ISSN 2321-9017
Volume 1, No.1, June - July 2013
International Journal of Bio-Medical Informatics and e-Health
Available Online at http://warse.org/pdfs/ijbmieh01112013.pdf

hiCLUMP: A hybrid Implementation of the CLUMP Algorithm for Clustering Microarrays Data

Dina Elsayad[1], Amal Khalifa[2], Essam Khalifa[3], El-Sayed M. El-horbaty[4]
Faculty of Computer and Information Sciences
Ain Shams University
Abbassyia, Cairo - Egypt
[1]dina.elsayad@fcis.asu.edu.eg, [2]amal@fcis.asu.edu.eg, [3]esskhalifa@eun.eg, [4]shorbaty@cis.asu.edu.eg

ABSTRACT

Microarrays technology allows us to measure the expression level of hundreds of thousands of genes simultaneously. The microarrays data analysis process involves various heavy computational tasks such as clustering. The clustering can be defined as partitioning a dataset into groups where objects in the same group are similar in somehow. CLUMP (clustering through MST in parallel) is one of the minimum spanning tree (MST) -based clustering techniques. It employed a parallel approach to reduce the MST construction time. An enhanced version of CLUMP (iCLUMP) was proposed to further improve the MST construction phase using cover tree data structure. Despite that modification, the MST construction phase is still a bottleneck since it is a time consuming task. Both CLUMP and iCLUMP are based on a distributed parallel computing model. Therefore, the objective of this paper is to study a different approach of enhancement using a hybrid parallel model. The proposed algorithm; hiCLUMP (hybrid CLUMP), considers utilizing multithreading on some of the distributed partitions suggested by the CLUMP algorithm. The experimental results on six different microarrays datasets show that the load balancing strategy used in hiCLUMP succeeded to decrease the MST construction in a range between 8% and 17% on 36 processing node. Moreover, the results showed that the hiCLUMP could not outperform the iCLUMP emphasizing that using another data structure is more effective than increasing the processing power of the underlying parallel machine.

Key words : Clustering , Microarrays , Minimum spanning tree, Parallel.

1. INTRODUCTION

Microarrays is one of the hottest fields related to bioinformatics. Microarrays is a multiplex lab-on-a-chip. Which is a 2D array on a solid substrate (usually a glass slide or silicon thin-film cell) that assays large amounts of biological material using high-throughput screening methods [1]. The types of that biological material determines the type of microarrays. Therefore, there are a number of types of microarrays including DNA microarrays [2], MMChips for surveillance of microRNA populations [3], Protein microarrays [4], Tissue microarrays [5], Cellular microarrays [6], Chemical compound microarrays [6], Antibody microarrays [7], and Carbohydrate arrays (glycoarrays) [1].

The biological material in DNA microarrays is DNA fragments, cDNA or oligonucleotide. The DNA microarrays providing a high-throughput experimental technique that can measure expression levels of hundreds of thousands of genes simultaneously. Expression level of the gene is estimated by measuring the amount of mRNA for that gene. A gene is active if it is being transcribed. More mRNA usually indicates more gene activity [8]. This makes the DNA microarrays of a special importance since it can be used not only for efficient screening and diagnosis of cancer in early stages of development but also for the identification of disease genes and therapeutic targets for human cancers. In other words, the goal of gene expression analysis is to discover subsets of genes that are associated with occurrence of certain diseases, for example breast cancer, leukemia or lymphoma by comparing gene expression profiling between tumor cell or tissues and corresponding normal cells or tissues in humans [9]. This allows biologists to infer gene function even when sequence similarity alone is insufficient to infer that functionality.

However, the analysis of the data resulting from microarrays remains a big challenge for the huge volume of data it produces. In addition, the data analysis process itself involves various computational tasks such as extracting differentially expressed genes, searching similar patterns of genes with a target gene, network analysis, clustering, and component analysis [10]. For example, the task of clustering aims to organize genes that those with similar expression patterns are grouped together to identifying biologically relevant groups of genes. It is believed that genes that behave similarly might have coordinated transcriptional response, possibly inferring common function or regulatory element.

Most of the clustering techniques are sequential. However with the massive volume of data produced by microarrays, parallel algorithms should be considered. The CLUMP algorithm (clustering through MST in parallel) proposed a parallel solution to the clustering problem. However, its MST construction phase was not efficiently handled. An enhanced version of CLUMP algorithm was proposed in [11]. According to the authors the enhanced CLUMP (iCLUMP) successes to improve the performance of the MST construction phase using the cover tree data structure. However, the implementation of both CLUMP and iCLUMP was based on distributed memory architecture.

