Fast and space efficient PNN algorithm with delayed distance calculations

Timo Kaukoranta[1], Pasi Fränti[2] and Olli Nevalainen[1]

[1] Turku Centre for Computer Science (TUCS)
Department of Computer Science, University of Turku
Turku, Finland

[2] Department of Computer Science, University of Joensuu
Joensuu, Finland

Abstract

Clustering of a data set can be done by the well-known Pairwise Nearest Neighbor (PNN) algorithm. The algorithm is conceptionally very simple and gives high quality solutions. A drawback of the method is the relatively large running time of the original (exact) implementation. Recently, an efficient version of the exact PNN algorithm has been introduced in literature. In this paper we give a faster implementation of this algorithm. The idea is to postpone the updating of the nearest neighbor information in order to reduce the number of cluster distance calculations. Correctness of the algorithm follows from the monotony of the cluster distances. Practical tests show that the new organization of the algorithm decreases the running time of PNN by ca. 35 per cent.

Keywords: *Vector quantization, Codebook generation, Clustering algorithms, PNN algorithm.*

1. INTRODUCTION

We study the problem of generating a *codebook* for a *vector quantizer* (VQ). The aim is to find *M code vectors* (*codebook*) for a given set of *N training vectors (training set*) by minimizing the average pairwise distance between the training vectors and their representative code vectors. The problem of generating an optimal codebook is a combinatorial optimization problem and it is NP-complete [1]. In other words, there is no known polynomial time algorithm for finding the globally optimal solution, but reasonable suboptimal solutions are typically obtained by heuristic algorithms [2-7]. The most cited and widely used algorithm is the *generalized Lloyd algorithm* (GLA) [2, 3]. It starts with an initial solution, which is iteratively improved using two optimality criteria in turn until a local minimum has been reached.

A different approach is to build the codebook hierarchically. The *pairwise nearest neighbor* (PNN) algorithm [4] starts by constructing an initial codebook in which each training vector is considered as its own code vector. Two nearest code vectors are merged at each step of the algorithm and the process is repeated until the desired size of the codebook has been reached. The PNN has the benefit of conceptual simplicity and the high quality of the solutions. The method has also the advantages that the bit rate of the vector quantizer is easier to control because the hierarchical approach produces codebooks of differing sizes as a side-product. The algorithm can also be used to produce an initial codebook for another optimizer (such as the GLA), or it can be embedded into hybrid methods such as genetic algorithm [6], or iterative split-and-merge method [7].

A drawback of the PNN is the relatively high running time in its exact form. There are a large number of steps because typically $M<<N$, and at each step all pairwise distances must be calculated for finding the pair of vectors to be merged. This is very slow for large training sets. Most of the computation originates from the calculation of the pairwise distances. However, only two code vectors are changed at each step of the PNN and therefore most of the distance calculations are unnecessary. A fast and space efficient implementation has been recently given independently by [8] and [9]. The idea is to keep track of the nearest neighbor of each cluster. After the merge operation, the pointers must be updated only for clusters whose nearest neighbor is one of the merged clusters.

In this paper we propose an improved version of the above nearest neighbor variant by [8]. The main idea is to reduce the distance calculations further by delaying the updates. We can do this because, as we will show, the cluster distances increase monotonically by the time. It follows from this property that if a given distance value is not the minimum distance prior to a certain update, it will not be the minimum after the update either. It is therefore sufficient to update a distance value only when it becomes the minimum distance. The new method, referred here as *Lazy PNN*, reduces the number of updates considerably. Empirical tests show that the Lazy PNN improves the running time by 30% on average.

The rest of the paper is organized as follows. The problem formulation and the structure of the PNN are given in Section 2. The Lazy PNN is then introduced in Section 3. Simulation results for various training sets are shown in Section 4, and conclusions are finally drawn in Section 5.

