การหารูปแบบที่มีความถี่สม่ำเสมอเคอันดับแรกจากฐานข้อมูลรายการ

นายโกเมศ อัมพวัน

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2553

MINING TOP-K REGULAR-FREQUENT ITEMSETS FROM TRANSACTIONAL DATABASE

Mr. Komate Amphawan

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2010

Thesis Title	MINING TOP-K REGULAR-FREQUENT ITEMSETS FROM TRANSACTIONAL DATABASE
By	Mr. Komate Amphawan
Field of Study	Computer Engineering
Thesis Advisor	Assistant Professor Athasit Surarerks, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

..... Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

..... Chairman
(Associate Professor Wanchai Rivepiboon, Ph.D.)

..... Thesis Advisor
(Assistant Professor Athasit Surarerks, Ph.D.)

..... Examiner
(Assistant Professor Pizzanu Kanongchaiyos, Ph.D.)

..... External Examiner
(Assistant Professor Krisana Chinnasarn, Ph.D.)

..... External Examiner
(Assistant Professor Arnon Rungsawang, Ph.D.)

โกเมศ อัมพวัน: การหารูปแบบที่มีความถี่สม่ำเสมอเคอันดับแรกจากฐานข้อมูลรายการ. (MINING TOP-K REGULAR-FREQUENT ITEMSETS FROM TRANSACTIONAL DATABASE) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผู้ช่วยศาสตราจารย์ อรรถสิทธิ์ สุรฤกษ์, 197 หน้า.

การหากฎความสัมพันธ์ภายใต้ค่าสนับสนุนและค่าความเชื่อมั่นที่ผู้ใช้ระบุเป็นงานสำคัญของกลุ่มวิจัยด้านการทำเหมืองข้อมูล อย่างไรก็ตามความถี่ของการเกิดข้อมูลในรูปแบบต่างๆอาจไม่เพียงพอที่จะเป็นเงื่อนไขของการหารูปแบบที่มีความหมาย ดังนั้นลักษณะการเกิดของรูปแบบที่มีความสัมพันธ์กันจึงเป็นประเด็นสำคัญของงานในหลายๆด้าน รูปแบบหนึ่งของการเกิดความสัมพันธ์ของข้อมูลก็คือการเกิดขึ้นอย่างสม่ำเสมอ นั่นคือข้อมูลจะปรากฏขึ้นห่างกันเป็นช่วงๆ ในระยะห่างที่ผู้ใช้ให้ความสำคัญ ซึ่งในการหาข้อมูลที่มีการเกิดขึ้นในลักษณะดังกล่าวจะต้องมีการกำหนดค่าสนับสนุนเพื่อใช้ในการเลือกกรองความสม่ำเสมอของข้อมูลที่เกิดขึ้น แต่ในทางปฏิบัติแล้วการกำหนดค่าขีดแบ่งที่เหมาะสมนี้ทำได้ยาก เนื่องจากถ้ากำหนดค่าขีดแบ่งนี้น้อยเกินไปจำนวนผลลัพธ์ที่ได้ก็จะมีปริมาณมากและในทางกลับกันถ้ากำหนดค่าขีดแบ่งมากผลลัพธ์ที่ได้ก็จะมีจำนวนน้อยหรืออาจจะไม่พบคำตอบที่สอดคล้องกับค่าขีดแบ่งนี้ ดังนั้นแทนที่จะกำหนดค่าขีดแบ่งนี้การกำหนดจำนวนผลลัพธ์ที่ต้องการจะเป็นการเหมาะสมกว่าจากการสำรวจงานวิจัยที่มีอยู่ในปัจจุบันพบว่าไม่มีวิธีใดที่ให้ผู้ใช้ระบุจำนวนการเกิดข้อมูลแบบสม่ำเสมอที่ผู้ใช้ต้องการ ดังนั้นวิทยานิพนธ์นี้จึงนำเสนอการใช้จำนวนผลลัพธ์เป็นเป้าหมายในการค้นหาข้อมูลที่มีการเกิดอย่างสม่ำเสมอ นั่นคือการหาผลลัพธ์ที่มีค่าสนับสนุนสูงที่สุดเคอันดับแรก ซึ่งวิธีที่วิทยานิพนธ์นี้นำเสนอมีดังนี้ (1) วิธีการหาคำตอบโดยอ่านข้อมูลจากฐานข้อมูลเพียงครั้งเดียว (2) การใช้การค้นหาแบบเลือกทางดีที่สุดเพื่อลดปริภูมิการค้นหา (3) การแบ่งข้อมูลและการประมาณค่าสนับสนุนเพื่อช่วยลดการคำนวณที่ไม่จำเป็นลง (4) การใช้วิธีการแทนข้อมูลแบบใหม่เพื่อลดเวลาและหน่วยความจำที่ใช้ในการประมวลผล ซึ่งจากการวิเคราะห์ประสิทธิภาพในเชิงเวลาและหน่วยความจำของวิธีการที่นำเสนอพบว่า ไม่ว่าจะกำหนดจำนวนของผลลัพธ์มากหรือน้อย ไม่ว่าข้อมูลจะมีการกระจุกหรือกระจายตัว ขั้นตอนวิธีที่นำเสนอนี้ก็สามารถให้ผลลัพธ์ได้อย่างมีประสิทธิภาพ

ภาควิชา .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต .....

สาขาวิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก ....

ปีการศึกษา ..... 2553 .....

4871857021: MAJOR COMPUTER ENGINEERING
KEYWORDS: DATA MINING / ASSOCIATION RULES / TOP-*K* SIGNIFICANT ITEMSETS / REGULAR-FREQUENT ITEMSETS / TOP-*K* REGULAR-FREQUENT ITEMSETS

KOMATE AMPHAWAN : MINING TOP-K REGULAR-FREQUENT ITEMSETS FROM TRANSACTIONAL DATABASE. THESIS ADVISOR : Assistant Professor Athasit Surarerks, Ph.D., 197 pp.

Association rule based on support-confident framework is an important task in data mining community. However, the occurrence frequency of a pattern may not be sufficient criterion for mining meaningful patterns. The occurrence behavior can be revealed as an important key in several applications. A pattern is a regular pattern if it regularly occurs in a user-given period (regularity threshold). To mine regular itemsets, a support threshold is used to filter some regular itemsets. However, in practice, it is often difficult for users to provide an appropriate support threshold. Indeed, a too small support threshold could yield a number of regular-frequent itemsets impractically large while a too large threshold could yield very few or no regular-frequent itemsets. Therefore, the use of a support threshold tends to produce a large number of regular-frequent itemsets and it could be better to ask for the number of desired results.

Currently, from the deep survey, there is no existing approach permitting users to specify the number of regular-frequent itemsets to be mined. Therefore, a new approach allowing the users to control the number of results (*i.e.* k regular itemsets with the highest supports) is presented. There are several techniques proposed in this dissertation to mine this kind of itemsets: (*i*) a single-scan approach, (*ii*) a best-first search strategy used to prune the search space, (*iii*) the partitioning and estimation techniques assisting in reducing unnecessary computational costs, and (*iv*) a new concise representation helping to save runtime and memory. From the performance study, the proposed approaches are efficient and scalable in terms of time and memory for small and large values of desired results on sparse and dense datasets.

Department: ..... Computer Engineering ..... Student's Signature .....
Field of Study: .... Computer Engineering .... Advisor's Signature .....
Academic Year: .....2010.....

Acknowledgements

First and foremost, I wish to thank the Higher Education Commission of Thailand who gave me a research scholarship. I would like to express my deep gratitude to my supervisor, Athasit Surarerks, Ph.D., to whom with his guidance, advices and helps me to overcome the difficulties of the process of conducting this dissertation. He patiently and thoroughly guided me for an almost 5-year period from a starting point to reading the chapters of this thesis. I also would like to express my deepest gratitude to my co-supervisor, Philippe Lenca, Ph.D. at Telecom Bretagne, France. His invaluable advice and patience allowed me to undergo and finish the challenging process of my doctoral degree program. I also express my thankfulness to my dissertation committees: Wanchai Rivepiboon, Ph.D., Pizzanu Kanongchaiyos, Ph.D., Krisana Chinnasan, Ph.D. and Arnon Rungsawang, Ph.D., for their advices and guidance to help me focus on my research activities. I am also grateful to Sunisa Rimchareon, Ph.D., and Miss Warisa Sritriratanarak whom always encourage me to do research, give me moral support and help me everything as they can do.

I also would like to thank all of my lovely friends (e.g. Pai, Phae, Te, Yring, Rin, Chan, Aui, Krit, Tuk, Kai, Kik, Camp, Oat, Zeng, Wat, Tar, Trung, Kwan, Pum, etc.) and the graduate students in Computer Engineering department (e.g. P' Chai, Koh, Pook, Pla, Woot, Wut, P' Tom, P' Chang, P' Jim, P' Phueng, P' Chit, P'Vic, Tei, Pae, Pun, Dong, Kai, Puh, Pair, Na, Pong, Dear, Oat(Elite), Oat(CG), Petch(Van), Petch(Dek), P'Keng, Tair, Chris, Tuy, Jumbo, Noot, P' Ae, P' Aoe, Matt, P' Mao,P'Aui P' Sakorn, P' Yui, Petch(Mhee), P' Dae, P' Jung, J' Nhan, P'Fu, June, Pia, Vit, etc.) whose advice and friendship gave me a lot of encouragement for my studying at Chulalongkorn University.

I would like to thank the Department of Computer Engineering of Chulalongkorn University for giving me the opportunity to pursue a Ph.D. degree in computer engineering. I am also thankful to the support staffs in Computer Engineering Department for always being helpful.

Last but not least, I am very grateful to my family: Father, Mother and my Sister for their love, patience, continuous moral support and encouragement. Without all of these I could not have accomplished my doctoral degree.

Contents

Chapter Page

Chapter Page

List of Tables

List of Figures

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Data mining, also known as Knowledge Discovery in Databases (KDD), is concerned with the extraction of previously unrecognized and interesting information contained within (usually large) data repositories. The interesting is of course a subjective concept, and a definition of what is interesting is required: it is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity (Fayyad et al., 1996). Almost always, what is being sought is some relationship which can be observed between categories of information in the data. A particular way to describe such a relationship is in the form of an association rule which relates attributes within the database.

The problem of mining association rules has been defined by Agrawal (Agrawal et al., 1993) as first proposed for market basket analysis in the form of association rule mining. It analyzes customers buying habits by finding associations between the different items that customers place in their "shopping baskets". For instance, if customers are buying milk, how likely are they going to also buy yogurt (and what kind of yogurt) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and arrange their shelf space. Association mining applications have been applied to many different domains including market basket and risk analysis in commercial environments, epidemiology, clinical medicine, fluid dynamics, astrophysics, crime prevention, and counter-terrorismall areas in which the relationship between objects can provide useful knowledge.

The process of mining association rules consists of two steps: (i) Find the frequent itemsets that have minimum support; (ii) Use the frequent itemsets to generate association rules that meet the confidence threshold. Among these two steps, step (i) is the most expensive since the number of itemsets grows exponentially with the number of items. Consequently, the task of frequent itemsets discovery (also called frequent patterns discovery) is widely studied in data mining as a mean of generating association rules (Agrawal et al., 1993), correlations (Brin et al., 1997a), sequential patterns (Agrawal and Srikant, 1995), emerging patterns (Dong and Li, 1999), dense regular patterns (Engler, 2008), frequent patterns with maximum length (Hu et al., 2008), frequent patterns with temporal dependencies (Tatavarty et al., 2007), negative rules (Wu et al., 2004), causality (Silverstein et al., 2000), weighted pattern mining (Tao et al., 2003)(Yun and Leggett, 2006) and classification rules (Li et al., 2001) (Liu et al., 1998).

Over the past decade a large number of research works have been published presenting new algorithms or improvements on existing algorithms to solve the frequent pattern mining problem more efficiently through the refinement of search strategies (depth first/breath first search (Agrawal and Srikant, 1994), top down/bottom up traversals (Grahne and Zhu, 2005)), pruning techniques, data structures (trees/other data structures (Han et al., 2004)), the use of alternative dataset organizations (vertical/horizontal formats (Zaki and Gouda, 2003)) and constraints ((Bonchi and Lucchese, 2005) (Pei et al., 2001a)). Recent surveys may be found in (Goethals, 2005) and (Han et al., 2007). However, two main bottlenecks exist: (i) A huge number of patterns are generated and (ii) Most of them are redundant or uninteresting. To tackle these problems, various approaches have been developed.

Frequent-closed pattern mining algorithms have been proposed to reduce redundant patterns (Pasquier et al., 1999) and to mine a compact set of frequent patterns which cover all frequent patterns (Pei et al., 2000). When a data set is dense, the number of frequent closed patterns extracted can be orders of magnitude fewer than the number of corresponding frequent patterns since they implicitly benefit from data correlations. Nevertheless, they concisely represent exactly the same knowledge. From closed patterns, it is in fact trivial to generate all the frequent patterns along with their supports. More importantly, association rules extracted from closed patterns have been proven to be more meaningful for analysts, because all redundancies are discarded (Zaki, 2004). A recent survey may be found in (Yahia et al., 2006).

While the previous approaches work at the algorithmic level, another strategy is to rank patterns in a post-algorithmic phase with objective measures of interest (Hilderman and Hamilton, 2000). A large number of interestingness measures have been proposed such as mining frequent patterns with tougher constraints (Bonchi and Lucchese, 2005), mining dense regular patterns (Engler, 2008), mining frequent patterns with maximum length (Hu et al., 2008), mining frequent patterns with convertible constraints (Pei et al., 2001a), mining frequent patterns with constraints using pattern growth approach (Pei and Han, 2002), mining temporal dependencies between frequent patterns (Tatavarty et al., 2007), integrating classification and association rule mining (Li et al., 2001)(Liu et al., 1998), etc. Interesting surveys and comparisons may be found in (Geng and Hamilton, 2006)(Lenca et al., 2008) and (Suzuki, 2008).

On constraint-based patterns mining, pushing the constraints using objective measures deeply into the patterns mining process is a very interesting approach (Bonchi and Lucchese, 2005) (Pei et al., 2001a). This approach uses efficient pruning strategies to discover interesting patterns such as optimal rule mining (Li, 2006)(Le Bras et al., 2009). It is important to notice that most of the previous mentioned works, except mainly (Li, 2006) and (Le Bras et al., 2009), are

always subject to the dictatorship of support for the frequent pattern mining step. Avoiding the use of support has been recognized as a major challenge, such as mining high confidence association without support pruning (Cohen et al., 2001), (Bhattacharyya and Bhattacharyya, 2007), (Le Bras et al., 2010), and mining rules without support threshold (Li et al., 1999)(Koh, 2008).

However, without specific knowledge, the setting of minimum support threshold is quite tricky and it leads to the following problem that may hinder its popular use. There are two challenges of minimum support based mining: (i) if the value of minimum support is set to be too small, the pattern mining algorithm may lead to the generation of thousands of patterns; (ii) if the value of minimum support constraint is set to be too big, the mining algorithm may often generate a few patterns or even no answers. In which case, the user may have to guess a smaller threshold and do the mining again, which may or may not give a better result. As it is difficult to predict how many patterns will be mined with a user-defined minimum support threshold, the top-*k* pattern mining has been proposed. As a consequence many works have focused on avoiding the use of a support threshold (e.g. (Li et al., 1999; Cheung and Fu, 2004; Koh, 2008)), or avoiding the use of the support itself (e.g. (Cohen et al., 2001), (Bhattacharyya and Bhattacharyya, 2007) and (Le Bras et al., 2010)). Another solution involves asking the number of desired outputs (Fu et al., 2000). Therefore, mining top-*k* patterns has become a very popular task. In particular, top-*k* frequent closed patterns (e.g. (Han et al., 2002), (Wang et al., 2005) and (Pietracaprina and Vandin, 2007)) and top-*k* patterns (e.g. (Fu et al., 2000) and (Yang et al., 2008)) have motivated a lot of works. Nowadays, mining from data streams also offers a new challenge because one cannot save all the patterns and their related information, due to the limitation of memory space. Thus mining top-*k* patterns from data streams becomes of great interest (e.g. (Metwally et al., 2005), (Li, 2009b), (Li, 2009a) and (Tsai, 2010)).

Recently, Tanbeer et al. (Tanbeer et al., 2009) proposed a pattern mining approach with a regular constraint on patterns appearance and a minimum support constraint. As pointed out by the authors, there are several applications to apply regular frequent patterns mining: in a retail market, among all frequently sold products, the sales manager may be interested only on the regularly sold products compared to the rest; for web site design or web administration, an administrator may be interested on the click sequences of heavily hit web pages; in genetic data analysis the set of all genes that not only appear frequently but also co-occur at regular interval in DNA sequence may carry more significant information to scientists; for stock market, the set of high stocks indices that rise regularally may be of special interest to traders, etc. Thus the occurrence regularity plays an important role in discovering some interesting frequent patterns in such applications (Engler, 2008) (Laxman and Sastry, 2006).

This dissertation here focuses on these two bottlenecks and extend the work of (Tanbeer et al., 2009). Thus, a new kind of pattern, namely the top-*k* regular-frequent itemsets, which is discovered from transactional databases is proposed. From this kind of pattern, the users can control the number of regular-frequent itemsets to be mined. At first, MTKPP algorithm (Mining Top*K* Periodic(Regular)-frequent Patterns) is introduced. It based on the use of top-*k* list structure and a best-first search strategy to quickly discover the regular itemsets with high supports. To calculate support of each itemset, a set of transaction-ids (that each itemset occurs) is collected. MTKPP also applies intersection process to collect and calculate a set of transaction-ids, a regularity and a support of each larger itemset. Next, TKRIMPE algorithm (Top-*K* Regular Itemset Mining using database Partitioning and support Estimation) is presented. TKRIMPE based on the database partitioning and support estimation techniques. By using these techniques, TKRIMPE can achieve a good performance especially on sparse datasets. Further, a new concise representation named interval tidset representation is devised. Then, a new efficient algorithm, called TKRIMIT (Top-*K* Regular-frequent Itemsets Mining based on Interval Tidset representation), is also proposed. With interval tidset representation, TKRIMIT can reduce the processing time and memory usage on dense datasets. Finally, an efficient and scalable algorithm named H-TKRIMP (Hybrid representation on Top-*K* Regular-frequent Itemsets Mining based on database Partitioning), based on the combination between normal tidset and interval tidset representations and the database partitioning, is devised. By comparing with other algorithms, H-TKRIMP can achieve a good performance for the small and large values of desired results on both sparse and dense datasets.

1.1 Objectives of Study

The objectives of study are as follows:

- To develop algorithms to mine top-*k* regular-frequent patterns that are very efficient in the terms of computational time and memory consumption.
- To develop a new technique to collect tidset of each itemset which can be applied in various problems such as frequent pattern mining, frequent closed pattern mining, and weighted frequent pattern mining.
- To propose an analysis of the performance of various techniques to maintain and intersect tidsets by making a comparison to a see trade-off between time and space.

1.2 Scopes of Study

The scopes of this study are as follows:

- This work considers the problem of top-*k* regular-frequent pattern mining.
- The datasets from *UCI* repository are used as a benchmark to test the proposed algorithms.
- Performance measurement can be either an actual running time (an actual memory consumption) or a complexity analysis.

1.3 Research Methodology

- Survey literature and review related works about association rules mining, frequent itemsets mining, frequent closed-itemsets mining, top-*k* frequent itemsets mining and regular-frequent itemsets mining.
- Study the principle theories and various techniques to mine frequent and other kinds of itemsets.
- Study various proposed representations used to maintain the content of databases.
- Collect the sparse and dense datasets from standard and existing benchmark datasets.
- Design an appropriate algorithms and perform the experiments to validate the algorithms
- Conclude the experimental results by comparing the results with those from other methods

1.4 Organization

The remainder of this dissertation is structured as follows: In Chapter 2, general background on association rules mining and its variant are introduced. Further, the frequent pattern mining, top-*k* itemsets mining and regular-frequent itemsets mining problems and related works are described. In Chapter 3, the formal notations and definitions used to mine a set of top-*k* regular-frequent itemsets are mentioned. An efficient algorithm, named MTKPP (Mining Top-*K* Periodic(Regular)-frequent Patterns), used a normal-tidset representation and applied a best-first search strategy is introduced. Chapter 4 presented a new efficient algorithm, called TKRIMPE (Top-*K* Regular-frequent Itemsets Mining using database Partitioning and support Estimation), applied the database partitioning and support estimation techniques in order to reduce computational time of MTKPP algorithm. Besides, a new concise interval tidset representation named interval tidset representation and an efficient algorithm called TKRIMIT (Top-*K* Regular-frequent

Itemsets Mining using Interval Tidset representation algorithm) is also proposed in Chapter 5. As further extensions, in Chapter 6, the database partitioning technique and the interval tidset representation are merged in order to devise a new algorithm, named H-TKRIMP (Hybrid representation on Top-*K* Regular-frequent Itemsets Mining based on database partitioning), that have a good performance on both dense and sparse datasets. Finally, Chapter 7 concludes this dissertation and describes future extension of this work.

CHAPTER II

RELATED WORK

The association rule mining problem has been extensively studied from various aspects over the past decade. As mentioned in the previous chapter, association rule mining consists of 2 steps: frequent itemsets mining and association rules generation. Most of previous works focus on frequent itemset mining which is the most time consuming step. They have been proposed to extend frequent itemsets mining for many purposes. One of interesting approaches is to control the number of itemsets to be mined (Fu et al., 2000), called top-*k* significant itemsets mining approach. Besides, many researchers try to find more interesting patterns (itemsets) by using other criteria instead of using only a support threshold. Recently, the regularity measure have been devised to discover itemsets that occur very frequent and regularly in transactional databases. Thus, this chapter surveys on previous works on the frequent itemsets mining, top-*k* significant itemsets mining and regular-frequent itemsets mining.

2.1 Frequent itemsets mining

The frequent itemsets and association rule mining is first introduced by (Agrawal et al., 1993). Most of association rule mining algorithms adopt the two-phase approach and focus on the frequent itemset mining in the context of transaction databases. A transaction database is a database containing a set of transactions and each transaction is associated with a transaction-id. The basic terms needed for describing association rules and frequent itemsets mining are given by using the formalism of (Agrawal and Srikant, 1994).

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items, that have been used as information units in an application domain and TDB be a database which is a set of transactions, where a transaction t is a subset of I ($t \subseteq I$). Each transaction is identified by a transaction-id tid. A set $X = \{i_{j_1}, \ldots, i_{j_l}\} \subseteq I$ is called an *itemset* or a *l-itemset* (an itemset of size l). If $X \subseteq Y$, it is said that t_q contains X (or X occurs in t_q). The support of an itemset X in a database is denoted by s^X, and is defined as $s^X = |\{t_q | 1 \leq q \leq TDB, t_q \in TDB \text{ and } X \subseteq t_q\}|/|TDB|$. An itemset X is called a *frequent itemset* if its support is greater than or equal to a support threshold specified by the user, otherwise the itemset is not frequent. An association rule is an expression of the form $X \rightarrow Y$, where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \phi$. Note that each of X and Y is a frequent itemset which contains a set of one or more items and the quantity of each item is not considered.

The itemset X is referred to as the antecedent of the rule and the itemset Y as the consequence. An example of association rule is the statement that 80% of transactions that purchase A also purchase B and 10% of all transactions contain both of them. Here, 10% is the support of the itemset A, B and 80% is the confidence of the rule $A \rightarrow B$. An association with the confidence greater than or equal a confidence threshold is considered as a valid association rule.

In association rules mining, two types of database layouts are employed: horizontal and vertical databases. As shown in Table 2.1, the traditional horizontal database contains a set of transactions and each transaction consists of a set of items. Most Apriori-like algorithms use this type of layout. On the other hand, as illustrated in Table 2.2, each item in the vertical database layout maintains a set of transaction-ids (denoted by tidset) where it occurs. Based on the vertical representation of database, various algorithms were devised to mine the results such as Eclat (Zaki et al., 1997), VIPER (Shenoy et al., 2000) and Mafia (Burdick, 2001). Lastly, as pointed out in (Zaki et al., 1997) and (Shenoy et al., 2000), they have been shown the trade-off between both layouts. They claimed that the vertical layout performs generally better than the horizontal format.

Table 2.1: Horizontal representation

tid	items
1	*a b d e*
2	*c d e*
3	*b c f g*
4	*a b d f g*
5	*c e g*
6	*a b c d g*
7	*a b c d*
8	*a b c e*
9	*b c d*
10	*a c e g*
11	*a b f*
12	*a b d g*

A large number of efficient algorithms to mine frequent itemsets have been developed over the years. The strategies developed to speed up frequent itemset mining process can be divided into two approaches. The first is based on the candidate generation-and-test approach. The Apriori algorithm and its several variations belong to this category. Apriori employs a bottom-up, breadth-first search that enumerates every single frequent itemset. It also provides the Apriori property also known as anti-monotone property that any subset of a frequent itemset must be a frequent. In this approach, a set of candidate itemsets of length $n + 1$ is generated from the set of frequent itemsets of length n and then each candidate itemset is checked to see if it meets the support threshold. Some algorithms adopt an Apriori-like method, and are focused on reducing

Table 2.2: Vertical Tidset representation

items	*tidset*
a	1 4 6 7 8 10 11 12
b	1 3 4 6 7 8 9 11 12
c	2 3 5 6 7 8 9 10
d	1 2 4 6 7 9 12
e	1 2 5 8 10
f	3 4 11
g	3 4 5 6 10 12

the number of candidate itemsets, which in turn reduce the time required for scanning databases, are briefly described as follows. Park et al. (Park et al., 1995) proposed an efficient Direct Hashing and Pruning (DHP) algorithm to control the number of candidate 2-itemsets and prune the size of the database by utilizing a hash technique. The inverted hashing and pruning (IHP) algorithm (Holt and Chung, 2002) was proposed. It is similar to the Direct Hashing and Pruning (DHP) algorithm (Park et al., 1995) in the sense that both use hash tables to prune candidate itemsets. In DHP, every k-itemset within each transaction is hashed into a hash table. In IHP, the transaction identifiers of each item of the transactions that contain the item are hashed into a hash table associated with the item. The Tree-Based Association Rule (TBAR) algorithm (Berzal et al., 2001) employs an effective data-tree structure to store all itemsets to reduce the time required for database scans. Further, Cheung et al. proposed Fast Distributed Mining (FDM) of association rules (Cheung et al., 1996) to efficiently discover frequent itemsets, which is a parallelization of the Apriori algorithm. At each level, a database scan is independently performed. In 1997, Brin et al. proposed the Dynamic Itemset Count (DIC) (Brin et al., 1997b) algorithm to locate frequent item sets, which uses fewer passes over the database than classic algorithms, and fewer candidate itemsets than methods based on sampling. Agarwal et al. presented the TreeProjection method (Agarwal et al., 2001) using the hierarchical structure of a lexicographic tree to project transactions at each node of the tree, and matrix counting on this reduced set of transactions for mining frequent itemsets. Another efficient method (Tsay and Chang-Chien, 2004) uses the techniques of clustering transactions and decomposing larger candidate itemsets for mining frequent itemsets. Tsay et al. proposed the Cluster-Based Association Rule (CBAR) method(Tsay and Chiang, 2005), which uses cluster tables to load the database into a main memory that requires only a single scan of the database. Its support count is performed on cluster tables, and thus, does not need to rescan the whole database. The Efficient Dynamic Database Updating Algorithm (EDUA) (Zhang et al., 2007) is designed for mining dynamic databases when some data are deleted. A special database structure BitTableFI (Dong and Han, 2007) is used horizontally and vertically to compress the database for quickly generating candidate itemsets and counting support.

Apriori-inspired algorithms show good performance with sparse datasets such as market basket data, where the frequent patterns are very short. However, with dense datasets such as telecommunications and census data, where there are many, long frequent patterns, the performance of these algorithms degrades incredibly. This degradation is due to the following reasons: these algorithms perform as many passes over the database as the length of the longest frequent pattern. Secondly, they have to generate and test the huge number of candidate itemsets. Thirdly, a frequent pattern of length l implies the presence of $2^l - 2$ additional frequent patterns as well, each of which is explicitly examined by such algorithms. When l is large, the frequent itemset mining methods become CPU bound rather than I/O bound. In other words, it is practically unfeasible to mine the set of all frequent patterns for other than small l. On the other hand, in many real world problems (e.g., patterns in biosequences, telecommunications data, census data, etc.) finding long itemsets of length 30 or 40 is not uncommon (Bayardo, 1998).

The second approach of pattern-growth has been proposed more recently. It also uses the Apriori property, but instead of generating candidate itemsets, it recursively mines patterns in the database to count the support for each pattern. Han et al. (Han et al., 2004) proposed the FP-growth method to avoid generating candidate itemsets by building a FP-tree with only two scans over the database. This milestone development of frequent itemsets mining avoids the costly candidate itemsets generation phase, which overcomes the main bottlenecks of the Apriori-like algorithms. Some algorithm analogy for FP-growth without generating candidate itemsets is briefly described as follows. The H-mine method (Pei et al., 2001b) uses a memory-based hyper structure to store a sparse database in the main memory, and builds an H-structure to invoke FP-growth in mining dense databases. An inverted matrix approach uses an inverted matrix to store the transactions in a special layout, then builds and mines relatively small structures, which are called COFI-trees (El-hajj and Zaiane, 2003). The CFP-tree structure (Liu et al., 2007) is designed to store pre-computed frequent itemsets on a disk to save space. A CFP-tree stores discovered frequent itemsets, but a FP-tree stores transactions. They both use prefix and suffix sharing in the CFP-tree, but only prefix sharing in the FP-tree. Maximum length frequent itemsets are generated by adapting a pattern fragment growth methodology (Hu et al., 2008) based on the FP-tree structure. For most data sets, these algorithms perform better than Apriori. Among the existing pattern-growth algorithms, H-Mine runs faster than FP-Growth on several commonly used test data sets.

2.2 Top-*k* significant itemsets mining

From mining frequent itemsets, a major problem is that user has to define a support threshold. However it is quite difficult for users to set a definite support threshold if they have no special

knowledge in advance. If the threshold is set too small, there will be a large number of itemsets having been found, which not only consumes more time and space resource, but also brings much burden to users on analyzing the mining results. On the contrary, if the threshold is set too large, there might be much few even no frequent itemsets, which implies some interesting patterns are hidden owing to the improper determination of support threshold. In some cases of application, it is natural for user to specify a simple threshold on the amount of mining results, say the most 100 frequent itemsets should be found.

Therefore it is of interest to mine the most k frequent itemsets over transactional databases with the highest supports.

Definition 2.1 *An itemset X is a top-k frequent itemset if there exist no more than $(k-1)$ itemsets whose support is higher than that of X.*

To focus on mining top-*k* patterns, Chueung et al. proposed N-most interesting itemsets mining (Cheung and Fu, 2002)(Cheung and Fu, 2004). This task mines only the N *k*-itemsets with highest supports for 1 up to a certain k_{max}, where k_{max} is the upper bound of the length of itemsets, and N is the desired number of *k*-itemsets. Three algorithms were proposed for this mining: *LOOPBACK, BOLB, and BOMO*. All three algorithms are adapted from the *FP-tree* approach. *BOMO* has two phases. First, it builds the complete *FP-tree* with all items in the database to find minimum support threshold of each *k*-itemset. Then, it mines itemsets. During the mining process, the support threshold of all itemsets is increased by considering the minimum value among the support of the N^{th} most frequent *k*-itemset discovered. It is used to prune unpromising itemsets. *LOOPBACK* builds the *FP-tree* and initializes the support threshold to be the support of the N^{th} sorted largest *1*-frequent. If the number of any *k*-itemsets is less than N, the tree will be rebuilt to find the smaller support in order to mine more itemsets in the mining phase. *BOLB* is a hybrid approach of *BOMO* and *LOOPBACK*. Like *BOMO*, it builds the complete *FP-tree* only once. The mining process is applied from the technique of *LOOPBACK*. Wang et al. proposed mining top-*k* frequent closed itemsets of length no less than min_l (Han et al., 2002) (Wang et al., 2005), where k is the desired number of frequent closed itemsets to be mined, and min_l is the minimal length of closed itemsets. *TFP* starts with minimum support threshold at 0. It constructs an *FP-tree* to raise the threshold and uses the threshold to prune the tree. It may take a long time to construct the *FP-tree* to find the final threshold and to prune the tree if the database contains many transactions and long patterns. Moreover, *TFP* has to maintain candidates to ensure that they are really closed. Top-k closed itemset mining was extended to mine top-k closed sequence in (Tzvetkov et al., 2005). Pietracaprina et al. proposed *TopKMiner*

(Pietracaprina and Vandin, 2007) to mine top-*k* closed itemsets without considering the minimal length of them. From this mining, it can allow a user to dynamically raise the value k with no need to restart the computation. *TopKMiner* mines top-*k* closed itemsets by combining the *LCM* algorithm (Uno et al., 2004a)(Uno et al., 2004b) and priority queue to avoid closed checking. In addition, it adopts best first search to generate closed itemsets with highest support first. This idea is supported by C. Wu (Wu, 2006). He proved that a heuristic algorithm is preferred over an exact algorithm to solve top-*k* closed itemset mining.

2.3 Regular-frequent itemsets mining

Mining frequent patterns, periodic pattern (Elfeky et al., 2005)(Maqbool et al., 2006)(Lee et al., 2006) and cyclic patterns (Özden et al., 1998) in static database have been well-addressed over the last decade. Periodic pattern (also called regular pattern) mining problem in time-series data focuses on the cyclic behavior of patterns either in the whole (Elfeky et al., 2005) or at some point (Lee et al., 2006) of time-series. Such pattern mining has also been studied as a wing of sequential pattern mining (Maqbool et al., 2006)(Lee et al., 2006) in recent years. However, although periodic pattern mining is closely related with regular-frequent pattern mining, it cannot be directly applied for finding regular patterns from a transactional database because of two primary reasons. First, it considers either time-series or sequential data. Second, it does not consider the support threshold which is the only constraint to be satisfied by all frequent patterns. Tanbeer (Tanbeer et al., 2009) proposed the regular-frequent pattern mining technique, on the other hand, introduces a new interesting measure of regularity and provides the set of patterns that satisfy both of the regularity and support thresholds in a transactional database.

Ozden et. al. (Özden et al., 1998) proposed a method to discover the association rules (Agrawal et al., 1993) occurring cyclically in a transactional database. It outputs a set of rules that maintains a cyclic behavior in appearance among a predefined non-overlapping database segments. The main limitation of this method is segmenting the database into a series of fixed sized segments, which may suffer from border effect. That is, if the sufficient number of occurrences of a pattern (to become frequent) occurs in the borders of two consecutive segments, the pattern might be ignored to generate association rules.

The problem of regular-frequent itemsets mining which has similar definition to (Tanbeer et al., 2009) is defined as follows:

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of $n \geq 1$ literals, called items. A set $X = \{i_1, \ldots, i_k\} \subseteq I$ is called an itemset (or a pattern), or a k-itemset if it contains k items. A transactional database

$TDB = \{t_1, t_2, \ldots, t_m\}$ over I is a set of $m = |TDB|$ transactions. Each transaction $t_j = (tid, Y)$ is a tuple where $tid = j$ represents the transaction-id and $Y \subseteq I$ is an itemset. If $X \subseteq Y$, it is said that t_j contains X (or X occurs in t_j) and is denoted as t_j^X. Therefore, $T^X = \{t_j^X, \ldots, t_k^X\}$, where $j, k \in [1, m]$ and $j \leq k$, is the set of all *ordered tids*, called *tidset*, where X occurs. The support of an itemset X in TDB, denoted as $Sup(X) = |T^X|$, is the number of transactions in TDB that contains X.

Let t_j^X and t_k^X be two consecutive tids in T^X, i.e. where $j < k$ and there is no transaction t_i, $j < i < k$, such that t_i contains X. Thus, $rtt^X = t_k^X - t_j^X$ is the regularity value which represents the number of missing transaction of X between two consecutive transactions t_j^X and t_k^X. We denote as $RTT^X = \{rtt_1^X, rtt_2^X, \ldots, rtt_z^X\}$, the set of all regularities of X between each pair of two consecutive transactions. Then, the regularity of X can be defined as $r^X = max(rtt_1^X, rtt_2^X, \ldots, rtt_z^X)$.

Therefore, an itemset X is called a *regular-frequent itemset* if (i) its regularity is no greater than a user-given maximum regularity threshold (σ_r); (ii) its support is no less than a user-given minimum support threshold (σ_s). Thus, the regular-frequent itemsets mining problem is to discover the complete set of regular-frequent itemsets from *TDB* with two user-given thresholds: minimum support and maximum regularity threshold which are defined in the percentage of $|TDB|$.

However, as pointed out before it is quite difficult for users to set a definite support threshold if they have no special knowledge in advance. In addition, in some cases, it is natural for user to specify a simple threshold on the amount of regular-frequent patterns, say the most 100 frequent patterns with regularity less than $1,000$ transactions*(i.e. occur at least once in every* $1,000$ *transactions)*. It is thus of interest to mine the most frequent *k* regular patterns over transactional databases without the minimum support threshold requirement.

2.4 Benchmark datasets

To validate the performance of the variant of association rule mining algorithms, several real (*i.e.* accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb*, and retail) and synthetic database benchmarks (*i.e.* T10I4D100K, T20I6D100K, and T40I10D100K), publicly available from IBM Almaden (http://www.almaden.ibm.com/cs/quest/syndata.html), FIMI repository (http://fimi.cs.helsinki.fi/data/), and UC-Irvine Machine Learning Database Repository (http://www.ics.uci.edu/ mlearn/MLRepository.html), are utilized.

Data set accidents contains (anonymous) traffic accident data from the National Institute of Statistics (NIS) for the region of Flanders (Belgium) for the period 1991-2000. The BMS-POS dataset is a real world dataset containing several years worth of point-of-sale data from a large electronic retailer, aggregated at the product category level. The connect and chess datasets are derived from UCI Machine Learning Repository. Each transaction in connect and chess contains legal 8-ply positions in the game where neither player has won yet and the next move is not forced. The kosarak contains click-stream data of a hungarian on-line news portal. The mushroom database consists of records describing the characteristics of various mushroom species. The PUMS datasets (pumsb and pumsb*) contain census data. Pumsb* is the same as pumsb without items with 80% or more support. The retail market basket dataset is obtained from Belgian retail supermarket store from December 1999 to November 2000. The synthetic datasets (T10I4D100K, T20I6D100K and T40D100K), using the IBM generator, mimic the transactions in a retailing environment.

The classical characteristics of datasets were studies in (Gouda and Zaki, 2001). According to (Gouda and Zaki, 2001), the density was used to categorize the characteristics of datasets. A dataset is dense when it produces many long frequent itemsets for all values of support threshold. The authors studied on seven datasets, (*i.e.* chess, connect, mushroom, pumsb, pumsb*, T10I4D100K, and T40I4D100K) and then categorized these datasets according to the density. The density is estimated by using the characteristics of maximal frquent itemsets and more precisely their distribution.

However, as pointed out in (Flouvat et al., 2010), there are two limitations of Gouda's classification. First, its variability with respect to support threshold values. Second, there is no clear relationship between the proposed classification and algorithms performance. Therefore, (Flouvat et al., 2010) proposed a new classification which differs from the classification of (Gouda and Zaki, 2001). It takes into account both the negative border and the positive border of itemsets. The positive border of frequent itemsets is the set of all maximal frequent itemsets w.r.t. set inclusion. The negative border of frequent itemsets is the set of all minimal unfrequent itemsets w.r.t. set inclusion.

These different types of datasets have been identified by taking advantage of the "distance"between positive and negative borders distributions of frequent itemsets. As a consequence, the authors introduced a new classification of datasets made of three types:

- Type I datasets contain long itemsets in the positive border and a negative border closed to the positive border, *i.e.* the mean of the negative border curve is not far from the mean

of the positive border curve. In other words, most of the itemsets in the two borders have approximately the same size.

- Type II datasets contain long itemsets in the positive and a large distance between the two borders distributions. In other words, the itemsets in the negative border are much smaller than those of the positive border.
- Type III is a very special case of type O: the two distribution are very close, but they are concentrated in very low levels. This type allows to catch the notion of sparseness.

The Table 2.3 summarizes this new classification and shows how this study could also used for FIMI.

Table 2.3: Datasets classification from (Flouvat et al., 2010)

Type	Type I	Type II	Type III
Distance between the borders	Small	Large	Small
Itemsets size	Long	Long	Small
Examples of datasets	accidents, chess, pumsb	connect, mushroom, pumsb*	BMS-POS, kosarak, retail, T10I4D100K, T20I6D100K, T40I10D100K

Based on the classification of (Flouvat et al., 2010), the Table 2.4 shows the characteristics of the real and synthetic datasets used in the evaluation of this dissertation. It shows the number of items, the average transaction length, and the number of records in each database. The table additionally shows the tpye of datasets that are classified.

Table 2.4: Database characteristics

Database	#items	Avg.length	#Transactions	type
accidents	468	338	340,183	dense
BMS-POS	1,156	7.5	515,597	sparse
chess	75	37	3,196	dense
connect	129	43	67,557	dense
kosarak	41,270	8.1	990,002	sparse
mushroom	119	23	8,124	dense
pumsb	2,113	74	49,046	dense
pusmsb*	2,088	50.5	49,046	dense
retail	16,469	10.3	88,162	sparse
T10I4D100K	1,000	10.3	100,000	sparse
T20I6D100K	1,000	20.2	100,000	sparse
T40I10D100K	1,000	40.1	100,000	sparse

CHAPTER III

MINING TOP-*K* REGULAR-FREQUENT ITEMSETS

Based on the idea of "Controlling the number of regular-frequent itemsets to be mined"motivated from (Fu et al., 2000) and (Tanbeer et al., 2009), a problem of mining k regular-frequent with highest supports is introduced and defined in this chapter. Besides, an efficient single-pass algorithm named *Mining Top-K Periodic(Regular)-frequent Patterns (MTKPP)*, used to mine this kind of itemsets is also presented. To discover a set of top-*k* regular-frequent itemsets, the users can specify only a regularity threshold and a number of desired results instead of setting a support threshold. By avoiding the setting of a support threshold, this approach might help the users from the difficulty of specifying an appropriate support threshold to mine regular-frequent itemsets.

3.1 Top-*k* regular-frequent itemsets mining

This section introduces the basic notations and definitions needed to define top-*k* regular-frequent itemsets as defined in (Amphawan et al., 2009).

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of $n \geq 1$ literals, called *items*. A set $X = \{i_{j_1}, \ldots, i_{j_l}\} \subseteq I$ is called an *itemset* or an *l-itemset* (an itemset of size l). A transactional database $TDB = \{t_1, t_2, \ldots, t_m\}$ is a set of transactions in which each transaction $t_q = (q, Y)$ is a tuple containing unique transaction identifier q (tid in the latter) and an itemset Y. If $X \subseteq Y$, it is said that t_q contains X (or X occurs in t_q) and is denoted as t_q^X. Therefore, $T^X = \{t_p^X, \ldots, t_q^X\}$, where $1 \leq p \leq q \leq |TDB|$, is the set of all ordered tids (called *tidset*) where X occurs. The support of an itemset X, denoted as $s^X = |T^X|$, is the number of tids (transactions) in TDB where X appears.

Definition 3.1 (Regularity of an itemset X) *Let t_j^X and t_k^X be two consecutive tids in the tidset T^X of an itemset X,* i.e. *where $j < k$ and there is no transaction t_i, $j < i < k$, such that t_i contains X. Then, $rtt^X = t_k^X - t_j^X$ is the regularity value which represents the number of transactions not containing X between two consecutive transactions t_j^X and t_k^X. Thus, $RTT^X = \{rtt_1^X, rtt_2^X, \ldots, rtt_{m+1}^X\}$ is denoted as the set of all regularities of X. Then, the regularity of X can be defined as*

$$r^X = max(RTT^X) = max(rtt_1^X, rtt_2^X, \ldots, rtt_{m+1}^X)$$

Definition 3.2 (Regular-frequent itemset) *An itemset X is called a* regular-frequent itemset *if (i) its regularity is no greater than a user-given regularity threshold (σ_r); (ii) its support is no less than a user-given support threshold (σ_s).*

Thus, the regular-frequent itemsets mining problem is to discover a complete set of regular-frequent itemsets from transactional database with two user-given support and regularity thresholds. However, as mentioned in the previous chapter the user may prefer to specify a simple threshold on the amount of results instead of a support threshold. The following definition of a *top-k* regular-frequent itemsets mining problem is thus proposed.

Definition 3.3 (Top-*k* regular-frequent itemset) *An itemset X is called a* top-k regular-frequent itemset *if (i) its regularity is no greater than a user-given regularity threshold (denoted as σ_r) and (ii) there exist no more than $k-1$ itemsets whose their supports are higher than that of X.*

Therefore, the top-*k* regular-frequent itemsets mining problem is to discover a set of top-*k* regular-frequent itemsets from transactional database with two user-given parameters: a number k of expected outputs and a regularity threshold σ_r.

3.2 Preliminary of MTKPP

In this section, details of the MTKPP algorithm which is an efficient single-pass algorithm used to discover a set of k regular itemsets with highest supports from a transactional database are introduced. It adopts a best-first search strategy to quickly find regular itemsets with the highest values of support. MTKPP is based on the use of a top-*k* list (with hash table) structure to maintain top-*k* regular-frequent itemsets during mining process.

3.3 MTKPP: Top-*k* list structure

Top-*k* list is a linked-list used to maintain *k* periodic(regular)-frequent patterns with highest supports. A hash table is also used with the top-*k* list in order to quickly access information in the top-*k* list. At any time during mining process, the top-*k* list contains not much more than k regular-frequent itemsets in main memory. Each entry in a top-*k* list consists of 4 fields: an item or itemset name (I), a total support (s^I), a regularity (periodicity) (r^I) and a tidset where I occurs (T^I). For example in Figure 3.1, an item a has a support of 8, a regularity of 3. Its tidset is $\{1, 4, 6, 7, 8, 10, 11, 12\}$ which means the item a occurs in $\{t_1, t_4, t_6, t_7, t_8, t_{10}, t_{11}, t_{12}\}$.

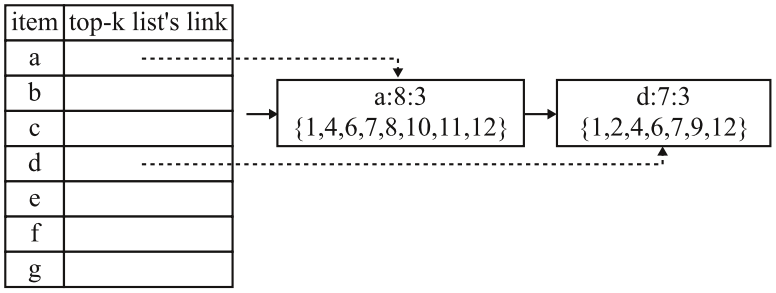


Figure 3.1: MTKPP: Top-*k* list with hash table

3.4 MTKPP algorithm

MTKPP consists of two steps: (*i*) Top-*k* list initialization: scan a database once to obtain k regular items (with highest support) and collect them into the top-*k* list with their supports, regularities and tidsets; and (*ii*) Top-*k* mining: merge each pair of entries in the top-*k* list by using the best-first search strategy (*i.e.* finding the itemsets with the highest support first in order to reduce search space) to generate a larger candidate itemset and then sequentially intersect their tidsets to calculate support and regularity of the new generated itemset.

3.4.1 MTKPP: Top-*k* list initialization

To create the top-*k* list, the database is scanned once to obtain all items. At the first occurrence of each item, the MTKPP algorithm creates a new entry in the top-*k* list and then initializes its support, regularity and tidset. For other occurrences, the hash table is looked up to find the existing entry in the top-*k* list and update the entry values. All items that have regularity greater than σ_r are removed from the top-*k* list and the top-*k* list is sorted in support descending order. Finally, all items that have support less than the support of the k^{th} item in top-*k* list (s_k) are removed from the top-*k* list. The details of the top-*k* list initialization process are described in Algorithm 1.

3.4.2 MTKPP: Top-*k* mining

To mine a set of top-*k* regular-frequent itemsets from the top-*k* list, the best-first search strategy is adopted first to generate regular itemsets with the highest supports. To generate a new candidate itemset, MTKPP starts from considering the most regular-frequent item to the least regular-frequent item in the top-*k* list. It then combines two entries in the top-*k* list under the following two constraints: (i) the size of the itemsets of both considered entries must be equal; (ii) both itemsets must have the same prefix (*i.e.* each item from both itemsets is the same, except the last item). When both itemsets satisfy the two constraints above, MTKPP will sequentially intersect their tidsets in order to calculate the support, the regularity, and the tidset of the new

Algorithm 1 (MTKPP: Top-k list initialization)

Input:
(*1*) A transaction database: TDB
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r
Output:
(*1*) A top-*k* list

create a hash table for all 1-items
for each transaction j in TDB **do**
  for each item i in the transaction j **do**
    if the item i does not have an entry in the top-*k* list **then**
      create a new entry for the item i with $s^i = 1, r^i = t_j$ and create a tidset T^i that contains t_j
      create a link between the hash table and the new entry
    else
      add the support s^i by 1
      calculate the regularity r^i by t_j
      collect t_j as the last tid in T^i
for each item i in the top-*k* list **do**
  calculate the regularity r^i by $|TDB|-$ the last tid of T^i
  if $r^i > \sigma_r$ **then**
    remove the entry i out of the top-*k* list
sort the top-*k* list by support descending order
remove all of entries after the k^{th} entry in the top-*k* list

Algorithm 2 (MTKPP: Top-k mining)

Input:
(*1*) A top-*k* list
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r
Output:
(*1*) A set of top-*k* regular-frequent itemsets

for each entry x in the top-*k* list **do**
  for each entry y in the top-*k* list $(x > y)$ **do**
    if the entries x and y have the same size of itemsets and the same prefix **then**
      merge the itemsets of x and y to be itemset $Z = I^x \cup I^y$
      for each t_p in T^{I^x} and t_q in T^{I^y} **do**
        if $t_p = t_q$ **then**
          calculate the regularity r^Z by t_p
          add the support s^Z by 1
          collect t_p as the last tid in T^Z
      calculate the regularity r^Z by $|TDB|-$ the last tid of T^Z
      if $r^Z \leq \sigma_r$ and $s^Z \geq s_k$ **then**
        remove the k^{th} entry from the top-*k* list
        insert the itemset Z ($I^x \cup I^y$) into the top-*k* list with r^Z, s^Z and T^Z

generated candidate itemset. If the regularity of the new candidate itemset is not greater than σ_r and the support is greater than the support of the k^{th} regular itemset in the top-*k* list, then the k^{th} regular itemset will be removed from the top-*k* list and the newly generated candidate itemset is inserted into the top-*k* list. The details of the mining process are described in Algorithm 2.

3.5 Example of MTKPP

Let consider the TDB presented in Table 3.1. The regularity threshold σ_r and the number of required results k are 4 and 5, respectively. Figure 3.2 illustrates the creating of the top-*k* list process from the TDB.

Table 3.1: A transactional database as a running example of MTKPP

tid	items
1	*a b d e*
2	*c d e*
3	*b c f g*
4	*a b d f g*
5	*c e g*
6	*a b c d g*
7	*a b c d*
8	*a b c e*
9	*b c d*
10	*a c e g*
11	*a b f*
12	*a b d g*

With the scanning of the first transaction $t_1 = \{a, b, d, e\}$, the entries for items a, b, d and e are initialized in the top-*k* list as shown in Figure 3.2(a). The next transaction $(t_2 = \{c, d, e\})$ initializes a new entry in the top-*k* list for item c. It then updates the values of support and regularity for items d and e to be $2 : 1$ and their tidsets to be $\{1, 2\}$ (Figure 3.2(b)). As shown in Figure 3.2(c), after scanning the third transaction $(t_3 = \{b, c, f, g\})$, the regularity r^b of the item b changes from 1 to 2. The top-*k* list after scanning all transactions is given in Figure 3.2(d). Next, the item f which has the regularity $r^f = 7$ greater than $\sigma_r = 4$ is removed from the top-*k* list. Finally, the top-*k* list is sorted by support descending order and item e is removed from the top-*k* list, since the support of e $(s^e = 5)$ is less than support of g $(s^g = 6)$ which is the $k^{th}(5^{th})$ pattern in the top-*k* list. The top-*k* list after initialization phase is shown in Figure 3.2(e).

MTKPP mines the top-*k* regular-frequent itemsets from the top-*k* list of Figure 3.2(e). Since item b is the first item in the top-*k* list and it has no items in the previous sequence, MTKPP starts by considering item a and search for identical size and prefix items (in the previous sequence), item b. Then, item b is combined with item a and their tidsets are intersected to find the support $(s^{ba} = 7)$, the regularity $(r^{ba} = 3)$ and the tidset $(T^{ba} = \{1, 4, 6, 7, 8, 11, 12\})$ of itemset ba. Since the regularity of ba is less than $\sigma_r = 4$ and the support of ba is more than $s_k = 6$, the itemset ba is inserted into the top-*k* list and item g (the k^{th} itemset) is removed from the top-*k* list (Figure 3.3). Next, the third element, item c, is considered. There are two entries which are in the previous sequence and have the same prefix as c: b and a. Thus, item c is combined with

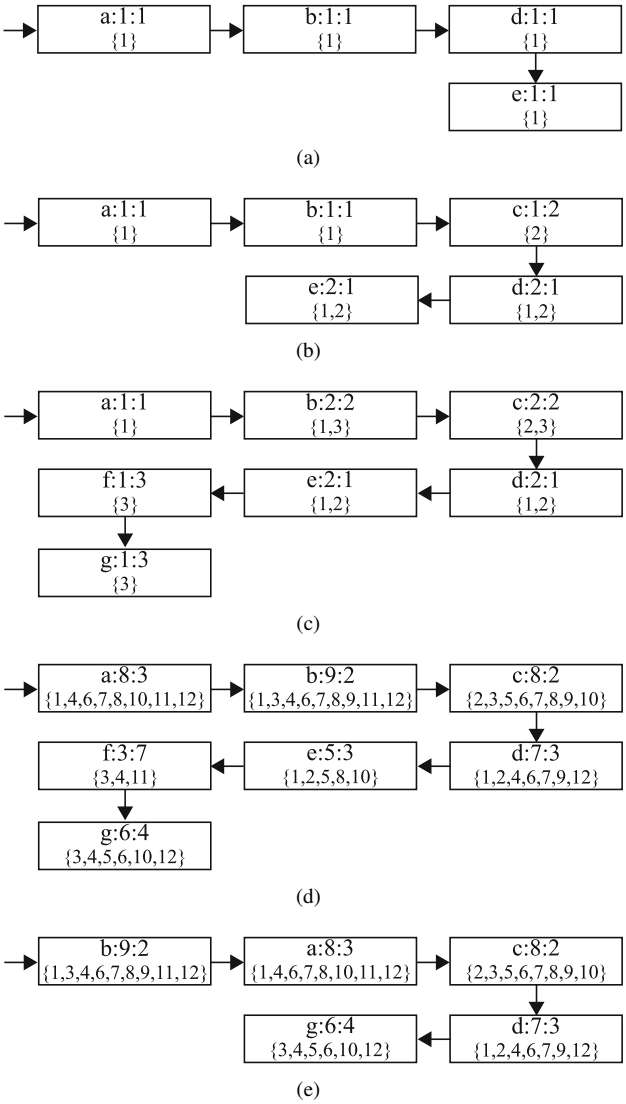


Figure 3.2: Top-*k* list initialization

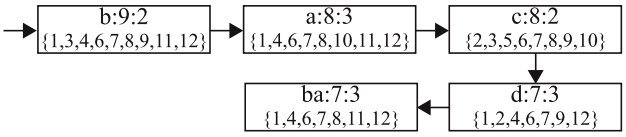


Figure 3.3: Top-*k* regular-frequent itemsets

item b and their tidsets are intersected. The tidset and the regularity of cb are $\{3, 6, 7, 8, 9\}$ and 3, respectively. Because the support of cb ($s^{cb} = 5$) is less than the support of $s_k = 7$, the itemset cb is no longer considered. Next, item c and item a are combined and their tidsets are intersected. The tidset of ca is then $\{6, 7, 8, 10\}$. Since the regularity of $ca(r^{ca} = 6)$ is greater than 4, itemset ca cannot be a regular itemset. Next, item d and itemset ba are considered in the same manner. When all itemsets in the top-*k* list have been considered, the top-*k* regular-frequent itemsets are stored in the top-*k* list with their occurrence information. The final result is shown in Figure 3.3.

3.6 Performance evaluation

In this section, the experimental studies are reported in order to evaluate the performance of the MTKPP algorithm. From the best of our knowledge, there is no other existing approach to discover top-*k* regular-frequent itemsets. Then, the effectiveness of MTKPP algorithm is focused and compared with PF-tree (Tanbeer et al., 2009) which is a regular-frequent itemset mining algorithm. It should be noticed that PF-tree mines the regular-frequent itemsets with a user-given support threshold whereas MTKPP requires the number of regular-frequent itemsets to be mined (k). Then, the support threshold is fixed in the way that PF-tree mines the same set of regular-frequent itemsets with highest supports as MTKPP (*i.e.* it is specified as $\sigma_s = s_k$ which is equal to the lowest support of the set of top-*k* regular-frequent itemsets). To demonstrate the performance of MTKPP, the processing time (*i.e.* CPU and I/Os costs) is investigated to compare the performance of the two algorithms with the small and large values of k and various values of regularity threshold (σ_r). Furthermore, a study of memory consumption of MTKPP is also considered because of the use of the top-*k* list structure. Lastly, the scalability of MTKPP on the number of transactions in the database is evaluated.

3.6.1 Experimental setup

As shown the characteristics in Chapter 2, nine real (*i.e.* accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb*, retail) and three synthetic (*i.e.* T10I4D100K, T20I6D100K, and T40I10D100K) datasets were employed to examine the performance of MTKPP. The simulations were performed on a Intel®Xeon 2.33 GHz and with 4 GB main memory on a Linux platform and the program of MTKPP and PF-tree implemented in C. In the experiments, the value of σ_r is set depending on the characteristic of each dataset for illustrative purpose. Therefore, the value of σ_r is specified to be different values. In fact, the number of regular itemsets for each database increases with the value of the regularity threshold. On sparse datasets, each itemset does not occur frequently thus the value of σ_r should be set to be large when the value of k is large. While, each itemset appears very often in dense dataset, a small value of σ_r should be applied. Hence, the value of k is divided into two rages: (*i*) [50,500] for the small values; and (*ii*) [1,000, 10,000] the large values, respectively.

3.6.2 Execution time

Figure 3.4 to Figure 3.21 show the runtime of MTKPP and PF-tree on real dense datasets (*i.e.* accidents, chess, connect, mushroom, pumsb, and pumsb*). From these figures, it can be observed that in almost cases, MTKPP outperforms PF-tree with the small and large values of

k. However, in some cases especially on connect and mushroom datasets when the value of k is large, MTKPP cannot significantly reduce the computational time from PF-tree. This happen because these two datasets have a small number of transactions (in some cases the number of transaction is less that the number of desired results). Then, PF-tree can reduce time to merge tidset from children to parent nodes, while MTKPP cannot take the advantage of using a top-k list.

Figure 3.22 to Figure 3.36 illustrate the processing time of two real sparse datasets (*i.e.* BMS-POS and retail) and the three synthetic datasets (T10I4D100K, T20I6D100K and T40I10D100K). One can observe that the computation time of MTKPP increases as k increases. When the value of k increases, MTKPP has to find more results, therefore the computation time increases as well. By comparing with PF-tree, MTKPP can save a large amount of time for small and large value of k. MTKPP runs very fast on sparse datasets since each itemset occur rarely (*i.e.* the number of tids that each itemset occurs is few). As a result, MTKPP spent a little time to intersect tidsets while PF-tree take time to merge and order tids. Therefore, these results confirm the advantage of MTKPP over PF-tree for the real and synthetic sparse datasets where the item distributes not regularly.

3.6.3 Memory consumption

The variation of memory usage of MTKPP with the number of regular-frequent itemsets to be mined, k, is shown in Figure 3.37 to Figure 3.47.

From these figures, it is obvious that the memory usage increases as k increases. In fact, the desired memory of MTKPP depends on the support of each itemset in the top-*k* list because MTKPP has to maintain the tidsets (*i.e.* sets of tids) of all itemsets in the top-*k* list in order to calculate their support and the regularity. For dense datasets, the memory usage linearly increases because the supports of itemsets in the top-*k* list are very close. For sparse datasets, the memory usage increases slightly as k increases because the supports of itemsets in the top-*k* list are quite different. However, based on the used of the top-*k* list structure, the memory usage of MTKPP is efficient for the top-*k* regular-frequent itemsets mining using the recently available gigabyte range memory.

3.6.4 Scalability test

The scalability of MTKPP algorithm is also studied on execution time and memory consumption by varying the number of transactions in database. The kosarak dataset is used to test scalability with the number of transactions. The kosarak dataset is a huge dataset with a large

number of distinct of items ($41,270$) and transactions ($990,002$). First, the database was divided into six portions (*i.e.* $100K$, $200K$, $400K$, $600K$, $800K$ and $990K$ transactions). Then, the performance of MTKPP was investigated on each portion. Second, the value of k is specified to be 500 and $10,000$ to investigate the scalability on the small and the large values of k. The regularity threshold was fixed to 6% of the number of transactions in each portion.

The experimental results shown in Figures 3.48 and 3.49. It is clear from the graphs that as the database size increases, overall top-*k* list initialization time and top-*k* mining time are linearly increased. The performance between MTKPP and PF-tree is similar when the number of transactions is between 0 and $200K$ transactions. Besides, MTKPP runs faster than PF-tree with the large number of transactions for the small and the large values of k. As shown the memory consumption of MTKPP in the figures, the memory requirement increases as the database size increases. However, MTKPP shows stable performance of about linearly increase of the runtime and memory usage with respect to the database size. Therefore, it can be observed from the scalability test that MTKPP can mine the top-*k* regular-frequent patterns over large datasets and distinct items with considerable amount of runtime and memory.

3.7 Summary

This chapter introduced and studied the problem of mining the top-*k* regular (periodic)-frequent itemsets from transactional databases without setting a support threshold. This problem allows users to control (or specify) the number of regular itemsets (*i.e.* the regularly-occurred itemsets) to be mined.

To discover this kind of itemset, an efficient one-pass algorithm, called *MTKPP (Mining Top-*K *Periodic(Regular)-frequent Patterns)*, is presented. Since the minimum support to retrieve top-*k* regular-frequent itemsets cannot be known in advance, a new best-first search strategy is devised to efficiently retrieve the top-*k* regular-frequent itemsets and the intersection process is applied to compute the support and the regularity of each itemset. By using these techniques, MTKPP first considers the itemsets with the highest support and then combines candidates to build the top-*k* regular-frequent itemsets list.

In the experiments, the empirical studies on both real and synthetic data (with the small and large values of k) show that the MTKPP algorithm is efficient for top-*k* regular-frequent itemset mining. It is also linearly scalable with the number of transactions comparing with PF-tree.

