LEARNING-BASED MODEL PREDICTIVE CONTROL FOR MARKOV DECISION PROCESSES

Rudy R. Negenborn [*,1] **Bart De Schutter** [*]
Marco A. Wiering [**] **Hans Hellendoorn** [*]

[*] *Delft Center for Systems and Control*
Delft University of Technology, Delft, The Netherlands
[**] *Institute of Information and Computing Sciences*
Utrecht University, Utrecht, The Netherlands

Abstract: We propose the use of Model Predictive Control (MPC) for controlling systems described by Markov decision processes. First, we consider a straightforward MPC algorithm for Markov decision processes. Then, we propose value functions, a means to deal with issues arising in conventional MPC, e.g., computational requirements and sub-optimality of actions. We use reinforcement learning to let an MPC agent learn a value function incrementally. The agent incorporates experience from the interaction with the system in its decision making. Our approach initially relies on pure MPC. Over time, as experience increases, the learned value function is taken more and more into account. This speeds up the decision making, allows decisions to be made over an infinite instead of a finite horizon, and provides adequate control actions, even if the system and desired performance slowly vary over time.

Keywords: Markov decision processes, predictive control, learning.

1. INTRODUCTION

Over the last decades Model Predictive Control (MPC) has become an important technology for finding control policies for complex, dynamic systems, as found in, e.g., the process industry (Camacho and Bordons, 1995; Morari and Lee, 1999). As the name suggests, MPC is based on models that describe the behavior of a system. Typically, these models are systems of difference or differential equations. In this paper we consider the application of MPC to systems that can be modeled by Markov decision processes, a subclass of discrete-event models. Moreover, we propose a learning-based extension for reducing the on-line computational cost of the MPC algorithm, using reinforcement learning to learn expectations of performance 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 infinite horizon.

We consider an agent controlling a dynamic system at discrete decision steps. At each decision step, the agent observes the state of the system and determines the next action to take based on the observation and a policy. A policy maps states to actions and it is the agent's task to determine a policy that makes the system behave in an optimal way.

This paper is organized as follows. We introduce conventional MPC in Section 2. Then we propose MPC for systems that can be modeled by Markov decision processes in Section 3. We consider the use of value functions in MPC in Section 4. To improve computational and decision making performance we improve the method with reinforcement learning in Section 5.

[1] Corresponding author, e-mail: r.negenborn@dcsc.tudelft.nl

2. MODEL PREDICTIVE CONTROL

MPC (Camacho and Bordons, 1995; Morari and Lee, 1999; Maciejowski, 2002) is a model-based control approach that has found successful application, e.g., in the process industry. In MPC, a control agent uses a system model to predict the behavior of a system under various actions. The control agent finds a sequence of actions that bring the system in a desired state, while minimizing negative effects of the actions, and taking constraints into account. In order to find the sequence of appropriate actions, the control agent uses a performance function. This performance function evaluates the preferability of being in a certain state and performing a certain action by giving rewards. Let us denote by r_k the reward given by the performance function at decision step k, by $a_0, \ldots, a_\infty$ the actions to be determined by the agent, and by E the expectancy operator taking the system uncertainty into account. We may then 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 function, and the constraints.

Basing actions on the model predictions introduces issues with robustness due to the fact that models are inherently inaccurate and thus predictions further in the future are more and more uncertain. To deal with this, MPC uses a *rolling* or *receding horizon*, which involves reformulating the optimization problem at each decision step using the latest observation of the system state. However, the rolling horizon increases computational costs, since at each decision step a sequence of actions has to be determined to make sure no constraints are violated. In practice this is intractable for many applications. To reduce computational costs, MPC uses a *control horizon*, a *prediction horizon*, and a *performance-to-go*. The control horizon determines the number of actions to find. The prediction horizon determines how far the behavior of the system is predicted. The performance-to-go gives the sum of the reward obtained from the state at the end of the prediction horizon until infinity under a certain policy. With these principles (1) can be rewritten as:

$$\max_{a_{k_0},\ldots,a_{k_0+N_\mathrm{c}}} \Big[E\Big\{\sum_{k=k_0}^{k_0+N_\mathrm{c}} r_k\Big\} + E\Big\{\sum_{k=k_0+N_\mathrm{c}+1}^{k_0+N_\mathrm{p}} r_k\Big\} + V\big(x_{k_0+N_\mathrm{p}+1}\big)\Big], \tag{2}$$

where V is the performance-to-go function, indicating the expected sum of future rewards when in a certain state. In general the performance-to-go 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, as we shall discuss in Section 5.

