Artificial Multi-Bee-Colony Algorithm for k-Nearest-Neighbor Fields Search

Yunhai Wang
Shandong University
Shandong, China
wang.yh@sdu.edu.cn

Yiming Qian
University of Alberta
Edmonton, Canada
yqian3@ualberta.ca

Yang Li
Memorial University
St. John's, Canada
yl5026@mun.ca

Minglun Gong
Memorial University
St. John's, Canada
gong@cs.mun.ca

Wolfgang Banzhaf
Memorial University
St. John's, Canada
banzhaf@mun.ca

ABSTRACT

Searching the k-nearest matching patches for each patch in an input image, i.e., computing the k-nearest-neighbor fields (k-NNF), is a core part of various computer vision/graphics algorithms. In this paper, we show that k-NNF can be efficiently computed using a novel artificial multi-bee-colony (AMBC) algorithm, where each patch uses a dedicated bee colony to search for its k-nearest matches. As a population-based algorithm, AMBC is capable of escaping local optima. The added communication among different colonies further allows good matches to be quickly propagated across the image. In addition, AMBC makes no assumption about the neighborhood structure or communication direction, making it directly applicable to image sets and suitable for parallel processing. Quantitative evaluations show that AMBC can find solutions that are much closer to the ground truth than the generalized PatchMatch algorithm does. It also outperforms the PatchMatch Graph over image sets.

CCS Concepts

•Computing methodologies → Matching; Bio-inspired approaches; Image processing;

Keywords

Image Matching; k-NNF Search; Artificial Bee Colony Algorithm; Patch Match Algorithm

1. INTRODUCTION

Patch-based sampling has emerged as an important component in many computer vision and graphics applications, such as image denoising [5], texture synthesis [25] and object recognition [31]. Such sampling techniques rely on computing k nearest neighbors (k-NN) for each patch in the input image, where the neighboring patches can be from the image itself, another image, or even k different images in a large image set. Since a large image contains millions of small patches[1], computing the k nearest neighbor field (k-NNF) is a computationally expensive process.

By treating each patch as a point in a high-dimensional space, traditional k-NN search algorithms first organize the candidates into a tree [1, 26] or a hash table [15] and then query candidates using these structures. Although these methods are much faster than exhaustive search, they are still too slow for interactive image and video editing applications. The seminal algorithm PatchMatch (PM) [3] offers at least an order-of-magnitude improvement over previous methods in 1-NNF search. Its extended version, generalized PatchMatch (GPM) [4], can compute k-NNF in real time and supports searching in high dimensional spaces, e.g. with rotation and scaling.

The success of PM and GPM lies in two novel operators: propagation and random search. The former propagates good solutions to adjacent patches, whereas the latter tries to improve the current solutions using a coarse-to-fine search. Starting from initial random solutions, these two operators allow PM and GPM to converge quickly.

Nevertheless, these two operators have their limitations. First, designed for quickly locating reasonably good solutions, PM and GPM often converge prematurely to local optima. As the search space gets bigger, this tendency for premature convergence becomes more of a problem. Second, the coarse-to-fine random search operator implicitly assumes that patches adjacent in search space are also similar. While this assumption often holds for single images, it is no longer valid when the search space covers an image set, where adjacent patches along the dimension of the image index are unrelated. Finally, the data dependency in the propagation operator requires PM and GPM to be performed sequentially. Although jump flooding schemes [27] can be applied, the result is not as accurate as a sequential implementation.

Several approaches have been proposed to address these shortcomings under the 1-NNF search setting [14, 23]. However, as far as we know, few can be extended to the more general k-NNF search problem. Motivated by these findings, we introduce a population based approach, the artificial bee colony (ABC) algorithm [18], to compute the k-NNF. By representing each food source as a potential solution, the ABC algorithm employs the foraging behavior of honeybee colonies to solve numerical optimization problems. With a good balance between local and global search, it is not only effective in escaping local optima but also performs well in high dimensional search spaces [21].

[1]Patch size is often set to 8×8 for image processing applications. Different patches are allowed to partially overlap each other.

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908835

Given the need to search k-NN for each patch in the input image, a straightforward way of applying the ABC algorithm is to attach a bee colony to each patch and search within each colony independently. However, such an approach ignores the coherence between image patches, leading to slow convergence. Accordingly, we present an artificial multi-bee-colony (AMBC) algorithm, where a given bee colony can collect food sources from its adjacent colonies in both domain space and range space. This allows good solutions to be propagated in both local and non-local manners and can greatly speed up the convergence.

Quantitative evaluations on benchmark datasets show that our approach can find k-NNFs that are much closer to the ground truth than GPM does. The improvement becomes more significant as the size of the search space increases, i.e., from 1-NNF search to k-NNF search to search with rotation and scaling. Moreover, our algorithm can be directly used for computing k-NNF on an image set and shows better performance than PatchMatch Graph [11]. Since the processing for different colonies is independent, the algorithm can be easily parallelized on GPUs.

2. RELATED WORK

We here limit our review to the most relevant related work in the areas of patch-based k-NN computation, PatchMatch and its variants, and to the ABC algorithm and its applications.

2.1 k-NN Search

The most widely used method for k-NN search is kd-tree [1], which works well in low-dimensions but loses its effectiveness as dimensionality increases. To handle high dimensional image patches, dimensionality reduction methods such as PCA are often applied before using kd-tree. To better organize patch candidates, various types of tree structures have been proposed, including TSVQ [30] and vp-trees [24]. By combining multiple randomized kd-trees, the FLANN method [26] can automatically choose the best algorithm and optimal parameters depending on the data. To accelerate the k-NN search for texture synthesis, Tong et al. extended the concept of coherence [2] into *k-coherence* [29] which pre-computes k-NN for each patch in the input sample texture and uses them for synthesizing a new texture.

