Fast Implementations of Markov Clustering for Protein Sequence Grouping*

László Szilágyi[1,2] and Sándor Miklos Szilágyi[3]

[1] Sapientia - Hungarian Science University of Transylvania,
Faculty of Technical and Human Science, Tîrgu-Mureş, Romania
lalo@ms.sapientia.ro
[2] Budapest University of Technology and Economics, Department of Control Engineering and Information Technology, Budapest, Hungary
[3] Petru Maior University of Tîrgu-Mureş, Romania

Abstract. Two efficient versions of a Markov clustering algorithm are proposed, suitable for fast and accurate grouping of protein sequences. First, the essence of the matrix splitting approach consists in optimal reordering of rows and columns in the similarity matrix after every iteration, transforming it into a matrix with several compact blocks along the diagonal, and zero similarities outside the blocks. These blocks are treated separately in later iterations, thus significantly reducing the overall computational load. Alternately, a special sparse matrix architecture is employed to represent the similarity matrix of the Markov clustering algorithm, which also helps getting rid of a severe amount of unnecessary computations. The proposed algorithms were tested to classify sequences of protein databases like SCOP95. The proposed solutions achieve a speed-up factor in the range 15-300 compared to the conventionally implemented Markov clustering, depending on input data size and parameter settings, without damaging the partition accuracy. The convergence is usually reached in 40-50 iterations. Combining the two proposed approaches brings us close to the 1000 times speed-up ratio.

Keywords: Markov clustering, bioinformatics, protein sequence classification, unsupervised classification.

1 Introduction

By definition, protein families represent groups of molecules with relevant sequence similarity [3]. Establishing protein families in large databases is one of the fundamental goals of functional genomics. A successful classification may contribute to the delineation of functional diversity of homologous proteins, and can provide valuable evolutionary insights as well [5]. Members of such protein families may serve similar or identical biological purposes [9]. Identifying these

* This work was supported by the Hungarian National Research Funds (OTKA) under grant no. PD103921, the Hungarian Academy of Science through the János Bolyai Fellowship Program.

V. Torra et al. (Eds.): MDAI 2013, LNAI 8234, pp. 214–225, 2013.

families is generally performed by clustering algorithms [6], supported by pairwise similarity or dissimilarity measures. Well established properties of some proteins in the family may be reliably transferred to other members whose functions are not well known [8].

TRIBE-MCL is an efficient clustering method proposed for protein sequence classification [5], based on Markov chain theory [4]. It assigns a graph structure to the protein database such a way that each protein has a corresponding node, while initial edge weights in the graph represent computed pairwise similarity values, obtained via BLAST search methods [1]. Clusters are then obtained by alternately applying two matrix operations called inflation and expansion.

In this paper we introduce two efficient approaches aimed to accelerate the execution speed of the algorithm, without damaging the outcome of the clusters. The first proposed approach optimizes the execution via splitting the similarity matrix into several smallers ones once the graph has been disintegrated into isolated subgraphs. The second one uses a special sparse matrix structure to model the similarity matrix, reducing the computational burden by eliminating the unnecessary computations with zeros. Further on, these two approaches are combined in a third one, which will be formulated after the numerical tests.

The remainder of this paper is structured as follows: Section 2 takes into account the functional details of the TRIBE-MCL algorithm. Section 3 presents the details of the proposed efficient TRIBE-MCL algorithms. Section 4 evaluates and discusses the efficiency of the proposed method. Section 5 presents the conclusions and gives some hints for further research.

2 Background

TRIBE-MCL is an iterative algorithm, which operates on a directional graph. Each of the n nodes of the graph represents a protein sequence from the set we wish to cluster, while each edge length S_{ij}, $i, j = 1 \dots n$, shows the similarity between protein sequences of index i and j, respectively. Edge lengths are stored in the $n \times n$ similarity matrix S. Initial edge lengths usually come from pairwise sequence alignment. During the iterations, S behaves as a column stochastic matrix, whose elements represent probabilities of transitions (evolution).

The TRIBE-MCL algorithm consists of two main operations, namely the inflation and expansion, which are repeated alternately until a convergence is reached, that is, the similarity matrix becomes invariant during a cycle:

1. Inflation has the main goal to differentiate among connections within the graph, favoring more likely direct walks along the graph in the detriment of less likely walks. It is computed via taking each element of the similarity matrix to the power of $r > 1$. The strength of this differentiation is controlled by the so called inflation rate r: large values express the preference of likely walks more severely, causing sudden ruptures within the graph, possibly not in the ideal place. Low inflation rates are more likely to yield smooth partitions, but the convergence may become rather slow.

