Delft University of Technology

Delft Center for Systems and Control

Technical report 04-020

Experience-based model predictive control using reinforcement learning*

R.R. Negenborn, B. De Schutter, M.A. Wiering, and J. Hellendoorn

If you want to cite this report, please use the following reference instead:

R.R. Negenborn, B. De Schutter, M.A. Wiering, and J. Hellendoorn, "Experience-based model predictive control using reinforcement learning," *Proceedings of the 8th TRAIL Congress 2004 – A World of Transport, Infrastructure and Logistics – CD-ROM*, Rotterdam, The Netherlands, 18 pp., Nov. 2004.

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.24.73 (secretary)
URL: https://www.dcsc.tudelft.nl

* This report can also be downloaded via https://pub.bartdeschutter.org/abs/04_020.html

Experience-based model predictive control using reinforcement learning

TRAIL Research School, Delft, November 2004

Authors

Drs. R.R. Negenborn*, Dr. ir. B. De Schutter*,
Dr. M.A. Wiering+, Prof. dr. ir. J. Hellendoorn*
* Delft Center for Systems and Control, Delft University of Technology
+ Institute of Information and Computing Sciences, Utrecht University

Contents

Abstract

Model predictive control (MPC) is becoming an increasingly popular method to select actions for controlling dynamic systems. Traditionally MPC uses a model of the system to be controlled and a performance function to characterize the desired behavior of the system. The MPC agent finds actions over a finite horizon that lead the system into a desired direction. A significant problem with conventional MPC is the amount of computations required and suboptimality of chosen actions.

In this paper we propose the use of MPC to control systems that can be described as Markov decision processes. We discuss how a straightforward MPC algorithm for Markov decision processes can be implemented, and how it can be improved in terms of speed and decision quality by considering value functions. We propose the use of reinforcement learning techniques to let the agent incorporate experience from the interaction with the system in its decision making. This experience speeds up the decision making of the agent significantly. Also, it allows the agent to base its decisions on an infinite instead of finite horizon.

The proposed approach can be beneficial for any system that can be modeled as Markov decision process, including systems found in areas like logistics, traffic control, and vehicle automation.

Keywords

Markov decision process, model predictive control, reinforcement learning

1 Introduction

Controlling Dynamic Systems In this paper we consider an agent that interacts with a controllable dynamic system at some discrete, low-level time scale. At each decision step the agent observes the state of the system, and based on this observation it chooses an action to perform on the system. The agent decides which action is selected based on a policy. The policy indicates what action the agent will make in each state.

The goal of the control agent is to find a policy that maps states to actions in such a way that the system behaves in the best way possible. In order for the agent to evaluate how well the system is behaving, there typically is some kind of performance function. This function indicates how well it is to make a certain state-action-state transition. The agent has to find a policy that chooses actions in states in such a way that the overall performance is maximized.

Model Predictive Control Over the last decades Model Predictive Control (MPC) has become an important technology for finding policies for complex, dynamic systems. MPC is popular mainly due to its easy way of integrating constraints on actions and states in the control of a system.

MPC has found wide application in the process industry. Recently, MPC has also started to be used in traffic control. Van den Berg et al. (2004) consider using MPC for controlling mixed urban and freeway traffic, where the objective is to make traffic in urban areas and on freeways smoothly interact with each other in such a way that total time spent in the network is minimized. Bellemans et al. (2002) consider MPC for control of ramp meter installations. Dunbar & Murray (2002) use MPC in a distributed setting to control multiple vehicles that have to drive in a specific formation. Hegyi (2004) considers using MPC to control different traffic measures at the same time: ramp metering and variable speed limits, route guidance and ramp metering, and ramp metering and main-stream metering.

As the name suggests, MPC is based on models that describe the behavior of a system. These models can be systems of difference or differential equations, hybrid models, or discrete-event models. In this paper we extend the application of MPC to systems that can be modeled as Markov decision processes, a subclass of discrete-event models. We consider how MPC can be used for this kind of systems and propose an experience-based extension for improving on-line computational costs of the MPC algorithm. This extension uses reinforcement learning to update the action selection policy on-line. The approach allows for system models to change gradually over time, results in fewer computations than conventional MPC, and improves decision quality by making decisions over an longer horizon.

Outline This paper is organized as follows. We start with an introduction to MPC in Section 2. We then discuss modeling systems as Markov decision processes in Section 3. We propose how MPC can be applied to these kind of models in Section 4. To improve computational and decision making performance we propose an extension to the MPC method for Markov decision processes that is based on reinforcement learning in Section 5.

