UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Graph-based agglomerative clustering

Pasi Fränti, Olli Virmajoki and Ville Hautamäki

Report A-2003-4

ACM I.5.3
ISSN 0789-7316
ISBN 952-458-390-0

Graph-based agglomerative clustering

Pasi Fränti, Olli Virmajoki and Ville Hautamäki

Department of Computer Science
University of Joensuu
P.O. Box 111, FIN-80101 Joensuu
FINLAND
Email: {franti,ovirma,villeh}@cs.joensuu.fi

Abstract: The search for nearest neighbor is the main source of computation in most clustering algorithms. A common solution is to calculate distance to all candidates (full search) and select the one with the smallest distance. In this paper, we propose the use of nearest neighbor graph for reducing the number of candidates to be considered. The number of distance calculations per search can be reduced from O(N) to O(k) where N is the number of clusters, and k is the number of neighbors in the graph. We apply the proposed scheme within agglomerative clustering algorithm known as the PNN algorithm, and show that remarkable reduction is obtained in the running time at the cost of slight increase in the distortion.

Keywords: Clustering, agglomeration, vector quantization, codebook generation, nearest neighbor graph, PNN.

Statistics: 28 pages, 13 figures, 8 tables, 6652 words, 32263 characters.

1. Introduction

Clustering is an important problem that must often be solved as a part of more complicated tasks in pattern recognition, image analysis and other fields of science and engineering [1, 2, 3, 4]. The clustering task is formalized here as a combinatorial optimization problem, in which the goal is to find the partition that minimizes a given distortion function.

Agglomerative clustering is popular method for generating the clustering hierarchically by a sequence of merge operations. The clustering starts by initializing each data vector as its own cluster. Two clusters are merged at each step and the process is repeated until the desired number of clusters is obtained. *Ward's method* [5] selects the cluster pair to be merged that minimizes the increase in the distortion function value. In the vector quantization context, this method is known as the *pairwise nearest neighbor* (*PNN*) method due to [6]. In the rest of this paper, we denote it as the *PNN method*.

The *PNN* is interesting method for the clustering task because of its conceptual simplicity and good results [7]. The *PNN* can also be combined with the *K-means* clustering such as the *Generalized Lloyd algorithm* (GLA) [8] as proposed in [9], or used as a component in more sophisticated optimization methods. For example, the *PNN* has been used in the merge phase in the *split-and-merge* algorithm [10] resulting in to a good time-distortion performance, and as the crossover method in *genetic algorithm* [11], which has turned out to be the best clustering method among a wide variety of algorithms in terms of the quality of the codebook [12].

The main drawback of the PNN is its slowness. The original implementation requires O(N^3) distance calculations [13]. An order of magnitude faster algorithm has been introduced in [7]

but the method is still lower bounded by $\Omega(N^2)$. The main source of computation originates from the search of the nearest neighbor cluster because the PNN always calculates distance to all candidates when finding the nearest cluster.

Another approach for clustering is to use graph theoretical methods [14]. For example, by first creating a complete undirected graph where the nodes correspond to the data vectors and the edges correspond to vector distances according to a given *similarity* or *dissimilarity* measure. The resulting graph can be trimmed to a *minimal spanning tree*, which can be interpreted as one large cluster. The clustering can then be generated by iteratively dividing the cluster by removing longest edges from the graph. In the final graph, clusters can be determined by finding the separate components in the graph. This algorithm can be seen as a variation of split-based methods with similar criterion as in the *single-linkage* agglomerative clustering.

In this paper, we introduce fast agglomerative clustering algorithm motivated by the graph-based approaches. In our approach, we process the data at the cluster level so that every node in the graph represents a cluster and not as a single vector as in the previous approaches. The edges of the graph represent inter cluster connections between nearby clusters. The graph is used merely as a search structure for reducing the number of distance calculations. We can therefore choose the distortion function freely instead of the previous approach, which can be interpreted as a heuristic implementation of the classical single-linkage clustering.

The proposed approach has two specific problems to solve: how to generate the graph efficiently, and how to utilize it. For example, standard solutions for solving minimum spanning tree takes $O(N^2)$ time, which would prevent any speed-up. We propose solutions for the first problem by considering the *mean-distance ordered partial search* [15], and by *divide-and-conquer* approach [16]. We study the second sub-problem and find out how much speed-up can be gained by using the graph for reducing the number of calculations in the PNN. We will show by experiments that a relatively small neighborhood size is sufficient for preserving the good quality clustering results. It is also possible that the idea could be generalized to other clustering algorithms that include large number of nearest neighbor searches.

The rest of the paper is organized as follows. In Section 2, we define the clustering problem considered here, and recall the *PNN* method. In section 3, we propose the new graph-based PNN algorithm. Two solutions for creating the nearest neighbor graph are introduced in Section 4. Experimental results are reported in Section 5, and conclusions drawn in Section 6.

2. Agglomerative clustering

The *clustering problem* is defined here as a combinatorial optimization problem. Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$, partition the data set into M clusters so that a given distortion function is minimized. Partition $P=\{p_1, p_2, \ldots, p_N\}$ defines the clustering by giving for each data vector the index of the cluster where it is assigned to. *A cluster* s_a is defined as the set of data vectors that belong to the same partition a.

$$s_a = \left\{ x_i \middle| p_i = a \right\} \tag{1}$$

The clustering is then represented as the set $S=\{s_1, s_2, ..., s_M\}$. In vector quantization, the output of the clustering is a codebook $C=\{c_1, c_2, \ldots, c_M\}$, which is usually the set of cluster

centroids. We assume that the vectors belong to Euclidean space, and use the mean square error (*MSE*) as the distortion function:

$$MSE(C,P) = \frac{1}{N} \cdot \sum_{i=1}^{N} \left\| x_i - c_{p_i} \right\|^2 \quad (2)$$

The *pairwise nearest neighbor* (*PNN*) method [5, 6] generates the clustering hierarchically using a sequence of merge operations. At each step two nearby clusters are merged:

$$s_a \leftarrow s_a \cup s_b \quad (3)$$

The cost of merging two clusters s_a and s_b is the increase in the *MSE*-value caused by the merge. It can be calculated using the following formula [5, 6]:

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \left\| c_a - c_b \right\|^2 \quad (4)$$

where n_a and n_b are the corresponding cluster sizes. The *PNN* applies local optimization strategy: all possible cluster pairs are considered and the one increasing *MSE* least is chosen:

$$a,b = \arg \min_{\substack{i,j \in [1,m] \\ i \neq j}} d_{i,j} \quad (5)$$

where m is the current number of clusters. There exist many variants of the *PNN* method. Straightforward implementation recalculates all distances at each step of the algorithm. This takes $O(N^3)$ time because there are $O(N)$ steps in total, and $O(N^2)$ cluster pairs to be checked at each step.

