A Load Balancing Knapsack Algorithm for Parallel Fuzzy c-Means Cluster Analysis

Marta V. Modenesi; Alexandre G. Evsukoff; Myrian C. A. Costa.

COPPE/Federal University of Rio de Janeiro,
P.O.Box 68506, 21945-970 Rio de Janeiro RJ, Brazil
Tel: (+55) 21 25627388, Fax: (+55) 21 25627392
modenesi@hotmail.com, evsukoff@coc.ufrj.br,
myrian@nacad.ufrj.br,

Abstract. This work proposes a load balance algorithm to parallel processing based on a variation of the classical knapsack problem. The problem considers the distribution of a set of partitions, defined by the number of clusters, over a set of processors attempting to achieve a minimal overall processing cost.

The work is an optimization for the parallel fuzzy c-means (FCM) clustering analysis algorithm proposed in a previous work composed by two distinct parts: the cluster analysis, properly said, using the FCM algorithm to calculate of clusters centers and the PBM index to evaluate partitions, and the load balance, which is modeled by the multiple knapsack problem and implemented through a heuristic that incorporates the restrictions related to cluster analysis in order to gives more efficiency to the parallel process.

Topics of Interest: Unsupervised Classification, Fuzzy c-Means, Load Balance, Optimization.

1. Introduction

Cluster analysis is the unsupervised classification of data into groups (clusters) and it is one of the most intensive computational tasks in data mining. It is thus very attractive for parallel processing and many parallel and distributed clustering algorithms have been recently studied [1][2][3].

There are several approaches that have been studied for cluster analysis algorithms [4]. In the partition approach, two main optimization problems are addressed: to find the number of clusters presents in the data and the location of clusters centers. The later problem is much easier to solve, and iterative greedy optimization algorithms, such as the k-means algorithm and its variants, are widely used for that purpose, being well known by the data mining community.

The k-means algorithm has been extended to the fuzzy c-means algorithm by Bezdek in the early eighties [5]. The fuzzy c-means (FCM) algorithm computes a

"fuzzy" partition where data records may be related to more than one cluster but with different membership values. The FCM solution is very useful in real applications because it provides soft boundaries for clusters taking classification uncertainty into account.

The problem of determining the number of clusters in a dataset is much more difficult to solve, both for crisp and fuzzy clustering algorithms. In general, a validation index that reflects the quality of the result is used as an optimization index and the clustering algorithm is executed for a range of clusters. Several cluster validation indexes have been proposed [6][7][8] and most of them are based on geometrical approaches with the aim of finding dense and separated clusters.

In a previous work, Modenesi et al. [3] have presented a parallel clustering algorithm that computes the location of clusters centers and the validation index simultaneously. In their approach, the FCM algorithm is iterated within the cluster validation loop, in which the clustering quality is computed by the PBM index, recently proposed by Pakhira et al. [8]. In the parallel implementation, the dataset is equally divided among the available processors, which compute the iterative steps, and the cluster and validation results are integrated by the master processor [3]. This approach causes a natural load balance in the parallel processing, but it has a high communication cost due to the need of frequent interaction among processors. The results show that parallelization is not efficient for a low number of clusters but the greater the dataset the better is the speed-up.

In cluster analysis, the question of minimizing communication costs and maximizing the parallel processing efficiency can be understood as a knapsack problem where computational capacity must be fulfilled minimizing the cluster analysis computation cost.

The knapsack problem, proposed initially by Dantzig [9], consists in the problem of selecting, from a collection of items, with distinct benefits and costs, the ones that fit in a knapsack resulting in the maximum possible value and minimal cost. It is a well known problem in the area of combinatorial analysis with extensive applicability in many practical cases, such as: production and logistics planning, financial engineering, vehicles shipment, budget, among others. The knapsack problem is a NP-complete problem and, therefore, there is no global optimum solution known to be computed in polynomial time. Many methods, called heuristics, are studied for solving the knapsack problem. They only give approximate solutions to the problem [10][11][12].

In this work, the parallel FCM cluster analysis proposed in a previous work [3] is extended with a load balancing computed by the solution of the knapsack problem. A heuristic algorithm is proposed, based on the bin packing problem, which is a variation of the multiple knapsacks problem.

