

COMPUTATIONAL CHALLENGES IN COMPUTING NEAREST NEIGHBOR ESTIMATES OF ENTROPY FOR LARGE MOLECULES

E. James Harner¹, Harshinder Singh^{1,2}, Shenggiao Li^{1,2} and Jun Tan¹

¹ Department of Statistics, West Virginia University, Morgantown 26506

² National Institute for Occupational Safety and Health, Morgantown 26505

ABSTRACT

Entropy is a statistical thermodynamic property of molecules; its evaluation is important for studying the properties of biological molecules (such as peptides, proteins, and DNA molecules) and chemical molecules. Entropy evaluation is also important in drug designs and for investigating the effect of toxins on human skin. The entropy of a molecule depends mainly on random fluctuations in its torsional (also called the rotational or dihedral) angles. The traditional approach assumed a multivariate normal distribution for the torsional angles of large molecules (Karplus and Kushik, *Macromolecules*, 1981). However, the assumption of normality is not valid in many situations, particularly when there are large fluctuations in the torsional angles.

Demchuk and Singh (2001) introduced a circular probability approach to modeling torsional angles in molecules and illustrated the modeling of the torsional angle of methanol using von Mises distributions. A bathtub shaped probability distribution was derived for the potential energy of the methanol molecule. Singh *et al.* (2002) introduced a bivariate circular model, which is a natural torus version of the bivariate normal distribution to which it reduces when the fluctuations in the angles are small. The marginal distributions are symmetric and are either unimodal or bimodal. This model was used for modeling two angles of a pentapeptide. In general, the torsional angles can have arbitrary shapes and macromolecules have a large number of torsional angles, which are interdependent. Thus a nonparametric approach appears to be a natural choice for entropy estimation of large molecules. However, entropy evaluation using histogram and kernel density estimates also has problems in high dimensions.

Estimates of entropy based on nearest neighbor distances between sample points (Kozachenko and Leonenko (1987)) and estimates of entropy based on k^{th} nearest neighbor distances (Singh *et al.* (2003)) offer hope for estimating entropy for large molecules. However, evaluating nearest neighbor distances is computationally challenging when the number of torsional angles is large and the data obtained using molecular dynamic simulations on the molecule is huge. We discuss computational approaches for obtaining estimates of entropy based on nearest neighbor distances. Our approaches use Rmpi to parallelize n^2 and $n \ln n$ k^{th} nearest neighbor algorithms, where n is the number of dynamic simulations. We illustrate this approach using data on torsional angles of some large molecules.

1. INTRODUCTION

Let \mathbf{X} be a p -dimensional random vector with probability density function $f(\mathbf{x})$. The entropy of probability distribution of \mathbf{X} is defined as

$$H(f) = E[-\ln f(\mathbf{X})]. \quad (1)$$

$H(f)$ measures the uncertainty associated with the distribution of \mathbf{X} . In the molecular sciences, the entropy associated with random vibrations in a molecular conformation is an important statistical thermodynamic property for studying the functions of biological molecules such as proteins, peptides, DNA and for studying the properties of chemical molecules such as toxins. Accurate prediction of molecular conformation stability and binding affinity requires a precise method of estimating conformational entropy. Estimation of conformational entropy is also important in studying problems relating to protein folding and binding of peptic ligands to proteins having applications to the field of drug designs.

The entropy of a molecular conformation depends on random fluctuations in its internal coordinates. These internal coordinates are bond lengths, bond angles and torsional angles. Fluctuations in bond lengths and bond angles are relatively small and thus the entropy of a molecular conformation is mainly determined by random fluctuations in its torsional angles.

Let $\Theta_1, \Theta_2, \dots, \Theta_p$ denote p torsional angles of a molecule. The traditional approach to estimate entropy for large molecules is based on the assumption of multivariate normal distribution of the p torsional angles. The probability density function of the torsional angles is assumed to be the multivariate normal density given by

$$f(\theta_1, \theta_2, \dots, \theta_p) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\theta - \mu)' \Sigma^{-1} (\theta - \mu)}, \quad (2)$$

where μ is the mean vector and Σ is the dispersion matrix of the distribution. Under this assumption of multivariate normality, the entropy of the distribution of torsional angles is given by

$$H = E[-\ln f(\Theta)] = \frac{p}{2}(\ln 2\pi + 1) + \frac{1}{2} \ln |\Sigma|. \quad (3)$$

Thus the entropy can be estimated by using an estimate of $\ln |\Sigma|$. A common approach in molecular sciences is to estimate $\ln |\Sigma|$ by $\ln |\hat{\Sigma}|$ where $\hat{\Sigma}$ is the sample variance-covariance matrix. Karplus and Kushik (1981) estimated entropy differences between two configurations of butane and decaglycine using the this estimator computed from data on internal coordinates obtained from molecular dynamic simulations. Misra *et al.* (2003) obtained best affine equivariant estimator of $\ln |\Sigma|$ and interestingly this turned out to be the minimum variance unbiased estimator and gives significant improvement over the maximum likelihood estimator in high dimensions. They proposed further improvement of the best affine equivariant estimator using Stein and Brewster Zidek methods.

