SeqClusMerge - A framework for sequence cluster merging [1]

Arvind Gopu
School of Informatics
&
Department of Computer Science
Indiana University, Bloomington IN
agopu@cs.indiana.edu

August 29, 2004

[1]Towards partial requirement of author's Master of Science degree (Bioinformatics)

Contents

Abstract

With the growing number of biological sequences, thanks to world-wide genome sequencing projects, automatic sequence clustering has become increasingly important. Various bottom-up and top-down approaches have been devised to do sequence clustering. In a bottom-up approach clusters are merged recursively, while in the top-down approach clusters are split up to specific levels. Though the top-down approach has been effectively used by various sequence clustering algorithms, it has an accompanying problem of cluster fragmentation. Methods to merge fragmented clusters prove to be very useful to work against this problem. We present a robust and modularly designed framework for merging fragmented clusters bottom-up. One of our methods uses a combination of statistical and machine learning techniques while the other two, more effective ones, use properties of phylogenetic trees. Finally, we present test results that prove the effectiveness of our framework.

Contents

1 Introduction & Motivation

Genome sequencing projects all over the world have proceeded at a fast pace, leading to a large increase in the number of biological sequences. Analyzing this ever increasing number of sequences has become more and more challenging; sequence clustering has a big part to play in helping out biologists who want to dig deep into particular classes of sequences. To this effect, there have been various bottom-up and top-down clustering approaches that have been devised. In a bottom-up approach, clusters are merged recursively, while in the top-down approach clusters are split up to specific levels.

Though the top-down approach has been effectively used by various sequence clustering algorithms – BAG [1], CASTOR [2], etc. – it has an accompanying problem of cluster fragmentation. Thus, it might be possible to achieve very high accuracy or clustering correctness but the flip side of such high accuracy is that many clusters which do not share a high degree of similarity get split further into many small fragments. For example, the BAG clustering algorithm is capable of producing clusters with up to 99% accuracy, but 40 % of those clusters are fragmented. Methods to these merge fragmented clusters prove to be very useful.

To get a better understanding, assume that clusters $\{C_1, C_2, ..., C_k\}$ are generated by a top-down clustering algorithm. Sequences within a cluster are generally related in terms of sequence similarity. However, there are often cases in which sequence similarity relationship can be observed across multiple clusters. For example, two sequences, $s_k \in C_i$ and $s_l \in C_j (i \neq j)$, may share some degree of sequence similarity. So the question is, should C_i and C_j be merged? This decision is often not clear since there are many pairs of sequences, one each from C_i and C_j respectively, that are related only with very weak similarity and then there are many that share no sequence similarity whatsoever. Instead of relying on each individual pairwise relationship, the decision on cluster merging can be guided by considering all-pairwise sequence relationship between pairs of clusters, one pair at a time.

There are different ways of achieving this kind of guidance. One of them is to generate profiles or multiple sequence alignments of clusters and then work on the properties of those profiles. Another way is to build phylogenetic trees of the combined profile and work on its properties. In this document, we present a robust and modularly designed framework for merging fragmented clusters bottom-up using the above mentioned two methods.

In section 2, we illustrate the various modules of our framework. One of the them is the merge'bility test module; it uses what we call a 'merge-test' to decide on the merge'bility of two fragment clusters . Three merge-tests are presented in this document. One of the tests uses a combination of statistical and machine learning techniques (see section 3) while the other two, more effective ones, use properties of phylogenetic trees (see sections 4 and 5). In section 7, we present test results that prove the effectiveness of our framework before concluding in section 8.

2 Basic Framework

The basic framework of SeqClusMerge is shown in figure 1. Once a set of sequences $S = s_1, s_2, \dots s_n$ are clustered into a bunch of clusters $C = C_1, C_2, \dots C_k$, fragment clusters are considered for merge'bility testing. At this point, we assume the clustering algorithm will provide hints as to what clusters constitute fragments (for example the BAG clustering algorithm [1] provides merge suggestions).

Let us consider two fragment clusters C_i and C_j. There are *pre* and *post* processing steps associated with the SeqClusMerge framework. The main focus, though, are the merge'bility tests - we will refer to them as "merge-test" in the remainder of this document. The trivial modules of the framework (*pre* and *post* processing steps) are explained below in the rest of this section. Three merge-tests that we developed are explained in sections 3, 4, 5.

Preprocessing
Before any merge-test processing can be done, there are some format requirements for the data that is fed into our merge-tests. The first step, as will be explained in the next three sections, is multiple sequence alignment (MSA). *Clustalw*, which is what we have used, and most other MSA programs take in a set of sequences in fasta format to generate the alignments. So we have a script which generates sequence files for fragmented clusters produced by the clustering algorithm. This step might or might not be needed, depending on the type of output a clustering algorithm generates.

Also it should be noted that the efficacy of our merge-tests depends up to an extent on the meaningfulness of merge suggestions supplied by the clustering algorithm. Most suggestions will be rejected if the suggestions are inherently bad ones. We have done the best part of testing using artificial datasets that we generated; more detail is available in section 7.