Locality sensitive hashing (LSH) [15] is another popular method for patch based k-NN search. Unlike kd-tree's deterministic partition, LSH [7, 17] produces random partitions using a large number of hash functions. With well designed hash functions, LSH offers sub-linear time search by hashing highly similar examples together in a hash table. Although efficient in searching, this method spends much time in building the hash tables, limiting its applicability in interactive image and video editing tasks.

k-NN search is applied to many real world applications associated with evolutionary computing algorithms in the literature. De La Vega *et al.* [8] combine k-NN search with differential evolution for time series forecasting. Cheng *et al.* [6] formulate Chinese text categorization as a nearest-neighbor-based optimization problem, which is then solved by the particle swarm algorithm. Jabbar *et al.* [16] solve the problem of heart disease classification by integrating k-NN classifier and genetic algorithm. In this paper, we focus on PatchMatch, which searches the k nearest patches for a given query patch in an image.

2.2 PatchMatch

By taking advantage of spatial coherence, PatchMatch (PM) has shown speedups of 20-100 times over previous state-of-the-art algorithm in 1-NNF computation. This is achieved by searching for novel solutions using random search and propagating good ones to nearby patches. To make long distance propagation, coherency sensitive hashing (CSH) [23] replaces the random search step with a hashing scheme used in LSH and propagates good candidates to patches with similar appearance. Propagation-assisted kd-trees [14] organize the candidates with a kd-tree and then use a non-iterative propagation method for fast querying. Since this method can propagate a group of candidates, it is 10-20 times faster than PM while maintaining the same accuracy. Although the recently introduced social-CSH approach [10] resembles our multi-bee colony, it only computes 1-NNF and is not applicable to k-NNF problem.

Generalized PatchMatch (GPM) [4] extends PM on three fronts: to perform k-NNF search, to support high dimensional search spaces, and to allow arbitrary similarity functions. Recently, GPM has been applied to color transformation [12] and computed k-NNF for an image set using PatchMatchGraph [11]. These approaches expand the search space but the core optimization algorithm has not been improved. In our experiments, we found GPM to be often trapped in local optima, especially when the parameter k or the dimensionality of the search space is large. With the goal of locating globally optimal solutions, we borrow ideas from population based search algorithms [19]. Quantitative evaluations show that our approach can find solutions that are up to 10 times closer to the ground truth than those of GPM.

2.3 The ABC Algorithm

As a swarm intelligence approach, the ABC algorithm [18] and its variants [32] can handle both unconstrained and constrained optimization problems [21, 20]. Although having several similarities with other population-based algorithms, the ABC algorithm has comparable or better performance and has the advantage of employing fewer control parameters [19]. These characteristics have motivated the use of ABC for various applications, such as numerical optimization [33], data clustering [22, 28] and feature selection [13].

As far as we know, we are the first to apply the ABC algorithm to solve the k-NNF search problem. In addition, we introduce the new concept of search with multiple bee colonies, where a dedicated colony is used to search the k-NN for each query patch. We also show that letting different colonies search independently does not yield satisfactory results since spatial coherence is ignored. A novel inter-colony onlooker bees operator is presented, which can effectively communicate good solutions among both local and non-local neighboring patches.

3. ARTIFICIAL MULTI-BEE-COLONY ALGORITHM

Here we start by explaining the original ABC algorithm [18] and how it can be applied to search the k-nearest neighbors (k-NN) for a single patch. Our multi-colony version of the ABC algorithm developed for searching the k-nearest neighbor fields (k-NNF) is presented next, followed by a discussion of its relationship with GPM.

3.1 ABC-based Algorithm for k-NN Search

The original ABC algorithm, as proposed by Karaboga [18], is an efficient population based optimization method inspired by the intelligent foraging behavior of bee colonies. By representing each solution as a food source, it searches the near optimal solutions with three kinds of bees: employed bees, onlookers and scouts. Each employed bee explores one food source and improves its position by searching nearby positions. After the employed bee completes the search process, it shares the food information with onlooker bees through waggle dance. Onlooker bees evaluate the nectar information taken from all employed bees and choose a food source

(a) Query patch in one of the input images (b) The k-nearest-neighbor found by GPM (c) The k-nearest-neighbor found by our approach

Figure 1: Visualization on a query patch (blue in (a)) from the k-nearest-neighbor field ($k = 16$) computed between two images. Matches found by our approach (red in (c)) are widely spread and include the corresponding feature, indicating the approach's global search power.

with a probability related to its nectar amount. Once a food source is chosen, an onlooker bee searches a better food source in the same manner used by employed bees. Hence, the number of food sources is equal to the numbers of both employed bees and onlooker bees. When a food source has been fully explored and cannot be improved anymore, the associated employed bee becomes a scout bee that carries out a random search for discovering new food sources. By default, there is at most one scout bee in each cycle.

3.1.1 k-NN Problem Formulation

The search for the k-NN of a query patch p can be formulated as the following optimization problem: Let $\mathcal{Q}$ denote the range space, i.e., the set of all possible patches that p can match to. We compute a subset Ω_p for patch p that minimizes the following average patch distance measure:

$$\Omega_p = \underset{\substack{\Omega_p \subset \mathcal{Q} \\ |\Omega_p| = k}}{\operatorname{argmin}} \left(\frac{1}{k} \sum_{q \in \Omega_p} D(p,q) \right) \tag{1}$$

where $|\cdot|$ denotes the size of a set and $D(\cdot,\cdot)$ symbolizes the distance between two input patches. Through treating patches as high dimensional vectors, $D(\cdot,\cdot)$ is computed here using Euclidean distance.

