## A PATCHY DYNAMIC PROGRAMMING SCHEME FOR A CLASS OF HAMILTON–JACOBI–BELLMAN EQUATIONS\*

# SIMONE CACACE<sup>†</sup>, EMILIANO CRISTIANI<sup>†</sup>, MAURIZIO FALCONE<sup>‡</sup>, AND ATHENA PICARELLI<sup>†</sup>

**Abstract.** In this paper we present a new algorithm for the solution of Hamilton–Jacobi– Bellman equations related to optimal control problems. The key idea is to divide the domain of computation into subdomains which are shaped by the optimal dynamics of the underlying control problem. This can result in a rather complex geometrical subdivision, but it has the advantage that every subdomain is invariant with respect to the optimal dynamics, and then the solution can be computed *independently* in each subdomain. The features of this dynamics-dependent domain decomposition can be exploited to speed up the computation and for an efficient parallelization, since the classical transmission conditions at the boundaries of the subdomains can be avoided. For their properties, the subdomains are *patches* in the sense introduced by Ancona and Bressan [*ESAIM Control Optim. Calc. Var.*, 4 (1999), pp. 445–471]. Several examples in two and three dimensions illustrate the properties of the new method.

 ${\bf Key}$  words. patchy methods, Hamilton–Jacobi equations, parallel methods, minimum time problem, semi-Lagrangian schemes

### AMS subject classifications. 65N55, 49L20

#### **DOI.** 10.1137/110841576

**1.** Introduction. The numerical solution of partial differential equations (PDEs) 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, and 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 an optimal 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 mentioning that the characterization via the Pontryagin principle gives only necessary conditions for the optimal trajectory and optimal open-loop control. In addition, the numerical procedures for solving the associated system of ordinary differential equations can be very complicated. In real applications, finding a good initial guess for the co-state often requires a long and tedious trial-and-error procedure. This is why it can be useful to combine the DPP and the Pontryagin approaches, using the approximate value function to compute a suitable initial guess for the co-state, as proposed in [21].

<sup>\*</sup>Submitted to the journal's Methods and Algorithms for Scientific Computing section July 20, 2011; accepted for publication (in revised form) May 8, 2012; published electronically October 2, 2012. This work was supported by AFOSR grant FA9550-10-1-0029, ITN - Marie Curie grant 264735-SADCO, and SAPIENZA 2009 "Analisi ed approssimazione di modelli differenziali nonlineari in fluidodinamica e scienza dei materiali."

http://www.siam.org/journals/sisc/34-5/84157.html

<sup>&</sup>lt;sup>†</sup>Dipartimento di Matematica, SAPIENZA - Università di Roma, 00185 Rome, Italy (cacace@ mat.uniroma1.it, emiliano.cristiani@gmail.com, athena.picarelli@gmail.com).

 $<sup>^{\</sup>ddagger}$ Corresponding author. Dipartimento di Matematica, SAPIENZA - Università di Roma, 00185 Rome, Italy (falcone@mat.uniroma1.it).

In this paper we mainly focus on the *minimum time problem*, which is associated to the following Hamilton–Jacobi–Bellman equation:

(1.1) 
$$\begin{cases} \max_{a \in A} \left\{ -f(x,a) \cdot \nabla u(x) - 1 \right\} = 0, & x \in \mathbb{R}^d \setminus \Omega_0, \\ u(x) = 0, & x \in \Omega_0, \end{cases}$$

where d is the dimension of the state,  $A \subset \mathbb{R}^m$  is the set of admissible controls,  $\Omega_0$  is the target to be reached in minimal time, and  $f : \mathbb{R}^d \times A \to \mathbb{R}^d$  is the dynamics of the system. The value function  $u : \mathbb{R}^d \to \mathbb{R}$  at the point x is the minimum time to reach the target starting from x (note that  $u(x) = +\infty$  if the target is not reachable). For numerical purposes, the equation is solved in a bounded domain  $\Omega \supset \Omega_0$ , so that boundary conditions on  $\partial\Omega$  are also needed. A rather standard choice when one has no additional information on the solution and deals with target problems is imposing state constraint boundary conditions.

The techniques used to obtain a numerical approximation of the viscosity solution of (1.1) have been mainly based on finite differences [19, 33] and semi-Lagrangian schemes [22, 24]. More recently, finite element methods based on discontinuous Galerkin approximations have been proposed, due to their ability to deal with nonregular functions, which is the typical case in the framework of viscosity solutions [17, 18, 35]. It is important to note that traditional approximation schemes, presented, for example, in [19] and [22], are based on fixed-point iterations, meaning that the solution is computed at each node of the grid at every iteration until convergence. 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 these full-grid schemes is  $O(M^{d+1})$ . We easily conclude that classical algorithms are very expensive when the state dimension is  $d \geq 3$ , although they are rather efficient for low-dimensional control problems as shown in [22] (see also the book [24]).

The "curse of dimensionality" has been attacked in many ways, and new techniques have been proposed to accelerate convergence and/or to reduce memory allocation. In [9] the authors proposed an algorithm that allows one to allocate only a small portion of the grid at every iteration. Another proposal to reduce the computational effort is given by the Fast Marching method, introduced in [32, 37] for the Eikonal equation. 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  $O(M^d \log M^d)$ . Despite the efficiency of the Fast Marching method, at present its application to more general equations of the form (1.1) is not an easy task, and it is still under investigation [13, 16, 20, 34].

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. Unlike Fast Marching 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 [38] and, more recently, for rather general Hamiltonians [36].

Another strategy is based on the decomposition of the domain  $\Omega$ . The problem is actually solved in subdomains  $\Omega^j$ ,  $j = 1, \ldots, R$ , 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. Depending on the choice of the subdomains  $\Omega^j$ , we can have overlapping regions or interfaces between the subdomains. This is a delicate point, since at each iteration it is necessary to exchange information between processors to make subdomains communicate with each other. Without this communication the result will not be correct. The interested reader can find in the book [31] a comprehensive introduction to domain decomposition techniques, whereas for an application to Hamilton–Jacobi equations we refer the reader to [14, 25].

Finally, a decomposition of the domain based on the concept of "patchy feedbacks" has been proposed. It was introduced by Ancona and Bressan in [2], where the authors studied the problem of the asymptotic stabilization of a control system. Their main result (see Theorem 2.1 in section 2.3) states that, under suitable assumptions on the control system, stabilization can be obtained by means of a special feedback control which is piecewise constant on a particular partition of the domain. Such a partition has the fundamental property that each part, or "patch," is positive-invariant with respect to the optimal dynamics driving the system. This is the spirit of what we call the "patchy method." Unfortunately, the result of Ancona and Bressan is purely theoretical, and their patchy decomposition turns out not to be constructive. Thus, one has to face the problem of a numerical approximation of such a dynamics-invariant domain decomposition.

A first example of the discrete patchy method has been proposed by Navasca and Krener in [28, 29]. The authors adopt a formal method developed by Al'brekht [1] that essentially translates the Hamilton–Jacobi–Bellman equation associated to a control problem into a system of algebraic equations, whose unknowns are the coefficients of the expansions in power series of the cost and optimal feedback. This gives an approximate solution in a small neighborhood of the origin, which is the first patch of their domain decomposition. The solution is then extended to new patches around the first one by picking some boundary points from where optimal trajectories emanate (they are computed numerically backward in time). Those points define the centers of new neighborhoods that can be used to restart the method. The solution is then obtained iteratively by fitting together the approximations in all the patches. More recently, it has been shown that this technique can be extended to obtain highorder accuracy in the regions where the value function is smooth (see [26] for more details).

Despite the high speed of the method, which actually does not use any grid, there are many open questions regarding its application. The first limitation is the strong regularity assumptions on the solution necessary to set the problem in these terms. Indeed, it is well known that the simplest control problems may have optimal controls which are not even continuous [5]. The second crucial point is the construction of the patchy decomposition that, in the examples contained in [28, 29], appears not to be completely invariant with respect to the optimal dynamics. This makes the solution rather inaccurate, especially near the boundaries of the patches.