Another approach is to maintain an $N \times N$ matrix of the merge cost values. The merge cost values are needed to be updated only for the newly merged cluster. Nevertheless, the algorithm still requires $O(N^3)$ because the search of the minimum cluster pair takes $O(N^2)$ time [13]. Kurita's method maintains an $N \times N$ matrix but it also utilizes a heap structure for searching the minimum distance [17]. The method runs in $O(N^2 \log N)$ time. The storage of the matrix, however, requires $O(N^2)$ memory, which makes these variants impractical for large data sets.

A fast implementation of the PNN with linear memory consumption has been obtained by maintaining a pointer from each cluster to its nearest neighbor, and the corresponding merge cost value [7]. The cluster pair to be merged can be found in $O(N)$ time, and only a small number (denoted by τ) of the nearest neighbor needs to be updated after each merge. The implementation takes $O(\tau N^2)$ time in total. Further speed-up can be achieved by using lazy update of the merge cost values [18], and by reducing the amount of work caused by the distance calculations [19].

3. Graph-based PNN

The proposed Graph-PNN is based on the exact PNN method, but we utilize the graph structure in the search of nearest neighbor clusters. In the PNN, the search of nearest neighbor cluster is repeated many times, and every search requires $O(N)$ distance calculations. The graph is utilized so that the search is limited only to the clusters that are directly connected by the graph structure. This reduces the time complexity of every search from $O(N)$ to $O(k)$. If the number of edges k is small, significant speed-up is obtained.

3.1 Simple implementation

The main structure of the algorithm is given in Fig. 1. The algorithm starts by initializing every data vector as its own clusters, and by constructing the neighborhood graph. The algorithm then iterates by removing nodes from the graph until the desired number of clusters has been reached. The graph stores the merge costs, i.e. the amount of distortion if the two neighbor clusters are merged. The edge cost values are stored in a heap structure.

At first, the edge with smallest weight is found, and the nodes (s_a and s_b) are merged. The algorithm creates a new node s_{ab} from the clusters s_a and s_b, which are removed from the graph. The corresponding edges are updated by calculating new cost values between the nodes that were connected to the merged nodes. The algorithm must also calculate cost values for the outgoing edges from the newly created node s_{ab}. The k nearest neighbors is found among the $2k$ neighbors of the previously merged nodes s_a and s_b.

We illustrate the merge procedure in Fig. 2 for a sample 2NN graph (k=2). The nearest neighbors for the merged cluster is found among the neighbors of a and b: the clusters c and e. We also update the links that pointed to a or b to point to the new cluster and update the associated cost values. In practice, the new cluster replaces a, and b is removed. The pointers $c \rightarrow b$ and $d \rightarrow b$ are replaced by pointers $c \rightarrow a$ and $d \rightarrow a$. A sample graph is shown in Fig. 3.

```
GraphPNN(X, M) → S
    FOR i←1 to N DO
        s_i ← {x_i};
    FOR ∀(s_i; i ∈ [1, N]) DO
        Find k nearest neighbors;
    REPEAT
        (s_a, s_b) ← SearchNearestClustersInGraph(S);
        s_ab ← Merge(s_a, s_b);
        Find the k nearest neighbors for s_ab;
        Update the nodes that had s_a and s_b as
        neighbors;
    UNTIL |S|=M;
```

Fig. 1. Structure of the Graph-PNN.

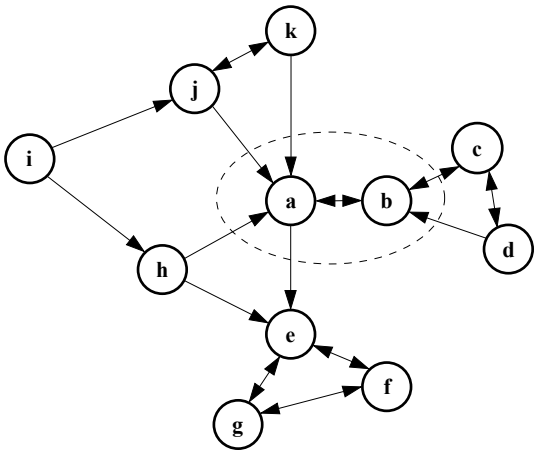


Fig. 2. Illustration of the graph (k=2) where a and b are to be merged.

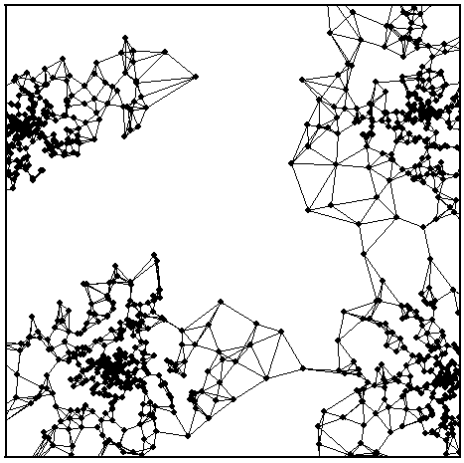


Fig. 3. Sample data set (black dots) and the corresponding graph (k=4).

3.3 Double linked list

To sum up, one step requires O(kN) time, which sums up to O(τN^2). The number τ denotes the number of the incoming pointers to the merged cluster. In general, this is too much and we therefore consider the double linked list (Fig. 4), in which we maintain for every node two lists: the first list points to the neighbor clusters, and the second list contains so called "back pointers" to clusters that have the node as their nearest neighbors. In this way, we can eliminate O(kN) time loops, and the time complexity becomes O($\tau N \log N / k$), see Tables 1, 2, 3 and 4.

