

Recent Results in the Approximation of Nonlinear Optimal Control Problems

Maurizio Falcone(✉)

Dipartimento di Matematica, SAPIENZA - Università di Roma,
P.za Aldo Moro, 2, 00185 Roma, Italy
`falcone@mat.uniroma1.it`

Abstract. This survey paper presents recent advances for the numerical solution of Hamilton-Jacobi-Bellman equations related to optimal control problems. The Dynamic Programming approach suffers for the “curse of dimensionality” and the solution of the nonlinear partial differential equations characterizing the value function of optimal control problems in high dimension is out of reach. However, a combination of various techniques can circumvent this difficulty and find the solution of optimal control problems up to dimension 10, a range of dimensions which could be enough for many applications. We illustrate here some of these techniques: patchy domain decomposition, fast marching and fast sweeping and an acceleration method based on the coupling between value and policy iteration. Numerical examples will illustrate the main features of those methods.

1 Introduction

The numerical solution of partial differential equations obtained by applying the Dynamic Programming Principle (DPP) to nonlinear optimal control problems is a challenging topic that can have a great impact in many areas, e.g. robotics, aeronautics, electrical and aerospace engineering. Indeed, by means of the DPP one can characterize the value function of a fully nonlinear control problem (including also state/control constraints) as the unique viscosity solution of a nonlinear Hamilton-Jacobi equation, and, even more important, from the solution of this equation one can derive the approximation of a feedback control. This result is the main motivation for the PDE approach to control problems and represents the main advantage over other methods, such as those based on the Pontryagin minimum principle. It is worth to mention that the characterization via the Pontryagin principle gives only necessary conditions for the optimal trajectory and optimal open-loop control. Although from the numerical point of view the control system can be solved via shooting methods for the associated two point boundary value problem, in real applications a good initial guess

The author wish to acknowledge the support obtained by the following grants: AFOSR Grant no. FA9550-10-1-0029, ITN - Marie Curie Grant no. 264735-SADCO.

for the co-state is particularly difficult and often requires a long and tedious trial-error procedure to be found.

In this paper we focus our attention on efficient methods to implement the DP approach for nonlinear control problems governed by ordinary differential equations. In particular, our presentation will be centered on the *minimum time problem*, which is associated to the following Hamilton-Jacobi-Bellman equation

$$\begin{cases} \max_{a \in A} \{-f(x, a) \cdot \nabla u(x)\} - 1 = 0, & x \in \mathbb{R}^d \setminus \mathcal{T} \\ u(x) = 0, & x \in \mathcal{T} \end{cases} \quad (1)$$

where d is the dimension of the state, $A \subset \mathbb{R}^m$ is a compact set defining the admissible controls, \mathcal{T} is the target set to be reached in minimal time and $f : \mathbb{R}^d \times A \rightarrow \mathbb{R}^d$ is the dynamics of the system. For this classical problem the value function $T : \mathbb{R}^d \rightarrow \mathbb{R}$ at the point x is the minimal time to reach the target starting from x (note that $T(x) = +\infty$ if the target is not reachable). For numerical purposes, the equation is solved in a bounded domain $\Omega \supset \mathcal{T}$, so that also boundary conditions on $\partial\Omega$ are needed. A rather standard choice when one does not have additional informations on the solution and deals with target problems is to impose state constraints boundary conditions.

The techniques used to obtain a numerical approximation of the viscosity solution of Eq. (1) have been mainly based on Finite Differences [12, 22] and Semi-Lagrangian schemes [15, 17]. It is rather important to note that traditional approximation schemes presented for example in [12] and [15] are based on a fixed point iteration scheme, which computes the solution at each node of the grid at every iteration. Denoting by M the number of nodes in each dimension and considering that the number of iterations needed for convergence is of order $O(M)$, the total cost of this full-grid scheme is $O(M^{d+1})$. We easily conclude that this algorithm is very expensive when the state dimension is $d \geq 3$, although it is rather efficient for low dimensional control problems as shown in [15] (see also the book [17]).

The “curse of dimensionality” is a typical drawback of Dynamic Programming and can not be eliminated. However, several techniques have been introduced in order to solve the DP equations in a rather high dimension (see [10] for a first tentative in this direction). Typically $1 \leq d \leq 10$ is an interesting range which can allow to solve many problems coming from applications, moreover a model reduction technique can be applied to the original dynamics in order to get a new dynamical system of lower dimension still catching the behavior of the dynamics. This remark is the main motivation which has driven the search for new computational techniques aimed to accelerate convergence and/or to reduce the memory allocation requirements.

Let us give some examples. One possible strategy is based on the decomposition of the domain Ω . The problem is actually solved in subdomains Ω^j , $j = 1, \dots, R$, whose size is chosen in order to reduce the number of grid nodes to a manageable size. Therefore, rather than solving a unique huge problem, one can solve R smaller subproblems working simultaneously on several processors. This produces a simple parallel algorithm. Depending on the choice of the subdomains

Ω^j we can have some overlapping regions or a number of interfaces between the subdomains. The presence of interfaces and/or overlapping regions is a delicate point, since at each iteration of the algorithm it will be necessary to exchange information between processors to properly define the values at the interfaces. Without this communication the result will not be correct. The interested reader can find in the book [26] a comprehensive introduction to domain decomposition techniques, whereas for an application to Hamilton-Jacobi equations we refer to [8, 18]. In this approach the choice of the division into subdomains is aimed to choose rather simple boundaries and geometries (typically an hypercube is divided into small hypercubes). A recent improvement has been made in [9] trying to adapt the geometry to the optimal dynamics of the system in order to obtain a subdivision made by “almost” invariant subdomains (the patches), this allows to eliminate the transmission load due to the exchange of informations between different processors. Previous patchy decompositions based on different ideas have been proposed first by Navasca and Krener in [20].