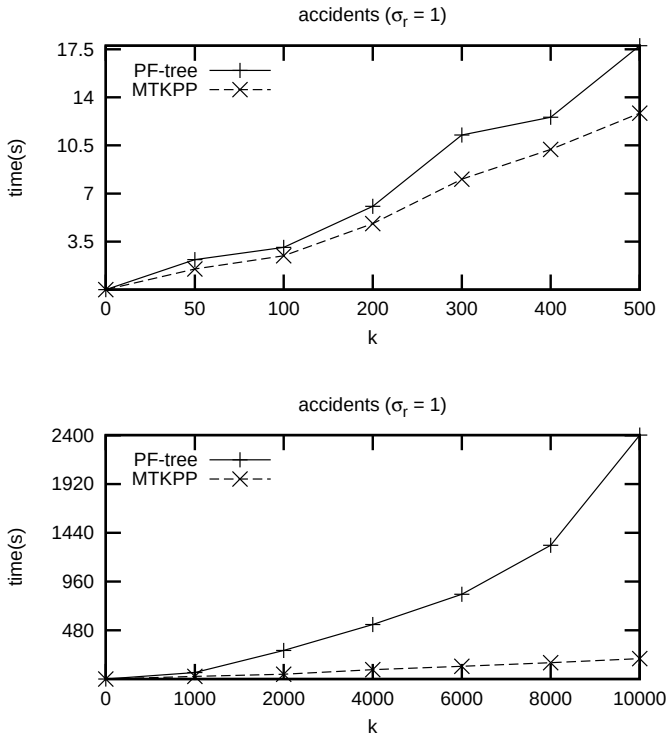


Figure 3.4: Runtime of MTKPP on *accidents* ($\sigma_r = 1\%$)

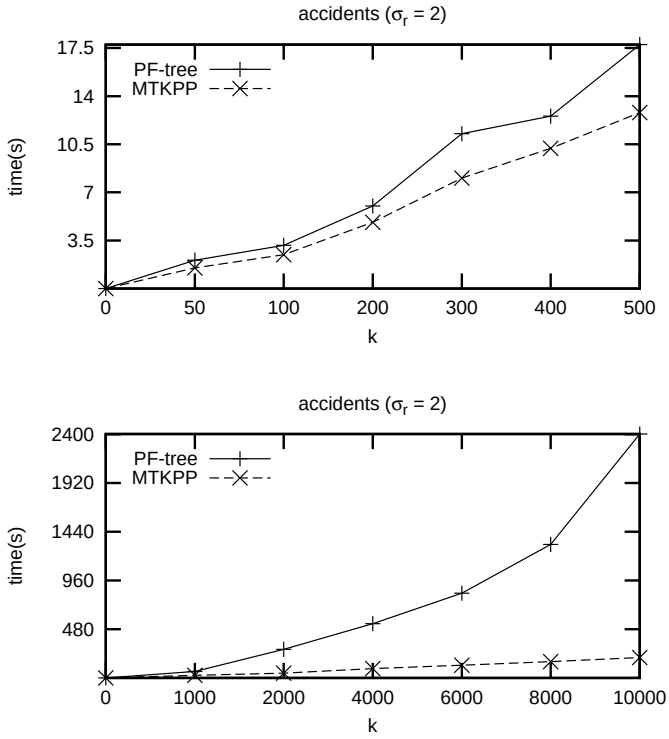


Figure 3.5: Runtime of MTKPP on *accidents* ($\sigma_r = 2\%$)

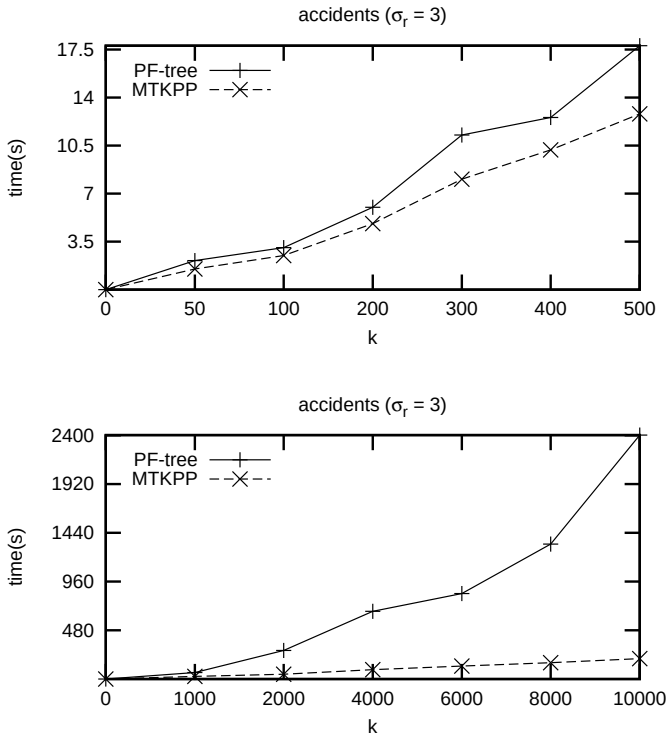


Figure 3.6: Runtime of MTKPP on *accidents* ($\sigma_r = 3\%$)

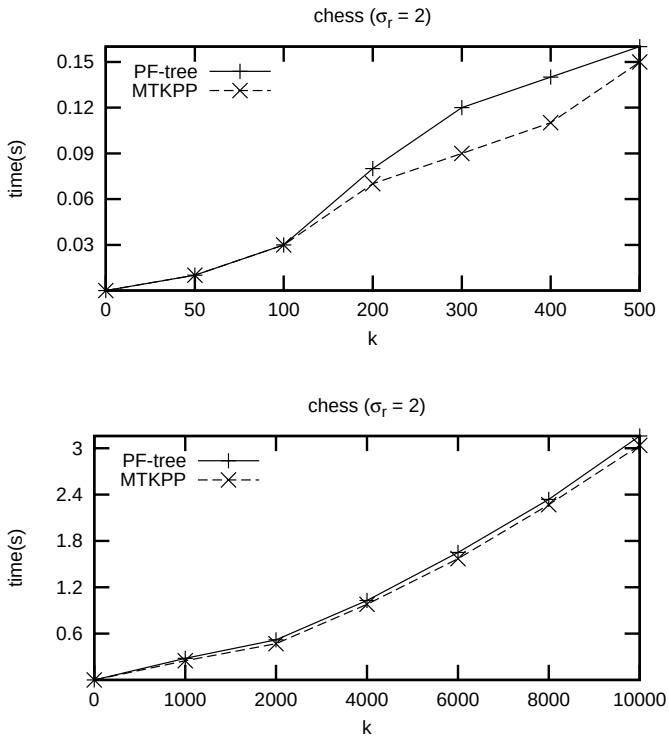


Figure 3.7: Runtime of MTKPP on *chess* ($\sigma_r = 2\%$)

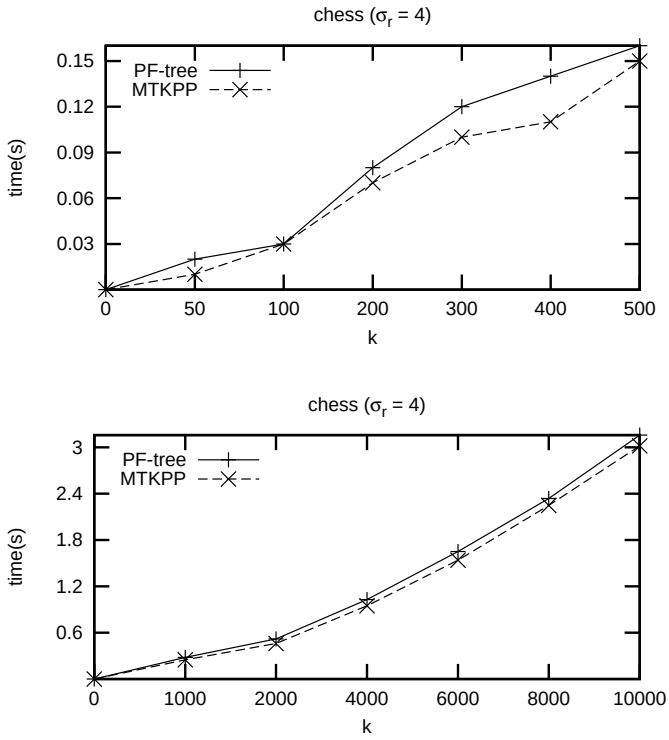


Figure 3.8: Runtime of MTKPP on *chess* ($\sigma_r = 4\%$)

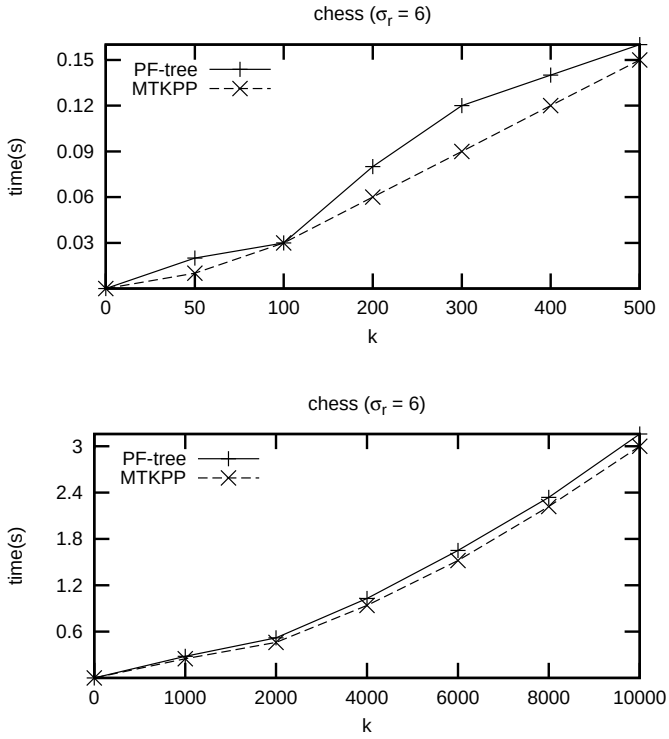


Figure 3.9: Runtime of MTKPP on *chess* ($\sigma_r = 6\%$)

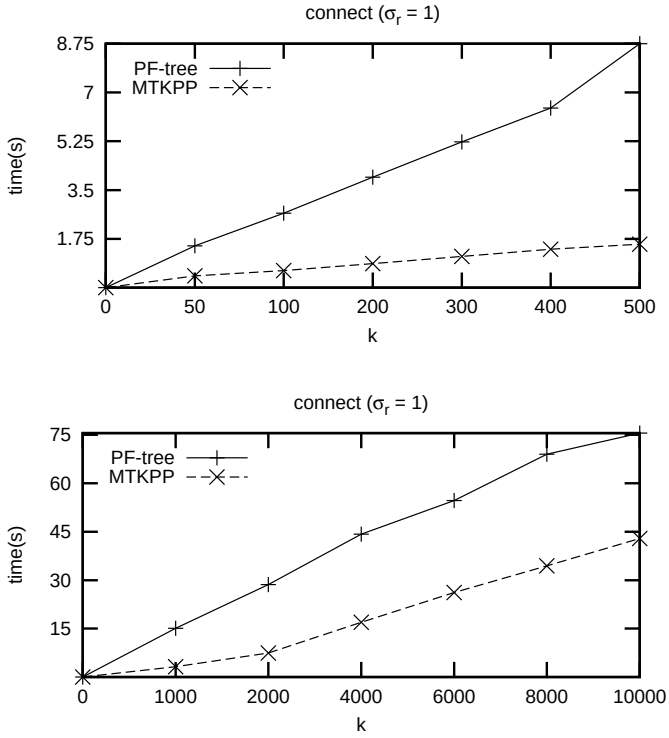


Figure 3.10: Runtime of MTKPP on *connect* ($\sigma_r = 1\%$)

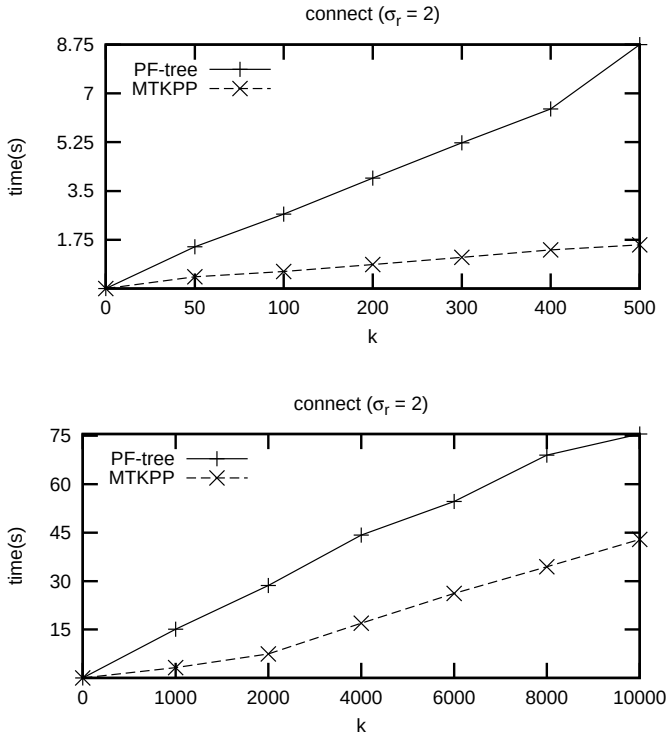


Figure 3.11: Runtime of MTKPP on *connect* ($\sigma_r = 2\%$)

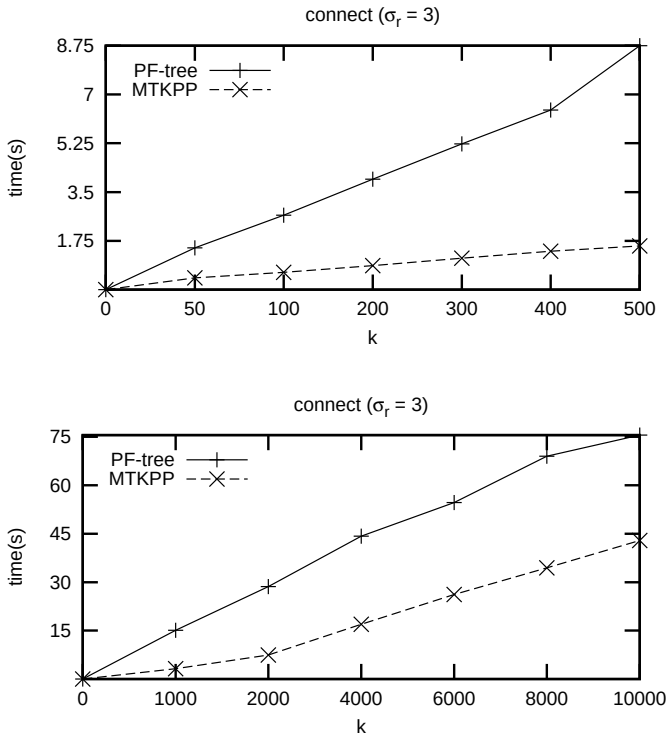


Figure 3.12: Runtime of MTKPP on *connect* ($\sigma_r = 3\%$)

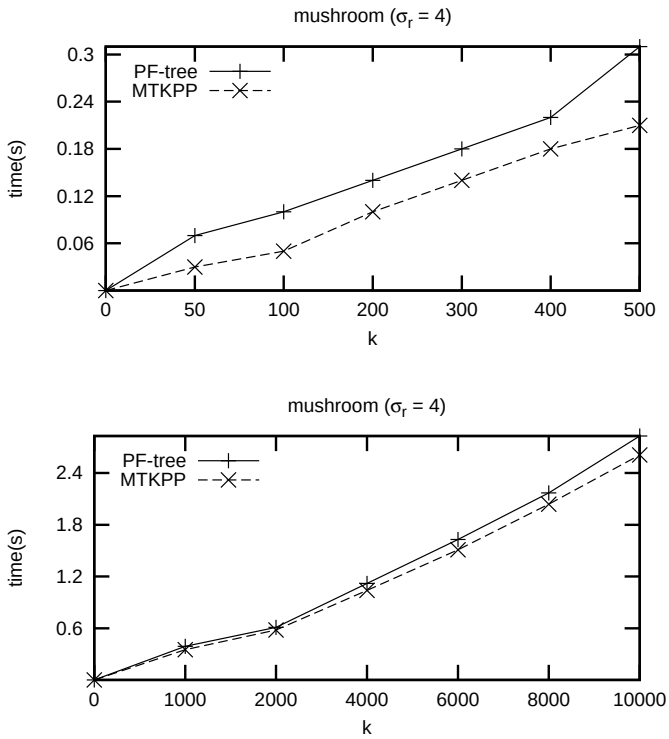


Figure 3.13: Runtime of MTKPP on *mushroom* ($\sigma_r = 4\%$)

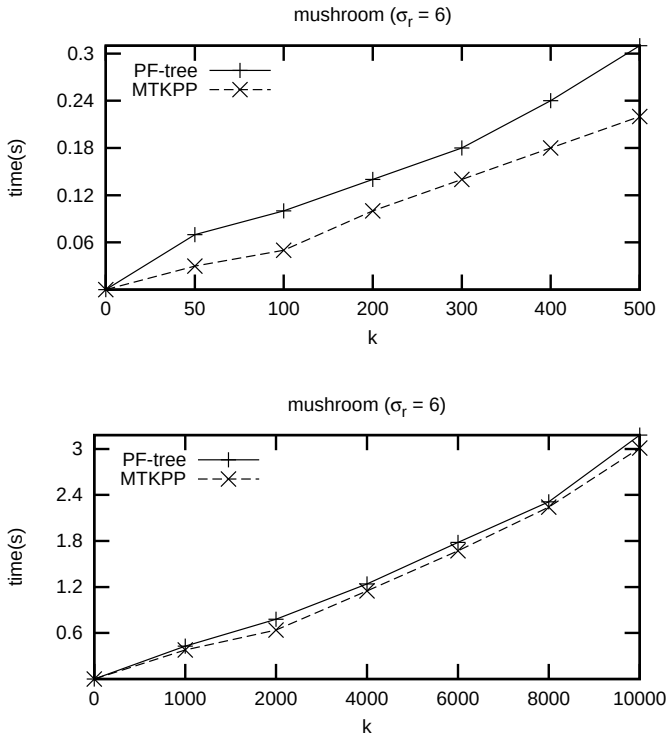


Figure 3.14: Runtime of MTKPP on *mushroom* ($\sigma_r = 6\%$)

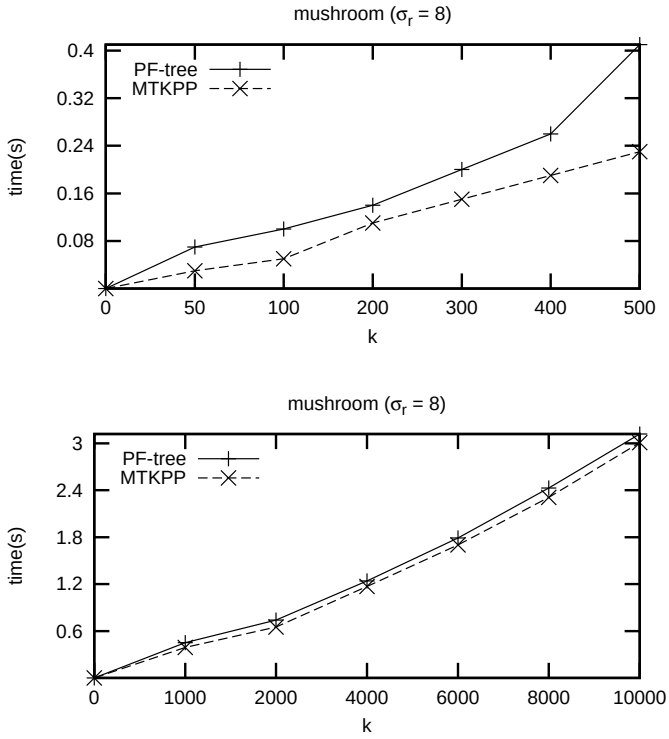


Figure 3.15: Runtime of MTKPP on *mushroom* ($\sigma_r = 8\%$)

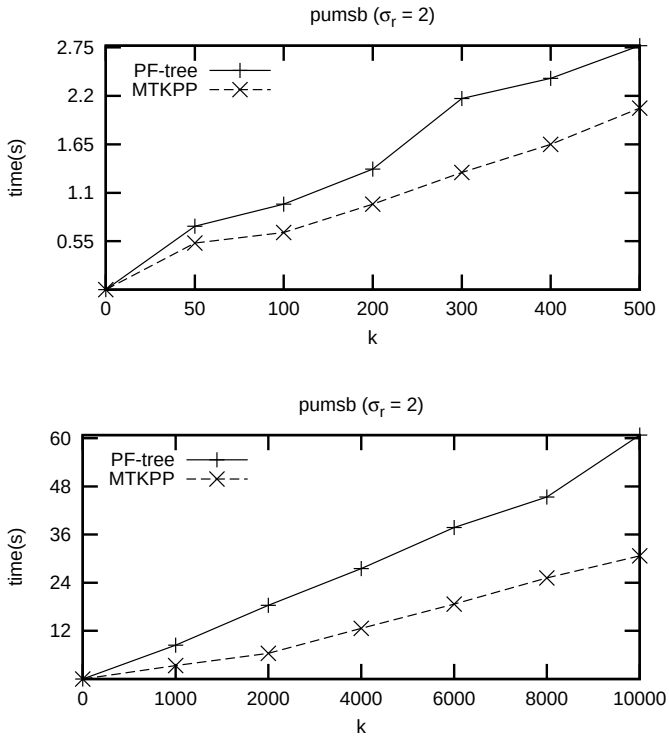


Figure 3.16: Runtime of MTKPP on *pumsb* ($\sigma_r = 2\%$)

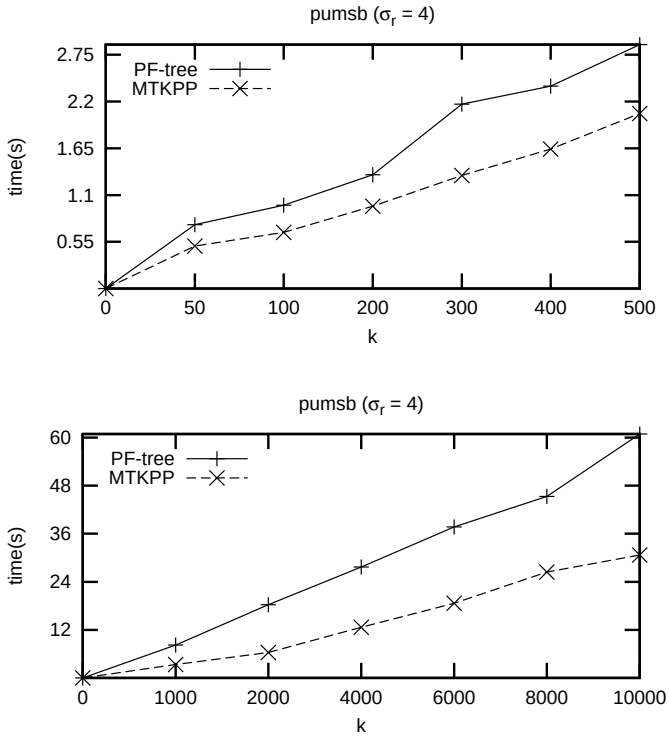


Figure 3.17: Runtime of MTKPP on *pumsb* ($\sigma_r = 4\%$)

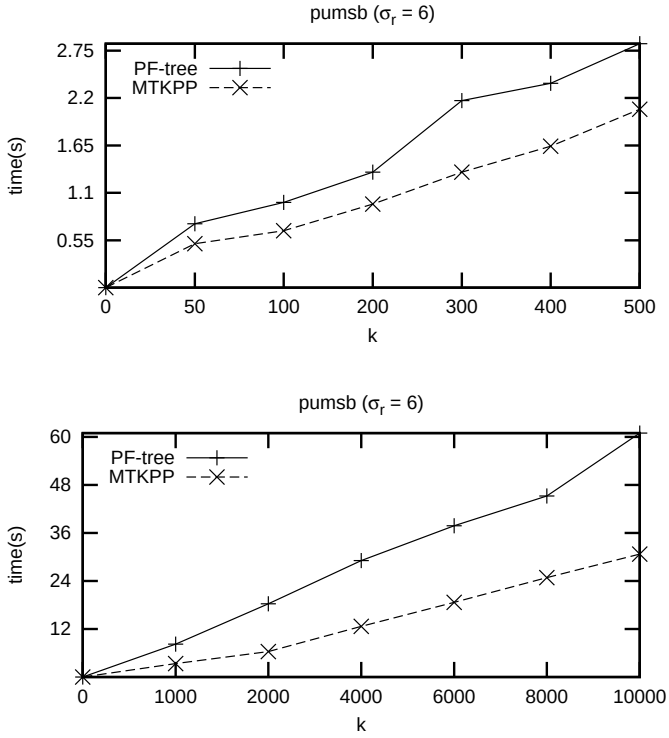


Figure 3.18: Runtime of MTKPP on *pumsb* ($\sigma_r = 6\%$)

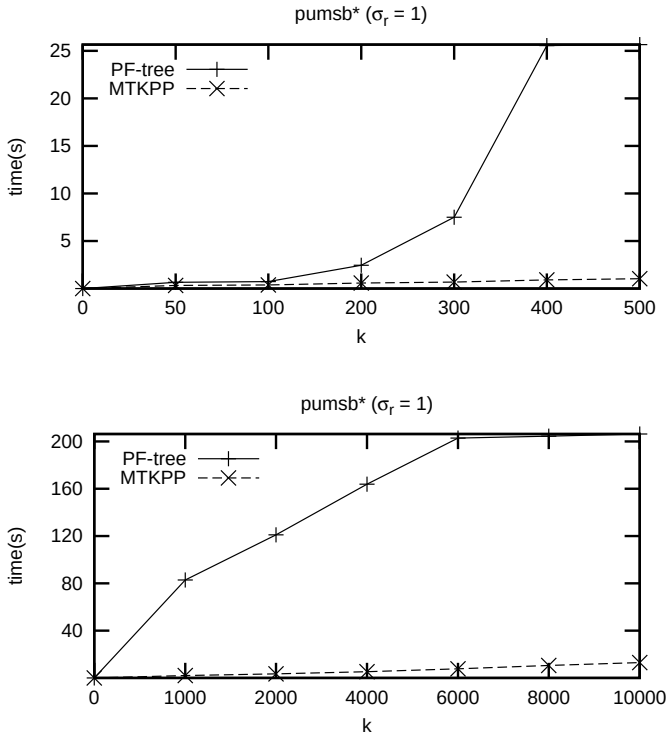


Figure 3.19: Runtime of MTKPP on *pumsb** ($\sigma_r = 1\%$)

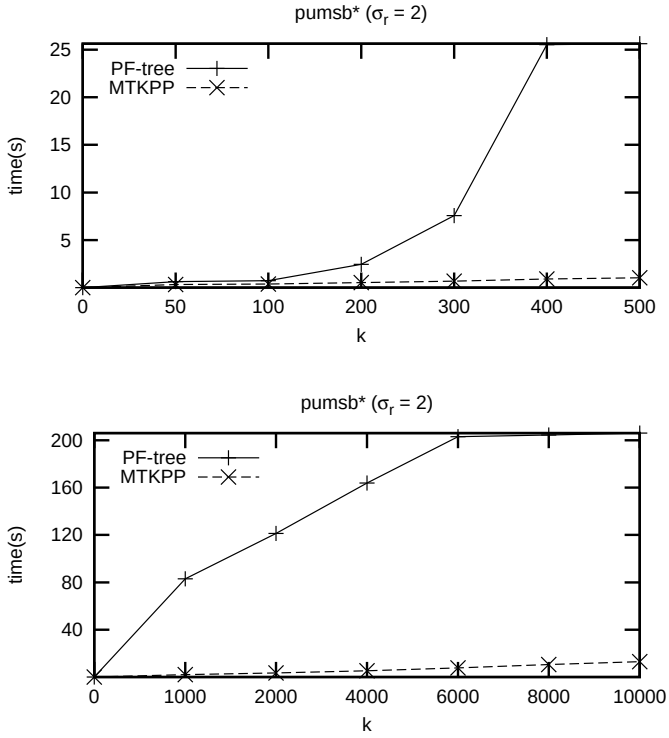


Figure 3.20: Runtime of MTKPP on *pumsb** ($\sigma_r = 2\%$)

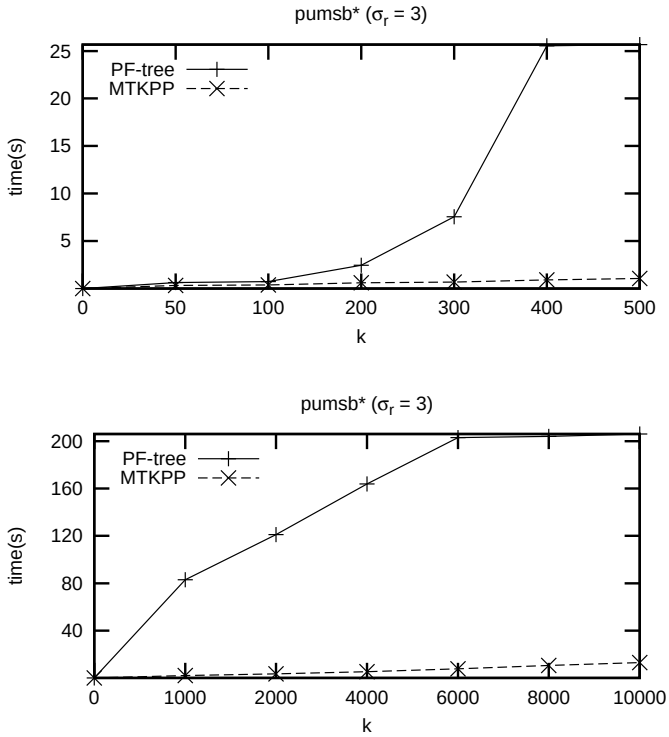


Figure 3.21: Runtime of MTKPP on *pumsb** ($\sigma_r = 3\%$)

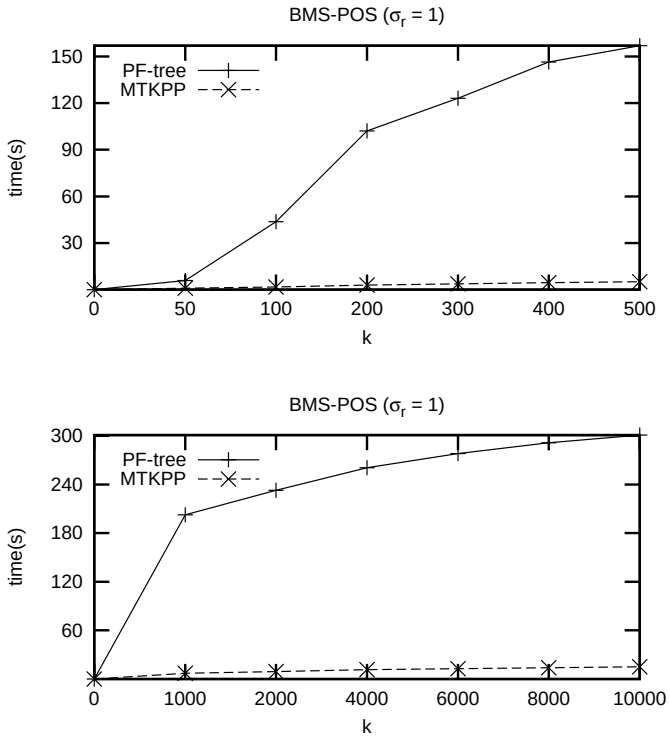


Figure 3.22: Runtime of MTKPP on *BMS-POS* ($\sigma_r = 1\%$)

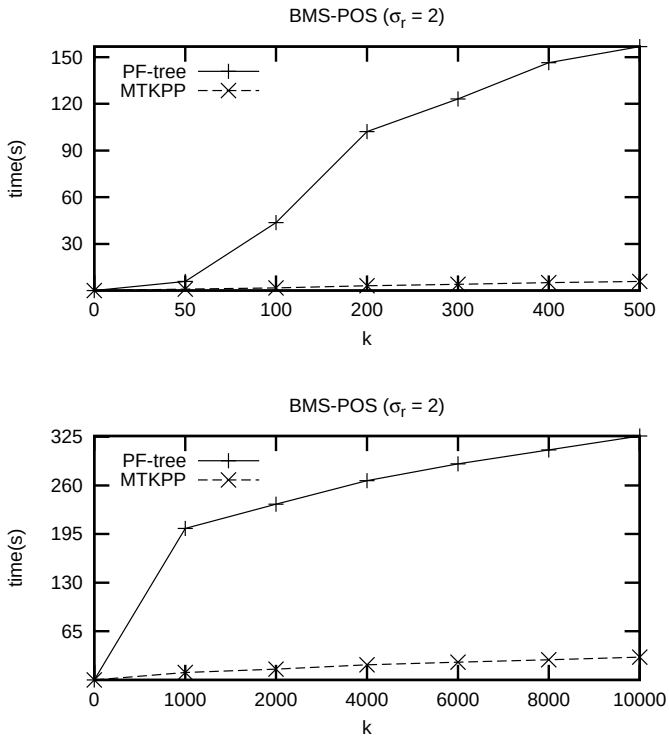


Figure 3.23: Runtime of MTKPP on *BMS-POS* ($\sigma_r = 2\%$)

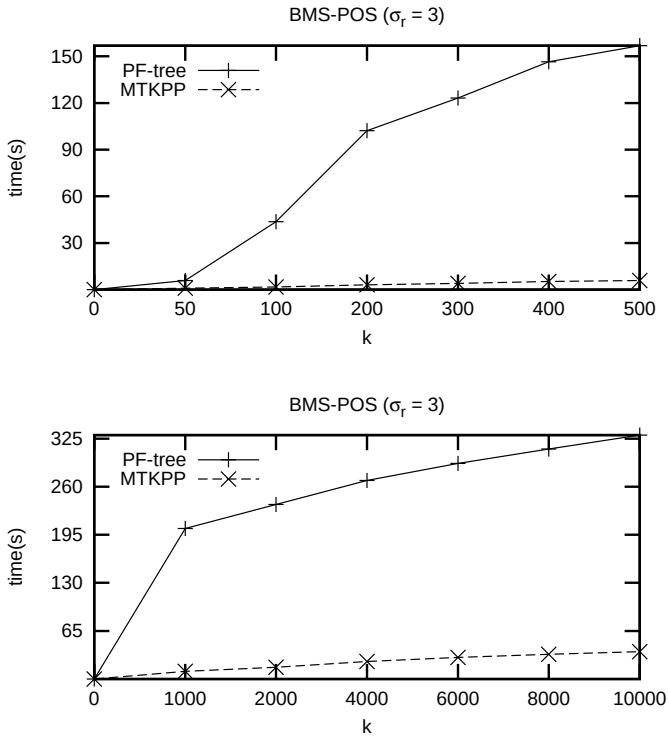


Figure 3.24: Runtime of MTKPP on *BMS-POS* ($\sigma_r = 3\%$)

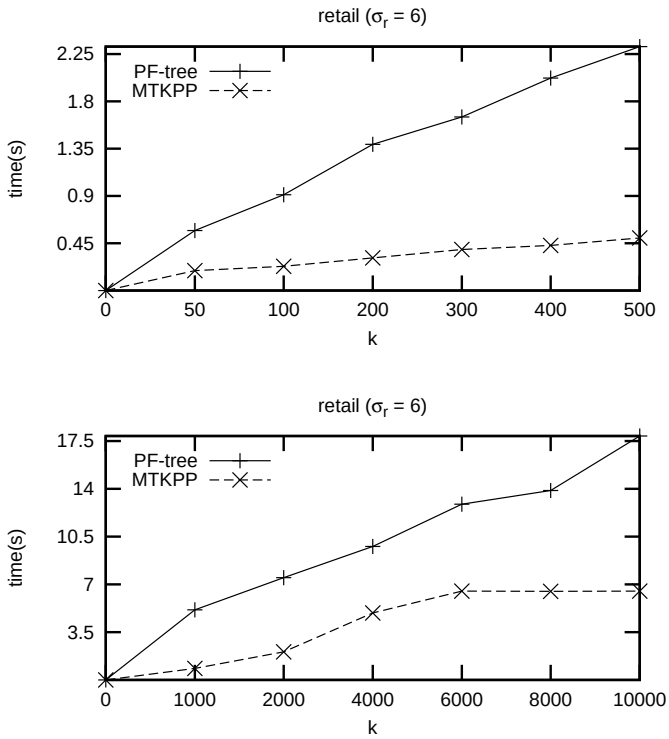


Figure 3.25: Runtime of MTKPP on *retail* ($\sigma_r = 6\%$)

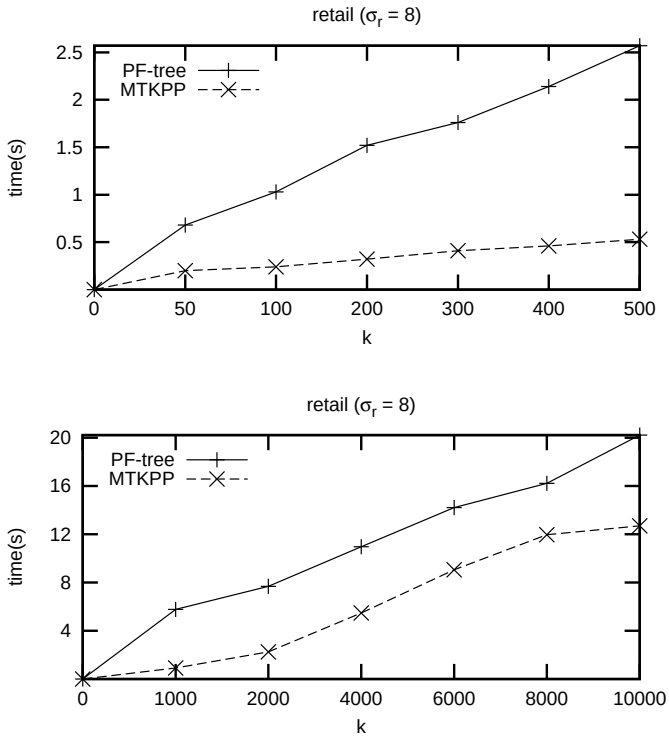


Figure 3.26: Runtime of MTKPP on *retail* ($\sigma_r = 8\%$)

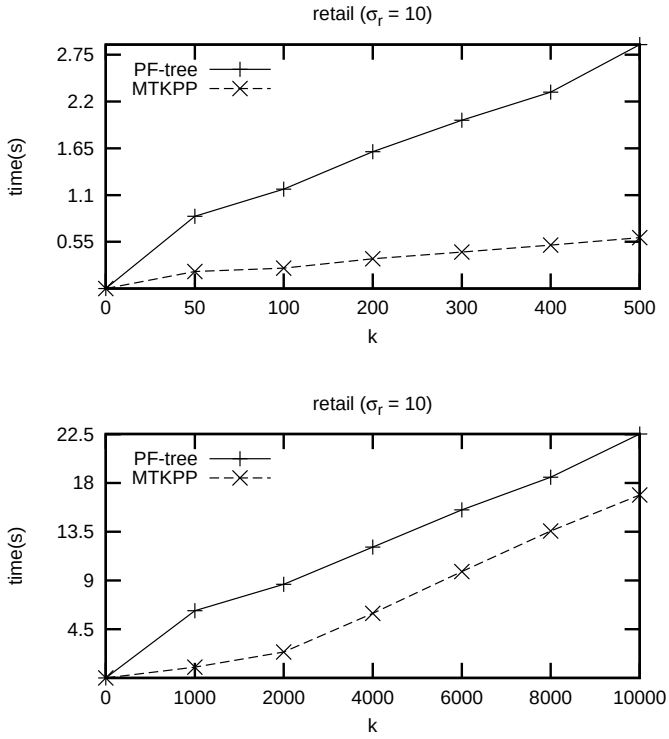


Figure 3.27: Runtime of MTKPP on *retail* ($\sigma_r = 10\%$)

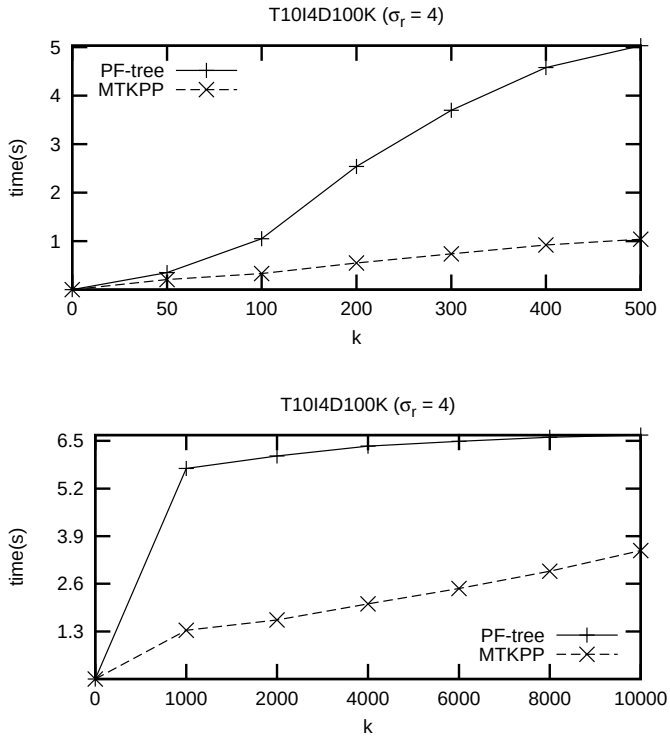


Figure 3.28: Runtime of MTKPP on *T10I4D100K* ($\sigma_r = 4\%$)

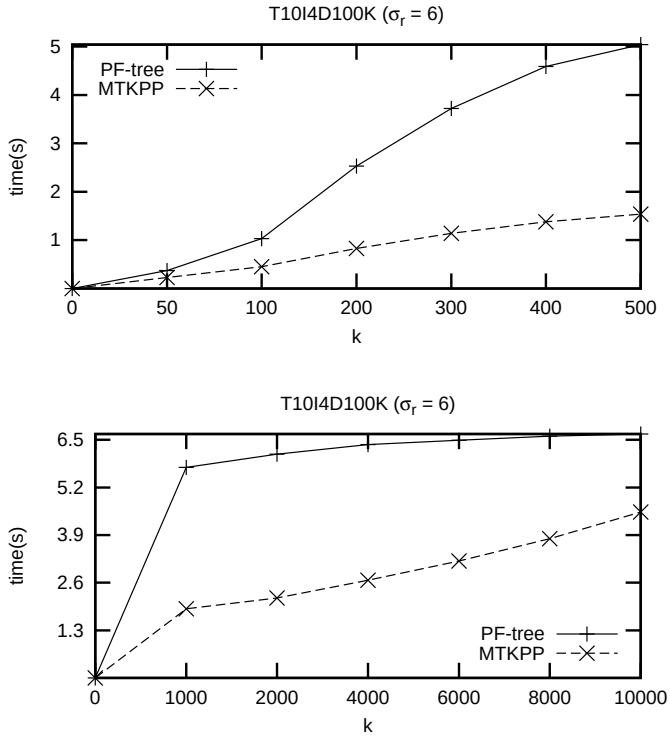


Figure 3.29: Runtime of MTKPP on *T10I4D100K* ($\sigma_r = 6\%$)

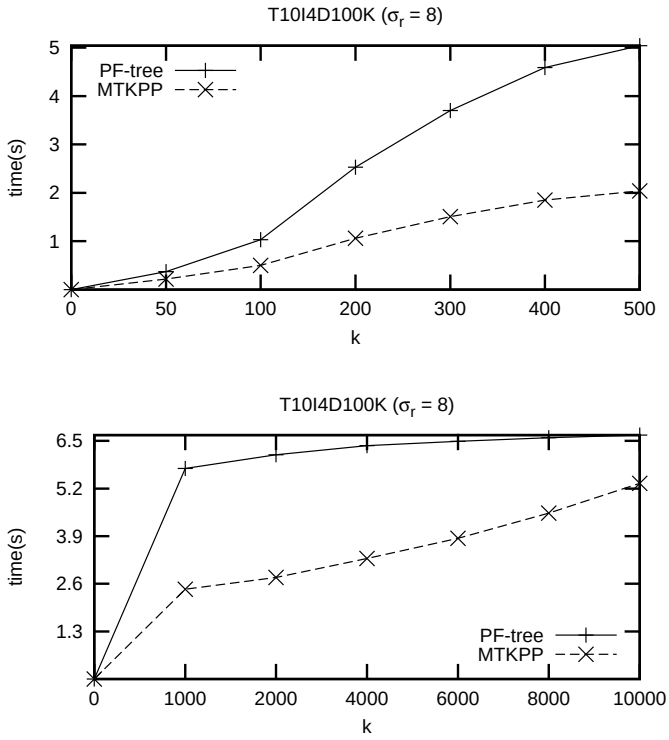


Figure 3.30: Runtime of MTKPP on *T10I4D100K* ($\sigma_r = 8\%$)

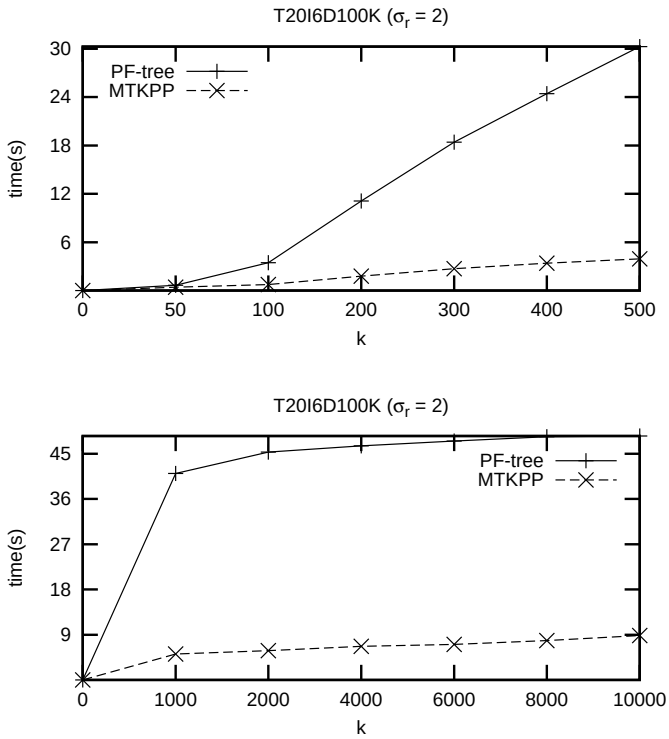


Figure 3.31: Runtime of MTKPP on *T20I6D100K* ($\sigma_r = 2\%$)

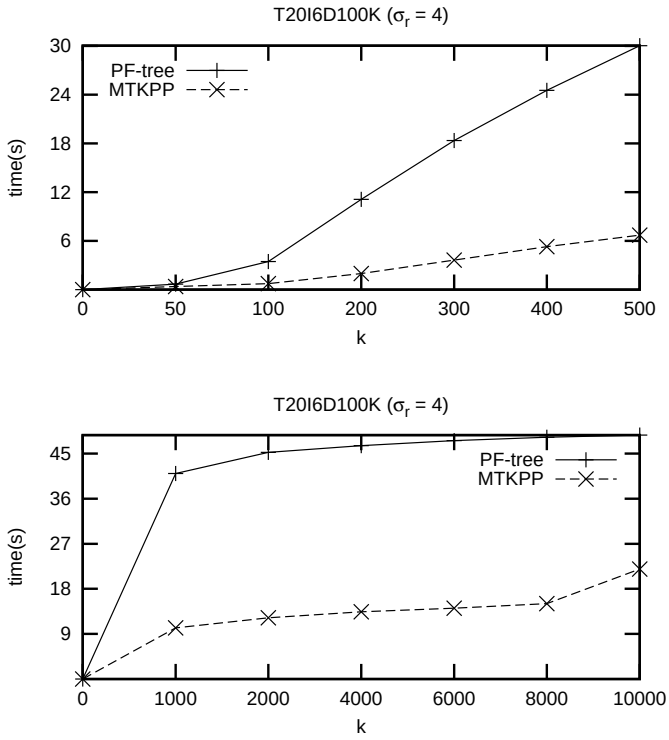


Figure 3.32: Runtime of MTKPP on *T20I6D100K* ($\sigma_r = 4\%$)

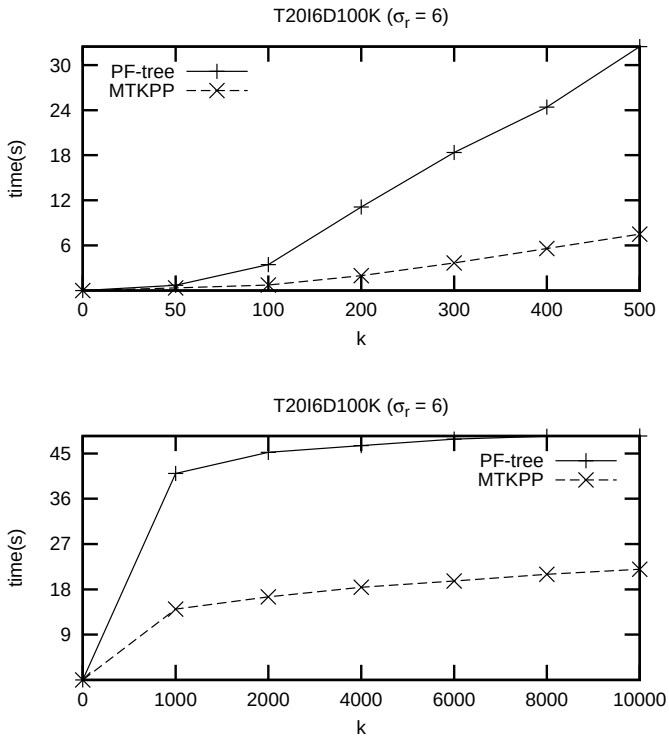


Figure 3.33: Runtime of MTKPP on *T20I6D100K* ($\sigma_r = 6\%$)

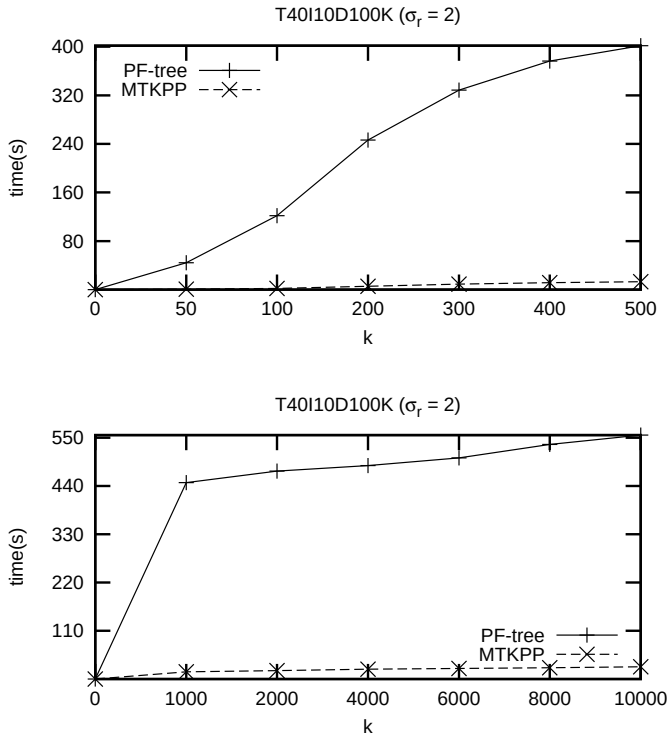


Figure 3.34: Runtime of MTKPP on *T40I10D100K* ($\sigma_r = 2\%$)

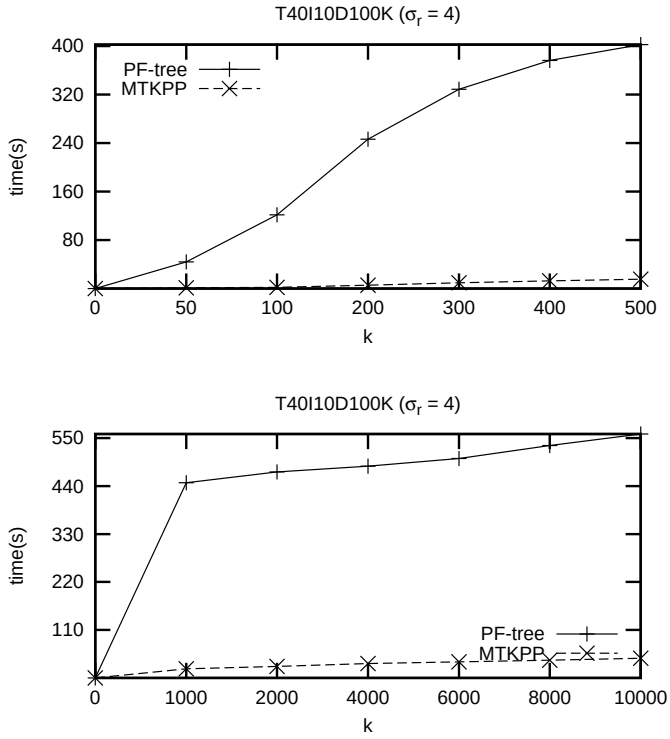


Figure 3.35: Runtime of MTKPP on *T40I10D100K* ($\sigma_r = 4\%$)

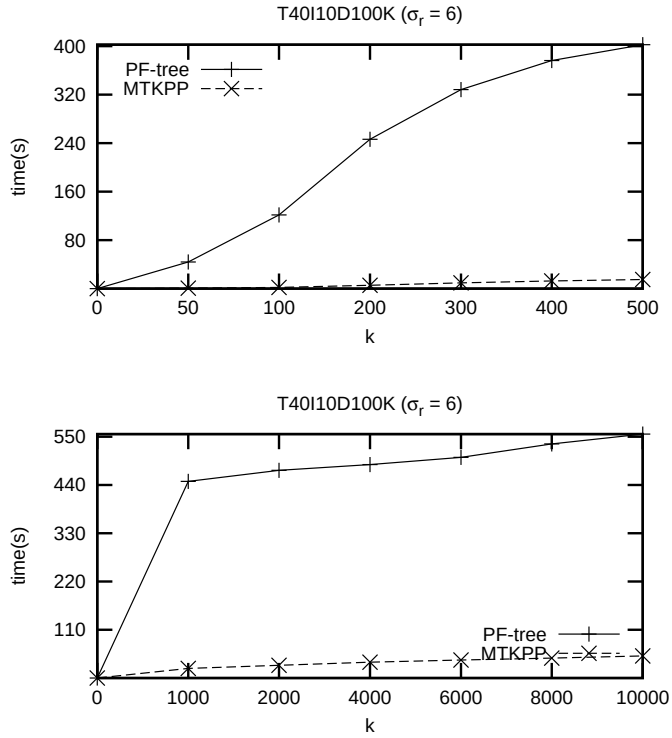


Figure 3.36: Runtime of MTKPP on *T40I10D100K* ($\sigma_r = 6\%$)

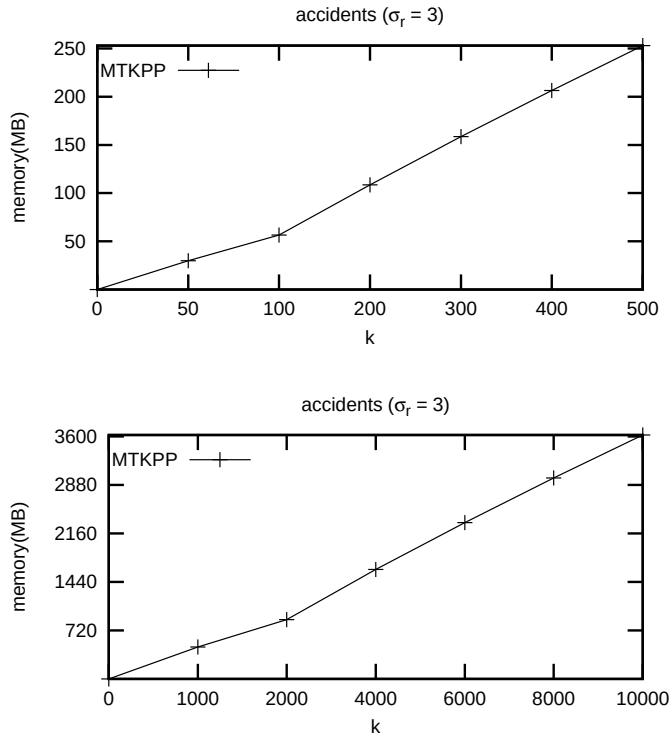


Figure 3.37: Memory usage of MTKPP on *accidents*

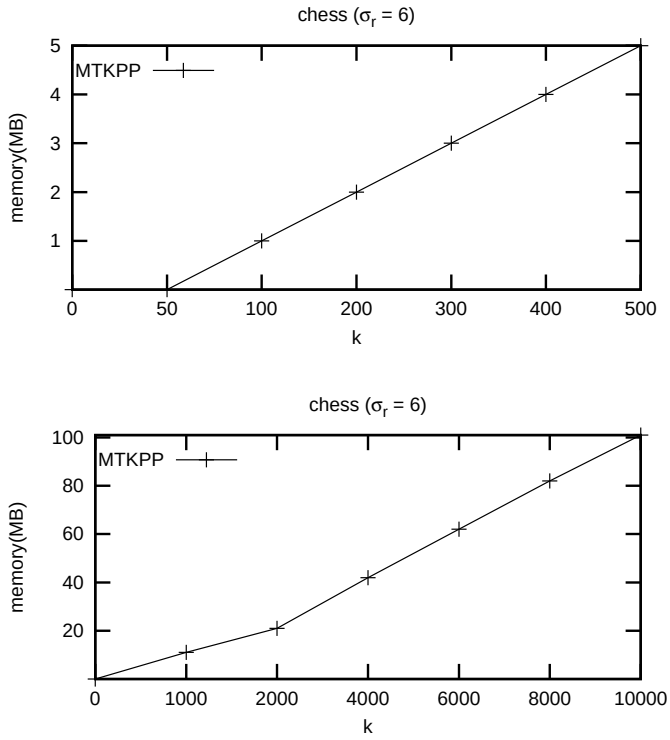


Figure 3.38: Memory usage of MTKPP on *chess*

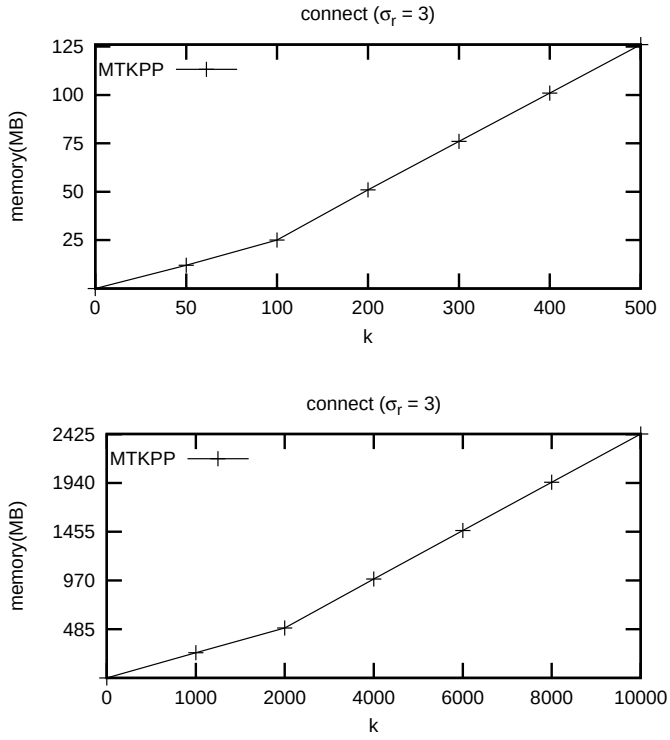


Figure 3.39: Memory usage of MTKPP on *connect*

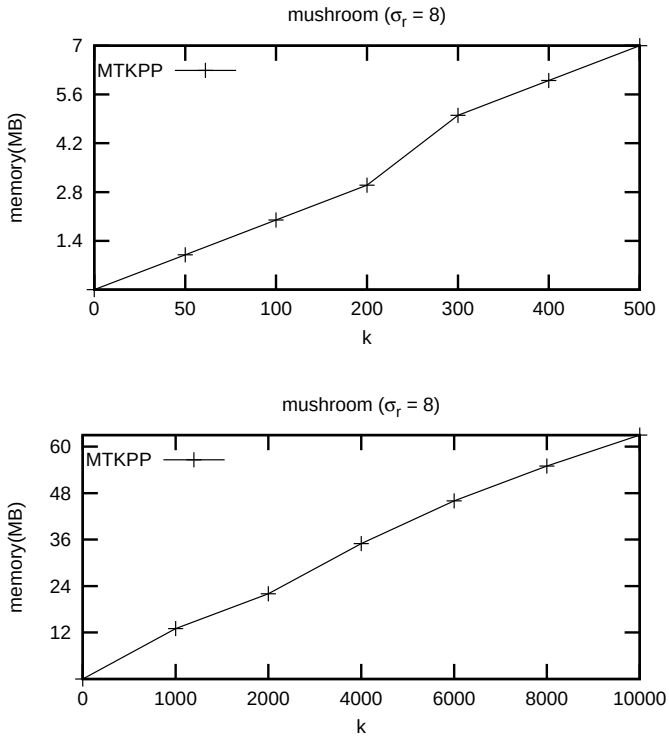


Figure 3.40: Memory usage of MTKPP on *mushroom*

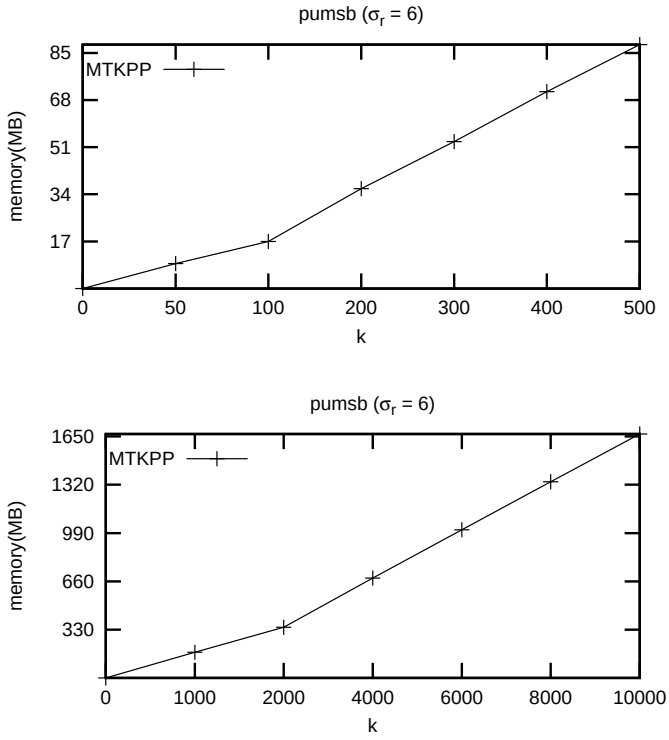


Figure 3.41: Memory usage of MTKPP on *pumsb*

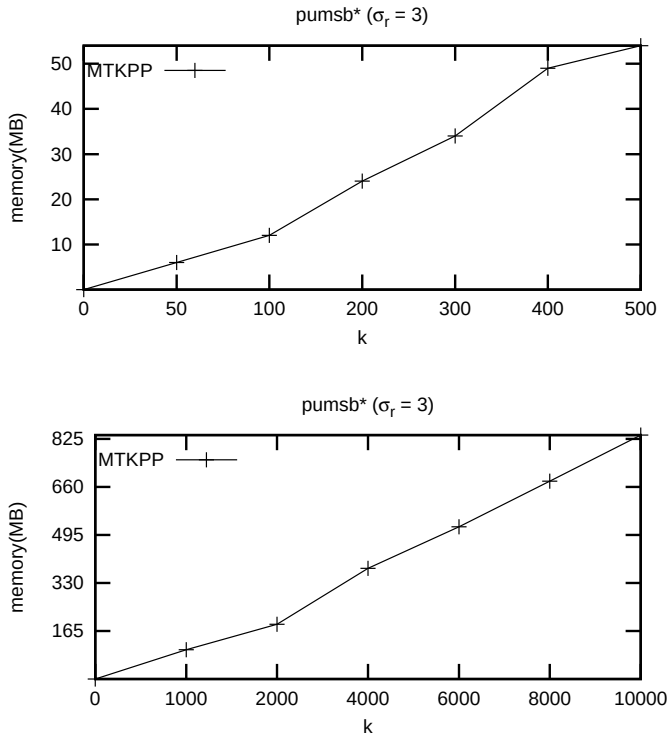


Figure 3.42: Memory usage of MTKPP on *pumsb**

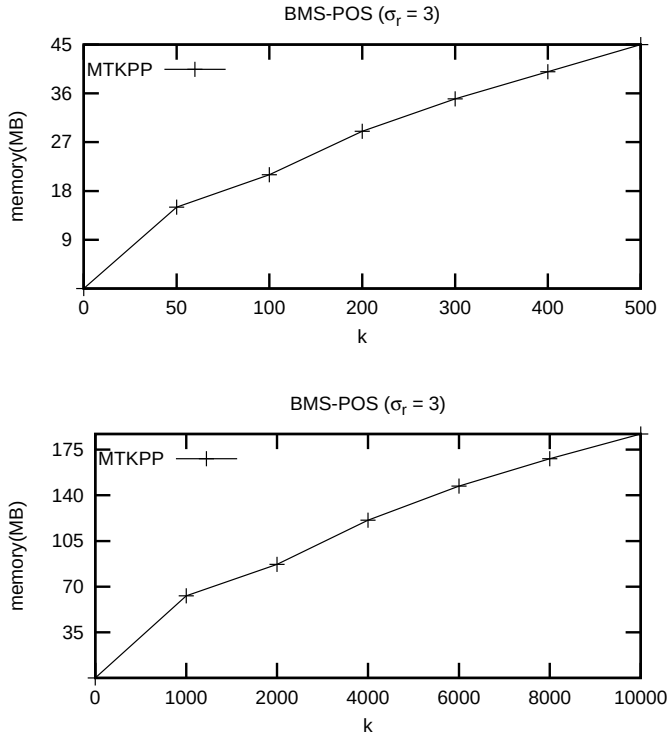


Figure 3.43: Memory usage of MTKPP on *BMS-POS*

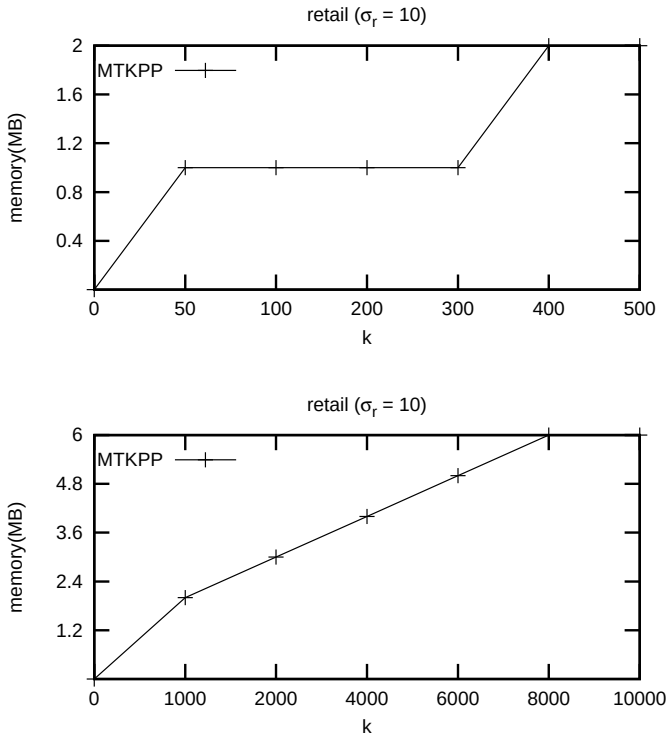


Figure 3.44: Memory usage of MTKPP on *retail*

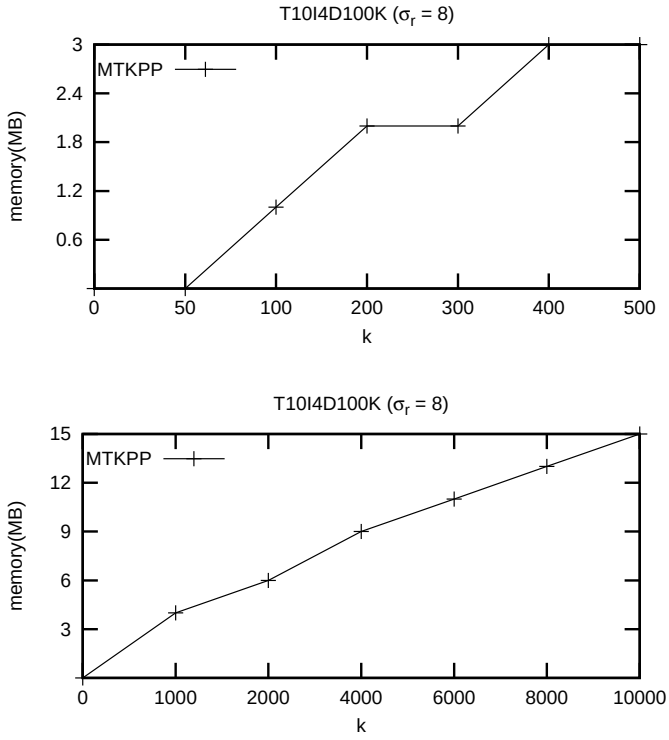


Figure 3.45: Memory usage of MTKPP on *T10I4D100K*

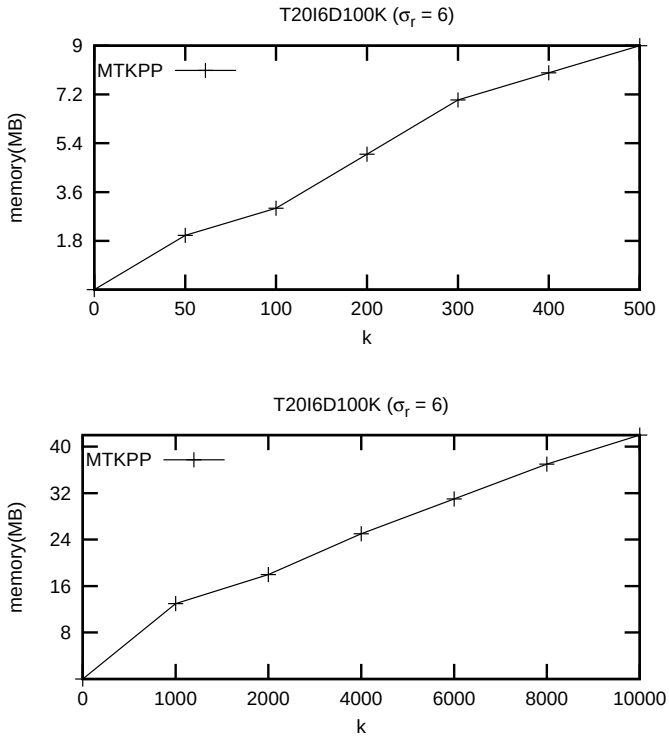


Figure 3.46: Memory usage of MTKPP on *T20I6D100K*

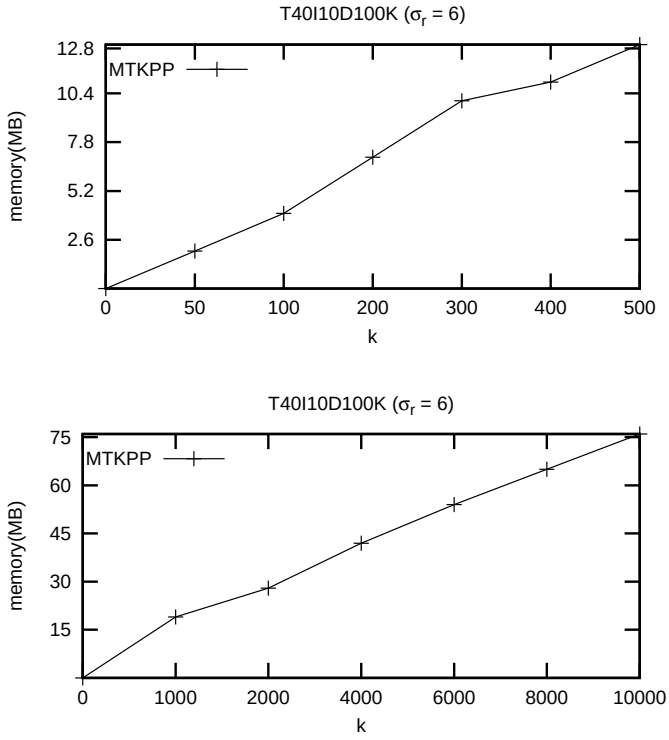


Figure 3.47: Memory usage of MTKPP on *T40I10D100K*

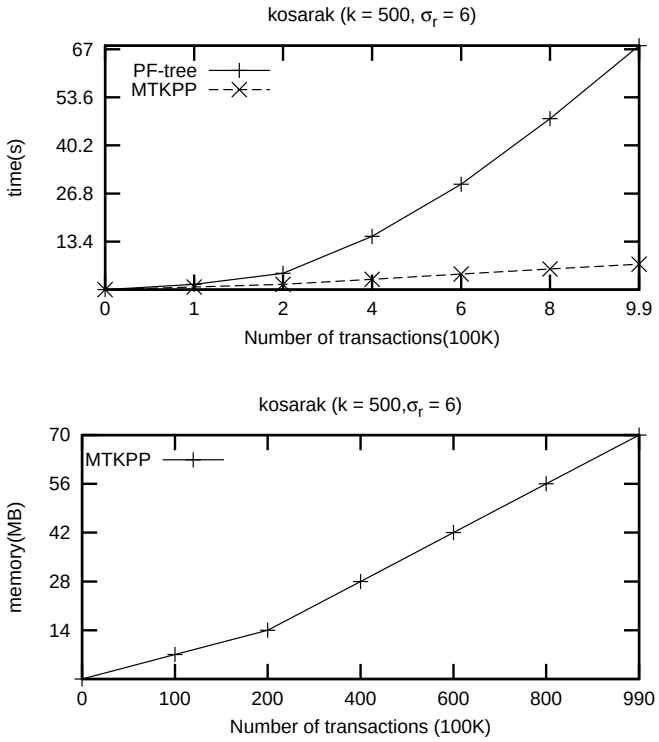


Figure 3.48: Scalability of MTKPP ($k : 500, \sigma_r = 6$)

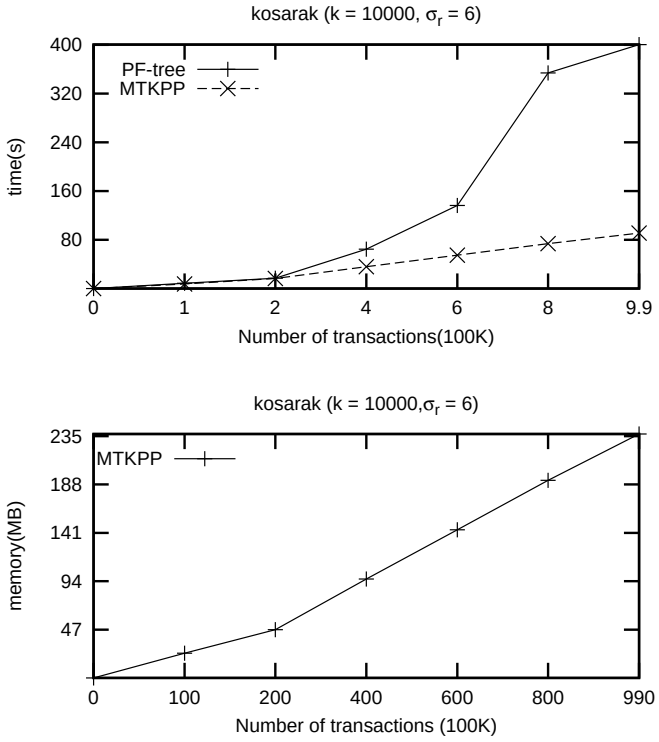


Figure 3.49: Scalability of MTKPP ($k : 10,000, \sigma_r = 6$)

CHAPTER IV

TKRIMPE: TOP-*K* REGULAR-FREQUENT ITEMSETS MINING USING DATABASE PARTITIONING AND SUPPORT ESTIMATION

As mentioned in the previous chapter, the MTKPP algorithm scans the database once to collect a set of transaction-ids (tidset) where each itemset occurs, and then MTKPP applies an intersection operation on the tidsets to collect the tidset and to calculate the support and the regularity of each larger itemset. Unfortunately, MTKPP spends a lot of time to intersect the tidsets comparing to the whole execution time.

Therefore, the aim of this chapter is to reduce the computational time on the intersection process of the MTKPP algorithm. Then, a new efficient algorithm, called *Top-K Regular-frequent Itemsets Mining using database Partitioning and support Estimation (TKRIMPE)*, to mine a set of top-*k* regular-frequent itemsets is proposed. The partition and estimation methods used to dismiss some inessential computing are also described in details. Besides, the data structure used to maintain the top-*k* regular-frequent itemsets and the complexity analysis of TKRIMPE are also discussed.

The experimental studies illustrate that TKRIMPE provides significant improvements, in particular for sparse datasets, in comparison with MTKPP on both small and large number of required results.

4.1 Preliminary of TKRIMPE

To mine the top-*k* regular-frequent itemsets, TKRIMPE employs a top-*k* list to maintain top-*k* regular-frequent itemsets during mining process. Besides, a best-first search strategy is also applied to quickly mine the itemsets with the highest supports (*i.e.* to raise up the support of the k^{th} itemset in the top-*k* list which helps to reduce the search space). Furthermore, the database partitioning technique is utilized to reduce the time to intersect tidsets. Meanwhile, the support estimation technique is used to early terminate the intersection process and to prune the set of candidates.

