PGP-mc: Towards a Multicore Parallel Approach for Mining Gradual Patterns

Anne Laurent[1], Benjamin Negrevergne[2], Nicolas Sicard[3], and Alexandre Termier[2]

[1] LIRMM - UM2- CNRS UMR 5506 - 161 rue Ada - 34392 Montpellier Cedex 5
laurent@lirmm.fr
http://www.lirmm.fr
[2] LIG - UJF-CNRS UMR 5217 - 681 rue de la Passerelle, B.P. 72, 38402 Saint Martin d'Hères
Benjamin.Negrevergne@imag.fr, Alexandre.Termier@imag.fr
http://www.liglab.fr
[3] LRIE - EFREI - 30-32 av. de la république, 94 800 Villejuif
nicolas.sicard@efrei.fr
http://www.efrei.fr

Abstract. Gradual patterns highlight complex order correlations of the form *"The more/less X, the more/less Y"*. Only recently algorithms have appeared to mine efficiently gradual rules. However, due to the complexity of mining gradual rules, these algorithms cannot yet scale on huge real world datasets. In this paper, we propose to exploit parallelism in order to enhance the performances of the fastest existing one (GRITE). Through a detailed experimental study, we show that our parallel algorithm scales very well with the number of cores available.

1 Introduction

Frequent pattern mining is a major domain of data mining. Its goal is to efficiently discover in data patterns having more occurrences than a pre-defined threshold. This domain started with the analysis of transactional data (frequent itemsets), and quickly expanded to the analysis of data having more complex structures such as sequences, trees or graphs. Very recently, a new pattern mining problem appeared: mining frequent *gradual itemsets* (also known as *gradual patterns*). This problem considers transactional databases where attributes can have a numeric value. The goal is then to discover frequent co-variations between attributes, such as: "The higher the age, the higher the salary". This problem has numerous applications, as well for analyzing client databases for marketing purposes as for analyzing patient databases in medical studies. Di Jorio et al. [1] recently proposed GRITE, a first efficient algorithm for mining gradual itemsets and gradual rules capable of handling databases with hundreds of attributes, whereas previous algorithms where limited to six attributes [2]. However, as gradual itemset mining is far more complex than traditionnal itemset mining,

GRITE cannot yet scale on large real databases, having millions of lines and hundreds or thousands of attributes.

One solution currently investigated by pattern mining researchers for reducing the mining time is to design algorithms dedicated for recent multi-core processors [3, 4]. Analyzing their first results shows that the more complex the patterns to mine (trees, graphs), the better the scale-up results on multiple cores can be. This suggests that using multicore processors for mining gradual itemsets with the GRITE algorithm could give interesting results. We show in our experiments that indeed, there is a quasi-linear scale up with the number of cores for our multi-threaded algorithm.

The outline of this paper is as follows: In Section 2, we explain the notion of gradual itemsets. In Section 3, we present the related works on gradual patterns and parallel pattern mining. In Section 4, we present our parallel algorithm for mining frequent gradual itemsets, and Section 5 shows the results of our experimental evaluation. Last, we conclude and give some perspectives in Section 6.

2 Gradual Patterns

Gradual patterns refer to itemsets of the form *"The more/less X_1, ..., the more/less X_n"*. We assume here that we are given a database DB that consists of a single table whose tuples are defined on the attribute set $\mathcal{I}$. In this context, gradual patterns are defined to be subsets of $\mathcal{I}$ whose elements are associated with an ordering, meant to take into account increasing or decreasing variations. Note that $t[I]$ hereafter denotes the value of t over attribute I.

For instance, we consider the database given in Table 1 describing fruits and their characteristics.

Id	Size (S)	Weight (W)	Sugar Rate (SR)
t_1	6	6	5.3
t_2	10	12	5.1
t_3	14	4	4.9
t_4	23	10	4.9
t_5	6	8	5.0
t_6	14	9	4.9
t_7	18	9	5.2

Table 1. Fruit Characteristics

Definition 1 *(Gradual Itemset) Given a table DB over the attribute set $\mathcal{I}$, a gradual item is a pair (I, θ) where I is an attribute in $\mathcal{I}$ and θ a comparison operator in $\{\geq, \leq\}$.*

A gradual itemset $g = \{(I_1, \theta_1), ..., (I_k, \theta_k)\}$ is a set of gradual items of cardinality greater than or equal to 2.

