



香港城市大學
City University of Hong Kong

專業 創新 胸懷全球
Professional · Creative
For The World

CityU Scholars

Calibur

A tool for clustering large numbers of protein decoys

Li, Shuai C.; Ng, Yen K.

Published in:

BMC Bioinformatics

Published: 13/01/2010

Document Version:

Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

License:

CC BY

Publication record in CityU Scholars:

[Go to record](#)

Published version (DOI):

[10.1186/1471-2105-11-25](https://doi.org/10.1186/1471-2105-11-25)

Publication details:

Li, S. C., & Ng, Y. K. (2010). Calibur: A tool for clustering large numbers of protein decoys. *BMC Bioinformatics*, 11, Article 25. <https://doi.org/10.1186/1471-2105-11-25>

Citing this paper

Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights

Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission

Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy

Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

SOFTWARE

Open Access

Calibur: a tool for clustering large numbers of protein decoys

Shuai Cheng Li^{1*}, Yen Kaow Ng²

Abstract

Background: Ab initio protein structure prediction methods generate numerous structural candidates, which are referred to as decoys. The decoy with the most number of neighbors of up to a threshold distance is typically identified as the most representative decoy. However, the clustering of decoys needed for this criterion involves computations with runtimes that are at best quadratic in the number of decoys. As a result currently there is no tool that is designed to exactly cluster very large numbers of decoys, thus creating a bottleneck in the analysis.

Results: Using three strategies aimed at enhancing performance (proximate decoys organization, preliminary screening via lower and upper bounds, outliers filtering) we designed and implemented a software tool for clustering decoys called Calibur. We show empirical results indicating the effectiveness of each of the strategies employed. The strategies are further fine-tuned according to their effectiveness.

Calibur demonstrated the ability to scale well with respect to increases in the number of decoys. For a sample size of approximately 30 thousand decoys, Calibur completed the analysis in one third of the time required when the strategies are not used.

For practical use Calibur is able to automatically discover from the input decoys a suitable threshold distance for clustering. Several methods for this discovery are implemented in Calibur, where by default a very fast one is used. Using the default method Calibur reported relatively good decoys in our tests.

Conclusions: Calibur's ability to handle very large protein decoy sets makes it a useful tool for clustering decoys in ab initio protein structure prediction. As the number of decoys generated in these methods increases, we believe Calibur will come in important for progress in the field.

Background

In ab initio protein structure predictions [1-5], it is often the case that a large set of candidates (called decoys) is generated, and one is required to select a single (or a small selection of) best candidate(s) from the set. One criterion for this selection is to choose decoys with more neighbors over decoys with fewer neighbors. The use of this criterion is well justified [6,7], and there are a few tools which incorporate this strategy [1,8,9]. In the popular protein structure prediction systems I-TASSER [2,8] and ROSETTA [1], decoys are selected using the following procedure: Starting with the set of generated decoys, a threshold d is first decided. Then, from the set, the decoy with the most neighboring decoys within RMSD d from it is found, and is reported as the highest

ranking decoy. (Ties are broken arbitrarily.) This decoy and all of its neighbors (the first cluster) are then removed from the set, after which the decoy with the most neighbors within RMSD d is again found. This decoy is then reported as the second highest ranking decoy, and together with all its neighbors (the second cluster) are removed from the set. Similarly the third highest ranking decoy is then found, and so on.

Current implementations of this procedure evaluate pairwise RMSD (or approximate values) of the decoys, resulting in runtimes which are at best quadratic in the number of decoys. As the number of decoys grows to the tens of thousands, this method becomes infeasible, necessitating the development of faster methods. In this paper we propose three strategies to speed up the procedure, with no compromise on the clustering performed. That is, the resultant method produces exactly the same clusters that are produced based on pairwise

* Correspondence: scli@bioinformatics.uwaterloo.ca

¹David R. Cheriton School of Computer Science, University of Waterloo, Waterloo ON N2L 3G1 Canada

comparison, but only faster. In the first strategy we create auxiliary groups of proximate decoys. This allows us to, through the use of triangular inequality, deduce if a group of decoys is (or is not) within the threshold distance from a given decoy. Our second strategy is to use efficiently computable lower and upper bounds of the RMSD to preliminary screen out unlikely candidates. Thirdly, outlier decoys can be detected and removed prior to the clustering. These strategies are implemented in an open-source tool called Calibur.

Implementation

Coordinates of the C_α atoms on the protein fold backbone are used to represent the main structure of a protein. As distance measure between two protein structures we use the backbone C_α -carbon root mean squared deviation (C_α RMSD). Each C_α atom corresponds to a point in 3D space. For two protein structures $S_1 = (s_{1,1}, s_{1,2}, \dots, s_{1,n})$ and $S_2 = (s_{2,1}, s_{2,2}, \dots, s_{2,n})$, each $s_{i,j}$, $1 \leq i \leq 2$, $1 \leq j \leq n$, is a 3D point indicating a C_α atom in the backbone. The C_α RMSD of S_1 and S_2 is defined as:

$$C_\alpha \text{ RMSD}(S_1, S_2) = \min_{R \in \mathcal{R}, T \in \mathcal{T}} \sqrt{\frac{\sum_{i=1}^n \|Rs_{1,i} - s_{2,i} - T\|^2}{n}}$$

where \mathcal{R} is the set of all rotations and \mathcal{T} the set of all translations.

Strategy 1: Auxiliary grouping of decoys

To avoid pairwise C_α RMSD computation, proximate decoys can be considered collectively, in deciding whether they are within C_α RMSD d from a decoy. We illustrate this idea in Figure 1, where the input decoys are collected into five groups. When finding all the decoys that are within C_α RMSD d from the decoy A (which is itself in group 2), one can first consider each

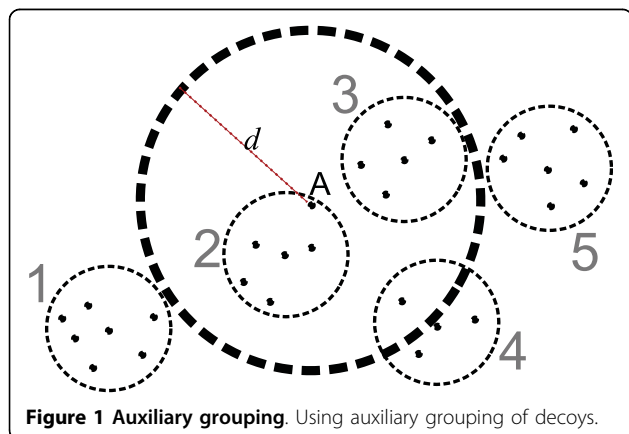


Figure 1 Auxiliary grouping. Using auxiliary grouping of decoys.

of the five groups as a whole. In this case, all the decoys in the groups 2 and 3 are within C_α RMSD d from A , while all the decoys in the groups 1 and 5 are further than d from A . No such conclusion can be collectively made about the decoys in group 4. This strategy is made possible by the fact that we can decide if an entire collection of decoys is within C_α RMSD d from a decoy A by comparing A to a representative decoy C for the collection. That is, if A is within a certain distance from C , then we conclude that the entire group is within d from X . Similarly, no decoy in the group is within d from A if A is further than some distance from C . How this can be done is as follows.

