On the convergence of PatchMatch and its variants

Thibaud Ehret
CMLA, ENS Cachan, CNRS,
Université Paris-Saclay, 94235 Cachan, France
thibaud.ehret@cmla.ens-cachan.fr

Pablo Arias
CMLA, ENS Cachan, CNRS,
Université Paris-Saclay, 94235 Cachan, France
pablo.arias@cmla.ens-cachan.fr

Abstract

Many problems in image/video processing and computer vision require the computation of a dense k-nearest neighbor field (k-NNF) between two images. For each patch in a query image, the k-NNF determines the positions of the k most similar patches in a database image. With the introduction of the PatchMatch algorithm, Barnes et al. demonstrated that this large search problem can be approximated efficiently by collaborative search methods that exploit the local coherency of image patches. After its introduction, several variants of the original PatchMatch algorithm have been proposed, some of them reducing the computational time by two orders of magnitude. In this work we study the convergence of PatchMatch and its variants, and derive bounds on their convergence rate. We consider a generic PatchMatch algorithm from which most specific instances found in the literature can be derived as particular cases. We also derive more specific bounds for two of these particular cases: the original PatchMatch and Coherency Sensitive Hashing. The proposed bounds are validated by contrasting them to the convergence observed in practice.

1. Introduction

Patch-based methods are among the state-of-the-art in several image/video processing and computer vision applications. Often these methods require finding for all patches of a query image, the (approximate) k nearest neighbors among the set of patches of a database image. This is referred to as an *approximate k nearest neighbors field* (k-ANNF) from the query image to the database image.

Examples of the application of k-ANNFs can be found for image completion (and editing) [2, 4], denoising of images [5] and video [7, 19], video stylization [6, 7], alpha matting [12], optical flow [3, 11, 14, 20] and stereo-vision [20, 9]. Also in the close field of computer graphics ANNFs of 3D surface patches have been applied to mesh tracking [16] and texture transfer [10].

The brute-force computation of the k-NNF scales linearly with the product of the number of pixels in the query and the database images, and is therefore prohibitively slow. The first practical approaches rely on data structures such as hash tables or partition trees (see [18] and references therein). The approximate k nearest neighbors of each query patch are computed independently. Even if these approaches greatly improve with respect to the brute-force search, they are still too slow (and moreover scale badly with the patch size) for many applications such as those requiring user interaction.

The introduction of the *PatchMatch* algorithm [4] and its generalized version [5] has represented a breakthrough in the field. It brings a speed-up of almost two orders of magnitude over previous search techniques. The main reason for this is that *PatchMatch* performs all queries simultaneously and in collaboration, exploiting the fact that overlapping query patches are likely to have overlapping matches in the database image.

PatchMatch is an iterative algorithm which starts from a random initial guess for the k-ANNF and gradually refines it. In each iteration, each patch *propagates* its current candidates to its neighboring patches on the query image grid. Additionally, new candidates are searched randomly in the database image. This is performed by uniformly sampling in a series of concentric squares with decreasing radius centered at the position of the current best candidate.

Several works have improved the original *PatchMatch* algorithm reporting gains of one order of magnitude or even more. The main theme of these works is to combine *PatchMatch* with classical search structures to improve the random initialization or the random search. *Coherency Sensitive Hashing* (CSH) [17] uses locality sensitive hashing [15] to improve both the propagation and the random search steps. KD-trees [8] are used in [21] instead of the random initialization (after reducing the dimensionality of the patches). In [13] a KD-tree is used for the random search.

In this work we extend and generalize the theoretical framework of [1] formalizing *PatchMatch*-like techniques. We apply it to study their convergence and give upper bounds on their convergence speed which are tighter than the ones of [1]. In particular we study the original *PatchMatch* [4],

its generalization to k nearest neighbors [5] and CSH [17]. Our estimates of a geometric convergence rate confirm the intuitions that led to the design of these algorithms. They also provide insight that might help improving current techniques.

We interpret *PatchMatch* in a general setting, as a collaborative optimization tool. While the results in [1] consider only the original *PatchMatch* algorithm with one nearest neighbor, our results apply to k-nearest neighbors and to more complex *PatchMatch* algorithms, such as CSH [17], propagation-assisted KD-trees [13] and RIANN [7].

We start by giving a precise definition of a generic *PatchMatch* algorithm in §2. In §3 we analyze the convergence of the generic *PatchMatch*, by bounding the probability of having energies higher than a given threshold after an iteration of the algorithm. In §4 we consider two specific algorithms, the original *PatchMatch* [4] and CSH [17] and derive specialized bounds for them. Finally, in §5 we compare the theoretical bound to empirical cases. The proofs of our results can be found in the supplementary material.

2. Generic fast patch matching algorithm

2.1. Notations

We denote the query image by $\mathbf{A}$ and the database image by $\mathbf{B}$. We write $x \in \mathbf{A}$ if $x \in \mathbb{R}^d$ is a patch of the image $\mathbf{A}$ (with this abuse of notation, $\mathbf{A}$ is both an image and the set of its patches). The k-NNF from image $\mathbf{A}$ to $\mathbf{B}$ is denoted by φ. For a patch $x \in \mathbf{A}$ the matches associated to x in $\mathbf{B}$ are written as φ_x, which is a set of k distinct patches of $\mathbf{B}$.

Definition 2.1 (Matching energy) *The quality of the matching of a patch $x \in \mathbf{A}$ is measured by an the energy function $U_x(.)$ defined for any patch $z \in \mathbb{R}^d$ with values between 0 and $+\infty$. We generalize this definition to a set of patches u by*

$$U_x(u) = \max_{y \in u} U_x(y). \tag{1}$$

We assume that the minimum value of U_x is over the sets of k elements without repetitions is 0.