2. Expansion operation is intended to reveal possible longer walks along the graph, to emphasize changes within the protein structures that happened in two or more evolutionary steps. Expansion in achieved via matrix multiplication, by taking similarity matrix S to the second power.

Auxiliary computations are also included in each iteration, in order to maintain the similarity matrix S as a symmetric column stochastic matrix.

Clusters are defined as connected subgraphs within the graph described by the similarity matrix, so a stable state of the similarity matrix means that the clusters don't change their contents during an iteration.

In a previous paper [14], we have proposed a series of generalizations of the conventional version of the TRIBE-MCL algorithm [5], e.g. time-variant inflation rate, generalized inflation scheme, singleton filter, etc. These changes brought slight improvements to the accuracy and efficiency of the algorithm.

3 Methods

In this paper we introduce two implementations of the TRIBE-MCL algorithm, with the aim of seriously reducing its computational load, without harming the accuracy of classification. We will test the proposed method on the proteins of the SCOP95 database.

3.1 The SCOP95 Database

The SCOP (Structural Classification of Proteins) database [12] contains protein sequences in order of tens of thousands, hierarchically classified into classes, folds, superfamilies and families [2]. The SCOP95 database involved in this study, is a subset of SCOP (version 1.69), which contains 11944 proteins, exhibiting a maximum similarity of 95% among each other. Pairwise similarity and distance matrices (BLAST [1], Smith-Waterman [13], Needleman-Wunsch [10], PRIDE [7], etc.) are available at the Protein Classification Benchmark Collection [11]. In this study we employ BLAST similarity measures, because that one suppresses low similarities, thus contributing to computational load reduction.

3.2 Matrix Splitting

Most of the computational load of the algorithm is caused by the matrix multiplication, which has a theoretical complexity of $\mathcal{O}(n^3)$. In order to reduce runtime, it would be beneficial at any time of the execution, to separate those proteins which no longer have any influence upon the others. This idea we employed in the previous paper [14], where we proposed to exclude the rows and columns of singletons from the similarity matrix in each iteration. This way we achieved $30\% - 50\%$ reduction of the overall processing time, depending on the percentage of singletons within the data.

In the following, we will formulate a more optimal separation scheme of clusters. Let us denote by Σ the initial set of proteins, which is intended to be

Data: similarity matrix $S = [s_{ij}]$ with $1 \leq i, j \leq n$
Result: reordering buffer R, number of clusters q, indexes of first elements of clusters $Q_1 \dots Q_q$

$m \leftarrow 0$;
$q \leftarrow 0$;
$M \leftarrow \Phi$;
while $m < n$ **do**
  Find smallest $i \in \{1, 2, ...n\}$ such that $i \notin M$;
  $m \leftarrow m + 1$;
  $R_m \leftarrow i$;
  $M \leftarrow M \cup \{i\}$;
  $q \leftarrow q + 1$;
  $Q_q \leftarrow m$;
  fifo.push(i);
  while fifo *not empty* **do**
    l = fifo.pop();
    for *each* $j \in \{1, 2, ...n\}$ *with* $j \notin M$ *and* $s_{lj} > 0$ **do**
      $m \leftarrow m + 1$;
      $R_m \leftarrow j$;
      $M \leftarrow M \cup \{j\}$;
      fifo.push(j);
    end
  end
end
$Q_{q+1} \leftarrow n + 1$;

