CHAPTER VI

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

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

6.1 Preliminary of H-TKRIMP

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

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

As previous algorithms, H-TKRIMP is also based on the use of a top-*k* list structure which is a simple linked-list with a hash table. The top-*k* list is used to maintain a set of k (or less than k) regular itemsets with the highest supports and their occurrence information during mining process. Meanwhile, the hash table is utilized to quickly access all the information of each itemset in the top-k list. As shown in Figure 6.1, each entry in the top-k list consists of 4 fields: item or itemset name (I), total support (s^I), regularity (r^I), and a set of tidsets ($T^I = \{T_1^I, \ldots, T_{pn}^I\}$) where pn is the number of partitions of considered database). From the figure, the item a has a support of 11, a regularity of 2, and tidsets as $\{\{1,-3\},\{6,-2\},\{9,-3\}\}$ which means the item a occurs in transactions $\{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$.

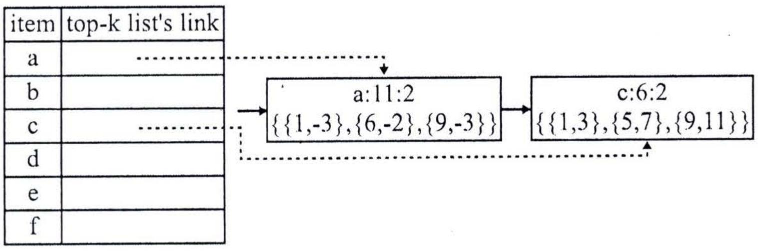


Figure 6.1: H-TKRIMP: Top-k list structure with hash table

6.3 Database Partitioning

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

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

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

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

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

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

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

By combining the partitioning technique with the hybrid representation together, H-TKRIMP can use two representations (i.e. normal tidset and interval tidset) to maintain tids of each partition. Then, $T^X = \{T^X_1, \ldots, T^X_{pn}\}$ is defined as the (global) tidset of an itemset X.

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

For example, consider an item a occurring in tids $\{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12\}$ (i.e. transactions $T^a = \{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$) from the transactional database of Table 6.1. Thus, based on the partitioning technique, the tidset of the first partition T^a_1 contains the set of tids $\{1, 2, 3, 4\}$ where the item a occurs. Meanwhile, the set of tids $\{6, 7, 8\}$ and $\{9, 10, 11, 12\}$ are maintained in T^a_2 and T^a_3, respectively. Thus, the (global) tidset of a is $T^a = \{\{1, 2, 3, 4\}, \{6, 7, 8\}, \{9, 10, 11, 12\}\}$. Besides, the support of a is $s^a = 4 + 3 + 4 = 11$.

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

6.4 Hybrid representation

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

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

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

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

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

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

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

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

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

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

$$ndt^X_m = \begin{cases} |T^X_m| - |TN^X_m| - 1 & \textit{if}\, t^X_{|T^X_m|,m} > 0 \\ |T^X_m| - |TN^X_m| & \textit{if}\, t^X_{|T^X_m|,m} < 0 \end{cases}$$

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

6.5 Calculation of Regularity and Support

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

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

$$rtt^X_{q,m} = \begin{cases} t^X_{q,m} & \textit{if}\, q = 1 \\ t^X_{q,m} - t^X_{p,m} & \textit{if}\, 2 \leq q \leq |T^X_m| \end{cases}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.6 H-TKRIMP algorithm

Based on the database partitioning and the hybrid representation mentioned above, the H-TKRIMP algorithm is also described. H-TKRIMP consists of two steps : (*i*) Top-*k* list initialization: partition the database, scan each partition to obtain top-*k* regular items and then transform each tidset into the suitable tidset (a normal tidset or an interval tidset); (*ii*) Top-*k* mining: merge (with two constraints) each pair of elements in the top-*k* list to produce new larger itemsets by using the best-first search strategy, sequentially intersect their tidsets (one by one partition) to find the k regular itemsets with the highest supports, and then transform each tidset of the new generated itemset into the proper representation.

6.6.1 H-TKRIMP: Top-*k* initialization

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

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

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

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

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

```
  (1) A transaction database: $TDB$
  (2) A number of itemsets to be mined: $k$
  (3) A regularity threshold: $\sigma_r$
Output:
  (1) A top-k list

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

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

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

6.6.2 H-TKRIMP: Top-*k* mining

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

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

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

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

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

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

  (*4*) if $t_{i,m}^X, t_{j,m}^X < 0$, add $t_{|T^{XY}|,m}^{XY} - (t_{i-1,m}^X - t_{i,m}^X)$ at the end of T_m^{XY} if $t_{i-1,m}^X - t_{i,m}^X < t_{j-1,m}^Y - t_{j,m}^Y$ otherwise add $t_{|T^{XY}|,m}^{XY} - (t_{j-1,m}^Y - t_{j,m}^Y)$ at the end of T^{XY}

- **One of them is a normal tidset and the another one is an interval tidset.** The conditions from the second case are applied with some different details; for example, T_m^X is a normal tidset and T_m^Y is an interval tidset.

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

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

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

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

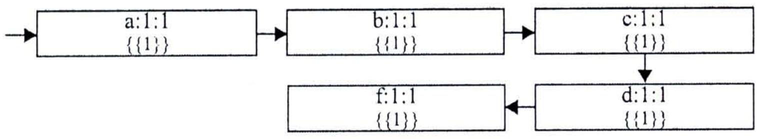
Based on the hybrid representation, H-TKRIMP can reduce time to intersect tidsets from TKRIMPE by reducing a number of tids in the dense tidsets. Moreover, H-TKRIMP can reduce time in the intersect process from TKRIMIT on the sparse tidsets by reducing the time to investigate each tid in the interval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

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

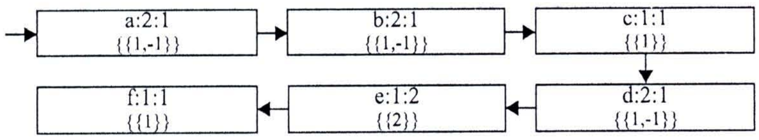
```
Input: top-k list, $\sigma_r$, $k$
Output: top-k regular-frequent itemsets
  for each entry $x$ in the top-k list do
    for each entry $y$ in the top-k list ($x > y$) do
      if the entries $x$ and $y$ have the same size of itemsets and the same prefix then
        merge the itemsets of $x$ and $y$ to be the itemset $Z = I^x \cup I^y$
        for each partition $m = 1$ to $pn$ do
          for each $t_p$ in $T_m^X$ ($p = 1$ to $|T_m^X|$) and $t_q$ in $T_m^Y$ ($q = 1$ to $|T_m^Y|$) do
            if $t_p > 0$ and $t_q > 0$ then
              if $t_p = t_q$ then
                calculate the regularity $r^Z$ by $t_p$ and check $r^Z$ with $\sigma_r$
                add the support $s_m^Z$ by 1
                collect $t_p$ as the last tid in $T_m^Z$
            else if $t_p > 0$ and $t_q < 0$ then
              if $t_p \leq t_{q-1} - t_q$ then
                calculate the regularity $r^Z$ by $t_p$ and check $r^Z$ with $\sigma_r$
                add the support $s_m^Z$ by 1
                collect $t_p$ as the last tid in $T_m^Z$
            else if $t_p < 0$ and $t_q > 0$ then
              if $t_{p-1} - t_p \geq t_q$ then
                calculate the regularity $r^Z$ by $t_q$ and check $r^Z$ with $\sigma_r$
                add the support $s_m^Z$ by 1
                collect $t_q$ as the last tid in $T_m^Z$
            else
              if $t_{p-1} - t_p > t_{q-1} - t_q$ then
                add the support $s_m^Z$ by $(t_{q-1} - t_q) - t_{|T_m^Z|}^Z$
                collect $t_{|T_m^Z|}^Z - (t_{q-1} - t_q)$ as the last tid in $T_m^Z$
              else
                add the support $s_m^Z$ by $(t_{p-1} - t_p) - t_{|T_m^Z|}^Z$
                collect $t_{|T_m^Z|}^Z - (t_{p-1} - t_p)$ as the last tid in $T_m^Z$
          add the support $s^Z$ by $s_m^Z$
          if $nrt_m^X < ndt_m^X$ then
            transform $T_m^Z$ to be a normal tidset // not contain $tid < 0$

