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 $\{t_1, t_4, t_6, t_7, t_8, t_{10}, t_{11}, t_{12}\}$.

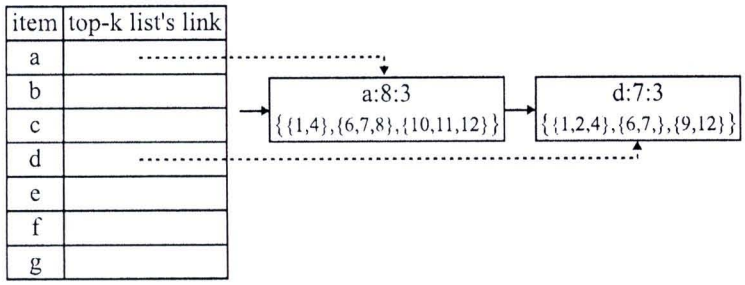

Figure 4.1: TKRIMPE: Top-k list with a hash table

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

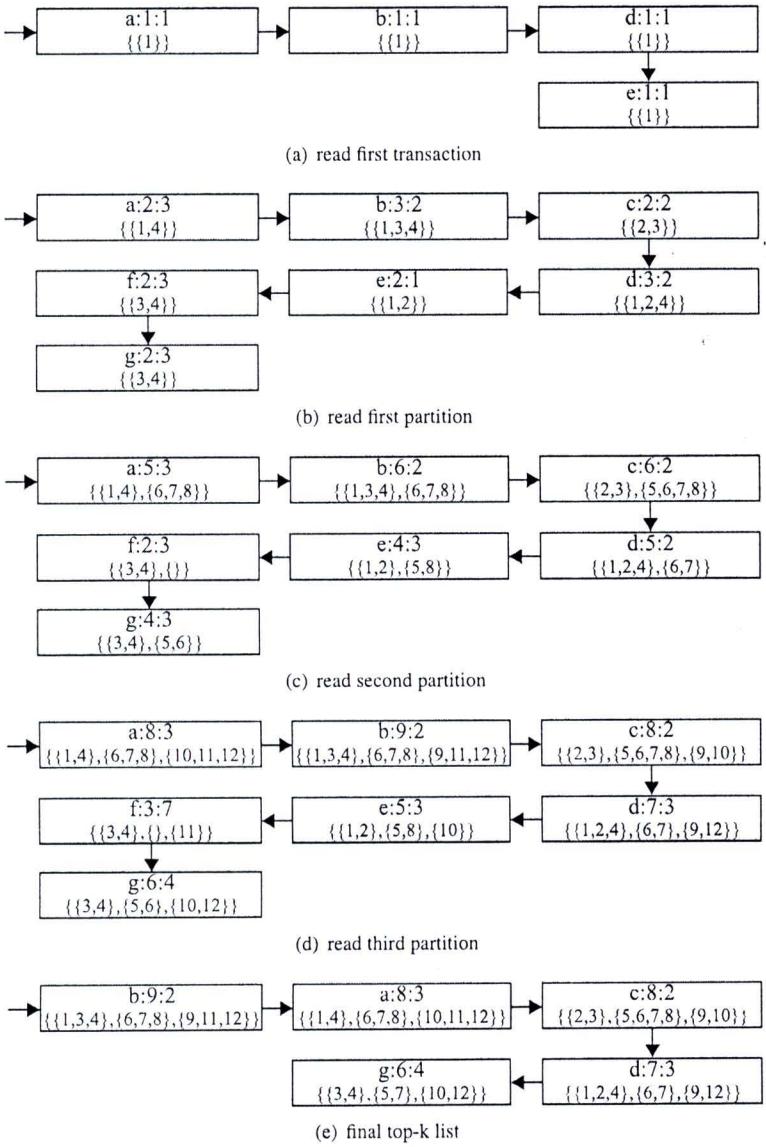

Figure 4.2: Top-k list initialization

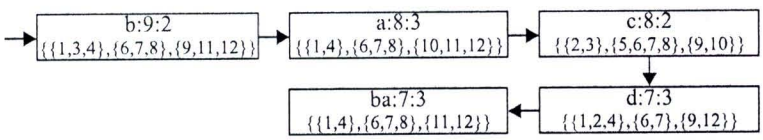

Figure 4.3: Top-k frequent itemsets

$lb_1^b = 1, lb_1^a = 1, rb_1^b = 3$ 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.

