

# Computation of the $k^{\text{th}}$ Nearest Neighbor Estimate of Entropy of Molecules Using Parallel Processing

E. James Harner<sup>1</sup>, Jun Tan<sup>1</sup>, Shengqiao Li<sup>1,2</sup> and Harshinder Singh<sup>1,2</sup>

<sup>1</sup> Department of Statistics, West Virginia University, Morgantown, WV

<sup>2</sup> Health Effects Laboratory Division, NIOSH, Morgantown, WV

## Abstract

Entropy is a statistical measure of the random fluctuations in molecules and its estimation is important for investigating the stability of molecular conformations, for modeling the binding of ligands to proteins, and for studying issues relating to drug designs. Singh *et al.* (2003) introduced a nonparametric approach for estimating entropy using the  $k^{\text{th}}$  nearest neighbor distances between sample points, which extends the first nearest neighbor approach of Kozachenko and Leonenko (1987). Entropy of a molecule depends on random fluctuations in the internal coordinates. The high dimension of the internal coordinates and a large number of observations on the coordinates cause computational challenges for computing the  $k^{\text{th}}$  nearest neighbor distances required for obtaining the  $k^{\text{th}}$  nearest neighbor estimate of entropy of a large molecule.

We are experimenting with computing the  $k^{\text{th}}$  nearest neighbor distances for the sample points on a high-performance computer having a large number of parallel processors (nodes). On each processor, we use the ANN method (Arya; 1994) for computing  $k^{\text{th}}$  nearest neighbor distances. ANN builds a tree structure and searches for nearest neighbors on it. In order to search for nearest neighbors concurrently using high-performance computing with many processors, we have developed a program to duplicate the search trees on multiple processors (slave nodes) and each node computes the  $k^{\text{th}}$  nearest neighbor distances for part of the data. When the dimension of the coordinates and the number of observations becomes large, the gain in performance becomes obvious. Using the ANN method, the whole data set needs to be loaded into the memory to build the tree. Therefore, the maximum size of the data set that ANN can handle is limited by the size of the available memory.

We also developed a program that breaks the data set into parts and builds tree structures on the slave nodes. To find the  $k^{\text{th}}$  nearest neighbor of a point, the program searches all the trees concurrently and then merges all the results to get the  $k^{\text{th}}$  nearest neighbor for the data point. This program enhances the capability of handling extremely large data sets, such as those obtained using molecular dynamics simulation, and it improves the efficiency of computing the  $k^{\text{th}}$  nearest neighbor distances for the sample points.

## 1 Introduction

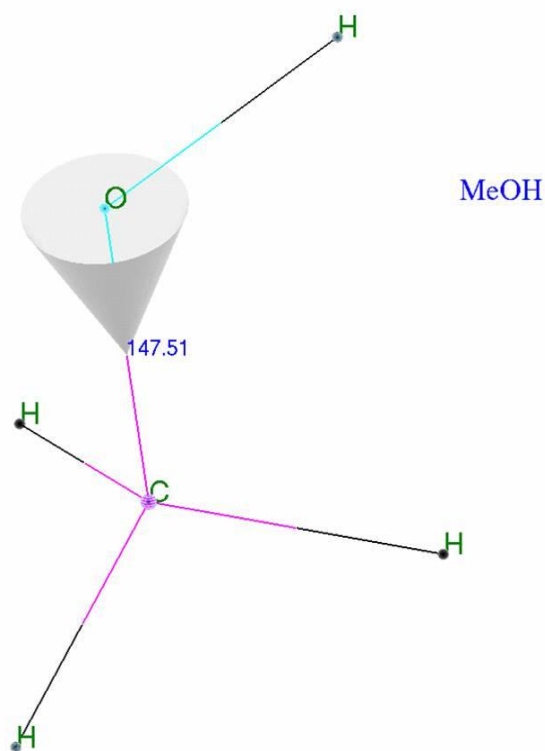
Researchers at NIOSH and WVU are modeling the random vibrations of molecules in order to study their properties and functions. We are focusing on estimating entropy, which measures the freedom of the molecular system to explore its available configuration space. Specifically, entropy evaluation is necessary to understand the stability of a conformation and the changes that take place in going from one conformation to another.

