Hindawi Publishing Corporation
ISRN Machine Vision
Volume 2013, Article ID 405680, 8 pages
http://dx.doi.org/10.1155/2013/405680

Research Article

Fast Exact Nearest Neighbour Matching in High Dimensions Using d-D Sort

Ruan Lakemond, Clinton Fookes, and Sridha Sridharan

Image and Video Research Laboratory, Queensland University of Technology, GPO Box 2434, 2 George Street, Brisbane, QLD 4001, Australia

Correspondence should be addressed to Ruan Lakemond; r.lakemond@qut.edu.au

Received 17 December 2012; Accepted 5 January 2013

Academic Editors: O. Ghita and S. Mattoccia

Copyright © 2013 Ruan Lakemond et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Data structures such as k-D trees and hierarchical k-means trees perform very well in approximate k nearest neighbour matching, but are only marginally more effective than linear search when performing exact matching in high-dimensional image descriptor data. This paper presents several improvements to linear search that allows it to outperform existing methods and recommends two approaches to exact matching. The first method reduces the number of operations by evaluating the distance measure in order of significance of the query dimensions and terminating when the partial distance exceeds the search threshold. This method does not require preprocessing and significantly outperforms existing methods. The second method improves query speed further by presorting the data using a data structure called d-D sort. The order information is used as a priority queue to reduce the time taken to find the exact match and to restrict the range of data searched. Construction of the d-D sort structure is very simple to implement, does not require any parameter tuning, and requires significantly less time than the best-performing tree structure, and data can be added to the structure relatively efficiently.

1. Introduction

The k nearest neighbour matching (kNN) problem is encountered in many applications of computer science. It is the problem of finding the k points in a database nearest to a given query point. The complexity of a simple linear search is proportional to dn, where n is the number of database entries and d is the number of data dimensions. Many attempts have been made to reduce the search time by implementing data storage and indexing structures so that the minimum number of data points has to be compared to the query point. Unfortunately, these methods are only effective in low dimensions or when using approximate nearest neighbour matching [1, 2]. Where an exact solution is required in dimensions greater than 20, linear search is only a fraction slower than the best existing search structure.

This paper proposes methods for improving the performance of linear search for the purpose of exact nearest neighbour matching using typical visual descriptors, such as the scale invariant feature transform (SIFT) [3, 4]. Many modern visual descriptors, such as gradient location and orientation histogram (GLOH) [5], are based on SIFT and have very similar characteristics from a search perspective. First, it is shown that simple modifications to the linear algorithm can allow it to outperform all existing search structures, without the need for any data preprocessing. Secondly, by presorting the data it is possible to reduce search time further. A multidimensional sorting data structure called d-D Sort is introduced that is trivially simple to construct, does not require any parameter tuning, and supports an efficient data addition operation. Thirdly, intersecting the search sphere with the data unit sphere is used to reduce the search space.

The proposed methods are compared to leading methods based on k-D trees and k-means trees [1, 2]. A set of SIFT descriptors were extracted from the MIRFLICKR-1 M image collection [6] for this purpose. Two approaches are recommended based on the results. The first requires no preprocessing and significantly outperforms leading tree structures that require a large amount of preprocessing. The second improves results further at the cost of additional memory

and preprocessing. Which method is best depends on the application.

2. Related Work

The most popular indexing and search methods are based on k-D trees [7] and hierarchical k-means trees [8, 9]. Due to the so-called curse of dimensionality [10], these methods become less effective as the data dimensionality increases. In the exact matching case, linear search is the best method for data with more than 20 dimensions. In many applications, such as where feature vocabularies are used instead of original features [11], an exact match is not essential. Many methods have been proposed to reduce computation time by finding approximate matches.

A probabilistic best-bin-first search algorithm for k-D trees was proposed in [12], where the total number of data points evaluated is fixed to limit computation time. This method does not enforce explicit bounds on the accuracy of the result. In [1] a method is proposed which ensures that, given tolerance ϵ, the distance between a query and a selected nearest neighbour is within a factor $(1 + \epsilon)$ of the distance between the query and the true nearest neighbour. The construction process does not depend on ϵ. A priority queue search is used, where the priority is determined by the distance of cells in the tree from the query point. Cells further than $\text{dist}(q, p)/(1 + \epsilon)$ are not evaluated, where q is the query and p is the current kth neighbour. These best-bin-first and approximate matching methods can also be applied to k-means trees in a very similar fashion.

