Sequential Pattern Mining by Multi- Threading Techniques

1Reza Noorian Talouki and 2Mehdi Sobhkhiz Talouki
1Department of Computer Engineering, Science and Research Branch, Islamic Azad University,
Tehran, Iran
2Sama Technical and Vocational Training College, Islamic Azad University,
Sari Branch, Sari, Iran

Abstract: Discovery of Sequential pattern mining is an important data mining mission with wide applications. One of the most important types of sequential patterns is closed sequential pattern, which holds all the information of the perfect patterns set but is much more compact than it. There is no model that used multi-threading techniques for parallel mining of closed sequential patterns. In this paper an algorithm called MTMCSP (multi-thread mining of closed sequential patterns) is recommended to conduct parallel mining of closed sequential patterns on a multi-processor system as a multi-threading technique. MTMCSP divides the works among the tasks by using the divide-and-conquer property. The proposed algorithm has used dynamic scheduling to avoid task idling, moreover we have employed a technique, called random selecting. The experimental results show that MTMCSP attains good parallelization efficiencies on various input datasets.

Keywords: Multi-threading, sequential patterns, thread scheduling

INTRODUCTION

The objective of sequential pattern mining is to discover frequent subsequences in a dataset. Sequential pattern mining has multiple applications, inspection of scientific or curative processes and analysis of web log registered acts. Several sequential pattern mining algorithms have been proposed so far (Agrawal and Srikant, 1995; De Amo et al., 2008; Pei et al., 2007; Wang et al., 2004).

Since a long sequence includes a combinatorial number of subsequences, sequential pattern mining creates an explosive number of frequent subsequences for long patterns, which is preventively costly in both time and space. Hence, instead of mining the whole set of sequential patterns, an alternative but equally powerful explanation is to mine closed sequential patterns only (Han and Kamber, 2006; Srikant and Agrawal, 1996; Yan et al., 2003).

A closed sequential pattern is a sequential pattern which has no super-sequence with the same incident frequency. Some algorithms have been recommended for mining closed sequential patterns usually work in the following two manners:

Some of these algorithms pursue a candidate maintenance-and-test example among the set of pattern candidates. These set are used to lessen the search space and verify if a recently found sequential pattern is probably to be closed. These algorithms tend to be percussively costly for mining long sequences and mining with very weak support thresholds. Clospan algorithms a kind of these algorithms (Yan et al., 2003).

The second group of algorithm (BIDE) pursues a closure checking scheme, called BI-Directional Extension, which mines closed sequential patterns without candidate maintenance (Afshar and Han, 2002; Wang and Han, 2004). Performance studies have shown that the 2nd kind algorithms are more efficient than the 1st group. To make sequential pattern mining applicable for large data sets, the mining processes must be efficient, scalable and have a short reaction time. Moreover, since sequential pattern mining requires repetitive searches of the sequence dataset with multiple data comparison and analysis operations and needs lots of computations. Furthermore, many applications are time-critical and entangle huge volumes of data. Such applications request more mining power than serial algorithms.

Thus, as we mentioned it is clearly important to study closed sequential patterns. Although an important amount of studies have been done on sequential pattern mining, there is still much room for improvement in its parallel implementation. Parallel implementation of the sequential pattern mining as a multi-processor system tends to be expensive. In this study, a suitable algorithm is proposed for parallel closed sequential pattern mining which is more efficient in hardware cost. To the best of our
knowledge, there is no parallel algorithm that targets closed sequential pattern mining as multi-threading.

**LITERATURE REVIEW**

**Problem definition:** Let $I = \{a_1, a_2, ..., a_n\}$ be a set of items. A sequence $s$ is a set of variables, represented as $<B_1, B_2, ..., B_l>$, where $Bc(1 \leq c \leq l)$, called events, or item sets. Each event is a set represented as $(y_1, y_2, ..., y_m)$ where $yk(1 \leq k \leq n)$ is an item. For briefness, the brackets are omitted if an element has only one item. A sequence dataset $S$ is a set of sequences. The total number of items in a sequence is called the length of the sequence and a sequence with length $l$ is called an $l$-sequence. A sequence $\beta = <b_1, b_2..b_n>$ is called a subsequence of another sequence $\mu = <m_1, m_2...m_n>$, represented as $\beta \preceq \mu$, if there exist integers $1 \leq c_1 \leq ... \leq c_l \leq n$ such that $b_1 \preceq m_{c_1}, b_2 \preceq m_{c_2},..., b_n \preceq m_{c_n}$. If $\alpha$ is a subsequence of $\mu$, we utter that $\mu$ includes $\beta$. The support of a sequence $\beta$ in a sequence dataset $S$, presented support ($\beta$), is the number of sequences in the dataset containing $\beta$ given a minimum support threshold, min sup, the a group of sequential pattern, SP, is the group of all the subsequences whose support values are no less than minsup. The set of closed sequential patterns, CSP is explained as $\text{CSP} = \{\beta | \beta \in \text{SP} \text{ and } \forall \mu \in \text{SP} \text{ such that } \beta \preceq \mu \text{ and support}(\beta) = \text{support}(\mu)\}$. The problem of closed sequential pattern mining is to find CSP with support value no less than a minimum support threshold (Dong and Pei, 2002; Han and Kamber, 2007).