        calculate the regularity $r^Z$ by $|TDB|-$ the last tid of $T_{pn}^Z$
        if $r^Z \leq \sigma_r$ and $s^Z \geq s_k$ then
          insert the itemset $Z$ ($I^x \cup I^y$) into the top-k list with $r^Z$, $s^Z$ and $T^Z$
          remove the $k^{th}$ entry from the top-k list
```

6.7 Example of H-TKRIMP

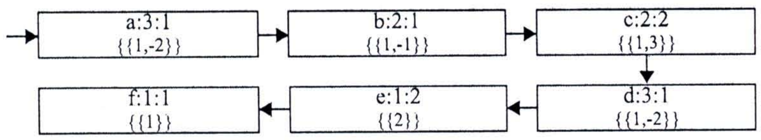
Consider the TDB of Table 6.1, the regularity threshold σ_r of 4 and the number of desired results k of 5. The database is separated into three partitions. Then, the process of initializing top-k list from the TDB of Table 6.1 is illustrated in Figure 6.2.



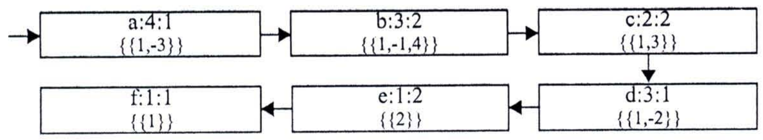
(a) read first transaction



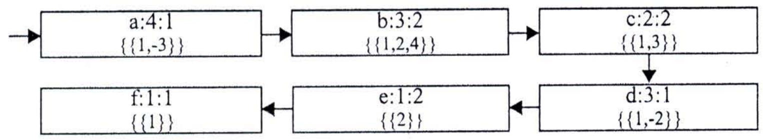
(b) read second transaction



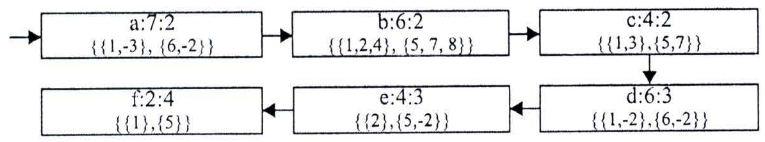
(c) read third transaction



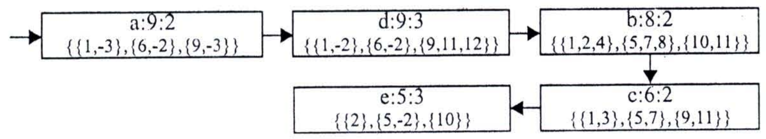
(d) read first partition



(e) converse tidset to be normal tidset



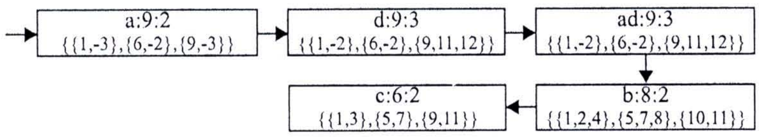
(f) read second partition



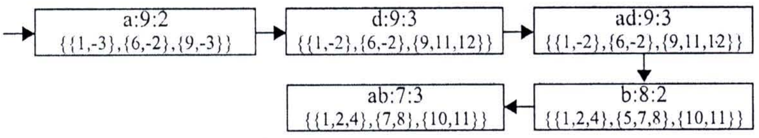
(g) Final Top-k list in initialization process

Figure 6.2: Top-k list initialization

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



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



(b) final top-*k* list

Figure 6.3: Top-*k* during mining process

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

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

6.8 Complexity analysis

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