The paper is organized as follows: section two reviews the parallel FCM cluster analysis algorithm; section three introduces the knapsack problem; section four presents the FCM cluster analysis with the load balance algorithm; section five describes tests with results and section six presents conclusions and suggests future works.

2. The Parallel FCM Algorithm

The cluster analysis FCM [3] aims to find the patterns present in data by processing a range of clusters by the calculation of distances of registers to clusters centers through the FCM algorithm and the selection of the best partition through the cluster validity index PBM.

The parallel FCM cluster analysis procedure is described by the following sequence:

Step 1. (Master processor): Splits the data set equally among the available processors so that each one receives N/p records, where N is the number of records and p is the number of processors.

Step 2. (All processors): Compute the geometrical center of its local data and communicate this center to all processors, so that every processor can compute the geometrical center of the entire database. Compute the global data density (factor E_1 of the PBM index) on local data and send it to master processor.

Step 3. (Master processor): Sets initial centers and broadcasts them, so that all processors have the same clusters' centers values at the beginning of the FCM looping.

Step 4. (All processors): Until convergence is achieved compute the distances from each record in the local dataset to all clusters' centers; update the partition matrix, calculate new clusters' centers.

Step 5. (All processors): Compute the cluster density (factor E_K of the PBM index) on its local data and send it to master processor.

Step 6. (Master Processor): Integrates the calculations for the PBM index and stores it. If the range of number of clusters is covered, stops, otherwise returns to *Step3*.

The procedure described above is computed for each number of clusters in the cluster analysis, so that the procedure is repeated as many times as the desired range of numbers of clusters, so that the PBM index, as a function of the number of centers, is computed. The best partition is the one corresponding to the largest value of the PBM index.

The computational cost of the algorithm is exponentially proportional to the number of patterns being analyzed. The bigger is the number of clusters to calculate, more cycles of processing of distances of registers to clusters centers are necessary and greater is the computational effort. Moreover, each partition has a different computational cost which is related to its number of clusters, meaning that processing partitions with bigger number of patterns (clusters) involve more computational effort than processing those with smaller ones.

An approach to minimize the communication cost in the prior algorithm is to distribute partitions over processors, making processors in charge of calculating a partitions´ group with no need of communication during the FCM loop, having at the end of the processing a master processor consolidating the results and indicating which is the best partition.

To make this approach effective, a load balance policy must be implemented, ensuring that processors receive a balanced charge of partitions to process in order to minimize the total time of the parallel processing. The load balance policy must distribute partitions over processors considering the computational cost of each partition. At this point the knapsack problem comes as a model of how to arrange partitions minimizing the processing cost.

3. The Knapsack Problem

All knapsack problems variations refer to a set of n items where each item has an associated profit p_j and weight $w_j, 1 \le j \le n$, which are generally positive integers. The problem consists in selecting which items must be included into the knapsack so that the total value is maximized without exceeding the knapsack capacity W. The selection value of an item is represented by the binary project variable $x_j \in \{0,1\}$.

The knapsack problem in its basic form can be formulated as:

$$\begin{aligned} &Maximize \sum_{j=1}^{n} p_j x_j \\ &s.t. \sum_{j=1}^{n} w_j x_j \le W \end{aligned} \tag{1}$$

Many kinds of knapsack problems arise from real situations where different constrains determine special cases. Among the knapsack problem direct generalizations, were applicable in this work the subset sum problem and the multiple knapsacks problem.

The subset sum problem is characterized by the situation where items costs (weights) are equal to profits (values), i.e. $w_j = p_j$ [11][12]. It occurs when a quantitative target should be reached, so that its negative deviation (or loss) must be minimized and a positive deviation is not allowed.

The knapsack problem becomes useful for load balancing in parallel processing when a set of m knapsacks are considered. This problem is known as the subset-sum partition problem when the capacities of all knapsacks are equal, so that each knapsack capacity is $W_i = W / m$, where W is the overall capacity. The solution to the problem is represented by the binary variable $x_{ij} \in \{0,1\}$ that assigns an item j to the knapsack i.