In this paper, we study using a hybrid parallel approach. In this model the computation tasks are assigned to

distributed nodes where each node executes its own task on a number of threads.

The rest of the paper is organized as follows: section two provides a brief review on related clustering techniques, section 3 describes the proposed algorithm, the implantation and the results are discussed in section 4, and finally the conclusions are given in section5.

2. BACKGROUND AND RELATED WORK

The clustering problem is defined as partitioning a dataset into groups called clusters where each cluster contains a set of objects that have common characteristics [12]. In case of a noisy background the identified clusters don't necessarily cover all objects. In general, clustering algorithms can be classified into fuzzy clustering and hard clustering. The objects in fuzzy clustering can belong to more than one cluster with associated membership level, while in hard clustering the object belongs to only one cluster.

As shown in Figure. 1, the hard clustering can be further classified into sub-categories such as hierarchical clustering [13] [14] [15], partitional algorithms [16], density-based clustering [17] [18] and graph based clustering. In graph based clustering techniques clusters are determined using graph representation such as bi-partite graph [19] or minimum spanning tree (MST) [20] [21] [22] [23] [24] [11]. The main advantage of MST-based clustering is the little impact of the cluster boundary shape.

As shown in Figure. 2, MST-based clustering algorithms consist of three main steps: MST construction, inconsistent edges identification, removing inconsistent edges from MST to get the clusters. Inconsistent edges are the edges that may connect objects belong to different clusters. The main difference between different MST-based clustering algorithms is the measuring of edge inconsistency. One approach of edge inconsistency measure is the longest edge removal [20]. The drawback of this approach is the partitioning without sufficient evidence.

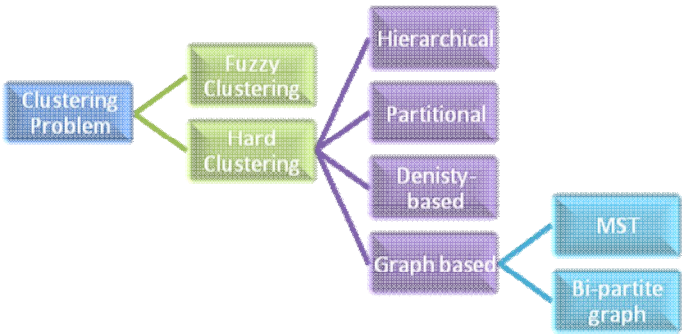

Figure 1: Types of clustering algorithms

Figure 2: MST-based clustering algorithm steps

Zhong et al. proposed a clustering method that is based on two rounds of minimum spanning trees [21], to avoid partitioning without sufficient evidence. In the first round a MST (T1) is constructed and in the second round another MST (T2) is constructed by considering only the edges that do not belong to (T1). Afterwards, both T1 and T2 are merged to construct the final MST (T). The clustering process works on the final MST (T). To ensure more evidence in each cut at least two edges must be removed, of which at least one edge comes from T1 and T2, respectively. Another MST- based clustering algorithm that is based on the direct clustering concept was proposed by Zhao and Zhang [22].

Xu et al. presented another MST-based clustering algorithm. The main idea of this approach is that each cluster corresponds to a sub-tree in the MST [23]. However, the size of the data this kind of clustering algorithms can effectively handle is limited. Therefore, Olman et al. [24] presented a parallel MST-based clustering algorithm called CLUMP (**clu**stering through **M**ST in **p**arallel) which identifies dense clusters in a noisy background. An enhanced version of CLUMP called iCLUMP was proposed in [11]. It enhances MST construction phase using a new data structure called cover tree [25]. The implementation of both CLUMP and iCLUMP was based on the distributed memory architecture. In this paper we study the performance of the CUMP algorithm using a hybrid parallel approach. The proposed algorithm; hiCLUMP (**h**ybr**i**d **CLUMP**), enhances the MST construction phase using threads.

