Solving large structured Markov Decision Problems for perishable inventory management and traffic control

Haijema, R.

Publication date
2008

Link to publication

Citation for published version (APA):
Haijema, R. (2008). *Solving large structured Markov Decision Problems for perishable inventory management and traffic control*. Thela Thesis.

Chapter 1

Introduction, outline and preliminaries

1.1 Introduction and outline

1.1.1 Large structured Markov decision problems

Many optimization problems arising in practice involve sequential decision making and can be formulated as a Markov decision problem (MDP). Solving MDPs numerically is however restricted to problems that have a relatively small state space, of say at most a few million states. An optimal solution, often called an optimal strategy or an optimal policy, prescribes a best action to take in each individual state that may occur. Many MDPs arising in practice tend to have too many states, due to the dimensionality of the state space. An optimal solution can then not be computed in reasonable time. Fortunately, often an approximate solution will do and might even be favored.

Constructing an approximation algorithm requires a good intuition and insight in the problem under consideration. The problem often exhibits some structure, which can be very helpful in solving the problem. A general approach of how to exploit the problem structure does not exist for all problems, since the structure may be problem specific. In this thesis, we illustrate how the formulation of a problem as an MDP helps to construct numerical solutions. Simulation turns out to be crucial as an evaluation tool.

1.1.2 Two applications

We focus on two different optimization problems:

I. the production-inventory management of perishables; in particular that of blood platelet pools, and

II. the dynamic control of traffic lights.

Production-inventory management of perishables

In the first optimization problem, we consider perishable products with a fixed maximal shelf life of m periods, say $m = 5$ days. One is interested in optimal production volumes, or order quantities, that balance the occurrence of outdating and shortages. Ordering too many products, results in excessive outdating after m periods. Ordering not enough products results in shortages; depending on the context unmet demand is considered as lost sales or is backlogged, eventually it may require an emergency order.

We study a perishable inventory problem that arises at blood banks and hospitals, which keep blood products in stock to meet the uncertain demand. Blood banks place production orders and hospitals place replenishment orders to keep their inventory at a safe level. Most inventory managers have difficulty in making the trade-off between the risk of shortages and outdating, since demand is uncertain and the ethical and economic impact of shortages and outdating is great.

So-called *blood platelet pools* (BPP) are thrombocyte concentrates that have a very short shelf life of only 5 to 7 days. The production of a single BPP requires platelets of 5 donors, expensive material and laboratory test to guarantee safe transfusions. This makes a BPP the most expensive blood products. Nevertheless, 10-20% of the BPP production becomes non-transfusable, because the transfusion date expires.

When solving this problem of finding good order sizes, a number of complicating aspects has to be acknowledged. We deliberate on them in the first part of this thesis, starting with Chapter 2. Without going into the details here, the state description shall contain a description of the number of so-called platelet pools of a specific age-category. Each age-category corresponds to a dimension of the state space. The resulting number of states, which is the product of the possible number of states in each dimension, becomes very large when applying real data. For example, suppose the maximal shelf life of a product is 5 periods (days), and at most 10 products are in stock of each of the 5 age categories, then the stock state consists already of 10^5 states.

In the practice of the Dutch blood banks, the state space is much larger and a number of complications have to be dealt with, such as the distinction of blood groups. A straightforward MDP approach is thus doomed to fail. But an approximate solution is of general interest, since it may reduce shortages and outdating of platelet pools significantly. The quality of the pools deteriorate over time. For some category of patients, very young platelet pools are strongly preferred to extend the period until the next transfusion. Therefore, we also study several issuing policies that prescribe which pools are selected from stock to meet the demand.

The structure of the perishable inventory problem allows to aggregate different states into a single state: e.g. multiple pools of different blood groups may be aggregated into batches. This reduces the state space significantly and thus we are able to solve the problem numerically. By simulation one may detect which states are visited most frequently and which actions are taken in these states. This way we may detect some structure in the thus obtained optimal strategy, maybe only after aggregating the age-categories. In a number of cases, we show that the optimal strategy turns out to resemble a simple rule. As we will show in Part I, there are a number of obstacles to take; such as the problem that the aggregation of states affects the transition law from one state to a next state. Therefore, the approach needs to be validated by an extensive sensitivity study, from which we obtain insights under what conditions the approach is valid.

Making a nearly optimal trade-off between high availability and low outdating figures is of general importance. Examples are found not only at blood banks and in hospitals, but also in the food-processing industry amongst other.

Dynamic control of traffic lights

The second problem that we study in this thesis, is the minimization of the expected waiting time per car at a traffic light. The optimization problem can be formulated as a MDP. For a detailed problem description, we refer to Chapter 6 in the second part. The state description contains, amongst other, the number of queued cars waiting at each stopping line. When cars approach the intersection from four directions and each direction consists of three separate lanes for respectively right turning, left turning and 'through' traffic, then the state space consists already of (at least) 12 dimensions. When at any time maximal 9 cars are queued at each lane, then the state space contains already (at least) 10^{12} states. Given the number of states, the MDP is intractable for most real intersections. An approximate solution is thus needed.