In practice, the L_2 distance in $\mathbb{R}^d$, written as $\|.\|_2$, is often used to define as matching energy. To have a minimum value of 0 over k-sets, we write it as

$$U_x(z) = \|x - z\|_2 - K_x \tag{2}$$

where if $N_k(x) \in \mathbf{B}$ is the actual k^{th} nearest neighbor of x in $\mathbf{B}$, $K_x = \|x - N_k(x)\|_2$. It follows that if u is a set of k distinct patches of $\mathbf{B}$, then $U_x(u) \geqslant 0$. We write $\overline{\varphi}_x$ to denote the worst match in this set, defined by

$$\overline{\varphi}_x = \operatorname*{argmax}_{y \in \varphi_x} U_x(y), \tag{3}$$

where $U_x(.)$ is the energy function of Definition 2.1.

The k-NNF φ assigns to each $x \in \mathbf{A}$ a k-set φ_x that minimizes U_x, i.e. such that $U_x(\varphi_x) = 0$. The goal of *PatchMatch* algorithms is to find an approximate solution to this optimization problem. In the following we define a level-set, written $\{U_z \geqslant \alpha\}$, for η a k-set of elements $\eta \in \{U_z \geqslant \alpha\}$ if and only if $U_z(\eta) \geqslant \alpha$.

The following operator selects the k patches with smaller energy U_x from a larger set.

Definition 2.2 (Merge operator) *We define* merge_x^k *as an operator transforming a set of more than k patches into a set of exactly k patches satisfying:*

$$|\text{merge}_x^k(u)| = k \qquad \textit{and}$$
$$\forall p \in u \backslash \text{merge}_x^k(u), U_x(p) \geqslant U_x(\text{merge}_x^k(u)). \tag{4}$$

We use basic graph notation in the next sections. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. By $y \sim x$ we denote that y is parent of x, *i.e.* $(y, x) \in \mathcal{E}$. We finally write $\mu(x)$ for the number of parents of a node x, i.e.

$$\mu(x) = |\{y \mid y \sim x\}|. \tag{5}$$

A path of size m from x to z corresponds to a set of vertices $(c_i)_{i=1,\dots,m} \in \mathcal{V}^m$ such that $c_1 = x$, $c_m = z$ and for all $i \in [\![1, m-1]\!]$, $(c_i, c_{i+1}) \in \mathcal{E}$. We write $\mathcal{P}(x, z)$ for the set of all paths (of any size) from x to z.

2.2. Propagation graph

In a *PatchMatch* algorithm all patches in $\mathbf{A}$ collaboratively search for their matches in $\mathbf{B}$. This collaboration is achieved through the propagation of candidate matches from each query patch to other patches in $\mathbf{A}$ following a specific order. This process can be described by defining a directed acyclic graph over the set of patches of $\mathbf{A}$, which we call the *propagation graph* following [1]. The vertices $\mathcal{V}$ in the propagation graph $\mathcal{G}$ are the set of patches of $\mathbf{A}$, and the directed edges $\mathcal{E}$ describe the propagation relationships: $(z, x) \in \mathcal{E}$ means that z propagates matches to x. We associate an action function to each edge to allow for the possibility of applying a transformation to the propagated matches.

Definition 2.3 (Propagation action) *An action associated to an edge $e \in \mathcal{E}$ in the propagation graph, written A_e, is a function which takes as argument a patch of an image and returns a patch. This patch can either be from the same image or be a new one generated by the action.*

The vertices in the propagation graph are indexed using a topological ordering (such an ordering exists because the graph is acyclic). The propagation follows this ordering. The complete definition of the propagation graph is the following, and it fully specifies the propagation in a *PatchMatch* algorithm.

Definition 2.4 (Propagation graph) *A propagation graph, written $\mathcal{G}$, corresponds to a triplet $(\mathcal{V}, \mathcal{E}, A)$ where $\mathcal{V}$ is the set of patches of image $\mathbf{A}$, $\mathcal{E}$ a set of edges such that the graph is connected and acyclic and A are the action functions associated to each edge (Def. 2.3).*

For matching image patches, the forward propagation follows the raster order from the top-left to the bottom-right. Each patch propagates matches to its neighbors on the right and down:

$$\mathcal{E} = \{(x, y) \in \mathcal{V} \times \mathcal{V} \mid y = \text{right}(x) \text{ or } y = \text{down}(x)\}\,. \quad (6)$$

The action shifts the candidate patch following the direction of propagation: if $x \in \mathbf{A}$ has a candidate $z \in \mathbf{B}$, it propagates $\text{right}(z) \in \mathbf{B}$ to $\text{right}(x)$. Thus, $A(x, \text{right}(x)) = \text{right}(\cdot)$, and analogously $A(x, \text{down}(x)) = \text{down}(\cdot)$.

Some variants of *PatchMatch* add additional edges to this basic propagation graph seeking to enhance the impact of the propagation [5, 17]. These additional transitions connect pairs of patches in $\mathbf{A}$ that are similar, and thus good matches for one of patches in the pair are natural candidates for the other. The action associated with these new edges is simply the identity function.

PatchMatch algorithms have also been applied on meshes [16, 10]. In those cases, the propagation can be defined by considering a DAG on a subgraph of the mesh.

2.3. Random search

For the specification of a *PatchMatch* algorithm we need a mechanism for gathering random samples around the current candidates.

