Random Grids: Fast Approximate Nearest Neighbors and Range Searching for Image Search

Dror Aiger, Efi Kokiopoulou, Ehud Rivlin
Google Inc.
aigerd@google.com, kokiopou@google.com, ehud@google.com

Abstract

We propose two solutions for both nearest neighbors and range search problems. For the nearest neighbors problem, we propose a c-approximate solution for the restricted version of the decision problem with bounded radius which is then reduced to the nearest neighbors by a known reduction. For range searching we propose a scheme that learns the parameters in a learning stage adopting them to the case of a set of points with low intrinsic dimension that are embedded in high dimensional space (common scenario for image point descriptors). We compare our algorithms to the best known methods for these problems, i.e. LSH, ANN and FLANN. We show analytically and experimentally that we can do better for moderate approximation factor. Our algorithms are trivial to parallelize. In the experiments conducted, running on couple of million images, our algorithms show meaningful speed-ups when compared with the above mentioned methods.

1. Introduction

Proximity problems in high dimensional spaces find many applications in computer vision. Image structures are commonly described by points in space and one is searching for similar structures by applying proximity searching in this space. A considerably larger effort has been invested in preprocessing-and-query problems, where the goal is to construct some data structure on the input point set which supports proximity queries of a certain kind. The most common of this kind of problems is *nearest neighbor searching*, where we are given a set P of n points in a high-dimensional space $\mathbb{R}^d$, and wish to construct a data structure that, given a query point q, finds the point(s) in P closest to q. Extensive research on this problem has led to a variety of interesting solutions, both exact and approximate. The dependence on d of the performance of the resulting algorithms is at least exponential.

Many of the known exact and approximate nearest neighbor searching data structures can be modified to report all (or most) points of P that are within a certain given distance r from a query point q. We give a brief review of the state-of-the-art approximate nearest neighbor data structures. For more information and related references see Har-Peled's recent book [7]. These approximate nearest neighbor data structures return, for any query point q, a point p whose distance from q is at most $(1+\varepsilon)$ times the distance between q and its nearest neighbor. A data structure based on Box-Decomposition Trees, partitioning space into axis-aligned boxes, was given by Arya et al. [4]. This structure takes $O(n)$ space, can be constructed in $O(n \log n)$ time, and answers a query in $O(\frac{1}{\varepsilon^d} \log n)$ time. To date, many trade-offs between query time and space have been achieved, and in all of the more efficient ones the product of the term depending on ε in the storage and the square of the term depending on ε in the query time is roughly $\frac{1}{\varepsilon^d}$ [5].

To overcome the exponential dependence on d of the performance of all these data structures, Indyk and Motwani introduced a different technique called *Locally Sensitive Hashing (LSH)* [9]. The first component in this method is a reduction from the problem of approximate nearest neighbor search to the problem of finding a neighbor at distance $\leq (1+\varepsilon)r$ if there exists a neighbor at distance r, for some pre-specified $r > 0$. Then the latter problem is solved using a family of hash functions that tend to map close points to the same bin (an LSH family in short). The solution of Indyk and Motwani answers a query in $O(n^{\frac{1}{1+\varepsilon}})$ time and takes $O(n^{1+\frac{1}{1+\varepsilon}})$ preprocessing time and storage. This was later improved, using more complex hash func-

tions, to $O(n^{\frac{1}{(1+\varepsilon)^2}})$ query time and $O(n^{1+\frac{1}{(1+\varepsilon)^2}})$ preprocessing and storage [3, 6].

From the more practical point of view in computer vision, finding the best matches for local image features in large datasets was considered by [14]. Nister and Stewenius [13] presented a fast method for nearest-neighbor feature search in very large databases. Their method is based on accessing a single leaf node of a hierarchical k-means tree. FLANN is a method proposed in [12] for automatically selecting the best method and its parameters for a given training set. It was shown to be fast in practice and is part of the OpenCV library. The C++ library of Arya and Mount [4] contains an implementation of their BBD-trees and kd-trees. For LSH, we also mention the E2LSH library by Andoni and Indyk [3]. We have used all these three for comparison with our new method.