For example, $(Size, \geq)$ is a gradual item, while $\{(Size, \geq), (Weight, \leq)\}$ is a gradual itemset.

The support of a gradual itemset in a database DB amounts to the extend to which a gradual pattern is present in a given database. Several support definitions have been proposed in the literature (see Section 3 below). In this paper, we consider the support as being defined as the number of tuples that can be ordered to support all item comparison operators:

Definition 2 *(Support of a Gradual Itemset) Let DB be a database and $g = \{(I_1, \theta_1), ..., (I_k, \theta_k)\}$ be a gradual itemset. The cardinality of g in DB, denoted by $\lambda(g, DB)$, is the length of the longest list $l = \langle t_1, \ldots, t_n \rangle$ of tuples in DB such that, for every $p = 1, \ldots, n-1$ and every $j = 1, \ldots, k$, the comparison $t_p[I_j]\, \theta_j\, t_{p+1}[I_j]$ holds.*

The support of g in DB, denoted by $supp(g, DB)$, is the ratio of $\lambda(g, DB)$ over the cardinality of DB, which we denote by $|DB|$. That is, $supp(g, DB) = \frac{\lambda(g, DB)}{|DB|}$.

In the example database, for the gradual itemset $g = \{(S, \geq), (SR, \leq)\}$, we have $\lambda(g, DB) = 5$, with the list $l = \langle t_1, t_2, t_3, t_6, t_4 \rangle$. Hence $supp(g, DB) = \frac{5}{7}$.

3 Related Works

Gradual patterns and gradual rules have been studied for many years in the framework of control, command and recommendation. More recently, data mining algorithms have been studied in order to automatically mine such patterns [1, 2, 5–8].

The approach in [7] uses statistical analysis and linear regression in order to extract gradual rules. In [2], the authors formalize four kinds of gradual rules in the form *The more/less X is in A, then the more/less Y is in B*, and propose an Apriori-based algorithm to extract such rules. Despite a good theoretical study, the algorithm is limited to the extraction of gradual rules of length 3.

In [1] and [5], two methods to mine gradual patterns are proposed. The difference between these approaches lies in the computation of the support: whereas, in [5], a heuristic is used and an approximate support value is computed, in [1], the correct support value is computed.

In [8], the authors propose another way to compute the support, by using ranking such as the Kendall tau ranking correlation coefficient, which basically computes, instead of the length of the longest path, the number of pairs of lines that are correctly ordered (concordant and discordant pairs).

To the best of our knowledge, there are no existing *parallel* algorithms to mine gradual itemsets. The most advanced works in parallel pattern mining have been presented by [3] for parallel graph mining and [4] for parallel tree mining. These works have showed that one of the main limiting factor for scalable parallel perfomance was that the memory was shared among all the cores. So if all the cores request a lot of data simultaneously, the bus will be saturated and the

performance will drop. The favorable case is to have compact data structures and complex patterns where a lot of computations have to be done for each chunk of data transfered from memory.

With its complex support computation and simple input data, gradual pattern mining is thus a favorable case for parallelization. The main difficulty will be to achieve a good load balance. We present our solution in the following section.

4 PGP-mc: Parallel Gradual Pattern Extraction

The sequential GRITE algorithm (see [1] for detailed algorithm) relies on a tree-based exploration, where every level $N+1$ is built upon the previous level N. The first level of the tree is initialized with all attributes, which all become itemset siblings. Then, *itemsets* from the second level are computed by combining *frequent itemsets* siblings from the first level through what we call the $Join()$ procedure. Candidates which match a pre-defined threshold - they are considered as *frequent* - are retained in level $N+1$.

In this approach, every level cannot be processed until the previous one has been completed, at least partially. So, we focused our efforts on the parallelization of each level construction where individual combinations of itemsets (through the $Join()$ procedure) are mostly independant tasks. The main problem is that the number of operations cannot be easily anticipated, at least for levels higher than 2. Moreover, the number of siblings may vary by a large margin depending of the considered itemsets. A simple parallel loop would lead to an irregular load distribution on several processing units.

In order to offset this irregularity, our approach dynamically attributes new tasks to a pool of threads on a "first come, first served" basis. At first, all frequent itemsets from the given level are marked unprocessed and queued in Q_i. A new frequent itemset i is dequeued and all its siblings are stored in a temporary queue Q_{si}. Each available thread then extracts the next unprocessed sibling j from Q_{si} and builds a new candidate k from i and j. The candidate is stored in level $N+1$ if it is considered frequent. When Q_{si} is empty, the next frequent itemset i is dequeued and Q_{si} is filled with its own siblings. The process is repeated until all itemsets i are processed (*e.g.,* Q_i is empty).