Algorithm 1. The subgraph identification function

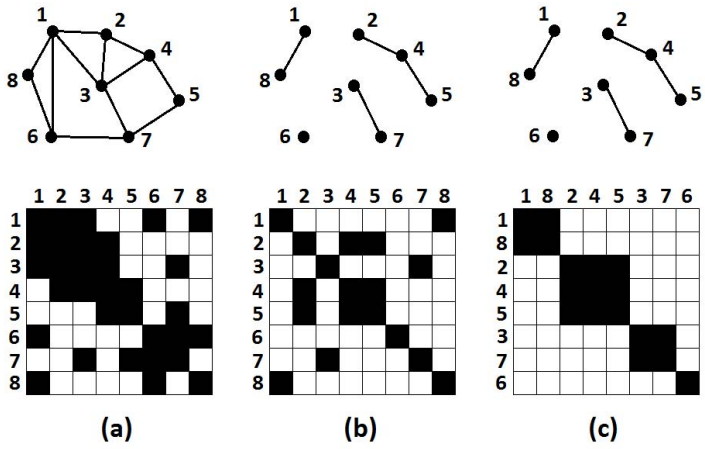

Fig. 1. Permutation of columns and rows: (a) an initial graph with several connections and the corresponding similarity matrix; (b) after a certain amount of iterations the graph breaks into pieces; (c) reordering the rows and columns in the matrix makes the similarity matrix contain non-zero blocks along the diagonal. At this given matrix splitting, the reordering buffer contains $R = [1, 8, 2, 4, 5, 3, 7, 6]$, the number of isolated subgraphs is $q = 4$, while the stored indexes of first elements are $Q = [1, 3, 6, 8]$.

classified. At any iteration t, we may look for isolated subgraphs in the graph represented by the similarity matrix S. Whenever we find a subset of proteins $\Sigma_1 \subset \Sigma$, corresponding to a connected subgraph isolated from the rest of the proteins ($s_{ij} = 0$, $\forall i \in \Sigma_1$ and $j \in \Sigma \setminus \Sigma_1$), in further iterations we may treat the proteins of Σ_1 separately from the others, because the rows and columns of S corresponding to these proteins will not interact with any other rows and columns. If we reorder all rows and columns of the similarity matrix S such a way, that isolated subgraphs are placed in consecutive rows and columns, we will have a similarity matrix formed by small square shaped blocks of nonzero elements placed along the main diagonal, and all other elements of the matrix will be zero.

In order to implement this idea, we need to define a reordering buffer R of size n, which will contain the permuted protein indexes corresponding to isolated subgraphs in the graph represented by S. Further on, we need a group buffer Q to store the indexes of initial elements of protein groups within the reordering buffer. The latter buffer will need a time-variant size of storage (denoted by q), but it will never exceed the limit of n items. Algorithm 1 presents the procedure of localizing isolated subgraphs within the graph. In this procedure, M represents the set of graph nodes already found during the process. The procedure sequentially looks for seed nodes which were not yet found and occupies the isolated subgraph using existing connections between nodes. Figure 1 exhibits the outcome of a column and row reordering, splitting an 8×8 matrix into four small matrices.

Having the isolated groups of nodes separated, we may reformulate the operations performed within each iteration as follows. For each square block along the diagonal of reordered matrix S, that is, for each $b \in \{1, 2, \ldots, q\}$, we consider the subset of proteins in the connected subgraph $\Sigma_b = \{R_{Q_b}, R_{Q_b+1}, \ldots, R_{Q_{b+1}-1}\}$ assuming that $Q_{q+1} = n + 1$, and then

– inflation is computed as:

$$s_{\alpha\beta}^{(\text{new})} = \left(s_{\alpha\beta}^{(\text{old})}\right)^r \qquad \forall \alpha, \beta \in \Sigma_b, \tag{1}$$

– expansion is given by the formula:

$$s_{\alpha\beta}^{(\text{new})} = \sum_{\gamma \in \Sigma_b} s_{\alpha\gamma}^{(\text{old})} s_{\gamma\beta}^{(\text{old})} \qquad \forall \alpha, \beta \in \Sigma_b, \tag{2}$$

– normalization is given by:

$$s_{\alpha\beta}^{(\text{new})} = s_{\alpha\beta}^{(\text{old})} \left(\sum_{\gamma \in \Sigma_b} s_{\gamma\beta}^{(\text{old})} \right)^{-1} \qquad \forall \alpha, \beta \in \Sigma_b, \tag{3}$$

– symmetry is approximated as:

$$s_{\alpha\beta}^{(\text{new})} = s_{\beta\alpha}^{(\text{new})} = \sqrt{s_{\alpha\beta}^{(\text{old})} s_{\beta\alpha}^{(\text{old})}} \tag{4}$$

$\forall \alpha, \beta \in \Sigma_b$, $\alpha < \beta$. After symmetrization, similarity values below ε are reduced to 0.

The proposed matrix splitting algorithm is summarized in Algorithm 2. Two parameters need to be set at the beginning: the inflation rate $r > 1$ and threshold ε around 10^{-3}. Generally 30-50 iterations are needed for a stable convergence. After 15 iterations most of the clusters are in their final form.

