Graph partitioning techniques for Markov Decision Processes decomposition

E0022

Abstract. Recently, several authors have proposed serial decomposition techniques for solving approximately or exactly large Markov Decision Processes. Many of these techniques rely on a clever partitioning of the state space in roughly independent parts. The different parts only communicate through relatively few communicating states. The efficiency of these decomposition methods clearly depends on the size of the set of communicating states : the smaller, the better. However, the task of finding such a decomposition with few communicating states is often (if not always) left to the user. In this paper, we present automated decomposition techniques for MDPs. These techniques are based on methods which have been developed for graph partitioning problems.

1 INTRODUCTION

Markov Decision Processes (MDPs) have become a standard model for decision-theoretic planning. However, traditional dynamic programming solution methods (e.g. [15]) do not scale to the size of problems that can be compactly represented by the factored representations traditionally used in AI. Therefore several methods of abstraction/agregation/decomposition for large problems have been proposed in the AI community (see [3] for an overview).

In this paper we focus on decomposition techniques, that allow a more efficient solving by splitting the problem into roughly independent subproblems, communicating through a small set of communicating states. Several authors [6, 5, 12] have proposed such techniques. They are based on Kernel decomposition [10], that [6] adapted to MDP decomposition. The authors were interested in, given a "good" decomposition, how to compute efficiently exact or approximate solutions to the original MDP. What we propose in this paper are methods and algorithms for finding such a good decomposition. Our work is in this respect complementary to theirs. To our knowledge not much effort has been devoted to this task, yet.

In the next two sections we recall the MDP framework and the Kernel decomposition technique for MDPs, as well as the basic features of the MDP solving methods exploiting this technique. Then in Section 4 we will expose the methods and algorithms that we suggest for computing automatically these decompositions. The methods are based on techniques which have been developed in the scientific computing community, for partitioning large graphs. These methods [8] are based either on *spectral decomposition* [13] or on local search [9].

2 MARKOV DECISION PROCESSES

The standard MDP model [15] is defined by : i) A set of stages (possibly infinite) in which decisions are taken. ii) For each stage t, a finite state space, S_t. iii) Sets $A_{s,t}$ (finite) of available actions in state s at stage t. iv) Rewards $r_t(s,a,s')$ that are obtained after a has been applied in state s and resulted in s'. v) Probability distributions $p_t(s,a,s')$ describing the uncertainty about the possible successor states (in S_{t+1}) of $s \in S_t$ when $a \in A_{s,t}$ is applied. A *policy* δ is an application from $\cup_t S_t$ to $\cup_{s\in S_t} A_{s,t}$ assigning an action to each possible state of the world in stage t. In the infinite horizon case or in the *stationary* finite horizon case, the parameter t has no influence on the decision problem, and thus, can be dropped. Then, the MDP is fully determined by $< S, A, p, r >$. For ease of exposition, we will restrict our study to stationary problems in this paper, but the techniques we propose could be extended to non stationary problems without much difficulty.

A policy δ, applied in an initial state s_0, defines a *Markov chain* that describes the sequence of states occupied by the system. The *discounted value of a policy* in a given state is the expected sum of the rewards gained along the possible trajectories :

$$v(s_0, \delta) = E(\sum_{t=0}^{n} \gamma^t \cdot r(s_t, \delta(s_t), s_{t+1})) \tag{1}$$

where $0 < \gamma < 1$ is the discounting factor and $0 \leq n \leq +\infty$ is the horizon.

Solving a MDP amounts to finding a policy δ^* maximizing $v(s,\cdot), \forall s$. The *dynamic programming* methods are based on the decomposition of the sequential decision problem into one-stage decision problems, by making use of the Bellman's equations [1].

In the finite horizon case, an optimal policy for a MDP is obtained as the solution of the following system of equations :

$$V^t(s) = \max_{a\in A_{s,t}} \{ \sum_{s'\in S_{t+1}} p(s,a,s') \cdot (r(s,a,s') + \gamma \cdot V^{t+1}(s'))\} \tag{2}$$

and $V^{n+1}(s) = 0, \forall s$.