In this paper we consider the problems of nearest neighbors and range searching for the application of image matching. We focus on the decision problem called the (c, r)-NN where we only interested in the neighbors if they are close enough to the query. While there is a known reduction (increasing the query time by a factor $\log n$) from the general nearest neihgbors to this problem, it seems that for practical applications, one can usually use this when r is specified in the input. We also present an algorithm for the range searching problem where we are interested in reporting all points at distance at most r from the query. Here we allow randomization and we obtain a randomized algorithm that is guaranteed to report points having pre-specified high probability. In many applications the input set P has a restricted structure, in the sense that its points have much fewer "degrees of freedom" than the ambient dimension d. For example, all the points of P might lie on, or near, some manifold of much smaller dimension. Low doubling dimension was exploited before for fast algorithms for approximate nearest neighbor searching [8, 11].

Our algorithms are based on recent results by the first author et al. [1]. In that work, a different problem is investigated where there is no index. The set of points, P, is given along with a pre-specified radius $r > 0$, and one wants to report all pairs in P at distance at most r. We show that the analysis is applicable for the problems we investigate in this paper and we show experimentally that the good practical results obtained there can be repeated also for the problems discussed in this paper.

2. The Problems

We give first a formal description of the problems:

(c, r)-NN The (c, r)-approximate near neighbor problem (or (c, r)-NN) with failure probability δ is to construct a data structure over a set of points P in metric space (X, D) supporting the following query: given any fixed query point $q \in X$, if $D_p(q) \leq r$, then report some $p \in P \cap B(q, cr)$, with probability $1 - \delta$ where $B(q, r)$ is the ball with radius r around q.

r-RRS The r-Randomized Range Searching (or r-RRS) problem with failure probability δ is to construct a data structure over a set of points P in metric space (X, D) supporting the following query: given any fixed query point $q \in X$, report all points $p \in P \cap B(q, r)$ where every such point is reported with probability $1 - \delta$.

By scaling, we will assume $r = 1$ for both problem. Our results from [1] have implications on both.

3. Proposed Algorithms

Next we describe the theory that covers the foundations for the proposed algorithms. We analyze the worst case scenario and compute the worse case running time. We then compare the proposed algorithm with the LSH scheme. We then describe the practical implementation of these algorithms and add remarks on the applicability to image search as to the way to parallelize the algorithms.

3.1. Theory

An important result from [1] states that if p and q are two points at distance at most 1 in d-dimensional Euclidean space and we impose a randomly rotated and shifted grid of cell size w on this space, then the probability of capturing both p and q in the same cell is at least $e^{-\sqrt{d}/w}$ for sufficiently large w. We use this result to construct an efficient index for fast query.

We impose a grid of cell size $w = \frac{c}{\sqrt{d}}$ on P. Based on the result above, a random unit vector in R^d will be captured in one cell with probability at least $e^{-\sqrt{d}/w} = e^{-\frac{d}{c}}$. We create $e^{\frac{d}{c}}$ copies of P, randomly rotated and shifted, and store them in the grid cells where each non empty cell contains a list of the points contained in it for every rotated/shifted copy. For each random translation/rotation, we use

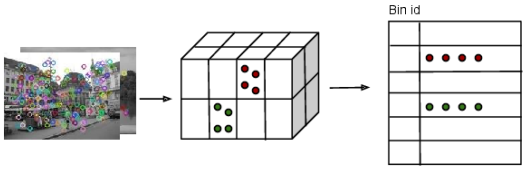


Figure 1: Randomized Grids indexing process: Feature descriptors are extracted from the key points of the dataset of images and represented as a set of points in R^d, where a series of random grids is applied. For each random grid, the points falling in the same bin hash to the same value.

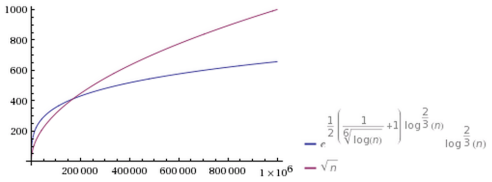


Figure 2: Numerical comparison to the p-stable LSH schemes for $c = 2, a = 2/3$. This is a plot of the query time bound we obtained and the p-stable LSH that has query time $O(n^{1/c})$.

a hash table for indexing, where points falling in the same bin hash to the same value (see Fig. 1 for an illustration). For a given query point q, we rotate and shift q, $e^{\frac{d}{c}}$ times by the corresponding transformations and report the first point found in the corresponding grid cells. With positive constant probability, if there is a point at distance at most 1 from q, it will be found in one of the grids cells. On the other hand, the reported point is at distance at most c from q. The space is $O(de^{\frac{d}{c}}n)$ and the preprocessing time is $O(d^2e^{\frac{d}{c}}n)$ (in computational model with floor function) and the query time is $O(d^2e^{\frac{d}{c}})$. The probability of success can be amplified to $1-\delta$ by repeating the process (increasing the data structure, space and query time) $\ln(1/\delta)$ times.