3. THE PROPOSED ALGORITHM

As discussed before, MST-based clustering algorithms consist of three phases MST construction, inconsistent edges identification and clusters identifications. Where the MST construction phase is considered the computational bottleneck of the algorithm [24]. In CLUMP [24] the MST is constructed using Prim algorithm and Fibonacci heap [26]. The MST construction phase was done on parallel distributed memory architecture. According to their approach, the graph is partitioned into n subgraphs G_i and each pair of graphs G_i and G_j are combined into bipartite graphs B_{ij} (as listed in Algorithm 1) . The MSTs T_i and T_{ij} are constructed for each subgraph G_i and each bipartite graph B_{ij} respectively. After that, all the MSTs are combined into one graph G from which the MST T is constructed (Algorithm 2). The complexity for constructing each sub-graph G_i can be expressed using (1), while (2) describes the complexity for each bipartite graph B_{ij}.

$$O(|E_i|+|V_i| \log(|V_i|)) \quad (1)$$

$$O(|V_i||V_j|+(|V_i|+|V_j|) \log(|V_i|+|V_j|)) \quad (2)$$

CLUMP distributes the task of MST construction for each subgraph G_i and bipartite graph B_{ij} among all the processing nodes. The number of the processing nodes P depends on the number of partitions n where P = n(n+1)/2. More specifically, n processing nodes are used to construct MST T_i of subgraphs Gi, while the rest of the processing nodes are used to construct the MST T_{ij} of the bipartite graphs

B_{ij}. Since each bipartite graph B_{ij} is a combination of two subgraphs G_i and G_j, the time needed to construct T_{ij} is nearly double the time needed to construct T_i. At this point, hiCLUMP implements a hybrid parallel model on the MST construction of the bipartite graphs B_{ij}. In other words the nodes assigned that task of constructing the MST of bipartite graph B_{ij} will actually implements the Prim algorithm using threads (Algorithm 5). This load balancing strategy is expected to reduce the MST construction time and hence will contribute to the overall execution time of the algorithm.

The reset of the CLUMP algorithm would proceed as it is. That is the clusters identification phase (Algorithm 3 and 4) follows the MST construction phase. Notes that the clusters identification is a recursive process where the first cluster consists of the whole MST edges and each cluster is partitioned recursively until the cluster size is less than or equal to the minimum cluster size. The cluster is identified by two edge indexes (left index and right index). Where, the inconsistent edge is the longest edge in the cluster range. So, the cluster is partitioned into a left valley and a right valley and the process is repeated recursively on each one of them.

Algorithm 1: Graph Construction

Input	Dataset of N points in R^d
Output	G(V,E): undirected fully connected graph, where V is the vertices set and E is the edges set

Step 1:
    Add each point v as a vertex to V
Step 2: d(u, v) = the distance between each pair of points u and v
Step 3: Connect each pair of points (u, v) by edge e(u, v) where the weight of the edge is d(u, v)
Step 4: Add e(u,v) to E

Algorithm 2 : MST Construction

Input	*G(V, E):* undirected fully connected graph n: the partitions number
Output	MST T of G(V, E)

Step 1: Calculate partition size K = |V|/n
Step 2: Partition G into n sub-graphs $G_i = (V_i, E_i)$ each sub-graph of size K vertex
Step 3: If |V| is not evenly divided by K
        Add the remaining vertices to the last sub-graph
    End if
Step 4: Construct the MSTs for sub-graphs G_i and bipartite graphs B_{ij} in parallel
  Step 4.1: Assume that the distributed system consists of a set of processing nodes P_i ($1 \le i \le n(n-1)/2$)
  Step 4.2: Distribute the work among the nodes :
  Step 4.3: If Master Node:
    Step 4.3.1: x = n
    Step 4.3.2: For i = 0 to n-1
            Send G_i to P_i where i > 0
            For j = i+1 to n-1
                Send G_i and G_j to P_x
                Increment x
            End for
        End for
    Step 4.3.3: Construct MST T_0 for G_0
    Step 4.3.4: Wait till all worker nodes send their MSTs
    Step 4.3.5: Reduce and merge the whole MSTs in one graph M
    End if
  Step 4.4: If Worker node:
    Step 4.4.1: If receive a graph G_i
            Construct MST T_i using original Prim algorithm
            Send T_i to the master node
        End if
    Step 4.4.2: If receive a two graphs G_i and G_j
            Define a bipartite graph $B_{ij} = (V_i \cup V_j, E_{ij})$ where $E_{ij} \supset E$ is the set of edges between V_i and V_j

Construct MST T_{ij} for bipartite graph B_{ij} using the multithreading version of Prim (Algorithm 5)
Send T_{ij} to the master node
End if
End if
Step 5: Construct MST T of M

