number of state-action pairs, but LSPI is appealing because it has no adjustable learning rate, delivers an optimal solution for every reward function, and its performance is independent of the MDP's connectivity and discount factor.

This paper introduces *Modular LSPI* (M-LSPI) which combines VI and LSPI to achieve the advantages of both: good scaling behavior in the number of states, discount factor, *and* connectivity. M-LSPI partitions the states into mutually exclusive regions (we explore several methods for doing so in Sec. 4). Separate modules perform LSPI within each region independently, and the value information between modules is then combined using a method that resembles VI. If the MDP exhibits regional structure, then the partitioning can lead to a considerable improvement in performance. The results indicate that M-LSPI is especially good on MDPs with *high connectivity* even when the assignment of states to regions is purely random. Experiments show benefits over previous methods on a variety of MDPs, including autonomous systems and simulated humanoid motor tasks, indicating that M-LSPI can be effective in any environment where a world-model can obtained. Finally, the modular approach appears promising for continual learning [10], as it provides a method for adding new regions without the need to recalculate the entire MDP.

2 Markov Decision Processes with Many States

Markov Decision Processes. A *Markov decision process* can be expressed as a 5-tuple (S,A,P,R,γ), where $S = \{s_1, s_2, \ldots, s_n\}$ is a finite set of states, $A = \{a_1, a_2, \ldots, a_m\}$ is a finite set of actions, $P(s, a, s')$ is a *Markovian transition model* quantifying the probability of ending up in state s' when taking an action a in state s, $R(s, a, s')$ is the expected reward when taking action a from state s and ending up in state s', and $\gamma \in [0, 1)$ is the discount factor, exponentially decreasing the impact of future rewards on the current action choices.

[1] Typically, LSPI operates on features; however, for an exact solution of an MDP, a single dimension of the feature vector can be dedicated to each state-action pair, which is how we use it here.

A learning agent increases its receipt of reward by improving its *policy* π, which specifies what action the agent takes in each state. The agent *solves* an MDP by finding the *optimal* policy π^*, which maximizes its receipt of reward. $Q^\pi(s,a)$, called the "state-action value," represents the expected sum of discounted future rewards the agent will receive for taking action a in state s and following policy π thereafter. These values for all (s,a) pairs are related mathematically by the Bellman equation:

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} P(s,a,s') Q^\pi(s', \pi(s')). \tag{1}$$

where $\mathcal{R}(s,a) = \sum_{s'} P(s,a,s') R(s,a,s')$. Thus, the optimal policy always chooses the action with the highest state-action value: $\pi^\star(a) = \max_a Q^{\pi^\star}(s,a)$.

Iterative MDP Solution Methods. The two main iterative methods for solving an MDP from P and $\mathcal{R}$ are value iteration (VI, described next) and policy iteration (PI, described in Sec. 3 in the context of LSPI).[2]

VI solves the system of Bellman equations described in Equation 1 by iteratively doing updates (called "backups") in the form:

$$Q(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V(s') \tag{2}$$

for all states and actions, where the *state value* $V(s) = \max_a Q(s,a)$. Each backup propagates the maximum reward information to each state from its successors. The agent's best policy, given the backups made at any point, is to take the action in each state with the highest state-action value. One loop through all state-action pairs yields the best policy with a one-step look-ahead; after two loops, the maximum two-step policy is obtained. The iteration process continues until *convergence*: when the difference between all state values over two subsequent loops is less than a certain threshold ϵ.

VI has a time complexity of $O(|A||S|^2)$ per iteration, with the number of iterations required for convergence growing with $1/(1-\gamma)$ [12], which makes VI troublesome as the discount factor γ approaches one. In contrast, PI requires $O(|S|^3)$ operations per iteration [3] and convergence does not seem to depend on the discount factor.