Optimal policies can be computed by *backwards induction*, i.e. solving the above equations in decreasing order of t. In the discounted infinite horizon case, optimal policies (which, by the way, are stationary) can be obtained as fixed points of equation 2, superscripts $t, t+1$ being dropped.

Methods such as the *value iteration* algorithm [2] can be used to compute optimal policies. This algorithm consists in, starting from an arbitrary initial value function V_0, applying iteratively the formula on the right-hand side of (2) to V, until convergence is obtained (to V^*). From the optimal value function V^*, an optimal policy δ^* can be obtained *greedily*[1]. Other algorithms have been designed to solve infinite horizon MDPs, such as *policy iteration* [15]. This algorithm operates on policies, by alternating Evaluation and Improvement phases. In the Evaluation phase, policy δ is fixed and iterations of the right-hand side of (2) are performed until convergence to V_δ. Once convergence is obtained, δ is modified in the Improvement phase: the new policy δ' is greedy with respect to V_δ. The algorithm stops when no modification of the policy results from two successive Evaluation/Iteration phases. *Modified policy iteration* works as *policy iteration* but iterations within the Evaluation phase are stopped before convergence.

3 DECOMPOSITION TECHNIQUES

In AI, planning problems can often be modeled as MDPs. However, traditional dynamic programming algorithms require time polynomial in the size of the state and action spaces *explicitly enumerated.* Unfortunately, these spaces are in general too large to be enumerated, and are rather described compactly in structured languages. This is the reason why much effort in the Decision Theoretic Planning community has been devoted to the search for feasible computational methods for solving large (multidimensional) spaces MDPs. Namely, three families can be inventored :

- The *state aggregation methods* group states in subsets sharing the same features, thus reducing the size of the MDP [7]. In the same family, actions are sometimes aggregated in *macro-actions* [14].

- The *decomposition methods* [6, 12, 5] aim at decreasing the complexity of the MDP by splitting the original problem into smaller subproblems solved independently. The elementary solutions are then combined in order to provide an (approximately) optimal solution to the global problem.

- The *multi-agent reinforcement learning methods* [11] combine RL [16] with multi-agent methods, used as a means of decomposing the initial problem.

In this paper, we focus on decomposition methods, based on a *partial decoupling* of the state space, initiated by [6] and then improved by [5] and [12].

3.1 Star topologies of a MDP

Let $< S, A, p, r >$ be a MDP. Decomposition methods consist in taking advantage of a given partition $\Pi = \{S_i\}_{i=1..n}$ of the state space S in order to solve the original MDP more efficiently than in the direct way. The works of [6, 12, 5] use a *star topology* of the state space based on a partition. This decomposition relies on the two following definitions:

Definition 1 *Periphery.*
$Per : \Pi \rightarrow 2^S$ *associates to each element S_i of the partition its periphery, defined as the states out of S_i immediately reachable from S_i by an action $a \in A$.*

$Per(S_i) = \{s' \notin S_i, \exists (s,a) \in S_i \times A, p(s,a,s') > 0\}$.

[1] Through the formula $\delta^*(s) = argmax_a \sum_{s' \in S} p(s,a,s') \cdot (r(s,a,s') + \gamma \cdot V^*(s')))$.

Then, the star topology decomposition of S induced by Π is defined as:

Definition 2 *Star topology decomposition* Π^*.
$\Pi^* = \{K_i\}_{i=1..n} \cup U$, *is defined as :*
$U = \cup_{i=1..n} Per(S_i)$ *and* $K_i = S_i \cap \bar{U}, \forall i$.

These definitions constitute a basis for the MDP solving techniques that we briefly review in the next paragraph.