Our goal is to study proteins, which are biological molecules of primary importance to all living organisms. Humans have over 30,000 types of proteins, which

interact in complex ways. We are interested in protein misfolding—the cause of diseases such as Alzheimer’s disease, mad cow disease, cystic fibrosis, and some types of cancer. Understanding the stability of proteins is the key to finding drugs (small molecules) that stabilize the normally folded structure.

The entropy of a molecular conformation depends on the following coordinates: bond lengths, bond angles, and torsional (or dihedral) angles. Entropy is mainly determined by the fluctuations in torsional angles since bond lengths and angles are rather hard coordinates. Probabilistic modeling of the torsional angles of a molecular system is the basis for entropy evaluation.

Methanol is a simple molecular system which has only one torsional angle. The following figure illustrates the molecular dynamics of methanol:



In the classical molecular biology literature, the torsional angles are assumed to have a multivariate Gaussian (Normal) distribution, e.g., see Karplus and Kushik (1981). The configurational entropy is then given by

$$S_c = \frac{pk_B}{2} + \frac{k_B}{2} \ln[(2\pi)^p |\Sigma|],$$

where  $k_B$  is Boltzman’s constant.  $S_c$  is estimated by using the maximum likelihood estimate of the determinant of the variance-covariance matrix using molecular dynamics data on the torsional angles of the system. Misra *et al.* (2004) discussed a decision theoretic approach for estimating the entropy of the multivariate nor-

mal distribution and obtained the best affine equivariant estimator and some more efficient estimators of entropy.

Unfortunately, assuming a multivariate Gaussian distribution for torsional angles is not realistic if the probability density has more than one peak or if too much free energy is available, e.g., in gases. Demchuk and Singh (2001) and Singh *et al.* (2002) proposed circular (von Mises) probability models which are more appropriate for certain molecules, e.g., methanol. However, complex models, e.g., bio-molecules, are not modeled well by either the Gaussian or von Mises distributions. Hnizdo *et al.* (2003) proposed a Fourier series expansion approach for modeling torsional angles, which is computationally intensive even for a moderate number of angles. A more flexible approach is needed.

The focus in this paper is in estimating entropy using a nonparametric approach based on nearest neighbor distances. Only the nearest neighbor approach proposed by Kozachenko and Leonenko (1987), and extended by Singh *et al.* (2003), is practical for molecules with a large number of torsional angles.

## 2 Nearest Neighbor Estimate of Entropy

Let  $X_1, X_2, \dots, X_n$  be a random sample from a population with pdf  $f(x)$  of a random vector with  $p$  dimensions. Let  $R_{i,k}$  = Euclidean distance from  $X_i$  to its  $k^{\text{th}}$  closest neighbor. A natural estimate of  $f(X_i)$  is then given by:

$$\hat{f}(X_i) \frac{R_{i,k}^p \pi^{p/2}}{\Gamma(p/2 + 1)} = \frac{k}{n},$$

or

$$\hat{f}(X_i) = \frac{k\Gamma(p/2 + 1)}{nR_{i,k}^p \pi^{p/2}}, i = 1, 2, 3, \dots, n.$$

Thus, a reasonable estimator of entropy of the population with pdf  $f(x)$  is given by:

$$\hat{G}_k = -\frac{1}{n} \sum_{i=1}^n \ln[\hat{f}(X_i)] = -\frac{1}{n} \sum_{i=1}^n T_i,$$

where

$$T_i = \ln\left(\frac{k\Gamma(p/2 + 1)}{nR_{i,k}^p \pi^{p/2}}\right), i = 1, 2, 3, \dots, n.$$

The asymptotic mean of the estimator is given by

$$\lim_{n \rightarrow \infty} E[\hat{G}_k] = L_{k-1} - \gamma - \ln k + E[-\ln f(X)],$$

where

$$L_0 = 0; L_j = \sum_{i=1}^j \frac{1}{i}, j \geq 1; \gamma = 0.5772.$$

Thus, the estimator  $\hat{G}_k$  is asymptotically biased. We consider the modified estimator:

$$\hat{H}_k = \hat{G}_k - L_{k-1} + \gamma + \ln k.$$

In terms of the  $k^{\text{th}}$  nearest neighbor distances,

$$\hat{H}_k = \frac{p}{n} \sum_{i=1}^n \ln R_{i,k} + \ln \frac{\pi^{p/2}}{\Gamma(p/2 + 1)} + \gamma - L_{k-1} + \ln n$$

Properties:  $\hat{H}_k$  is asymptotically unbiased;  $\hat{H}_k$  is a consistent estimator of the entropy,  $E[-\ln(f(X))]$ ;  $\hat{H}_k$  reduces to the estimator proposed by Kozachenko and Leonenko (1987) for  $k = 1$ .