2 Model Predictive Control

MPC (Camacho & Bordons (1995); García et al. (1989); Maciejowski (2002)) uses a model of the system and a model of the desired behavior to determine what actions to take. The system model is used to make predictions of the behavior of the system under various actions. The agent tries to find a sequence of actions that both bring the system in a desired state, and minimize negative effects of the actions. It considers a sequence of actions that avoids that any constraints on system states or actions are violated. By looking further ahead, the agent can anticipate possible constraint violations. In order to find the appropriate sequence of actions, MPC techniques have access to a model of desired behavior, in the form of a performance function. The performance function evaluates the preferability of being in a certain state and/or performing a certain action. Using these ingredients, the agent's task is to:

> Determine a sequence of actions based on predictions using the system model, that maximize the performance of the system in terms of the desired behavior model, while preventing violation of system and action constraints.

Thus, let us denote by r_k the performance or reward that the agent obtains at step k, by $a_0, \ldots, a_\infty$ the actions that the agent has to determine, and by E the expectancy operator taking the stochastic nature of the system into account. Then, we may write the task of the agent as solving the optimization problem:

$$\max_{a_0,\ldots,a_\infty} E\Big\{ \sum_{k=0}^{\infty} r_k \Big\}, \tag{1}$$

subject to the system model, the performance model, and constraints.

Basing actions on the predictions made by the system model introduces the issues of robustness and computational costs. MPC uses two principles to deal with these issues: the rolling horizon, and the finite horizon.

Rolling Horizon The problem of robustness is due to the fact that models are inherently inaccurate. A model is always an approximation of the system under consideration. Predictions about the behavior of the system become more and more inaccurate when considered further in the future. MPC techniques use a *rolling horizon* to increase robustness. The rolling horizon principle consists of synchronizing the state of the model with the state of the true system at every decision step. At every decision step the MPC agent observes the state of the true system, updates its model of the system and tries to find the best sequence of actions given the updated model. Typically the agent only executes the first action of this sequence. It then observes the system's state again and finds a new sequence of actions. Thus, the rolling horizon principle implements (1) as a sequence of optimization problems for each decision step k_0:

$$\max_{a_{k_0},\ldots,a_\infty} E\Big\{ \sum_{k=k_0}^{\infty} r_k \Big\}. \tag{2}$$

Finite Horizon The increase of robustness comes at the price of increased *computational costs*. Due to the rolling horizon, the MPC agent has to find a sequence of actions

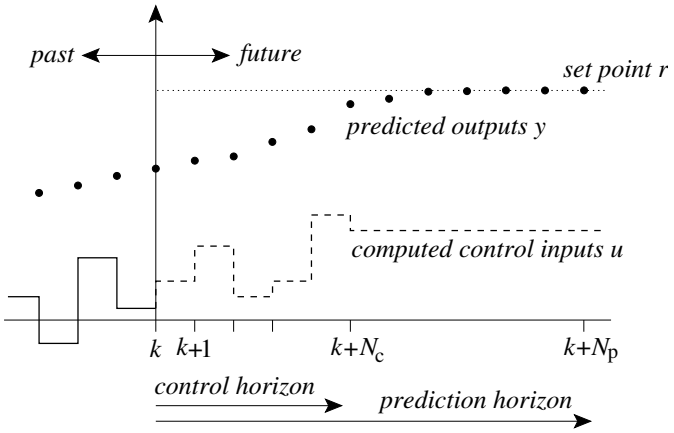

Figure 1: Example of Model Predictive Control.

at each decision step. This can be an intractable procedure when the horizon over which the agent has to find actions is infinite. Ideally the agent does consider an infinite horizon, since then the agent knows for sure that no constraints are violated. However, in practice considering an infinite horizon problem at each step is intractable for many applications. Therefore, conventional MPC techniques make the horizon finite by using a *control horizon*, a *prediction horizon*, and a *performance-to-go* part. The control horizon is the horizon over which the agent finds actions. The prediction horizon is the horizon over which the agent predicts the autonomous behavior of the system. The performance-to-go specifies the performance that the agent will obtain from the state at the end of the prediction horizon. Typically both horizons are much shorter than infinity, while the prediction horizon is larger than the control horizon. This is the case since in particular when considering systems that have autonomous behavior, actions are used to steer the system in a certain direction after which it can autonomously evolve further. By considering a prediction horizon that is larger than the control horizon, the agent can analyze where the system ends up after the agent has executed its actions over the control horizon. The finite horizon principle rewrites (2) as:

$$\max_{a_{k_0},\ldots,a_{k_0+N_c}} \left[E\left\{ \sum_{k=k_0}^{k_0+N_c} r_k \right\} + E\left\{ \sum_{k=k_0+N_c+1}^{k_0+N_p} r_k \right\} + V\left(x_{k_0+N_p+1}\right) \right], \tag{3}$$

where V is the performance-to-go function that indicates what the expected sum of future performance will be when in a certain state. In general this function is not known in advance. It may be assumed zero, approximated with a Lyapunov function (Jadbabaie et al. (1999)), or be learned from experience. We get back to this in Section 5.

