Parallel Sequence Mining on Cycle Stealing Networks

Calum Robertson
Centre for IT Innovation,
Queensland University of Technology,
Queensland, Australia
cs.robertson@student.qut.edu.au

Shlomo Geva
Centre for IT Innovation,
Queensland University of Technology,
Queensland, Australia
s.geva@qut.edu.au

Wayne Kelly
Centre for IT Innovation,
Queensland University of Technology,
Queensland, Australia
w.kelly@qut.edu.au

Abstract

Cycle stealing networks can convert a network of computers into a virtual supercomputer. However, harnessing this power for sequence mining is not a simple process. Most parallel sequence mining algorithms suffer from high inter-process communication costs which we effectively eliminate with our data partitioning algorithm. We show that this solution, running on the G2 [1] cycle stealing network, offers substantial performance gains, finds nearly all maximal sequences, with only a small percentage of false positives.

1 INTRODUCTION

Cycle stealing applications such as SETI@HOME allow the unutilized compute power of PCs spread across the Internet to be exploited. Data mining applications such as sequence mining are computationally intensive and therefore could benefit from parallelisation in such an environment. Unfortunately, existing data mining algorithms, such as (CD, DD, IDD) [2], pSPADE [3] and (DPF, STPF) [4], require all or at least some candidates to be passed between the nodes. While this works well on a traditional supercomputer or on a dedicated cluster computer, it doesn't work well when utilising PCs spread across the Internet. This is because direct communication between fellow "volunteers" is extremely difficult if not impossible, due to the presence of firewalls and the fact that volunteers can come and go during the computation without notice.

We propose a new approach to parallelising sequence mining problems that requires no communication between fellow volunteers and therefore results in much better performance in an Internet environment. The algorithm is based on a sampling heuristic and so in general will produce only an approximation of the results obtained by an exhaustive approach, but we show that in practice only a small percentage of incorrect results are produced, which we believe is acceptable for most applications.

In this paper we will briefly define the sequence mining problem, and outline our serial algorithm, before introducing the G2 [1] grid computing environment in Section 4. We will then briefly outline our parallel algorithms, and evaluate their performance, before presenting our conclusions.

2 PROBLEM DEFINITION

The sequence mining problem introduced by Agrawal and Srikant [5] processes large transaction datasets to find frequent sequential patterns with minimum support. A sequence mining dataset contains a history for a number of customers, each of which consists of a time ordered sequence of transactions. Each transaction consists of a set of individual items. The same items can appear in multiple transactions. An itemset is a subset of the items that make up a transaction. A sequence is an ordered list of itemsets that derive from successive transactions of a given customer. A sequence is said to have p% support if it exists in at least p% of all customers. A sequence is said to be maximal if no super-sequence exists with enough support.

3 SERIAL ALGORITHM

Our aim is to divide the sequence mining problem up into manageable parts and allow each to operate independently prior to combining results. We decided to partition the dataset into several small sub-datasets and perform a complete sequence mining operation on each. This approach not only reduces the memory requirement but also reduces I/O costs as the entire dataset doesn't need to be shifted in and out of memory. We chose the Apriori [5] algorithm for processing the data, as it is a widely known algorithm with well documented performance.

3.1 Apriori

The Apriori algorithm solves the sequence mining by finding all itemsets, converting the dataset so each transaction represents the itemsets it contains, and finally finding all sequences. Itemsets and Sequences are both found by recursively generating candidates of increasing length, counting their support and then subsequently pruning all candidates that don't have support.

A candidate containing n items/itemsets is called an n-length candidate. All 2-length candidates are formed by appending 1-length candidates. As items within an itemset have to be ordered, items are appended in order, though itemsets within a sequence can be appended in any order. A n-length candidate is formed by finding two patterns

with the same (n-1)-length prefix and combining, such that (0,1,2,3) and (0,1,2,4) make (0,1,2,3,4). Agrawal and Srikant [6] propose that it is cheaper to validate that (1,2,3,4) exists than it is to count the support of (0,1,2,3,4), so we incorporated this into our implementation.

Support for a given itemset/sequence is only counted once per customer to ensure that the minimum support requirement is met.

3.2 DPA

The Data Partitioning Apriori (DPA) algorithm divides the dataset into various equally sized sub-datasets that are processed independently before all results are combined. Partitioning is constrained because the size of the dataset partitions must not be so small that even a single instance of a sequence satisfies the minimum support constraint. Therefore there is a strict limit to the number of times the dataset can be partitioned.