For sample sizes of 10, 25, 50 and for some standard distributions (using simulations) we studied the performance of  $\hat{H}_k$  for various  $k$ 's under the mean square criterion. The estimator based on  $k = 4$  performs well. These estimators were used to evaluate entropy of the methanol and diethyl ether molecules.

Searching for the  $k^{\text{th}}$  nearest neighbor in large data sets (in terms of both the number of observations and dimensions) is computationally intensive and hence time consuming. ANN (Arya, *et al.*, 1998; <http://www.cs.umd.edu/mount/ANN/>) is an experimental algorithm that can greatly speed up nearest neighbor searching. It is a C++ library for exact and approximate  $k^{\text{th}}$  nearest neighbor searching in  $p$ -dimensional space. The basic idea of ANN is to build a tree structure based on the spacial locations of the data points. When a query point is given, instead of computing the distance between this point and each data point in the data set and picking the smallest, ANN first searches the tree for a small sub-set of data points near the query point, computes the distances (any Minkowski metric), and picks the  $k^{\text{th}}$  smallest one. In theory, given any positive real  $\epsilon$  and integer  $k \geq 1$ , the  $(1 + \epsilon)$  approximations to the  $k^{\text{th}}$  nearest neighbors of a single query point can be computed in  $O(kp \log(n))$  time. However, when dealing with data sets with large number of observations and high dimensions, ANN suffers from two problems:

1. The performance drops dramatically when the data set gets large. After the size of the data set exceeds a certain number, the performance of ANN drops much faster than what the theory shows. For example, on an Apple Cube (450MHZ G4, 512MB Memory), ANN took 25 minutes to search a  $24 \times 200000$  data set. In contrast, it took more than a day to search a  $24 \times 300000$  data set.
2. ANN needs to load the whole data set into memory in order to build the search tree. Thus, the maximum data set that ANN can handle is limited by the memory size of the machine on which ANN runs.

Using high performance computational techniques, we developed two programs based on ANN to overcome the difficulties mentioned above. The basic idea is to use multiple nodes to do nearest neighbor searching concurrently to improve the performance. The MPI library was used to handle the communications between nodes.

### 3 Parallelizing ANN I: Dividing the Query Points

For any given data set, the nearest neighbors of different querying points are independent. Therefore, if there are multiple computational nodes available, it is natural to divide the data into groups and assign them to the computation nodes so that they can be searched concurrently. For each querying point, the time spent on searching for its nearest neighbor does not change. The overall performance is improved if the extra time used for transferring data to the computation nodes is smaller than the time saved because of the concurrency.

Suppose the sample size is  $n$  and  $m$  slaves are available. Let  $l = n/m$ . Then, the steps for Algorithm I are:

1. build the tree for the entire data set on each slave concurrently;

Number of Nodes	Time Used (in seconds)
4	122
8	104
16	98

Table 1: Nearest searching on a  $200,000 \times 24$  data set using algorithm I

Number of Nodes	Time Used (in seconds)
4	17769
8	9003
16	4521
32	2383

Table 2: Nearest searching on a  $300,000 \times 24$  data set using algorithm I

- find the nearest neighbors for data points  $i + j * m$  ( $0 \leq j < l$ ) for slave  $i$  ( $1 \leq i \leq m$ );
- send the search results back to the master for calculating the final result.

For example, if  $m = 4$ , the nearest neighbors of data points 1, 5, 9, 13, ... will be searched on slave 1; 2, 6, 10, 14, ... will be found on slave 2, etc.