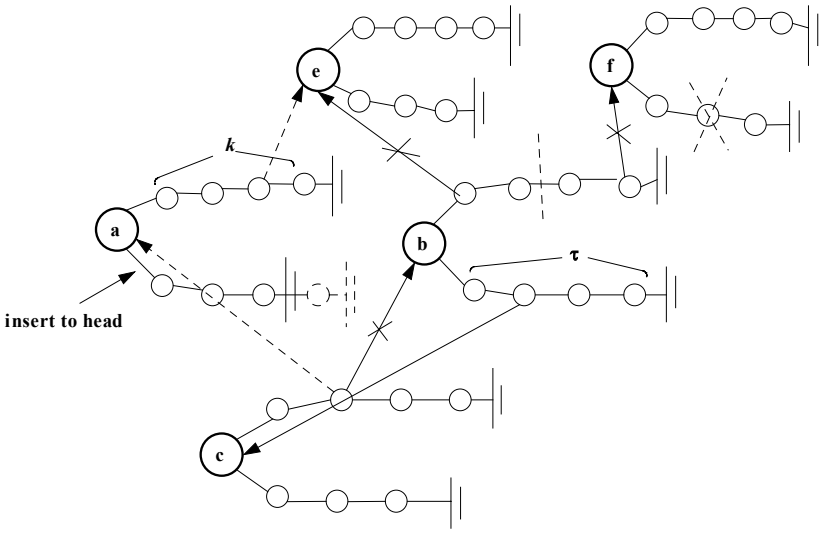


Fig. 4. Illustration of the update of the linked list in the merge procedure of the clusters a and b in the neighborhood graph.

Table 1: Estimated number of steps and distance calculations of the PNN iterations.

Steps:	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Dists	Steps	Dists
SearchNearest( )	N	-	1	-	1	-
Merge(a, b)	N	-	$2k^2 + \log N$	$2k$	$2k^2 + \tau k + \log N$	$2k$
FindNeighbors(a, b)	N	-	kN	-	τk	-
RemoveLast( )	N	-	$k + 2\log N$	-	$\log N$	-
UpdateDistances( )	$N(1+\tau)$	τN	$kN + \tau \log N / k$	τ	$\tau k + \tau \log N / k$	τ

Table 2: Observed number of the steps and distance calculations of the PNN iterations for *Bridge* ($k = 3$).

	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Distances	Steps	Distances
SearchNearests( )	8 357 760	-	3 840	-	3 840	-
Merge(a, b)	8 357 760	-	100 636	13 302	181 159	13 316
FindNeighbors(a, b)	8 357 760	-	25 078 280	-	63 765	-
RemoveLast( )	8 349 185	-	94 779	-	45 514	-
UpdateDistances( )	48 538 136	40 166 328	25 196 128	34 068	223 627	34 097
Total	81 960 601	40 166 328	50 468 663	47 370	517 905	47 413

Table 3: Observed number of the steps and distance calculations of the PNN iterations for *House* ($k = 3$).

	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Distances	Steps	Distances
SearchNearests( )	581 798 432	-	33 856	-	33 856	-
Merge(*a*, *b*)	581 798 432	-	960 410	112 923	1 245 180	112 942
FindNeighbors(*a*, *b*)	581 798 432	-	1 745 395 296	-	255 519	-
RemoveLast( )	580 354 971	-	1 026 580	-	497 511	-
UpdateDistances( )	2 237 529 292	1 655 663 346	1 746 170 412	162 850	1 313 939	163 757
Total	4 563 279 559	1 655 663 346	3 493 586 554	275 773	3 346 005	276 699

Table 4: Observed number of the steps and distance calculations of the PNN iterations for *Miss America* ($k = 3$).

	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Distances	Steps	Distances
SearchNearests( )	20 965 544	-	6 224	-	6 224	-
Merge(*a*, *b*)	20 965 544	-	172 616	25 797	273 994	25 819
FindNeighbors(*a*, *b*)	20 965 544	-	62 896 632	-	81 537	-
RemoveLast( )	20 955 451	-	160 544	-	77 194	-
UpdateDistances( )	128 322 309	107 331 780	63 078 024	44 418	326 677	44 331
Total	212 174 392	107 331 780	126 314 040	70 215	765 626	70 150

4. Creation of the neighborhood graph

We define *k-nearest neighbor graph* (*kNN graph*) as a weighted directed graph, in which every node represents a single cluster, and the edges correspond to pointers to neighbor clusters. Every node has exactly k edges to the k nearest clusters according to a given distance function. The distance of clusters is defined by the merge distortion function of the PNN of Eq. (4). Note that this is not the only possible definition of the graph: other definitions have been given in [20, 21].

The graph can be constructed by brute force by considering all pairwise distances but at the cost of $O(N^2)$ time. We therefore consider two faster methods: the *mean-distance ordered partial search* (MPS) [15], and a new *divide-and-conquer* technique [16].

4.1 MPS for searching nearest neighbor

We propose to use the *mean-distance ordered partial search* (MPS) as was originally proposed to be used with the k-means clustering (GLA) in [15] but then generalized to the PNN distance function in [19]. The method stores the component sums of each cluster centroid (code vector). Let s_a be the one, for which we seek its nearest neighbors, and s_j the candidate to be considered. The distance of their corresponding code vectors c_a and c_j can be approximated by the squared distance of their component sums:

$$\hat{e}_{a,j} = \left(\sum_{k=1}^{K} c_{ak} - \sum_{k=1}^{K} c_{jk} \right)^2 \tag{6}$$

The component sums correspond to the projections of the vectors to the diagonal axis of the vector space. In typical data sets, the code vectors are highly concentrated along the diagonal axis, and therefore, the distance of their component sums highly correlate to their real distance. Then, given the cost function value of the best candidate found so far, vectors outside the radius defined by a given pre-condition can be excluded in the calculations, see Fig. 5.