The multiple knapsacks' problem is formulated as:

$$Maximize \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij} \qquad (2)$$

$$s.t. \sum_{j=1}^{n} w_j x_{ij} \le W_i, \; 1 \le i \le m$$

A special case of the multiple knapsacks problem is the bin packing problem, where items must be placed in the smallest number of knapsacks.

The formulation of the multiple knapsacks problem for load balancing in the FCM parallel cluster analysis is discussed next.

4. The Proposed Algorithm

4.1. Problem formulation

The load balancing problem of the parallel FCM cluster analysis algorithm can be understood as a knapsack problem whose items are partitions to be processed by the algorithm, each partition defined by the number of clusters in the fuzzy partitions range $2 \le j \le n$. The load balancing problem is a multiple knapsacks problem where each processor represents a knapsack and the overall processing capacity is defined by the number of the available processors p.

The main issue in the load balancing problem is to minimize the overall parallel processing time. The problem thus becomes a minimization problem, stated as follows:

$$Minimize \sum_{i=1}^{p} \sum_{j=2}^{n} w_{ij} x_{ij} \qquad (3)$$

$$s.t. \sum_{i=1}^{p} x_{ij} = 1, \; 2 \le j \le n$$

$$\sum_{i=1}^{p} \sum_{j=2}^{n} x_{ij} = n$$

The project variable $x_{ij} \in \{0,1\}$ selects processor i to evaluate the partition j, so that each partition is evaluated only once as it is expressed by the first constraint. The second constraint states that all partitions should be evaluated.

The load w_{ij} represents the processing cost to evaluate the partition j at processor i and depends mainly on the number of partitions itself, besides the number of records and variables of the dataset [3].

In this work, the efficiency results obtained in [3] were used to define the loads as:

$$w_{ij} = \frac{1}{\varepsilon_{ij}} \tag{4}$$

where ε_{ij} is the efficiency, i.e. the ratio between the speed-up and the number of processors, computed by the evaluation of the partition j on processor i.

4.2. Description of the algorithm

To ensure the optimization of the parallel processing time the definition of the knapsacks size (processing lines) is a critical factor. In the context of the FCM cluster analysis all partitions must be processed and is reasonable to conclude the ideal load the average of the partitions values, where, being K_i a set of partitions where $2 \le i \le n$ and p the number of processors, the average processing load would be:

$$S = \frac{1}{p} \sum_{i=1}^{p} \sum_{j=2}^{n} w_{ij} \tag{5}$$

This average size, however, cannot be always obtained in an exact manner. To ensure an efficient load balance strategy scalable for any set of partitions and number of processors, this work proposes a heuristic that uses two values: a lower limit defined by the partitions average (the ideal load for the distribution) and a superior limit, which is the maximum load in the distribution that indicates the least possible cost for the parallel processing.

The heuristic assemblies the knapsacks through an iterative process where a distribution is generated and evaluated by a variation coefficient passed as a parameter at the beginning of the parallel process. If the group of knapsacks generated achieves a good variation coefficient, the algorithm goes on processing the FCM cluster analysis. Otherwise, the initial average value S is increased. If the S adjusted value surpasses the distribution maximal load, the algorithm goes on processing the FCM cluster analysis. On the contrary, a new distribution is generated, in an attempt to push the loads to a smaller number of processors, trying to free processors that will be reallocated to the evaluation of the bigger loads in order to reduce the overall parallel processing time. The looping ends when the distribution evaluation is considered ok or when the lower limit gets equal to the superior limit.

The heuristic core is based on the well known first fit decreasing algorithm much used to treat the bin packing problem [13][14].

The parallel FCM cluster analysis balancing scheme is described by the following sequence:

Step1. (Master processor): **Initial values calculation**
Sort partitions in decreasing order
Calculate the average S as (5).