We want a grouping such that each decoy belongs to exactly one group, and is at most C_α RMSD r from the group's center (i.e. the representative decoy). This is done as follows: First a distance r less than d is decided, and an arbitrary decoy is set as a center. (Let $r = \frac{d}{2}$ for Case 1 below.) Repeatedly we take an ungrouped decoy, and try to find from all current centers for one which it is within distance r from. If and when any such center C is found the decoy is grouped with C and its distance to C is recorded. Otherwise the decoy is declared as a new center.

To locate the decoys in a group that are within distance d from a decoy A , one can consider the following five cases (denote by C the group's center and X an arbitrary decoy in the group):

Case 1: A is in the group of C (including when A is the group's center), given that $r = \frac{d}{2}$.

Case 2: $C_\alpha \text{ RMSD}(A, C) + r \leq \frac{d}{2}$.

Case 3: $C_\alpha \text{ RMSD}(A, C) > d + r$.

Case 4: $C_\alpha \text{ RMSD}(A, C) + C_\alpha \text{ RMSD}(C, X) \leq d$

Case 5: $|C_\alpha \text{ RMSD}(A, C) - C_\alpha \text{ RMSD}(C, X)| > d$

These cases are depicted in Figure 2. Since C_α RMSD is a metric [10], triangular inequality applies. Hence in Cases 1 and 2, all the decoys grouped with C must be within distance d from A . In Case 3 the converse is true.

In Cases 4 and 5, we take advantage of the already computed distance from the group's center to each member of the group. Again, triangular inequality implies that in Case 4, the decoy X is within distance d from A , while in Case 5 the converse is true. The C_α RMSD between X and A is computed if and only if none of the cases applies.

Strategy 2: Lower and upper bounds to C_α RMSD

Given any two decoys X and A , an efficiently computable lower bound of C_α RMSD(X, A) can be used to detect if C_α RMSD(X, A) is larger than a given threshold d . Likewise, an upper bound can be used to detect the case where C_α RMSD(X, A) is smaller than d . Our strategy is to use multiple such efficiently computable bounds as preliminary checks to reduce

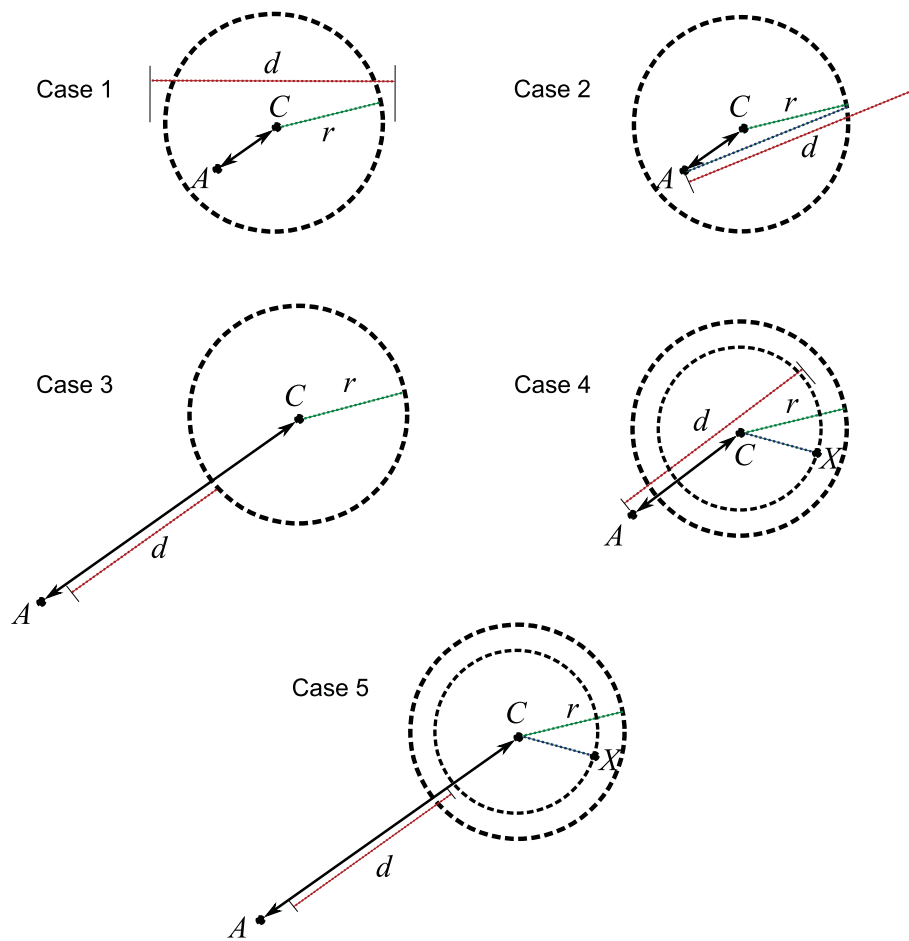


Figure 2 Deciding if decoys in a group is within d from A . Deciding if decoys' distances are within the given threshold without exhaustive C_α RMSD computation.

the much more expensive C_α RMSD computations. We will first propose a few of such upper and lower bounds, and then demonstrate how they are applied. First consider any three decoys, O , X and Y . By triangular inequality,

$$\begin{aligned} C_\alpha \text{ RMSD}(X, O) + C_\alpha \text{ RMSD}(Y, O) &> C_\alpha \text{ RMSD}(X, Y) \\ |C_\alpha \text{ RMSD}(X, O) - C_\alpha \text{ RMSD}(Y, O)| &\leq C_\alpha \text{ RMSD}(X, Y) \end{aligned}$$

Hence we can efficiently compute an upper and a lower bound to $C_\alpha \text{ RMSD}(X, Y)$, through an arbitrarily chosen *reference decoy* O and pre-computed $C_\alpha \text{ RMSD}(X, O)$ values for each decoy X . In practice, one can use n reference decoys O_1, O_2, \dots, O_n to obtain n upper bounds and n lower bounds.

The Euclidean distance between two decoys, after they are re-orientated to minimize their C_α RMSDs to a fixed arbitrary decoy, yields another upper bound to their $C_\alpha \text{ RMSD}$ [9]. This upper bound distance is referred to as rRMSD.

Another lower bound can be obtained as follows. Denote the centroid of a protein structure S_x as c_x . The *signature* Sig_x for a protein structure $S_x = (s_{x,1}, s_{x,2}, \dots, s_{x,n})$ is defined as:

$$Sig_x = \langle v_{x,1}, v_{x,2}, \dots, v_{x,n} \rangle \quad (1)$$

where $v_{x,i} = \|s_{x,i} - c_x\|$, $1 \leq i \leq n$. Define the distance between two signatures Sig_1 and Sig_2 , called *signature distance*, as:

$$dist(Sig_1, Sig_2) = \frac{1}{\sqrt{n}} \sqrt{\sum_{j=1}^n (v_{1,j} - v_{2,j})^2} \quad (2)$$

The signature distance of two protein structures is a lower bound of their $C_\alpha \text{ RMSD}$, that is:

Lemma 1. $C_\alpha \text{ RMSD}(S_1, S_2) \geq dist(Sig_1, Sig_2)$

Proof. Let R and T be the optimal rotation and translation found in computing the $C_\alpha \text{ RMSD}$ of two

structures S_1 and S_2 . Let $r_k = ||Rs_{1,k} - s_{2,k} - T||^2$, $u_{1,k} = \langle s_{1,k}, c_1 \rangle$ and $u_{2,k} = \langle s_{2,k}, c_2 \rangle$, $1 \leq k \leq n$. $u_{1,k}$ and $u_{2,k}$ are line segments with lengths $v_{1,k}$ and $v_{2,k}$ respectively.

