Implementation of Dynamic Programming for n-Dimensional Optimal Control Problems With Final State Constraints

Philipp Elbert, Soren Ebbesen, and Lino Guzzella

***Abstract*— Many optimal control problems include a continuous nonlinear dynamic system, state, and control constraints, and final state constraints. When using dynamic programming to solve such a problem, the solution space typically needs to be discretized and interpolation is used to evaluate the cost-to-go function between the grid points. When implementing such an algorithm, it is important to treat numerical issues appropriately. Otherwise, the accuracy of the found solution will deteriorate and global optimality can be restored only by increasing the level of discretization. Unfortunately, this will also increase the computational effort needed to calculate the solution. A known problem is the treatment of states in the time–state space from which the final state constraint cannot be met within the given final time. In this brief, a novel method to handle this problem is presented. The new method guarantees global optimality of the found solution, while it is not restricted to a specific class of problems. Opposed to that, previously proposed methods either sacrifice global optimality or are applicable to a specific class of problems only. Compared to the basic implementation, the proposed method allows the use of a substantially lower level of discretization while achieving the same accuracy. As an example, an academic optimal control problem is analyzed. With the new method, the evaluation time was reduced by a factor of about 300, while the accuracy of the solution was maintained.**

***Index Terms*— Backward reachability, curse of dimensionality, dynamic programming (DP), level-set function.**

I. Introduction

DYNAMIC programming (DP) is a powerful numerical method for solving optimal control problems [1], [2]. Its main advantage above other methods is that global optimality of the found solution is guaranteed, regardless of the type of problem. The main drawbacks are noncausality and the fact that the computational effort grows exponentially with the number of state variables and inputs of the underlying dynamic system. When treating systems with continuous state variables, the time–state space typically needs to be discretized. It is possible to circumvent discretization, e.g., when the cost-to-go function can be expressed analytically, however, in general, discretization is the only viable option. Discretization inherently introduces numeric errors, which will degrade the accuracy of the found solution. In order to avoid these numerical errors without increasing computational effort, a careful implementation is important.

If a problem includes *final* state constraints, any solution trajectory is bound to lie inside the backward-reachable space, i.e., the space from which the final state constraint can be met within the given final time. This backward-reachable space represents a dynamic state constraint that coincides neither with the state constraints of the problem definition nor with the grid that is used for discretization. It was shown in [3] that it is important to carefully distinguish between backward-reachable and non-backward-reachable space in the numeric algorithm. Otherwise, a significant error might be introduced to the solution.

The contribution of this brief is a seemingly simple yet nontrivial combination of the basic DP algorithm with the reachability theory developed by [4]–[6]. The proposed algorithm avoids numerical errors that are due to the interpolation between backward-reachable and non-backward-reachable grid points. This improves the accuracy of the found solution significantly. In a case study, we demonstrate that the basic implementation of DP needs a much higher resolution in order to reach the same accuracy as the proposed method. In our example, the computational effort to solve the problem at a given accuracy was more than 300 times lower with the proposed method than with basic DP. In contrast to that, previously proposed methods to reduce the computational effort of DP, such as iterative [7], adaptive [8], approximate [9], neuro- [10] or state increment DP [11], either sacrifice global optimality or are applicable to a specific class of problems only. In [3] and [12], algorithms are proposed that are based on the same basic idea of calculating the backward-reachable space, however, these algorithms are applicable only to problems where the underlying system has only one dynamic state variable.

This brief is organized in the following way. After defining the optimal control problem in Section II, the numerical issues associated with the standard DP algorithm are reviewed in Section III. Then in Section IV, the method to calculate the boundaries of the backward-reachable space is reviewed briefly. The main contribution of this article is presented in Section V, where the standard DP algorithm is combined with the level-set method from [4]–[6] in a nontrivial way such that numeric errors are avoided. Finally, a case study demonstrates the improved performance of the proposed method in terms of accuracy and computational effort.

II. Optimal Control Problem

In this brief, we consider only optimal control problems that have a fixed final time and a partially constrained final state.

Manuscript received June 9, 2011; revised December 1, 2011; accepted February 11, 2012. Manuscript received in final form February 28, 2012. Date of publication April 4, 2012; date of current version April 17, 2013. This work was supported in part by the AHEAD Project, which is a common project of Carrosserie Hess AG and Eidgenössische Technische Hochschule (ETH) Zürich, sponsored by the Swiss Federal Innovation Promotion Agency (KTI). Recommended by Associate Editor C. Lagoa.

The authors are with the Department of Mechanical and Process Engineering, ETH Zürich, Zürich 8092, Switzerland (e-mail: elbert@idsc.mavt.ethz.ch; ebbesen@idsc.mavt.ethz.ch; guzzella@idsc.mavt.ethz.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2012.2190935

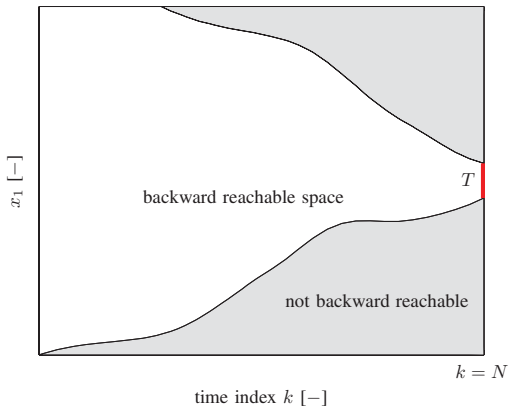


Fig. 1. Backward-reachable space for a system with only one dynamic state variable. The final state constraint is given by the set T.