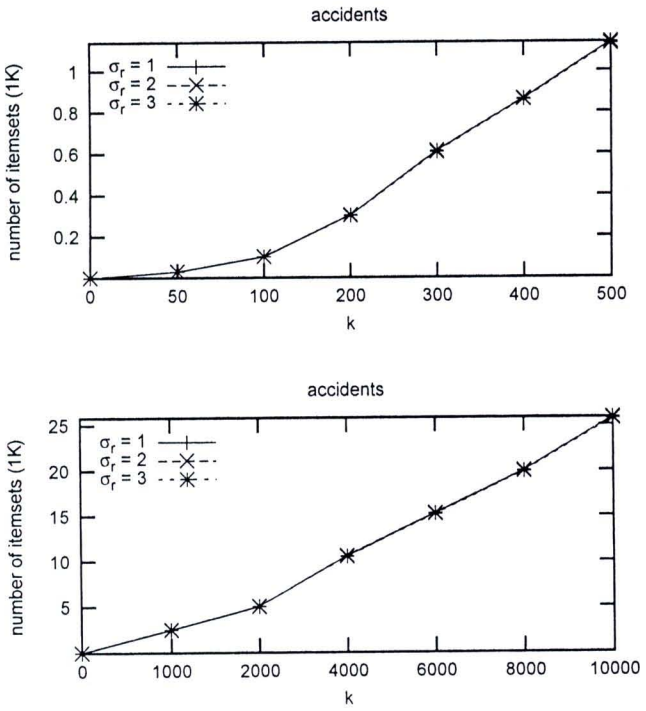

Figure 4.4: The number of early terminated itemsets on *accidents* dataset

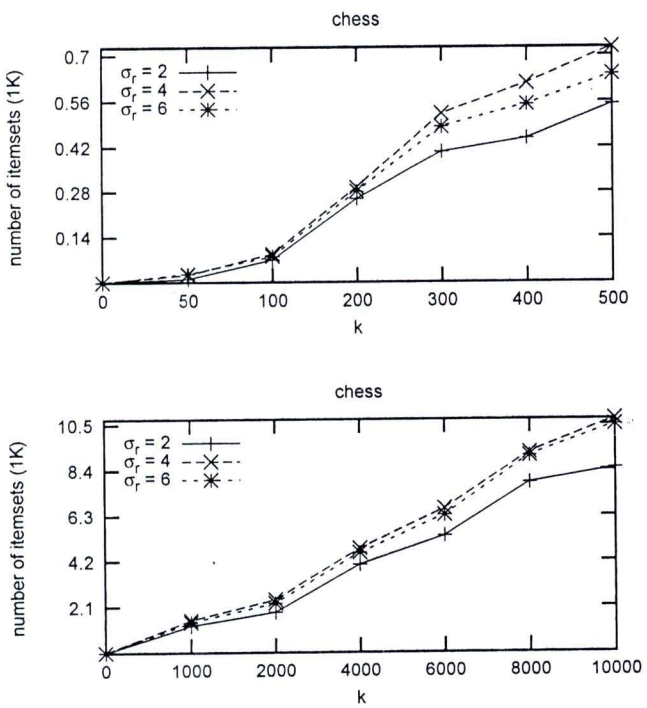

Figure 4.5: The number of early terminated itemsets on *chess* dataset

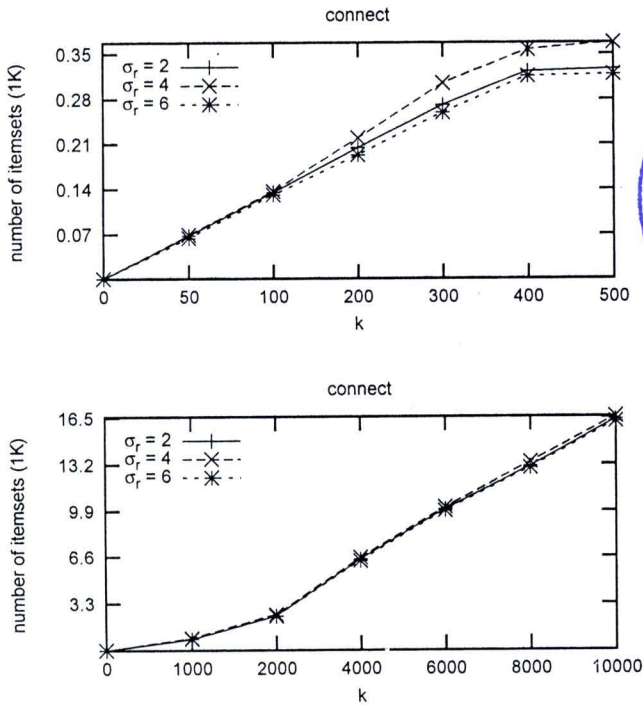

Figure 4.6: The number of early terminated itemsets on *connect* dataset

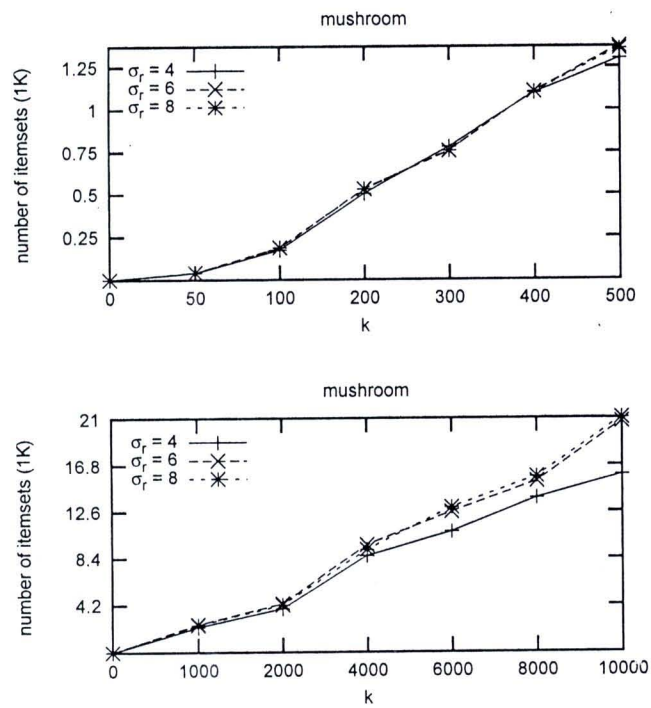

Figure 4.7: The number of early terminated itemsets on *mushroom* dataset

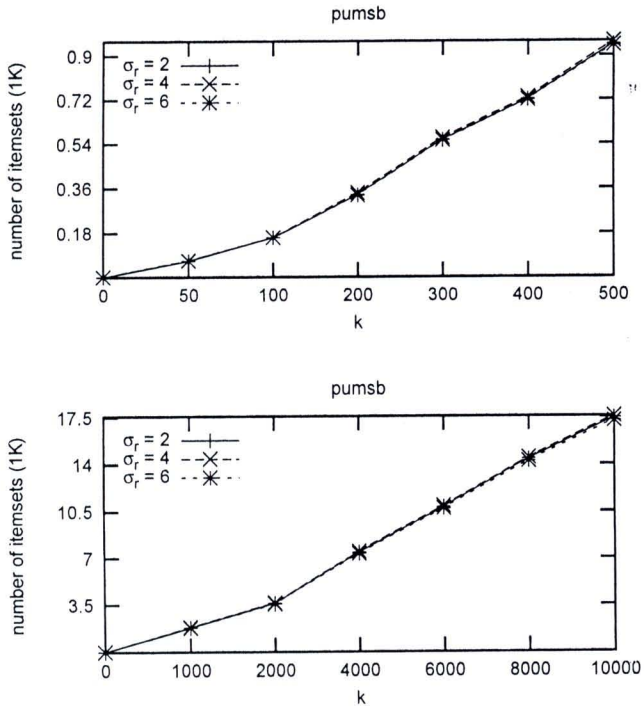

Figure 4.8: The number of early terminated itemsets on *pumsb* dataset

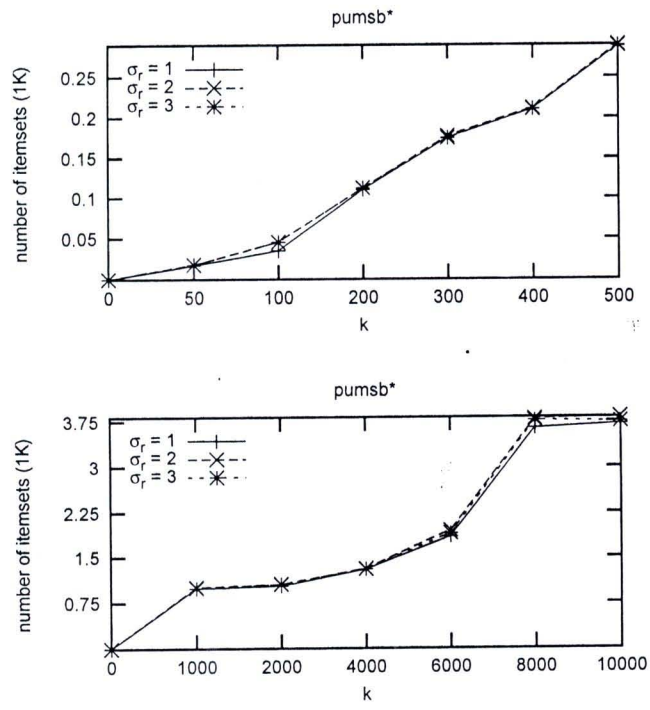

Figure 4.9: The number of early terminated itemsets on *pumsb** dataset

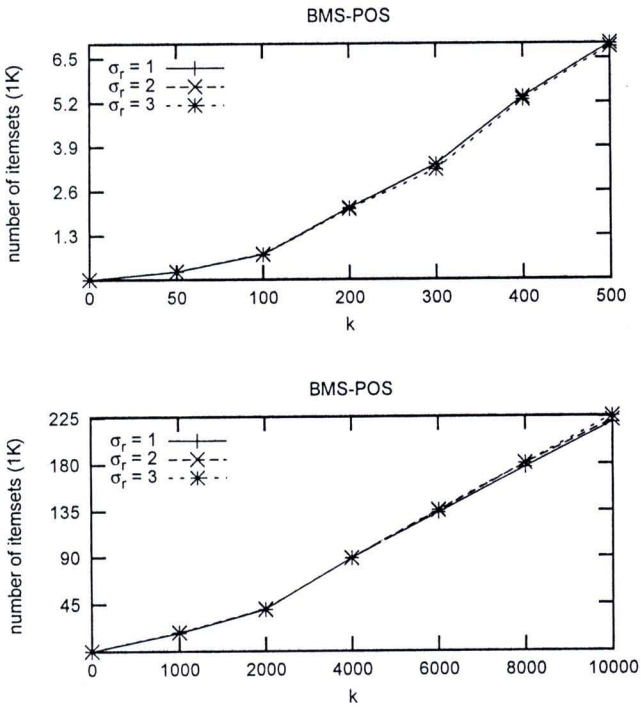

Figure 4.10: The number of early terminated itemsets on *BMS-POS* dataset

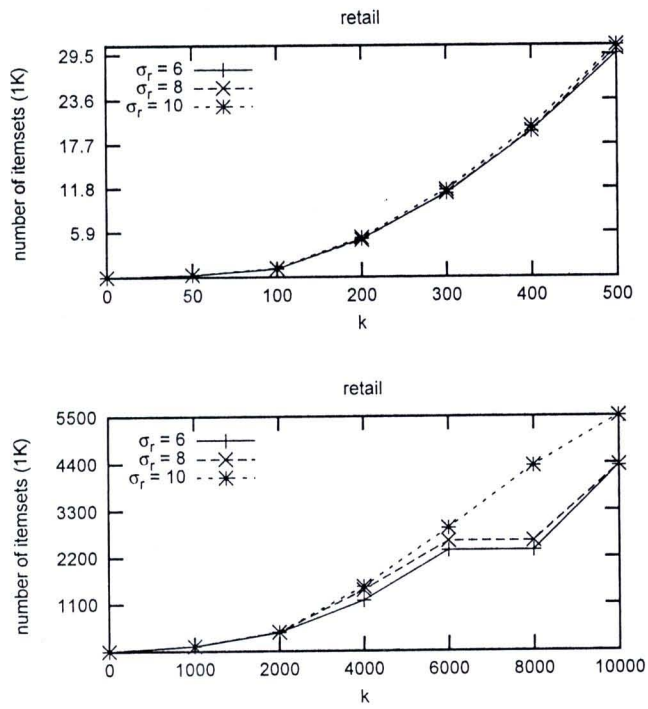

Figure 4.11: The number of early terminated itemsets on *retail* dataset

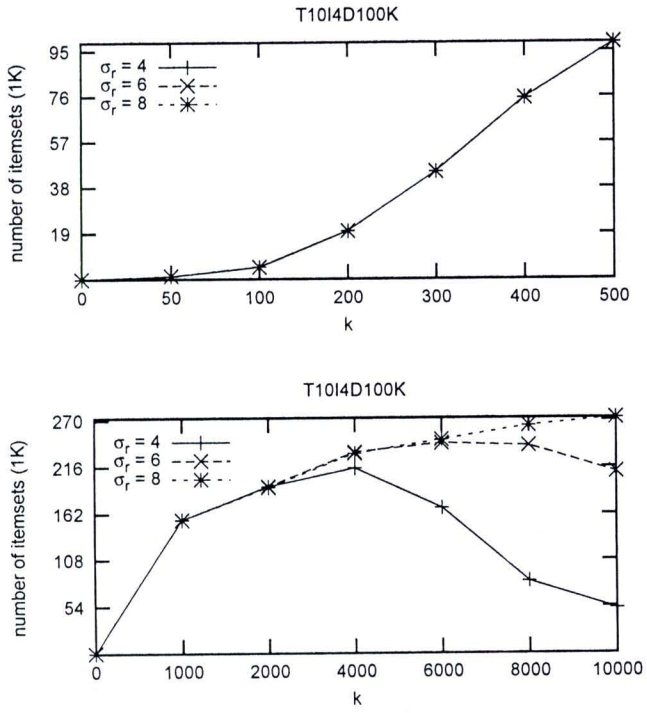

Figure 4.12: The number of early terminated itemsets on *T10I4D100K* dataset

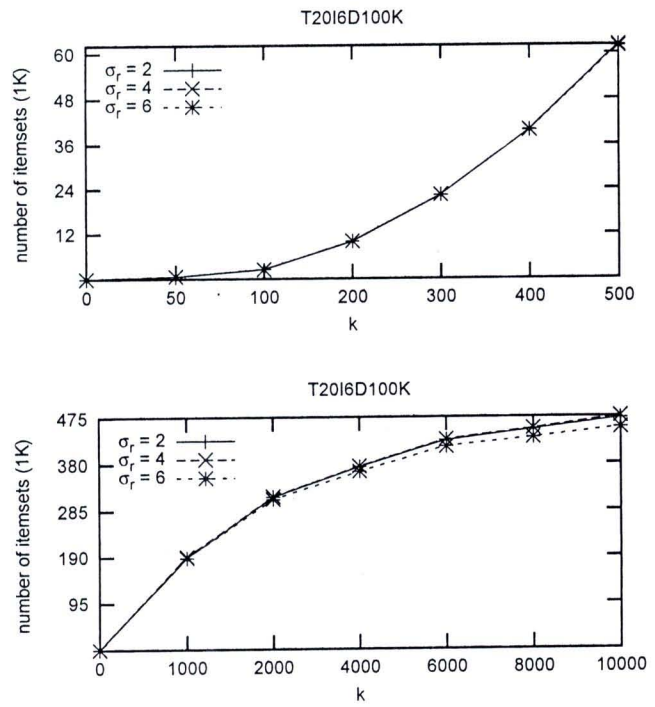

Figure 4.13: The number of early terminated itemsets on *T20I6D100K* dataset

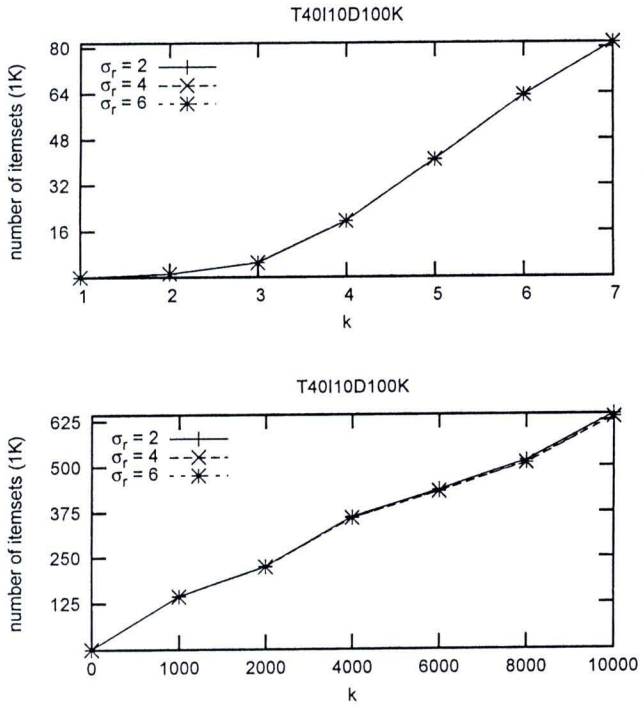

Figure 4.14: The number of early terminated itemsets on *T40I10D100K* dataset

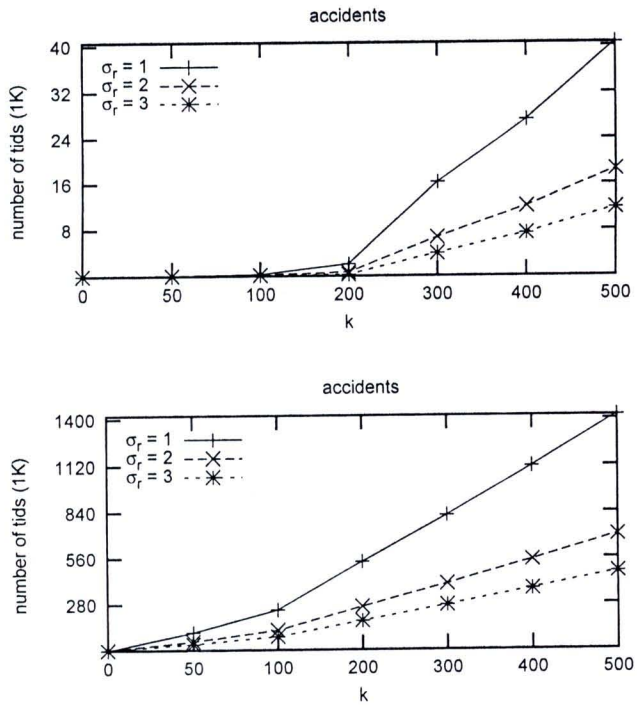

Figure 4.15: The number of non-regarded tids during intersection process on *accidents* dataset

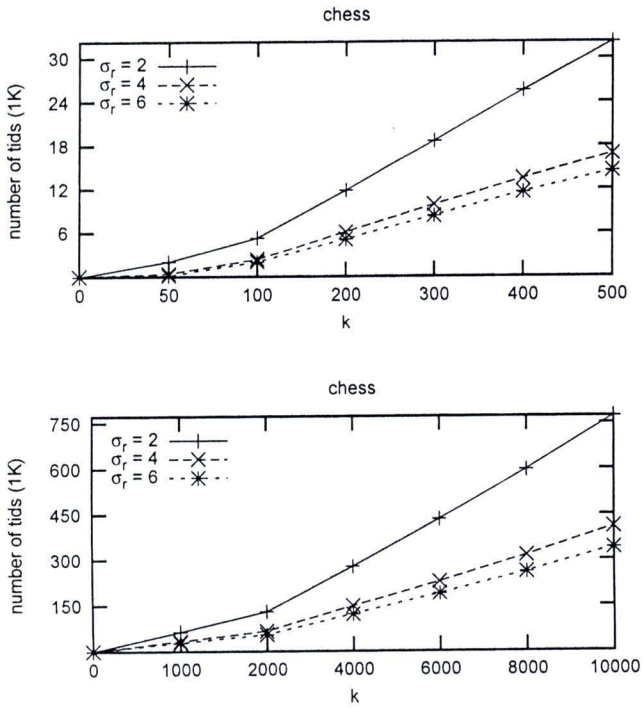

Figure 4.16: The number of non-regarded tids during intersection process on *chess* dataset

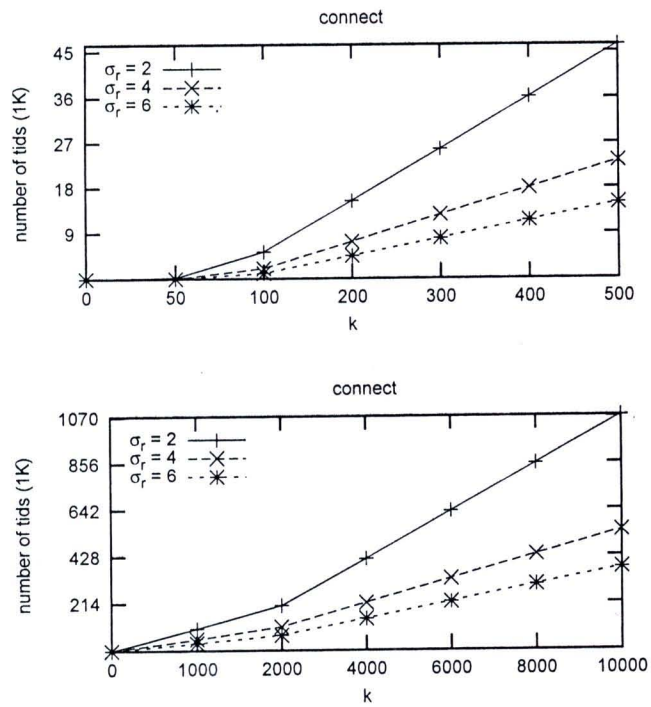

Figure 4.17: The number of non-regarded tids during intersection process on *connect* dataset

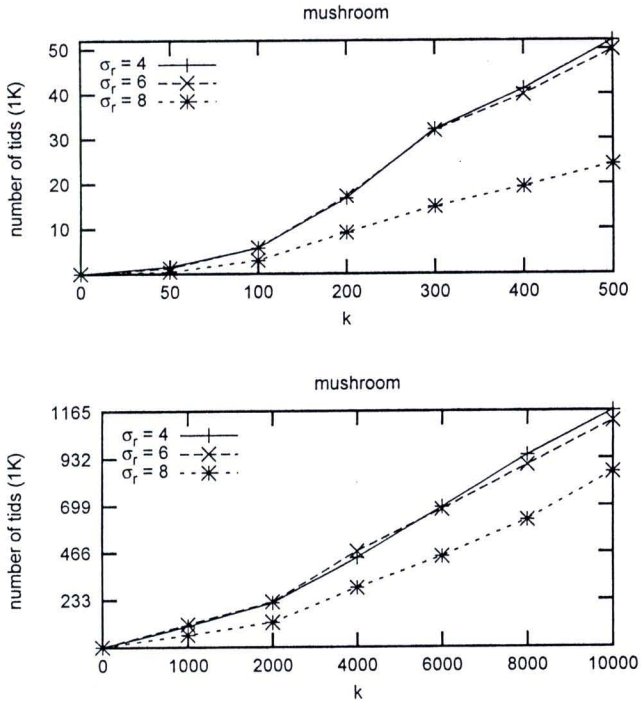

Figure 4.18: The number of non-regarded tids during intersection process on *mushroom* dataset

