A New Access Method for Text Mining and its Parallelization

Un nuevo método de acceso en minería de textos y su paralelización

Fernando Artigas Fuentes[1], José Manuel Badía Contelles[2] y Reynaldo Gil García[1]
[1]Centro de Reconocimiento de Patrones y Minería de Datos, Universidad de Oriente, Santiago de Cuba, Cuba.
[2]Departamento de Ingeniería y Ciencias de los Computadores. Universidad Jaume I, Castellón, España.
artigas@cerpamid.co.cu; badia@icc.uji.es; gil@cerpamid.co.cu

Abstract

In this paper, a new access method for text mining and its parallelization is proposed. The new method is based on the use of multiple subgraphs as index structure, instead a single graph, as we propose in a previous work. The method uses a search algorithm to obtain the k-nearest neighbors for novel query objects. This method shows a good selectivity over very-high dimensional spaces, such as documents in text mining, and a similar or even better performance than other methods when applied to objects with fewer dimensions. Although it is an approximate method, it shows a low error rate. We parallelize both the graph index structure generation and the search process. We implement our algorithms on a shared memory multiprocessor using the OpenMP interface. We evaluate our proposal on data sets from the well-known Reuters collection and working with thousands of dimensions. Experimental results show that the new proposal overcomes our previous work in the use of the main memory, quality of results and the cost to generate the index structure. It shows almost optimal performances on a SGI Altix 350.

Keywords: text mining, parallel computation, approximated search, graphs.

Resumen

En este trabajo presentamos un nuevo método de acceso para la minería de textos. El nuevo método se basa en el uso de múltiples subgrafos como estructura de indexado, a diferencia del método presentado en un trabajo anterior que se basa en el uso de un solo grafo. El método usa un algoritmo de búsqueda para obtener los *k* vecinos más cercanos de un objeto de consulta. Este método muestra una buena selectividad sobre espacios de muy alta dimensionalidad, como son los documentos en la minería de textos, y un desempeño similar o mejor al de otros métodos, cuando es aplicado a objetos con menos dimensiones. Aunque es un método aproximado muestra un nivel de error muy bajo. Hemos paralelizado tanto la generación de la estructura de indexado como el proceso de búsqueda. Implementamos nuestros algoritmos sobre un multiprocesador con memoria compartida usando la interfaz OpenMP. Evaluamos nuestra propuesta usando conjuntos de datos de la bien conocida colección de Reuters y trabajamos con miles de dimensiones. Los resultados experimentales muestran que la nueva propuesta mejora nuestro trabajo previo en cuanto al uso de la memoria principal, la calidad de los resultados y el coste de la generación de la estructura de indexado. Este muestra un desempeño casi óptimo sobre un SGI Altix 350.

Palabras clave: minería de textos, computación paralela, búsqueda aproximada, grafos.

1 Introduction

Nowadays there is a well-know information explosion. The rapidly increasing amount of published information causes more and more problems to manage it. For a single user it is very difficult to separate the relevant data from the huge quantity of available information. The search engines such as Google or Yahoo are useful, but those tools only permit to search using a short list of keywords combined with a simple list of operators. Besides, the solution retrieved is a large list of documents (or links), containing the keywords but with a high probability to deal with different themes very distant from the original interest of the user.

An exciting alternative would be to search using a sample document in order to obtain only a short list of most related ones. The search of the most similar documents to a query is a basic operation in Text Mining. However, there are few access methods with the ability to manage data

spaces with a very high number of dimensions, such as documents.

The performance of exact methods applied to text documents decreases rapidly as the dimensionality of the data space grows due the curse of dimensionality [13]. There are a variety of solutions of this problem in the literature. The most common is the application of techniques to reduce the dimensionality to a value more manageable by currents methods. Examples of this kind of access methods are [7,8,12,15].

Other proposals involve the relaxation of the index structure construction or the search mechanism in order to obtain approximate solutions. Examples of access methods that use this variant are [11,14]. Those methods have a good performance but a low the accuracy. An inexact method that works fine for a kind of objects has a very bad precision for others.