Step2. (Master processor): **Knapsacks generation**
2.1 Assign higher cost partitions
Place each partition with $w_{1j} > S$ into a single processor and consider the processor line full
Assign remaining partitions
For each partition j, place the partition in the next processor where lines sum >= partition cost
2.3 Assign partitions that did not fit in processing lines
Until all unassigned partitions are placed
Sort processing lines by load size
Place partition in the processor with the smaller total load
If it is the first distribution identify maximum load for the processing

Step3. (Master processor): **Knapsacks evaluation**
3.1Calculate average, standard deviation and the load variation coefficient
If variation coefficient > input variation
3.2 Adjust distribution parameters
a. $S = S + 1$
b. Cancel processors with maximum load with idle processors to find out the iteration maximum load
3.3 Evaluate if knapsack can be reorganized
a. If S < iteration maximal load
b. Returns to Step 2

Step4. (Master processor): **Assign idle processors**
If there is any idle processor
For each idle processor, assign to processor the next greatest load

Step5. (Master processor): **Communicate to processors**
Communicate processors´ partition lines
Communicate group information to processors who are part of a group

Step6. (Master processor): **Knapsack processing** *(All processors)*
6.1 If processor belongs to a group
Split data among the group processors
6.2 For each partition in processor calculate FCM and PBM loop
6.3 Send results to master processor

Step7. (Master processor): **Select partition with the greatest PBM index**

When the rate between number of partitions and number of processors is high the load balancing generated is usually a good one and presents very small variation coefficient (Figure 1). When the distribution using the average does not achieve a good result, such as in cases where the number of partitions is very close to the number of processors or, when the processors numbers are bigger than the number of partitions, the algorithm recalculates the knapsacks distribution, "pushing" the loads to a point, trying to group them in fewer processors, so that some processors can be freed to process along others the higher cost loads in an attempt of improve de overall load balance (Figure 2).

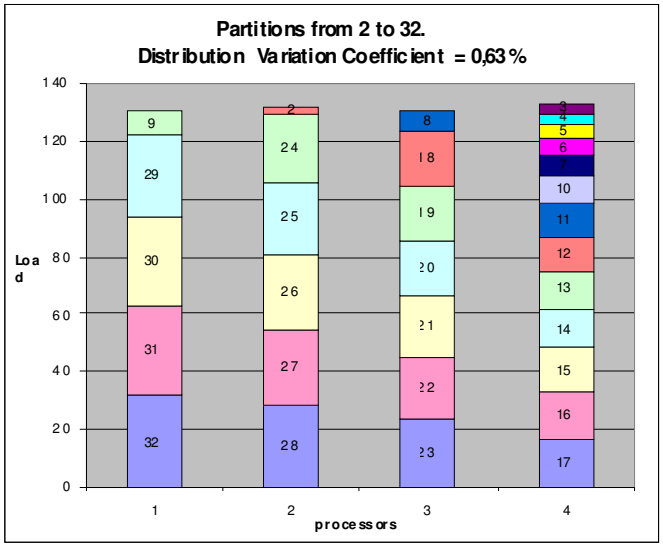


Figure 1 – Knapsacks generated for partitions range of 2 to 32 partitions distributed among four processors.

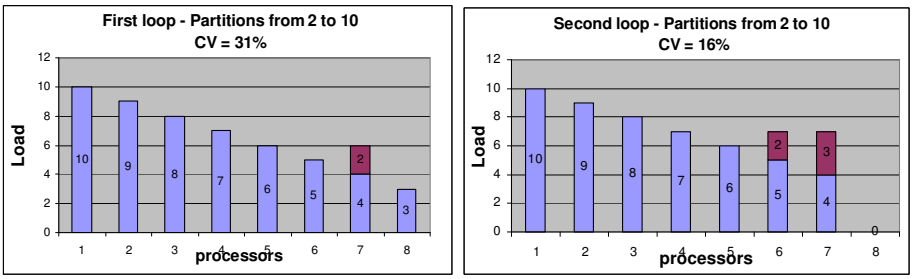


Figure 2 -Knapsacks generated in the first two iterations of the processing of the load balance algorithm when input variation coefficient = 20%.

5. Results and Discussion

5.1. Environment and test description

The SGI ALTIX 450 Venus machine with 32 Intel Itanium2 processor cores (1.6 GHz) e 64 GB of memory from the High Performance Computing Center (NACAD) of COPPE/UFRJ was used for execution and performance analysis of this work. Jobs execution was controlled by PBS (Portable Batch System) job scheduler. The application was developed using the C programming language and the Message Passing Interface (MPI) for processors communication.