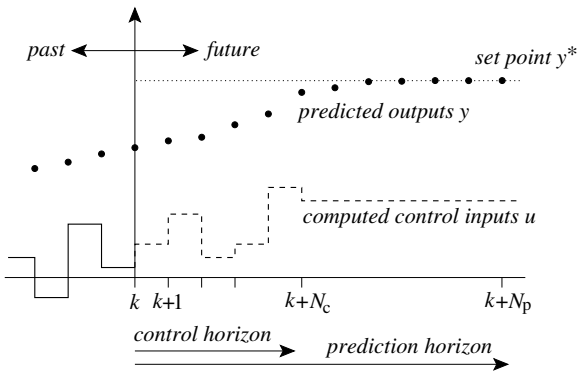

Fig. 1. Example of conventional MPC. The control problem is to find actions u_k to u_{k+N_c}, such that after N_p steps the system behavior y approaches the desired behavior y^*. In this example, y indeed reaches the desired set point y^*.

Implementation details of (2) depend on the structure of the system model and performance function. In general, MPC methods have the following scheme (see Figure 1):

(1) The horizon is moved to the current decision step k_0 by observing the state of the true system and reformulating the optimization problem of (2) using the observed state as initial state x_{k_0}.
(2) The formulated optimization problem is solved, often using general solution techniques (e.g., quadratic programming, sequential quadratic programming, ...). The optimization problem is solved taking into account constraints on actions and states.
(3) Actions found in the optimization procedure are executed until the next decision step. Typically only one action is performed.

Advantages of MPC lie in the explicit integration of input and state constraints. Due to the rolling horizon MPC adapts easily to new contexts and can be used without intervention for long periods. Moreover, only few parameters need to be tuned, i.e., the prediction and control horizon. However, the optimization problem may still require too many computations, e.g., when the control horizon becomes large. Resources required for computation and memory may be high, increasing more when the prediction horizon or system complexity increases. Besides that, solutions to the finite horizon problems do not guarantee solutions to the problem over the infinite horizon.

Research in the past has addressed these issues for conventional MPC, typically using models that are systems of difference or differential equations. In the following sections we propose MPC for systems modeled by Markov decision processes and consider improving speed and decision quality using the performance-to-go function and experience.

3. MPC FOR MARKOV DECISION PROCESSES

3.1 Markov Decision Processes

Markov decision processes (Puterman, 1994) are applicable in fields characterized by uncertain state transitions and a necessity for sequential decision making, e.g., robot control, manufacturing, and traffic signal control (Wiering, 2000). Markov decision processes satisfy the *Markov property*, stating that state transitions are conditionally independent from actions and states encountered before the current decision step. An agent can therefore rely on a policy that directly maps states to actions to determine the next action. After execution of an action, the system is assumed to stay in the new state until the next action, i.e., the system has no autonomous behavior. Figure 2 shows the graph representation of some Markov decision process.

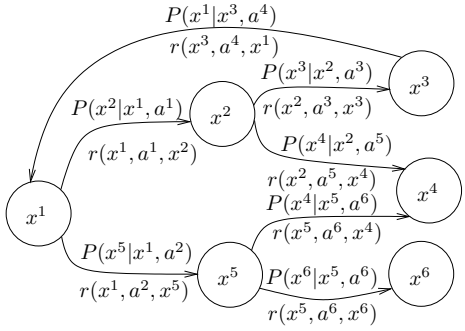

Fig. 2. Example of a Markov decision process. A node represents a state. An arc represents a transition from one state to another under a certain action. An arc is labeled with a transition probability and a reward obtainable under the transition.

We use k as counter that indicates the decision step. At each step the system is in one out of a finite set of states $X = \{x^1, x^2, \ldots, x^N\}$. In each state $x \in X$ there is a finite set of actions A_x that the agent can perform ($A_x = \{a^1, a^2, \ldots, a^{M_x}\}$). The system evolves according to system model $\Sigma : P(x'|x,a)$, where $P(x'|x,a)$ is the probability of transitioning from state x to state x' after action a is performed. The performance function is given by r, where $r(x,a,x')$ is the reward obtained with the transition from state x to state x' under action a.