Reward Horizon. As the number of states in an MDP increases, the discount factor also needs to increase towards one if all the rewards are to impact all of the states. For example, if $\gamma = 0.9$ a reward value of 1.0 has an impact (a discounted value) of only 10^{-23} on states 500 steps away; when $\gamma = 0.99$, that value is .007, but is 0.6 when $\gamma = 0.999$. Therefore in even moderate-sized MDPs (>1000 states) the discount factor must be higher than $\gamma = 0.9$ if the rewards in one region of the MDP are to have a significant impact on the agent's choices in distant states. Thus, planning very far into the future makes VI computationally

[2] Linear programming [11] can represent any MDP and solve it in polynomial time, but in practice the polynomials are often too large to be solved in reasonable time [12], while iterative methods are feasible.

costly: convergence for $\gamma = 0.99$ takes ten times longer than for $\gamma = 0.9$. And while PI handles the discount factor well, it scales poorly with the number of states. Therefore, both approaches seem prohibitive for large MDPs. Yet it is possible to get the best of both worlds in some cases by doing PI locally and VI globally, particularly if the MDP has local structure that allows regional decomposition.

Regional Decomposition. Decomposing the MDP into smaller regions has been used before to increase solution speed, whether it is to achieve sub-optimal policies quickly [13] or to organize backups to reach optimal values faster [6]. Generally performance improves when the partitioning reflects the structure of the MDP. Prior knowledge, such as the positions of the states in a 2D grid-world, can allow decomposition by hand (*e.g.,* into a coarser-resolution 2D grid-world). For the general case, when such prior knowledge is not available, there are regional decomposition algorithms such as the Chinese Whispers method [14], but these can be computationally demanding and their performance is not guaranteed. The opposite extreme is to assign states to regions randomly, which is computationally trivial yet still occasionally beneficial simply because smaller regions can be solved faster.

3 Modular Least-Square Policy Iteration

Algorithm 1 shows M-LSPI as pseudo-code. Prior to execution, the MDP is assumed to have been decomposed into non-overlapping regions of states (see Sec. 4).

The algorithm's main loop, repeated until convergence, combines a VI step with a PI step for each region. The VI step collects state values from the neighboring regions (lines **5**–**8**). The PI step calls LSPI-Model (the model-based version of LSPI, line **10**) which treats the region as though it were a complete MDP and finds its optimal policy (given the current information). The key of M-LSPI is the way the VI and PI steps are combined. The trick here is to sum up the state values from all the successor states in the neighboring regions and then *to add these values into the reward vector* $\mathbf{b}$ for the region to be solved. When LSPI solves the region using this modified reward vector, that integrates the VI and PI steps, yielding exact state-action values for the region using all the latest information from the local region and its neighbors.

Looking at the MDP as a graph, LSPI is guaranteed to find the correct values [7] for all the edges within the local region (given current information), while the value-iteration step updates the edges connecting the region to its neighbors. Since VI itself is guaranteed to converge [9], we conjecture that M-LSPI will iteratively converge to the optimal value function. This conjecture is supported by the empirical results in Sec 4.

M-LSPI scales well, handling MDPs with a large number of states and γ close to one. It avoids the cubic scaling complexity of global PI by keeping down the number of states within each region, and avoids VI's explosive reaction to γ by using LSPI to get exact results in each region. LSPI converges reliably and quickly to the optimal policy for small MDPs and does not require tuning.

Algorithm 1. M-LSPI($\mathbf{M}, \mathcal{P}, \mathcal{R}, Q, \gamma$)

```
// M_i : Region assignments
// P_{i,j} : Regional transition models (s, a, s') for s ∈ M_i and s' ∈ M_j
// R_i : Regional reward vectors for all (s, a), where s ∈ M_i
// Q : Initial state-action values
// γ : Discount factor
repeat
    Q' ← Q                                          // save latest state-action values
    for each region i in M do
        b ← R_i                                     // local reward vector
        for each region j in NeighborsOf (i) do
            for each (s, a, s') ∈ P_{i,j} do
                b_{s,a} ← b_{s,a} + γ P_{i,j}(s, a, s') · max_{a'∈A} Q_{s',a'}
            end
        end
        Q_i ← LSPI-Model(P_{i,i}, b, Q_i, γ)        // update Q values for s ∈ M_i
    end
until (max_{s,a} |Q_{s,a} − Q'_{s,a}| < ε)          // check convergence globally
return Q
```