3.3 Sparse Matrix

The sparse matrix is a memory saving representation for matrices which contain a low amount of non-zero values. The sparse matrix stores only the non-zero values together with its coordinates (row and column). In our case, a non-zero element in the similarity matrix requires at least twice more bytes than an element of an two-dimensional array. Whenever using matrices of low density, employing sparse matrices will reduce the necessary storage space.

Sparse matrices also contribute to the efficiency of the algorithm. While computing the normalization of a column, zero elements are not added to the sum, thus reducing the number of additions. In fact, a zero element can only change to non-zero during the expansion. But also in case of the expansion, zero elements in the input do not affect the outcome of any element of the output matrix.

In a conventional sparse matrix structure, the non-zero elements of each column are stocked in a chained list, ordered by row coordinate. Thus the sparse matrix has an array of list head pointers, each one pointing to the first non-zero element of the corresponding column. Each non-zero element is represented by the structure $(row, value, next)$. The latter variable in the structure is a pointer to the next non-zero element in the column.

In a conventional sparse matrix, the inflation operation requires a single parsing of each column and thus the power computation is only performed for non-zero elements. The normalization needs to parse twice each column: first it computes the sum of each column and then it divides all non-zero elements by the sum of the column. Assuring matrix symmetry is more complicated, because it requires searching for the transposed for each non-zero element.

Expansion requires a new sparse matrix for the output. During the computation of the expanded matrix, the elements of each column are determined in such an order, that new non-zero elements are always placed at the end of the list. That is why, it is worth to have a pointer to the tail of the column list as well (see Fig. 2). Further on, as expansion is computed right after having made the similarity matrix symmetric, we may approximate the element s_{ij} as:

$$s_{ij}^{(\text{new})} = \sum_{k=1}^{n} s_{ik} s_{kj} \approx \sum_{k=1}^{n} s_{ik} s_{jk} \quad , \tag{1}$$

which is easier to compute as columns are way easier to parse than rows in this data structure.

Data: similarity matrix $S = [s_{ij}]$ with $1 \leq i, j \leq n$
Result: same similarity matrix S
$m \leftarrow n$; $q \leftarrow 1$; $M \leftarrow \Sigma$; $Q_q \leftarrow 1$; $Q_{q+1} \leftarrow n + 1$;
repeat
  for $b \in \{1, 2, ...q\}$ **do**
    $\Sigma_b \leftarrow \{R_{Q_b}, R_{Q_b+1}, \ldots, R_{Q_{b+1}-1}\}$;
    `Inflation`;
    for $\alpha, \beta \in \Sigma_b$ **do**
      $s_{\alpha\beta} \leftarrow s^r_{\alpha\beta}$;
    end
    `Normalization`;
    $S' \leftarrow \mathbf{0}$;
    for $\beta \in \Sigma_b$ **do**
      $z \leftarrow 0$;
      for $\gamma \in \Sigma_b$ **do**
        $z \leftarrow z + s_{\gamma\beta}$;
      end
      for $\alpha \in \Sigma_b$ **do**
        $s'_{\alpha\beta} \leftarrow s_{\alpha\beta}/z$;
      end
    end
    `Symmetry`;
    for $\alpha, \beta \in \Sigma_b$ *with* $\alpha < \beta$ **do**
      $z \leftarrow \sqrt{s'_{\alpha\beta} s'_{\beta\alpha}}$;
      if $z < \varepsilon$ **then**
        $z \leftarrow 0$;
      end
      $s_{\alpha\beta} \leftarrow z$; $s_{\beta\alpha} \leftarrow z$;
    end
    `Normalization again, as above`;
    `Expansion`;
    $S \leftarrow \mathbf{0}$;
    for $\alpha, \beta \in \Sigma_b$ **do**
      $z \leftarrow 0$;
      for $\gamma \in \Sigma_b$ **do**
        $z \leftarrow z + s'_{\alpha\gamma} s'_{\gamma\beta}$;
      end
      $s_{\alpha\beta} \leftarrow z$;
    end
  end
  `Call Subgraph Identification function`;
until *convergence occurs*;

Algorithm 2. Algorithm for Tribe-MCL via matrix splitting. S and S' are two instances of the similarity matrix, necessary for the correct handling of data