2. THE PNN METHOD

We next use the following notations:

T	Set of N training vectors $T=\{T_1,T_2,\ldots,T_N\}$.
C	Codebook of m code vectors $C=\{C_1, C_2,\ldots,C_m\}$.
M	Size of the final codebook.
K	The dimension of the vectors.
S_i	Cluster (set) of n_i training vectors.
NN_i	Index of the nearest neighbor of the cluster S_i.
d_i	Increase of the distortion if the clusters i and Nn_i are merged.
R_i	Validity indicator; R_i=true if and only if d_i is valid.

We consider a set of N training vectors in a K-dimensional Euclidean space. The task of the codebook construction is to find a set of M code vectors (i.e. a codebook) by minimizing the average squared distance D between the training vectors T_i and their representative code vectors C_j:

$$D=\sum_{j=1}^{M}\sum_{T_i\in S_j}\left\|T_i-C_j\right\|^2 \tag{1}$$

Here $S=\{S_1,\ldots,S_M\}$ defines the clustering of the training set T. For a given codebook C, the optimal clustering can be constructed by assigning each training vector T_i to the cluster j_0 for which:

$$\left\|T_i-C_{j0}\right\|^2=\min_{j=1,\ldots,M}\left\|T_i-C_j\right\|^2 \tag{2}$$

The basic structure of the PNN is shown in Fig. 1. The method starts by initializing each training vector T_i as its own cluster S_i. At each step of the algorithm, two nearest clusters (S_a and S_b) are searched and merged. The distance (or *merge cost*) d between two clusters is defined as the increase in the distortion of the codebook if the clusters are merged. It is calculated as the squared Euclidean distance of the cluster centroids (code vectors) weighted by the number of vectors in the two clusters [4]:

$$d(S_a,S_b)=\frac{n_a n_b}{n_a+n_b}\cdot\left\|C_a-C_b\right\|^2 \tag{3}$$

The chosen clusters S_a and S_b are then merged. The size of the combined cluster S_{a+b} is $n_{a+b}=n_a+n_b$, and the corresponding code vector is the centroid of the training vectors in the cluster. It can be calculated as the weighted average of C_a and C_b:

$$C_{a+b}=\frac{n_a C_a+n_b C_b}{n_a+n_b} \tag{4}$$

It is thus sufficient to maintain only the cluster centroids (C_i) and the sizes of the clusters (n_i) in the implementation of the algorithm. The merge process is repeated until the codebook reaches the size M.

```
Let each training vector be a code vector ($m=N$).
Repeat
    Find two nearest clusters $S_a$ and $S_b$ to be merged.
    Merge $S_a$ and $S_b$; $m \leftarrow m-1$.
    Update data structures.
Until $m=M$.
```

Figure 1: Structure of the PNN method.

The (exact) PNN applies local optimization strategy where all possible cluster pairs are considered and the one increasing the distortion least (smallest cost function value) is chosen for merge. Straightforward implementation [4] recalculates all distances at each step of the algorithm. No additional data structures are needed but the algorithm takes $O(N^3K)$ time because there are $O(N)$ steps in total, and there are $O(N^2)$ cluster pairs to be checked at each step.

Most of the computation of the PNN originates from the calculation of the pairwise distances. Since only two code vectors are changed at each step, majority of the distance calculations is unnecessary. To reduce the number of distance calculations, previous pairwise cluster distances can be stored in an $N\times N$ matrix. The minimum cluster distance is searched from the matrix and the corresponding cluster pair is merged. New distances are then calculated between the new cluster and remaining clusters only. The algorithm runs in $O(N^2K+N^3)$ time where the former term originates from the distance calculations and the latter from the search for the minimum [10]. The disadvantages of this approach are cubic running time and quadratic memory consumption.

Kurita's method [11] stores all pairwise distances into a matrix, as above, but it utilizes a heap structure for searching the minimum distance. The merged clusters can be found by popping the smallest element from the top of the heap in $O(\log N)$ time. Only $O(N)$ distance updates are needed after each merge step; each of these updates takes $O(K + \log N)$ time because of the distance calculation and the heap operation. The method thus runs in $O(N^2K + N^2 \log N))$ time. The method still requires $O(N^2)$ memory, which is impractical for large training sets.