It is known that the superposition in computing the C_α RMSD of any two structures results in the centroids of the structures to coincide [11].

Let θ be the angle between $u_{1,k}$ and $u_{2,k}$. By trigonometry, $r_k = v_{1,k}^2 + v_{2,k}^2 - 2v_{1,k}v_{2,k} \cos \theta \geq (v_{1,k} - v_{2,k})^2$. Hence,
$$C_\alpha \text{ RMSD}(S_1, S_2) = \frac{1}{\sqrt{n}} \sqrt{(r_1 + \dots + r_n)} \geq \frac{1}{\sqrt{n}} \sqrt{((v_{1,1} - v_{2,1})^2 + \dots + (v_{1,n} - v_{2,n})^2)} = \text{dist}(\text{Sig}_1, \text{Sig}_2).$$

To decide if a decoy X is within C_α RMSD d of a decoy A , we first compute the bounds and examine the following.

- If any of the upper bounds of C_α RMSD(X, A) is smaller than or equal to d . If so, clearly C_α RMSD(X, A) $\leq d$.
- If any of the lower bounds of C_α RMSD(X, A) is larger than d . If so, clearly C_α RMSD(X, A) $> d$.

We compute C_α RMSD(X, A) if and only if these two checks fail.

The upper and lower bounds can also be applied to the conditions in Case 2 and Case 3 of Strategy 1, as follows.

- In Case 2, if any of the upper bounds of C_α RMSD(A, C) is smaller than $d - r$, then the condition C_α RMSD(A, C) $+ r \leq d$ holds.
- In Case 3, if any of the lower bounds of C_α RMSD(A, C) is larger than $d + r$, then the condition C_α RMSD(A, C) $> d + r$ holds.

We compute C_α RMSD(A, C) for Case 2 and Case 3 if and only if these two checks fail.

Strategy 3: Filtering outlier decoys

Another possible enhancement to performance is to discard decoys with low similarity to other decoys in the set, prior to the clustering. Here we propose an efficient technique to quickly identify such decoys. Our aim is to retain all of the high ranking decoys, and the decoys which are within distance d from them. We identify these as “good” decoys. Assume that every high ranking decoy is within distance d from 10% of all the decoys. For a random sample of n decoys, the probability for a good decoy to be within a distance $2d$ from at least one of the sampled decoys is $1 - 0.9^n$, which is > 0.9999 when $n = 100$. Hence decoys that are not within $2d$ from any one of 100 randomly sampled decoys are likely not good, and are removed from the set.

Overall program design

We designed a program based on the three strategies. On a given set S of n input decoys, the program does the following:

Step 1: Discover a suitable threshold distance d for clustering S .

Step 2: Filter outlier decoys using 100 randomly selected decoys, as in Strategy 3.

Step 3: Create auxiliary groups out of the decoys as required by Strategy 1. Compute the signature (Sig_x), and the distances (C_α RMSD(X, O) for each decoy X and reference decoy O ; $r\text{RMSD}(X, Y)$ for each decoy X, Y) as required by Strategy 2.

Step 4: Find for each decoy A a neighbor set N_A which contains all the decoys in S within distance d from A (A inclusive), using Strategy 1 with the preliminary checks of Strategy 2. This is done in a straight-forward fashion as follows.

1. Set N_A to an empty set.
2. For each auxiliary group of decoys G (C denote the center of G),
 - (a) If A is in G , add all decoys in G into N_A and go for the next auxiliary group.
 - (b) Examine if C_α RMSD(A, C) $+ r \leq d$ using each of the upperbounds of C_α RMSD(A, C).
If true, add all decoys in G into N_A . Go for the next auxiliary group.
 - (c) Examine if C_α RMSD(A, C) $> d + r$ using each of the lowerbounds of C_α RMSD(A, C).
If true, skip G . Go for the next auxiliary group.
 - (d) Compute C_α RMSD(A, C).
 - (e) Examine if C_α RMSD(A, C) $+ r \leq d$.
If true, add all decoys in G into N_A . Go for the next auxiliary group.
 - (f) Examine if C_α RMSD(A, C) $> d + r$. If true, skip G . Go for the next auxiliary group.
 - (g) For each decoy X in G ,
 - i. Examine if C_α RMSD(A, C) $+ C_\alpha$ RMSD(C, X) $\leq d$.
If true, add X into N_A . Go for the next decoy in G .
 - ii. Examine if $|C_\alpha$ RMSD(A, C) $- C_\alpha$ RMSD(C, X)| $> d$.
If true, skip X . Go for the next decoy in G .
 - iii. Examine if C_α RMSD(A, X) $\leq d$ using each of the upperbounds of C_α RMSD(A, X).
If true, add X into N_A . Go for the next decoy in G .
 - iv. Examine if C_α RMSD(A, X) $> d$ using each of the lowerbounds of C_α RMSD(A, X).
If true, skip X . Go for the next decoy in G .

- v. Compute C_α RMSD(A, X).
 - vi. If C_α RMSD(A, X) $\leq d$, add X into N_A .
3. Output N_A .

Step 5: Start with an empty sequence *Output*. Repeatedly find $A \in S$ with the largest N_A , appending A to *Output* while removing N_A from S and all the neighbor sets.

Step 6: Output the decoys in *Output*. (For brevity the program is set to output only the first 3 decoys.)

The threshold selection in Step 1 is addressed in the next sub-section.

Steps 2 and 3 are performed straightforwardly.

Step 5 is performed by repeating the following until S is empty: Find the decoy $X \in S$ with the largest N_X (breaking ties arbitrarily) and append the decoy to *Output*. Then, remove N_X from S and for each $Y \in N_X$, remove Y from N_Z for each $Z \in N_Y$.

Selection of a suitable threshold

We consider two decoys to be significantly related if and only if their C_α RMSD is *relatively small* among all pairwise C_α RMSDs of the decoys. Hence our strategy to threshold finding is to identify a value d such that only x percent of pairwise C_α RMSD distances are below d , for some suitable x . Given x , a straightforward way to determine such a d exactly is to compute all $n \times n$ C_α RMSDs and find the $(0.01xn^2)$ -th shortest distance. Alternatively, a reasonable approximation to the x -percentile value can be obtained efficiently using only a relatively small random sample of the decoys. In our tests, around 10 samplings of 100 decoys each sufficed to determine this value quickly and consistently in general. Our program uses this method by default, with x set to $\min\{100n^{-1/4}, 10\}$. The expression $100n^{-1/4}$ is heuristic. It's aim is to reduce the percentile when more decoys are available, in order to speed up the clustering (e.g., $x = 10$ when $n = 10000$, $x = 5$ when $n = 160000$).