Constraints can be included explicitly by restricting actions and reachable states, or implicitly by imposing a highly negative reward for certain transitions; as we will see, the agent will try to avoid these transitions.

As an example, in local traffic signal control at an intersection, a state can consist of the number of cars in front of the traffic signals. Actions in each state consist of traffic signal configurations. Transition probabilities may depend on the number of cars leaving the crossroad during a green signal. Rewards may depend on the average waiting time, with lower waiting time indicating higher reward. Constraints on actions consist of admissible, safe, traffic signal configurations.

3.2 Straightforward MPC Approach

Let us consider the straightforward application of MPC to Markov decision processes. Similar to alternative approaches, the rolling horizon principle is easily included by letting the agent synchronize at each decision step its current estimate of the system state with a new observation of the system state. The control horizon should equal the prediction horizon, since the systems we consider have no autonomous behavior and the set of possible actions can change per state. Therefore, as is usually assumed in conventional MPC, assuming constant actions between the end of the control horizon and the prediction horizon is not reasonable in our case.

The agent uses the Markov decision process 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 reward. This yields the following straightforward MPC algorithm for Markov decision processes:

(1) Roll the horizon to the current step by observing the state of the system. Define the optimization problem of finding the actions over the control horizon that maximize the sum of the rewards starting from the observed state.
(2) Find all paths of length N_c and accumulate the rewards. Determine the sequence of actions that leads to the path with the highest accumulated reward.
(3) Implement the first action of this sequence and move on to the next decision step.

The proposed MPC algorithm can 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 intractable 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. This is in particular the case since we ignored the performance-to-go V, as is commonly done in conventional MPC.

As a solution we can take a small control horizon. However, this may result in increased sub-optimal decision making, in particular when we keep ignoring the performance-to-go. In the following we will not ignore this performance indicator. We will from now on refer to the performance-to-go as *value function*, and use the information from this value function to improve the computations required at each step.

4. MPC WITH VALUE FUNCTIONS

4.1 Value Functions

A value function V gives the expected accumulated future reward for each state x and a policy π. The optimal value function V^* gives the highest possible expected accumulated future reward for each state. This highest possible future reward is obtained by following the actions that an optimal policy π^* prescribes [2]. Whereas in previous sections we considered a deterministic policy, from now on we consider a probabilistic policy. The optimal value function V^* is then obtained by solving for each x_{k_0}:

$$V^*(x_{k_0}) = \max_{\pi} E\Big\{ \sum_{k=k_0}^{\infty} r(x_k, \pi(x_k), x_{k+1}) \Big\}.$$

Assume the optimal value function is known. From the graphical viewpoint of Markov decision processes, we can label each node with a value, or expected accumulated future reward. In that case, the agent has to consider only the actions $a \in A_x$ possible in current state x and find the action that gives the highest sum of directly obtainable reward plus expected accumulated future reward of the resulting state after the action would have been executed. This sum, called the Q value for the (x, a)-pair, is used by the agent to find the action that gives the highest Q value as follows:

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

Thus, when the optimal value function is known, instead of considering N_c steps, the agent has to consider only a one-step optimization procedure at each decision step, i.e., the control horizon becomes $N_c = 1$. Moreover, since the value function is optimal over the infinite horizon, also the chosen actions are optimal over the infinite horizon.

In general neither optimal policies nor optimal value functions are known in advance. In our case, value functions cannot be computed easily in a straightforward way, since the reward over an infinite horizon cannot be summed explicitly. Instead, the value function can be approximated. Dynamic-programming methods (Bellman, 1957) use one way of approximating the value function. Dynamic-programming methods approximate the value function by introducing a *discount factor*. This discount factor lets the infinite sum of rewards converge. Using a discount factor, the value function is approximated as:

$$V^{\pi}(x_{k_0}) = E\Big\{ \sum_{k=k_0}^{\infty} \gamma^{k-k_0} r(x_k, \pi(x_k), x_{k+1}) \Big\}, \tag{3}$$

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 value function (3) can be written as:

$$V^{\pi}(x_{k_0}) = \sum_{a \in A_{x_{k_0}}} P_{\pi}(a|x_{k_0}) \times \Big[r(x_{k_0}, a, x') + \gamma \sum_{x'} P(x'|x_{k_0}, a) V(x') \Big],$$

where $P_{\pi}(a|x)$ is the probability that the policy π will select action a in state x. This kind of equation is called a Bellman equation. Dynamic-programming methods 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 and Barto, 1998).