Model-Based LSPI. In LSPI, the state-action values Q are approximated as a linear combination of *basis functions* $\mathbf{\Phi}$:

$$\hat{Q}(s, a, \omega) = \mathbf{\Phi}\omega.$$

Writing the Bellman equation (1) in matrix form

$$Q^\pi = \mathcal{R} + \gamma \mathbf{P\Pi} Q^\pi,$$

where $\mathbf{\Pi}$ is a matrix representation of policy π, and replacing Q^π with the approximation $\hat{Q} = \mathbf{\Phi}\omega$ yields

$$\mathbf{\Phi}\omega \approx \mathcal{R} + \gamma \mathbf{P}^\pi \mathbf{\Phi}\omega.$$

where $\mathbf{P}^\pi = \mathbf{P\Pi}$. Parameter values ω are calculated with a single matrix inversion:

$$\omega = \mathbf{A}^{-1} b, \tag{3}$$

where

$$\mathbf{A} = \mathbf{\Phi}^\intercal (\mathbf{\Phi} - \gamma \mathbf{P}^\pi \mathbf{\Phi})$$
$$b = \mathbf{\Phi}^\intercal \mathcal{R}.$$

In our model-based case, each basis function is dedicated to a single state-action pair, and thus $\mathbf{\Phi}$ is an $(|S||A| \times |S||A|)$ identity matrix, and ω is identical to the estimated state-action values (shown in Algorithm 1 as Q). However, for completeness we include the full derivation here.

Regional Model-Based LSPI. Given a state space that has been decomposed into non-overlapping regions of states, M-LSPI decomposes the above equations into subsets corresponding to those regions and uses LSPI to solve each subset separately (line **10** in Algorithm 1):

$$\begin{aligned}\omega_i &= \mathbf{A}_i^{-1} b_i \\ \mathbf{A}_i &= \mathbf{\Phi}_i^\intercal (\mathbf{\Phi}_i - \gamma \mathbf{P}_{i,i}^\pi \mathbf{\Phi}_i) \\ b_i &= \mathbf{\Phi}_i^\intercal V_i\end{aligned}$$

where $\mathbf{\Phi}_i$ is the set of basis functions for the states in region i, and $\mathbf{P}_{i,i}^\pi$ contains only information about transition probabilities within region i. V_i combines the reward values for region i with the discounted state-action values from all successor states in the neighboring regions, following Bellman's optimality equation rewritten (in matrix form) in terms of regions rather than states:

$$V_i = \mathcal{R}_i + \gamma \sum_{j \neq i} \mathbf{P}_{i,j}^\pi V_j \tag{4}$$

where R_i is the reward vector for region i (Algorithm 1, line **4**), $\mathbf{P}_{i,j}^\pi$ is the transition probability matrix between the regions i and j, and V_j is the vector of state values for region i's successor states in region j. As usual with VI, the state values V_j are computed as the maximum state-action value for each state (Algorithm 1, line **7**).

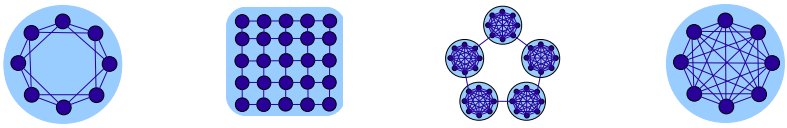
In the simplest case, where there is only one region, the second term on the right hand side of Equation 4 is 0, and the first term ($\mathcal{R}_i$) is the reward vector $\mathcal{R}$, which thus becomes the b vector used to solve Equation (3).

As ω_i depends on information from all the other $\omega_{j \neq i}$ it is now clear that the global solution must be reached through successive iterations, similar to VI. When the values have converged, the collection of regional weights ω_i taken together represent the full set of weights ω.

4 Experimental Results

This section compares M-LSPI with other methods for discovering the optimal policy π^* and calculating its value function over a variety of MDPs. All comparisons are in terms of wall-clock time (seconds) until convergence, $\epsilon = 10^{-6}$. In our case, this time includes the overhead of building regions (but this cost would not be incurred again if the calculation were repeated with different reward vectors).