3.2 Solving a decomposed MDP

Exact methods for solving MDPs under star-topology form are essentially iterative. Basically, the central component U of the star topology is used in order to make the local processes over the S_i independent.

Definition 3 *Local MDPs.*
Let $MDP =< S, A, p, r >$. Let $\Pi = \{S_i\}_{i=1..n}$ be a partition of the state space, and $U = \cup Per(S_i)$. Let λ, a $|U|$-dimensioned vector, be an estimate of the global value function V on subspace U. Then for each subspace S_i we define a subproblem $MDP_\lambda^i =< S_i \cup Per(S_i), A, p_i', r_i' >$ where p_i' and r_i' are such that :

- $\forall (s,a) \in S_i \times A, \forall s' \in S, p_i'(s,a,s') = p(s,a,s')$, *and* $\forall (s,a) \in Per(S_i) \times A, p_i'(s,a,s) = 1$,

- $\forall (s,s',a) \in S_i^2 \times A, r_i'(s,a,s') = r(s,a,s')$, $\forall (s,a,s') \in S_i \times A \times Per(S_i), r_i'(s,a,s') = \lambda(s')$, $\forall (s,a,s') \in Per(S_i) \times A \times S_i, r_i'(s,a,s') = 0$.

The following proposition holds (see e.g. [6]):

Proposition 1 *If $\lambda(s) = V^*(s), \forall s \in U$ (where V^* is the optimal value function for the global MDP), then the policy $\pi = \cup \pi_{V^*}^i$ obtained by gluing together the solutions of the component $MDP_\lambda^i s$ is itself optimal.*

Of course, since the optimal value function V^* is what we are trying to compute, it is of no use in order to initialize λ. So, the algorithms proposed in [6, 12, 5] are all iterative. They initialize λ arbitrarily (or heuristically), then solve the MDP_λ^i and use the solutions in order to update λ.

In [6], it is proposed to use linear programming in order to solve the component MDP_λ^i. [12] notes that just any scheme alternating improvement of the local policies and of λ would lead to an optimal policy, as soon as no region starves (such schemes are implementations of *asynchronous dynamic programming* [2]). So, *value iteration*, *policy iteration*, even if not led to full convergence, are allowed as local improvement steps. [12] and [5] suggest to use the star topology in order to transform the initial MDP into an *abstract MDP* which state space is U, and actions are local policies (also called *macro actions*) over the S_i. The transition and reward functions of the new MDP are derived in a coherent way from the ones of the initial MDP.

An important thing to notice is that the methods proposed by [6, 12, 5] are all the more efficient as the size of U, i.e. the dimension of λ is small and the S_i are of balanced sizes. However, to our knowledge, no algorithms have been proposed in order to automate the construction of a "good" partition of the state space. We propose such methods, inspired by works on graph-partitioning, in the next section.

4 AUTOMATED STATE SPACE DECOMPOSITION

As we mentioned in the previous section, the problem of finding a good k-partition of the state space can be stated as:

Find a partition $(S_1, \ldots, S_k)$, as balanced as possible, such that the cardinal of $U = \bigcup_{i=1}^{k} Per(S_i)$ is minimum.

A similar kind of problem is encountered, concerning graphs: the *minimal cuts graph partitioning problem.*

Definition 4 *Minimal cuts graph partitioning problem. Let $\mathcal{G} = (V, E)$ be a graph, where V is the set of vertices, and E the set of edges. The problem is : Find a partition $(V_1, \ldots, V_k)$ of V, as balanced as possible, minimizing the size of $Cuts = \{e = (v_1, v_2) \in E, s.t. \exists i \neq j, v_1 \in V_i, v_2 \in V_j\}$.*

This problem is NP-complete even for bipartition ($k = 2$), but can be approximately solved through spectral theory techniques. The solution obtained can then be locally refined, in order to find the solution to the minimal cut bipartition problem. When k-partition is concerned, recursive applications of spectral bipartition/refinement can be performed, although spectral quadrisection/octasection are possible, to the price of a higher computational cost [8].