The assumption of multivariate normality for torsional angles greatly helps in the estimation of entropy for large molecules. However, this assumption does not necessarily hold for torsional angles, particularly when there is more than one peak in the marginal distributions or when there are large fluctuations in the angles. Demchuk and Singh (2001) proposed circular probability approaches to modeling torsional angles in molecules. As an illustration, they modeled the torsional angle of methanol using a tri-mode von Mises distribution and derived a bathtub shaped

probability distribution for the torsional potential energy of methanol. The three-mode von Mises distribution provided an excellent fit to the torsional angle data obtained using molecular dynamic simulations. They obtained an expression for the entropy in terms of the concentration parameter of the von Mises distribution, which can be estimated using maximum likelihood. In general, molecules have more than one torsional angles. Singh *et al.* (2002) proposed a bivariate circular distribution for modeling two torsional angles of a molecule. This model is a five parameter natural torus analogue of the bivariate normal distribution to which it reduces when the fluctuations in the angles are small. The conditional distributions of the model are von Mises. The marginal distributions are symmetrical and are either unimodal or bimodal depending on the configuration of parameters. The proposed bivariate distribution was used to model two angles of a pentapeptide.

Torsional angles of molecules can have arbitrary shape. To model an arbitrary distribution of torsional angles in molecules, Singh *et al.* (2002) and Hnizdo *et al.* (2003) proposed a Fourier series expansion of the potential function approach. The parameters of the approximating finite terms in Fourier series are estimated by using the maximum likelihood estimation procedure. This approach was used to model some molecules having either one or two torsional angles. However, this approach breaks down when we go to higher dimensions as a large number of terms are required in the Fourier series to get a good fit. Thus a nonparametric approach to estimation of entropy in high dimensions is necessary. In Section 2 of this paper we discuss an entropy estimation approach based on nearest neighbor distances in sample points introduced by Kozachenko and Leonenko (1987) and a generalization of this approach based on k^{th} nearest neighbor distances introduced by Singh *et al.* (2003). In Section 3 we discuss the computational challenges in the computation of k^{th} nearest neighbor distances required for obtaining estimate of entropy.

2. NEAREST NEIGHBOR ESTIMATES OF ENTROPY

Let X_1, X_2, \dots, X_n be n independent copies of a p -dimensional random variable X and let $H = E[-\ln f(X)]$ denote the entropy of X . Let $d(x, y)$ denote the Euclidean distance between two points x and y in the p -dimensional space. Let $\rho_i = \min \{d(X_i, X_j), j \in \{1, 2, \dots, n\} - \{i\}\}$ denote the distance of X_i to its nearest neighbor, $i = 1, 2, \dots, n$. Kozachenko and Leonenko proposed the following estimator H_n for estimating H based on the nearest neighbor distances

$$H_n = \frac{p}{n} \sum_{i=1}^n \ln \rho_i + \ln \left[\frac{\pi^{p/2}}{\Gamma(\frac{p}{2} + 1)} \right] + \gamma + \ln(n - 1). \quad (4)$$

where $\Gamma(\cdot)$ is the gamma function and γ is Euler's constant with an approximate value of 0.5772. They established asymptotic unbiasedness and consistency of H_n .

For any fixed positive integer k , let

$$R_{i,k,n} = \text{Euclidean distance from } X_i \text{ to its } k^{th} \text{ closest neighbor.} \quad (5)$$

Singh *et al.* (2003) proposed an estimator of H based on the k^{th} nearest neighbor distances defined above. The estimator proposed by them is given by

$$\hat{H}_k^{(n)} = \frac{p}{n} \sum_{i=1}^n \ln R_{i,k,n} + \ln \left[\frac{\pi^{p/2}}{\Gamma(p/2 + 1)} \right] + \gamma - L_{k-1} + \ln n, \quad (6)$$

where $L_j = \sum_{i=1}^j \frac{1}{i}$. They established the asymptotic unbiasedness and consistency of the proposed estimator. Estimators were also compared for different values of k with respect to the mean squared error criterion for some standard distributions using statistical simulations. These estimators were used to estimate the entropy of methanol having one torsional angle and entropy of diethyl ether having four torsional angles. Note that for $k = 1$, the estimator given by (6) is almost the same as the estimator of Kozachenko and Leonenko given by (4).