Worst case analysis We derive below the theoretical upper bound on the running time for the worst case scenario. We also compare it with state-of-the-art methods from the literature. If the dimension d is high, we can reduce the query/space bound using the Johnson-Lindenstrauss (JL) lemma [10] following the same direction as in [3]. We reduce d to $t = O(\log^a n)$ for some fixed small $a < 1$. For any p, q such that $\|p-q\| = 1$, the distortion of the projection to dimension t, is consider to be *high* when it is either greater than $1+\varepsilon$ or smaller than $1-\varepsilon$, for some ε. The probability of high distortion is upper bounded by $f = e^{-\Omega(\varepsilon^2 t)}$, using standard bound on distortion under random projections [10, 3, 9]. As in [3], we use $\varepsilon = t^{-1/4}$ and then we obtain that with high probability the distortion is smaller than ε. We have to tune the grid size accordingly to still have c-approximation with high probability when we first project P to dimension t. For this we decrease the grid cell size by a factor of $1 + t^{-1/4}$ to $c/((1 + t^{-1/4})\sqrt{t})$. Then the query time becomes $O(tde^{t(1+t^{-1/4})/c})$. The projection time can be improved to $O(d \log d)$ instead of $O(td)$ by the FJLT [2]. Setting $t = O(\log^a n)$ we get that with high probability, a c-approximation near neighbor is obtained in time

$$O((d \log^a n)e^{\frac{(\log^a n)(1+\log^{-a/4} n)}{c}})$$

In general, one can compute the best a that minimizes this expression. Even better, one can set t to the value that minimizes the query expression.

We compare now with LSH. Our algorithms are essentially Locality Sensitive Hashing and share many aspects with both the p-stable LSH [6] and [3]. The goal of LSH is to remove the exponential dependency in the dimension, while still allowing sub-linear query time and good dependency on the approximation factor, c. Indeed, [3] was shown to be close to optimal with respect to the dependency on c. However, for a certain application, one usually determines c and is interested in the best dependency on the size of the dataset. This is exactly our goal and we show analytically and experimentally that we can do better for these applications. Our scheme can be thought of as a combination between [3] and [6]. The multiple random projection to a line is conceptually similar to the regular rotated and shifted grids and like [3] we use a grid in some smaller dimension t to which we project the dataset. Based on our analysis in [1], we show that one can get better dependency in the dataset size for moderate approximation factor.

The analysis above is worst case analysis. In practice, we learn the best t for random projection from a sample of the data as we do for the other two parameters, the cell size and the number of random grids. We conduct experiments that show our results in practice for image search application. Figure 2 shows numerical comparison to p-stable LSH[1] for $c = 2$ which is realistic in our application (as shown in experiments). For example, for

[1] For $c = 2$ the near optimal LSH is far slower and was omitted for better visualization

$t = \log^{2/3} n$ and $c = 2$ below is numerical comparison of the number of hash access for our proposal and the p-stable LSH where $n = |P|$ goes up to 10^6. The real query runtime (for both range search and 1-NN) of course depends also on the number of points in the dataset near the query at distance at most 1. For this reason, the query time cannot be bounded theoretically. Experimental comparison is given in Section 4.

3.2. Practical algorithms

As mentioned above the theoretical analysis covered the worst case scenario. In real applications this is not usually the case and this makes the worst case analysis less relevant. Next we describe practical ways to implement our solutions. For practical applications, we propose a more realistic scheme. More specifically, we propose to learn the best parameters from a sample of a training set. Note that this is a common strategy and the same approach is used in FLANN [12] and E2LSH [3]. We convert the approximation formulation to a randomized one. We again use (by scaling) 1 as the radius to simplify the presentation (r should be specified in advanced). Our scheme is essentially a kind of LSH and the techniques bellow are quite similar in any LSH scheme. The difference is the hashing being used. Details on theoretical aspects can be found in [9].