Another proposal to reduce the computational effort is the so-called Fast Marching method introduced in [25, 27]. While the full-size grid is always allocated, the computation is restricted to a small portion of the grid, thus saving CPU time. The cost of this method is of order $O(M^d \log M^d)$. In the original version, the Fast Marching method was derived for the Eikonal equation, corresponding (under a suitable change of variable) to Eq. (1) with $f(x, a) = a$ and $A = B_d(0, 1)$, the unit ball in \mathbb{R}^d centred in 0 (see [14] for details). Despite the efficiency of the Fast Marching method, at present its application to more general equations of the form $H(x, u(x), \nabla u(x)) = 0$ is not an easy task and it is still under investigation (see [7, 11, 13, 23]) because the causality principle which is behind the ordering of the grid nodes is not easy to detect for general control problems. Other methods have been proposed exploiting the idea that one can accelerate convergence by alternating the order in which the grid nodes are visited giving rise to the so-called “sweeping methods”. These methods do not require a special ordering of the grid nodes and are somehow blind, so it could be difficult to prove that they converge after a finite number of sweeps. However, they are easy to implement and they have been shown to be efficient for the Eikonal equation [28] and, more recently, for rather general Hamiltonians [24].

The third method is based on a coupling between two classical methods: value and policy iteration. It is well known that the value iteration is globally convergent but the rate of convergence is rather slow, whereas the policy iteration is locally convergent with a super-linear (or quadratic) rate of convergence. Then, a natural idea is to combine these methods in order to obtain a globally convergent method which starts using the value iteration to switch into the policy iteration when it reaches a “small” neighborhood of the solution.

The survey is organized as follows. Section 2 is devoted to the general presentation of two computational methods: the value iteration and the policy iteration. The semi-Lagrangian scheme associated to these methods will be the first building block for the following improvements. Section 3 is devoted to the patchy

domain decomposition method. Section 4 will briefly sketch Fast Marching and Fast sweeping methods. Finally, Sect. 5 is devoted to an acceleration method based on the coupling between value iteration and policy iteration.

2 Two Classical Algorithms for Dynamic Programming

In this section we will summarize the basic results for the two methods as they will constitute the starting point for our new algorithms. The essential features will be briefly sketched, more details can be found in the original papers and in some monographs, e.g. in the classical books by Bellman [6], Howard [19] and for a more recent setting in the framework of viscosity solutions in [3, 15]. Let us present the method for the *minimum time problem* where the dynamics is

$$\begin{cases} \dot{y}(t) = f(y(t), \alpha(t)) \\ y(0) = x \end{cases} \quad (2)$$

where $y \in \mathbb{R}^d$, $\alpha \in \mathbb{R}^m$ and $\alpha \in \mathcal{A} \equiv \{a : \mathbb{R}_+ \rightarrow A, \text{ measurable}\}$. If f is Lipschitz continuous with respect to the state variable and continuous with respect to (x, α) , the classical assumptions for the existence and uniqueness result for the Cauchy problem (2) are satisfied. To be more precise, the Carathéodory theorem implies that for any given control $\alpha(\cdot) \in \mathcal{A}$ there exists a unique trajectory $y(\cdot; \alpha)$ satisfying (2) almost everywhere. Changing the control policy the trajectory will change producing a family of infinitely many solutions of the controlled system (2) parametrized with respect to α .

In the minimum time problem one has to drive the controlled dynamical system (2) from its initial state to a given target \mathcal{T} . Let us assume that the target is a compact subset of \mathbb{R}^d with non empty interior and piecewise smooth boundary. The major difficulty dealing with this problem is that the time of arrival to the target starting from the point x and applying the control strategy α , denoted by $t(x, \alpha(\cdot))$, can be infinite at some points (if the strategy does not bring to \mathcal{T}), i.e.

$$t(x, \alpha(\cdot)) := \begin{cases} \inf_{\alpha \in \mathcal{A}} \{t \in \mathbb{R}_+ : y(t, \alpha(\cdot)) \in \mathcal{T}\} & \text{if } y(t, \alpha(t)) \in \mathcal{T} \text{ for some } t, \\ +\infty & \text{otherwise,} \end{cases} \quad (3)$$

As a consequence, the minimum time function defined as

$$T(x) = \inf_{\alpha \in \mathcal{A}} t(x, \alpha(\cdot)) \quad (4)$$

is not defined everywhere if some controllability assumptions are not satisfied. In general, this is a free boundary problem where one has to determine at the same time, the couple (T, Ω) , i.e. the minimum time function and its domain. Nevertheless, by applying the Dynamic Programming Principle and the so-called Kruzkov transform

$$v(x) \equiv \begin{cases} 1 - \exp(-T(x)) & \text{for } T(x) < +\infty \\ 1 & \text{for } T(x) = +\infty \end{cases} \quad (5)$$

the minimum time problem is characterized in terms of the unique viscosity solution of

$$\begin{cases} v(x) + \sup_{a \in A} \{-f(x, a) \cdot Dv(x)\} = 1 & \text{in } \mathbb{R} \setminus \mathcal{T} \\ v(x) = 0 & \text{on } \partial\mathcal{T}, \end{cases} \quad (6)$$

The *semi-Lagrangian scheme* for the approximation of (6) is obtained coupling a discretization in time along the trajectories with a local reconstruction in space via interpolation. Several coupling are possible and the interested reader can find in [17] all the details. Here we just sketch the one dimensional case where the integration along the trajectory is obtained using the Euler method. We introduce a grid G on Ω with nodes x_i , $i = 1, \dots, N$. Without loss of generality, throughout this paper we will assume that the numerical grid G is a regular equidistant array of points with mesh spacing denoted by Δx . We also denote by \mathring{G} the internal nodes of G and by ∂G its boundary, whose nodes act as *ghost nodes*. We map all the values at the nodes onto a N -dimensional vector $U = (U_1, \dots, U_N)$. Let us denote by $h_{i,a} > 0$ a (fictitious) time step, possibly depending on the node x_i and control a , and by $k = \Delta x > 0$ the space step. For every internal node of the grid we follow the dynamics using one step of the Euler scheme [4, 5] then we compute the values at the points $x_i + h_{i,a}f(x_i, a)$ via an interpolation operator denoted by $I[U]$ [15]. Finally, we obtain the following scheme in fixed point form for of (6)

$$U = F(U), \quad (7)$$

where $F : [0, 1]^N \rightarrow [0, 1]^N$ (due to the Kruzkov change of variable) is defined componentwise by

$$[F(U)]_i = \begin{cases} \min_{a \in A} \{I[U](x_i + h_{i,a}f(x_i, a)) + h_{i,a}\} & x_i \in \mathring{G} \setminus \mathcal{T}, \\ 0 & x_i \in \mathcal{T}, \\ 1 & x_i \in \partial G. \end{cases}$$