We introduced in [1, 4] an approximate access method for indexing collections of objects representing a very high-dimensional space. The method uses a graph structure for indexing objects, such as documents in text mining, and a search algorithm that uses distance, similarity or dissimilarity based functions in order to obtain the k-nearest neighbors for novel query objects. Although the method can be applied over any objects with a very high number of dimensions, we are specifically interested in applying it over text mining.

The method has the ability to increase its selectivity with the size of text collection used to generate the index structure. Although it is an approximate method, it has a low error rate, that is, we obtain over 70% of exact solutions during the search of k nearest neighbors.

Although our method is static and the index structure is generated offline only once, the time to build it grows with the size of object collection. On the other hand, although the search of the neighborhood of a novel query object is very fast, when the number of objects to process (for example, a large flood of documents) increases, the processing time becomes also large. One solution to both problems is to reduce the cost by parallelizing the algorithms.

We introduced a parallelization of the index structure generation in [3], and two parallel search algorithms in [2]. However, those proposals have the problem of consuming a large amount of memory as the number of processors increases.

In this paper we introduce a new access method that uses multiple subgraphs as index structure, and we show the results of its parallelization. This new methods shows very good performances on a NUMA virtual shared memory multiprocessor and allows us to deal with larger problems.

The rest of the paper is organized as follows: in Section 2 our original sequential access method is briefly described, en Section 3 the new access method based in subgraphs is presented. In Section 4 a parallelization of this new method is described. In Section 5 our experimental results are shown, and Section 6 shows our conclusions.

2 Sequential access method

In this section, the two main phases of our sequential access method are briefly explained. In the first phase, the index graph structure is generated by using the objects in a database (a repository of documents, for example). In the second phase, the neighborhood of a novel object is found.

2.1 Index structure generation

The main idea of this phase is to construct a connected and non-oriented graph using all the objects in the database. This graph must fulfill the following conditions:

- Each vertex is an object of the database, represented as a vector.
- Each edge is a relationship between the two objects connected by the graph. This relationship can be a similarity, dissimilarity or distance value.
- Each vertex must be connected at least with a previously selected number of its nearest neighbors. This is the connectivity factor θ.

The main difference of our structure with others is that our graph has several entry points, and the best one is selected to start the second phase of the method: the search process.

The construction of the graph involves the followings main steps:

1. A connected graph is generated using all database objects.
2. The graph is completed by adding edges in such a way that each object is connected with its θ nearest neighbors (calculated using an exhaustive way).
3. The entry-points set is created with u_k objects from the graph that fulfill one of the following conditions:

    a. C1: $u_k \in U$ is external to $O \subset U$, for Ψ, iif $\Psi(u_k, \bar{u}) < \Psi(o_i, \bar{u})\ \forall o_i \in O$.
    b. C2: $u_k \in U$ is internal to $O \subset U$, for Ψ, iif $\Psi(u_k, \bar{u}) > \Psi(o_i, \bar{u})\ \forall o_i \in O$.

were U is the database, $\bar{u}$ is the global centroid of U, O is the set of objects connected with u_k by the graph, and ψ is the particular distance, similarity or dissimilarity measure used in the specific problem.

A more detailed description of the method can be found in [1].

2.2 Search method

In this phase, given a novel object q as query, its k nearest neighbors in the graph structure are found. In order to avoid the exhaustive graph traversal, a fast search algorithm is used.

The search is performed in two main steps. If only one solution is required, it is obtained using the first step of the following algorithm. If more solutions are needed the second step is used.

Step 1: The nearest neighbor object (NN) of q in the graph is calculated.

a) The EP most related objects with q by Ψ are selected from the entry-points set. Those objects belong to the initial NN solution set.
b) The most related object with q by Ψ is selected from the indexed objects connected with any object in the current NN, and it becomes the new NN solution.
c) Step b) is repeated until no object connected to the current NN solution is more related to q by Ψ than it.
d) Set kNN, the initial solution set, equal to NN solution.