Another approach (τ-*PNN*) has been recently studied by Fränti and Kaukoranta in [8], and by Shen and Chang in [9]. The main idea is to maintain only a nearest neighbor pointer for each cluster. The index of the nearest cluster (NN_i) and the corresponding cost function value d_i are stored in the *nearest neighbor table*. The optimal cluster pair (S_a, S_b) to be merged can be found by a linear search among the d_i-values. After the merge operation, the nearest neighbor pointers must be updated for those clusters for which $NN_i=a$ or $NN_i=b$. Fortunately, in practice, there are only a small number (denoted by τ) of pointers to be updated on average. The method thus takes $O(\tau N^2K)$ time in total. In addition to that, the memory requirement of this approach is only $O(N)$.

3. LAZY PNN ALGORITHM

We propose next an improved version of the nearest neighbor variant of the PNN. The main idea is to reduce the distance calculations even further. Although the total number of updates (τ) is rather small on average, the search of the nearest neighbor is still an expensive $O(NK)$ time operation and it dominates the running time of the algorithm.However, the distance calculations can be delayed and therefore a remarkable number of updates may be avoided. The new method is referred here as *Lazy PNN*.

The application of delayed distance calculations is based on the *monotony property* of the cluster distances, which is defined as follows. Suppose that at a certain moment the minimal merge cost is $d(S_a, S_b)$ and the clusters S_a and S_b are merged. It is possible that the centroid of the merged cluster (C_{a+b}) may become closer to the centroid of a third cluster S_c than C_c was in respect to the original cluster centroids (C_a and C_b), see Fig. 2. However, the mass of the merged cluster increases so much that the merge cost $d(S_{a+b}, S_c)$ can never become smaller than both of $d(S_a, S_c)$ and $d(S_b, S_c)$. The cost function d is therefore monotonically increasing as a function of time. This is formalized in the following lemma:

Lemma 1. Consider the clusters S_a, S_b, S_c with centroids C_a, C_b, C_c, and frequencies n_a, n_b, n_c. Assume that $d(S_a,S_b) \le d(S_a,S_c) \le d(S_b,S_c)$ and $n_a, n_b, n_c \ge 1$. Then it holds that $d(S_a,S_c) \le d(S_{a+b},S_c)$.

Proof. See Appendix.

Because of the monotony property, we know that if a given distance value is not the minimum distance before the update, it will not be the minimum after the update either. We may therefore delay the distance calculations until the old cost function value becomes a candidate for being the smallest distance. We therefore mark each value whether it is up to date or not. The optimal cluster pair to be merged can now be searched as before with only one difference; when an out-dated distance value is found to be minimal it is recalculated. This practice does not compromise the exactness of the algorithm but it may remarkably reduce the number of expensive distance calculations.

The lazy processing can be applied to the nearest neighbor method as such. Instead, we take one step further and maintain a min-heap of the distance values (d_i) and the corresponding nearest neighbor pointers (NN_i). The heap elements contain an additional flag (R_i), which indicates whether the distance value is up-to-date or not. The difference to Kurita's method is that we only store one element per cluster whereas Kurita stores all distances. The use of the heap has no asymptotic influence on the running time because the recalculation of the distance values still dominates the running time. The heap, however, may speed-up the practical implementation because we need to consider only the root of the heap and therefore may potentially avoid some distance recalculations.

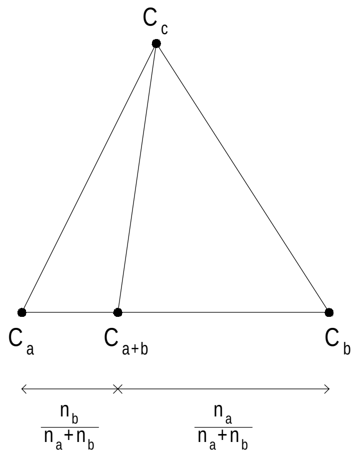

Figure 2: Illustration of the clusters in 2-dimensional space. C_a and C_b are the centroids Illustration of the clusters in 2-dimensional space. of the two clusters to be merged, C_{a+b} is the centroid of the merged cluster, and C_c is the centroid of any other cluster.