As will be shown in Part II, we may add additional structure to this problem by imposing a special strategy that allows the decomposition of the state space. Through a single policy improvement step, we obtain an approximate solution that appears to perform quite well: it results in low average waiting times compared to other control policies. The approach for a single intersection allows the inclusion of information on arrival times of near future car arrivals. Applying the approach to networks of intersections shows significant reduction in the overall average waiting times compared to the several existing control policies.

1.1.3 Simulation

Throughout this thesis (discrete) simulation plays three important roles:

- Due to the high dimensionality and the scale of the problem, the obtained approximate policies can be evaluated by simulation only.
- One may obtain a profound insight into (nearly) optimal strategies, by studying simulation results.
- Simulation enables to bridge the gap between the real system and the abstract mathematical models. By simulation one may validate in great detail the use of simplified decision models and approximate solutions.

1.1.4 Outline

In the next sections of this first chapter we discuss some preliminaries on Markov decision problems (MDP) and how to solve them numerically by Stochastic Dynamic Programming (SDP). In addition, we explain how numerical solution procedures can be exploited to evaluate a Markov chain (MC) of a specific (fixed) strategy. The concept of relative values of states is discussed, and finally we show how (numerically computed) relative values can be used in a one-step policy improvement algorithm. Readers not interested in, or already familiar with, these topics may skip the remainder of this chapter and may find the overview in Table 1.1 helpful in reading the thesis.

The remainder of the thesis is divided into two parts related to the two applications that are discussed. Part I consists of Chapters 2 to 4, and addresses the production-inventory management of perishables with a focus on blood platelet pools. Chapter 2 provides a detailed problem description and an extensive review of the history of perishable inventory models. In Chapter 3 a combined SDP-Simulation approach for solving the

Table 1.1: Outline of thesis

1	Introduction and Overview of this thesis
PART I	PERISHABLE INVENTORY MANAGEMENT
2	Perishable inventory theory
3	Combined SDP-Simulation approach
4	Extensions
PART II	DYNAMIC CONTROL OF TRAFFIC LIGHTS
5	Inroduction
6	Single intersection in isolation
7	Using arrival information
8	Arterials and Networks of intersections
9	Epilogue
	Appendices
	Bibliography
	Summary in English and Dutch

production-inventory problem at blood banks is introduced and applied to a case study, including a detailed sensitivity study. In Chapter 4 the problem of interest is extended to include non-stationary periods and fixed order costs, which may apply to hospitals. Despite the technical details, the results reported in this first part are of interest to managers of perishable inventory, and blood bank managers in particular, next to Operations Researchers.

Part II consists of Chapters 5 to 8 and introduces an MDP approach for controlling traffic lights. In Chapter 6, we explain how the waiting times at single intersections in isolation can be reduced by a one-step policy improvement algorithm. In Chapter 7, the approach is extended to include arrival information. Finally, in Chapter 8, we apply the approach to a network of intersections. In Chapter 9 we conclude the thesis by postulating some conclusions and discussing to what extent the two approaches can be generalized. After the Appendices, one finds summaries of the thesis in English and in Dutch.

Notations – Throughout this thesis some notations and conventions are in use as summarized in Appendix A. In some occasions, we allow ourselves using different notations (sometimes for typographical reasons and sometimes to avoid confusion).

1.2 Preliminaries on Markov Decision Problems

In this section we will present the main modeling assumptions of a Markov Decision problem (MDP), we introduce some solution procedures and discuss the computational complexity. We restrict the discussion to discrete time MDPs, in which the time is slotted into time intervals of fixed length, called slots. For a more general and more accurate discussion of MDP theory we refer to text books such as [14], [68], [120] and [144].

1.2.1 Modeling MDPs

Suppose one is responsible for controlling a dynamic system in discrete time. Therefore decisions, or actions, are taken at fixed discrete points in time, say at the start of each slot. Just before an action is selected, the state of the system is inspected. As a result of an action and a number of uncertain events, the state of the systems changes according to a probability distribution. The probability distribution captures the uncertainty of events that may happen between two decision epochs.

Related to a slot are (expected) direct or immediate costs, which depend on the state at the start of that slot, and possibly on the decision taken and the transition probabilities. In addition to direct costs, direct rewards may be included. In this discussion, we consider costs only; rewards can be seen as negative costs. The objective is to minimize the costs over some finite horizon, or to minimize the average costs per slot over an infinite horizon. An alternative, which we do not consider, is the minimization of the total discounted costs by discounting future costs.

Below we deliberate on the components of an MDP: the states, decisions, transitions, and related probabilities and costs.

States − The above sequential decision problem is a Markov decision problem if the so-called Markov-property or memoryless property holds: the state description should contain all relevant information (about the past and the current situation) for taking a decision, for incurring (expected) direct costs and for deriving the transition probabilities. No information on previously visited states is needed, other than the information included in the description of the current state.