The interpolation operator $I[U] : \Omega \rightarrow \mathbb{R}$ extends the discrete value function U to the whole space Ω . In order to fix the ideas, one should think to the linear interpolation in \mathbb{R}^d described in [10] but other choices are possible [17]. We choose the time step $h_{i,a}$ such that $|h_{i,a}f(x_i, a)| = k$ for every $i = 1, \dots, N$ and $a \in A$, so that the point $x_i + h_{i,a}f(x_i, a)$ falls in one of the first neighboring cells. In the simplest case, the minimum over A is evaluated by direct comparison, discretizing the set A with N_c points but other (more expensive and accurate) methods are available. Note that defining $F(U) = 1$ on ∂G corresponds to impose state constraint boundary conditions. The final iterative scheme reads

$$U^{(n+1)} = F(U^{(n)}), \quad U^{(0)} = \begin{cases} 0 & \text{on } \mathcal{T} \\ 1 & \text{otherwise} \end{cases}. \quad (8)$$

We refer to [15, 17] for details on the building blocks of this construction and for the convergence analysis. With the discrete value function U in hand, we can obtain a feedback map $\Phi_h : \Omega \rightarrow A$ just defining

$$\Phi_h(x) := \arg \min_{a \in A} \{I[U](x + h_{x,a}f(x, a)) + h_{x,a}\}. \quad (9)$$

Under rather general assumptions (see [16]), it can be shown that this is an approximation of the feedback map constructed for the continuous problem. A detailed discussion on the construction of feedback maps via the value function is contained in [3, p. 140–143]. It is important to note that weak convergence results apply also for Lipschitz continuous value functions.

Then, the *value iteration* based on the semi-Lagrangian method leads to following iterative scheme:

```

Data: Mesh  $G$ ,  $\Delta t$ , initial guess  $V^0$ , tolerance  $\epsilon$ .
forall the  $x_i \in \mathcal{T}$  do
  | set  $V_i = 0$ 
end
forall the  $x_i \in \partial G$  do
  | set  $V_i = 1$ 
end
while  $\|V^{k+1} - V^k\| \geq \epsilon$  do
  | forall the  $x_i \in G$  do
  |   |  $V_i^{k+1} = \min_{a \in A} \{e^{-\Delta t} I[V^k](x_i + \Delta t f(x_i, a)) + 1 - e^{-\Delta t}\}$ 
  |   end
  |  $k = k + 1$ 
end

```

Algorithm 1: (VI) Value Iteration method for minimum time problem

Here V_i^k represents the values at a node x_i of the grid at the k -th iteration and I is an interpolation operator acting on the values of the grid.

Algorithm 1 is referred in the literature as the *value iteration method* because, starting from an initial guess V^0 , it modifies the values on the grid according to the nonlinear rule in the loop. It is well-known that the convergence of the value iteration can be very slow, since the contraction constant $e^{-\Delta t}$ is close to 1 when Δt is close to 0. This means that a higher accuracy will also require more iterations. Then, there is a need for an acceleration technique in order to cut the link between accuracy and complexity of the value iteration. Note that similar ideas can be applied to other classical control problems with small changes [17].

A classical acceleration technique is the *approximation in the policy space* (or policy iteration), it is based on a linearization of the Bellman equation. This method is due to Howard [19] and dates back to the origin of dynamic programming. First, an initial guess for the control for every point in the state space is chosen. Once the control has been fixed, the Bellman equation becomes linear (no search for the minimum in the control space is performed), and it is solved as an advection equation. Then, an updated policy is computed and a new iteration starts. This leads to the following algorithm.

Note that the solution of the policy evaluation step can be obtained either by a linear system (assuming a linear interpolation operator) or as the limit

$$V^k = \lim_{m \rightarrow +\infty} V^{k,m}, \quad (10)$$

Data: Mesh G , Δt , initial guess V^0 , tolerance ϵ .
forall the $x_i \in \mathcal{T}$ **do**
 | set $V_i = 0$
end
forall the $x_i \in \partial G$ **do**
 | set $V_i = 1$
end
while $\|V^{k+1} - V^k\| \geq \epsilon$ **do**
 Policy evaluation step:
 forall the $x_i \in G$ **do**
 | $V_i^k = \Delta t + e^{-\Delta t} I \left[V^k \right] \left(x_i + \Delta t f \left(x_i, a_i^k \right) \right)$
 end
 Policy improvement step:
 forall the $x_i \in G$ **do**
 | $a_i^{k+1} = \arg \min_a \left\{ \Delta t + e^{-\Delta t} I \left[V^k \right] \left(x_i + \Delta t f(x_i, a) \right) \right\}$
 end
 $k = k + 1$
end

Algorithm 2: (PI) Policy Iteration method for the minimum time problem

of the linear time-marching scheme

$$V_i^{k,m+1} = \Delta t + e^{-\Delta t} I \left[V^{k,m} \right] \left(x_i + \Delta t f \left(x_i, a_i^k \right) \right). \quad (11)$$

Although this scheme is still iterative, the lack of a minimization phase makes it faster than the original value iteration. The sequence $\{V^k\}$ turns out to be monotone decreasing at every node of the grid. At a theoretical level, policy iteration can be shown to be equivalent to a Newton method, and therefore, under appropriate assumptions, it converges with quadratic speed (see [21]). On the other hand, convergence is local and this may represent a drawback with respect to value iterations.

3 The Patchy Domain Decomposition

In this section we introduce our new domain decomposition method for solving equations of Hamilton-Jacobi-Bellman type, in particular (6). The main feature of the new method is the technique we use to construct the subdomains of the decomposition, which are approximate “patches” in a sense inspired by Ancona and Bressan [2] in their study of feedback stabilization. They introduced and investigate the properties of a particular class of *discontinuous feedbacks*, the so-called *patchy feedbacks*.

The following definition gives the fundamental concept of a patch.

