Parallel Clustering Algorithm for Large-Scale Biological Data Sets

Minchao Wang[1], Wu Zhang[1,2], Wang Ding[1], Dongbo Dai[1], Huiran Zhang[1], Hao Xie[3], Luonan Chen[1,4], Yike Guo[1,5], Jiang Xie[1]*

1 School of Computer Engineering and Science, Shanghai University, Shanghai, P.R.China, **2** High Performance Computing Center, Shanghai University, Shanghai, P.R.China, **3** College of Stomatology, Wuhan University, Wuhan, P.R.China, **4** Key Laboratory of Systems Biology, Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, Shanghai, P.R.China, **5** Department of Computing, Imperial College London, London, United Kingdom

Abstract

Backgrounds: Recent explosion of biological data brings a great challenge for the traditional clustering algorithms. With increasing scale of data sets, much larger memory and longer runtime are required for the cluster identification problems. The affinity propagation algorithm outperforms many other classical clustering algorithms and is widely applied into the biological researches. However, the time and space complexity become a great bottleneck when handling the large-scale data sets. Moreover, the similarity matrix, whose constructing procedure takes long runtime, is required before running the affinity propagation algorithm, since the algorithm clusters data sets based on the similarities between data pairs.

Methods: Two types of parallel architectures are proposed in this paper to accelerate the similarity matrix constructing procedure and the affinity propagation algorithm. The memory-shared architecture is used to construct the similarity matrix, and the distributed system is taken for the affinity propagation algorithm, because of its large memory size and great computing capacity. An appropriate way of data partition and reduction is designed in our method, in order to minimize the global communication cost among processes.

Result: A speedup of 100 is gained with 128 cores. The runtime is reduced from serval hours to a few seconds, which indicates that parallel algorithm is capable of handling large-scale data sets effectively. The parallel affinity propagation also achieves a good performance when clustering large-scale gene data (microarray) and detecting families in large protein superfamilies.

Citation: Wang M, Zhang W, Ding W, Dai D, Zhang H, et al. (2014) Parallel Clustering Algorithm for Large-Scale Biological Data Sets. PLoS ONE 9(4): e91315. doi:10.1371/journal.pone.0091315

Editor: Peter Csermely, Semmelweis University, Hungary

Received November 13, 2013; **Accepted** February 10, 2014; **Published** April 4, 2014

Funding: This research is partially supported by the Specialized Research Fund for the Doctoral Program of Higher Education [SRFDP 20113108120022], the Key Project of Science and Technology Commission of Shanghai Municipality [No. 11510500300], and the Major Research Plan of NSFC [No. 91330116]. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: jiangx@shu.edu.cn

Introduction

Data clustering is to group a set of objects in such a way that objects in the same group (cluster) have higher similarity with each other than those in the other groups (clusters). It is a common technology for data mining and analysis in many fields, such as pattern recognition, machine learning, bioinformatics and so on. Many novel clustering algorithms have been introduced to handle biological problems, including protein families/superfamilies detecting [1,2], metabolic networks analysis [3–5] and protein-protein interaction (PPI) networks analysis [6,7].

In the last decade, many clustering algorithms were proposed and widely applied to the biological researches [8–13] (see [14–16] for a comprehensive survey). One of the most successful clustering algorithms is the Markov Cluster algorithm (Tribe-MCL) [1]. In the original publication, it was used to detect the protein families in the protein-protein interaction networks based on the graph theory. The algorithm simulates the random walks within the graph by alternation of two operations, called the Expansion and the Inflation. Spectral clustering, was first introduced into the image processing [17]. It was recently applied to the protein sequence clustering problems [2]. Spectral clustering requires a long runtime, and the cluster number is required to be specified manually. In microbial community analysis, some classic clustering algorithms such as linkage, graph partition are taken to handle the difference between microbial sequences. The correlation between the comparison of the human microbiome and various disease can be extracted from the clustering results.

The rapid increment in biological data sets scale poses great challenges for sequential algorithms, and makes the parallel clustering algorithms more attractive [18–20]. Chen et al. implemented a parallel spectral clustering [18]; Ekanayake et al. applied the cloud technologies into the clustering [19]. Bustamam et al. designed the sparse data structure and implemented the sparse MCL algorithm on the GPU [20].

This paper focuses on the parallel biological clustering for large-scale data sets in the distributed system. A promising algorithm called Affinity Propagation [21] is considered in our work. In the biological researches, the affinity propagation algorithm has been

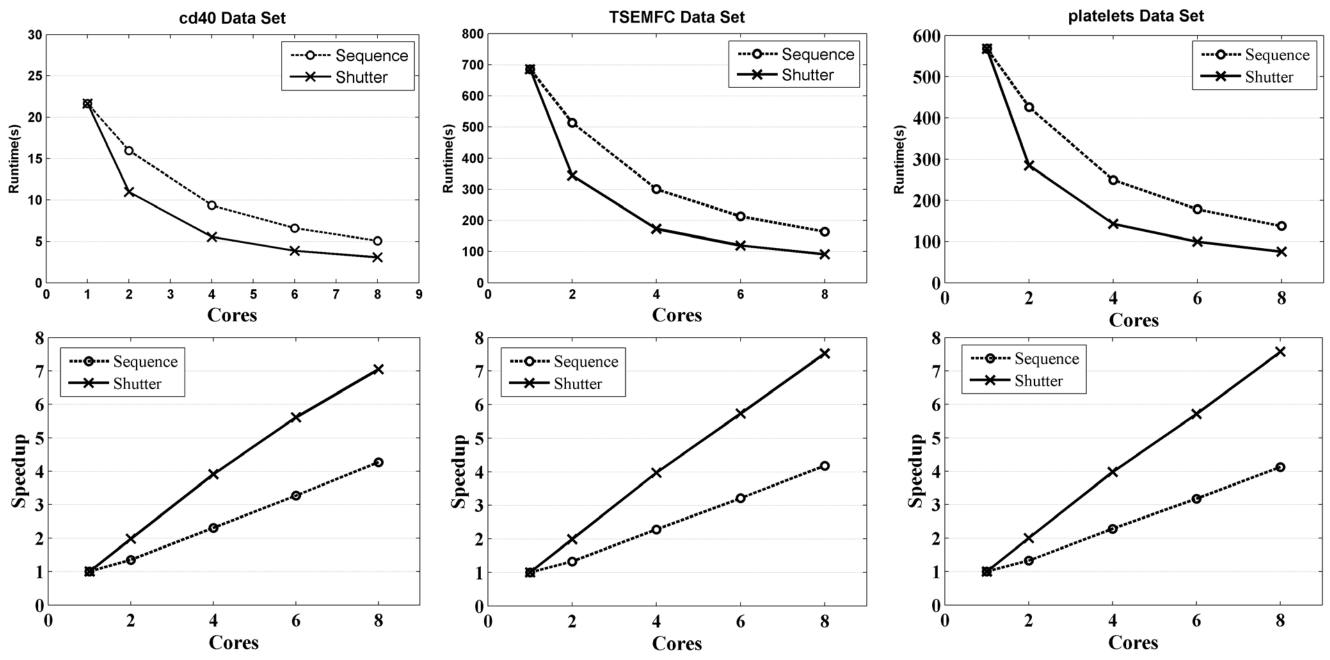

Figure 1. Runtime and speedup of constructing similarity matrix for different data partition ways. The X axis represents the number of cores, and the Y axis represents the runtime and speedup.
doi:10.1371/journal.pone.0091315.g001