The system has n state variables and m inputs. The state variables are assumed to be continuous variables[1] and the control inputs can be either continuous or discrete. The underlying dynamic system can be either a continuous-time or a discrete-time system. The optimal control problem is summarized as follows: find an admissible control sequence u_k, $k = 0, 1, \ldots, N$ such that the cost functional (1) is minimized and the constraints (2)–(6) are satisfied

$$\min_{u_k \in U_k} \left\{ g_N(x_N) + \sum_{0}^{N-1} g_k(x_k, u_k) \right\} \tag{1}$$

$$x_{k+1} = f_k(x_k, u_k) \tag{2}$$

$$x_k \in X_k \subseteq \mathbb{R}^n \tag{3}$$

$$x_0 = x_{\mathrm{IC}} \tag{4}$$

$$x_N \in T \subseteq \mathbb{R}^n \tag{5}$$

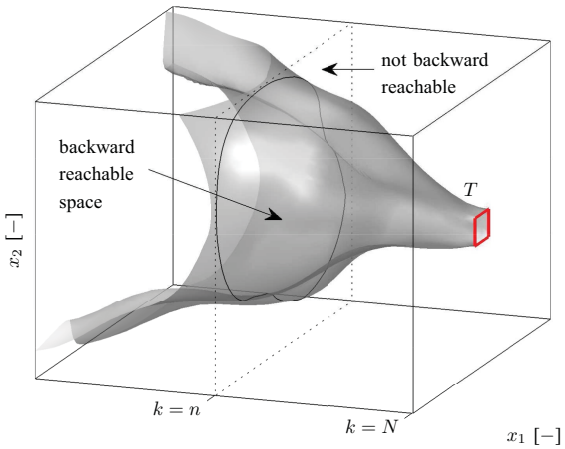
$$u_k \in U_k \subseteq \mathbb{R}^m \tag{6}$$

$$\text{for all } k = 0, 1, \ldots, N.$$

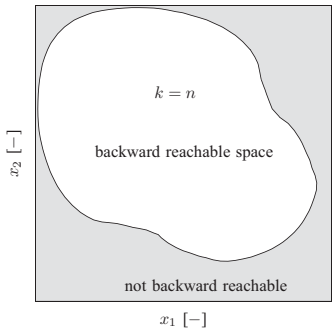
The function $g_N(x)$ is the final cost term and $g_k(x, u)$ is the stage cost, i.e., the cost of applying the control signal u at discrete time k to the dynamic system given by (2). Note that g_k and f_k are allowed to be time-variant, hence the index k. The state variables are constrained to the time-variant set X_k. The initial condition is given by x_{IC}, and the final value is partially constrained by the target set T. Additionally, the input signals are constrained by the time-variant set U_k.

Since DP is discrete in nature, the time, the state space, and the control space need to be discretized. Therefore, the functions f_k and g_k are discrete-time representations of the dynamic system and the stage-cost function. At time k, the state space is discretized to the set $X_k = \{x_k^1, x_k^2, \ldots, x_k^q\}$. The superscript i in x_k^i denotes the state variable in the discretized state–time space at the node with time index k and state index i. The continuous state vector is denoted by x_k. Analogously, the control space is represented by the discrete set $U_k = \{u_k^1, u_k^2, \ldots, u_k^r\}$. Note that the control inputs can be either discrete or continuous. In the latter case, U_k is a discrete approximation of the true control space.

[1]In fact, our method can also handle problems where some or all state variables are discrete. However, the benefits of the proposed method are apparent only when the optimal control problem contains at least one continuous state variable.



(a)



(b)

Fig. 2. (a) Backward-reachable space for a system with two dynamic state variables. (b) Note that in this example, the backward-reachable space at time $k = n$ is a nonconvex set.

Typically, the final state constraint defined by (5) cannot be met starting from every point in the search space. The evolution of the backward-reachable space and its boundary is depicted in Figs. 1 and 2 for typical 1- and 2-D problems. In a 1-D problem, the exact calculation of the backward-reachable space does not present any problem with the use of model inversion techniques, as was done in [3] and [12]. In contrast to that, the numerical representation of the backward-reachable space of higher dimensional problems is very difficult. Note that, even if the target set T is chosen to be a convex set, the backward-reachable space at any time $k \neq N$ might be nonconvex, e.g., at time $k = n$ in Fig. 2.

III. Basic DP

DP evaluates the optimal cost-to-go function $\mathcal{J}_k(x^i)$ at every node in the discretized time-state space by proceeding backwards in time.

1) Initialization of cost-to-go function

$$\mathcal{J}_N(x^i) = \begin{cases} g_N(x^i), & \text{for } x^i \in T \\ \infty, & \text{else.} \end{cases} \tag{7}$$

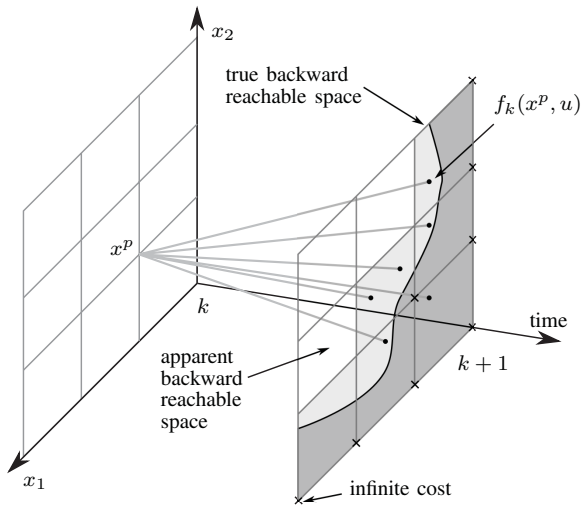


Fig. 3. Illustration of the numerical problems of the basic DP algorithm for a system with two dynamic state variables. If infinite cost values are used, together with interpolation the backward-reachable space will appear smaller than it actually is.