5 Experimental Results and Discussion

In this section we report experimental results from the execution of our program on two different workstations with up to 32 processing cores : COYOTE, with 8 AMD Opteron 852 processors (each with 4 cores), 64GB of RAM with Linux Centos 5.1, g++ 3.4.6 and IDKONN, with 4 Intel Xeon 7460 processors (each with 6 cores), 64GB of RAM with Linux Debian 5.0.2, g++ 4.3.2.

Most of the experiments are led on synthetic databases automatically generated by a tool based on an adapted version of IBM Synthetic Data Generation Code for Associations and Sequential Patterns[4]. This tool generates numeric

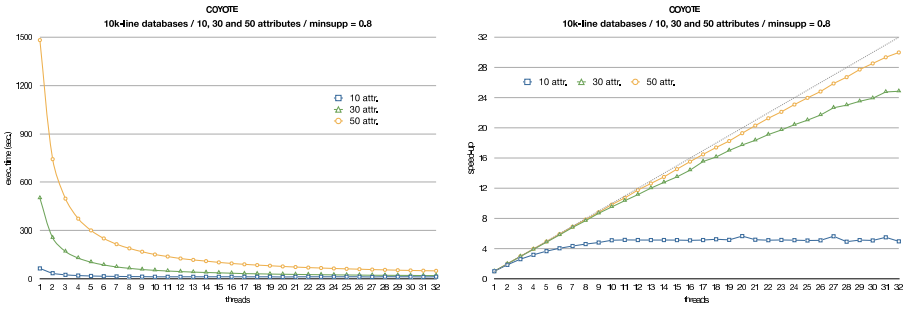
[4] www.almaden.ibm.com/software/projects/hdb/resources.shtml

databases depending on the following parameters: number of lines, number of attributes/columns and average number of distinct values per attribute.

5.1 Scalability

The following figures illustrate how the proposed solution scales with both the increasing number of threads and the growing complexity of the problem. The complexity comes either from the number of lines or from the number of attributes in the database as the number of individual tasks is related to the number of attributes while the complexity of each individual task - itemsets joining - depends on the number of lines. In this paper, we report results for two sets of experiments.

The first experiment set involves databases with relatively few attributes but a significant number of lines. This kind of databases usually produces few frequent items with moderate to high thresholds. As a consequence the first two level computations represent the main part of the global execution time. Figures 1(a) and 1(b) show the evolution of execution time and speed-up respectively for 10000-line databases - ranging from 10 to 50 attributes - on COYOTE.



(a) Exec. time (b) Speed-up

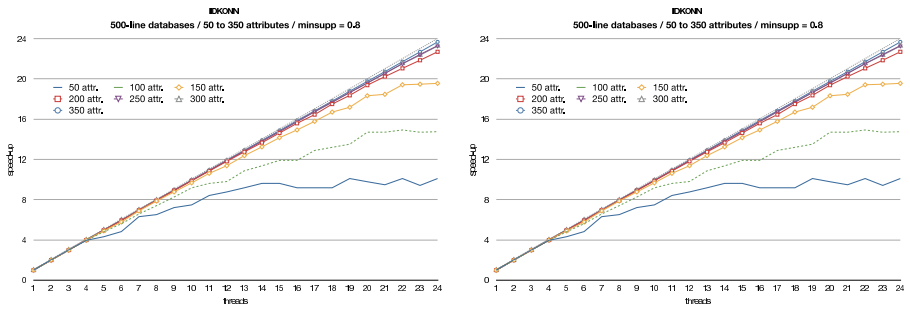
Fig. 1. Execution time and speed-up related to the number of threads. Test databases ranging from 10 to 50 attributes with 10k lines, on COYOTE.

As shown by Figure 1(a), speed-ups can reach very satisfying values in sufficiently complex situations. For example, speed-up is around 30 with 50 attributes where the theoretical maximum is 32. The upper limit for 10 and 20 attributes is not really surprising and can be explained by the lower number of individual tasks. As the number of tasks decreases and the complexity of each task increases, it becomes more and more difficult to reach an acceptable load balance. This phenomenon is especially tangible during the initial database loading phase (construction of the first level of the tree) where the number of tasks is exactly the number of attributes. For example, the sequential execution on the

10-attribute database takes around 64 seconds from which the database loading process takes 9 seconds. With 32 threads, the global execution time goes down to 13 seconds but more than 5.5 seconds are still used for the loading phase.