3.1.2 Initial Population

To apply the ABC algorithm, we start with a randomly distributed population of n ($n > k$ and $n = k+2$ by default) food sources sampled from $\mathcal{Q}$. In practice, a food source that corresponds to a matching patch solution q is represented using an m-dimensional vector $\mathbf{y}_q$, where the value of m depends on the range space $\mathcal{Q}$. For example, when searching for matches within a given image, $\mathbf{y}_q$ is a 2D vector that stores the 2D coordinates of the patches. When searching within an image set, $\mathbf{y}_q$ is a 3D vector, where the additional dimension encodes the image ID. Similarly, when rotation and scaling are allowed, additional dimensions are used to store the corresponding settings. Under such a representation, a random solution can be generated by:

$$\mathbf{y}[i] = R(\mathbf{y}^{\min}[i], \mathbf{y}^{\max}[i]), 0 \le i \le m, \tag{2}$$

where $\mathbf{y}^{\min}[i]$ and $\mathbf{y}^{\max}[i]$ are the lower and higher bounds of $\mathcal{Q}$ in the i^{th} dimension, respectively. $R(a,b)$ returns a random number that is uniformly distributed within the range $[a,b]$. To prevent duplicate entries in the population, we store all food sources in a hash table, which allows us to quickly identify whether a food source already exists.

In addition, we also associate a fitness value $F(\mathbf{y}_q)$ and a trial count $T(\mathbf{y}_q)$ to each food source $\mathbf{y}_q$ in the population. The fitness value is computed by:

$$F(\mathbf{y}_q) = \frac{1}{1 + D(p,q)}, \tag{3}$$

which outputs a higher value for patches with smaller distance to the query patch p. The trial count $T(\mathbf{y}_q)$ stores the number of iterations that $\mathbf{y}_q$ has gone through without being updated. It is initialized to 0.

3.1.3 Employed Bees

The task of employed bees is to explore the solution space by going through all food sources one by one and randomly perturbing each one. For each food source $\mathbf{y}_q$, we first randomly select another food source $\mathbf{y}_s$ within the population and compute:

$$\mathbf{y}'_q = \mathbf{y}_q + R(0,1)(\mathbf{y}_s - \mathbf{y}_q). \tag{4}$$

If $\mathbf{y}'_q$ does not exist in the population, then its fitness $F(\mathbf{y}'_q)$ is evaluated. $F(\mathbf{y}'_q)$ is compared with the fitness of the worst food source r in the population. Similar to GPM [4], a heap is maintained at each colony, which helps to quickly identify the worst food source r.

3.1.4 Onlooker Bees

To simulate the behavior of onlooker bees, which focus the search in the neighborhood of food sources with higher fitness, the probability of a food source $\mathbf{y}_q$ being chosen by an onlooker bee is computed as:

$$P(\mathbf{y}_q) = \frac{F(\mathbf{y}_q)}{\sum_{i=1}^{n} F(\mathbf{y}_i)}. \tag{5}$$

Once a food source $\mathbf{y}_q$ is picked based on this probability, a new food source $\mathbf{y}'_q$ is generated using Eq.(4). $\mathbf{y}'_q$ is used to replace the worst food source r if $F(\mathbf{y}'_q) > F(r)$. This importance-based sampling is repeated n times. Due to the randomness and non-uniform distribution of probabilities, some food sources may be selected more than once, whereas some others may not be used at all.

3.1.5 Scout Bees

When all food sources converge to a small neighborhood, the new food sources found by both employed bees and onlooker bees using Eq.(4) are constrained to be within the neighborhood. To prevent the search from being trapped in local optima, the scout bees are used to abandon a food sources that cannot be improved after T trails, where T is a threshold of the trial count. That is, a

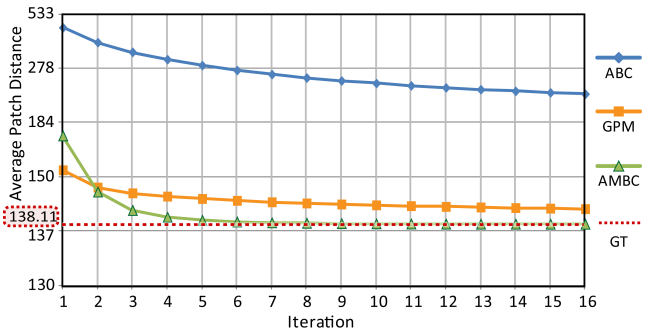


Figure 2: When computing the k-NNF that maps Figure 1(b) to itself, the performances of these methods (GPM, straightforward ABC and our AMBC) are compared with the ground truth (GT, red dash line). Note that log scale is used for the Y axis.

scout bee picks a food source $\mathbf{y}_q$ when its trial count is lager than T and then replaces $\mathbf{y}_q$ with a new solution generated using Eq.(2).

Since our objective is to search for the best k solutions, not just the best one, we do not want the top k food sources in the population to be replaced with generally worse random solutions. Hence, rather than picking the foods based on the trial counts, our scout bees selects among the the $n-k$ food sources with the lowest fitness values. In addition, we allow multiple food sources that pass the threshold to be replaced, which we found works better in our experiments.