The DPA algorithm requires that a percentage of partitions (β) support a sequence for it to be included. This condition ensures that sequence that is prominent in only a minor partition does not appear in the overall results.

4 THE G2 CYCLE-STEALING FRAMEWORK

G2 [1] is an application framework, also developed at QUT, for creating parallel applications that exploit the idle cycles of networked PCs. The framework is designed to make it both easy to develop simple new parallel applications (such as the sequence mining application described in this paper) and easy to deploy such applications.

The G2 framework is designed for use on the new Microsoft .NET platform. This allows application programmers to choose from a number of implementation languages, while still providing a managed runtime environment in which the operations attempted by user code can be carefully vetted so as to prevent malicious or accidental damage to volunteered workstations (this is typically referred to as "sand-boxing").

4.1 The G2 Programming Model

The G2 framework presents the application programmer with a programming model directly analogous to the implementation and use of web services using ASP.NET. To implement a section of code that is to be executed in parallel, a G2 application programmer simply creates a method and annotates it with a WebMethod attribute as shown in the trivial example in Figure 1:

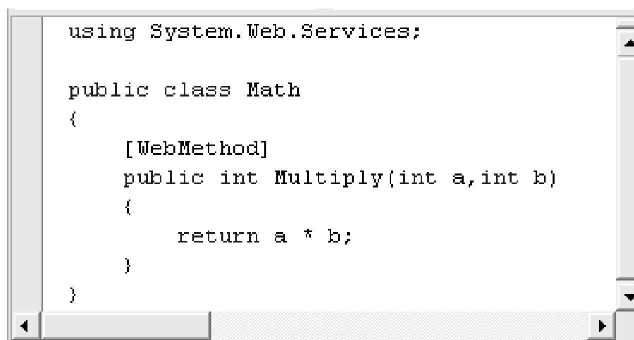

Figure 1: Sample G2 Method

This code is then used as input to a G2 utility program that generates a proxy class that allows clients to make multiple invocations of the designated Web Methods in parallel (on volunteered machines). This is exactly analogous to how proxy classes are generated in ASP.NET from WSDL specifications. Client programs simply create an instance of the automatically generated proxy class and invoke its methods (asynchronously) in exactly the same manner that they would invoke a regular local method. The methods are invoked asynchronously in order to provide the possibility for them to be evaluated in parallel. The standard .NET design pattern for asynchronous method invocation is followed, so it should again be familiar to many .NET programmers.

4.2 G2 Internal Architecture

Each G2 Web Method invocation is transparently converted by the client-side proxy class into a SOAP message, which is then transported using HTTP to a G2 server machine. There the SOAP message is stored in a job repository (a relational database) until a volunteer machine becomes available to execute it (see Figure 2).

Volunteered machines, when they would otherwise be idle, request a job from the G2 server machine, de-serialize the SOAP message, execute the method and send a SOAP encoded result back to the server. The result is then fetched by the client machine where it is used to provide a result for the original asynchronous method call.

With the wide spread use of firewalls, we assume, pessimistically, that clients and volunteers may be located on opposite sides of a firewall. We require only that the G2 Server is universally accessible by all clients and volunteers. Note that all communication is initiated by either the clients or the volunteers; the clients push jobs to the server and pull down results, while the volunteers pull down jobs and push back results. This arrangement is mandated by the assumption that the server cannot "push" data to either the clients or the volunteers.

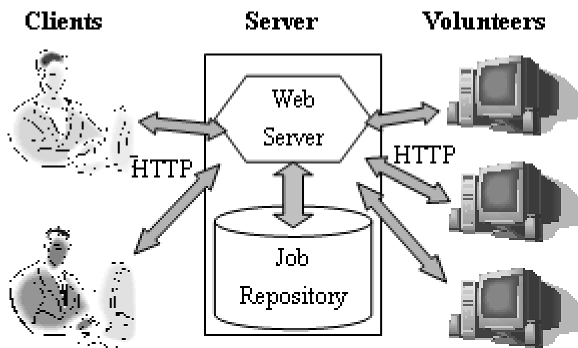

Figure 2: G2 Internal Architecture

4.3 G2 Application Deployment

Normally with ASP.NET web services, the software components (referred to as assemblies in .NET) that implement a web service need to be pre-deployed on a web server. In our case, the "Web Methods" are actually executed on a large collection of volunteer machines that are owned by various individuals that may have no particular relationship with the people wishing to run client applications. We cannot therefore hope to pre-deploy all client applications to all volunteers, so we instead adopt a lazy (and transparent) deployment strategy. All of the code, both the portion to execute on the client machine and the portion ultimately intended to execute on volunteer machines, initially resides only on the client machine. At runtime, it is dynamically uploaded to the server machine (if it is not already present) and from there dynamically downloaded to the volunteer machines and stored in their download assembly caches.