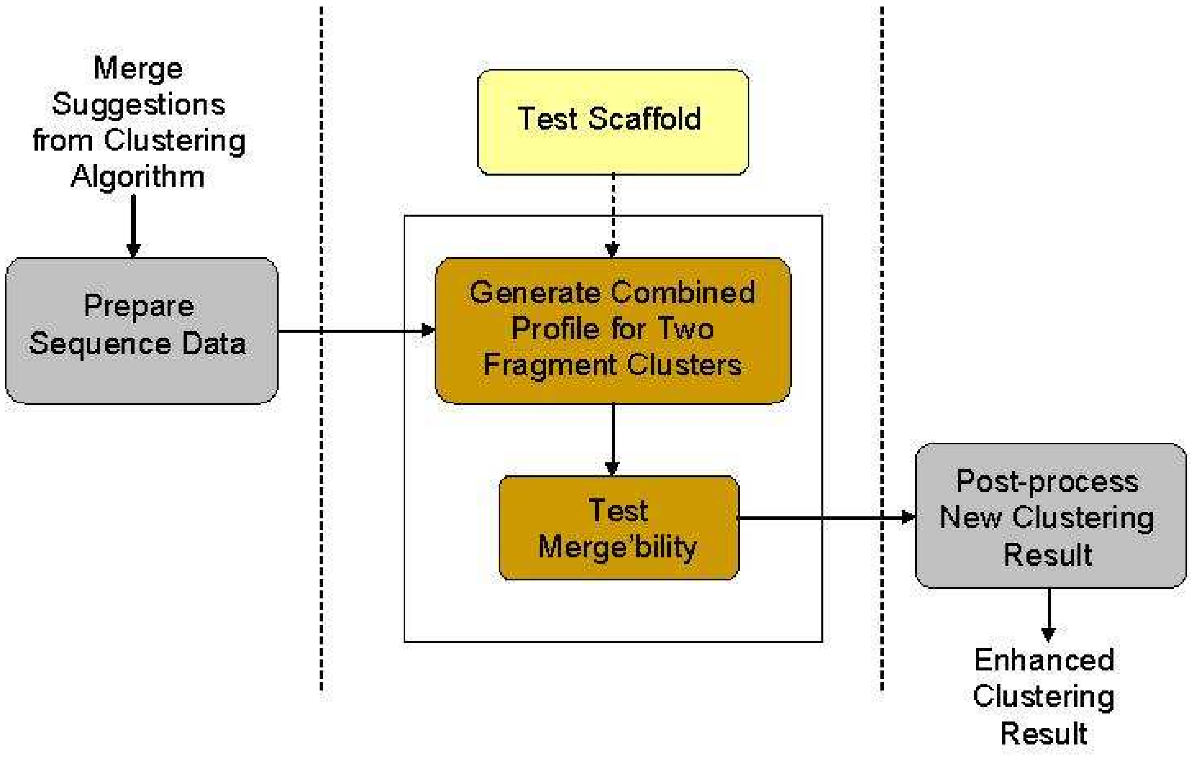

Merge Suggestions from Clustering Algorithm
Test Scaffold
Prepare Sequence Data
Generate Combined Profile for Two Fragment Clusters
Test Merge'bility
Post-process New Clustering Result
Enhanced Clustering Result

Figure 1: SeqClusMerge framework schematic

Merging Multiple Fragment Clusters
To merge multiple fragment clusters $\{C_{i_1}, C_{i_2}, ..., C_{i_l}\}$, we just apply one of the merge-tests incrementally, considering two fragments or super-fragments – a cluster which is formed out of already merging two fragments – in a greedy fashion.

3 Model Comparison based Merge-test

Recent studies show that the profile-profile alignment methods can effectively relate distantly or even structurally related group of sequences [3]. We propose a cluster merging procedure that combines a relative entropy based model comparison technique with another from statistics called Runs test to decide on merge'bility.

3.1 Combined Profile Generation

Starting with two sequence files corresponding to two fragment clusters, the following steps are carried out:

1. Individual MSA profiles are generated for the two sequence files. We used *Clustalw* for this step.

2. The two individual profiles are input to a profile alignment program to produce a single combined profile. Again *Clustalw* was used to generate the combined profile.

3.2 Relative Entropy (From Machine Learning)

Consider two clusters C_i and C_j. The main idea is that there should be a significant number of matching columns between the two multiple sequence alignments of C_i and C_j to merge the clusters into one. Similarity between two matching columns is measured using relative entropy (also referred to as Kullback-Liebler distance; refer to equation 1) of corresponding and matching columns. It is worth noting that our merging procedure is general enough, so different alignment tools or similarity measures could be used instead of *Clustalw* and relative entropy respectively. For each set of fragment clusters (C_i and C_j in the scenario explained above), the following steps are carried out:

1. Matching columns: Assume that MSA profiles P_i and P_j for fragment clusters C_i and C_j have already been generated (as explained in section 3.1). Also assume the combined profile is called P_m. We can easily identify matching columns in P_i and P_j by looking at P_m as shown in Figure 2. We number matching columns in P_i and P_j with index k. Let P_{ik} and P_{jk} denote matching columns with index k. Columns with more than C_{gap}% gaps are not matched.

2. Probabilistic Model and Column Distances: A probabilistic model is constructed for each matching column. It would be sensible to consider similar amino acid characters and match them appropriately while a model for each column is constructed. Similar characters can be defined by a scoring matrix; we used BLOSUM62 in our implementation. This technique of matching similar characters was successfully implemented in the SPLASH pattern discovery algorithm [4].

