

Recent Advances in Improving the Memory Efficiency of the TRIBE MCL Algorithm

László Szilágyi^{1,2,3(✉)}, Lajos Loránd Nagy², and Sándor Miklós Szilágyi^{1,4}

¹ Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary

² Faculty of Technical and Human Sciences, Sapientia University of Transylvania, Tîrgu-Mureş, Romania

`lalo@ms.sapientia.ro`

³ Canterbury University of Christchurch, Christchurch, New Zealand

⁴ Department of Informatics, Petru Maior University of Tîrgu-Mureş, Tîrgu-Mureş, Romania

Abstract. A fast and highly memory-efficient implementation of the TRIBE-MCL clustering algorithm is proposed to perform the classification of huge protein sequence data sets using an ordinary PC. Improvements compared to previous versions are achieved through adequately chosen data structures that facilitate the efficient handling of symmetric sparse matrices. The proposed algorithm was tested on huge synthetic protein sequence data sets. The validation process revealed that the proposed method extended the data size processable on a regular PC from previously reported 250 thousand to one million items. The algorithm needs 10–20 % less time for processing the same data sizes than previous efficient Markov clustering algorithms, without losing anything from the partition quality. The proposed solution is open for further improvement via parallel data processing.

Keywords: Protein sequence clustering · Markov clustering · Markov processes · Efficient computing · Sparse matrix

1 Introduction

Markov clustering performs a hierarchical grouping of input data based on a graph structure and its associated connectivity matrix. It has a series of successful applications in the field of protein sequence and interaction network analysis [6], video processing [8], image processing [4], language modeling [9], community detection [12], human action categorization [18], and FPGA circuit design [3].

When the input data consists of protein sequences, each sequence will be associated to a node of the graph, and edge weights will be the pairwise similarity values computed with existing alignment methods like: Needleman-Wunsch [11],

Research supported by the Hungarian National Research Funds (OTKA), Project no. PD103921. S. M. Szilágyi is a Bolyai Fellow of the Hungarian Academy of Sciences.

Smith-Waterman [13], BLAST [1], and PRIDE [7]. In case of large-scale data sets, the BLAST similarity measures are preferred due to its sparse nature, which allows for quick and memory-efficient processing.

TRIBE-MCL [6] is a clustering method based on Markov chain theory [5], which assigns a graph structure to the protein set such a way that each protein has a corresponding node. Edge weights are stored in the so-called similarity matrix \mathbf{S} , which acts as a stochastic matrix. At any moment, edge weight s_{ij} reflects the posterior probability that protein i and protein j have a common evolutionary ancestor. TRIBE-MCL is an iterative algorithm, performing in each loop two main operations on the similarity matrix: inflation and expansion. Inflation raises each element of the similarity matrix to power $r > 1$, which is a previously established fixed inflation rate, favoring higher similarity values in the detriment of lower ones. Expansion, performed by raising matrix \mathbf{S} to the second power, is aimed to favor longer walks along the graph. Further operations like column or row normalization, and matrix symmetrization are included to serve the stability and robustness of the algorithm, and to enforce the probabilistic constraint. Similarity values that fall below a previously defined threshold value ε are rounded to zero. Clusters are obtained as connected subgraphs in the graph.

Handling matrices of several hundreds of thousand rows and columns is prohibitively costly in both runtime and storage space. Recent fast TRIBE-MCL implementations (e.g. [16,17]) significantly reduced runtime, but the memory limitations still exist. The main goal of this paper is to introduce a novel memory efficient TRIBE-MCL approach that uses only sparse matrices to store similarity values and a one-dimensional array to store intermediate values of a single row during expansion. This change can significantly upgrade the size of processable data sets, and may also improve processing speed. The proposed method will be validated using large synthetic protein data sets derived from the SCOP95 database [2,10,14] using our method presented in [15].

The remainder of this paper is structured as follows. Section 2 presents the details of the proposed memory efficient TRIBE-MCL algorithm. Section 3 evaluates the behavior of the proposed method and discusses the achieved results and outlines the role of each parameter, while Sect. 4 concludes this study.

2 Methods

Making the TRIBE-MCL processing of graphs of up to $n = 10^6$ nodes accessible for ordinary PCs requires adequate data structures to store the nonzero values on the similarity matrix (SM), and also an adequate organization of the algorithmic steps. Compared to our previous solution, it is necessary to redefine the operations performed during each cycle. The main proposals in this order are listed in the following.