While k-D trees and hierarchical k-means trees on average yield similar results, their relative performance depends on the data and type of search query. A procedure for selecting between these algorithms and automatically tuning their parameters is presented in [2]. This procedure constructs a set of sample structures with various parameters from a subset of the data. The parameters of the best performing structure are then used to build the complete data structure. The construction time can be significantly increased by the parameter selection process.

More recently, locality sensitive hashing (LSH) has become a very popular approach to approximate matching. Hashing functions are used to map high-dimensional vectors to buckets. The hashing functions are designed to maximise the likelihood that similar descriptors are mapped to the same bucket. The query point is also hashed multiple times to select target buckets. The descriptors in the target buckets are then compared to the query using the original descriptors in a post-verification stage. The choice of hashing functions needs to be made with the aid of training data that is very similar to the final application data. This method is highly effective for approximate matching, but a large number of buckets need to be selected to ensure that the exact match is found, leading to unremarkable performance in exact matching [13, 14].

This paper focuses on the exact matching case, where the true nearest neighbour is required. Systems based on k-D trees and k-means trees support guaranteed exact search, but at best outperform linear search by only a fraction. Instead of modifying these structures, this paper investigates methods for improving linear search and shows that it can outperform the above methods.

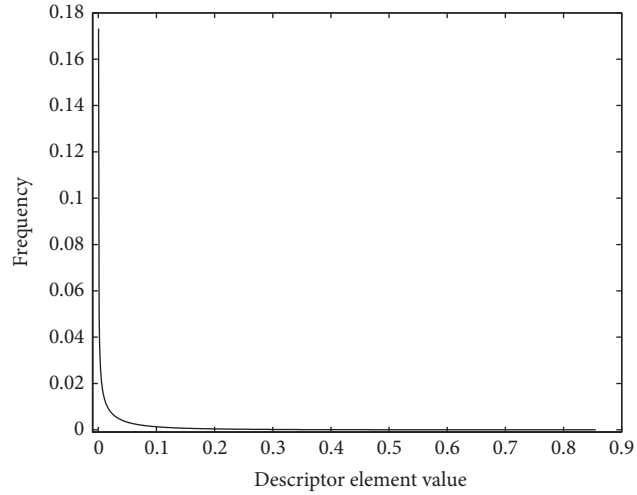


Figure 1: Distribution of descriptor element values for a 128-dimensional SIFT descriptor computed from a large set of MSER features.

3. Improved Linear Search

The objective of visual descriptor extraction is usually to capture some structure. Consequently, the elements of the resulting descriptor vector are not uniformly distributed, but exhibit some structure as well. Additionally, these feature vectors are frequently normalised to have unit length and only have positive values. Figure 1 shows the distribution of descriptor element values for a 128-dimensional SIFT descriptor computed from a large set of maximally stable extremal region (MSER) features [15], extracted from a natural scene. It can be seen that this distribution is approximately exponential. For an evenly distributed unit vector descriptor, one would expect the median value to be near $d^{-1/2}$; however, only 15% of the elements of the SIFT descriptor are above this value. It can therefore be said that most descriptors consist of a small set of relatively large values and a majority of small values (and often many zeros). Since the larger values of any descriptor are both rare and responsible for the majority of the distance between unit vector descriptors, it is possible to evaluate a large component of the distance by considering only a small number of dimensions.

3.1. Partial Distance Evaluation. While the exact distance between a query descriptor and the nearest k descriptors is usually of interest, the distance to other descriptors is not, except to determine that all other descriptors are more distant than the kth descriptor. Evaluation of the distance measure can therefore be aborted once it exceeds the distance to the current best candidate for descriptor k (used as a threshold). In the case of normalised vectors, the number of evaluated dimensions can be further reduced by evaluating the distance measure in order of the significance of the

query dimensions. This does not generalise to unnormalised, uniformly distributed data. While partial distances have been used before [16], the computation in order of significance is novel. All that is required for ordered partial distance computation is to find the order of significance of the query dimensions. This can be done using a sorting operation in $O(d\log(d))$ time.

The effectiveness of this method depends on how long it takes the search algorithm to encounter the nearest neighbour. At the start of the search, a relatively large search threshold is used (tight search thresholds have been shown to be ineffective [4]), resulting in complete distance computation for at least the first descriptor. As descriptors closer to the query are encountered, the search threshold is decreased and the partial evaluation becomes more effective.