The goal of this paper is to present and investigate a new patchy technique based on a semi-Lagrangian scheme that leads to a dynamics-dependent partition of the domain  $\Omega$ . Subdomains turn out to be, up to a discretization error, *invariant with respect to the optimal dynamics*, meaning that optimal dynamics do not cross the boundaries of the subdomains or, equivalently, that boundaries of the subdomains are optimal trajectories to the target. The algorithm consists of three main steps: First, a rough solution of (1.1) is computed on a coarse grid. Then, the feedback optimal control is used to obtain the dynamics-dependent domain decomposition. Finally, the solution of the equation is computed on a (much) finer grid, *independently* in each subdomain, by serial or parallel computation. We will see that the invariance of the subdomains can be exploited to lower the iterations needed to reach convergence and to parallelize efficiently the computation on distributed-memory architectures, since there is no need for communication among processors.

The paper is organized as follows. Section 2 is devoted to the presentation of the semi-Lagrangian scheme and the classical domain decomposition technique for (1.1). Moreover, we briefly describe the results on patchy methods which have been proved by Ancona and Bressan [2]. In section 3 we present the patchy domain decomposition method and our algorithm to split the domain into invariant subdomains. We discuss there several issues related to the implementation of the method and its parallelization. Section 4 is devoted to the numerical tests on control problems in dimension two, and section 5 presents some improvements of the basic algorithm. Finally, in section 6 we present some tests in dimension three.

2. Background. In this section we briefly introduce the numerical scheme used to discretize (1.1) and the classical domain decomposition technique, including an algorithm that will be used in the following. Next, we recall the notion of patchy decomposition and the result by Ancona and Bressan [2] concerning the asymptotic stabilization of control systems by means of patchy feedbacks, which inspired our patchy numerical method.

**2.1. The semi-Lagrangian scheme.** We introduce a structured grid G on  $\Omega$  with nodes  $x_i$ , i = 1, ..., N. We also denote by  $\mathring{G}$  the internal nodes of G and by  $\partial G$  its boundary, whose nodes act as *ghost nodes*. We map all the values at the nodes onto an N-dimensional vector  $U = (U_1, ..., U_N)$ . Let us denote by  $h_{i,a} > 0$  a (fictitious) time step, possibly depending on the node  $x_i$  and control a (see the book [24] for details), and by k > 0 the space step. By a standard semi-Lagrangian discretization [6, 7, 22] of (1.1), it is possible to obtain the following scheme in fixed-point form:

$$(2.1) U = F(U),$$

where  $F : \mathbb{R}^N \to \mathbb{R}^N$  is defined componentwise by

$$[F(U)]_i = \begin{cases} \min_{a \in A} \left\{ I \left[ U \right] \left( x_i + h_{i,a} f(x_i, a) \right) + h_{i,a} \right\}, & x_i \in \mathring{G} \setminus \Omega_0, \\ 0, & x_i \in \Omega_0 \cap G, \\ +\infty, & x_i \in \partial G. \end{cases}$$

The discrete value function U is extended on the whole space  $\Omega$  by the interpolation operator I. In order to fix ideas, one should consider the linear interpolation in  $\mathbb{R}^d$  described in [15], but other choices are available [24].

We choose the time step  $h_{i,a}$  such that  $|h_{i,a}f(x_i, a)| = k$  for every  $i = 1, \ldots, N$ 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 around  $x_i$ . The minimum over A is evaluated by direct comparison, discretizing the set A with  $N_c$  points. Note that defining  $F(U) = +\infty$  on  $\partial G$  corresponds to imposing state constraint boundary conditions. The final iterative scheme reads

(2.2) 
$$U^{(n+1)} = F(U^{(n)}), \qquad U^{(0)} = \begin{cases} 0 & \text{on } \Omega_0 \cap G, \\ +\infty & \text{otherwise.} \end{cases}$$

We refer the reader to [22, 24] for details and convergence results. With the discrete value function U in hand, we can obtain a discrete feedback map  $a_k^* : \Omega \to A$  just defining

(2.3) 
$$a_k^*(x) := \arg\min_{a \in A} \{ I[U](x + h_{x,a}f(x,a)) + h_{x,a} \} \,.$$

Under rather general assumptions [23], it can be shown that this is an approximation of the feedback map constructed for the continuous problem as

(2.4) 
$$a^*(x) := \arg \max_{a \in A} \{-f(x, a) \cdot \nabla u(x) - 1\}.$$

A detailed discussion on the construction of the feedback maps and of the optimal trajectory solution of the minimum time problem can be found in [22, 23].

**2.2. Domain decomposition method.** The domain decomposition method allows one to split the problem in  $\Omega$  into R subproblems in subsets  $\Omega^j$ ,  $j = 1, \ldots, R$ , such that  $\Omega = \Omega^1 \cup \cdots \cup \Omega^R$ . For every pair  $j \neq \ell$  of indices corresponding to adjacent subdomains  $\Omega^j$  and  $\Omega^\ell$ , let us denote by  $\Omega^{j\ell}$  the nonempty overlapping zone  $\Omega^j \cap \Omega^\ell$ , which is assumed to contain at least one grid cell.

We also denote by  $N^j$  the number of nodes of  $\Omega^j$ , by  $U^j$  the restriction of U to  $\Omega^j$ , and by  $F^j : \mathbb{R}^{N^j} \to \mathbb{R}^{N^j}$  the restriction of the operator F in (2.1) to  $\Omega^j$ . Then, we define globally in  $\Omega$  the following splitting operator  $F_{\text{SPLIT}} : \mathbb{R}^N \to \mathbb{R}^N$ , given componentwise by

$$[F_{\text{SPLIT}}(U)]_i \equiv \begin{cases} [F^j(U^j)]_i & \text{if } \exists j \text{ such that } x_i \in \Omega^j \setminus \bigcup_{\ell \neq j} \Omega^{j\ell}, \\ \\ \min_{j \,:\, x_i \in \Omega^j} \{ [F^j(U^j)]_i \} & \text{otherwise.} \end{cases}$$

Following [25], it is easy to prove that fixed-point iterations for F and  $F_{\text{SPLIT}}$  lead to the same solution.

We now describe a simple algorithm to compute the fixed point of  $F_{\text{SPLIT}}$ . DOMAIN DECOMPOSITION ALGORITHM.

Step 1. (Initialization) For n = 0 the initial guess  $U^{(0)} \in \mathbb{R}^N$  is fixed to 0 on the nodes corresponding to the target  $\Omega_0$  and  $+\infty$  elsewhere.

Step 2. (Computation)  $U^{(n+1/2)}$  is computed separately in every subdomain  $\Omega^{j}$  by

$$U^{j,(n+1/2)} = F^j(U^{j,(n)}), \qquad j = 1, \dots, R.$$

Step 3. (Coupling)

$$U_i^{(n+1)} = \begin{cases} U_i^{j,(n+1/2)} & \text{if } \exists j \text{ such that } x_i \in \Omega^j \setminus \bigcup_{\ell \neq j} \Omega^{j\ell} \\ & \\ \min_{j \,:\, x_i \in \Omega^j} \left\{ U_i^{j,(n+1/2)} \right\} & \text{otherwise.} \end{cases}$$

Step 4. (Stopping criterion) If  $||U^{(n+1)} - U^{(n)}||_{\infty} > tol$ , go to Step 2 with  $n \leftarrow n+1$ ; otherwise stop.

In order to speed up the convergence of the above algorithm, we use iterations of Gauss–Seidel type, meaning that we employ the updated values of the nodes as soon as they are available.

The domain decomposition method can be used in a natural way to parallelize the computation. Simply, each subdomain is assigned to a processor. Synchronization among processors is performed at each iteration after step 3. Note that, as a limiting case, the algorithm can also be used in serial computation (one processor).

From now on the final solution computed by the domain decomposition algorithm will be denoted by  $U_{DD}$ .

## A2630 S. CACACE, E. CRISTIANI, M. FALCONE, A. PICARELLI

2.3. Patchy feedbacks and stabilization for control systems. The inspiring idea at the basis of the numerical method we present in the next section is the notion of a "patch." It was introduced by Ancona and Bressan in [2] in the context of the stabilization of control systems. Let us recall here for completeness their main definitions and results. We refer the interested reader to [2] for details and to [3, 4, 10, 11] for further results on this subject.