Algorithm 3: CLUMP Cluster identification

Input	N: the number of edges in MST min_size: cluster minimum size
Output	C: the set of the data clusters where each cluster C_i is defined by edge ranges {Li, Ri}

Step 1: Initialization:
let the first cluster C_0 consists of the whole MST edges
$L_0 = 1$, $R_0 = N$
Step 2: Call the routine Cluster_Partition (L_0,R_0,min_size)
Step 3: Build the hierarchical structure of the clusters in C
Step 4: Clean the clusters (C`)
Step 5: Rebuild the hierarchical structure of clusters in (C`)

Algorithm 4: Cluster_Partition Routine

Input	L_i, R_i: the left and right cluster ranges min_size: cluster minimum size
Output	C: set of clusters

Step 1: Let max_index be the index of edge with maximum weight in range {L_{i+1}, R_i}
Step 2: Let Left_valley contains all edges in the range {L_i , max_index -1}
Step 3: Add the cluster Left_valley to C
Step 4: If size of Left_valley >= min_size
Step 4.1: Cluster_Partition(L_i, max_index -1, min_size)
End if
Step 5: Let Right_valley contains all edges in the range { max_index, R_i}
Step 6: Add the cluster Right_valley to C
Step 7: If size of Right_valley >= min_size
Step 7.1: Cluster_Partition(max_index , R_i, min_size)
End if

Algorithm 5: Prim algorithm using Multithreading

Input	G(V, E): undirected fully connected graph
Output	T(V_T, E_T): Minimum spanning tree of G where V_T is the vertices set and E_T is the edges set

Step 1: Initialization:
Step1.1: Let V_T consists of an arbitrary node from V: $V_T = \{X\}$
Step1.2 Let E_T equal empty set: $E_T = \{\}$
Step 2: Parallel for i=1 to number of vertices -1
Step 2.1: Choose an edge e(u, v) with the minimal weight such that u is in V_T and v is not
Step 2.2: Add v to V_T: $V_T = V_T \cup v$
Add e(u, v) to E_T: $V_T = E_T \cup e(u, v)$
End

4. IMPLEMENTATION AND RESULTS

This section demonstrates some experiments to evaluate the performance of the proposed MST-based clustering algorithm hiCLUMP against the original CLUMP and its modified version iCLUMP. The CLUMP and iCLUMP are implemented using C++ with MPI. While, hiCLUMP is implemented using C++ with MPI and OpenMP. The experiments were conducted on a 36 processing nodes cluster where each node is Intel ® Xeon® CPU E5620 @ 2.40 GHZ. Six large microarrays datasets are used for comparison, five of them are breast cancer datasets and one ovarian cancer. These datasets are publicly available from the GEO database (http://www.ncbi.nlm.nih.gov/) through their accession numbers. Table 1 shows the accession number and the size of each dataset.

The run times of the three algorithms are measured for the six datasets using different numbers of processing nodes. Table 2 shows the runtime for the three algorithms applied on the 5th and 6th datasets. As indicated from the results, the hiCLUMP algorithm outperformed the original CLUMP achieving better runtime. For example, at p = 36 the achieved runtime was 40.67and 35.85 compared to 42.78 and 40 by CLUMP algorithm for the 5th and 6th dataset respectively. The results also show that iCLUMP algorithm outperformed both the original CLUMP and hiCLUMP achieving better runtime for all the tested values of the processing nodes p. For example, at p = 36 the achieved runtime was 39.36 second and 31.23 second compared to CLUMP algorithm for the 5th and 6th dataset respectively. Figures 3, 4 depict the runtime of each dataset separately.

Figure 5 shows the MST construction time of the three algorithms at p = 36, where only 28 nodes actually handled the bipartite graphs using 2 threads. The results show that hiCLUMP success to decrease the CLUMP MST construction time in range between 7% and 18%. While, Figure 6 shows the three algorithms overall execution time at p = 36. One more time, hiCLUMP is better than CLUMP providing up to 13% decrease in the overall execution time. Again the iCLUMP still outperforms both CLUMP and hiCLUMP.