Implementing the above improvements is trivial and requires only the following three modifications.

(1) Sort the query dimensions according to significance.

(2) Modify the distance function so that the distance measure is computed in order of query dimension significance.

(3) The distance function should terminate as soon as the partially computed distance exceeds a threshold.

3.2. Searching on Sorted Arrays. Presorting the data according to one of the dimensions can be used to improve performance in two ways. Firstly, the sorted dimension can be used as a search priority queue. Bilinear search can be used to find a starting point that is close to the query point in at least one dimension, from where the search progresses outwards. If the sorted dimension is one of the more significant query dimensions, then the linear search is more likely to encounter the nearest neighbour early in the search, thereby improving the effectiveness of partial distance evaluation. The inverse is true if selecting a dimension that is one of the least significant query dimensions (this is demonstrated experimentally in Section 5). The best results are achieved by sorting the data on every dimension and by selecting the order associated with the query's dominant dimension.

Secondly, the sorted dimension can be used to restrict the search range. A naive approach to defining the search limits would be to simply search in the range $[q_j - r, q_j + r]$, where $\mathbf{q}$ is the query, r is the current search threshold (distance to kth nearest neighbour), and j is the query's dominant dimension. This is equivalent to stopping the search when the ordered partial distance exceeds the threshold on the first dimension. Because the data is normalised to unit vectors, much tighter constraints are possible. The search limits on q_j are set to the minimum and maximum values that satisfy the intersection of the data unit hypersphere and the search radius hypersphere.

The data space can be defined as

$$\begin{aligned} &\mathbf{x} \in \mathbb{R}^d, \\ &x_i = \mathbf{x}\{i\} \in [0,1], \\ &\mathbf{x}^\top \mathbf{x} = 1. \end{aligned} \tag{1}$$

The boundary, $\mathbf{b} \in \mathbf{x}$, of the search space around query point $\mathbf{q} \in \mathbf{x}$ additionally satisfies the constraint:

$$\begin{aligned} (\mathbf{b}-\mathbf{q})^\top (\mathbf{b}-\mathbf{q}) &= r^2, \\ \therefore \mathbf{b}^\top \mathbf{q} &= 1 - \frac{r^2}{2}. \end{aligned} \tag{2}$$

Let $\mathbf{b} = \mathbf{p} + \mathbf{s}$, where $\mathbf{p} = \mathbf{q}\sqrt{1 - r^2/2}$ is the component of $\mathbf{b}$ parallel to $\mathbf{q}$ and $\mathbf{s}$ is the orthogonal component such that $\mathbf{s}^\top \mathbf{q} = \mathbf{s}^\top \mathbf{p} = 0$. The length of $\mathbf{s}$ is found as

$$\begin{aligned} \mathbf{b}^\top \mathbf{b} &= 1 \\ (\mathbf{s}+\mathbf{p})^\top (\mathbf{s}+\mathbf{p}) &= 1, \\ \mathbf{s}^\top \mathbf{s} + 2\mathbf{s}^\top \mathbf{p} + \mathbf{p}^\top \mathbf{p} &= 1, \\ \mathbf{s}^\top \mathbf{s} &= \frac{r^2}{2}. \end{aligned} \tag{3}$$

In order to maximise b_j, choose $\mathbf{s}$ so that $|b_i|$ is minimised for all $i \neq j$, that is, $s_i \propto -q_i$. Add scale factor a to correct the length and select s_j to satisfy the orthogonality constraint:

$$\begin{aligned} s_i &= -aq_i, \quad \forall i \neq j, \\ \mathbf{s}^\top \mathbf{q} &= s_j q_j - a \sum_{\forall i \neq j} q_i^2 = 0, \\ \therefore s_j &= a \frac{\sum_{\forall i \neq j} q_i^2}{q_j}. \end{aligned} \tag{4}$$

Let $c = \sum_{\forall i \neq j} q_i^2$ and solve for a using the known length of $\mathbf{s}$:

$$\begin{aligned} s_j &= a\frac{c}{q_j}, \\ \mathbf{s}^\top \mathbf{s} &= s_j^2 + \sum_{\forall i \neq j} s_i^2, \\ \frac{r^2}{2} &= a^2 \frac{c^2}{q_j^2} + a^2 c, \\ a^2 &= \frac{r^2}{2\left(c^2/q_j^2 + c\right)}, \\ \therefore s_j &= \frac{rc}{q_j\sqrt{2\left(c^2/q_j^2 + c\right)}} = rg. \end{aligned} \tag{5}$$

Similarly, b_j can be minimised, resulting in $s_j = -rg$. Note that g depends only on q and can be computed once per query.