4.2 TKRIMPE: Top-*k* list structure

TKRIMPE is based on the use of a top-*k* list as proposed in (Amphawan et al., 2009). The top-*k* list is simply a linked-list with a hash table for efficiency reasons (two main operations - which are required to frequently access the information of itemsets- will be operated: initialization and updating the information of the top-*k* regular-frequent itemsets). At any time, the top-*k* list only contains not much more than k regular-frequent itemsets in main memory. Each entry in a top-*k* list consists of 4 fields: an item or itemset name I, a total support s^I, a regularity r^I and a set of tidsets T^I where I occurs in each partition, respectively. For example in Figure 4.1, an item a has a support of 8, a regularity of 3. Its set of tidsets is $\{\{1,4\},\{6,7,8\},\{10,11,12\}\}$ which means the item a occurs in transactions $\{t_1,t_4,t_6,t_7,t_8,t_{10},t_{11},t_{12}\}$.

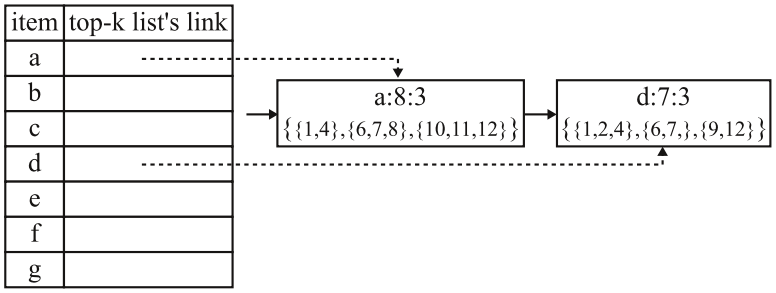


Figure 4.1: TKRIMPE: Top-*k* list with a hash table

4.3 Database Partitioning

In TKRIMPE, the database is first separated into several disjoint partitions of an equal size as presented in (Brin et al., 1997b). Then, TKRIMPE collects the tidsets (there is one tidset by partition) of each itemset in one database scan in order to calculate its support and regularity. Partitioning technique allows to reduce some unnecessary computational costs.

Given a regularity threshold σ_r, the database is split into $pn = \lceil |TDB|/\sigma_r \rceil$ partitions. Each partition will then contains σ_r transactions. For example, consider the transactional database of Table 4.1 with 12 transactions. A regularity threshold of 4 will split the database into 3 partitions of 4 transactions each.

TKRIMPE will fully exploit the partitioning of the database. Thus, a new local tidset, a local support and a local regularity related to a partition are considered. The (local) tidset of an itemset X in the m^{th} partition P_m, denoted as T_m^X, is the set of tids in m^{th} partition that contains itemset X:

$$T_m^X = \{t_q | X \subseteq t_q, t_q \in P_m\}$$

Table 4.1: A transactional database as a running example of TKRIMPE

tid	items
1	*a b d e*
2	*c d e*
3	*b c f g*
4	*a b d f g*
5	*c e g*
6	*a b c d g*
7	*a b c d*
8	*a b c e*
9	*b c d*
10	*a c e g*
11	*a b f*
12	*a b d g*

Then, the (global) tidset of an itemset X, T^X, is defined as $T^X = \{T_1^X, \ldots, T_{pn}^X\}$.The (local) support of an itemset X in the m^{th} partition, denoted s_m^X, is the number of transactions (also denoted tids) in the m^{th} partition that contains itemset X, i.e. $s_m^X = |T_m^X|$. Then, the (global) support s^X of the itemset X is equal to $\sum_{m=1}^{pn} s_m^X$.

For example, consider an item a occurring in the set of tids $\{1, 4, 6, 7, 8, 10, 11, 12\}$ (*i.e.* transactions $T^a = \{t_1, t_4, t_6, t_7, t_8, t_{10}, t_{11}, t_{12}\}$) from the transactional database of Table 4.1. Thus, the set of tids $\{1, 4\}$ is contained in T_1^a which is the tidset of the first partition. Meanwhile, the sets of tids $\{6, 7, 8\}$ and $\{10, 11, 12\}$ are stored in T_2^a and T_3^a, respectively. Thus, the tidset of the item a is $T^a = \{\{1, 4\}, \{6, 7, 8\}, \{10, 11, 12\}\}$. Besides, the support of the item a is $s^a = 2 + 3 + 3 = 8$.

By using the partition technique, the tidset of each itemset is spilt into several small tidsets. As a consequence, the original definition of the regularity of an itemset (see Definition3.1) cannot find the regularity between partitions. It is suitable on only one tidset for each itemset as in (Amphawan et al., 2009). Then, three new definitions are proposed to calculate the regularity in each partition, regularity between two consecutive partitions and the total regularity of an itemset. The effect of the partition technique is evaluated in Section 4.8.2.

Definition 4.1 (Regularity of an itemset X in a partition) *Let $t_{j,m}^X$ and $t_{k,m}^X$ be two consecutive tids in T_m^X, i.e. where $j < k$ and there is no tid $t_{i,m}^X$ in T_m^X, $j < i < k$, such that the transaction of $t_{i,m}^X$ contains X. Thus, $rtt_j^X = t_{k,m}^X - t_{j,m}^X$ is the regularity value between two consecutive tids $t_{j,m}^X$ and $t_{k,m}^X$. Therefore, the regularity of the itemset X in the m^{th} partition is defined as: $rp_m^X = max(rtt_1, rtt_2, \ldots, rtt_{|T_m^X|})$.*

Proposition 4.2 *The regularity of an itemset X in any partition P_m is strictly less than regularity threshold: $rp_m^X < \sigma_r$.*

Proof: Following Definition 4.1 it is obvious that the maximum regularity of an itemset in a partition is equal to $\sigma_r - 1$, which is the gap between the first and the last tids in the partition. ■

Definition 4.3 (Regularity of an itemset X between two consecutive partitions) *Let $t^X_{|T^X_m|,m-1}$ be the last tid where X occurs in the $(m-1)^{th}$ partition and $t^X_{1,m}$ be the first tid where X occurs in the m^{th} partition. Then, $rtp^X_m = t^X_{1,m} - t^X_{|T^X_{m-1}|,m-1}$ is the number of tids (transactions) that do not contain X between the $(m-1)^{th}$ and m^{th} partitions. Thus, a regularity of X between the two partitions is defined. Obviously, rtp^X_1 is $t^X_{1,m}$. To find the exact regularity between two consecutive partitions of X on the entire database, the number of transactions that do not contain X between the last tid where X occurs and the last tid (transaction) of database has to be calculated by: $rtp^X_{pn+1} = |TDB| - t^X_{|T^X_{pn}|,pn}$. Lastly, the regularity between any two consecutive tidsets T^X_{m-1} and T^X_m can be defined as:*

$$rtp^X_m = \begin{cases} t^X_{1,m} & \text{if } m = 1 \\ t^X_{1,m} - t^X_{|T^X_{m-1}|,m-1} & \text{if } 2 \leq m \leq pn \\ |TDB| - t^X_{|T^X_{m-1}|,m-1} & \text{if } m = pn+1 \end{cases}$$

Therefore, the regularity of an itemset is defined with the help of Definitions 4.1 and 6.8.

Definition 4.4 (Regularity of an itemset X) *The regularity of an itemset X is defined as*

$$r^X = max(max(RP^X), max(RTP^X))$$

where $RP^X = \{rp^X_1, rp^X_2, \ldots, rp^X_{pn}\}$ is the set of regularities of X in each partition (Definition 4.1) and $RTP^X = \{rtp^X_1, rtp^X_2, \ldots, rtp^X_{pn+1}\}$ is the set of regularities of X between two consecutive partitions (Definition 6.8).

For example consider the transactional database of Table 4.1 and the case of an item a: $T^a = \{\{1,4\},\{6,7,8\},\{10,11,12\}\}$. The set of regularities in each partition of the item a is $RP^a = \{(4-1), max(7-6, 8-7), max(11-10, 12-11)\} = \{3,1,1\}$. The set of regularities between two consecutive partitions of a is $RTP^a = \{1, 6-4, 10-8, 12-12\} = \{1,2,2,0\}$. Thus, the regularity of the item a is $r^a = max(max(3,1,1), max(1,2,2,0)) = 3$.

4.4 Support Estimation

The support estimation is used when the number of itemsets in the top-*k* list is equal or greater than k. The support estimation requires less computational efforts than the computing of the real support. When the estimated support of an itemset is less than the support of the k^{th} itemset in the sorted the top-*k* list, TKRIMPE can conclude that the support of the itemset is less than the support of the k^{th} element in the top-*k* list, and then TKRIMPE can prune the itemset out of a search space without intersection all of tids.

The support estimation is based on the notion of the left and right boundaries in each partition of two itemsets when these itemsets are merged. It will be also useful for regularity estimation. The left (right) boundary of itemsets X and Y in the m^{th} partition is simply the first (last) index of tids in T_m^X and T_m^Y such that the corresponding tids are equal for the two itemsets.

Formally: given the tids $t_{i,m}^X \in T_m^X$ and $t_{j,m}^Y \in T_m^Y$ ($1 \leq i \leq |T_m^X|$, $1 \leq j \leq |T_m^Y|$), the left boundaries lb_m^X and lb_m^Y of itemsets X and Y at the m^{th} partition during merging are given by:

$$lb_m^X = \begin{cases} 0 & \text{if } T_m^X \cap T_m^Y = \phi \\ min(i) & \text{if } t_{i,m}^X = t_{j,m}^Y \end{cases}$$

$$lb_m^Y = \begin{cases} 0 & \text{if } T_m^X \cap T_m^Y = \phi \\ min(j) & \text{if } t_{i,m}^X = t_{j,m}^Y \end{cases}$$

Obviously, the right boundaries are defined in a very similar way:

$$rb_m^X = \begin{cases} 0 & \text{if } T_m^X \cap T_m^Y = \phi \\ lb_m^X & \text{if } |T_m^X \cap T_m^Y| = 1 \\ max(i) & \text{if } t_{i,m}^X = t_{j,m}^Y \end{cases}$$

$$rb_m^Y = \begin{cases} 0 & \text{if } T_m^X \cap T_m^Y = \phi \\ lb_m^Y & \text{if } |T_m^X \cap T_m^Y| = 1 \\ max(j) & \text{if } t_{i,m}^X = t_{j,m}^Y \end{cases}$$

Thus, the estimated support is defined as the minimum distance between the left and the right boundaries of itemsets X and Y.

Definition 4.5 (Estimated support) *The estimated support of an itemset XY in the m^{th} partition, denoted as es_m^{XY}, is the minimum distance between the left and the right boundaries of itemsets X and Y in the m^{th} partition,* i.e.,

$$es_m^{XY} = \begin{cases} 0 & \text{if } lb_m^X = 0 \text{ or } lb_m^Y = 0 \\ 1 + min(rb_m^X - lb_m^X, rb_m^Y - lb_m^Y) & \text{otherwise} \end{cases}$$

Lemma 4.6 *The estimated support es_m^{XY} of an itemset XY in the m^{th} partition can be bounded with the help of the real support of XY in the m^{th} partition and the size of partitions:*

$$s_m^{XY} \leq es_m^{XY} \leq s_m^{XY} + ((\sigma_r - 2)/2)$$

Proof: Obviously, since the left and the right boundaries are indexes of the first and the last tids where itemsets X and Y occur together, the support of itemset XY could not be greater than the difference between the right and the left indices(*i.e.* the estimated support). Indeed, one can notice that the support of the itemset XY is equal to the estimated support of XY if XY occurs in every tid between the boundaries and that the support of XY is less than estimated support if there is at least one tid between the boundaries where the itemsets X and Y do not occur together. Thus, $s_m^{XY} \leq es_m^{XY}$.

In any partition, the maximum number of tids between the left and right boundaries of the itemset XY is $\sigma_r - 2$. This is the case when itemsets X and Y occur together in the first and the last transactions of the partition. The difference between the estimated support and the real support ($es_m^{XY} - s_m^{XY}$) corresponds to the number of tids where X and Y do not occur together between the left and the right boundaries. Then, in the worst case, this difference is equal to $(\sigma_r - 2)/2$. It happens when all the tids (between the left and the right boundaries) are totally different. Otherwise, the maximum of the difference is less than $(\sigma_r - 2)/2$. Thus $es_m^{XY} \leq s_m^{XY} + ((\sigma_r - 2)/2)$. ■

Definition 4.7 (The estimated support of an itemset XY) *The estimated support of an itemset XY, denoted es^{XY}, is the summation of estimated support in every partition, i.e.,*

$$es^{XY} = \sum_{m=1}^{pn} es_m^{XY}$$

Lemma 4.8 *Let es^{XY} be the estimated support of XY and s^{XY} be the support of XY, then $es^{XY} \geq s^{XY}$.*

Proof: Based on Lemma 4.6, in each partition, the estimated support of XY is greater than or equal to the real support of XY. Therefore, the estimated support of XY is no less than the real support because $es^{XY} = \sum_{m=1}^{PN} es_m^{XY} \geq s^{XY} = \sum_{m=1}^{pn} s_m^{XY}$. ■

Theorem 4.1 *An itemset XY is not a top-k regular-frequent itemset if $es^{XY} < s_k$, where s_k is the support of the k^{th} element in the sorted top-k list.*

Proof: Based on Lemma 4.8, the estimated support of an itemset XY is always no less than its support. If the estimated support of XY is less than s_k, then the support of XY is also less than s_k. Therefore, the itemset XY is not a top-*k* regular-frequent itemset. ■

Theorem 4.1 has clear practical implications. Indeed, for all situations where Theorem 4.1 holds, TKRIMPE can early prune the search space. The effect of this pruning strategy is evaluated in Section 4.8.2.

4.5 TKRIMPE algorithm

As MTKPP, TKRIMPE consists of two steps : (*i*) Top-*k* list initialization: partition database, scan each partition to obtain all regular items, and collect them into the top-*k* list with their supports, regularities and sets of tidsets; (*ii*) Top-*k* mining: merge each pair of entries in the top-*k* list using a best-first search strategy (*i.e.* finding the itemsets with the highest support first) and then intersect their tidsets (one by one partition) in order to find the top-*k* regular-frequent itemsets using the proposed support estimation technique.

4.5.1 TKRIMPE: Top-*k* list initialization

To create the top-*k* list, each partition of the database is scanned (one by one) to obtain all items. A new entry in the top-*k* list is created for any item that occurs in the first σ_r transactions (*i.e.* in the first partition), and then a new tidset for the first partition is built. Finally, the tidset and

the values of a support and a regularity are updated. For the following partitions, TKRIMPE first looks if the considered item is already existed in the top-*k* or not. This is done with the help of a hash function for efficiency reasons. For a first occurrence of an item in the partition, a new tidset of the partition is created and the values of support, regularity and tidset are initialized. If the item was already seen in the partition, TKRIMPE only updates its values in the top-*k* list. When the entire database has been read, the top-*k* list is trimmed by removing the items with regularity greater than σ_r. Then, the top-*k* list is sorted in descending order of support. Finally, TKRIMPE removes the items that have a support less than s_k (the support of the k^{th} item in the top-*k* list) from the top-*k* list. Details are given in Algorithm 3.

Algorithm 3 (TKRIMPE: Top-*k* list initialization)

(*1*) A transaction database: TDB
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r
Output:
(*1*) A top-*k* list

create a hash table for all 1-items
for each transaction j in the first partition **do**
  for each item i in the transaction j **do**
    if the item i does not have an entry in the top-*k* list **then**
      create a new entry for item i with $s^i = 1, r^i = t_j$ and create a tidset T_1^i that contain t_j
      create a link between the hash table and the new entry
    else
      add the support s^i by 1
      calculate the regularity r^i by t_j
      collet t_j as the last tid in T_1^i
for each partition $m = 2$ to pn **do**
  for each transaction j in the m^{th} partition **do**
    for each item i in the transaction j **do**
      if the item i has an entry in the top-*k* list **then**
        if t_j is the first tid that i occurs in the m^{th} partition **then**
          add the support s^i by 1
          calculate the regularity r^i by t_j and check r^i with σ_r
          create a tidset T_m^i and collect t_j as an element in T_m^i
        else
          add the support s^i by 1
          calculate the regularity r^i by t_j
          collect t_j as the last tid in T_m^i
for each item i in the top-*k* list **do**
  calculate the regularity r^i by $|TDB|-$ the last tid of T_{PN}^i ($t^i_{|T^i_{PN}|,PN}$)
  if $r^i > \sigma_r$ **then**
    remove the entry of i out of the top-*k* list
sort the top-*k* list by support descending order
remove all of entries after the k^{th} entry in top-*k* list

4.5.2 TKRIMPE: Top-*k* mining

As described in Algorithm 4, TKRIMPE starts from the most frequent itemset to the least frequent itemset in the top-*k* list to generate a new regular itemset with a best-first search strategy to quickly generate the regular-frequent itemsets with the highest support. It then combines two elements X and Y in the top-*k* list under the following two constraints: (*i*) the number of items in the itemsets of both elements must be equal; (*ii*) both itemsets must have the same prefix (*i.e.* each item from both itemsets is the same, except the last item). When both itemsets satisfy the two constraints, the tidsets of X and Y of each partition are sequentially intersected in order to find the regularity, the support and the tidsets of the new generated regular itemset XY. When the number of itemsets in the top-*k* list is greater than or equal to k, the estimation technique is performed in each partition (see Definition 4.5). Following Definition 4.7, the estimated support es^{XY} of the candidate itemset XY is then evaluated. If $es^{XY} < s_k$ (the support of the k^{th} itemset in the sorted top-*k* list), TKRIMPE will stop to consider the itemset XY (thanks to Theorem 4.1). Otherwise, the remaining tids between the left and the right boundaries of each partition are continuously intersected to find the (real) support and regularity. If the regularity of the new generated itemset XY is no greater than σ_r and its support is greater than s_k, then XY is inserted in the top-*k* list and the k^{th} itemset is removed from the top-*k* list. Lastly, one have to notice that thanks to the partitioning technique TKRIMPE can reduce the time to intersect some tids of each partition when at least one of the tidsets does not contains a regular sequence of transactions. This will particularly happens often in sparse datasets.

The advantages of the database partitioning and support estimation techniques will be illustrated in Section 4.8. The partitioning technique allows to reduce the number of tids to compare during intersection, and the support estimation allows to early reduce the number of candidate itemsets.

4.6 Example of TKRIMPE

Let consider the TDB presented in Table 4.1, the regularity threshold σ_r be 4 and the number of required results k be 5. The database is thus separated into three partitions.

The initialization of the top-*k* list from TDB is illustrated in Figure 4.2. After scanning the first transaction $t_1 = \{a, b, d, e\}$, the entries for items a, b, d and e are initialized in the top-*k* list as shown in Figure 4.2(a). Then the second, the third and the fourth transactions are considered. The tidsets for the first partition, the values of support and regularity of each element are initialized or updated as shown in Figure 4.2(b). The next partition (transactions 5 to 8) initializes or updates

Algorithm 4 (TKRIMPE: Top-k mining)

Input:
(*1*) A top-*k* list
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r

Output:
(*1*) A set of top-*k* regular-frequent itemsets

for each entry x in the top-*k* list **do**
  for each entry y in the top-*k* list $(x > y)$ **do**
    if the entries x and y have the same size of itemsets and the same prefix ($|I^x| = |I^y|$ and $i_1^x = i_1^y, i_2^x = i_2^y, \ldots, i_{|I^x|-1}^x = i_{|I^x|-1}^y$) **then**
      merge the itemsets of the entries x and y to be itemset $Z = I^x \cup I^y$
      $es^Z = 0, r^Z = 0, s^Z = 0$
      for each partition m **do**
        calculate the left lb_m^x, lb_m^y and the right boundaries rb_m^x, rb_m^y of the m^{th} partition
        calculate the estimated support es_m^Z from lb_m^x, lb_m^y and rb_m^x, rb_m^y
        calculate the regularity r^Z from lb_m^x, lb_m^y and rb_m^x, rb_m^y and check r^Z with σ_r
        add the estimated support es^Z by es_m^Z
      if $es^Z < s^k$ **then**
        stop considering Z $\{s^Z < s_k\}$
      for each partition m **do**
        for each t_p in T_m^X ($p = lb_m^x$ to rb_m^x) and t_q in T_m^Y ($q = lb_m^y$ to rb_m^y) **do**
          if $t_p = t_q$ **then**
            calculate the regularity r^Z by t_p
            add the support s^Z by 1
            collect t_p as the last tid in T_m^Z
        recalculate the estimated support $es^Z = es^Z - es_m^Z + |T_m^Z|$
        if $es^Z < s_k$ **then**
          stop considering Z $\{s^Z < s_k\}$
      calculate the regularity r^Z by $|TDB|-$ last tid of pn^{th} partition ($t^Z_{|T^Z_{pn}|,pn}$)
      if $r^Z \leq \sigma_r$ and $s^Z \geq s_k$ **then**
        remove the k^{th} entry from the top-*k* list
        insert the itemset Z ($I^x \cup I^y$) into the top-*k* list with r^Z, s^Z and T^Z

the tidsets for the second partition for each element as illustrated in Figure 4.2(c). Finally, the third partition is considered and the top-*k* list after scanning all transactions is given in Figure 4.2(d). Then, the item f which has the regularity $r^f = 7$ greater than $\sigma_r = 4$ is removed from the top-*k* list. The top-*k* list is sorted by support descending order and item e is removed, since the support of e ($s^e = 5$) is less than the support of g ($s^g = 6$) which is the k^{th} (5^{th}) item in the top-*k* list. The top-*k* list after initialization is shown in Figure 4.2(e). It will be the starting point for the mining process.

Since the item b is the first item in the top-*k* list, TKRIMPE starts by considering the item a and then looks in the previous items which have the same size and same prefix. Thus, the item b is combined with the item a and their tidsets are intersected (partition by partition). Since the number of itemsets in the top-*k* list is greater or equal to $k = 5$, TKRIMPE determines the estimated support of ba. The left and the right boundaries of b and a in the first partition are

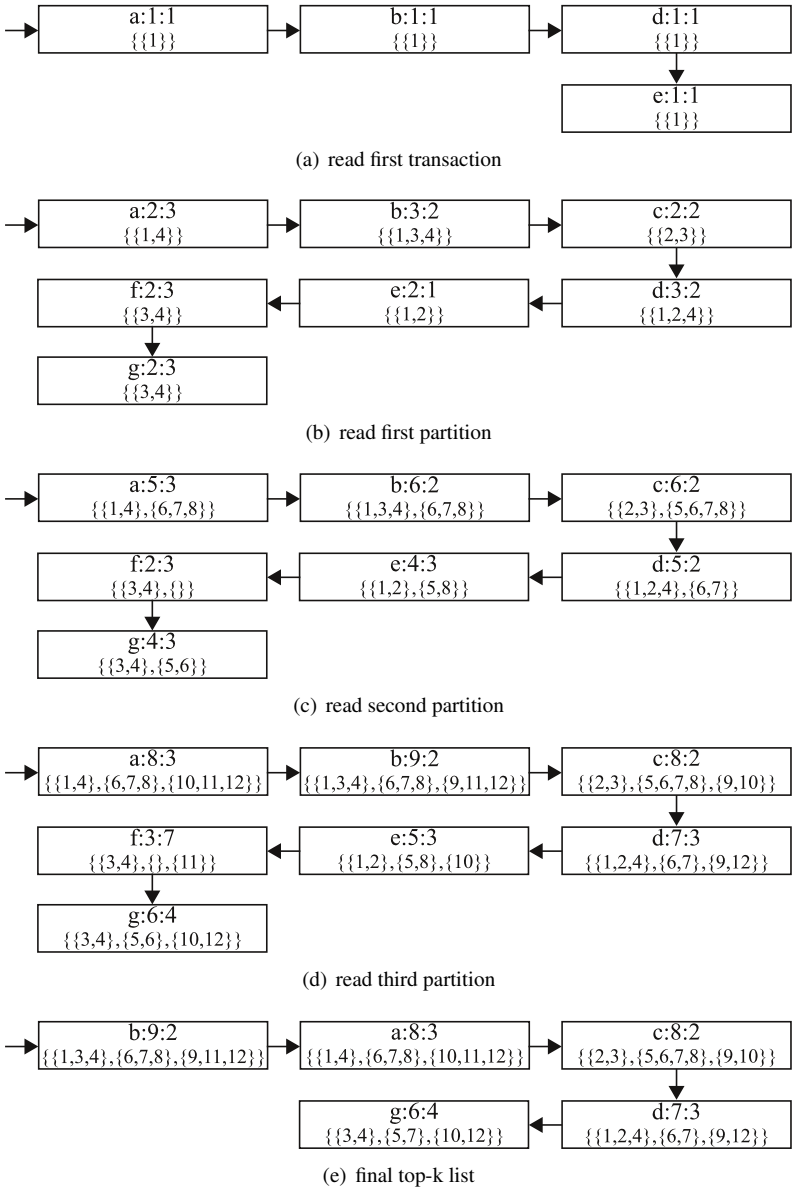


Figure 4.2: Top-*k* list initialization

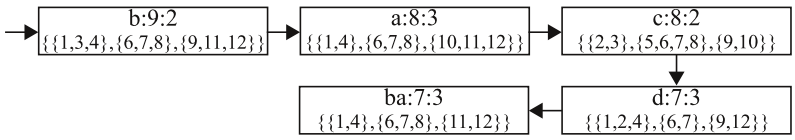


Figure 4.3: Top-*k* frequent itemsets

$lb_1^b = 1, lb_1^a = 1, rb_1^b = 3$ and $rb_1^a = 2$, respectively. Therefore, the estimated support es_1^{ba} of the first partition is $1 + min(3-1, 2-1) = 2$. The estimated supports of the second and the third partition are obtained in the same manner: $es_2^{ab} = 3$ and $es_3^{ab} = 2$. Finally, the estimated support es^{ba} of ba is equal to $2+3+2 = 7$. The itemset ba is still a candidate because $es^{ab} = s_k = 7 > s_k$. Thus, TKRIMPE intersects the remaining tids between the left and the right boundaries of each

partition to discover the support, regularity and tidsets of ba: 7, 3 and $\{\{1,4\},\{6,7,8\},\{11,12\}\}$. Since the regularity of ba is less than $\sigma_r(4)$ and the support of ba is more than $s_k = 6$, itemset ba is inserted in the top-*k* list and item g (the k^{th} itemset) is removed from the top-*k* list (Fig. 4.3). Next, the third element, item c, is considered. There are two elements in the previous sequence and have the same prefix as c: b and a. The item c thus is combined with b. The itemset bc is early pruned because its estimated support $es^{bc} = 1 + 3 + 1 = 5$ is less than s_k. Next, the item d and the itemset ba are treated in the same manner. When all itemsets in the top-*k* list have been considered, all top-*k* regular-frequent itemsets are obtained as shown in Figure 4.3.

4.7 Complexity analysis

In this section, the computational complexity for TKRIMPE is discussed in the terms of time and space. Extensive experimental studies will complement this analysis in Section 4.8.

Proposition 4.9 *The time complexity for creating the top-k list is $O(nm)$ where m is the number of transactions in the database and n is the number of items occurring in the database.*

Proof: Since the proposed algorithm scans each transaction in the database once, the entry of each item that occurs in the transaction is also looked up once in order to collect the tid into tidset ($O(nm)$). The cost for sorting all (in the very worst case) the entries is $O(n \log n)$. Then, the time complexity to create the top-*k* list is formally $O(nm + n \log n)$. In fact, the number of items (n) is, for the considered applications, always less than the number of transactions(m). Thus, the time complexity to create the top-*k* list is $O(nm)$. ■

Proposition 4.10 *The time complexity for mining top-k regular-itemset is $O(mk^2)$ where m is the number of transactions in the database and k is the number of results to be mined.*

Proof: The mining process merges each itemset in the top-*k* list with only the former itemset in the top-*k* list. Then, the tidsets of the two merged itemsets are intersected. Therefore, the combination of all itemsets in the top-*k* list is $k * (k+1)/2$ and the time to intersect tidset at each step is $O(m)$. Thus, the overall time complexity of mining process is $O(mk^2)$. ■

Proposition 4.11 *The memory space required by the top-k list is $O(km)$, where m is the number of transactions in the database and k is the number of results to be mined.*

Proof: The top-*k* list contains only k itemsets during the mining process and the maximum tids contained in each element of the top-*k* list is m. Therefore, the space complexity of using top-*k* list is $O(km)$. ■

4.8 Performance Evaluation

In this section, the experimental studies are here reported to investigate the performance of the TKRIMPE algorithm over various datasets. As illustrated in previous chapter, MTKPP algorithm (Amphawan et al., 2009) outperforms the PF-tree algorithm (Tanbeer et al., 2009) for all datasets. Then, experiments are conducted to evaluate the performance of TKRIMPE by comparing with MTKPP which are the top-*k* regular-frequent itemsets mining. To investigate the effectiveness of TKRIMPE, the advantages of database partitioning and support estimation techniques used in TKRIMPE are first illustrated. the processing time (*i.e.* CPU and I/Os costs) is examined to compare the performance of the two algorithms with the small and large values of k and various values of regularity threshold (σ_r). Furthermore, a study of memory consumption of TKRIMPE is also considered because of the use of the top-*k* list structure. Lastly, the scalability of TKRIMPE on the number of transactions in the database is evaluated.

4.8.1 Experimental setup

All the experiments are performed on a Linux platform with a Intel®Xeon 2.33 GHz and with 4 GB main memory. The experiments are done on nine real datasets (accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb* and retail) and three synthetic datasets (T10I4D100K, T20I6D100K and T20I6D100K) which were described their details and characteristics in Chapter 2. Programs for MTKPP and TKRIMPE are written in C based on the use of the top-*k* list structure.

In the experiments, the value of σ_r is set depending on the characteristic of each dataset for illustrative purpose. Therefore, the value of σ_r is specified to be different values. In fact, the number of regular itemsets for each database increases with the value of the regularity threshold. On sparse datasets, each itemsets does not occur frequently thus the value of σ_r should be set to be large when the value of k is large. While, each itemset appears very often in dense dataset, a small value of σ_r should be applied. Hence, the value of k is divided into two rages: (*i*) [50,500] for the small values; and (*ii*) [1,000, 10,000] the large values, respectively.

4.8.2 Advantages of the database partitioning and the support estimation techniques applied in TKRIMPE

The advantages of applying partitioning and estimation techniques in the TKRIMPE algorithm are first investigated. To do this, the numbers of early terminated itemsets (*i.e.* the intersection processes of these itemsets are not completed) by using the estimation technique and the numbers of non-considered tids during intersection process (*i.e.* the summation of non-regarded tids for each time of intersection) are considered. This analysis is done in an absolute manner and does not depend on the implementation.

Figure 4.4 to Figure 4.14 show the numbers of itemsets that are early terminated (pruned) by using the support estimation technique.For dense dataset, TKRIMPE is not so efficient, neither for the small value nor the large values of k, where the number of early terminated itemsets are in ranges: $[10, 1, 370]$ for the small values of k and $[980, 25, 799]$ for the large values, respectively. The reason is that the support of each top-*k* regular-frequent itemset is quite close to each other. Then, TKRIMPE cannot take benefit from the estimation technique which is an over estimation method.

However, on sparse datasets (*i.e.* BMS-POS, retail, T10I4D100K, T20I6D100K and T40I10D100K) shown in Figure 4.10 to Figure 4.14, the numbers of early terminated itemsets of the small and large values of k are varied between ranges: $[0.2K, 98K]$ and $[17K, 5500K]$, respectively. Obviously, from these figures, it could be seen that the use of estimation technique achieves high number of pruned itemsets for sparse datasets because each itemset occurs very few and not together. Thus, TKRIMPE cannot use the benefit of support estimation to prune such itemsets.

To show the benefit of partitioning technique, the number of of non-regarded tids (*i.e.* the summation of non-considered tids in each iteration of the intersection process) are illustrated in Figure 4.15 to Figure 4.25 illustrate the benefit of using the partitioning and the estimation techniques which is the summation of the number of non-considered tids in each iteration of intersection process. There are between $9,000$ and $38,000,000$ non-regarded tids for sparse datasets and between 200 and $11,000,000$ non-regarded tids for dense datsets.

4.8.3 Execution time

As mentioned above, TKRIMPE can save a lot of operations on itemsets using the database partitioning and the support estimation techniques. Recall that the performance of MTKPP is always better than that of PF-tree, a comparison on total execution times between top-*k* regular-frequent itemsets mining algorithms: MTKPP and TKRIMPE is thus now only provided.

Let first consider the runtime of TKRIMPE on the six real dense datasets (*i.e.* accidents, chess, connect, mushroom, pumsb, pumsb*) as shown in Figure 4.26 to Figure 4.43. From these figures, the execution times of MTKPP and TKRIMPE are always ranked in the same order on both the small and large values of k, due to TKRIMPE can only reduce a few number of comparison among the borders of a partition (*i.e.* the number of non-regarded tids is very few for each dense dataset). However, in some cases with the small values of k, TKRIMPE is faster than MTKPP because it can take advantage from the estimation technique. On the BMS-POS, retail and three synthetic sparse datasets (see Figure 4.44 to Figure 4.58), TMRIMPE outperforms MTKPP on both the small and large values of k. For the real retail dataset, one can notice that TMRIMPE significantly outperforms MTKPP algorithm, since TKRIMPE fully takes advantage of the partition and the support estimation techniques. On synthetic datasets, TMRIMPE outperforms MTKPP for the small and large values of k. However, on T40I10D100K, TKRIMPE has similar performance as MTKPP when k is large. Since this dataset is neither sparse nor dense dataset, TKRIMPE cannot take advantage of partitioning and estimation technique for this kind of dataset.

As a whole these results illustrate that TMRIMPE is very efficient when compared with MTKPP for sparse datasets as it was suggested in the description of the Top-k mining algorithm. In addition, TMRIMPE has better, but not significant, performance for dense datasets.

4.8.4 Memory consumption

Now, the memory consumption of TKRIMPE and MTKPP algorithms are examined. Both algorithms use a top-*k* list which contains item-name, a set of tidsets, support and regularity values for each entry. Obviously, the memory usage of the two algorithms is similar. Figures 4.59 to 4.69 show the memory usage for several values of k on the dense and sparse databases.

These experiments show that the memory consumption is low enough to be able to mine classical databases within the current available gigabyte-range memory. Indeed, in both imple-

mentations the top-*k* list structure is handled in very efficiently way.

Lastly, it is obvious that the memory usage increases when k increases. In fact, the memory usage of the proposed algorithm depends on the support value of each element in the top-*k* list because the algorithm has to maintain the tidsets of all itemsets in the top-*k* list in order to compute the support and the regularity.

4.8.5 Scalability test

the scalability of the TKRIMPE algorithm is studied on execution time and memory consumption by varying the number of transactions in database. The *kosarak* dataset is used to test the scalability of TKRIMPE and compared it with MTKPP. Since the *kosarak* is a huge dataset with a large number of distinct of items $(41,270)$ and transactions $(990,002)$, the database is firstly divided into six portions (*i.e.* $100K$, $200K$, $400K$, $600K$, $800K$ and $990K$ transactions) and the value of desired itemsets (k) is specified to be 500 and $10,000$ to investigate the scalability on the small and large values, respectively. Finally, the regularity threshold is set to 6% of number of transactions in each portion for each experiment.

From Figures 4.70 and 4.71, TKRIMPE has very good linear scalability against the number of transactions in the dataset. In comparison with MTKPP, TKRIMPE not only runs faster, but it also has much better scalability in terms of database size: the slope ratio for MTKPP is higher than that for TKRIMPE. This is because TKRIMPE can take the advantage from database partitioning and support estimation techniques.

The scalability of TKRIMPE is also investigated in terms of memory. From figures 4.70 and 4.71, these two algorithms have very similar memory requirement for all datasets because they use the same representation (tidset) to maintain tids that each itemsets occurs. Once the number of transactions increases, the memory usage of TKRIMPE and MTKPP also increase. However, TKRIMPE shows stable performance of about linearly increase of the memory requirement with respect to the database size. Therefore, it can be observed from the scalability test that TKRIMPE can mine the top-*k* regular-frequent patterns over large datasets and distinct items with considerable amount of runtime and memory.

4.9 Summary

In this chapter, an efficient algorithm to mine a set of top-*k* regular-frequent itemsets, TKRIMPE, is proposed which is based on: (*i*) the best-first search strategy that allows to mine the most frequent itemsets as soon as possible and to raise quickly the k^{th} support (*i.e.* the support of the k^{th} itemset in the sorted top-*k* list) dynamically which is then used to prune the search space; (*ii*) the partitioning of the database in order to reduce the number of comparison of certain tids at the end of each partition during the intersection process and (*iii*) the support estimation technique used to prune the search space.

The performance studies on both real and synthetic datasets show that the proposed algorithm is efficient. TKRIMPE is also compared with MTKPP, which are at the moment the only one efficient algorithms for mining top-*k* regular-frequent patterns. From the results, TKRIMPE outperforms MTKPP, for small and large value of k when the dataset is sparse, and have similar performance for dense datasets.

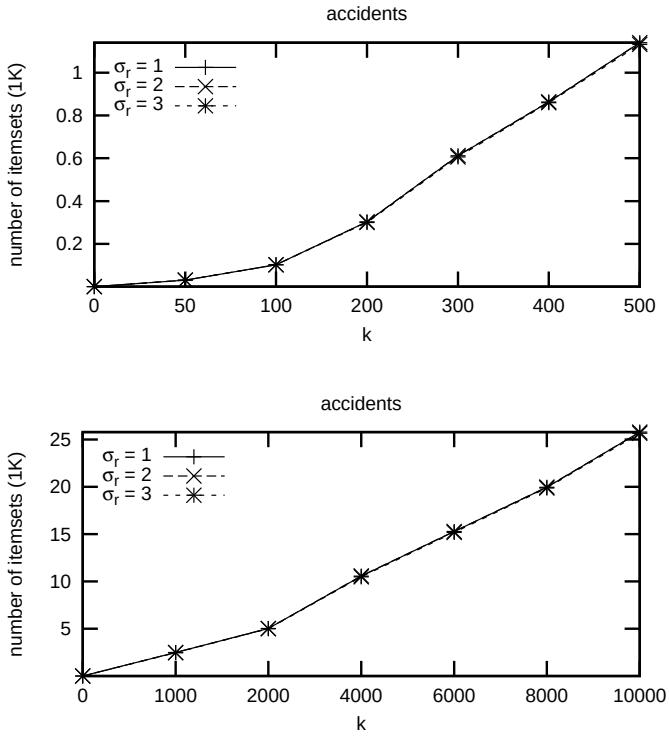


Figure 4.4: The number of early terminated itemsets on *accidents* dataset

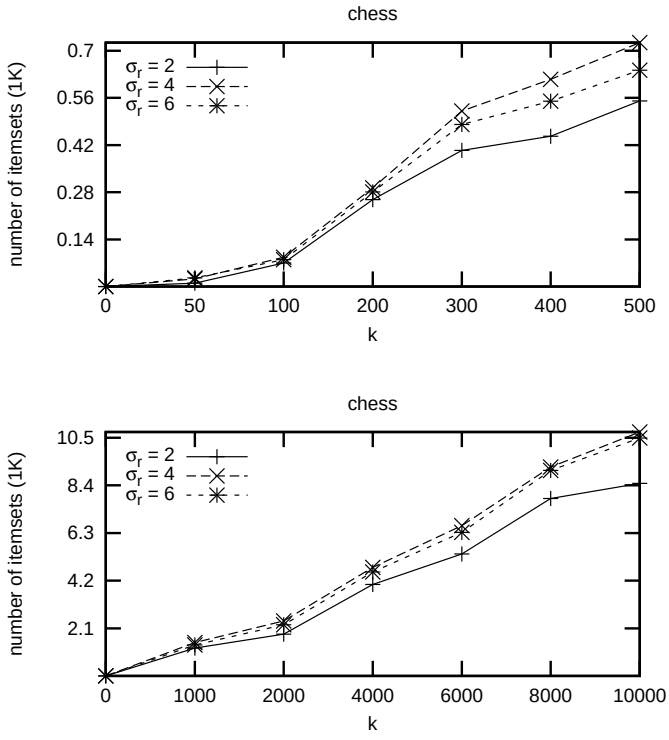


Figure 4.5: The number of early terminated itemsets on *chess* dataset

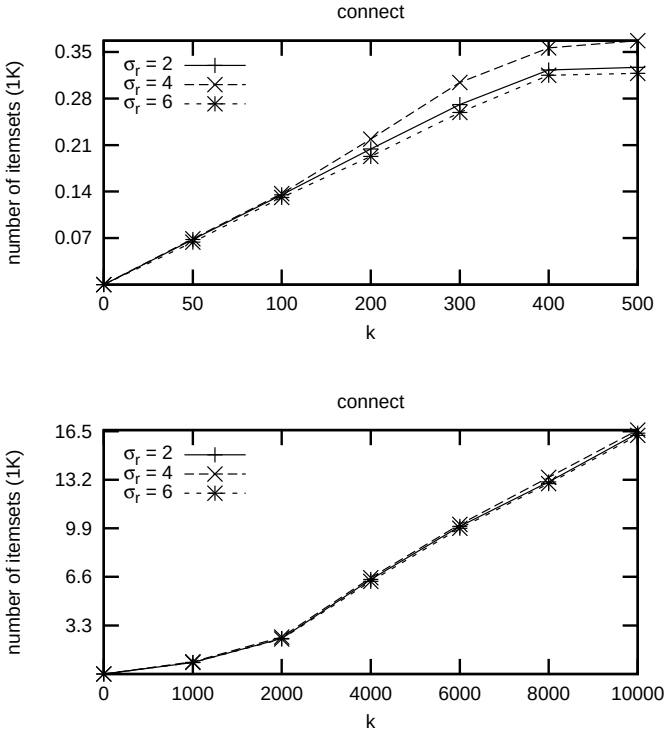


Figure 4.6: The number of early terminated itemsets on *connect* dataset

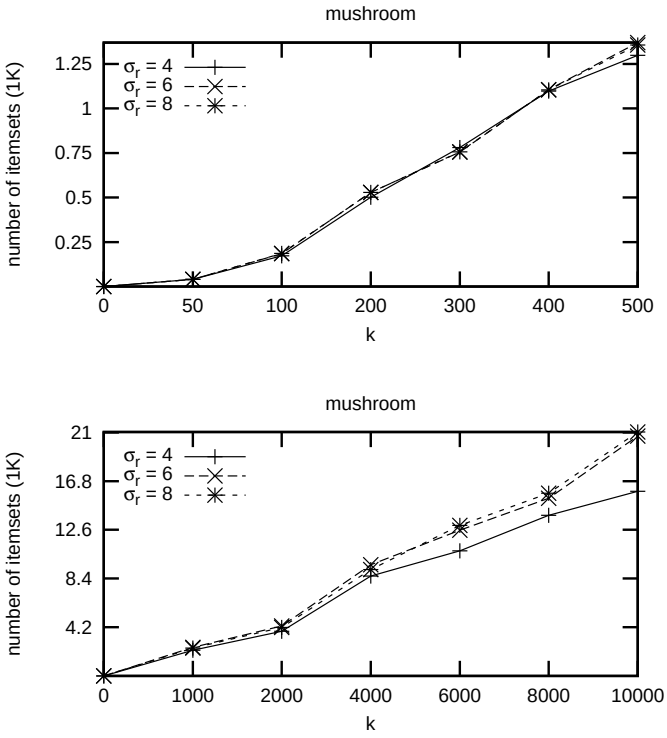


Figure 4.7: The number of early terminated itemsets on *mushroom* dataset

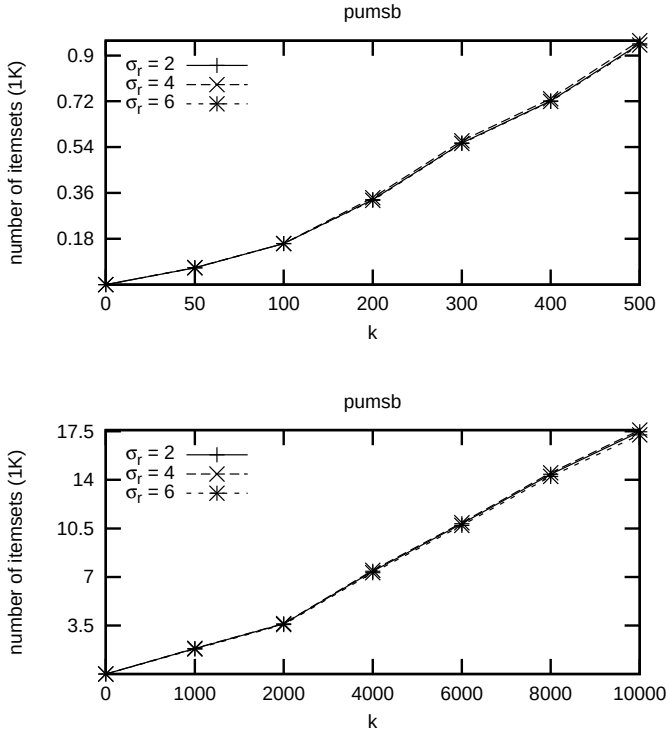


Figure 4.8: The number of early terminated itemsets on *pumsb* dataset

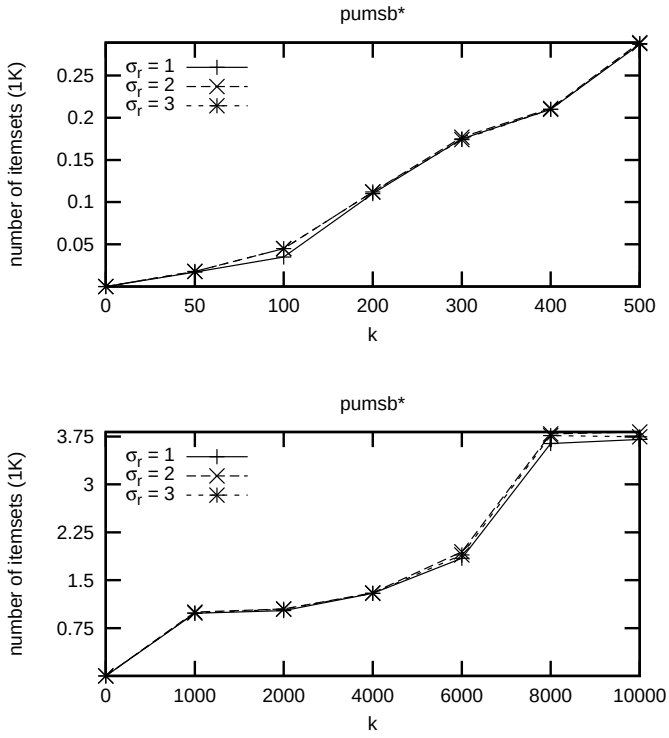


Figure 4.9: The number of early terminated itemsets on *pumsb** dataset

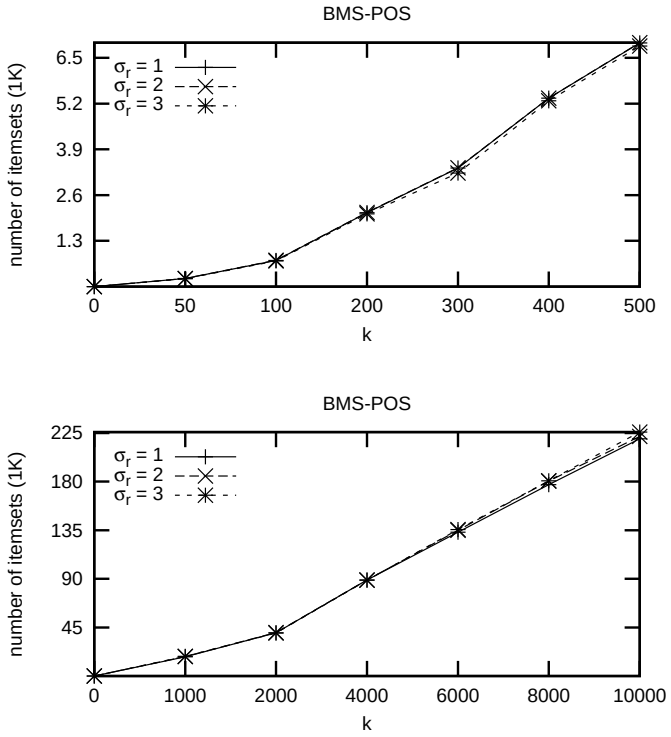


Figure 4.10: The number of early terminated itemsets on *BMS-POS* dataset

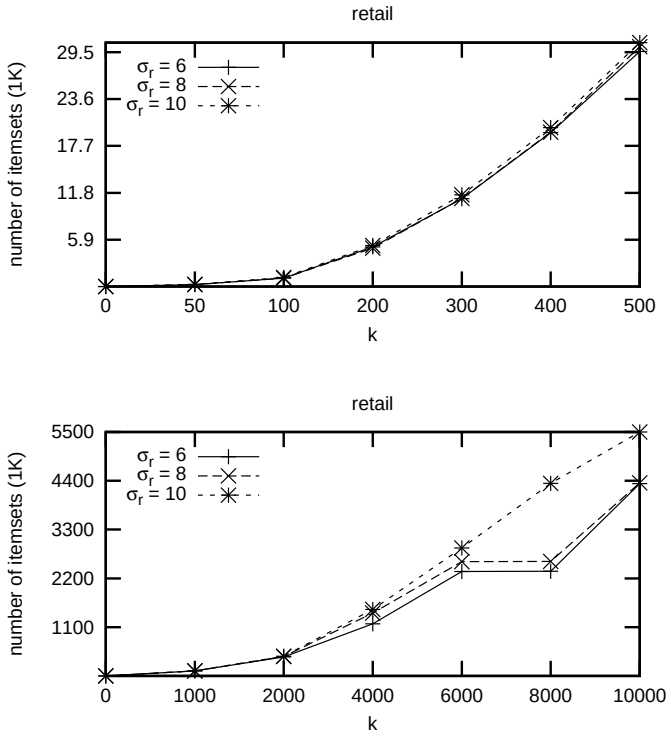


Figure 4.11: The number of early terminated itemsets on *retail* dataset

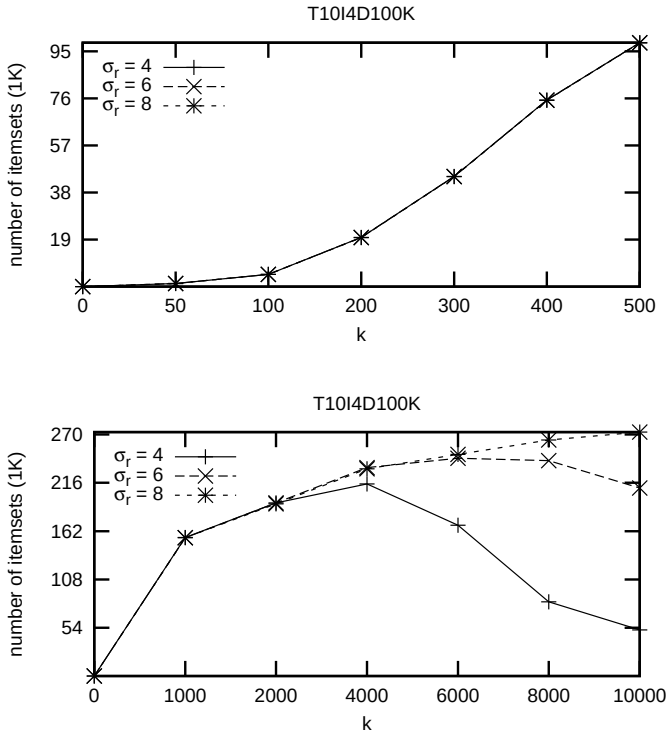


Figure 4.12: The number of early terminated itemsets on *T10I4D100K* dataset

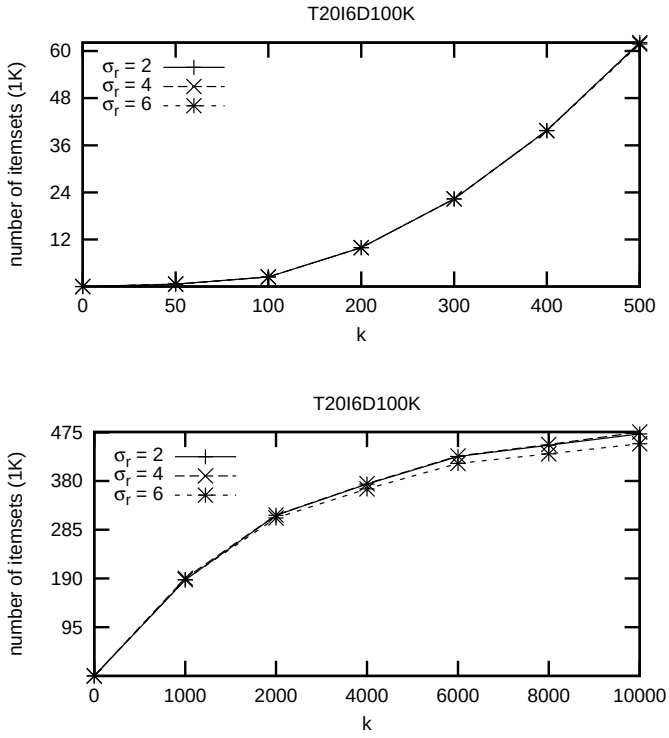


Figure 4.13: The number of early terminated itemsets on *T20I6D100K* dataset

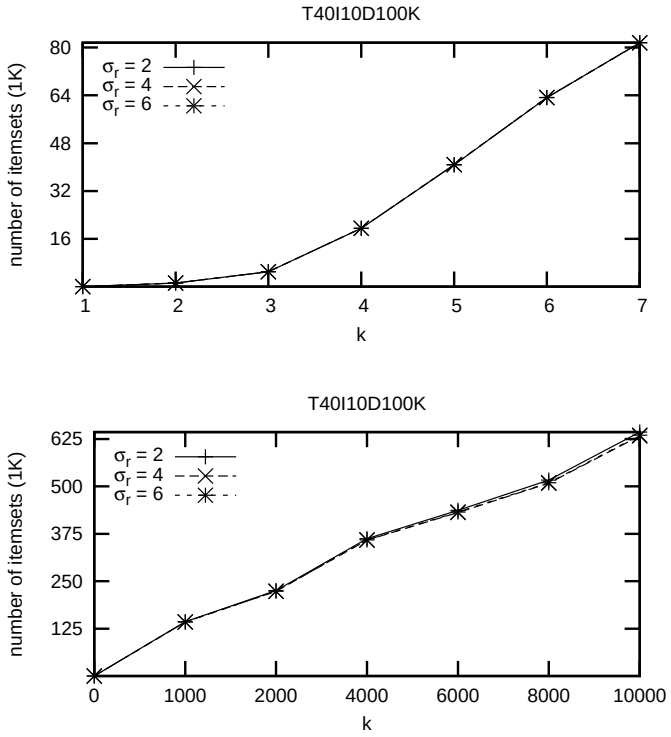


Figure 4.14: The number of early terminated itemsets on *T40I10D100K* dataset

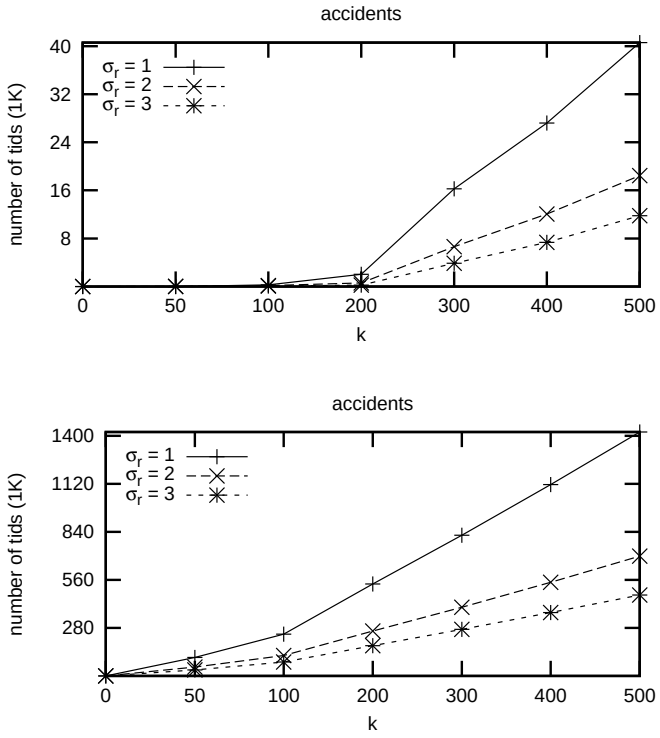


Figure 4.15: The number of non-regarded tids during intersection process on *accidents* dataset

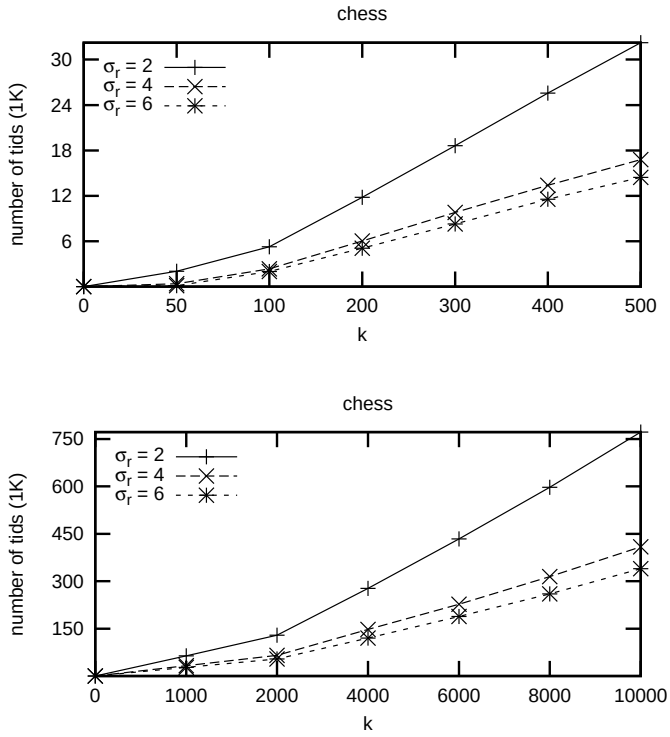


Figure 4.16: The number of non-regarded tids during intersection process on *chess* dataset

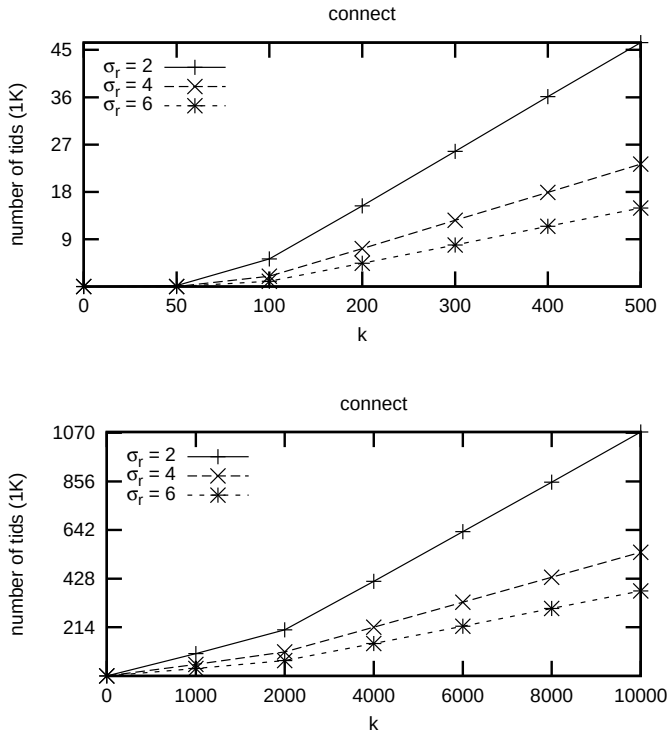


Figure 4.17: The number of non-regarded tids during intersection process on *connect* dataset

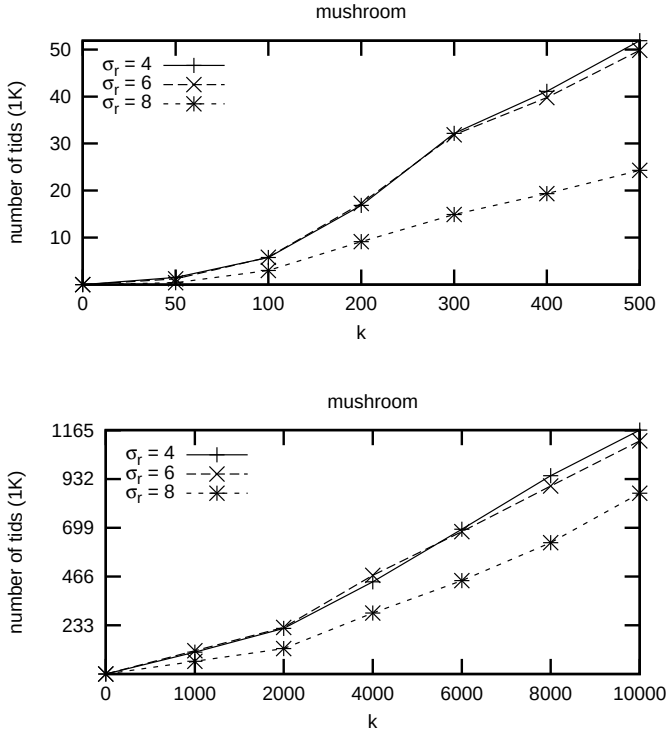


Figure 4.18: The number of non-regarded tids during intersection process on *mushroom* dataset

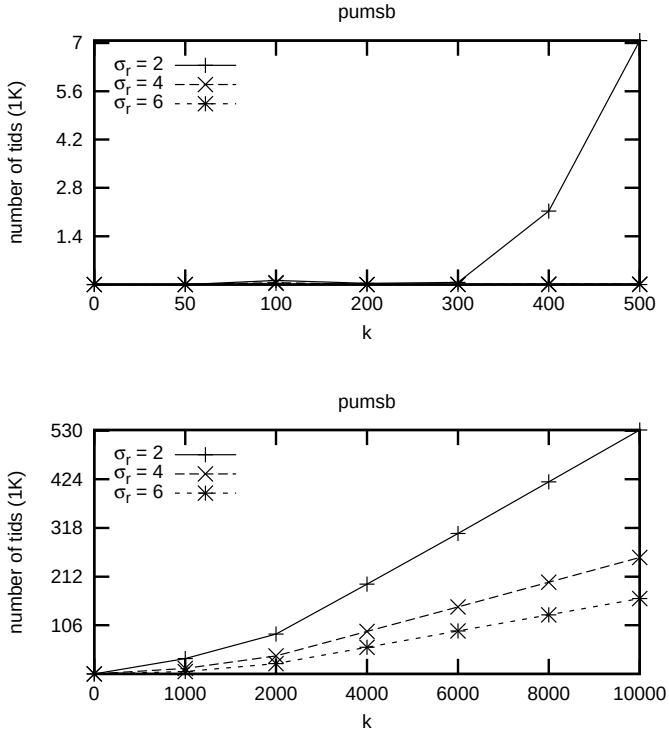


Figure 4.19: The number of non-regarded tids during intersection process on *pumsb* dataset

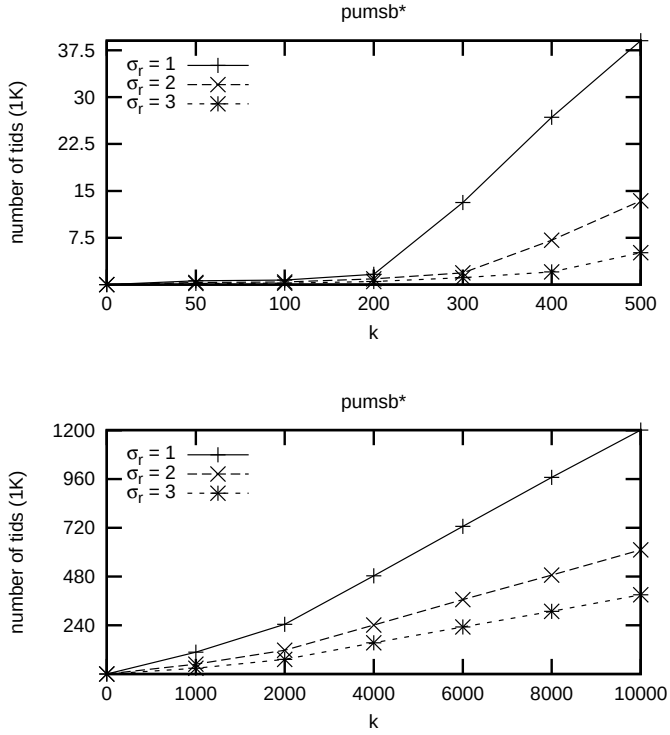


Figure 4.20: The number of non-regarded tids during intersection process on *pumsb** dataset

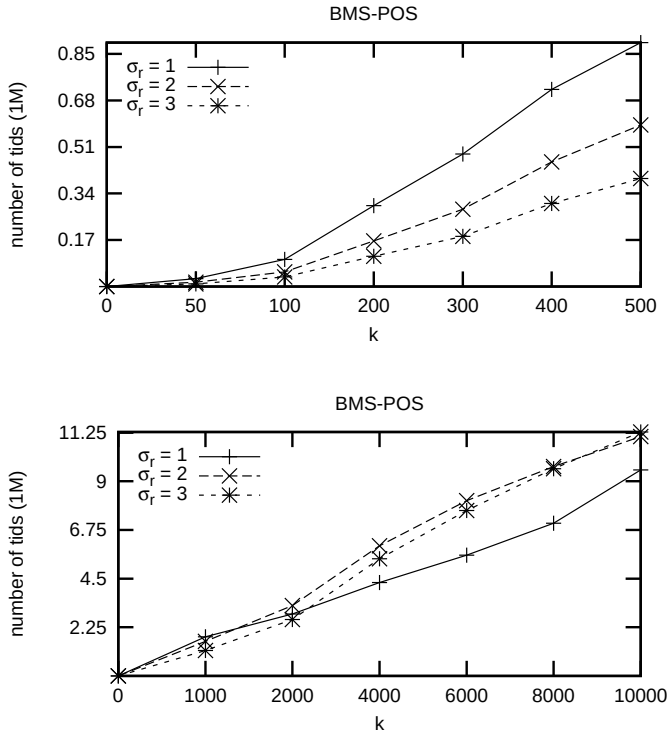


Figure 4.21: The number of non-regarded tids during intersection process on *BMS-POS* dataset

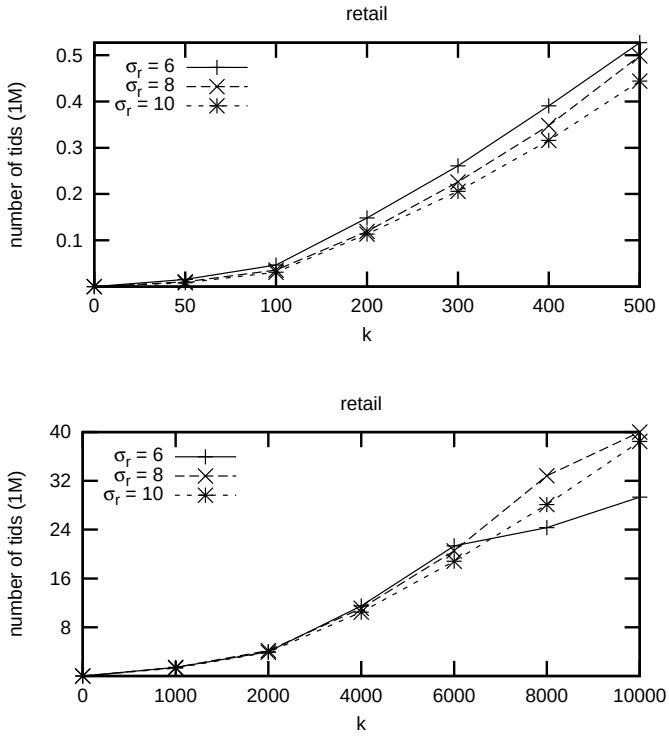


Figure 4.22: The number of non-regarded tids during intersection process on *retail* dataset

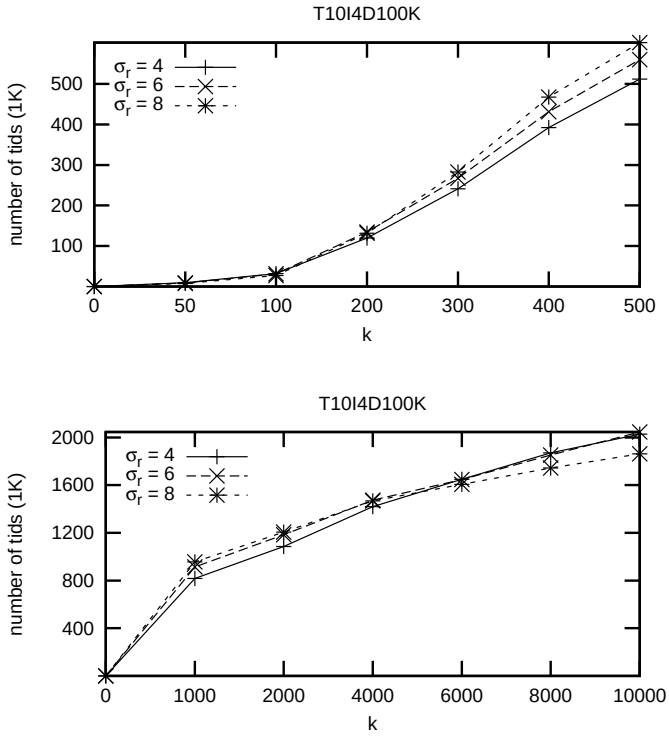


Figure 4.23: The number of non-regarded tids during intersection process on *T10I4D100K* dataset