We focus on MDPs where the state space is multi-dimensional. States will be represented by a vector $\mathbf{x} = (x_1, ..., x_m)$, where x_i can take values in $\mathcal{X}_i$. We allow a state space $\mathcal{X} = \prod_{i=1}^{m} \mathcal{X}_i$ to be an infinite countable state space, but we will truncate it to a finite one such to allow for the numerical computation of an optimal state-dependent policy. (In finite horizon problems, the slot or stage number may be included in the state space, resulting in different mutually exclusive subspaces: one for each time slot.)

Decisions – Every decision epoch, say at the start of each slot, a decision a is selected out of the action set $\mathcal{A}$, which may be state-dependent: $\mathcal{A}(\mathbf{x}) \subseteq \mathcal{A}$. In some states $\mathbf{x}$ only one single action is feasible, then the action set $\mathcal{A}(\mathbf{x})$ consists of one element only. (In principle, when multiple decisions are to be taken simultaneously, the action space may be multi-dimensional. In this thesis, we will not deal with multi-dimensional action spaces.)

Transitions – For determining the transition from one state to a next state, one needs to specify the course of events that may happen during a slot. The state of the system, $\mathbf{x}$, is inspected at the start of a slot. Based on the observed state $\mathbf{x}$, an action a is selected. Next, any certain and uncertain events are modeled in a pre-specified order. Finally, one computes the resulting state $\mathbf{y}$ at the end of the slot, or say at the start of the next slot. When action a is taken in state $\mathbf{x}$, only states $\mathbf{y}$ in $\mathcal{X}(\mathbf{x}, a) \subseteq \mathcal{X}$ can be reached. Different sequences of events that may happen can result in the same next state $\mathbf{y}$.

At the start of the next slot, the system will be in state $\mathbf{y}$ with probability $P_{\mathbf{x},\mathbf{y}}(a)$, given (state, action)-pair $(\mathbf{x}, a)$. These transition probabilities are computed from known probability distributions of the uncertain events that may happen during a slot.

Direct costs – The total costs to incur over a planning horizon, can be broken down in cost directly related to the successive slots. When at the start of a slot action a is taken in state $\mathbf{x}$, then direct costs $c(\mathbf{x}, a, \mathbf{y})$ are incurred when the next state is $\mathbf{y}$. As multiple uncertain events may result in the same transition from state $\mathbf{x}$ to $\mathbf{y}$, the direct costs are the expected costs over all possible events related to that transition.

As multiple states y can be visited after taking decision a in state $\mathbf{x}$, one often computes the expected direct costs: $\mathbb{E}C(\mathbf{x}, a)$. The expected direct costs $\mathbb{E}C(\mathbf{x}, a)$ to incur when action a is selected in state $\mathbf{x}$ depend on the transition probability $P_{\mathbf{x},\mathbf{y}}(a)$, and is computed as follows:

$$\mathbb{E}C(\mathbf{x}, a) = \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x}, a)} c(\mathbf{x}, a, \mathbf{y}) P_{\mathbf{x},\mathbf{y}}(a). \tag{1.1}$$

Criterion – The planning horizon of the MDP is either finite or infinite. In finite horizon MDPs, one usually minimizes the expected cost over the planning horizon (with or without discounting). In infinite horizon models, one minimizes either the long-run average cost or the expected total discounted costs.

1.2.2 Solving an MDP by Stochastic Dynamic Programming

When the planning horizon is finite, say N time slots, then an MDP is solved by (discrete) stochastic dynamic programming (SDP) in N stages. In general the time slots may differ in length, but we assume them to be of fixed identical length. When the planning horizon is infinite, a long-run-optimal policy can be derived by a Successive Approximations (SA) algorithm, which is in essence an SDP algorithm. We briefly present the finite and infinite horizon problems and discuss how they are solved by SDP. We limit the discussion to unichain cases. From the discussion of an SA algorithm, we learn about the complexity of solving infinite horizon MDPs.

Another solution procedure called Policy Iteration (PI) is discussed in Section 1.2.3. A third solution procedure, which we do not discuss, is based on Linear Programming (LP). For a more complete and rigorous treatment of this subject we refer to standard textbooks (such as [68], [120], and [144]).

Finite horizon problem

In a finite horizon problem one often labels the time slots backwards from 1 to N, and so do we in the discussion below. These labels refer to the N stages or iterations of the solution process by SDP. Usually the action space, the expected direct costs or the transition probabilities are stage dependent. An optimal strategy will then be non-stationary: i.e. optimal actions depend on the stage number $n \in \{1, 2, \ldots, N\}$. We add the slot number n to the state description. State $(n, \mathbf{x})$, thus denotes state $\mathbf{x}$ at the n-th stage of the solution process. Let $\mathcal{X}$ be the set of all state vectors $\mathbf{x}$ that are feasible at one ore more stages. The state space at the start of slot n, denoted by $\mathcal{X}(n)$, is then a subspace of $\mathcal{X}$. Further, we change the notation $\mathbb{E}C(\mathbf{x}, a)$ into $\mathbb{E}C(n, \mathbf{x}, a)$, and the transition probability will be $P_{n,\mathbf{x},\mathbf{y}}(a)$.