3.1.6 Termination Conditions

As explained above, employed bees search within the neighborhood defined by all solutions in the population, onlooker bees focus on the areas around the best solutions in the population, whereas scout bees conduct global and random search. With a good balance between global exploration and local exploitation using these three levels of search, the above ABC algorithm can quickly converge to a population that gives the near-optimal k-NN for the query patch. In practice, we terminate the iterative process once a given number of iterations has been reached.

3.2 AMBC Algorithm for k-NNF Search

To compute the k-NNF for an input image I or image set S, we associate a bee colony to each patch p in I or S and use it to search for p's k-NN. While we could use the ABC-based algorithm above to search for solutions independently, such an approach ignores the coherence among neighboring patches. We here show that through modeling the communication among neighboring bee colonies using a novel inter-colony onlooker bee operator, we can efficiently propagate good solutions across different patches and hence find the optimal k-NNF much faster. The benefit of this novel inter-colony onlooker bee is shown in Figure 2, which compares the average patch distance (see [4] or Eq.(6) for definition) for the k-NNF computed using different methods. It shows that the solutions obtained by the straightforward ABC algorithm are far from the ground truth (GT) and is worse than GPM. In comparison, our AMBC algorithm reduces the error of GPM (the discrepancy with the ground truth) by 90%.

Since the operations of employed and scout bees remain the same in the AMBC algorithm, the rest of this section focuses on how inter-colony onlooker bees work.

3.2.1 Problem Formulation

The search for the k-NNF can be formulated as the following optimization problem: Let $\mathscr{P}$ denote the domain space, i.e., a set of all patches in input image I or image set S, and $\mathscr{Q}$ again denote the range space. We like to compute a superset $\Omega = \{\Omega_p | p \in \mathscr{P}\}$ that minimizes the following average patch distance measure:

$$\Omega = \underset{\substack{p\in\mathscr{P},\Omega_p\subset\mathscr{Q}\\ |\Omega_p|=k}}{\operatorname{argmin}} \left(\frac{1}{k|\mathscr{P}|} \sum_{p\in\mathscr{P}} \sum_{q\in\Omega_p} D(p,q) \right) \tag{6}$$

3.2.2 Neighborhood Setup

To enable communication among neighboring bee colonies, we first need to define the neighborhood for a given patch p. Here we consider two types of neighbors: those that are adjacent to p in the domain space $\mathscr{P}$ and those adjacent to p in the range space $\mathscr{Q}$. Patch q is considered as p's range space neighbor if q is one of the k-NN found for p so far. Hence, a patch always has k range space neighbors.

The number of p's domain space neighbors, denoted as h, depends on the configuration of the domain space. Here we identify a patch p using its coordinate $\mathbf{x}_p$ in the domain space. When the task is to compute k-NN for patches of a fixed size, the domain space has a dimension of 2 and we have $h = 4$. These four neighbors are $\mathbf{x}_p + \Delta\mathbf{x}, \Delta\mathbf{x} \in \{(1,0),(-1,0),(0,1),(0,-1)\}$. When the task is to compute k-NN for patches of different sizes, the domain space has an extra dimension on patch size and $h = 6$. The two additional neighboring patches center at the same location as p does but have different sizes. That is, we have $\Delta\mathbf{x} \in \{(1,0,0),(-1,0,0),(0,1,0),(0,-1,0),(0,0,1),(0,0,-1)\}$. Please note that even though computing k-NN for an image set also requires the domain space having an extra dimension for the image ID, we do not consider patches from different images to be adjacent. Hence, h does not always equal twice the dimension of $\mathbf{x}_p$.

3.2.3 Inter-colony Onlooker Bees

With the neighborhood defined for each patch p in $\mathscr{P}$, we now discuss how to communicate among the bee colonies associated with these neighbors. It is worth noting that in some applications, e.g., computing k-NNF that map an image to itself or searching k-NNF within an image set [11], the domain and the range spaces are the same, i.e., $\mathscr{P} = \mathscr{Q}$. As a result, all p's neighbors have their k-NN calculated and have bee colonies associated with them. In some other applications, such as mapping patches in an image I to another image J, the two spaces are different. In these cases, we require that the k-NNF mapping from $\mathscr{Q}$ onto itself is already calculated with the AMBC algorithm during a pre-processing step. Hence, we can assume there are bee colonies associated with all of p's neighbors, even though some of these bee colonies may have already converged and are no longer active.

The core idea of our inter-colony onlooker bee operator is that onlookers shall not only watch the dances performed by the employed bees from its own colonies, but also the dances of employed bees from neighboring colonies. We here denote the onlooker bees watching domain and range space neighbors as domain and range onlookers, respectively. After evaluating the fitness of these food sources, onlooker bees will select the best food sources and use them to replace existing ones. Note that, due to the large number of candidate food sources available, we found that it is more efficient to let onlooker bees just pick the best n candidates in a greedy manner, rather than based on the probabilities defined in Eq.(5).

The food sources collected from neighboring colonies may need to be adjusted, before letting onlooker bees evaluate their fitness. The actual adjustment depends on whether a neighbor is from the

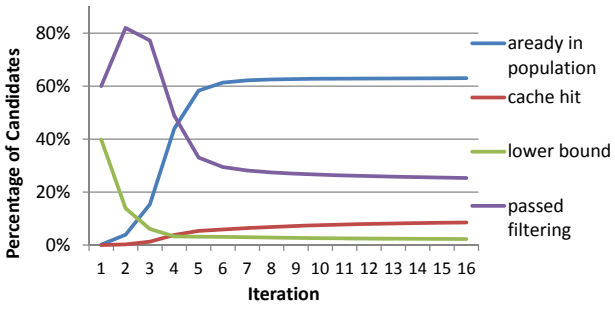


Figure 3: The percentage of candidates that are either removed by the three filtering steps or are left for fitness evaluation.