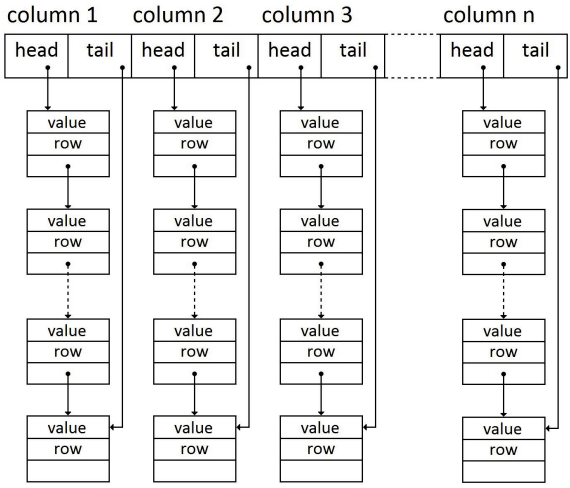

Fig. 2. Data structure used by the sparse matrix implementation

4 Results and Discussion

The main goal of protein clustering is to reveal hidden similarities among proteins. When evaluating the accuracy of the output, one can count the number of mixed clusters (those which contain proteins from two or more different families) and their cardinality. We have shown in the previous work [14], that the inflation rate is the main factor to influence the amount of mixed clusters. The approach proposed here computes exactly the same partitions as the conventional TRIBE-MCL, in a more efficient way. That is why the evaluation of accuracy is unnecessary in this study. The reader interested in accuracy details is referred to [14].

We have employed the proposed algorithms to classify either the whole set of 11944 proteins in the SCOP database, or selected subsets. At the selection of subsets, whole families were chosen from the hierarchical data structure, in order to keep all connections of each selected protein. The hierarchical structure of the SCOP database was only used to select input data and verify the final partition accuracy. Partitioning only uses the pairwise similarity data.

Fig. 3 summarizes some efficiency tests performed on a set of 908 proteins (all families from SCOP95 which have 11 to 14 proteins): varying the inflation rates between 1.3 and 2.0, the duration of each iteration was recorded and plot in this figure. In case of the matrix splitting approach, after only 4-6 iterations completed, the large connected block within the similarity graph is broken into small subgraphs, enabling us to compute subsequent iterations on very small matrices. Late iterations are performed approximately 1000 times quicker. Although the computation load stabilizes at a low level after the initial few iterations, the convergence of the output data requires around 40-50 cycles. Without this proposed efficient scheme, all iterations would need the same amount of computations as

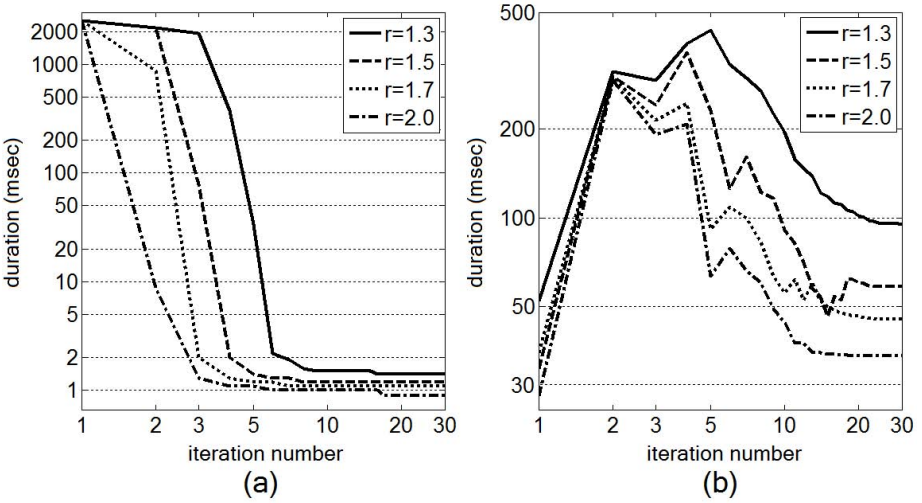

Fig. 3. The duration of the first 50 iterations, using the proposed method at various inflation rates, to classify 913 proteins from SCOP95: (a) matrix splitting approach; (b) sparse matrix implementation

the first one. This way we are able to approximate the speed-up ratio reached via fragmenting the similarity matrix. On the other hand, the sparse matrix implementation provides more efficiently computed initial loops, but the late iterations will require more computations than the matrix splitting approach. It is also visible that the duration of loops initially rises in the case of sparse matrix representation, which happens due to the growing amount of non-zero elements is the similarity matrix. After having performed 10-15 slower iterations, the duration of later iterations stabilizes at a low level.