Algorithm MPC is not one algorithm, but a class of algorithms that are based on the earlier described principles. The details of the implementations differ from each other depending on the structure of the system and performance models. In general, an agent employing MPC to determine its actions performs the following steps at each decision step k:

1. It rolls the horizon forward to the current decision step. That is, it measures the state of the true system and synchronizes its system model with this measurement. It formulates the optimization problem of finding the actions that give the

best performance over the control horizon considering the evolution of the system model over the prediction horizon, subject to constraints on system state and actions. In Figure 1, the desired behavior is indicated by the set points r, the control horizon by N_c, and the prediction horizon by N_p. The optimization problem consists of finding the actions from step k to $k+N_c$, such that at the end of the prediction horizon the system behavior y is close to the desired behavior r.

2. It solves the formulated optimization problem, often using general solution techniques (e.g., quadratic programming, sequential quadratic programming, ...), while taking into account constraints on actions and states. In Figure 1, the computed control inputs u are the actions found over the control horizon. After the control horizon the actions are assumed to be constant. Under this sequence of actions, the predicted outputs y approach the desired set points r.

3. It implements the actions found in the optimization procedure until the beginning of the next decision step. Typically this means that only one action is implemented before the horizon is rolled forward to step $k+1$.

Discussion MPC has found wide success in many different applications, mainly in the process industry. Advantages lie in the fact that the framework handles input and state constraints explicitly in a systematic way. This is due to the control problem formulation being based on the system model which includes the constraints. Also, MPC agents can operate without intervention for long periods. This is due to the rolling horizon principle, which makes that the agent looks ahead to prevent the system from going in the wrong direction. Finally, MPC agents adapt easily to new contexts due to the rolling horizon and require only few parameters to tune.

However, the use of MPC also has some disadvantages. The optimization problem of finding the sequence of actions can be of large size. In particular, when the control horizon over which actions are computed becomes larger, the number of variables of which the agent has to find the optimal value increases quickly. Also, the resources needed for computation and memory may be high, increasing more when the prediction horizon increases. The amount of resources required also grows with increasing system complexity. Finally, the feasibility of the solution to the overall control problem of the system is not guaranteed. Solutions to the problems considered over finite horizons do not guarantee solutions to the problem over the infinite horizon.

Research in the past has addressed these issues, resulting in conditions for feasibility and stability, using e.g. contracting constraints and classical stabilizing controllers at the end of the horizon. Most of the research has focused on computations carried out by one agent. In Negenborn et al. (2004) we survey how a distributed, multi-agent MPC setting can reduce the computations of a single MPC agent. In this paper we propose the use of experience to decrease the computational costs and improve decision making. However, first we propose the use of MPC to the class of systems that can be modeled as Markov decision processes.

3 Markov Decision Processes

As mentioned before, the MPC controller or agent has access to a model that describes the behavior of the system under actions and a model that describes the performance of the system and actions. Typically these models are systems of difference or differential equations, hybrid models, or discrete-event models. Discrete-event models are models in which transitions from one state to another occur with the execution or appearance of a specific action. Here we look at one type of discrete-event models: *Markov decision processes* (Puterman (1994)).

Assumptions Markov decision processes satisfy the *Markov property*, which states that the state evolution of the system only depends on the last state and the last action chosen. In other words, the transitions of the system are conditionally independent from actions and states encountered before the last decision step. To determine the next action, an agent uses its policy, which maps states to actions. No autonomous system behavior is assumed. Thus, the system only reacts in response to the action that the agent performed. After the action is performed the system stays in the new state until the next action occurs. The execution of each action indicates an event that transitions the system from one state to another.

We assume that the system evolves at discrete event steps $k = 0,1,2,3,\ldots$ At each step the system is in one out of a finite set of possible states X ($X = \{x^1, x^2, \ldots, x^N\}$). In each state $x \in X$ there is a finite set of actions A_x that the agent can perform ($A_x = \{a^1, a^2, \ldots, a^M\}$).

Typically Markov decision processes model stochastic systems. We assume that the system evolves according to a system model $\Sigma : P(x'|x,a)$, where $P(x'|x,a)$ denotes the probability of the system transitioning from state x to state x' after the agent has performed action a. The performance model is given by a function r, where $r(x,a,x')$ indicates the performance obtained by transitioning from state x to state x' under action a.

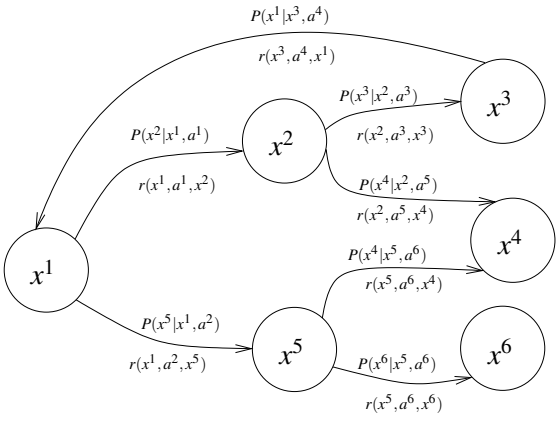

Figure 2: Example of a Markov decision process.