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

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

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

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

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

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

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

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

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

6.9 Performance evaluation

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

6.9.1 Experimental setup

The experiments of H-TKRIMP are done on three synthetic datasets (T10I4D100K, T20I6D100K and T20I6D100K) and nine real datasets (accidents, BMS-POS, chess, connect, kosarak, mushroom, pumsb, pumsb* and retail) which were described their details and characteristics in Chapter 2. Program for H-TKRIMP is written in C in the same manner as the previous algorithms: MTKPP, TKRIMPE and TKRIMIT using a top-*k* list. All experiments are performed on a Linux platform with a Intel®Xeon 2.33 GHz and with 4 GB main memory.

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

6.9.2 Execution time

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

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

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

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

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

6.9.3 Memory consumption

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

Figure 6.37 to Figure 6.42 show the memory usage of H-TKRIMP compared to other proposed algorithms on dense datasets. It can be seen from the figures that the memory usage of H-TKRIMP increases as the value of k increases and H-TKRIMP consumes the same size of memory as TKRIMIT in the most cases. From the figures, H-TKRIMP can down size memory usage from MTKPP and TKRIMPE with the large value of k because the advantage of using interval tidset representation. However, in some cases, H-TKRIMP uses more memory than that of TKRIMIT because it has to convert some tidsets that have too few tids to be normal tidset (*i.e.* based on the use of hybrid representation). By this way of doing, H-TKRIMP can save computational time from the use of single representation (only normal tidset or interval tidset representation) but it takes little more memory than TKRIMIT using only interval tidset representation. Meanwhile, the memory usage on sparse datasets are illustrated in Figure 6.43 to Figure 6.47. From these figures, the memory usage of the four proposed algorithms are similar due to the fact that each itemset does not occur in consecutive continuous tids. Therefore, H-TKRIMP and TKRIMIT cannot take the advantage from the interval tidset representation. However, from the results, it can be seen that based on the used of the top-*k* list structure and maintaining tidset, the memory usage of H-TKRIMP is efficient for the top-*k* regular-frequent itemsets mining using the recently available gigabyte range memory.