Table 1: Microarrays datasets used for comparison

No.	Accession number	Size (number of genes x number of samples)
1	GSE6008	22283 x 104
2	GSE7390	22283 x 189
3	GSE2034	22283 x 256
4	GSE3494	22645 x 252
5	GSE9195	54675 x 78
6	GSE6532	54675 x 88

Table 2: The runtime of the tested algorithms measured in seconds as applied on two datasets with accession numbers GSE9195 and GSE6532

p	GSE9195			GSE6532		
	CLUMP	hiCLUMP	iCLUMP	CLUMP	hiCLUMP	iCLUMP
3	232.68	212.42	173.78	172.78	152.13	102.87
6	118.28	107.58	101.14	90.10	81.75	61.11
10	82.28	75.39	70.93	80.42	67.78	46.93
15	66.97	63.43	60.40	53.45	48.08	39.92
21	56.99	54.38	52.25	48.43	42.87	37.08
28	47.12	45.25	43.11	43.98	37.65	32.82
36	42.78	40.67	39.36	40.00	35.85	31.23

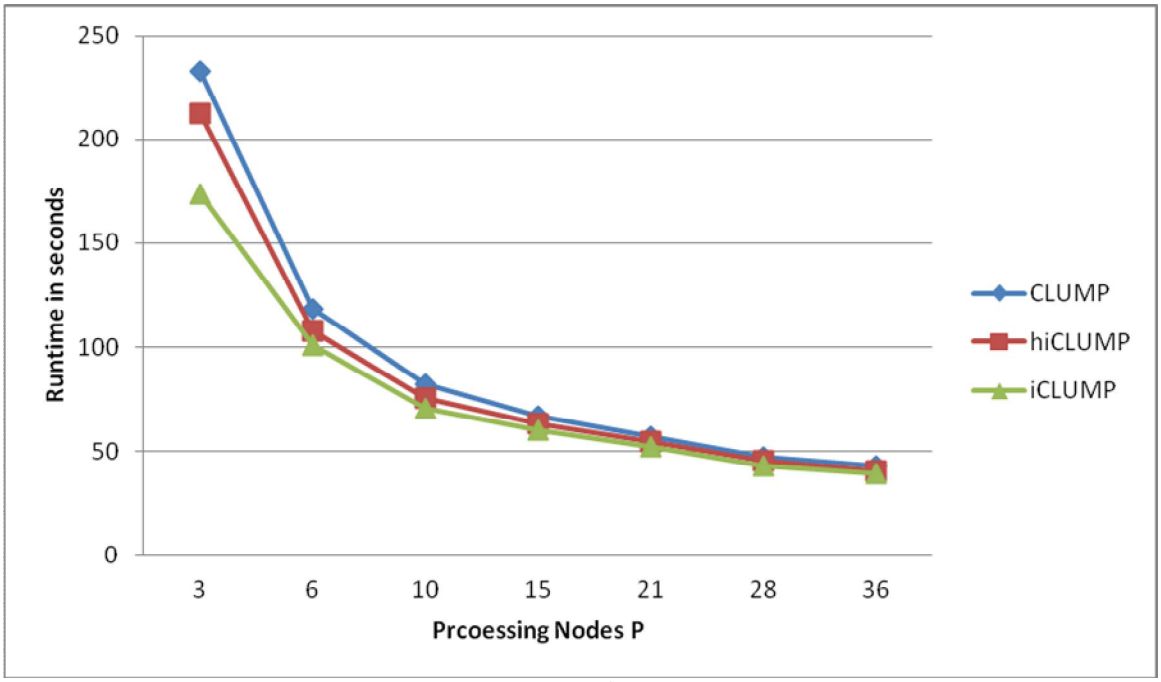

Figure 3: Runtime for dataset GSE9195

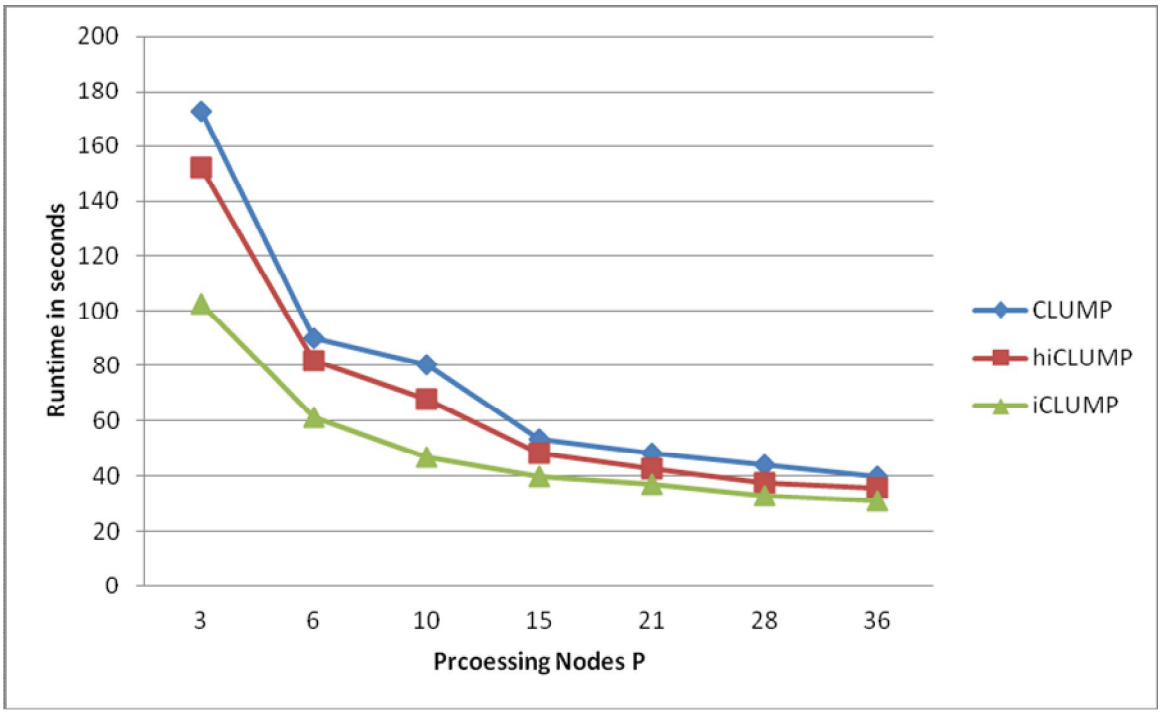

Figure 4: Runtime for dataset GSE6532

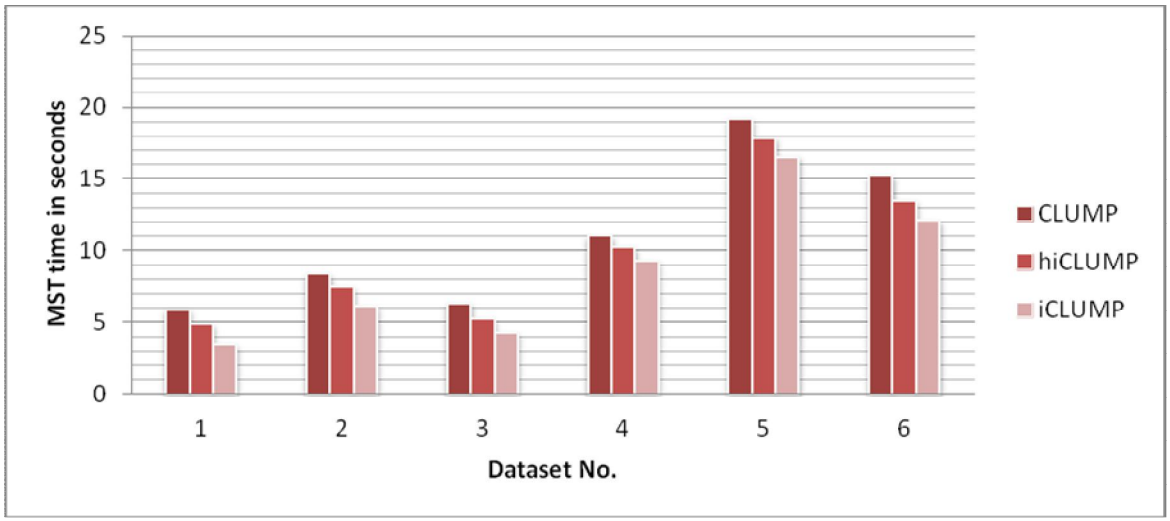

Figure 5: MST construction time of the tested algorithms measured in seconds as applied on all the datasets