A similar strategy would be to use the most frequently occurring C_α RMSD among decoys, f say, as a reference to decide a threshold distance d . (If the pairwise distances are distributed normally, f would correspond to the 50th percentile.) As a selectable option the program includes a simple method based on this, in which we let $d = cf + b$, where c is set to $\frac{2}{3}$ and b is set to the minimum pairwise distance discovered through random sampling.

Memory usage

In Steps 1-3 and 5, the memory required is linear in n . For Step 4, in the theoretical worst case, $|N_X| = n$ for each X , resulting in $O(n^2)$ memory usage. However, such a scenario is unlikely to occur in the program's intended use. In practical use, $|N_X|$ is seldom above $0.2n$, and small for most X . Note that in the case that the number of neighbor sets of a given size falls off geometrically with the size, the memory required to store

all neighbor sets would in fact be linear in n . In our tests, the actual growth in memory usage is closer to $O(n)$ than $O(n^2)$.

If one is interested in only the highest ranked decoy from the clustering, it is unnecessary to construct the neighbor sets, since the sizes of the neighbor sets suffice to determine such a decoy. In this case, the total memory usage would be linear in n . We include this mode of operation as an option.

Results

Our C++ implementation of the program is called Calibur. Calibur accepts as input a list of names of PDB files (each for a decoy) and an optional threshold d . No pre-processing is required of the PDB files. If no threshold is given, Calibur automatically finds a suitable threshold for the input decoys, as discussed. The method which Calibur uses for threshold discovery can be altered through commandline arguments. A list of all the implemented methods is shown when Calibur is called without any input arguments.

Effectiveness of strategies

The effectiveness of each of the strategies was evaluated with decoys predicted by FALCON (reported in Alipaliani *et al.*: A protocol for automated NMR protein structure determination, submitted for publication) on the proteins TM1112 from the Arrowsmith Lab at University of Toronto (herein the set is referred to as TM1112) and SH3 from Donaldson's Lab at York University (herein referred to as CASKIN). Each of these two sets contains 9999 decoys.

Auxiliary grouping, lower and upper bounds

Each of the different cases contributed in reducing the runtime, although the amounts differed at different thresholds (see Figures 3 and 4). At low thresholds, the chances of decoys being further than the threshold distance are high. Hence evaluations via Case 3 and the lower bounds are more effective. For a similar reason, the effects of evaluations through Case 1 and the upper bounds become elevated at larger thresholds.

In Calibur, the order in which evaluations are performed, as well as the range of thresholds to use for the evaluations has been optimized based on these observations.

Filtering

On the data sets TM1112 and CASKIN, filtering did not affect the clusters formed by the highest ranking decoys. Their rankings remained the same. This is true even in the cases where more than 70% of decoys had been filtered prior to the clustering. Figure 5 and 6 show, for TM1112 and CASKIN respectively, the number of decoys filtered (out of the total of 9999 decoys) at various threshold values.

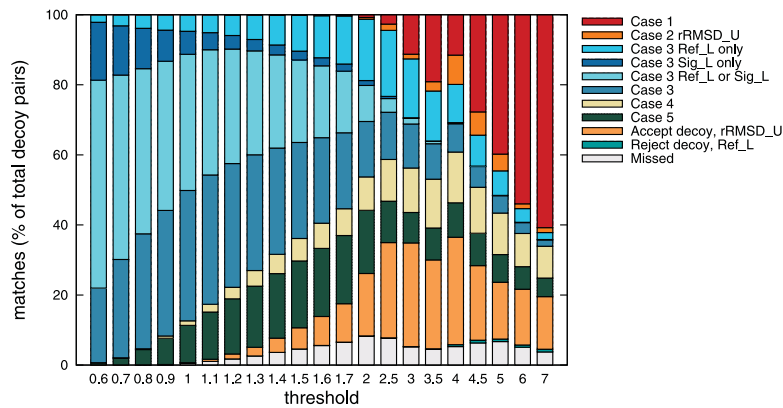


Figure 3 Contribution of Strategies 1 and 2 on TM1112. The number of C_{α} RMSD computations avoided (percentage over 9999×9998 computations) due to each of the cases considered, at different threshold values. For Case 2, the upper bounds are used for condition evaluation prior to the actual C_{α} RMSD. The contribution from the upper bounds via the reference decoys can be completely accounted for by rRMSD, while C_{α} RMSD evaluations contributed insignificantly. Only the contribution from rRMSD (label "Case 2 rRMSD_U") is shown. For Case 3, the lower bounds are used for condition evaluation prior to the actual C_{α} RMSD. While the contributions from both kinds of lower bounds overlapped (label "Case 3_Ref_L or Sig_L"), there were contributions entirely due to evaluations using the signature (label "Case 3 Sig_L only") as well as the reference decoys (label "Case 3 Ref_L only"). The contribution from evaluating the actual C_{α} RMSD was highly significant as well (label "Case 3"). In evaluating individual decoys, the upper bound obtained from rRMSD was highly effective at high thresholds (label "Accept decoy, rRMSD_U"). The lower bounds from the reference decoys demonstrated noticeable effects (label "Reject decoy, Ref_L"). Other contributions were insignificant.

Strategies' effects on Calibur's performance

To evaluate the strategies' effects on Calibur at various thresholds, the runtimes when the strategies are used ("Calibur") and when they are not used ("pairwise") were compared. For reference purposes, the runtime for ROSETTA's pairwise evaluation based clustering program ("cluster_info_silent") is also shown. ROSETTA is currently the most popular system for protein structure prediction. All the tests are run on a 3 GHz Intel Core 2 Duo PC with 2.98 G RAM running CentOS 5.3. All three tools are compiled using GCC 4.1.2 with

optimization -O. The same codes is used for computing C_{α} RMSD.

All the tools were given input such that the output would be exactly the same. Hence we compare only their runtimes. For *pairwise* and *cluster_info_silent*, the CPU time is taken to be the total time needed for neighbors finding and the recursive search for largest clusters. For Calibur, the CPU time is the sum of the times taken for signature computation, decoys re-orientation, filtering, auxiliary grouping, neighbors finding, and the recursive search for the largest clusters. Figure 7 shows the results

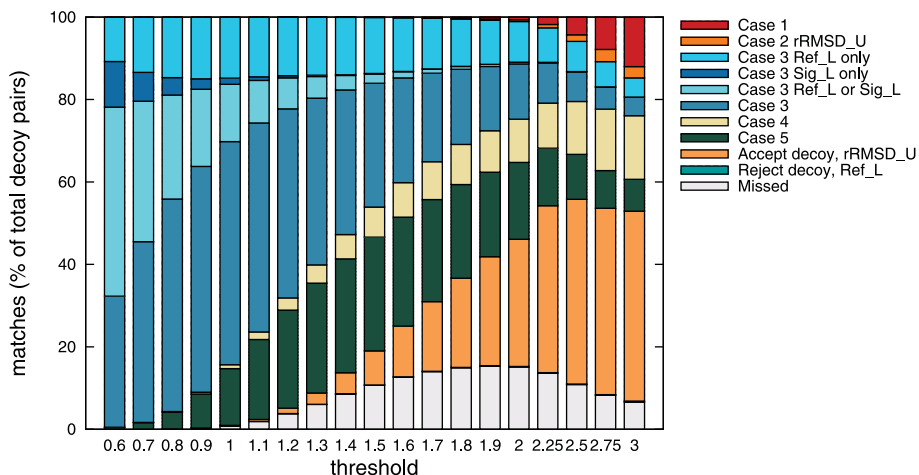


Figure 4 Contribution of Strategies 1 and 2 on CASKIN. Same as Figure 3, but on the CASKIN data set.