In the PNN, the cost function (4) consists of the squared Euclidean distance ($e_{a,j}$) of the code vectors and the weighting factor ($w_{a,j}$), which can be calculated separately. The following inequality holds true:

$$w_{a,j} \cdot \hat{e}_{a,j} \leq K \cdot w_{a,j} \cdot e_{a,j} \tag{7}$$

It was originally shown to hold in Euclidean distances in [15], which we have then generalized to the cluster distances in [13]. Given the cost function value $d_{\min}$ of the best candidate found so far, the inequality (7) can be utilized in the search of nearest neighbor by using the following *pre-condition*:

$$K \cdot d_{\min} < w_{a,j} \cdot \hat{e}_{a,j} \tag{8}$$

In other words, if the squared Euclidean distance of the component sums (multiplied by the weighting factor) exceeds the distance to the best candidate found so far (multiplied by K), the value cannot be smaller than $d_{\min}$, according to (7). This is illustrated in Fig. 2, where the distance from A to B is the current minimum. All potential candidates and their projections must therefore lie inside the circle.

The pre-condition is utilized as follows. The vectors are sorted according to their component sums, and then proceed in the order given by the sorting. The search starts from the cluster s_a and proceeds bi-directionally along the projection axis. The weighting factor $w_{a,j}$ and the distance of the component sums ($\hat{e}_{a,j}$) are first calculated, and the pre-condition (8) is evaluated. If it holds true, the calculation of the actual cost function value can be omitted and the candidate cluster s_j rejected. The pre-condition can be calculated fast in $O(1)$ time as the component sums and weights are known.

In k-means clustering, the search in any of the two directions can be terminated immediately when the pre-condition is met first time. In the PNN, however, this is not possible because of the weighting factor. Even in the initialization, there may be weighted vectors as the data set can be a result of preprocessing step where duplicate vectors have been merged and weighted by their frequency. The search is therefore terminated only if the weight of the candidate cluster equals to 1. See [19] for details.

The pseudo code of the algorithm is given in Fig. 6. For simplicity, we assume that the clusters have already been sorted before the call of the routine.

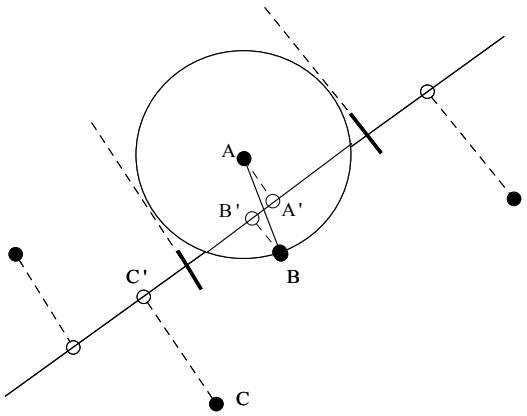


Fig. 5. Vectors (black dots) and their projections (empty dots) according to the component sums.

```
SearchNearestNeighborUsingMPS(c_a, c_j, d_min) → nn_a, d_a;
    d_min ← ∞;
    up ← TRUE;
    down ← TRUE;
    j_1 ← a;
    j_2 ← a;

    WHILE (up OR down) DO

        IF up THEN
            j_1 ← j_1 + 1;
            IF j_1 > N THEN up ← FALSE
            ELSE CheckCandidate(s_a, s_j1, n_a, d_min, nn, up);

        IF down THEN
            j_2 ← j_2 - 1;
            IF j_2 < 1 THEN down ← FALSE
            ELSE CheckCandidate(s_a, s_j2, n_a, d_min, nn, down);

        END-WHILE;

    RETURN nn, d_min;

CheckCandidate(s_a, s_j, n_a, d_min, nn, direction);
    IF PreCondition(s_a, s_j, d_min) THEN
        IF n_a = 1 THEN direction ← FALSE
    ELSE
        d ← MergeCost(s_a, s_j, d_min);
        IF d < d_min THEN
            d_min ← d;
            nn ← j;
    RETURN;

PreCondition(s_a, s_j, d_min) → BOOLEAN;
    w ← n_a · n_j / (n_a + n_j);
    ê ← (sum_a - sum_j)^2;
    RETURN( K·d_min < w · ê );
```

Fig. 6. Pseudo code of the MPS method used for the graph creation in Graph-PNN.

4.2 MPS for searching *k* neighbors

We apply the MPS method for finding the k nearest clusters as follows. We relax the condition of the graph and find any k neighbors instead of the nearest ones. This is a reasonable modification because the optimality of the graph cannot be guaranteed during the process of the PNN algorithm. Thus, by relaxing the definition of the k-nearest neighbor graph, additional speed-up can be obtained at a slight increase in the distortion function value.

In particular, we use the exact MPS method for finding the nearest neighbor but stop the search immediately when it has been found. In addition to this, we maintain ordered list of the k best candidates found so far. The rest of the neighbors are then chosen simply from the list of the candidates no matter whether they are actually the k-1 nearest or not. It is expected that the rest of the candidates are nearby vectors although not necessarily the nearest ones. Even if some links were missing, vectors in the same cluster are most likely to be connected anyhow.

Another way to limit the search is to set up a fixed search range. In this case, the limit must be set up experimentally. We will study these two alternatives (full search and limited search) later in Section 5.

The advantages of the MPS method are its simplicity and that it is expected to be fast on data sets with correlated vectors. The main disadvantage of the method is that the worst case time complexity is still $O(N^2)$, which is not any better than that of the *Brute force*. The actual speed-up is expected to be smaller on data sets with uncorrelated vectors.

4.3 Closest pair problem

Even though the MPS method is faster than the brute force, it still dominates the running time according to the experiments made in [22]. We therefore consider another approach based on the *closest pair problem* [16], which is stated as follows: given N points in d-dimensional space, find the two whose mutual distance is the smallest. The problem can be solved by recursive algorithm:

1. Divide X into X_1 and X_2 by the median hyper plane H normal to some axis.
2. Recursively solve the problem for X_1 and X_2.
3. Compute $\delta = \min(\delta_1, \delta_2)$, where δ_1 and δ_2 are the found distances in X_1 and X_2.
4. Let X_3 be the set of points that are within δ of H.
5. Use the δ*–sparsity* condition to recursively examine all pairs in X_3.