6.9.4 Scalability test

In this experiment, two primary factors, the scalability of execution time and memory usage, are examined. The kosarak dataset which is a huge dataset with a large number of distinct items $(41,270)$ and transactions $(990,002)$ is used. To test the scalability with the varied number of transactions, the database is first divided into six portions. Each portion contains: $100K$, $200K$, $400K$, $600K$, $800K$ and $990K$ transactions, respectively.The value of k (i.e. the number of itemsets to be mined) is specified to 500 and $10,00C$ (*i.e.* each is the instance of the small and the large values of k), and the regularity threshold is set to 6% of the number of transactions in each portion.

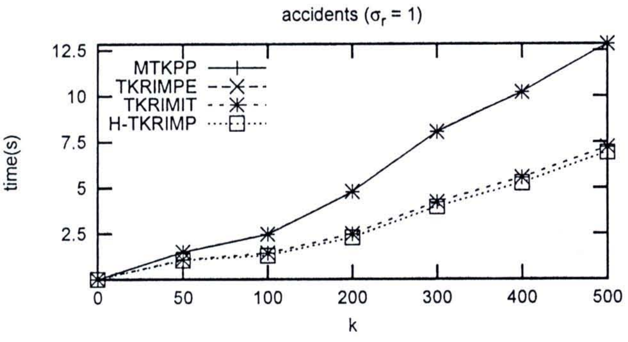
As shown in the plots in Figures 6.48 and 6.49, H-TKRIMP outperforms the three competitors in all the tests conducted. All the execution time linearly grows as the dataset size increases from $100K$ to $990K$. For the large values of k, H-TKRIMP is much more scalable than the others due to the fact that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

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

6.10 Summary

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

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



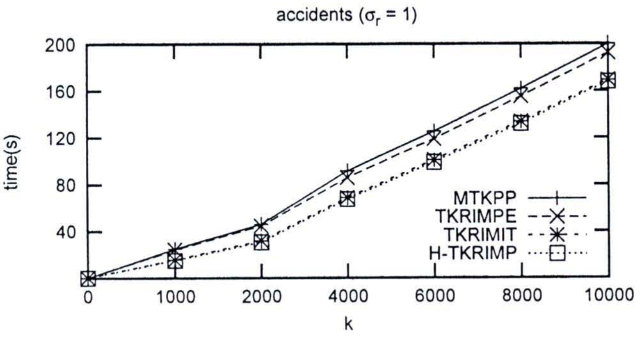
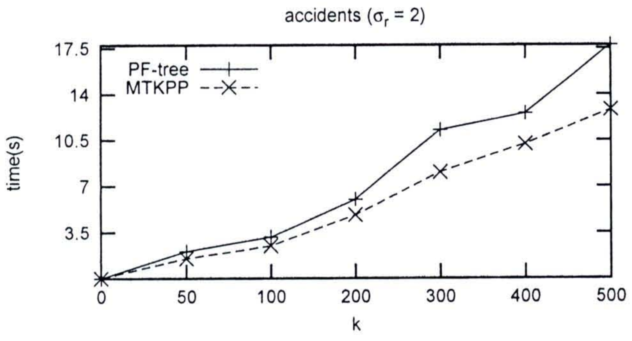


Figure 6.4: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 1\%$)