We consider the control system

(2.5) 
$$\dot{y}(t) = f(y(t), a(t)), \quad a(t) \in A,$$

assuming that the control set  $A \subset \mathbb{R}^m$  is compact and the dynamics  $f : \mathbb{R}^d \times A \to \mathbb{R}^d$ is sufficiently smooth. Moreover, we choose as admissible controls all the functions *a* belonging to

$$\mathcal{A} := \{ a : (0, \infty) \to \mathcal{A} \mid a \text{ is measurable} \}.$$

For every point  $x \in \mathbb{R}^d$  and admissible control  $a_0 \in \mathcal{A}$ , we denote by  $y(\cdot; x, a_0)$  the absolutely continuous function defined on some maximal interval  $[0; \tau^{\max}(x, a_0))$  satisfying the system (2.5) with initial condition y(0) = x and control  $a_0$ .

The following definition extends to control systems the classical notion of stability. DEFINITION 2.1. The system (2.5) is said to be globally asymptotically control-

- lable (to the origin) if the following hold:
  - 1. For each  $x \in \mathbb{R}^d$  there exists some admissible control  $a_0$  such that the trajectory  $t \to y(t) = y(t; x, a_0)$  is defined for all  $t \ge 0$  (i.e.,  $\tau^{\max}(x, a_0) = \infty$ ) and  $y(t) \to 0$  as  $t \to \infty$ .
  - 2. For each  $\varepsilon > 0$  there exists  $\delta > 0$  such that for each  $x \in \mathbb{R}^d$  with  $|x| < \delta$  there is an admissible control  $a_0$  as in condition 1 such that  $|y(t)| < \varepsilon$  for all  $t \ge 0$ .

Given an asymptotically controllable system, a classical problem is finding a feedback control  $a = \psi(y) : \mathbb{R}^d \to A$  such that all the trajectories of the corresponding closed-loop system

$$(2.6) \qquad \qquad \dot{y} = f(y, \psi(y))$$

tend asymptotically to the origin. Since this problem may not admit any solution in the class of *continuous feedbacks*, Ancona and Bressan introduce and investigate the properties of a particular class of *discontinuous feedbacks*, the so-called *patchy feedbacks*.

The following definition introduces the fundamental concept of a patch.

DEFINITION 2.2. Let  $\mathcal{P} \subset \mathbb{R}^d$  be an open domain with smooth boundary  $\partial \mathcal{P}$ , and let g be a smooth vector field defined on a neighborhood of  $\overline{\mathcal{P}}$ . We say that the pair  $(\mathcal{P}, g)$  is a patch if  $\mathcal{P}$  is a positive-invariant region for g, i.e., at every boundary point  $y \in \partial \mathcal{P}$  the inner product of g with the outer normal n satisfies

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

Then, by means of a superposition of patches, we get the notion of a patchy vector field on a domain  $\Omega \subset \mathbb{R}^d$ .

DEFINITION 2.3. We say that  $g: \Omega \to \mathbb{R}^d$  is a patchy vector field if there exists a family of patches  $\{(\Omega^{\alpha}, g^{\alpha}) : \alpha \in \mathcal{I}\}$  such that

- *I* is a totally ordered index set,
- the open sets  $\Omega^{\alpha}$  form a locally finite covering of  $\Omega$ ,

• the vector field g can be written in the form

$$g(y) = g^{\alpha}(y) \qquad if \ y \in \Omega^{\alpha} \setminus \bigcup_{\beta > \alpha} \Omega^{\beta}.$$

We use  $(\Omega, g, (\Omega^{\alpha}, g^{\alpha})_{\alpha \in \mathcal{I}})$  to denote the patchy vector field and the family of patches. By applying the previous definitions to the closed-loop system (2.6), we define a *patchy feedback* control as a piecewise constant map  $\psi : \mathbb{R}^d \to A$  such that the vector field  $g(y) := f(y, \psi(y))$  is a patchy vector field. More precisely, we have the following definition.

DEFINITION 2.4. Let  $(\Omega, g, (\Omega^{\alpha}, g^{\alpha})_{\alpha \in \mathcal{I}})$  be a patchy vector field. Assume that there exist control values  $\psi^{\alpha} \in A$  such that, for each  $\alpha \in \mathcal{I}$ ,

$$g^{\alpha}(y) = f(y, \psi^{\alpha}) \qquad \forall y \in \Omega^{\alpha} \setminus \bigcup_{\beta > \alpha} \Omega^{\beta}.$$

Then, the piecewise constant map

$$\psi(y) = \psi^{\alpha} \qquad if \ y \in \Omega^{\alpha} \setminus \bigcup_{\beta > \alpha} \Omega^{\beta}$$

is called a patchy feedback control on  $\Omega$ .

DEFINITION 2.5. A patchy feedback control  $\psi : \mathbb{R}^d \setminus \{0\} \to A$  is said to asymptotically stabilize the closed-loop system (2.6) with respect to the origin if the following hold:

- 1. For each  $x \in \mathbb{R}^d \setminus \{0\}$  and for every trajectory  $y(\cdot)$  of (2.6) starting from x one has  $y(t) \to 0$  as  $t \to \tau^{\max}(x)$ .
- 2. For each  $\varepsilon > 0$  there exists  $\delta > 0$  such that, for each  $x \in \mathbb{R}^d \setminus \{0\}$  with  $|x| < \delta$ and for every trajectory  $y(\cdot)$  of (2.6) starting from x, one has  $|y(t)| < \varepsilon$  for all  $0 \le t < \tau^{\max}(x)$ .

Finally, the main result of Ancona and Bressan can be summarized as follows.

THEOREM 2.1. If the system (2.5) is asymptotically controllable, then it admits an asymptotically stabilizing patchy feedback control.

**3.** The patchy domain decomposition. In this section we introduce our new numerical method for solving equations of Hamilton–Jacobi–Bellman type. In particular we focus on the minimum time problem (1.1). 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 the definitions of the previous section. Indeed, we will see that these patches turn out to be (almost) invariant with respect to the optimal dynamics driving the system, meaning that the optimal dynamics do not cross their boundaries. Even if this construction could lead to a rather complicated domain decomposition, it has the clear advantage that we do not need to apply any transmission condition between subdomains.

Let us introduce two rectangular (structured) grids. The first grid is rather *coarse* because it is used for preliminary (and fast) computations only. It will be denoted by  $\tilde{G}$  and its nodes by  $\tilde{x}_1, \ldots, \tilde{x}_{\tilde{N}}$ , where  $\tilde{N}$  is the total number of nodes. We will denote the space step for this grid by  $\tilde{k}$  and the approximate solution of (1.1) 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

 $x_1, \ldots, x_N$ , where N is the total number of nodes  $(N \gg N)$ . We will denote the space step for this grid by k and the solution of (1.1) 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  $\Omega_0$  into R parts denoted by  $\Omega_0^j$ , with  $j = 1, \ldots, R$ .

The patchy method can be described as follows.

PATCHY ALGORITHM.

- Step 1. (Computation on  $\widetilde{G}$ ) We solve the equation on  $\widetilde{G}$  by means of the classical domain decomposition algorithm described in section 2.2. For coherence we choose a (static) decomposition with R subdomains (as the number of patches). This leads to the function  $\widetilde{U}_P$ .
- 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

(3.1) 
$$a_{\widetilde{k}}^*(x_i) = \arg\min_{a \in A} \{ I[U_P^{(0)}](x_i + h_{i,a}f(x_i, a)) + h_{i,a} \}, \quad x_i \in G.$$

Even if  $a_{\tilde{k}}^*$  is defined on G, we still use the subscript  $\tilde{k}$  to stress that the optimal control is computed using only coarse information. We delete  $\tilde{G}$  and  $\tilde{U}_P$ .

- Step 3. (Main cycle) For every j = 1, ..., R:
  - Step 3.1. (Creation of the *j*th patch) Using the (coarse) optimal control  $a_{\tilde{k}}^*$ , we find the nodes of the grid *G* that have the part  $\Omega_0^j$  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 the *j*th patch) We iteratively apply the scheme (2.2) in the *j*th patch until convergence. Boundary conditions will be discussed later in this section.
- Step 4. (Merging) All the solutions are merged together. This leads to the final solution  $U_P$ .