Step 2: If more of one object is needed in the solution:

a) Set a search radio equal to the worst possible relationship between two objects as it is defined by Ψ.
b) Select at most k new objects with relationship value equal or greater value than the radio from indexed objects connected with any objects in kNN solution set. (Note: if you use a measurement like distance or dissimilarity you need to change the relation to "equal or minus")
c) Select the k objects most related to q from the union of objects in current kNN and objects obtained in the previous step. These objects become the new kNN solution set. If no new objects were found, return kNN and end the algorithm.

3 Sequential access method using subgraphs

In this section we describe a new variant of our access method that instead of using a single graph consists of several subgraphs containing together all the objects.

The goal of this proposal is to reduce the complexity of the index structure, and therefore to decrease the time and storage required to build it. Besides, we can directly apply our parallel algorithm in [3] to build each subgraph and this new index structure allows us a natural parallelization of the search process.

The main difference with the previous sequential approach is the following. The database objects are divided into a number of disjoint subsets. Then, the algorithm described in section 2.1. is applied to build a connected and non-oriented graph containing the objects of each subset.

This change not only affects the index structure generation process, but also the search phase. Each query must now be evaluated for each subgraph, using the algorithm described in section 2.2, getting several partial solutions of the

search. The final solution is obtained by selection of the best values from the partial solutions.

The first improvement of this new method is the reduction of the time to build the index structure. This is due to the quadratic cost of the generation process. That implies that when the original problem is divided into subproblems, the sum of costs needed to generate all the subgraphs is smaller than of the cost of generating the single graph containing all the objects. The relation (1) shows that.

$$\sum_{i=1}^{P}\left(\frac{N}{P}\right)^2 < N^2,\ P > 1 \qquad (1)$$

Also, this cost must be smaller as the value of P increases, as is shown by relation (2):

$$N^2 > \sum_{i=1}^{2}\left(\frac{N}{2}\right)^2 > \sum_{i=1}^{3}\left(\frac{N}{3}\right)^2 \ldots \qquad (2)$$

The second improvement of the new method is to reduce the dimensionality of the representation space. When the database is divided into subsets, each subset produces an index structure that represents a data space with a smaller number of dimensions than the original one.

As we decrease the number of objects on each graph, the number of dimensions involved also decreases. This must benefit the search algorithm, since the paths between pairs of objects that are not directly connected by the graph must be shorter. Therefore, the algorithm will find them with less iterations. This partially compensates for the fact that we must now search through all subgraphs.

In summary, with the use of subgraphs we expect to obtain the following main improvements:

1. Building a set of subgraphs is clearly faster than building one graph containing all the objects.
2. The maximum dimensionality of the data spaces of the subgraphs is smaller than the dimensionality involved by the whole graph.
3. The accuracy of solutions must be similar or better that using a single graph.

4 Parallel access method

In this section, we describe the parallelization of both phases of the access method.

4.1 Preliminary work

As previous work, a parallelization of the graph index structure generation, using the OpenMP interface [6], was presented in [3]. With this parallelization superspeedup were obtained (Figure 1), although the step 2 of the algorithm, the completion of the graph, was not parallelized.

We think this result is due the better utilization of memory hierarchy when data is distributed in local memories of processors.

Speedup is defined by the following formula:

$$S_p = \frac{T_1}{T_p} \qquad (3)$$

where T_1 is the execution time of the sequential algorithm and T_p is the execution time of the parallel algorithm with *p* processors.

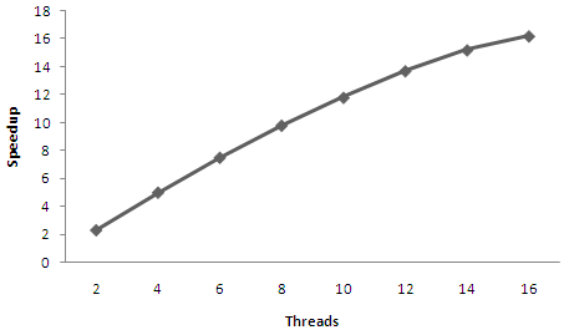

Figure 1. Performance of the parallel construction of the graph in [3].

