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:*

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

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).

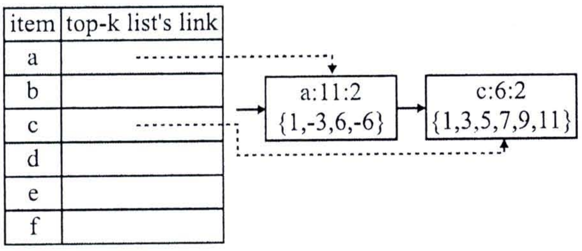


Figure 5.1: TKRIMIT: Top-*k* list structure with hash table

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: $\sigma_r$
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 $i_1^x = i_1^y, i_2^x = i_2^y, \ldots, i_{|I^x|-1}^x = i_{|I^x|-1}^y)$ then
      merge the itemsets of $x$ 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 $\sigma_r$
            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 $\sigma_r$
            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 $\sigma_r$
            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 $t_{|T^Z|}^Z - (t_{q-1} - t_q)$ as the last tid in $T^Z$
            add the support $s^Z$ by $(t_{q-1} - t_q) - t_{|T^Z|}^Z$
          else
            collect $t_{|T^Z|}^Z - (t_{p-1} - t_p)$ as the last tid in $T^Z$
            add the support $s^Z$ by $(t_{p-1} - t_p) - t_{|T^Z|}^Z$
      calculate the regularity $r^Z$ by $|TDB|-$ the last tid of $T^Z$ (in case of the last tid $> 0$, otherwise
      $|TDB|- (t_{|T^Z|-1}^Z - t_{|T^Z|}^Z))$
      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 $t_{|IT^{XY}|}^{XY} - (t_{i-1}^X - t_i^X)$ at the end of IT^{XY} if $t_{i-1}^X - t_i^X < t_{j-1}^Y - t_j^Y$, otherwise add $t_{|IT^{XY}|}^{XY} - (t_{j-1}^Y - t_j^Y)$ 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 $(1 : 1 : \{1\})$

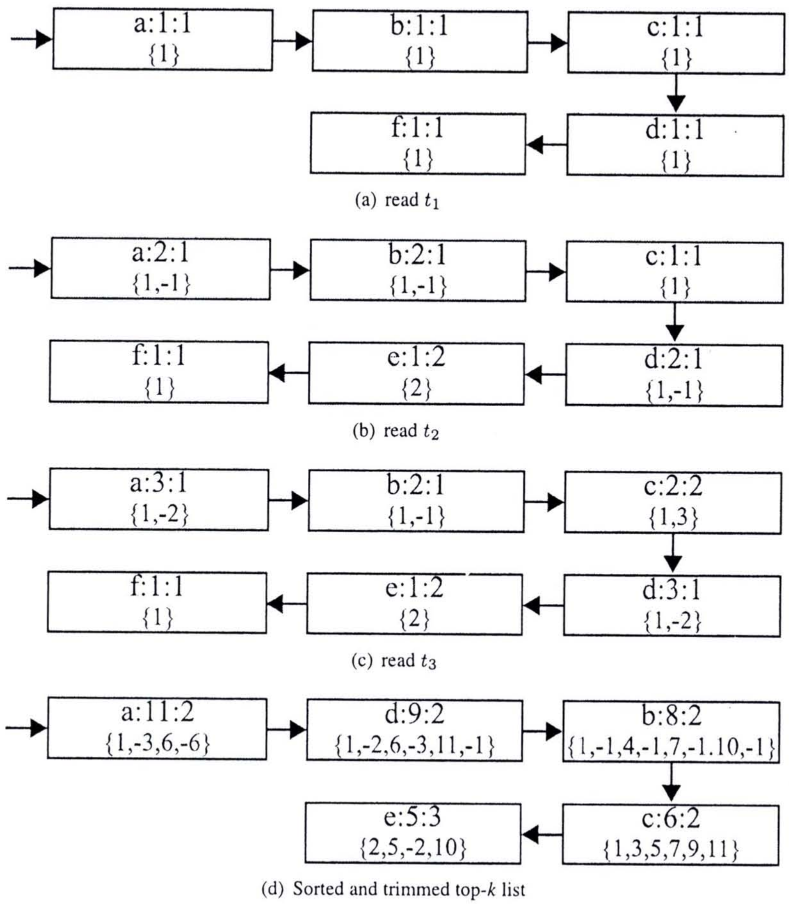
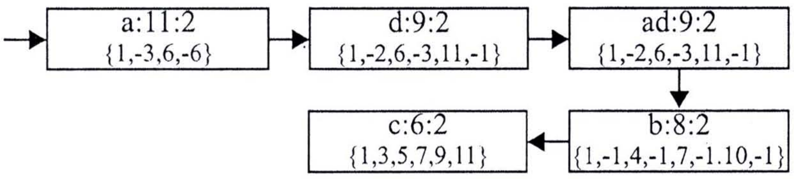


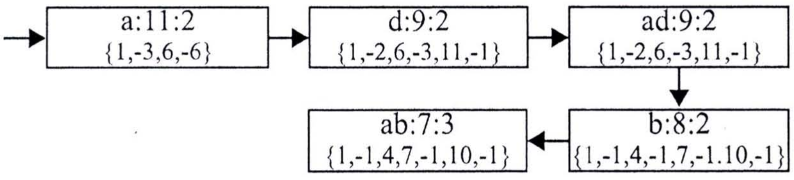
Figure 5.2: Top-k list initialization

(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.

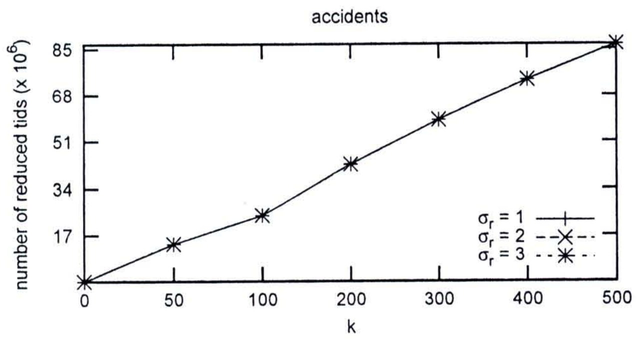
In Figure 5.59 and Figure 5.60, the scalability of TKRIMIT, MTKPP and TKRIMPE are tested in terms of runtime with different number of transactions in the database. From these figures, the runtime of TKRIMIT scales linearly increase when the size of database increases. Based on the interval tidset representation, 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.



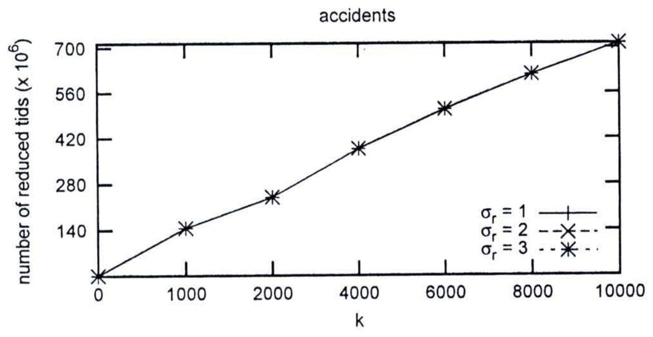
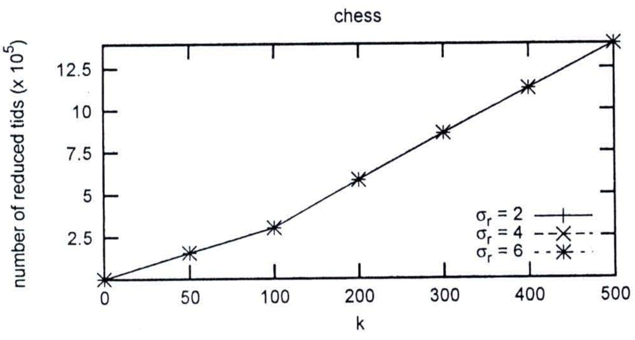


Figure 5.4: The number of reduced tids from TKRIMIT on *accidents* datasets



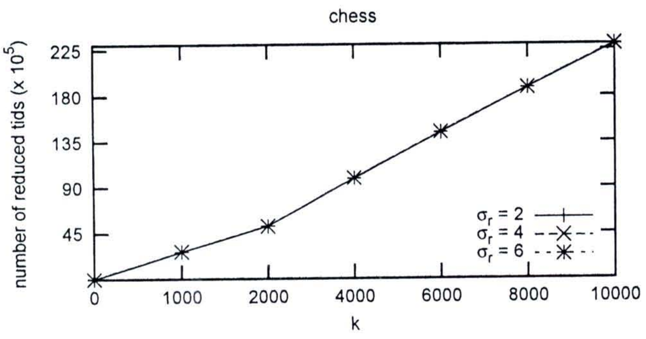


Figure 5.5: The number of reduced tids from TKRIMIT on *chess* datasets

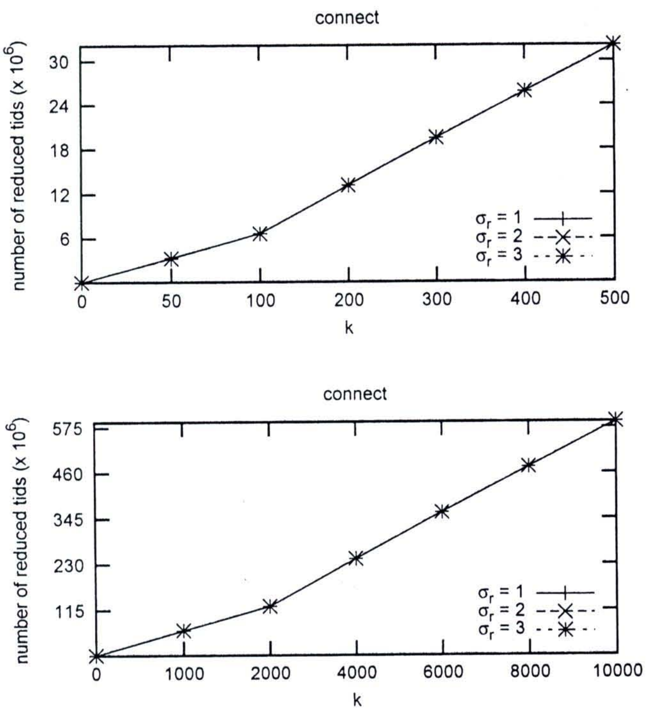


Figure 5.6: The number of reduced tids from TKRIMIT on *connect* datasets

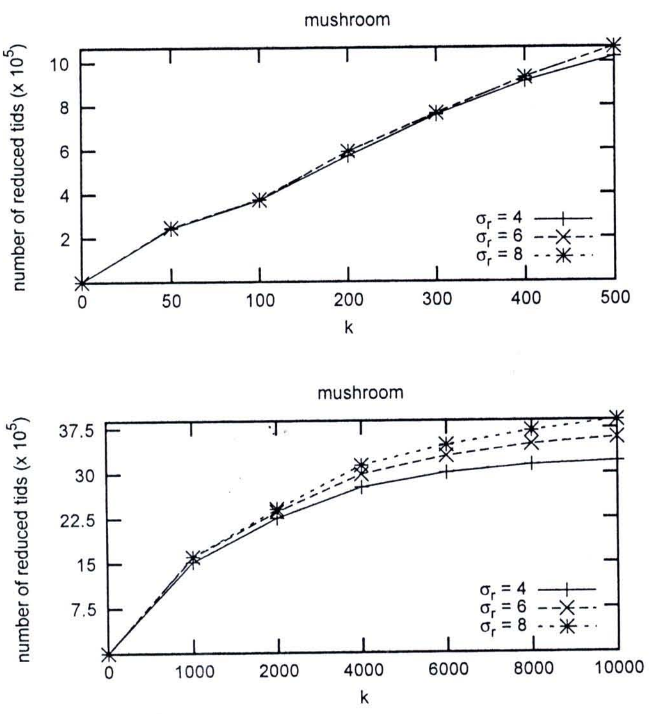
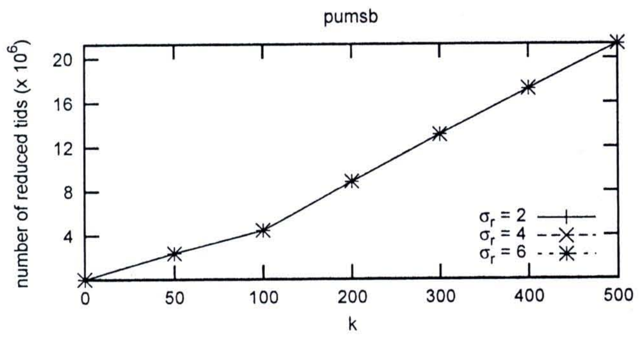