<table>
<thead>
<tr>
<th>Seq id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MFKALRTIPVILNMKDSKLCPN</td>
</tr>
<tr>
<td>2</td>
<td>MSPNPTNHTGKTLR</td>
</tr>
</tbody>
</table>

**MTMCSP ALGORITHM**

In this section, we introduce an algorithm called MTMCSP to mine the closed sequential-patterns in parallel. We address the following questions: How to decompose algorithm into tasks? How to schedule the results of thread? How to balance the load?

**Task decomposition:** Algorithm follows three steps:

**Stage 1:** Recognize the frequent 1-sequences

**Stage 2:** Suggest the dataset among each frequent 1-sequence

**Stage 3:** Search the result of projected datasets

The projected datasets of the frequent 1-sequences are independent. Consider a 1-sequence, say $j$, only the suffixes that pursue the first incidents of $j$ in each sequence are the projection of the dataset along $a$. So, the closed sequential-patterns mined from the dataset projection along $j_1$ all begin with $j_1$ as the prefix while the samples found from $j_2$’s projections all start with $j_2$. A partition strategy like the one just described is convenient.

![Fig. 1: Extraction of frequent pattern for projection N](image-url)
for task decomposition. Since the projected datasets are independent, they can be assigned to different threads. Then, each thread can mine the assigned projected datasets independently by using the algorithm. No interprocessor communication is needed during the local mining. Work procedure of the algorithm was shown in Fig. 1, using the dataset given in Table 1.

Our strategy for parallel mining of closed sequential patterns is as follows:

Each thread counts the occurrence of 1-sequences in a different part of the dataset. A function called com is executed to obtain the occurrence overall counts. The frequent 1-sequences, those that occur at least min sup (the support threshold) times, are identified.

For each frequent 1-sequence a very compact representation of the dataset projections, called pseudo-projections, is built. This is done in parallel by assigning different part of the dataset to each thread. In fact preparation of pseudo-projections from purposed data set and abandon of data set to each thread both done in the same time. Dynamic scheduler distributes the projection across the thread for processing.

We assume that the complete dataset is accessible to all threads. In the second step, each thread applies the pseudo-projection or purposed pseudodata set to construct the purposed datasets. A pseudo-projection consists of a set of pointers to the starting positions within the dataset of each sequence conforming the pseudo projected dataset. After constructing the pseudo-projections, they are broadcast to all threads. In our implementation, we found that it is more efficient to carry out the broadcast using a virtual structure because this act consumes no more than 0.6% of the mining time for send and receive.

Threads scheduling: Next, we discuss the mechanism that we use to assign projections to threads. To reduce load imbalance threads, MTMCSP uses dynamic scheduling. In our implementation, there is a master thread which maintains a queue of pseudo-projection identifiers. Each thread is initially assigned a projection. After a thread completes the mining of a projection, it sends a request to the master thread for another projection. The master thread replies with the index of the next projection in the queue and removes it from the queue. This process continues until the queue of projections becomes empty. The requests and replies are little messages and need usually little time relative to the mining time. Dynamic scheduling is quite effective when the sub-threads are of similar size and the numbers of threads are equal with the number of processors.

For the datasets we used in our tests, the cost of mining the projections may growth. The relatively voluminous mining time of some projected datasets may result in many static workloads.

Relative mining time estimation: Our approach to improving the efficiency of the dynamic scheduling is to recognize which projections require long mining time and to further decompose them. For this aim, we need to appraise the relative mining time of the projections. Our method to appraise mining time is to use runtime sampling. By mining a small sample of the primary dataset and timing the mining time of the projected databases of the sample, we should be able to recognize the projections whose mining time is longer. We appraise sampling strategies by the exactness of their estimation and the overhead they announce. The most natural sampling method is random sampling which progresses by gathering a randomly selected subset of the sequences in the dataset, reckons the projections and uses the mining time of this subset to appraise the mining time of each projection (Cong et al., 2004). Instead of randomly selecting a subset of sequences from the dataset, selective sampling potentially uses ingredients of every sequence in the dataset. Selective sampling first discards all infrequent 1-sequences and then discards the 1 frequent

Fig. 2: Selective sampling(Cong et al., 2004)
The part of the dataset allotted to it. Its first significant operation counts the 1-sequences for sequential patterns. In MTMCSP algorithm each thread in MTMCSP, the parallel algorithm to mine closed sequences without meaningfully affecting the relative mining time of their projections will not be time consuming and therefore, we can safely omit these 1-sequences. Variable S is allotted these relative mining times. After the sampling, the top time-consuming projections are divided into smaller ones. For example, if the projection along A is one of the top time consuming projections, it will be divided into the projections along zy, zx and so on. Then the master thread schedules these projections as subtasks by preserving a task queue. The projections appraised to take longer time in sampling are to be scheduled earlier. Those very small projections can be scheduled in pieces to prevent communication contention. Thread 0 is treated as the master thread and taking charge of the task scheduling while the other threads (slave threads) mine the allotted projections independently without communication to each other. Whenever a slave thread finishes the allotted subtask, it sends a request to the master thread for another one, until the task queue is vacant. Each slave thread outputs the closed sequential patterns in a file. The final closed sequential patterns are the concatenation of these files.