4.4 Isolated Storage

The same is true of any large data files that need to be read by the code executing on the volunteers. They are lazily uploaded to the server and then downloaded to the volunteer machines where they stored in a special part of the volunteer's local file system called isolated storage that is strictly managed by the .NET runtime environment. Application code executing on the volunteer machines is able to access files in this isolated storage, but are otherwise unable to read or write to the local file system.

4.5 Volunteer Host

All of the G2 code that executes on the volunteer, including the framework code that fetches jobs from the server is hosted within a web browser. Volunteers simply need to start their web browser and point it at a G2 Server and they are instantly volunteering. Absolutely no code needs to be pre-deployed on the volunteer machines apart from a web browser and the (free) .NET framework SDK. All G2 and application code that needs to execute on the volunteer is automatically downloaded as required inside a security context set up by the browser. If a volunteer wishes to stop volunteering, they can simply shutdown or kill the web browser process. If the volunteer was in the middle of executing a job, that job will be automatically picked up and re-executed (from the beginning) by some other volunteer – i.e. the system is intrinsically fault tolerant to volunteers disappearing without notice.

5 PARALLEL ALGORITHMS

The G2 client program for DPA sequence mining will create a number of jobs to be executed in parallel on the volunteers. Each job consists of performing the standard Apriori algorithm on a subset of the customers. Each volunteer needs therefore only part of the total dataset in order to perform its task. We investigate two alternative approaches to distributing this data to the volunteers.

The first approach involves sending to each volunteer (using isolated storage) only the part of the data (PDDPA) that it needs to complete the job it has currently been assigned. This requires the client to perform some initial processing in order to create individual files for each job, however, it minimizes the amount of data sent over the network.

The second approach involves sending to each volunteer (again using isolated storage) the entire dataset (EDDPA). This requires a lot more data to be sent over the network initially, but since this data is cached on the volunteers, subsequent mining operations on the same dataset will require no data to be sent to those volunteers. In other words, we pay a high once off cost which is then (hopefully) amortized over a number of sequence mining operations.

When a volunteer receives the job it initially needs to pre-process the data to extract the part it needs. In our implementation we first calculate all items, within the subdataset, with minimum support. We then transform the data so that each transformed transaction contains at least one supported item, and each customer contains at least one transformed transaction. This process effectively reduces the search space, as there is no need to search through items that don't have support.

In our early work with PDDPA we attempted to transfer the data to volunteers via a string SOAP variable in the method call. Whilst this process requires none of the I/O costs associated with isolated storage, it places unacceptable loads on the client, server and volunteer, due to the serialization process. The variable to be serialized must be memory resident, as must the generated XML. Deserialization normally requires the XML to be memory resident, along with the created variable. For small datasets this is not a major concern, though our 1,000,000 customer DS4

takes up 350MB in its base format. For a string variable this effectively means 700MB of data must exist in memory simultaneously, which caused problems with the 512MB machines we tested on. Therefore isolated storage is the most efficient solution for transmission of large datasets.

6 RESULTS

The results section is divided into two distinct sections. The first analyses the loss rate of the DPA algorithm, whilst the second examines the performance gains obtained on G2.

6.1 Lossy Performance

To analyse the effect of β we produced four synthetic datasets, using software provided by the IBM Quest group [5], defined in Table 2 (refer to Table 1 for parameter definitions). For all our tests we set N = 10,000, N_I = 25000, and N_S = 5000.

Table 1: Synthetic Dataset Paramters

D	Number of Customers
C	Av. Transactions per Customer
T	Av. Items per Transaction
S	Av. Potentially Maximal Sequence Length
I	Av. Potentially Maximal Itemset Length
N	Number of Items
N_S	Number of Maximal Sequences
N_I	Number of Maximal Itemsets

Table 2: Dataset Definitions

	Dataset			
Parameter	DS1	DS2	DS3	DS4
C	10	10	10	20
T	2.5	5	5	2.5
S	4	4	4	4
I	1.25	1.25	2.5	1.25

The results in Figure 3 show the percentage of maximal sequences found for 100,000 customer datasets partitioned 100 times at 1% support. This is close to the limit for partitioning yet it highlights that requiring too many sub-datasets to support the sequence reduces the number of maximal sequences found.