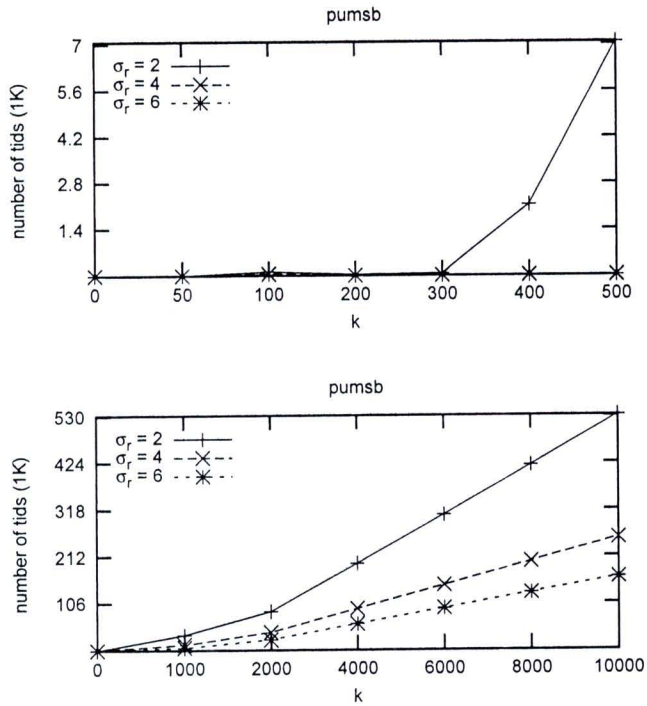

Figure 4.19: The number of non-regarded tids during intersection process on *pumsb* dataset

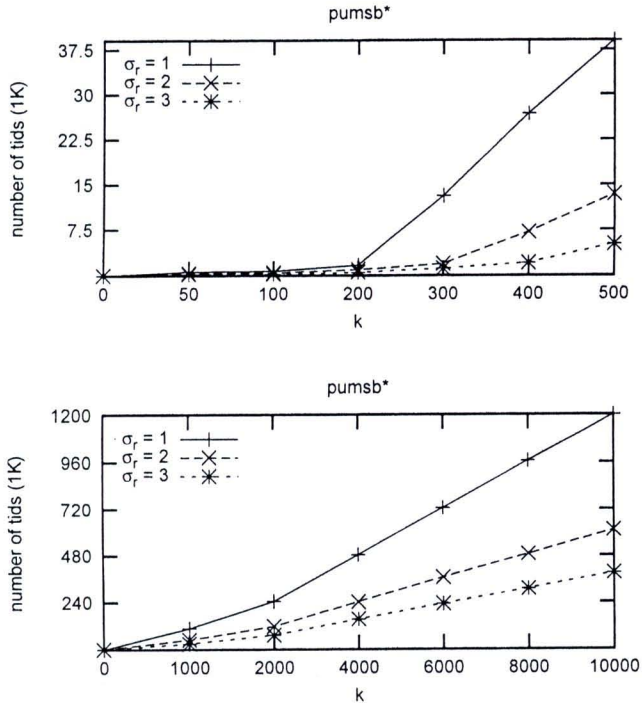

Figure 4.20: The number of non-regarded tids during intersection process on *pumsb** dataset

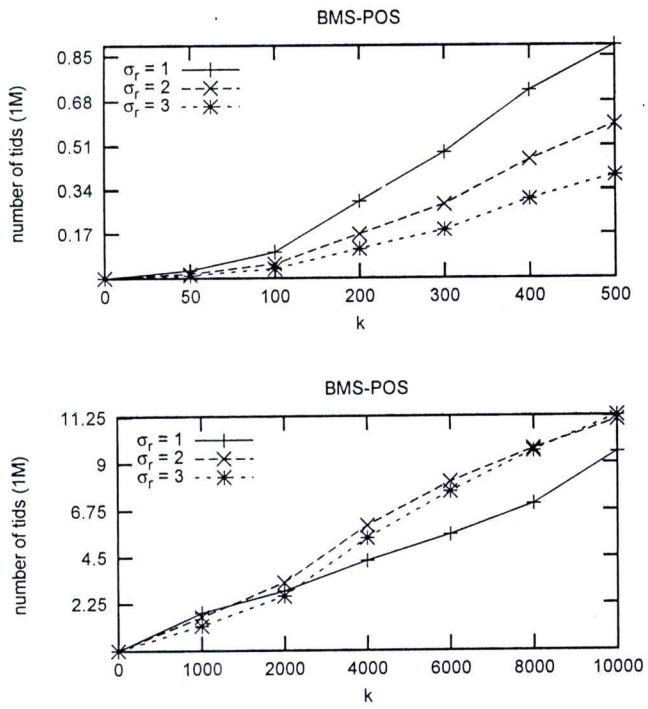

Figure 4.21: The number of non-regarded tids during intersection process on *BMS-POS* dataset

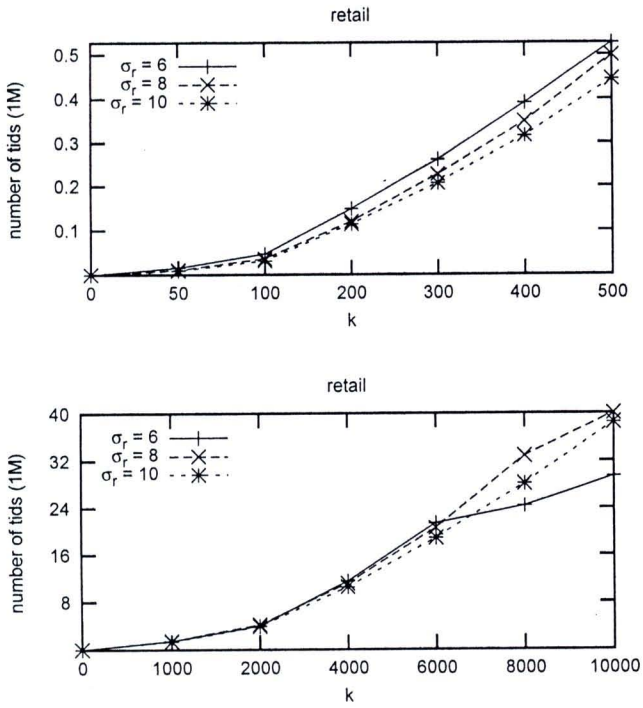

Figure 4.22: The number of non-regarded tids during intersection process on *retail* dataset

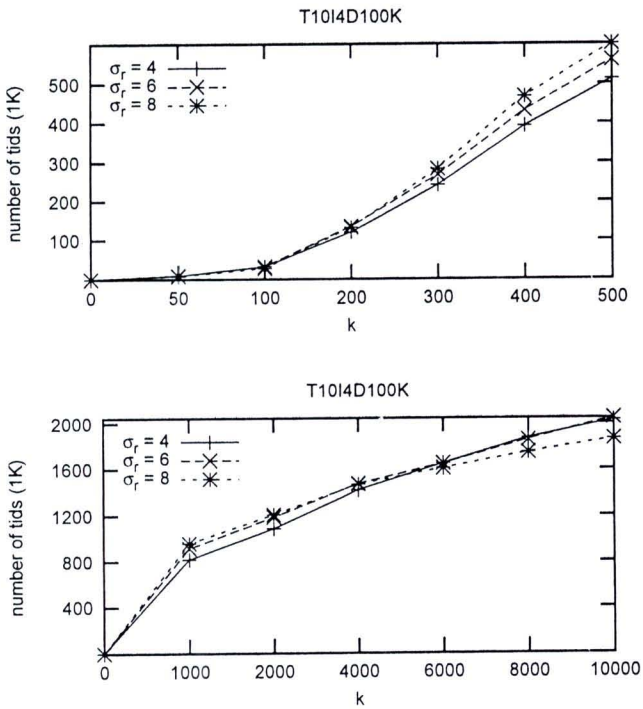

Figure 4.23: The number of non-regarded tids during intersection process on *T10I4D100K* dataset

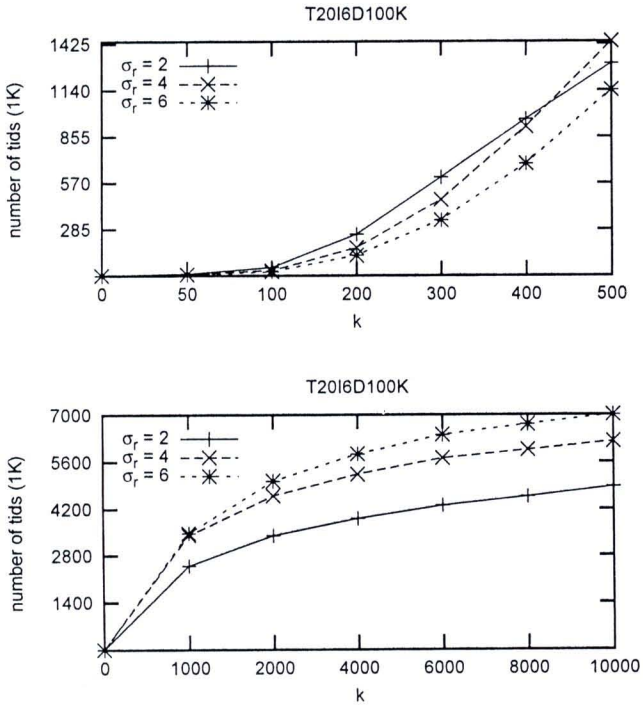

Figure 4.24: The number of non-regarded tids during intersection process on *T20I6D100K* dataset

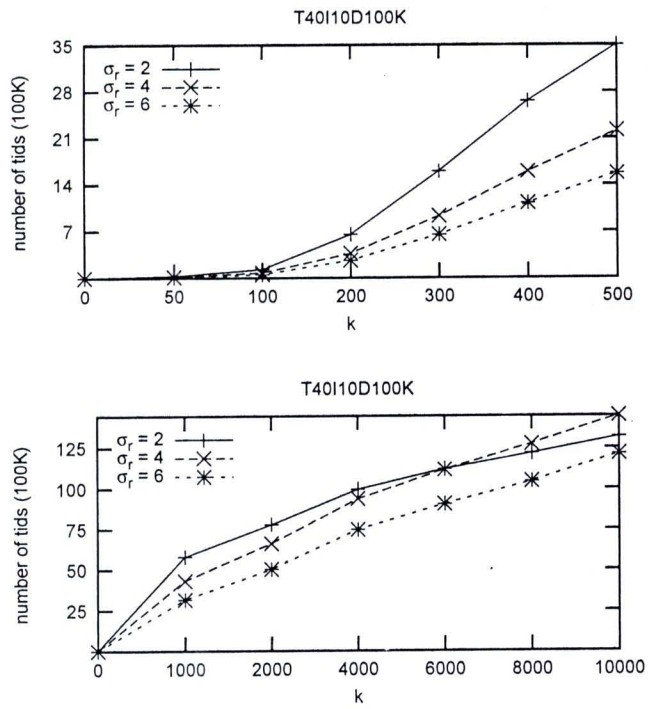

Figure 4.25: The number of non-regarded tids during intersection process on *T40I10D100K* dataset

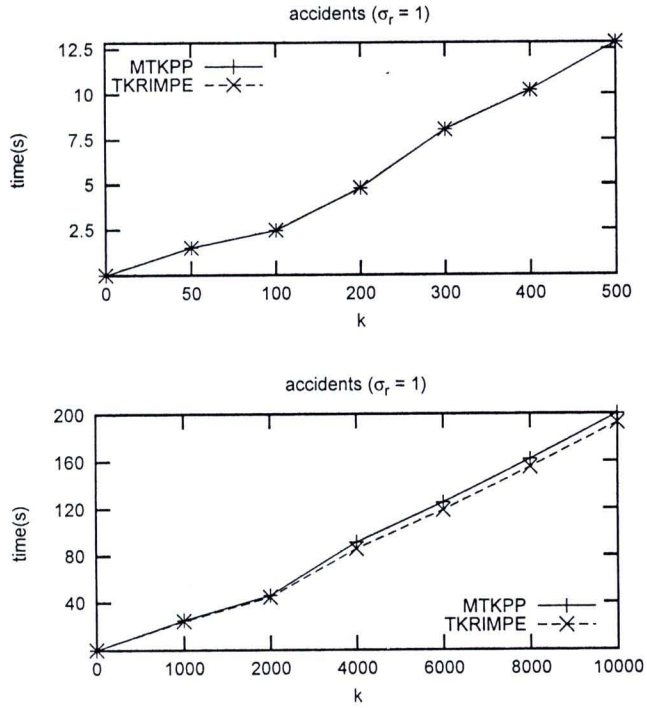

Figure 4.26: Runtime of TKRIMPE on *accidents* ($\sigma_r = 1\%$)

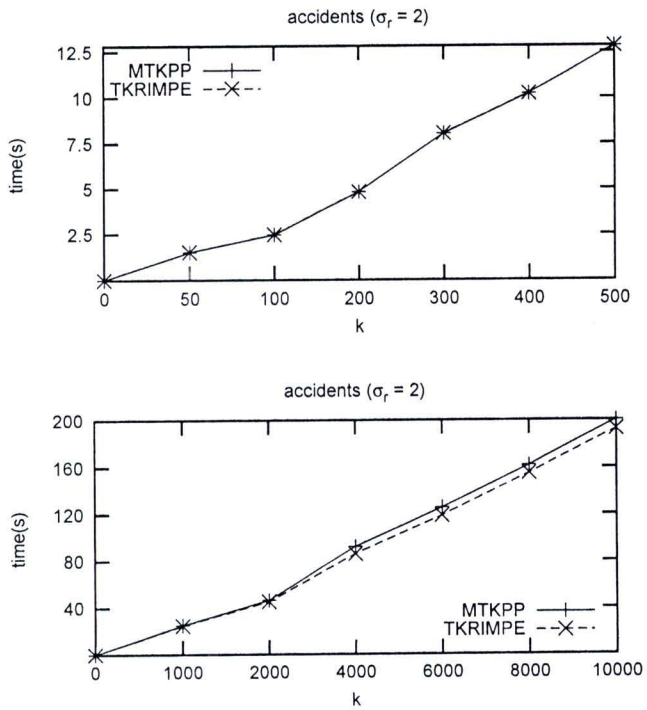

Figure 4.27: Runtime of TKRIMPE on *accidents* ($\sigma_r = 2\%$)