The pseudo code of the Lazy PNN algorithm is presented in Fig. 3. The algorithm starts by initializing each cluster (or code vector) with one training vector. For each cluster S_i, the nearest neighbor according to (3) is searched among the other clusters S_i ($i \ne j$). Information of the nearest neighbor is stored and marked valid. The nearest neighbor distances d_i of all clusters are inserted in the minimum heap H. The algorithm is then iterated until the size of the codebook reaches M. At each step, the cluster S_a with the smallest d-value is deleted from H. If the d-value of S_a is out-of-date (R_a=False) its nearest neighbor is recalculated and reinserted in the heap. The process is repeated until a valid minimal distance is obtained from the top of H.

The cluster S_a and its nearest neighbor S_b ($b=NN_a$) are merged according to (4) and the number of vectors in the new cluster is calculated. The nearest neighbor of the new cluster is determined and inserted in the heap according to its d-value. The information of the merged cluster is stored to S_a leaving S_b unused. The non-existing cluster S_b is removed from the heap. This can be done in $O(\log N)$ time by maintaining a position index to the heap for each cluster. To avoid gaps in the indexing, the last cluster S_m replaces S_b. All the nearest neighbor pointers NN_i to S_m are reassigned to S_b. Finally, the size of the intermediate codebook m is subtracted by one.

```
Input:          {$T_i$} and $M$.
Output:         {$C_i$}.
Main program
$H \leftarrow \emptyset$;
$m \leftarrow N$;
for $\forall\ i \in [1, m]$: $C_i \leftarrow T_i$; $n_i \leftarrow 1$;
for $\forall\ i \in [1, m]$: UpdateNNpointer($H, i$);
repeat
    PickPair($H,a,b$);
    MergeClusters($a,b$);
    UpdateNNpointer($H, a$);
until $m = M$;

Procedure PickPair(H,a,b):
$a \leftarrow$DeleteMin($H$);
while $R_a$=False
    UpdateNNpointer($H,a$);
    $a \leftarrow$DeleteMin($H$);
$b \leftarrow NN_a$; Remove($H, b$);

Procedure MergeClusters(a,b):
$C_a \leftarrow (n_a C_a + n_b C_b) / (n_a + n_b)$;
$n_a \leftarrow n_a + n_b$;
for $\forall\ i \in [1, m]$: if $NN_i = a \vee NN_i = b$ then $R_i \leftarrow$False;
$C_b \leftarrow C_m$; $n_b \leftarrow n_m$; $NN_b \leftarrow NN_m$; $d_b \leftarrow d_m$; $R_b \leftarrow R_m$;
for $\forall\ i \in [1, m-1]$: if $NN_i = m$ then $NN_i \leftarrow b$;
$m \leftarrow m-1$;

Procedure UpdateNNpointer(H, a)
$NN_a \leftarrow$ FindNearestCluster($a$);
$d_a \leftarrow d(a, NN_a)$;
$R_a \leftarrow$True;
Insert($H, a$);
```

Figure 3. Pseudo code of the Lazy PNN.

4. PRACTICAL RESULTS

We generated training sets from six different images: *Bridge*, *Camera*, *Miss America*, *Table tennis*, *Airplane* and *House*, see Fig. 4. The vectors in the first two sets (*Bridge*, *Camera*) are 4×4 pixel blocks from the image. The third and fourth sets (*Miss America*, *Table Tennis*) have been obtained by subtracting two subsequent image frames of the original video image sequences, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames have been used. The fifth and sixth data sets (*Airplane*, *House*) consist of color values of the *RGB* images, prequantized to 5 bits per each color component. Applications of this kind of data sets is found in image and video image coding (*Bridge*, *Camera*, *Miss America*, *Table tennis*), and in color image quantization (*Airplane*, *House*).