At any moment of the processing, the stored SM is symmetric. The input SM is symmetric. Inflation, consisting in raising each element of the matrix to a given power, does not influence matrix symmetry. Expansion, consisting in raising the SM to the second power, also gives symmetric output if the input

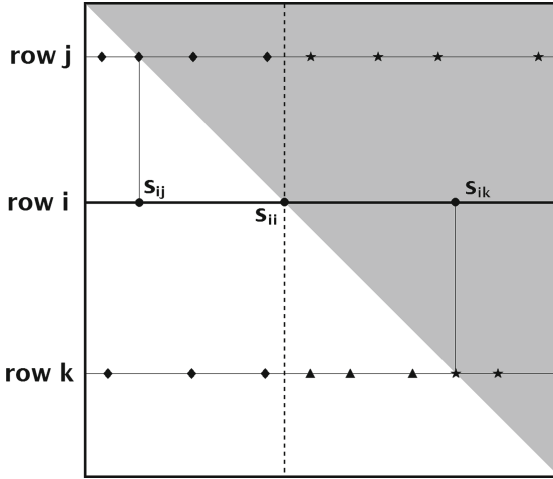


Fig. 1. Expansion using the employed data structures. Computing row i of the expanded matrix requires reading all rows with index $j < i$ and $k > i$ for which nonzeros s_{ij} or s_{ik} exist. Values indicated by stars (★) are directly accessible from the stored SM. Those drawn as triangles (▲) need information on the lower half of the matrix. Nonzeros indicated by diamonds (◆) are irrelevant at this step, as only that part of row i is to be computed, which falls in the upper diagonal half of the SM.

was symmetric. The only operation that pushes similarity values s_{ij} out of symmetry is the normalization of rows. However, each normalization is followed by a symmetrization, and by unifying these two steps, the intermediary output of normalization does not need to be stored.

Having the stored SM always symmetric, it is not necessary to store the whole matrix, only one half of it including the diagonal, as all further elements are deductible from these ones. Our choice was to store the upper diagonal half of sparse SM, thus in each row the first stored element is the one situated on the diagonal. All nonzero elements are stored in a row-wise order, together with their column coordinate. The number of nonzeros in each row, and a pointer to the first element of each row are stored separately.

Inflation is performed by parsing the array of stored nonzero elements and raising them to the power indicated by the inflation rate.

The unified normalization and symmetrization is performed in two steps. In a first step, the array of nonzeros is parsed and the sum of each row (Σ_i , $i = 1 \dots n$) is computed. Non-diagonal elements s_{ij} ($i \neq j$) in the stored upper half matrix are added to the sum of both rows i and j . In the second step, for each stored element s_{ij} , the normalized values become $\bar{s}_{ij} = s_{ij}/\Sigma_i$ and $\bar{s}_{ji} = s_{ij}/\Sigma_j$, respectively. The new s_{ij} value after symmetrization will be $\sqrt{\bar{s}_{ij}\bar{s}_{ji}}$, and will only be stored if it exceeds the similarity threshold ε .

Expansion is the only operation of TRIBE-MCL, which requires two instances of the similarity matrix: one for the input and one for the output of expansion.

Data: Initial similarity matrix \mathbf{S} , inflation rate $r > 1$, similarity threshold ε

Result: Final similarity matrix \mathbf{S}

repeat

Inflate(\mathbf{S})

Normalize-and-Symmetrize with elimination(\mathbf{S})

Normalize-and-Symmetrize(\mathbf{S})

Build auxiliary data structure for the lower diagonal half matrix

Expand(\mathbf{S})

until *convergence*;

Identify clusters

Algorithm 1. The steps of the proposed algorithm

Similarly to our solution given in [16], the expanded matrix is computed row by row, and during the expansion of a row, the computed output needs non-sparse storage. This requires a float valued array of n elements, denoted by E initialized with zero values for each row.

During expansion, the new row with index i , denoted by $\bar{\mathbf{S}}_i$ is obtained as follows:

$$E = \left(\begin{array}{cccc} \sum_{j \in \text{row}_i} s_{ij}s_{j1} & \sum_{j \in \text{row}_i} s_{ij}s_{j2} & \dots & \sum_{j \in \text{row}_i} s_{ij}s_{jn} \end{array} \right), \quad (1)$$

which in sparse matrix notations becomes

$$\bar{\mathbf{S}}_i = \sum_{j \in \text{row}_i} s_{ij}\mathbf{S}_j. \quad (2)$$

When the row is computed, its nonzero values falling in the upper half of the SM are transferred to the output sparse matrix.