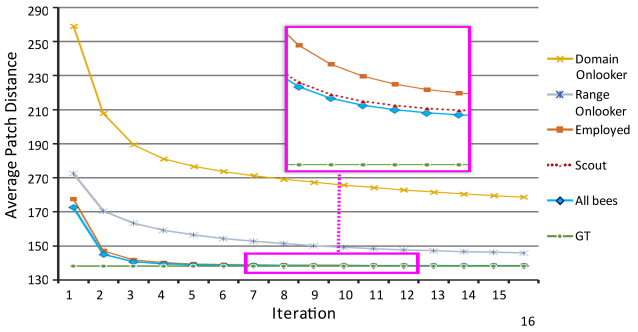


Figure 4: The contribution of each kind of bees to the AMBC algorithm.

domain space or the range space, as well as whether the solution space includes patches with different sizes and orientations. For example, when no scaling or rotation is allowed, all food sources $\mathbf{y}_q$ collected from domain space neighbor $\mathbf{x}_p + \Delta\mathbf{x}$ are adjusted to $\mathbf{y}_q - \Delta\mathbf{x}$, whereas the ones collected from range space neighbors do not need to be adjusted. On the other hand, when scaling or rotation is allowed, food sources collected from range space neighbors need to be adjusted using composition operators. Please refer to [11] for details.

3.2.4 *Candidate Filtering*

The h domain space neighbors and k range space neighbors provide a total of $(h+k)\times n$ candidates in total. Blindly evaluating the fitness for all these candidates results in high computation cost. To reduce unnecessary testing, a number of strategies are applied.

First of all, candidates collected from neighboring colonies may already exist in the current population. These candidates are filtered out first using the hash table mentioned in Section 3.1.

Next, even if a candidate does not exist in the population, it may have already been evaluated during either the previous iteration or the current one, and may have been found not good enough. Hence, there is no need to repeat the fitness calculation. To detect these candidates, we maintain a fixed sized cache using a hash table. A food source $\mathbf{y}_q$ is inserted into the cache when i) $F(\mathbf{y}_q)$ is computed but is not competitive; or ii) $\mathbf{y}_q$ is currently in the population but is being replaced by another candidate. With such a cache, a candidate is ignored if there is a cache hit.

Finally, the patch distance function in Eq.(1) can be taken as the Euclidean distance defined in high dimensional space, which satisfies the triangle inequality [9]. Hence, for each candidate patch t that we collected from p's range space neighbor s, we can estimate the lower bound for patch distance $D(p,t)$ using $D(p,s)$ and $D(s,t)$, both of which are already calculated. That is, we have $D(p,t) \geq |D(p,s) - D(s,t)|$. If the fitness estimated based on $|D(p,s) - D(s,t)|$ is lower than the worst food source in the population, then there is no need to compute the precise fitness $F(\mathbf{y}_t)$.

Figure 3 depicts the percentage of candidates being filtered by the above three strategies. It shows that, during initial iterations, up to 40% of the candidates can be removed using the lower bound computed based on triangle inequality. As the population converges, the first two filtering steps are more effective and can remove up to 70% of the candidates together. Hence, the three strategies complement each other very well.

3.2.5 *Algorithm Outline*

As shown in Alg. 1, the AMBC algorithm consists of four kinds of bees: employed bees, scouts, domain space onlookers, and range space onlookers. Figure 4 further illustrates the contribution of each kind of bee to the overall optimization by commenting out the corresponding bee operation when running the AMBC. It demonstrates that both onlooker bees play important roles in searching for good solutions. Without them, the optimization converges much slower, which explains why the straightforward ABC approach does not work well. The employed bee and scout bee also help to bring solutions closer to the global optimum, with the scout bee being the least useful.

Since there is no data dependency between different stages, our algorithm can be easily parallelized on CPUs or GPUs. On a multi-core CPU, our algorithm can obtain a linear speed-up, while our unoptimized GPU implementation is roughly 20x faster than the single core CPU algorithm on a NVIDIA Quadro K4000 card.

Algorithm 1 Artificial multiple bee colonies algorithm

```
for each query patch p in 𝒫 do in parallel
    Initialize p's population using Eq.(2)
    Evaluate all food sources in the population
    cycle=1
    repeat
        Synchronize all threads before employed bee
        for each food source y_q in p's population
            Produce a new solution y'_q using Eq.(4)
            Calculate F(y'_q)
            Replace y_q if F(y'_q) > F(y_q)
        Synchronize all threads before onlooker bees
        for each neighbor s of p
            for each y_t in the population of s
                Adjust y_t if necessary
                Apply candidate filtering
                Calculate F(y_t) if needed
                Add y_t to p's population if needed
        Synchronize all threads before scout bees
        for each n − k food sources with lowest fitness
            Find food source y_q with T(y_q) > T
            Replace y_q with a new solution using Eq.(2)
        cycle = cycle+1
    until cycle=MCN
    Output the best k food source in p's population
```

3.3 Comparison with GPM

Although derived from the ABC algorithm, the AMBC approach presented above bears many similarities with the GPM approach. In what follows we review these similarities, as well as the differences.