Details on Step 3.1. The basic idea we adopt here is dividing the whole domain starting from a partition of the target only, and letting the dynamics make a partition of the rest of the domain (see [12] for a similar idea in the context of parallelization of the Fast Marching method for the Eikonal equation). More precisely, once the target  $\Omega_0$  is divided into R parts, we associate each part to a color indexed by a number  $j = 1, \ldots, R$ . Assume, for instance, that  $\Omega_0$  is a ball at the center of the domain, and focus on the subset of the target with a generic color j, denoted by  $\Omega_0^j$ ; see Figure 3.1(a). The goal is finding the subset of the domain  $\Omega$  which has  $\Omega_0^j$  as numerical domain of dependence. First, we initialize the grid nodes with the values  $\phi_i$  as follows:

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

Then, employing the approximation of the optimal control given by  $a_{\tilde{k}}^*$ , we solve the following ad hoc discrete equation:

(3.2) 
$$\phi_i = I[\phi](x_i + h_i f(x_i, a^*_{\widetilde{\mu}}(x_i))), \qquad i = 1, \dots, N_i$$

which is similar to the fixed-point scheme (2.1) for the main equation. Here  $h_i > 0$  is chosen in such a way that  $|h_i f(x_i, a_{\tilde{k}}^*(x_i))| = k$ . Once the computation is completed,



FIG. 3.1. Creation of patches for a test dynamics, R = 4,  $\Omega_0 = small ball in the center:$  $(a) Select a subdomain <math>\Omega_0^j$  of the target  $\Omega_0$ . (b) Find the nodes which depend, at least partially, on  $\Omega_0^j$ . (c) Define  $\Omega^j$  projecting the color in a binary value. (d) Assemble all patches.

the whole domain will be divided into three zones:

$$\Lambda_1^j = \{x_i : \phi_i = 1\}, \quad \Lambda_2^j = \{x_i : \phi_i = 0\}, \quad \Lambda_3^j = \{x_i : \phi_i \in (0, 1)\};$$

see Figure 3.1(b). Note that  $\Lambda_3^j$  will be nonempty because the interpolation operator I in the scheme (3.2) mixes the values  $\phi_i$  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 color j into a binary value

(3.3) 
$$\widehat{\phi}_i = \begin{cases} 1, & \phi_i \ge \frac{1}{2}, \\ 0, & \phi_i < \frac{1}{2}, \end{cases} \quad i = 1, \dots, N,$$

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 Figure 3.1(c). Once all the patches  $j = 1, \ldots, R$  are computed, they are assembled together on the grid G. Thus the grid ends up being divided into R patches, each associated to a different color, as shown in Figure 3.1(d).

The main point here is that patches  $\Omega^{j}$  are constructed to be invariant with respect to the optimal dynamics, meaning that the solution of the equation in each patch will not depend on the solution in other patches. This is equivalent to stating that there is no information crossing 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 this is the employment of the precomputed optimal control  $a_{\tilde{k}}^*$  in (3.2), which avoids the evaluation of the minimum (see the scheme (2.2)). Moreover, the stop criterion for the fixed-point iterations used to solve (3.2) can be very rough, since we project the colors at the end and then we do not need precise values.

Details on boundary conditions. In Step 3.2 of the algorithm, the computation of the value function is performed independently in each patch; thus we have to impose boundary conditions on the boundaries of the patches. A natural choice is the employment of  $U_P^{(0)}$  (obtained in Step 2) as Dirichlet boundary condition, but in some early tests [30] we observed that this choice leads to reasonable results for  $U_P$  even if the domain decomposition is completely incorrect (i.e., patches are not at all invariant). More precisely, if the decomposition is not invariant, the accuracy of the final solution  $U_P$  is comparable to that of  $U_P^{(0)}$ ; otherwise it is comparable to that of  $U_{DD}$  computed on the same fine grid (which is the best one can do). This point will be discussed later in section 4.2. As a first study we prefer to impose a boundary condition which does not require any a priori information on the solution outside the patches, in order to check whether they are genuinely independent. This motivated us to use state constraint boundary conditions, which force the optimal direction  $f(x_i, a_k^*)$  to point inside the patch (see Figure 3.2 and its caption). This choice produces an error that can be evaluated comparing  $U_P$  with  $U_{DD}$  and that we consider as a degree of invariance of the patchy decomposition.



FIG. 3.2. (a) Discretization versus decomposition invariance: The node  $x_i$  belongs to the dark gray patch, but it is influenced by the light gray patch (through the coarse control  $a_k^*(x_i)$ ). (b) The cure: State constraint boundary conditions force the patches to be completely independent but change the optimal vector field  $f(x_i, a_k^*(x_i))$ , producing an error in the final solution.

Let us remark that once we have shown that patches are actually (almost) invariant, we *can* impose Dirichlet boundary conditions, further improving the quality of the solution. This also allows one to handle the case of dynamics such that state constraint condition cannot be satisfied everywhere (i.e., there is no control allowing the state to remain in the patch), as in the example discussed in section 4.6. Remark 3.1 (patches as a partition of G). We have no guarantee that patches  $\Omega^{j}$  do not overlap or that they cover the whole domain. On the overlapping zones we can simply choose a color at random. Instead, if they do not cover all the domain, we can repeat the computation in the uncolored nodes by relaxing the condition in (3.3), i.e., choosing a different value for 1/2. Alternatively, in the case of isolated uncolored points, we can assign to them the color of their neighbors.

How to parallelize the algorithm. The patchy algorithm can be parallelized in two ways.

- *Method* 1. Patches are processed *one after the other*, and the computation in each patch is parallel, assigning a batch of nodes to each processor.
- *Method* 2. Patches are *distributed among processors*, and the computation of each patch is serial.

The first strategy is designed for shared-memory architectures and gives priority to *saving CPU time*, while the second strategy is designed for distributed-memory architectures and gives priority to *saving memory allocation*. The difference in CPU time comes from the fact that, using the first method, processors are active all the time, while, using the second method, it can happen that one processor finishes its jobs and there are no more patches to be computed, so it remains idle. We stress again that, employing Method 2, the independence of the patches allows processors to not communicate until the end of their task, saving heavy overhead in distributed-memory architectures.

All tests presented in this paper are performed by implementing Method 1 on a shared-memory architecture, and in the following we will always refer to this choice.

4. Numerical investigation in two dimensions. In this section we first list the dynamics considered for the numerical tests. Then we investigate the optimality of the patchy decomposition and the performance of the algorithm with respect to the classical domain decomposition.

Numerical tests were performed on a server Supermicro 8045C-3RB using 1 CPU Intel Xeon Quad-Core E7330 2.4 GHz with 32 GB RAM, running under the Linux Gentoo operating system.

4.1. Choice of benchmarks. We will test the method described above against three minimum time problems of the form (1.1). They are listed in Table 4.1. The numerical domain is  $\Omega = [-2, 2]^2$  for all tests. In Figure 4.1 we show the patchy decomposition for the three dynamics described above in the case R = 8,  $\tilde{N} = 50^2$ , and  $N = 100^2$ . The number of points used for discretizing A is  $N_c = 32$ . We also superimpose the coarse optimal vector field  $f(x, a_{\tilde{k}}^*)$  to show that patches are (almost) invariant with respect to the optimal dynamics. Indeed, only a few arrows cross from one patch to another, which is exactly the case discussed in Figure 3.2(a). Note that patches cover the whole domain, but in general they are not equivalent in terms of area, even if the target  $\Omega_0$  was divided into R = 8 equal parts to generate the decomposition.

TABLE 4.1Two-dimensional numerical tests.

Name	d	$f(x_1, x_2, a)$	A	$\Omega_0$
Eikonal	2	a	$B_2(0,1)$	$B_2(0, 0.5)$
Fan	2	$( x_1 + x_2  + 0.1)a$	$B_2(0,1)$	$\{x_1 = 0\}$
Zermelo	2	2.1a + (2,0)	$B_2(0,1)$	$B_2(0, 0.5)$



FIG. 4.1. Patchy decompositions with R = 8,  $N_c = 32$ ,  $\tilde{N} = 50^2$ , and  $N = 100^2$ . For visualization purposes not all the arrows are shown. (a) Eikonal, (b) Fan, (c) Zermelo, (d) a detail of Fan.

4.2. Optimality of the patchy decomposition. In this section we 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 (2.2). Let us denote by E the difference

$$E := U_P - U_{DD},$$

which in the following will be referred to as  $patchy\ error.$  In particular we study the quantities

$$E_1 := \frac{1}{N} \sum_{i=1}^{N} |E_i|$$
 and  $E_{\infty} := \max_{i=1,\dots,N} |E_i|$ 

as the grid nodes  $\tilde{N}$  and N change. Error E is exclusively due to the fact that patches are not completely dynamics-invariant, and then it will be considered as a degree of the

TABLE 4.2 Patchy error  $E_1$  ( $E_{\infty}$ ). Dynamics: Eikonal,  $N_c = 32$ , R = 16.