Graphical Representation Markov decision processes are intuitively represented as directed graphs. Figure 2 shows an example of the graphical representation of a Markov decision process. The representation consists of nodes and arcs. The nodes represent the possible states; the arcs represent transitions between states. Each arc corresponds to an action and is labeled with the probability that under the given action performed in the state represented by the node at the beginning of the arc, the system transitions to the node represented by the node at the end of the arc. The arcs are also labeled with the performance, indicating the performance obtained when the transition is made.

Constraints Constraints on system states and actions can be incorporated in Markov decision processes by restricting the set of possible states and by restricting the actions that are possible from individual states. Constraints can also be imposed by adjusting the performance function with highly negative performance for certain state-action-state transitions. In that case the agent will try to avoid those transitions as much as possible, since it is not interested in highly negative performance.

Example Systems Systems that can be modeled as Markov decision processes can be found in many fields. Markov decision processes have been used in ecology, economics, communications engineering, and also in the areas of traffic control, logistics, and transportation. The fields where Markov decision processes are applicable are characterized by uncertain state transitions and a necessity for sequential decision making. Let us look at examples from logistics and traffic control.

In logistics, Markov decision processes have been used, e.g., in the field of logistics for airline meal planning (Goto et al. (2004)). At several decision steps up to 3 hours before an airplane departs, the meal planner observes how many people will be flying and what they will be eating in the plane. Depending on that he adjusts the quantity of allocated meal. In the last couple of hours before departure, the adjustments of meal quantities are much higher and limited by the capacity of the van that has to deliver the meals directly to the plane. In this example the states of the Markov decision process consist of number of meals already allocated and the number of booked and stand-by passengers. The actions in each state consist of choices about how many extra meals to prepare or deliver. The direct performance for performing an action in a certain state depends on the meal costs, the return meal penalty, the van delivery charge, and the being late penalty. The transition probabilities of the system model give the likelihood of changes in the number of people scheduled to fly with the plane.

In traffic control Markov decision processes can be found, e.g., in the traffic signal control at a crossroad (Wiering (2000)). The state of the system can be the number of cars standing in front of the traffic signals. Actions that the agent can perform (in each state) consist of choosing traffic signal configurations. In local control context, the direct performance that the agent obtains may depend, e.g., on the average waiting time for each car. The lower this waiting time is, the better the performance. The transition probabilities of the system model depend on the number of cars that during a green signal period manage to leave the crossroad.

Further examples of the use of Markov decision processes deal with queuing networks, highway pavement repair, inventory control, knapsack problems, network control, and robotic motion.

4 MPC for Markov Decision Processes

In this section we propose the use of MPC for controlling models that can be modeled as Markov decision processes. Thus, we assume that we are given a system model and a performance model as described in the previous section. We can now implement the principles of MPC to come up with an MPC strategy for controlling systems described with such models. Recall that for finding a sequence of actions the MPC approach relies on control and prediction horizons that are rolling. Let us look at these elements.

4.1 Basic Approach

Rolling Horizon Similar to alternative approaches, the rolling horizon principle is easily included in the MPC for Markov decision processes by observing at each decision step the state of the true system and synchronizing the estimate that the agent has of the state of the system with this.

Finite Horizon Typically it is assumed that at the steps between the end of the control horizon and the prediction horizon the action stays the same. However, in our case the set of possible actions might change per state and this is thus not a reasonable approach. Also, since we do not consider systems that have autonomous behavior an action has to be chosen in each state of the system, otherwise there is no system evolution. Therefore we conclude that the control horizon should equal the prediction horizon in MPC for Markov decision processes.

Algorithm We now need to define the way in which the agent can find an optimal action sequences. The agent somehow has to use the system and performance model to find a sequence of N_c actions that gives the best performance over the control horizon. From the graphical viewpoint of Markov decision processes this comes down to finding the path of N_c steps that has the highest expected accumulated performance. A straightforward approach that achieves this simply considers all possible paths consisting of N_c actions. It sums all the accumulated performance and gives as result the sequence with highest accumulated performance. The performance-to-go V is for now assumed to be zero, as commonly done in MPC. These considerations lead us to the following straightforward algorithm for MPC for Markov decision processes:

1. Roll the horizon to the current decision step by observing the state of the system and define the optimization problem of finding the actions over the control horizon that maximize the performance starting from the observed state.

2. Find all paths of length N_c and accumulate the expected performance. Determine the sequence of actions that leads to the path with the highest accumulated expected performance.

3. Implement the first action of this sequence and move on to the next decision step.

Discussion The proposed MPC algorithm might suffer from the disadvantages discussed earlier for general MPC techniques. The amount of computational resources required to consider all paths over a length of the control horizon depends on N_c and the number of actions possible from each encountered state. In particular when there is a

very large number of actions from each state, it may be impossible to consider all paths. Also whether or not the system model or the performance model are deterministic or stochastic has influence on the speed at which the paths can be evaluated. Furthermore, because of the limited horizon over which actions are considered, the resulting policy may be suboptimal.