Experimental Data. Comparisons were done with four classes of MDP (see Figure 1) and two sets of real data (named "RobotArm" and "Autonomous-system," see below). All MDPs have fixed (but randomly generated) transition probabilities and reward values; *i.e.,* when the MDP is created, each state-action pair is assigned a fixed reward value drawn from a uniform distribution in $(-1, 1)$.
1D-MDP (Fig 1, a): A one-dimensional Markov ring with sparse, local connectivity, having N states and five possible actions. One action has no effect; each



(a) **1D-MDP** (b) **2D-MDP** (c) **Cx-MDP** (d) **FC-MDP**

Fig. 1. Illustration of connectivity in different MDPs: (a) one- and (b) two-dimensional MDPs (**1D-MDP** and **2D-MDP**) with connections only to closest neighbors, (c) clusters of fully connected states with sparse connections to neighboring clusters (**Cx-MDP**) where x denotes the cluster size, and (d) fully connected MDP (**FC-MDP**).

of the other four move the agent either clockwise or counterclockwise, either one step or two. There is a 10% random chance one of the actions not chosen is taken instead. **2D-MDP** (Fig 1, b): A 2D (toroidal) version of the previous, also having five actions (up, down, left, right and stay), again with a 10% chance a different action is taken. **Cx-MDP** (Fig 1, c): Clusters of x fully connected states with sparse connections between the clusters. This is an MDP version of *modular small-world graphs* [15]. **FC-MDP** (Fig 1, d): A fully connected MDP where each state is connected to each other state. Transition probabilities for Cx and FC are assigned randomly, drawn uniformly from (0,1) and then normalized such that $\sum_{s'} P(s, a, s') = 1$ for each (s, a) pair. **RobotArm.** A dataset generated with the iCub robot simulator [16]. Each state corresponds to the configuration of three arm joints (two shoulder and one elbow). A *precision* value controls the number of states; higher precision means finer quantization and more states. There are six deterministic actions, each resulting in a movement to the next closest configuration for each joint in both directions. **Autonomous-system**. Datasets from the Stanford Large Networks dataset collection.[3] These datasets represent communication graphs of routers comprising the internet (known as "autonomous systems") and are available in a variety of sizes. We assigned random reward values to the edges, drawn uniformly from (-1,1), and the transition probabilities are split evenly. In this special case there are no actions, so there is no policy improvement, and only the value function is solved.

Methods Compared. The featured competitors are value iteration (VI; [2]), prioritized sweeping (PS; [4]), and LSPI itself on the model [7]. We do not compare experimentally to Wingate's value iteration with regional prioritization [6], nor the prioritization method of Dai [5], since we study the general case of arbitrary-valued reward functions, for which they are poorly suited.

Figure 2 shows a comparison of the three methods on two classes of MDPs with a small number of states. It is clear that only VI merits a large-scale comparison to M-LSPI.

Experiment Setup. Each model has an adjustable size (number of states). For every model size, we ran five tests and report average time to convergence.

[3] `snap.stanford.edu/data/`

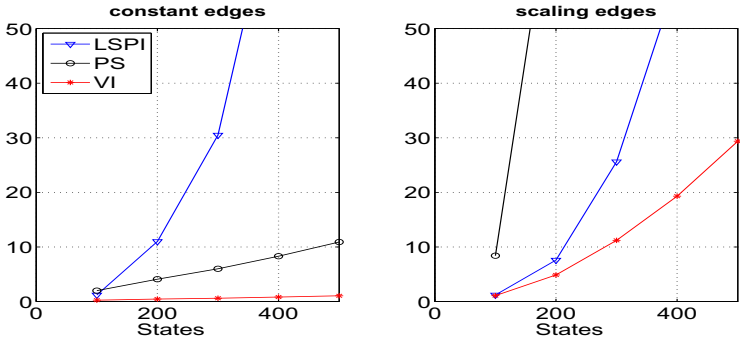


Fig. 2. Comparison of LSPI, PS, and VI on MDPs with complex reward functions (nonzero reward for each state-action pair), with $\gamma = 0.99$. Left: Time required to find the optimal policy on a 1D-MDP (see text) where the number of connections per state is constant. LSPI's near-cubic complexity leads to poor performance. VI does quite well with complex reward functions and low-connectivity. PS does not deal well with complex reward functions since the priority queue introduces needless overhead. Right: Results on FC-MDPs (see text) where the number of connections per state scales with the number of states. Again, LSPI scales very poorly, and high connectivity leads to extremely poor performance of PS. VI is also the best of the three in this type of MDP.