Definition 2.5 (Random sampling operator) *Given a database image $\boldsymbol{B}$ we define the transition kernel Q such that for any k-set φ of patches from image $\boldsymbol{B}$, $Q(\varphi, \cdot)$ is a probability on the k-sets of patches from $\boldsymbol{B}$. A set of k patches drawn from $Q(\varphi, \cdot)$ will be denoted by $S\varphi$, i.e. $S\varphi \sim Q(\varphi, \cdot)$. We consider transition kernels with the property of having a non-zero probability of transitioning to a k-set of matches with arbitrary positive energy:*

$$Q(\varphi, \{U_x < \varepsilon\}) > 0, \qquad \forall \varphi, \forall x \in \boldsymbol{A}, \forall \varepsilon > 0. \quad (7)$$

Defined alongside the kernel Q is the *worst case transition probability* for an energy level ε at $z \in \mathbf{A}$, as the highest probability of transitioning between two sets of patches with energy level higher than ε:

$$C(z, \varepsilon) := \sup_{\eta \in \{U_z \geqslant \varepsilon\}} Q(\eta, \{U_z \geqslant \varepsilon\}). \quad (8)$$

Lemma 2.6 *For all $z \in \boldsymbol{A}$, $C(z, .)$ is a non-increasing function such that for $\varepsilon < 0$, $C(z, \varepsilon) \in [0, 1[$ and $C(z, \varepsilon) = 1$ for $\varepsilon \leqslant 0$.*

Algorithm 1: Generic patch matching algorithm

1 Initialize propagation graph $\mathcal{G}$
2 Initialize matching φ^0
3 **for** $n \in \mathbb{N}$ **do**
4 **Update candidates**
5 **for** $x \in \mathcal{V}$ *following the topological ordering* **do**
6 $\varphi_x^{n+1/2} = \text{merge}_x^k \left(\varphi_x^n \cup \bigcup_{y \sim x} A_{y,x} \varphi_y^{n+1} \cup \bigcup_{y \sim x} S_2 A_{y,x} \varphi_y^{n+1} \right)$
7 $\varphi_x^{n+1} = \text{merge}_x^k \left(\varphi_x^{n+1/2} \cup S_1 \varphi_x^{n+1/2} \right)$
8 **end**
9 **Reverse propagation graph**
10 **Update propagation graph**
11 **end**

2.4. Algorithm

The generic *PatchMatch* is presented in Algorithm 1. It starts by initializing the propagation graph and the candidate matches. The propagation graph is initialized by defining a set of edges $\mathcal{E}$ and the associated propagation actions A.

The core of the algorithm is the iterative process that cycles through the nodes in the topological ordering updating the candidate matches according to the steps 5 and 7. The update in step 5 is the result of selecting the best k matches from a set of candidates given by: the current k matches φ_x^n; the matches $A_{y,x}\varphi_y^{n+1}$ propagated from the parent nodes $y \sim x$ in the propagation graph; and samples $S_2 A_{y,x} \varphi_y^{n+1}$ gathered around them. We call the latter set of samples the *randomized propagation*. In step 7 k random samples $S_1 \varphi_x^{n+1/2}$ around the new current matches are drawn to finish the update process.

The random samplings S_1 and S_2 are drawn from transition kernels Q_1 and Q_2 which might differ. The parent nodes $y \sim x$ precede x in the topological ordering, therefore they propagate to x the updated candidates φ_y^{n+1}.

After each iteration the propagation graph is reversed. The last step considers the possibility of modifying the propagation graph by adding/removing relevant/irrelevant edges. This is done for example in CSH [17].

3. Convergence of the patch matching algorithms

In this section we study the convergence of the generic *PatchMatch* algorithm described in the previous section. We do so by upper-bounding the probability that the energy U_x at a node x is larger than a threshold ε after an iteration of *PatchMatch*.

3.1. Point-wise energy decay

Our main result is Theorem 3.2 which bounds the decay of the probability of $U_x(\varphi_x^{n+1} > \varepsilon)$. The propagation makes this probability smaller, since any of the ancestors of x could find a candidate that when propagated to x would yields an energy below ε. Indeed, as a consequence of the propagation, if φ_x has energy higher than ε after a propagation pass, then the energies of the ancestors z of x need to be higher than a series of levels ε_{zx}. The violation of any of these restrictions would cause the propagation of a candidate match to x with energy U_x below ε. The higher these levels, the smaller the probability of not sampling random candidates with energies below them, and thus the smaller $P(U_x(\varphi_x^{n+1}) > \varepsilon)$.

In Lemma 3.1 we show that imposing a lower bound ε of the matching energy of a node x results in lower bounds $\varepsilon_{z,x}$ for all its ancestors z that can be calculated recursively starting from x and following the reversed propagation ordering. In Theorem 3.2 we bound the probability that none of x's ancestors draw a random candidate with energy lower than the corresponding $\varepsilon_{z,x}$.

Lemma 3.1 (Constraints propagation) *Consider an assignment φ^{n+1} resulting from an iteration of Algorithm 1. Then for each pair of nodes $x, z \in \mathcal{V}$,*

$$U_x(\varphi_x^{n+1}) \geqslant \varepsilon \Rightarrow U_z(\varphi_z^{n+1}) \geqslant \varepsilon_{z,x}, \tag{9}$$

where the levels $\varepsilon_{z,x} \geqslant 0$ are as follows. For the ancestors of x (i.e. $\mathcal{P}(z,x) \neq \emptyset$) the levels $\ell^{\varepsilon}_{z,x}$ are defined via the following recursion starting from x and following the inverse propagation order:

$$\begin{cases} \varepsilon_{z,x} = \min \left\{ U_z(\theta) \;\middle|\; \theta \in \bigcap\limits_{y \text{ s.t. } z \sim y} A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\}) \right\} \\ \varepsilon_{x,x} = \varepsilon. \end{cases} \tag{10}$$

For the rest of the nodes $\varepsilon_{z,x} = -1$.

Once we have established this "allowed" sets for the ancestors of x, the idea of the proof is to determine the probability of not escaping the allowed sets in any of the random searches.