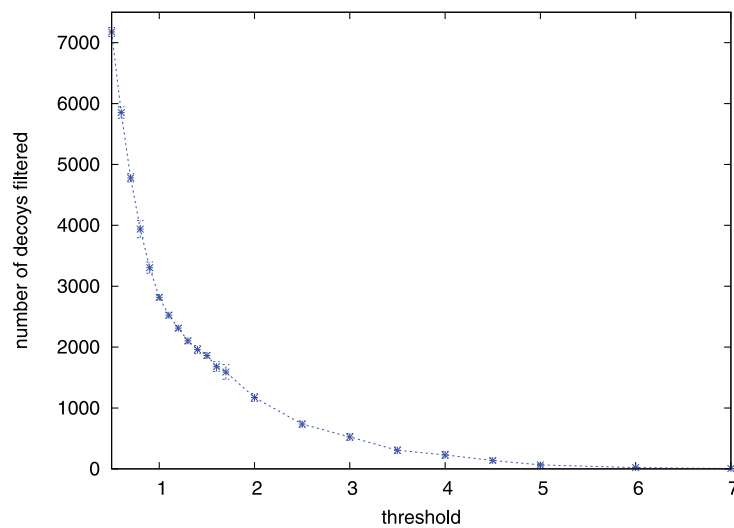


Figure 5 Number of decoys filtered from TM1112. The number of decoys filtered from the set TM1112 using 100 randomly selected decoys at different thresholds. Each value is an average of 10 numbers from 10 different trials using the same threshold. Error bars show the standard deviations.

on the data set TM1112. The largest sizes of the clusters at the thresholds 1, 2, 3, 4, 5, 6, 7 are respectively 1796, 6017, 7744, 8186, 8671, 9120, 9368.

Calibur's performance on a large data set

Calibur's performance in handling large numbers of decoys was evaluated using a set of 29770 decoys for the TM1112 protein generated by FALCON. For each threshold in 0.5, 1, 1.5, 2, 3, 4, 5, we performed 10 trial runs over a UNIX cluster. (More precisely, an HP XC cluster with 378 nodes, each with 8× Xeon 3.0 GHz

CPUs and 16 GB memory, running RHEL 5.1.) All the runs resulted in the same decoy clusters. Table 1 shows the average CPU times (in sec).

In practice the largest clusters typically contain around 10% of the decoys. In the present case, the largest clusters found at 1.5 threshold distance already contain more than 18% of all the decoys. At this point, the corresponding CPU time required by Calibur is about one third of the time required when the strategies are not used.

As a further reference on Calibur's performance in high load use, Calibur completed in around 15000

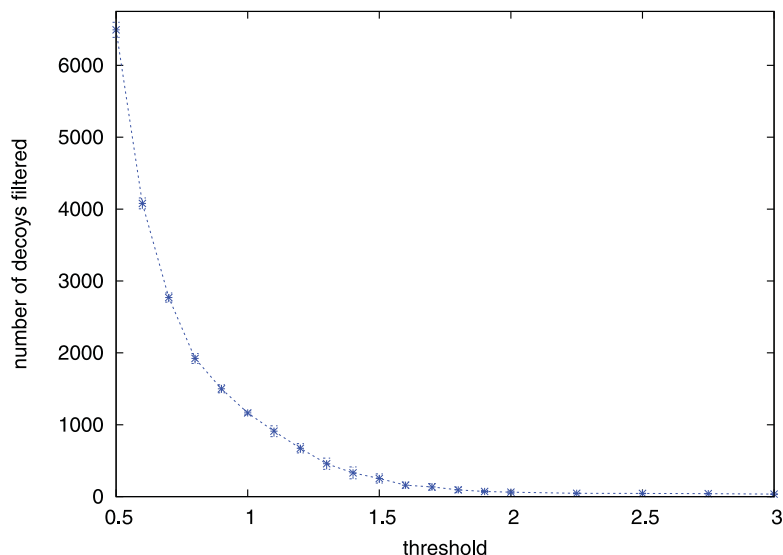


Figure 6 Number of decoys filtered from CASKIN. Same as Figure 5, but on the CASKIN data set.

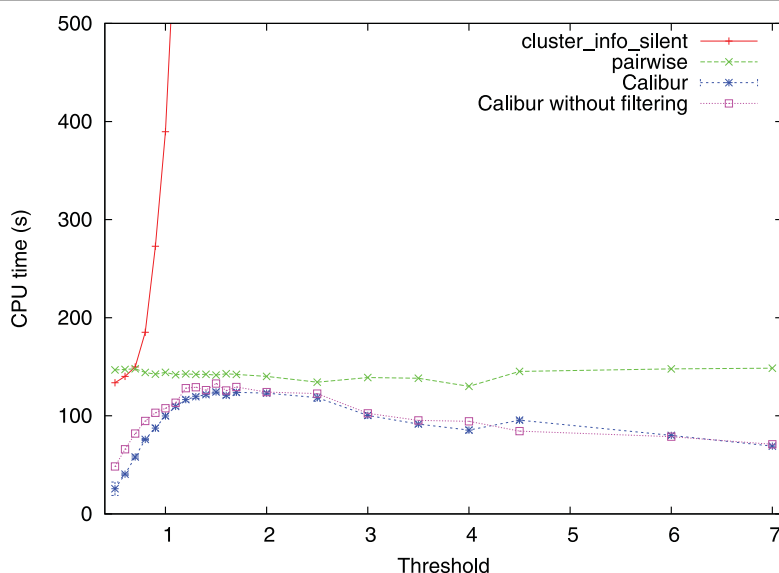


Figure 7 Runtime of cluster_info_silent, pairwise and Calibur on TM1112. The CPU times used to obtain cluster at different thresholds on the TM1112 data set of 9999 decoys, by (1) cluster_info_silent (label "cluster_info_silent"). (2) Calibur without using any of the strategies (label "pairwise"), (3) Calibur (label "Calibur") (To account for variations caused by the filtering each point is an average of 10 trials), (4) Calibur with the filtering mechanism disabled (label "Calibur without filtering").

seconds CPU time under its default settings in our recent tests using 100,000 decoys.

Evaluation of Calibur's output decoys

To evaluate the decoys produced by Calibur, we compared them to that obtained using SPICKER [8], the clustering tool used in the leading ab initio protein structure prediction system I-TASSER [2]. We used the decoy sets, natives and SPICKER's results published on I-TASSER's website [12], downloaded on the 24th of July, 2009. The data consists of decoys for 56 targets. The number of decoys for each target is shown in Table 2.