Definition 1. Let $\Omega \subset \mathbb{R}^d$ be an open domain with smooth boundary $\partial\Omega$ and f be a smooth vector field defined on a neighborhood of $\overline{\Omega}$. We say that the pair

(Ω, f) is a patch if Ω is a positive-invariant region for f , i.e. at every boundary point $y \in \partial\Omega$ the inner product of f with the outer normal n satisfies

$$\langle f(y), n(y) \rangle < 0.$$

By means of a superposition of patches, we get the notion of a patchy vector field on a domain $\Omega \subset \mathbb{R}^d$ and they have shown that these can be used to define discontinuous feedbacks stabilizing the system. However, their method is not constructive so an effort has been made to transform this approach into an algorithm. Clearly, from the numerical point of view, the approximation will produce patches which will be “almost invariant” with respect to the optimal dynamics driving the system. Their boundaries can be rather complicated, but this has the advantage that we do not need to apply any transmission condition between them.

Following [9], let us introduce two rectangular (structured) grids. The first grid should be rather *coarse* because it is used for preliminary (and fast) computations only. It will be denoted by \tilde{G} and its nodes by $\tilde{x}_1, \dots, \tilde{x}_{\tilde{N}}$, where \tilde{N} is the total number of nodes. We will denote the space step for this grid by $\tilde{k} := \Delta x_{\text{coarse}}$ and the approximate solution of the Eq. (6) on this grid by \tilde{U}_P .

The second grid is instead *fine*, being the grid where we actually want to compute the numerical solution of the equation. It will be denoted by G and its nodes by x_1, \dots, x_N , where N is the total number of nodes ($N \gg \tilde{N}$). We will denote the space step for this grid by $k := \Delta x_{\text{fine}}$ and the solution of the Eq. (6) on this grid by U_P . We also choose the number R of subdomains (patches) to be used in the patchy decomposition and we divide the target Ω_0 in R parts denoted by Ω_0^j , with $j = 1, \dots, R$.

The patchy method can be described as follows.

Patchy Algorithm

- Step 1. (Computation on \tilde{G}). We solve the equation on \tilde{G} by means of the classical domain decomposition algorithm (e.g. where the subdomains are rectangles). For coherence we choose the (static) decomposition made by R subdomains (as the number of patches). This leads to \tilde{U}_P .
- Step 2. (Interpolation on G). We define the function $U_P^{(0)}$ on the fine grid G by interpolation of the values \tilde{U}_P . Then, we compute the approximate optimal control

$$\tilde{a}^*(x_i) = \arg \min_{a \in A} \{I[U_P^{(0)}](x_i + h_{i,a}f(x_i, a)) + h_{i,a}\}, \quad x_i \in G. \quad (12)$$

Even if \tilde{a}^* is defined on G , we still use the symbol “tilde” to stress that optimal controls are computed using only coarse information. We delete \tilde{G} and \tilde{U}_P .

- Step 3. (Main cycle) For every $j = 1, \dots, R$,

Step 3.1. (Creation of j -th patch). Using the (coarse) optimal control \tilde{a}^* , we find the nodes of the grid G that have the part Ω_0^j of the target in

their numerical domain of dependence. This procedure defines the j -th patch, naturally following the (approximate) optimal dynamics. This step will be detailed later in this section.

Step 3.2. (Computation in j -th patch). As initial guess we initialize the j -th solution equal to $+\infty$ on the j -th patch and equal to 0 on the part Ω_0^j of the target. Then, we apply iteratively the scheme (8) in the j -th patch until convergence is reached. Finally, the j -th solution is copied in the matrix that will contain the global solution U_P .

Details on Step 3.1. The basic idea we adopt here is to divide the whole domain starting from a partition of the target only, and let the dynamics make a partition of the rest of the domain. To this end we use the approximation of the optimal control given by \tilde{a}^* to obtain a domain decomposition fully compliant to the dynamics. More precisely, we divide the target Ω_0 in R parts, each associated to a colour indexed by a number $j = 1, \dots, R$. Assume for instance that Ω_0 is a ball at the center of the domain and focus on the subset of the target with a generic colour j , denoted by Ω_0^j , see Fig. 1(a). The goal is to find the subset of the domain Ω which has Ω_0^j as numerical domain of dependence. To do that, we initialize the grid nodes with the values ϕ_i as follows:

$$\phi_i = \begin{cases} 1, & x_i \in \Omega_0^j \\ 0, & x_i \in G \setminus \Omega_0^j \end{cases}, \quad i = 1, \dots, N.$$

Then we solve the following *ad hoc* discrete equation,

$$\phi_i = I[\phi](x_i + h_i f(x_i, \tilde{a}^*(x_i))), \quad i = 1, \dots, N, \quad (13)$$

which is similar to the fixed-point scheme (7) for the main equation. Here $h_i > 0$ is chosen in such a way that $|h_i f(x_i, \tilde{a}^*(x_i))| = k$. Once the computation is completed, the whole domain will be divided in three zones:

$$A_1^j = \{x_i : \phi_i = 1\}, \quad A_2^j = \{x_i : \phi_i = 0\}, \quad A_3^j = \{x_i : \phi_i \in (0, 1)\},$$

see Fig. 1(b). Note that A_3^j will be nonempty because the interpolation operator I in the scheme (13) mixes the values ϕ_i through a convex combination, thus producing values in $[0, 1]$ even if the initial datum is in $\{0, 1\}$. Since we need a sharp division of the domain, we “project” the colour j into a binary value

$$\hat{\phi}_i = \begin{cases} 1, & \phi_i \geq \frac{1}{2} \\ 0, & \phi_i < \frac{1}{2} \end{cases}, \quad i = 1, \dots, N \quad (14)$$

and then we define the subdomain $\Omega^j = \{x_i \in G \setminus \Omega_0^j : \hat{\phi}_i = 1\}$ as the j -th patch, see Fig. 1(c). Once all the patches $j = 1, \dots, R$ are computed, they are assembled together on the grid G . Thus the grid results to be divided into R patches, each associated to a different colour, as shown in Fig. 1(d). Note that the boundaries of every patch are aligned with the coordinate axes.