Suppose z is an ancestor of x with a candidate $\varphi_z^n \in \{U_z \geqslant \varepsilon_{z,x}\}$ (for simplicity assume $k = 1$). The probability of keeping the energy higher than ε after one iteration decreases because there is a non-zero probability of taking a random sample outside this level set. The probability of sampling a candidate in an upper-level set is

$$\mathbb{P}(S\varphi_z^n \in \{U_z \geqslant l\} \mid \varphi_z^n \in \{U_z \geqslant l\}) = \frac{1}{\int_{\{U_z \geqslant l\}} \mathbb{P}(d\varphi_z^n)} \int_{\{U_z \geqslant l\}} Q(\varphi_z^n, \{U_z \geqslant l\}) \mathbb{P}(d\varphi_z^n). \tag{11}$$

If we knew the probability distribution of the candidate φ_x^n at iteration n, we could compute the above probability exactly. Instead, we bound it by assuming that all the mass of the distribution of φ_z^n concentrates on a single point: the one from which it is more unlikely to draw a sample with energy lower than l. The resulting probability is given by the worst case transition probability C in (8). This is the main intuition in the proof of our main result.

Theorem 3.2 (Point-wise convergence) *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 1. Then, for all $\varepsilon > 0$, for all $x \in \mathbf{A}$, we have*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \left(C_2(z, \varepsilon_{z,x})^{\mu(z)} C_1(z, \varepsilon_{z,x}) \right), \tag{12}$$

where $\mu(z)$ was defined in (5) as the number of parents of node z and C_i denotes the worst case transition probability for kernel Q_i, as in Eq. (8).

For notational simplicity the product in (12) is over all patches $z \in \mathbf{A}$, but the corresponding C_is are 1 for those z that are not ancestors of x in the propagation graph.

The result from Theorem 3.2 reflects some of the intuitive ideas that led the design of patch matching algorithms. Due to the propagation, all ancestors of x contribute to the probability of x of improving its energy. But not all nodes contribute the same to the bound. The larger $\varepsilon_{z,x}$ the better, as the $C_i(z, .)$ are non-increasing functions (Lemma 2.6). It is therefore important to design the propagation graph and its actions to maximize the $\varepsilon_{z,x}$. The shift propagation actions introduced in [4] in the patch matching application can be interpreted under the light of Theorem 3.2 as an heuristic to maximize the levels $\varepsilon_{z,x}$.

Indeed, $\varepsilon_{z,x}$ is given by the minimum energy U_z in the intersection of the sets $A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\})$ for y in the set of nodes to which z propagates. Say y is the right neighbor of z. Then the propagation action from z to y is a right shift $A_{z,y} = \text{right}(\,\cdot\,)$. Suppose that the patches are of size $s \times s$ and the matching cost is a function of the sum of pixel-wise matching errors over the patch (such as any L_p norm). Since the patches at z and y overlap, the energies $U_y(\eta)$ and $U_z(A_{z,y}^{-1}\eta)$ have $s(s-1)$ terms in common, and can expected to be similar. Therefore the minimum value of U_z in $A_{z,y}^{-1}(\{U_y \geqslant \varepsilon_{y,x}\})$ should not be much lower than $\varepsilon_{y,x}$. By assuming certain regularity conditions on the image it is possible to bound the difference between $\varepsilon_{z,x}$ and $\varepsilon_{y,x}$. This is out of the scope of this paper.

A result that is found in practice is that the rate at which the energy decreases becomes slower as the matches improve. Less contributions from other nodes are taken into account and mostly the random search improves the matching. This

is in agreement with the theory, since as ε decrease, the sizes of the allowed set increase and the $\varepsilon_{z,x}$ decrease as well. According to Theorem 3.2, this results in a slower energy decrease.

We can also note that adding more edges to the propagation graph can only improve the convergence rate, requiring a smaller number of iterations to achieve a desired precision in the result (in probability). However, having more edges implies more computation each iteration. The optimal number of propagation links results from a trade-off between the computational effort per iteration and its impact on reducing the energy.

Most variants of PatchMatch use the same propagation graph and change the random search steps (including the initialization). This can have a dramatic effect of the convergence. In our bound the choice of the random search determines the coefficients C_1 and C_2 which depend directly on the search transition kernel. Two examples of random searches will be reviewed in §4.

In the case of the single nearest neighbor ($k = 1$) the bound from Theorem 3.2 can be improved. The improvement comes from a better version of the C_i coefficients that considers the transition probability of merge$_x^k(\eta \cup S_i\eta)$. The details are provided in the supplementary material.

3.2. Uniform decay and convergence in the mean

One of the advantages of *PatchMatch* algorithms is that the NNF converges rapidly as a whole. We now give bounds on uniform convergence and convergence in the mean.

Theorem 3.3 *Consider the field of candidate matches at iteration n, φ^n. Define φ^{n+1} by applying an iteration of the Generic PatchMatch in Algorithm 1. Then, for all $\varepsilon > 0$ we have*

$$\mathbb{P}(\|U_\cdot(\varphi_\cdot^{n+1})\|_\infty \geqslant \varepsilon) \leqslant \mathbb{P}(\|U_\cdot(\varphi_\cdot^{n})\|_\infty \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \Big(C_2(z, \{U_z \geqslant \varepsilon\})^{\mu(z)} C_1(z, \{U_z \geqslant \varepsilon\}) \Big). \quad (13)$$

Theorems 3.2 and 3.3, together with the assumption (7) on the transition kernels Q_i, imply the convergence in probability of *PatchMatch* algorithms. Assumption (7) is necessary to ensure that $C_i(z, \varepsilon) < 1$ for $\varepsilon > 0$. A stronger convergence can also be shown once we considered transition kernels Q_i having this good property, the following result shows convergence in the mean, both point-wise and uniformly for the whole energy field.

Corollary 3.4 *Assume that for any pair (η, ξ) of sets of k candidate matches $Q_1(\eta, \xi) > 0$ (or $Q_2(\eta, \xi) > 0$). Let (φ^n) be a sequence defined by Algorithm 1. Then $\forall x \in \boldsymbol{A}$, $\mathbb{E}[U_x(\varphi_x^n)] \xrightarrow[n\to\infty]{} 0$ and $\mathbb{E}[\|U_\cdot(\varphi_\cdot^n)\|_\infty] \xrightarrow[n\to\infty]{} 0$.*

This assumption on Q_i is reasonable when the universe of candidates is finite, such as when searching matching patches in a database image $\mathbf{B}$. Some variants of *PatchMatch* have been used to minimize a field of functions over continuous parameters (scales and rotations [5], planes in 3D [9]). The previous corollary does not apply in these cases. Nevertheless both Theorems 3.2 and 3.3 remain valid even without assumption (7).