On the other hand Figure 4 shows that as β is reduced, more false positives are found. This in turn reduces the number of maximal sequences found, as false positives tend to be supersets of actual maximal sequences. Therefore, it seems that a value of β=60% is near optimal.

We tried reducing the minimum support requirement for each partition, in an attempt to find more sequences, but found that too many false positive results were found.

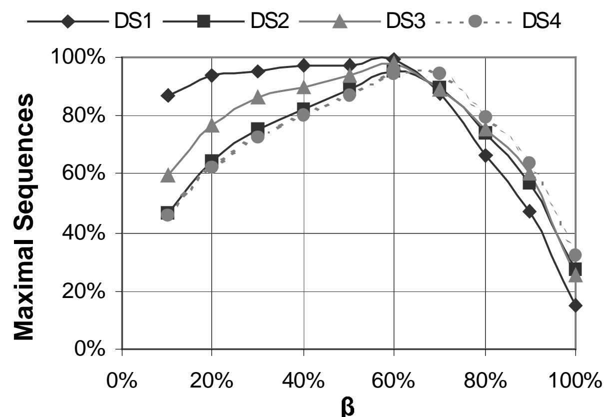

Figure 3: Results Vs β

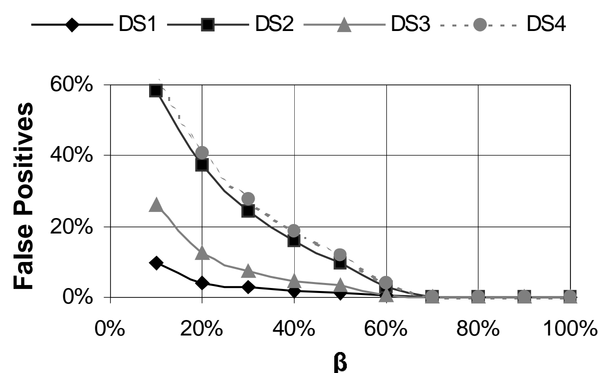

Figure 4: False Positives Vs β

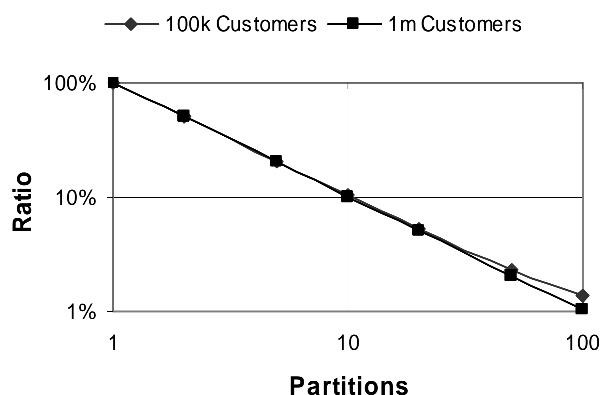

Figure 5: Candidates Checked Vs Partitions

To analyse the benefits of the DPA algorithm, we ran it on the four datasets with both 100,000 and 1,000,000 cus-

tomers at 1% support. We measured the number of candidates generated, and the number of candidates that were counted for support (the two major costs in the Apriori algorithm). The results in Figure 5 show that for more partitions there is a linear reduction in the number of candidates checked for support. This is due to the linear decrease in customers.

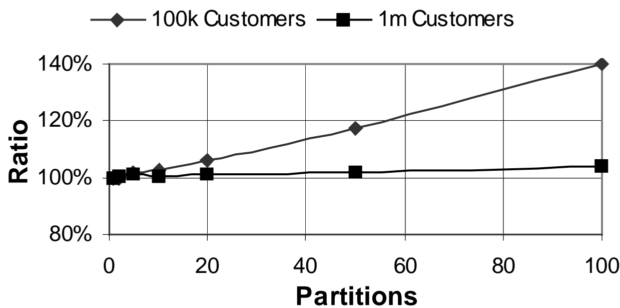

Figure 6: Candidates Vs Partitions

Figure 6 shows that the number of candidates generated starts to increase as the data is partitioned more. The effect on the 100,000 customer dataset is very prominent because at 100 partitions the 1% support requirement means that only 10 customers must contain the candidate for it to be significant. This emphasises our earlier statement that there is a strict limit to the number of times the dataset can be partitioned.