For n running from 1 to N, one computes for all $\mathbf{x} \in \mathcal{X}(n)$ the following dynamic programming (DP) recursion:

$$V(n, \mathbf{x}) = \min_{a \in \mathcal{A}(n,\mathbf{x})} [\, \mathbb{E}C(n, \mathbf{x}, a) \; + \sum_{\mathbf{y} \in \mathcal{X}(n-1|\mathbf{x},a)} P_{n,\mathbf{x},\mathbf{y}}(a)\, V(n-1, \mathbf{y}) \,], \tag{1.2}$$

where $\mathcal{X}(n-1|\mathbf{x}, a)$ is the set of states that can be reached from state $\mathbf{x}$ when action a is selected at the n-th stage. Thus, $\mathcal{X}(n-1|\mathbf{x}, a)$ is a subset of the state space at the start of the next slot (= slot $n-1$): $\mathcal{X}(n-1|\mathbf{x}, a) \subseteq \mathcal{X}(n-1)$.

Element $V(n, \mathbf{x})$ of the value vector equals the total expected costs over the last n slots of the planning horizon, when starting slot n in state $\mathbf{x}$ and only optimal actions are taken until the end of the horizon. This includes terminal costs, which will be zero when $V(0, \mathbf{y})$ is set to 0 for each state $\mathbf{y}$.

An optimal strategy π^* prescribes for each stage $n \in \{1, 2, \ldots, N\}$ and every state $\mathbf{x} \in \mathcal{X}(n)$ an optimal action according to Equation (1.2).

Remark – Note that the numerical computations of (1.2) can only be executed when the state space is finite. When this is not natural for the underlying problem, one needs to truncate the state space. Then one accepts a modeling error, for which one may need to compensate in the direct cost structure. Anyway, the truncation should not affect the optimal policy too much.

Infinite horizon – the Successive Approximation (SA) algorithm

An infinite horizon MDP is usually assumed to be *stationary* and *aperiodic*, i.e. the cost structure, transition probabilities, and the action space are identical in each stage. As a result the optimal policy is stationary. Solving an infinite horizon MDP is done in a way very similar way to solving an finite horizon MDP. An optimal policy can be computed by a Successive Approximations (SA) algorithm, which is based on SDP. The next four steps constitute an SA algorithm for determining a policy that minimizes the (long-run) average cost per slot for a stationary, aperiodic, infinite-horizon problem.

Successive Approximations algorithm

Step 1. Set $n = 0$, $\forall \mathbf{x} \in \mathcal{X}:\ V_0(\mathbf{x}) = 0$, and $\epsilon =$‘small’, i.e. ϵ is small compared to the (unknown) minimal long-run average cost per slot, that relates to the values that $\mathbb{E}C(\mathbf{x}, a)$ can take,

Step 2. $n = n + 1$, and
Compute for all $\mathbf{x} \in \mathcal{X}$:

$$V_n(\mathbf{x}) = \min_{a \in \mathcal{A}(\mathbf{x})} [\ \mathbb{E}C(\mathbf{x}, a)\ +\ \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x},a)} P_{\mathbf{x},\mathbf{y}}(a)\ V_{n-1}(\mathbf{y})\ ] \tag{1.3}$$

and store a minimizing action a: $\pi(\mathbf{x}) = a$,

Step 3. Compute:

$$U_n = \max_{\mathbf{x} \in \mathcal{X}}[V_n(\mathbf{x}) - V_{n-1}(\mathbf{x})], \quad \text{and} \quad L_n = \min_{\mathbf{x} \in \mathcal{X}}[V_n(\mathbf{x}) - V_{n-1}(\mathbf{x})],$$

Step 4. If $span(\mathbf{V}_n - \mathbf{V}_{n-1}) \equiv U_n - L_n\ \geq \epsilon$ then return to Step 2,
else $N = n$ and π approximates a (nearly) optimal stationary policy π^*. The average costs g^π can be approximated by $\frac{L_N + U_N}{2}$, and will be ϵ-close to the optimal average cost level g^*, as $g^\pi - g^* < \epsilon$.

Explanation – An optimal policy π^* is thus approximated by successively extending the planning horizon n by one time slot per iteration. Since the length of the planning horizon is indefinite, the range of n is in principle unbounded: $n \in \{1, 2, ...\}$. After a large number of, say N, iterations, the difference between two successive value vectors $\mathbf{V}_n$ and $\mathbf{V}_{n-1}$ has converged, with ϵ-precision, to a vector of constants equal to the long-run average costs per slot, g.

The convergence of the vector $\mathbf{V}_n - \mathbf{V}_{n-1}$ is checked by the so-called span, which is defined in Equation (1.4):

$$span(\mathbf{V}_n - \mathbf{V}_{n-1}) = \max_{\mathbf{x} \in \mathcal{X}} [\, V_n(\mathbf{x}) - V_{n-1}(\mathbf{x})\,] - \min_{\mathbf{x} \in \mathcal{X}} [\, V_n(\mathbf{x}) - V_{n-1}(\mathbf{x})\,]. \tag{1.4}$$

The SA algorithm stops when $span(\mathbf{V}_n - \mathbf{V}_{n-1}) < \epsilon$. The convergence of the span happens exponentially fast, but the speed of convergence, i.e. the required number of iterations N, depends on the problem data.