It has been shown that, in the case of 2-dimensional vector space, only a constant number of points can be neighbor in any cell in the set X_3 [23]. Assuming that the same primary axis is used in the division, the points can be pre-sorted and the analysis step can be performed in linear time. It has been proven that the algorithm takes $O(N \log N)$ time and the algorithm generalizes to multi-dimensional spaces but at the cost $O(N \log^{d-1} N)$ time [24], where d is the number of dimensions.

4.4 Divide-and-conquer method

We consider next an algorithm applicable for finding k-near neighbors based on the above divide-and-conquer approach with the following differences [25]. Firstly, we search several closest pairs for every vector in the data set. Secondly, we use *principle component analysis* (PCA) for calculating the projection axis with the maximum deviation. Thirdly, we use a distance-based heuristic for selecting the vectors to be included in the third sub set.

The pseudo code of the algorithm is given in Fig. 7. At each step of the recursion, we divide the data set X into two sub sets X_1 and X_2 of equal sizes as follows. We first calculate the principle axis of the data vectors in X, and then select (d-1)-dimensional hyper plane H perpendicular to the principal axis. The hyper plane is selected so that approximately half of the vectors belong to one side of the space, and the rest to the other side. Once the dividing procedure has been done, the two sub problems X_1 and X_2 are solved recursively. Sub problems smaller than c_k are solved by brute force search.

After the sub problems have been solved, we generate a third sub set X_3 consisting of vectors that are closer to the dividing hyper plane H than to its nearest neighbor in the corresponding sub set (X_1 or X_2). By using the control parameter c we can control the number of vectors chosen in the sub set. Once the sub set is created, the algorithm is recursively applied for X_3. Finally, the results of the three sub problems are combined. In Fig. 8 we illustrate the division of the set X to three overlapping sub sets (X_1, X_2, X_3) according to the dividing hyper plane H. The arrows indicate the nearest neighbors of the vectors.

```
Divide-and-Conquer(X, k, c_k ) → kNN
   IF ( |X|  > c_k ) THEN
      X1, X2, proj ← Divide(X);
      KNN1 ← Divide-and-conquer(X1, k, c_k);
      KNN2 ← Divide-and-conquer(X2, k, c_k);
      KNN ← KNN1 ∪ KNN2;
      X3 ← GenerateThirdSet(X, kNN, proj);
      kNN3 ← Divide-and-conquer(S3, k, c_k);
      kNN ← CombineResults(kNN, kNN3);
   ELSE
      kNN ← BruteForce(X, kNN, k);
   END-IF
RETURN kNN;

GenerateThirdSet(X, kNN, proj) → X3
   X3 ← ∅;
   FOR i ← 1 TO |X| DO
      δ ← ProjectionDistance(X[i], proj);
      IF cδ < kNN[i,1] THEN
         X3 ← X3 ∪ X[i];
RETURN X3;
```

Fig. 7. Sketch of the divide-and-conquer algorithm.

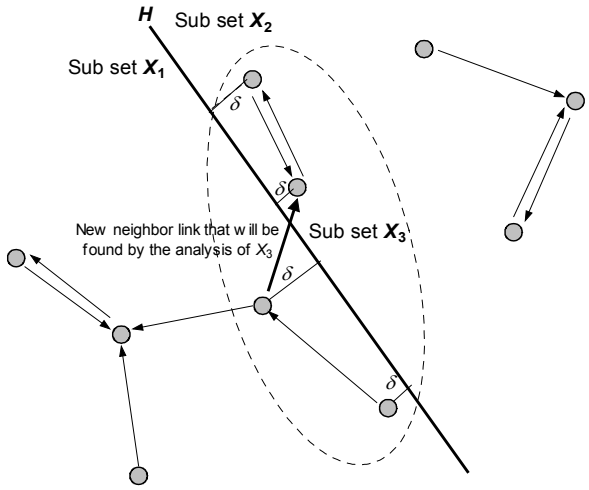


Fig. 8. Division to three overlapping sub sets (X_1, X_2, X_3) according to the dividing hyper plane H. The arrows indicate the nearest neighbors of the vectors.

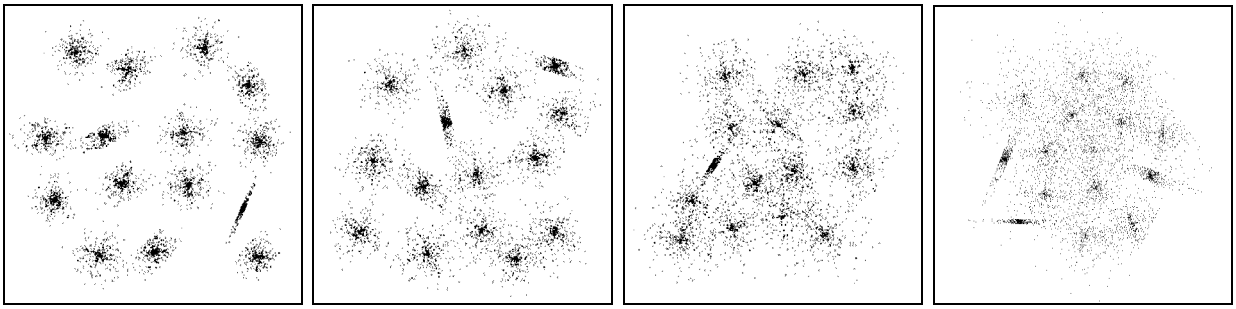
The time complexity of the proposed divide-and-conquer algorithm is estimated here by a recurrence $T(N) = 3 \cdot T(N/2) + O(N \cdot d^2)$ assuming that the size of the third sub set is less than equal to that of the other sub sets X_1 and X_2. The second term originate from the calculation of the principal axis. The rest of the calculations can be performed in linear time. The recurrence solves to $O(d^2 \cdot N^{1.58} \cdot \log N)$. It might be possible to squeeze the complexity to $O(N \cdot \log N)$ by selecting the dividing hyper plane by some simpler method, and by making tighter bounds for the third sub set. Note that the size of the X_3 can be controlled by the parameter c.