The tests were conducted with a synthetic file of one million record and seven variables. The file was processed for ranges of 3, 7, 15 e 31 partitions. The range of three partitions had 2, 3 and 4 clusters, the range of seven partitions had values from 2 up to 8 clusters, the range of fifteen partitions had values from 2 up to 16 clusters and the range of 31 partitions had values from 2 up to 32 clusters.

5.2. Results and analysis

The load balanced FCM cluster analysis algorithm shows a significant reduction in processing time when compared to the prior approach [3] as showed in Table 1.

Table 1: Processing times of the synthetic dataset.

Processors	Partitions Range / Time (seconds)			
Balaced Algorithm	*2 - 4*	*2 - 8*	*2 - 16*	*2 - 32*
1	62.576763	238.519065	957.624869	4,042.916829
4	27.659887	65.000000	246.093375	1,026.675303
8	13.440102	46.530077	132.000000	517.987720
12	9.103440	31.832853	90.000000	352.376751
Unbalanced Algorithm				
1	119.448050	475.618360	1,794.868058	7,350.881991
4	30.383583	114.260539	451.347466	1,837.258185
8	15.636881	57.642829	224.130619	909.555979
12	10.358894	38.253603	148.851859	616.004967

Table 2: Ratio between partitions number and processors numbers

Ratio Partitions / Processors				
Procs	Number of Partitions			
	3	7	15	31
1	3.00	7.00	15.00	31.00
4	0.75	1.75	3.75	7.75
8	0.38	0.88	1.88	3.88
12	0.25	0.58	1.25	2.58

The best processing times happen when the rate of number of partitions by number of processors has the highest values (Table 2), which means that communication

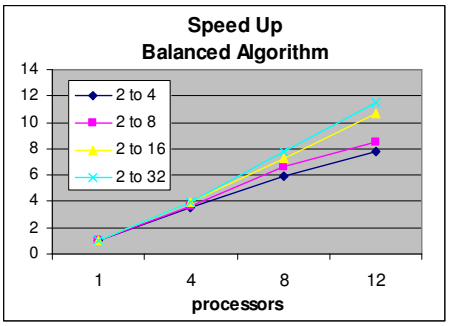
is the biggest hindrance to the good performance of the algorithm. On the other hand, when the values rate go lower, the time reduction rate decreases.

The balanced algorithm reduced the processing time in all tests (Table 3), but the best time savings advantages are for the biggest rates of number of partitions by number of processors.

Table 3: Percentage reduction values from comparing balanced and unbalanced algorithms when processing the one million lines dataset

	Partition´s Range			
Processors	**4**	**8**	**16**	**32**
1	**47.61%**	**49.85%**	**46.65%**	**45.00%**
4	**8.96%**	**43.11%**	**45.48%**	**44.12%**
8	**14.05%**	**19.28%**	**41.11%**	**43.05%**
12	**12.12%**	**16.78%**	**39.54%**	**42.80%**

The unbalanced algorithm speed up and efficiency values where compared to the balanced algorithm single processor time because the sequential processing time of this algorithm was the best of them.



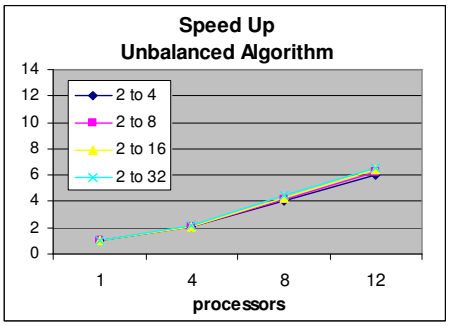
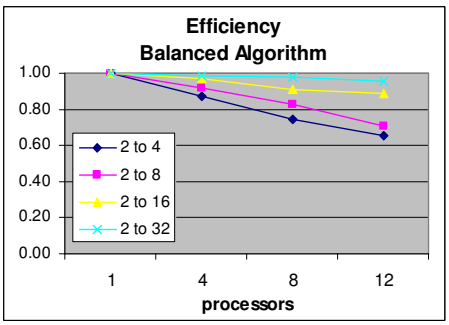


Figure 3 & Figure 4 - Balanced and unbalanced algorithms speed up values for one million lines dataset processing.