   For each matching column k, the Kullback Liebler distance d_k between P_{ik} and P_{jk} is computed using the equation shown below:

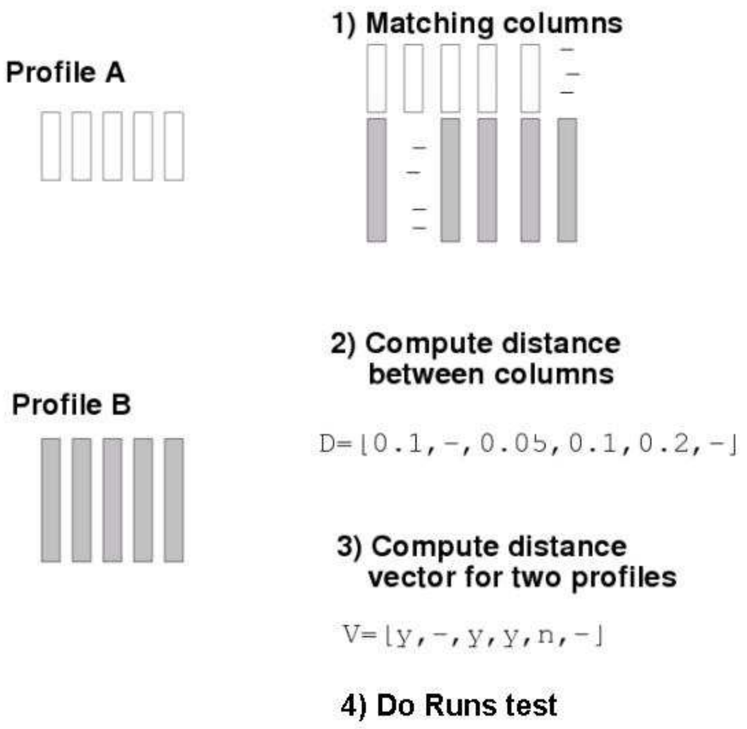


Figure 2: Steps involved in doing model comparison merge-test

$$d_k(P_{ik}||P_{jk}) = \sum_{a \in AA} P_{ik}(a) \times log(\frac{P_{ik}(a)}{P_{jk}(a)}) \tag{1}$$

where AA denotes the alphabet of all amino acid characters, and $P_{ik}(a)$ and $P_{jk}(a)$ can be computed simply by counting the number of a's in the column k. [1]

Since $d_k(P_{ik}||P_{jk})$ is not symmetric, we also compute $d_k(P_{jk}||P_{ik})$. Then we set the distance between P_{ik} and P_{jk} as below:

$$D_k(P_{ik}, P_{jk}) = Minimum\,(d_k(P_{ik}||P_{jk}), d_k(P_{jk}||P_{ik})) \tag{2}$$

3. Screening columns of random character composition: It is possible that both P_{ik} and P_{jk} are of random character composition, in which case

[1]To prevent division by zero, it is a common and necessary step to add a pseudo count to each character, per Laplace rule.

the distance between the two will be small *purely by chance.* To filter such cases, we compute distances, $d_k(P_{ik}||R)$ and $d_k(P_{jk}||R)$, of the two profiles P_{ik} and P_{jk} respectively, from a random model R. If both of these distances are small, i.e., both are close to a random model, then we set D_k to maximum possible distance (an arbitrarily high value).

4. Distance vector V_d of two profiles: Then we make a distance vector V_d where for each matching column k, the entry $V_d(k)$ is defined as below:

$$\text{if } D_k(P_{ik}, P_{jk}) < C_{distance},\ V_d(k) = `y';\ \text{otherwise } V_d(k) = `n' \tag{3}$$

Following this, we perform statistical analysis of the distance vector and details about that is given in the next sub-section.

3.3 Runs Test (From Statistics)

Once we have a distance vector consisting of 'y' and 'n' patterns, we subject the vector to Runs test [7], a well known technique in statistical circles. Simply put, we look for trend patterns and also figure out how random (or non-random) the patterns are. For example, it is quite intuitive while looking at the example runs shown below, to predict which of them is random.

Non-random run
. . . y . y . y . n . y . y . y . n . . .
Random run
. . . y . n . y . y . n . y . n . n . . .

If each pattern of 'y' or 'n' is considered to be an individual run, then too few runs or too many runs would generally constitute a non-random pattern. But in our case, we are only looking for small number of runs, i.e. too few runs. For 'N' values in the distance vector, we form all possible sequences consisting of N_1 'y' symbols and N_2 'n' symbols, such that $N_1 + N_2 = N$. We end up with a sampling distribution, which has a number of runs V associated with it. Then, mean and standard distribution are computed using equations 4 and 5. More detail on Runs test is available in [7].

$$\mu_V = \frac{2N_1N_2}{N_1 + N_2} + 1 \tag{4}$$

$$\sigma_V{}^2 = \frac{2N_1N_2(2N_1N_2 - N_1 - N_2)}{(N_1 + N_2)^2(N_1 + N_2 - 1)} \tag{5}$$

And finally, a Z-score, for distribution V, is computed using the usual formula:

$$Z = \frac{V - \mu_V}{\sigma_V} \tag{6}$$

We expect the follow up of Runs test to be a normal distribution. We apply a threshold based on the Z-score and its associated table of probabilities to decide if two fragment clusters need to be merged or not.