In order to compare Calibur with SPICKER in terms of both output and speed we ran Calibur under the same conditions as SPICKER. Both programs were compiled with optimization -O3 and were made to cluster exactly the same set of decoys. Filtering was disabled in Calibur. We noticed a limit on the number of decoys that SPICKER handles. When the number of decoys is larger than 13000, SPICKER samples only 13000 decoys for clustering. To test Calibur with the same set of

decoys that SPICKER clusters, we obtained 13000 decoys from each decoy set that is larger than 13000 (using the same procedure as in SPICKER's source codes) and tested Calibur with these decoys.

When decoys are sampled, they may not be sufficiently representative and the quality of the best decoy obtained may be compromised. To investigate this effect, we randomly sampled 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500 decoys from each of the original sets and ran SPICKER and Calibur with these sampled sets. Since only 6119 decoys are available for 1mkyA3, the full set was used as the sampled set at sizes above 5500.

All the tests were performed on the same UNIX cluster as in the previous section. Calibur used its default method for automatic threshold distance discovery. Table 3 shows, at different sample sizes, the average TM-scores and total C_{α} RMSDs (to native) for the best decoys reported by both tools, as well as the total CPU times used, as reported by the UNIX servers. These results are shown as histograms in Figures 8, 9 and 10. Detailed results are given in Additional File 1. The

Table 1 CPU times of Calibur on large data set.

Threshold	0.5 (27)	1.0 (1966)	1.5 (5531)	2.0 (8560)	3.0 (14397)	4.0 (17915)	5.0 (19905)
Calibur	74 ± 7	506 ± 14	1047 ± 27	1482 ± 42	2369 ± 154	3109 ± 266	3616 ± 290
no filtering	225 ± 11	717 ± 22	1250 ± 35	1629 ± 42	2495 ± 180	3166 ± 233	3501 ± 272
Pairwise	2628 ± 72	2624 ± 69	2651 ± 66	2741 ± 83	3293 ± 205	4014 ± 130	4425 ± 324

Numbers in brackets are the sizes of the largest clusters at the corresponding threshold.

CPU times of (1) Calibur, (2) Calibur with the filtering mechanism disabled ("no filtering"), and (3) pairwise, on a set of 29770 decoys for the TM1112 protein. Shown with ± standard deviation.

Table 2 Sizes of the decoy set for each target.

Target	#Decoys	Target	#Decoys	Target	#Decoys	Target	#Decoys
1abv_	12500	1dtjA_	20000	1mkyA3	6119	1shfA	20000
1af7_	12499	1egxA	20000	1mla_2	12500	1sro_	20000
1ah9_	27498	1fadA	12599	1mn8A	12500	1ten_	20000
1aoy_	32000	1fo5A	20000	1n0uA4	12499	1tfi_	32000
1b4bA	12500	1g1cA	19997	1ne3A	12500	1thx_	32000
1b72A	12499	1gjxA	12500	1no5A	12500	1tif_	12500
1bm8_	20000	1gnuA	17533	1npsA	20000	1tig_	12500
1bq9A	20000	1gpt_	32000	1o2fB_	12500	1vcc_	20000
1cewl	19830	1gyvA	11508	1of9A	20000	256bA	20000
1cqkA	19999	1hbkA	20000	1ogwA_	19998	2a0b_	32000
1csp_	12500	1itpA	12500	1orgA	20000	2cr7A	12500
1cy5A	32000	1jnuA	20000	1pgx_	20000	2f3nA	19999
1dcjA_	20000	1kjs_	20000	1r69_	20000	2pcy_	20000
1di2A_	20000	1kviA	20000	1sfp_	19985	2reb_2	12500

The number of decoys for each target.

average TM-score and total C_{α} RMSD reported by both tools are similar, showing the decoys reported by both tools to be comparable. The CPU times required by Calibur are significantly less than SPICKER for sample sizes above 2500, even with PDB files processing time included. There is an observable trend of increase in average TM-score as well as decrease in total C_{α} RMSD, with an increase in sample size.

For each sample size, we counted the number of targets where the best decoy reported by SPICKER has a better TM-score than Calibur, as well as the number of targets where the best decoy reported by Calibur has a better TM-score. These numbers are shown in Figure 11. Figure 12 shows the numbers where instead of the TM-score, the C_{α} RMSD is compared.

Further experiment with Calibur using more decoys than 13000 suggests that decoy quality may improve when a fuller set of decoys is used. We randomly sampled 16000, 19000, 22000, 25000, 28000 and 31000

decoys from each of the target sets and ran Calibur on these sampled sets. At each of the sample sizes, if the set contains less decoys than the sample size, then the entire set of decoys was used. The results are shown in Table 4 (details given in Additional File 2). The highest average TM-score and total C_{α} RMSD is observed at sample size 28000. Rank sum test shows the average TM-scores obtained at the sample sizes above 13000 to be higher than those obtained at sample sizes less than or equal to 13000 ($p \leq 0.1$).