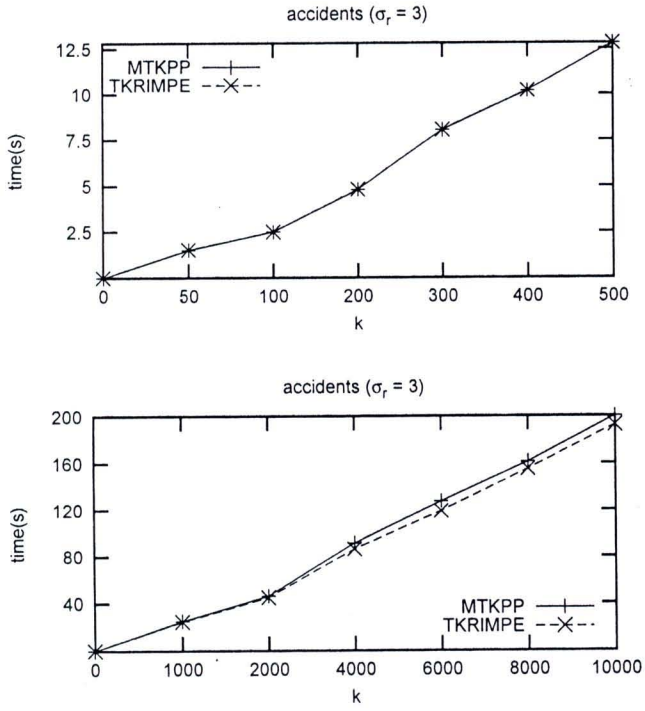

Figure 4.28: Runtime of TKRIMPE on *accidents* ($\sigma_r = 3\%$)

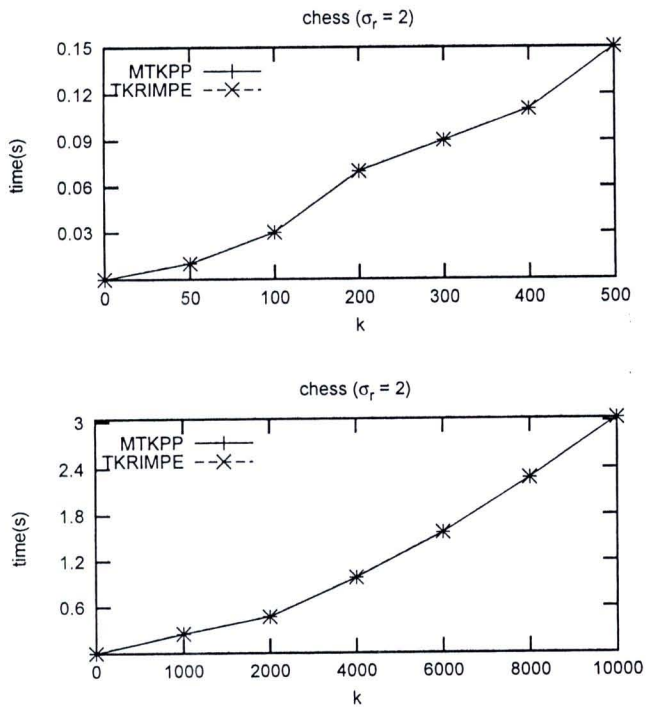

Figure 4.29: Runtime of TKRIMPE on *chess* ($\sigma_r = 2\%$)

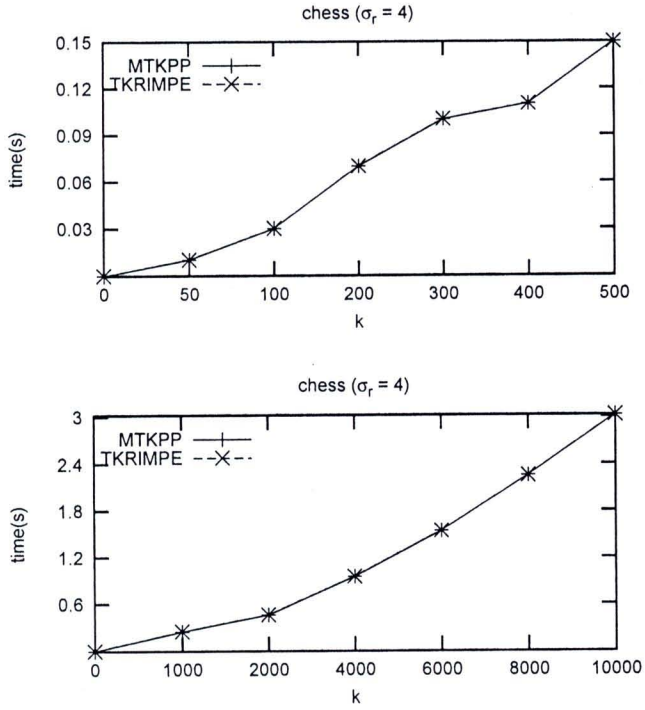

Figure 4.30: Runtime of TKRIMPE on *chess* ($\sigma_r = 4\%$)

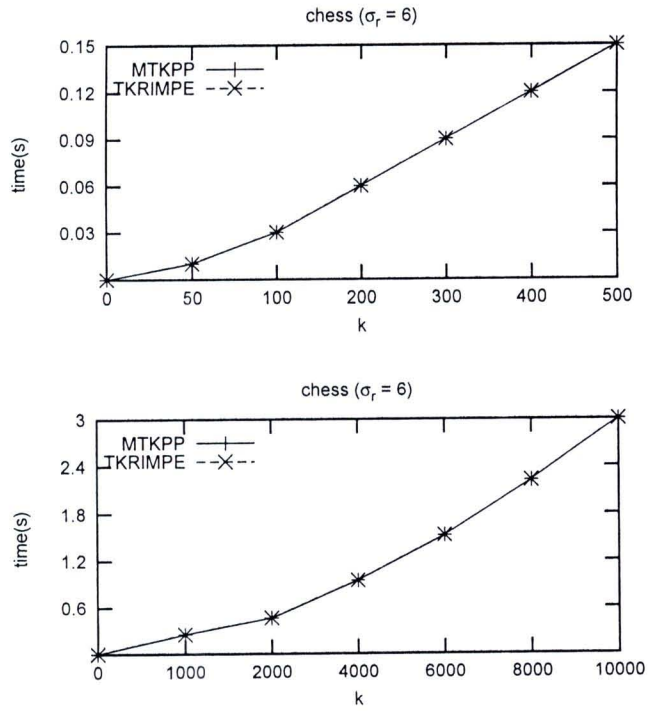

Figure 4.31: Runtime of TKRIMPE on *chess* ($\sigma_r = 6\%$)

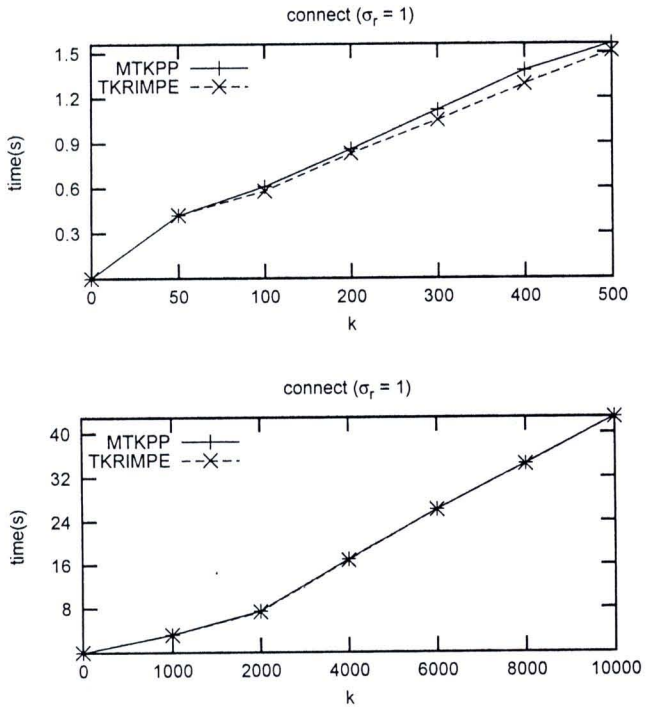

Figure 4.32: Runtime of TKRIMPE on *connect* ($\sigma_r = 1\%$)

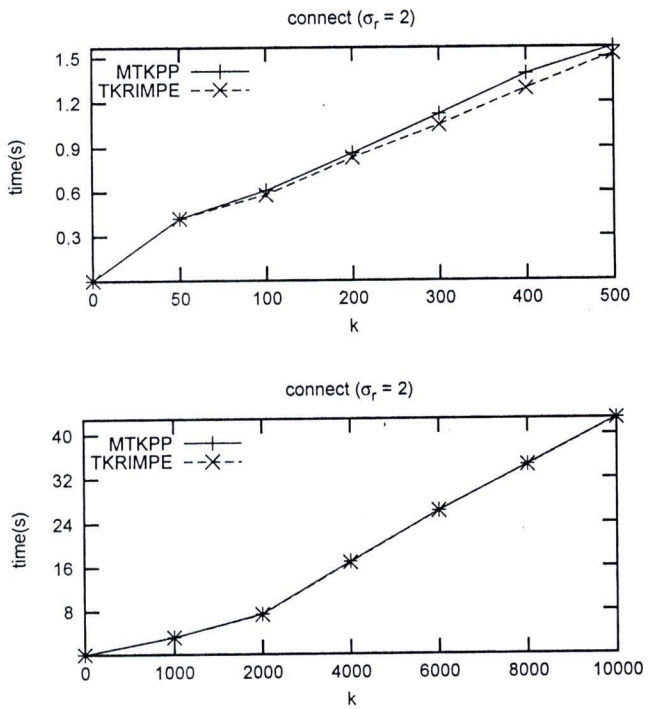

Figure 4.33: Runtime of TKRIMPE on *connect* ($\sigma_r = 2\%$)

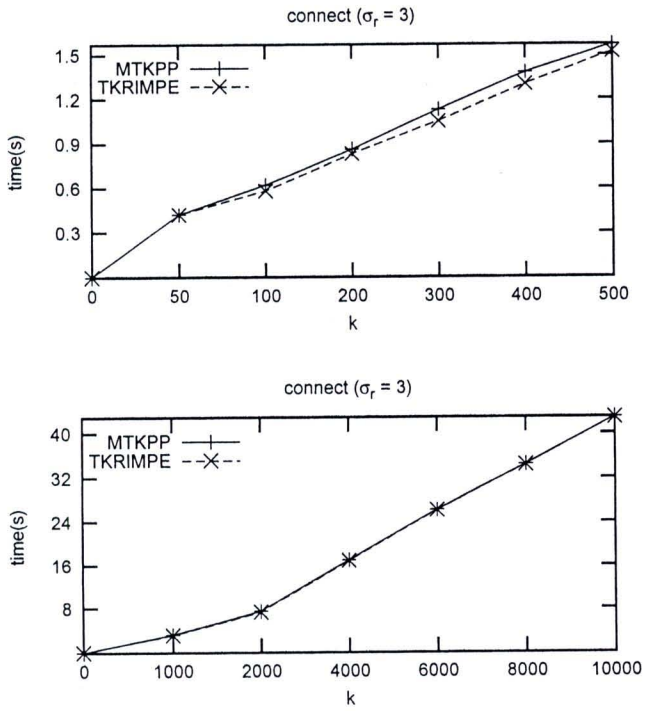

Figure 4.34: Runtime of TKRIMPE on *connect* ($\sigma_r = 3\%$)

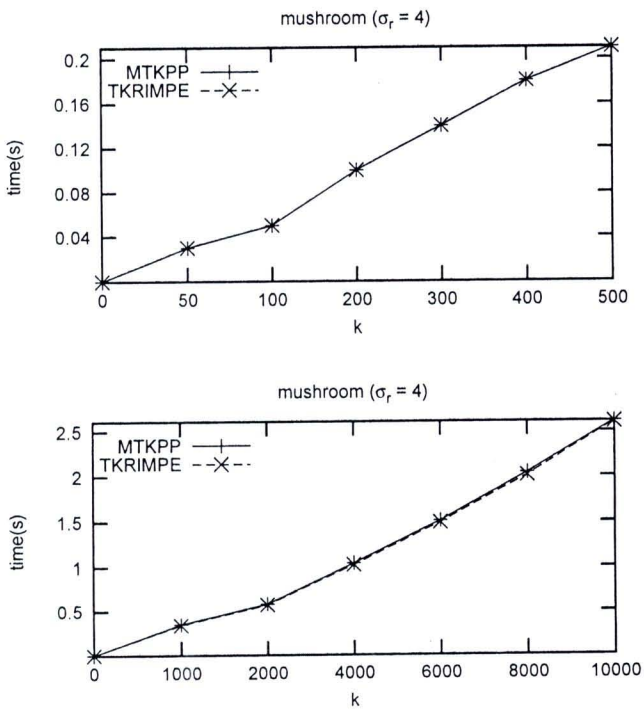

Figure 4.35: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 4\%$)