widely used [22–26]. Affinity propagation algorithm has many advantages and outperforms some famous algorithms such as the k-means, spectral clustering and super-paramagnetic clustering [27]. Moreover it doesn't require a specific cluster number. Compared with the Tribe-MCL algorithm, the result of affinity propagation is less sensitive to its input parameter. However, its time and space complexity become a great bottleneck when handling large-scale data sets. Given a data set with N data points, the algorithm has to treat three $N \times N$ matrices. Two messages are computed iteratively when algorithm is running, and the time complexity of computing each message is about $O(N^2)$. In order to address these challenges, we implemented our parallel affinity propagation algorithm in the distributed system. Distributed system can supply huge memory size and great computing capacity, so it is promising to design and implement large-scale biological applications on it. To our best knowledge, there are few works focusing on the parallel affinity propagation algorithm [28–32]. These works implemented the parallel affinity propagation algorithm on the memory-shared, GPU and MapReduce parallel architectures. The limitation on memory size and computing capacity of memory-shared parallel architecture make it difficult to handle large-scale data sets; While for the MapReduce parallel architecture, although it can supply the huge storage space, the runtime of parallel algorithm running on this architecture is much longer because of its running principle. The GPU parallel architecture has a great computing capacity, and its memory size also becomes larger in recent years. However, the GPU architecture is not good at doing the logic operations. Considering there are many logic operations in affinity propagation algorithms, we think that the GPU parallel architecture is not an appropriate parallel architecture. Compared with these parallel architectures, the distributed system is the most suitable for developing parallel affinity propagation. Compared with the works in publications [28,29], our method optimizes the global communication, so the parallel algorithm runs faster. Our parallel AP algorithm is compiled in the CentOS System using the gcc-4.1.4 compiler. The version of MPI is 3.0.1. All source code is available on the website: http://hpca.shu.edu.cn/mpiap.

Result and Conclusion

Assessment of the Algorithm Running Performance

In order to measure the efficiency of the parallel clustering algorithm, the runtime and speedup of the algorithm are recorded. Two parts of runtime are measured. The first one is the runtime of constructing the similarity matrix, and the second is the runtime of

Table 1. Comparison of accelerating efficiency for different data partition ways.

	Shutter Partition (%)				Sequence Partition (%)			
Data set	2 cores	4 cores	6 cores	8 cores	2 cores	4 cores	6 cores	8 cores
platelets	100	99.5	95.2	94.6	66.5	57.0	53.0	51.6
TSEMC	99.5	99.3	95.5	94.0	66.5	57.0	53.5	52.3
cd40	99.0	98.8	93.5	88.1	67.5	57.8	54.5	53.4

doi:10.1371/journal.pone.0091315.t001

Table 2. Runtime of the parallel affinity propagation algorithm.

	Runtime(sec)							
Data Sets	**1 core**	**2 cores**	**4 cores**	**8 cores**	**16 cores**	**32 cores**	**64 cores**	**128 cores**
cd40	9134.43	4995.28	2587.66	1334.10	687.59	346.21	178.57	91.05
TSEMFC	N/A	N/A	5113.52	2697.29	1357.79	711.86	404.08	236.88
platelets	N/A	N/A	N/A	N/A	7214.63	3489.62	2167.57	1357.54
Enolase	4531.59	2434.62	1277.32	627.66	335.98	176.79	92.66	48.70
Amidohydrolase	N/A	N/A	N/A	7660.48	3772.05	1894.40	1127.23	725.85

doi:10.1371/journal.pone.0091315.t002

the clustering algorithm. Our parallel clustering algorithm runs on the cluster computers environment. The cluster consists of 16 nodes, and in each node, there are 1 quad core Intel Xeon processor and 8 GB memory. All nodes are connected with each other by the gigabit Ethernet.

Five data sets (3 gene and 2 protein) were used in the experiments. Figure 1 gives the runtime and speedup (runtime of sequential algorithm over that of parallel algorithm) of two different ways of data partition for constructing the similarity matrix. The results shows that the shutter partition achieves much higher speedup than the sequence partition. Because the shutter partition makes the computing load more balanced, all cores can be fully utilized. Furthermore, because the memory-shared architecture is used, there are no communication cost. Table 1 lists the accelerating efficiency (speedup over the number of used cores).

Table 2 and Figure 2(a) show the runtime of the parallel clustering algorithm. In Table 2, N/A represents the runtime of that data set is not available because of the limitation of the memory size. As the number of cores increase, the runtime of parallel affinity propagation decreases greatly. For cd40 and enolase data sets, the sequential algorithm runtime is obtained, so the speedup is calculated and shown in Figure 2(b). Compared with the sequential algorithm, the parallel algorithm achieves a promising speedup, so it can handle the large-scale biological data sets effectively.

Assessment of the Clustering Performance on Gene Data

For gene data, Gene Ontology Overlap Score is used to measure the clustering results. The Gene Ontology (GO) project provides an ontology of defined terms representing gene product properties. The ontology covers three domains: molecular function, biological process and cellular component [33]. In the GO Overlap measurement, the Jaccard and PR indices are introduced to score the clustering results [34].

Jaccard: given two sets, the Jaccard score is defined as the size of the intersection over the size of the union. For clusters C_i and G_j, their Jaccard score is defined as $Jac_{ij} = \frac{|C_i \cap G_j|}{|C_i \cup G_j|}$.

PR: The PR score includes two parts, the Precision score and the Recall score. For clusters C_i and G_j, $PR_{ij} = \frac{|C_i \cap G_j|}{|C_i|} \cdot \frac{|C_i \cap G_j|}{|G_j|}$ denotes their PR score.