	$N = 50^{2}$	$N = 100^{2}$	$N = 200^2$	$N = 400^{2}$	$N = 800^{2}$
$\widetilde{N} = 50^2$	$0.02725 \\ (0.960)$	0.01719 (1.856)	0.00637 (0.048)	$0.00406 \\ (0.034)$	0.00300 (0.026)
$\widetilde{N} = 100^2$	-	$\begin{array}{c} 0.00550 \\ (0.046) \end{array}$	0.00181 (0.023)	$0.00087 \\ (0.042)$	0.00031 (0.008)
$\widetilde{N}=200^2$	_	_	0.00237 (0.029)	$0.00075 \\ (0.013)$	$0.00025 \\ (0.008)$
$\widetilde{N} = 400^2$			—	$0.00069 \\ (0.016)$	$0.00037 \\ (0.010)$
$\widetilde{N} = 800^2$			_		0.00025 (0.008)

TABLE 4.3 Patchy error  $E_1$  ( $E_{\infty}$ ). Dynamics: Fan,  $N_c = 32$ , R = 16.

	$N = 50^{2}$	$N = 100^{2}$	$N = 200^{2}$	$N = 400^{2}$	$N=800^2$
$\widetilde{N} = 50^2$	$0.08706 \\ (3.023)$	0.00769 (1.507)	$0.00231 \\ (0.315)$	$0.00106 \\ (0.263)$	$0.00069 \\ (0.263)$
$\widetilde{N}=100^2$	—	0.00712 (1.502)	$0.00200 \\ (0.149)$	$0.00069 \\ (0.095)$	$0.00037 \\ (0.095)$
$\widetilde{N}=200^2$	—		0.00200 (0.111)	$0.00069 \\ (0.061)$	$\begin{array}{c} 0.00025 \ (0.037) \end{array}$
$\widetilde{N} = 400^2$	—	_	—	$0.00069 \\ (0.079)$	$0.00025 \\ (0.037)$
$\widetilde{N} = 800^2$	_		_		$0.00025 \\ (0.037)$

TABLE 4.4 Patchy error  $E_1$  ( $E_{\infty}$ ). Dynamics: Zermelo,  $N_c = 32$ , R = 16.

	$N = 50^{2}$	$N = 100^{2}$	$N = 200^{2}$	$N = 400^2$	$N = 800^{2}$
$\widetilde{N} = 50^2$	0.01069 (0.293)	0.00994 (0.059)	$0.00606 \\ (0.057)$	0.00162 (0.027)	0.00037 (0.016)
$\widetilde{N} = 100^2$		$0.00631 \\ (0.063)$	$0.00206 \\ (0.041)$	$0.00069 \\ (0.023)$	$0.00025 \\ (0.016)$
$\widetilde{N}=200^2$	_	_	0.00244 (0.039)	0.00075 (0.023)	0.00025 (0.016)
$\widetilde{N} = 400^2$			—	$0.00069 \\ (0.020)$	0.00031 (0.015)
$\widetilde{N} = 800^2$	_		_	_	0.00025 (0.016)

invariance of the patchy decomposition. Let us stress that we employ state constraint boundary conditions, as discussed in the previous section (see Figure 3.2(b)).

We report the results for R = 16, which is the largest number of patches and also the worst case we tested. Indeed, error E necessarily increases as R increases because the number of boundaries increases. Results for the three dynamics are shown in Tables 4.2–4.4.

We see that the first row of each table  $(\tilde{N} = 50^2)$  reports in some cases unsatisfactory results caused by the excessive roughness of the grid  $\tilde{G}$ . Even the case  $\tilde{N} = N = 50^2$  (i.e., the grid is not refined at all) is not satisfactory. This can be explained by recalling that, even if  $\tilde{N} = N$ , the computations on the two grids are not identical because the second one employs state constraint boundary conditions. In the other cases, the behavior of the error is very good because *it decreases as* N *increases* (for any fixed  $\tilde{N}$ ). As pointed out in section 3, if the patches were not invariant with respect to the dynamics, as in the classical domain decomposition algorithm, we would not expect such a behavior here, because of the missing information across the patches. Thus, this shows that our patches are actually (almost) independent.

Tables 4.2–4.4 also show that the  $E_{\infty}$  is always larger than  $E_1$ , meaning that the error is concentrated in small regions. Indeed, 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 affects  $E_{\infty}$  but not  $E_1$ . Finally we note that the results are similar for the three dynamics, showing a good robustness even for highly rotating vector fields like that of Fan dynamics.

Figures 4.2(a)–(c) report the function E for the three tests, showing one of the most interesting features of the new method, i.e., the patchy error is concentrated along the boundaries of the patches and does not propagate in the interior. In the Eikonal and Zermelo cases the error starts from the target and increases as long as characteristics go away. In the Fan case, on the other hand, the largest error is found where patches and the target meet. Note that in the Eikonal case (Figure 4.2(a))



FIG. 4.2. Patchy error  $E, \tilde{N} = 50^2 \rightarrow N = 100^2$  for (a) Eikonal, (b) Fan, (c) Zermelo. In (d) we show a detail of the patchy decomposition for the Fan dynamics, together with the optimal vector field  $f(x, a_k^*)$  computed by means of the final patchy solution  $U_P$ .

no error is found where patch boundaries are aligned to the grid, since the optimal direction naturally points inside the patch and the state constraint boundary condition has no effect. Figure 4.2(d) shows a detail of the Fan decomposition along with the approximate optimal vector field computed by means of the final solution  $U_P$ . The effect of the state constraints is perfectly visible (arrows point unnaturally inward), confirming the fact that each patch is computed independently.

Figure 4.3 shows the value function  $U_P$  and its level sets for the Eikonal test. Here we see small perturbations where patches meet. It is interesting to note that they meet forming a hollow and not a discontinuity.



FIG. 4.3. Patchy solution for the Eikonal dynamics. (a) Value function and (b) a detail of its level sets. Small hollows are visible in correspondence of the lines  $\{x = y\}$  and  $\{x = -y\}$ ; cf. Figure 4.2(a).

**4.3.** Comparison of CPU times. In this section we compare the patchy algorithm with the classical domain decomposition algorithm in terms of CPU time. In the case of parallel computation, we will always refer to the *wall clock time*, and not to the sum of CPU times devoted to the task by each CPU running it.

Let us first explain why we expect that the patchy algorithm overcomes the domain decomposition algorithm, considering again the Eikonal case with R = 8; see Figure 4.1(a). If we visit the nodes in a single predefined order (i.e., we do *not* implement the fast sweeping technique [38] or similar ones), the eight subdomains need a different number of iterations to reach convergence. This is due to the fact that for some of them the visiting order corresponds to the upwind direction, while for the other subdomains the visiting order corresponds to the downwind direction. If we do not know a priori that the eight subdomains are invariant with respect to the optimal dynamics, we cannot stop the computation in a subdomain before computations in *all* subdomains are fully completed, because at any moment new information can enter, making necessary new computations. On the contrary, if we know a priori that subdomain as soon as the solution reaches convergence. Note that this argument is not related to the parallelization and holds also if only one core is employed. In addition, the higher the number of subdomains is, the more efficient is the computation.

## A2640 S. CACACE, E. CRISTIANI, M. FALCONE, A. PICARELLI

In Table 4.5 we report the execution times (in seconds) for the single steps of the patchy algorithm. Times for Step 3.1 are the most interesting because this step is expected to be the slowest one after Step 3.2 (main computation on the fine grid). Thus, time spent in Step 3.1 could completely neutralize the advantage we hope to get in the subsequent main computation. As we can see, Step 3.1 is much more costly than Steps 1 and 2, but not so much compared with the main computation.

TABLE 4.5 CPU time. Cores: 4,  $N_c = 32$ , Grid:  $\tilde{N} = 100^2 \rightarrow N = 800^2$ , R = 16.

	Step 1	Step 2	Step 3.1 (all $j$ 's)	Step 3.2 (all $j$ 's)
Eikonal	2	1	23	409
Fan	2	2	52	796
Zermelo	2	1	30	512

Tables 4.6–4.9 report the CPU time (in seconds) for the three dynamics of Table 4.1 as a function of the number of cores (1, 2, 4) and the number of patches (R = 2, 4, 8, 16). They also report the speed-up (in parentheses) obtained by the parallelization method, Method 1 (see section 3). For the Eikonal test we also vary the number of discrete controls ( $N_c = 16$  and 32). These results are compared with the best outcome of the domain decomposition method obtained by varying the number of domains (again 2, 4, 8, 16).<sup>1</sup>

TABLE 4.6 CPU time (speed-up). Dynamics: Eikonal.  $N_c = 16$ . Grid:  $\tilde{N} = 100^2 \rightarrow N = 800^2$ .