4. Specific *PatchMatch* algorithms

We now derive more specific bounds for two particular cases of the generic *PatchMatch* algorithm found in the literature.

4.1. The original *PatchMatch* algorithm

The original *PatchMatch* algorithm was introduced in [4] for $k = 1$, and then generalized to k-nearest neighbors in [5] (the *heap algorithm*) together with other generalizations. Most of them are covered by Theorem 3.2 as they are particular cases of Algorithm 1. In this section, we derive a more specific bound for one of these variants.

The field φ of k-matches is initialized at random, by uniform sampling from image $\mathbf{B}$. The propagation graph is the basic one presented in §2.2, with the shift actions. For the forward propagation these are $A(x, \text{right}(x)) = \text{right}(\cdot)$, $A(x, \text{down}(x)) = \text{down}(\cdot)$.

We define S_1 by taking samples around the current best candidate (the *RS best* variant in [5]). In [5] these samples are drawn from a sequence of transition probabilities. The qth sample is sampled uniformly from a box of size $[-d\alpha^q, d\alpha^q]^2$ centered at the current match, for $q = 0, 1, \ldots, q_{\max}$, $\alpha = 0.5$ and $d = \max\{W, H\}$ for a database image $\mathbf{B}$ of size $W \times H$. The number of samples is chosen so that the smallest box is larger than a pixel. The first sample is taken uniformly on the whole image $\mathbf{B}$, so that there is a positive probability of transitioning from and to any two patches in $\mathbf{B}$. For simplicity, we assume that the random search operator $S_1\varphi_x^n$ draws k independent patches in $\mathbf{B}$ with the same probability distribution $Q'(\tilde{\varphi}_x, .)$, where $\tilde{\varphi}_x$ is the best current match. For all $\phi \in \mathbf{B}$, the support of $Q'(\phi, \cdot)$ covers all patches in $\mathbf{B}$, so as in [4, 5], there is always a positive probability of transitioning between any two patches in $\mathbf{B}$.

Without loss of generality, to simplify the notation we consider that the propagation graph is the same for all iterations (i.e. no alternation between iterations).

Proposition 4.1 *The specific basic PatchMatch algorithm described above converges in probability to a NNF which minimizes the energy, namely*

$$\lim_{n\to\infty} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \quad (14)$$

with a geometric convergence rate. Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$, we have that

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \prod_{z \in \mathbf{A}} \left(1 - (1 - C'(z, \varepsilon_{z,x}))^k\right),$$

with $C'(z, \alpha) := \sup_\eta Q'(\eta, \{U_z \geqslant a\})$. For $\alpha > 0$ we can guarantee that $C'(z, \alpha) < 1$.

Corollary 4.2 *If $k = 1$ the upper bound can be written as*

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \prod_{z \in \boldsymbol{A}} C'(z, \varepsilon_{z,x}) \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon).$$

with $C'(z, \alpha) := \sup_{\eta \in \{U_z \geqslant a\}} Q'(\eta, \{U_z \geqslant a\})$.

Corollary 4.2 allows a direct comparison of our bound with the bound derived in [1]. Both bounds have the same structure. However, the energy levels ε_{zx} [1] are smaller than ours causing a looser bound.

4.2. The CSH algorithm

Coherency Sensitive Hashing (CSH) was introduced in [17]. It uses Locality Sensitive Hashing (LSH) [15] to improve the random search and to add propagation edges.

LSH is a method for nearest neighbors search based on partitioning the search space using a series of hash functions h drawn randomly from a family $\mathcal{H}$. Each hash function partitions the space in "bins" containing points with the same hash value. The family $\mathcal{H}$ has the property that nearby points collide in the same bin with high probability, while far away points do so with smaller probability. This is made precise by the following definition 4.3.

Definition 4.3 *A family of functions $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$, where U is the set of hashes, is called (R, cR, p_1, p_2)-sensitive if for any $p, q \in \mathbb{R}^d$ (for simplicity of notation, we write $\mathbb{P}_{\mathcal{H}}(\,.\,) = \mathbb{P}(\,.\mid \mathcal{H})$)*

(1) if $\|p - q\| \leqslant R$ then $\mathbb{P}_{\mathcal{H}}(h(q) = h(p)) \geqslant p_1$,

(2) if $\|p - q\| \geqslant cR$ then $\mathbb{P}_{\mathcal{H}}(h(q) = h(p)) \leqslant p_2$.

This is useful for nearest neighbors search when $p_1 > p_2$.

Instead of using directly the elements from $\mathcal{H}$, a second family of functions $\mathcal{G}$, called an OR family, is created. The function $g \in \mathcal{G}$ is based on a set of n random functions $h_1, \ldots, h_n$ from $\mathcal{H}$ such that for all p, q, $g(p) = g(q)$ if and only if there exist $i \in [\![1, n]\!]$ such that $h_i(p) = h_i(q)$. This will be the set of functions used to define the algorithm.