The application for the expansion formula given in Eq. (2) is not trivial, due to the structural properties of the stored SM. Explanation in this order is given in Fig. 1. During the computation of row i of the squared matrix, the area of interest is only the part of row i situated behind the diagonal element s_{ii} , as the others will not be stored anyway. Consequently all nonzeros outside row i , on the left side of the vertical dotted line are irrelevant.

The algorithm parses all nonzeros in row i . Those situated in columns with index $j \leq i$ will require reading row j , whose elements of interest are all accessible in the upper half sparse matrix structure. On the other hand, nonzeros situated in columns with index $k > i$, imply reading row k whose elements of interest are partly outside the stored zone. This is why, before proceeding to the expansion operation, another sparse matrix structure is created to store information on the elements in the lower diagonal half of the matrix. The latter structure does not store the actual value of similarity, but stores information on the existence and position of the corresponding element in the upper diagonal half matrix structure. Creating the lower diagonal half matrix structure in each iteration requires two parsings over the upper half matrix structure. During the first,

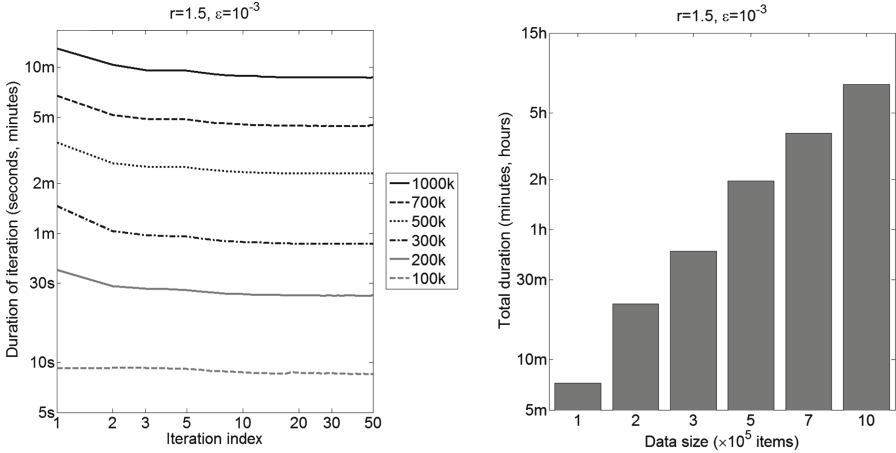


Fig. 2. Benchmark figures for a median density data sets of variable number of items, at constant inflation rate $r = 1.5$ at a constant value of similarity threshold $\varepsilon = 10^{-3}$: (left) duration of individual iterations; (right) overall runtime on fifty iterations.

we can count how many nonzeros are present in each row, and can allocate memory correspondingly and set row heads and element counts for all rows. During the second parsing, the actual information is extracted from the upper half nonzero elements (column information, and offset position with respect to row head) and added to the lower diagonal half matrix. This structure makes the lower part values, drawn in Fig. 1 as triangles (\blacktriangle), directly accessible, facilitating efficient data processing.

The proposed TRIBE-MCL algorithm is summarized in Algorithm 1.

3 Results and Discussion

The proposed method underwent a series of benchmark tests using synthetic test data sets of sizes ranging from 10 thousand to one million items. Data sets were generated using the method indicated in [15]. For each data size, 21 instances were created and the one with median density was chosen for the test.

Figure 2(left) exhibits the duration of each of the first fifty iterations in case of various matrix sizes, at inflation rate fixed at $r = 1.5$ and similarity threshold $\varepsilon = 10^{-3}$. As long as most nodes of the graph are connected together, namely in the first 5–6 loops, the computational load is somewhat higher and considerably falls thereafter, being virtually constant and low from the 10th loop. This difference between the duration of initial and late iterations gets more relevant as the data size grows. Figure 2(right) indicates the total runtime necessary to perform 50 loops in case of various data sizes. A comparison with our previous algorithms version [16] reveals that the main improvement is achieved in the size of data an ordinary PC can deal with. The new algorithm can easily handle a one-million-node graph, while the previous one was limited at 250 thousands. The execution time for the same amount of nodes in the graph is also reduced by 10–20 %.