Randomized-NN The goal is to build a data structure that for a given query point q, if there is a point in P at distance at most 1 from q, returns one such point with pre specified probability ρ. Note that the data structure does not necessarily return the nearest neighbour, it rather returns *some* point in the given range. We impose a grid of cell size w and use m randomly shifted and rotated grids to store the points in P. All non empty cells are hashed. For a query point q, we compute the m grid cells it falls into (for each of these m grids). Then we pick from the set of points in every such cell, a random set of k points where k is a parameter. We report the first point in this set that has distance at most 1 from q and stop when such point has been found. The three parameters, w, m, k are learned from a sample of a given training set by standard optimization. For a given pre-specified probability ρ we optimize the runtime, provided that for overall sampled query points we report at least ρN near neighbors where N is the true exact near neighbors (computed using a standard kd-tree).

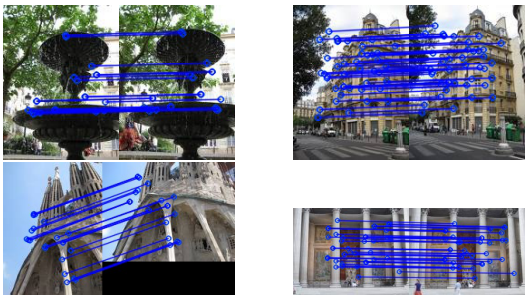


Figure 3: Samples of matched images.

Randomized Range Searching The construction is almost the same as for the Randomized-NN. We impose a grid of cell size w as before. For a given w, m, we construct m copies of the data structure. For given query point q, we rotate and shift q, m times by the corresponding transformations and report all points found in the corresponding grid cells that have distance at most 1 to q . We have to explicitly compute the distance from q for all points found in the cells and filter only the ones we need. If w, m are set correctly, with sufficiently large probability, every point at distance at most 1 from q will be found in one of the grid cells. Here, the runtime is determined by the number of "redundant" points we have to examine in all cells. In contrast to the analysis in [1] we cannot guarantee the runtime for arbitrary query. We rather want to learn the best value of w, m from the data in a learning stage. The analysis in [1] suggests that if the set P has small "intrinsic dimension" it is expected that the parameters m and w will be set such that the overall runtime will be smaller.

For learning, we have two sets of points P and Q where P is considered to be the index and Q is a set of queries (we can simply set $Q = P$). We sample a fixed fraction of the points from P and Q with uniform distribution getting P' and Q'. For a given probability ρ we seek w, m such that the overall number of reported points (when we query all points in Q' against the index built from P') is at least ρN where N is the exact number of neighbours (computed by e.g. a standard kd-tree). For all w, m satisfying this, we pick the one with the smallest runtime. For the optimization we can use a simple brute force grid scheme or we can apply any standard local optimization starting from a guess.

Application to image search The application of these tools to Image Search is straight forward. For a given set I of images, and a given distance radius

r, we first compute the local features by any appropriate method (e.g. SIFT, SURF). Each descriptor is considered as a point in high (64 in this case) dimensional space. We index all these points, the set P, in our structure (any of the above) along with the information on the image they come from. For a query image, we compute the same features set using the same method we used for the index. Let Q be the set of features that we extract from the query image. Then we query for each point in Q, all points in P that are closer than r to it. For every reported point we also record the image it comes from. For these images that have sufficient number of reporting points, we apply geometric verification by a robust (RANSAC) feature matching under perspective transformation or by finding all points consistent with the same fundamental matrix. We then report all images in I that has large support. Fig. 3 shows examples of matched images obtained with our algorithm when it is applied to a photo collection from a tourist trip to Europe (see Section 4 for more details). The matched descriptors (correspondences) are also illustrated.

Parallelization Our algorithm is straightforward to parallelize in contrast to other (e.g., tree-based) approximate nearest neighbors approaches. We propose below a simple parallelization based on the MapReduce framework. We assume that each map worker has access to the whole query set, which is typically much smaller than the training set. When the training set grows very large and does not fit into the memory of a single machine, one usually distributes the data in order to make the algorithm scalable. Hence, we assume that the training set of points is partitioned and distributed across the map workers.

- *Map phase*. Each map worker does the indexing using the partition of the data assigned to it and then queries each point from the query set using its (local) index set. The matched pairs are output to the reducer using the query point id as the key.

- *Reduce phase*. In the reduce phase all matches corresponding to the same query point will be collected in the same reduce worker. The job of the reducer is to collect together all matches of the query coming from the whole training set.