Lemma 4.4 *If $\mathcal{H}$ is (R, cR, p_1, p_2)-sensitive then an OR family $\mathcal{G}$ created using n functions from $\mathcal{H}$ is $(R, cR, 1 - (1 - p_1)^n, 1 - (1 - p_2)^n)$-sensitive.*

In CSH, hash functions drawn randomly at each iteration are used both to define the random search operator S_1 and the randomized propagation operator S_2. Let us define the projection bin of a patch $p \in \mathbb{R}^d$ as

$$\mathcal{B}_g(p) = \{q \in \mathbf{B} \mid g(p) = g(q)\}. \tag{15}$$

These projection bins depend on the random choice of the hash function, and are used to define the candidate set sampling operators. The random search S_1 is simply the projection bin of the query patch x, i.e. $S_1\varphi_x := \mathcal{B}_g(x)$. The randomized propagation is defined as the union of the projection bins corresponding to the best b propagated candidates, where $b \leqslant k$ is a parameter of the method, that is:

$$S_2 A_{y,x}\varphi_y = \cup_{l=1}^{b} \mathcal{B}_g([A_{y,x}\varphi_y]_l), \quad y \sim x. \tag{16}$$

Here $[A_{y,x}\varphi_y]_l$, $l = 1, \ldots, k$ are the propagated candidates.

Note that $S_1\varphi_x$ does not depend on the candidate list, but only on the query patch x. This particularity, together with the properties of the hashing family $\mathcal{H}$, can be exploited to derive a tighter upper bound than the one given in Theorem 3.2. CSH also uses hashing over the query image $\mathbf{A}$ to connect similar patches in the propagation graph. Our result is valid for any propagation graph.

Proposition 4.5 *For a (R, cR, p_1, p_2)-sensitive family of hashing functions such that $R \geqslant \max_{z \in \boldsymbol{A}} K_z$ (see Definition 2.1), the sequence (φ^n) defined by the CSH algorithm converges in probability to a minimizer of the total energy,*

$$\lim_{n \to \infty} \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) = 0, \forall \varepsilon > 0, x \in \boldsymbol{A}, \tag{17}$$

with a geometric convergence rate. Moreover for all $\varepsilon > 0$, for all $x \in \boldsymbol{A}$,

$$\mathbb{P}(U_x(\varphi_x^{n+1}) \geqslant \varepsilon) \leqslant \mathbb{P}(U_x(\varphi_x^n) \geqslant \varepsilon) \tag{18}$$

$$\prod_{z \in \mathbf{A}} C_2(z, \ell_{z,x}^{\varepsilon})^{\mu(z)} f(p_1, \varepsilon_{z,x}), \tag{19}$$

where $f(\alpha, \beta) = (1 - \alpha^k)$ if $\beta > 0$, 1 otherwise.

In practice it is possible to compute an (R, cR, p_1, p_2)-family of hash functions such that p_1 is large enough to outperform the transition kernels used by the original *PatchMatch* algorithm. This explains the improved convergence rate of CSH reported in [17].

5. Experiments and discussion

We show experiments comparing the convergence predicted by the theory with the one found in practice, for the case of the original PatchMatch algorithm with $k = 1$ [4]. The code to reproduce the experiments is available at https://github.com/pariasm/pm-bound.

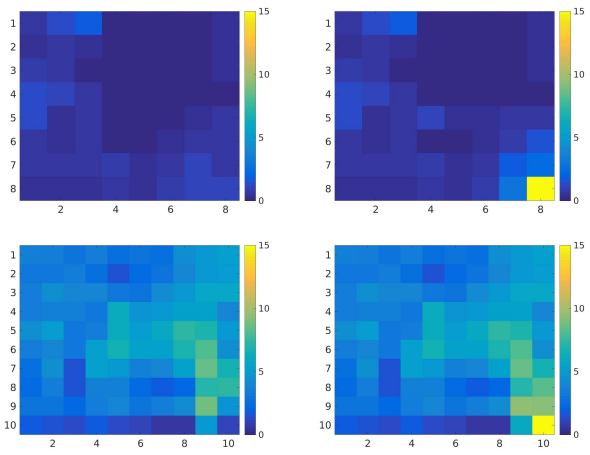

Figure 1: We represent here the $\varepsilon_{z,x}$s for two different points x (top and bottom rows) and for $\varepsilon = 1$ (left) and $\varepsilon = 15$ (right). The point x is the bottom right pixel.

5.1. Computation of theoretical bound

Given an energy value ε and a patch x in the query image $\mathbf{A}$, the computation of the bound requires computing the energy levels $\varepsilon_{z,x}$ and the worst case transition probability $C_1(z, \varepsilon_{z,x})$ for all ancestors z of x. The ancestors are all pixels located above and to the left of x during the forward propagation, and below and on the right of x during the backward pass. The original PatchMatch algorithm does not consider a randomized propagation $S_2 A_{z,y} \varphi_y^{n+1}$, thus in the following we will drop the subindex for C_1 and Q_1.

The computation of the levels $\varepsilon_{z,x}$ is straightforward. We loop on the ancestors starting from x. Following the inverse propagation ordering we apply the recursion (10). The inverse actions $A_{z,y}^{-1}$ are one pixel shifts in the opposite direction to the propagation. For instance if z propagates to y at his right, then $A_{z,y}^{-1}\{U_y \geqslant \varepsilon_{y,x}\}$ results from shifting to the left all elements of $\{U_y \geqslant \varepsilon_{y,x}\} \subseteq \mathbf{B}$.

Figure 1 shows examples of the resulting $\varepsilon_{z,x}$ at two image locations and for $\varepsilon = 1$ and $\varepsilon = 15$. The larger the $\varepsilon_{z,x}$ the smaller the coefficient $C(z, \varepsilon_{z,x})$ and the faster the convergence. We can see that the levels $\varepsilon_{z,x}$ at different points x can be very different. Moreover, increasing ε increases the levels $\varepsilon_{z,x}$ locally around x.

To compute the worst case transition probability $C(z, \varepsilon_{z,x})$, we compute for all $\eta \in \{U_z \geqslant \varepsilon_{z,x}\}$ the probability of drawing a sample from the upper-level set. As in the original PatchMatch, we take n independent samples following a sequence of uniform distributions $S_i\eta \sim Q_i(\eta, \cdot)$ on square boxes centered at η with a decreasing sequence of radii.[1] For the worst case transition, we need the n samples

[1]This scheme differs from the one considered in Proposition 4.1, which takes k equally distributed samples $S\eta \sim Q'(\eta, \cdot)$. But the theory applies the same as long as C is in accordance with the sampling procedure.