2) Backward iteration for $k = N-1$ to 0, $\forall x^i \in X_k$

$$\mathcal{J}_k(x^i) = \min_{u_k \in U_k} \left\{ g_k(x^i, u_k) + \mathcal{J}_{k+1}(f_k(x^i, u_k)) \right\}. \quad (8)$$

The optimal control signal at each node is given by the argument minimizing the right-hand side of (8), yielding the optimal control policy $\pi = \{\mu_0(x), \mu_1(x), \ldots, \mu_{N-1}(x)\}$.

Grid points that are not backward-reachable are, of course, infinitely expensive and therefore should have infinite cost, as in (7). However, this causes numerical problems. Consider the scenario depicted in Fig. 3. The cost-to-go $\mathcal{J}_{k+1}(x)$ is known for all grid points x^i at time $k+1$. In order to evaluate the cost to go at point x^p at time k, the algorithm simulates the system over one time step by applying all possible control candidates. Thereby, the system is driven into the points $f_k(x^p, u)$ with $u \in U_k$. Since these points do not generally coincide with the grid, interpolation is used to find the values of the cost-to-go at the points $\mathcal{J}_{k+1}(f_k(x^p, u))$. In this brief, we use multilinear interpolation. In the example shown in Fig. 3, interpolation will always rely on at least one grid point where the cost-to-go has an infinite value. Therefore, the backward-reachable space will appear smaller than it actually is and the cost-to-go at $\mathcal{J}_k(x^p)$ will be set to infinity, even though x^p is perfectly backward-reachable. When these effects continue during the DP iteration, the calculated backward-reachable space will be underestimated.

A first remedy to this problem is to choose a large but finite cost instead of an infinite value in (7). However, this technique results in a steep gradient in the cost-to-go function which distorts the found solution. This procedure will be referred to as "basic DP." Other techniques would be to increase the density of the grid toward the end of the problem, either by increasing the number of grid points (increase q, as k approaches N), or decreasing the size of the search space. We will show in our case study that these methods fail to produce useful results. This is mainly because the backward-reachable space is not known before initializing the algorithm.

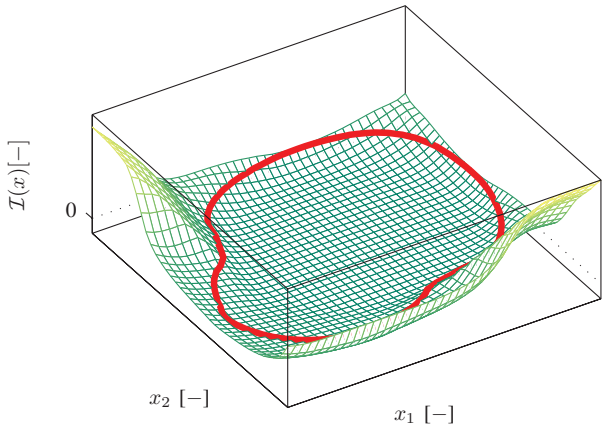


Fig. 4. Example of a level-set function $\mathcal{I}(x)$ which represents the feasible region from the lower part of Fig. 2 (thick line).

IV. Finding the Backward-Reachable Space

As a prerequisite for understanding the improved DP algorithm, which will be introduced in the next section, we will first briefly review how level-set functions can be used to calculate the backward-reachable space. This idea was developed by the authors of [4]–[6].

Generally, in an n-dimensional state space, it is not clear how the boundaries of the backward-reachable space evolve. Furthermore, at any time k, this space is not necessarily convex (see Fig. 2). Therefore, an exact numerical description is difficult. However, the feasible region at each time k can be conveniently estimated by an implicit surface function or level-set function, which is calculated by a DP algorithm.

The main idea is as follows. Let $\mathcal{I}$ be a function that acts on $\mathbb{R}^n$

$$\mathcal{I} : X \subseteq \mathbb{R}^n \to \mathbb{R}. \quad (9)$$

Such a function can be used to represent a region G that is defined as follows:

$$G = \{x \in X | \mathcal{I}(x) \leq 0\}. \quad (10)$$

Fig. 4 illustrates how a level-set function $\mathcal{I}(x)$ can be used to represent the backward-reachable space shown in Fig. 2. The advantages of such a representation are as follows. First, a Cartesian grid can be used for the evaluation of the function $\mathcal{I}(x)$. Together with (10), it is numerically easy (by interpolation) to evaluate whether a point x is backward-reachable or not. Second, such a function can represent regions of any shape (even nonconvex). Furthermore, the description is general and can be applied to systems with any number of state variables and control inputs. A drawback of this method is that the boundary of the backward-reachable space is not represented exactly, but rather approximated by the level-set function. The error of this approximation decreases with increasing grid density.