	R = 2	R = 4	R = 8	R = 16	Best DD
1 core	1547	1076	1058	933	1571
2 cores	845(1.83)	595(1.81)	574(1.84)	504(1.85)	820(1.92)
4 cores	459(3.37)	325(3.31)	317(3.34)	271(3.44)	415(3.79)

 $\begin{array}{c} \text{TABLE 4.7}\\ CPU \ time \ (speed-up). \ Dynamics: \ Eikonal. \ N_c=32. \ Grid: \ \widetilde{N}=100^2 \rightarrow N=800^2. \end{array}$ 

	R = 2	R = 4	R = 8	R = 16	Best DD
1 core	2702	1897	1843	1623	2785
2 cores	1462(1.85)	998(1.90)	968(1.90)	872(1.86)	1430(1.95)
4 cores	771(3.50)	532(3.57)	514(3.59)	435(3.73)	716(3.89)

TABLE 4.8 CPU time (speed-up). Dynamics: Fan.  $N_c = 32$ . Grid:  $\tilde{N} = 100^2 \rightarrow N = 800^2$ .

	R = 2	R = 4	R = 8	R = 16	Best DD
1 core	3712	3322	3049	3172	4163
2 cores	2020(1.84)	1746(1.90)	1596(1.91)	1559(2.03)	2124 (1.96)
4 cores	1032 (3.60)	900(3.69)	841(3.63)	852(3.72)	1069 (3.89)

We see that the speed-up is very satisfactory and proves that the parallelization method, Method 1, that we implement here is sound. Moreover, we see that the

<sup>&</sup>lt;sup>1</sup>The CPU time of the domain decomposition method does not vary a lot when the number of domains is varied, but small differences are present. They are due to the different order in which nodes are visited and to the synchronization overhead at the end of each iteration.

LE	4.9	
----	-----	--

CPU time (speed-up). Dynamics: Zermelo.  $N_c = 32$ . Grid:  $\tilde{N} = 100^2 \rightarrow N = 800^2$ .

TAB

	R=2	R = 4	R = 8	R = 16	Best DD
1 core	3113	2675	2126	2018	3209
2 cores	1651 (1.89)	1404(1.91)	1111(1.91)	1054(1.91)	1640(1.96)
4 cores	871(3.57)	721(3.71)	584(3.64)	545(3.70)	825 (3.89)

CPU time decreases remarkably as the number of patches R increases. For R = 16 the CPU time is considerably smaller than that of the best domain decomposition method. This is one of the main results of the paper.

Differences among  $N_c = 16$  and  $N_c = 32$  are instead less clear, although the patchy algorithm should have an advantage for large  $N_c$  because of the smaller ratio between CPU time for Step 3.1 (one discrete control) and Step 3.2 ( $N_c$  discrete controls).

*Remark* 4.1. The fast sweeping technique can mitigate the performances of the patchy method, since it clears the differences between domains with the same number of nodes, but it cannot neutralize them completely. Indeed, the patchy algorithm has the clear advantage that no synchronization or crossing information among processors is needed. This is a great advantage when using distributed-memory parallel computers (for which Method 2 is designed), where communications are performed via cables connecting cluster nodes. This advantage is not really included in our experiments because our cores share a common RAM.

4.4. Patchy method with obstacles. We have also tried to use the patchy algorithm to solve a minimum time problem with Eikonal dynamics and obstacles. In Figure 4.4 we show the obstacles (one circle and one rectangle), the level sets of the solution, the patchy decomposition, and the patchy error E. The behavior of the patchy decomposition is correct because the dynamics drives the patches around the obstacles. If not influenced by the obstacles, the error is concentrated around the boundaries of the patches as expected. Instead, when a boundary meets an obstacle, the error can either stop propagating (see the circle) or spread out (see the rectangle, right side).



FIG. 4.4. Patches bypass the obstacles driven by the dynamics. (a) Obstacles (black), level sets of the value function, and patchy decomposition. (b) Patchy error E.

**4.5. Limitations of the patchy method.** The overall efficiency of the patchy method depends on the dynamics and on the shape of the target  $\Omega_0$ . Unfortunately, it is not always possible to get a suitable patchy decomposition which allows one to run the algorithm. This can happen, for example, if the target is very small; then it cannot be divided into R subdomains.

Another issue, much more difficult to fix, emerges whenever there is a large difference between the sizes of the patches and possibly some of them degenerate into a subset of a few grid nodes. This is the case of the classical "lunar landing" problem

d = 2,  $f(x_1, x_2, a) = (x_2, a)$ ,  $A = \{-1, 1\}$ ,  $\Omega_0 = B_2(0, \varepsilon)$ .

In this case the patchy decomposition consists of two large domains and R-2 smaller domains; see Figure 4.5(a). The small domains degenerate into sets of one dimension when  $\varepsilon$  tends to zero, because all the optimal trajectories tend to meet in only two switching lines.



FIG. 4.5. (a) Decomposition in R = 4 patches, (b) error function and level sets of the patchy solution (the solution is truncated at value u = 3 in order to remove the boundary effects which are very important for this dynamics), (c) patchy solution  $U_P$ , (d) a detail of the error function shown in (b).

A third dangerous case arises when some regions in  $\Omega_0$  are not reachable. If the dynamics make it impossible to reach the target from some point  $x \in \Omega$ , the value function is set to  $u(x) = +\infty$ . From the numerical point of view, the solution stays frozen at the value given as initial guess. At these points the optimal control (3.1) is not uniquely defined, and then the patchy decomposition cannot be built. On the other hand, in the unreachable regions, the solution is in some sense already computed; then the issue can be easily fixed by a slight modification of the algorithm. After Step 1 we locate the regions where  $\widetilde{U}_P$  is very large, and then we do not consider those regions in the rest of the computations.

4.6. Patchy decomposition for nontarget problems. In the case of nontarget problems we cannot in principle build the patchy decomposition. Indeed, we recall that our patchy decomposition starts from a decomposition of the target  $\Omega_0$ . Nevertheless, in some special situations it is still possible to achieve the patchy decomposition, using some a priori knowledge of the solution of the problem. This is the case of the infinite horizon problem associated to the *linear-quadratic regulator*, studied by Krener and Navasca in [29] as a test for their patchy method (see also [28] and the introduction for a brief description):

(4.1) 
$$\min_{a \in \mathbb{R}} \int_0^\infty \left( \frac{1}{2} (y_1^2 + y_2^2) + \frac{1}{2} a^2 \right) dt \quad \text{subject to} \quad \begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = a \end{cases}$$

with  $y_1(t=0) = x_1$  and  $y_2(t=0) = x_2$ . The exact value function for this problem is

$$u(x_1, x_2) = \frac{1}{2} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} \sqrt{3} & 1 \\ 1 & \sqrt{3} \end{pmatrix} (x_1 \ x_2),$$

and it is easy to check that the origin (0,0) is the only source of all characteristic curves. Then, using a small ball  $B_2(0,\varepsilon)$  as a fictitious target, we are able to generate the patchy decomposition and then run the algorithm normally. In Figure 4.5 we show the outcome of the simulation with the following parameters:  $\Omega = [-1,1]^2$ ,  $\varepsilon = 0.05$ , R = 4,  $N_c = 101$ , A = [-3,3],  $\tilde{N} = 100^2$ ,  $N = 200^2$ . Note that the choice R = 4 is due to limitations discussed in section 4.5. Moreover, we cannot impose state constraint boundary conditions at the boundaries of the patches, since at some nodes the dynamics in (4.1) do not point inside the patch to which they belong for any  $a \in A$ . As already discussed in section 3, this issue is solved by using  $U_P^{(0)}$  as Dirichlet boundary condition:

(4.2) 
$$U_P|_{\partial\Omega^j} = U_P^{(0)}|_{\partial\Omega^j}, \qquad j = 1, \dots, R.$$

Patchy errors are  $E_1 = 0.00012$  and  $E_{\infty} = 0.009$ . Note that they are generally smaller than those in Tables 4.2–4.4 (computed for other dynamics) because of the more favorable boundary condition (4.2).