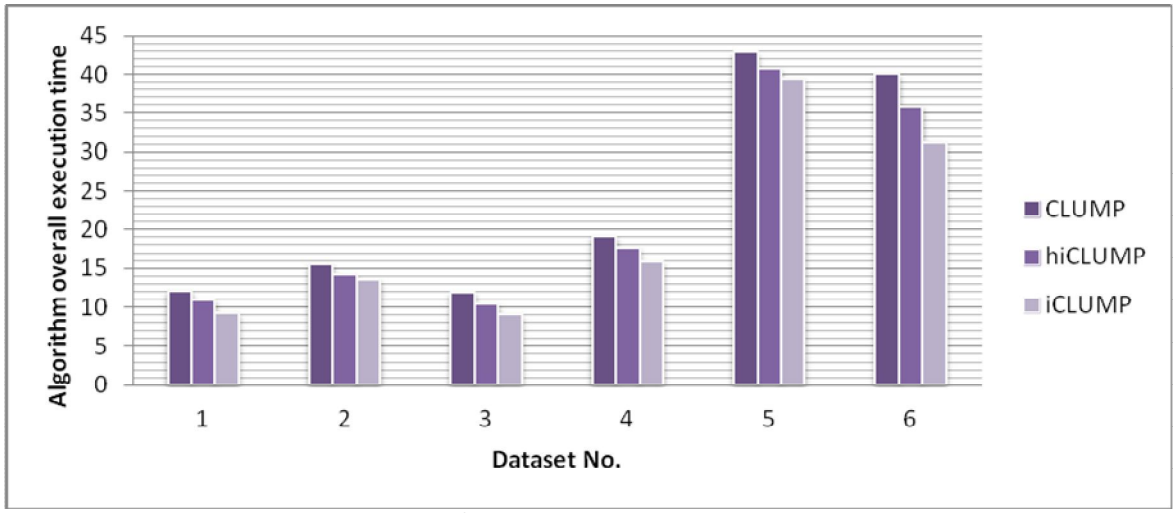

Figure 6: Algorithm overall execution time of the tested algorithms measured in seconds as applied on all the datasets

5. CONCLUSIONS AND FUTURE WORK

One of the MST-based clustering techniques is the CLUMP algorithm, which identifies dense clusters in a noisy background. iCLUMP is improved version of CLUMP, which enhances the CLUMP performance especially the MST construction phase. iCLUMP constructs MST using the cover tree data structure. This paper presents hiCLUMP which is another improved version of CLUMP algorithm. The idea of hiCLUMP is based increasing the processing power instead of applying another data structure. In other words, hiCUMP uses a hybrid parallel model in which the distributed and shared memory models are merged together. The MST construction phase in hiCLUMP is implemented using multithreading on the bipartite graphs since the size of each bipartite graph is double the size of an ordinary subgraph resulting an imbalance in the computational time. Experiments were conducted on a 36- processing nodes cluster powered by multithreading capability. The runtime and MST construction time were measured for the three algorithms using 6 microarrays datasets. At 36 processing nodes, the hiCLUMP overall runtime was 35.85 seconds instead of 40 for CLUMP using a dataset of size 54675 x 88. The MST construction time is decreased in range between 7% and 18%. However, the iCLUMP still outperforming both CLUMP and iCLUMP for all the tested cases. It implies that using an efficient data structure has much more impact on the performance than increasing the computational power of the parallel machine.

As future work we aim to apply different distance measure on the algorithms.

REFERENCES

1. Culf, A.S. and Cuperlovic-Culf, M. and Ouellette, R.J. Carbohydrate microarrays: survey of fabrication techniques. OMICS: A Journal of Integrative Biology, vol. 10, no. 3, pp. 289-310, 2006.

2. Schena, M. and Shalon, D. and Davis, R.W. and Brown, P.O. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. Science (Washington), vol. 270, no. 5235, pp. 467-470, 1995.

3. Meenakshisundaram, K. and Carmen, L. and Michela, B. and Diego, D.B. and Rosaria, V. and Gabriella, M. Existence of snoRNA, microRNA, piRNA characteristics in a novel non-coding RNA: x-ncRNA and its biological implication in Homo sapiens. Journal of Bioinformatics and Sequence Analysis, vol. 1, no. 2, pp. 031-040, 2009.

4. Stoevesandt, O. and Taussig, M.J. and He, M. Protein microarrays: high-throughput tools for proteomics. Expert Review of Proteomics, vol. 6, no. 2, pp. 145-157, 2009.

5. Kononen, J. and Bubendorf, L. and Kallionimeni, A. and Börlund, M. and Schraml, P. and Leighton, S. and Torhorst, J. and Mihatsch, M.J. and Sauter, G. and Kallionimeni, O.P. Tissue microarrays for high-throughput molecular profiling of tumor specimens. Nature Medicine, vol. 4, no. 7, pp. 844-847, 1998.