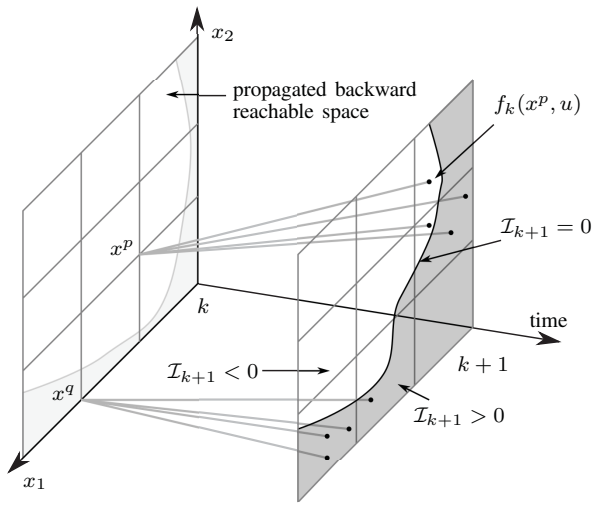


Fig. 5. Illustration of the algorithm that propagates in the backward-reachable space for a system with two dynamic state variables.

A. Calculating the Backward-Reachable Space Using DP

Assume that the final state constraint is given as a target set T, which is defined by a level-set function $h(x)$

$$h : X_N \to \mathbb{R}, \qquad \text{where } X_N \subseteq \mathbb{R}^n \tag{11}$$

$$T = \{x \in X_N | h(x) \leq 0\}. \tag{12}$$

Then, a DP algorithm is applied.

1) Initialization

$$\mathcal{I}_N(x^i) = h(x^i). \tag{13}$$

2) Backward iteration for $k = N-1$ to 0

$$\mathcal{I}_k(x^i) = \min_{u_k \in U_k} \left\{ \mathcal{I}_{k+1}(f_k(x^i, u_k)) \right\}. \tag{14}$$

The cost-to-go function of this algorithm is interpreted as a level-set function. Therefore, a new symbol $\mathcal{I}$ is used in order to distinguish it from the cost-to-go of the original problem. Fig. 5 illustrates the algorithm. Assume that the level-set function $\mathcal{I}_{k+1}(x)$ at time $k+1$ is known at all nodes x^i. Starting from a specific grid point x^p at time k, the algorithm applies all possible control candidates. This process yields the points $f_k(x^p, u)$ with $u \in U_k$. Then, as in (14), the algorithm assigns $\mathcal{I}_k(x^p)$ to be the minimum forward reachable value of $\mathcal{I}_{k+1}$, i.e., the minimum value of $\mathcal{I}_{k+1}(f_k(x^p, u))$. In the example, this value is negative for x^p, which indicates that x^p is a backward-reachable grid point. Contrarily, the point x^q is not backward-reachable.

V. Level-Set DP

This section presents the main contribution of this brief. It presents an improved version of the basic DP algorithm that minimizes the numerical problems described in Section III. The main idea is to use a level-set function to describe the backward-reachable space. This allows identifying grid points that lead to a violation of the final state constraints. Therefore, large penalty costs in the cost-to-go function are no longer required and a large gradient is avoided. The value of the cost-to-go function at grid points outside the backward-reachable space is not defined. Since interpolation might have to rely on such grid points, a reasonable approximation needs to be found. Our new algorithm simply uses the value of the cost of driving the system parallel to boundary of the backward-reachable space as close as possible to the final state constraint. This delivers a smooth cost-to-go function over the whole state space and, therefore, interpolation will not cause any particular numerical problem. This algorithm is referred to as "level-set DP algorithm."

A. Level-Set DP Algorithm

1) Initialize $k = N$ and the level-set and the cost-to-go functions as

$$\mathcal{I}_N(x^i) = h(x^i) \tag{15}$$

$$\mathcal{J}_N(x^i) = g_N(x^i) \tag{16}$$

where $h(x)$ is chosen to represent the target set T as in (11) and (12).

2) Then reduce k by 1 and update the level-set function by

$$\mathcal{I}_k(x^i) = \min_{u_k \in U_k} \left\{ \mathcal{I}_{k+1}(f_k(x^i, u_k)) \right\}. \tag{17}$$

3) For each grid point x^i, find the set of control signals for which the system trajectory ends up inside the backward-reachable space in the next time step

$$U_k^{\mathrm{F}}(x^i) = \{u_k \in U_k | \mathcal{I}_{k+1}(f_k(x^i, u_k)) \leq 0\} \tag{18}$$

and the one control candidate that minimizes the level-set function

$$\tilde{u}_k(x^i) = \operatorname*{argmin}_{u_k \in U_k} \left\{ \mathcal{I}_{k+1}(f_k(x^i, u_k)) \right\}. \tag{19}$$

4) Update the optimal cost-to-go by the following rule: if at least one valid control candidate is found, i.e., $U_k^{\mathrm{F}}(x^i) \neq \emptyset$, then calculate the cost-to-go based upon the optimal candidate

$$\mathcal{J}_k(x^i) = \min_{u_k \in U_k^{\mathrm{F}}(x^i)} \left\{ g_k(x^i, u_k) + \mathcal{J}_{k+1}(f_k(x^i, u_k)) \right\}. \tag{20}$$

If, however, the grid point is not backward-reachable, then calculate the cost-to-go based on the control input $\tilde{u}_k(x^i)$

$$\mathcal{J}_k(x^i) = g_k(x^i, \tilde{u}_k(x^i)) + \mathcal{J}_{k+1}(f_k(x^i, \tilde{u}_k(x^i))) \tag{21}$$

and repeat steps 2–4 until $k = 0$.