Experimental results on IDKONN are very similar to these figures as speed-ups go from a maximum of 4.8 with 24 threads on the 10-attribute database to a maximum of 22.3 with 24 threads on the 50-attribute database. Detailed results on IDKONN are available at `http://www.lirmm.fr/~laurent/DASFAA10`.

The second set of experiments reported in this article is about databases with growing complexity in term of attributes. Figures 2(a) and 2(b) show the evolution of execution time and speed-up respectively for 500-line databases with various number of attributes - ranging from 50 to 350 - on IDKONN.



(a) Exec. time (b) Speed-up

Fig. 2. Execution time and speed-up related to the number of threads. Test databases ranging from 50 to 350 attributes with 500 lines, on IDKONN.

As we can see, our solution is extremely efficient and scales very well for many attributes: we almost reach the theoretical maximum linear speed-up progression for 150 attributes or more. For example, the sequential processing of the 350 attributes database took more than five hours while it spend approximatively 13 minutes using 24 threads on IDKONN. Furthermore, speed-up results are particularly stable from one architecture to another[5], meaning that performances do not rely on very specific architectural features (caches, memory systems...).

With an execution time of less than 0.2 second with 16 threads, the 50-attribute database experiment illustrates how our approach can still achieve a very tangible acceleration on this particular case, which appears as crucial for real time or near real time data mining and applications (*e.g.,* intrusion/fraud detection).

[5] Complete experiments, detailed at *http://www.lirmm.fr/~laurent/DASFAA10*, show very similar results on COYOTE (with 32 threads).

6 Conclusion and Perspectives

In this paper, we propose an original parallel approach to mine large numeric databases for gradual patterns like *the oldest a people, the higher his/her salary*. Mining these rules is indeed very difficult as the algorithms must perform many time-consuming operations to get the frequent gradual patterns from the databases. In order to tackle this problem, our method intensively uses the multiple processors and cores that are now available on recent computers. The experiments performed show the interest of our approach, by leading to quasi-linear speed-ups on problems that were previously very time-consuming or even impossible to manage, especially in the case of databases containing a lot of attributes.

This work opens many perspectives, not only based on technical improvements depending on ad-hoc architectures of the machines, but also based on other data mining paradigms. Hence we will consider closed gradual patterns in order to cut down the computation runtimes. We will also study the use of another parallel framework: clusters (including clusters of multi-core machines in order to benefit from both architectures).

Acknowledgements

The authors would like to acknowledge Lisa Di Jorio for providing the source code of the implementation of the GRITE algorithm [1].

References

1. Di Jorio, L., Laurent, A., Teisseire, M.: Mining frequent gradual itemsets from large databases. In: Int. Conf. on Intelligent Data Analysis, IDA'09. (2009)
2. Berzal, F., Cubero, J.C., Sanchez, D., Vila, M.A., Serrano, J.M.: An alternative approach to discover gradual dependencies. Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS) **15**(5) (2007) 559–570
3. Buehrer, G., Parthasarathy, S., Chen, Y.K.: Adaptive parallel graph mining for cmp architectures. In: ICDM. (2006) 97–106
4. Tatikonda, S., Parthasarathy, S.: Mining tree-structured data on multicore systems. In: VLDB '09: Proceedings of the 35th international conference on Very large data bases. (2009)
5. Di Jorio, L., Laurent, A., Teisseire, M.: Fast extraction of gradual association rules: A heuristic based method. In: IEEE/ACM Int. Conf. on Soft computing as Transdisciplinary Science and Technology, CSTST'08. (2008)
6. Fiot, C., Masseglia, F., Laurent, A., Teisseire, M.: Gradual trends in fuzzy sequential patterns. In: Proc. of the Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU). (2008)
7. Hüllermeier, E.: Association rules for expressing gradual dependencies. In: Proc. of the 6th European Conf. on Principles of Data Mining and Knowledge Discovery, PKDD'02, Springer-Verlag (2002) 200–211
8. Laurent, A., Lesot, M.J., Rifqi, M.: Graank: Exploiting rank correlations for extracting gradual dependencies. In: Proc. of FQAS'09. (2009)