This algorithm has a simple computational model, which minimizes the communication between the master and slaves. The nodes execute independently, i.e., they don't need to wait for the results from other nodes to continue. We used this program to search for the 10 nearest neighbors from several data sets of different sizes<sup>1</sup> on the Pittsburgh supercomputer Lemieux. Some of the results are shown in Table 1 and Table 2. The performance improved as the number of nodes increased. In addition, the relative performance improvement was greater for the larger data set.

Although this algorithm improved the performance, it suffers from some obvious problems:

- The data set size is limited by the slave with the least amount of memory since the whole data set must be loaded into the memory of each node.
- Each slave node reads the whole data set initially, which can put pressure on both the network and file system. (Also, as the number of nodes increases, the required bandwidth increases, particularly if the data set is large.)

Since the data set from a molecular dynamics simulation can be extremely large (a data point is collected every  $10^{-9}$  second), an alternative algorithm—not limited by the memory of an individual slave—is needed.

## 4 Parallel ANN Algorithm II: Partitioning the Search Data Set

The second program uses a different strategy to divide nearest neighbor searching into smaller independent parts. It is built using the fact that the nearest neighbor

<sup>1</sup>We used the same data set as both querying data set and data set

of a data point is the smallest of the nearest neighbors of all the parts of the data set. More precisely, given the data set  $D$ , a querying data point  $q$ , and the partition  $D_1, D_2, \dots, D_m \subset D$ , where  $\cup_{i=1}^m (D_i) = D$ , let  $r$  be the nearest neighbor of  $q$  in  $D$  and  $r_i$  be the nearest neighbor of  $q$  in  $D_i$ . Then  $r = \min(r_1, r_2, \dots, r_m)$ . The algorithm first breaks the data set into parts and assigns each part to a node. For a given querying point, the algorithm searches for its nearest neighbors in all the parts simultaneously and then it returns the nearest neighbors, i.e., points with the smallest distance to the querying point for each of the parts.

As mentioned in the last section, the complexity of tree-based nearest neighbor searching is  $O(kpn \log(n))$  for all query points. Therefore, the complexity of algorithm II is  $O(kpn \log(n) - kpn \log(m) + kmn)$  for all query points, where  $m$  is the number of nodes. If  $kpn \log(m) < kmn$ , then the performance of this algorithm will increase if the time spent on communications is less than the time saved because of concurrency. Through our experience with using ANN, we learned that the performance of ANN was worse than the theoretical complexity formula shows when the size of data set is large. Therefore, we expect a performance increase when this program is used for large data sets. Another attractive feature of this algorithm is that the maximum data set is not bound to the available memory of each node. Since each node only holds a part of the data set, we can add more nodes to make each part small enough to be handled efficiently.

However, unlike what people would usually expect, adding more nodes will not always increase the performance of this algorithm. Consider two extreme situations:

1. If only one node is available, the program is nothing but a wrapper that simply calls ANN resulting in overall complexity of  $O(kpn \log(n))$ .
2. If  $n$  nodes are available, where  $n$  is the size of the data set, there is only one data point for each node and the algorithm reduces to a brute-force nearest neighbor search resulting in overall complexity of  $O(kpn^2)$ , which is much slower than the one-node case.

Therefore, if this algorithm increases the performance as the number of nodes increases, at least initially, there will be a optimal number of nodes  $m < n$ , which will give the best performance.

In this program, there are two types of nodes: one master node and one or more slave nodes. The master node controls the running of the program and generates the final result. In contrast, slave nodes receive instructions from the master node and do the nearest neighbor search. The following shows the tasks performed on master node and slave nodes, respectively:

Master node:

1. Based on the size of data set and the number of available slave nodes, the master node determines the actual number of slave nodes that will be used to do the computation. It is difficult to find the optimal number of slaves. Currently we avoid the situation in which small groups of data are scattered to a large number of nodes, which is known to be very inefficient. Once the number of slave nodes is determined, the size of the data groups that each slave will hold is computed.
2. The master node reads through the data set and determines which slave node should hold each point, which is then sent to the appropriate node.
3. The master node broadcasts each query point to all slave nodes and gathers the results from all slave nodes. The master node sorts the results from the