Consider the scenario depicted in Fig. 6. At time $k+1$, the cost-to-go and the level-set functions are known, e.g., from step 1. For the point x^p at time k, the algorithm applies all possible control candidates. This will drive the system to the points $f_k(x^p, u)$ with $u \in U_k$. Then, as in step 2, the value of the level-set function $\mathcal{I}_k$ at point x^p is set to be the minimal reachable value of $\mathcal{I}_{k+1}$, which will propagate the information about backward reachability to time k. In step 3, the algorithm identifies the valid control candidates

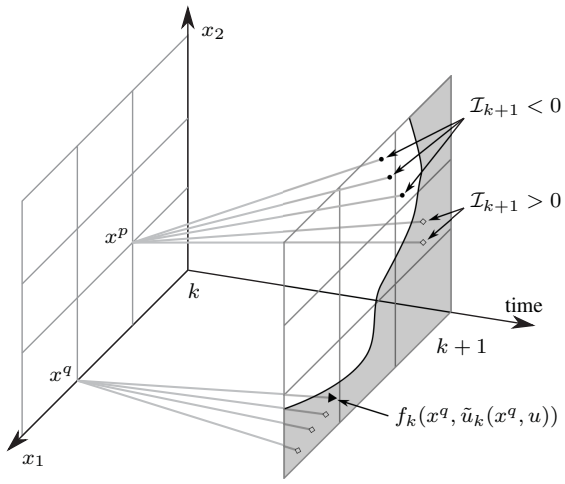


Fig. 6. Illustration of the level-set DP algorithm for a system with two state variables.

$U_k^F(x^p)$, based on the values of the level-set function at the points $\mathcal{I}(f_k(x^p, u))$ (black dots versus empty diamonds). In step 4, the cost-to-go at point x^p is calculated considering the valid control inputs only, as given by (20). The level-set function thus enables the algorithm to distinguish between those trajectories that end up inside the backward-reachable set and those that do not. Therefore, the cost-to-go function does not need an artificial large penalty cost. This way, numerical errors during the interpolation can be avoided.

Anyway, interpolation might still rely on grid points that are not part of the backward-reachable space. Since the cost-to-go at such points is not defined, a reasonable approximation needs to be found. If no valid control inputs can be found at all for a specific grid-point x^q, the algorithm concludes that this point does not lie inside the backward-reachable space. In this case, the value of the cost-to-go function is calculated based on the control $\tilde{u}_k(x^q)$ as in (21). This control input drives the system as closely as possible toward the backward-reachable space (black triangle). This approximation yields a cost-to-go function that is smooth over the whole state space. This gets clear when considering a point that lies exactly on the boundary of the backward-reachable space. For such a grid point, (20) and (21) would yield the same value and therefore the approximation is smooth. As a result, the algorithm can rely on grid points that are not part of the backward-reachable space when interpolating the cost-to-go function.

Note that it would generally be possible to exclude grid points that are not backward-reachable from evaluation, as in [12]. However, since the level-set function has to be evaluated for all grid points, the algorithm has to evaluate the expression $f_k(x^i, u_k)$ anyway. Therefore, the exclusion of non-backward-reachable grid points is not useful in the proposed method.

B. Forward Simulation

The result of the DP algorithm is the optimal cost-to-go function for all times k and all grid points x_k^i. In order to retrieve the sequence of optimal control inputs and the corresponding optimal state trajectory, a forward simulation has to be conducted. For the level-set DP, it takes the following form.

1) Initialization at $k = 0$

$$x_0 = x_{\mathrm{IC}}. \tag{22}$$

2) Find the feasible control candidates

$$U_k^{\mathrm{F}} = \{u_k \in U_k | \mathcal{I}_{k+1}(f_k(x_k, u_k)) \leq 0\}. \tag{23}$$

3) Find the optimal control input

$$u_k^o(x_k) = \begin{cases} \underset{u_k \in U_k^{\mathrm{F}}}{\operatorname{argmin}} \{g_k(x_k, u_k) \ldots \\ \qquad + \mathcal{J}_{k+1}(f_k(x_k, u_k))\} \\ \qquad \text{if } U_k^{\mathrm{F}} \neq \emptyset, \\ \underset{u_k \in U_k}{\operatorname{argmin}} \{\mathcal{I}_{k+1}(f_k(x_k, u_k))\} \\ \qquad \text{else.} \end{cases} \tag{24}$$

4) Simulate the system using the optimal control input

$$x_{k+1} = f_k(x_k, u_k^o) \tag{25}$$

and repeat steps 2–4 until $k = N$.

Note that in the forward simulation x_k can take continuous values, therefore the superscript i does not appear in (22)–(25).

The second case of (24) is introduced to improve the robustness of the algorithm. It becomes active only if x_k lies outside the feasible region. However, if x_{IC} is inside the feasible region, this case remains inactive.

C. Initialization of the Target Set

This section will briefly explain how the level-set function is to be set up so as to represent a specific target set T. Generally, there are infinitely many ways to initialize such a function. Here we present one approach that was found to deliver useful results. Ideally, the target set contains only one target point. Since only a discrete number of control signals are available, it is impossible to hit a target point exactly. Therefore, a target set

$$T = [x_1^{\min}, x_1^{\max}] \times \cdots \times [x_n^{\min}, x_n^{\max}] \in \mathbb{R}^n \tag{26}$$

is specified, where $x_n^{\min}$ and $x_n^{\max}$ are the minimum and maximum allowed values for the nth state variable. When using linear interpolation, it makes sense to define the function $h(x)$ such that its value at any point x is equal to the distance from x to the nearest bound

$$h(x) = \max_{j=1,\ldots,n} \left\{ \max \left(x_j^{\min} - x_j, x_j - x_j^{\max} \right) \right\}. \tag{27}$$

Inside T, $h(x)$ will have negative values; outside T, positive values. The minimum of $h(x)$ lies inside the set T.