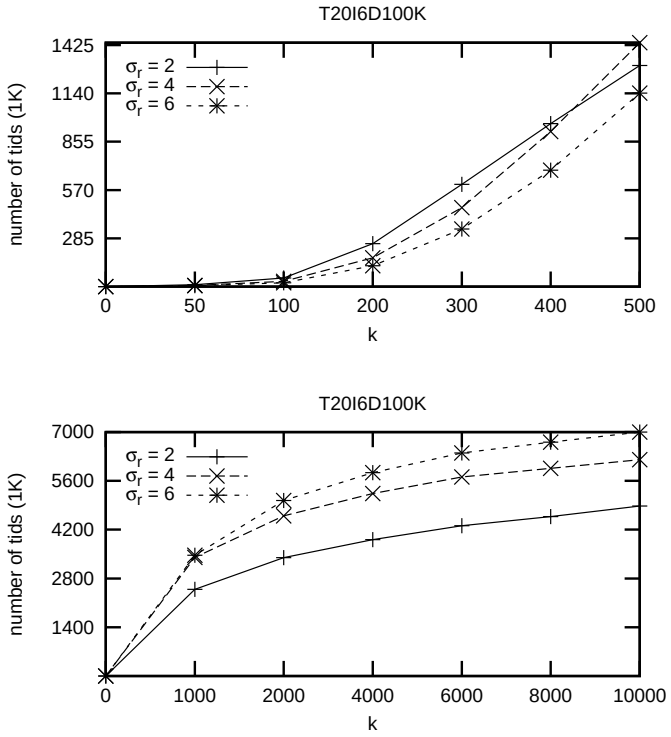


Figure 4.24: The number of non-regarded tids during intersection process on *T20I6D100K* dataset

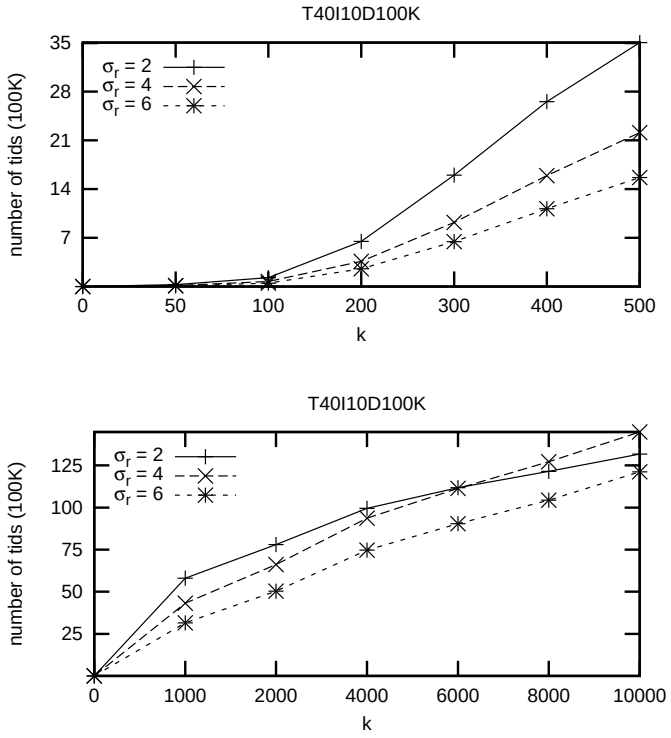


Figure 4.25: The number of non-regarded tids during intersection process on *T40I10D100K* dataset

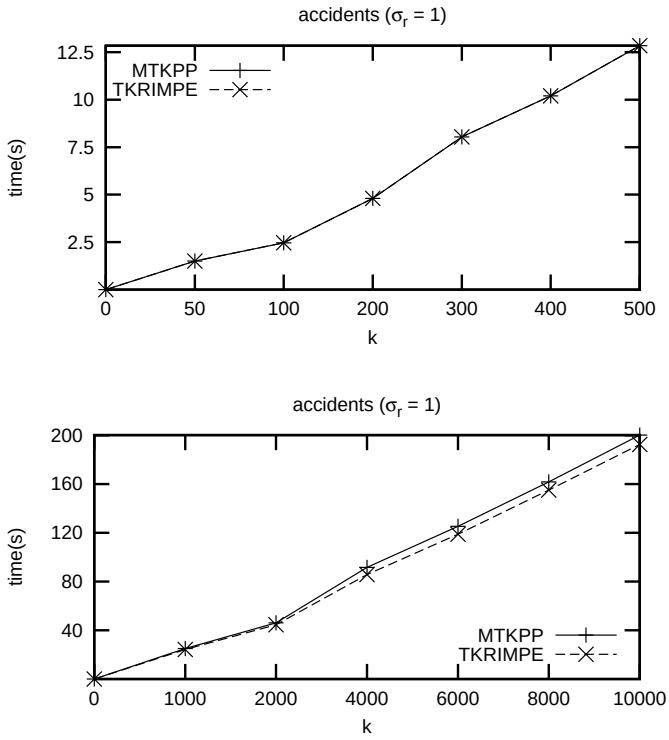


Figure 4.26: Runtime of TKRIMPE on *accidents* ($\sigma_r = 1\%$)

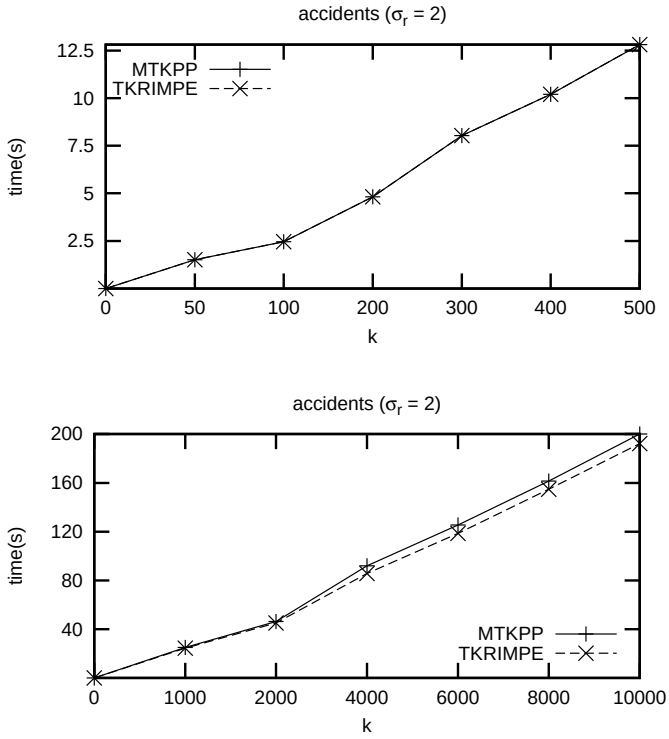


Figure 4.27: Runtime of TKRIMPE on *accidents* ($\sigma_r = 2\%$)

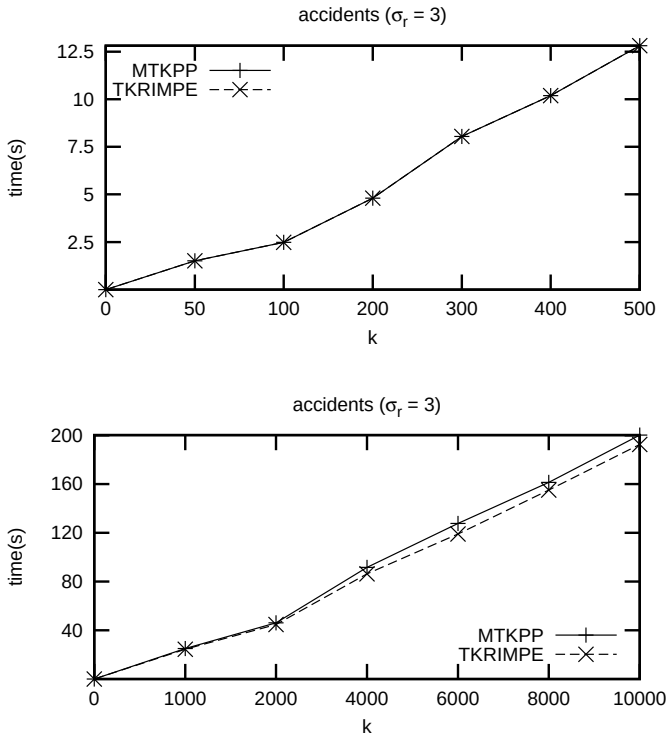


Figure 4.28: Runtime of TKRIMPE on *accidents* ($\sigma_r = 3\%$)

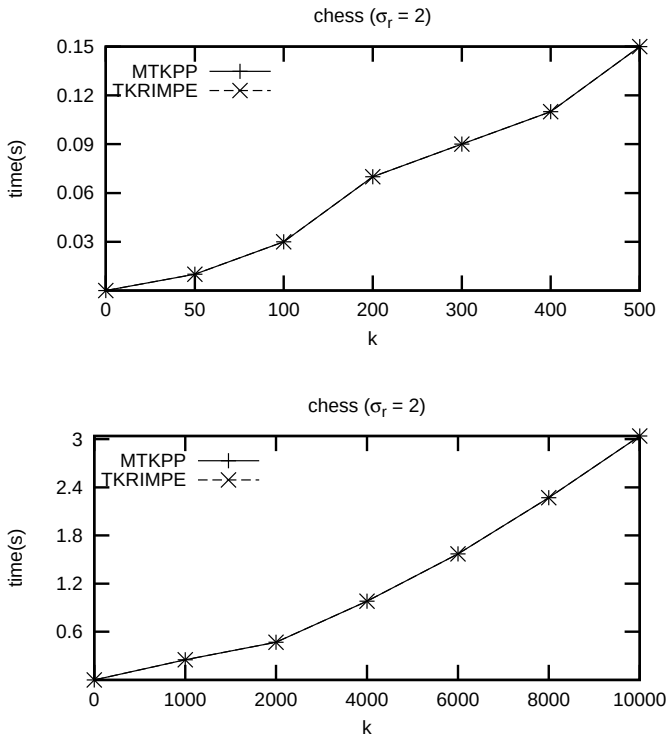


Figure 4.29: Runtime of TKRIMPE on *chess* ($\sigma_r = 2\%$)

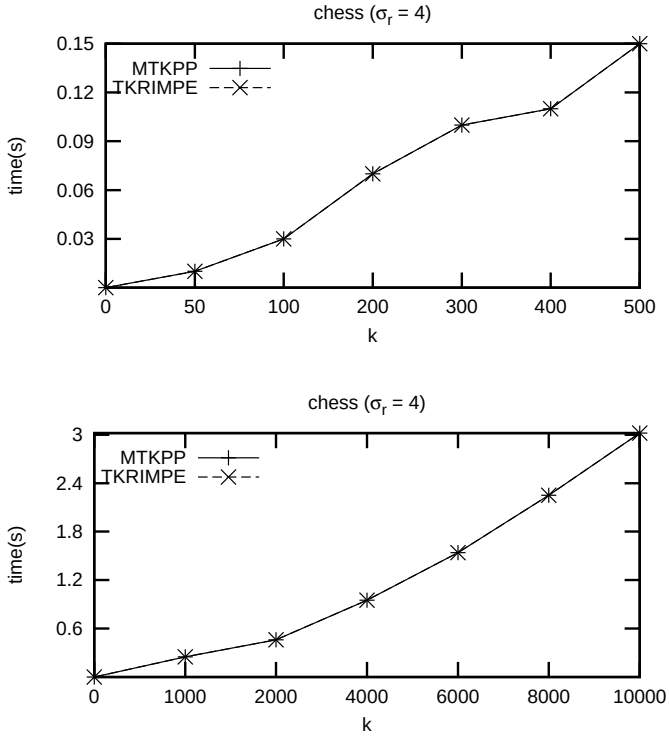


Figure 4.30: Runtime of TKRIMPE on *chess* ($\sigma_r = 4\%$)

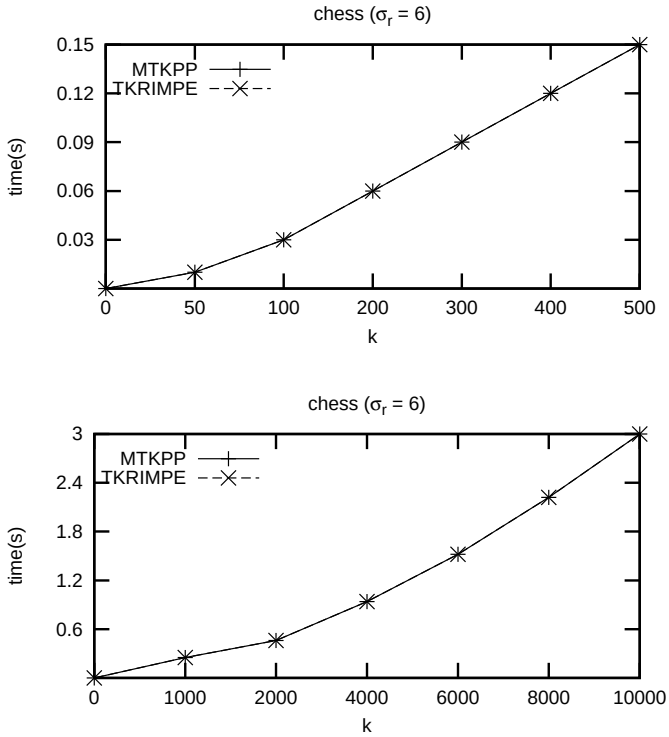


Figure 4.31: Runtime of TKRIMPE on *chess* ($\sigma_r = 6\%$)

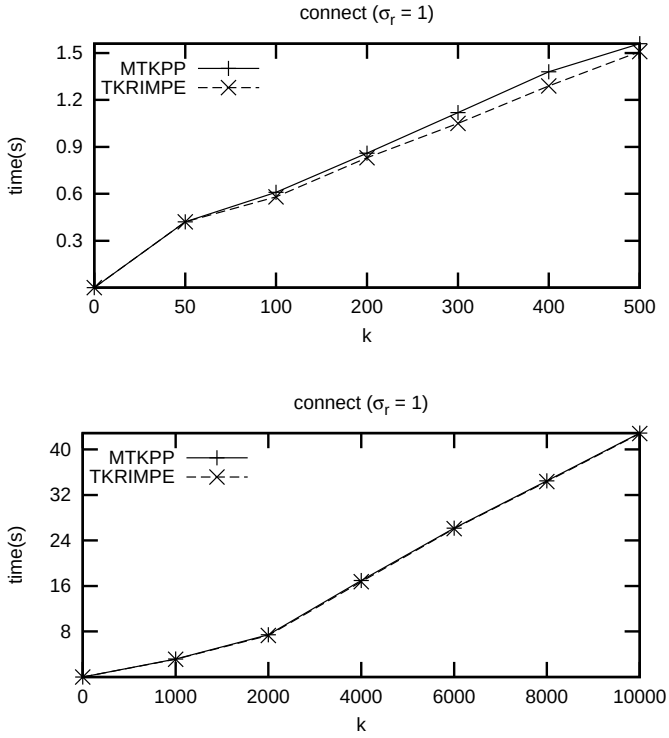


Figure 4.32: Runtime of TKRIMPE on *connect* ($\sigma_r = 1\%$)

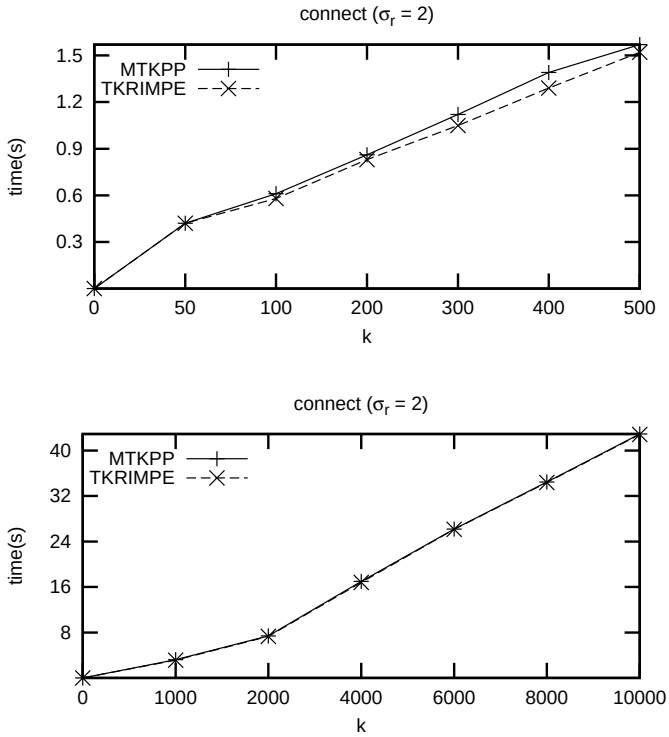


Figure 4.33: Runtime of TKRIMPE on *connect* ($\sigma_r = 2\%$)

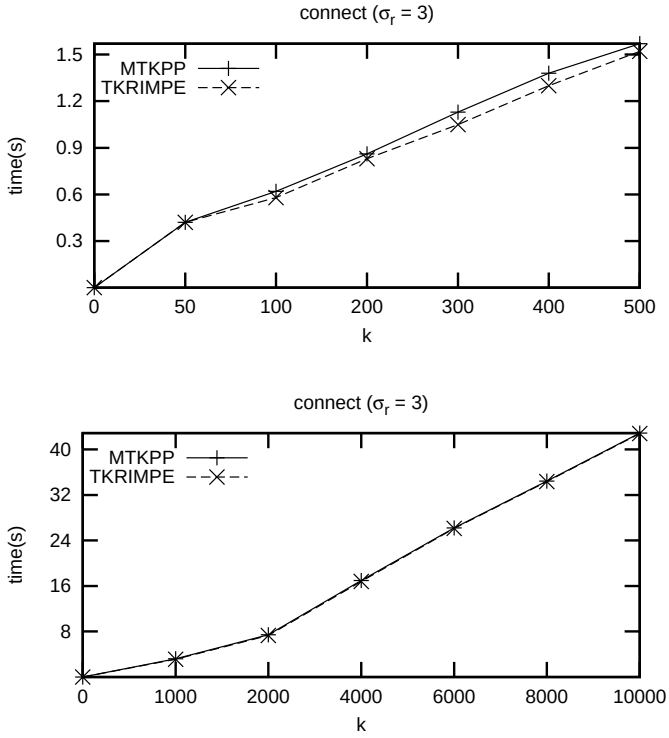


Figure 4.34: Runtime of TKRIMPE on *connect* ($\sigma_r = 3\%$)

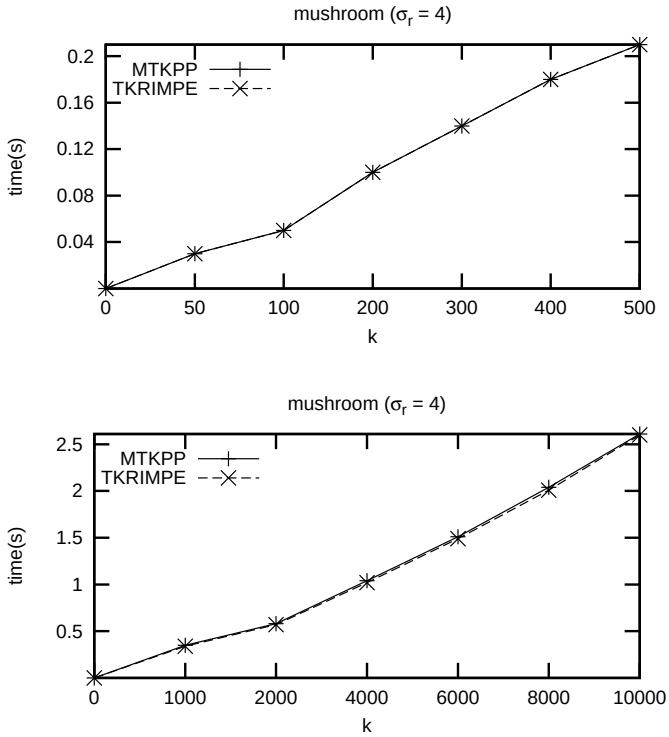


Figure 4.35: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 4\%$)

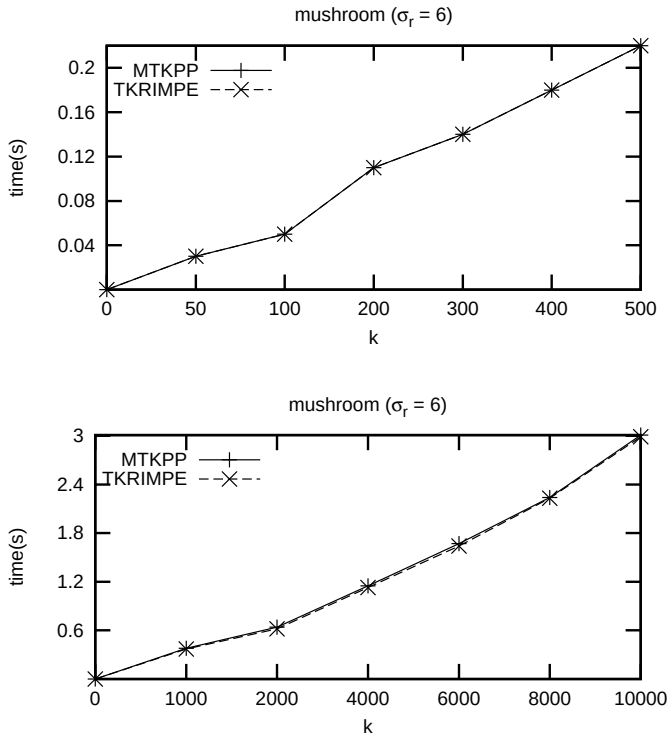


Figure 4.36: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 6\%$)

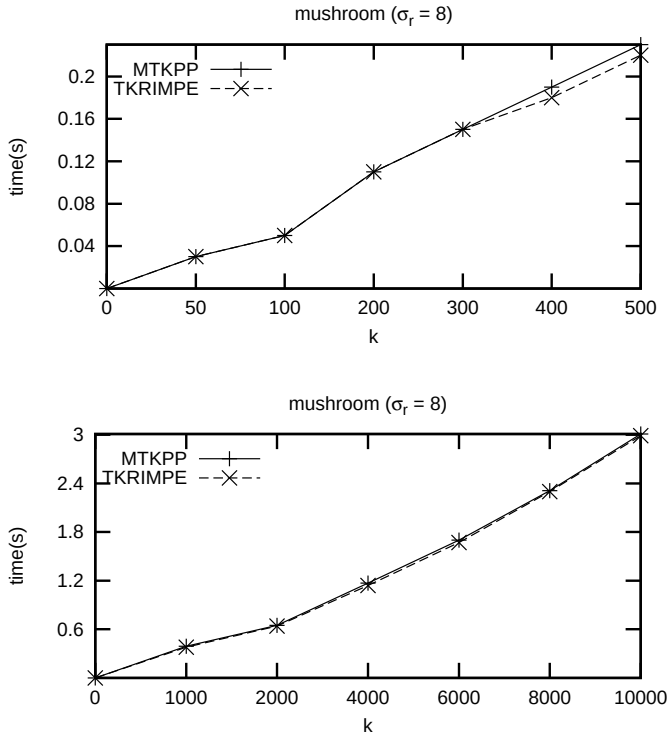


Figure 4.37: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 8\%$)

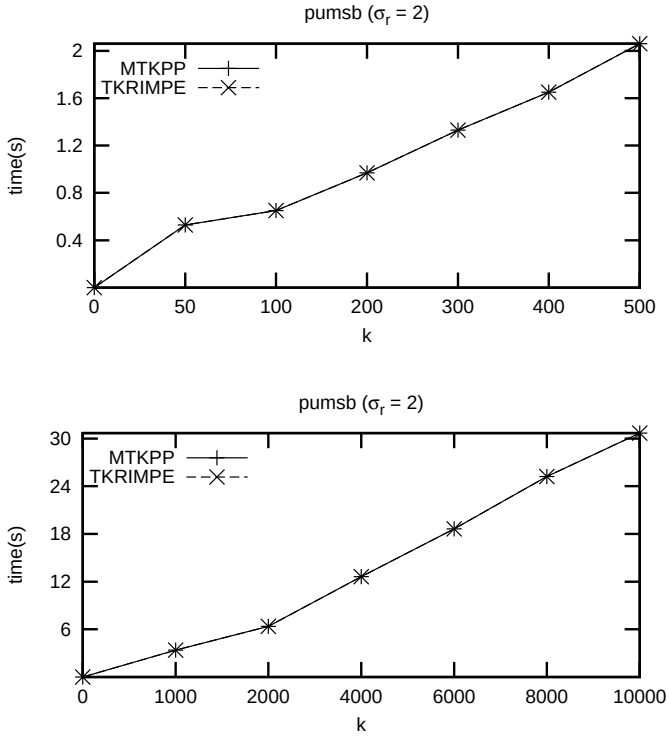


Figure 4.38: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 2\%$)

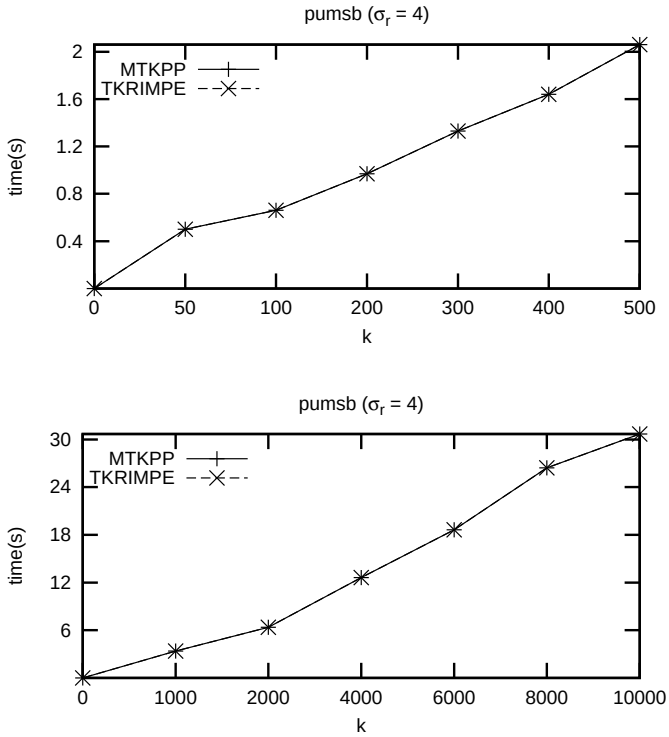


Figure 4.39: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 4\%$)

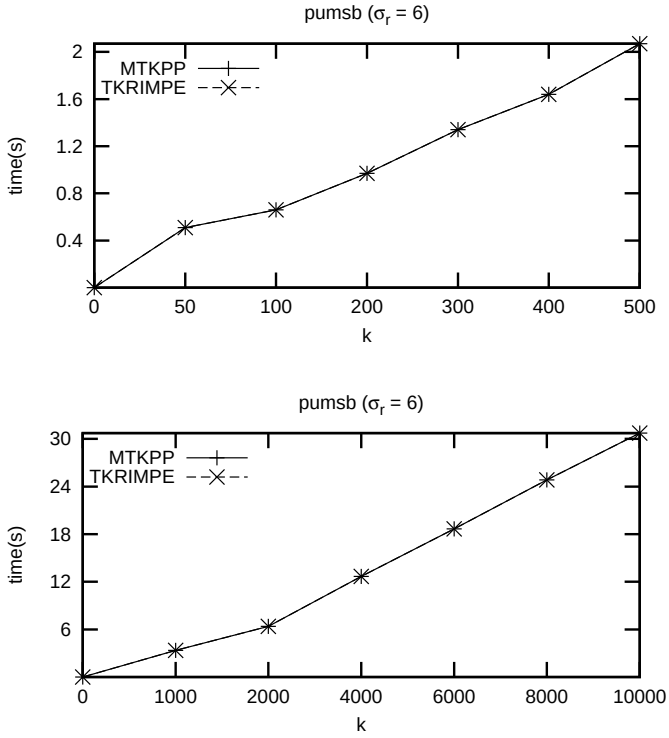


Figure 4.40: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 6\%$)

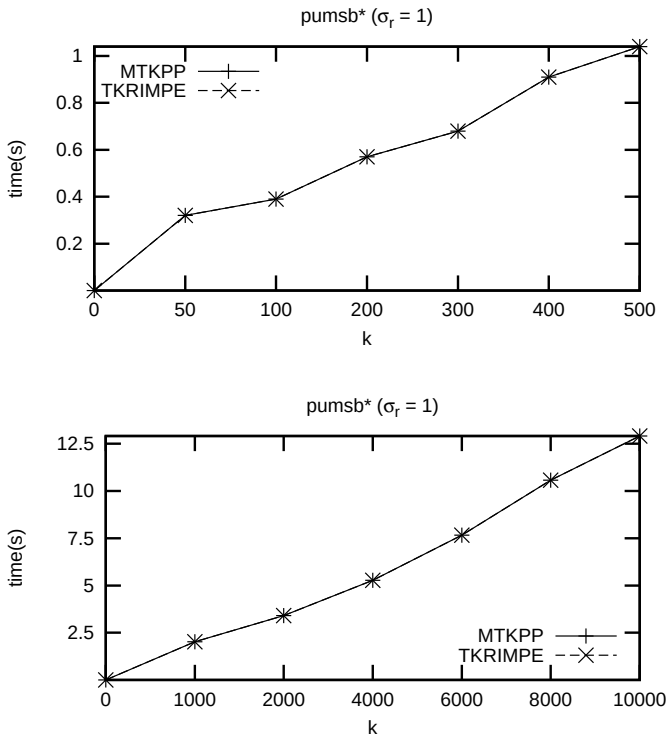


Figure 4.41: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 1\%$)

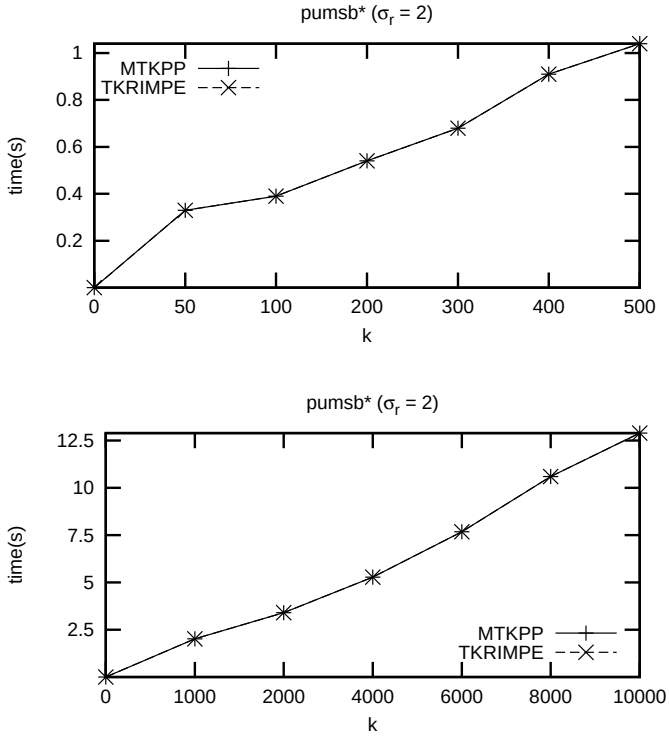


Figure 4.42: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 2\%$)

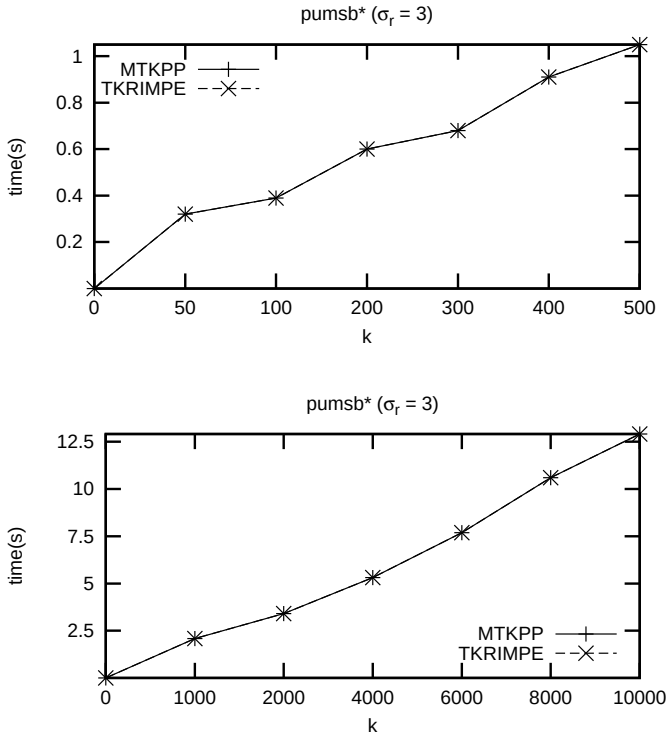


Figure 4.43: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 3\%$)

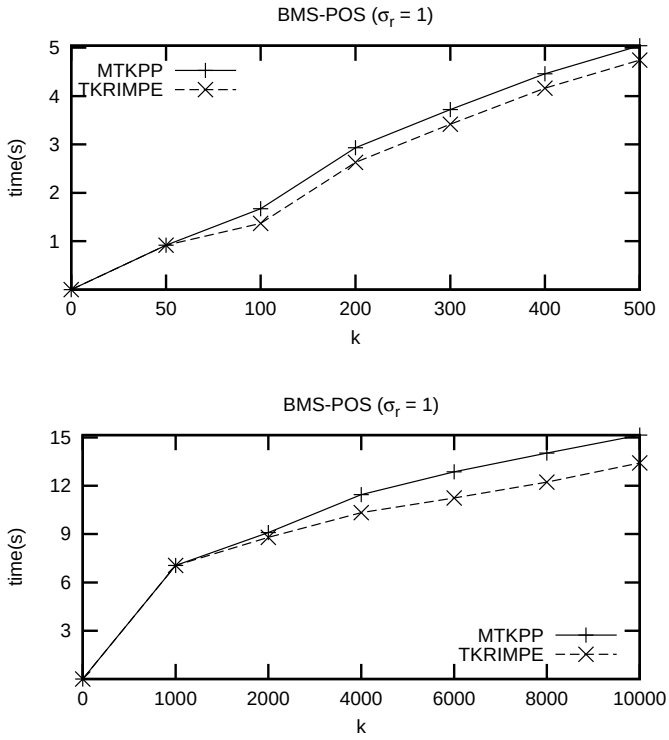


Figure 4.44: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 1\%$)

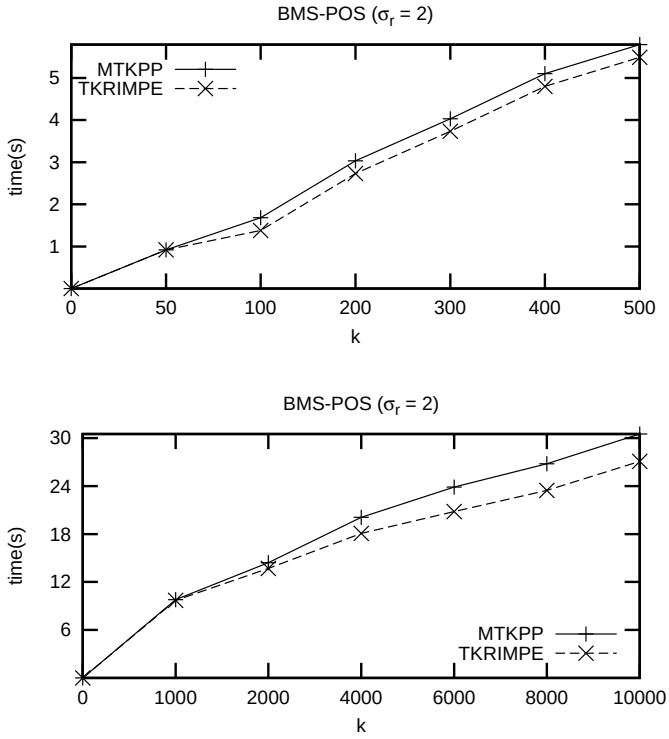


Figure 4.45: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 2\%$)

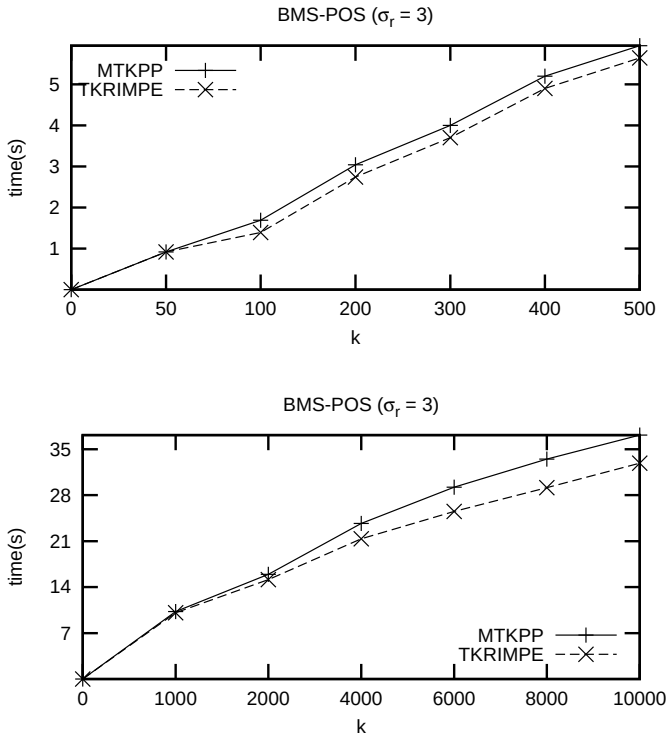


Figure 4.46: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 3\%$)

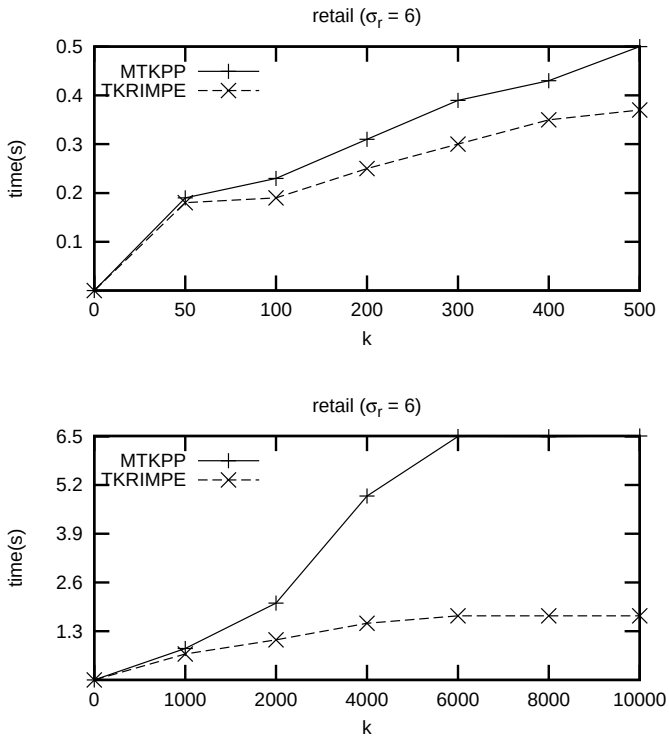


Figure 4.47: Runtime of TKRIMPE on *retail* ($\sigma_r = 6\%$)

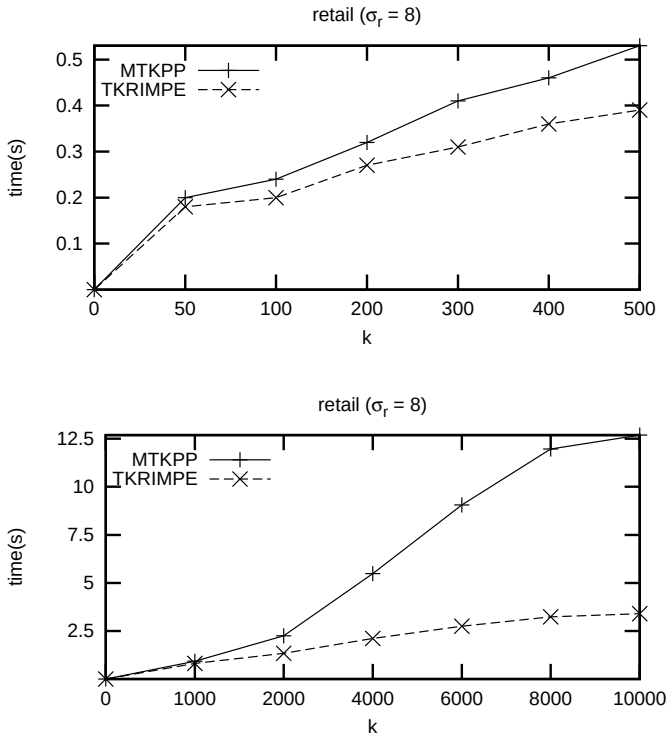


Figure 4.48: Runtime of TKRIMPE on *retail* ($\sigma_r = 8\%$)

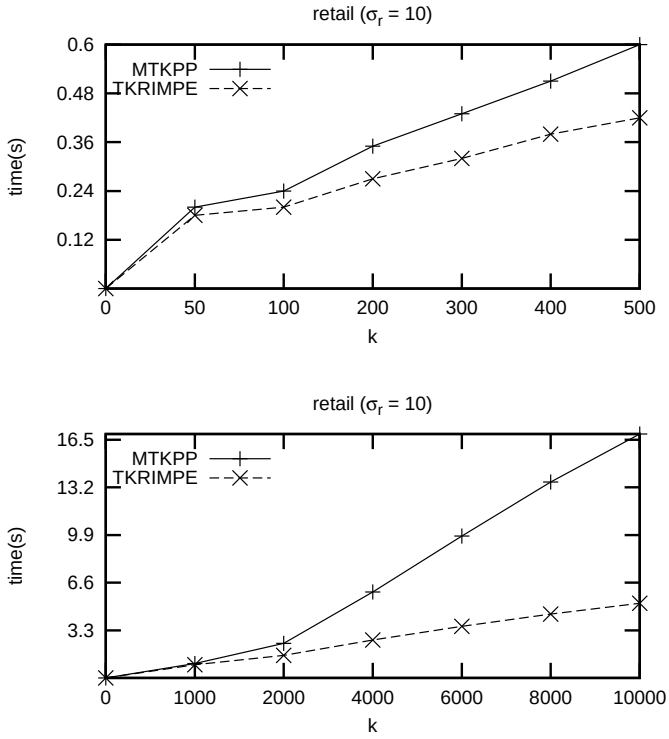


Figure 4.49: Runtime of TKRIMPE on *retail* ($\sigma_r = 10\%$)

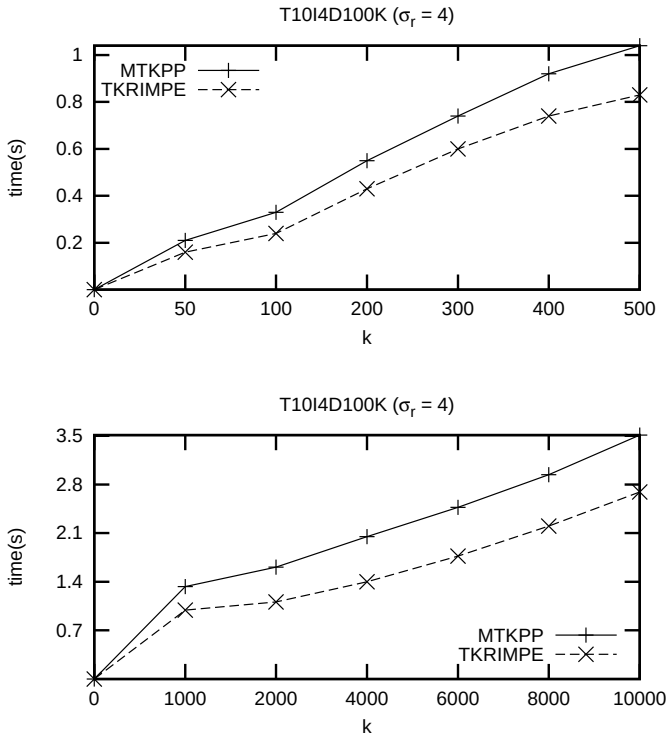


Figure 4.50: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 4\%$)

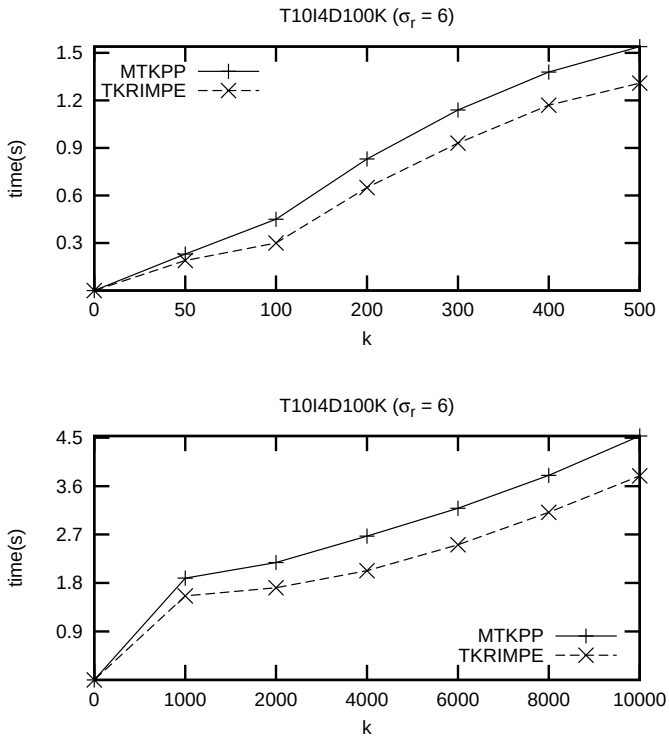


Figure 4.51: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 6\%$)

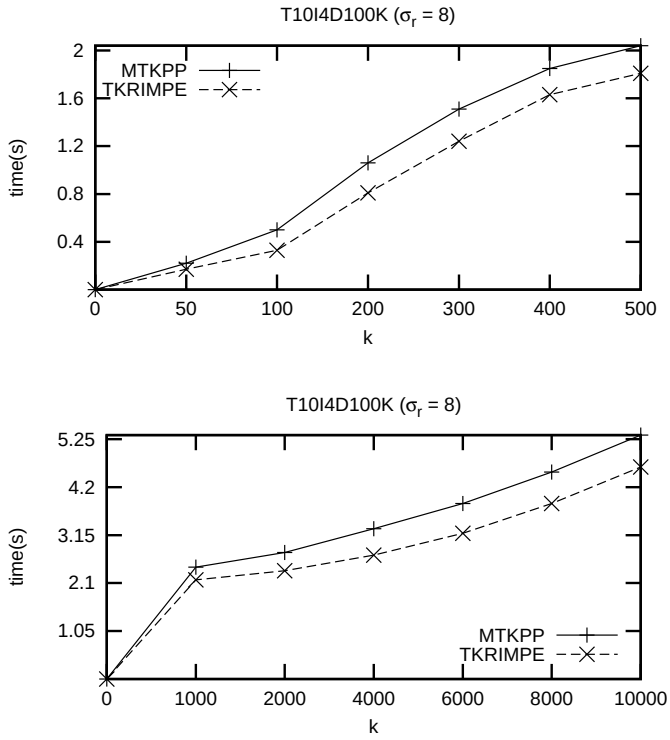


Figure 4.52: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 8\%$)

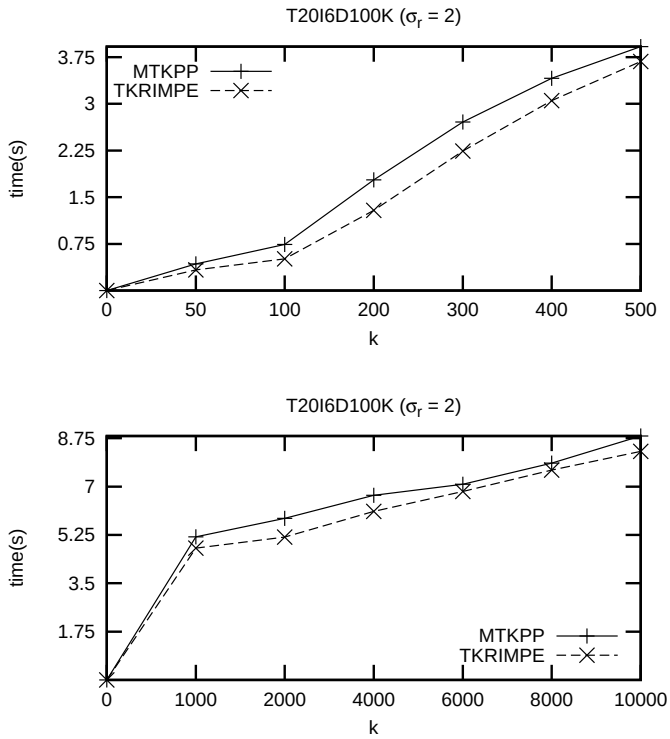


Figure 4.53: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 2\%$)

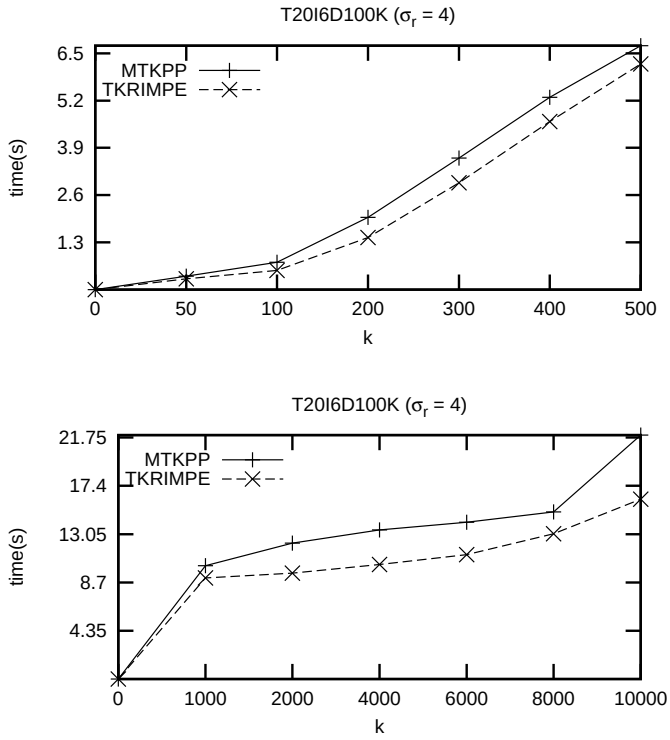


Figure 4.54: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 4\%$)

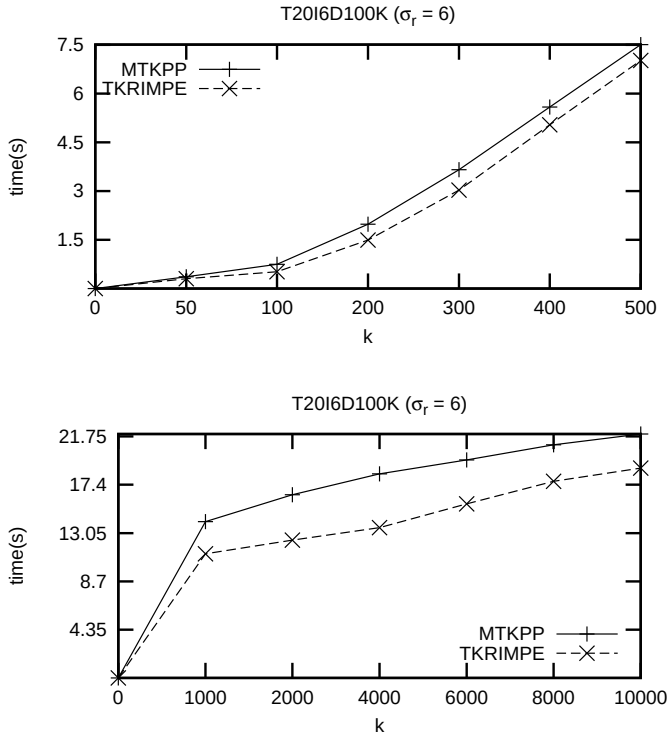


Figure 4.55: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 6\%$)

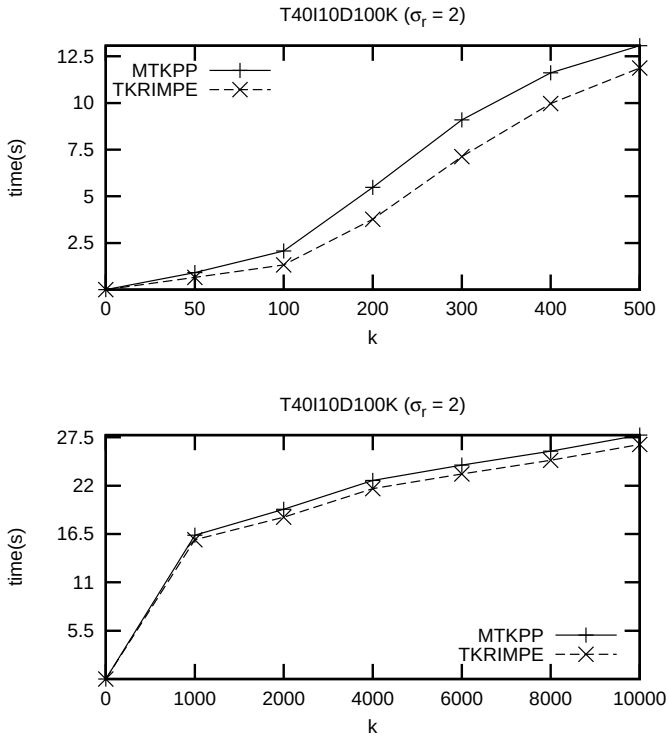


Figure 4.56: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 2\%$)

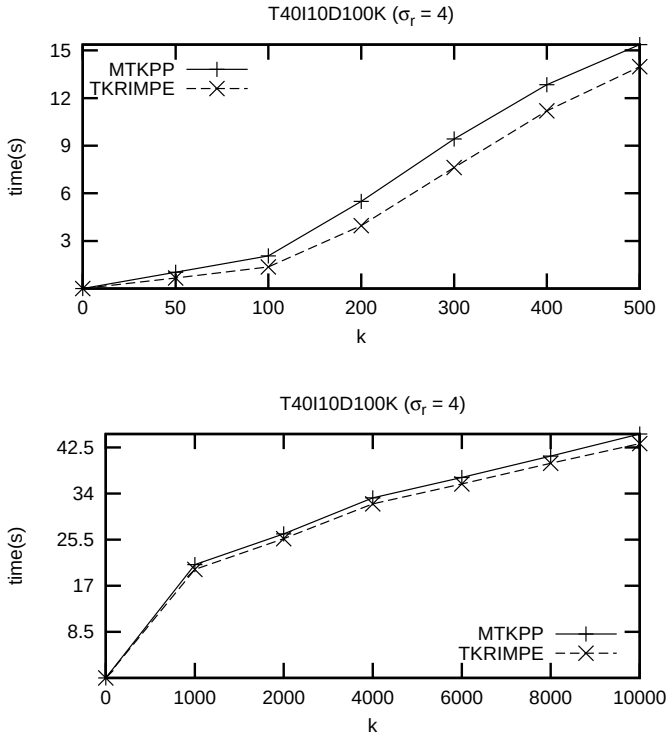


Figure 4.57: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 4\%$)

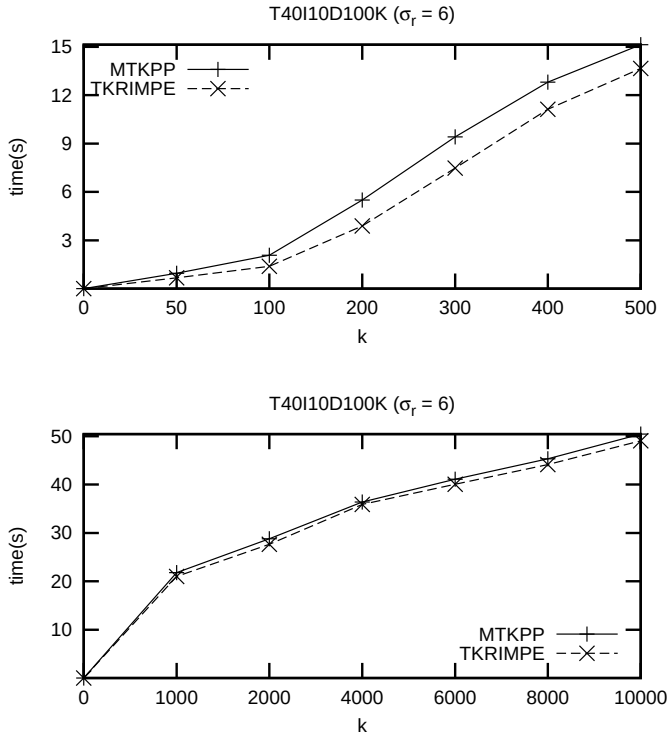


Figure 4.58: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 6\%$)

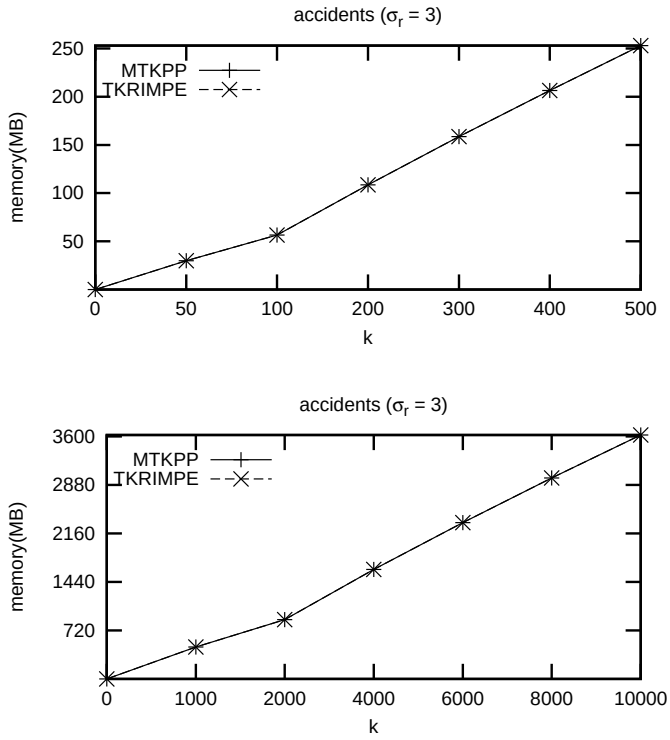


Figure 4.59: Memory usage of TKRIMPE on *accidents*

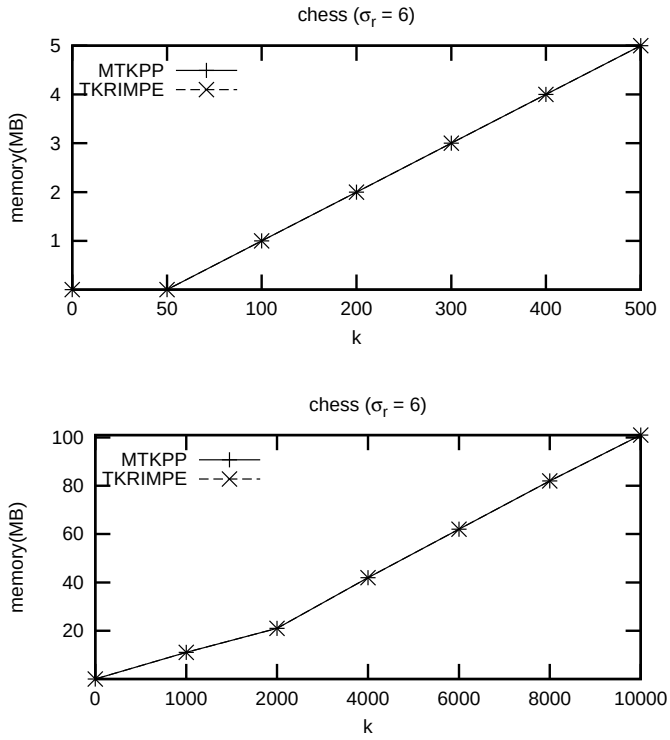


Figure 4.60: Memory usage of TKRIMPE on *chess*

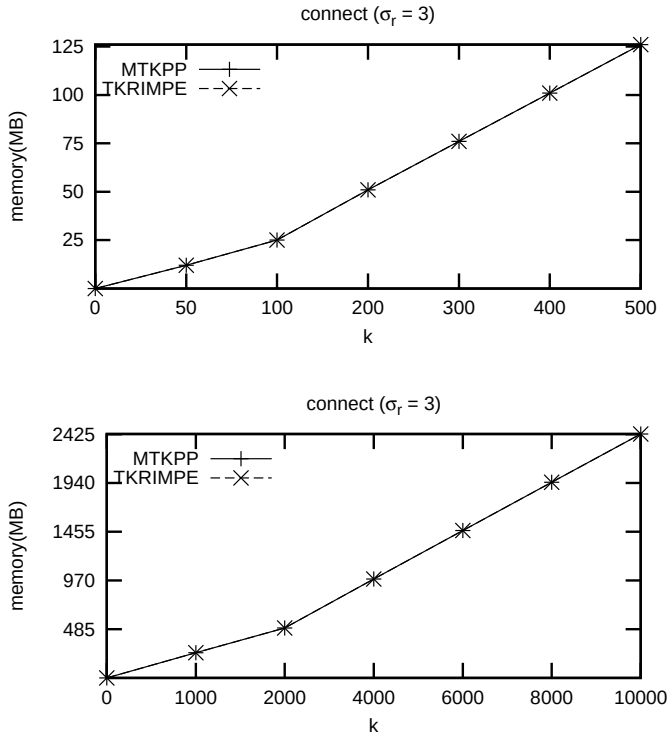


Figure 4.61: Memory usage of TKRIMPE on *connect*

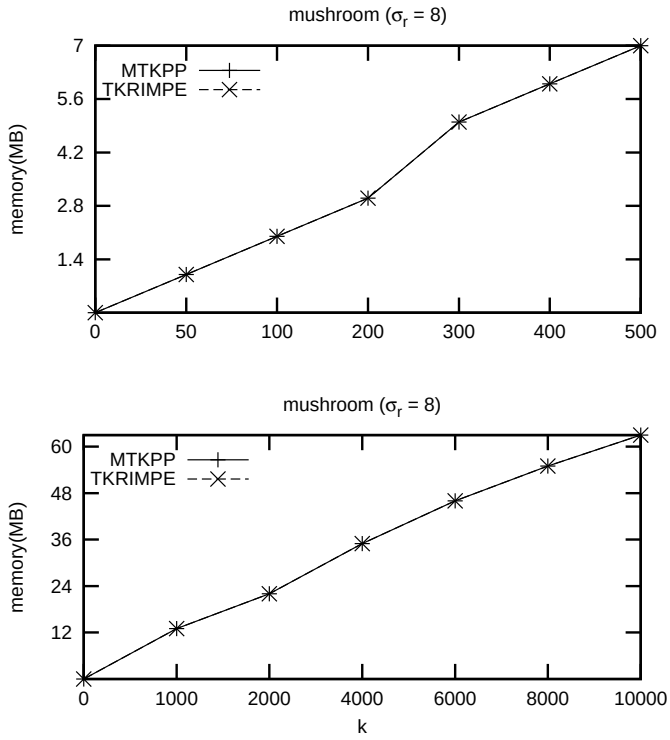


Figure 4.62: Memory usage of TKRIMPE on *mushroom*

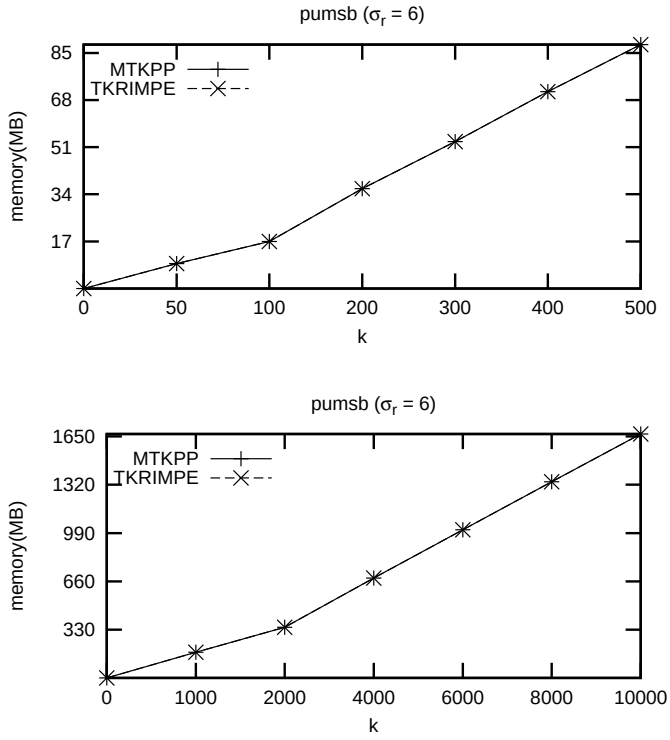


Figure 4.63: Memory usage of TKRIMPE on *pumsb*

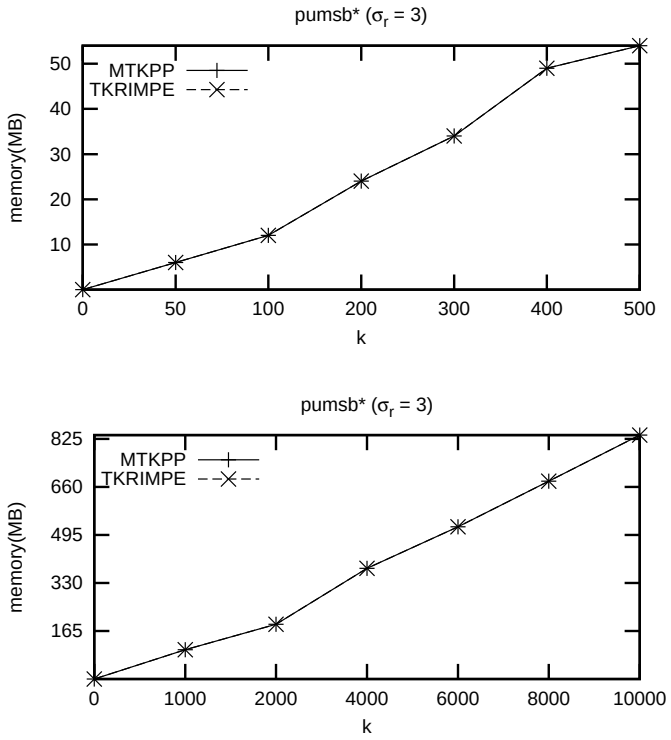


Figure 4.64: Memory usage of TKRIMPE on *pumsb**

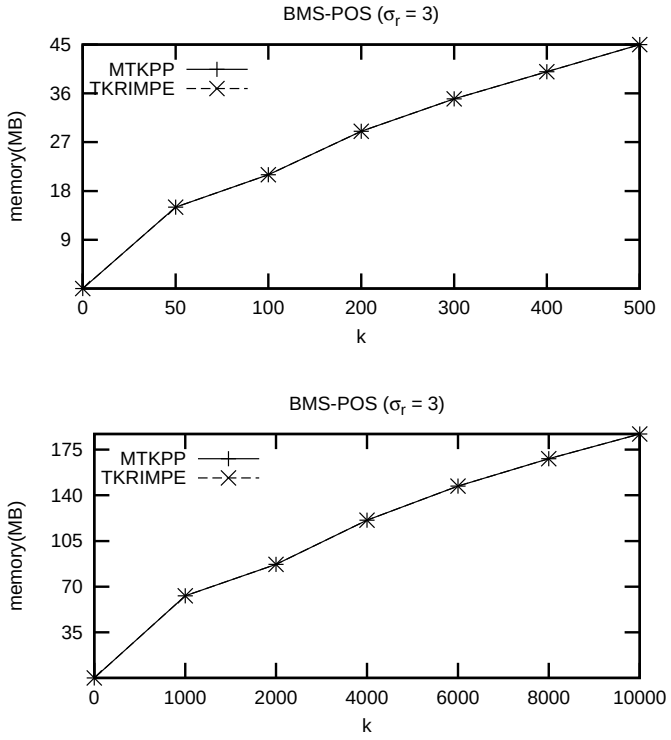


Figure 4.65: Memory usage of TKRIMPE on *BMS-POS*

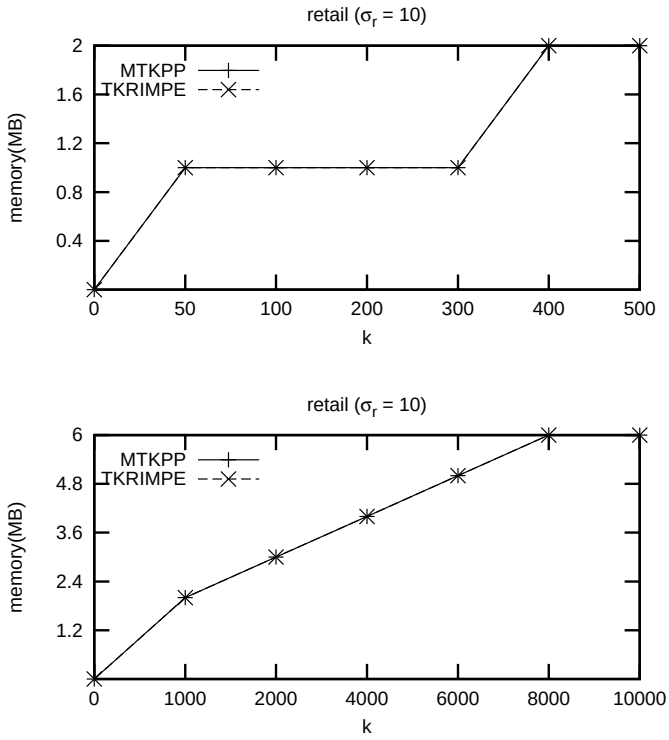


Figure 4.66: Memory usage of TKRIMPE on *retail*

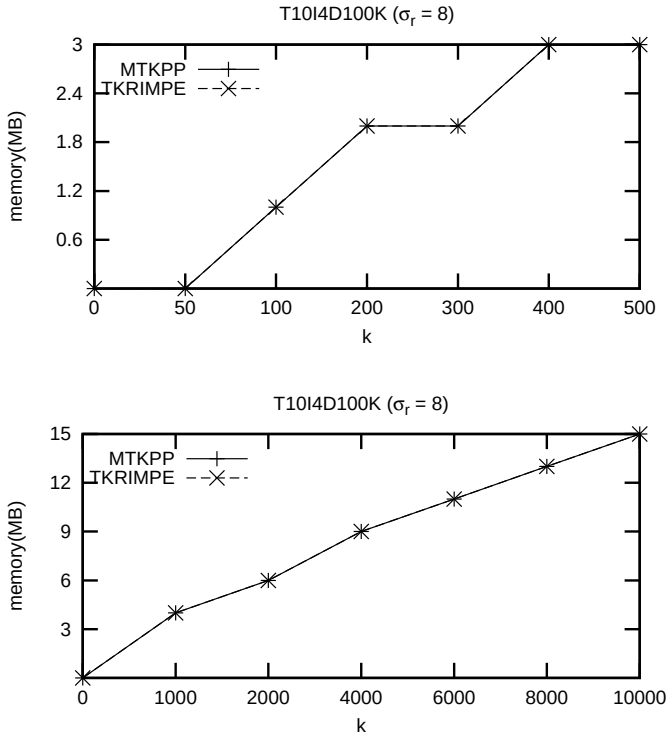


Figure 4.67: Memory usage of TKRIMPE on *T10I4D100K*

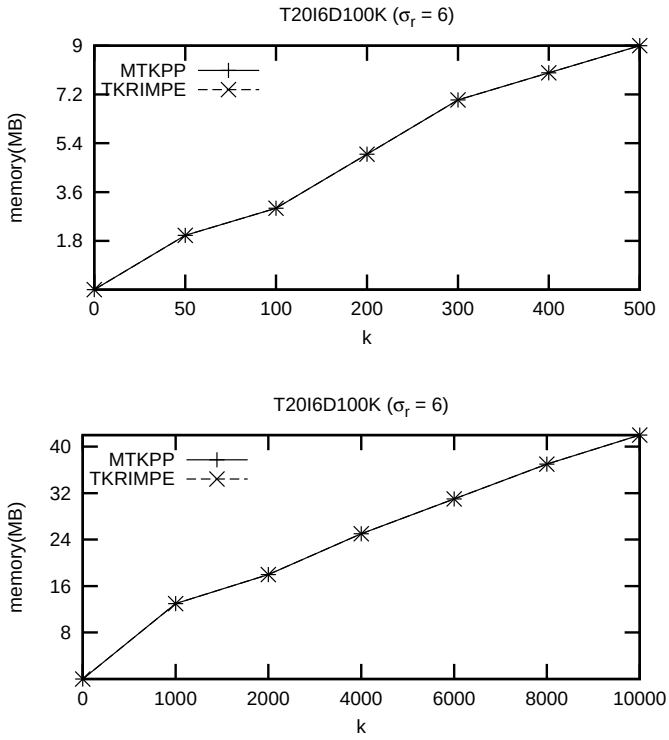


Figure 4.68: Memory usage of TKRIMPE on *T20I6D100K*

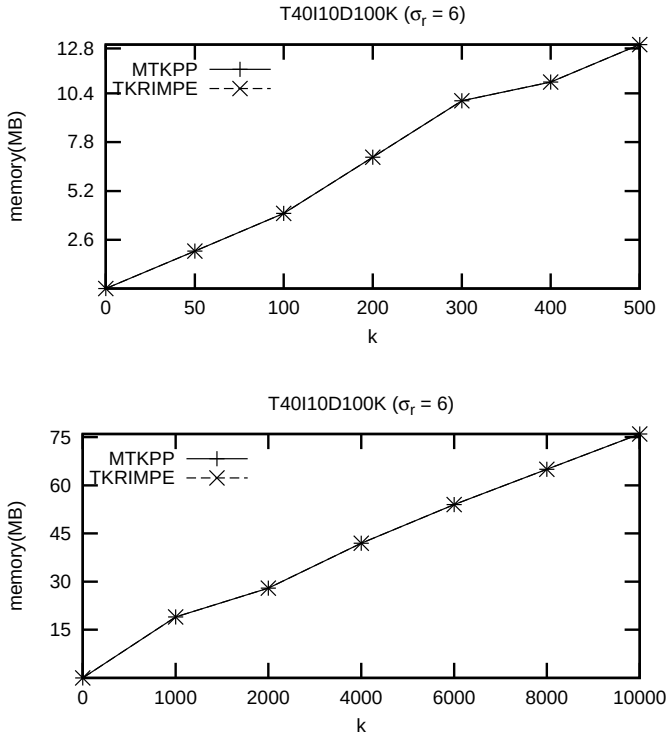


Figure 4.69: Memory usage of TKRIMPE on *T40I10D100K*

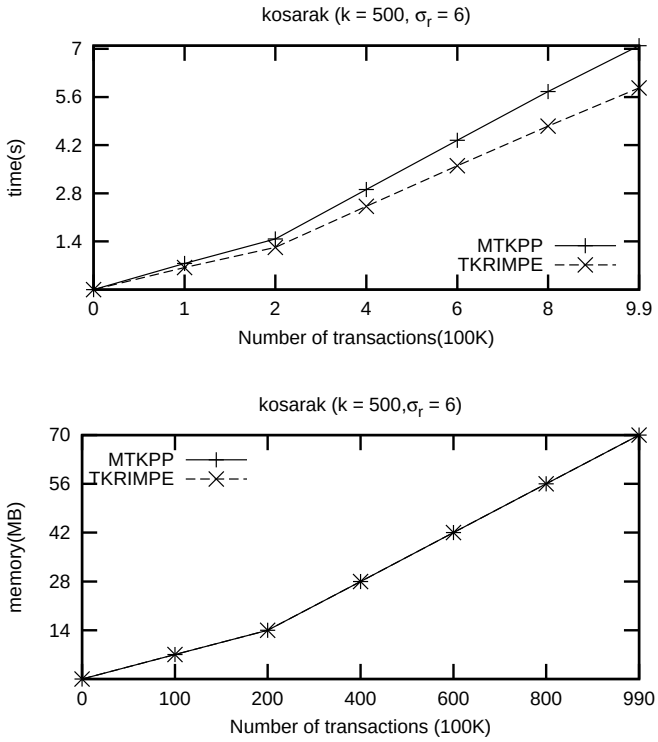


Figure 4.70: Scalability of TKRIMPE ($k : 500, \sigma_r = 6$)

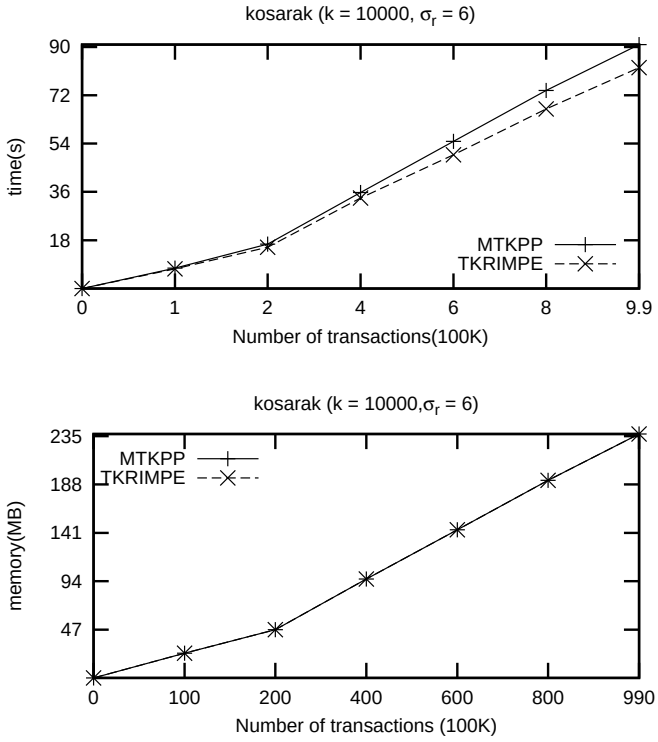


Figure 4.71: Scalability of TKRIMPE ($k : 10,000, \sigma_r = 6$)

CHAPTER V

TKRIMIT: TOP-*K* REGULAR-FREQUENT ITEMSETS MINING BASED ON INTERVAL TIDSET REPRESENTATION

The aim of this chapter is to reduce the computational time and memory consumption from the MTKPP and TKRIMPE algorithms by reducing the number of maintained tids during mining process. Hence, a new concise representation, called *interval transaction-ids set (interval tidset)*, used to maintain the occurrence information of each regular itemset is introduced and described in details. Based on the interval tidset representation, an interval tidset is employed instead of a normal tidset (*i.e.* maintaining all of tids that each itemset occurs) as used in MTKPP and TKRIMPE algorithms. In addition, an efficient algorithm, called *Top-K Regular-frequent Itemsets based on Interval Tidset representation (TKRIMIT)*, is also proposed. Lastly, the data structure and the complexity analysis of the TKRIMIT algorithm are discussed.