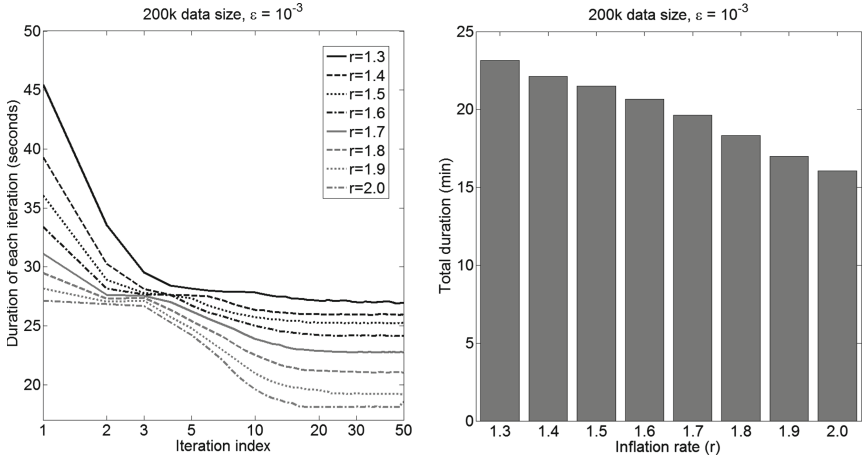


Fig. 3. Benchmark figures for a median density data set of 200k items, showing the influence of inflation rate r at a constant value of similarity threshold ε : (left) duration of individual iterations; (right) overall runtime on fifty iterations.

Figure 3 exhibits the effect of the inflation rate on the computational load of the algorithm. The input data here consisted of 200 thousand items having a similarity matrix of median density. Figure 3(left) shows the duration of individual iterations, while Fig. 3(right) indicates the total runtime of clustering performed in 50 loops. As the inflation rate grows, the similarity matrix becomes sparser and thus the total runtime and also the length of late iterations are shorter.

Figure 4 shows the influence of the similarity threshold ε on the computational load of the algorithm, using median density data sets of 200 thousand items. Figure 4(left) shows the duration of single iterations, while Fig. 4(right) exhibits the total runtime of clustering performed in 50 loops. A lower value of the similarity threshold keeps small similarity values longer in the matrix, and consequently the processing needs more time. The final outcome of clusters, and consequently the accuracy of clusters is hardly influenced by ε . Consequently ε should be kept high enough to support efficient data processing. However, if ε is too high, it may determine the algorithm to eliminate all non-zeros at once from a row of the matrix, leading to serious damage in the obtained partition.

All efficiency tests were carried out on a PC with quad core i7-4770 processor running at 3.4 GHz frequency and 16 GB RAM memory, using a single core of the microprocessor. The upper limit of processable data size is constrained by the memory of the employed computer. The main determining factor is the maximum number of nonzero values in the similarity matrix, usually reached after the first expansion operation. With the current version of the algorithm, an ordinary PC with 4 GB RAM can easily process a graph of 350,000 nodes, while one million nodes require an upper class PC with 16 GB RAM.

In case of huge input data sets, the approximative memory requirement is 11 bytes multiplied by the maximum number of nonzeros in the similarity matrix at

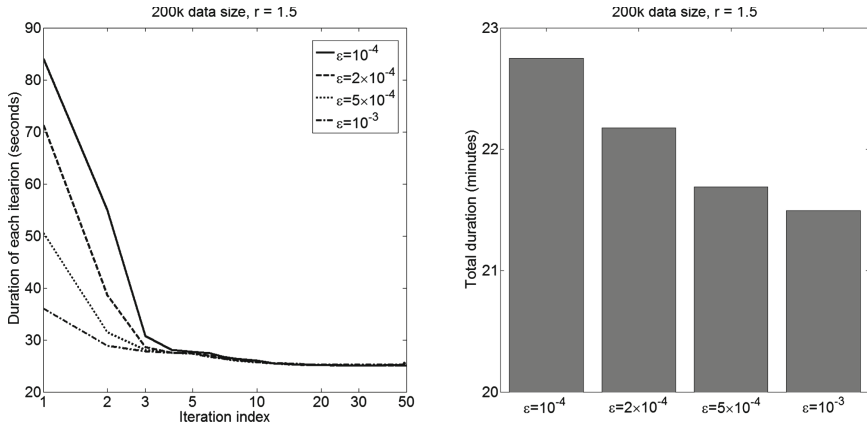


Fig. 4. Benchmark figures for a median density data set of 200k items, showing the influence of r and ε : (left) duration of individual iterations; (right) overall runtime on fifty iterations.

any time during the data processing. At a reasonably high inflation rate (e.g. $r \geq 1.3$), the maximum is likely to be reached right after the first expansion. Further improvement of the processing speed is achievable via parallel processing either with CPU or GPU. Further extension of the processable data size is achievable by temporary storage of data on solid state drives.