4.2 Value-Function MPC Approach

Using the value function we can formulate a new MPC algorithm for Markov decision processes as follows:

(1) Apply the rolling horizon principle, 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 the highest value. Solve the problem.
(4) Implement the found action and move on to the next decision step.

The advantage of this approach is that the control horizon is only of length one. Moreover, by using the most up-to-date system model to compute the value function at each decision step, actions are adequate, even in the event of (slowly) changing system and performance desires.

However, computing the optimal value function at each decision step can computationally be very expensive. Computing the optimal value function off-line before the agent starts controlling the system (e.g., as done in (Bemporad *et al.*, 2002) for linear systems) reduces on-line computations, but does not allow for the system to vary over time. Although the rolling horizon provides some robustness, structural changes in parameters of the system model are not anticipated.

Instead of recomputing the value function at each decision step, we could update the value function on-line using experience from the interaction between the agent and the true system. We propose to combine MPC for Markov decision processes with learning the value function on-line using reinforcement learning. This way, system changes are anticipated on-line while not computing the value function at every decision step.

[2] For the sake of simplicity we assume a unique optimal policy. Extension to the non-unique case is straightforward by choosing one of the optimums.

5. MPC WITH REINFORCEMENT LEARNING

5.1 Reinforcement Learning

In reinforcement learning (Sutton and Barto, 1998; Kaelbling *et al.*, 1996; Wiering, 1999) both the model of the stochastic system and the desired behavior are unknown *a priori*. To determine a policy, the agent incrementally computes the value function based on performance indications and interaction with the system, which implicitly contains the system model. At each decision step the value function of the last decision step is updated with the newly gained experience consisting of a state-action-state transition and reward. By obtaining sufficiently many experiences the agent can accurately estimate the value function.

In Temporal-Difference (λ) learning (TD(λ)) (Sutton, 1988) the difference between value estimates of successive decision steps is minimized, explicitly using value estimates of successive states. The parameter $\lambda \in [0, 1]$ weighs reward and value estimates further away in the future exponentially less. With probability 1 value estimates can be guaranteed to converge to the true values for all λ (Sutton, 1988).

TD(λ) learning uses eligibility traces to incrementally learn the value function, which we assume initially contains arbitrary (finite) values. The value of a state depends on the values of successor states. Therefore, the value update of a state also depends on the value updates of successive states. In fact, to compute the update for a state, all future updates need to be known, which is impossible for the infinite-horizon case. Instead, values can be updated incrementally as new updates become available using eligibility traces (Barto *et al.*, 1983). These traces indicate the amount a state is eligible to learn from new experience. This depends on λ, the recency of the state appearance, and the frequency of the state appearance. The update $\Delta V^l(x)$ of the learned value of a state using a reward received in the future can be shown to be:

$$\Delta V^l(x) = \alpha(x) e_k l_k(x)$$

where $\alpha(x)$ is a suitable learning rate, which can guarantee convergence; error $e_k = r_k + \gamma V^l(x_{k+1}) - V^l(x_k)$ indicates for a state the difference between the previously learned value $V^l(x_k)$ and the sampled value based on the obtained reward r_k and the previously learned value $V^l(x_{k+1})$ for the successor state; $l_k(x)$ represents the accumulating eligibility trace for x, which is initially zero and can recursively be updated as:

$$\begin{aligned} l_{k+1}(x) &\leftarrow \lambda\gamma l_k(x) && \text{if } x_k \neq x \\ l_{k+1}(x) &\leftarrow \lambda\gamma l_k(x) + 1 && \text{if } x_k = x. \end{aligned}$$

The uncertainty in the update can be computed using the error e_k. For the case $\lambda = 0$ the uncertainty (or variance) in the update is $\sigma_k^2 = e_k^2$. More general results on error bounds for TD learning are reported in (Kearns and Singh, 2000).