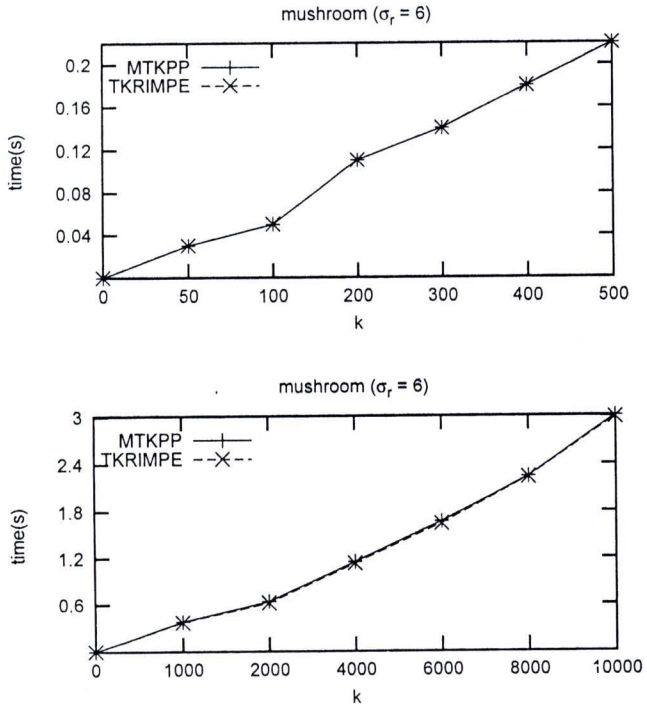

Figure 4.36: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 6\%$)

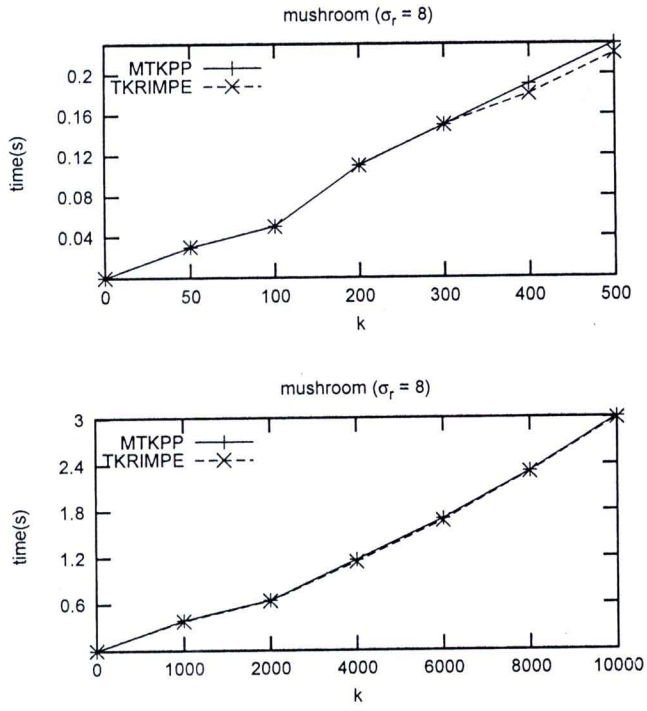

Figure 4.37: Runtime of TKRIMPE on *mushroom* ($\sigma_r = 8\%$)

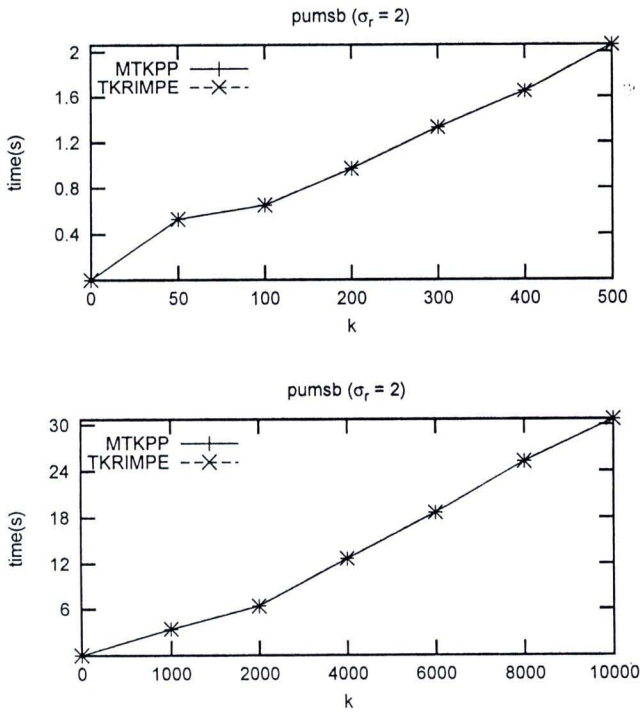

Figure 4.38: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 2\%$)

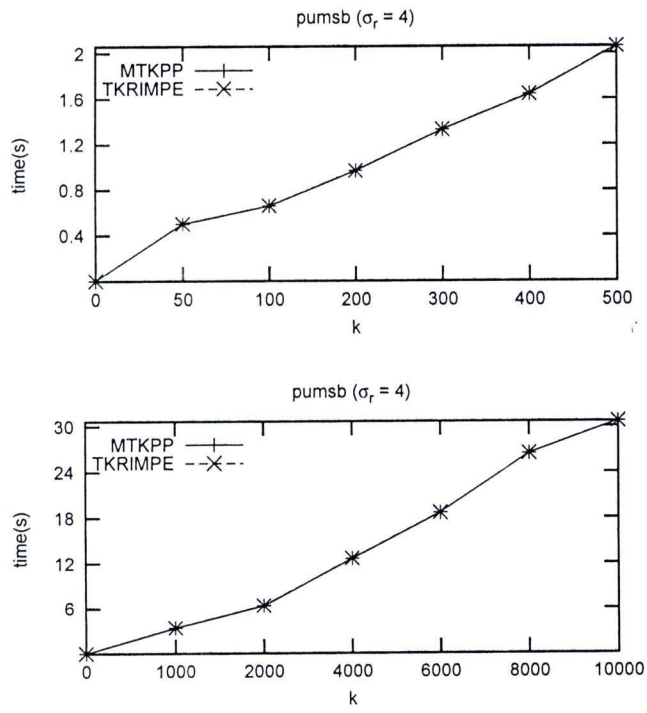

Figure 4.39: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 4\%$)

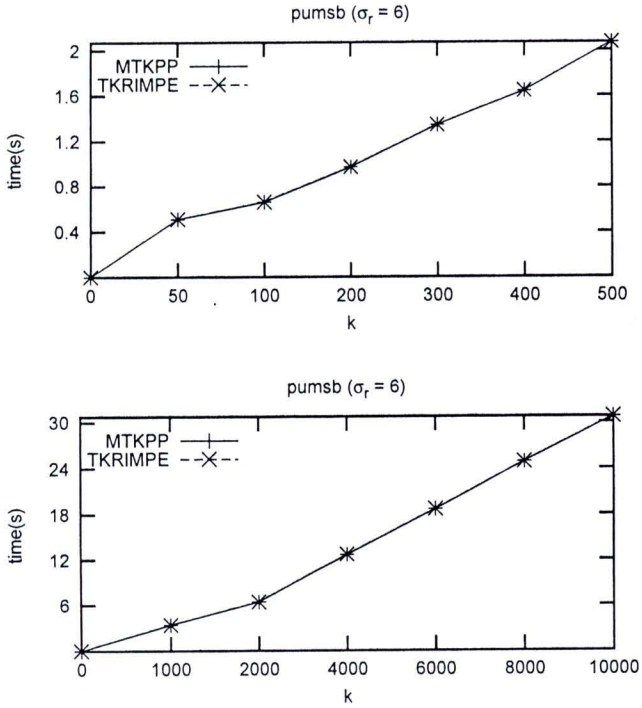

Figure 4.40: Runtime of TKRIMPE on *pumsb* ($\sigma_r = 6\%$)

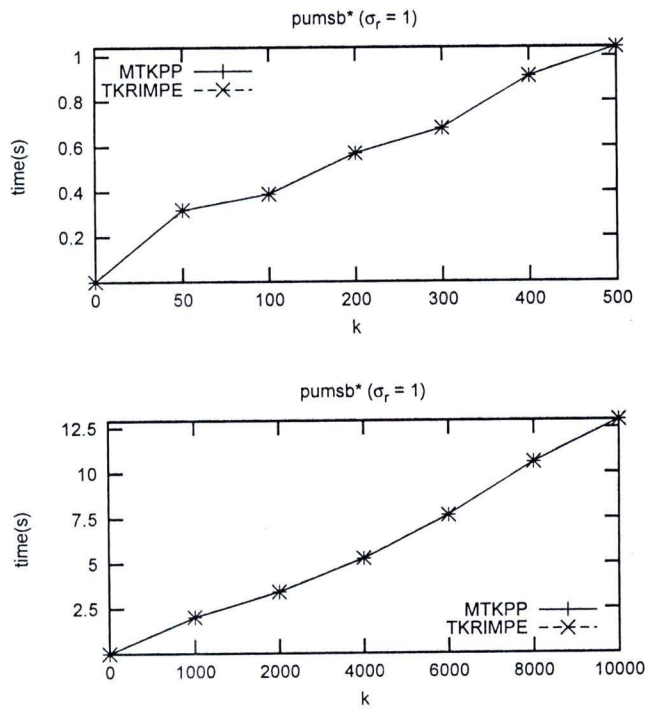

Figure 4.41: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 1\%$)

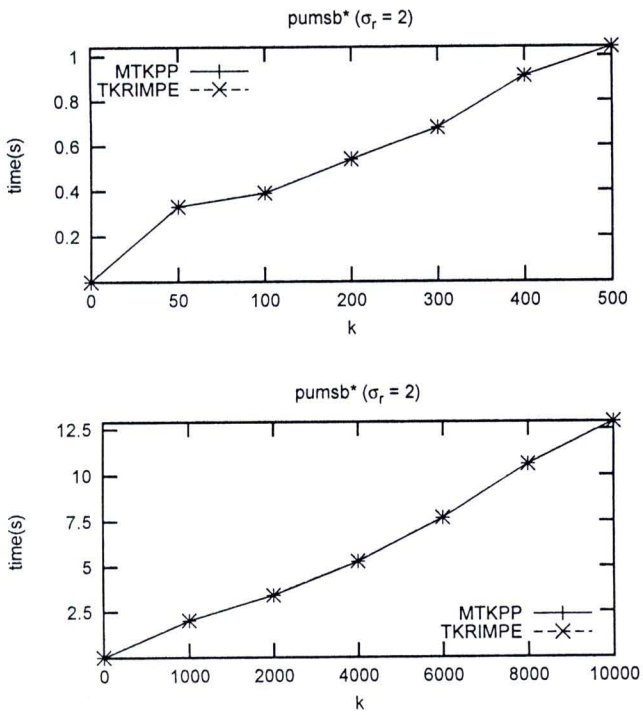

Figure 4.42: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 2\%$)

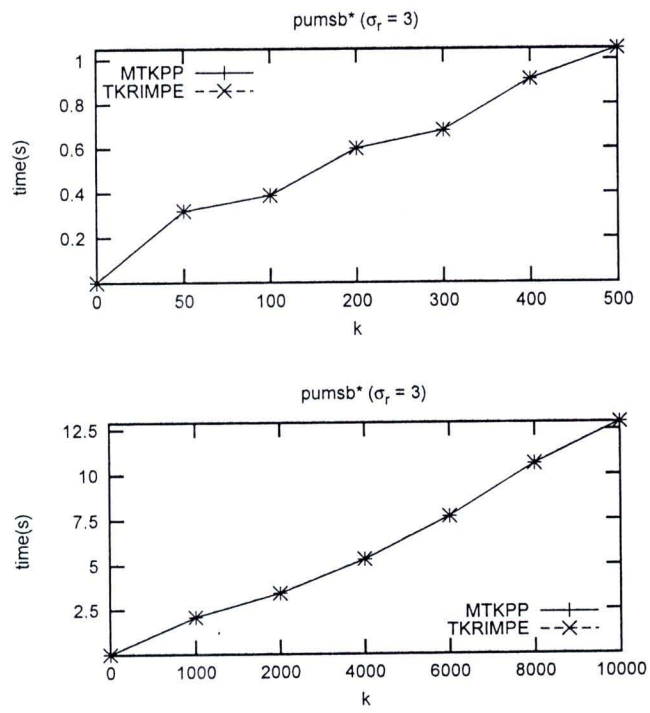

Figure 4.43: Runtime of TKRIMPE on *pumsb** ($\sigma_r = 3\%$)

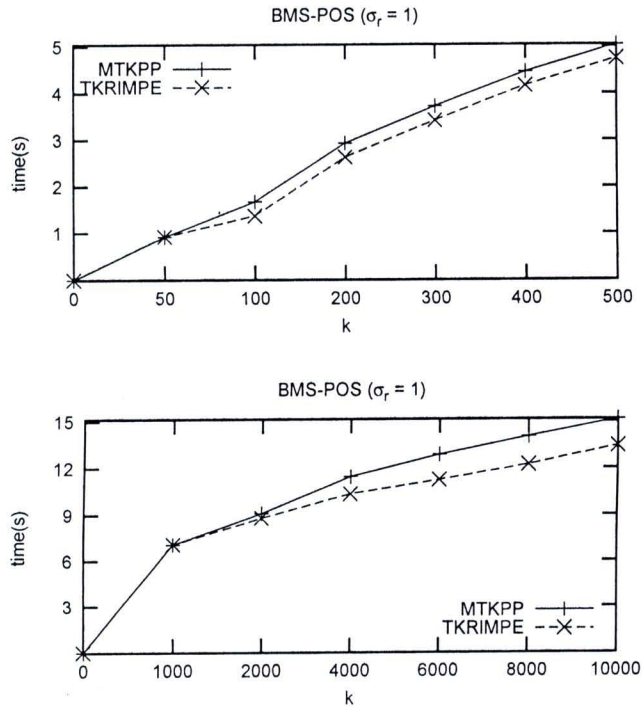

Figure 4.44: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 1\%$)