In addition, two parallelization of the search algorithm using OpenMPI [9] were presented and discussed in [2].

The first one, GCP, was based on the fully parallel scheme (Figure 2). Each processor has its own subset of queries and works independently from the rest.

The second parallel search algorithm, GME, was based on the farm scheme (or master-slave).

One of the processors works as master, and the rest, as workers. The master controls and dynamically distributes the queries between the workers as they are requested. Then workers compute solutions and send them to the master. This process is repeated while new queries are available (Figure 3).

With the first parallelization an almost perfect speedup was obtained. The master-slave scheme balances the workload among the processors, but introduces communications that reduce the speedup of the algorithm (Figure 4).

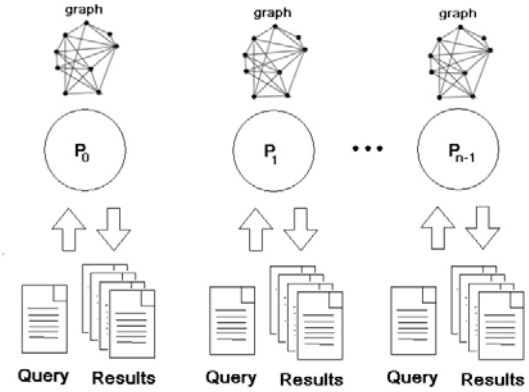

Figure 2. Fully parallel scheme of the search process, GCP.

The previous parallelizations have a major drawback. As the number of processors increases, so does the memory usage. This is because each processor needs a copy of the graph.

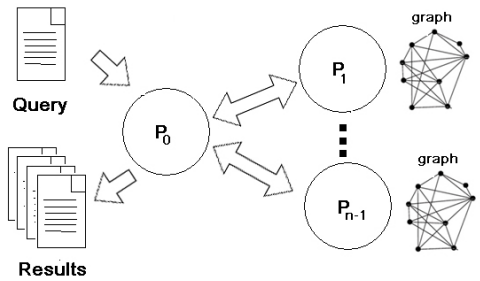

Figure 3. Master-slave parallel schema of the search process, GME.

This problem can be solved by applying the new algorithm described in Section 3. In this case, to build the index structure we only deal at each moment with the objects of one subgraph. Afterwards, during the search process, each processor only stores one subgraph and not a copy of the whole graph. Therefore, the memory usage is almost maintained as we increase the number of processors.

Besides, this new methods allows the processing of larger problems, including more objects.

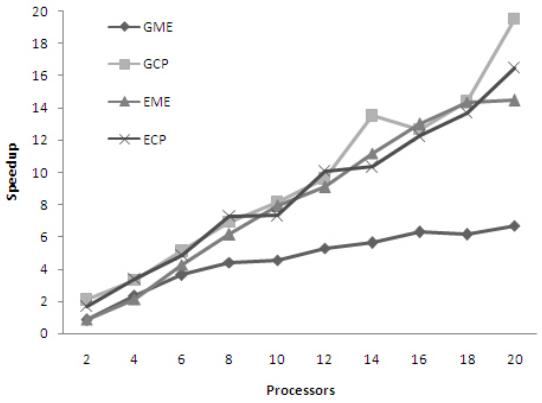

Figure 4. Speedup for both fully (GCP) and farm (GME) parallel searching schemas.

4.2 Parallel index structure generation.

In [3], we presented a parallelization of each phase of the index structure generation. This work is still valid, and in this paper we only introduce two changes.

The first major change is that we have to build in parallel the new index structure. Instead of building a single graph we build several subgraphs. A first approach to deal with this process, applied to the algorithm presented on this paper, is to use the parallel algorithm in [3] to build successively each of the subgraphs. We construct as much subgraphs as processors, so that during the search phase each processor locates the neighbors to the queries on a different subgraph.

A second approach to build the subgraphs in parallel is that each processor applies the sequential algorithm to generate one of the subgraphs. We are now finishing the implementation and analysis of the second approach.