The above remarked trends are also visible in Fig. 4, which presents efficiency results of the proposed methods on various data sets, using a fixed inflation rate $r = 1.5$. Data sets involved in the tests reported here were chosen as all protein families with cardinality between 10-18 (1795 proteins), 10-20 (2106 proteins), 8-30 (3887 proteins), 5-50 (6522 proteins), 3-99 (8920 proteins), and whole SCOP95 database (11944 proteins). All efficiency tests were run on PC with quad core Intel i7 processor running at 3.4GHz frequency.

Let us remark some trends identified from Figs. 3-4:

1. In every case, we needed a few iterations to break the similarity graph into several small isolated subgraphs. The larger the input data set, the more iterations are necessary. Using an inflation rate fixed at a reasonable value ($r = 1.5$) with the matrix splitting approach, a set of 1000 proteins requires 3 slow loops at the beginning, while at 5000 proteins, the fourth iteration is slow as well. One can expect that 10^5 proteins will need no more than 6-7 slow iterations. The trend of longer initial iterations is similar at the sparse matrix version, but it last a longer number of iterations.

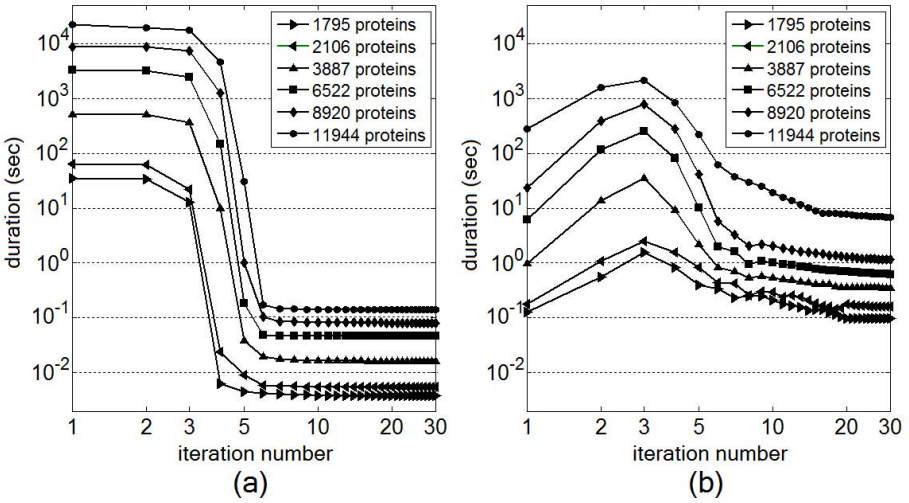

Fig. 4. The duration of the first 50 iterations, using the proposed method with various input data sets, plotted on logarithmic scale, using inflation rate $r = 1.5$: (a) matrix splitting approach; (b) sparse matrix implementation

2. Choosing a larger inflation rate reduces the number of slow iterations. However, it is not recommended to use very high inflation rates, because they yield small clusters in the output, which will hardly reveal any biologically relevant protein similarities.
3. Even though larger number of initial, longer lasting loops are performed by the sparse matrix version, this approach has the better overall runtime, because these initial loops have lower computational burden than the first loops of the matrix splitting approach. This is visible in Fig. 4, where the scales on the vertical axis of the two graphs (a) and (b) are identical.
4. Theoretically both approaches perform the same computations. If the input data set is the same, and the algorithm parameters are set equally, both approaches will lead to the same partition. Further on, we may also assert that after any number of iterations, the current partition of the two approaches are theoretically equivalent.
5. Based on the above assumption, we may combine the two approaches to provide a third, even more efficient one, which performs the initial iterations using the sparse matrix approach and switches to matrix splitting version thereafter, always using the version which performs the iterations quicker. Switching is performed when the largest connected subgraph is smaller than 5% of the total number of graph nodes. Table 1 refers to this switching method as combined approach.