As a solution the control horizon might be chosen small, but this comes at the expense of larger sub-optimality. Alternatively we can make use of the performance-to-go function, which we will from now on refer to as *value function*, that for each state gives the accumulated future expected performance. Using the information from this function the computations required at each decision step may decrease significantly.

4.2 Value-Function Approach

Value Functions Value functions give the expected accumulated future performance for each state x. Obviously the future performance depends on the actions taken in the future, and since the actions taken in the future are chosen by a policy, value functions depend on policies. The *optimal* value function gives for each state the highest possible future performance from that state. This highest possible future performance is obtained by following the actions that an optimal policy prescribes. The optimal value function is obtained by solving:

$$V^*\big(x_{k_0}\big) = \max E\Big\{ \sum_{k=k_0}^{\infty} r_k \Big\}, \tag{4}$$

where r_k is the performance received for the transition at decision step k. The optimal value function V^* has the most accurate estimates of the future performance to be expected when in a certain state.

Assume that the optimal value function is known. From the graphical viewpoint of Markov decision processes, this means that we can label each node with the future expected performance. In that case, when the system is in state x, all the agent has to do is consider the actions $a \in A_x$ possible in state x and find the action that gives the best combination of directly obtainable performance plus expected accumulated performance from the state where the system will end up in after execution of action a. This quantity is called the Q value for the (x,a) pair. The agent uses the performance model to determine the direct performance gain for choosing an action and the system model to determine the successor state. The agent has to find the action that gives the highest Q value, i.e.,

$$a_k = \arg\max_{a \in A_{x_k}} \left[\sum_{x'} P(x'|x_k,a)\big(r(x_k,a,x') + V^*(x')\big) \right]. \tag{5}$$

Thus, when the optimal value function is known, instead of considering N_c steps, the procedure that the agent has to perform at each decision step reduces to a single one-step optimization procedure, i.e. the control horizon becomes $N_\mathrm{c} = 1$. Moreover, since we assumed an optimal value function, the chosen actions are optimal over the infinite horizon.

The problem is that in general neither optimal policies nor optimal value functions for the control of dynamic systems are known in advance. Thus, the question is how the value function can be computed. Since we are considering the infinite-horizon case,

the agent cannot simply compute the value function, since it cannot explicitly sum the performance over an infinite horizon. So instead it has to approximate the value function in some way. One way of approximating the value function is by use of a *discount factor*. This discount factor makes that the infinite sum of performances converges. We can employ dynamic-programming methods to find the value function in this case.

Dynamic Programming Dynamic-programming (Bellman (1957)) methods approximate the value function when discount factors are used. Given a policy, dynamic-programming methods compute the value function as:

$$V(x_{k_0}) = E\Big\{ \sum_{k=k_0}^{\infty} \gamma^{k-k_0} r_k \Big\}, \tag{6}$$

where $\gamma \in (0,1)$ is the discount factor. The closer γ is chosen to 1, the more long-term performance expectations are taken into account. The discount factor makes that performance received earlier is more important than performance obtained further in the future. We can rewrite the value function as:

$$\begin{aligned} V(x_{k_0}) &= E\Big\{ \sum_{k=k_0}^{\infty} \gamma^{k-k_0} r_k \Big\} \\ &= E\{r_{k_0} + \gamma r_{k_0+1} + \gamma^2 r_{k_0+2} + \ldots\} \\ &= E\{r_{k_0} + \gamma V(x_{k_0+1})\} \\ &= \sum_{a \in A_{x_{k_0}}} P_\Pi(x_{k_0}, a) \left[r(x_{k_0}, a, x') + \gamma \sum_{x'} P(x'|x_{k_0}, a) V(x') \right], \end{aligned} \tag{7}$$

where $P_\Pi(x,a)$ is the probability that the policy assigns to choosing action a in state x. Equation (7) is called the *Bellman* equation form of the value function. Bellman equations relate value functions recursively to themselves.

Methods that find the optimal value function treat the values of the optimal values of the states as unknowns. In that case a system of Bellman equations for all states forms a system of equations whose unique solution is the optimal value function (Sutton & Barto (1998)).

Algorithm With the value function we can define an MPC algorithm for Markov decision processes as follows:

1. Apply the rolling horizon principle by updating the state estimate with a measurement of the state.

2. Compute the value function given the latest system model.

3. Formulate the optimization problem over a control horizon of $N_c = 1$ of finding the action that brings the state of the system into the state with highest value. Solve the optimization problem.

4. Implement the found action and move on to the next decision step.

Discussion Computing the optimal value function at each decision step is too computationally expensive. Computing the optimal value function offline before the agent starts controlling the system (e.g., as in Bemporad et al. (2002)) reduces on-line computations, but has as disadvantage that the system cannot vary over time. The system should be time invariant. As mentioned before, models are inherently inaccurate. In MPC, typically deterministic models are assumed. In that case, modeled transition probabilities and true transition probabilities will not completely match. Also, these probabilities might change over time; some transitions might not even have been modeled at all. Although the rolling horizon provides some robustness, structural changes in parameters of the system model, e.g., transition probabilities, or changes in the performance model are not anticipated. Instead, we may update the value function on-line using experience from the interaction between the agent and the true system. In the following section we propose the use of reinforcement learning to update the value function on-line.