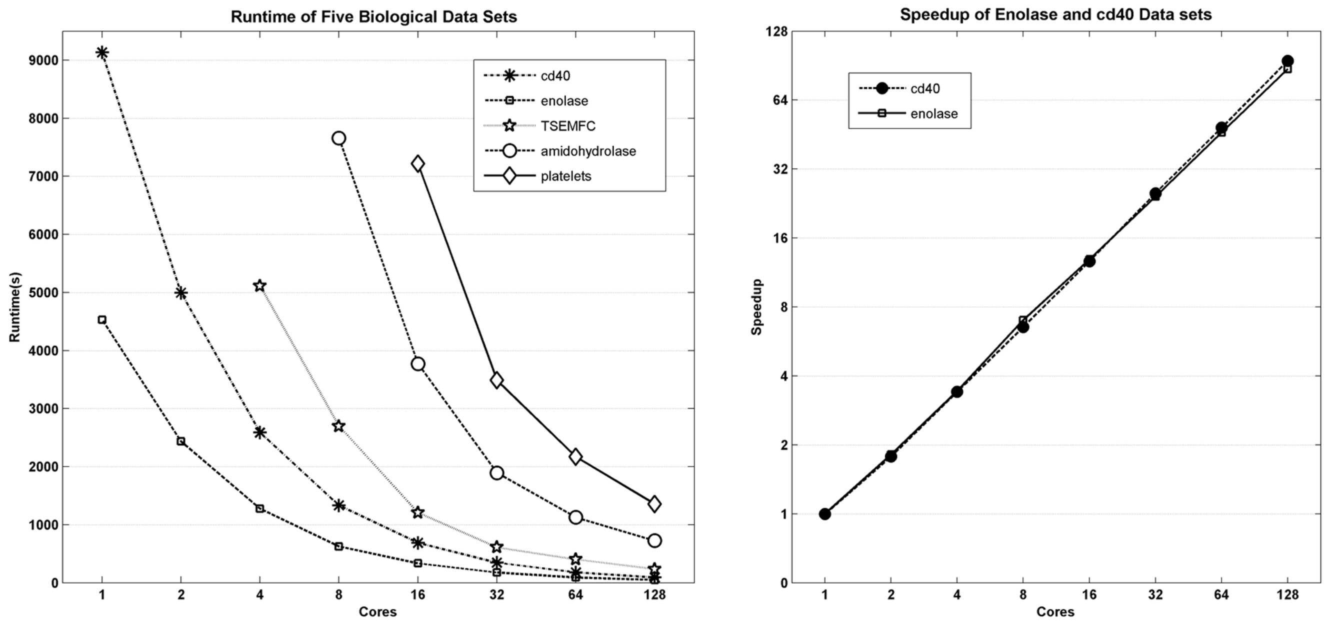

Figure 2. Runtime and speedup of parallel affinity propagation algorithm. (a) shows the runtime of the five biological data sets. (b) shows the speedup of cd40 and enolase data sets.
doi:10.1371/journal.pone.0091315.g002

Table 3. GO overlap analysis of high-throughput gene data sets.

	Molecular Function				**Biological Process**				**Cellular Component**			
Data Sets	Jac_A[a]	Jac_M[b]	PR_A[c]	PR_M[d]	Jac_A	Jac_M	PR_A	PR_M	Jac_A	Jac_M	PR_A	PR_M
TSEMFC	0.133	0.167	0.065	0.082	0.079	0.096	0.028	0.056	0.144	0.205	0.076	0.124
platelets	0.194	0.212	0.112	0.123	0.101	0.106	0.042	0.050	0.181	0.195	0.098	0.109
cd40	0.190	0.237	0.108	0.147	0.183	0.234	0.102	0.186	0.187	0.223	0.103	0.135

[a]Average Jaccard value: the average Jaccard value of all clusters.
[b]Maximum Jaccard value: the Jaccard value of one cluster in which its Jaccard value is maximum in all clusters.
[c]Average PR value: the average PR value of all clusters.
[d]Maximum PR value: the PR value of one cluster in which its Jaccard value is maximum in all clusters.
doi:10.1371/journal.pone.0091315.t003

For each data set, we take an average score over all clusters, weighted by cluster size, to get the Jac_A and PR_A score. Also, we take an maximum score from all clusters, to get the Jac_M and PR_M score. The Jaccard and PR scores range from 0 to 1, and the higher values indicate the better agreement of the uncovered clusters with functional modules corresponding to GO. The scores of three gene data sets are listed in TABLE 3.

Assessment of the Clustering Performance on Protein Data

For protein data, the protein families are detected by the clustering algorithm. The Recall and Precision Measure Score (F-Measure Score) is used to measure the clustering results. F-measure score considers both recall score and precision score. For a cluster C_i and a protein family G_j, the precision p_{ij} is the number of correct results divided by the cluster size $(\frac{|C_i \cap G_j|}{|C_i|})$ and the recall r_{ij} is the number of correct results divided by the protein family size $(\frac{|C_i \cap G_j|}{|G_j|})$. The F-measure score of C_i and G_j is calculated as $F_{ij} = \frac{2p_{ij} \cdot r_{ij}}{p_{ij} + r_{ij}}$.

Two protein subsets with family annotations, containing 14927 and 5431 protein sequences, are regarded as the gold-standard and are clustered to measure the algorithm performance in the experiments. The F-measure score varies from 0 to 1, where 0 is the worst and 1 is the best. For each cluster, the maximum value is taken as the F-measure score over all protein families, and the average value over all clusters, weighted by cluster size, is taken as the F-measure score of a data set. The F-measure scores of two protein superfamilies are listed in Table 4, and Figure 3 shows the detailed recall and precision scores of some larger clusters (larger cluster means there are more than 10 data points in this cluster). The recall score means how many proteins in this family are detected by the AP algorithm in a gold-standard protein family; the precise score means how many proteins in this family are divided right. As shown in Figure 3, the x-axis represents the cluster number. For instance, 1 in x-axis means that the 1^{st} larger cluster; 2 in x-axis means that the 2^{nd} larger cluster. The AP detects about 35 larger protein families and about 12 larger protein families for the Amidohydrolase and Enolase protein superfamily, respectively. The results show that the algorithm can detect the protein families in the large protein superfamilies effectively. Most clusters achieve a high precision score. Both precision and recall scores indicate that most proteins in the same families are detected and grouped correctly.

Table 4. Clustering performance on protein superfamily data sets.