Figure 5.7: The number of reduced tids from TKRIMIT on *mushroom* datasets



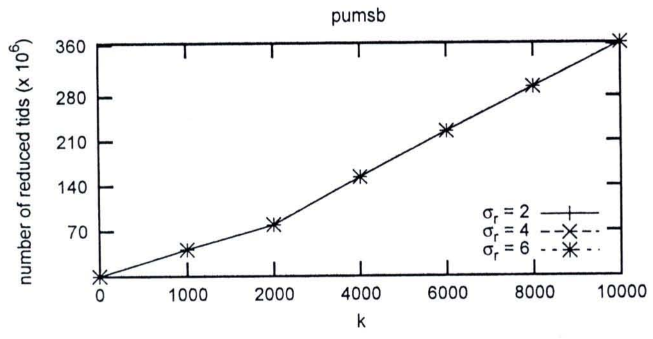
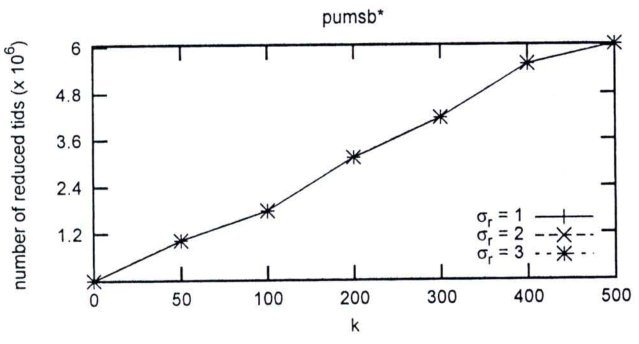


Figure 5.8: The number of reduced tids from TKRIMIT on *pumsb* datasets



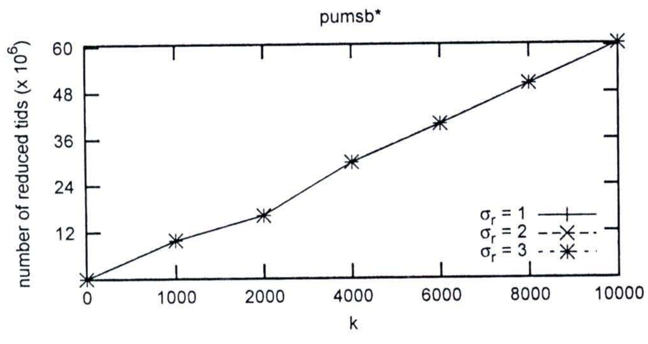


Figure 5.9: The number of reduced tids from TKRIMIT on *pumsb** datasets

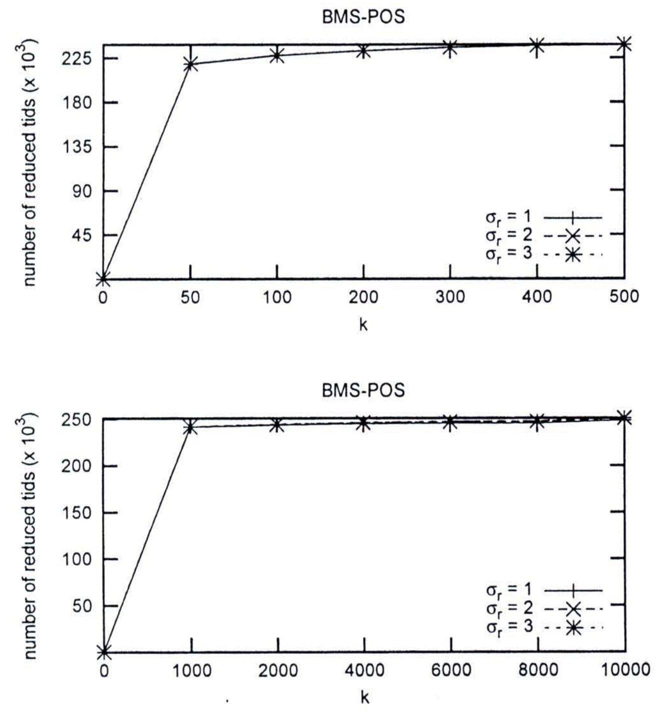


Figure 5.10: The number of reduced tids from TKRIMIT on *BMS-POS* datasets

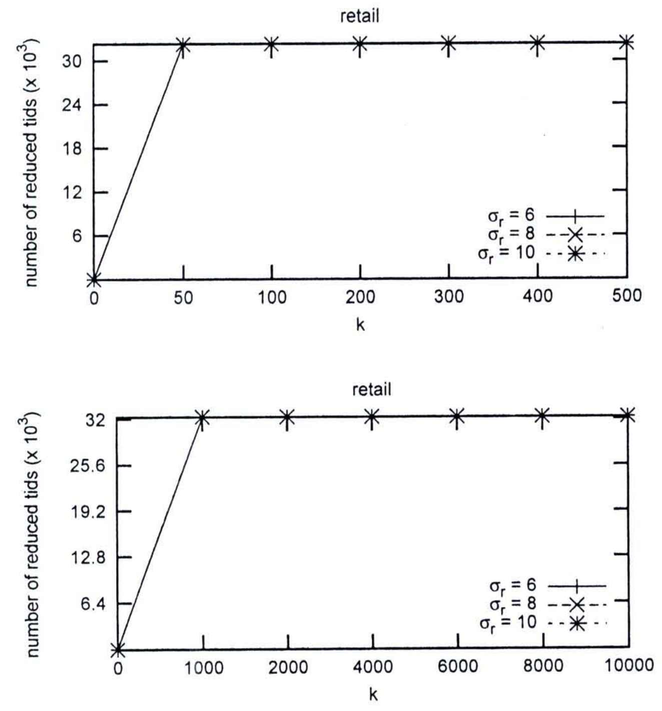


Figure 5.11: The number of reduced tids from TKRIMIT on *retail* datasets

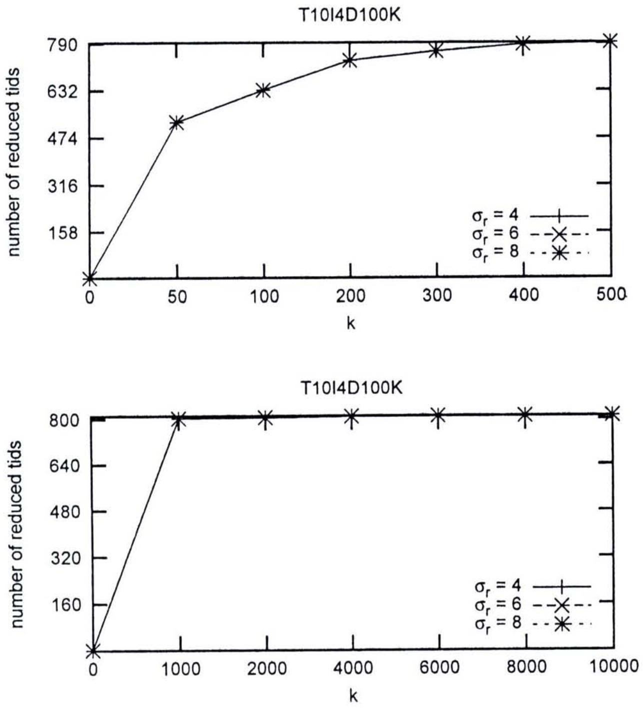


Figure 5.12: The number of reduced tids from TKRIMIT on *T10I4D100K* datasets

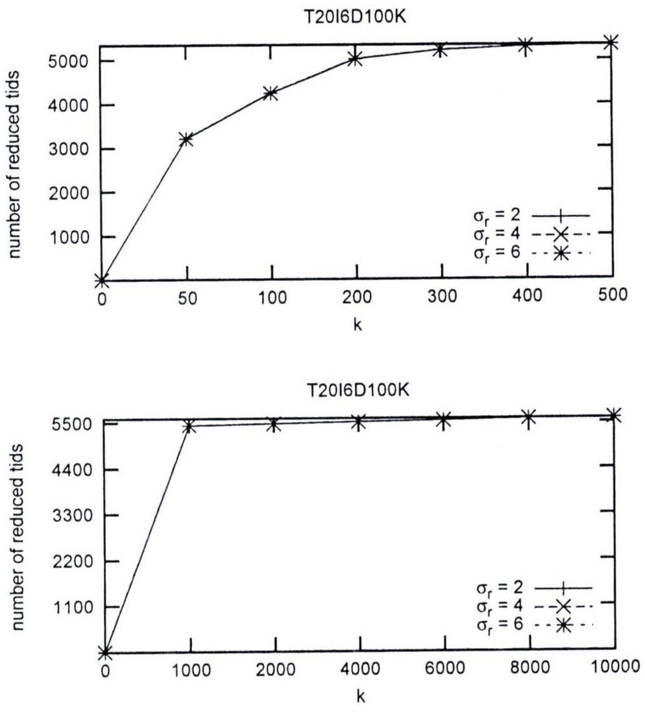
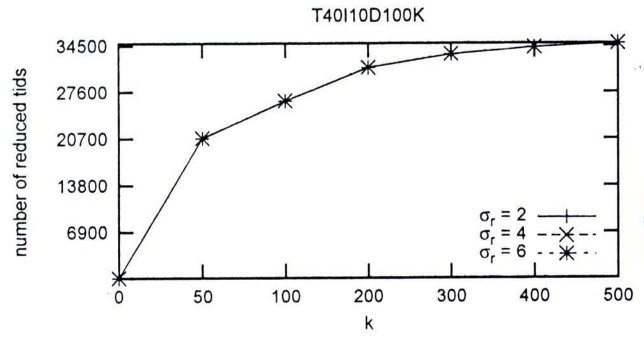


Figure 5.13: The number of reduced tids from TKRIMIT on *T20I6D100K* datasets



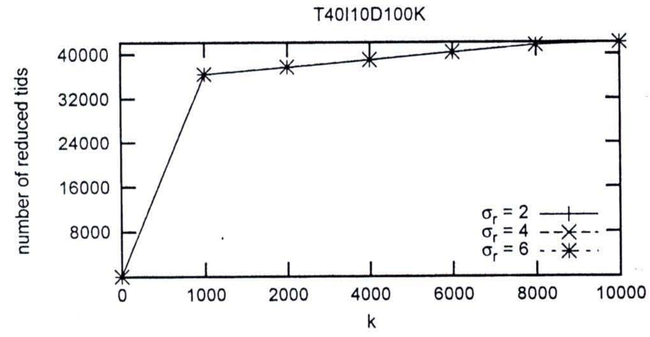
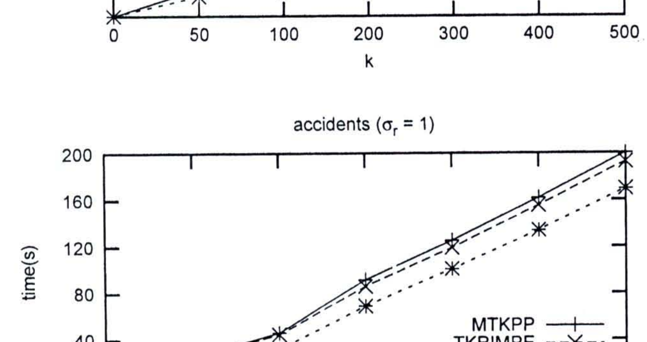