The above equations only find $\mathbf{b}$ with minimum or maximum value of b_j if the search area is not bounded by one of the coordinate axes. If the j-axis intersects the search area, then the maximum value of b_j is 1, and if any other axis intersects the search area, then the minimum value of b_j is 0. Let r_u be the distance between $\mathbf{q}$ and the unit vector along dimension j, and let r_l be the distance between $\mathbf{q}$ and the unit

vector along the minimum dimension of $\mathbf{q}$. The upper and lower search limits along dimension j are then,

$$t_u = \begin{cases} q_j\sqrt{1-\dfrac{r^2}{2}} + rg, & r < r_u, \\ 1, & r \geq r_u, \end{cases}$$
$$t_l = \begin{cases} q_j\sqrt{1-\dfrac{r^2}{2}} - rg, & r < r_l, \\ 0, & r \geq r_l. \end{cases} \quad (6)$$

4. The d-D Sort System

A search structure named d-D sort is used to implement the sorted linear search improvements presented in the previous section.

4.1. Construction. The d-D Sort structure consist of d index arrays, $o = \{o_1, o_2, \ldots, o_d\}$, with each $o_i = \{o_{i1}, o_{i2}, \ldots, o_{in}\}$ indexing the descriptor data, $\mathbf{x}$, in order of dimension i. It requires nd memory in addition to the data. Construction is extremely simple; sort the descriptor data according to each dimension and write the result in o (instead of actually reordering the data), requiring $O(dn\log(n))$ time.

4.2. Adding Data. Adding additional data to the d-D Sort structure is relatively simple as well. The new data, $\mathbf{x}'$, is appended to the existing data, $\mathbf{x}'' = \{\mathbf{x}, \mathbf{x}'\}$. The ordering arrays are updated by finding the position of the new data. This can be achieved efficiently using binary search in $O(d\log(n))$ time and array insertion operations in linear time. For a large batch of new descriptors, a more efficient solution is to sort the new data first ($O(dn'\log n')$ time) and then merge the result with the existing order arrays ($O(d(n+n'))$ time), rather than performing multiple separate insertions. This addition operation is faster than a complete reconstruction where the new data contains fewer descriptors than the existing data.

4.3. Queries. All matching queries are variants of a k nearest neighbours (kNNs) query with a maximum distance threshold. The algorithm is listed in Algorithm 1.

Other types of queries are essentially specialisations of the above algorithm. KNN without a threshold is achieved by setting r to a very large initial value. Nearest neighbour ratio matching [4] is equivalent to 1NN matching with the additional requirement that no second match is found in the radius $(1+\delta)e_r\{1\}$. In this case r is updated as $r \leftarrow (1+\delta)e$ in line (1.16), and the final match is invalidated if any second nearest neighbour is found to be within r.

Two possible methods can be applied to further speed up the query process while sacrificing exact results. Firstly, the number of descriptors visited can be limited. It is possible to compute the worst case accuracy of this approach for each individual query, but it is not possible to enforce a lower accuracy bound on all queries. Alternatively, the distance threshold can be reduced according to an accuracy requirement, to reduce the number of descriptors visited. The convention used in [1] for describing approximation tolerance is used: given tolerance ϵ, the distance between a query and a selected nearest neighbour must be within a factor $(1+\epsilon)$ of the distance between the query and the true nearest neighbour. This is implemented in the d-D Sort query algorithm in line (1.16). As is shown in the experimental evaluation, this method can reduce search time, but the search accuracy degrades more rapidly than with other search structures.

The computation time of the query operation is highly dependant on the data distribution. Sorting the query requires $O(d\log(d))$ operations and the initial binary search requires $O(\log(n))$ operations. Searching for k exact matches requires at best $\Omega(kd)$ operations and at worst $O(nd)$ operations (requiring an invalid query $\mathbf{q} = \mathbf{0}$ or data spherically symmetric around $\mathbf{q}$). In practice, the total number of data points visited is much smaller than n, and much fewer than d dimensions are evaluated for most visited points.