Data Sets	**Amidohydrolase**	**Enolase**
Superfamily Size	14927	5431
Family Number	80	19
Cluster Number	85	20
F-Measure Score	0.57	0.54

doi:10.1371/journal.pone.0091315.t004

Error Rates Analysis of Parallel and Non-Parallel Algorithms

In order to validate that the parallel affinity propagation algorithm performs well and its results are similar to the non-parallel affinity propagation algorithm, we make the error rates analysis of both algorithms. We get the non-parallel program and a relatively small data set ("Facesimilarity" data set, 900 data points) from the original publication of AP algorithm [21], and then run this non-parallel program and our parallel program on this data set. According to its description, it is appropriate to set the "preference value" (input of AP algorithm) close to median of the similarity values. So we set a range of the "preference value" which is close to the median of the similarity values, and use these "preference values" to test both algorithms. For all input values, both algorithms detect the same number of clusters, and converge at the same iteration steps. Furthermore, because the cluster structure is extracted from the latest message matrices, it is important to validate whether the two message matrices of both algorithms are the same. We calculate the 2-norm of the difference value of two message matrices got from both parallel and non-parallel algorithms. Detailed results are listed in Table 5.

Conclusion

The parallel affinity propagation clustering algorithm can address the large-scale biological problems such as clustering the microarrays gene data sets and detecting the protein families effectively. The Euclidean distance and BLAST E-value are used in the experiments to describe the similarities of gene data pairs and protein data pairs, respectively. In the similarity matrix construction, the memory-shared architecture minimizes the communication time between data pairs, and the shutter partition of data sets balances the computing loads on cores. Compared with the sequential algorithm, the parallel affinity propagation algorithm reduces the runtime significantly and achieves promising speedups. Also, it handles large-scale biological data sets

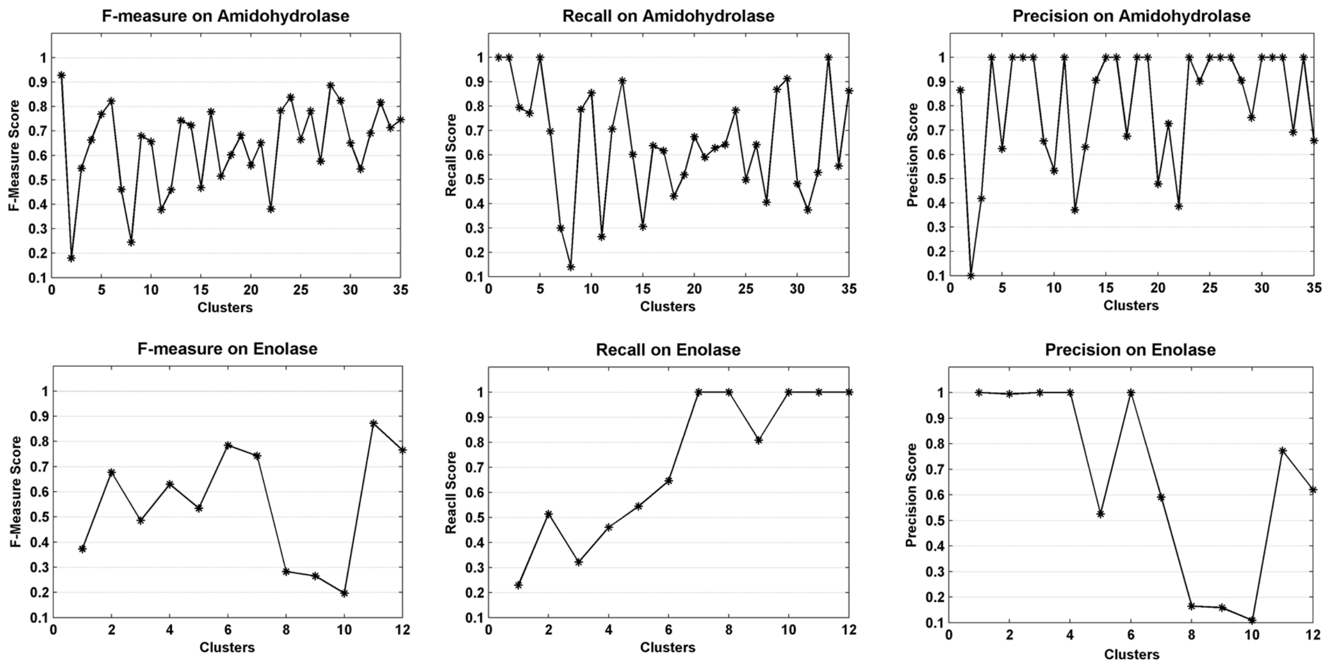

Figure 3. The F-Measure score of some large clusters on two protein data sets. The F-measure score, recall score and precision score are calculated. The X axis and Y axis represent the cluster and its corresponding best score value, respectively. Some tiny and singleton clusters are not considered in the figure.
doi:10.1371/journal.pone.0091315.g003

effectively. From the GO overlap score and F-measure score, the affinity propagation is validated so that it is very feasible in clustering the gene data sets and detecting the protein families. Besides the analysis of microarray gene data and protein families, it is possible and promising to apply the affinity propagation algorithm into some other researches like microbial communities.

Discussion

Appropriate Parallel Architecture

There are two steps for affinity propagation algorithm to cluster a data set. The first one is constructing the similarity matrix, and the second is exchanging messages (responsibility message and availability message). However, considering the strong correlations between each data points in the procedure of constructing similarity matrix, we implemented parallel similarity matrix construction algorithm in the memory-shared architecture. The parallelism in this kind of parallel architecture is on the thread level, and threads share the memory with each other, so there are nearly no communication cost among threads. For the affinity propagation algorithm, it always requires large memory space to store intermediate information when clustering large-scale data sets. Since the distributed system consisting of many computers, large memory size and great computing capacity are available, it become the best choice for the parallel affinity propagation algorithm.

Minimizing Global Communication

The increase of communication cost leads to a longer algorithm runtime and a lower speedup. In all kinds of communication, the global communication is the most time consuming. The affinity propagation algorithm is parallelized on the process-level, and each core can run one or several processes concurrently. According to two message computing rules, nearly N times global communication are require in each iteration to gather information from all processes. In order to reduce the global communication cost in each iteration, the similarity matrix is partitioned by the row and some variables are defined to store the intermediate results when computing the availability message. In this way, the responsibility message is computed independently without communication among processes; for the availability message, it requires only one time global communication in each iteration. In our methods, the global communication is minimized to 1 time in one iteration, so the communication cost is reduced significantly.

Table 5. Error rates analysis of parallel and non-parallel algorithms.