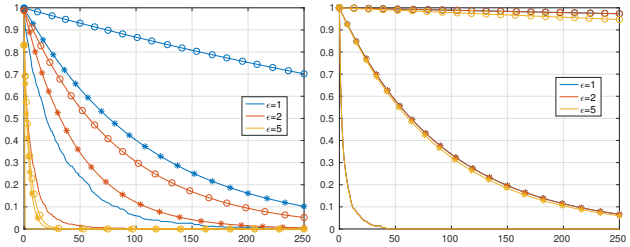

Figure 2: The different theoretical bounds are compared to the empirical one at the position of the two points from Figure 1 for different values of ε. The plots show the probability than the energy is above different energy levels. The line without marker corresponds to the empirical probability. The line marked with a circle corresponds to the bound presented in [1]. The line marked with a star correspond to the bound presented in this paper.

to be in $\{U_z \geqslant \varepsilon_{z,x}\}$, thus:

$$C(z, \varepsilon_{z,x}) = \max_{\eta \in \{U_z \geqslant \varepsilon_{z,x}\}} \prod_{i=1}^{n} Q_i(\eta, \{U_z \geqslant \varepsilon_{z,x}\}). \quad (20)$$

For each sampling radius r, the probability $Q_i(\cdot, \{U_z \geqslant \varepsilon_{z,x}\})$ can be computed efficiently via a convolution of the indicator function of the upper-level set with a box kernel of size $2r + 1 \times 2r + 1$[2]. We use an integral image implementation to speed-up the computation.

5.2. Experimental validation

To validate the theory we contrast the predicted evolution of the matching energy with the empirical evolution. Since PatchMatch is a randomized algorithm, we estimate for a given energy level ε the probability that the energy is above ε for each iteration. The empirical probability is computed by running N iterations of PatchMatch a number of trials M. For a patch $x \in \mathbf{A}$ we define $U_x^{n,m}$ the matching energy at iteration n of trial m. Then the empirical probability of $U_x \geqslant \varepsilon$ is estimated as

$$\widehat{P}(x, \varepsilon, n) = \widehat{\mathbb{P}}(U_x(\varphi_x^n) \geqslant \varepsilon) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}(U_x^{n,m} \geqslant \varepsilon).$$

The theoretical bound on this probability is given by

$$B(x, \varepsilon, n) = \mathbb{P}(U_x(\varphi_x^0) \geqslant \varepsilon) \cdot K(x, \varepsilon)^n, \quad (21)$$

where $K(x, \varepsilon) = \prod_{z \in \mathbf{A}} C(z, \varepsilon_{z,x})$. The candidates in the original PatchMatch are initialized by sampling uniformly over the database image $\mathbf{B}$, thus the initial probabilities can

[2]The kernel needs to be normalized by the area of the portion of the kernel that fits in the image domain.

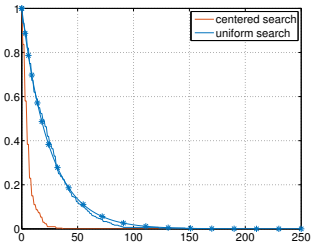

Figure 3: The effect of the random search. If we simplify the random search and make it uniform, then convergence is slower and the bound is tight. In this experiment, we match two random noise images. See text for details.

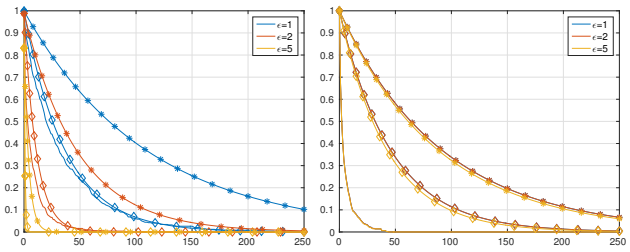

Figure 4: Comparison between the "average" and worst-case C. The curve with the diamonds corresponds to the "average" C whereas the stars show the worst case C.

be easily computed as the area ratio between the upper-level set $\{U_x \geqslant \varepsilon\}$ and the image domain **B**.

In Figure 2 we plot the estimated probabilities $\widehat{P}(x, \varepsilon, n)$ together with the theoretical bound $B(x, \varepsilon, n)$ for a patch $x \in \mathbf{A}$ and several values of ε. As expected, the estimated probability is below the worst case bound. For smaller ε the bound becomes looser. For example: for the energy to be below than $\varepsilon = 1$ with a probability of 0.2, the theoretical bound predicts 178 iterations but in practice 60 were needed.

For $k = 1$, the gap between the theoretical bound and the empirical decay is mainly due to upper-bounding the transition probability (11) by the worst case probability C. To verify this we use a simplified random search: the sample $S\eta$ is taken uniformly over **B** (as in the initialization). In this way, we eliminate the dependence between the sample $S\eta$ and the current candidate η, and we have that $\mathbb{P}(S\varphi_z^n \in \{U_z \geqslant l\} \mid \varphi_z^n \in \{U_z \geqslant l\})$ is the area ratio between the upper level set and the domain of image **B**, regardless of φ_z^n.

We generate two images of random noise. The query image **A** is of size 24×24. It contains $q = 20 \times 20 = 400$ patches of size 5×5. The database image **B** is of size 104×104, thus containing $p = 100 \times 100 = 10^4$ patches. As before, we compute the empirical probabilities $\widehat{P}$ and the bound for the bottom right patch. We show two results: one using only the uniform random search, and another one using the original centered random search. The plots correspond to an energy level of $\varepsilon = 0.5$, and only the unique global minimum is below ε. When using the uniform search, the empirical decay matches the theoretical prediction. The centered search shows a faster convergence.