5. Experimental Evaluation

The following eight search methods were compared experimentally:

(1) linear search,

(2) linear search using partial distance computation,

(3) linear search using significance ordered partial distance computation,

(4) d-D Sort structured search,

(5) 1-D Sort search,

(6) the k-D tree from the approximate nearest neighbor (ANN) library [1],

(7) the b-D tree from the approximate nearest neighbor (ANN) library [1],

(8) the fast library for approximate nearest neighbors (FLANN) [2].

Publicly available C/C++ implementations by the original authors were used for ANN and FLANN. The FLANN method was set to use 1000 points in its an automatic parameter selection process. The 1-D Sort method sorts the data on the first dimension only. This method is added to the evaluation to demonstrate the importance of using the order of the most significant query dimension as search priority queue.

A set of data was generated using a characteristic scale determinant of Hessian feature detector [17, 18] and SIFT descriptor [3, 4] to extract 1 million descriptors from a subset of the MIRFLICKR-1 M image collection [6]. These descriptors are normalised to have an L_2 norm of 1 and consists of strictly positive values.

Figure 2 plots construction times and average query time results for a set of queries produced by extracting features for an image not included in the dataset. This is the most difficult test case, since the queries are rarely very close to any points in the dataset. Query time results are expressed

Input: $\mathbf{x}$, o, $\mathbf{q}$, k, r, ϵ
Output: n_r, i_r, e_r
(1.1) **begin**
(1.2) Sort query, $\mathbf{q}$ in ascending order, yielding order array o_q.
(1.3) Select prime dimension, $j \leftarrow o_q\{d\}$.
(1.4) Select data order vector $o_j = o\{j\}$ as the search order.
(1.5) Compute r_u and r_l.
(1.6) Compute g according to (5).
(1.7) Compute initial search range, $[t_u, t_l]$, according to (6).
(1.8) Use binary search to find $i_0 \leftarrow \arg\min_i \left|\mathbf{x}\{o_j\{i\}\}\{j\} - q_j\right|$.
(1.9) **foreach** $i \in [1, n]$ in order of increasing distance from i_0 **do**
(1.10) If $\mathbf{x}\{o_j\{i\}\}\{j\}$ is out of search range $[t_u, t_l]$, terminate.
(1.11) Compute distance $e = \text{dist}\left(\mathbf{x}\left\{o_j\{i\}\right\}, \mathbf{q}\right)$ using the ordered partial distance method.
(1.12) **if** $e < r$ **then**
(1.13) Insert i, e in results list, i_r, e_r.
(1.14) Increment n_r (up to k).
(1.15) **if** $n_r = k$ **then**
(1.16) Reduce the search range, $r \leftarrow e_k/(1 + \epsilon)$.
(1.17) Recompute $[t_u, t_l]$ according to (6).
(1.18) **end**
(1.19) **end**
(1.20) **end**
(1.21) **end**

Algorithm 1: kNN Query Algorithm for the d-D Sort Structure.

Table 1: 1NN query performance using a 128000-point dataset. Each column shows results for query descriptors extracted from an image related to the data in a different way. Results are presented as the ratio between linear search time and the given method's search time.

Method	Not in database	Rotated copy	Exact copy
ANN b-D tree	0.297	0.292	0.2872
ANN k-D tree	0.503	0.498	0.4909
FLANN	1.450	2.483	23.13
Linear	1.000	1.000	1.000
Linear partial	1.392	2.124	12.62
Linear ordered partial	2.588	4.397	12.62
d-D sort	3.404	7.032	2246.0

in terms of a relative increase in speed over the time taken to do a simple linear search ($t_{\text{linear}}/t_{\text{test}}$). Table 1 lists 1NN performance for queries from an image that is not in the database (same as Figure 2(b)), a rotated version of an image that is in the database, and an exact copy of an image in the database.

Data structure construction times are plotted in Figure 2(a) for those methods that require preprocessing. The d-D Sort structure construction time is comparable to that of the b-D tree on average, while the construction time increases more slowly than the k-D Tree. FLANN takes the longest to construct due to the simulation process it uses to select parameters. Figure 3 shows the time taken to append 100 descriptors to a d-D Sort structure of increasing size. It can be seen that the addition method is more efficient when adding a relatively small number of points to the database, compared to rebuilding the structure completely.