The main point here is that the patches Ω^j 's are constructed to be invariant with respect to the optimal dynamics, meaning that the solution of the equation

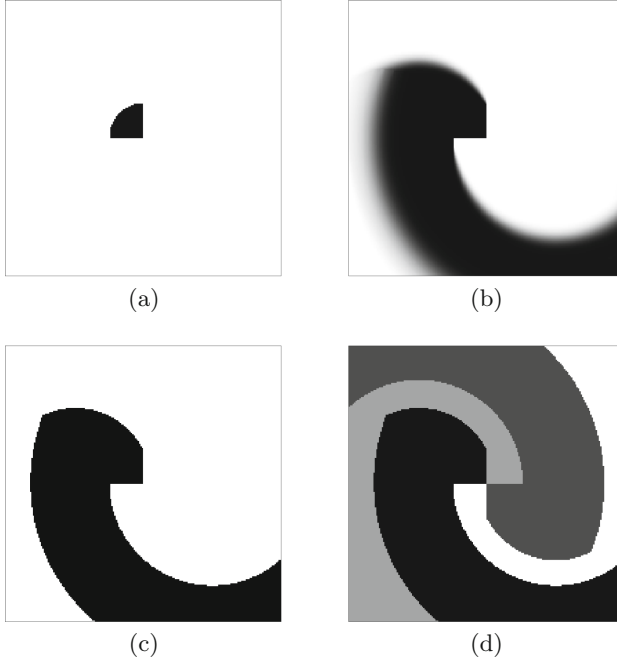


Fig. 1. Creation of patches for a test dynamics, $R = 4$, Ω_0 = small ball in the centre: (a) Select a subdomain Ω_0^j of the target Ω_0 . (b) Find the nodes which depend, at least partially, on Ω_0^j . (c) Define Ω^j projecting the color in a binary value. (d) Assemble all patches.

in each patch will not depend on the solution in other patches. This is equivalent to state that there is no crossing information through the boundaries of the patches.

We stress that Step 3.1 of the algorithm is not expensive, even if it is performed on the fine grid G . The reason for that is the employment of the pre-computed optimal control \tilde{a}^* in the Eq. (13), which avoids the evaluation of the minimum (see the scheme (8)). Moreover, the stopping rule for the fixed point iterations used to solve (13) can be very rough, since we project the colors at the end and then we do not need precise values.

Numerical examples

We will test the method described above against two minimum time problems of the form (1). The numerical domain is always $\Omega = [-2, 2]^2$.

Test 1 (Eikonal) : $d = 2$, $f(x_1, x_2, a) = a$, $A = B_2(0, 1)$, $\Omega_0 = B_2(0, 0.5)$.

Test 2 (Fan) : $d = 2$, $f(x_1, x_2, a) = |x_1 + x_2 + 0.1|a$, $A = B_2(0, 1)$, $\Omega_0 = \{x_1 = 0\}$.

In Fig. 2 we show the patchy decomposition for the two dynamics described above in the case $R = 8$, $N_c = 32$, $\tilde{N} = 50$ and $N = 100$. We also superimpose the optimal vector field $f(x, \tilde{a}^*)$ to show that patches are (almost) invariant with

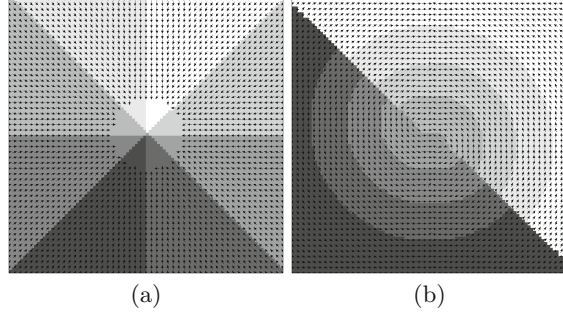


Fig. 2. Patchy decompositions with $R = 8$, $N_c = 32$, $\tilde{N} = 50$ and $N = 100$. For visualization purposes not all the arrows are shown. (a) Eikonal, (b) Fan.

respect to the optimal dynamics. Indeed, only a few arrows cross from a patch to another. Note that patches cover the whole domain but they are not equivalent in terms of area, even if the target Ω_0 was divided in $R = 8$ equal parts to generate the decomposition.

It is interesting to compare the solution U_P of the patchy algorithm with that of the classical domain decomposition method U_{DD} , both computed on the same fine grid by means of the scheme (8). Let us denote by E_P the difference $E_P := U_P - U_{DD}$ that in the following will be referred to as *patchy error*. In particular we study the norms

$$\|E_P\|_1 := k^d \sum_{i=1}^N |E_{P_i}| \quad \text{and} \quad \|E_P\|_\infty := \max_{i=1,\dots,N} |E_{P_i}|$$

depending on the space steps \tilde{k} and k . This error is exclusively due to the fact that patches are not completely dynamics-invariant and then it will be considered as a degree of the invariance of the patchy decomposition. Let us stress that we apply state constraint boundary conditions on the patches.

We report the results for $R = 16$, which is the largest number of patches and also the worst case we tested. Indeed, the error E_P necessarily increases as R increases because the number of boundaries increases. We present the results for the Test 2 in Table 1, similar errors appear in other tests.

Table 1. Patchy error $\|E_P\|_1$ ($\|E_P\|_\infty$). Dynamics: Fan, $N_c = 32$, $R = 16$

	$k = 0.08$	$k = 0.04$	$k = 0.02$	$k = 0.01$	$k = 0.005$
$\tilde{k} = 0.08$	1.393 (3.023)	0.123 (1.507)	0.037 (0.315)	0.017 (0.263)	0.011 (0.263)
$\tilde{k} = 0.04$	—	0.114 (1.502)	0.032 (0.149)	0.011 (0.095)	0.006 (0.095)
$\tilde{k} = 0.02$	—	—	0.032 (0.111)	0.011 (0.061)	0.004 (0.037)
$\tilde{k} = 0.01$	—	—	—	0.011 (0.079)	0.004 (0.037)
$\tilde{k} = 0.005$	—	—	—	—	0.004 (0.037)

We see that the first line of each table reports in many cases unsatisfactory results, caused by the excessive roughness of the grid \tilde{G} (see the case $\tilde{k} = 0.08$, corresponding to $\tilde{N} = 50$). Even the case $\tilde{k} = k = 0.08$ (i.e. the grid is not refined at all) is not satisfactory. This can be explained by recalling that the computations on the two grids are not identical because the second one employs state constraints boundary conditions. If the grid is not fine enough, the error due to the boundary conditions is large, and tends to propagate inside each patch. Conversely, if \tilde{G} has at least 100 nodes per dimension ($\tilde{k} \leq 0.04$), the behaviour of the error is surprisingly good because it decreases as k decreases (for any fixed \tilde{k}) and $\|E_P\|_1$ is of the same order of k itself. Note that the L^∞ error is always larger than the L^1 error. Quite often we find a very small number of nodes with a large error near the boundaries of the patches, especially at those nodes where two patches and the target meet. This mainly affect the L^∞ error but not the L^1 error.