Figure 5.14: The number of reduced tids from TKRIMIT on *T40I10D100K* datasets



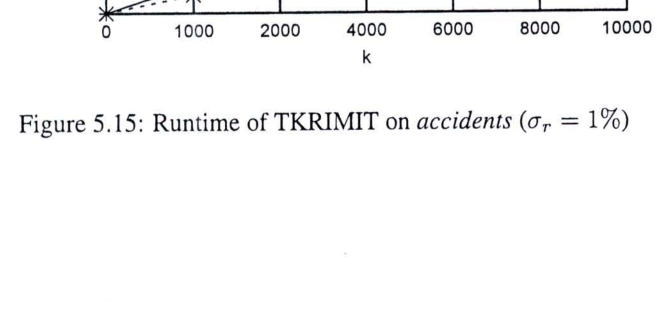
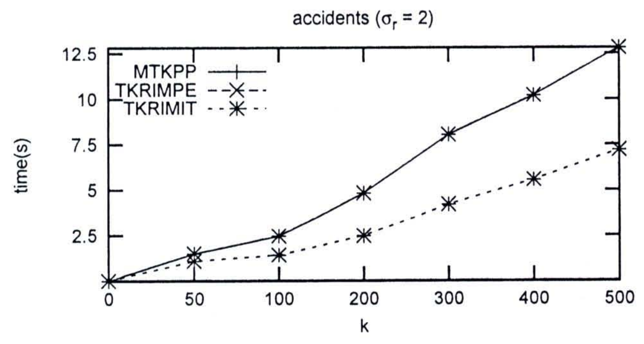


Figure 5.15: Runtime of TKRIMIT on *accidents* ($\sigma_r = 1\%$)



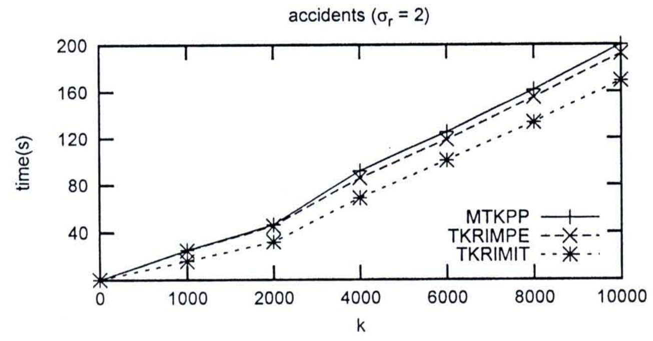
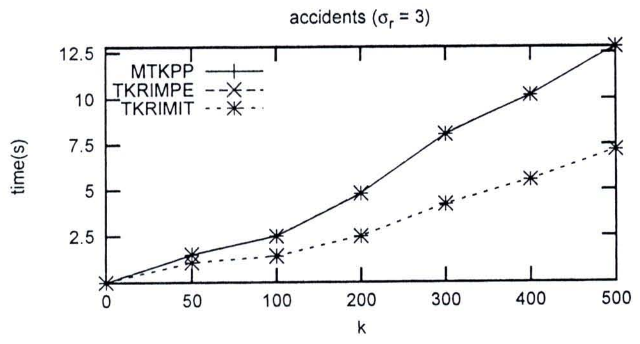


Figure 5.16: Runtime of TKRIMIT on *accidents* ($\sigma_r = 2\%$)



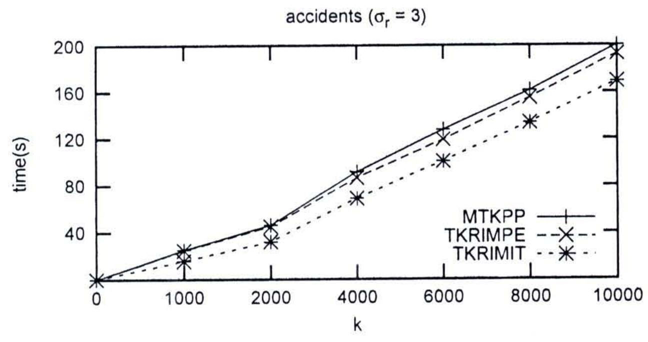


Figure 5.17: Runtime of TKRIMIT on *accidents* ($\sigma_r = 3\%$)

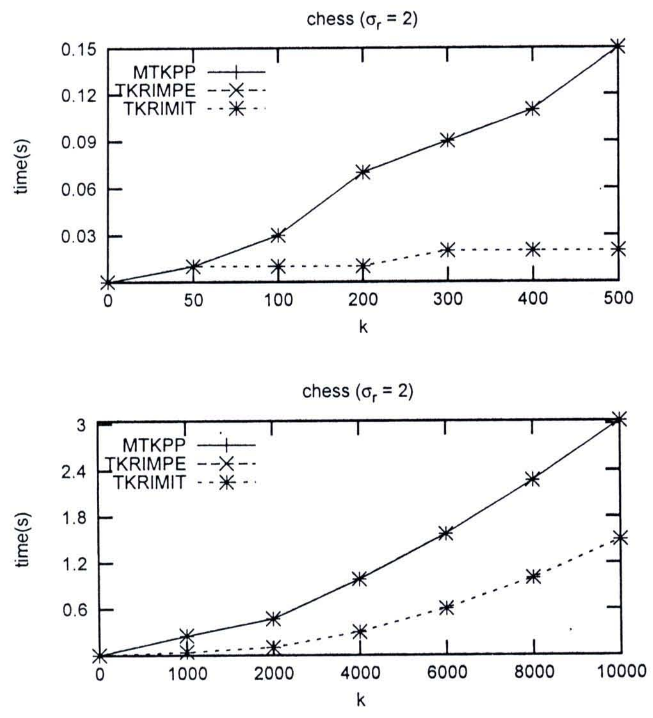


Figure 5.18: Runtime of TKRIMIT on *chess* ($\sigma_r = 2\%$)

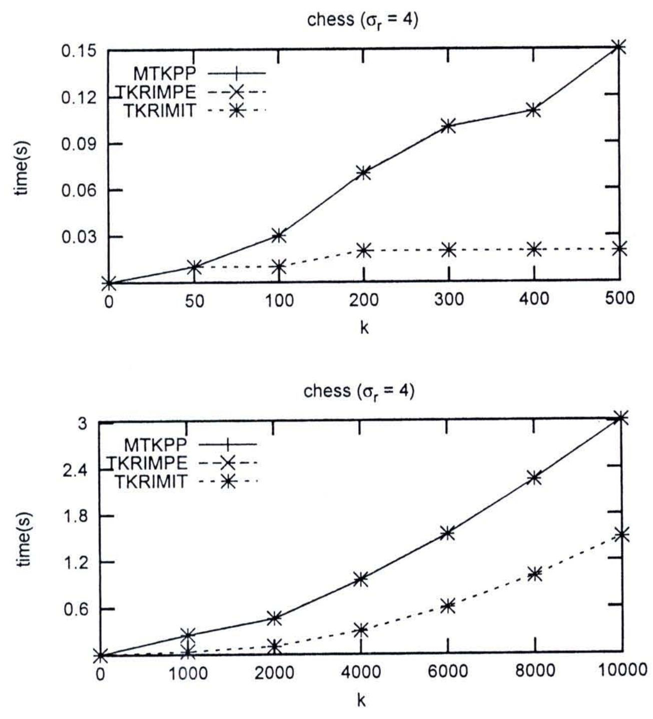
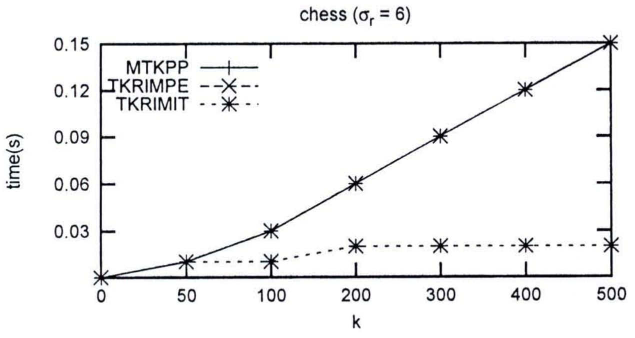


Figure 5.19: Runtime of TKRIMIT on *chess* ($\sigma_r = 4\%$)



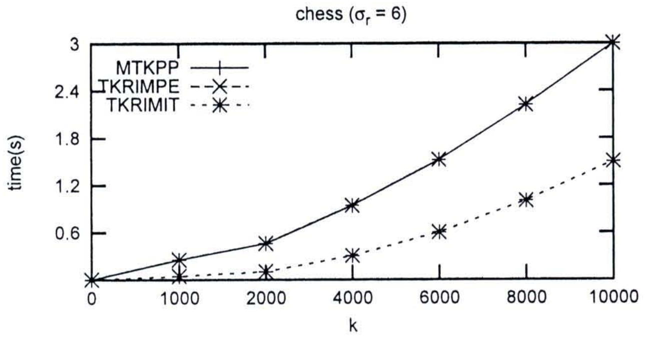
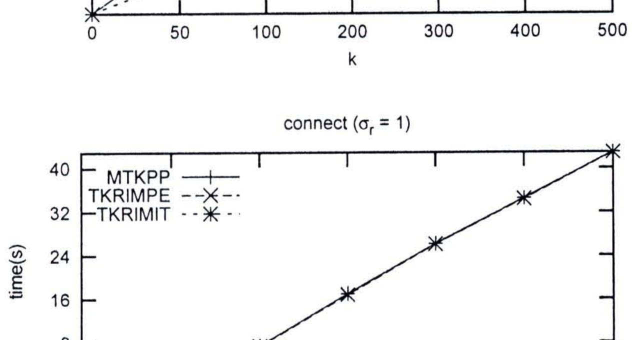


Figure 5.20: Runtime of TKRIMIT on *chess* ($\sigma_r = 6\%$)



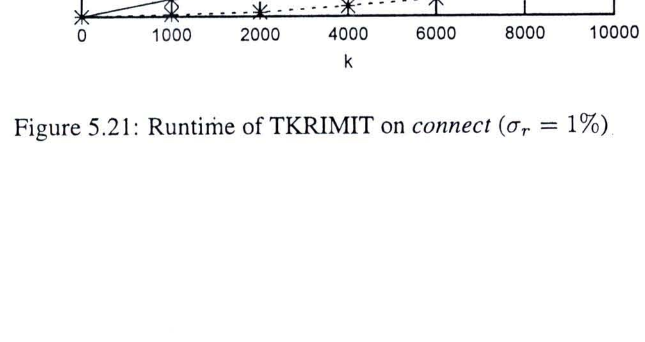
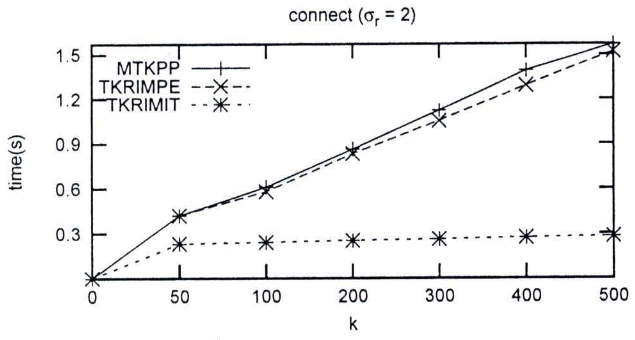


Figure 5.21: Runtime of TKRIMIT on *connect* ($\sigma_r = 1\%$)



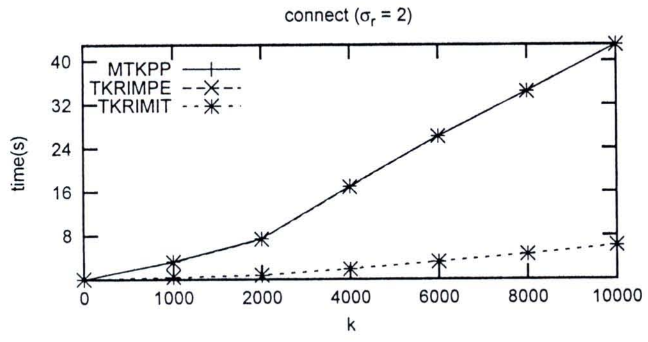
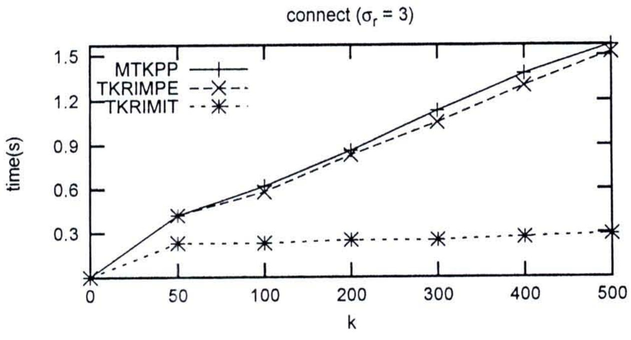