5. Experiments

We consider three image data sets (Fig. 9), four synthetically generated data sets (Fig. 10), and the *BIRCH* data sets [26]. The vectors in the first set (*Bridge*) are 4×4 blocks taken from gray-scale image, and in the second set (*Miss America*) 4×4 difference blocks of two subsequent frames in video sequence. The third data set (*House*) consists of color values of the *RGB* image. The number of clusters is fixed to M=256. The data sets S_1 to S_4 are two-dimensional sets with varying complexity in terms of spatial data distributions with M=15 clusters.

Spatial vectors:	Spatial residual vectors:	Color vectors:
		
Bridge (256×256) *K*=16, *N*=4096	*Miss America* (360×288) *K*=16, *N*=6480	*House* (256×256) *K*=3, *N*=34112*

Fig. 9. Image data sets. *Duplicate training vectors are combined and frequency information is stored. Note that when duplicates vectors are merged, all distance and merge cost calculations must always be multiplied by the frequency of the data vectors representing multiple instances of the original data set.



Data set S_1 Data set S_2 Data set S_3 Data set S_4

Fig. 10: Two-dimensional data sets with varying complexity in terms of spatial data distributions. The data sets have 5000 vectors scattered around 15 predefined clusters with a varying degree of overlap.

The effect of the neighborhood size (parameter *k*) on the running time and quality is shown in Fig. 11 with the data sets *Bridge* and *BIRCH1*. The results indicate that a very small neighborhood size such as *k*=3 is sufficient for obtaining high quality clustering for the image data sets, and larger neighborhood sizes would give only slight improvement. In the case two-dimensional data sets, however, too small neighborhood size can cause problems in the form of isolated sub clusters and somewhat larger (*k*=4 or *k*=5) neighborhood size is therefore recommended. The running time has linear dependency with the parameter *k* but the growing rate is relatively small.

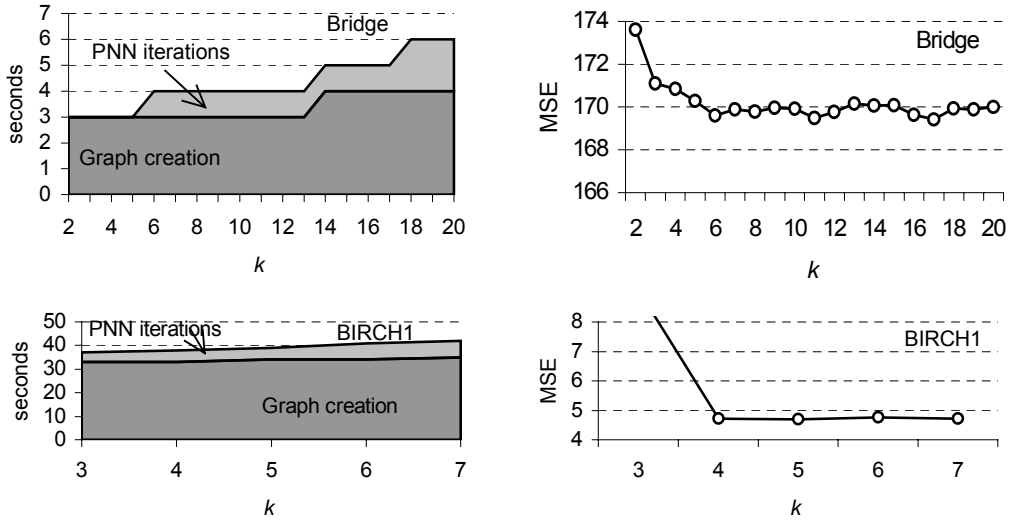


Fig. 11. The run time and quality of the graph PNN algorithm as a function of *k*. The MPS algorithm is used for graph creation and double linked list in the PNN iterations. The results are (from top to down) for *Bridge* and *BIRCH1*.

The running time consists of two parts: graph creation and the PNN iterations. The run times of four variants (*Brute force*, *Brute force fast*, *MPS, Divide-and-conquer*) for the graph creation are shown in Table 5. The first two variants (*Brute force*) calculate all pairwise distances in order to create the graph. The *Brute force fast* variant considers only half of the distances and is therefore about 50% faster than the *Brute force* itself.

The MPS variant, however, is clearly faster than the *Brute force fast* especially in the case of *Bridge* and *House*. In the case of *Miss America*, the MPS does not provide improvement. The divide-and-conquer technique is faster than the MPS with the 16-D image data sets (*Bridge* and *Miss America*) but slower with the 3-D data set (*House*). The small differences in the MSE-values are due to the different order of processing in the case of ties, and are reported merely to show the difference in quality between the *Graph-PNN* and the *Fast PNN*.

The running times of the PNN iterations are shown in Table 6 for the *simple* algorithm, and for the *double linked* algorithm. The results show that the *simple* algorithm is useful with *Bridge* and *Miss America*, but the *double linked* algorithm is significantly more efficient with *House*. The illustrations in Fig. 11 show that the PNN iterations can be performed efficiently and that the graph creation is still the bottleneck of the algorithm.

The overall running times and the corresponding number of distance calculations are summarized in Table 7. Comparative results are given for the fast exact *PNN* [7], and the fast exact *PNN* with several speed-up methods as proposed in [19]. The results show that the *graph PNN* is significantly faster than the fast exact *PNN* with all data sets. The results are most remarkable with the largest data set (*House*), for which the running time was reduced down to 9 % from that of the fastest comparative variant. The corresponding numbers for *Bridge* and *Miss America* are 33 % and 43 %.

Table 5. Running times of the graph creation.

		Bridge		*House*		*Miss America*	
		Time	MSE	Time	MSE	Time	MSE
Graph PNN	Brute Force	34	171.17	881	6.43	89	5.44
	Brute Force /fast	16	172.20	446	6.33	43	5.48
	MPS	3	171.11	18	6.37	44	5.44
	Divide-and-conquer	2	171.80	49	6.58	7	5.44

Table 6. Running times of the PNN iterations.

		Bridge		*House*		*Miss America*	
		Time	MSE	Time	MSE	Time	MSE
Graph PNN	Simple	4	171.12	542	6.40	12	5.45
	Double linked	< 1	171.11	1	6.37	< 1	5.44

Table 7. Summary of the running times and the number of distance calculations of the MPS method in comparison to the full search PNN.