	Convergence Steps[a]		Cluster Num[b]		Error Rates[c]	
Preference[d]	Non-P[e]	P[f]	Non-P	P	R msg[g]	A msg[h]
−40	123	123	94	94	1.2703e-04	1.7147e-04
−50	118	118	72	72	1.2967e-04	2.0939e-04
−60	129	129	62	62	1.2364e-04	1.9748e-04
−70	112	112	50	50	1.4744e-04	2.6443e-04
−80	117	117	46	46	1.1632e-04	2.4990e-04

[a]Convergence steps: the iteration steps when the algorithm is converged.
[b]Cluster Number: the number of clusters which the AP algorithm detects.
[c]Errot rates: the 2-norm of the difference value of message matrices from two algorithms.
[d]Preference value: the input value of AP algorithm.
[e]Non-Parallel algorithm: the non-parallel version of algorithm which get from the original publication of AP algorithm.
[f]Parallel algorithm: the parallel version of algorithm.
[g]Responsibility message: the responsibility message of AP algorithm.
[h]Availability message: the availability message of AP algorithm.
doi:10.1371/journal.pone.0091315.t005

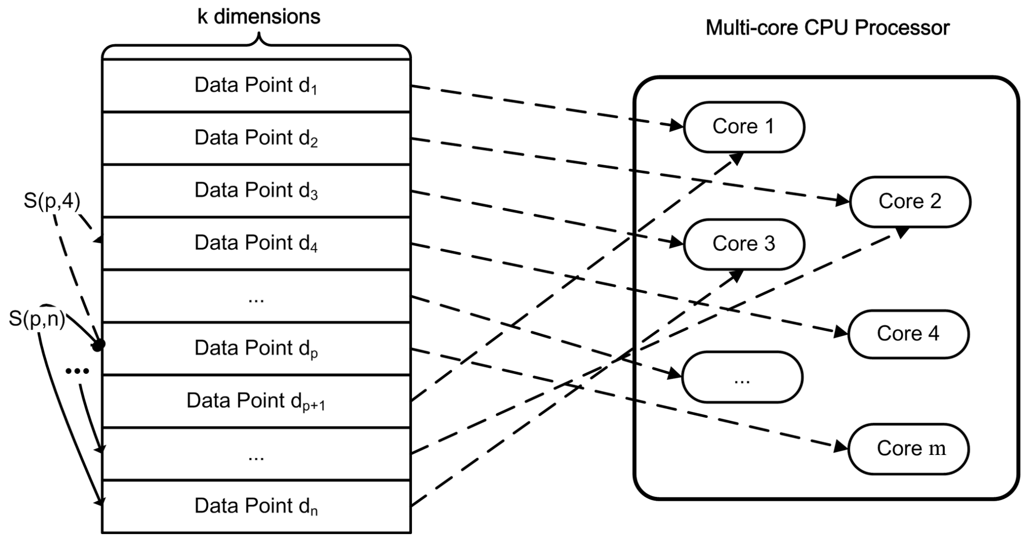

Figure 4. Partition of the input biological matrix. Rows of the input matrix are assigned to different cores to calculate the similarities with others (each row represents a data point; n and k represent the row number of the input matrix and the dimension of each row, respectively; $S(p,n)$ represents the similarity between data point p and n; m represents the number of available cores).
doi:10.1371/journal.pone.0091315.g004

Materials and Methods

Parallel Constructing Similarity Matrix

Since affinity propagation algorithm runs based on the similarities between data pairs, the similarity matrix is required to be constructed firstly. There exist many methods to construct the similarities matrix, including the Euclidean distance, Pearson correlation coefficient, E-value(output of BLAST software) and so on. The time complexity of constructing the similarity matrix is about $O(N^2k)$, where N and k represent the data point number and the dimension of each data points, respectively. The biological data sets are partitioned and several cores are used to accelerate the construction procedure of similarity matrix. Figure 4 presents the partition of data sets.

The shutter partition way rather than the sequence way, is used to to alleviate the unbalanced computing load on cores. The sequential partition means that rows of the input matrix are assigned to cores one by one; while the shutter partition means that rows of the input matrix assigned to cores one block by one block, and the length of block equals the number of available cores. For instance, given an input matrix, with N rows, the number of available cores is p. All cores, except for the last one, are assigned about $\lceil N/p \rceil$ rows ($\lceil x \rceil$ means round up×to an integer). For the sequential partition, the 1^{st}, 2^{nd}, 3^{rd},…, k^{th} rows are assigned to the 1^{st} core (k equals $\lceil N/p \rceil$); $(k+1)^{th}$, $(k+2)^{th}$,…, $2k^{th}$ rows are assigned to the 2^{nd} core, and so on. For the shutter partition, the length of block equals p (number of

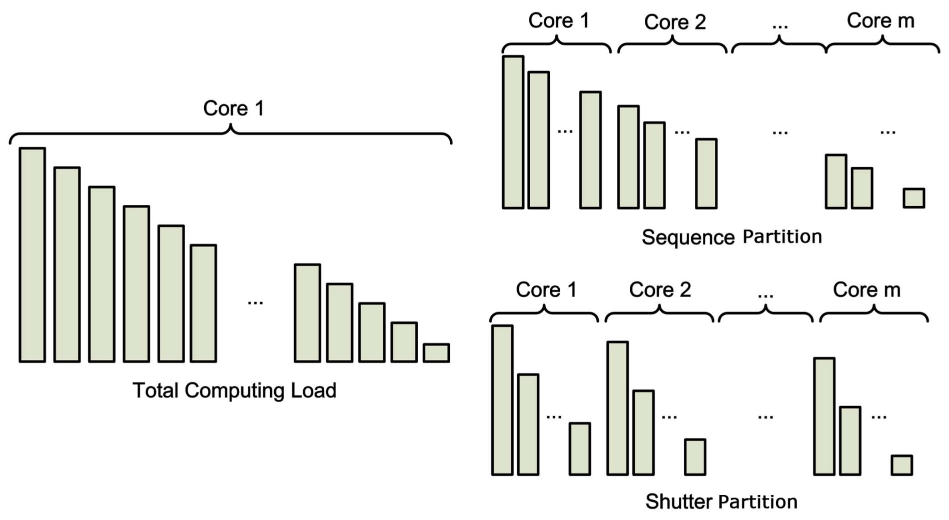

Figure 5. The computing load on cores for different data partition ways. (a) The computing load of whole data set on one core. The computing loads on the cores which are assigned the upper rows of input matrix are much more than that on the cores which are assigned the lower rows. (b) The computing load on cores when the input matrix is partitioned by the sequence partition. (c) The computing load on cores when the input matrix is partitioned by the shutter partition.
doi:10.1371/journal.pone.0091315.g005