4. Experimental Results

We show results comparing our methods to ANN (Arya and Mount), FLANN (with its many algorithms including LSH) and the p-stable LSH.

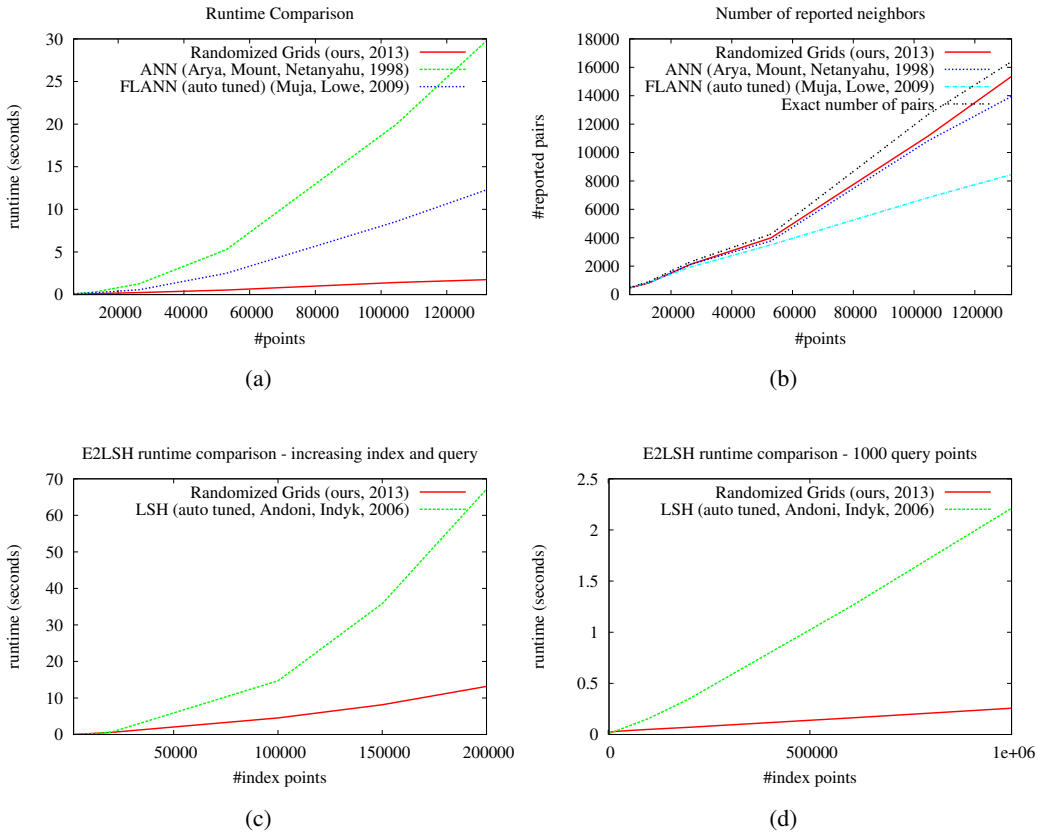
Random projection is an option for our algorithm.[2] Our comparisons to ANN and FLANN did not use random projection and the results are still good. We observed that the improvement is increasing for large size and we did use random projection in our larger scale comparison to E2LSH. For the same reason, we observed that rotating the grid for small sets does not help and we therefore did not use rotation in practice. Note that the overhead is still some constant for fixed d thus for sufficiently large n it would be better to use rotation and/or projection.

4.1. Range Searching

The first experiment is for the randomized range searching problem. The data set consists of 1M SURF descriptors (of dimension 64) extracted from a large set of 4000 images. The radius was 0.08 which was found to match the best the application of image matching. We measure the *accuracy* of the methods by comparing the total number of reported neighbors (across all query points) for each method with the exact number of neighbors (obtained using an exact method). In order to have a fair comparison, we set the parameters of the methods such that they return (about) the same number of reported neighbors. More specifically, FLANN was autotuned to select the best algorithm and parameters for this dataset (it selected kmeans as the best) with target precision set to 0.98. ANN was set with an error bound `epsilon` equal to 2.0. Imposing that index set and query set are of the same size, we randomly sample from the original data set and perform several measurements with different data sizes. The runtime comparison along with the accuracy is shown in Figs (a) and (b) respectively. Our method is much faster than both FLANN and ANN.