Our proposition for finding a good state space bipartition is to apply spectral bipartition to the underlying graph of the MDP, and then to locally refine the partition in order to minimize $|U|$. A k-partition can be obtained through successive bipartitions. Let us now describe in details the steps of the bipartitioning of a MDP :

- obtaining a symmetric graph from the MDP model,
- spectral bipartition of the graph or multilevel spectral bipartition for large graphs,
- local refinement of the bipartition.

4.1 Symmetric graph of the MDP

Let $< S, A, p, r >$ be a MDP. We define the symmetric graph G$= (V, E)$ of the MDP as follows.

Definition 5 *Symmetric graph of a MDP. $\mathcal{G} = (V, E)$ is the symmetric graph of MDP $< S, A, p, r >$ iff $V = S$ and $E = \{(s, s') \in S \times S / \exists a \in A, p(s, a, s') > 0$ or $p(s', a, s) > 0\}$.*

In other words, s and s' are adjacent in the graph iff there is an action leading possibly from one state to the other.

4.2 Spectral bipartition

We define the *Laplacian matrix* of the graph as follows:

Definition 6 *Laplacian Q of graph $\mathcal{G} = (V, E)$.*
- $Q(s, s') = -1 \; \forall (s, s') \in V^2$, $s \neq s'$ *and* $(s, s') \in E$,
- $Q(s, s') = 0 \; \forall (s, s') \in V^2$, $s \neq s'$ *and* $(s, s') \notin E$,
- $Q(s, s) = -(\sum_{s \neq s'} Q(s, s')), \forall s \in V$.

Q is of the form $Q = D - A$, where A is the adjacency matrix of the graph. Let now $\Pi = (V_1, V_2)$ be a bipartition of V and $X = (x_i)$ be a vector of size $|V|$, such that $x_i = 1$ if $s_i \in V_1$ and $x_i = -1$ if $s_i \in V_2$. The number of cuts induced by the partition (V_1, V_2), $\delta(V_1, V_2)$ is equal to (see, e.g. [13]):

Proposition 2

$$\delta(V_1, V_2) = \frac{1}{4}(X^t Q X).$$

As a consequence, the minimal cut bipartition problem can be stated as:

$$\text{Find } |\delta_{min}| = \min_{x_i \in \{-1,1\}, |\sum x_i| \leq \varepsilon} \frac{1}{4}(X^t Q X) \qquad (3)$$

where the constraint $|\sum x_i| \leq \varepsilon$ which is equivalent to $||V_1| - |V_2|| \leq \varepsilon$ is a constraint on the balance of the bipartition (if $\varepsilon = 0$, the two subsets have the same number of vertices).

The problem (3) is NP-complete, but an approximate solution can be obtained through the following steps :
1) Compute Z, the eigenvector of Q corresponding to the smallest, strictly positive eigenvalue[2].
2) Project Z on $\{-1, 1\}^{|S|}$ while respecting the balance constraint.

There exist efficient algorithms that perform step 1) and in our implementation we used the MATLAB built-in function *eigs* that does it for large, sparse, symetric matrices. Step 2) can be directly performed by setting to -1 the values of the negative elements of Z and to +1 the others. This can be slightly alterated if the result is not balanced.

4.3 Multilevel bipartition

Spectral bipartition is impractical for a graph of more than a few hundreds of vertices, which severely limits this approach for bisecting large MDP state spaces. [8] propose an improvement of the algorithm, that allows to deal with large graphs. This improvement is based on :

- A *coarsening* procedure which transforms a given weighted graph into a smaller, coarse weighted graph,
- an adaptation of the spectral bisection algorithm which enables it to deal with weighted graphs,
- an *uncoarsening* procedure, which generates a partition of the fine graph, from the partition of the coarse graph.