The second change to the method in [3] is the parallelization of the step 2 of the algorithm. In this step, each vertex of the graph is explored in order to determine if it is connected at least with its θ nearest neighbors. The problem is that when a vertex needs to be completed, two new edges for each new neighbor connected are created, one from and other to the new neighbor. If two vertices are explored in parallel and both need to be connected each one with the other, only one of the processes should create a particular edge, in order to avoid repeated edges into the graph.

In order to solve this problem, when we parallelized this step, we introduced a critical section in the insertion of new edges. One processor only inserts a new edge if it has not been previously inserted by another processor.

The following algorithm shows a simplified OpenMP coding of the process:

```
#pragma omp parallel for private(i)
for(i=0;i<TotalNumOfVertices;i++){

#pragma omp critical
{
  if(!InsertedEdge(edge_from))
       InsertEdge(edge_from);
}
#pragma omp critical
{
  if(!InsertedEdge(edge_to))
       InsertEdge(edge_to);
}

}
```

The use of critical section solves the problem of multiple insertions of the same edge, but involves additional testing code on each process.

4.3 Parallel search method

As we stated before, the parallel construction and storage of the subgraphs leads to a distributed search method.

We organized the processors with the same master-slave schema of Figure 3, but now, each processor will work with one of the subgraphs. Each processor will find a local solution for the query and communicate it to the master processor. Finally, the best solution will be selected by this one, in the same way as described in section 3 (Figure 5).

To exploit all available computing resources, the master processor will also have assigned a subgraph, and participate in the search process.

Local searches will be performed using the same algorithm described in section 2.2. We have also considered the possibility of parallelizing the searching of the nearest neighbors of each object on each graph. However, this is a very fast process and the small granularity of the computations involved would surely produce poor parallel performances.

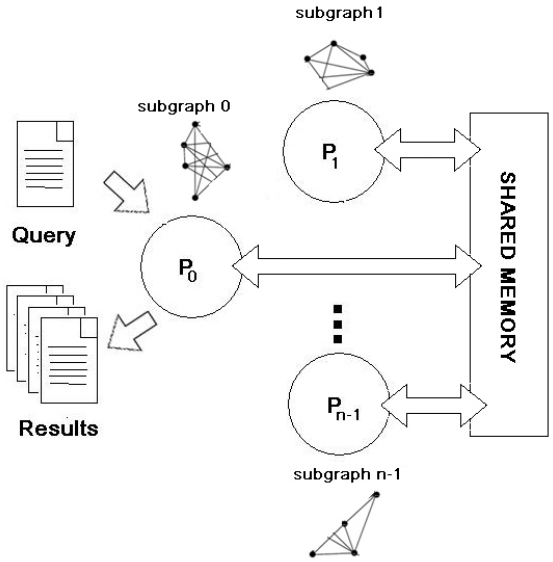

Figure 5. Master-slave search process using subgraphs.

5 Experimental analysis

In this section we show the results obtained when we evaluate our proposal on data sets from a well-known benchmark collection and working with thousands of dimensions.

5.1 Experimental environment

To perform our experiments, we use the well-known benchmark collection Reuters corpus version 1 (RCV1-v2) [10]. This collection has a set of documents represented as vectors.

The feature vector for each document was produced from the concatenation of text in the <headline> and <text> XML elements. Text was reduced to lowercase characters, after which tokenization, punctuation removal and stemming, stop word removal, term weighting, feature selection, and length normalization was applied.

The LYRL2004 partition, with 23.149 training, and 781.265 testing vectors, was used. The representation space of this partition has 47.152 of dimensions. But only the training subset was used, for both database and query objects.

All experiments were performed on a CC-NUMA virtual shared memory multiprocessor SGI Altix 350. This machine has 8 nodes, each containing 2 Intel Itanium2, 1.5 GHz processors. Each node has 4 GBytes of local memory connected via a SGI NUMAlink network, adding up to 32 GBytes of shared memory.

All the algorithms were implemented using C language, and compiled with ICC 9.0, over Linux OS.