		Bridge		*House*		*Miss America*	
		Distance calculations	Run time	Distance calculations	Run time	Distance calculations	Run time
Fast PNN		48 552 888	79	2 237 460 562	1574	128 323 740	229
Fast PNN	+MPS+PDS +lazy	6 167 439	9	37 752 863	190	83 323 889	106
Graph PNN	Graph creation	2 341 547	3	19 017 163	18	32 440 442	44
	Iterations	47 413	< 1	276 699	1	70 150	< 1
	Total	2 388 960	3	19 293 862	19	32 510 592	44

The graph creation is evidently a bottleneck in the *Graph-PNN*. We therefore consider also the faster but sub-optimal variants: *limited-search MPS*, and divide-and-conquer with the control parameter. The search is performed as discussed in Sections 4.2 and 4.4. The corresponding time-distortion performance is illustrated in Fig. 12 for Miss America. Even though the results favor the divide-and-conquer method, it has much narrower operative time marginal, and the limited-search MPS is better for all other sets.

Comparisons of the different algorithms are summarized in Table 8 with the following methods included:

- *Fast PNN*
- *Graph-PNN*
- *Graph-PNN + GLA*
- *GLA*

The *Fast PNN* has two variants: the fast implementation as proposed in [7] and the improved variant [19]. The latter one uses three speed-up techniques: the *PDS*, *MPS* and *Lazy evaluation* of the distances. The *GLA* has two variants: the original method [8], and a faster variant that uses PDS, MPS and activity detection for speed-up [27]. Results are given also for *Graph-PNN + GLA*, in which the data is first processed by the *Graph-PNN* and the result is input to the *GLA*. The results show that the *Graph-PNN* produces better result with an algorithm that is competitive to the GLA in speed.

The final clustering and the neighborhood graph of the Graph-PNN (k=5) are illustrated in Fig. 13. It shows that the Graph PNN achieves the correct clustering. The final graph does not have k=5 outgoing edges for every node although every cluster is still connected to the same graph. With a smaller number of neighbors (k=3), however, there would have been isolated components and in some cases, the algorithm degenerated to situation where there were not enough edges to achieve the optimal clustering.

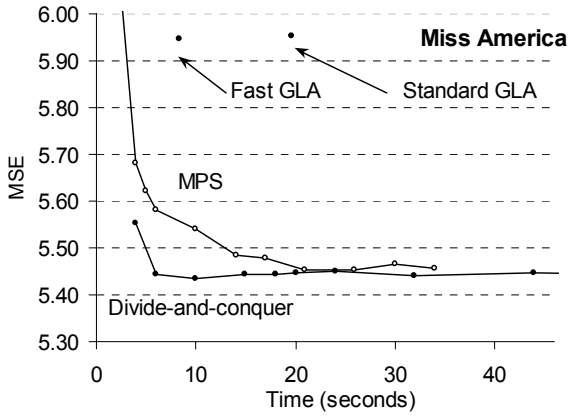


Fig. 12. The time-distortion performance of the Graph-PNN.

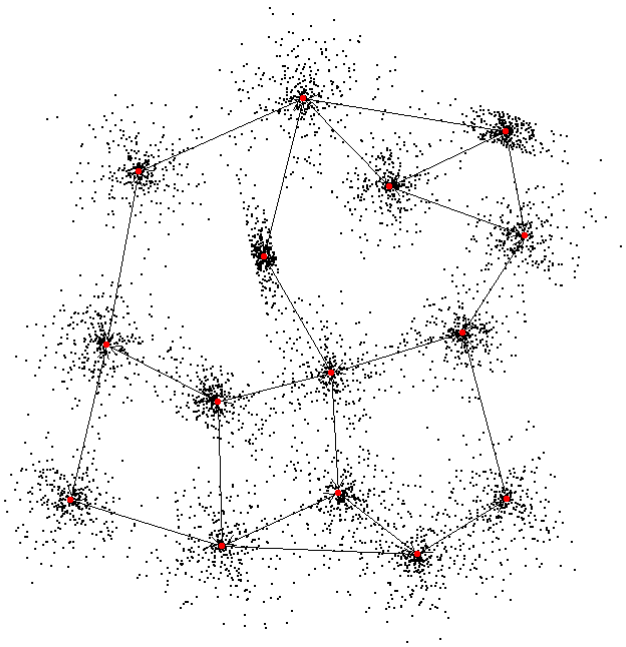


Fig. 13. Illustration of the final results of the Graph-PNN for set S_2.

Table 8. Comparison of the Graph-PNN (k=5) with other methods.

Image data sets		*Bridge*		*House*		*Miss America*	
		Time	MSE	Time	MSE	Time	MSE
Fast PNN	Full search	79	168.92	1574	6.27	229	5.36
	+PDS+MPS+Lazy	9	168.92	190	6.26	106	5.37
Graph PNN	Full MPS	3	170.28	19	6.33	45	5.11
	Limited search MPS	3	170.56	14	6.51	6	5.58
	Divide-and-conquer	2	170.69	51	6.37	8	5.43
Graph PNN + GLA	Full MPS	4	166.23	20	6.14	47	5.30
	Limited search MPS	4	166.38	15	6.18	9	5.34
	Divide-and-conquer	2	165.81	52	6.14	10	5.30
GLA	Standard	13	179.95	23	7.77	20	5.95
	+PDS+MPS+Activity	2	180.02	3	7.80	8	5.95

Birch data sets		*BIRCH 1*		*BIRCH 2*		*BIRCH 3*	
		Time	MSE	Time	MSE	Time	MSE
Fast PNN	Full search	N/A	4.73	N/A	2.28	N/A	1.96
	+PDS+MPS+Lazy	2397	4.73	2115	2.28	2316	1.96
Graph PNN	Full MPS	40	4.71	16	2.28	34	1.96
	Limited search MPS	37	4.73	15	2.28	28	2.02
	Divide-and-conquer	N/A	N/A	N/A	N/A	N/A	N/A
Graph PNN + GLA	Full MPS	44	4.64	17	2.28	51	1.87
	Limited search MPS	41	4.64	16	2.88	44	1.90
	Divide-and-conquer	N/A	N/A	N/A	N/A	N/A	N/A
GLA	Standard	209	5.51	43	7.42	171	2.41
	+PDS+MPS+Activity	29	5.34	8	7.85	35	2.50