6. Ma, H. and Horiuchi, K.Y. Chemical microarray: a new tool for drug screening and discovery. Drug discovery today, vol. 11, no. 13-14, pp. 661-668, 2006.

7. Rivas, L.A. and García-Villadangos, M. and Moreno-Paz, M. and Cruz-Gil, P. and Gómez-Elvira, J. and Parro, V. A 200-Antibody Microarray Biochip for Environmental Monitoring: Searching for Universal Microbial Biomarkers through Immunoprofiling. Analytical Chemistry, vol. 80, no. 21, pp. 7970-7979, 2008.

8. Schena, M. and Shalon, D. and Davis, R.W. and Brown, P.O. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. Science (Washington), vol. 270, no. 5235, pp. 467-470, 1995.

9. Li, S. and Li, D. DNA microarray technology and data analysis in cancer research. World Scientific Pub Co Inc, 2008.

10. Yang, Y. and Choi, J.Y. and Choi, K. and Pierce, M. and Gannon, D. and Kim, S. BioVLAB-Microarray: Microarray Data Analysis in Virtual Environment. IEEE Fourth International Conference on eScience, 2008.

11. Elsayad, D. Khalifa, A. Khalifs, M.E. El-Horbaty, E.-S. An improved parallel minimum spanning tree based clustering algorithm for microarrays data analysis. 8th International Conference on Informatics and Systems (INFOS), May 2012, pp. DE-66,DE-72.

12. D. Dembele and P. Kanstner. Fuzzy C-means method for clustering microarray data. Bioinformatics, vol. 19, pp. 973-980, 2003.

13. Ivan G. Costa, Francisco de A.T. de Carvalho and Marcilio C.P. de Souto. Comparative Analysis of Clustering Methods for Gene Expression Time Course Data. Genetics and Molecular Biology, vol. 27, no. 4, pp. 623-631, 2004.

14. Carlos Cotta, Pablo Moscato. A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny. Biosystems, vol. 72, no. 1, pp. 75-97, 2003.

15. Sudip Seal, Srikanth Komrina, Srinivas Aluru. An optimal hierarchical clustering algorithm for gene expression data. Elsevier, 2004.

16. C.M. Bishop, Neural Networks for Pattern Recognition. Oxford Univ.Press, 1995.

17. De Bin, R. and Risso, D. A novel approach to the clustering of microarray data via nonparametric density estimation. BMC bioinformatics, vol. 12, no. 1, pp. 49-56, 2011.

18. McNicholas, P.D. and Murphy, T.B. Model-based clustering of microarray expression data via latent Gaussian mixture models. Bioinformatics, vol. 26, no. 12, pp. 2705 – 2712, 2010.

19. Kanungo, S. and Sahoo, G. and Gore, M.M. A Co-Clustering Technique for Gene Expression Data Using Bi-Partite Graph Approach. International Conference on Bioinformatics and Biomedical Engineering, 2010, pp. 1-5.

20. Jana, PK and Naik, A. An efficient minimum spanning tree based clustering algorithm. Proceeding of International Conference on Methods and Models in Computer Science, 2009, pp. 1-5.

21. Zhong, C. and Miao, D. and Wang, R. A graph-theoretical clustering method based on two rounds of minimum spanning trees. Pattern Recognition, vol. 43, no. 3, pp. 752-766, 2010.

22. Zhao, W.L. and Zhang, Z.G. An Improved Algorithm for Clustering Gene Expression Data

Using Minimum Spanning Trees. Applied Mechanics and Materials, vol. 29, pp. 2656-2661, 2010.

23. XY. Xu, V. Olman, and D. Xu. Clustering Gene Expression Data Using a Graph-Theoretic Approach: An Application of Minimum Spanning Tree. Bioinformatics, vol. 18, no. 4, pp. 526-535, 2001.

24. Olman, V. and Mao, F. and Wu, H. and Xu, Y. Parallel clustering algorithm for large data sets with applications in bioinformatics. IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol 6, no. 2, pp. 344-352, 2009.

25. A. Beygelzimer, S. Kakade, and J.C. Langford. Cover Trees for Nearest Neighbor. Proceedings of the 23rd International Conference on Machine learning, 2006, pp. 97–104.

26. Handbook of Discrete and Combinatorial Mathematics, K.H. Rosen, ed. CRC Press, 1999.