5 MPC with Reinforcement Learning

5.1 Reinforcement Learning

Since there is uncertainty in the system and performance model, we do not want to compute the value function once at the beginning. On the other hand, we also do not want to compute it at every decision step because this takes too many computations. We propose to combine the MPC for Markov decision processes introduced earlier with learning the value function on-line using reinforcement learning (Sutton & Barto (1998); Kaelbling et al. (1996); Wiering (1999)). In reinforcement learning we have a stochastic system of which we do not know the transition probabilities and the performance of taking actions in individual states. Since an agent wants to find a policy that gives the maximal cumulative expected future performance, it has to interact with the system, which implicitly contains the system model of which the agent requires information to determine the optimal value function.

In reinforcement learning experience is built up over time, instead of assumed available in *a priori* knowledge. The experience is based on the performance indications that give information about how well a certain action was in a certain state of the system. The experience is also based on the state transitions of the system under actions taken. In reinforcement learning the value function is approximated by keeping track of the performance obtained at each decision step considering the system state, performed action, and resulting system state. At each decision step the value function of the last decision step is updated with the experience built up over that last decision step. This experience, that is, the obtained performance indication and state observation, are samples of the value function that the agent wants to acquire. Each performance obtained by executing a certain action in a certain state leading to another state gives some information about the true value function. By obtaining sufficiently many of these samples the agent may accurately estimate the true value function.

Temporal-Difference Learning One of the methods that computes the value function without knowledge about the system or reward model using samples of interaction is Temporal-Difference (TD) learning (Sutton (1988)). TD learning uses current value estimates of successive states to update the value of the current state. Implicitly the assumption of time continuity is made: states visited in the same interval predict about the same outcome.

TD methods try to minimize the difference between value estimates of successive decision steps, explicitly using value estimates of successive states. For this, the TD method TD(0) uses only the current performance indication and the estimate of the value of the next state. TD(0) updates the value of a state as:

$$V(x_k) = V(x_k) + \alpha_{x_k}(r_k + \gamma V(x_{k+1}) - V(x_k)), \tag{8}$$

where α_{x_k} is a learning rate, and the term $e_k = r_k + \gamma V(x_{k+1}) - V(x_k)$ is defined as the TD(0)-error, which gives the error in the current value estimate of state x_k. The learning rate is used to be able to make the estimated value function converge by letting it decay over time when more experience has been built up.

The variance in the TD-errors gives an indication of the uncertainty in the value estimates. We may use this to determine when experience is sufficiently trustworthy.

The variance σ of the value estimate assigned with state x_k is computed as:

$$\sigma(x_k) = (r_k + \gamma V(x_{k+1}) - V(x_k))^2. \tag{9}$$

TD(λ) with Eligibility Traces TD(λ) learning generalizes TD(0) learning in that the λ parameter allows performance and value estimates further away in the future to be taken into account as well. The λ factor is a real number in the interval $[0,1]$, which weighs performance and value estimates further in the future exponentially less. With probability 1 value estimates converge to the true values for all λ, when each state is visited by infinitely many times and the learning rate diminishes to zero. To compute the TD(λ)-errors we need to know the TD(0)-errors of all future steps. For an infinite horizon it is impossible to know these. Instead, we incrementally update the value estimates as new TD(0)-errors become available using eligibility traces (Barto et al. (1983)). Eligibility traces indicate how much a state is eligible to learn from a new TD(0)-error. How much this is depends on λ, the recency of the state appearance, and the frequency of the state appearance. It can be shown that the update of the value of a state using a performance indication received a number of steps in the future is

$$\Delta V(x) = \alpha(x) e_k l_k(x), \tag{10}$$

where $l_k(x)$ represents the accumulating eligibility trace for x, which can recursively be implemented as:

$$l_{k+1}(x) \leftarrow \lambda \gamma l_k(x) \qquad \text{if } x_k \neq x \tag{11}$$

$$l_{k+1}(x) \leftarrow \lambda \gamma l_k(x) + 1 \qquad \text{if } x_k = x. \tag{12}$$

Instead of using accumulating eligibility traces, we may also use *replacing traces*, in which case the eligibility for a state is simply reset to 1 when it is encountered, instead of incremented by 1.

5.2 MPC with TD(λ) Algorithm

We now combine the MPC for Markov decision processes with learning the value function. In the beginning the agent has no experience to rely on, so it should not rely on its value function. Instead the agent should fully rely on its MPC decisions. Later on it gains more experience and the decisions that it can make based on the value function it learned will become more trustworthy. In the extreme case, the agent has gained so much experience with the system that it should fully rely on the experience.