Each test had a unique set of randomly assigned reward values and transition probabilities, and we compared all methods using the same set of tests. We used two values of γ: 0.99 and 0.999, and convergence criteria was $\epsilon = 10^{-6}$ for all methods, which in all cases guaranteed discovery of the optimal policy.

Region Building. M-LSPI used several different region-building techniques for the tests. For the random MDPs, states were assigned to regions randomly, with the region size fixed at 30. We also assigned states randomly for the Autonomous-system data, which has high (but not full) connectivity. We used the Chinese Whispers (CW) technique for the modular MDPs (CX-100). CW breaks up regions based on bottlenecks, leading states with high mutual connectivity to belong to the same regions. In this particular case region sizes were typically 100 states. This is an excellent situation for M-LSPI, as LSPI can handle dense connectivity within the regions well (in a small number of iterations). However, VI will struggle when the connectivity is dense. For the 1D, 2D, and RobotArm MDPs, there are no bottlenecks, so we use a simple variant of CW where: (1) initially each state is its own region; and (2) until all states are tagged, a random untagged state is selected, tagged, and a few of its untagged connecting states are merged into a single region (unless it would exceed 30 states).

Results. Graphic comparisons of M-LSPI and VI are shown in Fig. 3. M-LSPI seems to scale better than VI except for (a) the 1D and 2D MDPs when $\gamma = 0.99$ and (b) the RobotArm data (which is also a type of structured MDP). VI is ideal for the RobotArm data where the MDP is deterministic with low connectivity. When γ is increased, M-LSPI scales better than VI in all cases, since it solves each region with LSPI. Possibly due its modular structure, M-LSPI excels on modular