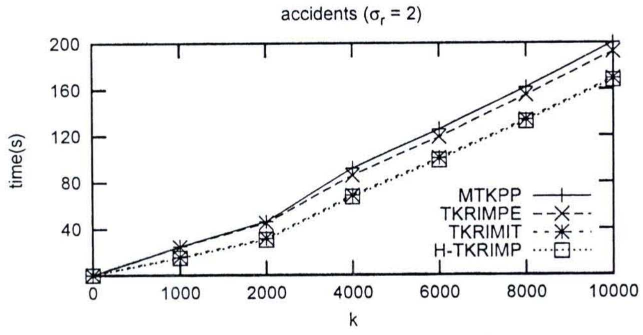


Figure 6.5: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 2\%$)

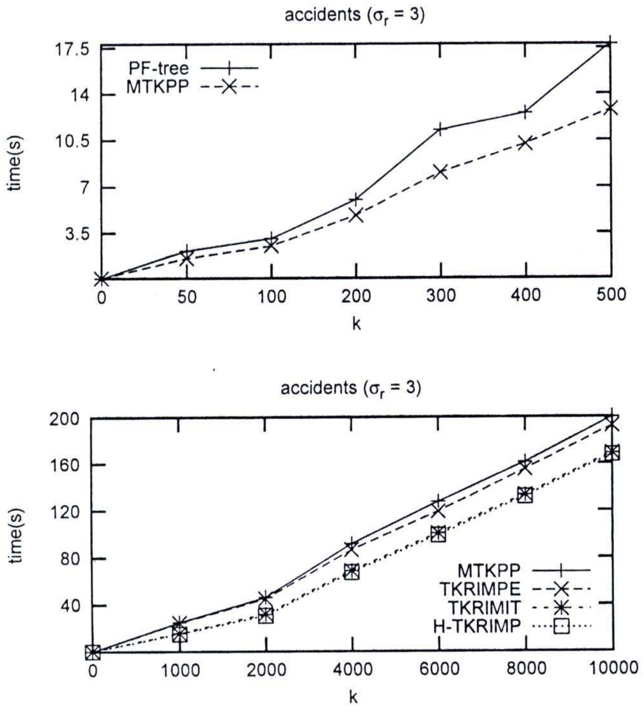


Figure 6.6: Runtime of H-TKRIMP on *accidents* ($\sigma_r = 3\%$)

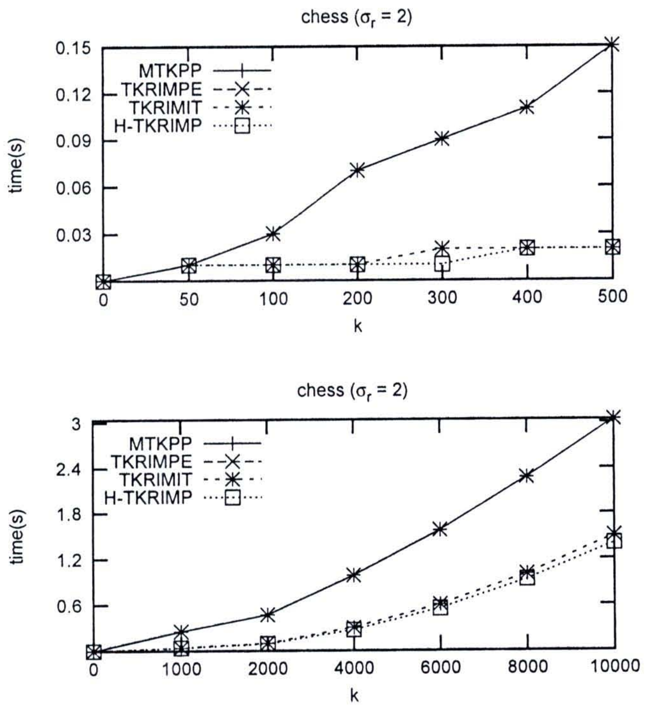
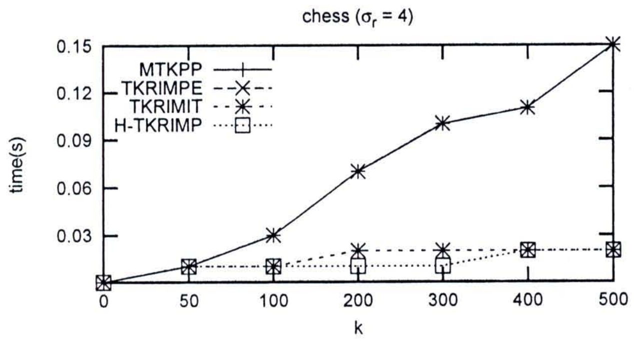


Figure 6.7: Runtime of H-TKRIMP on *chess* ($\sigma_r = 2\%$)



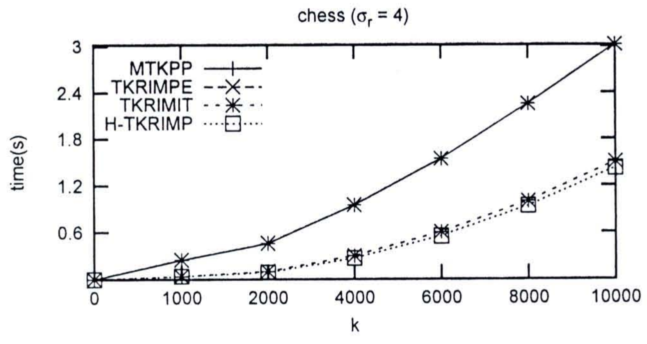
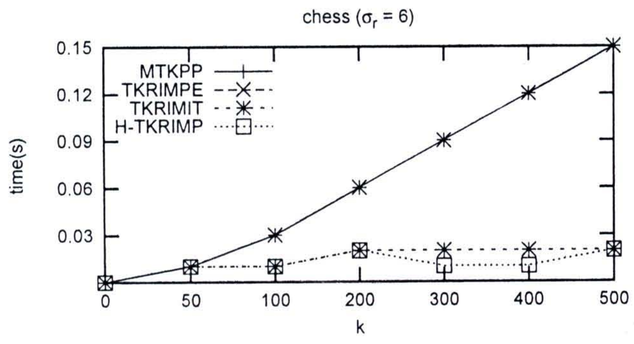


Figure 6.8: Runtime of H-TKRIMP on *chess* ($\sigma_r = 4\%$)



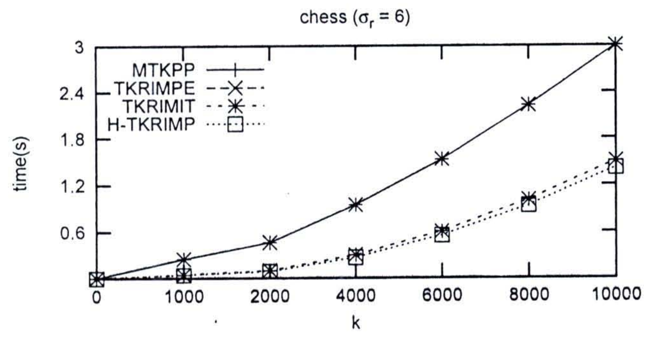


Figure 6.9: Runtime of H-TKRIMP on *chess* ($\sigma_r = 6\%$)

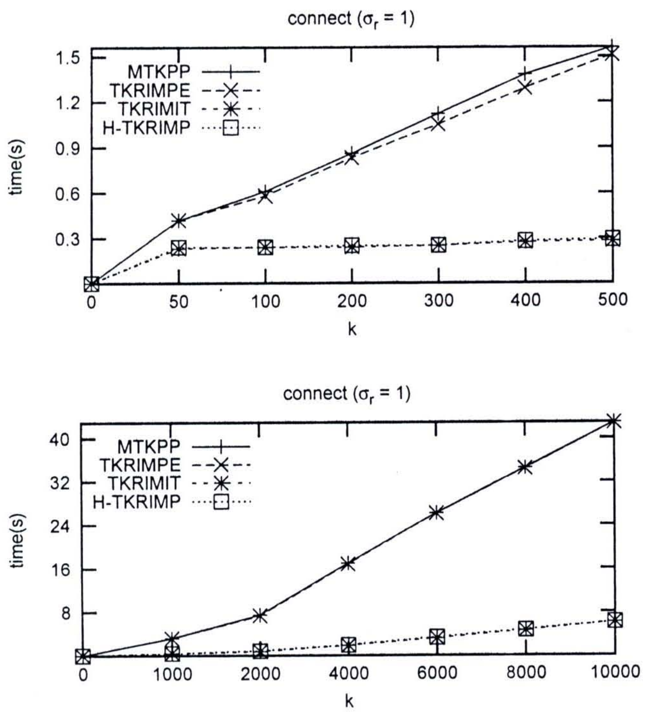


Figure 6.10: Runtime of H-TKRIMP on *connect* ($\sigma_r = 1\%$)

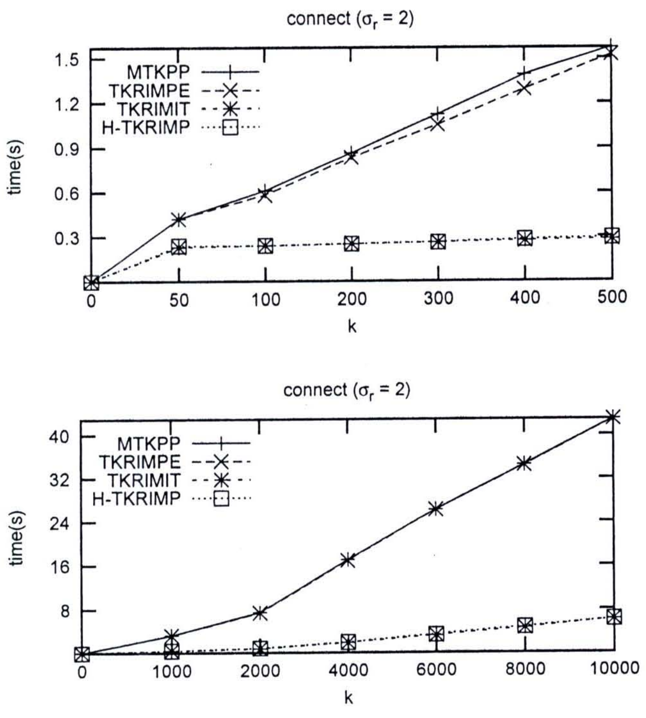
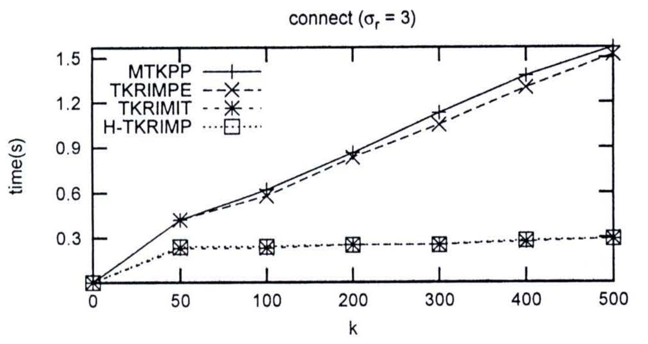


Figure 6.11: Runtime of H-TKRIMP on *connect* ($\sigma_r = 2\%$)



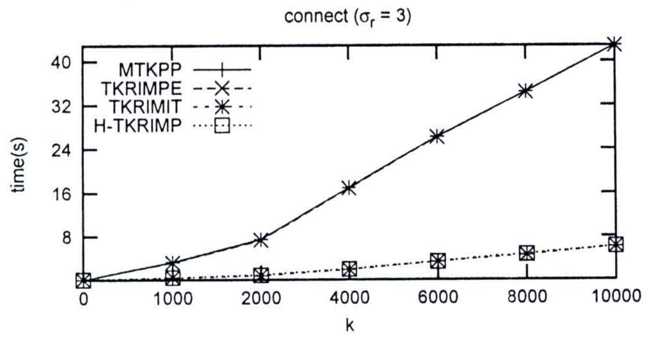
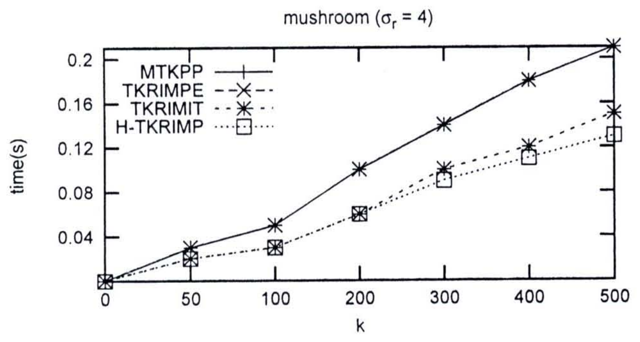


Figure 6.12: Runtime of H-TKRIMP on *connect* ($\sigma_r = 3\%$)



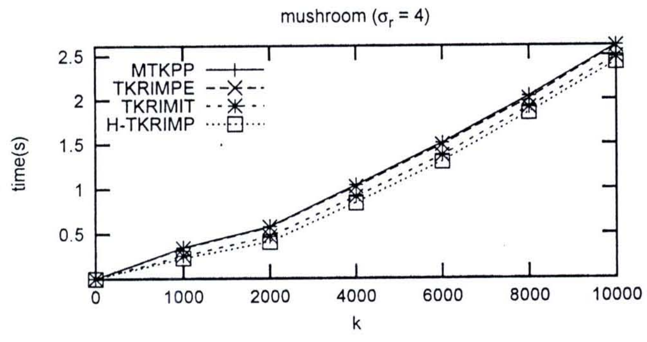


Figure 6.13: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 4\%$)

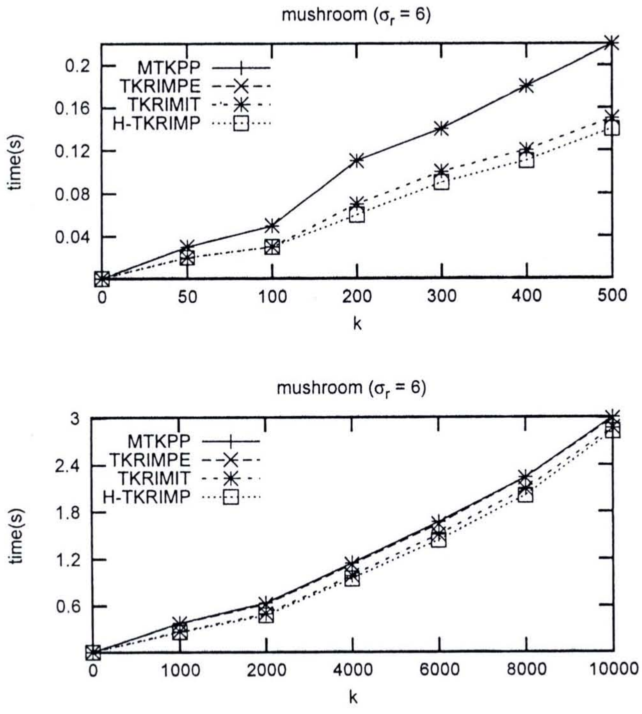


Figure 6.14: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 6\%$)

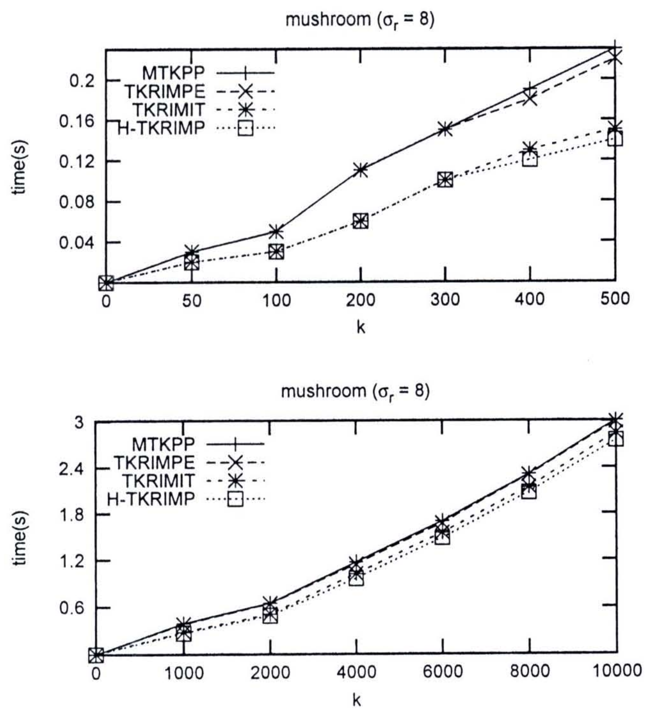


Figure 6.15: Runtime of H-TKRIMP on *mushroom* ($\sigma_r = 8\%$)

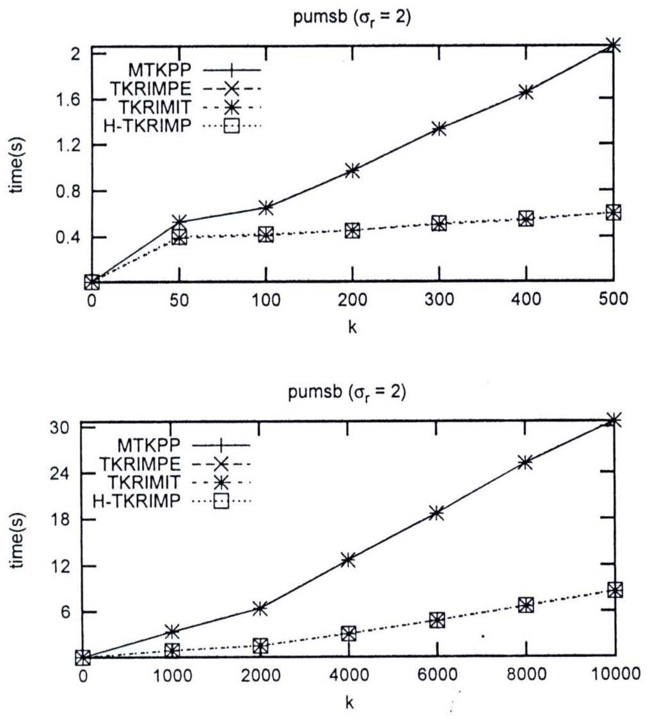


Figure 6.16: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 2\%$)