Before going any further, let us introduce the following notations : a weighted graph is now represented by $\mathcal{G} = (V, E, w_v, w_e)$ where V and E are the usual vertices and edges sets. w_v and w_e are integer valued vectors of strictly positive weights attached to the vertices and edges of the graph, respectively. We define the *weighted Laplacian* and *weight vector* of a weighted graph as :

Definition 7 *Weighted Laplacian and weight vector. Matrix Q is the weighted Laplacian of $\mathcal{G} = (V, E, w_v, w_e)$ iff $\forall (s, s') \in V^2, s \neq s'$, $Q(s, s') = -w_e(s, s')$ and $Q(s, s) = \sum_{s' \neq s} w_e(s, s')$. Similarly,* a vertices weight vector W *is defined as $W(s) = w_v(s), \forall s \in V$.*

Let us then briefly describe the different steps of the multilevel bisection algorithm:

Coarsening of a weighted graph. The first step in the coarsening of a graph is to find a matching $\mathcal{M}$ of the graph, that is a set of edges which have no vertices in common. The following simple procedure allows to find such

[2] Z is the solution to the minimization problem (3) without constraints.

a matching $\mathcal{M}$, which is furthermore *maximal*, in that no matching $\mathcal{M}'$ exists which strictly contains $\mathcal{M}$[3] :

```
    Maximal matching.
Let go through $V$, in a random but fixed order.
For each vertex $v$ encountered:
    - Remove $v$ from $V$,
    - if there exists $v'$ such that $(v,v') \in E$, remove
$v'$ from $V$ and add $(v,v')$ to $\mathcal{M}$,
    - remove all edges containing $v$ or $v'$ from $E$.
```

This procedure is linear in the size of V. Once the matching $\mathcal{M}$ is obtained, we can construct the coarsened graph (V', E', w'_v, w'_e). For each matching pair $(v,v') \in \mathcal{M}$, form a new vertex $v_{v,v'}$, of weight $w'_v(v_{v,v'}) = w_v(v) + w_v(v')$. If v and v' are both adjacent to vertex v'', $w'_e(v_{v,v'}, v'') = w_e(v,v'') + w_e(v',v'')$.

The adjacency matrix and weight vector of the coarsened graph can be obtained from the coarsening matrix M: M is an $m \times n$ matrix, where n is the cardinal of V, and m is the number of edges in $\mathcal{M}$, plus the number of unmatched vertices. Each line of M corresponds either : i) to an edge in $\mathcal{M}$, in which case the line has zeros everywhere, except on the places corresponding to the vertices in the associated edge where it has ones, ii) or to an unmatched vertex, in which case it has zeros everywhere, except on the place corresponding to the unmatched vertex.

If A and W are respectively the adjacency matrix and Weight vector of the original graph, $A' = MA$ and $W' = MW$ are the adjacency matrix and weight vector of the coarsened graph (V', E', w'_v, w'_e). This coarsening procedure is exemplified in Figure 1.

An important property of this coarsening procedure is that the weighted number of cuts for a given partition of the coarse graph is equal to the weighted number of cuts for the corresponding partition of the fine graph. Furthermore, the total weights of vertices in each part are equal in the coarse and fine graphs.

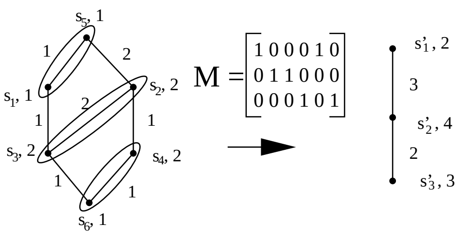

Figure 1. Coarsening of a graph.

Bisection of a weighted graph. Proposition 2 holds also for weighted Laplacians associated to weighted graphs, as shown in [8]. So, it is possible, exactly as in the unweighted case, to find a partition of a weighted graph, through spectral bisection, the balance constraint being replaced by a constraint on the difference of the sum of the weights of the vertices in the two parts of the bisection.