The optimal strategy π^* is approximated by the policy π obtained in the last iteration. When not stored in Step 2, the optimal actions follows from another improvement step:

$$\pi^*(\mathbf{x}) \approx \arg \min_{a \in \mathcal{A}(\mathbf{x})} \left[\, \mathbb{E}C(\mathbf{x}, a) \;+ \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x},a)} P_{\mathbf{x},\mathbf{y}}(a)\; V_N(\mathbf{y}) \,\right] \tag{1.5}$$

The minimal long-run average costs are estimated by that under policy π, i.e. by g^π, for which an upper bound U_N and a lower bound L_n is available: $U_N - g^\pi < \epsilon$ and $g^\pi - L_N < \epsilon$.

Periodicity – If the problem is by nature periodic, e.g. in the action space, in the cost structure, or in the transition probabilities, with period D, any feasible policy is periodic. Then the state space is divided into D periodic classes. Under any policy the long-run average cost per slot is then class dependent: the long-run average costs per slot in class $d \in \{1, 2, \ldots, D\}$ is g_d. The long-run average costs over D successive slots is $\sum_{d=1}^{D} g_d = D \cdot g$, where g is the *gain* of the underlying MC, which equals the (overall) long-run average cost per slot.

The limit $\lim_{n\to\infty} span(\mathbf{V}_n - \mathbf{V}_{n-1})$ does not exist, as the differences between $\mathbf{V}_n$ and $\mathbf{V}_{n-1}$ change periodically with n. Therefore, Step 2 in the SA algorithm is executed D times before the algorithm proceeds to Step 3 and Step 4. In Step 4, $span(\mathbf{V}_n - \mathbf{V}_{n-D})$ is checked rather than $span(\mathbf{V}_n - \mathbf{V}_{n-1})$.

Even when the problem itself is aperiodic, an optimal policy may behave virtually periodic. Then the convergence of the SA algorithm is slow but it will converge after many iterations. If one suspects the optimal policy to behave virtually periodic with period D, one may speed up the solution process by checking $span(\mathbf{V}_n - \mathbf{V}_{n-D})$ rather than $span(\mathbf{V}_n - \mathbf{V}_{n-1})$.

Computational complexity of an SA algorithm

Solving an infinite horizon problem by successive approximations requires the computation of (1.3) for all states $\mathbf{x} \in \mathcal{X}$. The number of state vectors is $|\mathcal{X}|$ and depends on the number of values each element of the state vector $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ can take: $|\mathcal{X}| = \mathcal{O}(\prod_{i=1}^{m} |\mathcal{X}_i|)$. For each state $\mathbf{x}$, all $|\mathcal{A}(\mathbf{x})|$ feasible decisions are considered. Given any state-action pair $(\mathbf{x}, a)$, there are $|\mathcal{X}(\mathbf{x}, a)| \leq |\mathcal{X}|$ 'next-states' $\mathbf{y}$ to consider.

Hence, an upper bound on the total number of transitions computed in a single iteration is $|\mathcal{A}| \cdot |\mathcal{X}|^2$. The number of iterations N depends on the problem (and data) under consideration. The overall computational complexity is $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{X}|^2)$.

1.2.3 Solving an MDP by Policy Iteration (PI)

An alternative way of solving an MDP is by a *Policy Iteration (PI) algorithm.* The algorithm is presented below. For a more detailed discussion we refer to text books such as [68] and [120]. The algorithm consists of the following four Steps I to IV.

Policy Iteration algorithm

Step I. Select a stationary initial policy π_0, and set $\pi = \pi_0$ and set $n = 1$,

Step II. Value determination step
In the n-th iteration of the PI algorithm, evaluate the MC for policy π: i.e. derive a so-called relative value vector $\mathbf{v}^\pi$ and the gain, g^π, which must satisfies at least the following set of equations (in matrix notation):

$$\mathbf{v}^\pi + g^\pi \cdot \mathbf{1} = \mathbf{c}^\pi + \mathbf{P}^\pi \mathbf{v}^\pi, \tag{1.6}$$

where $\mathbf{c}^\pi$ has elements $\mathbb{E}C(\mathbf{x}, \pi(\mathbf{x}))$ and $\mathbf{P}^\pi$ has elements $P_{\mathbf{x},\mathbf{y}}(\pi(\mathbf{x}))$.

Note that a solution $(\mathbf{v}^\pi, g^\pi)$ to (1.6) is not unique. Therefore one may set an arbitrary element of $\mathbf{v}^\pi$ to zero, say $v^\pi(\mathbf{x}) = 0$. Alternatively, one may add the constraint $\mathbf{P}^\pi \mathbf{v}^\pi = \mathbf{0}$, such that the relative values $\mathbf{v}^\pi$ are the so-called *bias terms* $\widetilde{\mathbf{v}}^\pi$ of policy π.