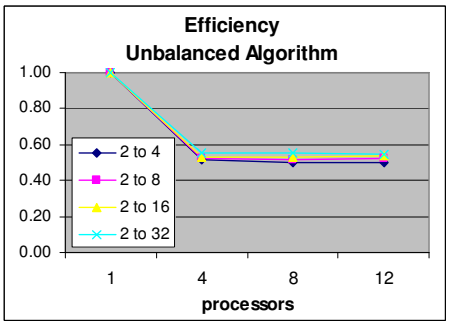


Figure 5 & Figure 6 – Balanced and unbalanced algorithms efficiency values for one million lines dataset processing.

The load balancing algorithm scales well for an increasing number of processors presenting good speed up and efficiency values, and in all cases presents better speed up and efficiency values than the unbalanced algorithm (Figure 4) (Figure 5).

The balanced algorithm presents best speed up values when processing bigger range of partitions revealing that minimizing communications is a very effective way of reducing parallel processing time.

6. Conclusions

This work presents a significant improvement to the performance of the parallel FCM cluster analysis algorithm [3]. The load balancing for FCM cluster analysis algorithm scales well and presents good efficiency for all parallel contexts, bringing new levels of performance to the parallel FCM cluster analysis.

Nevertheless, the applicability of this approach has to be improved with a strategy for establishing a lower bound for the algorithm, in order to not keep assigning processors to evaluate charges when there is no benefit from parallel execution, as in the situations of the analysis of small range of partitions. In such cases the parallel proposed approach would keep using machine capacity without any benefit.

Acknowledgements

This work has been supported by the Brazilian Research Council (CNPq), by the Brazilian Innovation Agency (FINEP) and by the National Petroleum Agency (ANP). The authors are grateful to High Performance Computing Center (NACAD-COPPE/UFRJ) where the experiments were performed.

References

[1] B. Boutsinas and T. Gnardellis (2002). On distributing the clustering process. *Pattern Recognition Letters* 23, pp. 999–1008.

[2] S. Rahimi, M. Zargham, A. Thakre and D. Chhillar (2004) A parallel Fuzzy C-Mean algorithm for image segmentation. *Proceedings of the IEEE Annual Meeting of the Fuzzy Information NAFIPS '04*, vol. 1, pp. 234 – 237.

[3] M. V. Modenesi, M. A. Costa, A. G. Evsukoff and N. F. F. Ebecken (2006). Parallel Fuzzy c-Means Cluster Analysis. *Proceedings of the 7th VecPar,* pp. 139-142.

[4] A. K. Jain, M. N. Murty and P. J. Flynn (1999). Data clustering: a review. ACM Computing Surveys, vol. 31, no. 3. pp. 264-323.

[5] J. C. Bezdek (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York, Plenum.

[6] X. L. Xie and G. A. Beni (1991). Validity measure for fuzzy clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3 no. 8, pp. 841–846.
[7] J. Bezdek and N.R. Pal (1998). Some new indexes of cluster validity. IEEE Trans. Systems Man and Cybernetics B, vol. 28, pp. 301–315.
[8] M. K. Pakhira, S. Bandyopadhyay and U. Maulik (2004). Validity index for crisp and fuzzy clusters. Pattern Recognition, vol. 37, pp. 487-501.
[9] G.B. Dantzig (1957) Discrete Variable Extremum Problems, Operations Research, 5, 266-277.
[10] L.G.Mitten (1970) Branch-And-Bound Methods: General Formulation and Properties *Operations Research*, Vol. 18, No.1, pp. 24-34
[11] S. Martello, P. Toth (1990). Knapsack Problems, Algorithms and Computer Implementations. John Wiley & Sons.
[12] D. Pissinger, P. Toth (1998). ”Knapsak problems” in Du, D-Z., Pardalos, P. (Eds), Vol 1, Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Dordrecht.
[13] Johnson, D., Near-Optimal Bin Packing Algorithms, Doctoral Thesis, MIT, Cambridge, 1973.
[14] R.L. Graham, "Bounds on multiprocessor timing anomalies" *SIAM J. Appl. Math.* , 17 (1966) pp. 416–429.