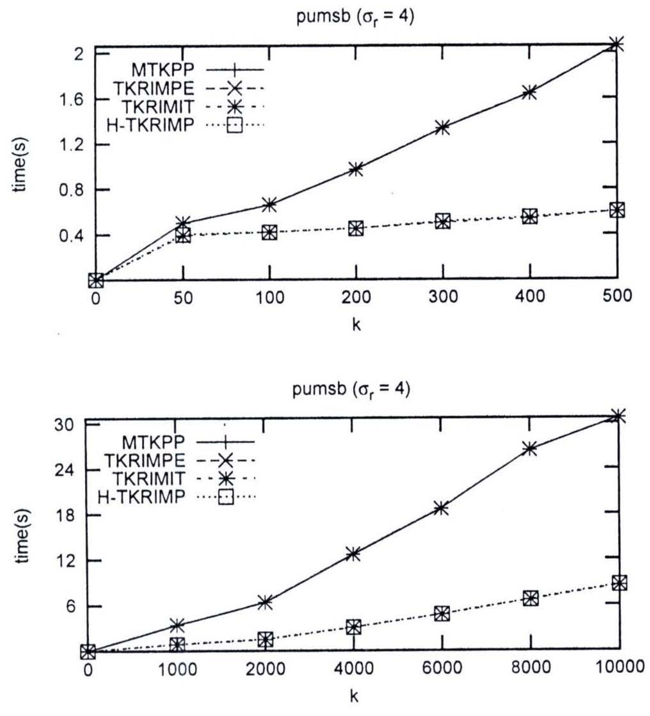


Figure 6.17: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 4\%$)

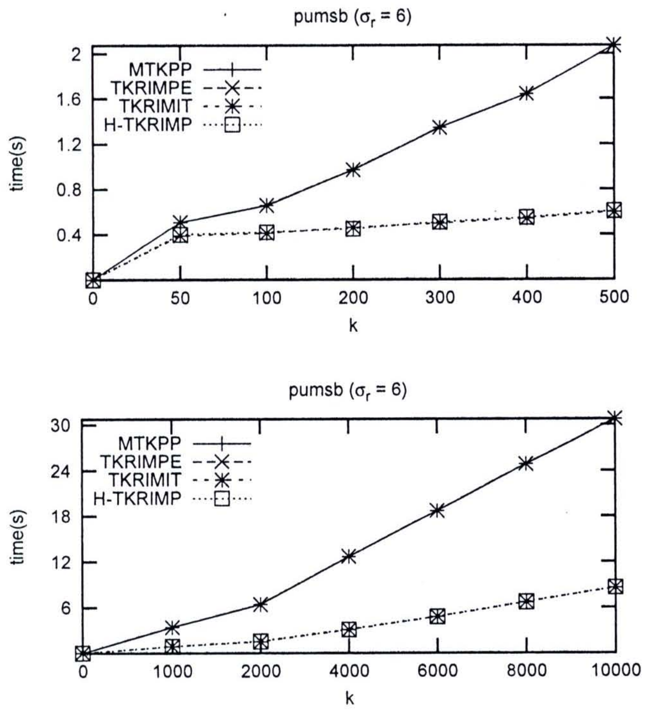


Figure 6.18: Runtime of H-TKRIMP on *pumsb* ($\sigma_r = 6\%$)

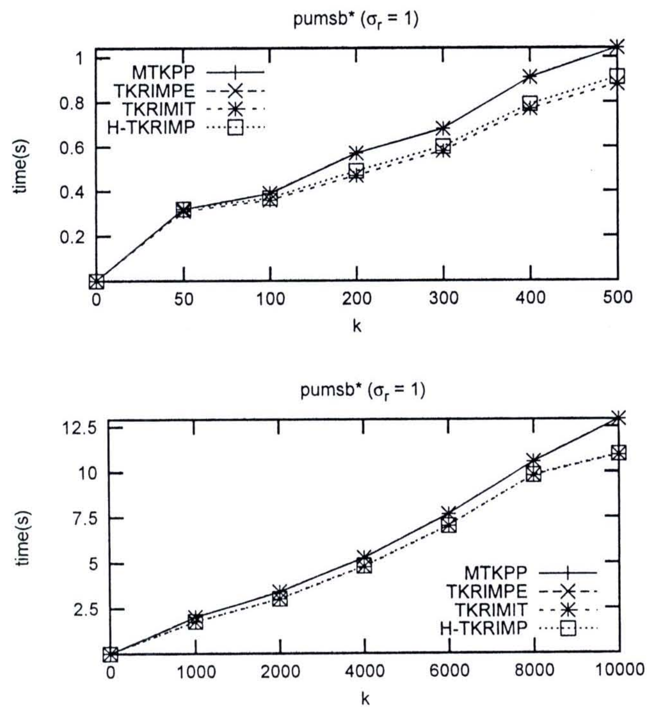
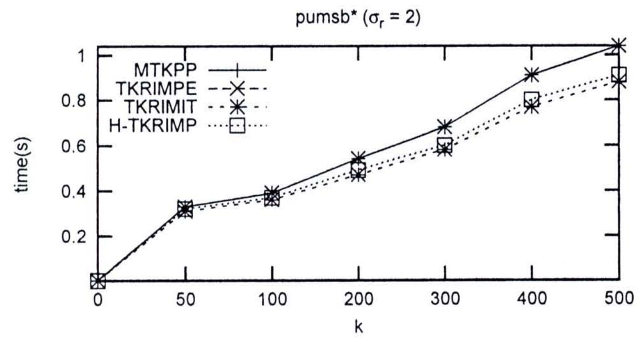


Figure 6.19: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 1\%$)



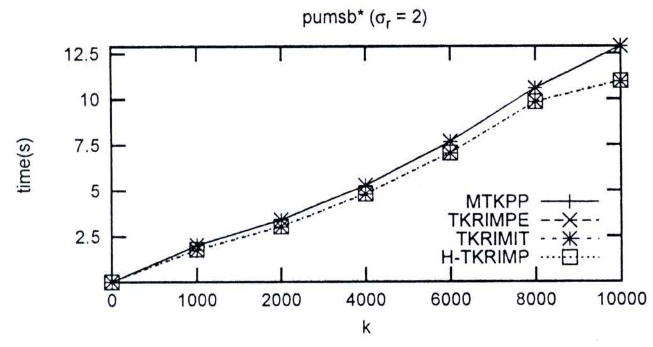
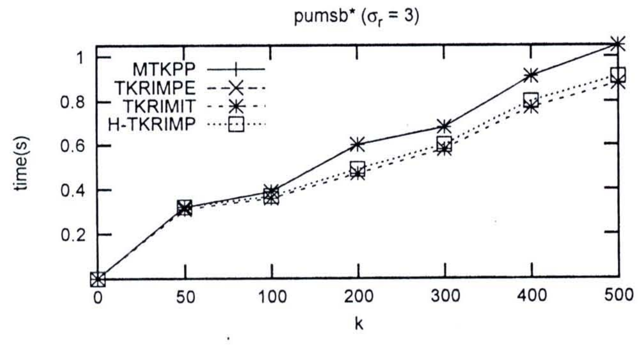


Figure 6.20: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 2\%$)



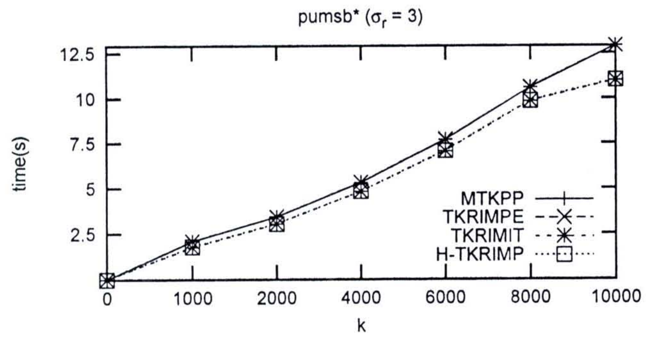


Figure 6.21: Runtime of H-TKRIMP on *pumsb** ($\sigma_r = 3\%$)

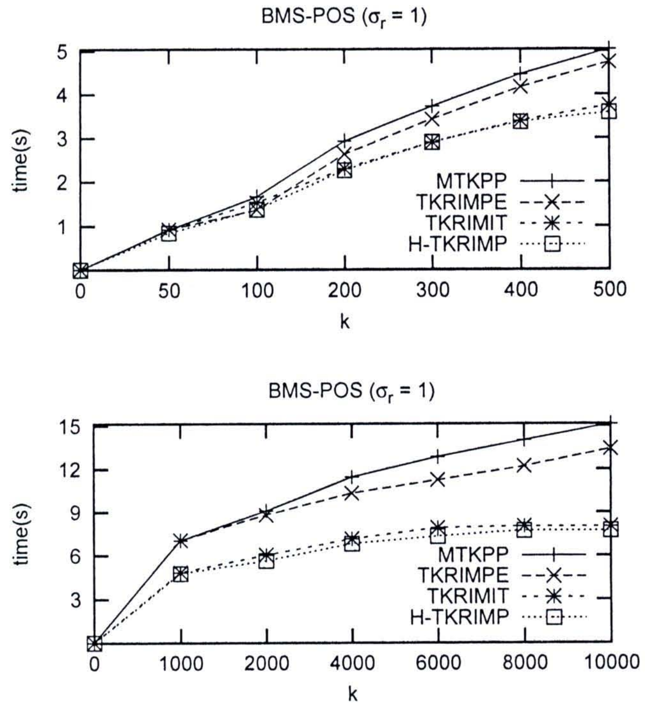


Figure 6.22: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 1\%$)

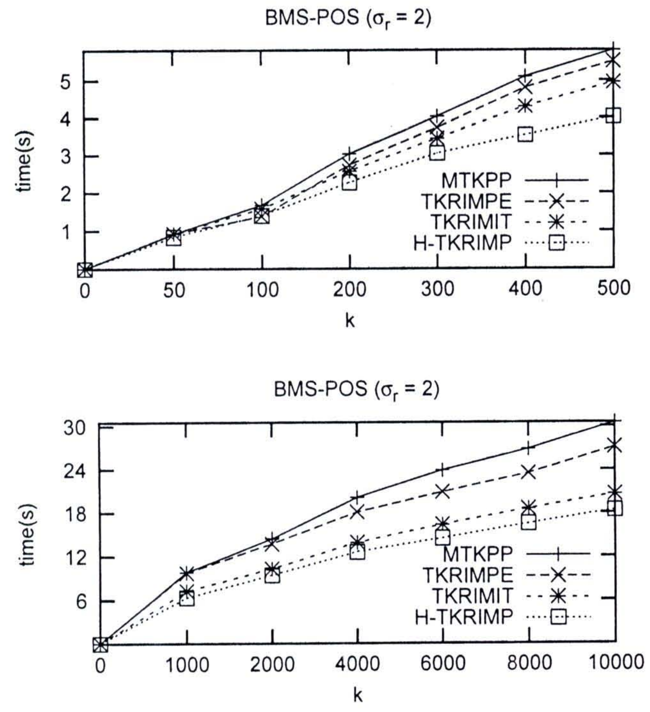
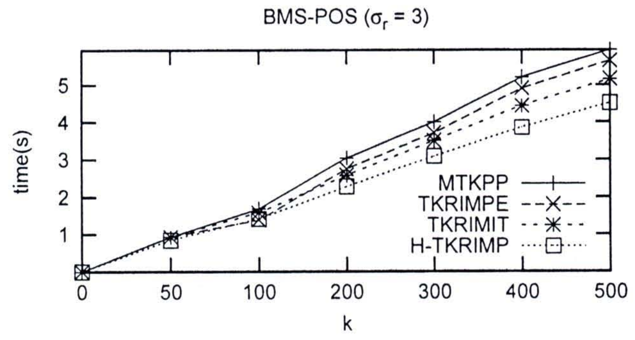


Figure 6.23: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 2\%$)



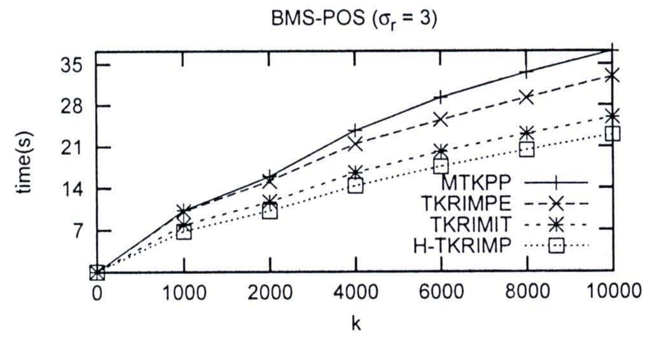
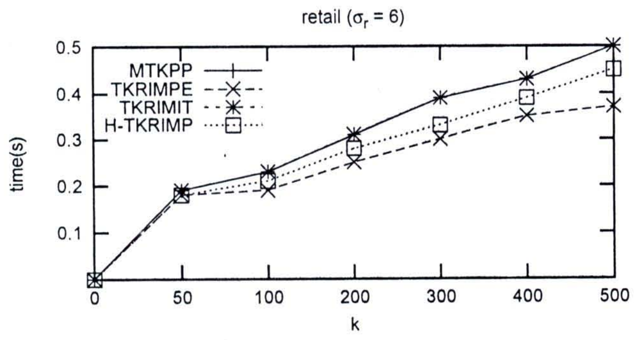


Figure 6.24: Runtime of H-TKRIMP on *BMS-POS* ($\sigma_r = 3\%$)



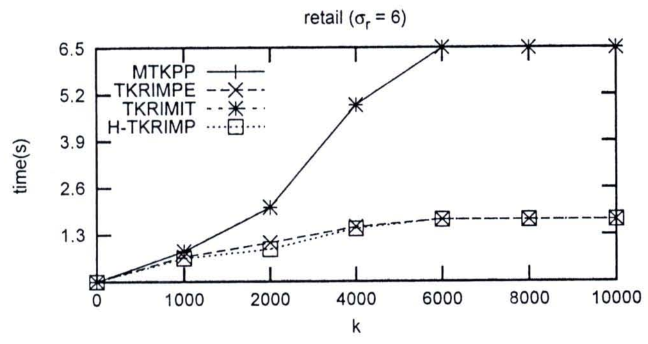


Figure 6.25: Runtime of H-TKRIMP on *retail* ($\sigma_r = 6\%$)

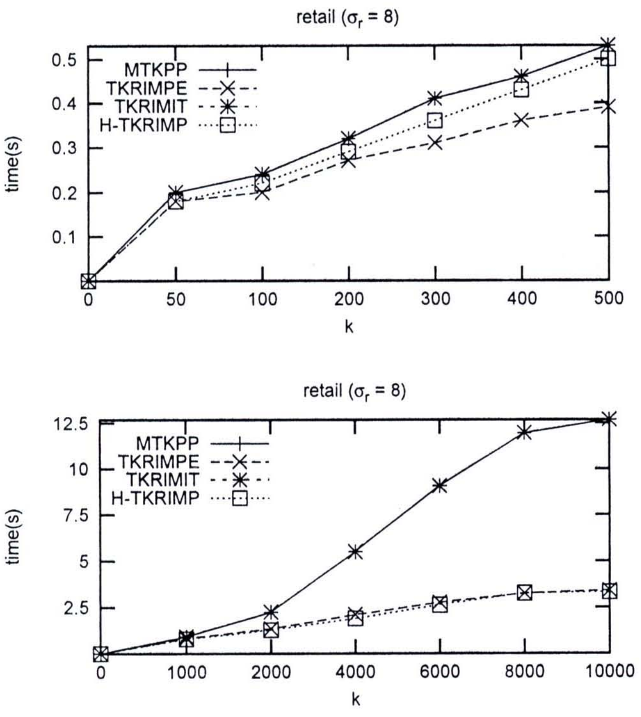


Figure 6.26: Runtime of H-TKRIMP on *retail* ($\sigma_r = 8\%$)

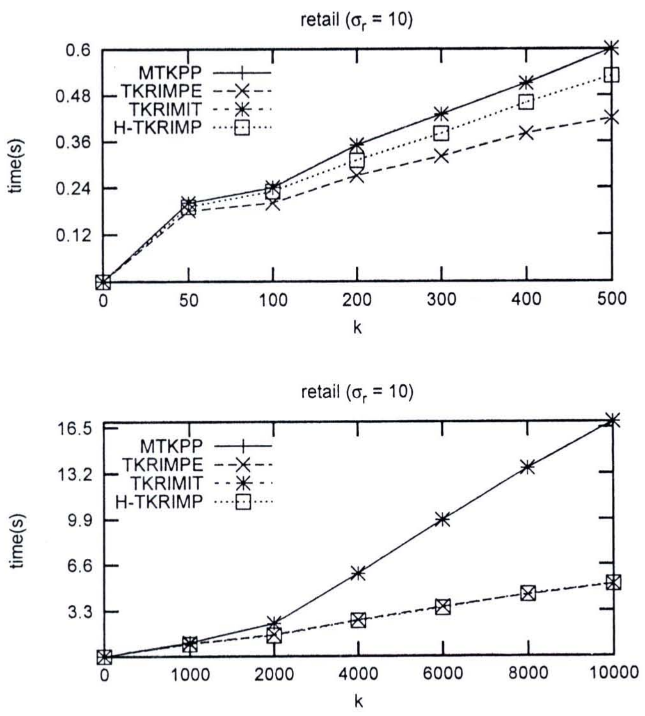


Figure 6.27: Runtime of H-TKRIMP on *retail* ($\sigma_r = 10\%$)

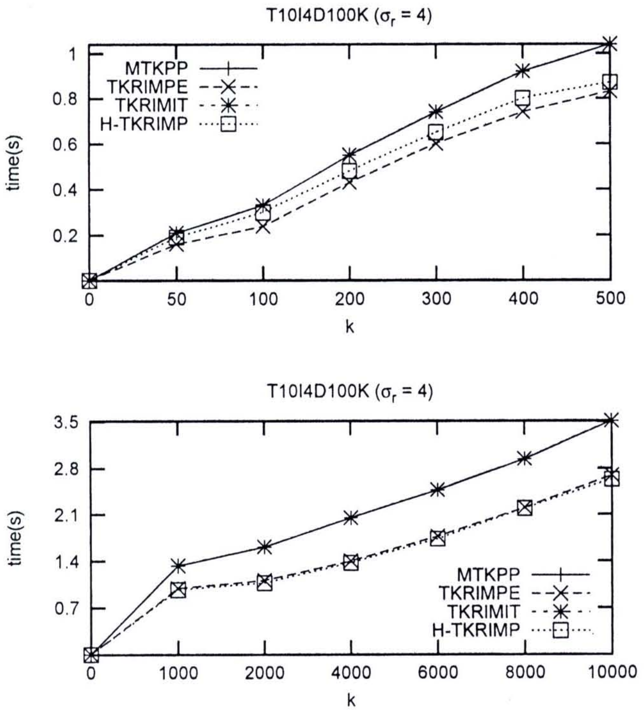


Figure 6.28: Runtime of H-TKRIMP on *T10I4D100K* (σ_r = 4%)