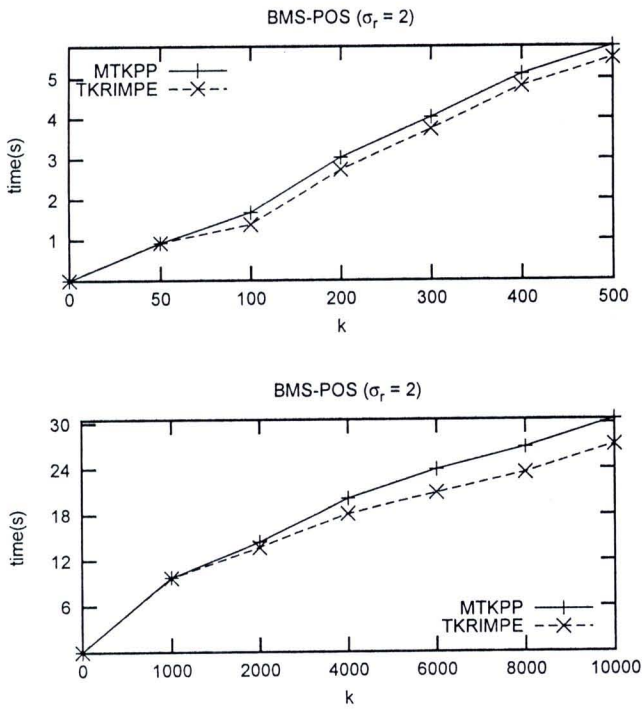

Figure 4.45: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 2\%$)

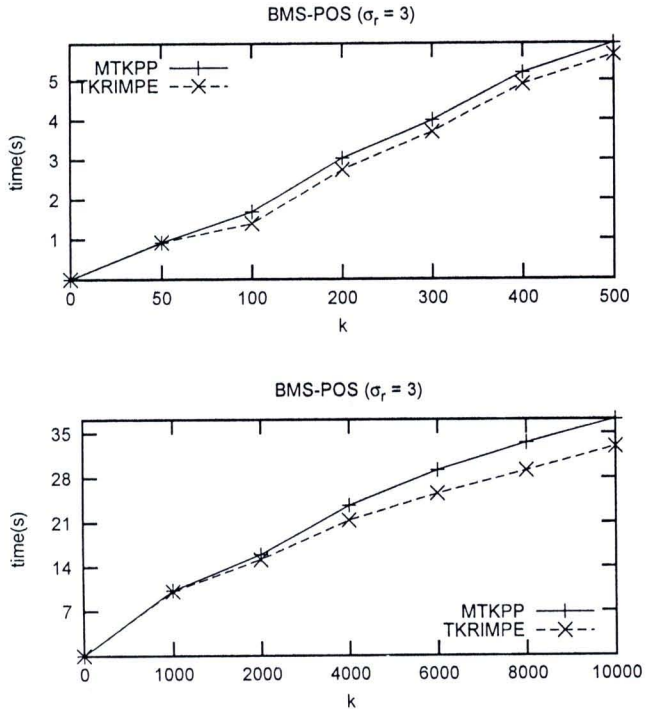

Figure 4.46: Runtime of TKRIMPE on *BMS-POS* ($\sigma_r = 3\%$)

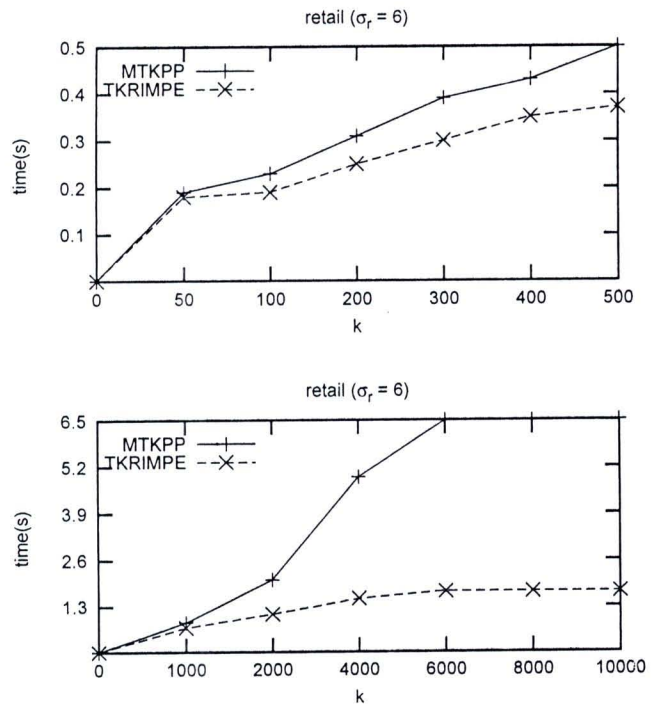

Figure 4.47: Runtime of TKRIMPE on *retail* ($\sigma_r = 6\%$)

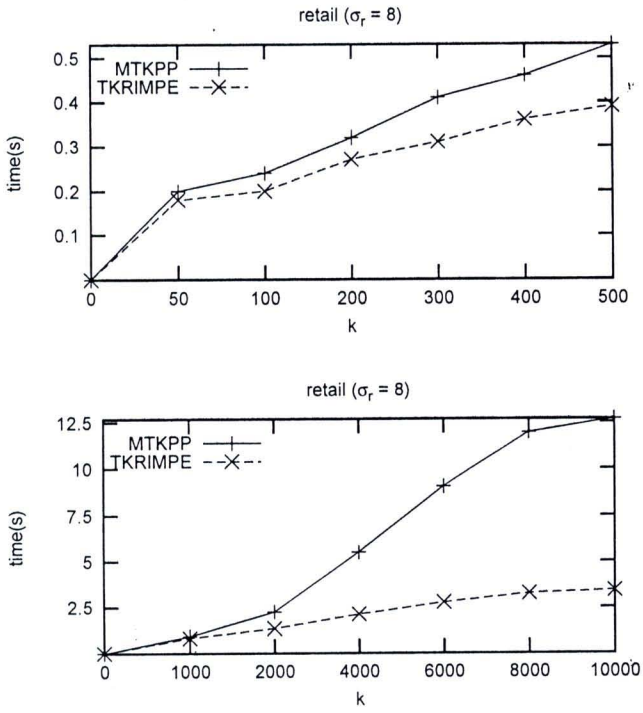

Figure 4.48: Runtime of TKRIMPE on *retail* ($\sigma_r = 8\%$)

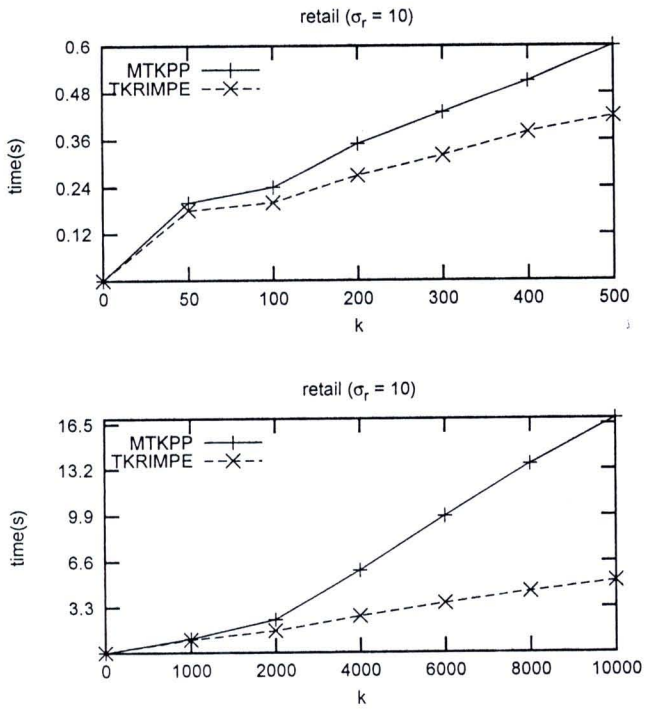

Figure 4.49: Runtime of TKRIMPE on *retail* ($\sigma_r = 10\%$)

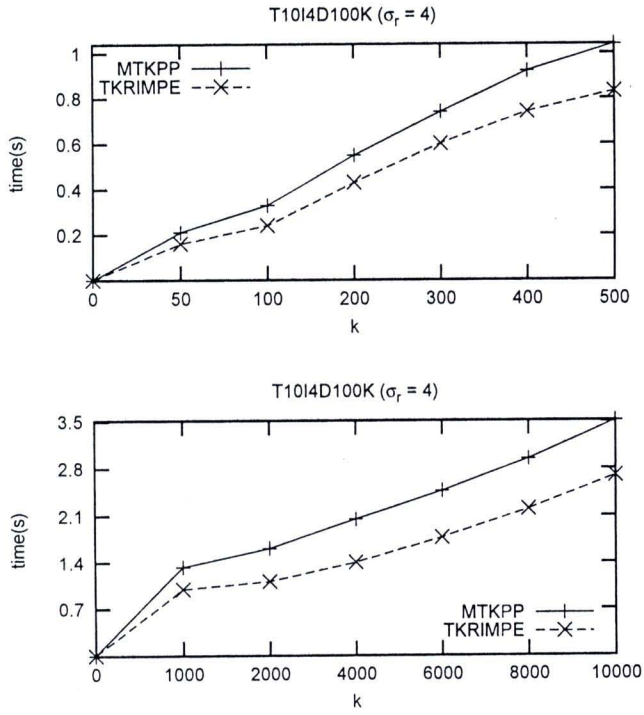

Figure 4.50: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 4\%$)

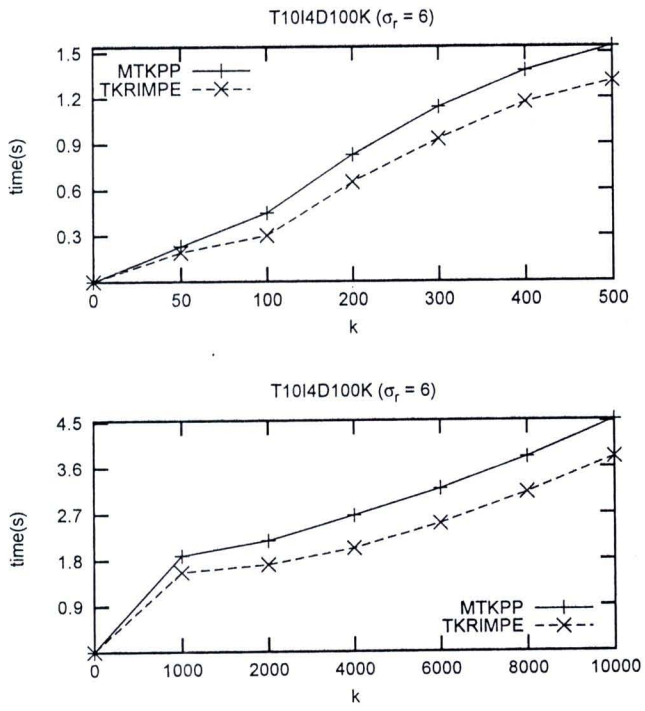

Figure 4.51: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 6\%$)

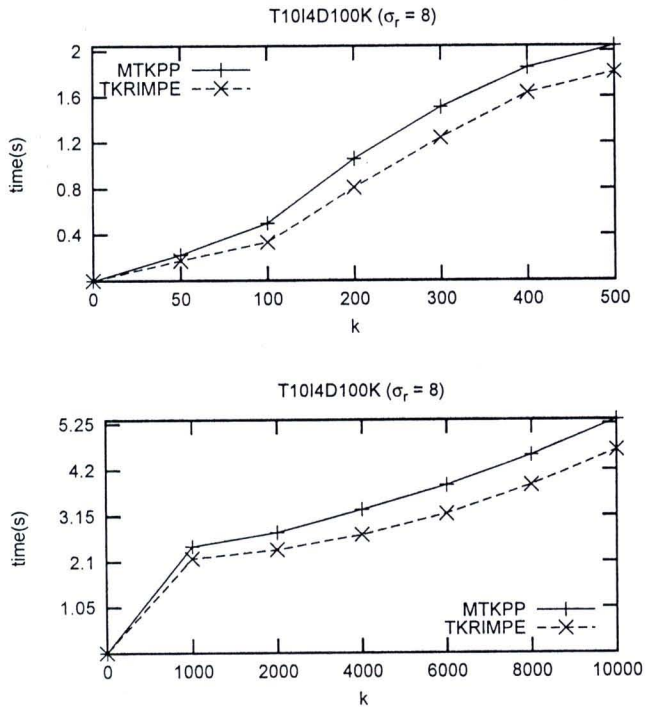

Figure 4.52: Runtime of TKRIMPE on *T10I4D100K* ($\sigma_r = 8\%$)

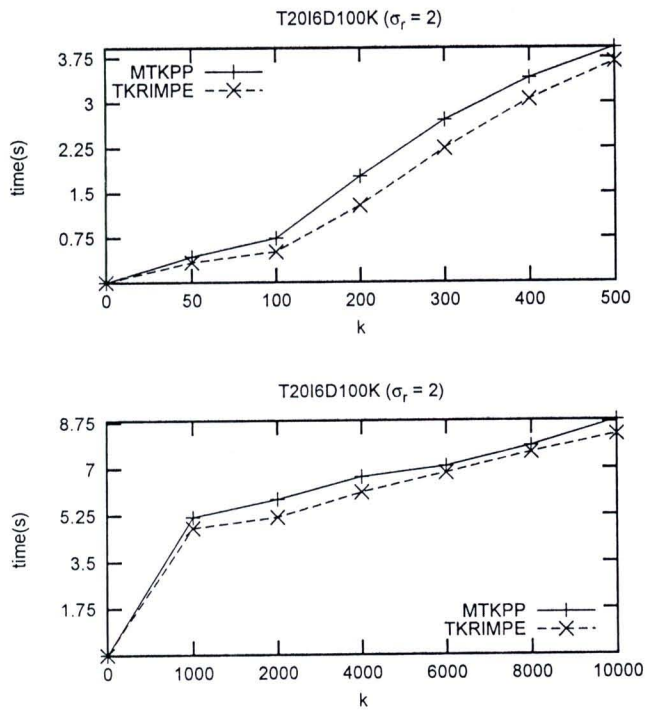

Figure 4.53: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 2\%$)