Further comparison to FLANN. We also experimented with FLANN using lower precision values. The index data set consists of 100K points and the query set contains 1K points. First, the precision was set to 0.8 (which is according to the literature the typical value used in vision applications).

[2]For small size, the need to project any query vector from dimension d to dimension t requires $O(dt)$ time with naive implementation. This can be further improved to $O(d \log d)$ by the FJLT [2]. We found this questionable in practice.

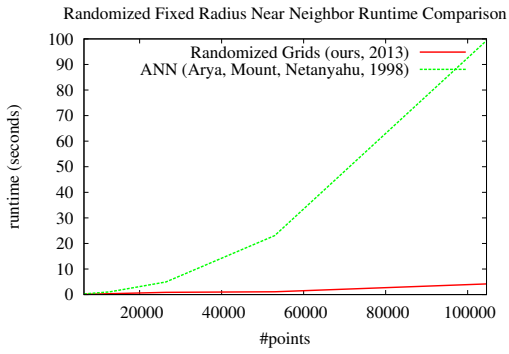


FLANN took 0.26 sec and reported 1104 neighbors, while our algorithm took 0.06 secs and reported 1112 neighbors. We also tried FLANN with 0.7 precision which resulted in 1102 reported neighbors with 0.2 secs query time. In both cases, the FLANN runtimes are still much higher than ours for a comparable number of reported neighbors. The exact number of points in the neighbourhood (for all queries) in this case was 1113.

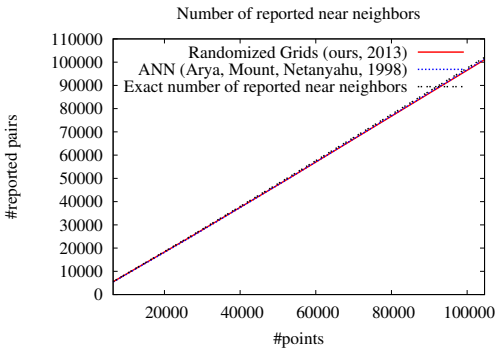
Comparison to LSH. We also compared our algorithm to the E2LSH library [3]. From the original data set we randomly sampled smaller sets of various sizes. The parameters for E2LSH were learned automatically from a sample of 4000 index points and 4000 for query points. We have used $\rho = 0.9$ as the probability of success. Taking the total number of returned points from all queries, we learned the best parameters for our algorithm, requesting the same number of returned points. The best parameters found for the random grids using this process are $w = 1.7, m = 19$ and the projection dimension was set to 14. We conducted two different tests: (i) the first uses increasing number of index and query sets *of the same size* and (ii) the second uses increasing number of index points, but fixed set of query points of size 1000. The purpose of the second experiment is to evaluate the rate of increasing running time as a function of the index in this application. Figs (c) and (d) show the results of the first and the second experiment respectively.

4.2. Randomized NN

We implemented the randomized NN algorithm introduced in Section 3.2. We looked for a practical value of r such that each query gets approximately one reported neighbor. r was found to be 0.3 for our dataset. Once r is fixed, we need to tune the parameters of the methods (for a fair comparison) such that the average total number of neighbors reported by each method is approximately ρN, where N is the exact number of neighbors (in the range r). We set the probability of success ρ to 0.9. In order to achieve that, we optimized our algorithm by selecting the parameters w (bin size), m (number of random translations) and k (number of randomly selected points from each bin) such that in the learning stage the probability of reporting a near neighbor is at least 0.9. The computed parameters for our algorithm were found to be $w = 1.4, m = 40$ and $k = 100$. For ANN, we had to set `epsilon` to



(e)



(f)

2.0. The results of the runtime comparison is shown in Fig (e). The number of returned neighbors in both algorithms in comparison to the exact number is shown in Figure (f).

4.3. Image Search Experiments

We use a small photo collection, which consists of about 3500 images from a tourist trip to Europe, where again we extract 64-dimensional SURF descriptors from each image. We perform several tests for an increasing total number of images. More specifically, for a given maximum number of images we randomly split half of the images in the training set and half of them in the query set. We index all the descriptors of the training images and then we query each descriptor of the query images using range searching. A query image is said to match a training image if the number of descriptor matches (filtered after geometric verification) is higher than 20. The descriptors match is done by different algorithms: RAND GRIDS, ANN and FLANN. We provide in Table 1 below the timings of each method for an increasing number of images used. We also report the number of matched image pairs. In order to tune the parameters of the methods for a fair comparison, we use the same process as before (i.e., using the rule that comparable number of neighbors are reported by all methods).