Brief Analysis
The above method of testing for merge'bility has some inherent flaws. That means the test is not robust as we would have liked. Some of the problems associated with the model comparison merge-test (as we inferred from our experiments as well as theoretical reasoning):

- Setting a threshold for Z-score computed using equation 6 is extremely hard and contentious. This is illustrated to a certain extent in section 7.
- The method does not take into account the fact that fragment cluster sizes may vary a great deal - building a probabilistic model using data of different sizes and comparing the models leads to unpredictable results.

These issues are addressed in the other two methods that we have developed. The distance based merge-test (section 4) addresses the above mentioned problems partially, while the least common ancestor based merge-test (section 5) is even more successful in taking care of them and churning out correct results.

4 Phylogenetic Tree - Evolutionary Distance based Merge-test

In this section, we explain the algorithm involved in generating neighbor joining phylogenetic trees for combined fragment cluster profiles and then how evolutionary distance, a property of the tree generated [9], can be used to test whether or not to merge two fragment clusters.

4.1 Neighbor Joining (NJ) based Phylogenetic Tree Generation

As was the case with the model comparison method explained in the previous section, we start with two sequence files corresponding to two cluster fragments. Then the following steps are carried out (note: the first two steps are identical to the ones on section 3.1):

1. Individual MSA profiles are generated for the two sequence files. We used *Clustalw* for this step.

2. The two individual profiles are fed to a profile alignment program to produce a single combined profile. Again *Clustalw* was used to generate the combined profile.

3. The combined profile generated in the previous step is then fed into a NJ phylogenetic tree generation program. We used *Clustalw*, once more, for the tree generation.

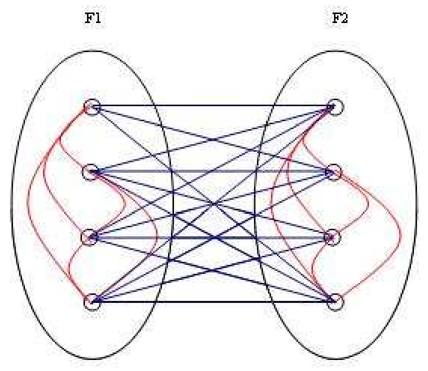
   It should be noted that our framework is flexible enough, so it should be straight forward to use other programs like the ones available in the *Phylip* suite to generate the trees.

4.2 Distance based test

Once a phylogenetic tree has been constructed for a combined profile of two fragment clusters C_i and C_j, we process the tree file to parse out a set of 'tree distance' values between pairs of sequences. This distance is nothing but the percentage evolutionary divergence for any two sequences under consideration. It is worth noting that the distance is length normalized as well [5]. Our decision on whether to merge or not is based on a very intuitive notion – and that is explained below.

If we consider two sequence clusters to be sets (in a venn diagram), then any score between sequences, like FASTA34 score or tree distance, corresponds to an edge between the members of the two sets. Edges between members of a single set are analogous to intra cluster-edges while ones between members of two clusters are analogous to inter-cluster edges in graph theory terms. This is illustrated in figures 3 and 4.

It is quite intuitive to expect intra-cluster edges to have shorter distances than inter-cluster distances. In our test, intra-cluster edges do not have much influence in deciding whether two clusters need to be merged or not, because they are hard to differentiate from inter-cluster sequence distances if both fragment clusters are indeed from the same super-cluster. We are more interested in the inter-cluster distances. The only hurdle is that though we know the fragment cluster a sequence belongs to, we do not know yet whether two fragments are sub-sets of a bigger super-cluster or if they are indeed two independent clusters; that is exactly what we are trying to figure out! So we have looked at the notion of inter-cluster distances in terms of "bad" distances and "good" distances. These two parameters could be made program parameters. Again the "good" inter-cluster distances get abstracted out along with the intra-cluster distances. But the "bad" distances stand out and that is what we have targeted our attention on. Once we figure the



Example of two fragment clusters with ***equal*** number of sequences.
Number of intra-cluster edges show in red = (6 + 6) = 12
Number of inter-cluster edges shown in blue = 16.

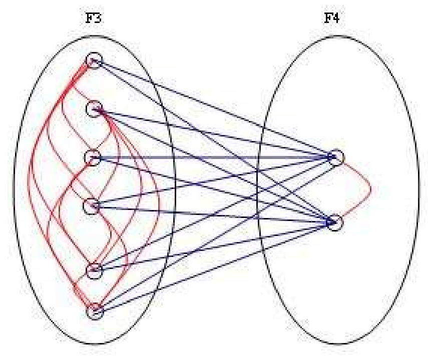
Figure 3: Equally sized fragment clusters - in venn diagram notation

number of "bad" distances, we compare it to a threshold value and reject any merge which has a number higher than the threshold.

There is one more consideration in deciding whether to merge or not. The number of inter-cluster edges is directly related to the product of number of sequences in the two individual fragment clusters. This gives raise to a need for normalization of our criterion i.e. "bad" distance. This is explained below.