5. Patchy method's add-ons. The patchy algorithm proposed in section 3 has a multigrid nature, meaning that the computation of the solution on a rough grid is needed to start the optimal domain decomposition. Once this preliminary effort is done, it appears to be natural to use all the information we have collected in order to speed up the algorithm. First, in the next tests we impose by default the boundary condition (4.2) (note that it becomes available only after the computation on the rough grid). Further multigrid advantages we can take into account are listed in the following:

,

- AO1. We use  $U_P^{(0)}$  computed in Step 2 as initial guess for Step 3.2. In this way we save some iterations to reach convergence.
- AO2. Before Step 3.2 we order the nodes belonging to each patch in such a way that they fit as closely as possible the *causality principle* [34]. For example, we can order the nodes with respect to their values. This ordering is optimal if the characteristic lines coincide with the gradient lines of the solution, as happens in the case of the Eikonal equation. In general this is not true; in any case, this ordering is often not too far from the optimal one.
- AO3. In Step 3.2 we reduce the number of discrete controls used in the numerical scheme, eliminating those controls which are "far" from the optimal one  $a_{\tilde{k}}^*(x_i)$  as computed by the first computation on the rough grid (Step 2). For example, if  $A = B_2(0, 1)$ , we can introduce a reduction factor r > 1 and replace A with the set

$$A_r = \left\{ a \in A : a \cdot a_{\widetilde{k}}^* \ge \cos\left(\frac{\pi}{r}\right) \right\}.$$

This is the only add-on which introduces a new error in the solution; in any case, it is negligible in most cases.

We point out that the patchy method can easily become an actual multigrid method. Indeed, we can in principle repeat the algorithm introducing a sequence of grids  $G_1, G_2, \ldots$ , each finer than the previous one, until the desired precision is reached.

In order to study the effect of the previously described add-ons, we introduce them separately and compare the CPU times with that of the basic algorithm. Then, we apply all the features together. Results are reported in Table 5.1.

Dynamics	Grid size	Basic	AO1	AO2	AO3	AO1 + AO2 + AO3
Eikonal	$100^2 \rightarrow 200^2$	20.0	19.2	9.6	9.1	5.7
Eikonal	$100^2 \rightarrow 400^2$	130.7	130.2	40.5	43.6	17.8
Eikonal	$100^2 \rightarrow 800^2$	928.1	924.6	238.8	298.1	100.6
Fan	$100^2 \rightarrow 200^2$	31.9	31.0	11.4	14.0	7.6
Fan	$100^2 \rightarrow 400^2$	209.8	205.7	43.5	72.3	20.6
Fan	$100^2 \rightarrow 800^2$	1571.9	1564.0	247.3	529.6	110.6
Zermelo	$100^2 \rightarrow 200^2$	23.2	22.6	11.5	10.7	6.7
Zermelo	$100^2 \to 400^2$	143.5	142.4	46.2	51.0	20.3
Zermelo	$100^2 \rightarrow 800^2$	1071.4	1057.9	290.1	345.5	111.3

 $\begin{array}{c} \text{TABLE 5.1}\\ \text{Effects of add-ons. Cores: 2, } R=8, \, N_c=32. \ \text{Controls reduced by a factor of 4.} \end{array}$ 

Note that CPU times for this test are lower than those in section 4.3 because of the more favorable boundary condition (4.2).

6. Numerical tests in three dimensions. We solve the three-dimensional (3D) minimal time problems of the form (1.1) listed in Table 6.1. The numerical domain is  $\Omega = [-2, 2]^3$  for all tests. For the first two tests we used  $N_c = 189$  discrete controls uniformly distributed on the unit sphere,<sup>2</sup> while for the last test we used only  $N_c = 9$  discrete controls in  $\{-5, 0, 5\}^2$ . The latter choice is motivated by the fact that using a larger number of discrete controls in  $[-5, 5]^2$  does not lead to a different

 $<sup>^{2}</sup>$ In order to define a uniform distribution of discrete controls on the unit sphere, we used the vertices of a geosphere obtained by recursion starting from an icosahedron.

## A PATCHY DP SCHEME FOR HJB EQUATIONS

	Table 6.1	
3D	$numerical\ tests.$	

Name	d	$f(x_1, x_2, x_3, a)$	A	$\Omega_0$
Eikonal 3D	3	a	$B_3(0,1)$	$B_3(0, 0.5)$
Fan 3D	3	$( x_1 + x_2 + x_3  + 0.1)a$	$B_3(0,1)$	$\{x_1 = 0\}$
Brockett 3D [8, 27]	3	$(a_1, a_2, x_1a_2 - x_2a_1)$	$[-5,5]^2$	$B_3(0, 0.25)$

TABLE 6.2 3D tests. Cores: 4, R = 8. Add-ons enabled (Eikonal and Fan: controls reduced by a factor of 4; Brockett: not reduced).

Dynamics	Grid size	CPU time	$E_1$	$E_{\infty}$
Eikonal 3D	$50^3 \rightarrow 100^3$	183	0.00052	0.035
Eikonal 3D	$50^3 \rightarrow 200^3$	1217	0.00045	0.042
Fan 3D	$50^3 \rightarrow 100^3$	165	0.00100	0.187
Fan 3D	$50^3 \rightarrow 200^3$	1269	0.00087	0.305
Brockett 3D	$50^3 \rightarrow 100^3$	132	0.00358	0.024
Brockett 3D	$50^3 \rightarrow 200^3$	1557	0.00258	0.020



FIG. 6.1. One level set of the value function for the Eikonal 3D dynamics.

result, since the optimal strategy always requires one to saturate the control to the extremal admissible values ( $\pm 5$  in this case).

Results are reported in Table 6.2. Considering the large number of discrete controls used for Eikonal 3D and Fan 3D, the CPU time is remarkable. Figure 6.1 shows a level set of the value function for the Eikonal 3D dynamics. It is perfectly apparent that error is located where the patches meet. Figure 6.2 shows instead the results for Fan 3D with 200<sup>3</sup> nodes. In Figures 6.2(a) and 6.2(b) we show the boundaries of the patches and some level sets of the solution, respectively. Level sets should be planes, but the state constraints imposed by the computational box  $\Omega$  bend them near  $\partial\Omega$ . In Figure 6.2(c) we show some optimal trajectories to the target.



FIG. 6.2. Fan 3D: (a) Decomposition with 8 patches, (b) level sets of the solution, and (c) some optimal trajectories to the target.

Results for the Brockett problem are different from those of the previous tests. First, CPU time turns out to be high with respect to the small number of discrete controls in use (just  $N_c = 9$  controls). This could be related to the fact that characteristics are broken lines (see Figure 6.3(c)) that neither go directly to the target, as in the Eikonal equation, nor bend slightly as for the Fan dynamics (see Figure 6.2(c)). Instead, they change direction instantaneously (see also control switch regions in Figure 6.3(b)), so that this dynamics takes much more time to move information through the domain. Second, the patchy error  $E_1$  is quite large if compared to the other dynamics (see Table 6.2). This depends on the fact that the patchy decomposition obtained for this dynamics is rather complicated (see Figure 6.3(a)); in particular, patches arrange themselves in (suggestive) sets with very large boundary areas, and this increases the number of nodes with large error E.



FIG. 6.3. Brockett 3D: (a) Decomposition with 8 patches, (b) regions with constant optimal controls, and (c) some optimal trajectories to the target.

**Concluding remarks and future directions.** In this paper we have proposed a new numerical method for optimal control problems which tries to mimic the "patchy decomposition" proposed by Ancona and Bressan [2]. We have investigated the serial implementation of the algorithm as well as one of the possible ways to parallelize it (Method 1), particularly suitable for shared-memory architectures. The new method is shown to be faster than the classical domain decomposition algorithm, since it avoids useless computations at nodes that have already reached convergence. At present, the main drawback of our approach is the fact that we have almost no control over the size of the patches, which depends only on the initial partition of the target and the dynamics.

Many points need to be investigated in the near future. The first is the parallelization on distributed-memory architectures, where patches are processed in parallel (Method 2). Moreover, we plan to improve the dynamics-dependent decomposition in such a way that the size of the patches is controllable. Finally, we aim at obtaining a convergence result for the scheme and, possibly, an a priori estimate for the patchy error. These results, coupled with the previous results by Ancona and Bressan, will produce a constructive provably convergent method for some classes of optimal control problems.