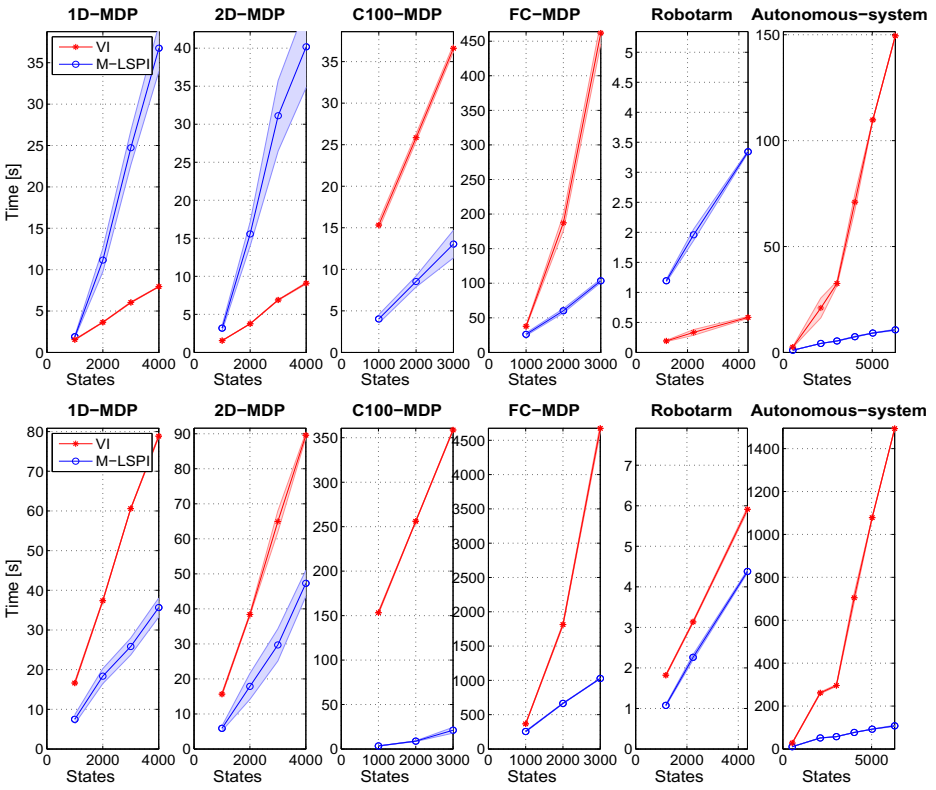


Fig. 3. Performance comparison of M-LSPI to value iteration on synthetic and real-data-based MDPs (described in text). For the upper panels, discount factor $\gamma = 0.99$, for the lower panels $\gamma = 0.999$.

Cx-MDPs, especially those with lower overall connectivity. It also does exceptionally well with fully-connected random MDPs and the autonomous-systems data, scaling far better than any alternate method we know.

Note that all results shown above *include* the cost of building the regions. In the RobotArm experiment, the region building time for {1176, 2238, 4356} states on average is {0.46, 1.2, 1.7} seconds. This can be done once, and the same regions can be used for multiple reward vectors. If the regions are already built, M-LSPI will be even faster than shown.

5 Conclusions

M-LSPI is a novel, modular, model-based MDP solver for large MDPs with complex reward functions. It efficiently yet optimally solves large MDPs, excelling on those with high connectivity or modular small-world connectivity. While approximate MDP solvers can sacrifice accuracy for speed, our method and others [5, 6]

have shown this sacrifice is not necessary if the structure of the solver matches the structure of the world. However, to our knowledge ours is the only method that can take advantage of such structure when the reward function is complex, which it will be for general-purpose, real-world intelligent agents, arising from multiple, anisotropic pain and pleasure signals.

Acknowledgments. The authors would like to thank Vincent Graziano for helpful discussions. This work was funded through the 7th framework program of the EU under grants #228844 (NanoBioTouch project), #231722 (IM-Clever project), and #270247 (NeuralDynamics project).

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. Cambridge Univ. Press (1998)
2. Bellman, R.: Dynamic Prog. Princeton University Press, Princeton (1957)
3. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
4. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning 13, 103–130 (1993)
5. Dai, P., Hansen, E.A.: Prioritizing bellman backups without a priority queue. In: ICAPS, pp. 113–119 (2007)
6. Wingate, D., Seppi, K.D.: Prioritization methods for accelerating mdp solvers. Journal of Machine Learning Research 6, 851–881 (2005)
7. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. The Journal of Machine Learning Research 4, 1107–1149 (2003)
8. Gisslén, L., Luciw, M., Graziano, V., Schmidhuber, J.: Sequential Constant Size Compressors for Reinforcement Learning. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) AGI 2011. LNCS, vol. 6830, pp. 31–40. Springer, Heidelberg (2011)
9. Bersekas, D.P.: Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall, Englewood Cliffs (1987)
10. Ring, M.B.: Continual learning in reinforcement environments. PhD thesis, University of Texas at Austin (1994)
11. D'Epenoux, F.: A probabilistic production and inventory problem. Management Science 10, 98–108 (1993)
12. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, UAI 1995, pp. 394–402. Morgan Kauffman, San Francisco (1995)
13. Kaelbling, L.P.: Hierarchical learning in stochastic domains: Preliminary results. In: Proceedings of the Tenth International Conference on Machine Learning, pp. 167–173. Citeseer (1993)
14. Biemann, C.: Chinese whispers. In: Workshop on TextGraphs, at HLT-NAACL, pp. 73–80. Association for Computational Linguistics (2006)
15. Bullmore, E., Sporns, O.: Complex brain networks: graph theoretical analysis of structural and functional systems. Nature Reviews Neuroscience 10(3), 186–198 (2009)
16. Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., Nori, F.: An open-source simulator for cognitive robotics research: The prototype of the icub humanoid robot simulator (2008)