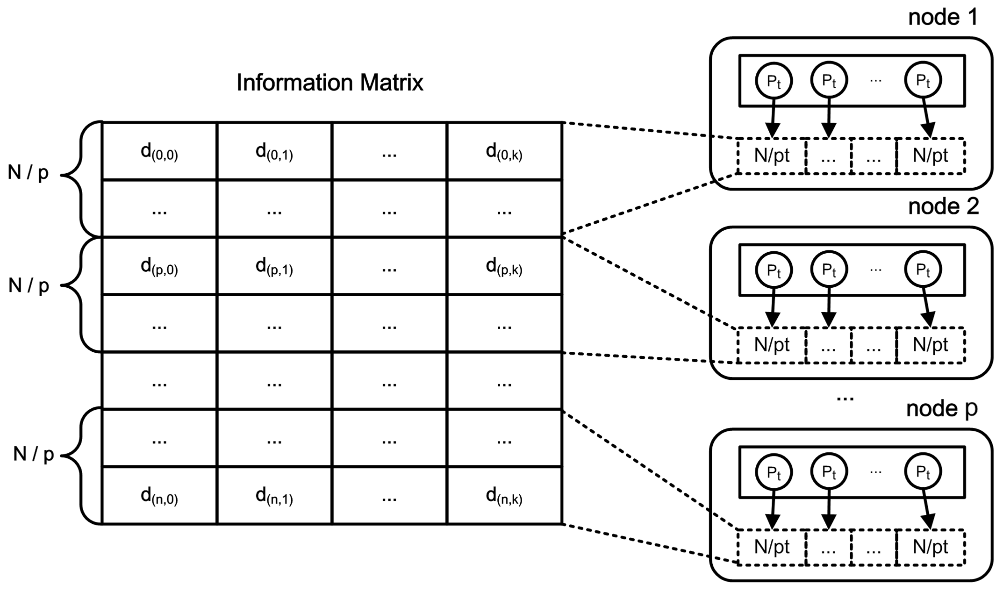

Figure 6. Partition of three information matrices. All three matrices are partitioned by the row. For a matrix with N rows and a computing cluster with p machine nodes, each node is assigned about N/p rows of each information matrix. In each node, the N/p rows are processed by t cores concurrently.
doi:10.1371/journal.pone.0091315.g006

available cores), and the 1st, $(p+1)^{th}$, $(2p+1)^{th}$,…, $(kp+1)^{th}$ rows are assigned to the 1st core; the 2nd, $(p+2)^{th}$, $(2p+2)^{th}$,…, $(kp+2)^{th}$ rows are assigned to the 2nd core, and so on.

In the experiments, the Euclidean distance is taken to describe the similarities between data points. As shown in Figure 4, data point d_p is required to compute the similarities with data points d_{p+1}, d_{p+2},…, d_n and doesn't require to compute with d_4. Because the value of $S(p,4)$ equals $S(4,p)$, and the value has been computed previously. The symmetry of calculating similarity leads to the unbalanced computing load of the rows of input matrix. The rows at the bottom of input matrix require less calculation than those at the top of input matrix. So for the sequential partition, the computing loads on the cores, which are assigned the upper rows of input matrix, are much more than that on the cores, which are assigned the lower rows. It causes unbalanced

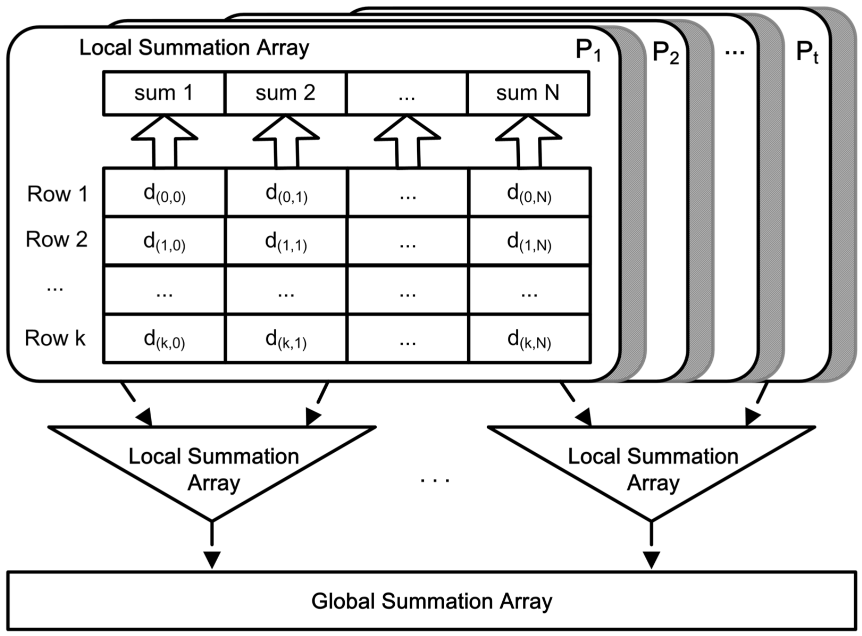

Figure 7. The procedure of computing availability messages. There are t processes P_1,P_2,…,P_t, and each process is assigned about k rows of the availability messages matrix after partitioning. In each row, there are N column. In order to reduce the communication cost, each process computes the local summation of the N columns and stores the intermediate values in the Local Summation Array firstly, and then these local values in all processes are gathered and scattered to compute the global summation of the N columns.
doi:10.1371/journal.pone.0091315.g007

Table 6. Algorithm 1.

```
1: function AVAIMSGCPT(Res, Avai, lam)
2:   for $i<colSize$ do
3:     for all $v \in Res[i]$ do
4:       $localsum[i] \leftarrow localsum[i]+max(0, v)$
5:     end for
6:   end for
7:   $globalsum \leftarrow MPI\_Allreduce(localsum)$
8:   MPI_Barrier ()
9:   for $i<colSize$ do
10:    for $k<rowSize$ do
11:      $v \leftarrow max(0, Res[k][i])$
12:      $A[k][i] \leftarrow min(0, globalsum[i]-v)$
13:    end for
14:  end for
15:  $Avai \leftarrow A\times(1-lam)+Avai\times lam$
16: end function
```

Availability message computation.
doi:10.1371/journal.pone.0091315.t006

computing loads on cores. However, for the shutter partition, cores are assigned rows of the whole input matrix, so the computing load is much more balanced. It makes the cores fully utilized. Figure 5 depicts computing load on cores by two partition ways.