Figure 5.22: Runtime of TKRIMIT on *connect* ($\sigma_r = 2\%$)



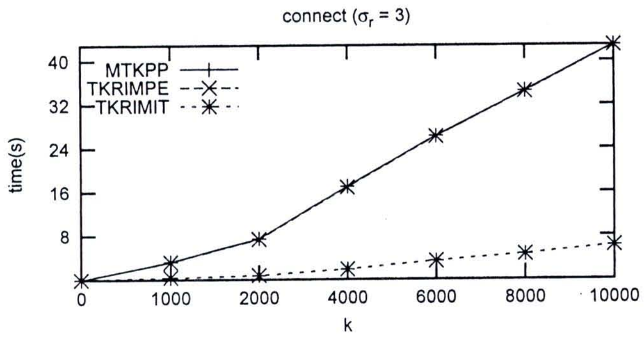
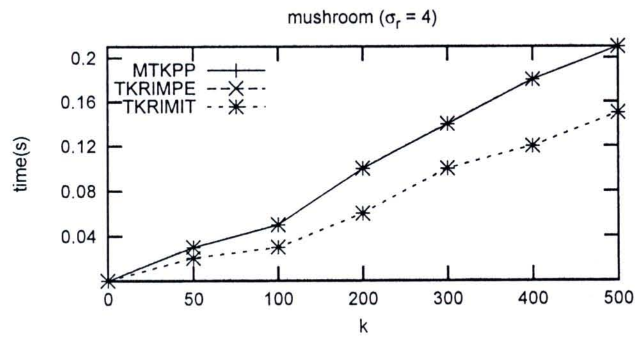


Figure 5.23: Runtime of TKRIMIT on *connect* ($\sigma_r = 3\%$)



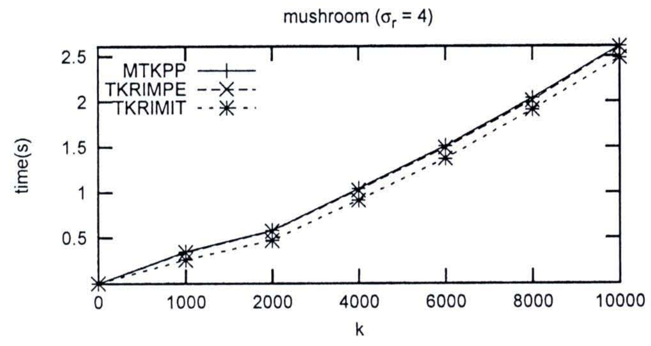
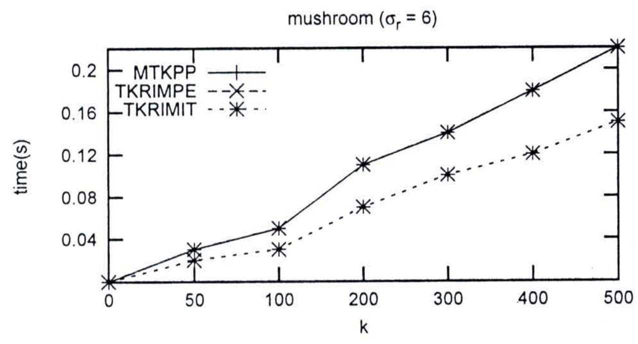


Figure 5.24: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 4\%$)



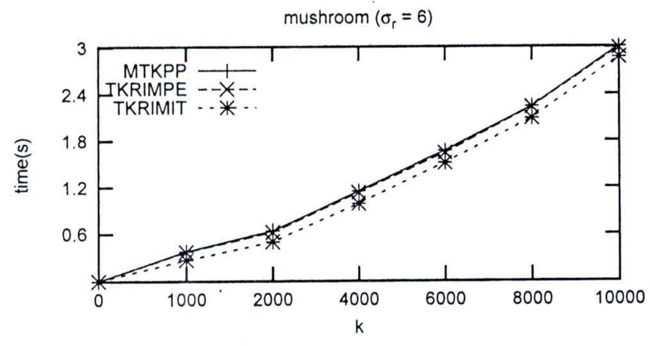


Figure 5.25: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 6\%$)

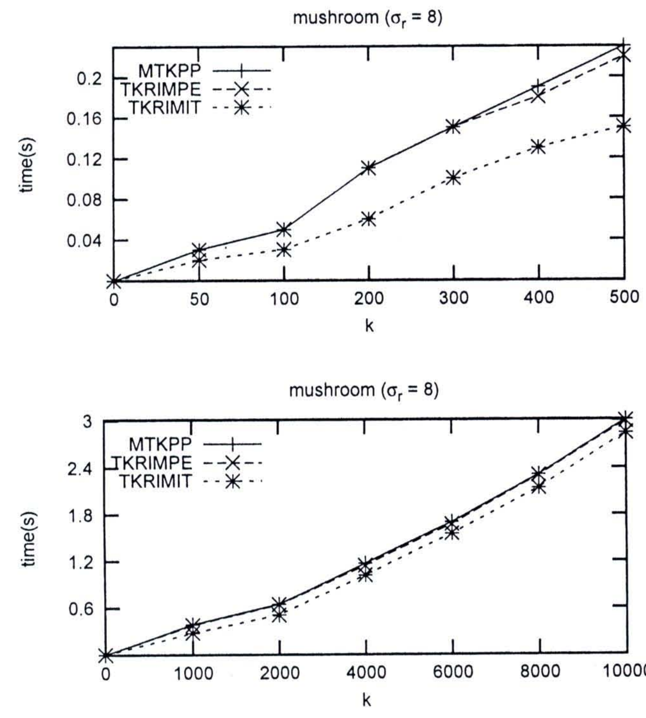


Figure 5.26: Runtime of TKRIMIT on *mushroom* ($\sigma_r = 8\%$)

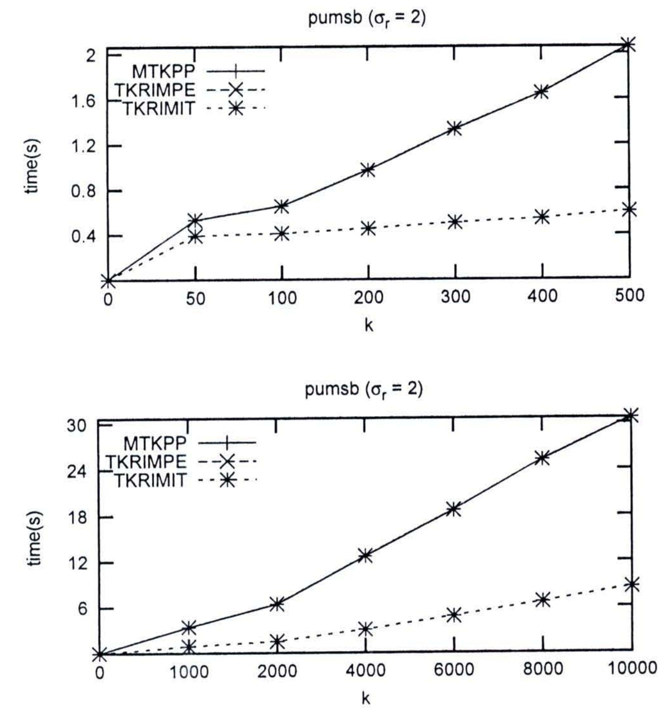


Figure 5.27: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 2\%$)

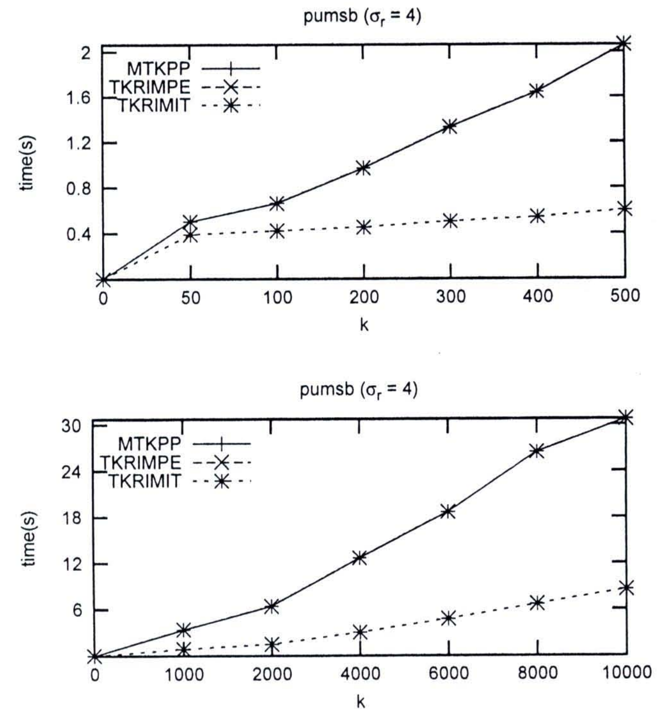


Figure 5.28: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 4\%$)

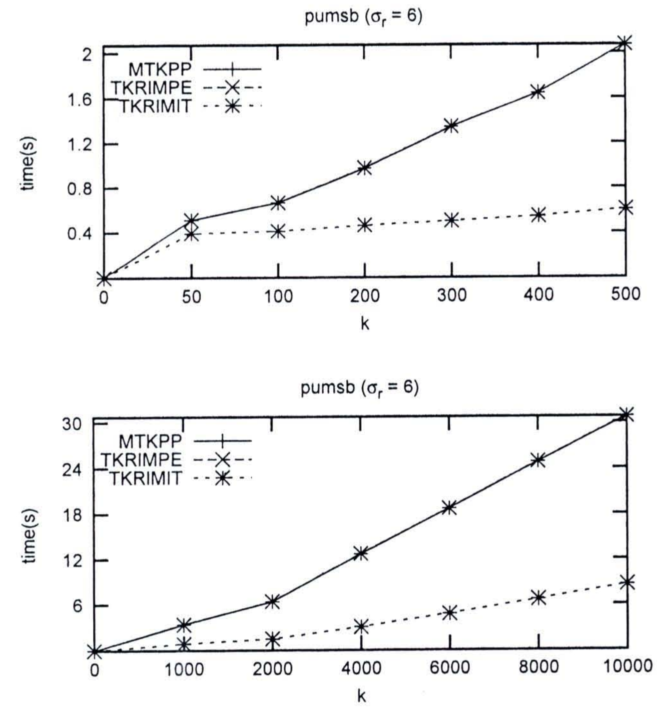


Figure 5.29: Runtime of TKRIMIT on *pumsb* ($\sigma_r = 6\%$)

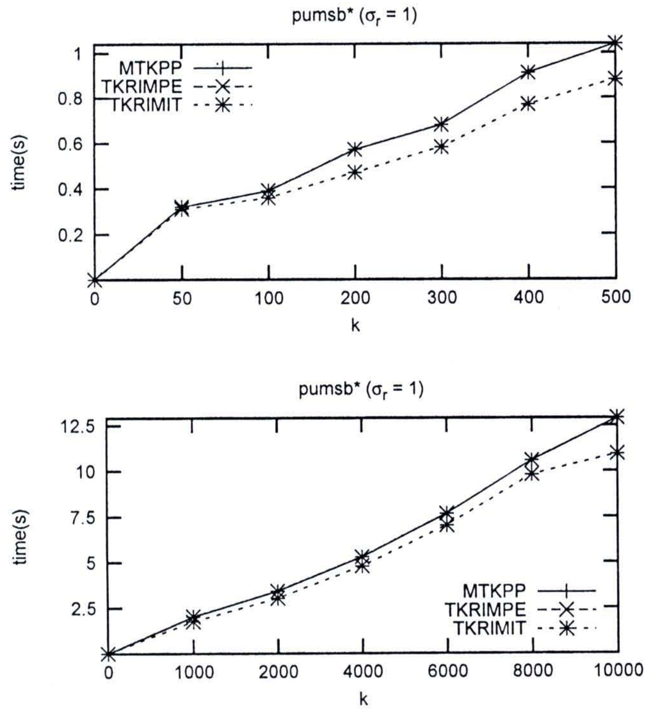


Figure 5.30: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 1\%$)