Uncoarsening of the coarse partition. Once the original graph has been coarsened, and the spectral bisection made, we are left with a partition of the coarse graph. This partition can be transformed in a partition of the fine graph in a direct way, by assigning the vertices of a matched pair to the same part as their coarse counterpart. This simple transformation ensures that the balance constraint is still fulfilled, as the number of cuts. The uncoarsening step can be easily expressed in matrix form. If $X' \in \{-1,1\}^m$ represents a bipartition of the coarse graph, $X =^t MX'$ is the corresponding partition of the fine graph.

4.4 Local refinement

In the multilevel spectral bipartition algorithm, several sources of imprecision arise in the search for minimal U-size bipartition:

- First, spectral bipartition aims at finding minimal edge-cuts bipartition, which is different from minimal U-size bipartition, although correlated.

- Second, when the NP-complete original problem is replaced with eigenvector search another approximation is made, since the search in a discrete space is replaced by a search in a continuous space followed by a projection of the result on the discrete space.

- Third, when a coarse partition is searched for, vertices that have been matched together can not be split by the bipartition algorithm, even if they would be separated by an optimal bipartition.

For these reasons, the result of a multilevel spectral bipartition should only be seen as an approximate solution to the minimal U-size bipartition problem, that deserves to be locally refined. Let us describe a refinement scheme that is inspired from the algorithm of [9], designed for the minimal cut bipartition problem. Notice that this scheme is local and may not lead to a global optimum.

```
    Local refinement.
Bestpart $\leftarrow (S_1, S_2)$ ;
Besteval $\leftarrow |Per(S_1) \cup Per(S_2)|$ ;
$Improve \leftarrow true$ ;
While $(Improve)$
    Perform a sequence of greedy improving swaps[4]
each vertex being swapped at most once ;
    If the best partition encountered is better
than Bestpart and balanced then
    Update Bestpart and Besteval
    Else $Improve \leftarrow false$ ;
End ;
```

Finally, multilevel bipartition goes as follows :
- Construct the graph $\mathcal{G} = (V, E)$ of the MDP.
- Perform successive coarsening steps $\mathcal{G}_{i+1} = Coarsen(\mathcal{G}_i)$, until $|V_i|$ is below a given threshold.
- Perform weighted spectral bipartition on $\mathcal{G}_i$.
- Perform uncoarsening steps, interleaved with local refinement steps if desired.

[3] Note that there may be several maximal matchings for a given graph, not having the same cardinality. The procedure we describe is the simplest for finding maximal matchings but there is no guarantee that it returns a matching with maximal cardinality.

[4] $Swap(s), s \in S_1$ is greedy improving iff it minimizes $|Per(S_1 - \{s'\}) \cup Per(S_2 \cup \{s'\})|$.

4.5 Example

We have illustrated the approach on the navigation example of Figure 2. The effect of actions are stochastic: when trying to move to an adjacent square, the two squares that are adjacent to the target (and diagonal to the source) may be reached with probability 0.2 each. Here, we have applied three successive multilevel bipartition steps, each followed by a local refinement step. For multilevel bipartition, successive coarsening steps were performed, until the number of vertices of the coarse graph was under 50. The balance constraint in the spectral bipartition phase was that there should be at least 40 squares in each part. Figure 2 should be read from left to right for the three bipartition steps, the results of the spectral bipartitions being on top, the refined results on bottom.