In order to compute $H_k^{(n)}$, the computations of k^{th} nearest neighbor distances $R_{i,k,n}$ are required for all the sample points. Computations of these distances pose a challenging problem in high dimensions. In Section 3 we discuss some algorithms/methods for obtaining these distances and describe parallel computing approaches for computing the distances in high dimensions for large data sets.

3. ALGORITHMS FOR COMPUTING k^{th} NEAREST NEIGHBOR DISTANCES

The principal challenge in computing the k^{th} nearest neighbor is that the running time and space, required by even efficient algorithms, is exponential in p . We used ANN (Approximate Nearest Neighbor) to compute exact nearest neighbors in various dimensional spaces (<http://www.cs.umd.edu/mount/ANN/>). In order to allow large problems to be solved, we modified ANN to allow computations to be spread over a cluster using MPI (Message Passing Interface).

3.1 Cluster Computing

Cluster computing is a form of parallel computing that can be implemented efficiently and inexpensively. However, parallel programmers face a number of issues: data partitioning, load balancing, deadlocks, program flow, etc. Tools are emerging, e.g., Rmpi, to assist the statistical programmer, but challenges remain.

Parallel processes must have some way to communicate, e.g., by using some type of message passing. The two common message-passing specifications are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). PVM is older, but it is still widely used due to its flexibility and fault tolerance. Also, nodes can be dynamically removed from and added to the cluster. However, MPI has a more complete set of communication routines and increasingly is being used for high-performance computing.

Interfaces to both PVM and Rmpi are available in R. Rpvms was developed by Na Li and Tony Rossini and Rmpi (<http://www.stats.uwo.ca/faculty/yu/Rmpi/>) was developed by Hao Yu. Na LI also developed an interface to the Scalable Parallel Random Number Generator (sprng) called rsprng (<http://gauss.est.ufpr.br/CRAN/src/contrib/PACKAGES.html>). Parallel random number generation is essential for studies requiring Monte Carlo methods. MPI and Rmpi were used in this paper to compute k^{th} nearest neighbor distances, which are used for estimating entropy.

Rmpi is a wrapper to MPI, which is a standardized and portable message-passing system. Rmpi requires the LAM implementation (<http://www.lam-mpi.org/>) of MPI. LAM is an open-source implementation of the MPI specification (<http://www.mpi-forum.org/>) intended for both production and research. It also supports GRID computing (clusters of clusters) and SMP clusters.

LAM/MPI and R/Rmpi must be installed on the master and on each of the slaves. The master controls the execution, but consumes few resources. The slaves do the actual work.

3.2 Brute-force Algorithms for Computing the k^{th} Nearest Neighbor

Brute-force algorithms for computing the k^{th} nearest neighbor are not practical except for relatively small problems. However, these algorithms are simple and provide a way of learning how to implement parallel computing using Rmpi.

Consider the following R code to compute entropy:

```
knnb<- function(x, k)
{
  if(is.matrix(x)){
    n<- nrow(x); p<- ncol(x);
  }
  else{
    n<-length(x); p<- 1;
  }
  k2<- k+1;
  R<-matrix(0, nrow=n, ncol=k);
  for(i in 1:n){
    if(p==1){
      R[i, ]<-sort( abs(x[i]-x), method="quick")[2:k2];
    }
    else{
      R[i, ]<-sort(rowSums(sweep(x, 2, x[i,])^2), method="quick")[2:k2];
    }
  }

  G<- p*colMeans(log(R)) + p/2*log(pi)- log(gamma(p/2+1))
    + log(n)-log(1:k);

  lk<-function(k)
  {
    if(k==0) {
      return (0);
    }
    sum(1/1:k);
  }
  EulerGamma<- 0.577216;
  H<- G - apply(rbind(0:(k-1)), 2, lk) + EulerGamma + log(1:k);
  return (list(G=G, H=H));
}
```

This is a simple algorithm in which distances are computed between each data point and each of the remaining data points. A quick sort is then done and the first k distances are selected for each data point. This algorithm requires a complete sort for each of the n data points, which can quickly become a limiting factor whatever the dimension p .