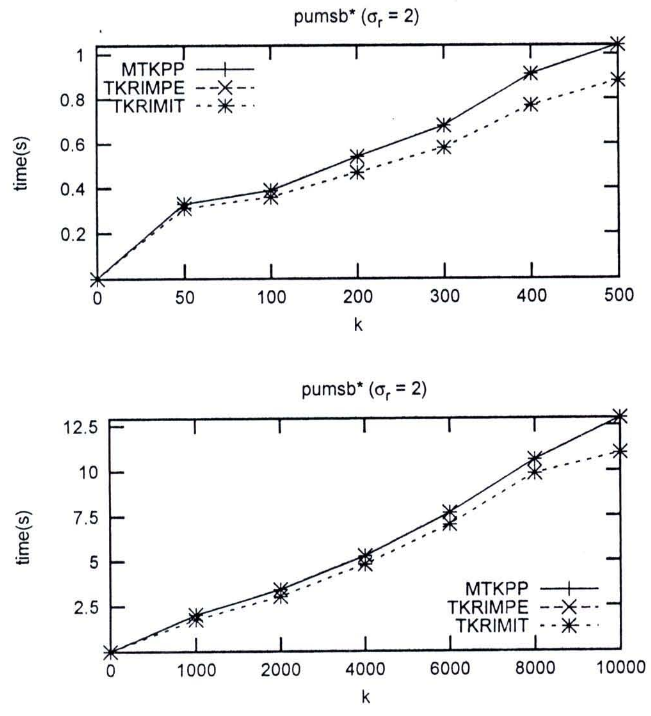
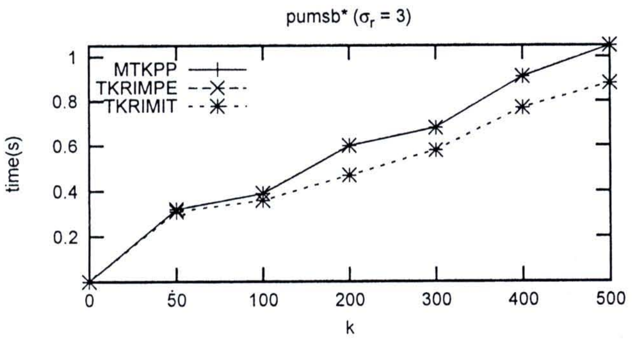


Figure 5.31: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 2\%$)



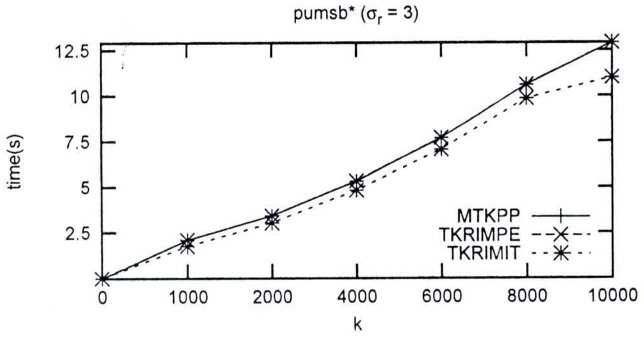
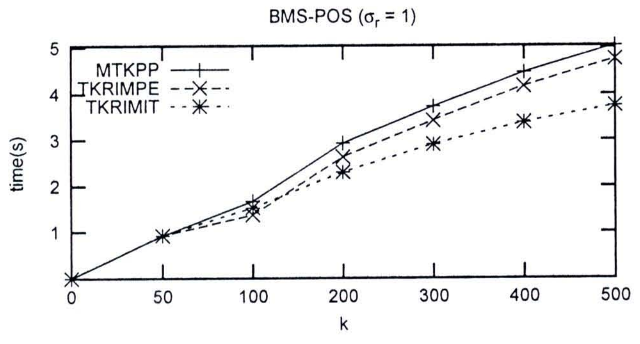


Figure 5.32: Runtime of TKRIMIT on *pumsb** ($\sigma_r = 3\%$)



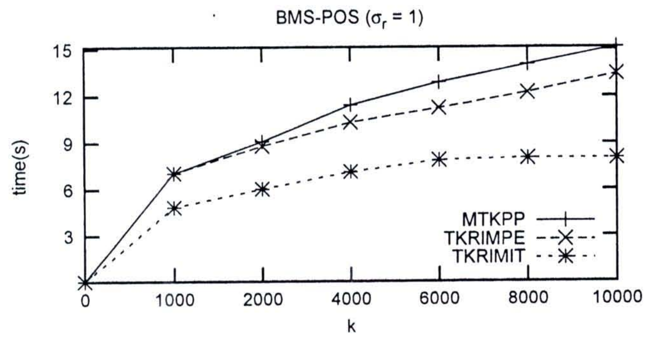
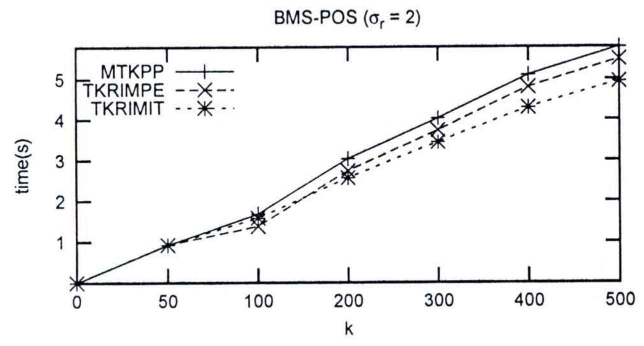


Figure 5.33: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 1\%$)



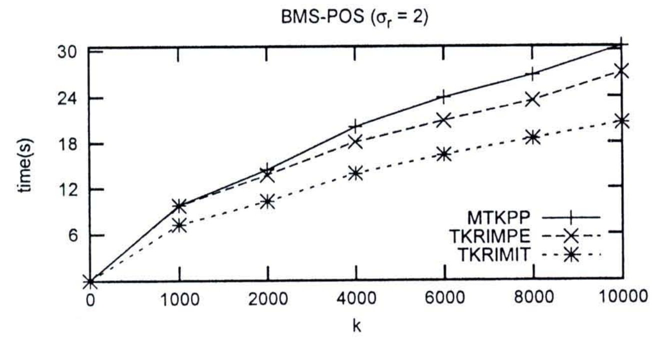
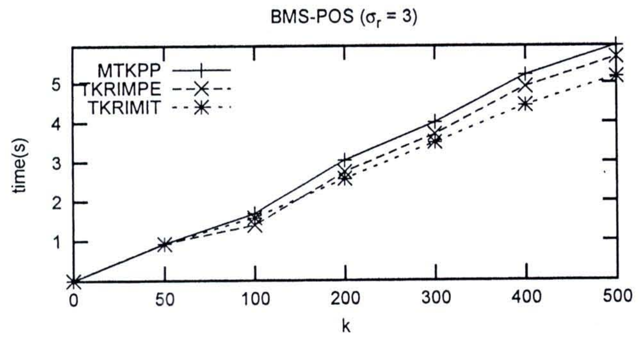


Figure 5.34: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 2\%$)



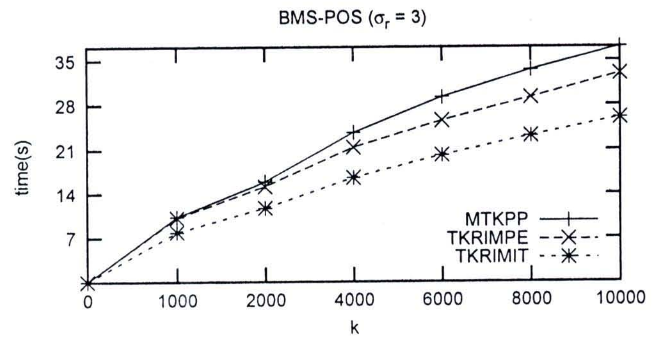
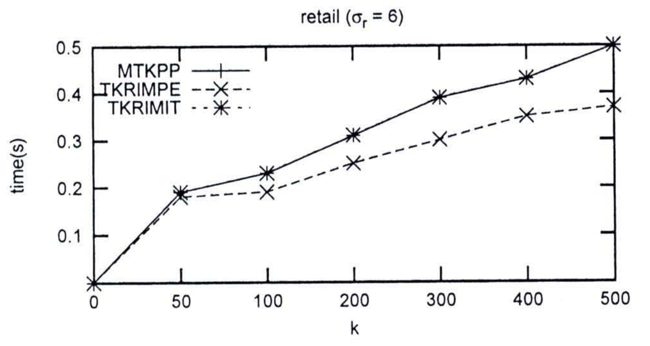


Figure 5.35: Runtime of TKRIMIT on *BMS-POS* ($\sigma_r = 3\%$)



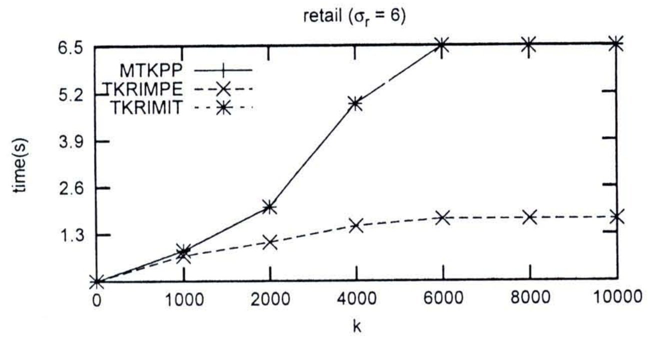
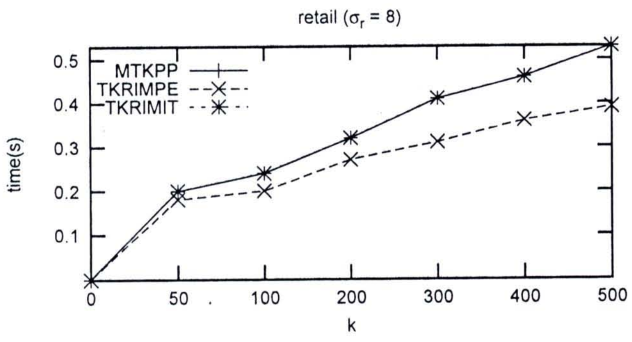


Figure 5.36: Runtime of TKRIMIT on *retail* ($\sigma_r = 6\%$)



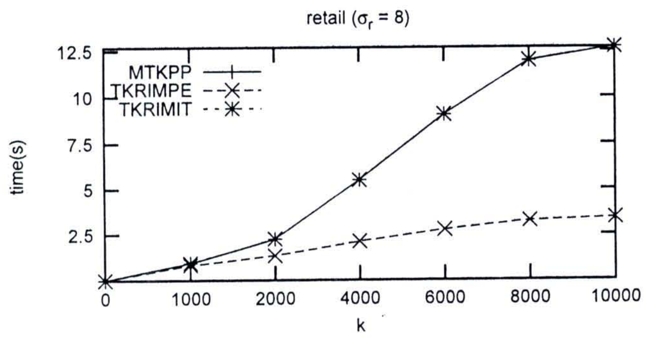
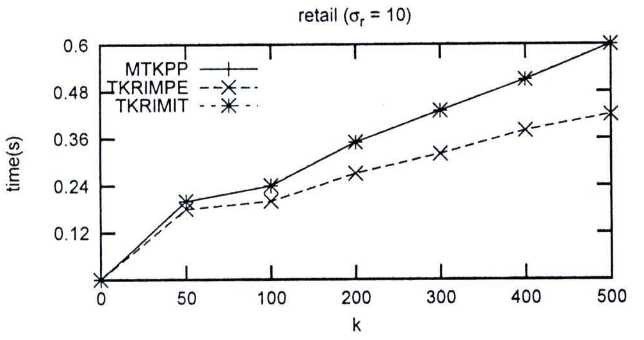