Competitive Approach As a first approach to combine MPC with learning we could consider a competitive situation between the MPC and the learning part of the agent. These two parts could compete over which of them gets to make the decision about which action the agent takes. In the beginning, the learning part will have a difficult time competing with the MPC part, since it has no experience at all with controlling the system. Over time though the learning part does get more experienced and at some point the learning part might propose actions to take that are better than those suggested by the MPC part. At that point the agent can either switch to completely using the experience-based decisions and not using the MPC part anymore, or it can at each decision step decide on which of the two to rely on. It may decide on which of the two it should rely by keeping track of the quality of earlier actions proposed by the two.

Collaborative Approach Alternatively, instead of seeing the decision making as a competition between the two parts we can see it as a collaboration between the parts. The MPC part provides basic robustness and decision making over the relatively short term, while the learning part provides experience and decision making over the long term.

The agent may gradually incorporate the experience built up in the value function in its decision making. In the beginning it should still not rely too much on the learned value function since it has not gathered much experience with the system yet. The uncertainty in the value estimates, that is, in the estimates of the expected accumulated future performance is high. Over time the agents builds up more experience, resulting in value estimates with higher accuracy. In that case the agent should use its experience in its decision making. In the algorithm we propose next the agent uses accurate value estimates to decrease the control horizon over which MPC computes paths, thus speeding up decision making, while increasing decision quality.

In our first algorithm for Markov decision processes (on page 7) the agent computed the accumulated performance for each path over the control horizon of length N_c. We then ignored the value (or, performance-to-go) of the resulting final path state. Now however we can incorporate the value function that is being learned in this process. When the agent is considering a certain path of length N_c it can at each step in that path consider the value of the state associated with the step in that path. If the uncertainty in the value of that state is low, then the agent can stop considering the rest of the path and instead take the given value as estimate of the obtainable future accumulated performance. We can do this since the value of that state indicates the expected accumulated future performance starting from that state.

The outline of the algorithm we propose is as follows:

1. The agent rolls the horizon of the last step forward to the current step k, measuring the current state of the system.

2. The MPC part of the agent considers each path of length N_c starting from the measured state. It finds the path of state-action-reward-states (x,a,r,x') that gives the maximal accumulated performance over the control horizon. While considering the paths, the agent may encounter a state that has a value estimate associated with it with uncertainty below some threshold. This estimate indicates the expected accumulated future performance from that state onwards, and thus the MPC part can use that value as indication for the expected accumulated future performance. It need not consider any further steps on that path.

3. The learning part of the agent incorporates the (x,a,r,x') samples generated by the MPC part of the agent. The samples generated by the MPC part are predictions about the behavior of the system and predictions about what is optimal to do considering the control horizon. The learning part uses these samples as idealized experience. It incorporates all generated performance indications in its value function, thereby decreasing the uncertainty in the values. The learning part will incorporate the predicted samples from the starting state to the state of which the value is used of the path that has maximal total performance. Note that since the agent uses a value estimate only when the uncertainty in it is below some threshold, it does not matter how the values are initialized.

4. The agent implements the first action in the sequence determined and moves on to the next decision step.

5.3 Discussion

The described algorithm has some attractive features. First of all, once the value function is computed with high enough accuracy, the computationally intensive MPC optimizations over the full control horizon using the system and performance model can be reduced to an optimization over one step using the system model and value function. Besides that, in the second phase of operation, the decisions the agent makes are based on an infinite horizon, since the values of the states represent the expected accumulated performance over the full future. This makes the actions chosen closer to optimal.

Also, the approach allows for the system and desired performance to change over time. In particular when we consider systems that have a long lifetime this is an advantage. The performance model can be updated at each decision step with some identification method. The MPC agent will then generate samples using these new rewards, and the learning part will incorporate these samples and slowly adjust to the new performance. The system model may also change over time and be updated at each decision step. The transition probabilities can change over time, e.g., due to aging. By monitoring the actual transition from a state to another under a certain action, a system model can be learned. This model will adapt to changing system behavior. The MPC part can use this model learned to compute its paths. The problem is that the value function is implicitly based on the system model. However since TD methods do not need a system model they will adjust to the new situation.

Finally, the proposed approach still acknowledges the constraints in the original model. Moreover, if constraints are violated, the reinforcement learning part might obtain a highly negative feedback. This will learn the agent that in the future it should avoid the same situation.

6 Conclusion

In this paper we have considered model predictive control for Markov decision processes. We have proposed a straightforward algorithm for these kind of models. However, this algorithm might suffer from too high computational requirements and suboptimality. As a first solution to this we have considered the use of value functions. Once value functions are known well, the computational requirements reduce to an MPC problem with a control horizon of only length one. At the same time, the decisions are based on infinite-horizon information. The problem is that in general the value function is not known well enough *a priori*. In this paper we have considered the use of experience to incrementally learn the value function over time. Using the reinforcement learning method called TD(λ) learning the agent can incorporate experience built up through interaction with the system it has to control. In this way it will over time get a sufficiently good estimate of the values of states. Once it has built up enough experience the agent can fully rely on it, thus reducing the computations required to do the MPC for Markov decision processes without learning. Additional advantages of the proposed approach lie in the fact that the agent will adapt to changing system characteristics or performance characteristics. The parameters of the performance function or system under control may slowly change over time. Since the agent can continuously incorporate newly gained experience, it will adapt to these changes.