Normalization:
As mentioned earlier, the number of edges across the two sets in figures 3 and 4 is directly dependent on the product of number of sequences in the two individual sets. If the difference in fragment cluster (set) size is huge, then the number of inter-edges is going to be skewed with respect to the net number of intra-cluster edges – which, as mentioned before, falls within the category of "good" distances.

To illustrate this, let us consider the two cases shown in figures 3 and 4. We assume first up that the two fragment clusters are indeed independent of each other. If figure 3 is considered, then we would ideally expect $(4*4) = 16$ edges across the sets (or clusters) to be "bad" and the number of intra-cluster distance values to be $(6+6) = 12$. But the case of figure 4, though we have the same *net* number of sequences, the number of inter-cluster distance values (which are "bad") would be only $(6*2) = 12$ while the number of intra-cluster distance values would be $(15+1) = 16$! Obviously, we need to normalize the number of "bad" edges before making a decision on whether if a merge is a



Example of two fragment clusters with ***unequal*** number of sequences.
Number of intra-cluster edges show in red = (6 + 6) = 16.
Number of inter-cluster edges shown in blue = 12.

Figure 4: Unequally sized fragment clusters - in venn diagram notation (Illustrates need for normalization when compared with figure 3)

bad one.

We used a very simple normalizing factor (or "normalizer") in our implementation. Consider fragment clusters C_i and C_j, we calculate a normalizer:

$$normalizer = \frac{|C_i|}{|C_j|} \text{ if } |C_i| > |C_j| \text{ ; inverse otherwise} \tag{7}$$

Brief Analysis

This method performed considerably better than the model comparison method explained in section 3 (please refer section 7 for details). But yet, there are some drawbacks attached to the distance based merge-test. The most obvious one has to do with the "normalizer". What constitutes a good normalizing factor? There are various possible answers to that question. We chose a simple "normalizer" (equation 7) because of its simplicity, but it is bound to be too strict thus giving raise to false negative results - rejection of possibly good merges. For this reason, it might be a good idea to come up with a better and more fair normalizing factor. The evolutionary distances are also dependent on the data size, this issue is better addressed by the LCA based merge-test explained in the next section.

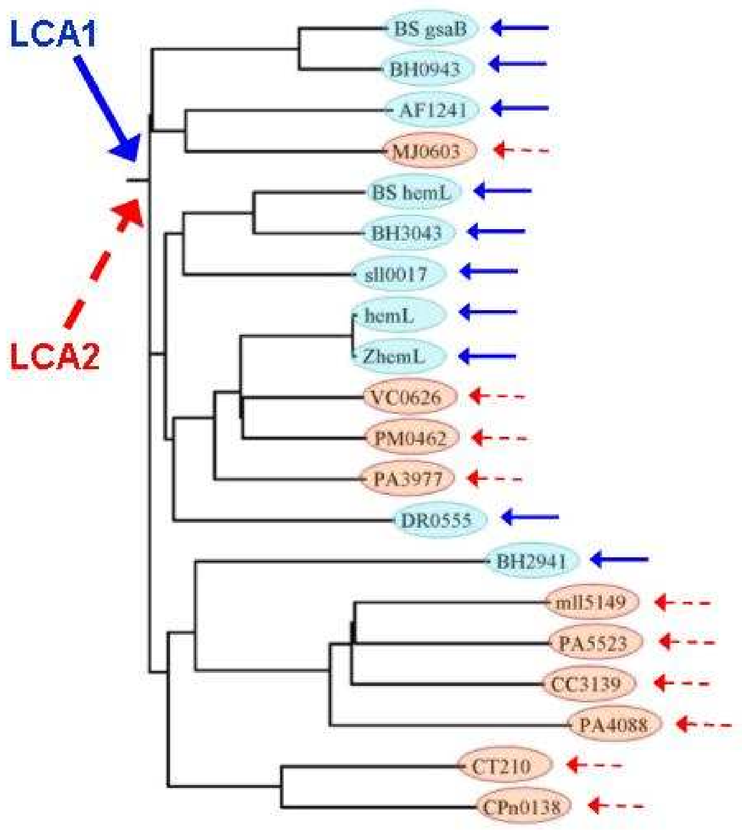


Figure 5: Phylogenetic tree for combined profile of two fragment clusters (Good merge)

5 Phylogenetic Tree - Least Common Ancestor (LCA) based Merge-test

In this section, we explain how the decision on whether, or not, to merge two fragment clusters can be taken by analyzing the coverage of the respective least common ancestor (LCA) sequences of the two fragment clusters.

5.1 Rooted Phylogenetic Tree Generation

The same procedure explained in section 4.1 is used to generate phylogenetic trees for the two fragment clusters in question, the only change being that a rooted tree in phylip format is generated instead of a neighbor joining tree.