In AMBC, the employed bees and the scout bees are used to conduct local and global random search, respectively. Hence, they roughly correspond to the random search step in GPM. However, the coarse-to-fine search strategy used in GPM always performs an equal number of global and local searches throughout the iterative process. In contrast, AMBC applies employed bees much more often than scout bees. To be precise, within each iteration, employed bees are performed n times, whereas scout bees at most $n-k$ times, with actual numbers depending on the trail threshold T. In addition, since the employed bees always search within the convex hull formed by the current population, its search scope is automatically adjusted during iterations. At the beginning when the population is widely spread, the employed bees explore a bigger area. When the population converges, the search scope gets much smaller.

Collecting information from domain space neighbors, the inter-colony onlooker bees' operation is closely related to the propagation step in GPM. However, unlike GPM, which requires propagation among patches being conducted in a sequential order, our onlooker bees are used for different patches independently, which facilitates parallel implementation. In theory, when propagation is performed in parallel, it takes more iterations to send information over a long distance. However, this is not a problem in AMBC since range space neighbors are also used, which generally propagate information in a non-local manner.

Our range space neighborhood is similar to the concept of forward enrichment in GPM. However, instead of performing forward enrichment only once or twice as post-processing, we use range space neighbors to enhance solutions in every iteration. While the number of candidates collected from all range space neighbors, i.e., $k \times n$, seems prohibitive, we show that a carefully designed filtering process can remove the majority of these candidates.

In general, AMBC spends more time evaluating candidates collected from neighbors (up to $(h+k) \times n$) than candidates obtained by random search (up to $2n-k$). In comparison, GPM tests $2k$ candidates that come from the two local neighbors and $k \times \log W$ candidates generated by random search, where W is the image resolution. Not only are there too many tests used by random search, which renders it less and less effective when converging, but also the computational costs increase as input images become larger.

Figure 5 plots the number of candidates that went through fitness evaluation vs. the candidates that are selected into the k-NN in the end. It shows that, after the initial four iterations, AMBC actually makes fewer patch distance calls than GPM, thanks to the effective filtering process. In addition, although fewer calls are made, more candidates are selected into the k-NN at each iteration, which suggests that the solution keeps improving. In comparison, after the first four iterations, on average GPM replaces less than one solution per patch at each iteration.

Since the ABC algorithm is found to be efficient in solving optimization problems in a high-dimensional search space [21], the performance gain of our AMBC algorithm over GPM also increases as the search space grows larger. As shown in Figure 6, with scaling and rotation options being added to the k-NNF search, both GPM and AMBC need more iterations to converge and the error differences of the corresponding converged solutions become bigger and bigger.

It is worth noting that even though only the best k solutions are needed for k-NNF search, AMBC maintains the best n food sources at each pixel. The additional $n-k$ solutions can be useful since i) random search may turn a solution into a top k solution; ii) after being propagated to neighboring pixels, a solution may be a top k solution. A similar idea can be applied in GPM, i.e., the best n solutions are maintained at each pixel but only the top k are used as output in the end. Figure 7 compares the impacts of parameter

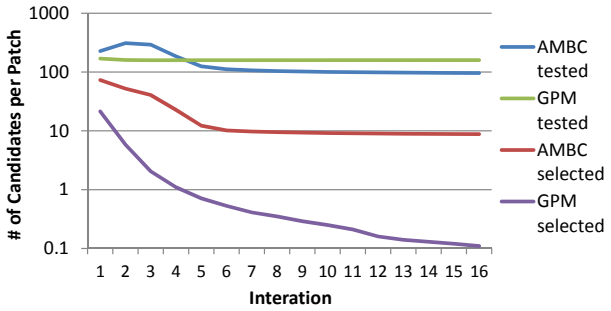


Figure 5: The number of candidates that have their fitness evaluated vs. candidates that actually become the k-NN.

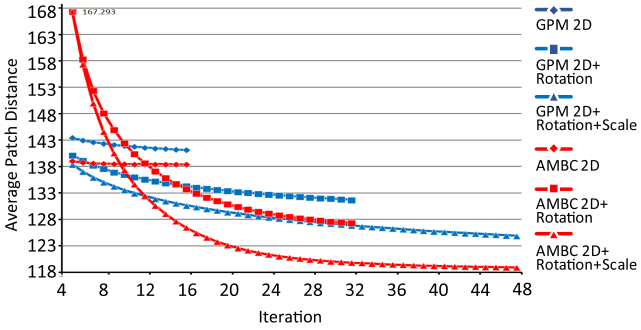


Figure 6: The convergence curves between GPM and AMBC on tasks with different search space dimensions.

n to both approaches. It shows that making n slightly larger than k (e.g., $n = k+2$) is sufficient for AMBC. On the other hand, while using larger n does improve the performance of GPM, the result is still suboptimal even when $n = 2k$, which noticeably slows down the GPM approach. This confirms that the performance gain of AMBC over GPM is not simply due to additional food sources.

4. EXPERIMENTS

Section 3.3 has shown a number of comparisons between AMBC and GPM using the test images shown in Figure 1. Here we further compare AMBC with both GPM and PatchMatch Graph on two sets of images. The implementations for GPM and PatchMatch Graph come from the respective authors and are written in C++. Although we have implemented AMBC on a GPU, we only use the CPU implementation (in C++) for testing to allow for a fair comparison on timing. When comparing with GPM, a PC with Intel Core i7 3.5GHz CPU and 4GB RAM is used for testing. Running PatchMatch Graph on image sets requires large memory and hence a PC with Intel Xeon E5-2687 3.4GHz CPU and 32GB RAM is used.