Figure 2(b) plots the relative exact 1NN query performance against the size of the database. A matching threshold of 2.0 was used. It can be seen that using only partial distance computation is sufficient to bring the speed of linear search up to that of the best tree structure method (FLANN). Ordered partial distance adds a further improvement, all without any data preprocessing. The d-D Sort structure achieves the best performance by a significant margin, demonstrating the benefits of presorting the data. In contrast, 1-D Sort performs worse than the ordered partial distance method. This shows that sorting on an arbitrary dimension is not beneficial for all queries and that it is necessary to sort on the query's most significant dimension.

The effect of matching threshold on query performance is examined in Figure 2(c). Smaller thresholds reduce the search space and improve matching speed in general. Of the linear methods, the d-D Sort structure is best able to take advantage of smaller thresholds. The k-D Tree and b-D Tree structures benefit the most from smaller thresholds, with k-D Tree surpassing d-D Sort at a relatively small threshold of 0.2.

The number of nearest neighbours selected impacts the partial distance computation, since the greater distance to the kth match leads to a greater search threshold. Figure 2(d) plots the query performance against k. A decrease in performance can be seen for d-D Sort and modified linear methods, while the ANN and FLANN methods are unaffected. The linear methods remain the most efficient.

The d-D Sort structure supports approximate matching by reducing the search threshold below what is needed to