Large-scale image matching We run our parallel algorithm testing it on a large-scale image matching application using MapReduce. We use an image set consisting of 15M images that are collected from public Picasa albums on which we apply our MR parallel algorithm. The query set consists of 3357 images (randomly) sub-sampled from the original data set. The algorithm found 2056 matches of the 3357 query images across 14,128,635 training images. Querying all descriptors from the query set takes about 100 sec for a single partition (or 0.03s per query image). A few samples of matched image pairs are shown in Fig. 4.

5. Discussion and Conclusion

We considered the problems of nearest neighbors and range searching for the application of image matching. We focused on the (c, r)-NN problem where we were only interested in neighbors that are close enough to the query. For the range searching problem we were interested in reporting all points of distance at most r from the query. We proposed a randomized algorithm that is guaranteed to report points having pre-specified high probability.

These two solutions for the nearest neighbors and range search problems belong to the LSH family. We compared our algorithms to LSH, ANN and FLANN and showed analytically and experimentally that we can do better for moderate approximation factor. Our proposed algorithms use a learning phase. For range searching we proposed a scheme that learns the parameters adopting them to the case of a set of points with low intrinsic dimension that are embedded in high dimensional space. We learned the best t for random projection from a sample of the data as we do for the other two parameters, the cell size and the number of random grids.

Our algorithms are trivial to parallelize. In the experiments we conducted, running on couple of million images, our algorithms show meaningful speed-ups when compared with the above mentioned methods.

Num. images	ANN		FLANN		RANDGRIDS	
	Time	num. pairs	Time	num. pairs	Time	num. pairs
1000	7.48s	5	11.4s	5	1.75s	5
1500	17.55s	6	27.63s	6	2.84s	5
2000	29.95s	14	44.98s	13	3.86s	16
2500	45.93s	19	62.34s	16	5.25s	17
3000	67.45s	29	91.13s	25	5.93s	22
3500	80.27s	30	112.51s	22	7.13s	29

Table 1: Timings of the methods for image matching.

Figure 4: Samples of matched images from the large-scale experiment.

References

[1] D. Aiger, H. Kaplan, M. Sharir, Reporting Neighbors in High-Dimensional Euclidean Space, in *Proc. Annu. ACM-SIAM Sympos. Discrete Algorithms*, 2013. 2, 3, 4

[2] N. Ailon, B. Chazelle, Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform, STOC 06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pages 557-563, New York, NY, USA, 2006. ACM. 3, 5

[3] A. Andoni and P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, in *Proc. 47th Annu. IEEE Sympos. Found. Comput. Sci.*, 2006, 459–468. 2, 3, 4, 6

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *J. ACM* 45(6) (1998), 891–923. 1, 2

[5] S. Arya, T. Malamatos, and D. M. Mount, Space-time tradeoffs for approximate nearest neighbor searching, *J. ACM*, 57(1) (2009), 1:1–1:54. 1

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in *Proc. 20th Annu. Sympos. Comput. Geom.*, 2004, 253–262. 2, 3

[7] S. Har-Peled, Geometric Approximation Algorithms, Mathematical Surveys and Monographs, volume 173, AMS Press, Providence, RI, 2011. 1

[8] S. Har-Peled, M. Mendel, Fast Construction of Nets in Low-Dimensional Metrics and Their Applications. SIAM J. Comput. 35(5): 1148-1184 (2006) 2

[9] P. Indyk and R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in *Proc. 30th Annu. ACM Sypos. Theory Comput.*, 1998, 604–613. 1, 3, 4

[10] W. B. Johnson and. J. Lindenstrauss, Extensions of Lipschitz mapping into Hilbert space. Contemporary Mathematics, 26:189206, 1984. 323, 341. 3

[11] R. Krauthgamer and J. R. Lee, Navigating nets: Simple algorithms for proximity search, in *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms*, 2004, 798–807. 2

[12] M. Muja, D. G. Lowe, Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009 2, 4

[13] D. Nister, H. Stewenius, Scalable recognition with a vocabulary tree, CVPR 2006, p. 21612168. 2

[14] J. Philbin, O. Chum, M. Isard, J. Sivic, A. Zisserman, Object retrieval with large vocabularies and fast spatial matching. CVPR 2007. 2