D. Initialization of the Cost-to-Go Function

This section will briefly explain an alternative way of initializing the cost-to-go function at time $k = N$, such as to further improve the accuracy of the algorithm.

As k approaches the final time N, the optimal trajectory is likely to move along the boundary of the backward-reachable space. In this case, the algorithm has to rely on grid

points that are outside the feasible region when interpolating the cost-to-go function. The value of the cost-to-go function at these grid points is calculated based on the control input that drives the system as close as possible to the target set T. Since the target set T cannot be reached from these grid points, it is beneficial to account for that circumstance in the initialization of the cost-to-go function. For grid points that lie outside the target set, the additional cost that would result from a trajectory leading into the target set T in some time $k > N$ has to be added to the final cost g_N. A linear approximation can easily be found and is sufficient in most cases. In the level-set DP, the initialization of the cost-to-go function (16) is therefore replaced by

$$\mathcal{J}_N(x^i) = \begin{cases} g_N(x^i), & \text{if } x^i \in T \\ g_N(x^i) + \lambda^T \cdot (x^i - x_T), & \text{else} \end{cases} \tag{28}$$

where $(x^i - x_T)$ is the minimum distance from point x^i to the target set T, and s is a vector of factors that convert the distance between the grid point x^i and the target set T into an equivalent cost. In the literature, the vector λ is typically referred to as the "equivalence factor" or "Lagrange multiplier," hence the symbol λ.

E. Computational Effort

Typically, the number of model evaluations is the factor that has the greatest influence on the calculation time. The number of model-function evaluations for the basic DP and the level-set DP with an equally spaced grid is given by

$$N_{\text{feval}} = N \cdot \prod_{i=1}^{n} N_{x_i} \cdot \prod_{j=1}^{m} N_{u_j} \tag{29}$$

where N_{x_i} is the number of grid points in the ith state variable and N_{u_j} represents the number of grid points in the jth control input.

Compared to the basic DP implementation, the level-set DP approach causes some additional calculation demand due to the interpolation of the level-set function (17) and the determination of the valid control candidates (18). However, the system dynamic equation f_k and the arc-cost g_k has to be evaluated only once for each grid point/control input combination. In the limit case, where the model evaluation takes up an infinitely small amount of time $t_{\text{model}} \to 0$, the total evaluation time doubles compared to that required by the basic DP approach. However, in a typical DP problem, the evaluation of the model itself takes up most of the calculation time. Therefore, when using both algorithms to solve the same problem on the same grid, the calculation time for the level-set DP algorithm is only slightly longer than for the basic DP approach.

VI. Case Study: Simple Dynamic System

The accuracy of a solution obtained by DP is highly dependent on the discretization of the state space. Therefore, choosing a good level of discretization is a tradeoff between accuracy and calculation time. If a finer grid is chosen, the solution becomes more precise, but also calculation time increases.

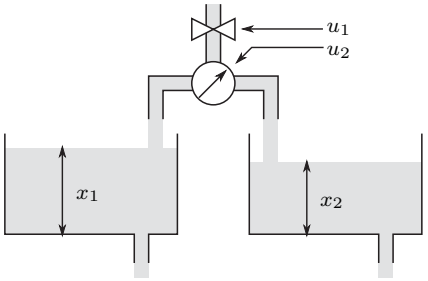


Fig. 7. Illustration of the example problem.

In this section, a simple optimal control problem based on a dynamic system with two state variables and two inputs is used to compare the accuracy of the basic DP and the level-set DP algorithms. Since the analytic solution to the problem is known, a fair comparison of the results is possible.

A. Problem Definition

The system is described by the following dynamic equations:

$$\dot{x}_1(t) = -\frac{1}{2}x_1(t) + u_1(t) \cdot u_2(t) \tag{30}$$

$$\dot{x}_2(t) = -\frac{1}{2}x_2(t) + u_1(t) \cdot (1 - u_2(t)) \tag{31}$$

with the constraints

$$x(t) \in [0,1] \times [0,1] \quad \forall t \in [0, t_f] \tag{32}$$

$$u(t) \in [0,1] \times [0,1] \quad \forall t \in [0, t_f] \tag{33}$$

and the initial and final conditions

$$x_1(0) = x_2(0) = 0 \tag{34}$$

$$x_1(t_f) = x_2(t_f) \geq 0.5 = x_{\min} \tag{35}$$

where $t_f = 2$ s. The cost functional to be minimized is given as

$$J = \int_0^{t_f} u_1(t) + 0.1 \cdot |u_2(t) - 0.5| dt. \tag{36}$$

The problem can be illustrated as in Fig. 7. Within time t_f, two water reservoirs have to be filled from empty to a certain level $x(t_f) = [0.5, 0.5]^T$, using a minimum amount of water. But both the reservoirs have a leak, where the outflow is proportional to the amount of water in the reservoir. The first control input determines the total water flow into the system, whereas the second input determines how the water is distributed between the two reservoirs. The second term in the cost functional penalizes the action on the second control input and ensures that the problem has a unique solution.