5.2 TD-MPC Approach

We consider a collaborative approach in which MPC provides basic robustness and decision making over the relatively short term, while learning provides robustness, adaptation, and decision making over the long term. The agent gradually incorporates the learned value function in its decision making as experience increases. Initially uncertainty in the value estimates is high, so it will just use MPC. Samples generated by the MPC part are predictions about the behavior of the system and predictions about what is optimal to do over the control horizon. Learning uses the samples as idealized experience, incorporating them in its value function. Over time the uncertainty in the value estimates decreases. When the uncertainty is below a threshold, the agent uses the value estimates, thereby decreasing the control horizon over which MPC computes paths. Since the agent uses a learned value only when the uncertainty in it is below a threshold, values can be initialized to any finite value. We propose the following algorithm:

(1) Roll the horizon to the current step k.
(2) For each path of N_c (x, a, r, x') 4-tuples starting from the current state, consider each state. If the uncertainty in the value estimate of an encountered state is below a threshold, use the value plus reward summed over earlier steps in that path as indication for the expected accumulated future reward, and stop considering the path. Else, add the given reward to the summed reward over earlier steps in the path and move to the next state.
(3) Incorporate the (x, a, r, x')-samples created by MPC in the value function as experience using TD learning and reduce the uncertainty in the value estimates.
(4) Implement the first action in the sequence determined and move to the next decision step.

The described algorithm has some attractive features. Once the value function is computed with high enough accuracy, the computationally intensive MPC optimizations over the full control horizon using the system model and the performance function are reduced to a one-step optimization using the system model and the value function. Moreover, using the experience, the decisions are based on an infinite horizon, since values of states represent expected accumulated reward over the full future. Constraint violations are thus anticipated better.

The agent will propose adequate actions, even if the system and desired performance slowly vary over time. In particular for systems with a long lifetime this is an advantage. The system model and performance function can be updated at each decision step. The agent will then generate samples using these updated models, and the learning part will incorporate these samples and adjust to the new situation.

6. CONCLUSIONS & FUTURE RESEARCH

In this paper we have considered Model Predictive Control (MPC) for Markov decision processes. We have first considered a straightforward algorithm for these kind of models. To deal high computational requirements and sub-optimality issues, we have proposed the use of the performance-to-go or value function. With optimal value functions the MPC control horizon becomes length one. Speed is increased, while decisions are based on infinite-horizon predictions.

In general however, optimal value functions are not known *a priori*. In this paper we have considered using experience to incrementally learn value functions over time. With reinforcement-learning methods like temporal-difference learning the agent incorporates experience built up through interaction with the system. It can over time get a good estimate of the value function. Once sufficient experience has been obtained, the agent uses this to its fullest, requiring less computations than the non-learning approach.

An additional advantage of the proposed approach lies in that the agent adapts to changing system and performance characteristics. The performance function or system under control may slowly change over time. Since the agent incorporates newly gained experience at each decision step, it will adapt to these changes and still produce adequate actions.

We note that 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 actor-critic methods (Sutton and Barto, 1998). Moreover, in this paper we have silently assumed an explicit tabular value-function representation. If an explicit representation is not available, we may use an implicit representation, e.g., a function approximator (Sutton and Barto, 1998). MPC may then still be combined fruitfully with learning.

Future research directions consist of considering alternative ways to include the uncertainty in the gained experience in the decision making. Also, accuracy bounds and comparisons with alternative adaptive and learning control approaches can be made. Furthermore, experiments need to be implemented to further investigate and show the potential of the proposed learning-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, the TU Delft spearhead program "Transport Research Centre Delft: Towards Reliable Mobility", and the European 6th Framework Network of Excellence "HYbrid CONtrol: Taming Heterogeneity and Complexity of Networked Embedded Systems (HYCON)".

REFERENCES

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

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

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

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

Jadbabaie, A., J. Yu and 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 and A. W. Moore (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* **4**, 237–285.

Kearns, M. and S. Singh (2000). Bias-variance error bounds for temporal difference updates. In: *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*. Stanford, California. pp. 142–147.

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

Morari, M. and J. H. Lee (1999). Model predictive control: past, present and future. *Computers and Chemical Engineering* **23**, 667–682.

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

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

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

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. PhD thesis. University of Amsterdam. The Netherlands.