The experimental studies illustrate that TKRIMIT provides significant improvements, in particular for dense datasets, in comparison with MTKPP and TKRIMPE on both small and large number of required results.

5.1 Preliminary of TKRIMIT

To mine the top-*k* regular-frequent itemsets, TKRIMIT also employs a top-*k* list as MTKPP and TKRIMPE in order to maintain a set of top-*k* regular-frequent itemsets during mining process. Besides, the best-first search strategy is adopted to quickly mine itemsets with the highest supports (*i.e.* to raise up the support of the k^{th} itemset in the top-*k* list which helps to cut down the search space). In addition, the interval tidset representation is devised and utilized to reduce the number of maintained tids. By this way of doing, TKRIMIT can reduce memory to maintain tidsets and time to intersect between tidsets.

5.2 Interval Tidset representation

Interval tidset representation is a new concise representation used to store the occurrence information (tidsets) of the top-*k* regular-frequent itemsets during mining process. The main concept of the interval tidset is to wrap up two or more consecutive continuous tids by maintaining only the first (with one positive integer) and the last tids (with one negative integer) of that group

of tids. By applying this representation, TKRIMIT can thus reduce time to compute support and regularity, and also memory to store occurrence information. In particular this representation is appropriate for dense datasets.

Definition 5.1 (Interval tidset of an itemset X) *Let a set of tids that itemset X occurs in TDB be $T^X = \{t_p^X, t_{p+1}^X, \ldots, t_q^X\}$ where $p < q$ and there are some consecutive tids $\{t_u^X, t_{u+1}^X, \ldots, t_v^X\}$ that are continuous between t_p^X and t_q^X (**where** $p \leq u$ **and** $q \geq v$). Thus, the interval tidset of the itemset X is defined as:*

$$IT^X = \{t_p^X, t_{p+1}^X, \ldots, t_u^X, (t_u^X - t_v^X), t_{v+1}^X, \ldots, T_q^X\}$$

For example, from the transactional database of Table 5.1, an item a occurs in the set of transactions: $T^a = \{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$ which is composed of two groups of consecutive continuous transactions. Thus, by using the interval tidset representation, the interval tidset of the item a is $IT^a = \{1, -3, 6, -6\}$. The first interval tids $(1, -3)$ represents $\{t_1, t_2, t_3, t_4\}$ whereas $(6, -6)$ represents the last seven consecutive continuous tids that the item a occurs in the database. For the item a, the use of interval tidset representation is efficient. It can reduce seven tids to be maintained comparing with the normal tidset representation. For items b and c, the sets of transactions that they occur are $T^b = \{t_1, t_2, t_4, t_5, t_7, t_8, t_{10}, t_{11}\}$ and $T^c = \{t_1, t_3, t_5, t_7, t_9, t_{11}\}$, respectively. Therefore, the interval tidsets of the items b and c are $IT^b = \{1, -1, 4, -1, 7, -1, 10, -1\}$ and $IT^c = \{1, 3, 5, 7, 9, 11\}$ which are the examples of the worst cases of interval tidset representation.

The interval tidset representation is efficient as soon as there are three consecutive continuous tids in the tidsets whereas in the worst cases, the interval tidset representation contains the same number of tids as the normal tidset representation.

Theorem 5.1 *Let $|IT^X|$ is the number of tids in the interval tidset of an itemset X and s^X is its support. The $|IT^X| < s^X$ where $s^X > \lceil \frac{2}{3} \times |TDB| \rceil$ and $|TDB| \geq 3$. Otherwise, $|IT^X|$ can be less than or eqaul to s^X.*

Proof: Let $s^X > \lceil \frac{2}{3} \times |TDB| \rceil$ and let the tidset T^X of the itemset X has no more than two consecutive continuous tids. In fact, the maximum value of s^X when the tidset of X has no more than two consecutive continuous tids is $\lceil \frac{2}{3} \times |TDB| \rceil$. It happens in the case that the itemset X occurs in every two transactions and misses one transaction. In contradiction, for any s^X which

$s^X > \lceil \frac{2}{3} \times |TDB| \rceil$ must have at least one group of tids that occurs in three or more consecutive continuous. Therefore, when the tidset T^X has a group of three or more consecutive continuous tids, TKRIMIT (based on the interval tidset representation) can group these tids together by using only one positive and negative tids. Thus, $|IT^X| < s^X$. ■

With this representation a tidset of any itemsets may contain some negative tids. Therefore, the original definition (Definition 3.1) is not suitable to calculate the regularity from this kind of tidsets. Thus, a new way to calculate the regularity of any itemsets from the interval tidset representation is proposed.

Definition 5.2 (Regularity of an itemset X from interval tidset) *Let t_p^X and t_q^X be two consecutive tids in interval tidset IT^X, i.e. where $p < q$ and there is no transaction t_r, $p < r < q$, such that t_r contains X (note that p, q and r are indeces). Then, rtt_q^X is denoted as the regularity between two consecutive tids t_p^X and t_q^X* (i.e. *the number of tids (transactions) between t_p^X and t_q^X that do not contain X). Obviously, rtt_1^X is t_1^X. Last, to find the exact regularity of X in the database, the regularity between the last tid of $IT^X$and the last tid of the database should be calculated. This leads to the cases as follows:*

$$rtt_q^X = \begin{cases} t_q^X & \text{if } q = 1 \\ t_q^X - t_p^X & \text{if } t_p^X \text{ and } t_q^X > 0, 2 \leq q \leq |IT^X| \\ 1 & \text{if } t_p^X > 0 \text{ and } t_q^X < 0, 2 \leq q \leq |IT^X| \\ t_q^X + (t_p^X - t_{p-1}^X) & \text{if } t_p^X < 0 \text{ and } t_q^X > 0, 2 \leq q \leq |IT^X| \\ |TDB| - t_{|IT^X|}^X & \text{if } t_{|IT^X|}^X > 0, \text{ (i.e. } q = |IT^X| + 1\text{)} \\ |TDB| + (t_{|IT^X|}^X - t_{|IT^X|-1}^X) & \text{if } t_{|IT^X|}^X < 0, \text{ (i.e. } q = |IT^X| + 1\text{)} \end{cases}$$

Finally, the regularity of X is defined as $r^X = max(rtt_1^X, rtt_2^X, \ldots, rtt_{m+1}^X)$.

For example, consider the interval tidset $IT^a = \{1, -3, 6, -6\}$ of the item a. The set of regularities between each pair of two consecutive tids is equal to $\{1, 1, 6+(-3-1), 1, 12-(-6-6)\}$ = {1,1,2,1,0} and the regularity of the item a is 2.

Table 5.1: A transactional database as a running example of TKRIMIT

tid	items
1	*a b c d f*
2	*a b d e*
3	*a c d*
4	*a b*
5	*b c e f*
6	*a d e*
7	*a b c d e*
8	*a b d*
9	*a c d f*
10	*a b e*
11	*a b c d*
12	*a d f*

5.3 TKRIMIT: Top-*k* list structure

As in (Amphawan et al., 2009), TKRIMIT is based on the use of a top-*k* list, which is an ordinary linked-list, to maintain the top-*k* regular-frequent itemsets. A hash table is also utilized with the top-*k* list in order to quickly access each entry in the top-*k* list. At any time, the top-*k* list only contains not more than k regular-frequent itemsets in main memory. As shown in Figure 5.1, each entry in a top-*k* list consists of 4 fields: (*i*) item or itemset name (I), (*ii*) total support (s^I), (*iii*) regularity (r^I) and (*iiii*) an interval tidset where I occurs (IT^I). For example, the item a has a support of 11, a regularity of 2 and its interval tidset is $IT^a = \{1, -3, 6, -6\}$ (see Figure 5.1).

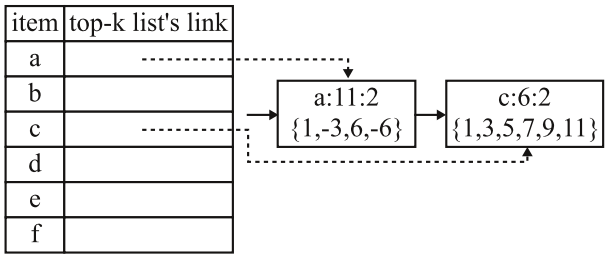


Figure 5.1: TKRIMIT: Top-*k* list structure with hash table

5.4 TKRIMIT algorithm

As mentioned above, the TKRIMIT is based on the interval tidset representation to maintain the occurrence information of each itemset and the use of a top-*k* list to collect the k regular itemsets during mining process. The TKRIMIT algorithm consists of two steps: (*i*) Top-*k* list initialization: scan database once to obtain and collect the all regular items(with highest support)

into the top-*k* list; (*ii*) Top-*k* mining: merge each pair of entries in the top-*k* list and then intersect their interval tidsets in order to collect tidset and to calculate the support and regularity of a new generated regular itemset.

5.4.1 TKRIMIT: Top-*k* list initialization

To create the top-*k* list, TKRIMIT scans the database once (transaction per transaction). Then, each item of the current transaction is then considered. With the help of the hash table, TKRIMIT can know quickly if the current item is already existed in the top-*k* list or not. In the first case, its support, regularity and interval tidset have just updated. If it is its first occurrence then a new entry is created and its support, regularity and interval tidset are initialized.

To update the interval tidset IT^X of an item X, TKRIMIT has to compare the last tid (t_i) of IT^X with the new coming tid (t_j). Thanks to the interval representation (see Definition 5.1) it simply consists of the following cases:

- if $t_i < 0$, i.e. there are some former consecutive continuous tids occurs with the exact tid of t_i, TKRIMIT calculates the exact tid of $t_i < 0$ (*i.e.* $t_{i-1} - t_i$) and compares it with t_j to check whether they are continuous. If they are consecutive continuous tids (*i.e* $t_j - t_{i-1} + t_i = 1$), TKRIMIT has to extend the interval tidset IT^X (it consists only of adding -1 to t_i). Otherwise, TKRIMIT creates a new element to take into account t_j (it simply consists of adding t_j after t_i in IT^X).
- if $t_i > 0$, i.e. there is no former consecutive continuous tid occurs with t_i, TKRIMIT compared t_i with t_j to check whether they are continuous or not. If they are consecutive continuous tids (i.e. $t_j - t_i = 1$), TKRIMIT creates a new interval in IT^X (it consists of adding -1 after t_i in IT^X). Otherwise, TKRIMIT creates a new element to take into account t_j (it simply consists of adding t_j after t_i in IT^X).

After scanning all transactions, the top-*k* list is trimmed by removing all the entries (items) with regularity greater than the regularity threshold σ_r, and the remaining entries are sorted in descending order of support. Lastly, TKRIMIT removes the entries after the k^{th} entry in top-*k* list. The details of the top-*k* list's construction are presented in Algorithm 5.

Algorithm 5 (TKRIMIT: Top-*k* list initialization)

(*1*) A transaction database: TDB
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r
Output:
(*1*) A top-*k* list

create a hash table for all 1-items
for each transaction j in TDB **do**
  for each item i in the transaction j **do**
    if the item i does not have an entry in top-*k* list **then**
      create a new entry for item i with $s^i = 1, r^i = t_j$, and $T^i = T^i \cup t_j$
      create a link between the hash table and the entry
    else
      calculate the regularity r^i by t_j
      add the support s^i by 1
      if the last tid in $T^i(t^i_{|T^i|}) < 0$ **then**
        if $t_j - (t^i_{|T^i|-1} - t^i_{|T^i|}) = 1$ **then**
          add the last tid in T^i by -1
        else
          collect t_j as the last tid in T^i
      else
        if $t_j -$ the last tid in $T^i = 1$ **then**
          collect -1 as the last tid in T^i
        else
          collect t_j as the last tid in T^i

for each item i in top-*k* list **do**
  calculate the regularity r^i by $|TDB| -$ the last tid of T^i (in case of the last tid > 0, otherwise $|TDB| - (t^i_{|T^i|-1} - t^i_{|T^i|})$)
  if $r^i > \sigma_r$ **then**
    remove the entry i out of the top-*k* list
sort the top-*k* list by support descending order
remove all of items having the support less than k^{th} item in the top-*k* list

Algorithm 6 (TKRIMIT: Top-*k* mining)

Input:

(*1*) A top-*k* list
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r

Output:

(*1*) A set of top-*k* regular-frequent itemsets

for each entry x in the top-*k* list **do**
  for each entry y in the top-*k* list $(x > y)$ **do**
    if the entries x and y have the same size of itemsets and the same prefix
    ($|I^x| = |I^y|$ and $i^x_1 = i^y_1, i^x_2 = i^y_2, \ldots, i^x_{|I^x|-1} = i^y_{|I^x|-1}$) **then**
      merge the itemsets of x and y to be itemset $Z = I^x \cup I^y$
      $r^Z = 0, s^Z = 0$
      for each t_p in T^{I^x} ($p = 1$ to $|T^{I^x}|$) and t_q in T^{I^y} ($q = 1$ to $|T^{I^y}|$) **do**
        if $t_p > 0$ and $t_q > 0$ **then**
          if $t_p = t_q$ **then**
            calculate the regularity r^Z by t_p and check r^Z with σ_r
            add the support s^Z by 1
            collect t_p as the last tid in T^Z
        else if $t_p > 0$ and $t_q < 0$ **then**
          if $t_p \leq t_{q-1} - t_q$ **then**
            calculate the regularity r^Z by t_p and check r^Z with σ_r
            add the support s^Z by 1
            collect t_p as the last tid in T^Z
        else if $t_p < 0$ and $t_q > 0$ **then**
          if $t_{p-1} - t_p \geq t_q$ **then**
            calculate the regularity r^Z by t_q and check r^Z with σ_r
            add the support s^Z by 1
            collect t_q as the last tid in T^Z
        else
          if $t_{p-1} - t_p > t_{q-1} - t_q$ **then**
            collect $t^Z_{|T^Z|} - (t_{q-1} - t_q)$ as the last tid in T^Z
            add the support s^Z by $(t_{q-1} - t_q) - t^Z_{|T^Z|}$
          else
            collect $t^Z_{|T^Z|} - (t_{p-1} - t_p)$ as the last tid in T^Z
            add the support s^Z by $(t_{p-1} - t_p) - t^Z_{|T^Z|}$
      calculate the regularity r^Z by $|TDB|-$ the last tid of T^Z (in case of the last tid > 0, otherwise $|TDB|- (t^Z_{|T^Z|-1} - t^Z_{|T^Z|})$)
      if $r^Z \leq \sigma_r$ and $s^Z \geq s_k$ **then**
        remove k^{th} entry from the top-*k* list
        insert the itemset Z into the top-*k* list with r^Z, s^Z and T^Z

5.4.2 TKRIMIT: Top-*k* mining

As described in Algorithm 6, a best-first search strategy (from the most frequent itemsets to the least frequent itemsets) is adopted to quickly generate the regular itemsets with the highest supports from the top-*k* list. This technique can help TKRIMIT to prune the search space when TKRIMIT can quickly find the top-*k* regular-frequent itemsets with the highest supports.

To find the top-*k* regular-frequent itemsets, two candidate itemsets X and Y in the top-*k* list are merged with the following two constraints: (*i*) the size of the itemsets of both elements must be equal; (*ii*) both itemsets must have the same prefix (*i.e.* each item from both itemsets is the same, except the last item). This way of doing will help the proposed algorithm to avoid the repetition of generating larger itemset and may help to prune the search space. Then, the interval tidsets of the two candidate itemsets are sequentially intersected in order to calculate the support, the regularity and to collect the interval tidset of the new generated itemset. To sequentially intersect the interval tidsets IT^X and IT^Y of X and Y, one have to consider four cases when comparing each pair of tids t_i^X and t_j^Y in order to construct IT^{XY} (see Definition 5.1):

(*i*) if $t_i^X = t_j^Y > 0$ add t_i^X at the end of IT^{XY}

(*ii*) if $t_i^X > 0, t_j^Y < 0, t_i^X \leq t_{j-1}^Y - t_j^Y$, add t_i^X at the end of IT^{XY}

(*iii*) if $t_i^X < 0, t_j^Y > 0, t_j^Y \leq t_{i-1}^X - t_i^X$, add t_j^Y at the end of IT^{XY}

(*iiii*) if $t_i^X, t_j^X < 0$, add $t^{XY}_{|IT^{XY}|} - (t_{i-1}^X - t_i^X)$ at the end of IT^{XY} if $t_{i-1}^X - t_i^X < t_{j-1}^Y - t_j^Y$, otherwise add $t^{XY}_{|IT^{XY}|} - (t_{j-1}^Y - t_j^Y)$ at the end of IT^{XY}

From IT^{XY}, the support s^{XY} and the regularity r^{XY} of XY (see definition 5.2) are easily computed. TKRIMIT then removes the k^{th} entry and inserts the itemset XY into the top-*k* list if s^{XY} is greater than the support of the k^{th} itemset in the top-*k* list and if r^{XY} is not greater than the regularity threshold σ_r.

5.5 Example of TKRIMIT

Consider the TDB of Table 5.1, a regularity threshold σ_r of 4 and the number of desired results k of 5. Then, the initialization of the top-*k* list from the TDB is illustrated in Figure 5.2.

After scanning the first transaction ($t_1 = \{a, b, c, d, f\}$), the entries for items a, b, c, d and f are created, their supports, regularities and interval tidsets are also initialized as $(1 : 1 : \{1\})$

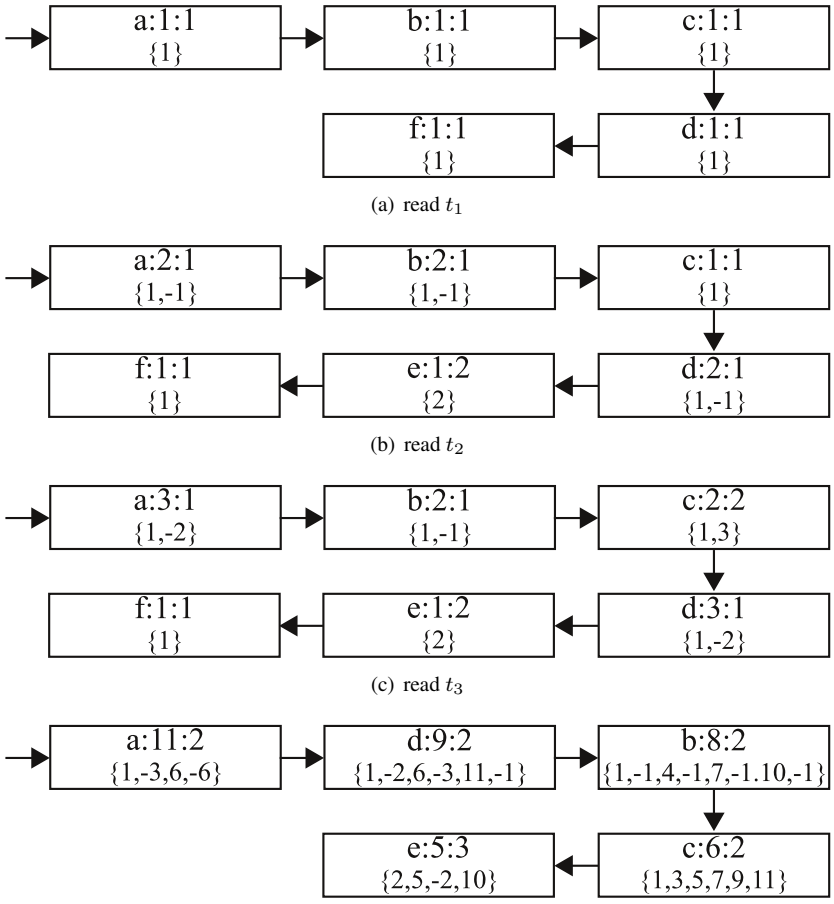
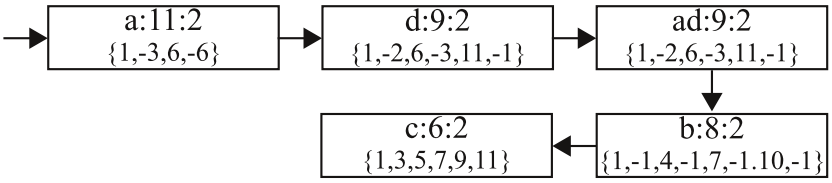


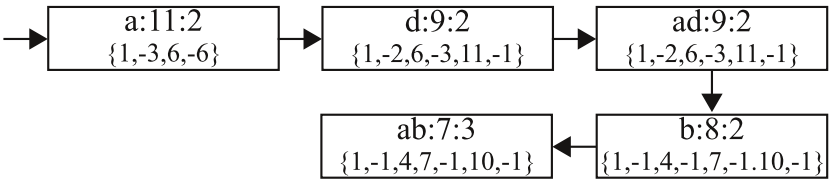
Figure 5.2: Top-*k* list initialization

(see Figure 5.2(a)). With the second transaction ($t_2 = \{a, b, d, e\}$), TKRIMIT adds -1 at the end of the interval tidsets of a, b and d, since these items occur in two consecutive continuous transactions. Then, the entry for the item e is created and initialized (see Figure 5.2(b)). For the third transaction ($t_3 = \{a, c, d\}$), as shown in Figure 5.2(c), the last tids of the items a and d are changed to -2 (*i.e.* they occur in the three consecutive continuous transactions t_1, t_2 and t_3) and the interval tidset of item c is updated by adding t_3 as the last tid. After scanning all the transactions, the top-*k* list is sorted by its support descending order and item f is removed (Figure 5.2(d)).

In the mining process, item d is firstly merged with the former item a. The interval tidsets IT^a and IT^d are sequentially intersected to calculate the support $s^{ad} = 9$, the regularity $r^{ad} = 3$ and to collect the interval tidset $IT^{ad} = \{1, -2, 6, -3, 11, -1\}$ of the itemset ad. Since the support s^{ad} is greater than $s^e = 5$ and the regularity r^{ad} is less than $\sigma_r = 4$, the item e is removed and ad is inserted into the top-*k* list as shown in Figure 5.3(a). Next, the third itemset *i.e.* the itemset ad is compared to the former itemsets a and b. These itemsets do not share the same



(a) top-*k* list when merging item a with item d



(b) final top-*k* list

Figure 5.3: Top-*k* during mining process

prefix and thus are not merged. TKRIMIT then considers the item b which is merged with a and d ($s^{ab} = 7$, $r^{ab} = 3$, $IT^{ab} = \{1, -2, 7, -1, 10, -1\}$; $s^{bd} = 5$, $r^{bd} = 5$, $IT^{bd} = \{1, -1, 7, -1, 11\}$). The itemset ab is thus added to the list and itemset c is removed. The itemset bd is eliminated, since its regularity is greater than σ_r. Lastly, the itemsets ab and ad are considered and the top-*k* regular-frequent itemsets are finally obtained as shown in Figure 5.3(b).

5.6 Complexity analysis

In this section, the complexity of TKRIMIT is further discussed in terms of time and space.

Proposition 5.3 *The time complexity for initializing the top-k list is $O(nm)$ where n is the number of items occurring in database and m is the number of transactions in database.*

Proof: Since the proposed algorithm scans each transaction in the database once, the entry of each item that occurs in the transaction is also looked up once in order to collect tids into tidsets. Hence, the cost for database scanning is $O(nm)$ whereas the cost for sorting all (in the very worst case) the entries is $O(n \log n)$. Then, the time complexity to create the top-*k* list is formally $O(nm + n \log n)$. In fact, the number of items (n) is, for the considered applications, always less than the number of transactions(m). Thus, the time complexity to create the top-*k* list is $O(nm)$. ■

Proposition 5.4 *The time complexity for mining top-k regular-itemsets is $O(m(k^2))$ where m is the number of transaction in database and k is the number of itemsets to be mined.*

Proof: The mining process merges each itemset in the top-*k* list with only the former itemset in the top-*k* list. Then, the interval tidsets of the two merged itemsets are intersected. Therefore, the combination of all itemsets in the top-*k* list is $k * (k + 1)/2$ and the time to intersect any two interval tidsets at each step is $O(m)$. Thus, the overall time complexity of mining process is $O(mk^2)$. ■

Proposition 5.5 *The memory space required for TKRIMIT is $O((\lceil \frac{2}{3}m \rceil)k)$ where m is the number of transaction in database and k is the number of itemsets to be mined.*

Proof: Base on Theorem 5.1, the maximum number of maintained tids of each itemset is $\lceil \frac{2}{3}|TDB| \rceil$. Then, all of desired memory to maintain interval tidsets for k itemsets is $O((\lceil \frac{2}{3}m \rceil)k)$. ■

5.7 Performance evaluation

In order to validate the effectiveness of the TKRIMIT algorithm based on the interval tidset representation, several experiments were conducted to compare the performance of TKRIMIT with the TKRIMPE and MTKPP algorithms. To measure the performance of the three algorithms, the processing time (*i.e.* included top-*k* list construction and mining processes) and space usage (*i.e.* memory consumption and the number of maintained tids during mining process) are considered.

5.7.1 Experimental setup

All experiments were performed on an Intel®Xeon 2.33 GHz with 4 GB main memory, running on Linux platform and all the programs were coded in C with the same structure as MTKPP (i.e. based on the use of top-*k* list). The experiments were performed on nine real datasets (accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb*, retail) and three

synthetic datasets (T10I4D100K, T20I6D100K, and T40I10D100K) of which some statistical information are shown in Chapter 2. The performance of TKRIMIT is evaluated by various values of k and σ_r. It can be observed that in all datasets the high value of regularity threshold (σ_r) will give a greater number of regular itemsets. This is due to the fact that as the σ_r increases, there is a greater possibility of getting more regular itemsets compared to low σ_r values. This is why the value of σ_r is specified for each datasets in the experiments is not equal. The value of regularity threshold is set between 1 to 10% of total number of transactions in database. The values of k are varied between 50 to $10,000$ to see the performance of the proposed algorithm for the small and large value of k.

5.7.2 Compactness of using interval tidset representation

Based on the interval tidset representation, TKRIMIT can generate more concise tidsets than the original tidsets (used in MTKPP and TKRIMPE) since the former maintains only the first and the last tids of the two or more consecutive continuous tids by using only one positive and one negative integer, respectively. Meanwhile, the latter collects all of tids that each itemset occurs. Thus, the number of tids that TKRIMIT can reduce on dense and sparse datasets are considered. To depict the result, the numbers of reduced tids by TKRIMPE is shown in Figure 5.4 to Figure 5.14.

It is observed from Figure 5.4 to Figure 5.9 that the TKRIMIT can reduce a lot of tids to store in the interval tidset on dense datasets. For the small values of k, TKRIMIT can reduce up to $86,000,000$ tids whereas the number of reduced tids is $713,000,000$ with the large values of k. However, as shown in Figure 5.10 to Figure 5.14, TKRIMIT cannot significantly reduce the number of maintained tids from MTKPP and TKRIMPE on sparse datsets. Because of the characteristics of sparse datasets, most of itemsets do not occur in consecutive continuous tids. Thus, the number of reduced tids is in range $[500, 34,000]$ for the small value of k and $[800, 42,000]$ for the large values of k.

5.7.3 Execution time

From Figures 5.15 to Figure 5.32, the evaluation results for real dense datasets are reported. From these figures, the performance of TKRIMIT is different from other algorithms such as MTKPP and TKRIMPE using normal tidsets (*i.e.* maintaining all of tids that each itemset

occurs). It can see from these figures that the performance of using interval tidset is better than using normal tidset on the small and large value of k. In addition, on dense dataset, each itemset occurs almost every transaction or occurs very frequent. Then, TKRIMIT can take the advantage from the use of interval tidset representation. However, on mushroom dataset and the large values of k, TKRIMIT cannot significantly reduce the runtime from MTKPP and TKRIMPE. This is because mushroom has a small number of transactions, then TKRIMIT cannot yield the benefit of grouping tids together.

Meanwhile, the execution time on sparse datasets is shown in Figure 5.33 to Figure 5.47. Note that the performance of TKRIMIT is similar with MTKPP and run slower than TKRIMPE from these figures. TKRIMIT cannot take the advantage from database partitioning and support estimation techniques as used in TKRIMPE. Due to each itemset in sparse datasets occurs not often and it does not occurs in the consecutive continuous transactions, TKRIMIT cannot take the advantages from grouping consecutive continuous tids from sparse datasets.

5.7.4 Memory consumption

As mentioned above, TKRIMIT can essentially reduce the number of maintained tids during mining. In this subsection, the memory usage of TKRIMIT is also investigated by comparing with MTKPP and TKRIMPE.

Figure 5.48 to Figure 5.53 show the memory usage of TKRIMIT, MTKPP and TKRIMPE on real dense datasets. From this figure, TKRIMIT can significantly save the memory usage from MTKPP and TKRIMPE. For the large value of k, TKRIMIT consumes over two orders of magnitude less memory than MTKPP and TKRIMPE. The memory usage of TKRIMIT increases linearly as the number of desired itemsets increases while memory used by MTKPP and TKRIMPE increase dramatically. This is because MTKPP and TKRIMPE use normal tidset that maintain all tids occurring in each itemset. The memory usage of MTKPP and TKRIMPE depend on the support (*i.e.* number of tids that each itemset occurs) of each itemsets. Meanwhile, TKRIMIT can take the advantage from the use of interval tidset representation which group several consecutive continuous tids together.

As shown in Figure 5.54 to Figure 5.58, the required memory of TKRIMIT on sparse datasets is examined. Followed by this figure, the memory usage of the three algorithms is quite similar. This is because each itemset on sparse dataset does not occur frequently and consecutively continuous. Then, TKRIMIT cannot group several tids together.

5.7.5 Scalability test

To study the scalability of the TKRIMIT algorithm, the execution time and memory consumption of TKRIMIIT are considered by comparing with MTKPP and TKRIMPE when the size of database increases. The kosarak dataset which is a huge dataset with a large number of distinct of items $(41,270)$ and transactions $(990,002)$ is used to test scalability by varying the number of transactions. The database is first divided into six portions (i.e. $100,000$, $200,000$, $400,000$, $600,000$, $800,000$ and $990,002$ transactions). Then, the performance of TKRIMIT is investigated on each portion. The values of desired itemsets (k) are also varied into small (i.e. 50, 100, 200, 300, 400, and 500) and large (i.e. $1,000$, $2,000$, $4,000$, $6,000$, $8,000$, $10,000$) values. Lastly, the regularity threshold is fixed to 6% of number of transactions in each portion.

In Figure 5.59 and Figure 5.60, the scalability of TKRIMIT, MTKPP and TKRIMPE are tested in terms of runtime with different number of transactions in the database. From these figures, the runtime of TKRIMIT scales linearly increase when the size of database increases. Based on the interval tidset representation, TKRIMIT can group many consecutive tids together and then TKRIMIT has a better scalability than that of MTKPP on the small and large values of k. Meanwhile, TKRIMIT cannot significantly reduce the runtime from TKRIMPE because TKRIMIT uses only grouping technique (i.e. interval tidset representation) and cannot take the advantages from the database partitioning and support estimation techniques.

Figures 5.59 and 5.60 also plot the high water mark of space usage of TKRIMIT, TKRIMPE and MTKPP with varying the size of the database. The three algorithms have linear scalability and TKRIMIT is a clear winner. Therefore, it can be seen from the figures that by based on the interval tidset representation, TKRIMIT is very efficient and scalable in terms of space usage with respect to the number of itemsets to be mined and the number of transactions in database.

5.8 Summary

This chapter have presented a new efficient and scalable algorithm named *TKRIMIT (Top-K Regular-frequent Itemsets Mining based on Interval Tidset representation)* to discover a set of k regular itemsets with the highest supports. A new concise representation, called *interval transaction-ids set (interval tidset)*, has also introduced. Based on the interval tidset representation, a set of tids that each itemset occuring consecutively continuous is transformed and

compressed to interval tids by using only one positive and negative integer. The top-*k* regular-frequent itemsets are found by intersection of interval tidsets along the order of top-*k* list. Besides, TKRIMIT is based on a best-first search startegy that can help TKRIMIT algorithm to raise quickly the support of the k^{th} itemsets in the sorted top-*k* list which help the proposed algorithm to prune the search space.

The analysis and experiment results show that TKRIMIT achieves high performance on both dense and sparse datasets. The proposed algorithm delivers competitive performance and, especially for dense datasets, outperforms MTKPP and TKRIMPE which are currently the most efficient algorithm for top-*k* regular-frequent. Based on this study, it is can be claimed that the proposed algorithm are superior to MTKPP and TKRIMPE on both the small and large values of k when the datasets are dense.

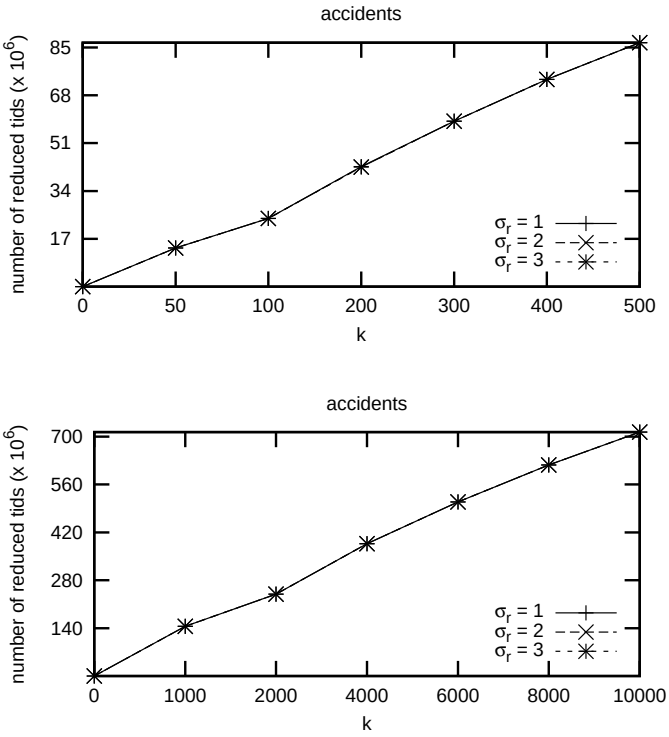


Figure 5.4: The number of reduced tids from TKRIMIT on *accidents* datasets

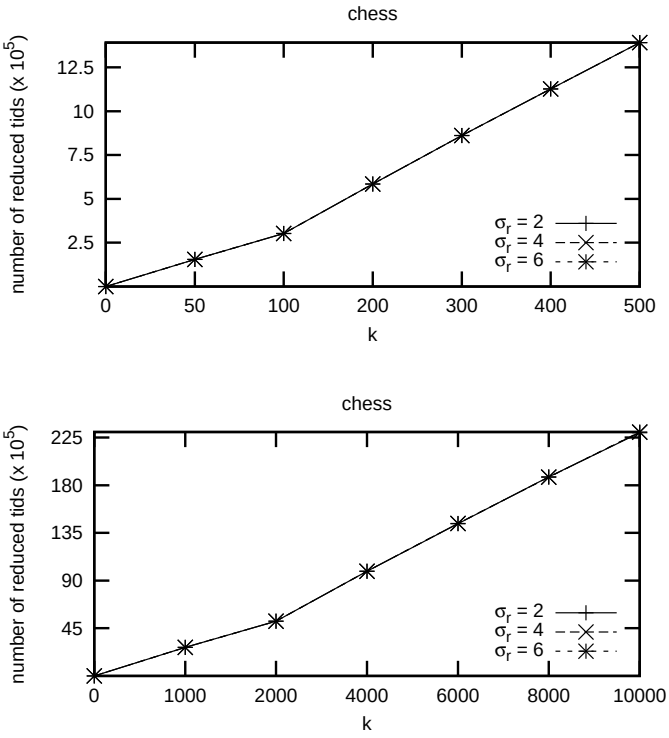


Figure 5.5: The number of reduced tids from TKRIMIT on *chess* datasets

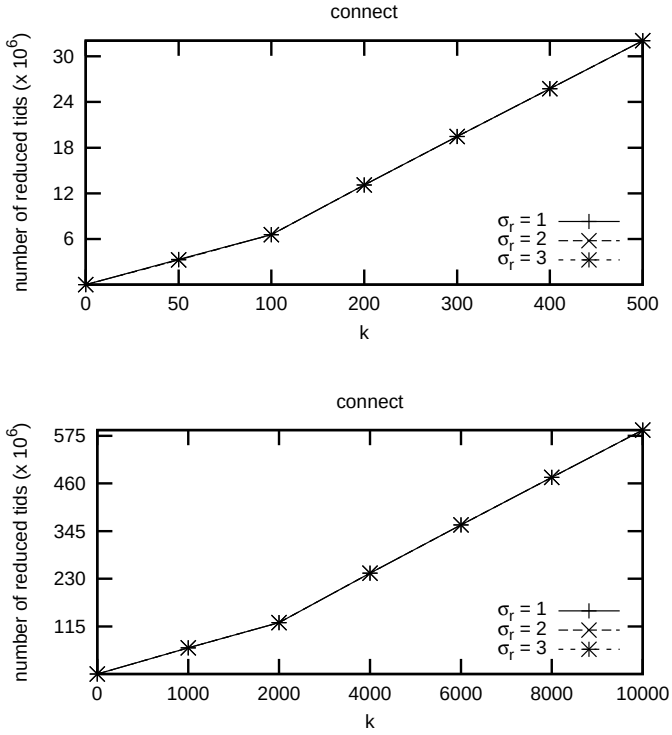


Figure 5.6: The number of reduced tids from TKRIMIT on *connect* datasets

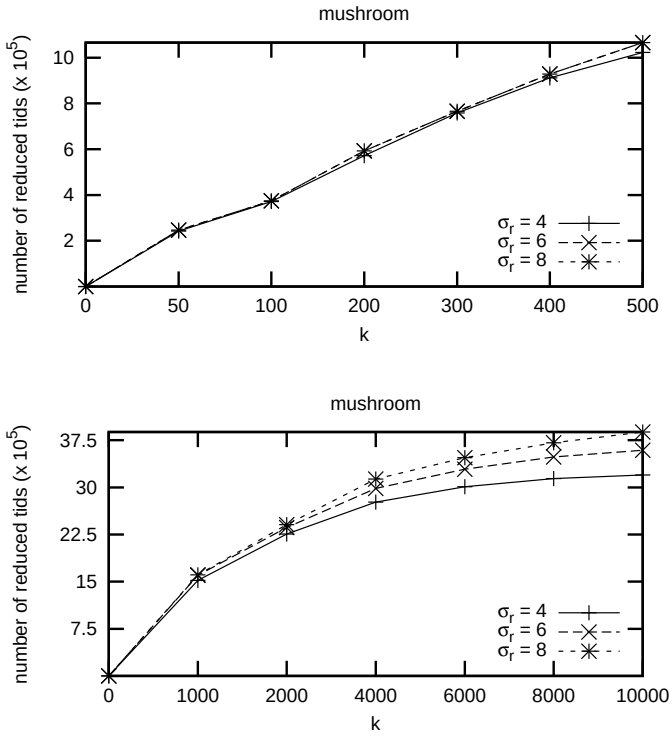


Figure 5.7: The number of reduced tids from TKRIMIT on *mushroom* datasets

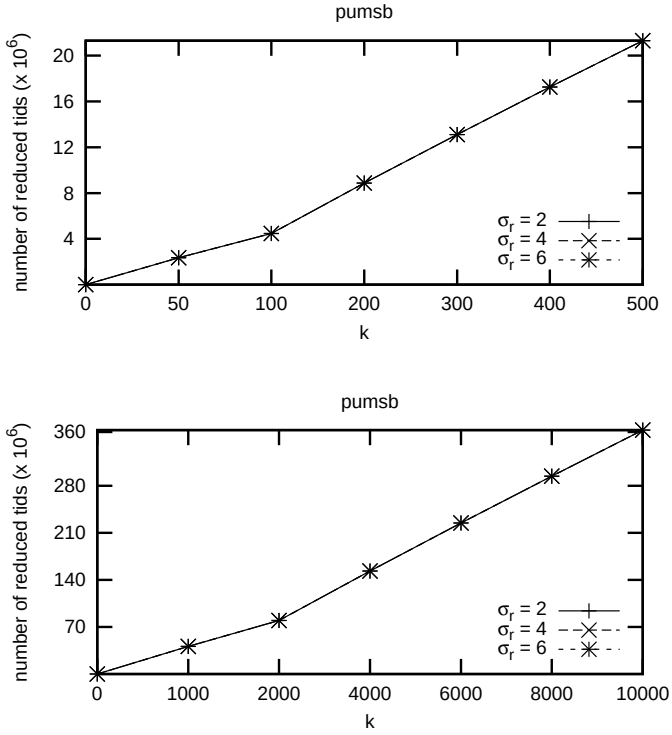


Figure 5.8: The number of reduced tids from TKRIMIT on *pumsb* datasets

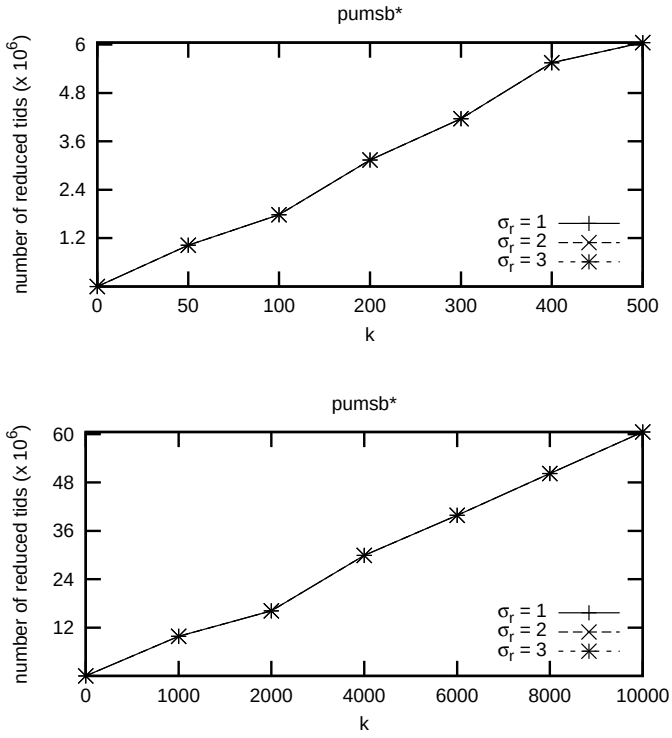


Figure 5.9: The number of reduced tids from TKRIMIT on *pumsb** datasets

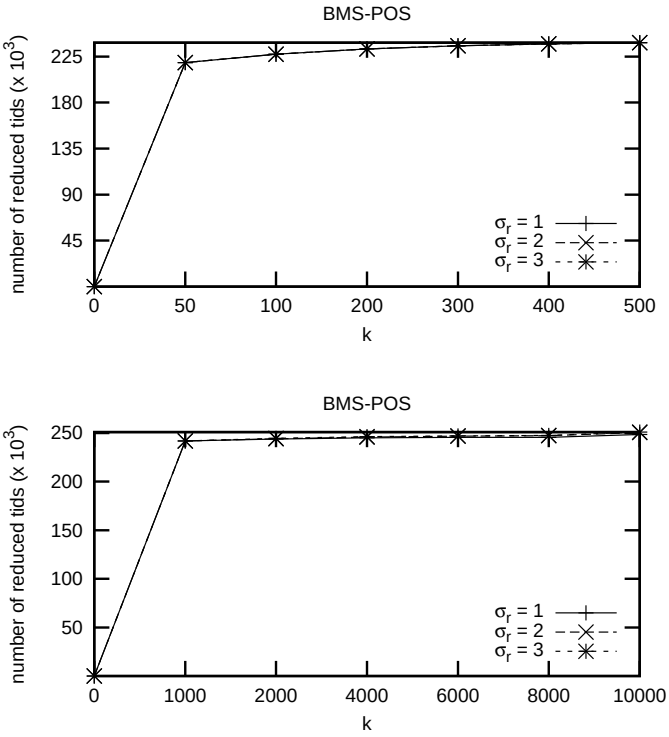


Figure 5.10: The number of reduced tids from TKRIMIT on *BMS-POS* datasets

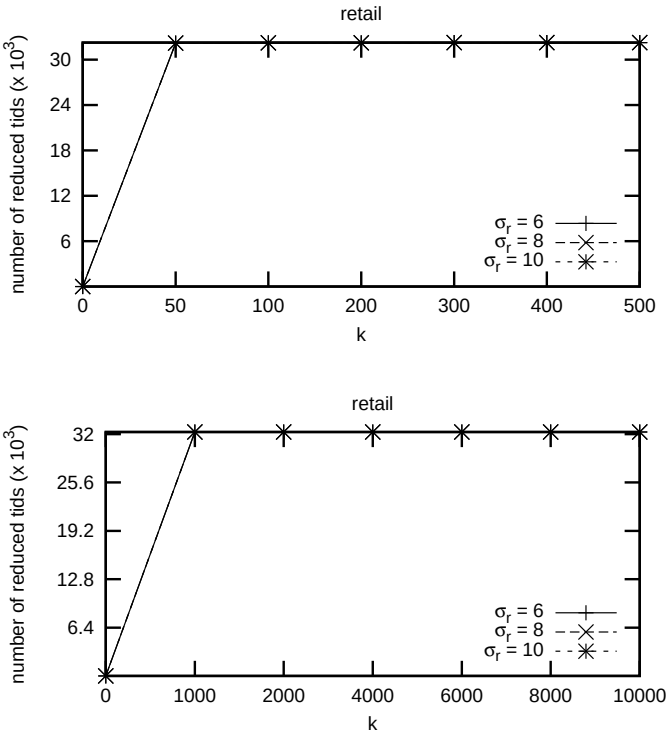


Figure 5.11: The number of reduced tids from TKRIMIT on *retail* datasets

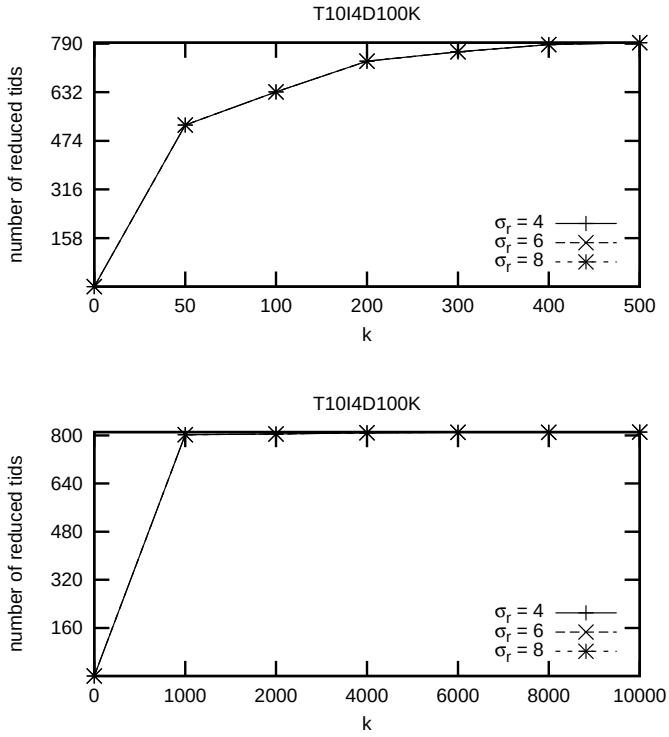


Figure 5.12: The number of reduced tids from TKRIMIT on *T10I4D100K* datasets

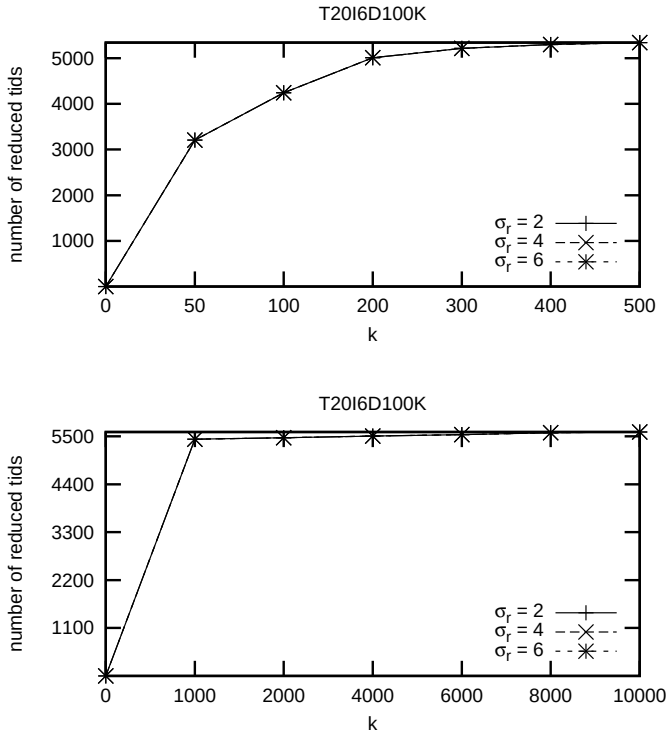


Figure 5.13: The number of reduced tids from TKRIMIT on *T20I6D100K* datasets

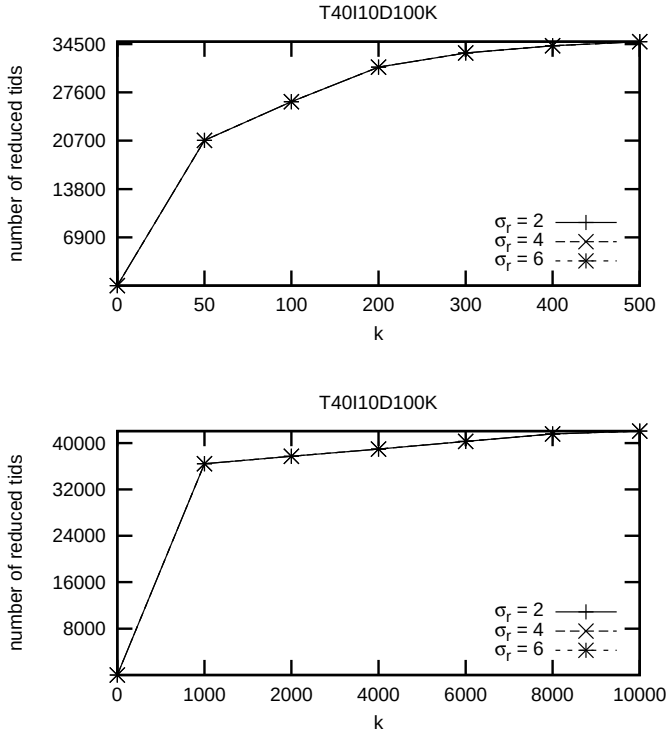


Figure 5.14: The number of reduced tids from TKRIMIT on *T40I10D100K* datasets

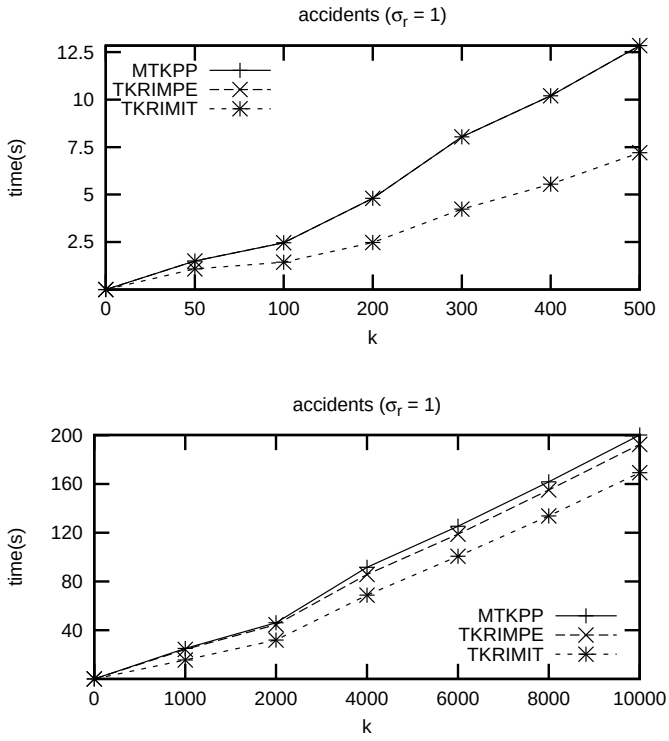


Figure 5.15: Runtime of TKRIMIT on *accidents* ($\sigma_r = 1\%$)

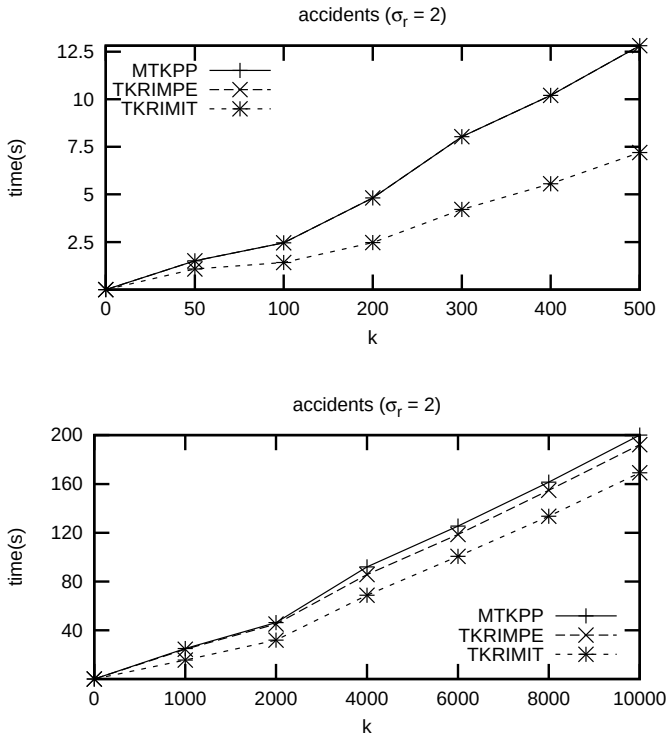


Figure 5.16: Runtime of TKRIMIT on *accidents* ($\sigma_r = 2\%$)

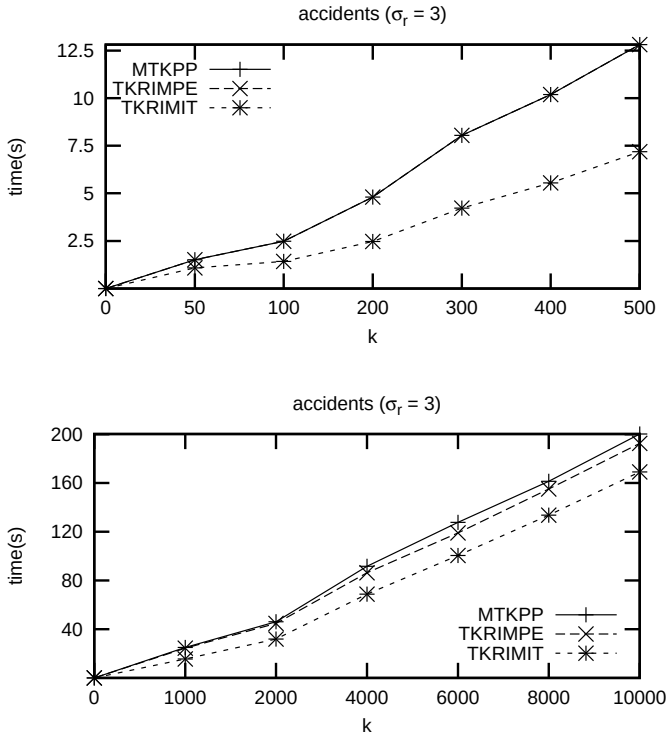


Figure 5.17: Runtime of TKRIMIT on *accidents* ($\sigma_r = 3\%$)

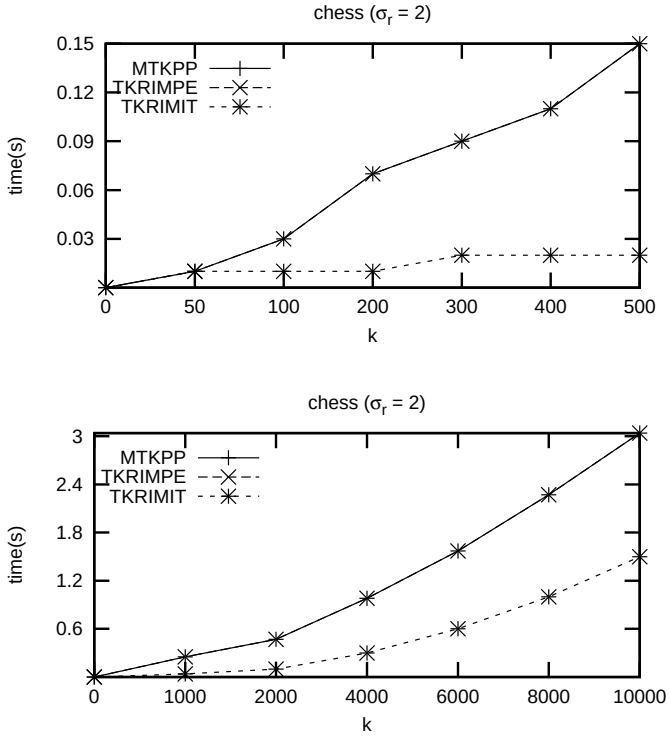


Figure 5.18: Runtime of TKRIMIT on *chess* ($\sigma_r = 2\%$)

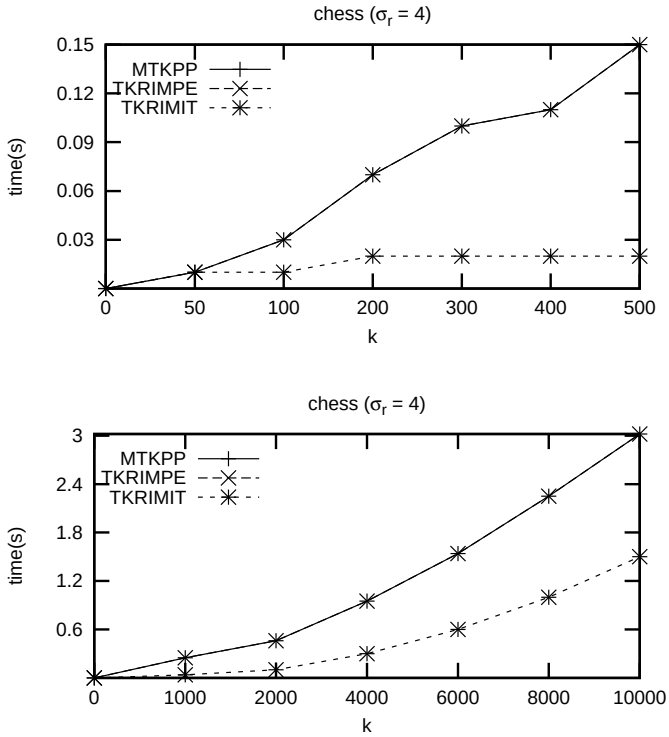


Figure 5.19: Runtime of TKRIMIT on *chess* ($\sigma_r = 4\%$)

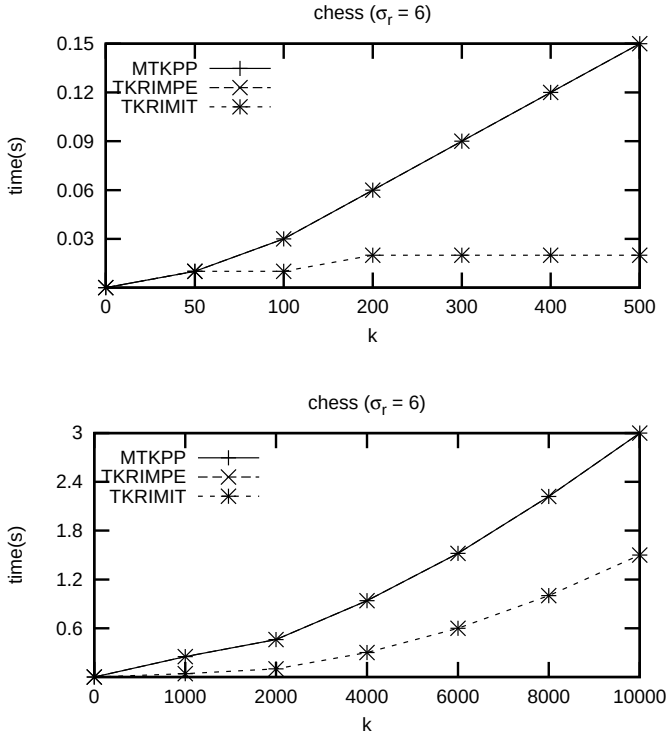


Figure 5.20: Runtime of TKRIMIT on *chess* ($\sigma_r = 6\%$)

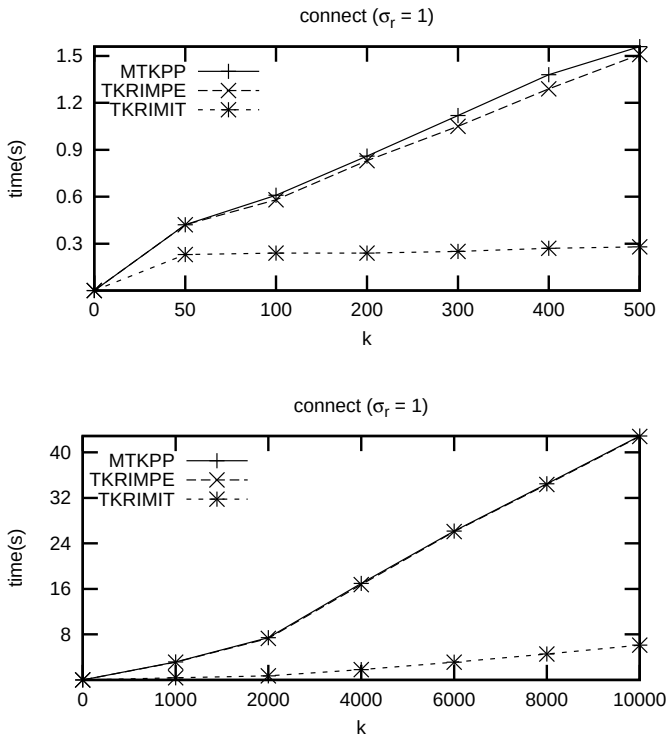


Figure 5.21: Runtime of TKRIMIT on *connect* ($\sigma_r = 1\%$)