Number of Nodes	Time (build tree)	Time (search)
2	64	235
4	63	242
6	63	242
8	63	219
10	63	199
12	63	204
14	63	208
16	63	208

Table 3: Nearest searching on a  $200,000 \times 24$  data set using algorithm II

Number of Nodes	Time (build tree)	Time (search)
8	76	8738
10	77	6114
12	77	4352
14	77	3172
16	76	2733
18	76	2446
20	83	2140
22	77	1946
24	76	1849
26	77	1634
32	75	1315
48	75	1300
64	75	1237

Table 4: Nearest searching on a  $300,000 \times 24$  data set using algorithm II

slave nodes using a merging algorithm. The  $k$  smallest values are the  $k$  nearest neighbors.

Slave node:

1. The slave node receives data from the master node. Using ANN, a tree structure is built based on received data.
2. The slave node receives a querying data point from the master node. Using ANN, the tree structure is searched for the  $k$  nearest neighbors. The result is sent back to the master. This step will repeat until there is no more querying data.

We searched for the 10 nearest neighbors on several data sets using the above algorithms on the Pittsburgh supercomputer Lemieux. From Table 3, we see that algorithm II is much slower than algorithm I on the smaller data set. When a data set is (relatively) small, the total time spent on nearest neighbor computing is small. The concurrent search does not save much time and the time spent on communications and merging the results from slave nodes is relatively significant. Note that using 10 nodes is optimal, i.e., both a smaller number of nodes and a larger number increase the search time.

Table 4 shows the benefit of using algorithm II when the data set is (relatively) large and a reasonable number of slave nodes are available. Notice that performance is still improving at  $m = 64$  nodes, i.e., the optimal number of nodes is  $m \geq 64$ .

## 5 Conclusion

Computing the entropy for moderately sized molecules, such as for peptides, is complex if the number of MD simulations is large. Parallel versions of the ANN algorithm greatly improve the running time for these macromolecules relative to the brute-force method and the basic ANN algorithm.

Using high-performance computing technology, we developed two parallel algorithms to increase the performance of nearest neighbor searching. Algorithm I is better for moderately sized  $n$ , whereas Algorithm II is preferred for large  $n$ . The cross-over point is being explored, but it may not be possible to find which algorithm is better in all possible situations since many factors are involved, such as the network speed. The first algorithm is simple, but the size of the data set it can handle is limited by the node with the least available memory. The second algorithm is more complicated. Although more time is spent controlling the nodes and transferring the results back and forth, the performance of this algorithm is better when the size of the data set is large. The largest data set that this algorithm can handle depends on the total available memory of all the nodes. Performance is also affected by the number of nodes used.

## REFERENCES

- Arya S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A.Y. (1998) An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, **45**, 891-923.
- Demchuk, E., and Singh, H. (2001) Statistical thermodynamics of hindered rotation from computer simulations. *Molecular Physics*, **99**, 627-636.
- Hnizdo, V., Fedorowicz, A., Singh, H. and Demchuk, E. (2003) Statistical thermodynamics of internal rotation in a hindering potential of mean force obtained from computer simulations. *Journal of Computational Chemistry*, **24**, 1172-1183.
- Karplus, M., and Kushik, J.N. (1981) Method for estimating the configurational entropy of macromolecules. *Macromolecules*, **14**, 325-332.
- Kozachenko, L. F., and Leonenko, N. N. (1987) Sample estimates of entropy of a random vector. *Problems of Information Transmission*, **23**, 95-101.
- Misra, N., Singh, H. and Demchuk, E. (2004). Estimation of the entropy of a multivariate normal distribution. To appear in the *Journal of Multivariate Analysis*.
- Singh, H., Demchuk, E., Hnizdo, V. and Sharp D. (2002) Stochastic modeling of rotational degrees of freedom in molecules. *Proceedings of the Hawaii International Conference on Statistics*, 1-9.
- Singh, H., Hnizdo, V., and Demchuk, E. (2002) Probabilistic model for two dependent circular variables. *Biometrika*, **89**, 719-723.

Singh, H., Misra, N., Hnizdo, V., Fedorowicz, A., and Demchuk, E. (2003).  
Nearest neighbor estimates of entropy. *American Journal of Mathematical and  
Management Sciences*, **23**, 301–321.