This experiment also shows that the theoretical bound captures the main intuition behind the design of the PatchMatch algorithm: if a region of image **B** is an exact copy of image **A** (or of a region in image **A**) it becomes very likely to find the copied region in the database image. Due to the propagation, as soon as one node in **A** finds its match, it will be propagated to all other nodes. In this case, one can compute exactly the probability that all the ancestors of x miss the copied region. The probability of missing the global optimum is $1 - 1/p$. The probability that all q ancestors of x miss the optimum is therefore $(1 - 1/p)^q$. This coincides with the theoretical bound.

As a final experiment we show results obtained computing C as an "average" transition probability instead of the worst case. This average transition results from assuming in (11) that φ_x^n is distributed uniformly over the upper-level set. Figure 4 compares the predictions of this average C with the worst case C. While we do not have theoretical guaranties on the average C, its behavior that is much closer to the empirical case.

6. Conclusions

We presented a theoretical analysis of the convergence of PatchMatch algorithms. For an energy level ε, we show that the probability of having an energy above ε converges to 0 with the number of iterations, and we provide worst case bounds on the convergence rate. Our analysis applies to the case of k nearest neighbors, and to most variants of PatchMatch proposed in the literature. We give specific bounds for two of these algorithms: the original PatchMatch [4, 5] and CHS [17]. For the case of the original PatchMatch (with $k = 1$) we validate our results by comparing the predicted convergence rate with the one found in practice. The setting of our framework is rather general: the task of patch matching is viewed as an optimization problem where the goal is to minimize several non-convex energies U_x over the same domain. This might allow the application of these techniques to similar optimizations problems in other areas. For the case of matching patches, it would interesting to precise the link between the regularity of the images and the convergence rate, at least for certain simple models of images.

Acknowledgment

Work partly financed by IDEX Paris-Saclay IDI 2016, ANR-11-IDEX-0003-02, Office of Naval research grant N00014-17-1-2552, DGA Astrid project «filmer la Terre» n^o ANR-17-ASTR-0013-01, MENRT and Fondation Mathématique Jacques Hadamard

References

[1] P. Arias, V. Caselles, and G. Facciolo. Analysis of a variational framework for exemplar-based image inpainting. *Multiscale Modeling & Simulation*, 10(2):473–514, 2012. 1, 2, 6, 7

[2] P. Arias, G. Facciolo, V. Caselles, and G. Sapiro. A variational framework for exemplar-based image inpainting. *International Journal of Computer Vision*, 93(3):319–347, 2011. 1

[3] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving patchmatch for large displacement optical flow. *Image Processing, IEEE Transactions on*, 23(12):4996–5006, Dec 2014. 1

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009. 1, 2, 4, 5, 6, 8

[5] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized patchmatch correspondence algorithm. In *Proceedings of the Europ. Conf. on Computer Vision (ECCV)*, pages 29–43. Springer, 2010. 1, 2, 3, 5, 8

[6] C. Barnes, F.-L. Zhang, L. Lou, X. Wu, and S.-M. Hu. Patchtable: Efficient patch queries for large datasets and applications. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, Aug. 2015. 1

[7] N. Ben-Zrihem and L. Zelnik-Manor. RIANN: Approximate Nearest Neighbor Fields in Video. In *Proceedings of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 2

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 1

[9] M. Bleyer, C. Rhemann, and C. Rother. Patchmatch stereo - stereo matching with slanted support windows. In *Proceedings of the British Machine Vision Conference (BMVA)*, pages 14.1–14.11. BMVA Press, 2011. 1, 5

[10] X. Chen, T. Funkhouser, D. B. Goldman, and E. Shechtman. Non-parametric texture transfer using meshmatch. Technical Report Technical Report 2012-2, Adobe, November 2012. 1, 3

[11] D. Fortun, P. Bouthemy, and C. Kervrann. Sparse aggregation framework for optical flow estimation. In *Proceedings of the 5th Int. Conf. on Scale Space and Variational Methods in Computer Vision (SSVM)*, pages 323–334. Springer, 2015. 1

[12] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun. A global sampling method for alpha matting. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2049–2056. IEEE, 2011. 1

[13] K. He and J. Sun. Computing Nearest-Neighbor Fields via Propagation-Assisted KD-trees. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 111–118. IEEE, 2012. 1, 2

[14] M. Hornáček, F. Besse, J. Kautz, A. Fitzgibbon, and C. Rother. Highly overparameterized optical flow using PatchMatch Belief Propagation. In *Proceedings of the 13th Europ. Conf. on Computer Vision (ECCV)*, pages 220–234. Springer, 2014. 1

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symp. on Theory of Computing*, pages 604–613. ACM, 1998. 1, 6

[16] M. Klaudiny and A. Hilton. Cooperative patch-based 3D surface tracking. In *Proceedings of the Conf. for Visual Media Production (CVMP)*, pages 67–76, 2011. 1, 3

[17] S. Korman and S. Avidan. Coherency sensitive hashing. In *Proceedings of the IEEE Int. Conf. on Computer Vision (ICCV)*, pages 1607–1614. IEEE, 2011. 1, 2, 3, 6, 8

[18] N. Kumar, L. Zhang, and S. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *Proceedings of the Europ. Conf. on Computer Vision*, pages 364–378. Springer, 2008. 1

[19] C. Liu and W. T. Freeman. A high-quality video denoising algorithm based on reliable motion estimation. In *Proceedings of the Europ. Conf. on Computer Vision*, pages 706–719. Springer, 2010. 1

[20] J. Lu, H. Yang, D. Min, and M. N. Do. Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1854–1861, 2013. 1

[21] I. Olonetsky and S. Avidan. TreeCANN - kd-tree Coherence Approximate Nearest Neighbor algorithm. In *Proceedings of the Europ. Conf. on Computer Vision (ECCV)*, pages 602–615. Springer, 2012. 1