4 Conclusions

In this paper we have proposed an ultimate efficient approach to the graph-based TRIBE-MCL clustering method, a useful tool in protein sequence classification. The proposed approach proved extremely quick, and its memory needs are strongly reduced since previous versions. This novel implementation represents a major step of TRIBE-MCL towards handling huge data sets in reasonable time. Further enhancement of the algorithm's efficiency may be achieved via parallel implementation in CPUs or GPUs. Our future efforts will be focused on developing TRIBE-MCL algorithm versions to efficiently handle huge networks described by sparse matrices of unloadable size.

References

1. Altschul, S.F., Madden, T.L., Schaffner, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search program. *Nucl. 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. *Nucl. Acids Res.* **36**, D419–D425 (2008)

3. Dai, H., Zhou, Q., He, O., Bian, J.: Markov clustering based placement algorithm for island-style FPGAs. In: IEEE International Conference on Green Circuits and Systems, pp. 123–128. IEEE Press, New York (2010)
4. Dhara, M., Shukla, K.K.: Characteristics of restricted neighbourhood search algorithm and Markov clustering on modified power-law distribution. In: 1st International Conference on Recent Advances in Information Technology, pp. 520–525. IEEE Press, New York (2012)
5. Eddy, S.R.: Profile hidden Markov models. *Bioinformatics* **14**, 755–763 (1998)
6. Enright, A.J., van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. *Nucl. Acids Res.* **30**, 1575–1584 (2002)
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. Hospedales, T., Gong, S.G., Xiang, T.: A Markov clustering topic model for mining behaviour in video. In: 12th IEEE International Conference on Computer Vision, pp. 1156–1172. IEEE Press, New York (2009)
9. Keensub, L., Ellis, D.P.W., Loui, A.C.: Detecting local semantic concepts in environmental sounds using Markov model based clustering. In: IEEE International Conference on Acoustics Speech and Signal Processing, pp. 2278–2281. IEEE Press, New York (2010)
10. Lo Conte, L., Ailey, B., Hubbard, T.J., Brenner, S.E., Murzin, A.G., Chothia, C.: SCOP: a structural classification of protein database. *Nucl. Acids Res.* **28**, 257–259 (2000)
11. 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)
12. Pons, P., Latapy, M.: Computing communities in large networks using random walks. In: Yolum, I., Güngör, T., Gürgen, F., Özturan, C. (eds.) *ISCIS 2005*. LNCS, vol. 3733, pp. 284–293. Springer, Heidelberg (2005)
13. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. *J. Mol. Biol.* **147**, 195–197 (1981)
14. Structural Classification of Proteins database. <http://scop.mrc-lmb.cam.ac.uk/scop>
15. Szilágyi, L., Kovács, L., Szilágyi, S.M.: Synthetic test data generation for hierarchical graph clustering methods. In: Loo, C.K., Yap, K.S., Wong, K.W., Teoh, A., Huang, K. (eds.) *ICONIP 2014, Part II*. LNCS, vol. 8835, pp. 303–310. Springer, Heidelberg (2014)
16. Szilágyi, L., Szilágyi, S.M., Hirsbrunner, B.: A fast and memory-efficient hierarchical graph clustering algorithm. In: Loo, C.K., Yap, K.S., Wong, K.W., Teoh, A., Huang, K. (eds.) *ICONIP 2014, Part I*. LNCS, vol. 8834, pp. 247–254. Springer, Heidelberg (2014)
17. Szilágyi, S.M., Szilágyi, L.: A fast hierarchical clustering algorithm for large-scale protein sequence data sets. *Comput. Biol. Med.* **48**, 94–101 (2014)
18. Zhu, X., Li, H.: Unsupervised human action categorization using latent Dirichlet Markov clustering. In: 4th International Conference on Intelligent Networking and Collaborative Systems, pp. 347–352. IEEE Press, New York (2012)