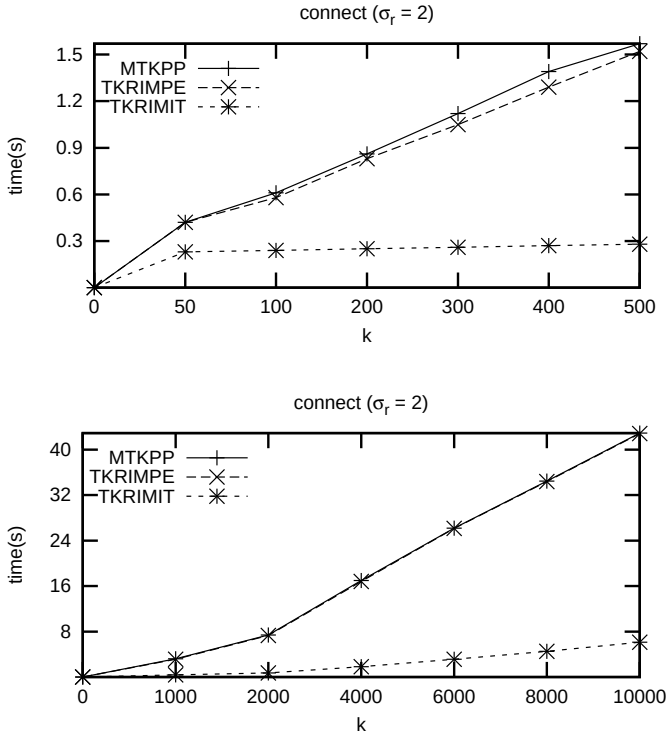


Figure 5.22: Runtime of TKRIMIT on *connect* ($\sigma_r = 2\%$)

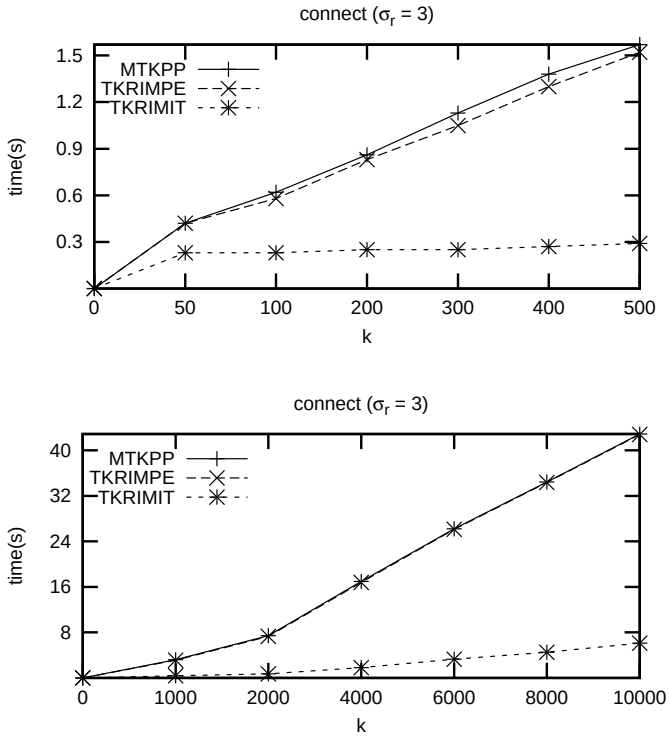


Figure 5.23: Runtime of TKRIMIT on *connect* ($\sigma_r = 3\%$)

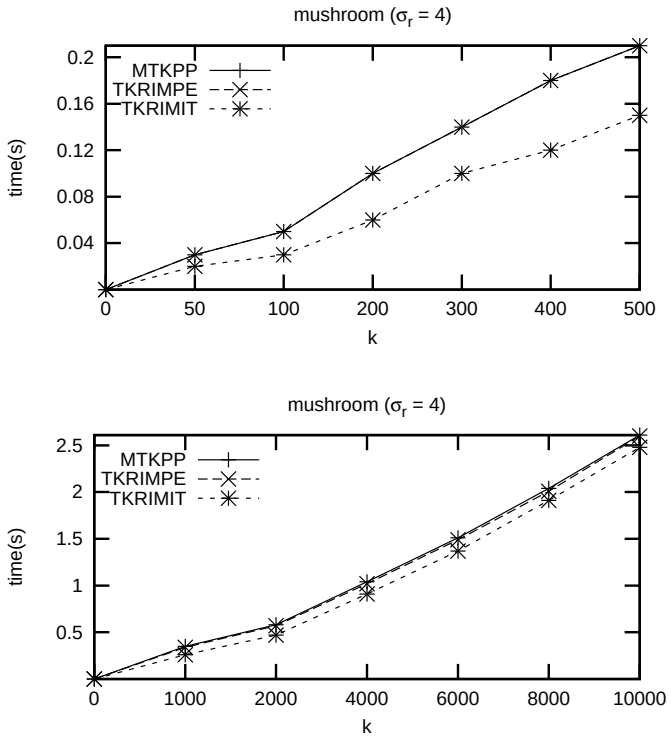


Figure 5.24: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 4\%$)

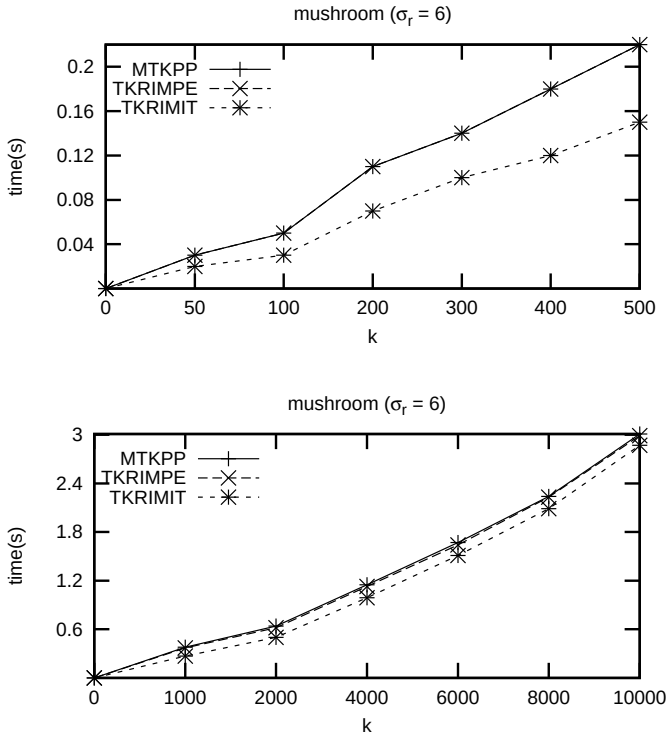


Figure 5.25: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 6\%$)

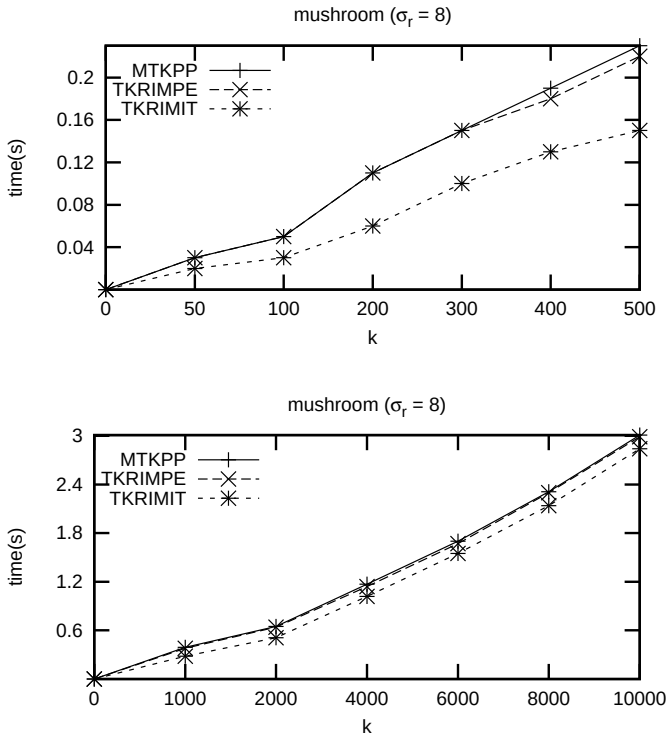


Figure 5.26: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 8\%$)

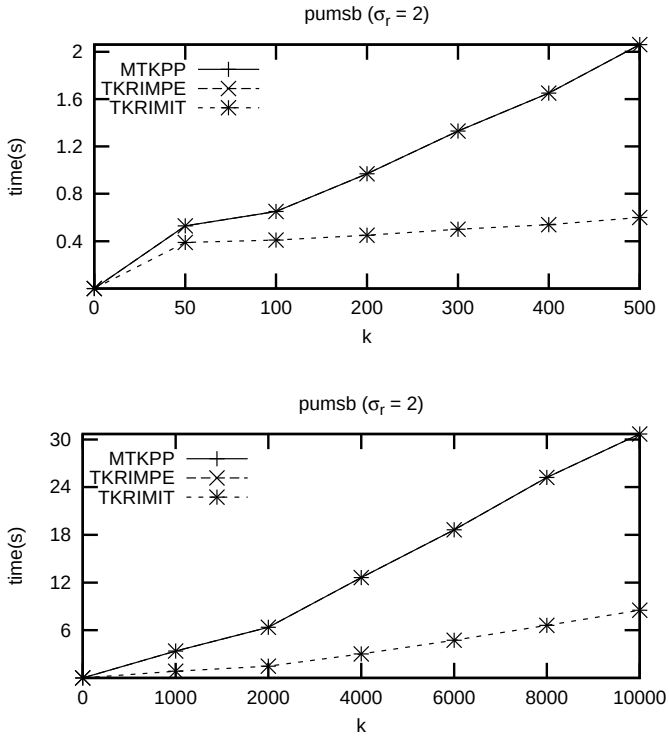


Figure 5.27: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 2\%$)

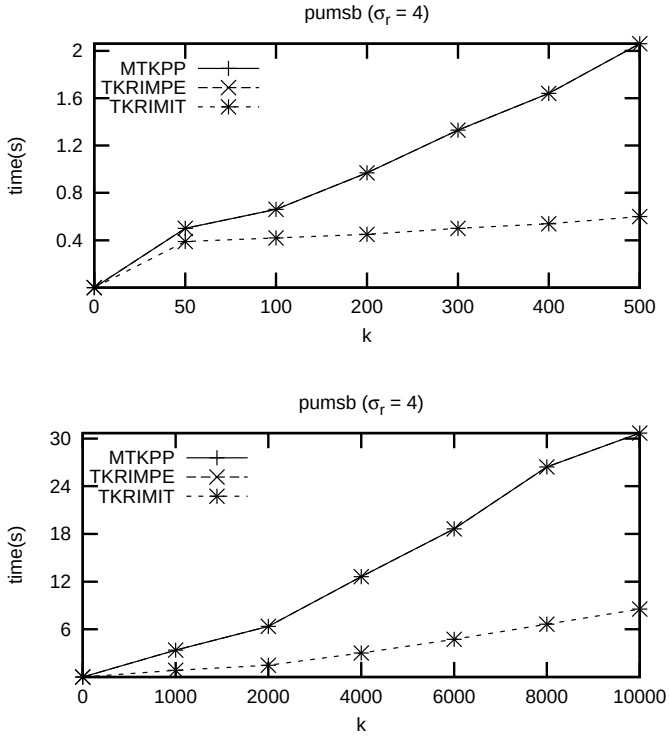


Figure 5.28: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 4\%$)

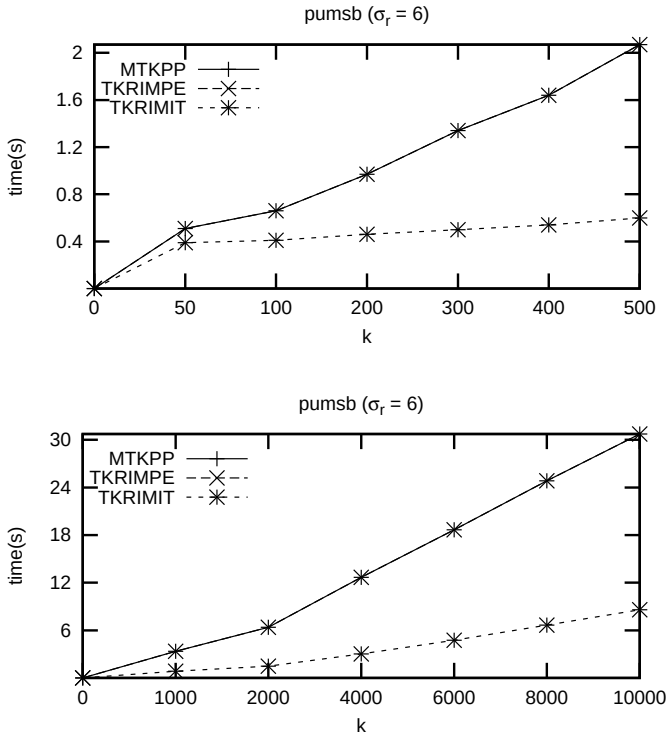


Figure 5.29: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 6\%$)

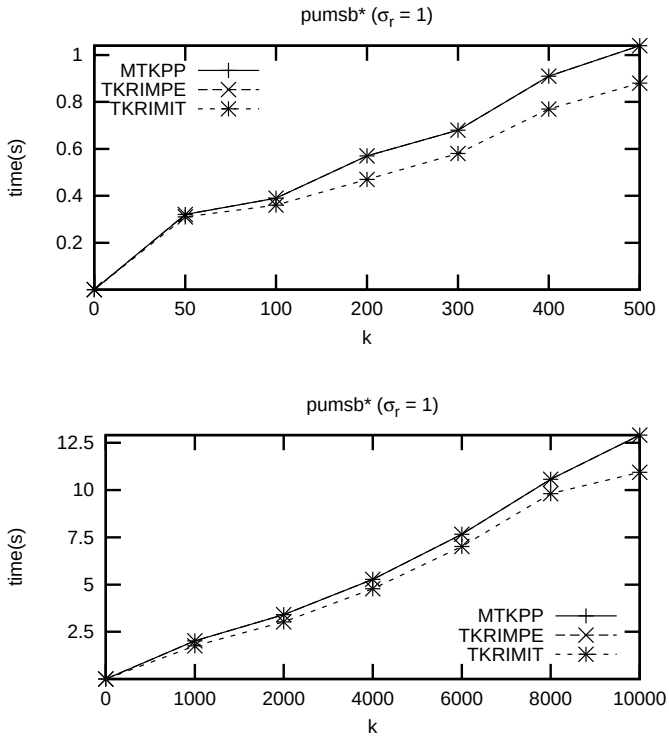


Figure 5.30: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 1\%$)

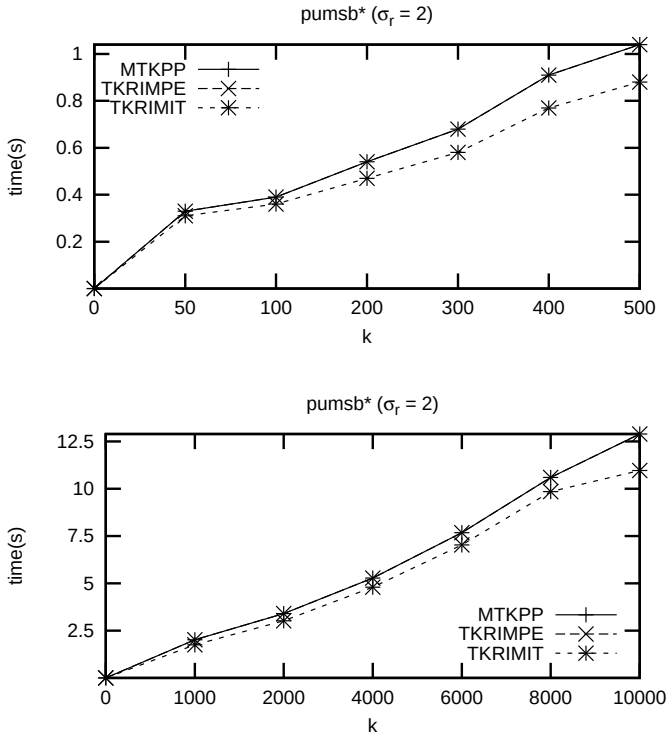


Figure 5.31: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 2\%$)

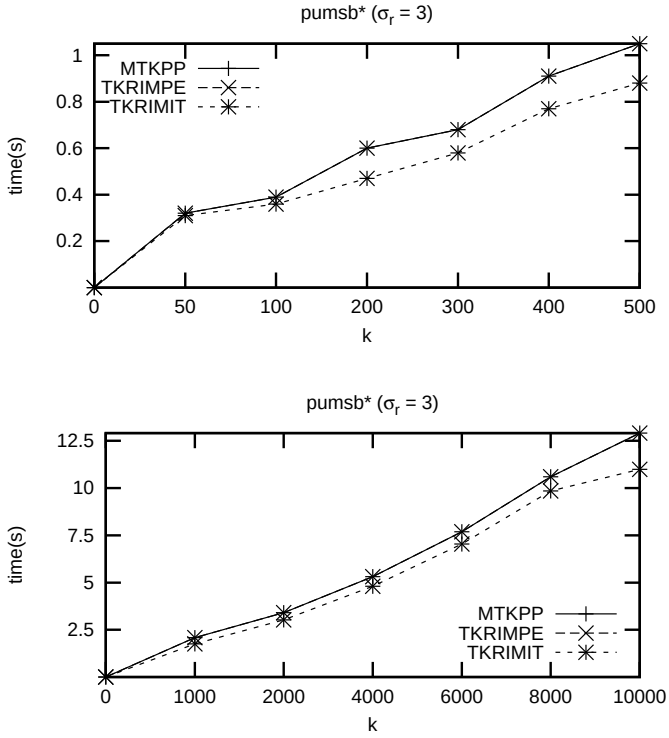


Figure 5.32: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 3\%$)

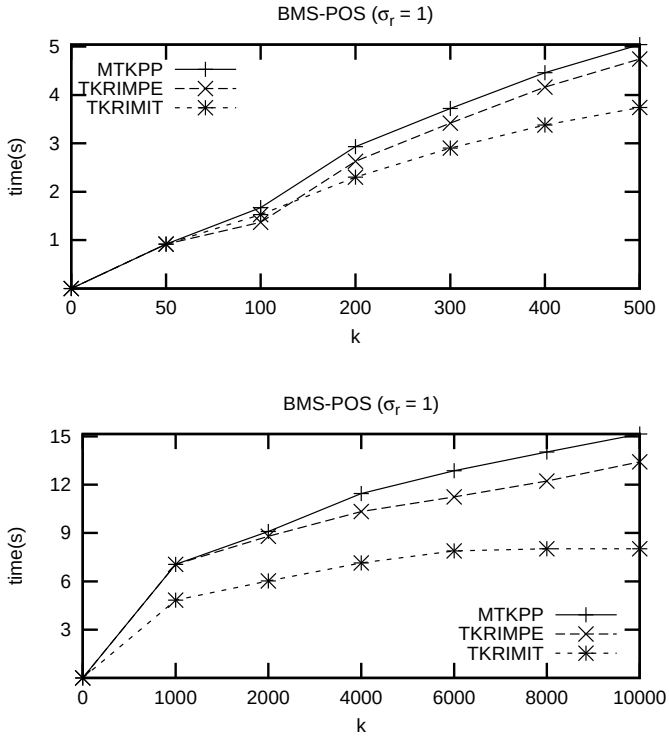


Figure 5.33: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 1\%$)

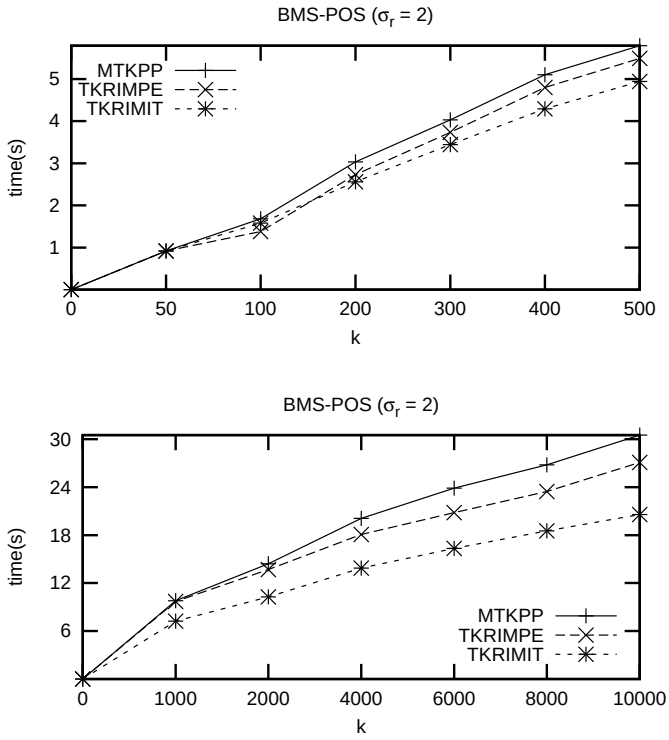


Figure 5.34: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 2\%$)

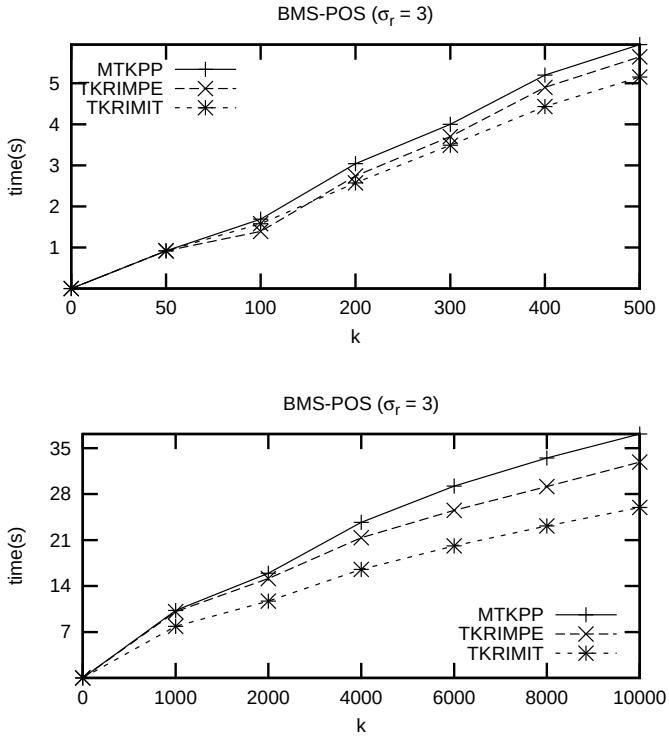


Figure 5.35: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 3\%$)

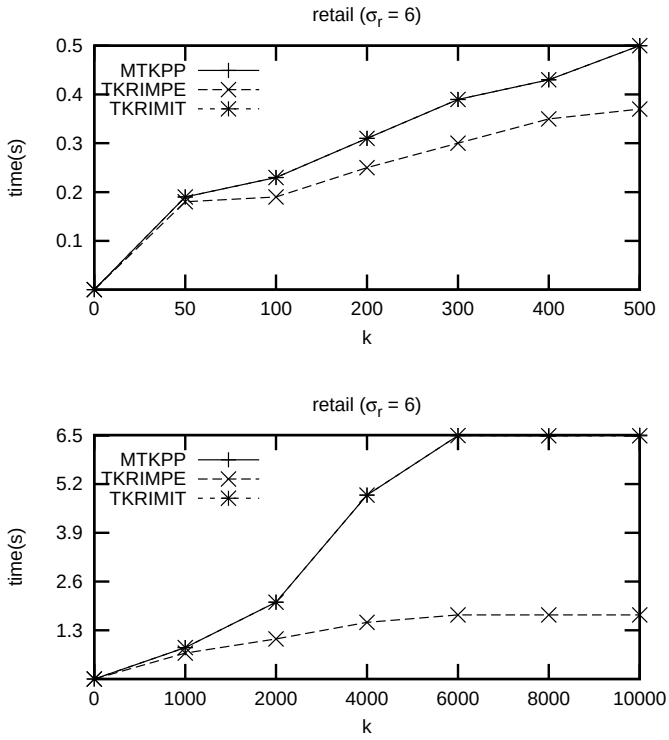


Figure 5.36: Runtime of TKRIMIT on *retail* ($\sigma_r = 6\%$)

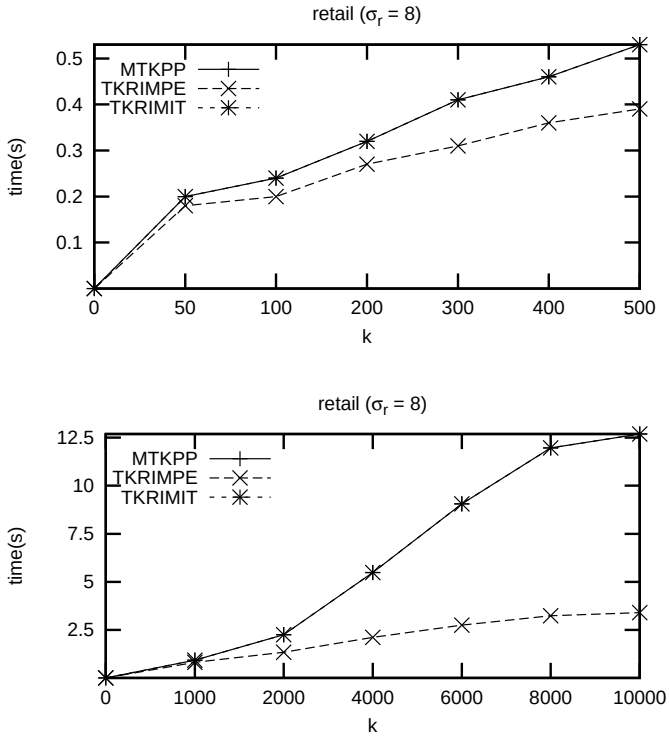


Figure 5.37: Runtime of TKRIMIT on *retail* ($\sigma_r = 8\%$)

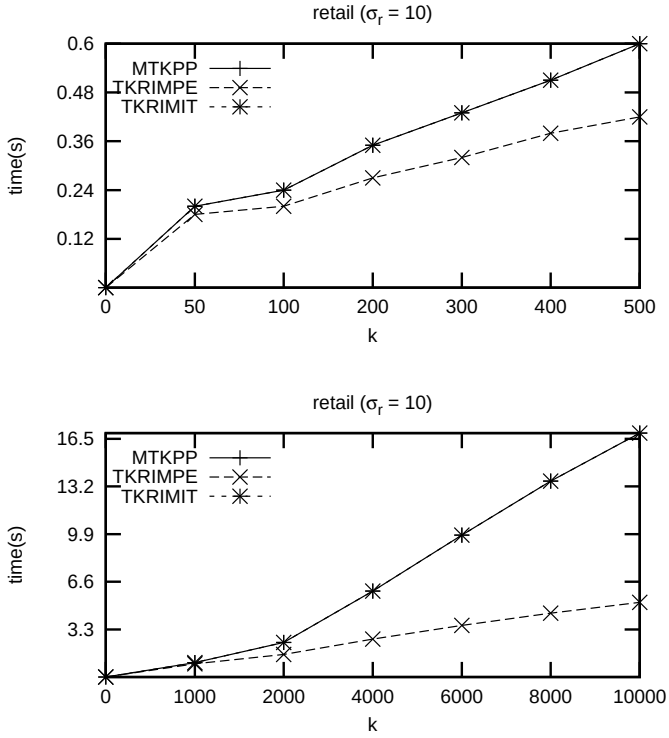


Figure 5.38: Runtime of TKRIMIT on *retail* ($\sigma_r = 10\%$)

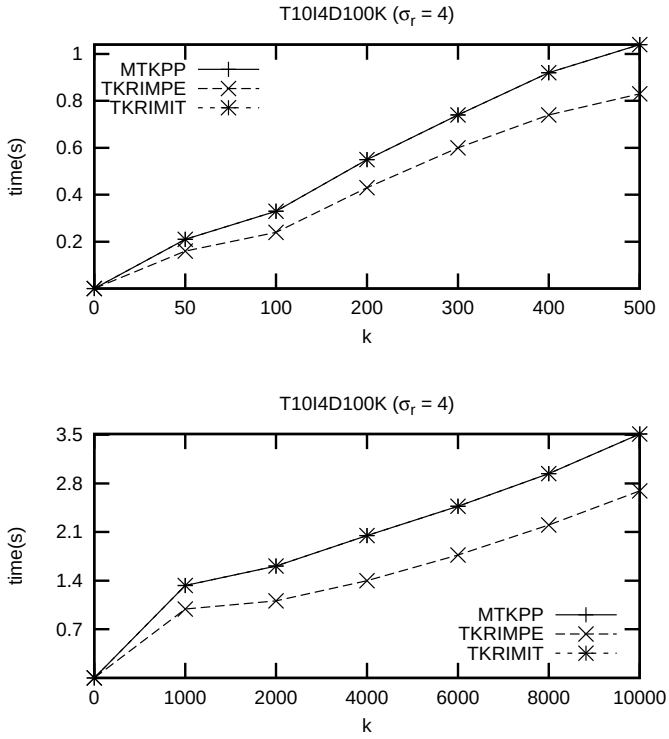


Figure 5.39: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 4\%$)

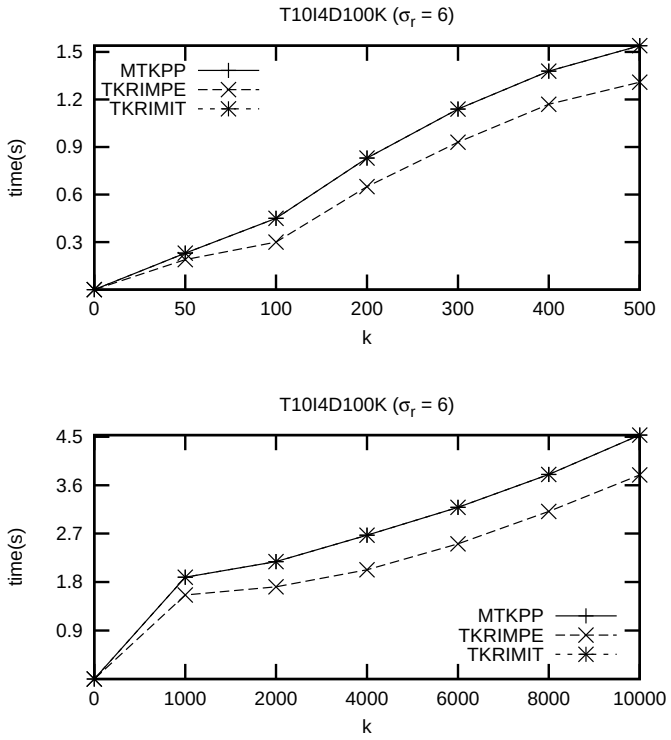


Figure 5.40: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 6\%$)

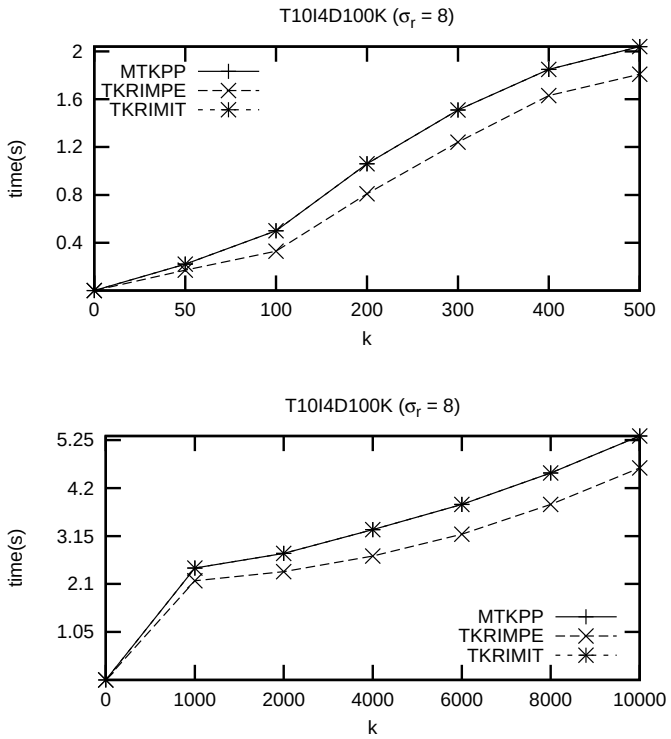


Figure 5.41: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 8\%$)

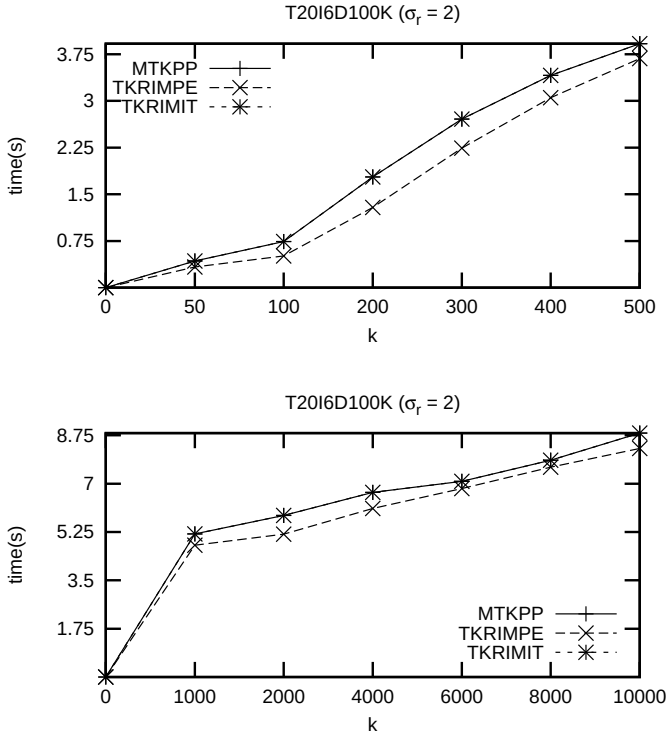


Figure 5.42: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 2\%$)

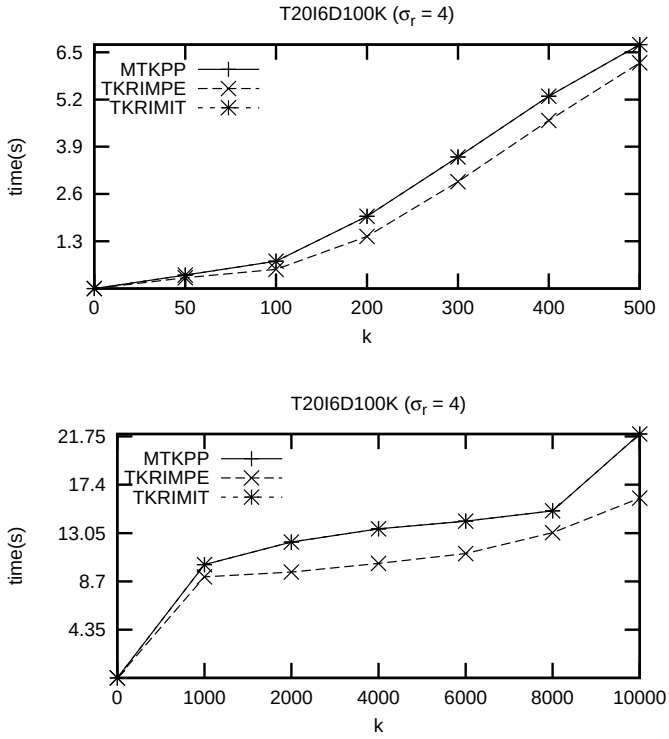


Figure 5.43: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 4\%$)

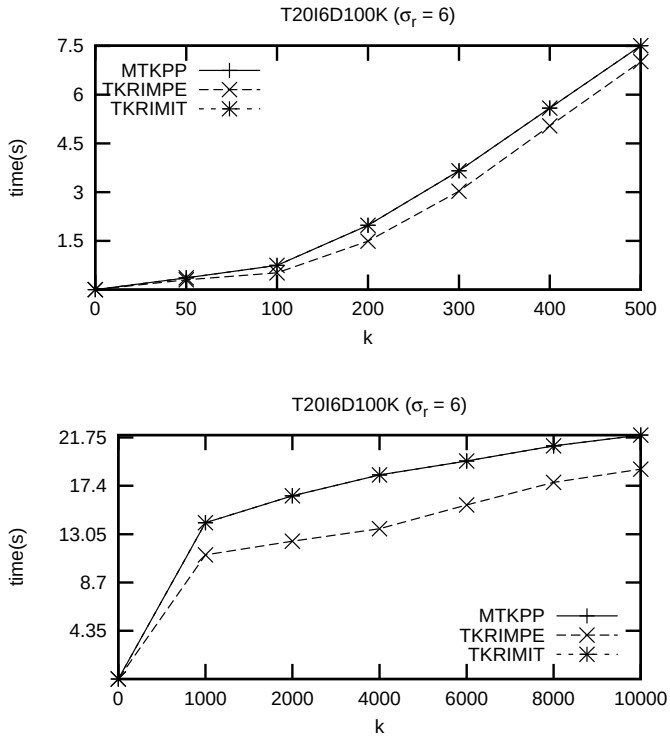


Figure 5.44: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 6\%$)

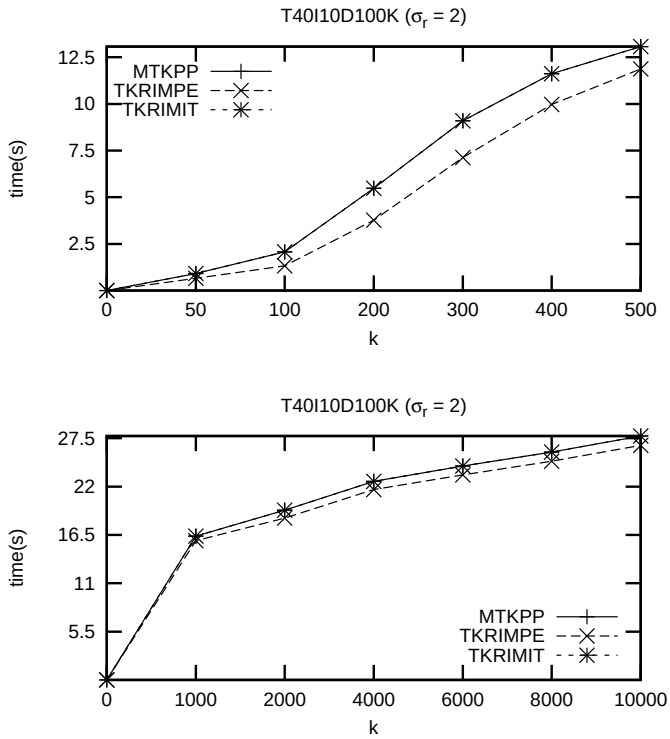


Figure 5.45: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 2\%$)

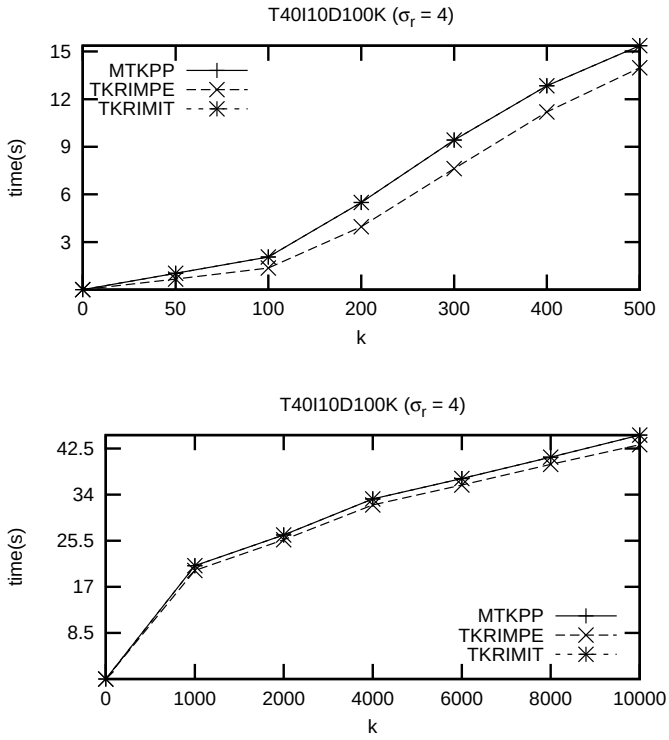


Figure 5.46: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 4\%$)

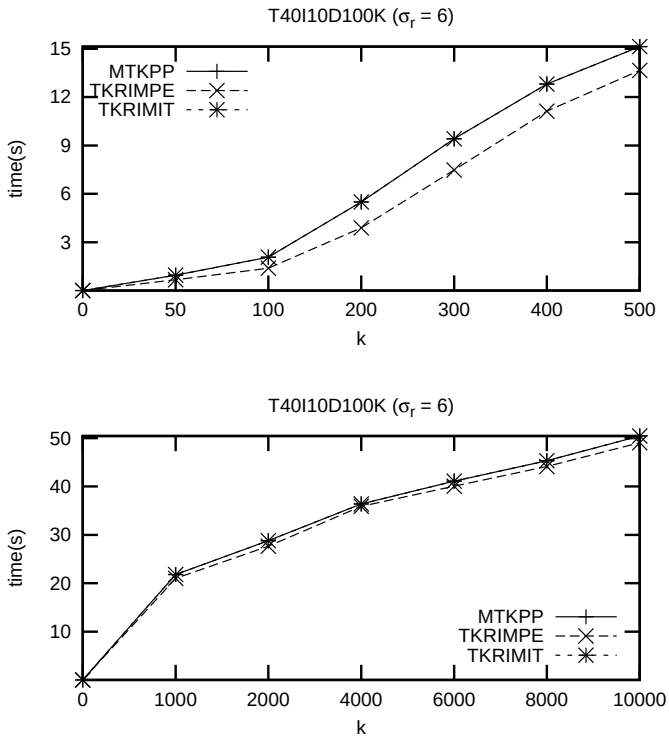


Figure 5.47: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 6\%$)

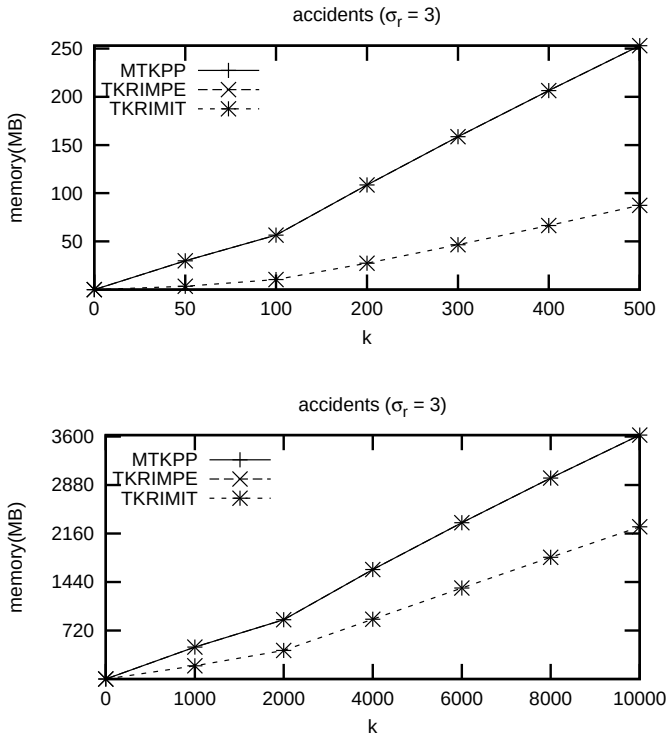


Figure 5.48: Memory usage of TKRIMIT on *accidents*

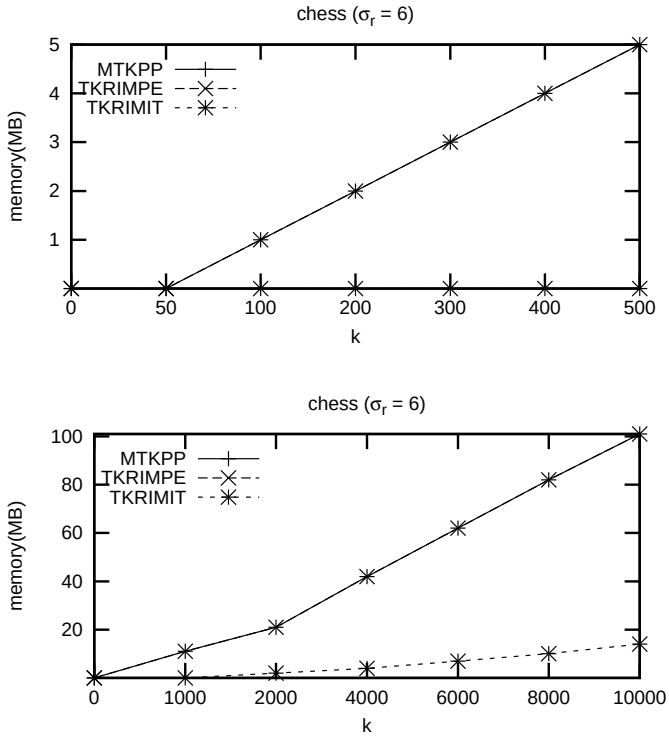


Figure 5.49: Memory usage of TKRIMIT on *chess*

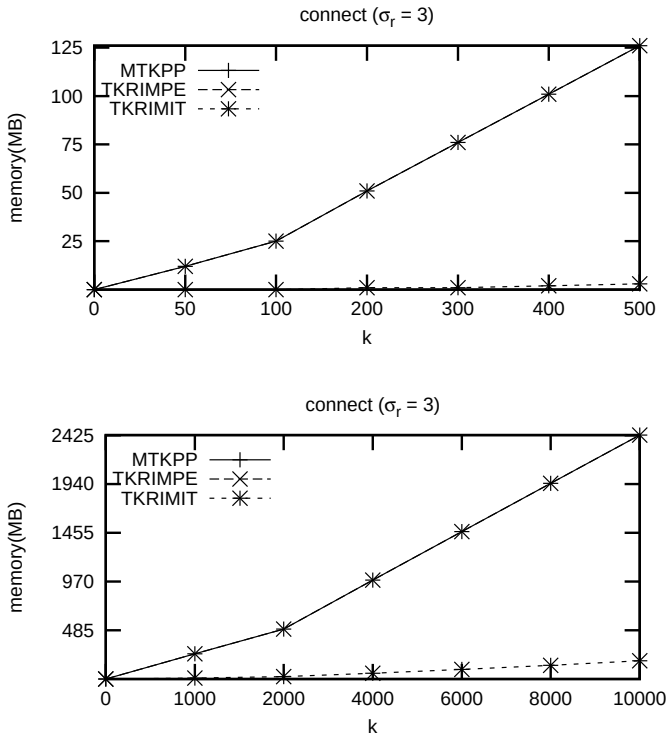


Figure 5.50: Memory usage of TKRIMIT on *connect*

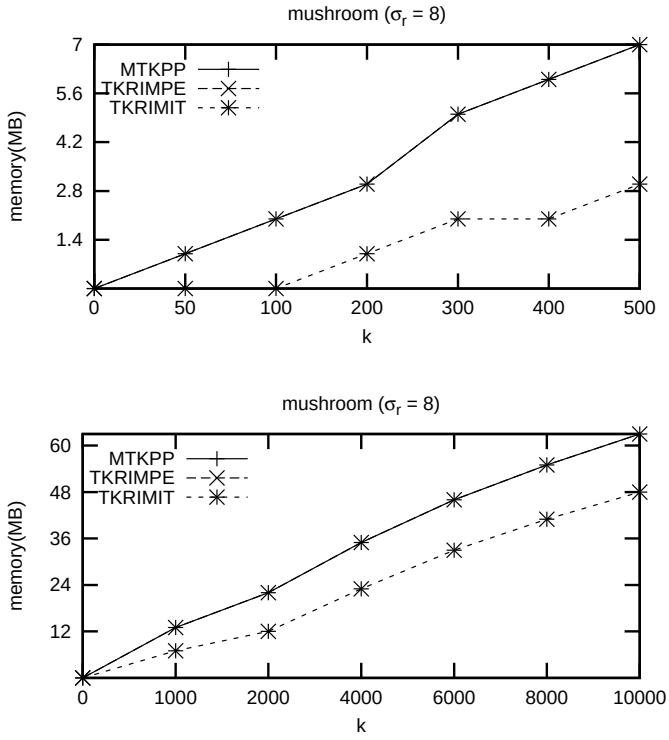


Figure 5.51: Memory usage of TKRIMIT on *mushroom*

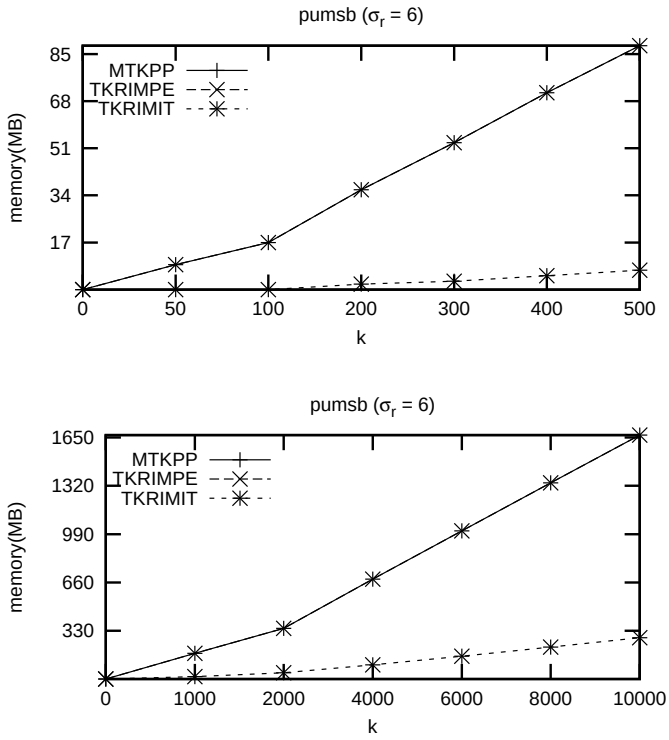


Figure 5.52: Memory usage of TKRIMIT on *pumsb*

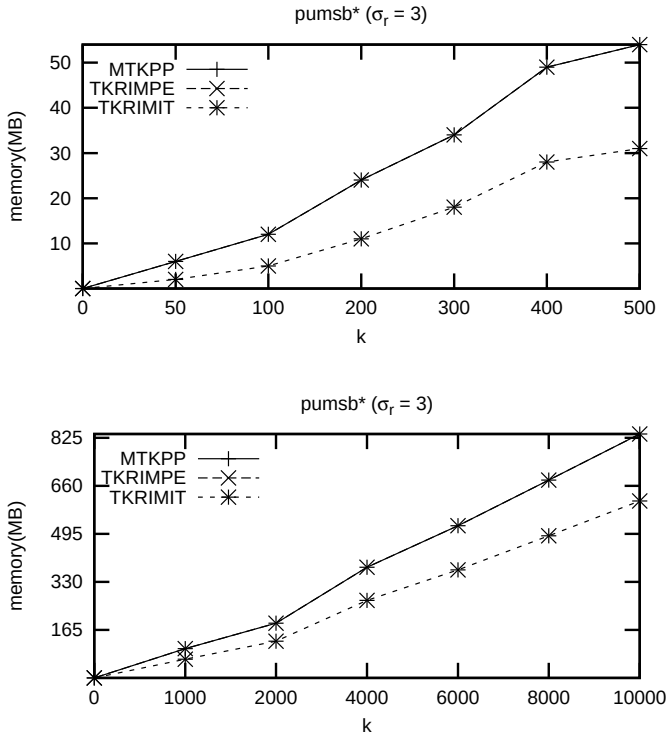


Figure 5.53: Memory usage of TKRIMIT on *pumsb**

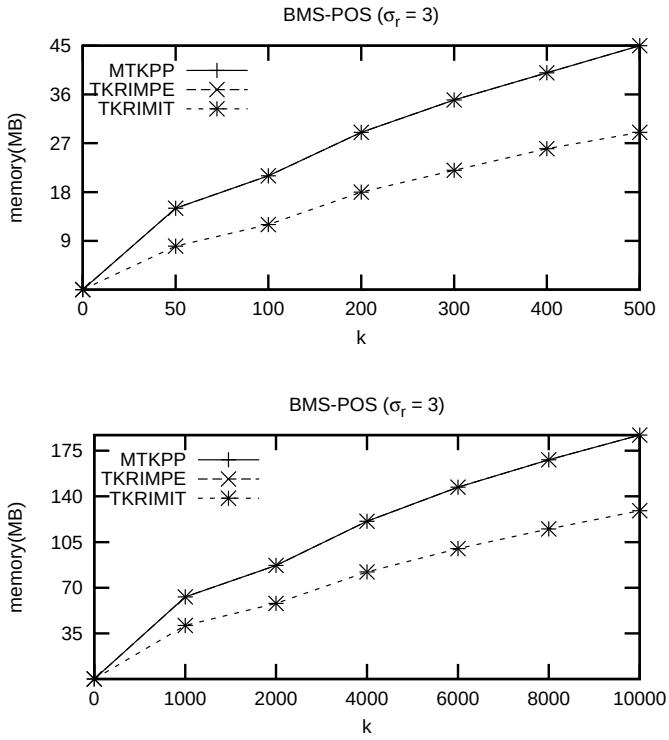


Figure 5.54: Memory usage of TKRIMIT on *BMS-POS*

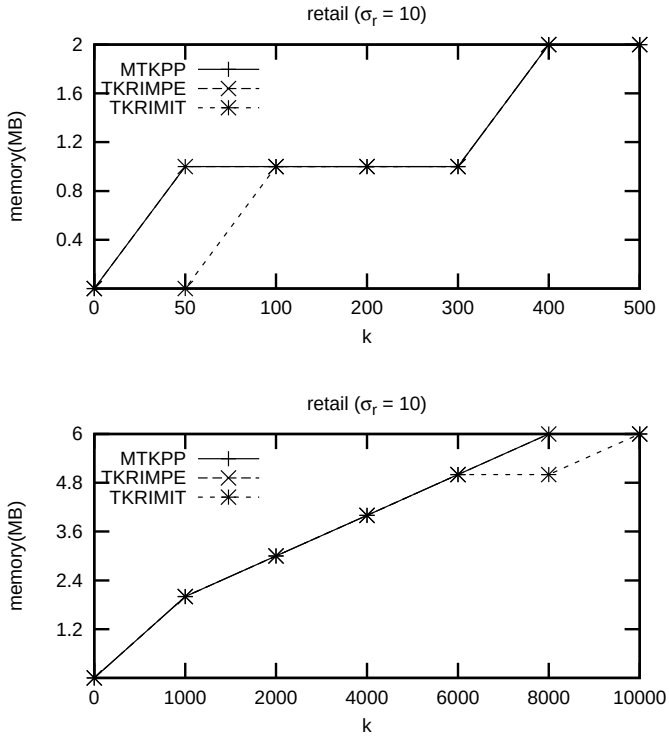


Figure 5.55: Memory usage of TKRIMIT on *retail*

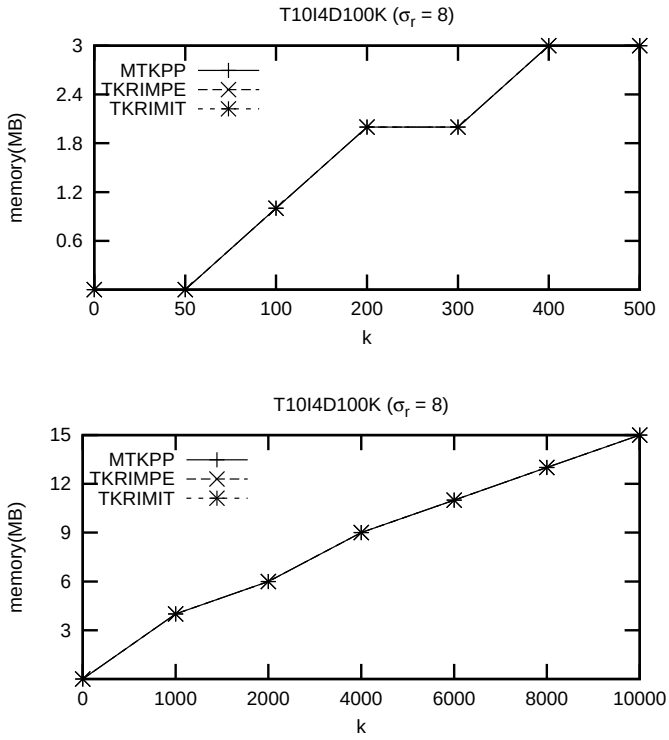


Figure 5.56: Memory usage of TKRIMIT on *T10I4D100K*

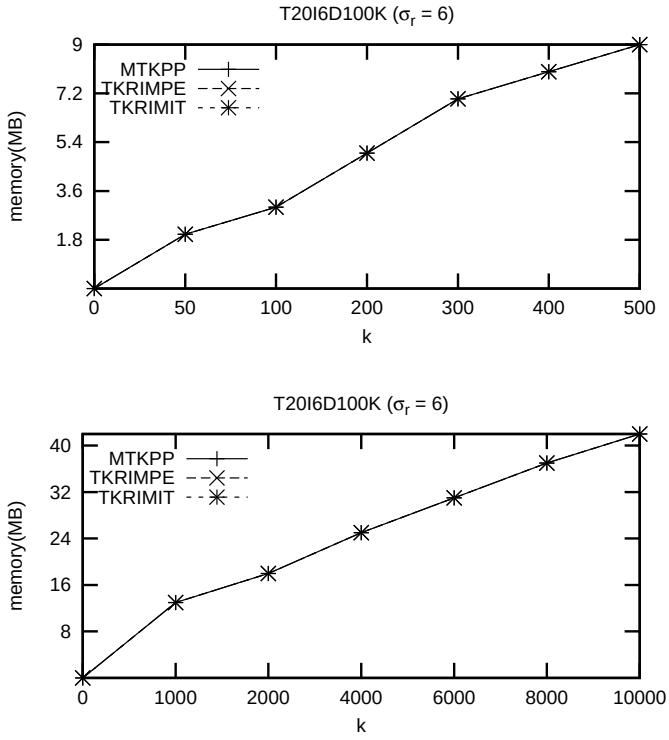


Figure 5.57: Memory usage of TKRIMIT on *T20I6D100K*

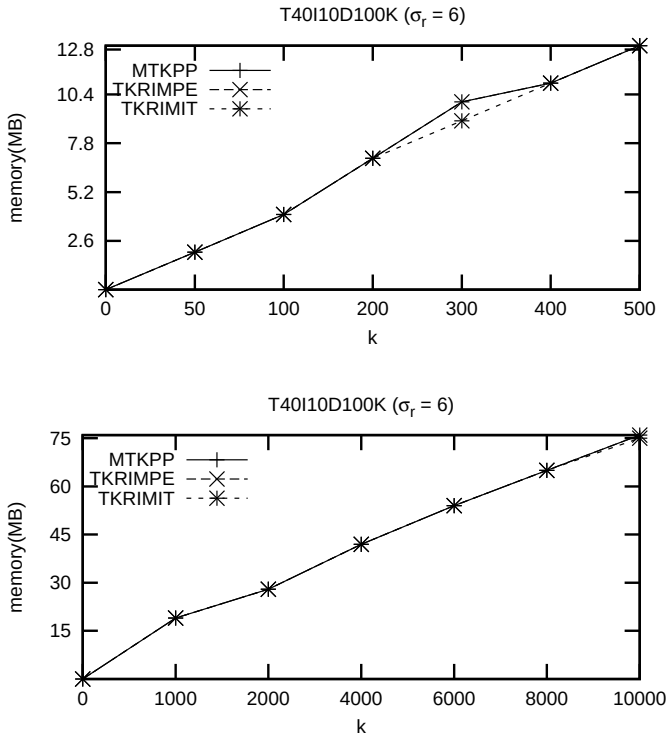


Figure 5.58: Memory usage of TKRIMIT on *T40I10D100K*

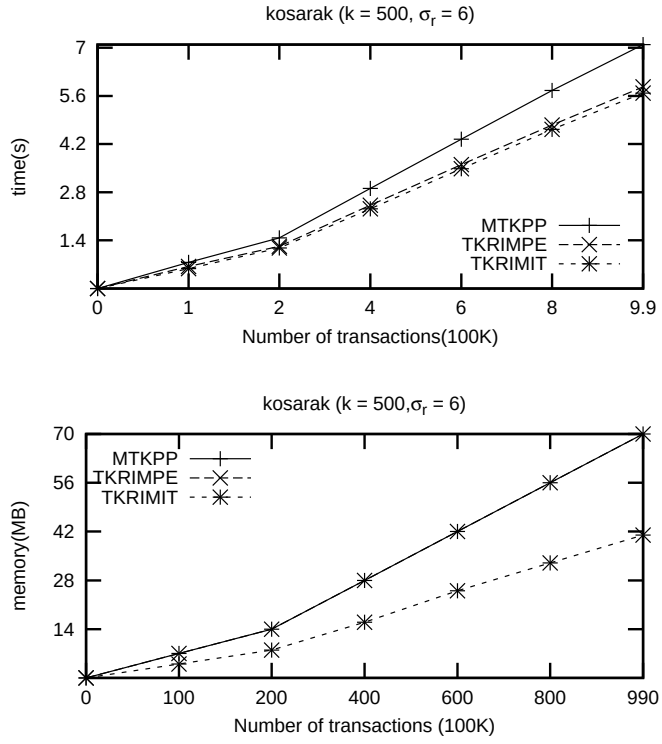


Figure 5.59: Scalability of TKRIMIT ($k : 500, \sigma_r = 6$)

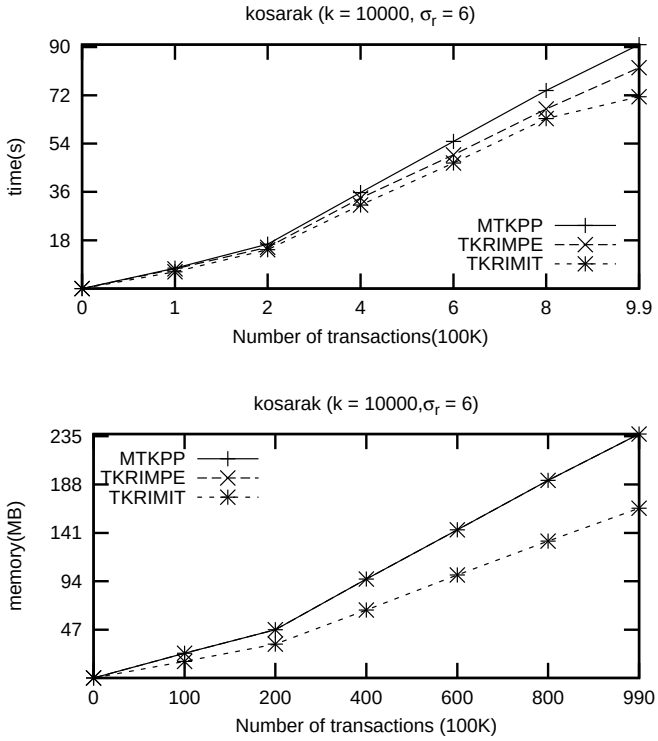


Figure 5.60: Scalability of TKRIMIT ($k : 10,000, \sigma_r = 6$)

CHAPTER VI

H-TKRIMP: HYBRID REPRESENTATION ON TOP-*K* REGULAR-FREQUENT ITEMSETS MINING BASED ON DATABASE PARTITIONING

As described in previous chapters, TKRIMPE based on the database partitioning and the support estimation technique works well on sparse datasets. Whilst, TKRIMIT based on the interval tidset representation achieves good performance on dense datasets. Therefore, the aim of this chapter is to devise a new efficient algorithm by combining the techniques from TKRIMPE and TKRIMIT. The database partitioning technique is integrated with the interval tidset representation to gain good performance on both sparse and dense datasets. Consequently, a new efficient single-pass algorithm, H-TKRIMP (Hybrid representation on Top-*K* Regular-frequent Itemsets Mining based on database Partitioning), is introduced. In this chapter, a database partitioning technique (as presented in Chapter 4) and a hybrid representation (i.e. a combination between normal tidset and interval tidset representations) are described in details. Besides, the data structure used to maintain the top-*k* regular-frequent itemsets during mining process and the complexity analysis of H-TKRIMPE are also discussed.

6.1 Preliminary of H-TKRIMP

To mine a set of top-*k* regular-frequent itemsets, H-TKRIMPE also employs a top-*k* list as the previous algorithms. The top-*k* list is used to maintain the top-*k* regular-frequent itemsets during mining. Besides, the best-first search strategy is applied to cut down the search space and quickly mine the regular itemsets with the highest supports. Further, the database partitioning technique is utilized to dismiss some unnecessary computing. Ultimately, a combination between normal tidset and interval tidset representation, Hybrid representation, is devised and included into H-TKRIMP to obtain good performance on all characteristics of datasets.

6.2 H-TKRIMP: Top-*k* list structure

As previous algorithms, H-TKRIMP is also based on the use of a top-*k* list structure which is a simple linked-list with a hash table. The top-*k* list is used to maintain a set of k (or less than k) regular itemsets with the highest supports and their occurrence information during mining process. Meanwhile, the hash table is utilized to quickly access all the information of each itemset

in the top-k list. As shown in Figure 6.1, each entry in the top-k list consists of 4 fields: item or itemset name (I), total support (s^I), regularity (r^I), and a set of tidsets ($T^I = \{T^I_1, \ldots, T^I_{pn}\}$) where pn is the number of partitions of considered database). From the figure, the item a has a support of 11, a regularity of 2, and tidsets as $\{\{1, -3\}, \{6, -2\}, \{9, -3\}\}$ which means the item a occurs in transactions $\{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$.

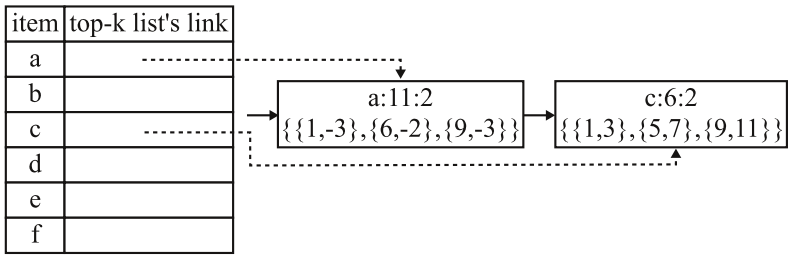


Figure 6.1: H-TKRIMP: Top-k list structure with hash table

6.3 Database Partitioning

In H-TKRIMP, the database is first divided into several disjoint partitions which have an equal number of transactions as presented in (Brin et al., 1997b). Then, the tidsets (there is one tidset for each partition) of each itemset are collected by using the proposed hybrid representation in order to calculate its support and regularity with one database scan. This partition technique allows reducing unnecessary computational costs.

Given the regularity threshold σ_r, the database is split into $pn = \lceil |TDB|/\sigma_r \rceil$ partitions. Each partition will then contains σ_r transactions. For example, consider the transactional database of Table 6.1 with 12 transactions. A regularity threshold of 4 will split the database into 3 partitions with 4 transactions each.

Table 6.1: A transactional database as a running example of H-TKRIMP

tid	items
1	*a b c d f*
2	*a b d e*
3	*a c d*
4	*a b*
5	*b c e f*
6	*a d e*
7	*a b c d e*
8	*a b d*
9	*a c d f*
10	*a b e*
11	*a b c d*
12	*a d f*

H-TKRIMP will fully exploit the partitioning of the database. Thus, each itemset has a (local) support, a (local) regularity, and a (local) tidset for each partition of database.

The tidset of an itemset X in the m^{th} partition P_m, denoted T_m^X, is the set of tids in m^{th} partition that contains the itemset X:

$$T_m^X = \{t_{q,m} | X \subseteq t_{q,m}, t_{q,m} \in P_m\}$$

By combining the partitioning technique with the hybrid representation together, H-TKRIMP can use two representations (i.e. normal tidset and interval tidset) to maintain tids of each partition. Then, $T^X = \{T_1^X, \ldots, T_{pn}^X\}$ is defined as the (global) tidset of an itemset X.

The (local) support of an itemset X in the m^{th} partition, denoted s_m^X, is the number of transactions (also denoted tids) in the m^{th} partition that contains the itemset X. Then, the (global) support s^X of the itemset X is equal to $\sum_{m=1}^{pn} s_m^X$.

For example, consider an item a occurring in tids $\{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$ (i.e. transactions $T^a = \{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$) from the transactional database of Table 6.1. Thus, based on the partitioning technique, the tidset of the first partition T_1^a contains the set of tids $\{1, 2, 3, 4\}$ where the item a occurs. Meanwhile, the set of tids $\{6, 7, 8\}$ and $\{9, 10, 11, 12\}$ are maintained in T_2^a and T_3^a, respectively. Thus, the (global) tidset of a is $T^a = \{\{1, 2, 3, 4\}, \{6, 7, 8\}, \{9, 10, 11, 12\}\}$. Besides, the support of a is $s^a = 4 + 3 + 4 = 11$.

As mentioned in Chapter 4, based on the use of database partitioning technique, H-TKRIMP can reduce some considered tids on mining process.

6.4 Hybrid representation

To allow for efficient calculating support and regularity of an itemset, a hybrid representation is applied in H-TKRIMP to collect its tidset that occurs in each partition. A hybrid representation is a combination between normal tidset (i.e. the exact value of the transaction-ids) and interval tidset (i.e. using only one positive and one negative integer to store a set of consecutive continuous transaction-ids).