Step III. Policy improvement step
Compute, $\forall \mathbf{x} \in \mathcal{X}$:

$$\pi_n(\mathbf{x}) = \arg \min_{a \in \mathcal{A}(\mathbf{x})} \left[\, \mathbb{E}C(\mathbf{x}, a) \; + \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x},a)} P_{\mathbf{x},\mathbf{y}}(a)\, v^\pi(\mathbf{y}) \,\right] \tag{1.7}$$

and set $\pi_n(\mathbf{x}) = \pi(\mathbf{x})$, whenever possible.

Step IV. If $\pi_n = \pi$ then stop as $\pi^* = \pi$ is an optimal policy,
else set $\pi = \pi_n$, and $n = n + 1$ and return to Step 2.

Remark – The core of the algorithm is Step II and Step III. In an alternative PI algorithm, Steps II and III are interchanged. Then the algorithm starts with an initial value vector $\mathbf{v}$, which may or may not be related to a (possibly unknown) strategy. Next, a policy improvement step is executed using the initial value vector $\mathbf{v}$. In the third step, the gain and relative values are determined for the resulting policy.

Computational complexity of a PI algorithm

The computational complexity of a PI algorithm is set by the value determination step. In that step the relative values of states for a given policy π, as well as the gain g of the underlying MC, are determined. Equation (1.6) represent a system of $|\mathcal{X}|$ equations in $|\mathcal{X}| + 1$ unknowns. To obtain a unique solution, one of the relative values can be set to zero (or any other constant). Then a solution can be obtained by applying a Gaussian elimination, or any other procedure to solve a set of linear equations.

Usually, the number of iterations that a PI algorithm requires is much smaller than that of an SA algorithm, but each iteration takes more time. When Gaussian elimination is used the computational complexity is $\mathcal{O}(|\mathcal{X}|^3)$. When the transition matrix $\mathbf{P}$ is sparse, more efficient procedures to solve the equations can be used. In the Section 1.3.2, we discuss how an approximate solution can be found by an SA algorithm with actions fixed to those set by policy π.

1.2.4 Computation times and memory usage

The numerical example below shows that deriving an optimal policy for a multi-dimensional MDP can be very time-consuming. For an SA algorithm we discuss both the (evolution of the) computation time as well as the memory usage.

Computation times – a numerical example

Consider an MDP with a 5-dimensional state space $\mathcal{X} = \prod_{i=1}^{5} \mathcal{X}_i$, and $|\mathcal{X}_i| = 20$ for all $i = 1, 2, \ldots, 5$. The total number of states is thus $20^5 = 3.2\,10^6$. When $|\mathcal{A}(\mathbf{x})|$ equals 15 for all states $\mathbf{x}$, then 15 possible actions are considered per state. When all $|\mathcal{X}|$ states can be reached from any state $\mathbf{x}$, then the number of transitions to evaluate per iteration is $15 \cdot (3.2\,10^6)^2 \approx 1.5\,10^{14}$. Fortunately, for many MDPs the transition matrix is sparse and well-structured: given action a, the number of states that can be reached from state $\mathbf{x}$ is much less than $|\mathcal{X}|$. If $|\mathcal{X}(\mathbf{x}, a)| = 30$ for all state-action pairs $(\mathbf{x}, a)$, then the number of transitions to consider is $30 \cdot 15 \cdot 3.2\,10^6 \approx 1.4\,10^9$. Suppose that a computer evaluates 1 million transitions per second, then evaluating all transition of a single iteration would take 1,400 seconds, or 23 minutes.

The total computation time depends also on the number of iterations N is problem specific and further depends on the required precision in the evaluation of $span(\mathbf{V}_n - \mathbf{V}_{n-1}) < \epsilon$. When $N = 30$ iterations are required, then in total about $4.2\,10^{10}$ transitions are to be considered. An optimal solution is thus obtained after 11.5 hours, which may or may not be acceptable depending on the context of the problem. Anyway, we are interested in solving problems of this size and much larger.

Evolution of computation time

When $|\mathcal{X}_i|$ doubles for all $i = 1, 2, \ldots, 5$, then the state space $|\mathcal{X}|$ will become 32 times as large. When $|\mathcal{X}(\mathbf{x}, a)|$, $|\mathcal{A}(\mathbf{x})|$ and N are unaffected, the computations will thus be 32 times as long. This dramatic increase of the number of states when the problem size increases is often referred to as the *curse of dimensionality*. Whether a problem can be solved in a reasonable time depends mainly on the total number of states. One should thus carefully investigate the structure of a problem to limit the number of states.

Memory usage

Another limitation may be the available memory. Fast RAM memory is used to store and update the value vectors, and to retrieve the transition probabilities and the direct costs. In 2008, the RAM memory on most computers was limited to a few Gb. For each state $\mathbf{x}$ two reals, $V_n(\mathbf{x})$ and $V_{n-1}(\mathbf{x})$, each of size 8 byte, are to be stored. In addition, one may store the transition probabilities in RAM memory. For the example given with $3.2\,10^6$ states one needs 51.2Mb fast RAM memory to store the two value vectors. (When $|\mathcal{X}_i|$ doubles for all $i = 1, 2, \ldots, 5$, then almost 1.7Gb RAM is to be used.)