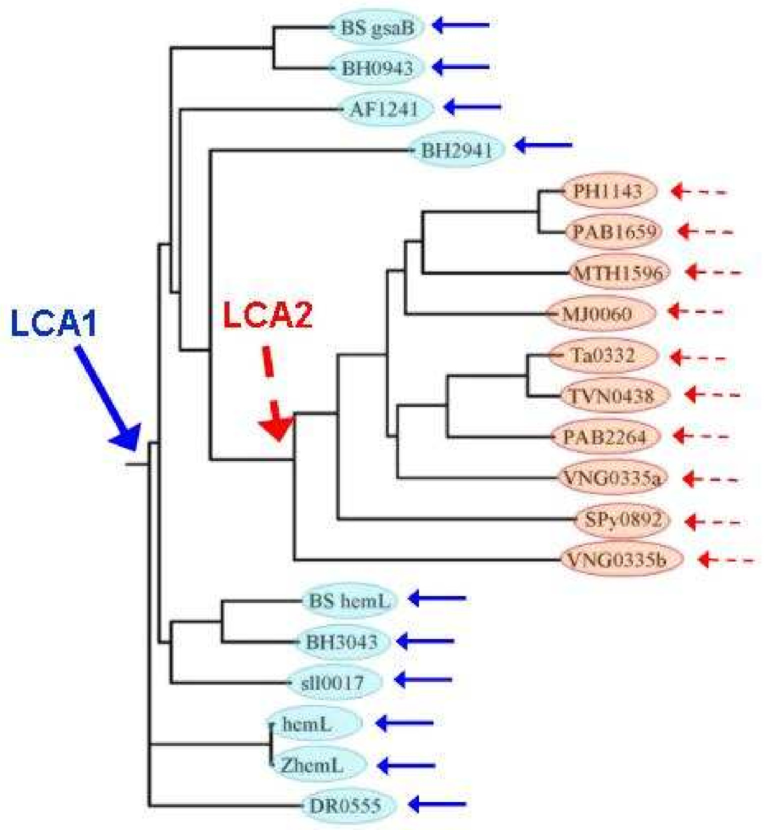


Figure 6: Phylogenetic tree for combined profile of two fragment clusters (Bad merge)

5.2 LCA based merge-test

Once a rooted phylip format phylogenetic tree has been constructed for the combined profile of two fragment clusters, we parse the tree file into a custom data structure. Let us assume fragment clusters C_i and C_j with sizes $|C_i|$ and $|C_j|$ respectively. We compute the least common ancestor for both fragment cluster's sequences, i.e., Compute $LCA(C_i)$ and $LCA(C_j)$.

Once this is done, we get a pretty good idea about whether the merge is a good one or not. If $LCA(C_i) == LCA(C_j)$ then, we argue it is a good merge. This is an easy case, and we proceed to the next set of fragments. An example of such a simple case is shown in figure 5: Both the fragment clusters (represented by blue and red colors respectively) share a common

LCA indicated by the thick blue and red (dashed) arrows[2].

But if the LCAs are not the same for C_i and C_j, then does that mean the merge is a bad one? The short answer to that question is ***no***, why it is so is explained below.

Coverage of LCA (and cross-coverage)

Let us hypothesize that not having a common LCA means the merge-test should return false. In a majority of cases, this assumption might be true. For example, figure 6 shows a case in which it is too obvious why the merge is not a good one. The blue cluster's LCA is the root of the tree (indicated by a blue arrow), while the red cluster has a totally independent sub-node as its LCA (indicated by a thick red dashed arrow). In such simple cases, we could reject the merge - but that would be possible only by visual inspection if we want to avoid false negatives as much as possible. But our framework's objective is to automate the whole process once parameters have been provided.

Now let us consider another example case in which the above hypothesis clearly does not hold true. Consider figure 7, in which again the two fragment clusters are represented by blue and red colors respectively. It is easy to see that the red cluster's LCA, indicated by a thick red (dashed) arrow, actually has a few sequences from the blue cluster. We use the term "coverage" to describe sequences under a certain node (possibly a LCA). Thus it is clear that in this case, the red cluster's LCA *covers* sequences from the blue cluster; another term we freely use is "cross coverage" - which again refers to the situation in which one cluster's LCA covers sequences from the other fragment cluster. It is clear we cannot reject merges outright if the two fragments do not share their LCAs, in cases like the one shown in figure 7.

To take care of such cases, we decided to check for "coverage" instead of checking for a common LCA before deciding on merge'bility of two fragment clusters. The test we perform is as follows:

$$\text{Check if } coverage(LCA(C_i)) > |C_i| \text{ given } |C_i| < |C_j| \qquad (8)$$

$$\text{Conversely, check if } coverage(LCA(C_j)) > |C_j| \text{ given } |C_j| < |C_i| \qquad (9)$$

If either of the conditions in equations 8 or 9 is true, then the merge-test returns true. Otherwise it rejects the merge as a bad one.

Brief Analysis

This method performed the best of the three merge-test methods discussed

[2]For the benefit of readers, we have also used *dashed* arrows for any thing associated with the cluster represented by color red