Synthetic data sets		S_1		S_2		S_3		S_4	
		Time	MSE	Time	MSE	Time	MSE	Time	MSE
Fast PNN	Full search	25	8.93	25	13.44	25	17.70	25	17.52
	+PDS+MPS+Lazy	3	8.93	3	13.44	3	17.70	3	17.52
Graph PNN	Full MPS	< 1	9.07	< 1	13.41	< 1	17.21	< 1	16.69
	Limited search MPS	< 1	9.07	< 1	13.41	< 1	17.21	< 1	16.69
	Divide-and-conquer	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Graph PNN + GLA	Full MPS	< 1	8.92	< 1	13.28	< 1	16.89	< 1	15.71
	Limited search MPS	< 1	8.92	< 1	13.28	< 1	16.89	< 1	15.71
	Divide-and-conquer	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
GLA	Standard	< 1	19.02	< 1	18.78	< 1	19.78	< 1	16.72
	+PDS+MPS+Activity	< 1	18.07	< 1	16.69	< 1	18.53	< 1	16.71

6. Conclusions

Graph-based agglomerative clustering algorithm was proposed. We found out that a relatively small neighborhood size (k=3-5) is sufficient to produce clustering with similar quality to that of the full search. At the same time, significantly fewer distance calculations were needed and, thus, remarkable speed-up was achieved.

Two graph creation algorithms were considered: the mean-distance ordered partial search (MPS), and a divide-and-conquer technique. The latter one is faster in the case of the high dimensional data sets. In the 2-dimensional or 3-dimensional color clustering, on the other hand, it was slower than the MPS method.

We conclude that the improvement due to the neighborhood graph is significant. The graph creation, however, is the bottleneck of the algorithm. It remains an open question whether faster method could be invented with better time-distortion performance than the proposed divide-and-conquer and the limited-search MPS algorithm.

References

[1] R. Dubes and A. Jain, *Algorithms that Cluster Data*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[2] B.S. Everitt, *Cluster Analysis* (3rd edition). Edward Arnold / Halsted Press, London, 1992.

[3] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht, 1992.

[4] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley Sons, New York, 1990.

[5] J.H. Ward, "Hierarchical grouping to optimize an objective function," *J. Amer. Statist.Assoc.*, 58, 236-244, 1963.

[6] W.H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 37 (10), 1568-1575, October 1989.

[7] P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, "Fast and memory efficient implementation of the exact PNN," *IEEE Trans. on Image Processing*, 9 (5), 773-777, May 2000.

[8] Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. on Communications*, 28 (1), 84-95, January 1980.

[9] D.P. de Garrido, W.A. Pearlman and W.A. Finamore, "A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids," *IEEE Trans. on Circuits and Systems for Video Technology*, 5 (2), 83-95, April 1995.

[10] T. Kaukoranta, P. Fränti and O. Nevalainen, "Iterative split-and-merge algorithm for VQ codebook generation," *Optical Engineering*, 37 (10), 2726-2732, October 1998.

[11] P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, "Genetic algorithms for large scale clustering problem," *The Computer Journal*, 40 (9), 547-554, 1997.

[12] P. Fränti, "Genetic algorithm with deterministic crossover for vector quantization," *Pattern Recognition Letters*, 21 (1), 61-68, January 2000.

[13] J. Shanbehzadeh and P.O. Ogunbona, "On the computational complexity of the LBG and PNN algorithms," *IEEE Trans. on Image Processing*, 6 (4), 614-616, April 1997.

[14] J.C. Gover and G.J.S. Ross, "Minimum spanning trees and single linkage cluster analysis," *Applied Statistics*, 18, 54-64, 1969.

[15] S.-W. Ra and J.K. Kim, "A fast mean-distance-ordered partial codebook search algorithm for image vector quantization," *IEEE Trans. on Circuits and Systems*, 40 (9), 576-579, September 1993.

[16] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[17] T. Kurita, "An efficient agglomerative clustering algorithm using a heap". *Pattern Recognition*, 24 (3), 205-209, March 1991.

[18] T. Kaukoranta, P. Fränti and O. Nevalainen, "Vector quantization by lazy pairwise nearest neighbor method," *Optical Engineering*, 38 (11), 1862-1868, November 1999.

[19] O. Virmajoki, P. Fränti, T. Kaukoranta, "Practical methods for speeding-up the pairwise nearest neighbor method", *Optical Engineering*, 40 (11), 2495-2504, November 2001.

[20] S. Arya and D.M. Mount, "Algorithm for fast vector quantization," *Proceedings Data Compression Conference*, Snowbird Utah, 381-390, 1993.

[21] A.D. Constantinou, R.D. Bull and C.N. Canagarajah, "A new class of VQ codebook design algorithms using adjacency maps," *SPIE Electronics Imaging 2000*, San Jose, 3974, 625-634, 2000.

[22] P. Fränti, O. Virmajoki and V. Hautamäki "Fast PNN-based clustering using k-nearest neighbor graph", *IEEE Int. Conf. on Data Mining* (*ICDM'03*), Melbourne, FL, USA, November 2003. (to appear)

[23] M.I. Shamos, "Geometric complexity", *Proc. 7th Annual ACM Symposium on the Theory of Computing*, 224-233, Albuquerque, New Mexico, 1975.

[24] J.L. Bentley and M.I. Shamos, "Divide-and-Conquer in Multidimensional Space", *8th Annual ACM Symposium on the Theory of Computing*, 220-230, New York, 1976.

[25] O. Virmajoki and P. Fränti, "Divide-and-conquer algorithm for creating neighborhood graph for clustering", *Int. Conf. on Pattern Recognition (ICPR'04*), Cambridge, UK, August 2004. (submitted)

[26] T. Zhang, R. Ramakrishnan, M. Livny, "BIRCH: A New Data Clustering Algorithm and Its Applications", *Data Mining and Knowledge Discovery*, 1 (2), 141-182, 1997.

[27] T. Kaukoranta, P. Fränti and O. Nevalainen, "A fast exact GLA based on code vector activity detection", *IEEE Trans. on Image Processing*, 9 (8), 1337-1342, August 2000.