In addition to the value vector, another 11 Gb RAM memory is needed to store all $1.4\,10^9$ transition probabilities $P_{\mathbf{x},\mathbf{y}}(a)$ for the given example. Although RAM memory is easily extended, memory consumption is an obstacle for large problems. To save RAM memory, one may decide to compute the transition probabilities online, rather than storing them all. Then the available computation time is usually the true bottleneck.

Parallel computing

Distributing the work over multiple PC gives a relatively small relieve of the computational trouble. For many problems encountered in practice, the state space is far too large to solve the problem to optimality, even when the computational burden is spread over many computers. For example, when a 5-dimensional problem can just be solved on a single PC, then a similar problem that is twice as large in each of the 5 dimensions requires $2^5 = 32$ computers connected to each other to solve the problem in the same time (ignoring any time lost in the communication between the 32 PCs).

1.2.5 Conclusion

For many multi-dimensional Markov decision problems, the curse of dimensionality necessitate the development of approximate solutions. Exploiting any structure present in a problem may be crucial for a successful solution of the problem. In the next section, we present a one-step policy improvement algorithm, which may be helpful in finding an approximate solution.

1.3 One-step policy improvement approach

1.3.1 Motivation

When computing an optimal policy for a stationary (infinite horizon) MDP is too time-consuming, one is interested in an approximate solution. Therefore, we discuss in this section how to evaluate and to improve an initial policy π through a single policy improvement step.

For a number of problems an approximate solution can be derived by execution of a single iteration of the PI algorithm starting with a well-structured policy π as the initial policy. That is, first a relative value vector $\mathbf{v}^\pi$ is computed for this initial policy π. Next, an improved policy π' is derived by executing a single policy improvement step:

$$\pi'(\mathbf{x}) = \arg \min_{a \in \mathcal{A}(\mathbf{x})} \left[\, \mathbb{E}C(\mathbf{x}, a) \;+ \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x},a)} P_{\mathbf{x},\mathbf{y}}(a)\, v^\pi(\mathbf{y}) \,\right]. \tag{1.8}$$

Such an approach is called a *one-step policy improvement approach*. Since the value determination step is the computationally most complex step in a PI algorithm, one looks for well-structured policies for which the relative values of states can be computed or approximated quickly. There are roughly two ways of approximating the relative values: one way is by solving the balance equations in an analytical way, another way is by the numerical computation of a relative value vector.

In either approach one looks for relative value functions or vectors that are separable. Therefore additional modeling assumptions may be needed to allow the decomposition of the state space of the MDP into subspaces. In Chapter 5, we give a number of examples in which a one-step policy improvement approach is followed after the decomposition of the state space.

For some problems closed-forms expressions or approximations of the relative value function exist for well-structured policies. The numerical computation of the relative values of states can be done under more general assumptions, but the initial policy must allow the decomposition of the state space. This means that, in a numerical approach, we look for a policy for which the relative values of states can be computed for each subspace in isolation of the other states. Therefore one has to analyze the MC for policy π using an SA algorithm with actions set to those specified by π. In the remainder of this section we discuss the numerical computation of relative values of states under any policy π.

1.3.2 Markov chain analysis for a fixed policy π

The SA algorithm of Section 1.2.2 can be used to evaluate a (MC) of a stationary policy π, when the action space $\mathcal{A}(\mathbf{x})$ in Equation (1.3) is replaced by a single action $\pi(\mathbf{x})$ as shown in (1.9):

$$\forall \mathbf{x} \in \mathcal{X}: \quad V_n^\pi(\mathbf{x}) = \mathbb{E}C(\mathbf{x}, \pi(\mathbf{x})) + \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x}, \pi(\mathbf{x}))} P_{\mathbf{x},\mathbf{y}}(\pi(\mathbf{x}))\, V_{n-1}^\pi(\mathbf{y}) \tag{1.9}$$

The long-run average costs or the gain g^π of the policy π follow from the difference between successive value vectors. When π is periodic with period D, the gain can be computed from $\lim_{n\to\infty} \left(\mathbf{V}_n^\pi - \mathbf{V}_{n-D}^\pi\right) = D \cdot g^\pi$. When π is aperiodic, then $D = 1$. To execute a policy improvement step the relative values of states need to be computed. As this is not straightforward for periodic policies, we split the discussion on how to compute a relative value vector: first for an aperiodic policy, next for a periodic policy.

Relative values of states for an aperiodic policy

If π is aperiodic, it can be shown [68] that the difference between the value vector $\mathbf{V}_n^\pi$ and the average cost over n slots converges to a vector $\widetilde{\mathbf{v}}^\pi$ that contains the bias terms of the states under policy π. The bias term $\tilde{v}^\pi(\mathbf{x})$ gives the relative appreciation of state $\mathbf{x}$:

$$\tilde{v}^\pi(\mathbf{x}) = \lim_{n\to\infty} (V_n^\pi(\mathbf{x}) - n \cdot g^\pi). \tag{1.10}$$

As such $\widetilde{\mathbf{v}}^\pi$ is a relative value vector that can be used in (1.8).