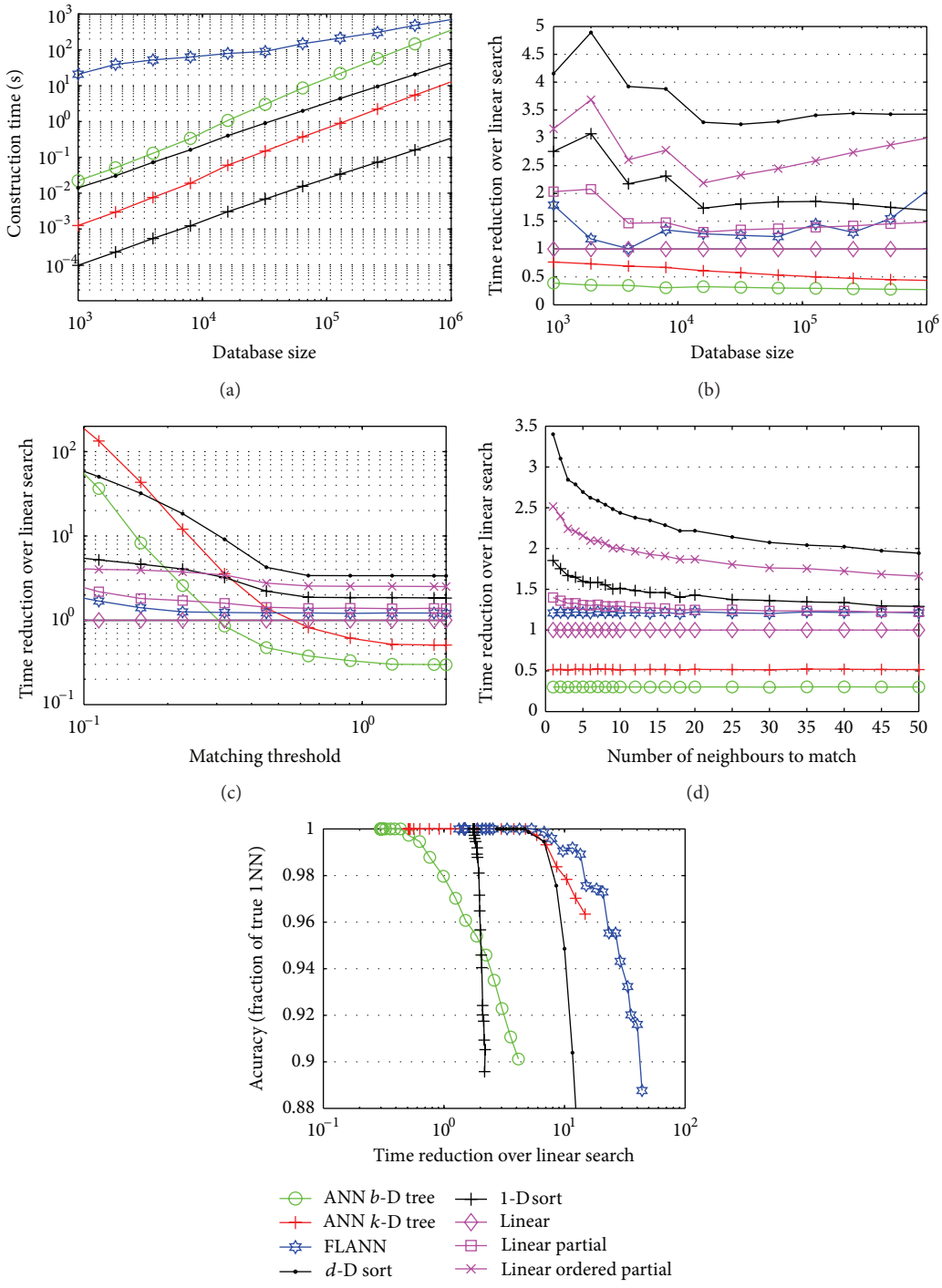


FIGURE 2: Results for SIFT descriptor data experiment using euclidean distance. (a) Search structure construction time plots against database size. (b) Exact 1NN query time, normalised according to linear search time, plots against database size. A threshold of 2.0 was used. (c) Exact 1NN query time, normalised according to linear search time, plots against threshold for 100 k descriptor database. (d) Exact query time, normalised according to linear search time, plots against the number of neighbours to find. A 100 k descriptor database and threshold of 2.0 was used. (e) Accuracy of approximate 1NN query plots against query time, normalised according to linear search time. A 100 k descriptor database and threshold of 2.0 were used.

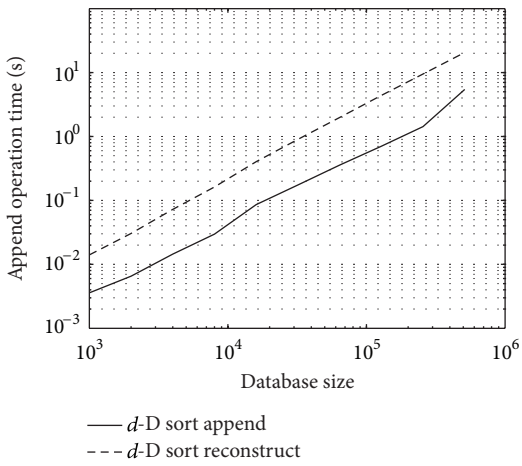


Figure 3: The time required to append 100 SIFT descriptors to a d-D sort structure plot against the number of descriptors already in the structure. The time taken to construct the database anew is included for reference.

prove an exact match. Figure 2(b) plots the match accuracy against the relative query performance for approximate nearest neighbour matching. Match accuracy is measured as the proportion of matches returned that are the true matches. It can be seen that approximate matching leads to more than an order of magnitude improvement in performance at the cost of a slight reduction in accuracy. The accuracy of the d-D Sort structure drops more quickly than that of ANN k-D tree or FLANN. While d-D Sort consistently outperforms the other methods in guaranteed exact matching, it is not the best choice for approximate matching.

The effect of the similarity between queries and the data is demonstrated by Table 1. The first column shows results from an image not in the database, which is the same as one of the points in Figure 2(b). The second column lists results of matching a rotated version of an image in the database and the third column is for an exact copy of a database image. The absolute time performance of linear search is the same for all three cases. The ANN methods perform the same for all three cases as well, which is unexpected. FLANN shows a significant increase in performance. This is the expected behaviour, since the closer true match allows for reduced search space. Both modified linear methods show an improvement in performance, though the improvement is less pronounced than with FLANN. In the exact copy case the modified linear methods practically become equivalent, since any nonzero dimension of the query is sufficient to exclude a potential match after the true nearest neighbour has been found. The d-D Sort method shows the best improvement in performance, with performance more than doubling between the outlier case and the rotated image case. In the exact case, the initial bilinear search always delivers the exact match, resulting in $\log(n)$ search time and extremely fast performance. Unfortunately the exact match case does not have many applications. This last result really shows not only that exact matching is possible using a one dimensional feature, but also that d-D Sort is able to take advantage of close similarity.

6. Conclusion

Data structures such as k-D trees and hierarchical k-means trees are only marginally more effective than linear search when performing exact k nearest neighbour matching in high-dimensional local image descriptor data. Of these, the best-performing method is the FLANN approach [2], which improves performance by a factor of 2.5 at best, but typically yields an improvement of less than 1.5-fold. At the same time, this data structure requires more than a minute to construct for even small datasets.