5.2 Experimental results

To study the behavior of our new proposal several index structures with different numbers of subgraphs were generated for both sequential and parallel schemas.

The results were obtained for the different folds of the 10-fold-cross technique. The experimental values analyzed are the average of those, if nothing else is indicated. For each experiment we measure the time taken for each phase, the memory consumption and the dimensionality of the represented space.

5.3 Accuracy of results using subgraphs

The first aspect to be studied is the quality of search results of the new approximate access method. We compare the results obtained for all the queries against the results obtained with an exhaustive search method that guarantees the exact solution.

Figure 6 shows that the accuracy of the results increases with the number of subgraphs. The increase was from 92 percent for 1 single graph to almost 97 percent for 16 subgraphs.

The increase in the accuracy of the results by using subgraphs is one of the advantages of this index structure.

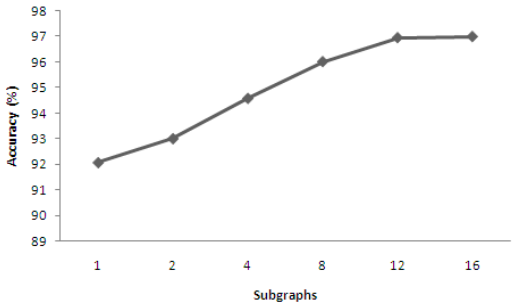

Figure 6. Accuracy of the results of searches against the obtained with the exhaustive method, varying the number of subgraphs.

5.4 Dimensionality of the solution

In order to analyze the dimensionality of problems associated to each subgraph, we use the size of its global centroid. The global centroid contains all the dimensions of the problem, because it is the sum of all vectors on the subgraph. Figure 7 shows the dimensionality of the largest centroid of all the subgraphs as we increase the number of subgraphs.

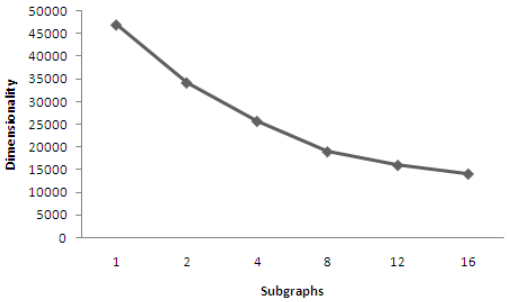

Figure 7. Maximum dimensionality of the problem, varying the number of subgraphs.

As expected, the maximum dimensionality of the representation spaces decreases as the number of subgraphs increases, because we are dealing with smaller subgraphs. Figure 7 shows that the dimensionality varies from almost 50.000 for the complete graph to approximately 14.000 dimensions with 16 subgraphs.

5.5 Parallelization of the completion process

Figure 8 shows the impact of parallelizing the step 2 of the algorithm of index structure generation. As it can be seen, the sequential time decreases as we increase the number of subgraphs. We can also see that the parallel time decreases as we increase the number of subgraphs and the number of processors. Although, the parallel time is lesser in all the cases, the difference is reduced as a larger number of processors and subgraphs are used.

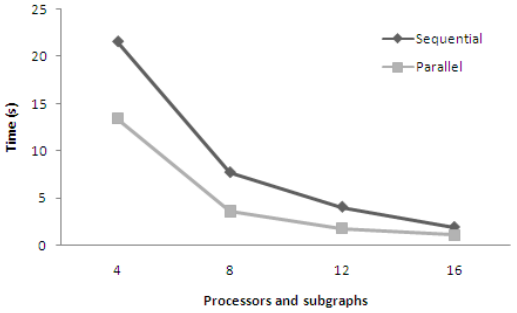

Figure 8. Time cost to complete the subgraphs (step 2) for both the sequential and parallel versions, incrementing the number of processors and subgraphs.

The cause of this effect is that increasing the number of processors the influence of the communication cost and the remaining sequential code (the critical section, for example) increases.

5.6 Time cost of subgraphs generation

Figure 9 shows the sequential time to generate from 1 to 16 subgraphs with the same set of objects. It also shows the parallel time to construct the same subgraphs using on each case as much processors as subgraphs are built. As it can be seen, both the sequential and parallel times decrease as we increase the number of subgraphs.