Parallel Affinity Propagation Algorithm

The affinity propagation algorithm clusters data set based on the similarity matrix, and exchanges two messages between data points. The two messages are responsibility and availability messages. The responsibility $r(i,k)$ and availability messages a(i,k) are computed by the following rules.

$$r(i,k) \leftarrow s(i,k) - \max_{k' s.t. k' \neq k} \{a(i,k') + s(i,k')\} \tag{1}$$

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' s.t. i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \tag{2}$$

When the algorithm runs, three information matrices (one for similarity and two for messages) are required to be stored in the memory. In the information matrices, each row or column represents one data point. For large-scale data sets, it is impossible for the stand-alone computer to run the sequential affinity propagation algorithm because of the nearly endless runtime and the limitation of memory space. Given a data set with more than 50000 data points, it requires nearly 56 GB (double precision) memory space to store three information matrices and the algorithm runtime is more than 30 hours. The parallel affinity propagation in the distributed system can alleviate this kind of limitation significantly.

In the distributed system, affinity propagation algorithm runs on the process-level parallelism. Each process is assigned a few rows of the three information matrices, and each core execute one or several processes concurrently. From the equation 1 and equation 2, we know that the responsibility message is calculated by the row, and the availability message is calculated by the column. Compared with the responsibility message, the availability message has less correlations among data points. As shown in Figure 6, in order to reduce the communication cost in parallel algorithm, we partition the information matrices by the row.

The key issue in the parallel affinity propagation algorithm is to parallel exchange messages among data points. According to the message computing rules, the responsibility message is computed by the rows and the availability message is computed by the columns. Since the information matrices are partitioned by the rows, the data chunks of each row are all stored in the local running process when computing the responsibility message. So the responsibility message can be computed independently without communication cost among processes. For the availability message, the summation values of all columns in the information matrices are computed. However, since data chunks of each column are distributed on the different running processes, data chunks required to be gathered from all processes for N times to compute the summation values of N columns, causes a massive communication cost.

In order to minimize the global communication times, the local summation of N columns is computed firstly in each process, and intermediate variables are defined to store these intermediate results. In this way, the root process requires only one time global reduction and scatter to finish computing availability message. The communication cost is optimized from N times global communication to 1 time. Figure 7 depicts the procedure of parallel computing availability message, as well as the pseudocode of parallel computing availability message listed in the Table 6.

Data Sets

Two kinds of data sets are used here, including the gene data sets and protein data sets. DNA microarrays are extremely powerful tools for such studies in which they allow one to probe virtually the entire transcriptome to give an overall picture of gene expression behavior [35]. Microarray data sets are commonly very large, and are influenced by a number of variables. Many clustering algorithms currently have been introduced to explore genes expression data [36–38]. Three microarray data sets are used in the experiments, including TSEMFC (Tobacco smoke effect on maternal and fetal cells, GEO Accession Number: GSE27272) [39], platelets (platelets sickle cell disease, GEO Accession Number: GSE11524) [40] and cd40 (CD40-Ligand screen in B cells, GEO Accession Number: GSE376) [41] which are downloaded from the Gene Expression Omnibus online database (GEO) [42]. TSEMFC data set comes from the analysis data of peripheral blood leukocytes and placenta of pregnant smokers, with 24526 data points and 183 samples. The platelets data set comes from the analysis data of platelets from patients with sickle cell disease, with 54675 data points and 30 samples. The cd40 data set is the data of effects induced by 65 nM CD40 in male c57BL/6 B cells and consists of 16273 data points and 12 samples.

For protein data sets, protein families are detected in the superfamily by the affinity propagation algorithm. Detecting protein families in large-scale superfamily is a difficult but important biological task. Proteins with the same functions or biological processes should be in the same protein families [43]. Moreover, the protein families are always defined as the groups of molecules which share significant sequence similarities [44], so it is promising to detect the protein families through the protein sequence similarities. Similarity matrices of protein data sets are constructed by carrying out the all-against-all BLAST in local databases, and the similarities between protein pairs are set to the $-log$ of the BLAST E-value. Two large protein superfamilies are downloaded from Structure-Function Linkage database [45]. They are Amidohydrolase [46] and Enolase [47] protein superfamilies, with about 36690 and 12509 protein sequences, respectively.

Acknowledgments

The authors would like to thank for the discussion with Professor Qing Nie of the Department of Mathematics, University of California Irvine, and the Professor Sanzheng Qiao of the Department of Computing and Software, McMaster University. Their precious advices give us lots of help.

Author Contributions

Conceived and designed the experiments: MW WD DD HZ HX LC WZ JX YG. Performed the experiments: MW WD. Analyzed the data: HX LC WZ JX YG. Contributed reagents/materials/analysis tools: MW WD DD HZ HX LC WZ JX. Wrote the paper: MW DD WZ.