Figure 5.37: Runtime of TKRIMIT on *retail* ($\sigma_r = 8\%$)



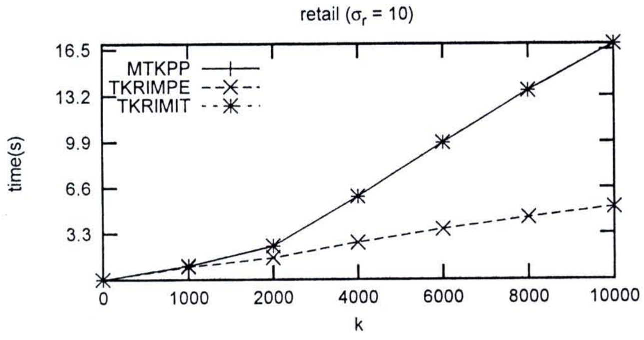
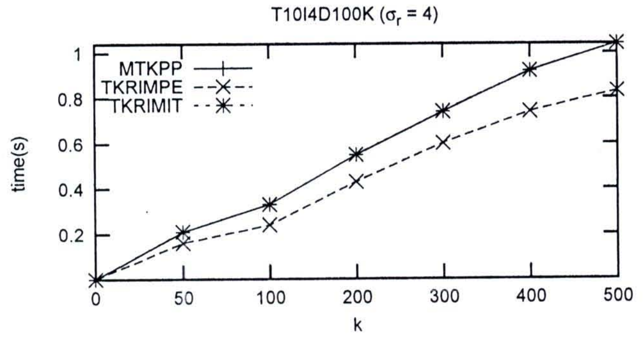


Figure 5.38: Runtime of TKRIMIT on *retail* ($\sigma_r = 10\%$)



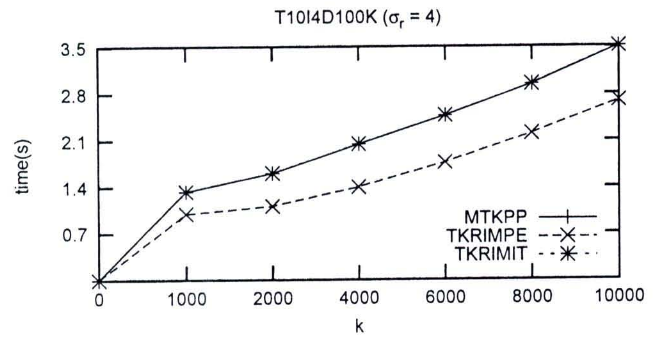
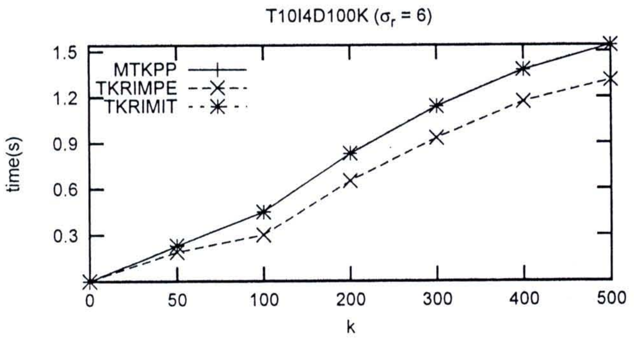


Figure 5.39: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 4\%$)



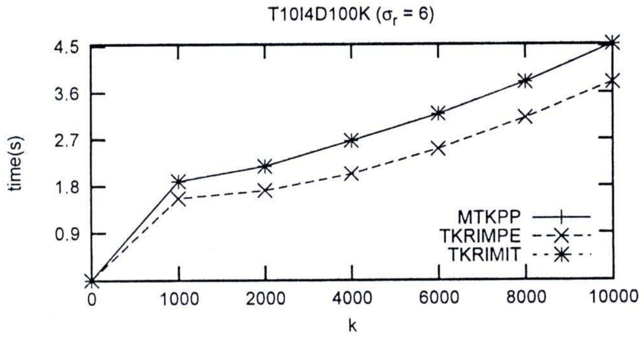
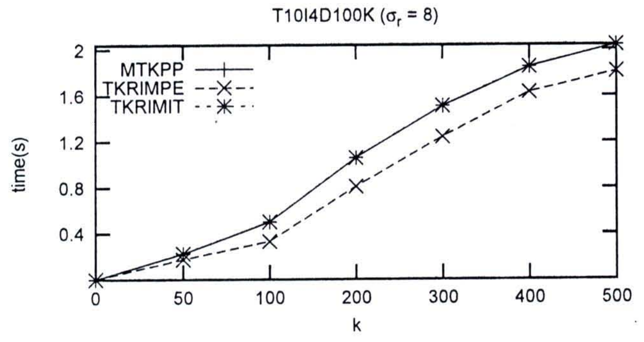


Figure 5.40: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 6\%$)



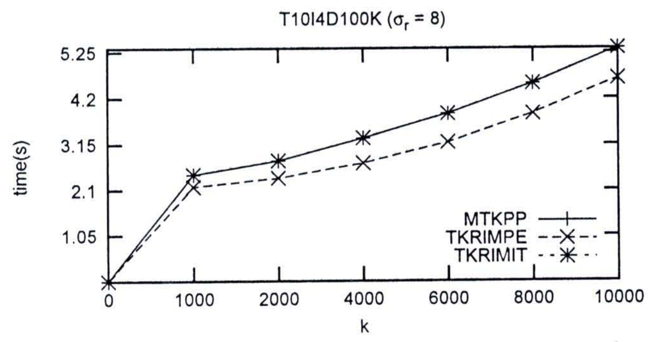
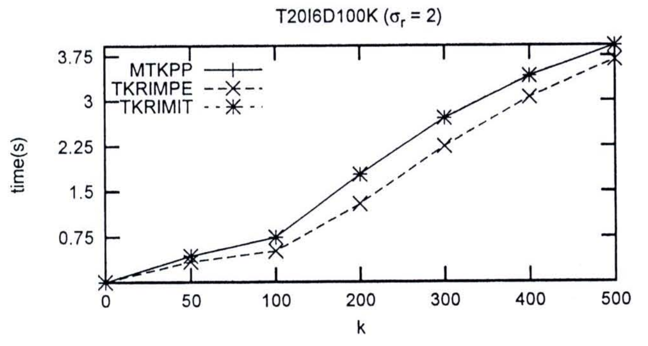


Figure 5.41: Runtime of TKRIMIT on *T10I4D100K* ($\sigma_r = 8\%$)



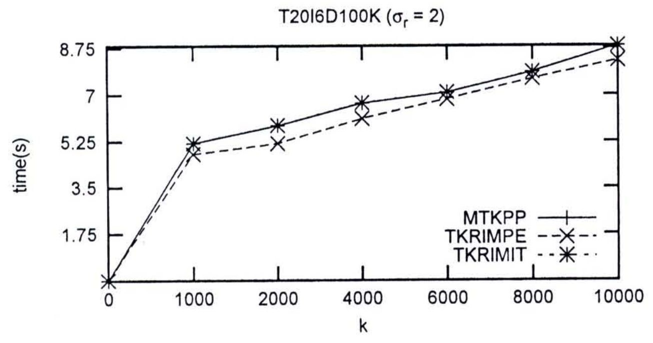
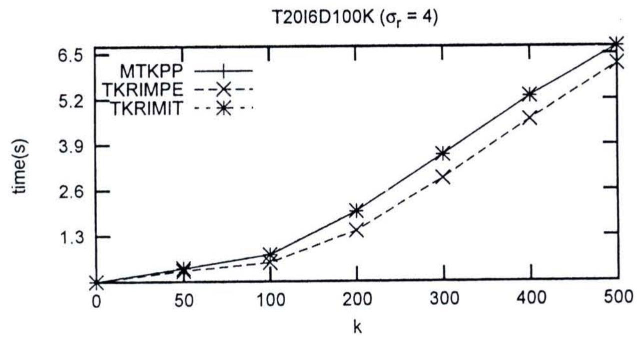


Figure 5.42: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 2\%$)



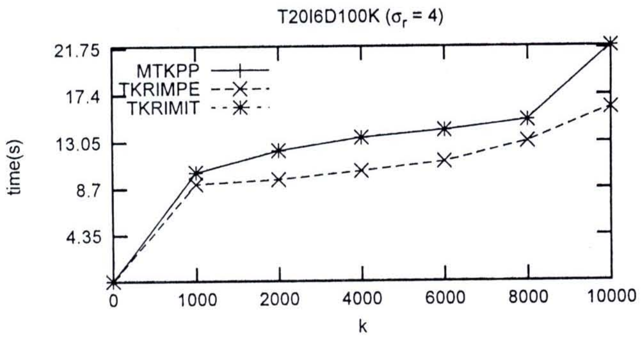
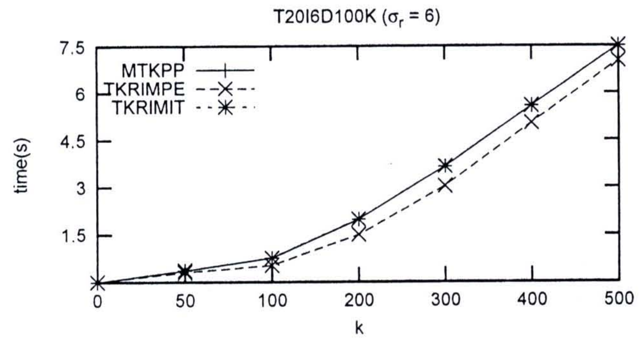


Figure 5.43: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 4\%$)



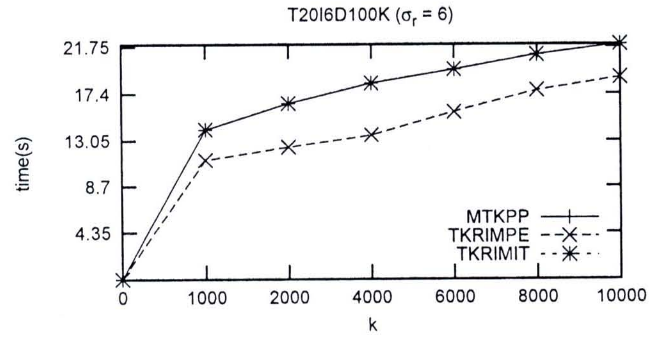
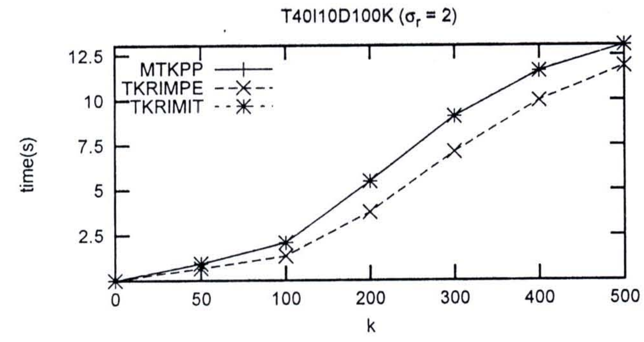


Figure 5.44: Runtime of TKRIMIT on *T20I6D100K* ($\sigma_r = 6\%$)



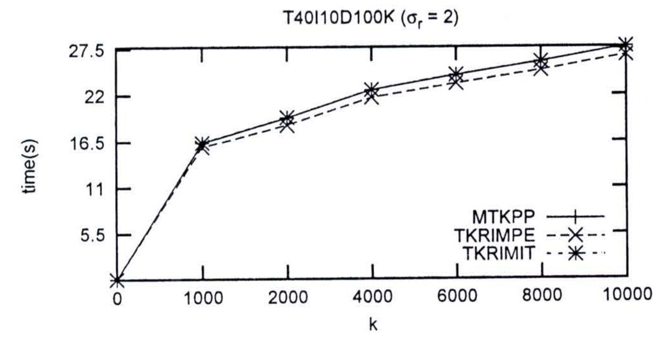


Figure 5.45: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 2\%$)