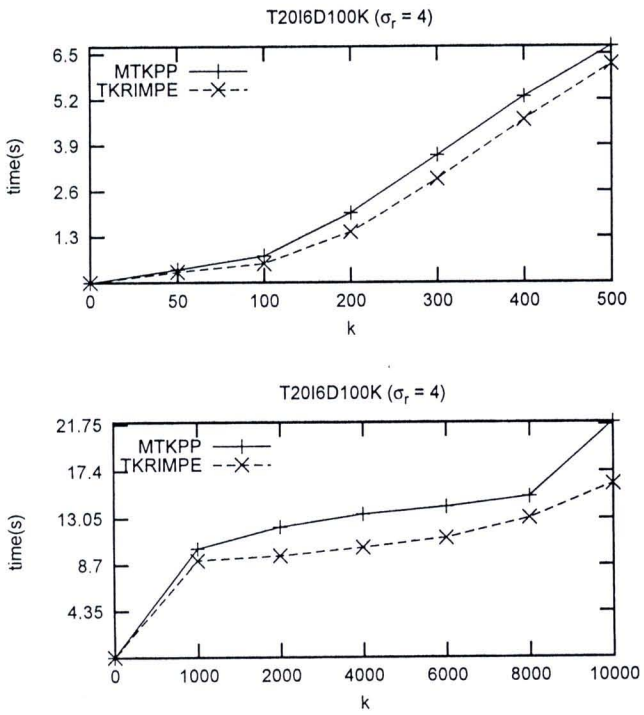

Figure 4.54: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 4\%$)

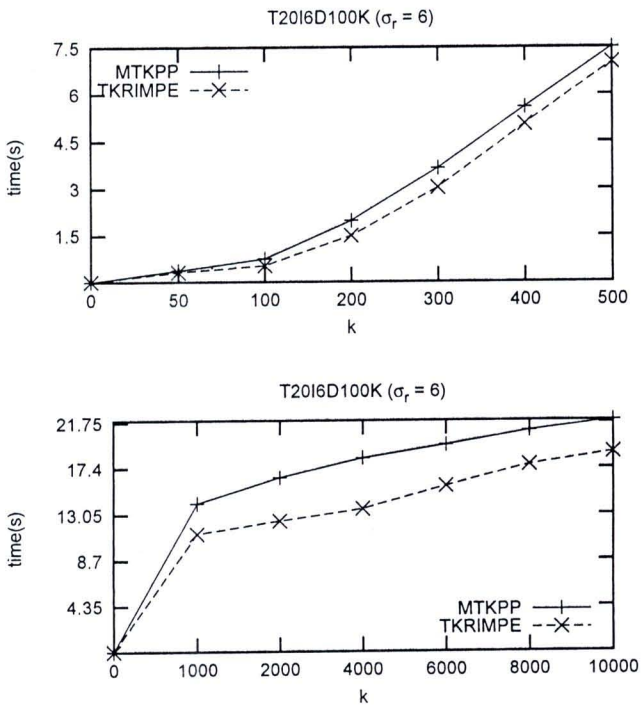

Figure 4.55: Runtime of TKRIMPE on *T20I6D100K* ($\sigma_r = 6\%$)

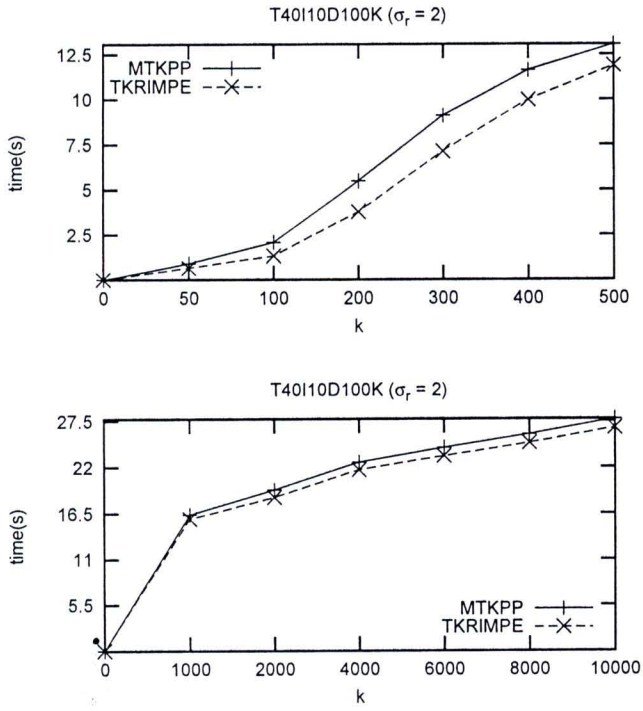

Figure 4.56: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 2\%$)

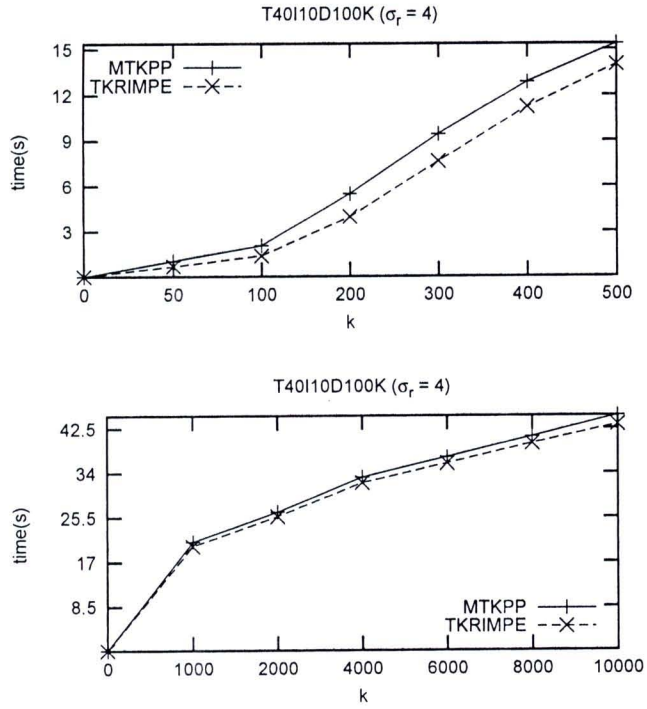

Figure 4.57: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 4\%$)

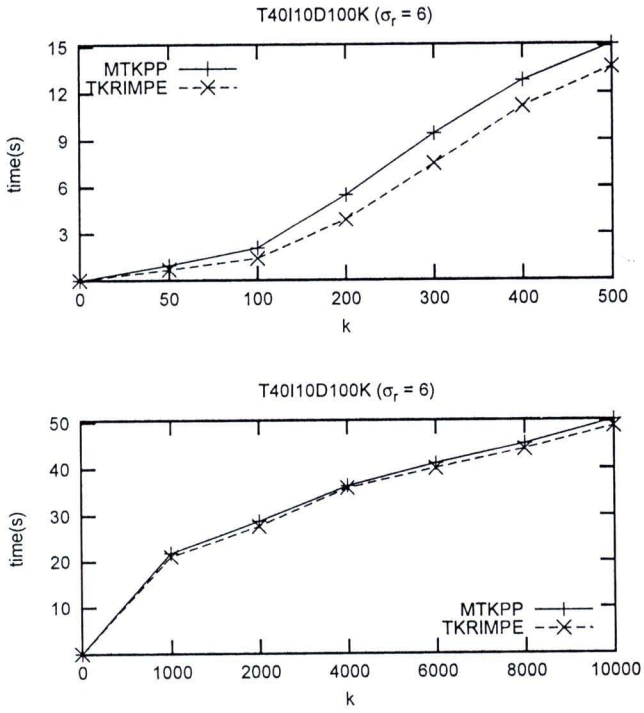

Figure 4.58: Runtime of TKRIMPE on *T40I10D100K* ($\sigma_r = 6\%$)