6.2 Parallel Performance

These algorithms were written in Microsoft C# .NET using the Microsoft .NET 1.1 Framework. All tests were performed on 800MHz Intel Pentium 3 PCs with 512MB of memory. A 100Mb/s switching network was used, though not exclusively for these tests.

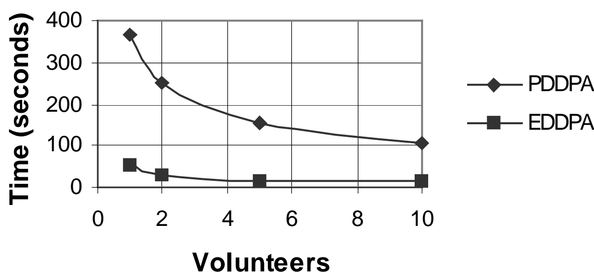

Figure 7: EDDPA Vs PDDPA

The process times in Figure 7, for the 100,000 customer DS1 at 1% support, demonstrates that, when data has been pre-deployed, EDDPA is a lot cheaper than PDDPA. However PDDPA speeds up reasonably well, so if only one sequence mining operation needs to be performed, PDDPA is preferable to EDDPA. Note that by speedup we refer to the time taken to execute the code on a single volunteer versus multiple volunteers.

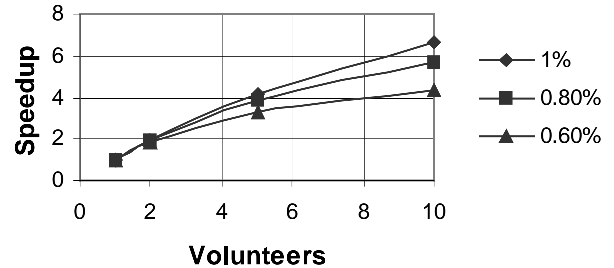

Figure 8: Mean EDDPA 100k customer Speedup

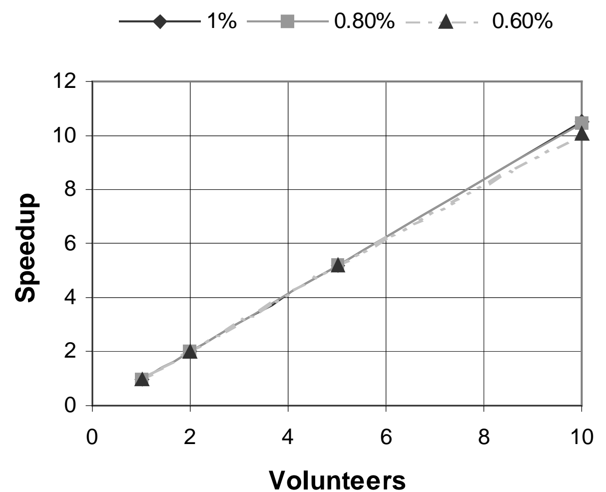

Figure 9: DS1 EDDPA 1m customer Speedup

The results in Figure 8 show the mean speedup for all datasets with 100,000 customers, whilst Figure 9 shows the speedup for the 1,000,000 customer DS1, with varying supports. They demonstrate that decreasing the support, also decreases the speedup, though the effect is less significant for larger datasets.

To analyse this effect we recorded the Fixed (candidate generation, and pruning) and Dynamic (pre-processing, transmission, support counting, and dataset conversion) costs for each volunteer. The costs in Figure 10 show that fixed costs become more significant, as the number of partitions increase. The results in Figure 5 highlight that the support count cost reduces proportionally to the number of partitions, and obviously the other dynamic costs reduce with more partitions, as there is less data. Also results in Figure 6 demonstrate that the candidate generation cost remains relatively constant.

Therefore these results indicate that the performance of parallel DPA algorithms are limited by the ratio of fixed to dynamic costs. If the fixed costs are equivalent to the dynamic costs at a 100,000 customer partition, then there is no benefit in creating smaller partitions.