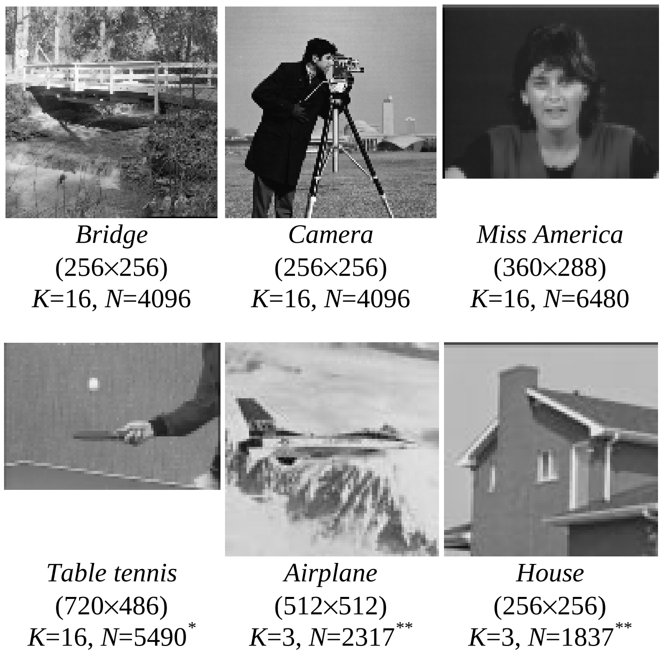

Figure 4. Sources of the training sets. *The training set *Table tennis* is constructed by random sampling only every fourth block. **The images *Airplane* and *House* are prequantized by 5 bits per each color component.

Properties of the compared PNN methods are presented in Table 1. Table 2 shows a summary of the test results for three main variants. Kurita's method was not applied because its memory consumption is too high for these training sets. The size of the codebook was fixed to M=256. Both nearest neighbor variants (t-PNN and Lazy PNN) are clearly superior to the original PNN being about 100 to 500 times faster. From these two variants, the Lazy PNN is about 35% faster. The speed-up originates mainly from the decreased number of distance recalculations; the average number of updates (t) varied from 4.4 to 5.6 in the t-PNN, and from 3.0 to 3.8 in the Lazy PNN. Small improvement is also due to the use of the heap structure.

In order to compare the nearest neighbor variants with the Kurita's method we generated subsets from *Bridge* and *House* by random sampling. The smaller training sets are of size N=(128, 256, 384, 512, 640, 768, 896, 1024). In these tests the codebook size was set to M=1 for getting the maximal number of iterations. The results for the two cases are illustrated in Fig. 5 and Fig. 6. The Lazy PNN is comparable to the Kurita's method in speed but it has the benefit of smaller memory consumption. The actual running times are virtually the same for training sets (e.g. *Bridge*) with large vector dimensions (K=16), whereas for training sets (e.g. *House*) with smaller dimensions (K=3) the Lazy PNN is faster.

Table 1. Summary of the compared exact PNN methods.

Method	Time	Extra data	Space
Original [4]	$O(N^3K)$	-	$O(N)$
Kurita [11]	$O(N^2K+N^2\log N)$	Dist.matrix	$O(N^2)$
t-PNN [8]	$O(\mathrm{t}N^2K)$	*NN*-table	$O(N)$
Lazy PNN	$O(\mathrm{t}N^2K)$	*NN*-table	$O(N)$

Table 2. Summary of the running times (in seconds).