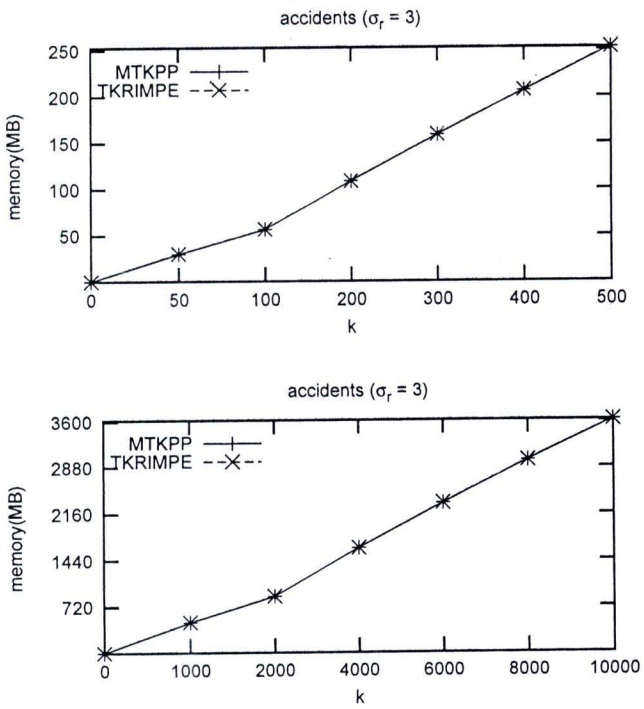

Figure 4.59: Memory usage of TKRIMPE on *accidents*

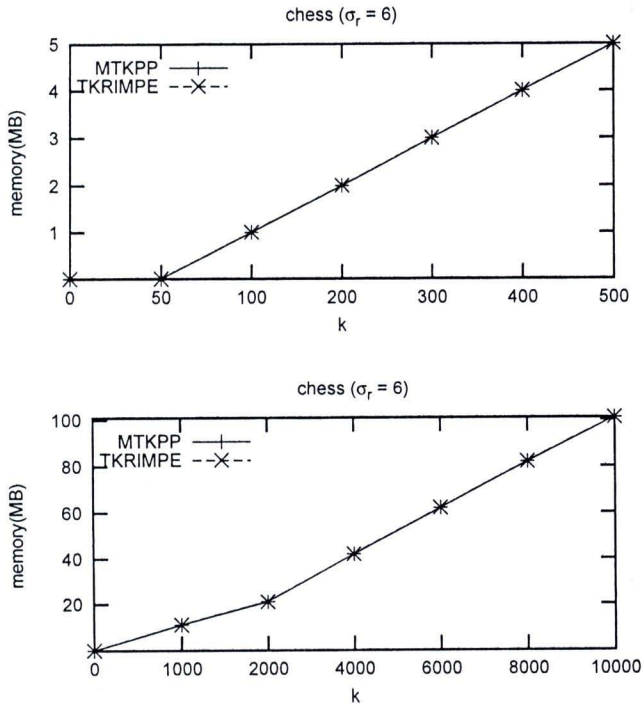

Figure 4.60: Memory usage of TKRIMPE on *chess*

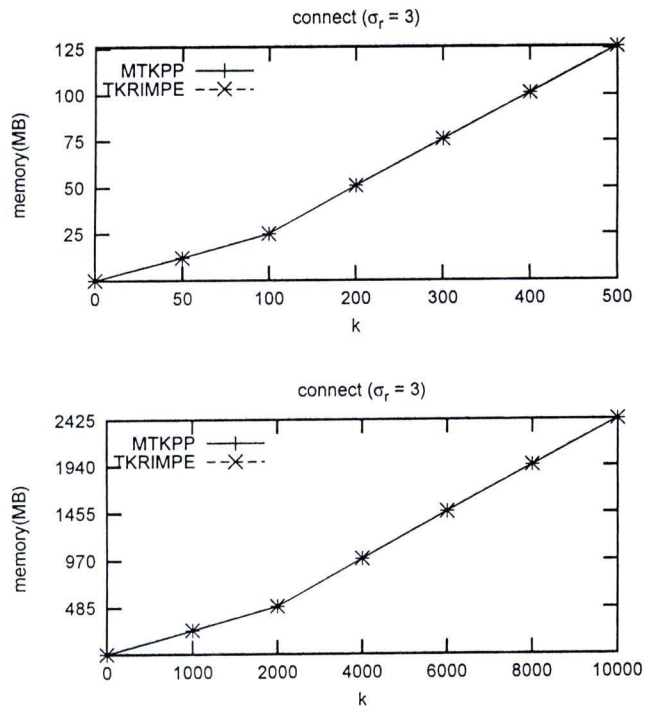

Figure 4.61: Memory usage of TKRIMPE on *connect*

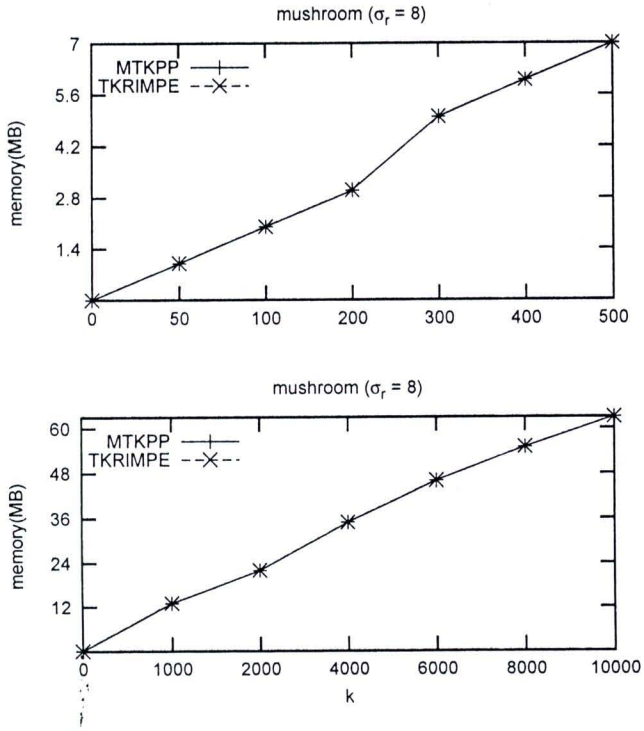

Figure 4.62: Memory usage of TKRIMPE on *mushroom*

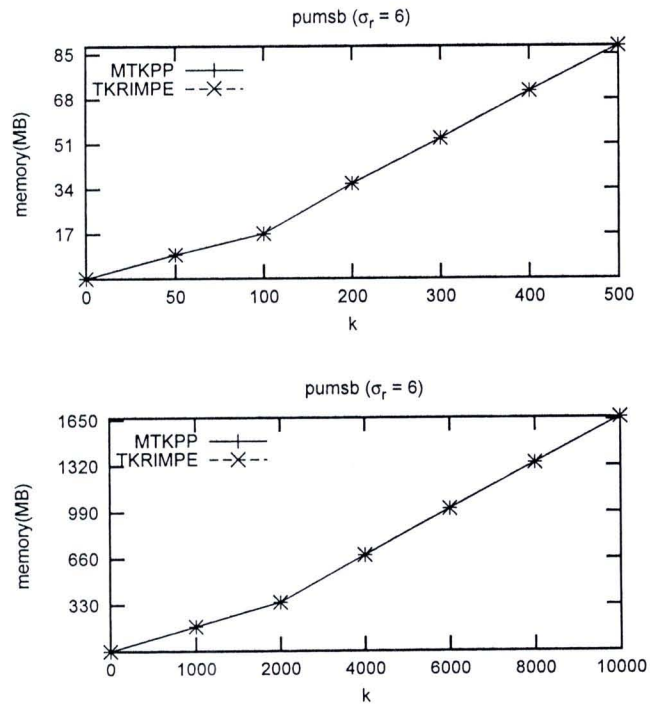

Figure 4.63: Memory usage of TKRIMPE on *pumsb*

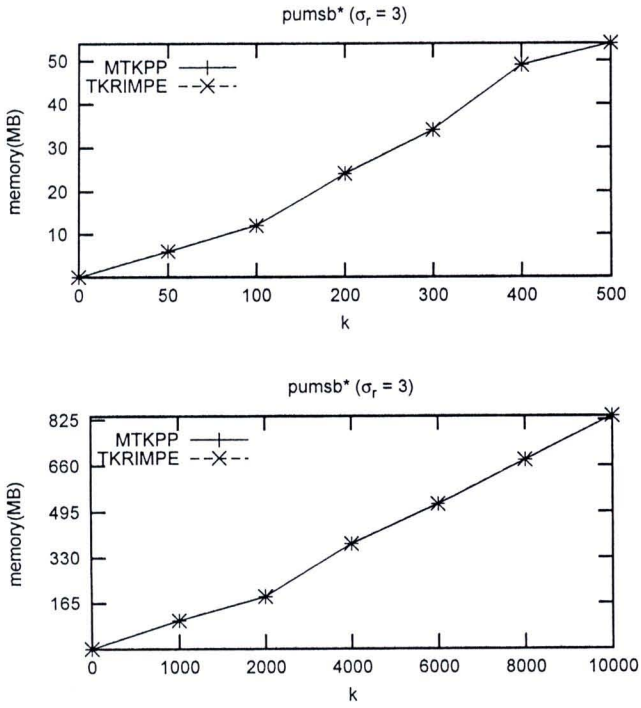

Figure 4.64: Memory usage of TKRIMPE on *pumsb**

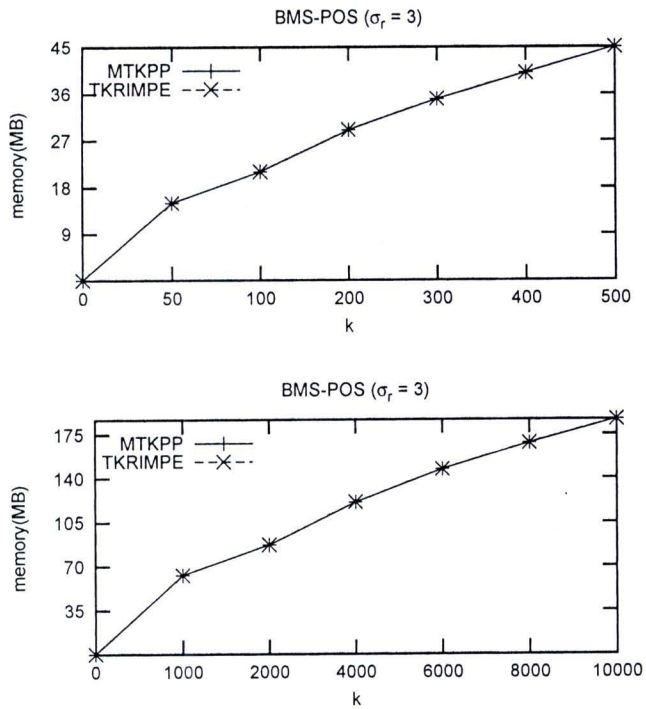

Figure 4.65: Memory usage of TKRIMPE on *BMS-POS*

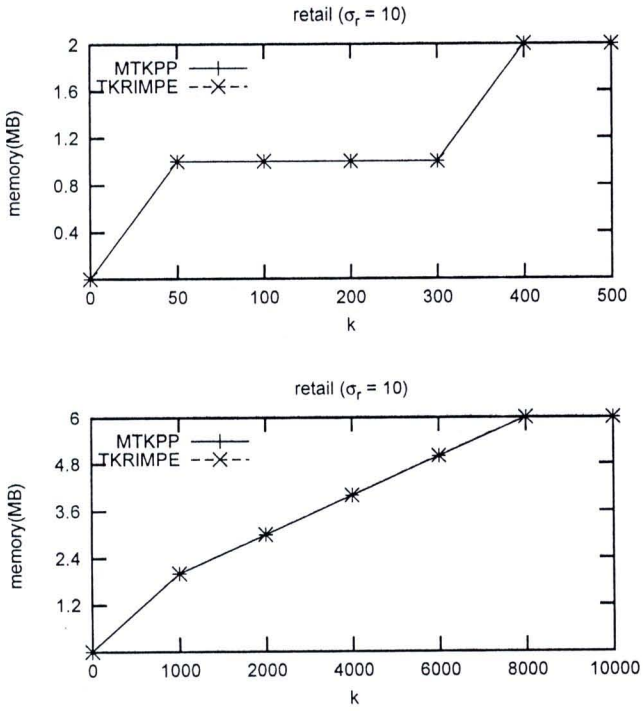

Figure 4.66: Memory usage of TKRIMPE on *retail*

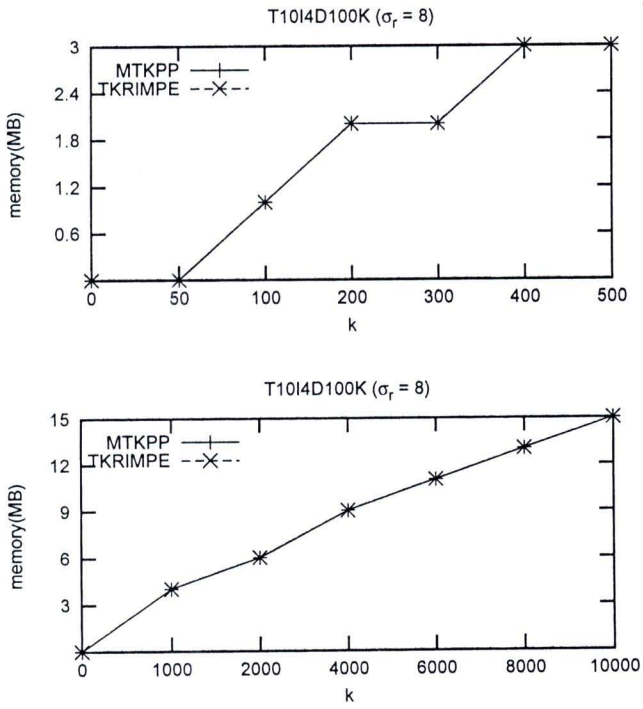

Figure 4.67: Memory usage of TKRIMPE on *T10I4D100K*

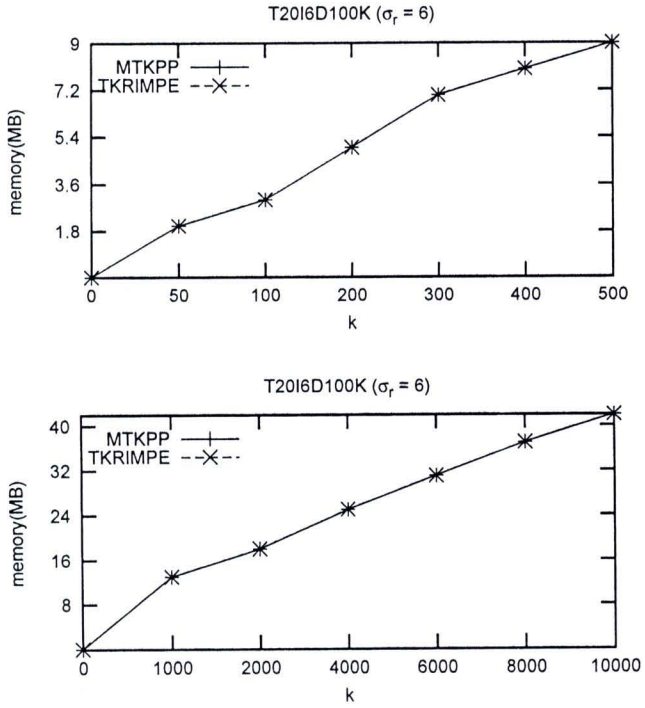

Figure 4.68: Memory usage of TKRIMPE on *T20I6D100K*

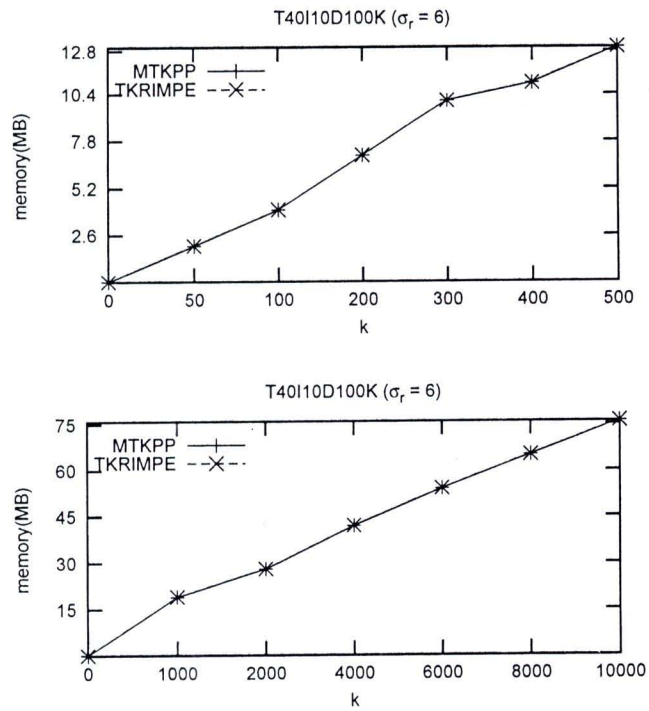

Figure 4.69: Memory usage of TKRIMPE on *T40I10D100K*

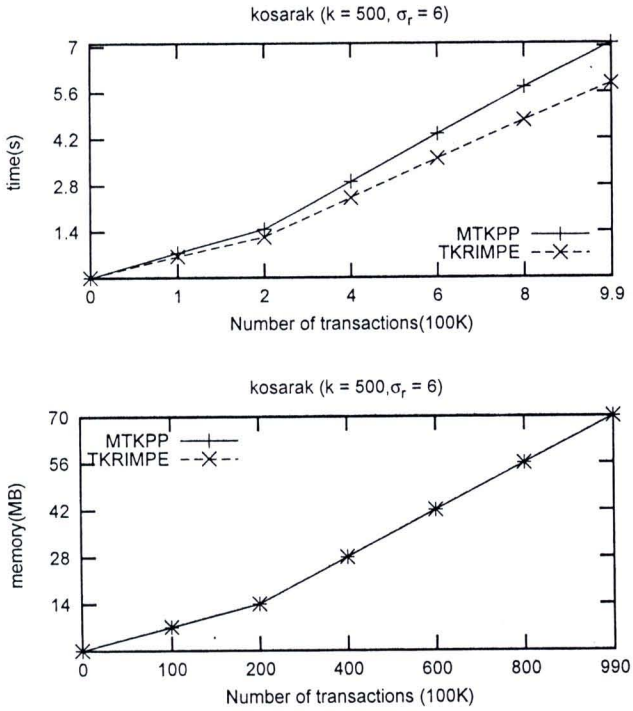

Figure 4.70: Scalability of TKRIMPE ($k : 500, \sigma_r = 6$)

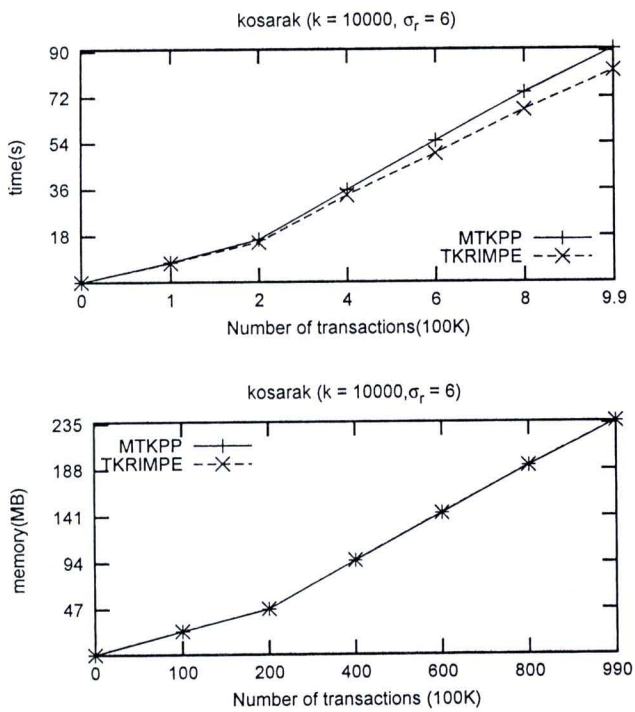

Figure 4.71: Scalability of TKRIMPE ($k : 10,000, \sigma_r = 6$)