**EXPERIMENTAL RESULTS**

**Experimental setup:** Our performance study includes both artificial and actual datasets. We used three artificial datasets generated by the Gen Bank dataset available in NCBI and an actual dataset zinc finger, which comes from protein sequence set and provided by NCBI. In Gen Bank we consider distinctive products as distinctive items and the sequence views as events. Also we assume all of the consecutive sequences of DNAs as a protein sequences. The characteristics of these datasets are shown in Table 2.

All of our experiments were performed on a AMD 4-core CPU, i.e., AMDphenomX4, with 4GB memory.

**Experimental results:** We first inspect the parallel performance of the MTMCSP algorithm. Figure 3 shows the total execution time and the speedup for each dataset. Execution time is measured in seconds throughout this process of mining the selective sample. But in place of producing the closed sequential patterns, it records the mining time for the projections of all frequent 1-sequences. Variable S is allotted these relative mining times. After the sampling, the top time-consuming projections are divided into smaller ones. For example, if the projection along A is one of the top time consuming projections, it will be divided into the projections along zy, zx and so on. Then the master thread schedules these projections as subtasks by preserving a task queue. The projections appraised to take longer time in sampling are to be scheduled earlier. Those very small projections can be scheduled in pieces to prevent communication contention. Thread 0 is treated as the master thread and taking charge of the task scheduling while the other threads (slave threads) mine the allotted projections independently without communication to each other. Whenever a slave thread finishes the allotted subtask, it sends a request to the master thread for another one, until the task queue is vacant. Each slave thread outputs the closed sequential patterns in a file. The final closed sequential patterns are the concatenation of these files.

In our implementation, we perform the mining of the dataset resulting from selective sampling in parallel following the same method that we apply to the complete dataset. As you may expect, there is a trade-off between the exactness and the overhead of selective sampling. The more frequent 1-sequences we discard, the less exactness selective sampling will be and the less processing will be announced by sampling. When building the projection along a frequent 1-sequence, only the suffixes (with the 1-sequence as prefix) will be gathered. The frequent 1-sequences in the tail of a sequence will appear in every projection of their prefixes. Thus, by omitting these frequent 1-sequences, the sequences in most of the projections become shorter and so, the mining time can be greatly decreased compared to the mining time of the primary dataset. At the same time, the suffixes of the frequent 1-sequences in the tails are shorter so that the mining time of their projections will not be time consuming and therefore, we can safely omit these 1-sequences without meaningfully affecting the relative times.

The **MTMCSP algorithm:** In this subsection, we explain MTMCSP, the parallel algorithm to mine closed sequential patterns. In MTMCSP algorithm each thread in its first significant operation counts the 1-sequences for the part of the dataset allotted to it.
We ran the old sequential algorithm and MTMCSP using some thread. As the charts show, MTMCSP achieves fairly good performance for all the tested datasets. MTMCSP substantially decreases the mining time comparing to the sequential algorithm. Datasets sequence_1 and sequence_3 whose sequential mining times are larger achieve better speedups than Kringle and zinc finger. Another factor that constraints the speedups is load imbalance. The mining time of some tasks are so large that the subtasks derived from them are still much bigger than the small tasks. The answer of this problem is multi-level task piece.

The selective sampling technique can be extended to complete this multi-level partitioning. Next, we examine the influence of changing minimum support threshold on the performance of MTMCSP. The results are shown in Fig. 4, in which we use the dataset sequence_1 with the minimum support threshold changing from a high of 0.08% to a low of 0.005%. We tested MTMCSP on different threads and compared the performance with sequential algorithm. MTMCSP shows steady parallel performance with distinctive support threshold. Similar results can be achieved for the other datasets. Our experiments show efficiency of the MTMCSP, based on the number of changing of threads. The best efficiency achieved when the number of threads and processors be equal, when the number of threads allotted to each processor is large enough so that it tends to balance the load and lessen the efficiency of processors. We also compare MTMCSP and prefix span (Pei et al., 2001).

**CONCLUSION**

In this study, we suggest a parallel closed sequential pattern mining algorithm MTMCSP. It is the first parallel paradigm for the sequence pattern mining problem. We apply multi-threading for patterns mining in which the divide-and-conquer feature is used to minimize the inter-processor communications and task assignment is performed by dynamic scheduling.
REFERENCES

Cong, S., J. Han and D. Padua, Year. Parallel mining of closed sequential patterns. Parallel Comput., 30(4): 443-472.