The optimal solution to this problem is to start filling in the water as late as possible at time t_{on} with the maximum inflow, such that the final state constraint is exactly met at time t_f. The inflow has to be distributed between the two reservoirs equally. This way, the amount of water lost through the leaks

is minimized over the time horizon. Therefore, the optimal control input is given as

$$u_1^o(t) = \begin{cases} 0, & \text{if } t < t_{\text{on}} \\ 1, & \text{if } t \geq t_{\text{on}} \end{cases} \quad (37)$$

$$u_2^o(t) = 0.5 \qquad \forall t \in [0, t_f]. \quad (38)$$

The optimal time to switch on the water is calculated by backward-integrating the system dynamics from the final state constraint using the optimal control inputs. It can be shown that

$$t_{\text{on}} = 2 + 2 \cdot \ln\left(\frac{1}{2}\right). \quad (39)$$

Therefore, the value of the optimal cost functional is

$$J^o_{\text{analytic}} = \int_{t_{\text{on}}}^{t_f} 1 \, dt = t_f - t_{\text{on}} = -2 \cdot \ln\left(\frac{1}{2}\right) \approx 1.3863. \quad (40)$$

Since the optimal solution is to move along the boundary of the feasible region, the DP algorithm is sensitive to the method used to account for the final state constraints.

B. Resolution Study

The problem stated above is solved using the basic and level-set DP algorithms using various levels of state space discretization. The time discretization is chosen to be $\Delta t = 0.01$ s. The number of points used to discretize each state variable are $N_x = N_{x_1} \cdot N_{x_2}$, while the control space discretization is kept at $N_u = N_{u_1} \cdot N_{u_2} = 21 \times 21 = 441$. Due to the discretization of time, an inherent minimal error is introduced

$$\frac{J^o_{\text{DP}} - J^o_{\text{analytic}}}{J^o_{\text{analytic}}} = \frac{1.39 + 2 \cdot \ln(\frac{1}{2})}{-2 \cdot \ln(\frac{1}{2})} = 0.0027. \quad (41)$$

For the basic DP algorithm, the penalty cost for non-backward-reachable states is chosen to be the maximum cost that can occur in the problem. The maximum cost accumulates when the water valve is opened at $t = 0$ s and not closed again before t_f, and therefore $\mathcal{J}^\infty = \int_0^{t_f} 1 dt = 2$.

For the level-set DP, the cost-to-go function was initialized using (28)

$$\mathcal{J}_N(x^i) = \sum_{n=1}^{2} \max(x_{\min} - x_n^i, 0) \quad (42)$$

where x_n^i is the nth element of the vector $x^i \in X_N$. This accounts for the fact that, if either of reservoirs n ends up at time $k = N$ with a level lower than $x_{\min}$, it has to be filled up to the level $x_{\min}$ in future, thereby causing a future cost of $(x_{\min} - x_n^i)$. From a physical standpoint, it therefore makes sense that the vector of equivalence factors is equal to $\lambda = [1, 1]^T$ in this example.

Fig. 8 shows the found system trajectory and the corresponding control inputs when the algorithms are used with $N_x = 51 \times 51$. In the upper graph, the optimal trajectories of $x_1^o(t)$ and $x_2^o(t)$ are equal and therefore coincide. Clearly, the solution found by the level-set algorithm is much closer to the analytic solution. With the basic DP approach, the water valve is opened much too early and therefore more water is lost through the leaks. The system is not able to move closely

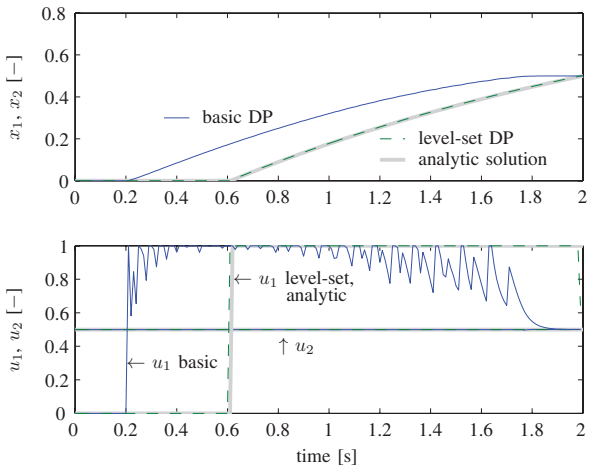


Fig. 8. System trajectory and control inputs when solving the problem with $N_x = 51 \times 51$.

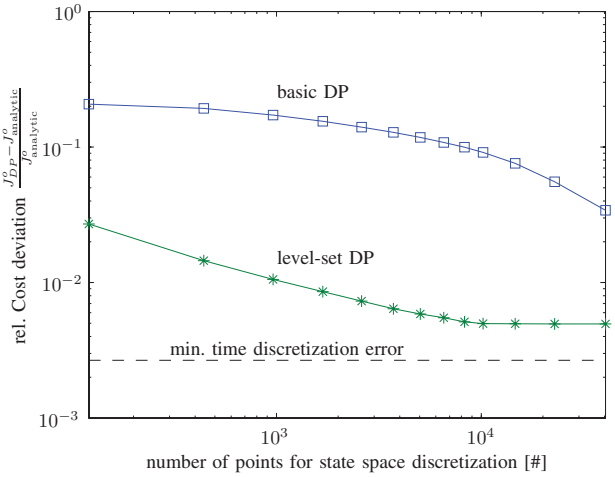


Fig. 9. Relative deviation of the cost computed by DP compared to the optimal cost obtained by the analytic solution for several levels of discretization ranging from $N_{x_1} = N_{x_2} = 11$ up to 201.

along the boundary of the backward-reachable space due to the numerical problems described earlier.