Table 1 gives us a summary of speed-up ratios reached on input data of various sizes, at different inflation rates. These values were computed against the performance of the conventional TRIBE-MCL algorithm, which computes

Table 1. Speed-up ratios reached by the proposed efficient execution scheme

Number of proteins	Inflation rate	Speed-up ratio		
		Matrix splitting	Sparse matrix	Combined
908	1.3	17.87	18.65	80.4
908	1.5	26.16	30.96	150.9
908	1.7	36.67	40.36	184.1
908	2.0	49.07	52.24	339.4
1795	1.5	21.29	189.0	705.5
2106	1.5	21.48	212.7	773.1
3877	1.5	18.39	320.4	423.4
6522	1.5	18.03	327.4	356.7
8920	1.5	16.84	278.7	297.9
11944	1.5	17.30	197.7	224.9

Table 2. Amount of proteins in mixed clusters, out of 11944

Inflation rate	Proteins in mixed clusters at the level of				Total
	classes	folds	superfamilies	families	
1.30	446	245	123	1237	2051
1.35	110	89	118	771	1088
1.40	29	50	51	507	637
1.45	0	35	39	448	522
1.50	0	8	13	356	377
1.55	0	0	10	239	249
1.65	0	0	10	184	194
1.75	0	0	0	97	97
1.85	0	0	0	31	31
2.00	0	0	0	19	19
2.10	0	0	0	3	3
2.35	0	0	0	0	0

the whole similarity matrix in every iteration, encoded in a two-dimensional array. Even higher speed-up ratios could be reached using parallel computing.

The proposed efficient implementations enabled us to perform several tests on the whole SCOP95 database, to evaluate the amount of obtained mixed clusters depending on the algorithm's parameters. Mixed clusters are clusters where proteins from different families are present. We can further distinguish mixtures at the level of classes, folds, superfamilies, and families. For example, a cluster mixed at the level of folds contains proteins from different folds but all its proteins are from the same class. Table 2 presents the amount of proteins situated in mixed clusters for various values of the inflation rate. All these tests were run for threshold value $\varepsilon = 10^{-3}$.

As it was expected, the number of proteins in mixed clusters decreases as the inflation rate grows. Mixtures at the level of classes, folds, superfamilies and families vanish at $r = 1.44$, $r = 1.52$, $r = 1.73$, and $r = 2.35$, respectively.

5 Conclusions

In this paper we have proposed two efficient implementation schemes and a combined third efficient procedure for the graph-based TRIBE MCL clustering method, a useful tool in protein sequence classification. With these novel formulations, late iterations of the algorithm are performed up to thousands times quicker, and the overall runtime becomes shorter by 2-3 orders of magnitude, than in the conventional case. This speed-up is achieved without any damage of the classification accuracy.

References

1. Altschul, S.F., Madden, T.L., Schaffen, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search program. Nucleic Acids Res. 25, 3389–3402 (1997)
2. Andreeva, A., Howorth, D., Chadonia, J.M., Brenner, S.E., Hubbard, T.J.P., Chothia, C., Murzin, A.G.: Data growth and its impact on the SCOP database: new developments. Nucleic Acids Res. 36, D419–D425 (2008)
3. Dayhoff, M.O.: The origin and evolution of protein superfamilies. Fed. Proc. 35, 2132–2138 (1976)
4. Eddy, S.R.: Profile hidden Markov models. Bioinformatics 14, 755–763 (1998)
5. Enright, A.J., van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res. 30, 1575–1584 (2002)
6. Everitt, B.S., Landau, S., Leese, M., Stahl, D.: Cluster Analysis, 5th edn. John Wiley & Sons, Chichester (2011)
7. Gáspári, Z., Vlahovicek, K., Pongor, S.: Efficient recognition of folds in protein 3D structures by the improved PRIDE algorithm. Bioinformatics 21, 3322–3323 (2005)
8. Heger, A., Holm, L.: Towards a covering set of protein family profiles. Prog. Biophys. Mol. Bio. 73, 321–337 (2000)
9. Hegyi, H., Gerstein, M.: The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. J. Mol. Biol. 288, 147–164 (1999)
10. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453 (1970)
11. Protein Classification Benchmark Collection, `http://net.icgeb.org/benchmark`
12. Structural Classification of Proteins database, `http://scop.mrc-lmb.cam.ac.uk/scop`
13. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. 147, 195–197 (1981)
14. Szilágyi, L., Medvés, L., Szilágyi, S.M.: A modified Markov clustering approach to unsupervised classification of protein sequences. Neurocomputing 73, 2332–2345 (2010)