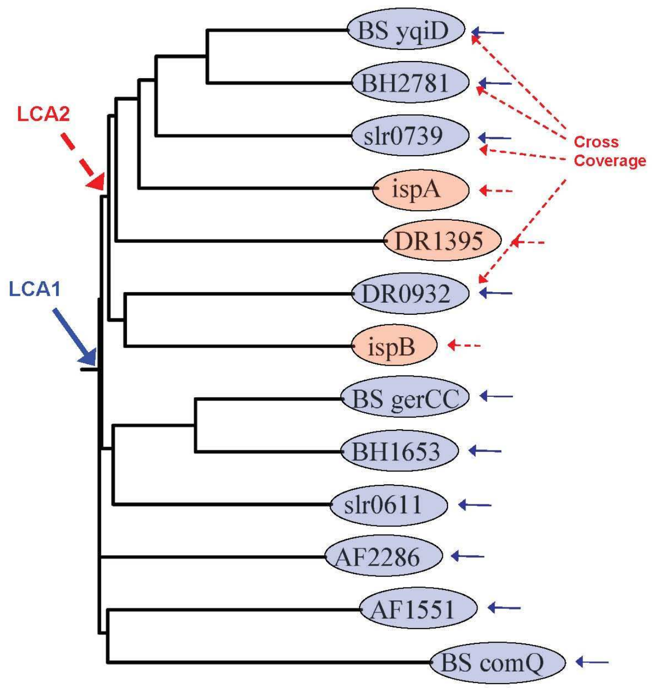


Figure 7: Phylogenetic tree for combined profile of two fragment clusters (Good merge – but unclear from just looking at LCAs; Looking at coverage of each LCA gives better idea)

in this document. It addresses the threshold setting problem, which was especially a drawback in the model comparison based method - this method does not have that problem since the coverage test is based on a clearly demarcatable criterion (i.e. is the coverage ratio greater than one?). The only drawback we can think of, in case of the LCA method, is if the fragments are extremely small in size ($number(sequences) <= 2$) then it can be expected to fail since it is really hard to have two different LCAs for a set of four or less sequences! This drawback is very minor though, since merging such small fragments does not decrease overall fragmentation all that much.

6 Implementation

We implemented the SeqClusMerge framework described in section 2 along with the merge-tests explained in sections 3, 4, 5, using a combination of Perl modules, C programs and Unix shell scripts. The framework is implemented as follows: Users who want to use BAG, to cluster biological sequences, can use our interactive framework; they can choose a cluster merging technique, then they can set various program parameters and then let the programs do the merge tests for them.

It should be noted that users can do their clustering using other algorithms as long as they can bridge data format to our framework's requirements. At present, SeqClusMerge expects clustering results in a specific format and also a set of 'merge suggestions'. The format of these things is just a question of changing parse-regexs, which should be quite straight forward. Once the data is supplied to the framework along with the choice of method for merging, it does the rest – producing MSA profiles, trees, etc. as per the requirements of the respective method. Then it prints a file with the merge candidates; then another program takes over to do some further post-processing. Finally, the user gets a clustering result (in BAG clustering algorithm's [1] output format) which includes merged fragment clusters and the rest of the original clustering result.

An open source implementation of SeqClusMerge is available up on request.

7 Experimental Results

We experimented on our merge-tests with test data that was constructed by extracting sequences from the COG database [8]. Essentially all we did was: select sets of two COG families and then construct 'good' & 'bad' test cases. The former contained sequences from just one of the chosen COG families while the latter contained sequences from both the chosen COG families.

The initial step, even before the construction of 'good' and 'bad' test cases, was to choose the COG families; we performed Pfam searches [10] on various COG families of interest and then chose families in increasing order of difficulty for our test data sets. Thus test dataset 1 includes two families which are evolutionarily very distant from each other - which means classifying them or testing merge'bility is expected to be very easy. Test dataset 4, on the other hand, includes two COG families which are reported as *one* family by the Pfam database, which indicates that the family got split *only due to human curation* of the COG database. Thus this dataset is very hard to test for merge'bility.

The results of our experiments are tabulated below. The two COG families used and then the size of the data extracted out for each of those are listed. The expected and observed merge-decisions are listed for each row. The tables below clearly show how robust each method is and also what their weaknesses may be.

Brief note about the formatting of data in the tables

- The words 'Good' and 'Bad' are used in the context of a merge-test - whether a particular dataset is supposed to be a good merge or not.
- <u>Underlined</u> entries indicate an excellent result - could be a case in which the merge-test achieved the expected outcome inspite of a hard dataset being used, etc or a case in which a merge-test produced correct results for the entire dataset.
- **Bold** entries indicate a wrong observed result i.e. the merge-test failed to achieve the expected outcome.
- *Italicized* entries indicate a case in which the merge-test in question produced the expected outcome because of parameter tweaking i.e. the results could not be generalized. This is mostly applicable to the model comparison based merge-test in which the z-score threshold had to be tweaked for each *dataset* (constant for various sizes of one data-set though)

Dataset 1: COG 0001 (size: 35) and 0005 (size: 30)

$n(F1)$	$n(F2)$	Expected	Model Comp.	Phy.Dist.	Phy.LCA
10	10	Good	Good	<u>Good</u>	<u>Good</u>
10	10	Bad	Bad	<u>Bad</u>	<u>Bad</u>
10	5	Good	Good	<u>Good</u>	<u>Good</u>
10	5	Bad	*Bad*	<u>Bad</u>	<u>Bad</u>
10	3	Good	Good	<u>Good</u>	<u>Good</u>
10	3	Bad	*Bad*	<u>Bad</u>	<u>Bad</u>
4	2	Good	Good	<u>Good</u>	<u>Good</u>
4	2	Bad	*Bad*	<u>Bad</u>	<u>Bad</u>
3	3	Good	Good	<u>Good</u>	<u>Good</u>
3	3	Bad	*Bad*	<u>Bad</u>	<u>Bad</u>

Dataset 2: COG 0142 (size: 74) and 0183 (size: 116)