Figure 10 shows the speedup of the parallel algorithm used to generate the subgraphs. We can see that the speedups are always near to the optimum.

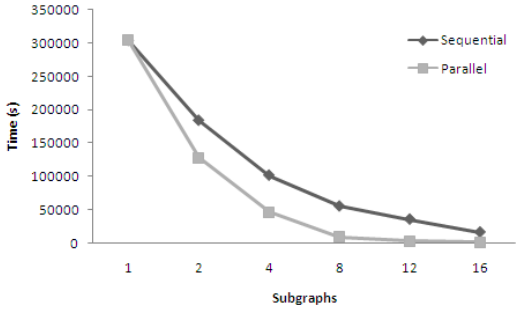

Figure 9. Time in seconds to build the index structure varying the number of subgraphs, for both sequential and parallel versions.

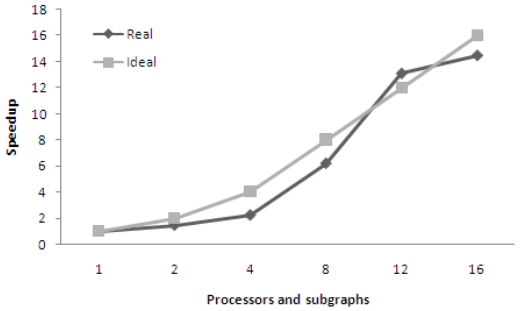

Figure 10. Speedup for generating subgraphs in parallel. The number of subgraphs is equal to the number of processors used.

5.7 Memory usage of the search method

Figure 11 shows the memory usage of the parallel search method in [3] and of the new parallel algorithm. As we noticed before the method in [3] stores one copy of the whole graph for each processor, and therefore its memory usage increases linearly. On the other hand, the memory usage of the new parallel algorithm based on subgraphs is almost maintained as we increase the

number of processors, because the subgraphs always store the same set of objects.

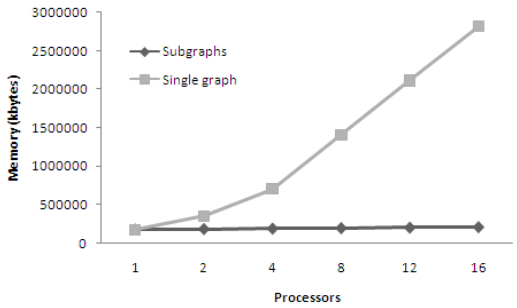

Figure 11. Memory usage of both the single graph and the subgraphs methods.

Figure 12 adapts the scale of the graphic to show the small increase of the memory usage of the subgraphs method as we increase the number of processors. This increase is due to additional data structures associated to each subgraph, such as the list of entry points and the global centroid of the subgraphs.

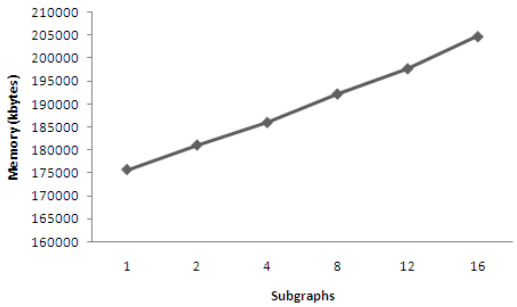

Figure 12. Memory usage of the subgraphs method.

5.8 Time cost of searches

Figure 12 shows the average sequential time of a query varying the number of subgraphs. As it can be seen, the cost of a query increases with the number of subgraphs. We have seen that the cost of searching in a graph is quite smaller than the cost of searching in several subgraphs containing together the same objects. That is, halving the number of objects in the graph does not halve the searching time.

However, the times observed are low enough to still consider the search algorithm on multiple subgraphs as fast.