Definition 6.1 (Normal tidset of an itemset X in m^{th} partition) *Let a set of tids that the itemset X occurs in TDB at the m^{th} partition be $\{t_{p,m}^X, t_{p+1,m}^X, \ldots, t_{q,m}^X\}$, where $p < q$. Thus, the tidset of the itemset X is defined as:*

$$T_m^X = \{t_{p,m}^X, \ldots, t_{q,m}^X\}$$

Definition 6.2 (Interval tidset of an itemset X in m^{th} partition) *Let a set of tids that itemsets X occurs in TDB at the m^{th} partition be $\{t_{p,m}^X, t_{p+1,m}^X, \ldots, t_{q,m}^X\}$ where $p < q$ and there are some consecutive tids $\{t_{u,m}^X, t_{u+1,m}^X, \ldots, t_{v,m}^X\}$ that are continuous between $t_{p,m}^X$ and $t_{q,m}^X$ (where $p \leq u$ and $q \geq v$). Thus, interval tidset of itemset X is defined as:*

$$T_m^X = \{t_{p,m}^X, t_{p+1,m}^X, \ldots, t_{u,m}^X, (t_{u,m}^X - t_{v,m}^X), t_{v+1,m}^X, \ldots, t_{q,m}^X\}$$

For example, consider an item a occurring in tids $\{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$ from the transactional database of Table 6.1. Thus, based on the partitioning technique and the hybrid representation, the tidset of the first partition T_1^a contains the set of tids 1,-3 where item a occurs. Meanwhile, thetidsets $\{6, -2\}$ and $\{9, -3\}$ are maintained in T_2^a and T_3^a, respectively. Therefore, the tidsets of a is $T^a = \{\{1, -3\}, \{6, -2\}, \{9, -3\}\}$.

For each tidset T_m^X of an itemset X, H-TKRIMP has to decide which representation should be used to achieve a good performance. To make a decision, the advantage and disadvantage of each representation are considered. The advantage of using an interval tidset representation is the number of reduced tids (as described in Chapter 5). Whereas, the disadvantage is the number of tids that have to be determined whether it is consecutive continuous tids.

Definition 6.3 (Number of reduced tids in the m^{th} partition) *Let T_m^X be the interval tidset of an itemset X in the m^{th} partition and let $TN_m^X = \{tn_{1,m}^X, \ldots, tn_{j,m}^X\}$, where $1 \leq j \leq |T_m^X|$, $tn_{j,m}^X \in T_m^X$, and $tn_{j,m}^X < 0$, be the set of negative tids in the interval tidset T_m^X. Then, nrt_m^X is defined as the number of reduced tids in the m^{th} partition from the interval tidset T_m^X:*

$$nrt_m^X = \sum_{i=1}^{|TN_m^X|} -(1 + tn_{i,m}^X)$$

Definition 6.4 (Number of determined tids (to check whether they are consecutive continues tids) in the m^{th} partition) *Let T_m^X be the interval tidset of an itemset X in the m^{th} partition and let $TN_m^X = \{tn_{1,m}^X, \ldots, tn_{j,m}^X\}$, where $1 \leq j \leq |T_m^X|$, $tn_{j,m}^X \in T_m^X$ and $tn_{j,m}^X < 0$, be the set of negative tids in the interval tidset T_m^X. Then, the number of tids that are determined as the consecutive continuous tids ndt_m^X can be defined as:*

$$ndt_m^X = \begin{cases} |T_m^X| - |TN_m^X| - 1 & \textit{if } t^X_{|T_m^X|,m} > 0 \\ |T_m^X| - |TN_m^X| & \textit{if } t^X_{|T_m^X|,m} < 0 \end{cases}$$

Therefore, the trade-off between nrt_m^X and ndt_m^X values is taken into account. If $nrt_m^X \geq ndt_m^X$, H-TKRIMP can take advantage from the interval tidset representation. Then, H-TKRIMP uses an interval tidset to maintain a tidset. Otherwise, a normal tidset is applied. By using a hybrid representation, H-TKRIMP can save time from the use of the combination between a normal tidset and an interval tidset in the mining process (in Section 6.6).

6.5 Calculation of Regularity and Support

By using the partition technique and the hybrid representation, the tidset of each itemset is splited into several tidsets, and these tidsets may contain some negative tids when the itemset occurs in consecutive continuous tids (as described in Definition 6.2). As a consequence, the original definition of the regularity of an itemset of (Tanbeer et al., 2009)) and that of (Amphawan et al., 2009) cannot find the regularity between two tidsets and between positive and negative tids. It is suitable for only one tidset in each itemset and only for positive tids. Accordingly, five new definitions is proposed to calculate the regularity of each itemset.

Definition 6.5 (Regularity of an itemset X between two consecutive tids in a normal tidset) *Consider the normal tidset T_m^X of an itemset X for the m^{th} partition. Let $t_{p,m}^X$ and $t_{q,m}^X$ be two consecutive tids in T_m^X, i.e. where $p < q$, and there is no tid $t_{o,m}^X$ in T_m^X, $p < o < q$, such that a transaction of $t_{o,m}^X$ contains X (note that p, q and o are indices). Thus, $rtt_{q,m}^X$ is defined as the regularity value between the two consecutive tids $t_{p,m}^X$ and $t_{q,m}^X$ by following cases:*

$$rtt_{q,m}^X = \begin{cases} t_{q,m}^X & \textit{if } q = 1 \\ t_{q,m}^X - t_{p,m}^X & \textit{if } 2 \leq q \leq |T_m^X| \end{cases}$$

Definition 6.6 (Regularity of an itemset X between two consecutive tids in an interval tidset) *Consider the interval tidset T_m^X of an itemset X for the m^{th} partition. Let $t_{p,m}^X$ and $t_{q,m}^X$ be two consecutive tids in T_m^X, i.e. where $p < q$ and there is no transaction t_o, $p < o < q$, such that t_o contains X (note that p, q and o are indices). Then, $rtt_{q,m}^X$ is denoted as the number of tids (transactions) between $t_{p,m}^X$ and $t_{q,m}^X$ that do not contain X. This leads to the following cases:*

$$rtt^X_{q,m} = \begin{cases} t^X_{q,m} & \text{if } q = 1 \\ t^X_{q,m} - t^X_{p,m} & \text{if } t^X_{p,m} \text{ and } t^X_{q,m} > 0, 2 \leq q \leq |T^X_m| \\ 1 & \text{if } t^X_{p,m} > 0 \text{ and } t^X_{q,m} < 0, 2 \leq q \leq |T^X_m| \\ t^X_{q,m} + (t^X_{p,m} - t^X_{p-1,m}) & \text{if } t^X_{p,m} < 0 \text{ and } t^X_{q,m} > 0, 2 \leq q \leq |T^X_m| \end{cases}$$

Definition 6.7 (Regularity of an itemset X in the m^{th} partition) *Let for a T^X_m, $RTT^X_m = \{rtt^X_{1,m}, \ldots, rtt^X_{|T^X_m|,m}\}$ be the set of regularity between each pair of consecutive tids in the m^{th} partition. Then, the regularity of X in the m^{th} partition can be denoted as:*

$$rp^X_m = max(rtt_{1,m}, rtt_{2,m}, \ldots, rtt_{|T^X_m|,m})$$

Definition 6.8 (Regularity of an itemset X between two consecutive tidsets) *Let $t^X_{|T^X_m|,m-1}$ be the last tid where X occurs in the $(m-1)^{th}$ partition and $t^X_{1,m}$ be the first tid where X occurs in the m^{th} partition. Then, rtp^X_m is denoted as the regularity of X (i.e. the number of tids (transactions) that do not contain X) between the two consecutive partitions, $(m-1)^{th}$ and m^{th}. Obviously, rtp^X_1 is $t^X_{1,m}$. Lastly, to find the exact regularity between two consecutive partitions of X on all the database, the number of transactions that do not contain X between the last tid where X occurs and the last transaction of database: rtp^X_{pn+1} is also considered. Thus, the regularity between any two consecutive tidsets T^X_{m-1} and T^X_m can be defined as:*

$$rtp^X_m = \begin{cases} t^X_{1,m} & \text{if } m = 1 \\ t^X_{1,m} - t^X_{|T^X_{m-1}|,m-1} & \text{if } 2 \leq m \leq pn, t^X_{|T^X_{m-1}|,m-1} > 0 \\ t^X_{1,m} - (t^X_{|T^X_{m-1}|-1,m-1} - t^X_{|T^X_{m-1}|,m-1}) & \text{if } 2 \leq m \leq pn, t^X_{|T^X_{m-1}|,m-1} < 0 \\ |TDB| - t^X_{|T^X_{m-1}|,m-1} & \text{if } m = pn+1, t^X_{|T^X_{m-1}|,m-1} > 0 \\ |TDB| - (t^X_{|T^X_{m-1}|-1,m-1} - t^X_{|T^X_{m-1}|,m-1}) & \text{if } m = pn+1, t^X_{|T^X_{m-1}|,m-1} < 0 \end{cases}$$

Then, the regularity of an itemset is defined with the help of definitions 6.7 and 6.8.

Definition 6.9 (Regularity of an itemset X) *The regularity of an itemset X is defined as:*

$$r^X = max(max(RP^X), max(RTP^X))$$

where $RP^X = \{rp_1^X, rp_2^X, \ldots, rp_{pn}^X\}$ *is the set of regularities of* X *in each partition (Definition 6.7) and* $RTP^X = \{rtp_1^X, rtp_2^X, \ldots, rtp_{pn+1}^X\}$ *is the set of regularities of* X *between two consecutive partitions (Definition 6.8).*

To calculate the support of each itemset from its tidsets, two definitions are used to compute the support in each partition and the total support of the itemset is also presented.

Definition 6.10 (Support of an itemset X in a partition) *Let* $t_{i-1,m}^X$ *and* $t_{i,m}^X$ *be the two consecutive tids in* T_m^X *Thus,* $stt_{i,m}^X$ *is defined as the support value between two consecutive tids* $t_{i-1,m}^X$ *and* $t_{i,m}^X$ *by following cases:*

$$stt_{i,m}^X = \begin{cases} 1 & \text{if } t_{i,m}^X > 0 \\ -t_{i,m}^X & \text{if } t_{i,m}^X < 0 \end{cases}$$

Therefore, the regularity of the itemset X *in the* m^{th} *partition is defined as follows.*

$$s_m^X = \sum_{i=1}^{|T_m^X|} stt_{i,m}^X$$

Definition 6.11 (Support of an itemset X) *The support of an itemset* X*, denoted* s^X*, is the summation of support in every partition, i.e.,*

$$s^X = \sum_{m=1}^{pn} s_m^X$$

For example, consider the transactional database of Table 6.1 and the case of an item a: $T^a = \{\{1, -3\}, \{6, -2\}, \{9, -3\}\}$. The set of regularities in each partition of the item a is $RP^a = \{1, 1, 1\}$. The set of regularities between two consecutive partitions of a is $RTP^a = \{1, 6-(1-(-3)), 9-(6-(-2)), 12-(9-(-3))\} = \{1, 2, 1, 0\}$. Thus, the regularity of item a is $r^a = max(max(1, 1, 1), max(1, 2, 1, 0)) = 2$. In addition, the set of supports of the item a in each partition is = $\{(1+(-(-3)), (1+(-(-2))), (1+(-(-3)))\} = \{4, 3, 4\}$. Consequently, the support s^a of the item a is equal to $4 + 3 + 4 = 11$.

6.6 H-TKRIMP algorithm

Based on the database partitioning and the hybrid representation mentioned above, the H-TKRIMP algorithm is also described. H-TKRIMP consists of two steps : (*i*) Top-*k* list initialization: partition the database, scan each partition to obtain top-*k* regular items and then transform

each tidset into the suitable tidset (a normal tidset or an interval tidset); (*ii*) Top-*k* mining: merge (with two constraints) each pair of elements in the top-*k* list to produce new larger itemsets by using the best-first search strategy, sequentially intersect their tidsets (one by one partition) to find the k regular itemsets with the highest supports, and then transform each tidset of the new generated itemset into the proper representation.

6.6.1 H-TKRIMP: Top-*k* initialization

To create the top-*k* list, each partition of the database is scanned (one by one transaction) to obtain all k (or less than k) regular items. A new entry in the top-*k* list is created for any item that occurs in the first σ_r transactions (i.e. occurs in the first partition). Each item of the current transaction is then considered. With the help of the hash table, H-TKRIMP quickly realizes whether the current item is already existed in the top-*k* list or not. For the first occurrence of an item in the partition, a new tidset for the partition is built and its support, regularity, and a tidset are initialized. Otherwise, H-TKRIMP updates its support, regularity and a tidset.

To update the tidset T_m^X of an item X in the m^{th} partition, H-TKRIMP has to compare the last tid $(t_{i,m}^X)$ of T_m^X with the new coming tid (t_j). It simply consists of the following cases:

- if $t_{i,m} < 0$, i.e. there are some former tids which are consecutive and continuous with the exact tid of $t_{i,m}$. H-TKRIMP calculates the exact tid of $t_{i,m} < 0$ (i.e $t_{i-1,m} - t_{i,m}$), and compares it with t_j to check if they are continuous or not. If they are consecutive continuous tids (i.e $t_j - t_{i-1,m} + t_{i,m} = 1$), H-TKRIMP has to extend the tidset T_m^X (it consists only of adding -1 to $t_{i,m}$), otherwise H-TKRIMP creates a new element to take into account t_j (it simply consists of adding t_j after $t_{i,m}$ in T^X).
- if $t_{i,m} > 0$, i.e. there is no former tid, consecutive and continuous with $t_{i,m}$. H-TKRIMP compares $t_{i,m}$ with t_j to check if they are continuous or not. If they are consecutive continuous tids (i.e. $t_j - t_{i,m} = 1$) H-TKRIMIP creates a new interval in T^X (it consists of adding -1 after $t_{i,m}$ in T_m^X); otherwise, H-TKRIMP creates a new element to take into account t_j (it simply consists of adding t_j after $t_{i,m}$ in T_m^X).

At the end of the m^{th} partition, if $nrt_m^X < ndt_m^X$, the interval tidsets T_m^X will be transformed to a normal tidset. When the entire database is read, the top-*k* list is trimmed by removing all the entries (items) with regularity greater than the regularity threshold σ_r, and the remaining entries are sorted in descending order of support. Lastly, H-TKRIMP removes the entries after the k^{th} entry in the top-*k* list. The detail of the top-*k* list's construction is presented in Algorithm 7.

Algorithm 7 (H-TKRIMP: Top-k list initialization)

(*1*) A transaction database: TDB
(*2*) A number of itemsets to be mined: k
(*3*) A regularity threshold: σ_r
Output:
(*1*) A top-*k* list

create a hash table for all 1-items
for each partition $m = 1$ to pn **do**
  for each transaction j in the the m^{th} partition **do**
    for each item i in the transaction j **do**
      if the item i does not have an entry in the top-*k* list **then**
        create a new entry for the item i with $s^i_m = 1, r^i = t_j$ and create a tidset T^i_m that contain t_j
        create a link between the hash table and the new entry
      else
        add the support s^i_m by 1
        if t_j and the last tid in T^i_m are two consecutive continuous tids **then**
          if the last tid in $T^i_m < 0$ **then**
            add the last tid in T^i_m by -1
          else
            collect -1 as the last tid in T^i_m
        else
          collect t_j as the last tid in T^i_m
          calculate the regularity r^i by t_j

  for each entry (item) i in the top-*k* list **do**
    add the support s^i by s^i_m
    if $nrt^X_m < ndt^X_m$ **then**
      transform T^i_m to be a normal tidset // not contain $tid < 0$

for each item i in the top-*k* list **do**
  calculate the regularity r^i by $|TDB|-$ the last tid of T^i_{pn}
  if $r^i > \sigma_r$ **then**
    remove the entry i out of the top-*k* list
sort the top-*k* list by support descending order
remove all of entries after the k^{th} entry in the top-*k* list

6.6.2 H-TKRIMP: Top-*k* mining

The top-k mining algorithm, shown in Algorithm 8, also adopts the best-best first search strategy (*i.e.* first consider from the most frequent itemsets to the least frequent itemsets in the top-*k* list) to quickly generate the regular itemsets with the highest supports and to raise up the support of the k^{th} itemset (s_k). This strategy can help the H-TKRIMP algorithm to prune the search space by using the support s_k.

To generate a new top-*k* regular-frequent itemsets, two candidate itemsets X and Y in the top-*k* list are merged to be an itemset XY with the following two constraints: (*i*) the size of the itemsets must be equal; (*ii*) both itemsets must have the same prefix (i.e. each item from both itemsets is the same, excepts the last item). These constraints can help H-TKRIMP avoid the repetition of generating top-*k* regular itemsets and help H-TKRIMP prune the search space. Consequently, the tidsets of itemsets X and Y are sequentially intersected in order to calculate

the support, the regularity and the tidsets of XY. To sequentially intersect interval tidsets T_m^X and T_m^Y, H-TKRIMP categorizes this process into three cases as follow:

- **Two of them are normal tidsets.** The two tidsets can be easily intersected by comparing each pair of tids. If they are equal, H-TKRIMP collects one of them into the tidset T_m^{XY} of the m^{th} partition.
- **Two of them are interval tidsets.** H-TKRIMP has to consider four cases when comparing each pair of tids t_i^X and t_j^Y in order to construct T^{XY} (see Definition 6.2):

  (1) if $t_{i,m}^X = t_{j,m}^Y > 0$ add $t_{i,m}^X$ at the end of T_m^{XY}

  (2) if $t_{i,m}^X > 0, t_{j,m}^Y < 0, t_{i,m}^X \leq t_{j-1,m}^Y - t_{j,m}^Y$, add $t_{i,m}^X$ at the end of T_m^{XY}

  (3) if $t_{i,m}^X < 0, t_{j,m}^Y > 0, t_{j,m}^Y \leq t_{i-1,m}^X - t_{i,m}^X$, add $t_{j,m}^Y$ at the end of T_m^{XY}

  (4) if $t_{i,m}^X, t_{j,m}^X < 0$, add $t_{|T^{XY}|,m}^{XY} - (t_{i-1,m}^X - t_{i,m}^X)$ at the end of T_m^{XY} if $t_{i-1,m}^X - t_{i,m}^X < t_{j-1,m}^Y - t_{j,m}^Y$ otherwise add $t_{|T^{XY}|,m}^{XY} - (t_{j-1,m}^Y - t_{j,m}^Y)$ at the end of T^{XY}
- **One of them is a normal tidset and the another one is an interval tidset.** The conditions from the second case are applied with some different details; for example, T_m^X is a normal tidset and T_m^Y is an interval tidset.

  (1) if $t_{i,m}^X = t_{j,m}^Y > 0$, add $t_{i,m}^X$ at the end of T_m^{XY}

  (2) if $t_{i,m}^X > 0, t_{j,m}^Y < 0, t_{i,m}^X \leq t_{j-1,m}^Y - t_{j,m}^Y$, add $t_{i,m}^X$ at the end of T_m^{XY}

From T_m^{XY}, the support s^{XY} and regularity r^{XY} of XY can be easily computed. If the regularity of the new generated itemset XY is no greater than σ_r and its support is greater than s_k, then XY is inserted in the top-*k* list and the k^{th} itemset is removed from the top-*k* list. Lastly, because of the partitioning technique, TKRIMPE can reduce the time to intersect some tids of each partition when at least one of the tidsets does not contain regular sequence of transactions. This will frequently happen particularly in sparse datasets.

By separating the intersection process into 3 cases, H-TKRIMP can reduce computational time in some cases. For the first case, the two tidsets are normal tidsets. This means that the two considered candidate itemsets occur sparsely in the partition. Thus, the computational time used to intersect these tidsets is equal to TKRIMPE which is the fastest algorithm for sparse datasets. For the second case, the two tidsets are interval tidsets. H-TKRIMP has similar perfomance as TKRIMIT which is the best algorithm for dense datasets. Finally, for the third case, one is a normal tidset and another one is an interval tidset. It is the case of the intersection between tidsets

that are sparse and dense, respectively. H-TKRIMP has similar performance since TKRIMPE as it consider only small size of tidsets.

Based on the hybrid representation, H-TKRIMP can reduce time to intersect tidsets from TKRIMPE by reducing a number of tids in the dense tidsets. Moreover, H-TKRIMP can reduce time in the intersect process from TKRIMIT on the sparse tidsets by reducing the time to investigate each tid in the interval tidset whether it is consecutive continuous or not.

Algorithm 8 (H-TKRIMP: top-k mining)

Input: top-k list, σ_r, k
Output: top-k regular-frequent itemsets

```
for each entry x in the top-k list do
  for each entry y in the top-k list (x > y) do
    if the entries x and y have the same size of itemsets and the same prefix then
      merge the itemsets of x and y to be the itemset Z = I^x ∪ I^y
      for each partition m = 1 to pn do
        for each t_p in T_m^X (p = 1 to |T_m^X|) and t_q in T_m^Y (q = 1 to |T_m^Y|) do
          if t_p > 0 and t_q > 0 then
            if t_p = t_q then
              calculate the regularity r^Z by t_p and check r^Z with σ_r
              add the support s_m^Z by 1
              collect t_p as the last tid in T_m^Z
          else if t_p > 0 and t_q < 0 then
            if t_p ≤ t_{q-1} − t_q then
              calculate the regularity r^Z by t_p and check r^Z with σ_r
              add the support s_m^Z by 1
              collect t_p as the last tid in T_m^Z
          else if t_p < 0 and t_q > 0 then
            if t_{p-1} − t_p ≥ t_q then
              calculate the regularity r^Z by t_q and check r^Z with σ_r
              add the support s_m^Z by 1
              collect t_q as the last tid in T_m^Z
          else
            if t_{p-1} − t_p > t_{q-1} − t_q then
              add the support s_m^Z by (t_{q-1} − t_q) − t^Z_{|T_m^Z|}
              collect t^Z_{|T_m^Z|} − (t_{q-1} − t_q) as the last tid in T_m^Z
            else
              add the support s_m^Z by (t_{p-1} − t_p) − t^Z_{|T_m^Z|}
              collect t^Z_{|T_m^Z|} − (t_{p-1} − t_p) as the last tid in T_m^Z
        add the support s^Z by s_m^Z
        if nrt_m^X < ndt_m^X then
          transform T_m^Z to be a normal tidset // not contain tid < 0

      calculate the regularity r^Z by |TDB|− the last tid of T_pn^Z
      if r^Z ≤ σ_r and s^Z ≥ s_k then
        insert the itemset Z (I^x ∪ I^y) into the top-k list with r^Z, s^Z and T^Z
        remove the k^th entry from the top-k list
```

6.7 Example of H-TKRIMP

Consider the TDB of Table 6.1, the regularity threshold σ_r of 4 and the number of desired results k of 5. The database is separated into three partitions. Then, the process of initializing top-*k* list from the TDB of Table 6.1 is illustrated in Figure 6.2.

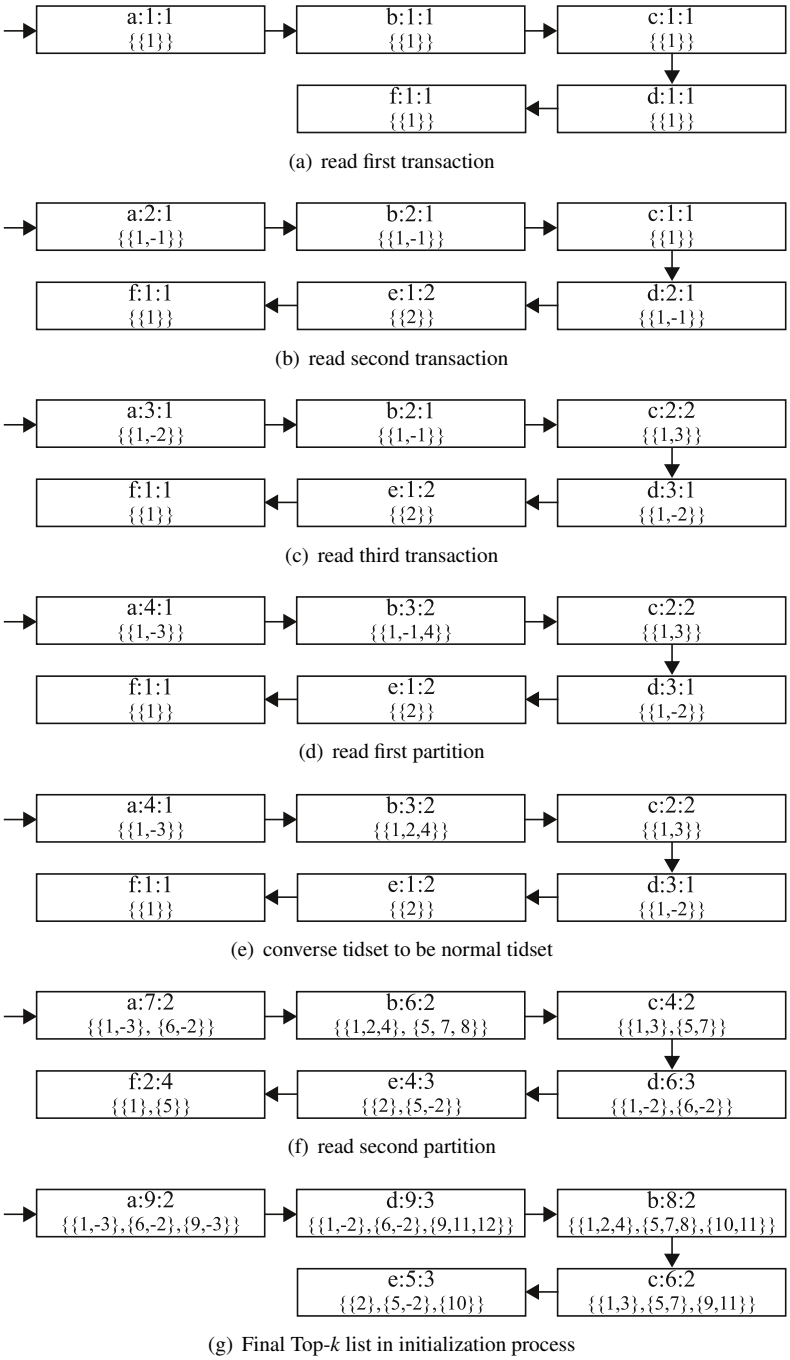
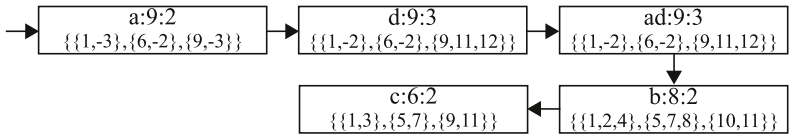
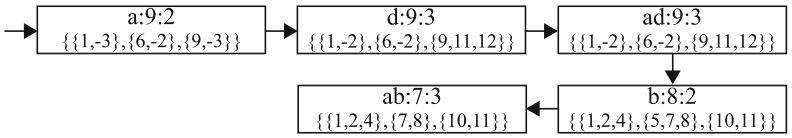


Figure 6.2: Top-*k* list initialization

By scanning the first transaction $t_1 = \{a, b, c, d, f\}$, the entries for items a, b, c, d, and f are created, and their supports, regularities and interval tidsets are initialized as $(1 : 1 : \{1\})$ (see



(a) top-*k* list when merging item a with item d



(b) final top-*k* list

Figure 6.3: Top-*k* during mining process

Figure 6.2(a)). Next, the second $t_2 = \{a, b, d, e\}$ is read, and H-TKRIMP adds -1 at the end of the interval tidsets of a, b and d, since these items occur in two consecutive continuous transactions. Then, the entry for the item e is created and initialized (Figure 6.2(b)). For the third transaction ($t_3 = \{a, c, d\}$), as shown in Figure 6.2(c), the last tids of item a and d are changed to -2 (they occur in three consecutive continuous transactions t_1, t_2 and t_3) and the interval tidset of the item c is updated by adding t_3 as the last tid in T_1^c. Now, the forth transaction is considered to update the tidset T_1 of items a and b as illustrated in Figure 6.2(d). However, if $nrt_1^b(=0) < ndt_1^b(=1)$, then H-TKRIMP transforms T_1^b to be a normal tidset (see Figure 6.2(e)). After the first partition is read, the next partition (transactions 5 to 8) initializes or updates the tidset T_2 for each item occurring in this partition as illustrated in Figure 6.2(f). Finally, the third partition is considered and then H-TKRIMP transforms the tidsets into suitable representation. After scanning all the transactions, the top-*k* list is sorted by support descending order and the item f is removed (see Figure 6.2(g)). It will be the starting point for the mining process.

In the mining process, the item d is first merged with the former item a. The tidsets T^a and T^d are sequentially intersected from the first to the last partition in order to calculate the support $s^{ad} = 9$, the regularity $r^{ad} = 3$. The tidsets of the first and the second are $T_1^{ad} = \{1, -2\}$ and $T_2^{ad} = \{6, -2\}$, respectively. Meanwhile, the tidset is $T_3^{ad} = \{9, 11, -1\}$ and $nrt_3^{ad}(=0) < ndt_3^{ad}(=2)$. Then, H-TKRIMP transforms T_3^{ad} into a normal tidset format ($T_3^{ad} = \{9, 11, 12\}$). The tidsets T^{ad} of itemset ad is $T^{ad} = \{\{1, -2\}, \{6, -2\}, \{9, 11, 12\}\}$. Since the support s^{ad} is greater than $s^e = 5$ and the regularity r^{ad} is less than $\sigma_r = 4$, the item e is removed and ad is inserted into the top-*k* list as shown in Figure 6.3(a). Next, the third itemset *i.e.* itemset ad is considered and compared to the former itemsets a and b. Since these itemsets do not have different size (and do not share the same prefix), they are not merged. Next, H-TKRIMP then considers the item b which is merged with a and d ($s^{ab} = 7$, $r^{ab} = 3$, $T^{ab} = \{\{1, 2, 4\}, \{7, 8\}, \{10, 11\}\}$; $s^{bd} = 5$, $r^{bd} = 5$, $IT^{bd} = \{\{1, 2\}, \{7, 8\}, \{11\}\}$). The itemset ab is thus added to the list and the

item c is removed. The itemset bd is eliminated. Lastly, the itemsets ab and ad are considered, and finally the top-*k* regular-frequent itemsets are obtained as shown in Figure 6.3(b).

6.8 Complexity analysis

In this section, we discuss the computational complexity for H-TKRIMP in terms of time and space. Extensive experimental studies will complement this analysis in Section 6.9.

Proposition 6.12 *The time complexity for creating the top-k list is $O(nm)$ where m is the number of transactions in the database and n is the number of items occurring in the database.*

Proof: Since the proposed algorithm scans each transaction in the database once, the entry of each items that occurs in the transaction is also looked up once in order to collect the tid into tidset ($O(nm)$). The cost for sorting all (in the very worst case) the entries is $O(n \log n)$. Then, the time complexity to create the top-*k* list is formally $O(nm + n \log n)$. In fact, the number of items (n) is, for the considered applications, always less than the number of transactions(m). Hence, the time complexity to create the top-*k* list is $O(nm)$. ■

Proposition 6.13 *The time complexity for mining top-k regular-itemset is $O(mk^2)$ where m is the number of transactions in the database and k is the number of results to be mined.*

Proof: The mining process merges each itemset in the top-*k* list with only the former itemset in the top-*k* list. Then, the tidsets of the two merged itemsets are intersected. Therefore, the combination of all itemsets in the top-*k* list is $k * (k+1)/2$ and the time to intersect tidset at each step is $O(m)$. Hence, the overall time complexity of mining process is $O(mk^2)$. ■

Proposition 6.14 *The memory space required by TKRIMIT is $O((\lceil \frac{3}{4}|TDB| \rceil)k)$ where σ_r is the number of transactions in each partition and k is the number of itemsets to be mined.*

Proof: Based on the interval tidset representation, the maximum number of maintained tids of an itemset X in TDB is $\lceil \frac{2}{3}|TDB| \rceil$. With the partitioning technique, the database is divided into several partitions. Thus, the maximum number of maintained tids of an itemset X in any m^{th} partition is $\lceil \frac{2}{3}\sigma_r \rceil$ where σ_r is the number of transactions in each partition. This case happens when the itemset X occurs in every two transactions and miss one transaction in the m^{th} partition.

Since, the interval tidset contains one positive and one negative tids alternately over the tidset, the maximum value of the number of determined tids ndt_m^X is equal to the number of negative tids in the interval tidset which is $ndt_m^X = \frac{\lceil \frac{2}{3}\sigma_r \rceil}{2} = \lceil \frac{1}{3}\sigma_r \rceil$.

To decide which representation should be used for the m^{th} partition, the value of nrt_m^X must be greater than ndt_m^X. As mentioned in Chpater 5, the use of interval tidset representation cannot reduce the number of tids to be maintained in the case that the number of tids is less than or equal to $\lceil \frac{2}{3}\sigma_r \rceil$. If the number of maintained tids is equal to $\lceil \frac{2}{3}\sigma_r \rceil + 1$, there are at least one group that have three or more consecutive tids in the tidset. Thus, for each increasing number of tids that more than $\lceil \frac{2}{3}\sigma_r \rceil$, the number of reduced tids is increased 3 and the number of determined tids is reduced to 1. Thus, when the number of tids that X occurs is grater than $\lceil \frac{2}{3}\sigma_r \rceil$ equal to $\frac{1}{4} * \lceil \frac{1}{3}\sigma_r \rceil = \lceil \frac{1}{12}\sigma_r \rceil$, the value of nrt_m^X and ndt_m^X are equivalent. This is the worst case of maintaining tids of the hybrid representation. Then the maximum number of maintained tids in the m^{th} partition is equal to $\lceil \frac{2}{3}\sigma_r \rceil + \lceil \frac{1}{12}\sigma_r \rceil = \lceil \frac{3}{4}\sigma_r \rceil$.

In addition, the maximum number of maintained tids of the itemset X in every partitions is equal to $\lceil \frac{3}{4}\sigma_r \rceil * pn = \lceil \frac{3}{4}|TDB| \rceil$ where pn is the number of partitions in database. Consequently, all of desired memory to maintain interval tidsets for k itemsets is $O((\lceil \frac{3}{4}|TDB| \rceil)k)$. ■

6.9 Performance evaluation

In this section, the performance of the H-TKRIMP algorithm is empirically studied and compared with the previous top-*k* regular-frequent itemsets mining algorithms: MTKPP, TKRIMPE and TKRIMIT to demonstrate the difference on performance of the algorithms to mine top-*k* regular-frequent itemsets. To measure the performance of H-TKRIMP, the processing time (including top-*k* list construction and mining processes), space usage (*i.e.* memory consumption) and scalability (with varied number of transactions in database) are considered.

6.9.1 Experimental setup

The experiments of H-TKRIMP are done on three synthetic datasets (T10I4D100K, T20I6D100K and T20I6D100K) and nine real datasets (accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb* and retail) which were described their details and characteristics in Chapter 2. Program for H-TKRIMP is written in C in the same manner as the previous algorithms: MTKPP, TKRIMPE and TKRIMIT using a top-*k* list. All experiments are performed

on a Linux platform with a Intel®Xeon 2.33 GHz and with 4 GB main memory.

To evaluate the performance of H-TKRIMP, the computational time (total execution time, including CPU and I/O costs) of the four algorithms with the small and the large values of k and various values of σ_r are considered. The value of k is divided into two ranges which are $50 - 500$ (for the small values) and $1,000 - 10,000$ (for the large values). Meanwhile, the value of σ_r is set depending on the characteristic of each dataset for illustrative purpose. Therefore, the value of σ_r is not the same in each dataset. In fact, the number of regular itemsets of each database increases with the regularity threshold. For sparse datasets, each itemset does not frequently occur, then the value of σ_r should be specified to be large when the value of k is large in order to gain a large number of results. For dense datasets, each itemset appears very often, then a small value of σ_r should be used. Due to the use of the top-*k* list and the proposed hybrid representation, the study of memory consumption for H-TKRIMP compared with the previous proposed algorithms is also discussed. Lastly, the scalability of H-TKRIMP on the number of transactions in the database is illustrated.

6.9.2 Execution time

Let first consider the six real dense datasets (*i.e.* accidents, chess, connect, mushroom, pumsb, and pumsb*). Figure 6.4 to Figure 6.21 demonstrate the the runtime on real dense datasets with varied regularity threshold. In most cases, H-TKRIMP has similar performance to TKRIMIT but outperforms MTKPP and TKRIMPE. When the value of k increases, the performance difference becomes larger. With the large values of k, H-TKRIMP and TKRIMIT can fully take advantage of the interval tidset representation.

Recall that H-TKRIMP and TKRIMIT employ the interval tidset representation to maintain tids that each itemset appears, then both algorithms can group consecutive continuous tids together and reduce the number of maintained tids of each itemset. Hence, H-TKRIMP and TKRIMIT can save time to intersect tids, calculate regularity and support, and collect tidsets of each new generated itemset.

The runtime on sparse datasets (*i.e.* BMS-POS, retail, T10I4D100K, T20I6D100K, and T40I10D100K) is illustrated in Figures 6.22 - 6.36. From these figures, it can be seen that H-TKRIMP outperforms MTKPP and TKRIMIT for the small value of k because H-TKRIMP employs the hybrid representation that maintains tidsets of each itemsets follows by the occurrence behavior of each itemset. On the other hand, TKRIMPE is faster than H-TKRIMP since

H-TKRIMP does not apply the support estimation technique, it cannot take advantage from early terminated intersection process. With the large values of k, H-TKRIMP consumes execution time as much as TKRIMPE (*i.e.* the fastest algorithm among the previous algorithms) because both of them employ the database partitioning technique which helps ignoring some tids in the intersection process. In some cases, especially on BMS-POS and T40I10D100K datasets, H-TKRIMP outperforms TKRIMPE, by using the hybrid representation which can also group the consecutive continuous tids in sparse datasets,

As mentioned above, based on the database partitioning technique and the hybrid representation, H-TKRIMP can reduce intersection process time on sparse datasets and reduce space used to maintain tids on dense datasets. On dense datasets, supports of most itemsets in the set of results are quite high, thus the interval tidset representation is applied to such itemsets. As a result, the processing time of H-TKRIMP is similar to TKRIMIT which performs best on most dense datasets with long items. On sparse datasets, with the small value of k, the processing time of H-TKRIMP is shorter than MTKPP and TKRIMIT but it is longer than TKRIMPE in some cases. Since, the H-TKRIMP and TKRIMIT contain some negative tids in the tidsets, it is very difficult to apply the estimation technique in the interval tidset representation. Accordingly, H-TKRIMP cannot take benefit of pruning search space from estimation technique as TKRIMPE in some datasets. With the large values of k, H-TKRIMP has the same performance as TKRIMPE which is still better than MTKPP and TKRIMIT due to the advantage of the database partitioning technique. By deeper analysis, in some datasets, *e.g.* BMS-POS and Mushroom, H-TKRIMP is the fastest algorithm on both small and large values of k. For each itemset, H-TKRIMP uses normal tidset to collect tids for sparse partitions and also apply interval tidset to maintain tids in the dense partitions. Therefore, H-TKRIMP can take benefit from the hybrid representation on fluctuated occurred datasets.

6.9.3 Memory consumption

Another issue related to the efficiency of H-TKRIMP is memory usage. To evaluate the space usage, the regularity threshold σ_r is set to be the highest value (used in previous subsection) for each dataset.

Figure 6.37 to Figure 6.42 show the memory usage of H-TKRIMP compared to other proposed algorithms on dense datasets. It can be seen from the figures that the memory usage of H-TKRIMP increases as the value of k increases and H-TKRIMP consumes the same size of memory

as TKRIMIT in the most cases. From the figures, H-TKRIMP can down size memory usage from MTKPP and TKRIMPE with the large value of k because the advantage of using interval tidset representation. However, in some cases, H-TKRIMP uses more memory than that of TKRIMIT because it has to convert some tidsets that have too few tids to be normal tidset (*i.e.* based on the use of hybrid representation). By this way of doing, H-TKRIMP can save computational time from the use of single representation (only normal tidset or interval tidset representation) but it takes little more memory than TKRIMIT using only interval tidset representation. Meanwhile, the memory usage on sparse datasets are illustrated in Figure 6.43 to Figure 6.47. From these figures, the memory usage of the four proposed algorithms are similar due to the fact that each itemset does not occur in consecutive continuous tids. Therefore, H-TKRIMP and TKRIMIT cannot take the advantage from the interval tidset representation. However, from the results, it can be seen that based on the used of the top-*k* list structure and maintaining tidset, the memory usage of H-TKRIMP is efficient for the top-*k* regular-frequent itemsets mining using the recently available gigabyte range memory.

6.9.4 Scalability test

In this experiment, two primary factors, the scalability of execution time and memory usage, are examined. The kosarak dataset which is a huge dataset with a large number of distinct items $(41,270)$ and transactions $(990,002)$ is used. To test the scalability with the varied number of transactions, the database is first divided into six portions. Each portion contains: $100K$, $200K$, $400K$, $600K$, $800K$ and $990K$ transactions, respectively.The value of k (i.e. the number of itemsets to be mined) is specified to 500 and $10,000$ (*i.e.* each is the instance of the small and the large values of k), and the regularity threshold is set to 6% of the number of transactions in each portion.

As shown in the plots in Figures 6.48 and 6.49, H-TKRIMP outperforms the three competitors in all the tests conducted. All the execution time linearly grows as the dataset size increases from $100K$ to $990K$. For the large values of k, H-TKRIMP is much more scalable than the others due to the fact that it benefits from the proposed hybrid representation. H-TKRIMP can reduce the number of maintained tids during mining based on interval tidset representation. Furthermore, the number of considered tids (in each iteration of intersection process) is also decreased by using database partitioning technique. Meanwhile, H-TKRIMP is also the most scalable on the small value of k. In most cases, the scalability of H-TKRIMP and TKRIMIT are similar since they are both based on the interval tidset representation.

The memory scalability is also considered. From the Figures 6.48 and 6.49, the slope of H-TKRIMP smoothly increases as the number of transactions increases. In some cases, H-TKRIMP consumes memory little more than that of TKRIMIT but it still better than MTKPP and TKRIMPE because H-TKRIMP employs the hybrid representation of normal and interval tidsets. However, H-TKRIMP has linearly scalability in term of memory usage for mining top-*k* regular-frequent itemsets.

6.10 Summary

In this chapter, we have proposed an efficient algorithm to mine a set of top-*k* regular-frequent itemsets, H-TKRIMP, which is based on: (*i*) a best-first search strategy that allows to mine the most frequent itemsets as soon as possible and to raise quickly the k^{th} support (i.e. the support of the k^{th} itemset in the sorted top-*k* list) dynamically which is then used to prune the search space; (*ii*) a partitioning of the database in order to reduce the number of comparison of certain tids at the end of each partition during the intersection process and (*iii*) a hybrid representation used to maintain tidset during mining process which is a combination between normal and interval tidset representations.

The performance studies on both real and synthetic datasets show that the proposed algorithm is efficient. The performance of H-TKRIMP is compared with MTKPP, TKRIMPE and TKRIMIT, which are at the moment the only three efficient algorithms for mining top-*k* regular-frequent patterns. From the performance studies, it can be concluded that with the small and the large value of k, H-TKRIMP has good overall performance for both dense and sparse datasets. In most time, H-TKRIMP can reduce the number of maintained tids in the top-*k* list on dense datasets. This is caused to save its processing time to mine results. On the other hand, H-TKRIMP is able to reduce the number of considered tids in each iteration of intersection process on sparse datasets, thus improves its running time instantly. By combining the partitioning and the hybrid representation together, H-TKRIMP is efficient in terms of time and space to mine top-*k* regular-frequent itemsets.

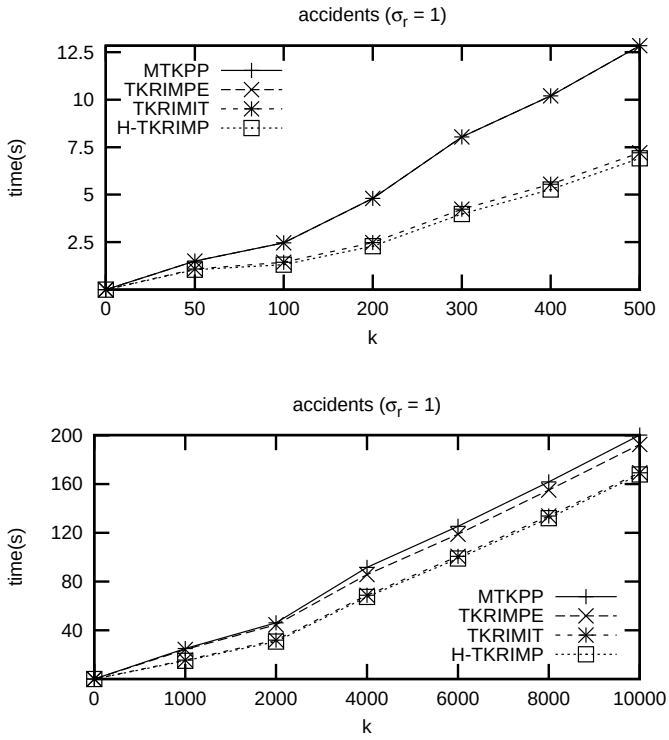


Figure 6.4: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 1\%$)

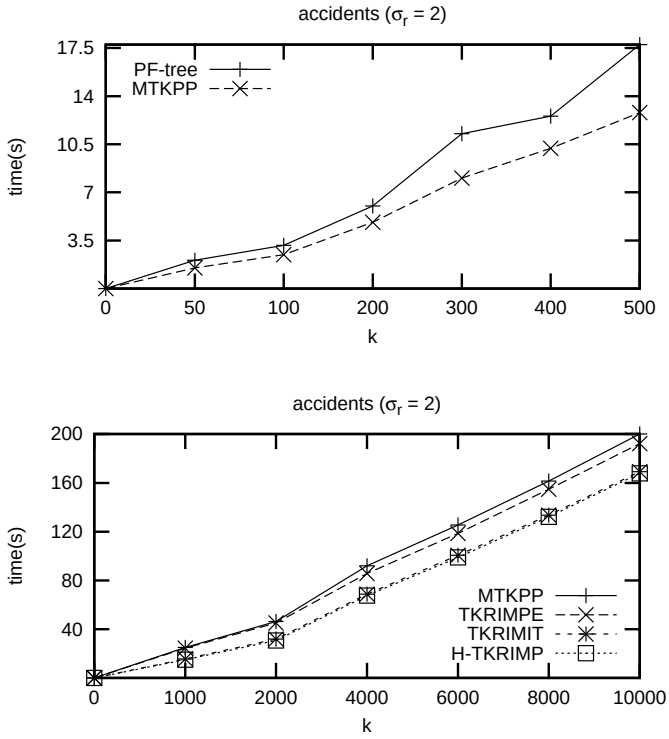


Figure 6.5: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 2\%$)

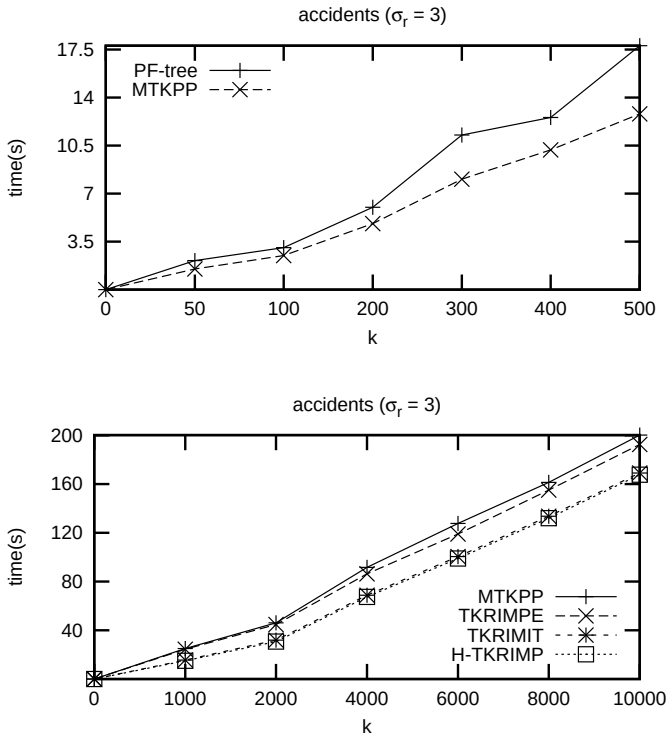


Figure 6.6: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 3\%$)

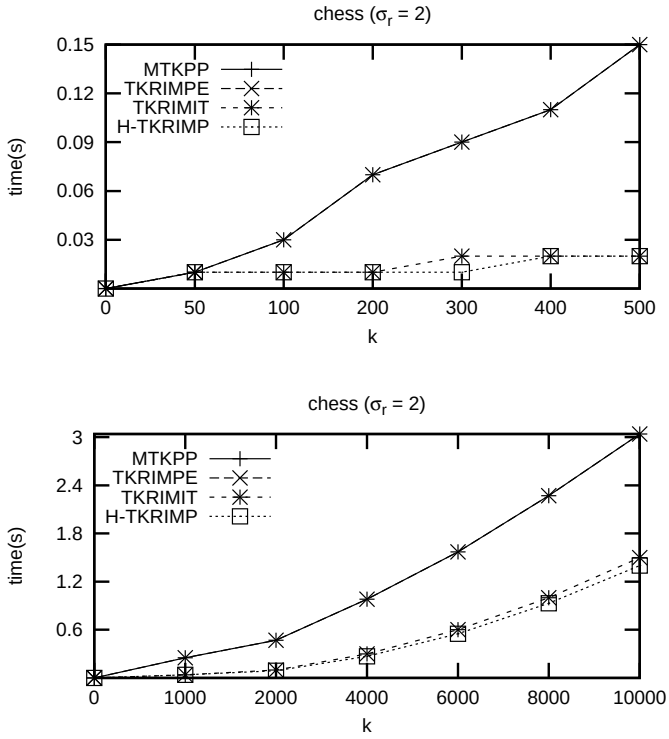


Figure 6.7: Runtime of H-TKRIMP on *chess* ($\sigma_r = 2\%$)

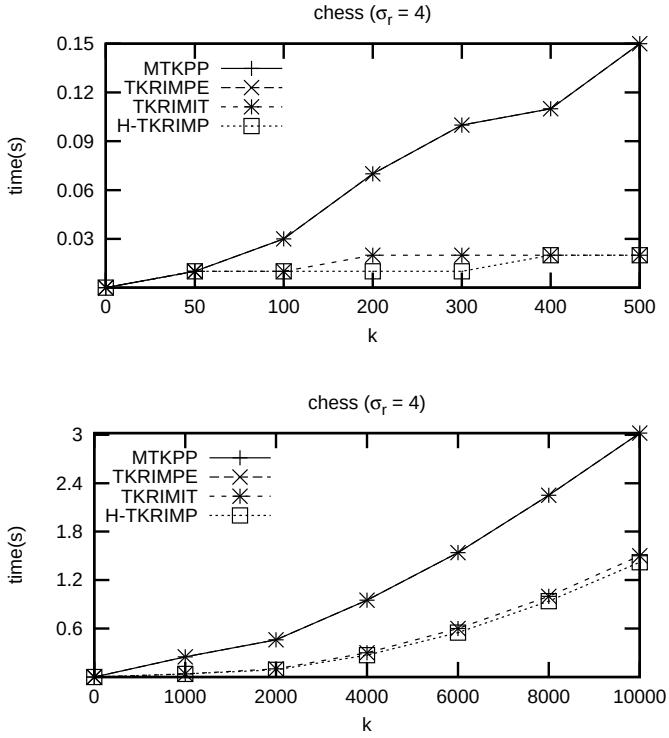


Figure 6.8: Runtime of H-TKRIMP on *chess* ($\sigma_r = 4\%$)

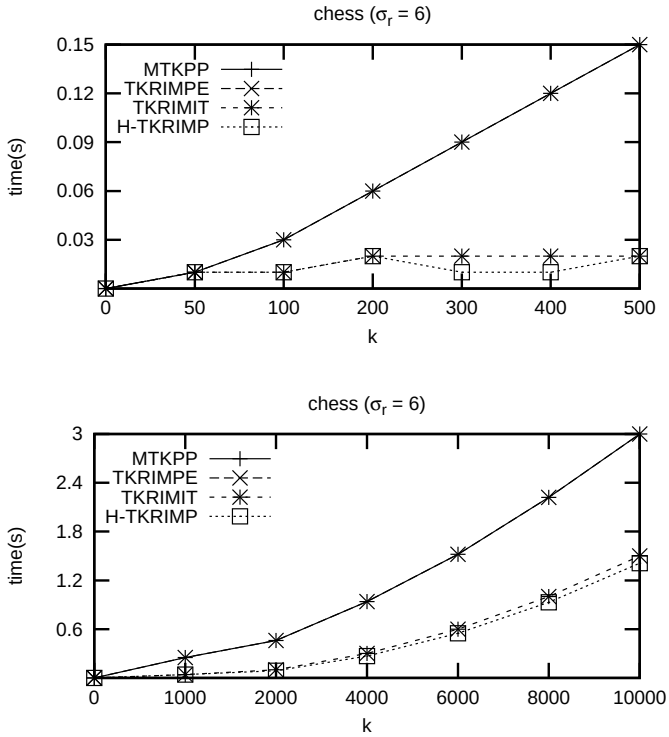


Figure 6.9: Runtime of H-TKRIMP on *chess* ($\sigma_r = 6\%$)

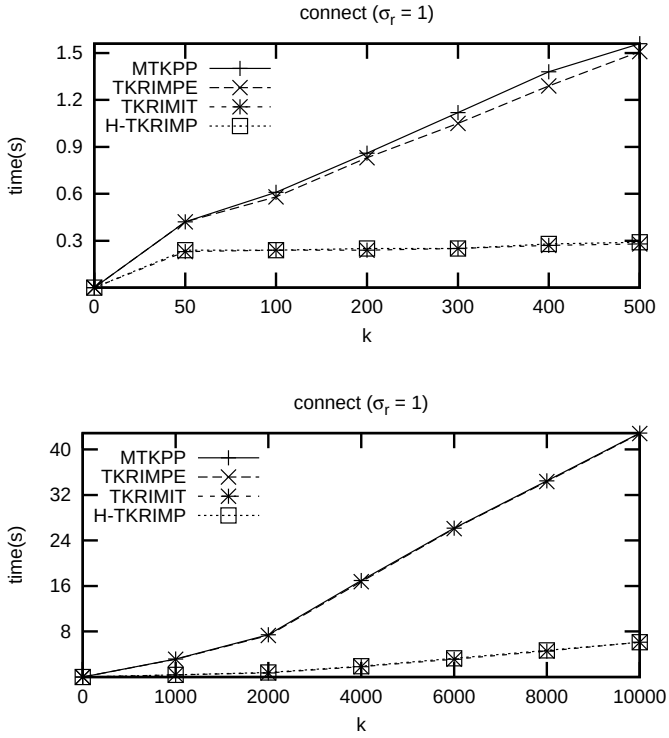


Figure 6.10: Runtime of H-TKRIMP on *connect* ($\sigma_r = 1\%$)

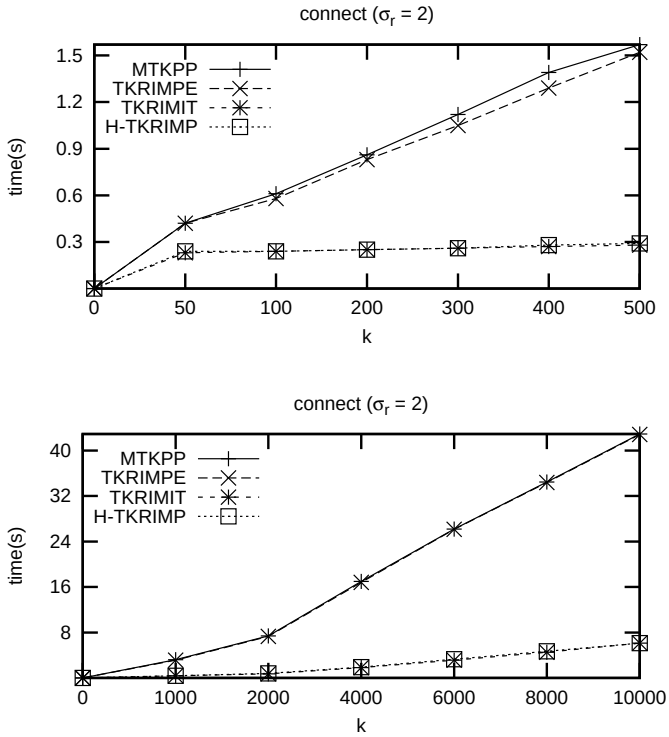


Figure 6.11: Runtime of H-TKRIMP on *connect* ($\sigma_r = 2\%$)

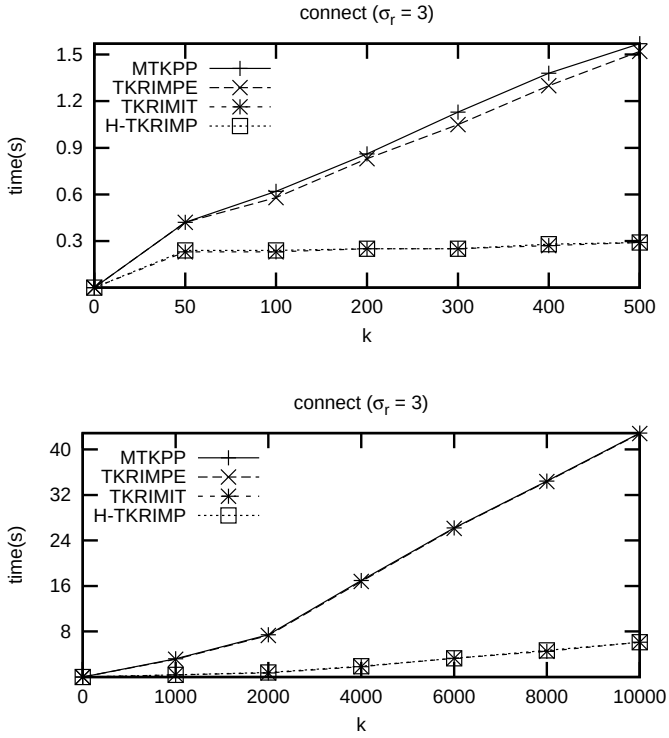


Figure 6.12: Runtime of H-TKRIMP on *connect* ($\sigma_r = 3\%$)

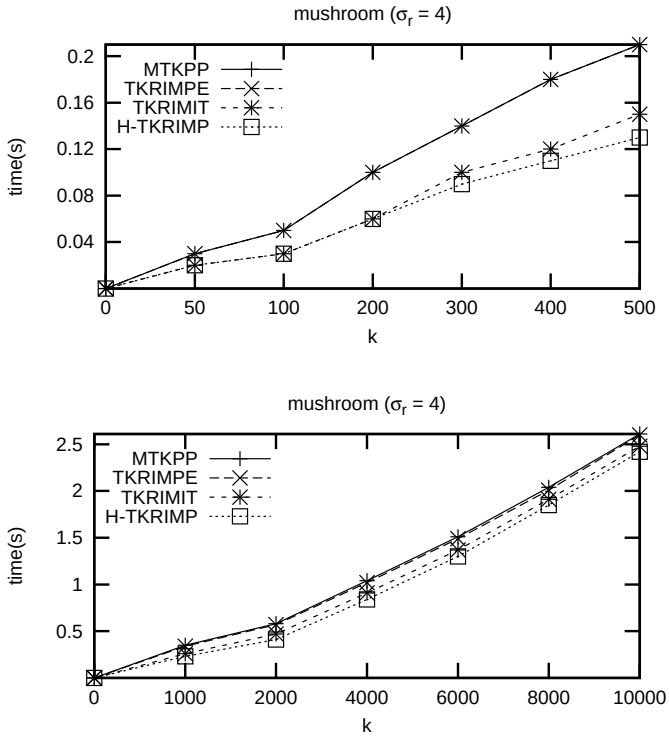


Figure 6.13: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 4\%$)

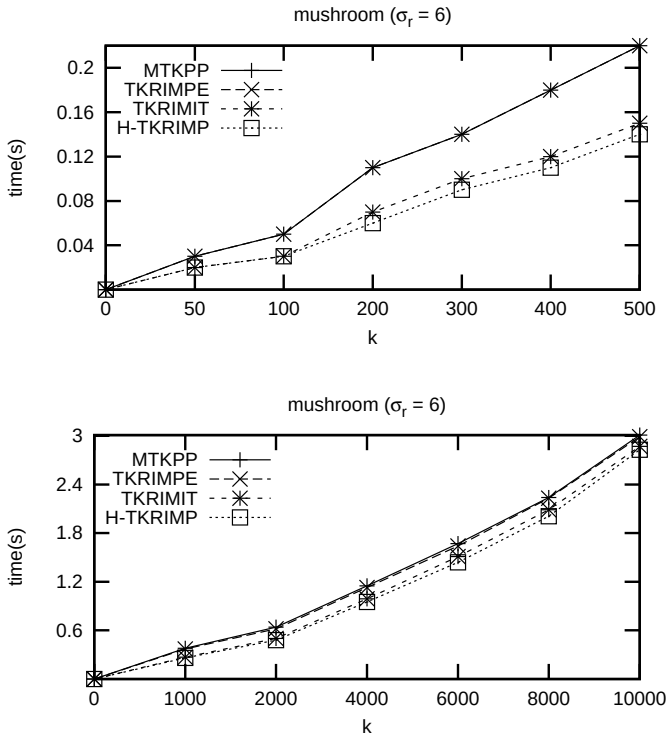


Figure 6.14: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 6\%$)

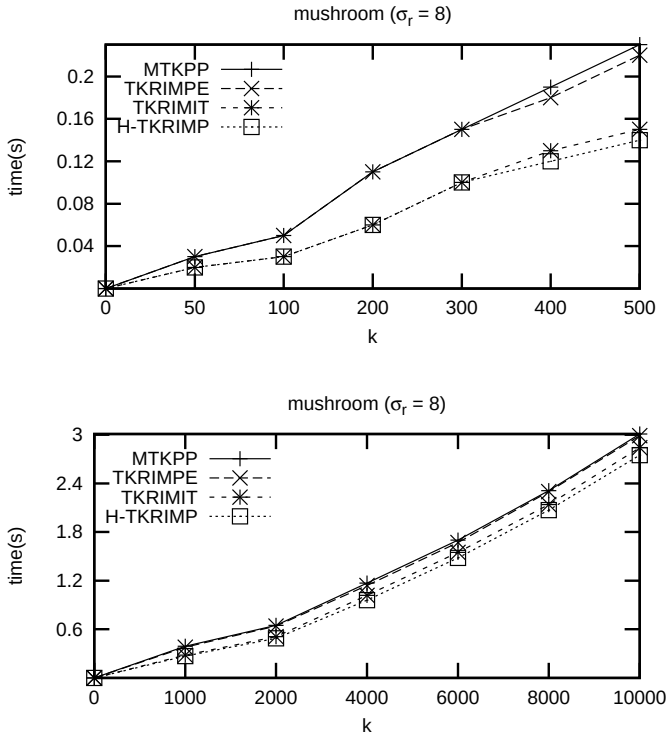


Figure 6.15: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 8\%$)

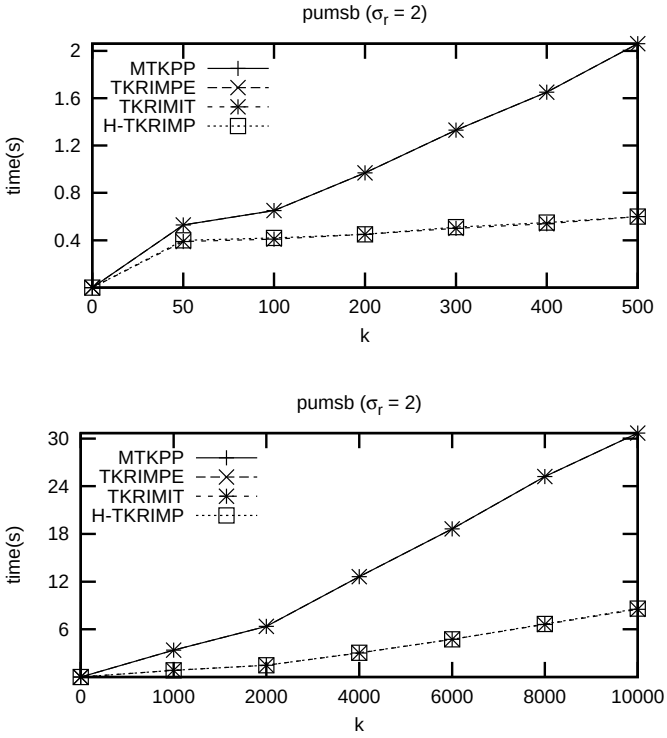


Figure 6.16: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 2\%$)

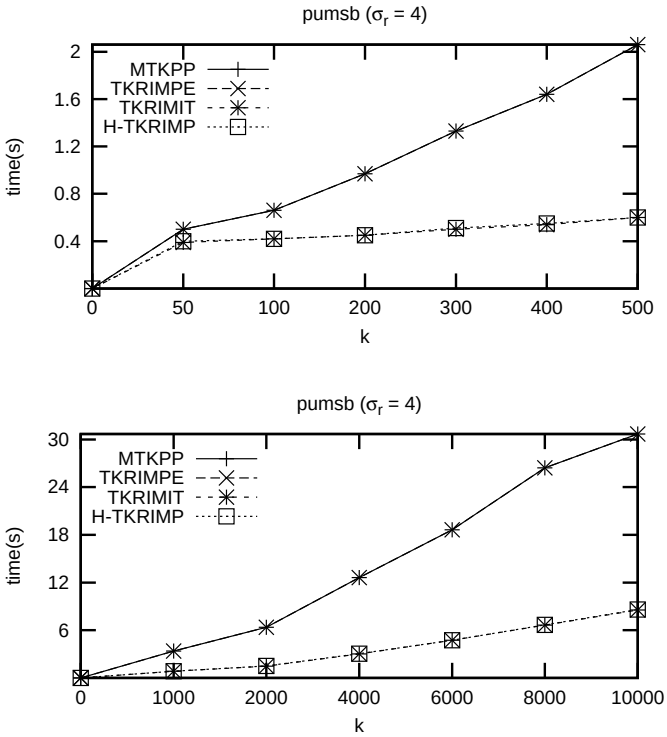


Figure 6.17: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 4\%$)

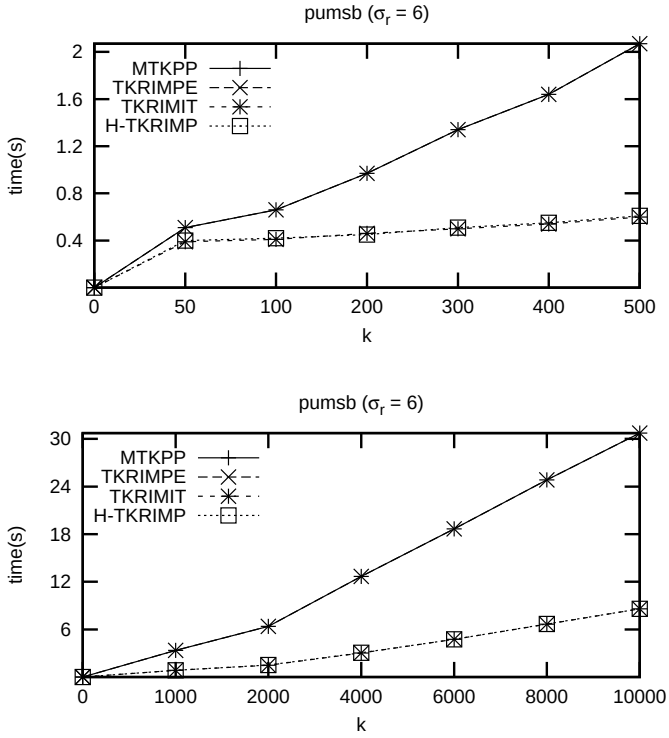


Figure 6.18: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 6\%$)

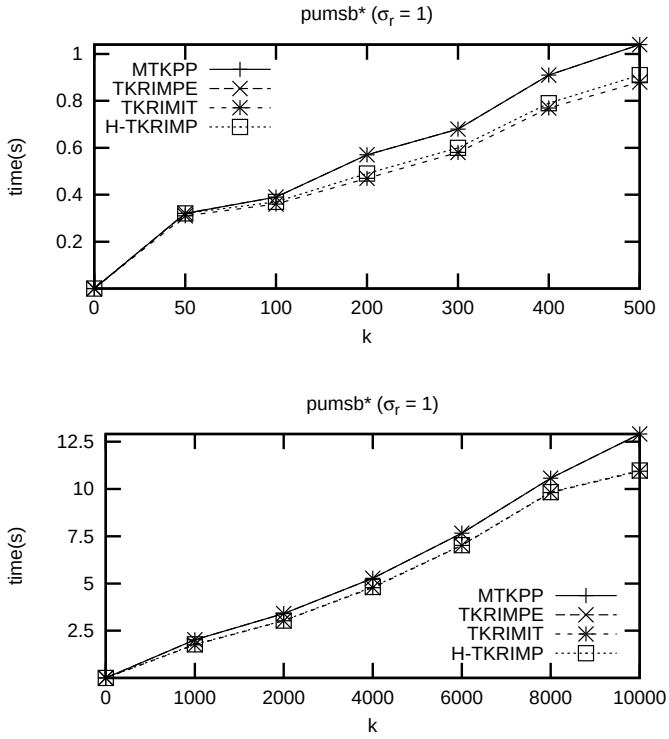


Figure 6.19: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 1\%$)

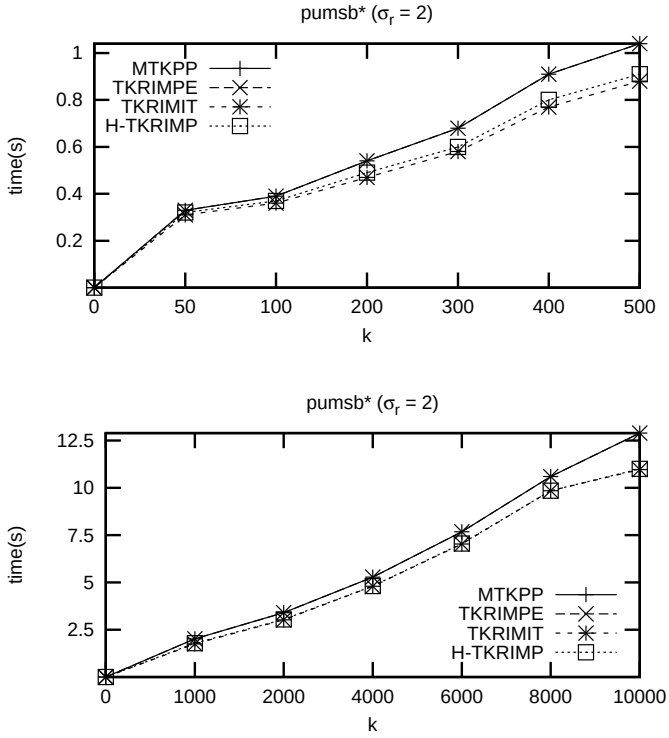


Figure 6.20: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 2\%$)