4.1 Comparison with GPM on Matching within Images or between two Images

We first test on the public data set VidParis [23], which contains 133 pairs of images. We downsample these images to 0.4M pixels for our experiments. Two experiments are conducted. The first one uses the second image in each pair to compute the k-NNF ($k = 16$, patch size $= 8 \times 8$) that matches among patches within the given image. For each image, three k-NNFs are computed using GPM, AMBC, and exhaustive search, respectively. The number of food sources is set to $n = 18$ for AMBC. The k-NNF obtained by exhaus-

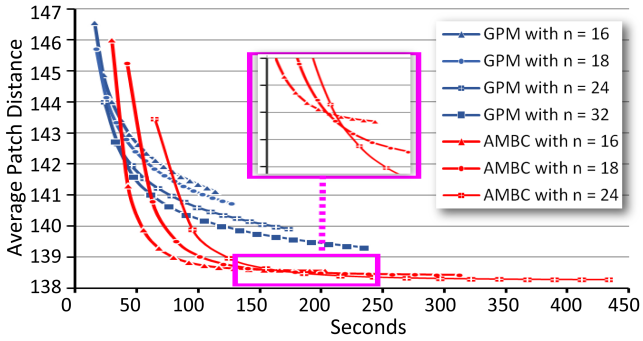


Figure 7: Comparing AMBC with GPMs under different values of n. Since the X-axis is processing time, the horizontal distance between adjacent data points indicates the time needed for each iteration, which increases as the value n does.

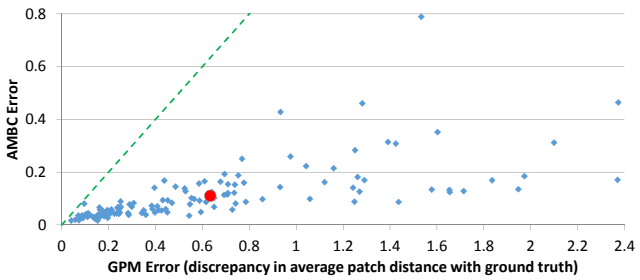
tive search is considered as the ground truth and is used to evaluate the errors of k-NNFs generated by GPM and AMBC, where the error is defined as the difference in average patch distances. Figure 8(a) compares the errors of GPM and AMBC results for each of the 133 images, where GPM error and AMBC error are drawn in X and Y axis for each image, respectively. It shows that the AMBC yields less errors for all these images. On average, the error of AMBC is only $1/5$ of the error produced by GPM, whereas in extreme cases, this ratio goes down to $1/14$.

The second experiment evaluates the k-NNF generated for mapping from the first image to the second image in each image pair. In this case, a pre-processing step is needed to establish the matching from $\mathscr{Q}$ to $\mathscr{Q}$, i.e., mapping the second image to itself, so that the range space onlooker bees can be applied. In practice, we found that there is no need to fully compute the k-NNF for $\mathscr{Q}$ to $\mathscr{Q}$ mapping. Instead, the results obtained by running the AMBC for three iterations are sufficient for guiding the range space onlookers. Figure 8(b) evaluates the k-NNFs obtained by GPM and AMBC using the ground truth. It shows that GPM has noticeable better performance than AMBC on only 2 out of 133 pairs. The average error of AMBC's results is $1/3$ of that of GPM's, whereas in extreme cases, the error of AMBC is about $1/10$ of GMP's.

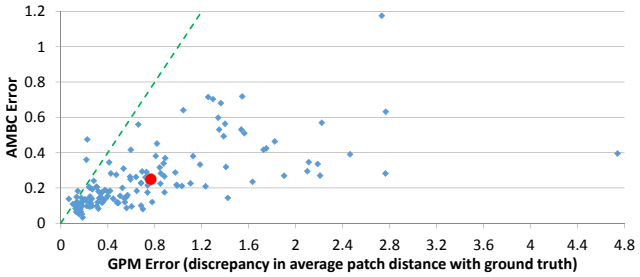
4.2 Comparison with PatchMatch Graph on Image Sets

Given an image set, PatchMatch graph searches k nearest neighbors for each image patch in an image set under the constraint that no two neighboring patches can come from the same image. The Polo dataset [11] is used here for testing, which contains 80 images downsampled to 50% of the original size. By replacing the searching portion of the authors' PatchMatch graph implementation with our AMBC approach, we are able to compare AMBC with PatchMatch graph under the same settings. These include $k = 10$, patch size $= 8 \times 8$, patch distance being computed in the CIELab color space using L^1 distance. The search space includes translation, scaling, and rotation (4D in total), where the rotation is limited to horizontal and vertical flips.

Figure 9 plots the convergence curves of PatchMatch graph and our AMBC approach under different number of food sources. The results demonstrate that PatchMatch graph converges prematurely after about 14 mins. AMBC, on the other hand, keeps improving results and generates much better final solutions.



(a) Evaluation on the k-NNF that maps the image b in each pair to itself



(b) Evaluation on the k-NNF that maps image a to image b within each pair

Figure 8: The error comparison between GPM and AMBC over a set of 133 image pairs $\{< a,b >\}$. Each blue data point represents one of the images, whereas the red dot shows the average over all images. The green dash line shows $X = Y$.

5. DISCUSSIONS AND FUTURE WORK

This paper proposes a novel AMBC algorithm for computing the k-NNF that maps patches from an input image to either the same image, a different image, or an image set. Quantitative evaluations show that AMBC can avoid local optima and converge to more accurate solutions than GPM and PatchMatch Graph. The key of the success is the proposed inter-colony onlooker bees operator, which propagates good solutions among different bee colonies in both local and non-local manners. The experimental results demonstrate that without using the inter-colony onlooker bees, the performance suffers significantly.