This function is invoked by:

```
> knnb(rnorm(1000),10)
$G
[1] 0.807352 1.180578 1.259807 1.308433 1.327055 1.333878
     1.342139 1.350061 1.356582 1.368295
```

```
$H
[1] 1.384552 1.450925 1.435619 1.438594 1.430360 1.419505
     1.415249 1.413846 1.413149 1.419112
```

The unadjusted, G , and adjusted, H , estimates of entropy are given up to the k^{th} nearest neighbor, which is 10 here. The estimates seem to have stabilized by $k = 4$.

This code can be parallelized using Rmpi, as discussed above. The R code for the master processor is:

```
library(Rmpi);
mpi.spawn.Rslaves(hosts=c(0, 0, 1, 1));

mpi.remote.exec(load, file = "Projects/comp/RNnbMpi.Rdata");

k <- 10;
G <- mpi.remote.exec(knnb.part, "Projects/comp/dim14_2k.dat", c(2000, 14), k, 1);
n <- G[1, 1]; p <- G[2, 1]; G <- p * rowSums(G) / n;

G <- G[3:length(G)] + p / 2 * log(pi) - log(gamma(p / 2 + 1)) + log(n) - log(1:k);

lk <- function(k){
  if(k == 0){
    return(0);
  }
  sum(1/1:k);
}

EulerGamma <- 0.577216;
H <- G - apply(rbind(0:(k-1)), 2, lk) + EulerGamma + log(1:k);
```

The `mpi.spawn.Rslaves` function creates slaves on remote processors. In this case, it creates a slave on each of the two processors on node 0 and on each of the two processors on node 1, i.e., 4 slaves are created. The `mpi.remote.exec` function initiates execution on the remote slaves. The first call loads the slave R code (`knnb.part`) and the second call loads the data (in this case the first 2000 rows of the 14-dimensional data).

The code executed on each of the slaves is:

```
knnb.part <- function(data.file = "", dimension = NULL, k = 1, comm = 1){
  x <- scan(data.file);
  dim(x) <- dimension;

  size <- mpi.comm.size(comm);
  rank <- mpi.comm.rank(comm);

  if(is.matrix(x)){
    n <- nrow(x); p <- ncol(x);
  }
  else{
    n <- length(x); p <- 1;
  }
}
```

```

    k2 <- k + 1;
    R <- matrix(1, nrow = n, ncol = k);

i <- rank;

while(i <= n){
  if(p == 1){
    R[i,] <- sort(abs(x[i] - x), method = "quick")[2:k2];
  }
  else{
    R[i,] <- sort(rowSums(sweep(x, 2, x[i,])^2), method = "quick")[2:k2];
  }
  i <- i + size - 1;
}

R <- colSums(log(R));
return(c(n, p, R));
}

```

The key here is that each slave only does part of the calculations, i.e, those determined by size, which is the number of processes retrieved by `mpi.comm.size` (including the master process). The starting observation is the rank position of the slave process (starting at 1 since the master process reserves position 0) determined by `mpi.comm.rank`. For example, the first slave (out of 4 above) would compute the nearest neighbors (the first k) for observations 1, 5, 9, ..., the second slave process would compute the nearest neighbors for 2, 6, 10, ..., etc. The nearest neighbors for each of these observations are returned to the master, which then combines them to compute the entropy.

3.3 ANN Algorithms for Computing the k^{th} Nearest Neighbor

ANN is a C++ library for exact and approximate nearest neighbor searching in a p -dimensional space. The n data points in the p -dimensional space are preprocessed into a data structure for efficient searching. The two data structures currently supported in ANN are the kd-trees and the box-decomposition (bd)-trees (see the ANN programming manual at:

<http://www.cs.umd.edu/~mount/ANN/>). The kd-tree recursively subdivided the space into regions called cells. It is the default data structure in ANN and was used for finding nearest neighbors in this paper. If data points are highly clustered, bd-tree structures may be preferred over kd-trees. ANN allows distances to be computed using any Minkowski metric.

The running time (and space) for nearest neighbor searching increases exponentially with the dimension. This becomes a major problem for computing the entropy of macromolecules, e.g., proteins, in which the number of dihedral angles is large. ANN allows the user to do approximate nearest neighbor queries. This option was not used in this paper since all calculations were done on relatively small problems.

Various strategies are available for parallelizing ANN. A simple way is to build the whole tree structure on each slave and then split the querying tasks among the slaves.