This paper presents several performance improvements to the linear search method for exact k nearest neighbour matching. It is shown that evaluating the distance measure in order of the significance of query dimensions and terminating when the search threshold is reached can improve linear search time by 1.7–4.4-fold (usually at least 2.6 fold). These modifications are simple to implement and do not require any data preprocessing. Secondly, the d-D sort structure is introduced. This structure essentially presorts the data according to every dimension. No parameter tuning is required and data can be added efficiently. Using the sort order associated with the query's most significant dimension as a priority queue and to limit the search range improves results further. The d-D Sort-based search showed an improvement over linear search of 2–7-fold (usually at least 3.2-fold), while the preprocessing time can be several orders of magnitude less than that of FLANN. While it is possible to implement approximate nearest neighbour search using the d-D Sort structure, results show that the accuracy of this method decreases more rapidly than that of ANN and FLANN and does not yield the same performance-accuracy ratio.

In summary, this paper proposes two approaches for exact nearest neighbour search in normalised high-dimensional descriptor data. The first is the use of partial distance computation in order of significance of the query dimensions. This does not require any data preprocessing and yields best results when preprocessing time would be a significant factor, for example, when matching between a small set of images. The second approach is to use the d-D sort structure and proposed query mechanism, which yield the best query time performance without the need for any parameter tuning. This yields the best performance where the preprocessing time is small compared to the number of queries that will be performed. The d-D Sort structure also supports data addition for problems where the matching database grows progressively.

Acknowledgment

This project was supported by Australian Research Council Grant no. LP0990135.

References

[1] 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," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[2] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP'09)*, pp. 331–340, February 2009.

[3] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV'99)*, vol. 2, pp. 1150–1157, Kerkyra, Greece, September 1999.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[6] M. J. Huiskes and M. S. Lew, "The MIR Flickr retrieval evaluation," in *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval (MIR'08)*, pp. 39–43, August 2008.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[8] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Transactions on Computers*, vol. 24, no. 7, pp. 750–753, 1975.

[9] S. Brin, "Near neighbor search in large metric spaces.," in *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB'95)*, pp. 574–584, 1995.

[10] M. Houle, H. Kriegel, P. Kröger, E. Schubert, and A. Zimek, *Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?*vol. 6187 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2010.

[11] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2161–2168, June 2006.

[12] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1000–1006, June 1997.

[13] M. Aly, P. Welinder, M. Munich, and P. Perona, "Scaling object recognition: benchmark of current state of the art techniques," in *Proceedings of the 12th IEEE International Conference on Computer Vision Workshops (ICCV Workshops'09)*, pp. 2117–2124, Kyoto, Japan, October 2009.

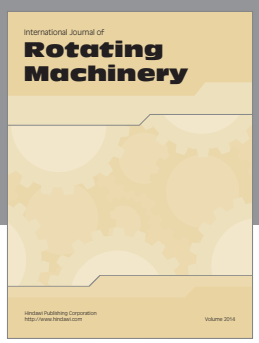
[14] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: a comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

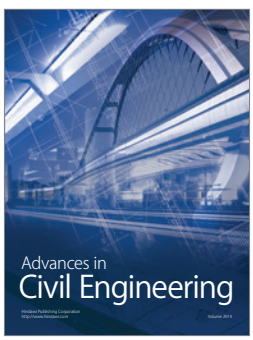
[15] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *Proceedings of the British Machine Vision Conference*, vol. 1, pp. 384–393, 2002.

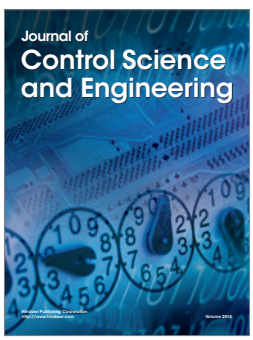
[16] C. D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Transactions on Communications*, vol. 33, no. 10, pp. 1132–1133, 1985.

[17] R. Lakemond, C. Fookes, and S. Sridharan, "Negative determinant of hessian features," in *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications*, pp. 530–535, Queensland, Australia, December 2011.

[18] R. Lakemond, D. N. R. McKinnon, C. Fookes, and S. Sridharan, "A feature clustering algorithm for scale-space analysis of image structures," in *Proceedings of the International Conference on Signal Processing and Communication Systems*, pp. 186–192, Gold Coast, Australia, December 2007.







Hindawi
Submit your manuscripts at
http://www.hindawi.com