4 Fast Marching and Fast Sweeping Methods

The second technique which has been proposed to reduce the computational load and memory allocations is based on the localization of the algorithm. At every iteration only a subset of the grid (the active region) is taken into account and the solution is computed just on the nodes belonging to this region. An important feature of this method is the fact that the value at a single node is computed only a finite number of times and this allows to show that the solution is obtained in a finite number of iterations. Here we list and briefly describe some iterative and *single-pass* methods for solving HJ equations.

Let us sketch the Fast Marching Method (FMM) [25, 27] introduced as a fast solver for the eikonal equation. Despite the standard global iterative method, the nodes are visited in a solution-dependent order, producing a *single-pass* method: the algorithm itself finds a correct order for processing the grid nodes. The order which is determined satisfies the *causality* principle, i.e. the computation of a node is declared completed only if its value cannot be affected by the future computation. To this end, at each step the grid is divided in three regions: *ACC*, where computation is definitively done, *CONS*, where computation is going on and *FAR*, where computation is not done yet. Then, the node in *CONS* with the minimal value enters *ACC*, its first neighbours enter *CONS* (if not yet in) and are (re)computed.

Following [23], we remark that this *minimum-value rule* corresponds to compute the value function T step by step in the ascending order (i.e., from the simplex containing $-\nabla T$). It follows that *CONS* expands under the gradient flow of the solution itself, which is exactly equivalent to say that *CONS* is, at each step, an approximation of a level set of the value function. In the case of isotropic eikonal Eq. (6), the gradient of the solution coincides with the characteristic field of the HJ equation, hence FMM computes the correct solution. Moreover, FMM still works for problems with mild anisotropy, where gradient lines and characteristics define small angles and lie, at each point, in the same

simplex of the underlying grid. On the other hand, when a strong anisotropy comes into play, as for a general anisotropic eikonal equation, FMM fails and there is no way to compute the viscosity solution following its level sets. Finally, we remark that FMM is also a *local* method, since each node is computed by means of first neighbors nodes only and *CONS* is one-cell thick. Moreover, FMM computes the same solution of the global fixed point method (ITM), provided the same scheme is employed.

The *Fast Sweeping Method (FSM)* [24, 28] is similar to the global fixed point iteration ITM, but the grid is visited in a multiple-direction predefined order. Usually, a rectangular grid is iteratively swept along four directions: $N \rightarrow S$, $E \rightarrow W$, $S \rightarrow N$, and $W \rightarrow E$, where N, S, E , and W stand for North, South, East, and West, respectively. This method has been shown to be much faster than ITM, but (as ITM) it is neither *local* nor *single-pass*. A well known exception is given by the eikonal equation, for which it is proved that only 1 sweep (i.e. four visits of the whole grid) is enough to reach convergence (see [28] for details). FSM computes the same solution of ITM, provided the same scheme and the same stopping rule are employed.

Numerical examples

Let us compare the methods in terms of velocity and accuracy.

Test 1. Let us choose $\mathcal{T} = (0, 0)$, $f(x, y, a) \equiv a$. We know the exact solution of the corresponding eikonal equation which is $T(x, y) = \sqrt{(x^2 + y^2)}$.

As one can see in Table 2 the two fast marching methods (FM-FD and FM-SL, respectively based on a finite difference and a semi-Lagrangian solver) give a big speed-up in the computation. The fast sweeping method (FS-SL) gives good results but is generally slower than the fast marching methods.

Test 2: state constraint problem. $\mathcal{T} = (-1, -1)$.

$$f(x, y, a) = \begin{cases} 0 & (x, y) \in ([0, 0.5] \times [-2, 1.5]) \cup ([1, 1.5] \times [-1.5, 2]) \\ a & \text{elsewhere.} \end{cases}$$

Table 2. Errors for Test 1.

Method	Δx	L^∞ error	L^1 error	CPU time (s)
FM-FD	0.08	0.0875	0.7807	0.5
FM-SL	0.08	0.0329	0.3757	0.7
SL (46 it)	0.08	0.0329	0.3757	8.4
FS-SL	0.08	0.0329	0.3757	0.8
FM-FD	0.04	0.0526	0.4762	2.1
FM-SL	0.04	0.0204	0.2340	3.1
SL (86 it)	0.04	0.0204	0.2340	60
FS-SL	0.04	0.0204	0.2340	3.2
FM-FD	0.02	0.0309	0.2834	9.4
FM-SL	0.02	0.0122	0.1406	14
SL (162 it)	0.02	0.0122	0.1406	443.7
FS-SL	0.02	0.0122	0.1406	12.5

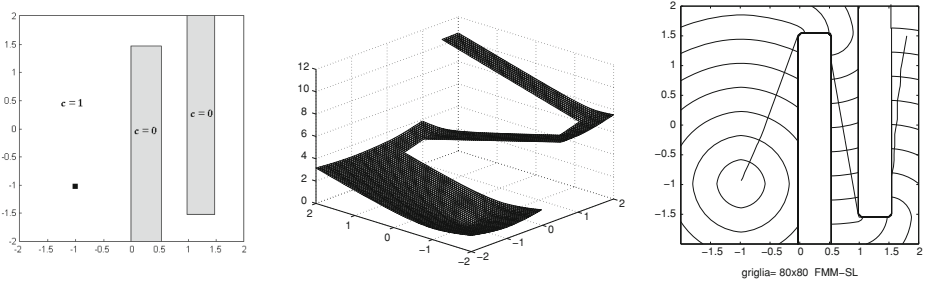


Fig. 3. Domain of the equation (left), value function T (center) and level sets of T with one optimal trajectory (right).