References

1. Enright AJ, Van Dongen S, Ouzounis CA (2002) An efficient algorithm for large-scale detection of protein families. Nucleic Acids Research 30: 1575–1584.
2. Paccanaro A, Casbon JA, Saqi MAS (2006) Spectral clustering of protein sequences. Nucleic Acids Research 34: 1571–1580.
3. Guimera R, Nunes Amaral LA (2005) Functional cartography of complex metabolic networks. Nature 433: 895–900.
4. Hanisch D, Zien A, Zimmer R, Lengauer T (2002) Co-clustering of biological networks and gene expression data. Bioinformatics 18: S145–S154.
5. Mazurie A, Bonchev D, Schwikowski B, Buck G (2010) Evolution of metabolic network organization. BMC Systems Biology 4: 59.
6. Bader G, Hogue C (2003) An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics 4: 2.
7. Brohee S, van Helden J (2006) Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinformatics 7: 488.
8. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Le Cam LM, Neyman J, editors, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1. University of California Press, Berkeley, CA, USA, pp. 281–297.
9. Mndez MA, Hodar C, Vulpe C, Gonzlez M, Cambiazo V (2002) Discriminant analysis to evaluate clustering of gene expression data. FEBS Letters 522: 24–28.
10. Pirim H, Eksioglu B, Perkins AD, Yceer E (2012) Clustering of high throughput gene expression data. Computers & Operations Research 39: 3046–3061.
11. Mishra VD, Mishra S, Mishra D, Satapathy SK (2012) A naive soft computing based approach for gene expression data analysis. Procedia Engineering 38: 2124–2128.
12. Lounkine E, Nigsch F, Jenkins JL, Glick M (2011) Activity-aware clustering of high throughput screening data and elucidation of orthogonal structureactivity relationships. Journal of Chemical Information and Modeling 51: 3158–3168.
13. Ding W, Xie J, Dai D, Zhang H, Xie H, et al. (2013) Cnncon: Improved protein contact maps prediction using cascaded neural networks. PLoS ONE 8: e61533.
14. Kerr G, Ruskin HJ, Crane M, Doolan P (2008) Techniques for clustering gene expression data. Comput Biol Med 38: 283–293.
15. Asyali MH, Colak D, Demirkaya O, Inan MS (2006) Gene expression profile classification: A review. Current Bioinformatics 1: 55–73.
16. Pham TD, Wells C, Crane DI (2006) Analysis of microarray gene expression data. Current Bioin- formatics 1: 37–53.
17. Shi J, Malik J (2000) Normalized cuts and image segmentation. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22: 888–905.
18. Chen WY (2011) Parallel spectral clustering in distributed systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 33: 568–586.
19. Ekanayake J, Gunarathne T, Qiu J (2011) Cloud technologies for bioinformatics applications. Parallel and Distributed Systems, IEEE Transactions on 22: 998–1011.
20. Bustamam A, Burrage K, Hamilton NA (2012) Fast parallel markov clustering in bioinformatics using massively parallel computing on gpu with cuda and ellpack-r sparse format. IEEE/ACM Trans Comput Biol Bioinformatics 9: 679–692.
21. Frey BJ, Dueck D (2007) Clustering by passing messages between data points. Science 315: 972–976.
22. Leone M, Sumedha, Weigt M (2007) Clustering by soft-constraint affinity propagation: applications to gene-expression data. Bioinformatics 23: 2708–2715.
23. Dueck D, Frey B, Jojic N, Jojic V, Giaever G, et al. (2008) Constructing Treatment Portfolios Using Affinity Propagation, Springer Berlin Heidelberg, volume 4955 of Lecture Notes in Computer Science, chapter 31. pp. 360–371.
24. Kiddle SJ, Windram OPF, McHattie S, Mead A, Beynon J, et al. (2010) Temporal clustering by affinity propagation reveals transcriptional modules in arabidopsis thaliana. Bioinformatics 26: 355–362.
25. Borile C, Labarre M, Franz S, Sola C, Refregier G (2011) Using affinity propagation for identifying subspecies among clonal organisms: lessons from m. tuberculosis. BMC Bioinformatics 12: 224.
26. Soria D, Garibaldi JM, Ambrogi F, Boracchi P, Raimondi E, et al. (2009) Cancer profiles by affinity propagation. International Journal of Knowledge Engineering and Soft Data Paradigms 1: 195–215.
27. Blatt M, Wiseman S, Domany E (1996) Superparamagnetic clustering of data. Physical Review Letters 76: 3251–3254.
28. Blasberg RV, Gobbert MK (2008) Parallel performance studies for a clustering algorithm. Technical Report HPCF-2008-5, UMBC High Performance Computing Facility, University of Maryland, Baltimore County.
29. Blasberg RV, Gobbert MK (2008) Parallel performance studies for a clustering algorithm. Technical Report HPCF-2008-8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County.
30. Kurdziel M, Boryczko K (2012) Dense affinity propagation on clusters of gpus. In: Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, volume 7203 of Lecture Notes in Computer Science. pp. 599–608.
31. Blasberg RV, Gobbert MK (2008). Mvapich2 vs. openmpi for a clustering algorithm.
32. Fong J, Kwan R, Wang FL (2008) Large scale of e-learning resources clustering with parallel affinity propagation. In: Proceedings of the 1st International Conference on Hybrid Learning and Education, ICHL 2008. Springer 2008 Lecture Notes in Computer Science.
33. Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, et al. (2000) Gene ontology: tool for the unification of biology. Nat Genet 25: 25–29.
34. Song J, Singh M (2009) How and when should interactome-derived clusters be used to predict functional modules and protein function? Bioinformatics 25: 3143–3150.
35. Spurgeon SL, Jones RC, Ramakrishnan R (2008) High throughput gene expression measurement with real time pcr in a microuidic dynamic array. Plos One 3: e1662.
36. Sturn A, Quackenbush J, Trajanoski Z (2002) Genesis: cluster analysis of microarray data. Bioinformatics 18: 207–208.
37. Tung-Shou C, Tzu-Hsin T, Yi-Tzu C, Chin-Chiang L, Rong-Chang C, et al. (2005) A combined k-means and hierarchical clustering method for improving the clustering efficiency of microarray. In: Intelligent Signal Processing and Communication Systems, 2005. ISPACS 2005. Proceedings of 2005 International Symposium on. pp. 405–408.
38. Dembl D, Kastner P (2003) Fuzzy c-means method for clustering microarray data. Bioinformatics 19: 973–980.
39. Votavova H, Dostalova Merkerova M, Fejglova K, Vasikova A, Krejcik Z, et al. (2011) Transcriptome alterations in maternal and fetal cells induced by tobacco smoke. Placenta 32: 763–770.
40. Raghavachari N, Xu X, Harris A, Villagra J, Logun C, et al. (2007) Amplified expression profiling of platelet transcriptome reveals changes in arginine metabolic pathways in patients with sickle cell disease. Circulation 115: 1551–1562.
41. Zhu X, Hart R, Chang MS, Kim JW, Lee SY, et al. (2004) Analysis of the major patterns of b cell gene expression changes in response to short-term stimulation with 33 single ligands. The Journal of Immunology 173: 7141–7149.
42. Edgar R, Domrachev M, Lash AE (2002) Gene expression omnibus: Ncbi gene expression and hybridization array data repository. Nucleic Acids Research 30: 207–210.
43. Hegyi H, Gerstein M (1999) The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. Journal of Molecular Biology 288: 147–164.
44. Dayhoff MO (1976) The origin and evolution of protein superfamilies. Fed Proc 35: 2132–8.
45. Pegg SCH, Brown SD, Ojha S, Seffernick J, Meng EC, et al. (2006) Leveraging enzyme structure-function relationships for functional inference and experimental design: The structure-function linkage database. Biochemistry 45: 2545–2555.
46. Seibert CM, Raushel FM (2005) Structural and catalytic diversity within the amidohydrolase superfamily. Biochemistry 44: 6383–6391.
47. Babbitt PC, Hasson MS, Wedekind JE, Palmer DRJ, Barrett WC, et al. (1996) The enolase superfamily: A general strategy for enzyme-catalyzed abstraction of the -protons of carboxylic acids. Biochemistry 35: 16489–16501.