While the GPM also utilize a propagation operator, it only propagates solutions between neighboring pixels under a predefined order. Hence, different pixels have to be processed sequentially. The inter-colony onlooker bees operator breaks the data dependency through propagating solutions in parallel manner. This not only facilitates a GPU implementation but also makes it possible to apply AMBC to problems where the domain spaces do not have naturally defined orders. Examples include finding the best matching nodes between two graphs, matching points between two 3D point clouds captured by laser scanning, and matching vertices between two 3D mesh models, where the nodes, points, and vertices are not ordered. In the future, we plan to apply AMBC to these problems. We would also like to explore other optimization problems, where the overall task can be decomposed to multiple sub-problems and the solutions found for one sub-problem can provide useful hints to related sub-problems. In terms of limitation, the k-NNF found by AMBC is not as good as that of GPM at the beginning of the search (first 50 secs for the example shown in Figure 7). This limits its usage in time critical applications, although the GPU implementation can reduce this time 20-fold.

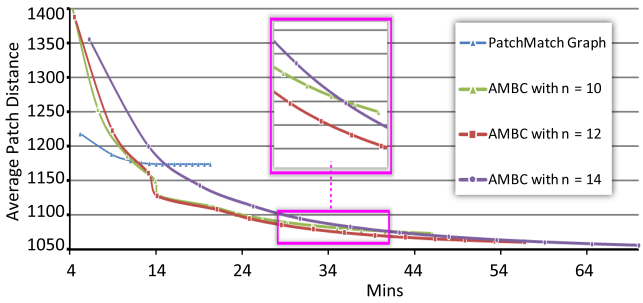


Figure 9: Search k-NNF ($k = 10$) for an image set using PatchMatch graph and AMBC under food sources number n.

6. ACKNOWLEDGMENTS

We thank NSERC, NSFC-Guangdong Joint Fund (U1501255), and 973 program (2015CB352501) for the financial support.

7. REFERENCES

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[2] M. Ashikhmin. Synthesizing natural textures. In *Proceedings of the symposium on Interactive 3D graphics*, pages 217–226. ACM, 2001.

[3] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. (SIGGRAPH)*, 28(3):24:1–24:10, 2009.

[4] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized patchmatch correspondence algorithm. In *Proceedings of European Conference on Computer Vision*, pages 29–43, 2010.

[5] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 60–65. IEEE, 2005.

[6] S. Cheng, Y. Shi, Q. Qin, and T. Ting. Particle swarm optimization based nearest neighbor algorithm on chinese text categorization. In *Swarm Intelligence (SIS), 2013 IEEE Symposium on*, pages 164–171. IEEE, 2013.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[8] E. De La Vega, J. J. Flores, and M. Graff. k-nearest-neighbor by differential evolution for time series forecasting. In *Nature-Inspired Computation and Machine Learning*, pages 50–60. Springer, 2014.

[9] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pages 147–153, 2003.

[10] Y. Eshet, S. Korman, E. Ofek, and S. Avidan. Dcsh-matching patches in rgbd images. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 89–96. IEEE, 2013.

[11] S. Gould and Y. Zhang. Patchmatchgraph: Building a graph of dense patch correspondences for label transfer. In *Proceedings of European Conference on Computer Vision*, pages 439–452. Springer, 2012.

[12] Y. HaCohen, E. Shechtman, D. B. Goldman, and D. Lischinski. Non-rigid dense correspondence with applications for image enhancement. *ACM Trans. Graph. (SIGGRAPH)*, 30(4):70, 2011.

[13] E. Hancer, B. Xue, D. Karaboga, and M. Zhang. A binary abc algorithm based on advanced similarity scheme for feature selection. *Applied Soft Computing*, 36:334–348, 2015.

[14] K. He and J. Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 111–118, 2012.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[16] M. A. Jabbar, B. Deekshatulu, and P. Chandra. Classification of heart disease using k-nearest neighbor and genetic algorithm. *Procedia Technology*, 10:85–94, 2013.

[17] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[18] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report 06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[19] D. Karaboga and B. Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132, 2009.

[20] D. Karaboga and B. Basturk. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In *Foundations of Fuzzy Logic and Soft Computing*, pages 789–798. Springer, 2007.

[21] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[22] D. Karaboga, S. Okdem, and C. Ozturk. Cluster based wireless sensor network routing using artificial bee colony algorithm. *Wireless Networks*, 18(7):847–860, 2012.

[23] S. Korman and S. Avidan. Coherency sensitive hashing. In *Proceedings of IEEE International Conference on Computer Vision*, pages 1607–1614, 2011.

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

[25] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001.

[26] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340, 2009.

[27] G. Rong and T.-S. Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, 2006.

[28] Q. Tan, H. Wu, B. Hu, and X. Liu. An improved artificial bee colony algorithm for clustering. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 19–20, New York, NY, USA, 2014. ACM.

[29] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Graph. (SIGGRAPH)*, 21(3):665–672, 2002.

[30] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH*, pages 479–488, 2000.

[31] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2005.

[32] W.-J. Yu, J. Zhang, and W.-N. Chen. Adaptive artificial bee colony optimization. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 153–158, New York, NY, USA, 2013. ACM.

[33] G. Zhu and S. Kwong. Gbest-guided artificial bee colony algorithm for numerical function optimization. *Applied Mathematics and Computation*, 217(7):3166–3173, 2010.