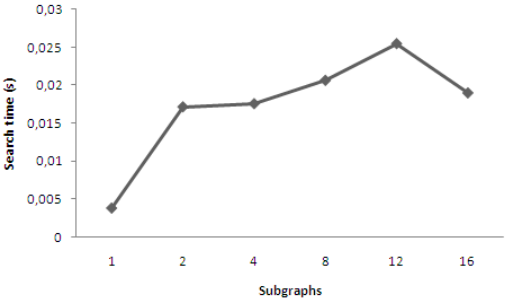

Figure 12. Average time of a query varying the number of subgraphs.

6 Conclusions and future work

In this work, a new sequential and parallel access method for text mining, a problem with a very high dimensionality, was proposed. It is based in the use of multiple subgraphs as index structure, instead a single graph as our previous access method presented in [1]. Experimental results show that our new access method improves the previous one in the use of memory, the quality of results and the cost to generate the index structure. However, using several subgraphs increases the time of the search process.

On the other hand, the huge reduction in the memory usage of the new methods allows us to deal with larger problems.

We are now finishing the implementation and analysis of a second approach that uses an alternative method to parallelize the index structure generation.

7 Acknowledgment

The authors acknowledge support from the Engineering and Computer Science Department of University Jaume I, Spain.

References

[1] **Artigas-Fuentes, F., Badía-Contelles, J. M. and Gil-García, R. (2010).** A High-dimensional Access Method for Approximated Similarity Search in Text Mining. *ICPR 2010, 20th International Conference on Pattern Recognition*, Istanbul, Turkey. 3155- 3158.

[2] **Artigas-Fuentes, F., Badía-Contelles, J. M. and Gil-García, R. (2010)**. Búsqueda paralela de los vecinos más cercanos en Minería de Textos. *Resumen de las XXI Jornadas de Paralelismo (SARTECO 2010) CEDI 2010,* UPV, Valencia, España, 93-100.

[3] **Artigas-Fuentes, F., Badía-Contelles, J. M., Gil-García, R. and Pons-Porrata, A. (2008)**. Algoritmo paralelo de generación de una estructura de indexado basada en grafos. *Actas de las XIX Jornadas de Paralelismo*, UJI, Castellón, España. 93-98.

[4] **Artigas-Fuentes, F., Badía-Contelles, J. M., Gil-García, R. and Pons-Porrata, A. (2008)**. Cálculo de la vecindad mediante grafos en minería de textos, *Universidad Ciencia y Tecnología*, 48, 1-10.

[5] **Berchtold, S. et al. (2000)**. Independent quantization: an index compression technique for high dimensional data spaces, *ICDE'00.* San Diego, California, USA. 577-588.

[6] **Chandra, R. et all. (2001)**. Parallel Programming in OpenMP. *Morgan Kaufmann Publisher.* San Francisco, CA. 230.

[7] **Ciecielski, K. et al. (2007).** Histogram-based dimensionality reduction of term vector space. *CISIM'07.* 28(30). 103-108.

[8] **Faloustus, C. and Bhagwat, P. (1993)**. Declustering using fractals. *PDIS Journal of parallel and distributed information systems.* 18-25.

[9] **Gabriel, E., et al. (2004)**. Open MPI: goals, concept and design of a next generation MPI implementation. *In proc. of 11th European PVM/MPI users' group meeting.* Budapest, Hungary.

[10] **Lewis, D. L. et al. (2004)**. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research.* 5, 361-397.

[11] **Navarro, G. (2002)**. Searching in metric spaces by spatial approximation. *VLD-VJ'02.* 711(1).

[12] **Pagel, B. et al (2000).** Deflating the dimensionality curse using fractal dimensions. *ICDE.* San Diego. California, USA. 589-598.

[13] **Schek, H. et al**. (1998) A quantitative analysis and performance study for similarity-search methods in high dimensional spaces. *VLDB'98*, New York, USA. 194-205.

[14] **Weber, R. and Blott, S. (1997)**. An approximation based data structure for similarity search. *Technical report 24, ESPRIT project ERMES (no. 9141).* 1-25.

[15] **Yang, Y. and Pederson, J. (1997)**. *ICML'97.* Tennessee, USA. 412-420.