Training set	**Original**	**τ-PNN**	**Lazy PNN**	**Time saved**
Bridge	73007	331	220	33.5%
Camera	73040	300	209	30.3%
Miss America	292351	870	557	36.0%
Table tennis	177019	649	419	35.4%
Airplane	4751	48	28	41.7%
House	2341	27	17	37.0%

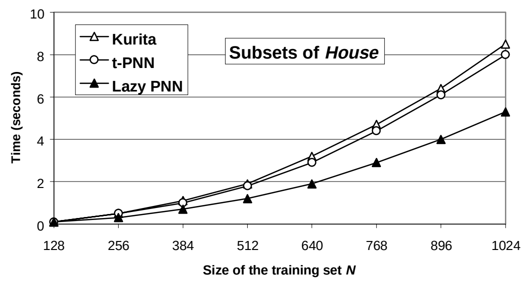

Figure 5. Comparison of the fast exact PNN methods for subsets of *House* (K=3) when M=1.

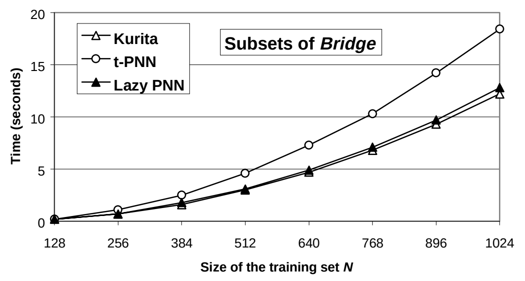

Figure 6. Comparison of the fast exact PNN methods for subsets of *Bridge* (K=16) when M=1.

5. CONCLUSION

A fast variant of the PNN algorithm was introduced. The main idea of the algorithm is to maintain a table of nearest neighbors as in the t-PNN algorithm. In addition to that we postpone the updating of the closest distance information to the moment when the (old) distance becomes the new tentative minimum among the cluster distances. This action is possible due to the monotony of the cluster distances. The monotony property is utilized further by using a heap structure as a priority queue to maintain the set of cluster distances.

Our practical tests indicate that Lazy PNN is about 100 to 500 times faster than the original PNN. The new method is comparable to the Kurita's algorithm in speed but it has the benefit of smaller memory by factor N. In the comparison to t-PNN the number of updated cluster distances was observed to reduce by 35% on average. The proposed method is rather simple to implement and practical because no distance matrix is needed for storing the pairwise distances.

We also gave a proof of the monotony property for the vectors in Euclidean space. It is an open question whether the result generalizes to other cluster distances. This would expand the usefulness of the new algorithm to the general clustering problem.

6. ACKNOWLEDGEMENTS

The work of Pasi Fränti was supported by a grant from the Academy of Finland.

7. REFERENCES

[1] M.R. Garey, D.S. Johnson, H.S. Witsenhausen, "The Complexity of the Generalized Lloyd-Max Problem". *IEEE Transactions on Information Theory*, Vol.28 (2), pp.255-256, March 1982.

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

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

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

[5] P. Fränti, T. Kaukoranta and O. Nevalainen, "On the splitting method for VQ codebook generation", *Optical Engineering*, **36** (11), November 1997.

[6] P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, "Genetic algorithms for codebook generation in VQ", *Proc. 3rd Nordic Workshop on Genetic Algorithms*, Helsinki, Finland, pp. 207-222, 1997.

[7] T. Kaukoranta, P. Fränti and O. Nevalainen, "Iterative split-and-merge algorithm for VQ codebook generation", *Optical Engineering*, 1998. (to appear)

[8] P. Fränti and T. Kaukoranta, "Fast implementation of the optimal PNN method", *Proc. Int. Conf. on Image Processing (ICIP)*, Chicago, Illinois, October 1998.

[9] D.-F. Shen and K.-S. Chang, "Fast PNN algorithm for design of VQ initial codebook", *Proc. SPIE 3309, Visual Communications and Image Processing '98*, San Jose, California, pp. 842-850, 1998.