In this test the dynamics has been set to 0 on the obstacles to enforce the state constraint. The results are shown in Fig. 3.

5 An Accelerated Policy Iteration Algorithm with Smart Initialization

Let us conclude with an accelerated iterative algorithm which is constructed upon the building blocks previously introduced in Sect. 2. We aim to an efficient formulation exploiting the main computational features of both value and policy iteration algorithms. As it has been stated in [21], there exists a theoretical equivalence between both algorithms, which guarantees a rather wide convergence framework. However, from a computational perspective, there are significant differences between both implementations. A first key factor can be observed in Fig. 4, which shows, for a two-dimensional minimum time problem, the typical situation arising with the evolution of the error measured with respect to the optimal solution, when comparing value and policy iteration algorithms. To achieve a similar error level, policy iteration requires considerable fewer iterations than the value iteration scheme, as quadratic convergent behavior is reached faster for any number of nodes in the state-space grid. Despite the observed computational evidence, a second issue is observed when examining the policy iteration algorithm in more detail. That is, the sensitivity of the method with respect to the choice of the initial guess of the control field. It can be seen that different initial admissible control fields can lead to radically different convergent behaviors. While some guesses will produce quadratic convergence from the beginning of the iterative procedure, others can lead to an underperformant value iteration-like evolution of the error. This latter is computationally expensive, because it translates into a non-monotone evolution of the subiteration count of the solution of Eq. (2).

A final relevant remark goes back to Fig. 4, where it can be observed that for coarse meshes, the value iteration algorithm generates a fast error decay up to a higher global error. This, combined with the fact that value iteration algorithms are rather insensitive to the choice of the initial guess for the value

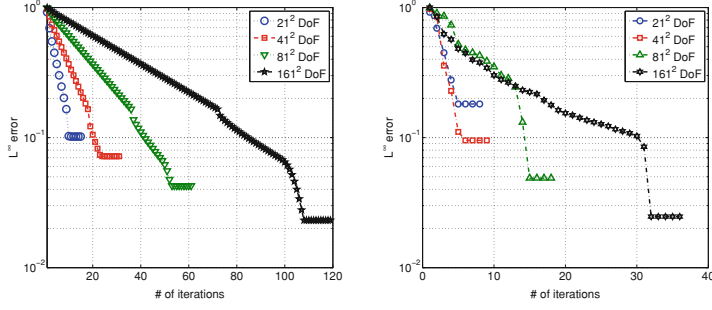


Fig. 4. Error evolution in a 2D problem: value iteration (left) and policy iteration (right).

function, are crucial points for the construction of our accelerated algorithm. The accelerated policy iteration algorithm is based on a robust initialization of the policy iteration procedure via a coarse value iteration which will yield to a good guess of the initial control field (see [1] for details).

Data: Coarse mesh G_c and Δt_c , fine mesh G_f and Δt_f , initial coarse guess V_c^0 , coarse-mesh tolerance ϵ_c , fine-mesh tolerance ϵ_f .

begin

Coarse-mesh value iteration step: perform Algorithm 1

Input: $G_c, \Delta t_c, V_c^0, \epsilon_c$

Output: V_c^*

forall the $x_i \in G_f$ **do**

$V_f^0(x_i) = I_1[V_c^*](x_i)$

$A_f^0(x_i) = \underset{a \in A}{\operatorname{argmin}} \{e^{-\Delta t} I_1[V_f^0](x_i + f(x_i, a)) + \Delta t\}$

end

Fine-mesh policy iteration step: perform Algorithm 2

Input: $G_f, \Delta t_f, V_f^0, A_f^0, \epsilon_f$

Output: V_f^*

end

Algorithm 3: (API) Accelerated Policy Iteration

Numerical examples

The next two cases are based on a two-dimensional eikonal equation. For both problems, common settings are given by

$$f(x, y, a) = \begin{pmatrix} \cos(a) \\ \sin(a) \end{pmatrix}, \quad A = [-\pi, \pi], \quad \Delta t = 0.8\Delta x.$$

What differentiates the problems is the domain and target definitions.

Test 1: $\Omega =]-1, 1[^2$, target $\mathcal{T} = (0, 0)$.

Test 2: $\Omega =]-2, 2[^2$, $\mathcal{T} = \{x \in \mathbb{R}^2 : \|x\|_2 \leq 1\}$.

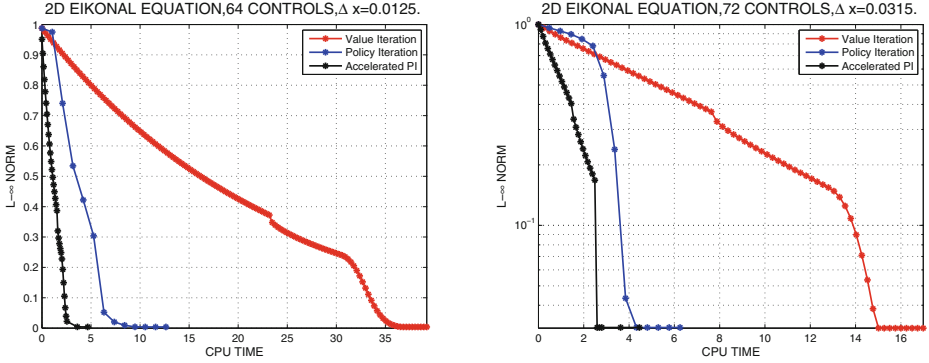


Fig. 5. Error evolution in 2D eikonal equations: Test 1 (left) and Test 2 (right).

Reference solutions are considered to be the distance function to the respective targets, which is an accurate approximation provided that the number of possible control directions is large enough. For Test 1, with a discretization of the control space into a set of 64 equidistant points, it can be seen that API provides a speedup of $8\times$ with respect to VI over fine meshes despite the large set of discrete control points. Figure 5 illustrates, for both problems, the way in which the API idea acts: pre-processing of the initial guess of PI leads to proximity to a “quadratic convergence neighborhood”; fast error decay that coarse mesh VI has in comparison with the fine mesh VI is clearly noticeable.