When policy π is stationary and aperiodic, the limit in (1.10) exists, for a prove see [68]. The bias terms $\widetilde{\mathbf{v}}^\pi$ satisfy (1.6), and so will do the vector $\widetilde{\mathbf{v}}^\pi - k \cdot \mathbf{1}$ for any k, as the relative value vector $\mathbf{v}^\pi$ is unique up to a constant. Hence when the number of iterations N of the SA algorithm is sufficiently large, $\mathbf{V}_N^\pi$ can be used directly in (1.8) to obtain an improved policy π':

$$\pi'(\mathbf{x}) = \arg\min_{a \in \mathcal{A}(\mathbf{x})} \left[\, \mathbb{E}C(\mathbf{x}, a) \;+ \sum_{\mathbf{y} \in \mathcal{X}(\mathbf{x}, a)} P_{\mathbf{x},\mathbf{y}}(a)\, V_N^\pi(\mathbf{y}) \,\right] \tag{1.11}$$

Policy π' is either an improvement over policy π or both π and π' are optimal. Note that $V_N^\pi(\mathbf{z}) - V_N^\pi(\mathbf{x})$ is the total expected cost difference between starting an horizon of N slots, in state $\mathbf{z}$ compared to starting in state $\mathbf{x}$.

Relative values of states for a periodic policy

When the policy π is periodic with period D, the state space is divided into D subspaces, or classes. The classes are numbered $1, 2, \ldots, D$ and visited in that order in a cyclic way, i.e. after class D, class 1 is visited. The long-run average cost to incur in class d is g^π_d. The gain of the MC for policy $^\pi$ is $g^\pi = \frac{1}{D}\sum_{d=1}^{D} g^\pi_d$.

The limit in (1.10) does not exist for periodic policies, as, when state $\mathbf{x}$ is in periodic class d, $\lim_{n\to\infty} \{\, V^\pi_{nD+1}(\mathbf{x}) - V^\pi_{nD}(\mathbf{x}) \,\} \;=\; g^\pi_d$, which is usually not equal to the gain g^π.

Therefore, no matter how large N is, $\mathbf{V}^\pi_N$ cannot act as an approximation of the value vector. In words, when a policy is periodic, the periodic class at which the system is at the end of the horizon depends on the class at which the horizon starts. Since the expected costs in a slot after many iterations is class dependent, a fair comparison of states based on $\mathbf{V}^\pi_N$ is only possible for states that are in the same class.

If states of different classes are to be compared, then one has to revert to a value vector that satisfies the Equations (1.6). We claim that for any policy π and for N sufficiently large, the vector

$$\mathbf{v}^\pi = \lim_{n\to\infty} \frac{1}{D}\sum_{d=0}^{D-1}\left(\mathbf{V}^\pi_{n+d} - (n+d)\cdot g^\pi \cdot \mathbf{1}\right). \tag{1.12}$$

can be used for this purpose. A proof will be given Appendix B. In fact, when N is sufficiently large, the vector $\displaystyle\frac{1}{D}\sum_{d=0}^{D-1}\mathbf{V}^\pi_{N+d} - k\cdot\mathbf{1}$ can be used for any constant k. When k is set to zero, an (improved) policy π' is set by:

$$\pi'(\mathbf{x}) = \arg\min_{a\in\mathcal{A}(\mathbf{x})} \left[\, \mathbb{E}C(\mathbf{x},a) \;+ \sum_{\mathbf{y}\in\mathcal{X}(\mathbf{x},a)} P_{\mathbf{x},\mathbf{y}}(a)\,\frac{1}{D}\sum_{d=0}^{D-1} V^\pi_{N+d}(\mathbf{y}) \,\right]. \tag{1.13}$$

The intuitive logic behind this definition is the following. A fair comparison of states in different periodic classes requires to consider each class as the last class of a planning horizon, independent of the initial class. Thus one needs to average over D successive horizon lengths. A formal derivation of Equation (1.12), as a correct definition of the relative value vector to compare states in periodic Markov chains is given in Appendix B.

1.3.3 Conclusions

In a one-step policy improvement approach one executes a single policy improvement step for a given initial policy π. The applicability of the algorithm relies on with whether one is able to find a policy π for which a vector, function or expression can be obtained that approximates the relative values of the states under π. If such a policy π with related relative values is found, an improvement over π is guaranteed [144], unless π itself is already optimal. Whether the improvement is substantial is however not guaranteed, but strongly depends on the problem and the selected policy π.

For the derivation of the relative values of a policy one may revert to an SA algorithm, but due to its computational complexity we may not be able to compute these values for problems with large states spaces. In some cases the state space can be decomposed but only when the problem and the initial policy are well-structured, such that the computational problem can be decomposed as well. We deliberate on the decomposition of multi-dimensional MDPs in Chapter 5. Although not guaranteed, a one-step policy improvement approach may yield a very good, or even nearly optimal strategy.