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

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

8. APPENDIX: PROOF OF LEMMA 1

We have the following relationships between the distances of the cluster centroids:

$$\|C_a - C_c\|^2 = \|C_a - C_b\|^2 + \|C_b - C_c\|^2 - 2(C_a - C_b)\cdot(C_b - C_c) \qquad \text{(A.1)}$$

and

$$\|C_{a+b} - C_c\|^2 = \left(\frac{n_a}{n_a+n_b}\right)^2 \|C_a - C_b\|^2 + \|C_b - C_c\|^2 - 2\left(\frac{n_a}{n_a+n_b}(C_a - C_b)\right)\cdot(C_b - C_c) \qquad \text{(A.2)}$$

We can thus write (A.2) in the form:

$$\|C_{a+b} - C_c\|^2 = \left(\frac{n_a}{n_a+n_b}\right)^2 \|C_a - C_b\|^2 + \|C_b - C_c\|^2 - \frac{n_a}{n_a+n_b}\left(\|C_a - C_b\|^2 + \|C_b - C_c\|^2 - \|C_a - C_c\|^2\right)$$

$$= \frac{n_a}{n_a+n_b}\|C_a - C_c\|^2 + \frac{n_b}{n_a+n_b}\|C_b - C_c\|^2 - \frac{n_a n_b}{(n_a+n_b)^2}\|C_a - C_b\|^2$$

This gives the condition

$$(n_a+n_b)\|C_{a+b} - C_c\|^2 = n_a\|C_a - C_c\|^2 + n_b\|C_b - C_c\|^2 - d(S_a, S_b).$$

Now, we can write the difference of the merge costs in form:

$$d(S_{a+b}, S_c) - d(S_a, S_c) = \frac{n_c(n_a+n_b)}{n_a+n_b+n_c}\|C_{a+b} - C_c\|^2 - d(S_a, S_c)$$

$$= \frac{n_a n_c}{n_a+n_b+n_c}\|C_a - C_c\|^2 + \frac{n_b n_c}{n_a+n_b+n_c}\|C_b - C_c\|^2 - \frac{n_c}{n_a+n_b+n_c}d(S_a, S_b) - d(S_a, S_c)$$

$$= \frac{n_a+n_c}{n_a+n_b+n_c}\cdot\frac{n_a n_c}{n_a+n_c}\|C_a - C_c\|^2 + \frac{n_b+n_c}{n_a+n_b+n_c}\cdot\frac{n_b n_c}{n_b+n_c}\|C_b - C_c\|^2 - \frac{n_c}{n_a+n_b+n_c}d(S_a, S_b) - d(S_a, S_c)$$

$$= \frac{n_a+n_c}{n_a+n_b+n_c}\cdot d(S_a, S_c) + \frac{n_b+n_c}{n_a+n_b+n_c}\cdot d(S_b, S_c) - \frac{n_c}{n_a+n_b+n_c}d(S_a, S_b) - d(S_a, S_c)$$

$$= \frac{n_b+n_c}{n_a+n_b+n_c}\cdot d(S_b, S_c) - \frac{n_c}{n_a+n_b+n_c}d(S_a, S_b) - \frac{n_b}{n_a+n_b+n_c}d(S_a, S_c)$$

This gives us the formula

$$d(S_{a+b}, S_c) - d(S_a, S_c) = \frac{n_b[d(S_b, S_c) - d(S_a, S_c)] + n_c[d(S_b, S_c) - d(S_a, S_b)]}{n_a+n_b+n_c}. \qquad \text{(A.3)}$$

The value of (3) is clearly now positive due to the assumptions made in lemma. This proofs the lemma. ■

Authors:

Timo Kaukoranta, PhD student in TUCS, in the computer science dept., university of Turku, Finland.
Address: Lemminkäisenkatu 14, 20520 Turku, Finland.
E-mail: tkaukora@cs.utu.fi

Pasi Fränti, research fellow in the computer science dept., university of Joensuu, Finland.
Address: P.O. box 111, 80101 Joensuu, Finland.
E-mail: franti@cs.joensuu.fi

Olli Nevalainen, professor in the computer science dept., university of Turku, Finland.
Address: Lemminkäisenkatu 14, 20520 Turku, Finland.
E-mail: olneva@cs.utu.fi