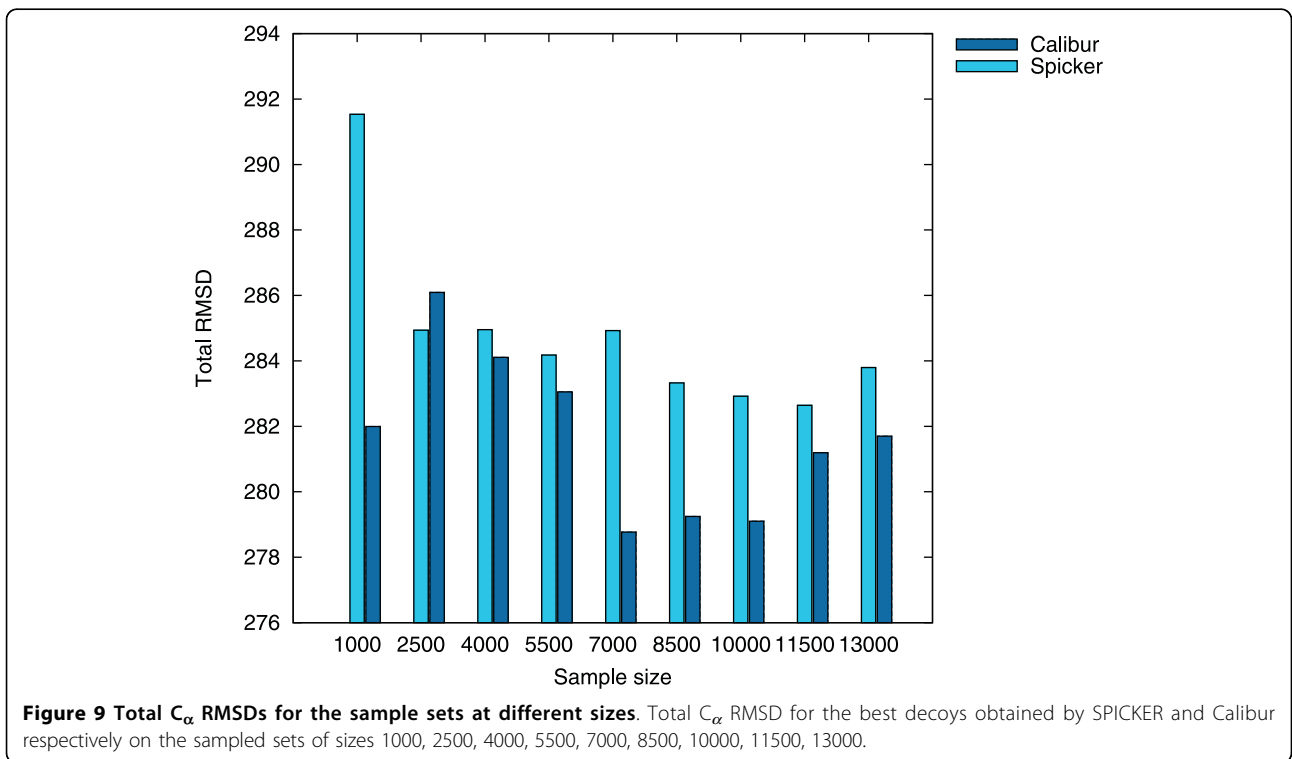
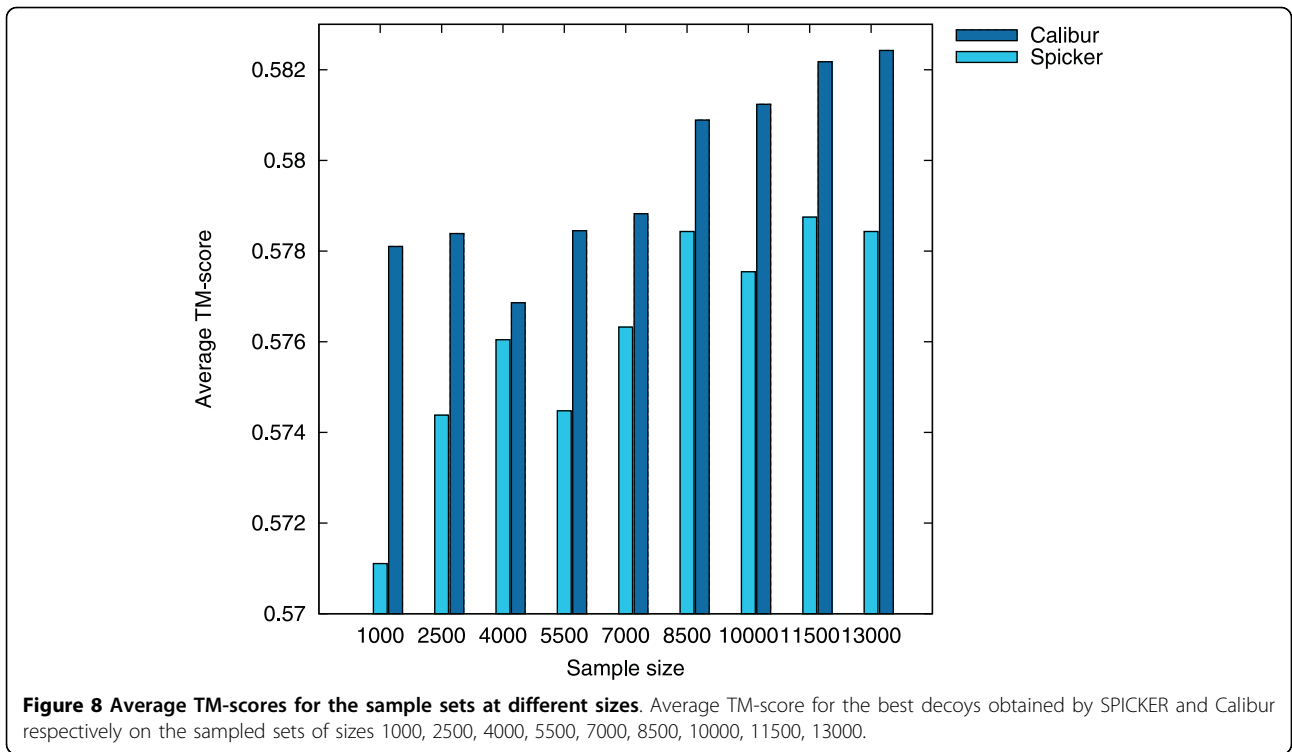
Conclusion

Calibur is a carefully implemented tool, dedicated to the purpose of clustering very large numbers of decoys. As methods in ab initio protein structure prediction advances, the number of decoys to be analyzed is expected to increase, and the disability to cluster decoys efficiently will eventually pose a hindrance to the analyses of various problems and subproblems in the

Table 3 Total scores and CPU times.

Sample size	Average TM-score		Total C_{α} RMSD		Total CPU Time (s)	
	SPICKER	Calibur	SPICKER	Calibur	SPICKER	Calibur
1000	0.571107	0.578102	291.534	281.994	136.05	237.25
2500	0.574379	0.578386	284.937	286.09	687.45	746.84
4000	0.576045	0.576859	284.953	284.108	1851.34	1533.81
5500	0.574475	0.57845	284.18	283.055	3276.36	2781.55
7000	0.576323	0.578823	284.927	278.769	7029.09	4320.95
8500	0.57843	0.580889	283.325	279.248	8132.9	5906.61
10000	0.577543	0.581236	282.919	279.1	10745.5	7822.75
11500	0.578748	0.582175	282.644	281.192	14293.8	10374.2
13000	0.578432	0.582425	283.795	281.701	16608.8	12420.5

Average TM-score and total C_{α} RMSD for the best decoys obtained, as well as the total CPU time used, on the sampled sets of sizes 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500, 13000. (SPICKER requires additional pre-processing of the input PDB files which are not added into the CPU times here.)



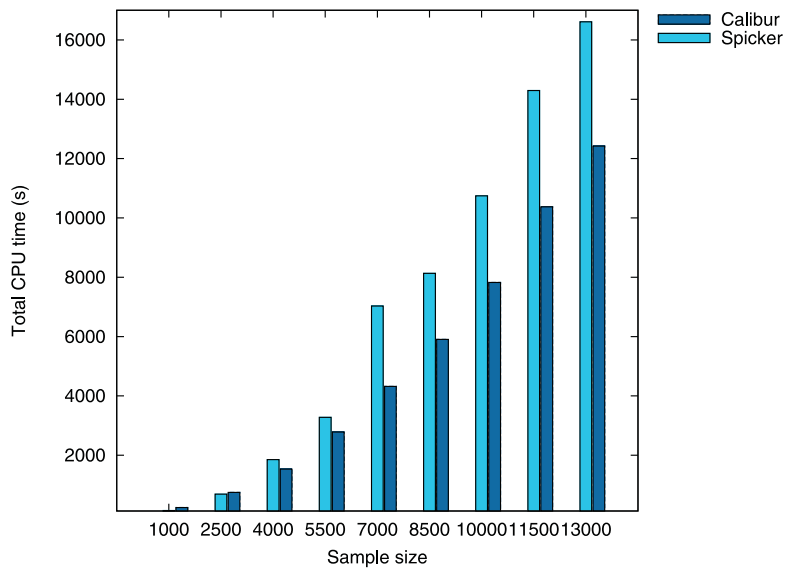


Figure 10 Total CPU times for the sample sets at different sizes. Total CPU time (sec) used by SPICKER and Calibur respectively on the sampled sets of sizes 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500, 13000.