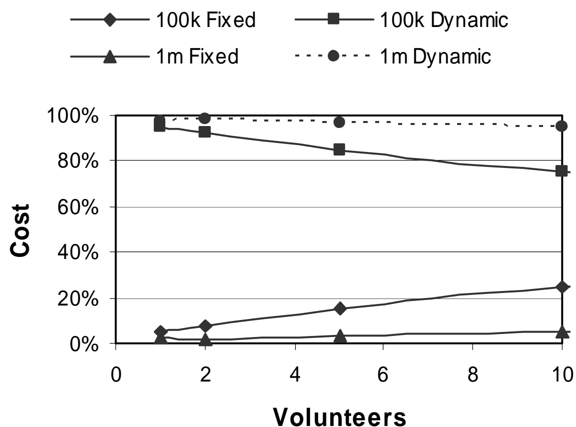

Figure 10: DS1 EDDPA at 1% Costs Per Volunteer

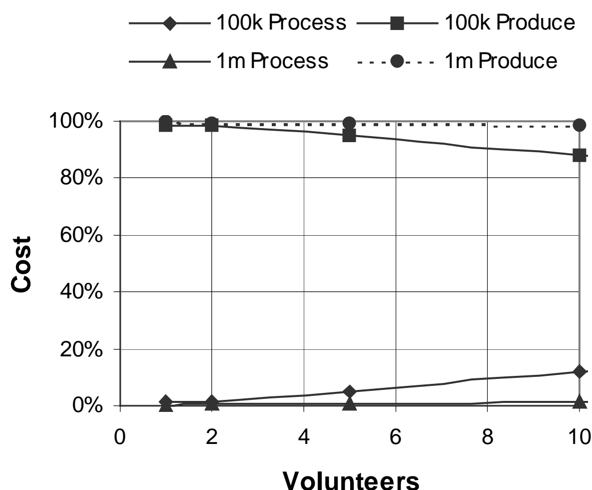

Figure 11: DS1 EDDPA at 1% Process Vs Produce

However, to verify this effect, we measured the cost of result processing versus production. Result production costs are the costs associated with producing all the jobs, and having the volunteers process them all. Result process costs are the costs associated with retrieving the volunteers' results, combining them and finally eliminating all but those with β% support amongst the partitions. The results shown in Figure 11 indicate that process costs start to increase with more partitions. This is due to an increase in the number of results, as each partition, no matter the size, will produce a similar number of results to that produced by processing the whole dataset. Of course Figure 6 shows that the more times the data is partitioned, the more candidates are generated, and therefore there is a higher chance of there being more results.

Interestingly the result process versus result production costs is not nearly as severe as the volunteer fixed versus dynamic costs. This effect can be attributed to the asynchronous nature of the G2 environment, where jobs are not created simultaneously, and therefore results are not submitted simultaneously. With this in mind it is possible to predict when an individual job will be inefficient, though is it is not as easy to predict when the whole process will become inefficient.

7 CONCLUSIONS

We have shown that the DPA algorithm can find the majority of maximal results, with a very small percentage of false positives. Our results show that substantial performance gains can be achieved for large datasets for EDDPA, though gains are limited by the complexity of the dataset. We have also shown that the PDDPA algorithm offers good gains for situations where EDDPA is not preferable. Though we used the Apriori algorithm, other algorithms could provide better gains, which we hope to establish in future work.

The G2 framework, outlined in this paper, enables the power of local intranets, and the internet, to be harnessed for parallel computing. Its application to the sequence mining problem has shown its benefits, and in future we will demonstrate its effect on other problems.

8 REFERENCES

[1] Kelly, W.A., P. Roe, and J. Sumitomo, *G2: A Grid Middleware for Cycle Donation using .NET.* In Proceedings of *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas. June 2002.

[2] Joshi, M.V., E. Han, K. G., and V. Kumar, *Efficient Parallel Algorithms for Mining Associations.* Lecture Notes in Artificial Intelligence, 2000: p. 83-126.

[3] Zaki, M., *Parallel Sequence Mining on Shared-Memory Machines.* Journal of Parallel and Distributed Computing, 2001(61): p. 401-426.

[4] Guralnik, V. and G. Karypis, *Parallel Formulations of Tree-Projection-Based Sequence Mining Algorithm.* Technical Report 03-0003, University of Minnesota, Minneapolis, 2003.

[5] Agrawal, R. and R. Srikant, *Mining Sequential Patterns.* In Proceedings of *11th International Conference on Data Engineering*, Taipei, Taiwan. 1995: p. 3-14.

[6] Agrawal, R. and R. Srikant, *Mining Sequential Patterns: Generalizations and Performance Improvements.* In Proceedings of *5th International Conference on Extending Database Technology*, Avignon, France. 1996. Springer-Verlag: p. 3-17.