$n(F1)$	$n(F2)$	Expected	Model Comp.	Phy.Dist.	Phy.LCA
10	10	Good	*Good*	Good	<u>Good</u>
10	10	Bad	*Bad*	Bad	<u>Bad</u>
10	5	Good	*Good*	**Bad**	<u>Good</u>
10	5	Bad	*Bad*	Bad	<u>Bad</u>
10	3	Good	*Good*	**Bad**	<u>Good</u>
10	3	Bad	**Good**	Bad	<u>Bad</u>
4	2	Good	*Good*	**Bad**	<u>Good</u>
4	2	Bad	*Bad*	Bad	<u>Bad</u>
3	3	Good	*Good*	**Bad**	**Bad**
3	3	Bad	*Bad*	Bad	Bad

Dataset 3: COG 0380 (size: 15) and 0383 (size: 13)

$n(F1)$	$n(F2)$	Expected	Model Comp.	Phy.Dist.	Phy.LCA
10	10	Good	*Good*	Good	<u>Good</u>
10	10	Bad	**Good**	Bad	<u>Bad</u>
10	5	Good	*Good*	**Bad**	<u>Good</u>
10	5	Bad	**Good**	Bad	<u>Bad</u>
10	3	Good	*Good*	**Bad**	<u>Good</u>
10	3	Bad	**Good**	Bad	<u>Bad</u>
4	2	Good	*Good*	<u>Good</u>	**Bad**
4	2	Bad	**Good**	<u>Bad</u>	Bad
3	3	Good	*Good*	<u>Good</u>	Good
3	3	Bad	**Good**	<u>Bad</u>	**Good**

Dataset 4: COG 0160 (size: 79) and 0161 (size: 49) ***[Same family according to Pfam search]***

$n(F1)$	$n(F2)$	Expected	Model Comp.	Phy.Dist.	Phy.LCA
10	10	Good	**Bad**	Good	Good
10	10	Bad	**Good**	**Good**	Bad
10	5	Good	**Bad**	Good	Good
10	5	Bad	**Good**	**Good**	Bad
10	3	Good	*Good*	Good	Good
10	3	Bad	**Good**	Bad	Bad
4	2	Good	*Good*	Good	Good
4	2	Bad	**Good**	**Good**	Bad
3	3	Good	*Good*	Good	Good
3	3	Bad	**Good**	**Good**	Bad

8 Conclusion

We presented a robust framework SeqClusMerge to merge fragmented clusters bottom-up; we also described the modular design of the framework. Following that, three merge-tests, one using a statistical and machine learning techniques and the other two using properties of phylogenetic trees, were described. Experimental results prove the effectiveness of the SeqClusMerge framework.

Another point which can be inferred from our thesis work is that complicated and well-studied techniques can fail in the absence of domain knowledge - as is clear from the rather mediocre results of our model comparison based merge-test (section 3). On the contrary, simple techniques which have the backing of domain knowledge return excellent results - both our phylogenetic tree based merge-tests have biological domain knowledge built into them (in the form of substitution matrices, etc.).

Further work includes making the entire system available on the web, designing and implementing more complicated merge-tests based on phylogenetic trees, structural analysis and so forth.

References

[1] Kim S., Gopu A., BAG: A Graph Theoretic Sequence Clustering Algorithm, *Submitted to J.Computational Biol.* (2004).

[2] Liu AH, Califano A., CASTOR: Clustering Algorithm for Sequence Taxonomical Organization and Relationships., J.Comput Biol. (2003).

[3] Yona G., Levitt M., Within the twilight zone: A sensitive profile-profile comparison tool based on information theory, J.Molecular Biol. (2002).

[4] Califano A., SPLASH: Structural Pattern Localization Analysis by Sequential Histograms, Bioinformatics (2000).

[5] Higgins D., Thompson J., Gibson T.Thompson J.D., Higgins D.G., Gibson T.J.(1994): CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting,position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22:4673-4680

[6] Kim S., Gopu A., Cluster Utility: A new metric to guide sequence clustering, Indiana University School of Informatics Technical Report 116 (2004; URL http://www.informatics.indiana.edu/research/publications/publications.asp?id=14).

[7] Spiegel M.R., Stephens L.J., Schaum's outlines Statistics, Third Edition.

[8] Tatusov RL., Fedorova ND., Jackson JD., Jacobs AR., Kiryutin B., Koonin EV., Krylov DM., Mazumder R., Mekhedov SL., Nikolskaya AN., Rao BS., Smirnov S., Sverdlov AV., Vasudevan S., Wolf YI., Yin JJ., Natale DA., The COG database: an updated version includes eukaryotes. Bioinformatics (2003).

[9] Gopu A., Cluster Merging using Phylogenetic tree distances, L529 term project report (2004; URL http://biokdd.informatics.indiana.edu/ãgopu/reports/L529_report.pdf)

[10] HmmPfam Online Search (URL http://pfam.wustl.edu/hmmsearch.shtml)

Acknowledgement

I'd like to thank the following people for valuable suggestions and discussions: Dr. Sun Kim (Advisor), Dr. Jeong-Hyeon Choi, Zhiping Wang, staff and faculty members at School of Informatics and at Computer Science.

I'd also like to thank mom, dad, my bro, my grandparents and some of my friends for their support.