The corresponding control inputs are depicted in the lower graph of Fig. 8. It can be seen that the basic algorithm causes a very erratic trajectory for the first control input. The level-set method has the additional benefit that the control inputs are much smoother.

The relative deviation of the cost computed by DP compared to the results obtained by the analytic solution is shown in Fig. 9. Over the whole range of levels of discretization, the results obtained by the level-set DP approach are substantially closer to the analytic solution than those of the basic DP approach. Even at the lowest level of discretization, the level-set algorithm yields better results than the basic DP approach at the highest level of discretization.

When looking at the results presented in Fig. 9, it gets immediately clear that an adaptive grid method, based on refining discretization toward the end of the problem, can never reach the accuracy of the proposed algorithm. An algorithm

TABLE I
RESULTS OBTAINED WITH THE TWO ALGORITHMS AT SIMILAR ACCURACY FOR THE SIMPLE DYNAMIC SYSTEM PROBLEM

		Basic DP	**Level set DP**
Level of discretization	N_x	201×201	11×11
Relative cost deviation	ΔJ	0.034	0.027
Function evaluations	N_{feval}	3'581'185'041	10'725'561
Measured eval. time	t_{eval} [min]	164	0.42

that starts with $N_x = 11 \times 11$ and ends with $N_x = 201 \times 201$ will never be more accurate than an algorithm with a constant $N_x = 201 \times 201$. Obviously, the new proposed method delivers superior results.

C. Computational Effort

The computational effort of the level-set DP and basic DP algorithms is shown in Table I for a similar level of accuracy. The level-set DP approach allows choosing a significantly lower level of discretization. Instead of a grid with 201×201 points, a grid of 11×11 points is sufficient. Assuming that the speed of the algorithm is mainly determined by the number of function evaluations, as defined in (29), the time savings can be calculated, with the result that the level-set DP algorithm is about 334 times faster than the basic DP approach, and yet the resulting cost is closer to that of the analytic solution.

In order to consolidate this statement, the calculation time to solve the problem on a 2.66-GHz processor was measured. The last row in Table I shows that the evaluation time is accelerated by a factor of about 390 when using the level-set DP algorithm instead of the basic DP algorithm. This result seems somewhat counterintuitive since the benefit of lowering the discretization is even greater than the value that was estimated using (28). The reason for this effect is that the handling of the large amounts of data, as required for a high level of discretization, causes an additional significant computational effort which was not accounted for in (28). Working with a low level of discretization therefore has an additional positive effect.

VII. CONCLUSION

The level-set DP algorithm presented in this brief considerably improves the efficiency of DP. The method is generic because it does not require any assumptions regarding the properties of the optimal control problem and it does not compromise global optimality of the found solution. It can handle nonlinear optimal control problems with any number of state variables and control inputs. The method can improve efficiency only if the underlying dynamic system contains at least one continuous state variable and if the final state is partially constrained. In fact, many real-world optimal control problems faced in engineering belong to this class. In such problems, the optimal state trajectory is close to the boundary of the backward-reachable space when the final time is approached. Computational cost is reduced since the proposed algorithm achieves the same accuracy as the basic implementation at a much lower state space resolution. Further, the calculated optimal control inputs are much smoother than the ones calculated with the basic algorithm at the same discretization.

ACKNOWLEDGMENT

The authors would like to thank O. Sundström for his original contribution [13]. A generic dynamic programming function for MATLAB is available online at www.idsc.ethz.ch/downloads. The most recent version incorporates the level-set method described herein.

REFERENCES

[1] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[2] D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995.
[3] O. Sundström, D. Ambühl, and L. Guzzella, "On implementation of dynamic programming for optimal control problems with final state constraints," in *Proc. IFP Conf. Adv. Powertrains*, 2008, pp. 91–102.
[4] P. Varaiya, "Reach set computation using optimal control," in *Proc. KIT Workshop Verification Hybrid Syst.*, Grenoble, France, 1998, pp. 377–383.
[5] A. Kurzhanski and P. Varaiya, "Dynamic optimization for reachability problems," *J. Opt. Theory Appl.*, vol. 108, no. 2, pp. 227–251, 2001.
[6] I. Mitchell, A. Bayen, and C. Tomlin, "Validating a hamilton-jacobi approximation to hybrid system reachable sets," in *Hybrid Systems: Computation and Control*. New York: Springer-Verlag, 2001, pp. 418–432.
[7] R. Luus, *Iterative Dynamic Programming*, 1st ed. Boca Raton, FL: CRC Press, 2000.
[8] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 39–47, May 2009.
[9] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York: Wiley, 2007.
[10] D. Bertsekas and J. Tsitsiklis, "Neuro-dynamic programming: An overview," in *Proc. IEEE Decision Control 34th Conf.*, Dec. 1995, pp. 560–564.
[11] R. E. Larson, *State Increment Dynamic Programming*. New York: American Elsevier, 1968.
[12] M. Back, S. Terwen, and V. Krebs, "Predictive powertrain control for hybrid electric vehicles," in *Proc. IFAC Symp. Adv. Autom. Control*, Salerno, Italy, Apr. 2004, pp. 451–457.
[13] O. Sundström and L. Guzzella, "A generic dynamic programming MATLAB function," in *Proc. IEEE Control Appl. Intell. Control*, Jul. 2009, pp. 1625–1630.