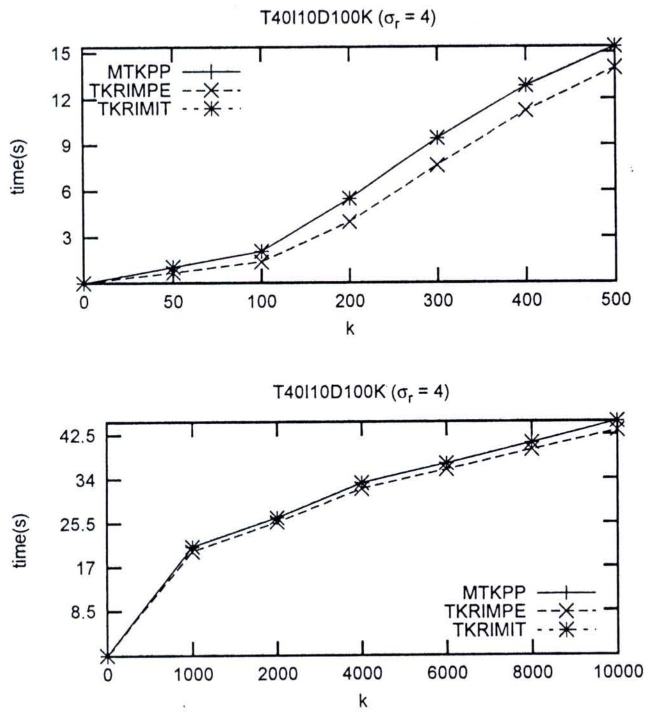


Figure 5.46: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 4\%$)

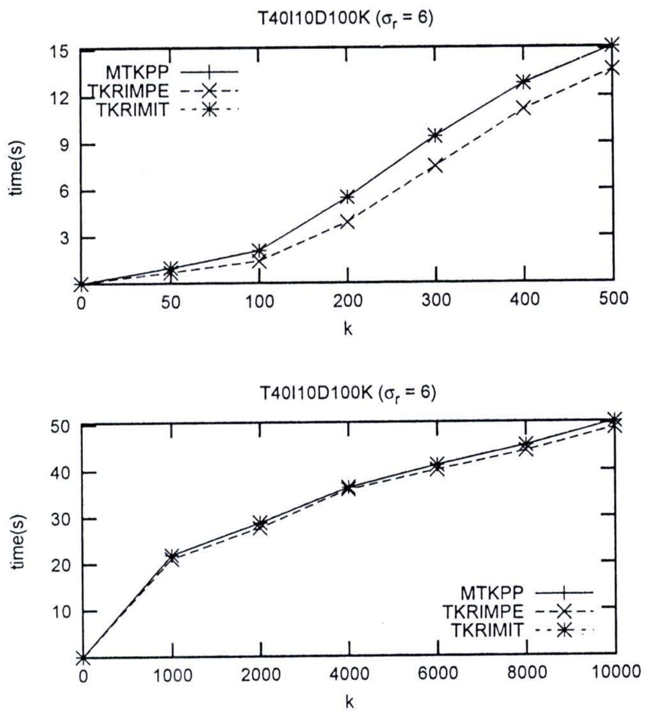
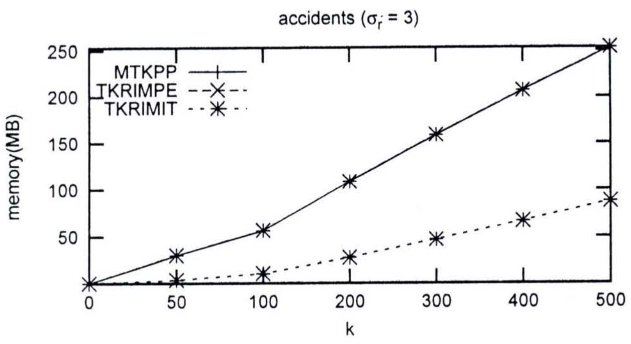


Figure 5.47: Runtime of TKRIMIT on *T40I10D100K* ($\sigma_r = 6\%$)



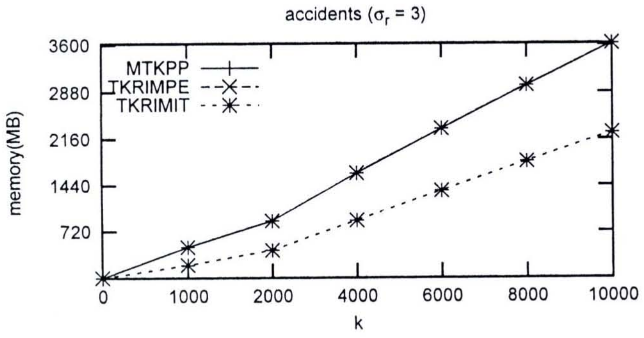
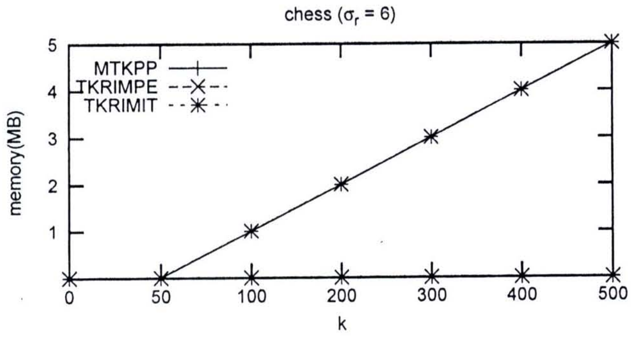


Figure 5.48: Memory usage of TKRIMIT on *accidents*



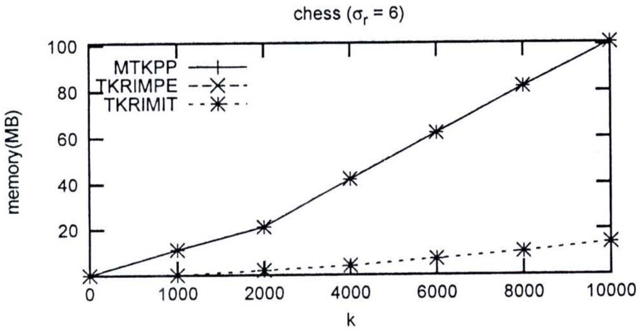
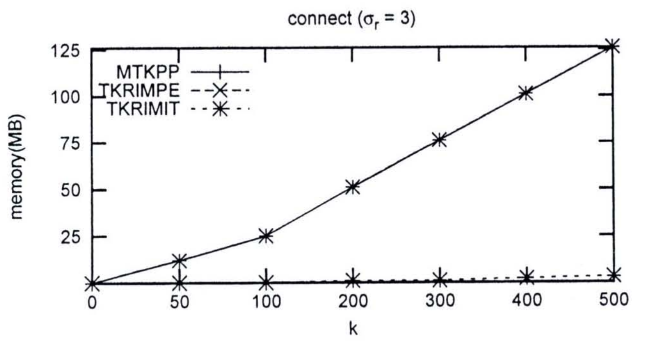


Figure 5.49: Memory usage of TKRIMIT on *chess*



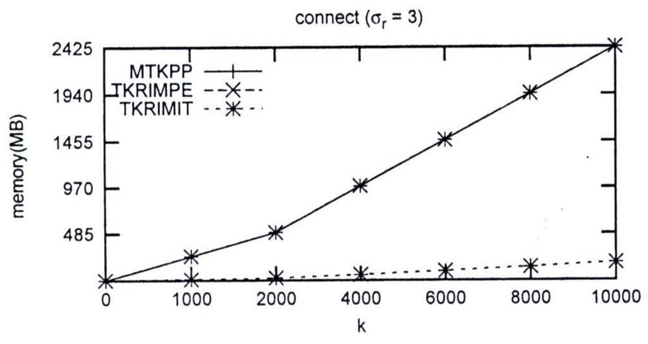
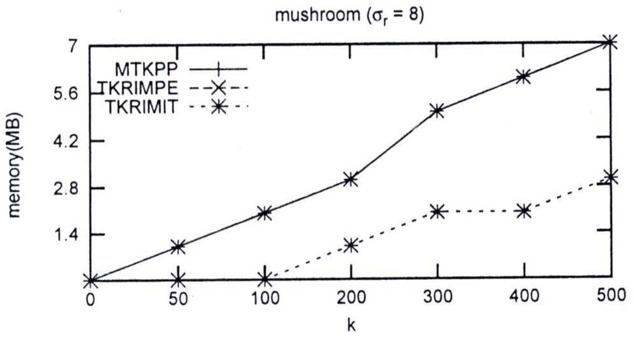


Figure 5.50: Memory usage of TKRIMIT on *connect*



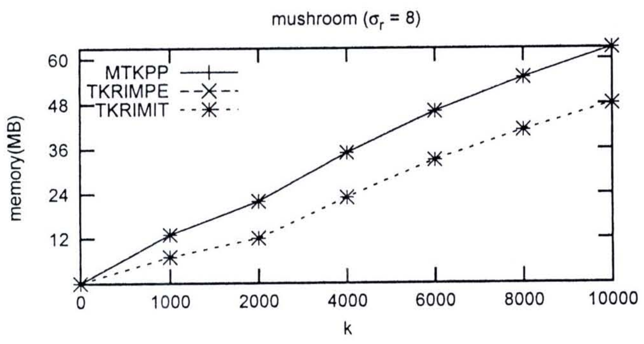


Figure 5.51: Memory usage of TKRIMIT on *mushroom*

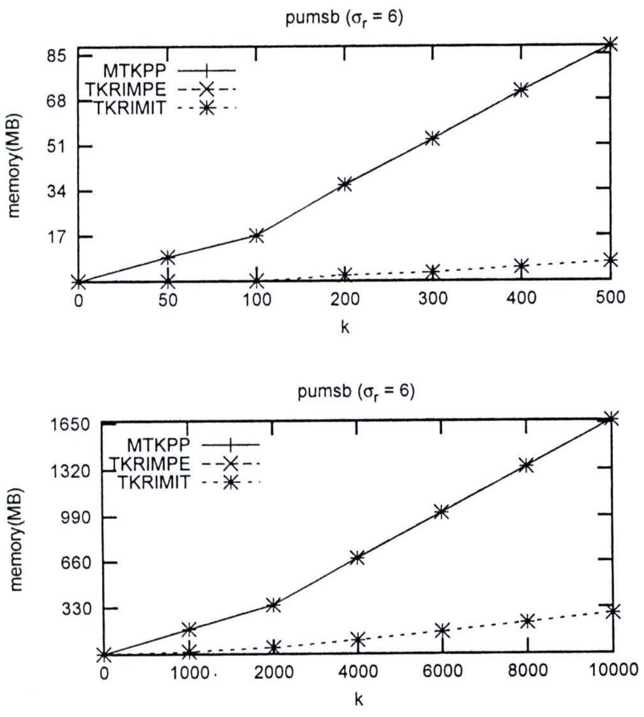


Figure 5.52: Memory usage of TKRIMIT on *pumsb*

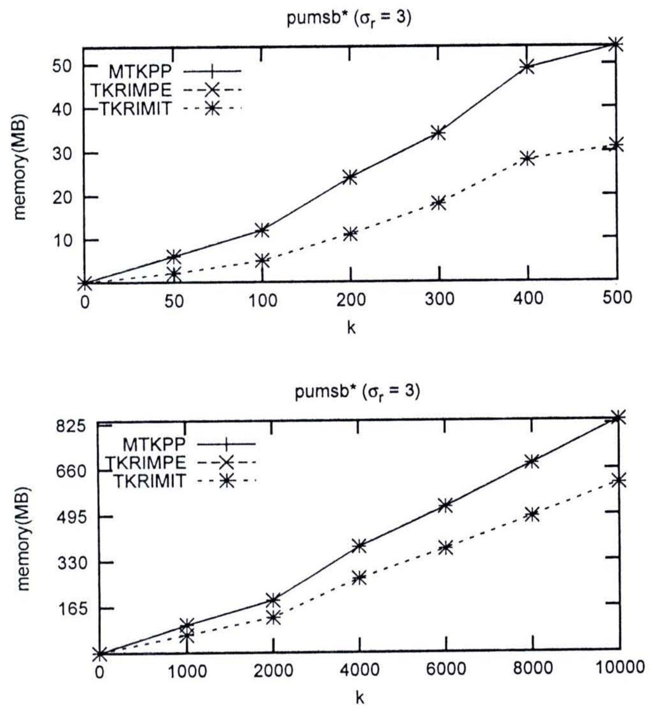
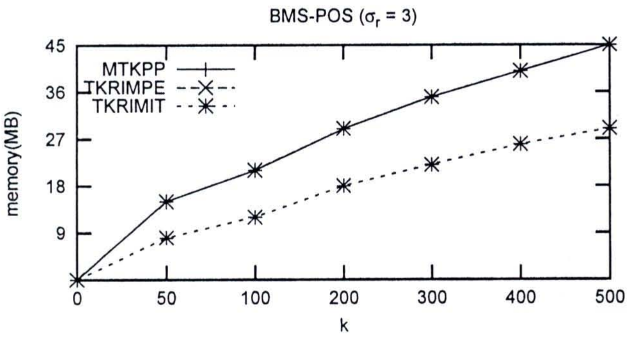