Acknowledgments. The authors would like to thank the anonymous referees for the useful suggestions which improved the quality of the paper. They also wish to thank the CASPUR Consortium for its technical support.

### REFERENCES

- E. G. AL'BREKHT, On the optimal stabilization of nonlinear systems, J. Appl. Math. Mech., 25 (1961), pp. 1254–1266.
- F. ANCONA AND A. BRESSAN, Patchy vector fields and asymptotic stabilization, ESAIM Control Optim. Calc. Var., 4 (1999), pp. 445–471.
- F. ANCONA AND A. BRESSAN, Flow stability of patchy vector fields and robust feedback stabilization, SIAM J. Control Optim., 41 (2003), pp. 1455–1476.
- [4] F. ANCONA AND A. BRESSAN, Nearly time optimal stabilizing patchy feedbacks, Ann. Inst. H. Poincaré Anal. Non Linéaire, 24 (2007), pp. 279–310.
- [5] M. BARDI AND I. CAPUZZO-DOLCETTA, Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations, Birkhäuser Boston, Boston, 1997.
- M. BARDI AND M. FALCONE, An approximation scheme for the minimum time function, SIAM J. Control Optim., 28 (1990), pp. 950–965.
- [7] M. BARDI AND M. FALCONE, Discrete approximation of the minimal time function for systems with regular optimal trajectories, in Analysis and Optimization of Systems, Lecture Notes in Control and Inform. Sci. 144, A. Bensoussan and J. L. Lions, eds., Springer-Verlag, Berlin, 1990, pp. 103–112.
- [8] A. M. BLOCH, M. REYHANOGLU, AND N. H. MCCLAMROCH, Control and stabilization of nonholonomic dynamic systems, IEEE Trans. Automat. Control, 37 (1992), pp. 1746–1757.
- O. BOKANOWSKI, E. CRISTIANI, AND H. ZIDANI, An efficient data structure and accurate scheme to solve front propagation problems, J. Sci. Comput., 42 (2010), pp. 251–273.
- [10] A. BRESSAN, Singularities of stabilizing feedbacks, Rend. Sem. Mat. Univ. Politec. Torino, 56 (1998), pp. 87–104.
- [11] A. BRESSAN AND F. S. PRIULI, Nearly optimal patchy feedbacks, Discrete Contin. Dyn. Syst., 21 (2008), pp. 687–701.
- [12] M. BREUSS, E. CRISTIANI, P. GWOSDEK, AND O. VOGEL, An adaptive domain-decomposition technique for parallelization of the fast marching method, Appl. Math. Comput., 218 (2011), pp. 32–44.
- [13] S. CACACE, E. CRISTIANI, AND M. FALCONE, A local ordered upwind method for Hamilton-Jacobi and Isaacs equations, in Proceedings of the 18th IFAC World Congress, 2011, pp. 6800–6805.
- [14] F. CAMILLI, M. FALCONE, P. LANUCARA, AND A. SEGHINI, A domain decomposition method for Bellman equations, in Domain Decomposition Methods in Scientific and Engineering Computing, Contemp. Math. 180, D. E. Keyes and J. Xu, eds., AMS, Providence, RI, 1994, pp. 477–483.
- [15] E. CARLINI, M. FALCONE, AND R. FERRETTI, An efficient algorithm for Hamilton-Jacobi equations in high dimension, Comput. Vis. Sci., 7 (2004), pp. 15–29.
- [16] E. CARLINI, M. FALCONE, N. FORCADEL, AND R. MONNEAU, Convergence of a generalized fastmarching method for an eikonal equation with a velocity-changing sign, SIAM J. Numer. Anal., 46 (2008), pp. 2920–2952.
- [17] Y. CHEN AND B. COCKBURN, An adaptive high-order discontinuous Galerkin method with error control for the Hamilton-Jacobi equations. I. The one-dimensional steady state case, J. Comput. Phys., 226 (2007), pp. 1027–1058.
- [18] Y. CHENG AND C. W. SHU, A discontinuous Galerkin finite element method for directly solving the Hamilton-Jacobi equations, J. Comput. Phys., 223 (2007), pp. 398–415.
- [19] M. G. CRANDALL AND P.-L. LIONS, Two approximations of solutions of Hamilton-Jacobi equations, Math. Comp., 43 (1984), pp. 1–19.
- [20] E. CRISTIANI, A fast marching method for Hamilton-Jacobi equations modeling monotone front propagations, J. Sci. Comput., 39 (2009), pp. 189–205.
- [21] E. CRISTIANI AND P. MARTINON, Initialization of the shooting method via the Hamilton-Jacobi-Bellman approach, J. Optim. Theory Appl., 146 (2010), pp. 321–346.
- [22] M. FALCONE, Numerical solution of dynamic programming equations (Appendix A), in Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations, Birkhäuser Boston, Boston, 1997, pp. 471–504.

- [23] M. FALCONE, Some remarks on the synthesis of feedback controls via numerical methods, in Optimal Control and Partial Differential Equations, J. L. Menaldi, E. Rofman, and A. Sulem, eds., IOS Press, Amsterdam, 2001, pp. 456–465.
- [24] M. FALCONE AND R. FERRETTI, Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations, SIAM, Philadelphia, to appear.
- [25] M. FALCONE, P. LANUCARA, AND A. SEGHINI, A splitting algorithm for Hamilton-Jacobi-Bellman equations, Appl. Numer. Math., 15 (1994), pp. 207–218.
- [26] T. HUNT, A Proof of the Higher Order Accuracy of the Patchy Method for Solving the Hamilton-Jacobi-Bellman Equation, Ph.D. Thesis, University of California Davis, Davis, CA, 2011.
- [27] K. A. MORGANSEN AND R. W. BROCKETT, Nonolonomic control based on approximate inversion, in Proceedings of the 1999 American Control Conference, 1999, pp. 3515–3519.
- [28] C. NAVASCA AND A. J. KRENER, Patchy solutions of Hamilton-Jacobi-Bellman partial differential equations, in Modeling, Estimation and Control, Lecture Notes in Control and Inform. Sci. 364, A. Chiuso, A. Ferrante, and S. Pinzoni, eds., Springer, Berlin, 2007, pp. 251–270.
- [29] C. NAVASCA AND A. J. KRENER, The patchy cost and feedback for the HJB PDE, in Proceedings of the 18th International Symposium on Mathematical Theory of Networks and Systems, Blacksburg, VA, 2008.
- [30] A. PICARELLI, Metodo patchy per la soluzione di alcuni problemi di controllo ottimo, Master thesis, SAPIENZA–Università di Roma, Rome, Italy, 2011 (in Italian); available online from https://sites.google.com/site/athenapicarelli/education.
- [31] A. QUARTERONI AND A. VALLI, Domain Decomposition Methods for Partial Differential Equations, The Clarendon Press, Oxford University Press, New York, 1999.
- [32] J. A. SETHIAN, A fast marching level set method for monotonically advancing fronts, Proc. Natl. Acad. Sci. USA, 93 (1996), pp. 1591–1595.
- [33] J. A. SETHIAN, Level Set Methods and Fast Marching Methods, Cambridge University Press, Cambridge, UK, 1999.
- [34] J. A. SETHIAN AND A. VLADIMIRSKY, Ordered upwind methods for static Hamilton-Jacobi equations: Theory and algorithms, SIAM J. Numer. Anal., 41 (2003), pp. 325–363.
- [35] C. W. SHU, High order numerical methods for time dependent Hamilton-Jacobi equations, in Mathematics and Computation in Imaging Science and Information Processing, Lect. Notes Ser. Inst. Math. Sci. Natl. Univ. Singap. 11, S. S. Goh, A. Ron, and Z. Shen, eds., World Scientific, Hackensack, NJ, 2007, pp. 47–91.
- [36] Y.-H. TSAI, L.-T. CHENG, S. OSHER, AND H.-K. ZHAO, Fast sweeping algorithms for a class of Hamilton–Jacobi equations, SIAM J. Numer. Anal., 41 (2003), pp. 673–694.
- [37] J. N. TSITSIKLIS, Efficient algorithms for globally optimal trajectories, IEEE Trans. Automat. Control, 40 (1995), pp. 1528–1538.
- [38] H. ZHAO, A fast sweeping method for eikonal equations, Math. Comp., 74 (2005), pp. 603–627.