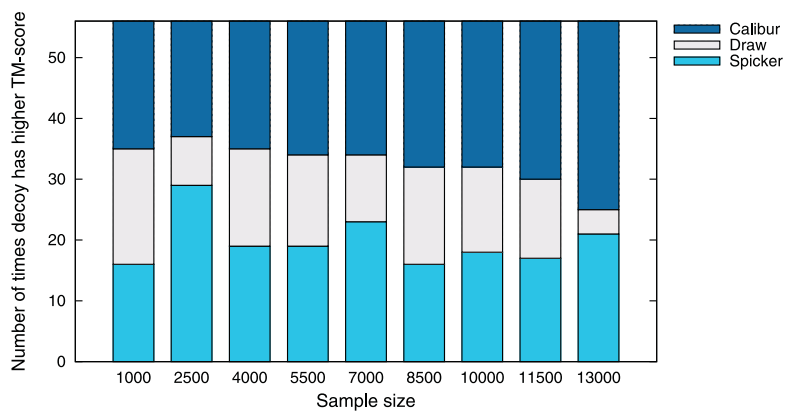


Figure 11 Number of times decoy with better TM-score is reported by each method. Number of times the decoy with better TM-score was reported by SPICKER and Calibur respectively, on the sampled sets of sizes 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500, 13000.

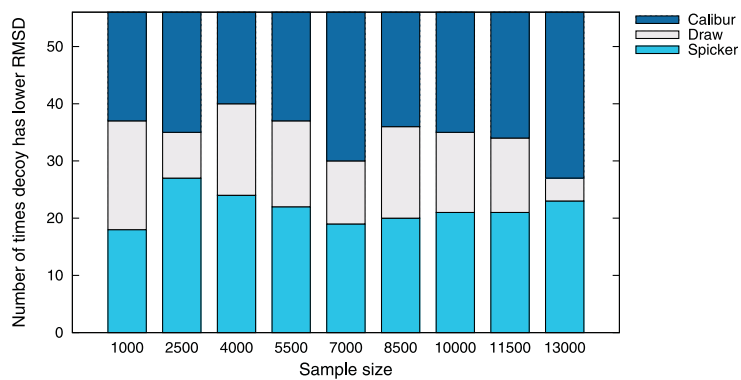


Figure 12 Number of times decoy with better C_α RMSD is reported by each method. Number of times the decoy with better C_α RMSD was reported by SPICKER and Calibur respectively, on the sampled sets of sizes 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500, 13000.

Table 4 Total scores and CPU times at larger sample sizes.

Max sample size	Average TM-score	Total C _α RMSD	Total CPU Time (s)
16000	0.581143	283.749	28857.2
19000	0.582325	281.589	38928.9
22000	0.580996	279.887	45916
25000	0.581545	283.549	47083.1
28000	0.58272	278.243	50669.4
31000	0.581805	282.32	53582.2

Average TM-score and total C_α RMSD for the best decoys obtained, as well as the total CPU time used, on the sampled sets of maximum sizes 16000, 19000, 22000, 25000, 28000 and 31000.

prediction of protein structures. It is our belief that Calibur, together with the methods it implements, will come in very useful when this situation arises. For this reason, we have decided to release the source codes of Calibur with an open license.

Availability and requirements

- **Project name:** Calibur
- **Project homepage:** <http://sourceforge.net/projects/calibur/>
- **Operating System(s):** Multiple platform (tested on Windows and Linux)
- **Programming Language:** C++.
- **Other requirements:** None.
- **License:** GNU General Public License

Additional file 1: Details of experiments using sample sets of sizes 1000, 2500, 4000, 5500, 7000, 8500, 10000, 11500, 13000.

Click here for file

[<http://www.biomedcentral.com/content/supplementary/1471-2105-11-25-S1.pdf>]

Additional file 2: Details of experiments using sample sets of sizes 16000, 19000, 22000, 25000, 28000 and 31000.

Click here for file

[<http://www.biomedcentral.com/content/supplementary/1471-2105-11-25-S2.pdf>]

Author details

¹David R. Cheriton School of Computer Science, University of Waterloo, Waterloo ON N2L 3G1 Canada. ²Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei, Tokyo 184-8588, Japan.

Authors' contributions

All authors jointly developed the methods and wrote the article. They read and approved the final manuscript.

Received: 24 June 2009

Accepted: 13 January 2010 Published: 13 January 2010

References

1. Simons KT, Kooperberg C, Huang E, Baker D: Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *J Mol Biol* 1997, **268**:209-25.

2. Wu S, Skolnick J, Zhang Y: Ab initio modeling of small proteins by iterative TASSER simulations. *BMC Biology* 2007, **5**(17).
3. Hamelryck T, Kent JT, Krogh A: Sampling realistic protein conformations using local structural bias. *PLoS Comput Biol* 2006, **2**(9):1121-1133.
4. Bystroff C, Shao Y: Fully automated ab initio protein structure prediction using I-SITES, HMMSTR and ROSETTA. *Bioinformatics* 2002, **18**(Suppl 1):54-61.
5. Li SC, Bu D, Xu J, Li M: Fragment-HMM: A new approach to protein structure prediction. *Protein Science* 2008, **17**(11):1925-34.
6. Shortle D, Simons KT, Baker D: Clustering of low-energy conformations near the native structures of small proteins. *Proc Natl Acad Sci* 1998, **95**:1158-62.
7. Betancourt MR, Skolnick J: Finding the needle in a haystack: Educating native folds from ambiguous ab initio protein structure. *J Comput Chem* 2001, **22**:339-353.
8. Zhang Y, Skolnick J: SPICKER: Approach to clustering protein structures for near-native model selection. *J Comput Chem* 2004, **25**:865-871.
9. Li H, Zhou Y: SCUD: Fast structure clustering of decoys using reference state to remove overall rotation. *J Comput Chem* 2005, **26**(11):1189-92.
10. Boris S: A revised proof of the metric properties of optimally superimposed vector sets. *Acta Crystallographica Section A* 2002, **58**(5):506.
11. Arun KS, Huang TS, Blostein SD: Least-squares fitting of two 3-D point sets. *IEEE Trans Pattern Anal Mach Intell* 1987, **9**(5):698-700.
12. I-TASSER protein structure decoys. <http://zhang.bioinformatics.ku.edu/I-TASSER/decoys/>.

doi:10.1186/1471-2105-11-25

Cite this article as: Li and Ng: Calibur: a tool for clustering large numbers of protein decoys. *BMC Bioinformatics* 2010 **11**:25.

Submit your next manuscript to BioMed Central and take full advantage of:

- Convenient online submission
- Thorough peer review
- No space constraints or color figure charges
- Immediate publication on acceptance
- Inclusion in PubMed, CAS, Scopus and Google Scholar
- Research which is freely available for redistribution

Submit your manuscript at
www.biomedcentral.com/submit