Figure 5.53: Memory usage of TKRIMIT on *pumsb**



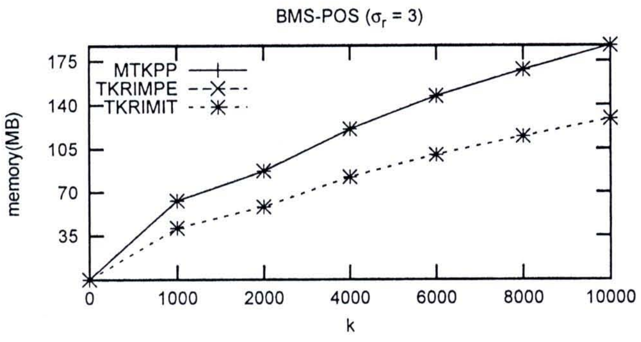
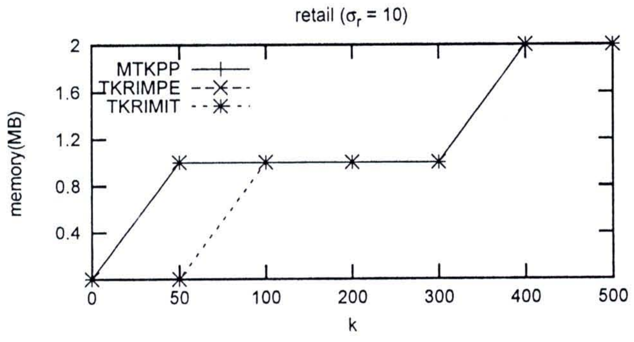


Figure 5.54: Memory usage of TKRIMIT on *BMS-POS*



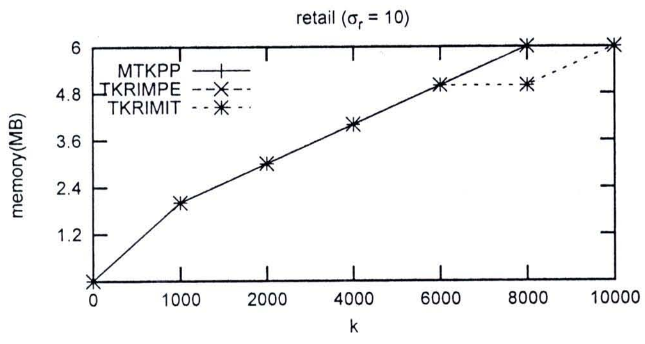
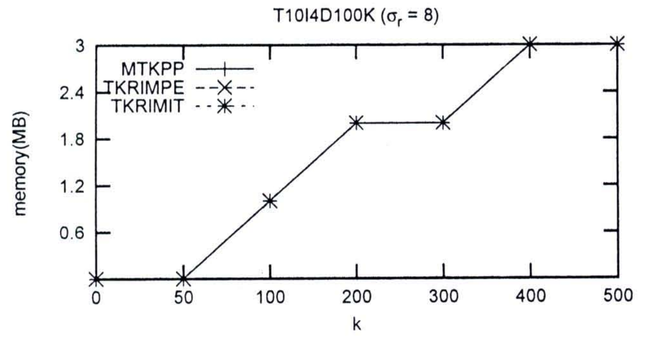


Figure 5.55: Memory usage of TKRIMIT on *retail*



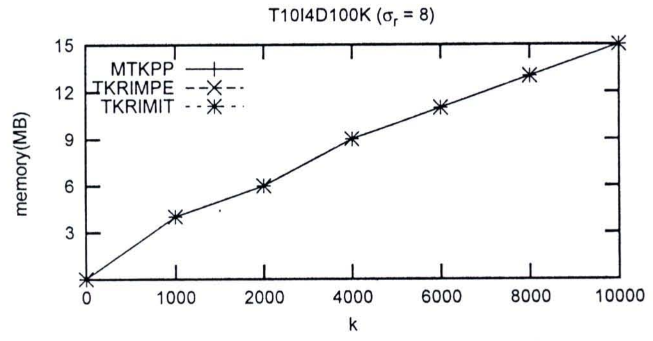
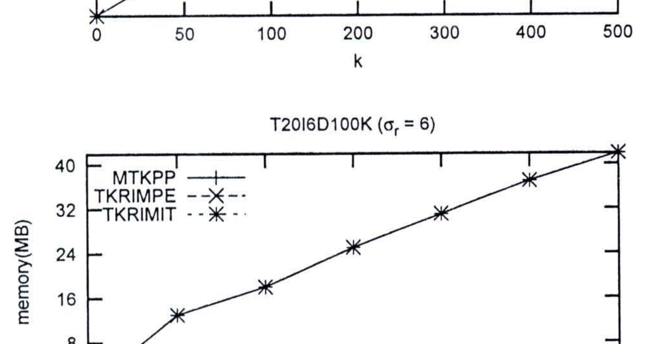


Figure 5.56: Memory usage of TKRIMIT on *T10I4D100K*



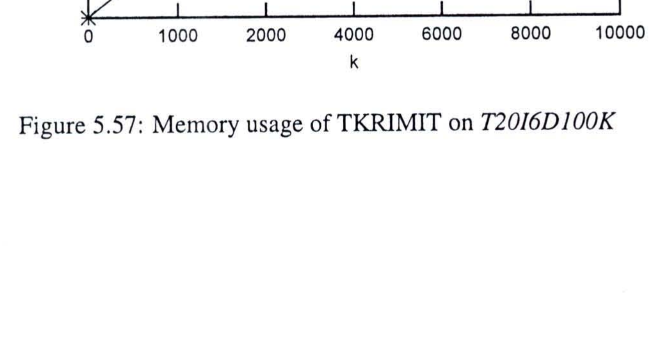
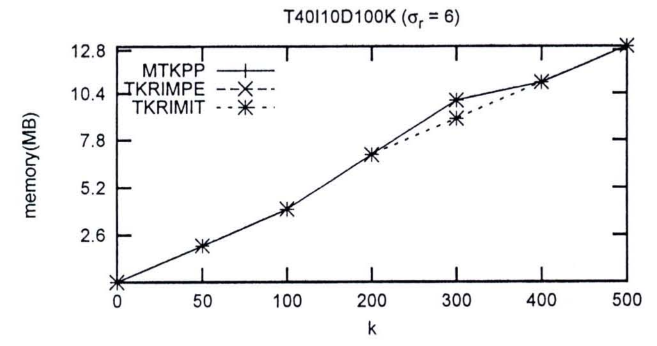


Figure 5.57: Memory usage of TKRIMIT on *T20I6D100K*



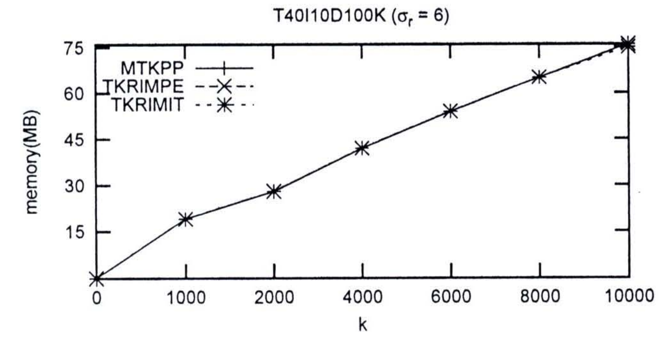
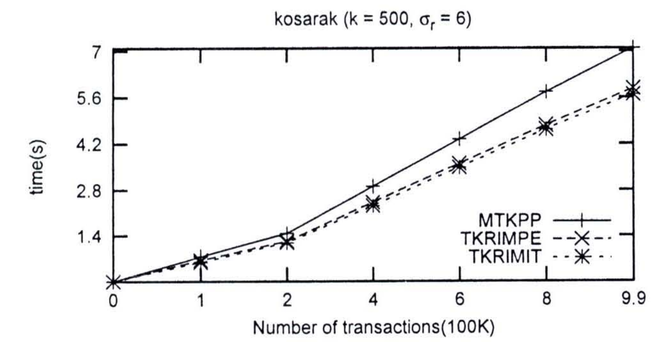


Figure 5.58: Memory usage of TKRIMIT on *T40I10D100K*



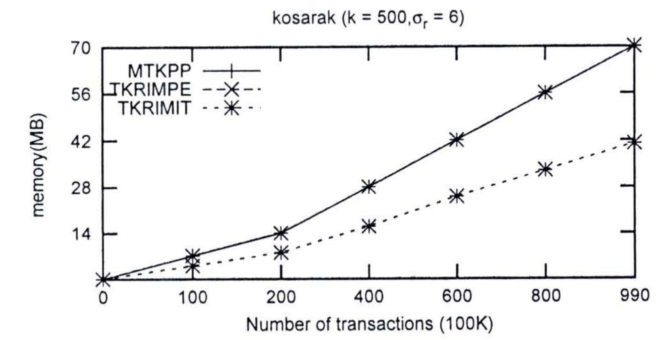
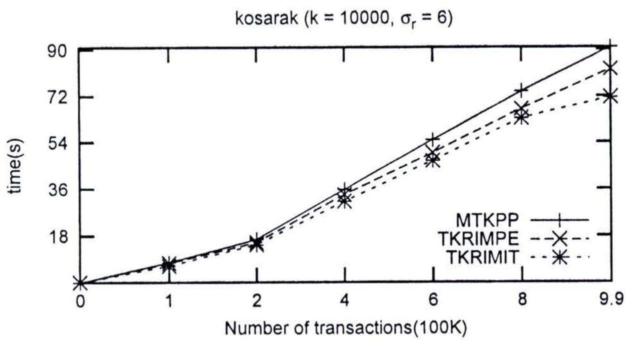


Figure 5.59: Scalability of TKRIMIT (k : 500, $\sigma_r = 6$)



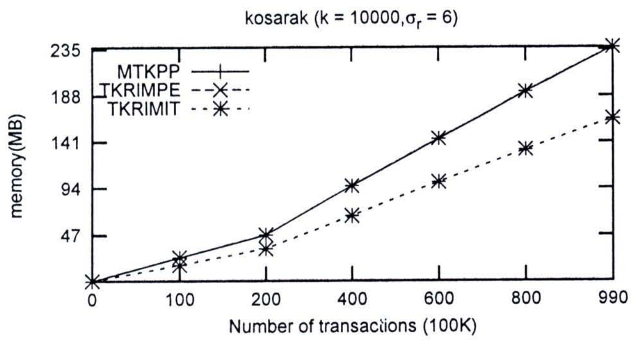


Figure 5.60: Scalability of TKRIMIT (k : $10,000, \sigma_r = 6$)