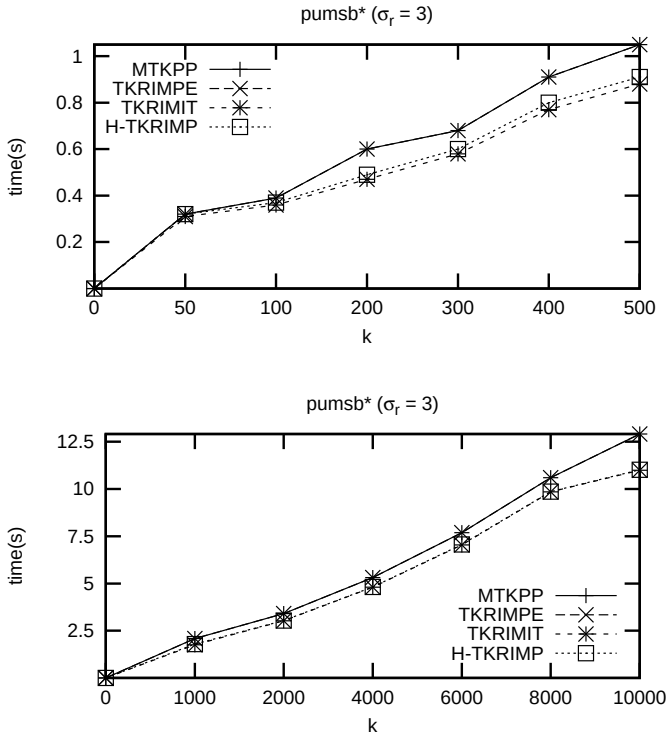


Figure 6.21: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 3\%$)

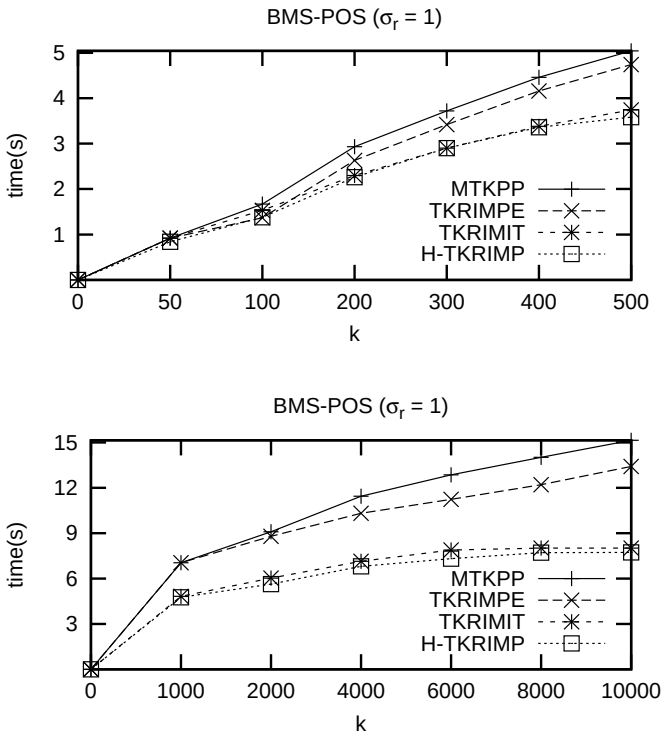


Figure 6.22: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 1\%$)

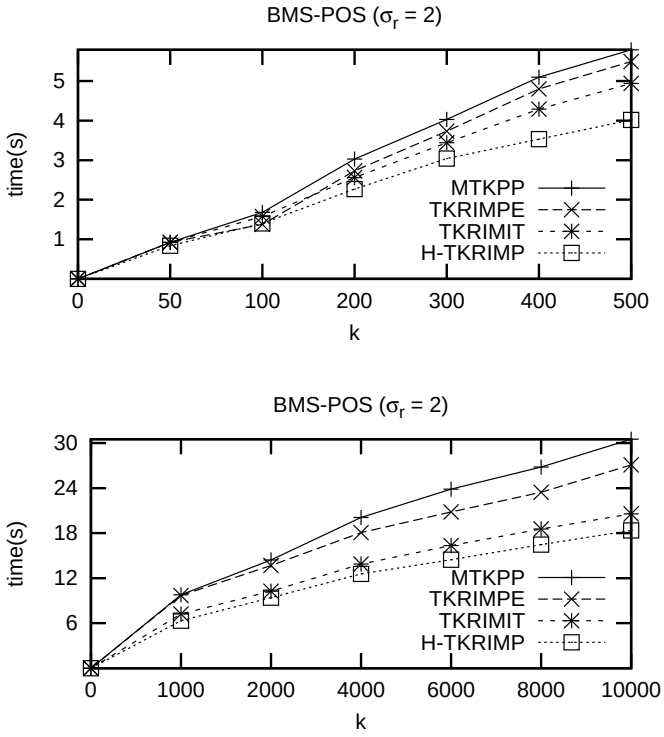


Figure 6.23: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 2\%$)

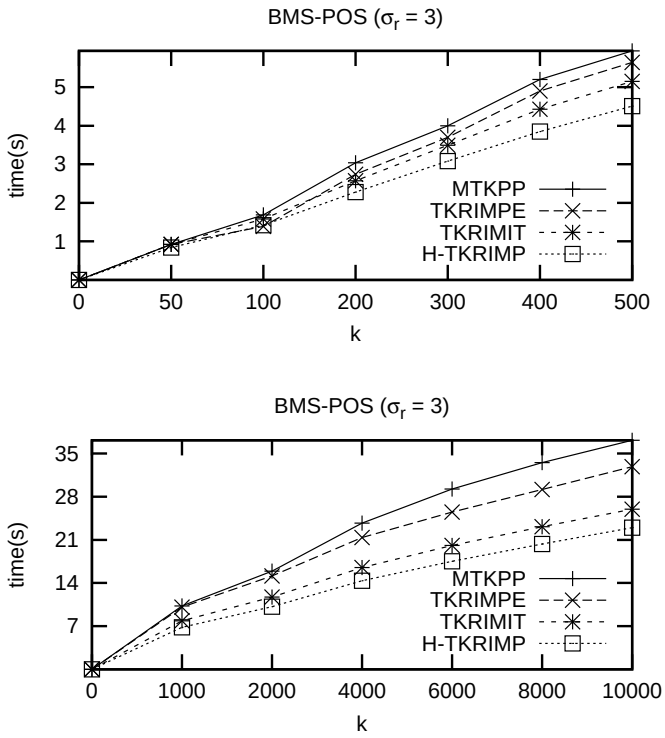


Figure 6.24: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 3\%$)

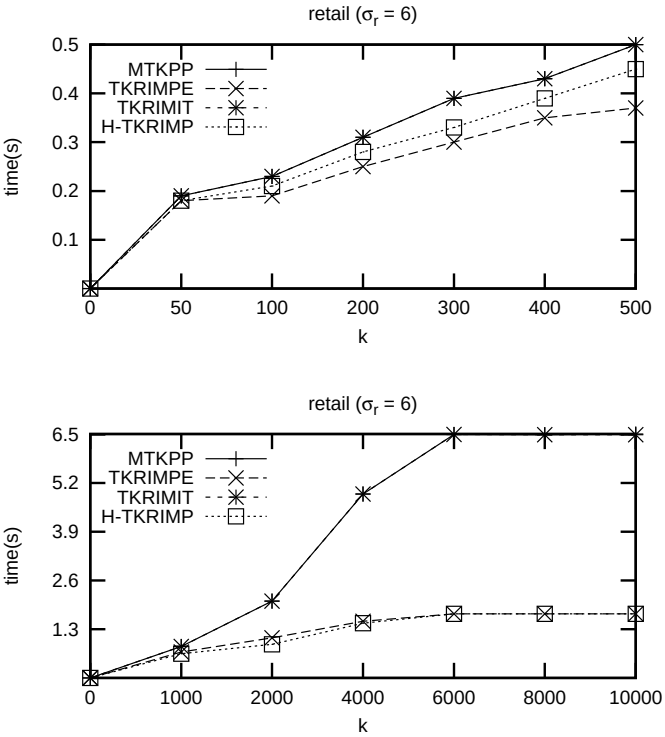


Figure 6.25: Runtime of H-TKRIMP on *retail* ($\sigma_r = 6\%$)

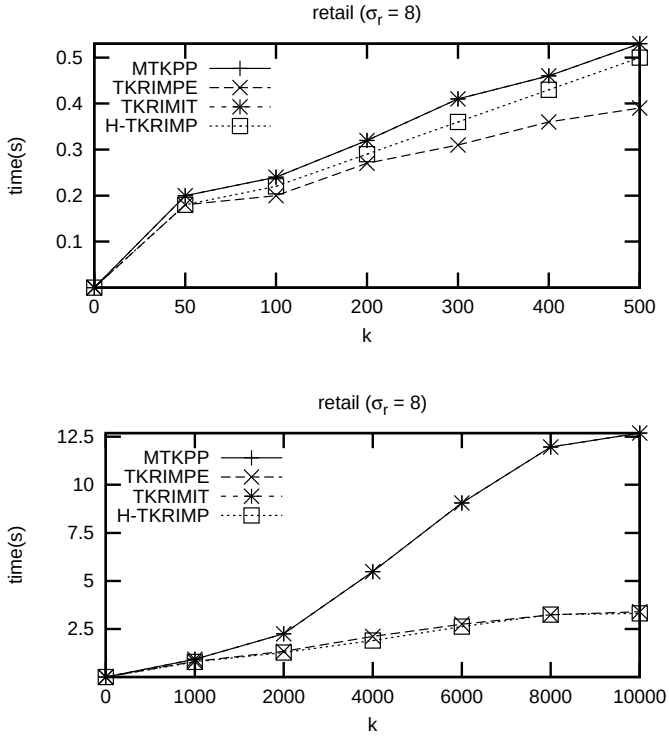


Figure 6.26: Runtime of H-TKRIMP on *retail* ($\sigma_r = 8\%$)

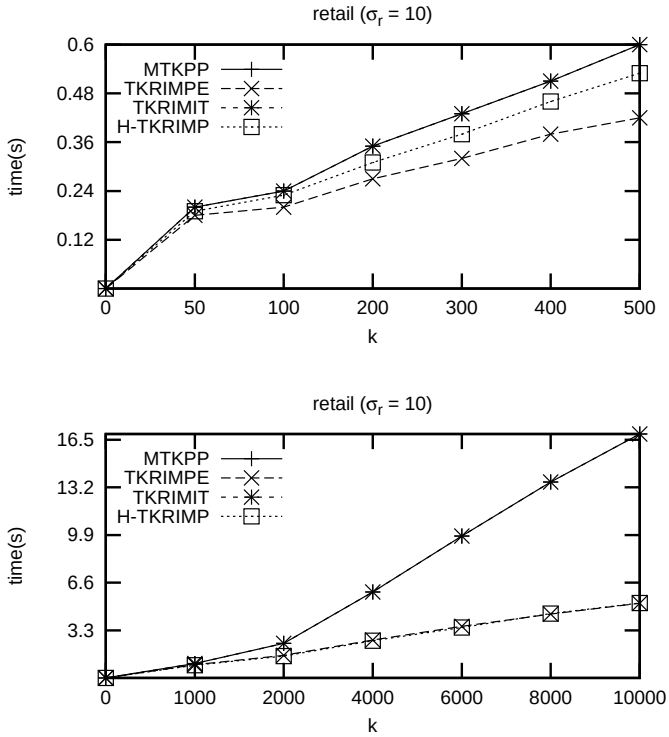


Figure 6.27: Runtime of H-TKRIMP on *retail* ($\sigma_r = 10\%$)

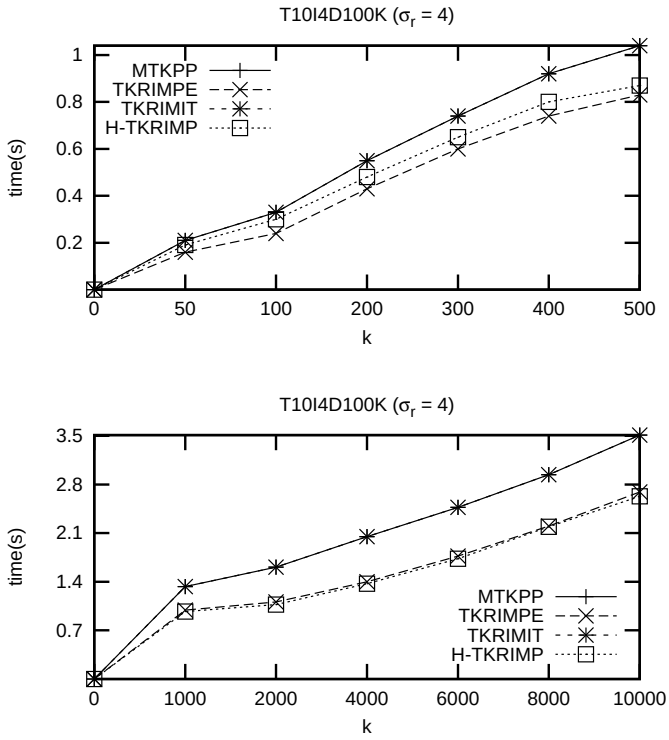


Figure 6.28: Runtime of H-TKRIMP on *T10I4D100K* ($\sigma_r = 4\%$)

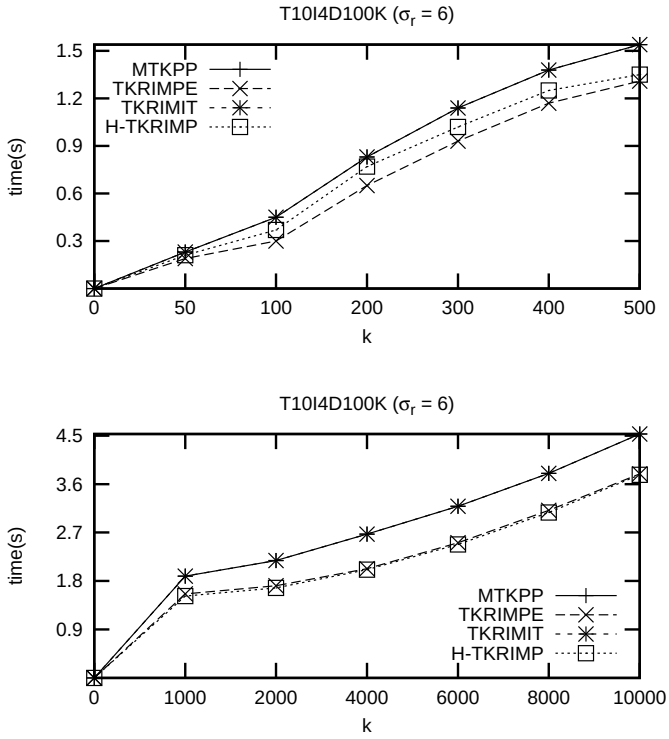


Figure 6.29: Runtime of H-TKRIMP on *T10I4D100K* ($\sigma_r = 6\%$)

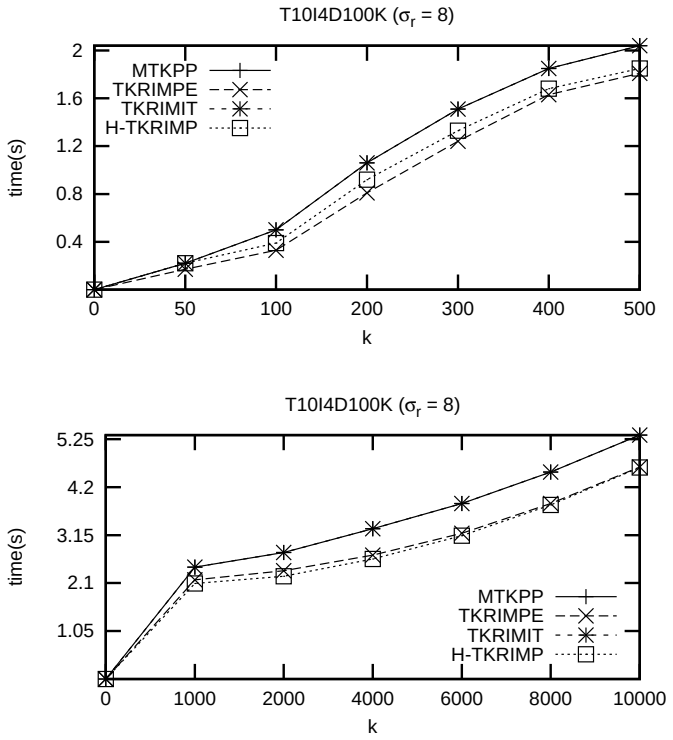


Figure 6.30: Runtime of H-TKRIMP on *T10I4D100K* ($\sigma_r = 8\%$)

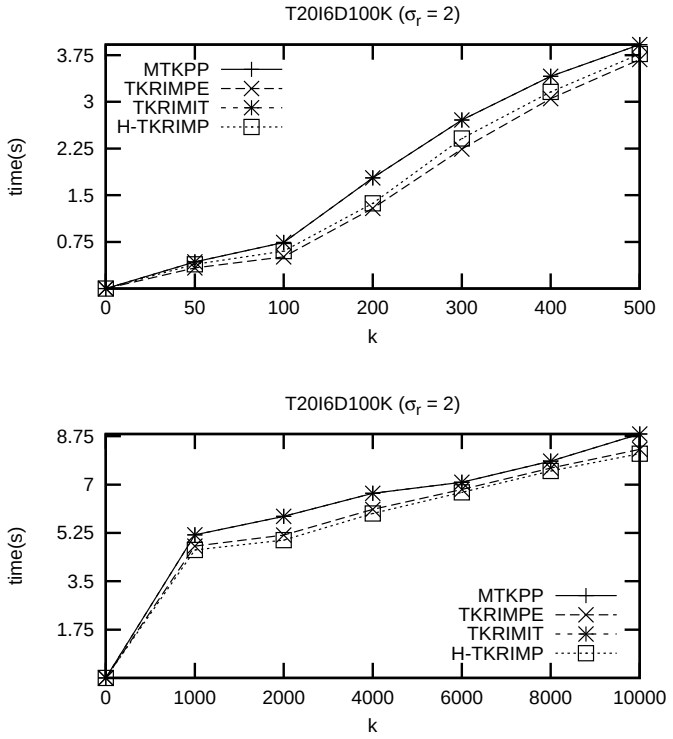


Figure 6.31: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 2\%$)

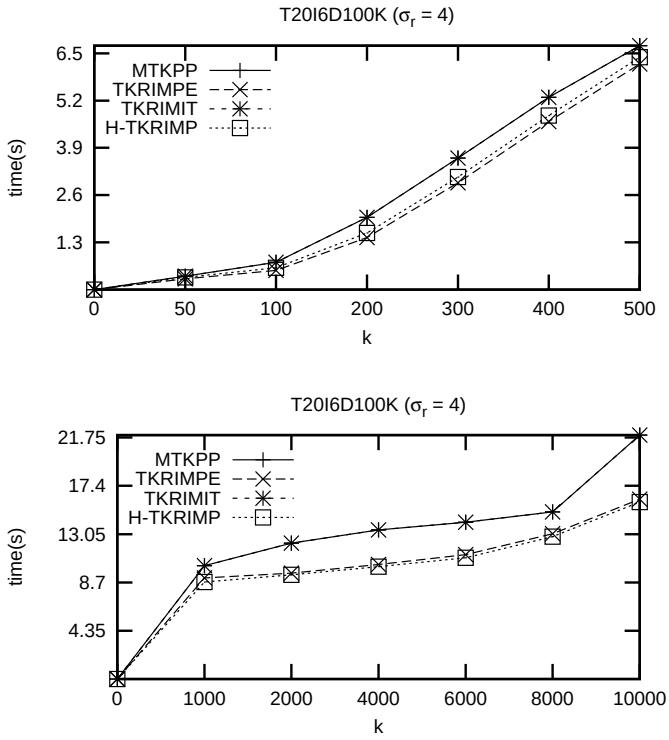


Figure 6.32: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 4\%$)

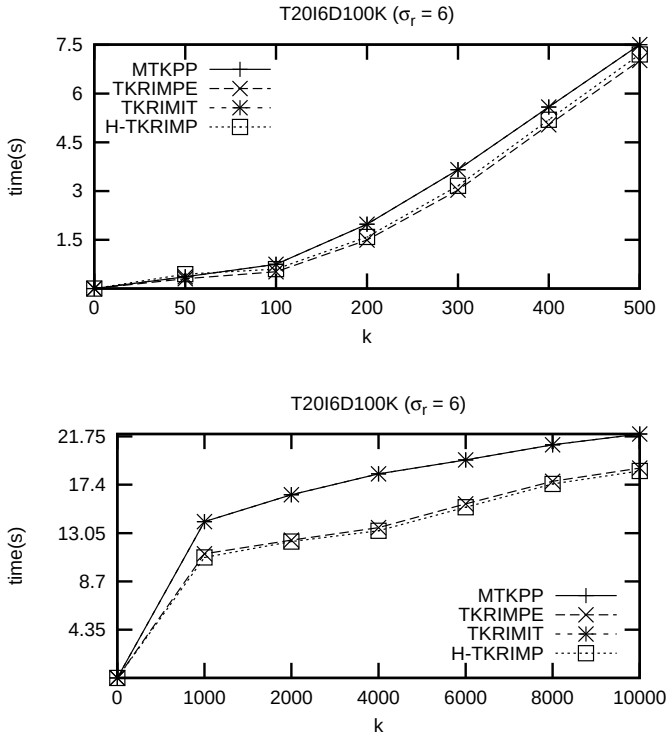


Figure 6.33: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 6\%$)

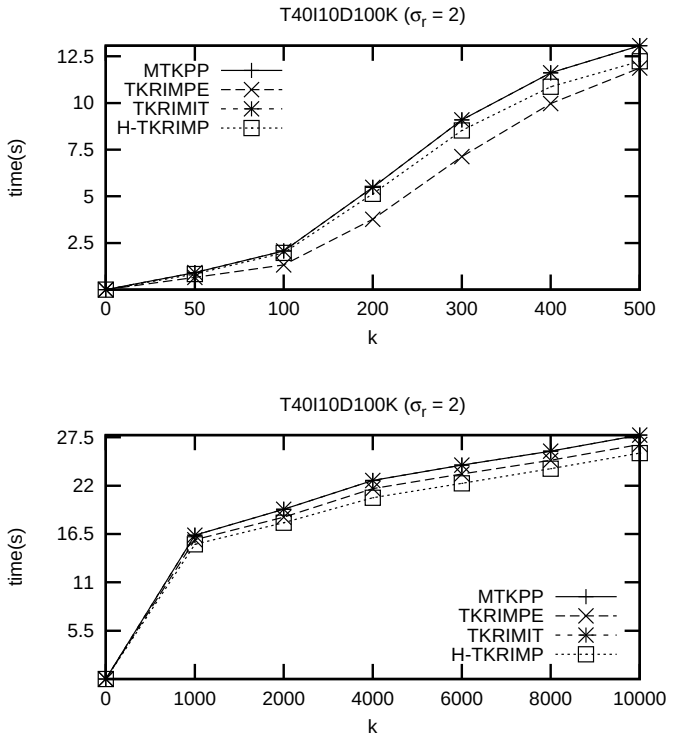


Figure 6.34: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 2\%$)

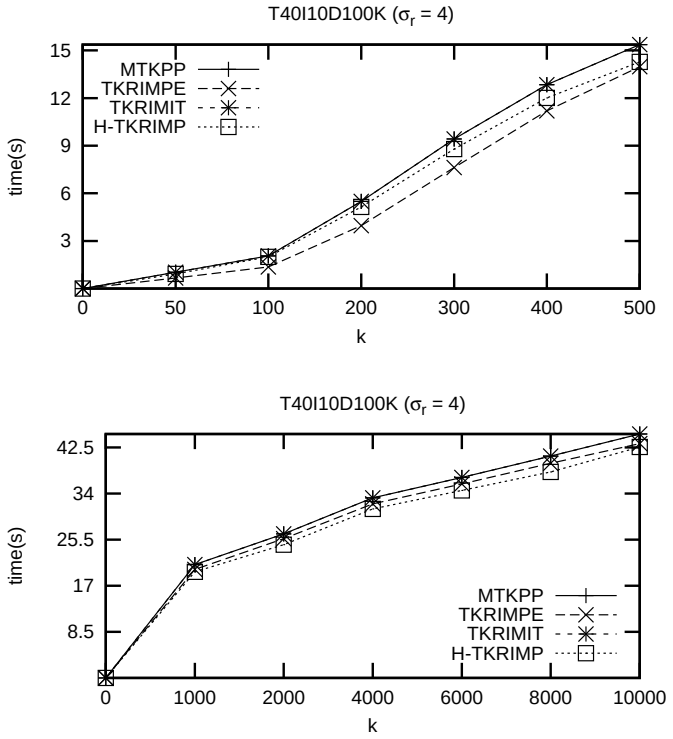


Figure 6.35: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 4\%$)

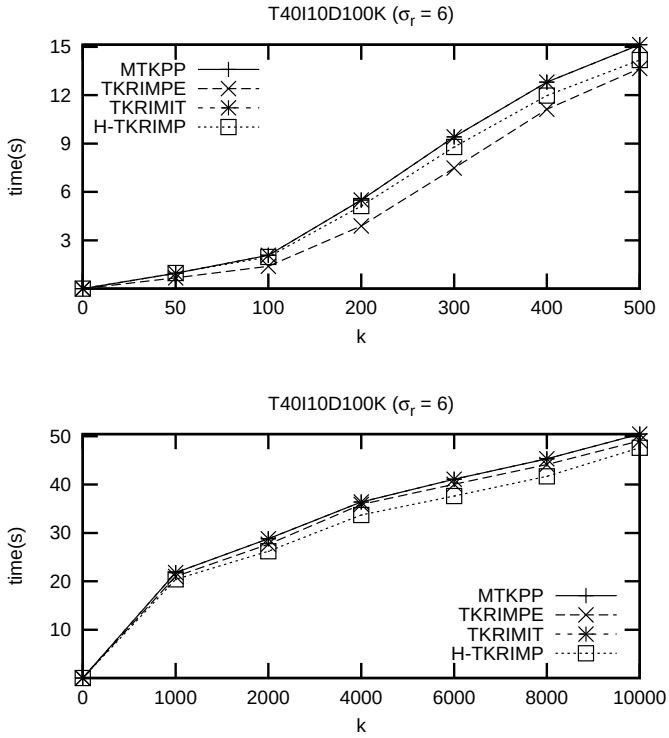


Figure 6.36: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 6\%$)

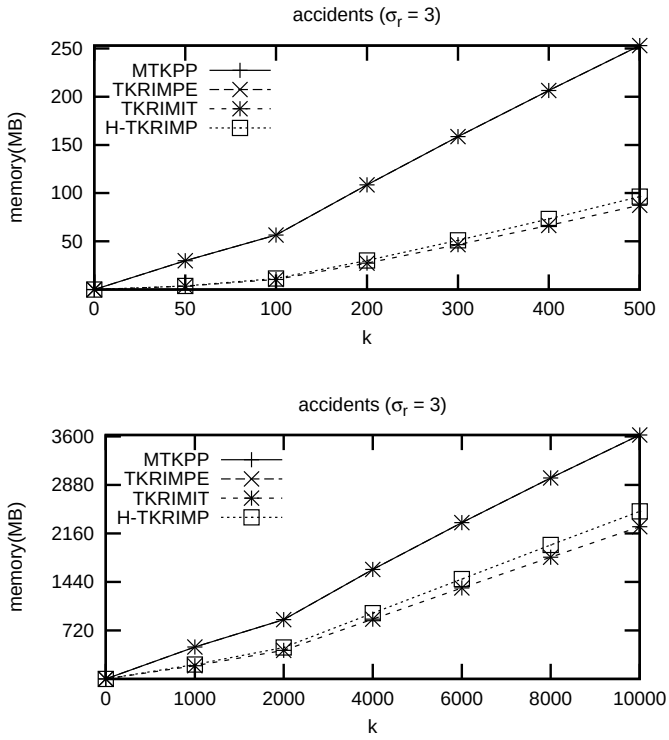


Figure 6.37: Memory usage of H-TKRIMP on *accidents*

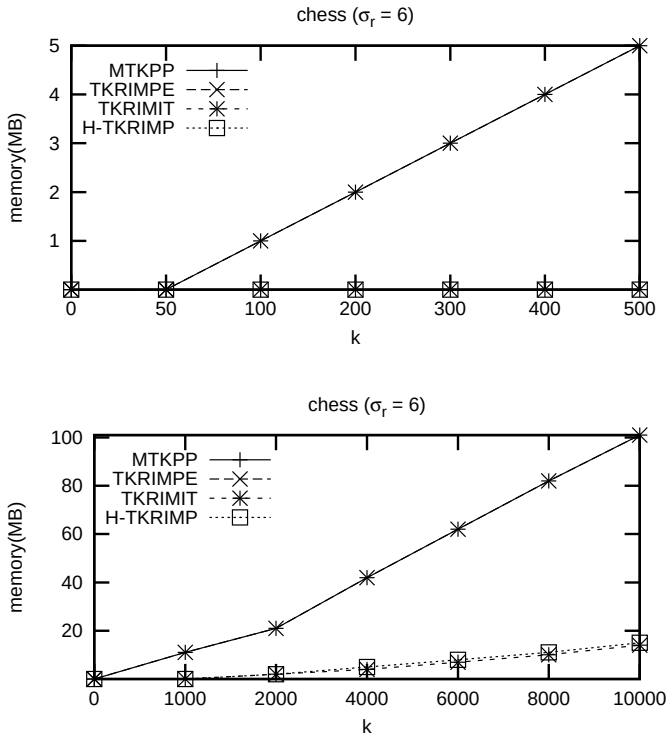


Figure 6.38: Memory usage of H-TKRIMP on *chess*

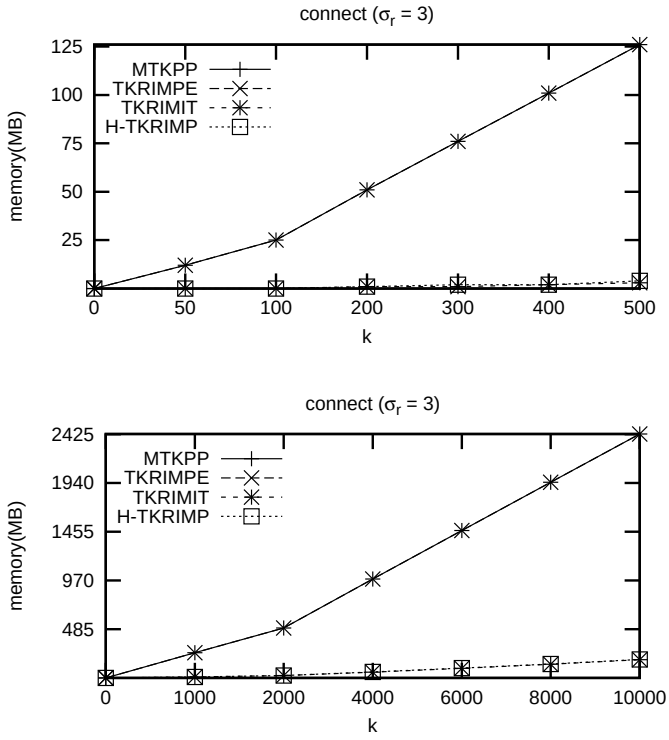


Figure 6.39: Memory usage of H-TKRIMP on *connect*

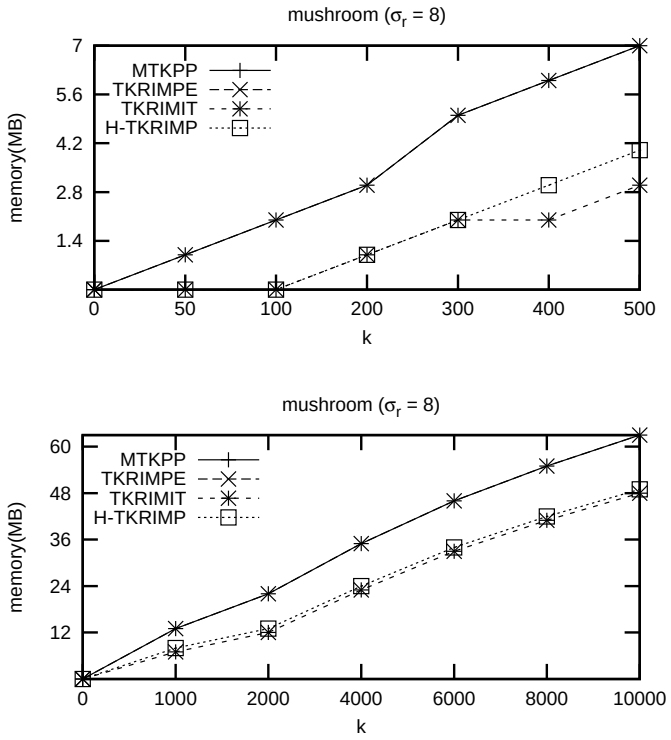


Figure 6.40: Memory usage of H-TKRIMP on *mushroom*

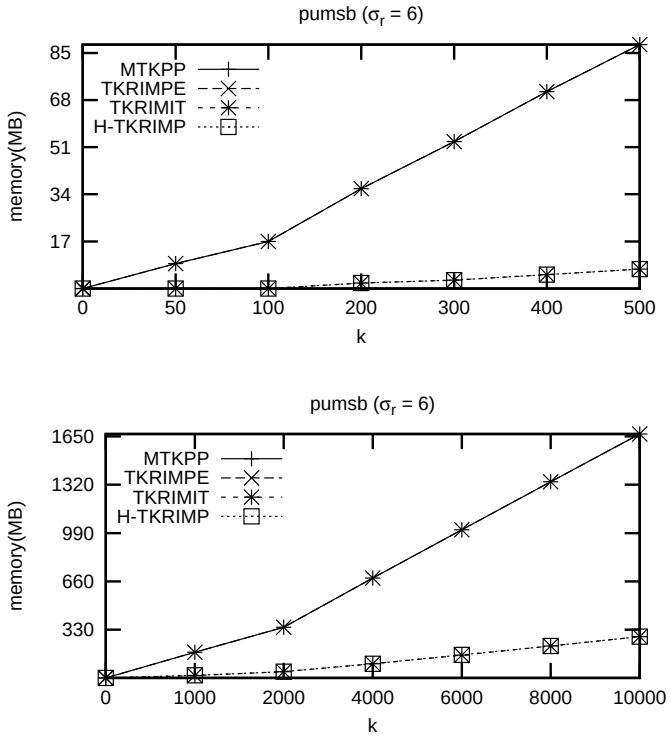


Figure 6.41: Memory usage of H-TKRIMP on *pumsb*

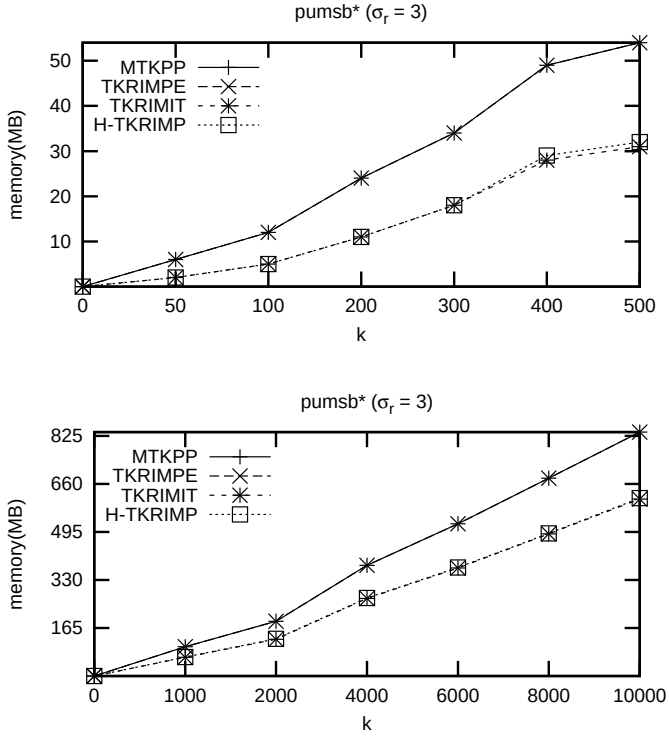


Figure 6.42: Memory usage of H-TKRIMP on *pumsb**

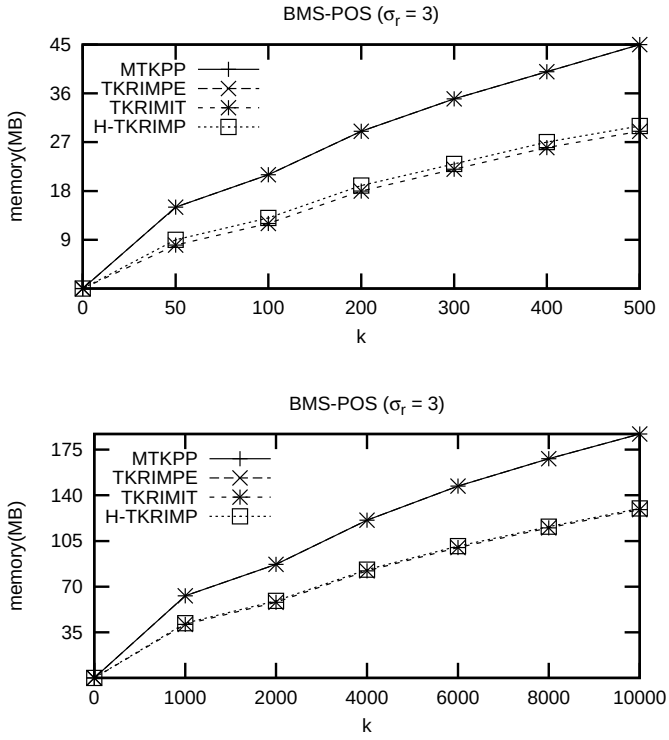


Figure 6.43: Memory usage of H-TKRIMP on *BMS-POS*

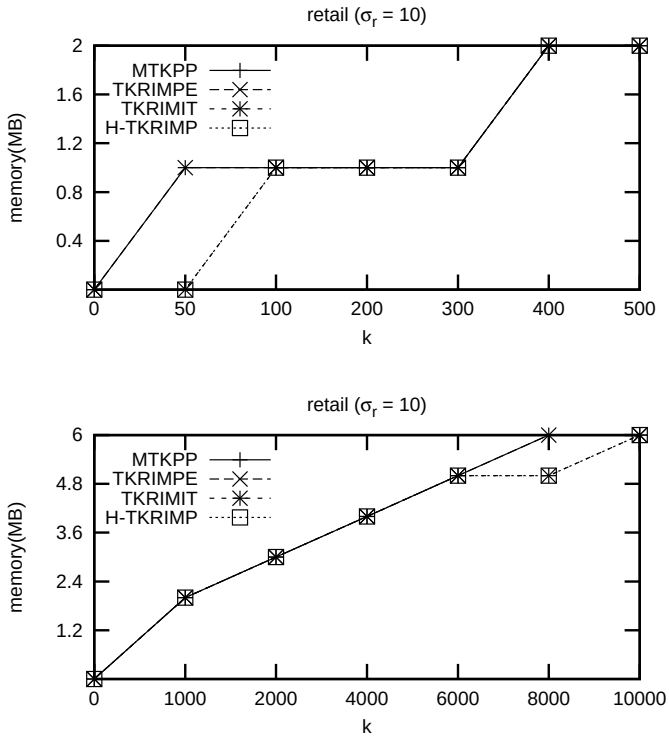


Figure 6.44: Memory usage of H-TKRIMP on *retail*

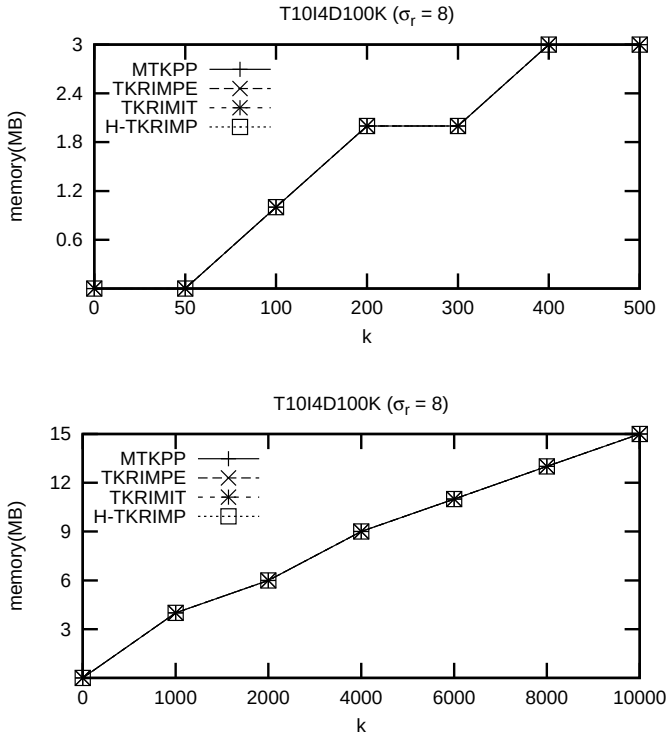


Figure 6.45: Memory usage of H-TKRIMP on *T10I4D100K*

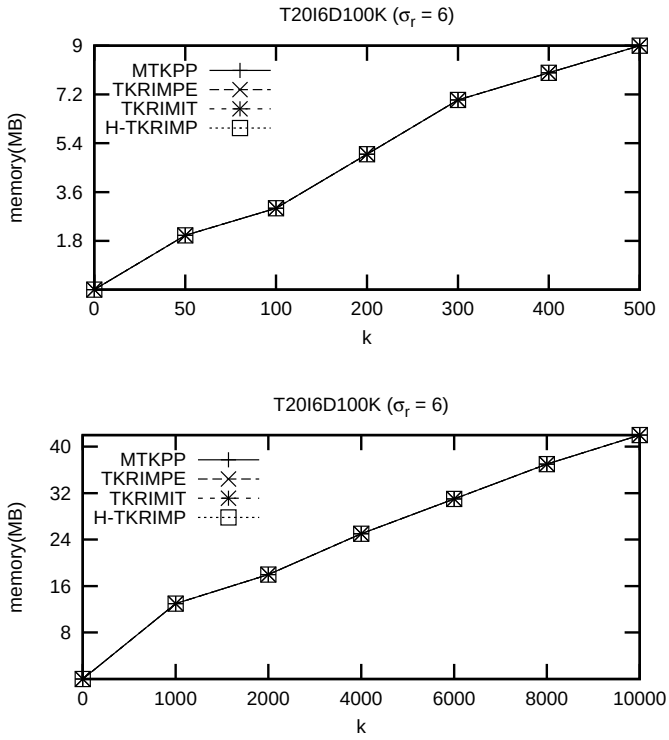


Figure 6.46: Memory usage of H-TKRIMP on *T20I6D100K*

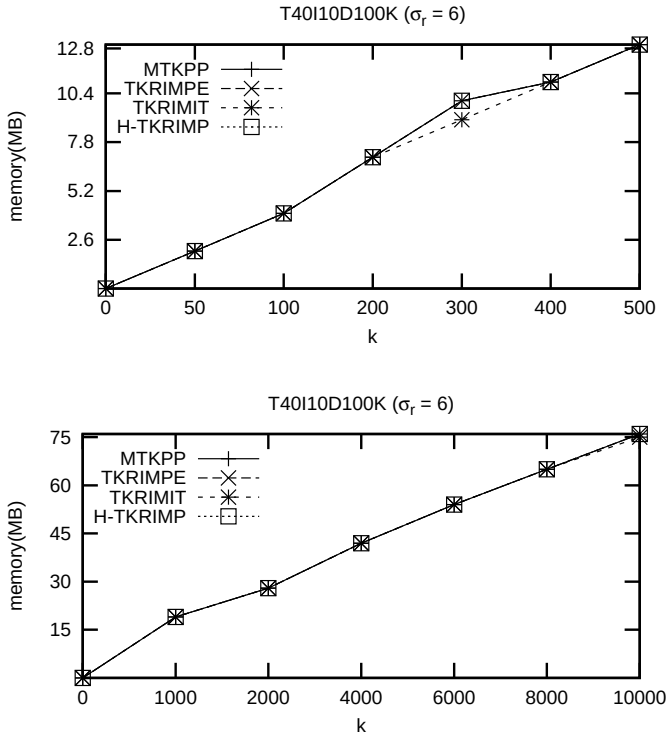


Figure 6.47: Memory usage of H-TKRIMP on *T40I10D100K*

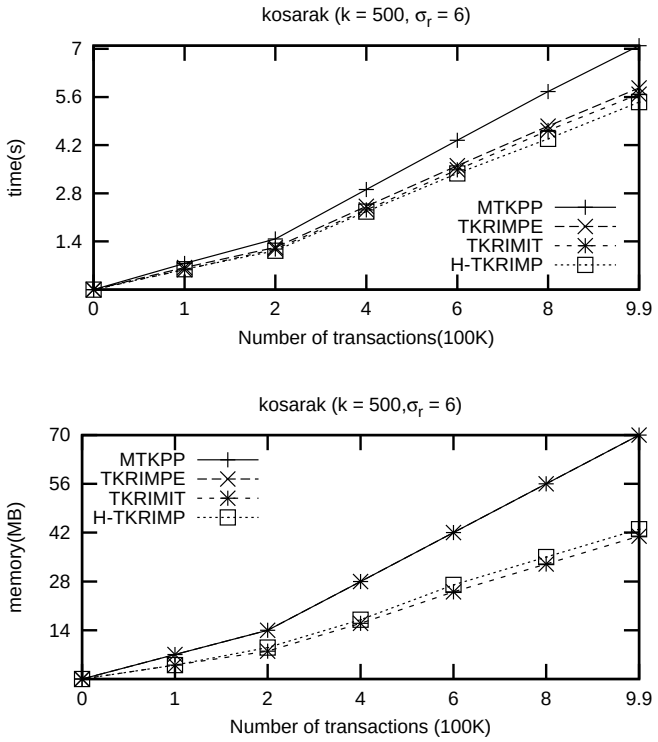


Figure 6.48: Scalability of H-TKRIMP ($k : 500, \sigma_r = 6$)

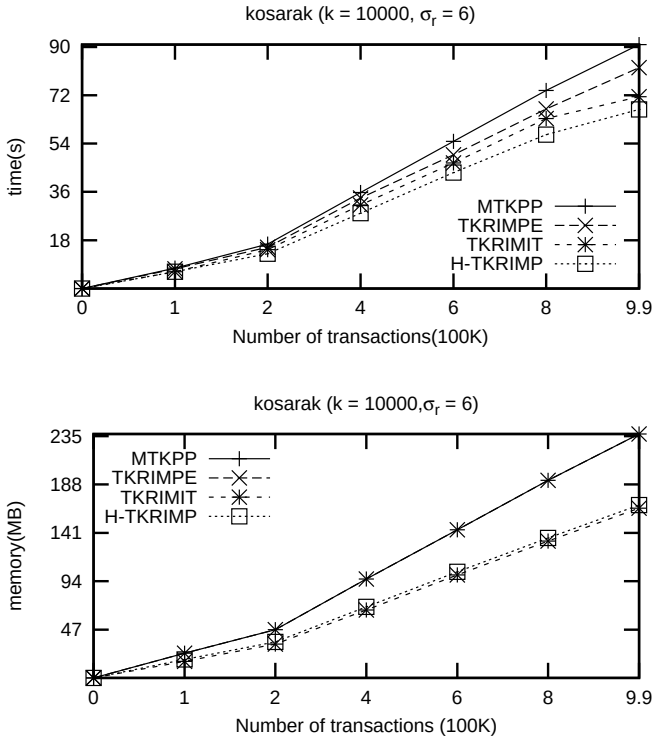


Figure 6.49: Scalability of H-TKRIMP ($k : 10,000, \sigma_r = 6$)

CHAPTER VII

CONCLUSION

7.1 Summary of Dissertation

Recently, Tanbeer et al. proposed an approach for considering the occurrence behavior of patterns (Tanbeer et al., 2009), *i.e.* whether the pattern occur regularly, irregularly or mostly in specific time period of a transactional database. Hence, a pattern is a regular-frequent if it is frequent in terms of the support measure, as defined in (Agrawal and Srikant, 1994), and if it regularly appears (measure of regularity/ periodicity of the pattern which considers the maximum period at which the pattern occurs). To discover a set of regular-frequent itemsets, the authors proposed a highly compact tree structure named *Periodic Frequent patterns tree (PF-tree)* to maintain the database content, and a pattern growth-based algorithm to mine a complete set of regular-frequent itemsets with user-given support and regularity thresholds.

However, it is well-known that the support-based approaches tend to produce a huge number of patterns and it is not easy for end-users to determine a suitable support threshold. Thus, the top-*k* significant patterns mining framework, which allows the users controlling the number of patterns (k) to be mined (which is easy to specify) without a support threshold, is an interesting approach (Han et al., 2002). Therefore, the contributions of this dissertation have focused on the problem of mining top-*k* regular-frequent itemsets as follows.

Chapter 3 introduced the the problem of mining k regular itemsets with the highest supports, called *Top-K Regular-frequent Itemsets Mining*, that allows users to specify the number of regular-frequent itemsets to be mined. From this problem, the users have to specify two parameters: (*i*) a number of desired results (k) (*i.e.* specify the value of k instead of setting the support threshold); and (*ii*) a regularity threshold (*i.e.* to see whether an itemset occurs regularly). Consequently, an efficient algorithm named *Mining Top-K Periodic(Regular)-Frequent Pattern (MTKPP)* was proposed. To mine top-*k* regular-frequent itemsets, the top-*k* list structure (with hash table) and the best-first search strategy were also devised for efficiency reasons. From the experimental results, it can be observed that MTKPP ran faster than PF-tree which exactly mines the same results (with the small and large values of k) and scaled linearly relative to the size of the input database. Thus, the MTKPP algorithm is recommended when the users desire to control the number of outputs.

Subsequently, an efficient algorithm called *Top-K Regular-frequent Itemsets Mining with database Partitioning and support Estimation (TKRIMPE)* to mine a set of k regular-frequent itemsets with the highest supports was presented in Chapter 4. TKRIMPE was devised to improve the performance of MTKPP by trying to reduce the processing time in the intersection process. In TKRIMPE, the database partitioning and support estimation techniques were also introduced to dismiss unnecessary computational costs and to cut down the search space. The experimental study shows that TKRIMPE ran faster than MTKPP when the database is sparse, and also has the similar performance on dense datasets.

Chapter 5 proposed an efficient algorithm, called *Top-K Regular-frequent Itemsets based on Interval Tidset representation (TKRIMIT)* to mine top-*k* regular-frequent itemsets. In addition, a new concise representation, *Interval Tidset* representation, used to collect the set of tids that each considered itemset occurs was introduced. Based on the interval tidset representation, TKRIMIT can reduce the number of maintained tids that each itemset occurs. This is caused to save the memory usage and runtime to mine the top-*k* regular-frequent itemsets. As shown by results from the experiments, the use of interval tidset representation gives the better performance from the use of normal tidset representation especially on dense datasets.

In Chapter 6, the combination between the database partitioning technique and interval tidset representation was proposed. The *Hybrid representation* was introduced to maintain the tids that each itemset occurs. It composes of normal tidset representation (*i.e.* sets of normal tids that each itemset occurs) and interval tidset representation (*i.e.* sets of concise (wrap up) tids that each itemset appears). By using this representation, a simple heuristic was devised to choose a proper presentation for the occurrence behavior of each itemset. Consequently, an efficient algorithm based on database partitioning technique and the hybrid representation called *Hybrid representation on Top-K Regular-frequent Itemsets Mining based on database Partitioning (H-TKRIMP)* was introduced. As shown in the experiments, H-TKRIMP can run faster than the other algorithms on both sparse and dense datasets with the small and large values of k.

In summary, this dissertation has studied the regular-frequent itemsets mining problem and then proposed the problem of top-*k* regular-frequent itemsets mining which allows users to control the number of results to be mined. To mine the top-*k* regular-frequent itemsets, the efficient and scalable single-pass algorithms based on the top-*k* list structure have been suggested. They consist of two steps: (*i*) top-*k* initialization: scan database to construct the top-*k* list of regular items with their supports, regularities, and tidsets; and (*ii*) top-*k* mining: merge each pair of entries in the top-*k* list using the best-first search strategy (*i.e.* first consider the itemsets with the highest supports), then intersect their tidsets to calculate regularity and support of each itemset.

From both steps, it can be observed that the mining process consumes high processing time in the intersection process. Therefore, the partitioning and estimation techniques were invented to reduce the cost of intersection by dismissing some unnecessary computing. From the experiment (as mentioned in Chater 4), it can be seen that applying both techniques can help algorithms to achieve a good performance especially on sparse datasets with the small and large values of k. In addition, to gain a better performance on dense datasets, the number of maintained tids was emphasized. If the number of maintained tids is very few, mining algorithms would spend few time in the intersection process. Thus, a new concise representation was devised to reduce the number of maintained tids of each itemset in the top-*k* list. It uses only one positive and one negative tids to represent a group of consecutive continuous tids. By using this representation, the performance of mining algorithms grow up in terms of time and space especially on dense datasets. Finally, to have a good performance on both sparse and dense datasets without knowing the characteristic of datasets in advance, the combination of database partitioning technique and the concise representation was proposed. Then, the hybrid representation was also devised to maintain tidsets followed by occurrence behavior of each itemset. If the itemset occurs frequent, the concise representation is applied. Otherwise, the original representation is employed. The results show that the use of hybrid representation and partitioning technique can give a good performance on both sparse and dense dataset with the small and large values of k comparing with the other proposed algorithms.

7.2 Discussion

Although, this dissertation introduced the top-*k* regular-frequent itemsets mining and some efficient algorithms that achieve a good performance on both sparse and dense datasets with the small and large values of k, there exists some limitation which can be categorized into several points.

First, the proposed algorithms are based on single scanning and maintaining the tids for each itemset in the top-*k* list. Though, there exists a concise representation which helps to save runtime and memory space, the mining algorithms still spend a lot of time in the intersection process and a lot of memory to maintain tids. Then, the interesting problem is to design a new algorithm that can share the common sets of tids among the itemsets in the top-*k* list. This way of doing may help the mining algorithms to save time to intersect and space to maintain tids during mining process.

Second, to discover the top-*k* regular-frequent itemsets in the presence memory constraint, the proposed algorithms would have a problem on memory consumption because they have to

maintain tidset for each itemset the set of results. Thus, a new approach using the secondary storage or using incremental technique to separately consider each partition of database should be discussed in the direction of reducing required memory.

Third, the problem of top-*k* regular-frequent itemsets mining require two parameters: (*i*) regularity threshold (σ_r) and (*ii*) the number of desired results (k). In some cases, users would suffer from the setting a suitable regularity threshold. Thus, the interesting problem is to automatically specify the regularity threshold to mine the top-*k* regular-frequent itemsets. To come up with an appropriate regularity threshold, one needs to have detailed knowledge about the mining query and the task-specific data, and be able to estimate, without mining, how many itemsets would be generated with a particular threshold. Unlike a support threshold, the setting of a regularity threshold is quite subtle: a too large threshold may lead to the generation of thousands of itemsets, whereas a too small one may often generate very few or no answers. Therefore, the avoiding of the setting of regularity threshold by using other criteria to find the suitable threshold might become an important task.

Fourth, the problem of top-*k* regular-frequent itemsets mining works only on static database (*i.e.* no updated record). Therefore, another interesting direction is to study the problem of mining top-*k* regular-frequent itemsets mining from incremental databases and data streams. In the past few years, research in data streams (also incremental databases) has attracted a lot of researchers. A data stream is a continuous, unbounded, and timely ordered sequence of data elements generated at a rapid rate. Unlike traditional static databases, stream data, in general, has additional processing requirements; *i.e.*, each data element should be examined at most once and processed as fast as possible with the limitation of available memory. Even though mining user-interest based patterns from data stream has become a challenging issue, interests in online stream mining for discovering such patterns dramatically increased. Hence, to find top-*k* regular-frequent itemsets efficiently from data streams, an efficient algorithm that can capture the stream content with one scan and can competently mine the resultant itemsets is required. Since the proposed algorithms scan database once, then it can be improved the algorithms to directly mine top-*k* regular-frequent itemsets from data streams.

The author strongly believe that, with the proposed algorithms and the proposed approach, it could seen many interesting, or the ultimate, solution to the mining regular patterns in the near future.

References

Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. 2001. A tree projection algorithm for generation of frequent item sets. Journal of Parallel and Distributed Computing 61 ,3:350–371.

Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In International Conference on Data Engineering, pp. 3–14, Los Alamitos, CA, USA. IEEE Computer Society.

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules in large databases. In VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile,September 12-15, pp. 487–499.

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. 1993. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, pp. 207–216.

Komate Amphawan, Philippe Lenca, and Athasit Surarerks. 2009. Mining top-k periodic-frequent patterns without support threshold. In The 3rd International Conference on Advances in Information Technology, IAIT'09, Bangkok, Thailand, December 1-5, volume 55 of CCIS, pp. 18–29. Springer.

Roberto J. Bayardo. 1998. Efficiently mining long patterns from databases. In SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, pp. 85–93.

Fernando Berzal, Juan-Carlos Cubero, Nicolás Marín, and José-María Serrano. 2001. Tbar: An efficient method for association rule mining in relational databases. Data and Knowledge Engineering 37,1:47–64.

Ramkishore Bhattacharyya and Balaram Bhattacharyya. 2007. High confidence association mining without support pruning. In Ashish Ghosh, Rajat K. De, and Sankar K. Pal, editors, Pattern Recognition and Machine Intelligence, Second International Conference, PReMI 2007, Kolkata, India, December 18-22, 2007, Proceedings, volume 4815 of Lecture Notes in Computer Science, pp. 332–340. Springer.

Francesco Bonchi and Claudio Lucchese. 2005. Pushing tougher constraints in frequent pattern mining. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia

Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, volume 3518 of Lecture Notes in Computer Science, pp. 114–124. Springer.

Sergey Brin, Rajeev Motwani, and Craig Silverstein. 1997a. Beyond market baskets: generalizing association rules to correlations. In ACM SIGMOD/PODS, pp. 265–276.

Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. 1997b. Dynamic itemset counting and implication rules for market basket data. In SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pp. 255–264.

Doug Burdick. 2001. Mafia: A maximal frequent itemset algorithm for transactional databases. In ICDE '01: Proceedings of the 17th International Conference on Data Engineering, p. 443.

David W. Cheung, Jiawei Han, Vincent T. Ng, Ada W. Fu, and Yongjian Fu. 1996. A fast distributed algorithm for mining association rules. In DIS '96: Proceedings of the fourth international conference on on Parallel and distributed information systems, pp. 31–43.

Yin-Ling Cheung and Ada Wai-Chee Fu. 2002. Fp-tree approach for mining n-most interesting itemsets. Data Mining and Knowledge Discovery: Theory, Tools, and Technology IV 4730,1:460–471.

Yin-Ling Cheung and Ada Wai-Chee Fu. 2004. Mining frequent itemsets without support threshold: With and without item constraints. IEEE Transactions on Knowledge and Data Engineering 16,9:1052–1069.

Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. 2001. Finding interesting associations without support pruning. IEEE Transactions on Knowledge and Data Engineering 13,1:64–78.

Guozhu Dong and Jinyan Li. 1999. Efficient mining of emerging patterns: discovering trends and differences. In KDD '99: ACM SIGKDD international conference on Knowledge discovery and data mining, San Diego, California, United States, pp. 43–52.

Jie Dong and Min Han. 2007. Bittablefi: An efficient mining frequent itemsets algorithm. Knowledge-Based Systems 20,4:329–335.

Mohammad El-hajj and Osmar R. Zaiane. 2003. Cofi-tree mining: A new approach to pattern growth with reduced candidacy generation. In Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM.

Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. 2005. Periodicity detection in time series databases. IEEE Trans. on Knowl. and Data Eng. 17,7:875–887.

Joseph Engler. 2008. Mining periodic patterns in manufacturing test data. In International Conference IEEE SoutheastCon, pp. 389–395.

Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. Knowledge discovery and data mining: Towards a unifying framework. In KDD, pp. 82–88.

Frdric Flouvat, Fabien De Marchi, and Jean-Marc Petit. 2010. A new classification of datasets for frequent itemsets. Journal of Intelligent Information Systems 34:1–19.

Ada Wai-Chee Fu, Renfrew W. w. Kwong, and Jian Tang. 2000. Mining n-most interesting itemsets. In ISMIS '00: Proceedings of the 12th International Symposium on Foundations of Intelligent Systems, pp. 59–67. Springer-Verlag.

Liquang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. ACM Comput. Surv. 38,3:9.

Bart Goethals. 2005. Frequent set mining. In The Data Mining and Knowledge Discovery Handbook, pp. 377–397. Springer.

Karam Gouda and Mohammed J. Zaki. 2001. Efficiently mining maximal frequent itemsets. Data Mining, IEEE International Conference on 0:163.

Gösta Grahne and Jianfei Zhu. 2005. Fast algorithms for frequent itemset mining using fp-trees. IEEE Transactions on Knowledge and Data Engineering 17,10:1347–1362.

Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. 2002. Mining top-k frequent closed patterns without minimum support. In IEEE International Conference on Data Mining, pp. 211–218.

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. 8,1: 53–87.

Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15,1:55–86.

Robert J. Hilderman and Howard J. Hamilton. 2000. Applying objective interestingness measures in data mining systems. In Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, volume 1910 of Lecture Notes in Computer Science, pp. 432–439. Springer.

John D. Holt and Soon M. Chung. 2002. Mining association rules using inverted hashing and pruning. Information Processing Letters 83,4:211–220.

Tianming Hu, Sam Yuan Sung, Hui Xiong, and Qian Fu. 2008. Discovery of maximum length frequent itemsets. Inf. Sci. 178,1:69–87.

Yun Sing Koh. 2008. Mining non-coincidental rules without a user defined support threshold. In Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, volume 5012 of Lecture Notes in Computer Science, pp. 910–915. Springer.

Srivatsan. Laxman and P.S. Sastry. 2006. A survey of temporal data mining. In Sādhanā, volume 31, Part 2, pp. 173–198.

Yannick Le Bras, Philippe Lenca, and Stéphane Lallich. 2009. On optimal rule mining: A framework and a necessary and sufficient condition of antimonotonicity. In Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, volume 5476 of Lecture Notes in Computer Science, pp. 705–712. Springer.

Yannick Le Bras, Philippe Lenca, and Stéphane Lallich. 2010. Mining interesting rules without support requirement: A general universal existential upward closure property. Information Systems .

Guanling Lee, Wenpo Yang, and Jia-Min Lee. 2006. A parallel algorithm for mining multiple partial periodic patterns. Information Sciences 176,24:3591–3609.

Philippe Lenca, Patrick Meyer, Benoît Vaillant, and Stéphane Lallich. 2008. On selecting interestingness measures for association rules: User oriented description and multiple criteria decision aid. European Journal of Operational Research 184,2:610–626.

Hua-Fu Li. 2009a. Mining top-k maximal reference sequences from streaming web click-sequences with a damped sliding window. Expert Systems with Applications 36,8: 11304–11311.

Hua-Fu Li. 2009b. Interactive mining of top-k frequent closed itemsets from data streams. Expert Systems with Applications 36,7:10779–10788.

Jinyan Li, Xiuzhen Zhang, Guozhu Dong, Kotagiri Ramamohanarao, and Qun Sun. 1999. Efficient mining of high confidence association rules without support thresholds. In Jan M. Zytkow and Jan Rauch, editors, Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD '99, Prague, Czech Republic, September 15-18, volume 1704 of Lecture Notes in Computer Science, pp. 406–411. Springer.

Jiuyong Li. 2006. On optimal rule discovery. IEEE Transactions on Knowledge and Data Engineering 18,4:460–471.

Wenmin Li, Jiawei Han, and Jian Pei. 2001. Cmar: Accurate and efficient classification based on multiple class-association rules. Data Mining, IEEE International Conference on 0:369.

Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating classification and association rule mining. In 4th International Conference on Knowledge Discovery and Data Mining, pp. 80–86.

Guimei Liu, Hongjun Lu, and Jeffrey Xu Yu. 2007. Cfp-tree: A compact disk-based structure for storing and querying frequent itemsets. Information Systems 32,2:295–319.

Fahad Maqbool, Shariq Bashir, and A. Rauf Baig. 2006. E-map: Efficiently mining asynchronous periodic patterns. IJCSNS International Journal of Computer Science and Network Security 6,8:174–179.

Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In Thomas Eiter and Leonid Libkin, editors, Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings, volume 3363 of Lecture Notes in Computer Science, pp. 398–412. Springer.

Banu Özden, Sridhar Ramaswamy, and Abraham Silberschatz. 1998. Cyclic association rules. In ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering, pp. 412–421.

Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. 1995. An effective hash-based algorithm for mining association rules. SIGMOD Rec. 24,2:175–186.

Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. 1999. Discovering frequent closed itemsets for association rules. In Database Theory - ICDT, 7th International Conference, Jerusalem, Israel, January 10-12, volume 1540, pp. 398–416.

Jian Pei and Jiawei Han. 2002. Constrained frequent pattern mining: a pattern-growth view. SIGKDD Explor. Newsl. 4,1:31–39.

Jian Pei, Jiawei Han, and Runying Mao. 2000. Closet: An efficient algorithm for mining frequent closed itemsets. In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21–30.

Jian Pei, Jiawei Han, and Laks V. S. Lakshmanan. 2001a. Mining frequent item sets with convertible constraints. In Proceedings of the 17th International Conference on Data Engineering, April 2-6, Heidelberg, Germany, pp. 433–442.

Jian Pei, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongquing Yang. 2001b. H-mine: Hyper-structure mining of frequent patterns in large databases. Data Mining, IEEE International Conference on 0:441.

Andrea Pietracaprina and Fabio Vandin. 2007. Efficient incremental mining of top-k frequent closed itemsets. In Discovery Science, volume 4755 of Lecture Notes in Computer Science, pp. 275–280. Springer.

Pradeep Shenoy, Jayant R. Haritsa, S. Sudarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. 2000. Turbo-charging vertical mining of large databases. SIGMOD Rec. 29,2: 22–33.

Craig Silverstein, Sergey Brin, Rajeev Motwani, and Jeff Ullman. 2000. Scalable techniques for mining causal structures. Data Mining and Knowledge Discovery 4,2-3:163–192.

Einoshin Suzuki. 2008. Pitfalls for categorizations of objective interestingness measures for rule discovery. In Statistical Implicative Analysis, Theory and Applications, volume 127, pp. 383–395. Springer.

Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. 2009. Discovering periodic-frequent patterns in transactional databases. In Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, volume 5476 of Lecture Notes in Computer Science, pp. 242–253. Springer.

Feng Tao, Fionn Murtagh, and Mohsen Farid. 2003. Weighted association rule mining using weighted support and significance framework. In KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 661–666.

Giridhar Tatavarty, Raj Bhatnagar, and Barrington Young. 2007. Discovery of temporal dependencies between frequent patterns in multivariate time series. In Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2007, part of the IEEE Symposium Series on Computational Intelligence 2007, Honolulu, Hawaii, USA, 1-5 April 2007, pp. 688–696. IEEE.

Pauray S.M. Tsai. 2010. Mining top-k frequent closed itemsets over data streams using the sliding window model. Expert Systems with Applications In Press, Corrected Proof:–.

Yuh-Jiuan Tsay and Ya-Wen Chang-Chien. 2004. An efficient cluster and decomposition algorithm for mining association rules. Information Sciences 160,1-4:161–171.

Yuh-Jiuan Tsay and Jiunn-Yann Chiang. 2005. Cbar: an efficient method for mining association rules. Knowledge-Based Systems 18,2-3:99–105.

Petre Tzvetkov, Xifeng Yan, and Jiawei Han1. 2005. Tsp: Mining top-k closed sequential patterns. Knowledge and Information Systems 7,4:438–457.

Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. 2004a. An efficient algorithm for enumerating closed patterns in transaction databases. In Discovery Science, volume 3245 of Lecture Notes in Computer Science, pp. 16–31. Springer.

Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. 2004b. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations. Brighton, UK, volume 126.

Jianyong Wang, Jiawei Han, Ying Lu, and Petre Tzvetkov. 2005. Tfp: an efficient algorithm for mining top-k frequent closed itemsets. In Proceeding of the IEEE Transactions on Knowledge and Data Engineering, volume 17, pp. 652–664.

Chienwen Wu. 2006. Mining top-k frequent closed itemsets is not in apx. In Advances in Knowledge Discovery and Data Mining, volume 3918 of Lecture Notes in Computer Science, pp. 435–439. Springer.

Xindong Wu, Chengqi Zhang, and Shichao Zhang. 2004. Efficient mining of both positive and negative association rules. ACM Trans. Inf. Syst. 22,3:381–405.

Sadok Ben Yahia, Tarek Hamrouni, and Engelbert Mephu Nguifo. 2006. Frequent closed itemset base algorithms: a thorough structural and analytical survey. SIGKDD Explorations 8 ,1:93–104.

Bei Yang, Houkuan Huang, and Zhifeng Wu. 2008. Topsis: Finding top-k significant n-itemsets in sliding windows adaptively. Knowl.-Based Syst. 21,6:443–449.

Unil Yun and John J. Leggett. 2006. Wip: mining weighted interesting patterns with a strong weight and/or support affinity. In Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA, pp. 623–627.

Mohammed Javeed Zaki. 2004. Mining non-redundant association rules. Data Mining and Knowledge Discovery 9,3:223–248.

Mohammed Javeed Zaki and Karam Gouda. 2003. Fast vertical mining using diffsets. In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, pp. 326–335.

Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. 1997. New algorithms for fast discovery of association rules. In KDD, pp. 283–286.

Shichao Zhang, Jilian Zhang, and Chengqi Zhang. 2007. Edua: An efficient algorithm for dynamic database mining. Information Sciences 177,13:2756–2767.

Biography

Komate Amphawan was born in Chonburi, Thailand, on October, 1981. He received B.Sc., in computer science, from Burapha University, Thailand, in 2002. He received M.Sc., in computer engineering, from Chulalongkorn University, Thailand, in 2005. His master degree have been supervised by Dr. Athasit Surarerks. His doctorate has been also under the supervision of Dr. Athasit Surarerks. During his research at Chulalongkorn University (october 2008 - October 2009), he had a great chance to visit a research institute : Telecom Bretagne, France. His field of interest includes various topics in association rules mining: frequent itemset mining, top-k significant itemset mining, regular-frequent itemset mining, weighted frequent itemset mining, utility itemset mining and parallelizing algorithms for all kind of itemsets.