We close with some remarks and future research directions. In this paper we have considered TD(λ) learning for finite Markov decision processes. To deal with high dimensional continuous action and state spaces we can use a subclass of reinforcement learning methods called actor-critic methods (Sutton & Barto (1998)). We also remark that in this paper we have silently assumed that we can store the value function in a table. In this way we have an explicit representation of the value function. Sometimes we may not be able to make an explicit representation and may have to use function approximators to make an implicit representation of the value function. Note that MPC may still be combined fruitfully with reinforcement learning using function approximators (Sutton & Barto (1998)). Future research directions consist of considering alternative ways to decide when the agent has gained enough experience to switch from relying on the MPC decisions to the decisions learned. Furthermore, experiments need to be implemented to further investigate and show the potential of the proposed experience-based MPC for Markov decision processes.

Acknowledgments

This research was supported by project “Multi-agent control of large-scale hybrid systems” (DWV.6188) of the Dutch Technology Foundation STW, Applied Science division of NWO, the Technology Programme of the Dutch Ministry of Economic Affairs, and by the TU Delft spearhead program “Transport Research Centre Delft: Towards Reliable Mobility”.

References

Barto, A. G., R. S. Sutton, C. W. Anderson (1983) Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man, and Cybernetics*, 13, pp. 834–846.

Bellemans, T., B. De Schutter, B. De Moor (2002) Model predictive control for ramp metering of motorway traffic: A case study, Tech. Rep. CSE02-022, Control Systems Engineering, Fac. of Information Technology and Systems, Delft University of Technology, Delft, The Netherlands, provisionally accepted for publication in *Control Engineering Practice*.

Bellman, R. (1957) *Dynamic Programming*, Princeton University Press, Princeton, New Jersey.

Bemporad, A., M. Morari, V. Dua, E. Pistikopoulos (2002) The explicit linear quadratic regulator for constrained systems, *Automatica*, 38(1), pp. 3–20.

Camacho, E., C. Bordons (1995) *Model Predictive Control in the Process Industry*, Springer-Verlag, Berlin, Germany.

Dunbar, W. B., R. M. Murray (2002) Model predictive control of coordinated multi-vehicle formations, in: *Proceedings of the 41st IEEE Conference on Decision and Control*, Las Vegas, Nevada, pp. 4631–4636.

García, C., D. Prett, M. Morari (1989) Model predictive control: Theory and practice — A survey, *Automatica*, 25(3), pp. 335–348.

Goto, J. H., M. E. Lewis, M. L. Puterman (2004) Coffee, tea, or ...?: A Markov decision process model for airline meal provisioning, *Transportation Science*, 38, pp. 107–118.

Hegyi, A. (2004) *Model Predictive Control for Integrating Traffic Control Measures*, Ph.D. thesis, TRAIL Thesis Series T2004/2, Delft University of Technology, Delft, The Netherlands, `http://www.dcsc.tudelft.nl/~ahegyi/thesis/`.

Jadbabaie, A., J. Yu, J. Hauser (1999) Stabilizing receding horizon control of nonlinear systems: a control Lyapunov function approach, in: *Proceedings of the 1999 American Control Conference*, San Diego, California, pp. 1535–1539.

Kaelbling, L. P., M. L. Littman, A. W. Moore (1996) Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, 4, pp. 237–285.

Maciejowski, J. M. (2002) *Predictive Control with Constraints*, Prentice Hall, Harlow, England.

Negenborn, R. R., B. De Schutter, J. Hellendoorn (2004) Multi-agent model predictive control: A survey, Tech. Rep. 04-010, Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands.

Puterman, M. L. (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New York.

Sutton, R., A. Barto (1998) *An Introduction to Reinforcement Learning*, MIT Press, Cambridge, Massachusetts.

Sutton, R. S. (1988) Learning to predict by the methods of temporal differences, *Machine Learning*, 3, pp. 9–44.

Van den Berg, M., B. De Schutter, A. Hegyi, J. Hellendoorn (2004) Model predictive control for mixed urban and freeway networks, in: *Proceedings of the 83rd Annual Meeting of the Transportation Research Board*, Washington, D.C.

Wiering, M. (2000) Multi-agent reinforcement learning for traffic light control, in: *Proceedings of the Seventeenth International Conference on Machine Learning*, Stanford, California, pp. 1151–1158.

Wiering, M. A. (1999) *Explorations in Efficient Reinforcement Learning*, Ph.D. thesis, University of Amsterdam, The Netherlands.