In Test 2 we have a “fat” target. In general, larger or more complicated targets represent a difficulty in terms of the choice of the minimizing control, which translates into a larger number of iterations. In this case, the CPU time spent in the pre-processing is significant to the overall CPU time, but increasing this ratio in order to reduce its share will lead to an underperformant PI part of the algorithm.

6 Conclusions

We illustrated some recent results concerning the numerical approximation of optimal control problems governed by ordinary differential equations. The above methods can be combined in order to obtain fast algorithms and accurate solutions. For example one can use a patchy domain decomposition to set up a parallel algorithm and inside every patch use an Accelerated Policy Iteration (API) or a Fast Marching method. Several open problems still remain. For example, we would like to prove error bounds for the patchy domain decomposition and for the API acceleration method. Moreover, we continue our investigations to extend these methods to differential games and to the control of partial differential equations.

References

1. Alla, A., Falcone, M., Kalise, D.: An efficient policy iteration algorithm for dynamic programming equations. *SIAM J. Sci. Comp.* (still to appear)
2. Ancona, F., Bressan, A.: Patchy vector fields and asymptotic stabilization. *ESAIM: Control Optim. Calc. Var.* **4**, 445–471 (1999)
3. Bardi, M., Capuzzo Dolcetta, I.: *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Birkhäuser, Boston (1997)
4. Bardi, M., Falcone, M.: An approximation scheme for the minimum time function. *SIAM J. Control Optim.* **28**, 950–965 (1990)
5. Bardi, M., Falcone, M.: Discrete approximation of the minimal time function for systems with regular optimal trajectories. In: Bensoussan, A., Lions, J.L. (eds.) *Analysis and Optimization of Systems. Lecture Notes in Control and Information Sciences*, vol. 144, pp. 103–112. Springer, Heidelberg (1990)
6. Bellman, R.: *Dynamic Programming*. Princeton University Press, Princeton (1957)
7. Cacace, S., Cristiani, E., Falcone, M.: A local ordered upwind method for Hamilton-Jacobi and Isaacs equations. In: *Proceedings of the 18th IFAC World Congress*, pp. 6800–6805 (2011)
8. Camilli, F., Falcone, M., Lanucara, P., Seghini, A.: A domain decomposition method for Bellman equations. In: Keyes, D.E., Xu, J. (eds.) *Domain Decomposition methods in Scientific and Engineering Computing, Contemporary Mathematics*, vol. 180, pp. 477–483. AMS, Providence (1994)
9. Cacace, S., Cristiani, E., Falcone, M., Picarelli, A.: A patchy dynamic programming scheme for a class of Hamilton-Jacobi-Bellman equations. *SIAM J. Sci. Comp.* **34**, 2625–2649 (2012)
10. Carlini, E., Falcone, M., Ferretti, R.: An efficient algorithm for Hamilton-Jacobi equations in high dimension. *Comput. Vis. Sci.* **7**, 15–29 (2004)
11. Carlini, E., Falcone, M., Forcadell, N., Monneau, R.: Convergence of a generalized fast marching method for an Eikonal equation with a velocity changing sign. *SIAM J. Numer. Anal.* **46**, 2920–2952 (2008)
12. Crandall, M.G., Lions, P.L.: Two approximation of solutions of Hamilton-Jacobi equations. *Math. Comput.* **43**, 1–19 (1984)
13. Cristiani, E.: A fast marching method for Hamilton-Jacobi equations modeling monotone front propagations. *J. Sci. Comput.* **39**, 189–205 (2009)
14. Cristiani, E., Falcone, M.: Fast semi-Lagrangian schemes for the Eikonal equation and applications. *SIAM J. Numer. Anal.* **45**, 1979–2011 (2007)
15. Falcone, M.: Numerical solution of dynamic programming equations, Appendix A in [3].
16. Falcone, M.: Some remarks on the synthesis of feedback controls via numerical methods. In: Menaldi, J.L., Rofman, E., Sulem, A. (eds.), *Optimal Control and Partial Differential Equations*, pp. 456–465. IOS Press (2001)
17. Falcone, M., Ferretti, R.: *Semi-Lagrangian approximation schemes for linear and Hamilton-Jacobi equations*. SIAM, Philadelphia (2014)
18. Falcone, M., Lanucara, P., Seghini, A.: A splitting algorithm for Hamilton-Jacobi-Bellman equations. *Appl. Numer. Math.* **15**, 207–218 (1994)
19. Howard, R.A.: *Dynamic programming and Markov processes*. Wiley, New York (1960)
20. Navasca, C., Krener, A.J.: Patchy solutions of Hamilton-Jacobi-Bellman partial differential equations. In: Chiuso, A., et al. (eds.) *Modeling, Estimation and Control. Lecture Notes in Control and Information Sciences*, vol. 364, pp. 251–270. Springer, Heidelberg (2007)

21. Puterman, M.L., Brumelle, S.L.: On the convergence of policy iteration in stationary dynamic programming. *Math. Oper. Res.* **4**(1), 60–69 (1979)
22. Sethian, J.A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge (1999)
23. Sethian, J.A., Vladimirsky, A.: Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms. *SIAM J. Numer. Anal.* **41**, 325–363 (2003)
24. Tsai, Y., Cheng, L., Osher, S., Zhao, H.: Fast sweeping algorithms for a class of Hamilton-Jacobi equations. *SIAM J. Numer. Anal.* **41**, 673–694 (2004)
25. Tsitsiklis, J.N.: Efficient algorithms for globally optimal trajectories. *IEEE Trans. Autom. Control* **40**, 1528–1538 (1995)
26. Quarteroni, A., Valli, A.: *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Oxford (1999)
27. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci. USA* **93**, 1591–1595 (1996)
28. Zhao, H.: A fast sweeping method for Eikonal equations. *Math. Comp.* **74**, 603–627 (2005)

Large-Scale Scientific Computing

9th International Conference, LSSC 2013, Sozopol,
Bulgaria, June 3-7, 2013. Revised Selected Papers

Lirkov, I.; Margenov, S.; Waśniewski, J. (Eds.)

2014, XV, 654 p. 173 illus., Softcover

ISBN: 978-3-662-43879-4