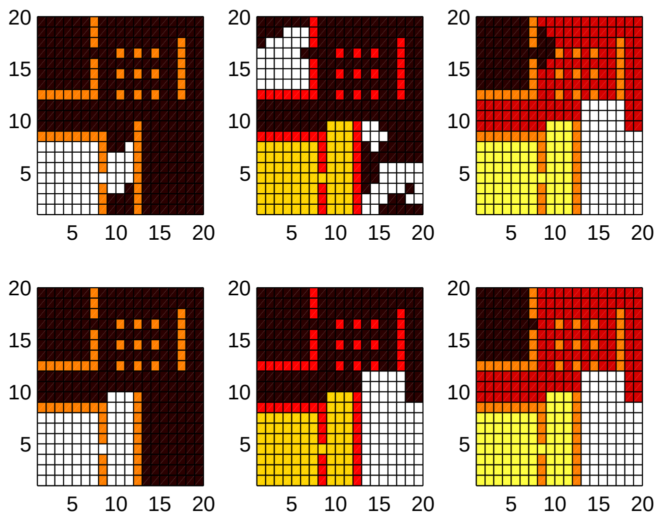

Figure 2. Successive bipartitions from left to right, up: spectral bipartition, down: refined bipartition.

To give an indication on the usefulness of the obtained decomposition, we applied an MDP decomposition method to this problem, where a positive reward is obtained when the state of coordinates $(3, 3)$ is reached, a null reward everywhere else and a discount factor of 0.9. In the case of the first bipartition, we compared direct application of value iteration[5] with a simple method alterning improvement steps of local policies (through the execution of a small number of policy improvement steps) and updates of the λ function on U.

The implementation was done in MATLAB on a PIII 600 biprocessor under Linux. Direct Value iteration took 46.14 secs, while spectral bipartition, local refinement and solving of the decomposed MDP took respectively 3.17 secs, 2.37 secs, and 20.27 sec. Different runs, leading to different bipartitions (there is a random factor in the coarsening process) gave results quite similar, except for the local refinement process, which execution time has a high variability.

5 CONCLUDING REMARKS

We have proposed algorithms for the automated decomposition of state spaces of large MDPs. This work is complementary to previous works on MDP decomposition. We have checked experimentally on a very restricted domain that the cost induced by automated decomposition is more than compensated by the gains obtained with decomposition-based solving of MDPs. However, more effort should be devoted to assess the efficiency of such methods.

Further developments of the methods we have proposed should involve factored representations of MDPs [4]. Indeed, large MDPs are especially encountered when structured representations (logical, Bayes net...) are used. [4] use aggregation methods for solving more efficiently factored MDPs. Automated decomposition methods such as the ones we propose could be usefully integrated to enhance the performances of the aggregation-based methods, by e.g. building graphs on aggregated states in order to decompose the aggregated MDPs.

[5] Which goes faster than policy iteration on this example

REFERENCES

[1] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[2] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, 1987.

[3] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[4] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

[5] C. Boutilier, M. Hauskrecht, N. Meuleau, L.P. Kaelbling, and T. Dean. Hierarchical solutions of markov decision processes using macro-actions. In *Proc. 14th conf. on Uncertainty in Art. Int. (UAI'98)*, pages 220–229, Madison, WI, 1998.

[6] T. Dean and S.H. Lin. Decomposition techniques for planning in stochastic domains. In *Proc. IJCAI'95*, pages 1121–1127, Montreal, Canada, 1995.

[7] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Art. Int.*, 89:219–283, 1997.

[8] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Stat. and Comput.*, 16:452–469, 1995.

[9] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 29:291–307, 1970.

[10] H.J. Kushner and C.H. Chen. Decomposition of systems governed by markov chains. *IEEE transactions on Automatic Control*, 5(19):501–507, 1974.

[11] M.L. Littman. Value-function reinforcement learning in markov games. *J. of Cognitive Sys. Research*, 2:55–66, 2001.

[12] R. Parr. Flexible decomposition algorithms for weakly coupled markov decision processes. In *Proc. UAI'98*, pages 422–430, Madison, WI, 1998.

[13] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Analysis*, 11:430–452, 1990.

[14] D. Precup, R. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract behaviors. In *Proc. ECML'98*, pages 382–393, 1998.

[15] M.L. Puterman. *Markov Decision Processes.* Wiley and Sons, New York, 1994.

[16] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.