Suppose the sample size is n and m slaves are available. Let $l = n/m$. More specifically, the details are:

1. build the tree on each slave concurrently;
2. find the nearest neighbors for data samples $i + j * m$ ($0 \leq j < l$) for slave i ($0 \leq i < m$); (for example, if $i = 0$ and $m = 4$, the nearest neighbors of data samples 0, 4, 8, 12, ... will be searched on slave 0)
3. send the results of searching back to the master for calculating the final result.

The advantages of this solution are that it:

1. fully utilizes the ability of each slave;
2. minimizes the communication between the master and slaves.

It is possible to call the ANN library from within R and to use the Rmpi interface to MPI. However, this involves foreign function calls and some efficiency is lost. As a result, the interface to MPI was programmed in C++. The code is not presented here, but the general idea is outlined above.

4. RESULTS

Preliminary results are available for comparing the brute-force method without MPI, the brute-force method with MPI, and the ANN algorithm with MPI.

Table 1 reports the performance of the brute-force algorithm without MPI using a single processor on a dual G4 computer, which serves as a reference. For any given dimension, the computing time (in seconds) increases $O(n^2)$ with the sample size n , allowing for suitable overhead for small n . Likewise, for any given sample size, the computing time increases exponentially with the dimension p .

Sample Size	Dimension	Seconds
1000	1	1.22
2000	1	2.77
10000	1	42.46
1000	7	5.27
2000	7	17.94
10000	7	717.06
1000	14	8.75
2000	14	32.87
10000	14	1518.88

Table 1: Brute-force Algorithm without MPI on One G4 Processor

When 2 slaves are run on a dual-processor CPU (Table 2), the time is reduced (but not by half) if the sample size is large enough to overcome the overhead of Rmpi. Running 4 slaves on two dual-processor CPUs (Table 3) further reduces the computing time for large sample sizes. However, molecular dynamics simulation often results in millions of observations and it would be then be infeasible to use the brute-force method whatever the number of dihedral angles.

ANN reduces the computing time dramatically over the brute-force method (Tables 4 and 5). The computing time for the ANN algorithms is $O(n \log n)$. However, increasing the number of slaves only marginally, if at all, reduces the computing time (for large sample size).

Sample Size	Dimension	Seconds
1000	1	2.11
2000	1	2.99
10000	1	28.27
1000	7	4.65
2000	7	12.85
10000	7	464.11
1000	14	7.05
2000	14	22.35
10000	14	991.65

Table 2: Brute-force Algorithm with MPI: 2 Slaves on One 2-CPU G4 Computer

Sample Size	Dimension	Seconds
1000	1	2.04
2000	1	2.48
10000	1	16.78
1000	7	3.35
2000	7	7.39
10000	7	231.74
1000	14	4.64
2000	14	12.44
10000	14	499.46

Table 3: Brute-force Algorithm with MPI: 4 Slaves on Two 2-CPU G4 Computers

Sample Size	Dimension	Seconds
1000	1	0.61
2000	1	0.64
10000	1	0.88
1000	7	0.73
2000	7	0.88
10000	7	2.22
1000	14	0.90
2000	14	1.31
10000	14	5.82

Table 4: ANN Algorithm with MPI: 2 Slaves on One 2-CPU Computer

5. DISCUSSION

Computing entropy for moderately sized molecules, such as peptides, is feasible depending on the number of MD simulations. The ANN algorithms greatly improve

Sample Size	Dimension	Seconds
1000	1	0.67
2000	1	0.74
10000	1	0.91
1000	7	0.79
2000	7	0.96
10000	7	2.11
1000	14	0.98
2000	14	1.30
10000	14	4.72

Table 5: ANN Algorithm with MPI: 4 Slaves on Two 2-CPU Computers

the run time for these macromolecules relative to the brute-force method.

However, it is less clear how much cluster computing improves performance for ANN algorithms. Although the preliminary results above show little improvement, this is likely to change when the number of slaves is increased for large n and/or p . For example, on a data set with 1.44M observations, the ANN kd-tree algorithm took 727.52 seconds to run with 2 slaves and 435.12 seconds to run with 4 slaves (using the same computers as the above tests). These results look promising, but more testing must be done before definitive answers can be given on the benefits of using MPI for large n and/or high dimensionality p .

REFERENCES

- 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. (2003). Estimation of the entropy of a multivariate normal distribution. Submitted for publication.
- 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. To appear in the *American Journal of Mathematical and Management Sciences*.

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.