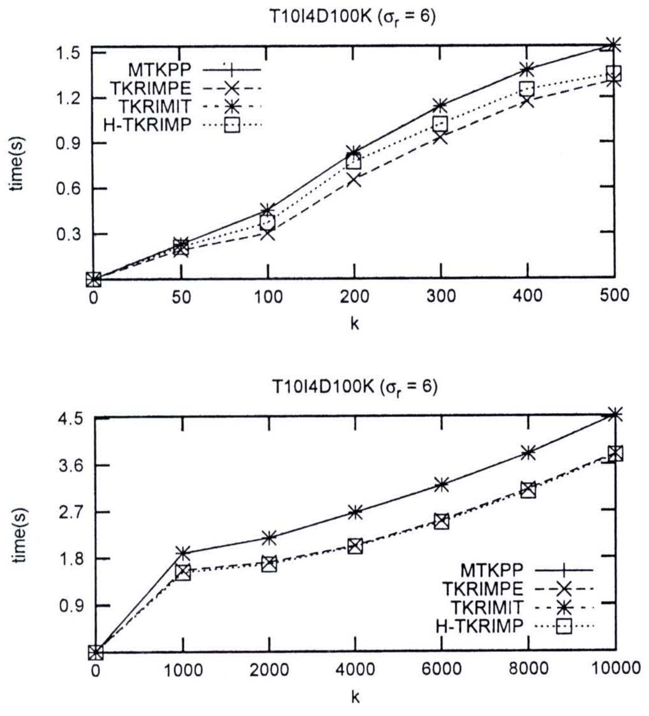


Figure 6.29: Runtime of H-TKRIMP on *T10I4D100K* (σ_r = 6%)

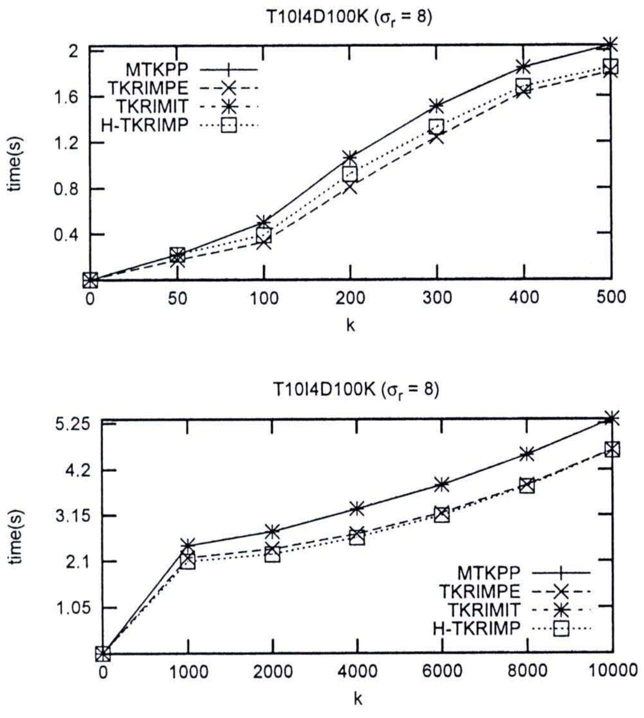


Figure 6.30: Runtime of H-TKRIMP on *T10I4D100K* ($\sigma_r = 8\%$)

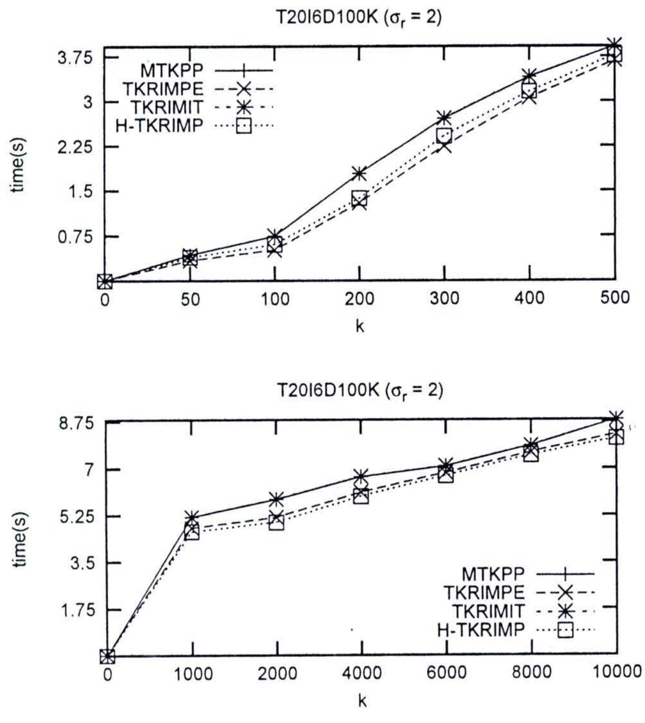
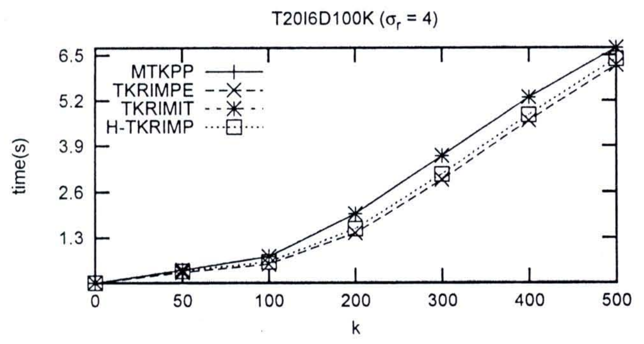


Figure 6.31: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 2\%$)



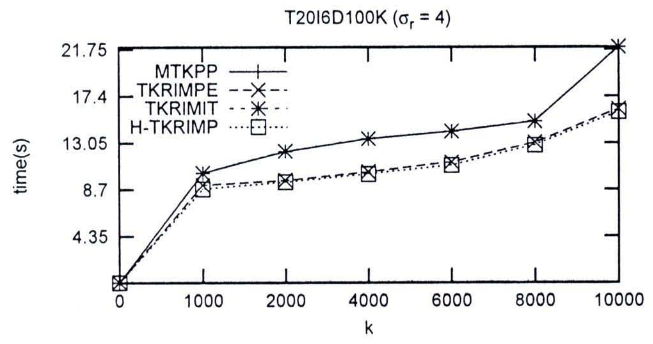
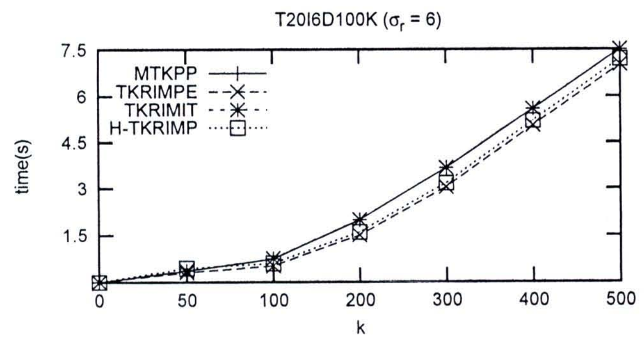


Figure 6.32: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 4\%$)



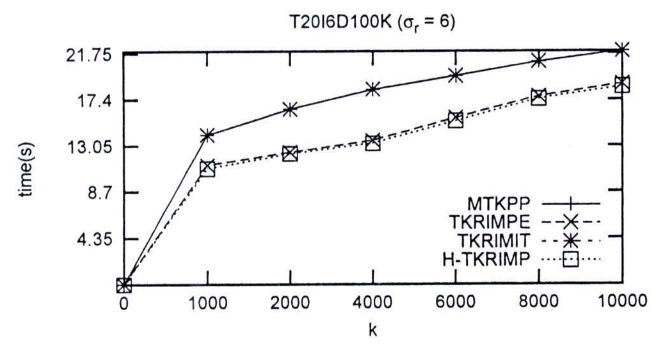
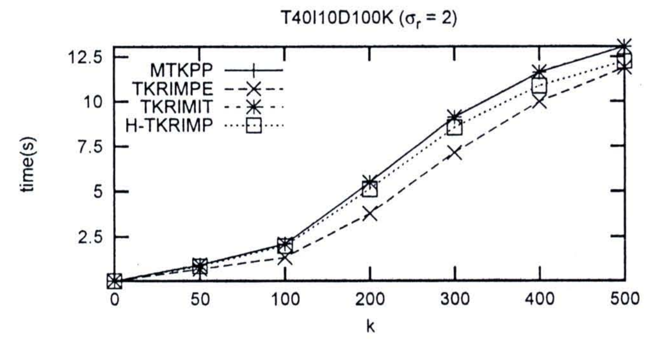


Figure 6.33: Runtime of H-TKRIMP on *T20I6D100K* ($\sigma_r = 6\%$)



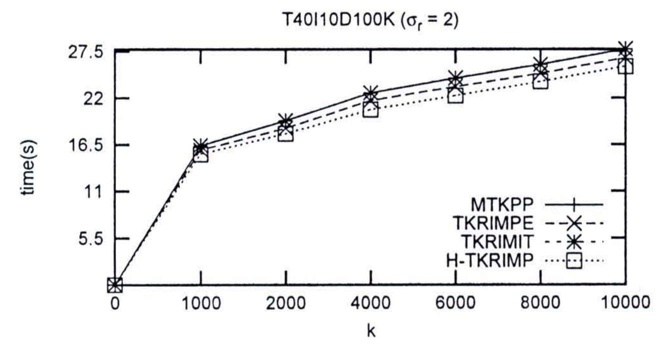
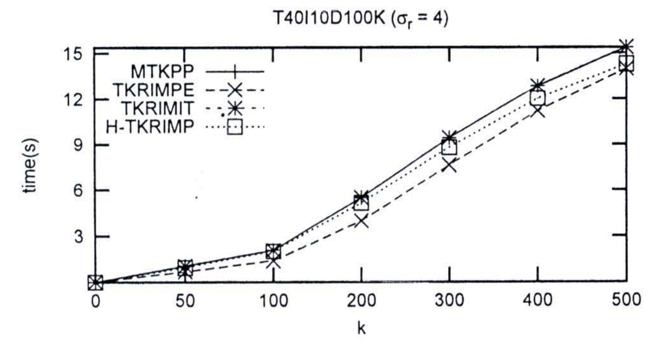


Figure 6.34: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 2\%$)



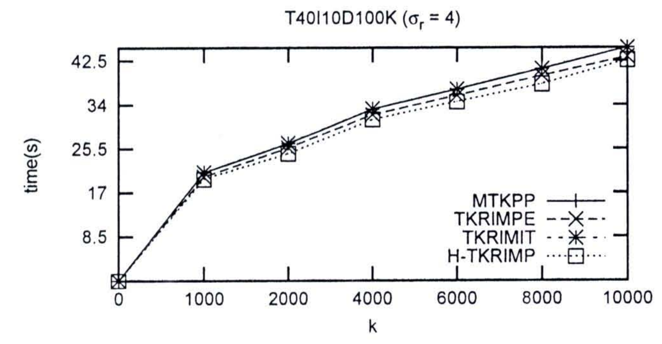


Figure 6.35: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 4\%$)

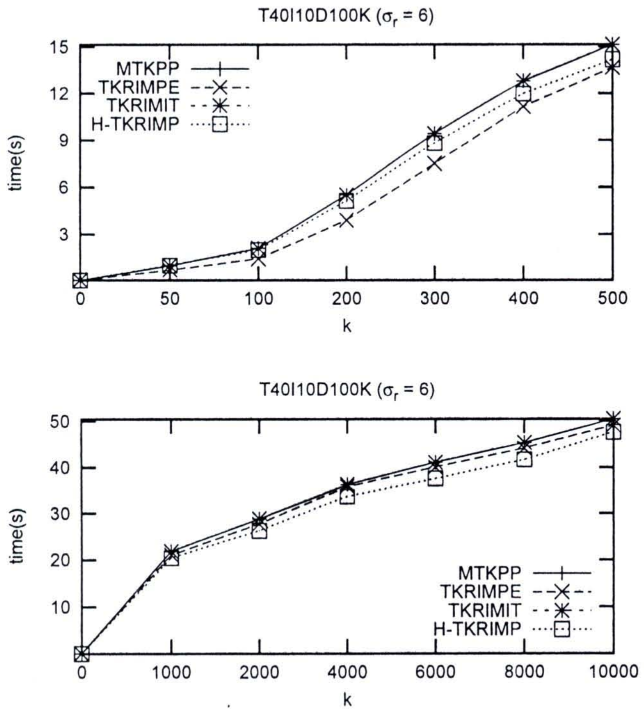


Figure 6.36: Runtime of H-TKRIMP on *T40I10D100K* ($\sigma_r = 6\%$)

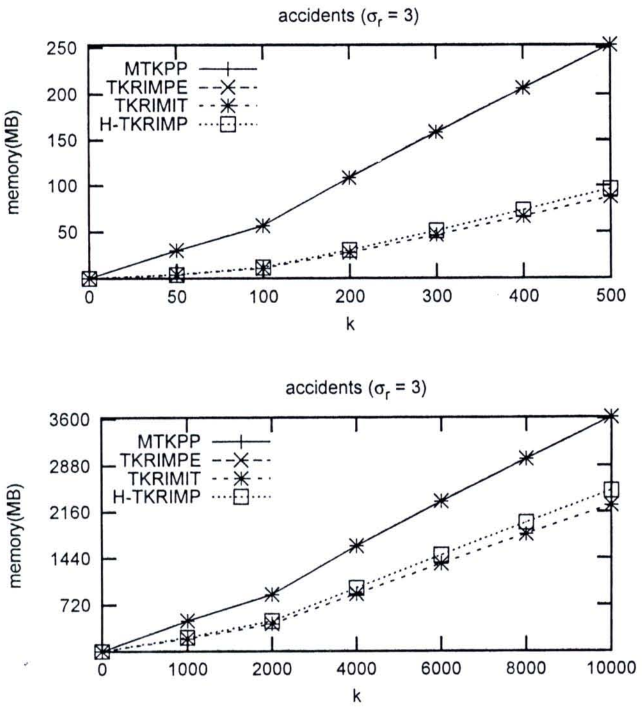
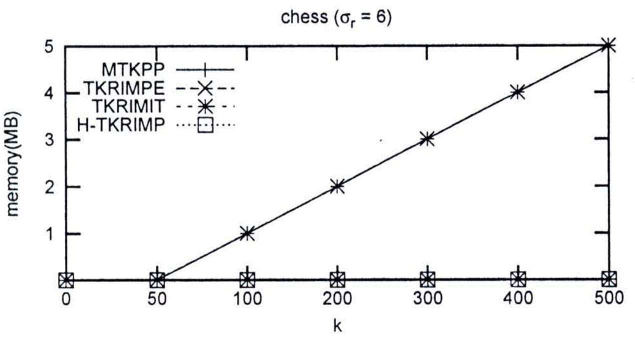


Figure 6.37: Memory usage of H-TKRIMP on *accidents*



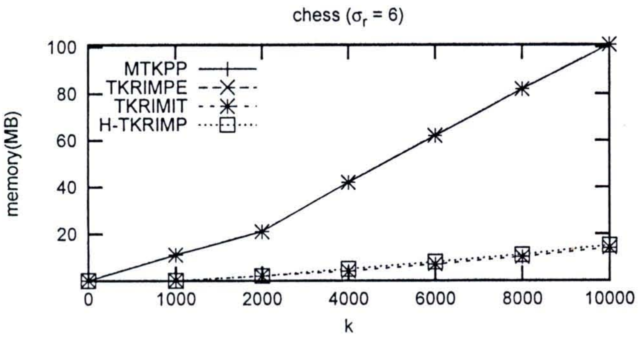
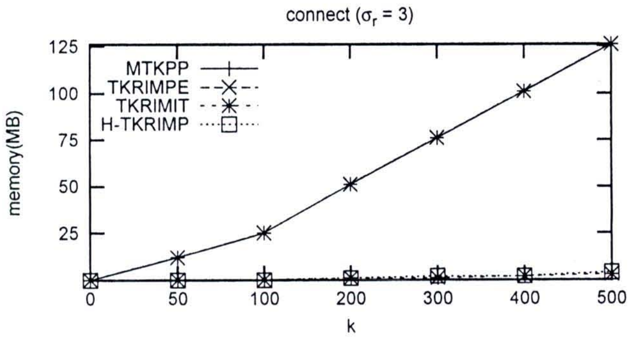


Figure 6.38: Memory usage of H-TKRIMP on *chess*



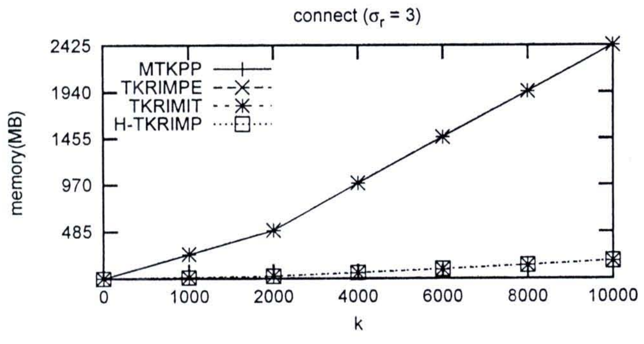
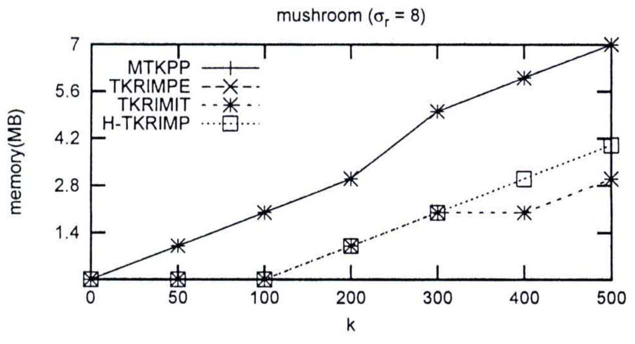


Figure 6.39: Memory usage of H-TKRIMP on *connect*



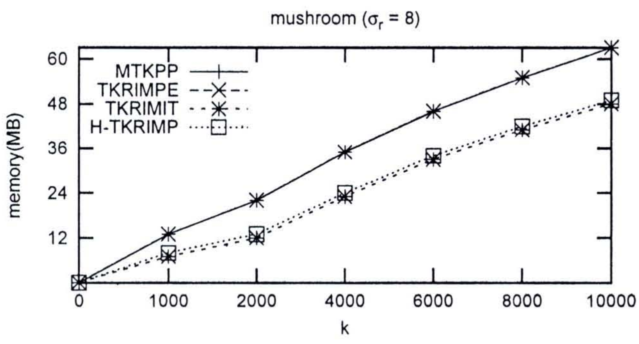
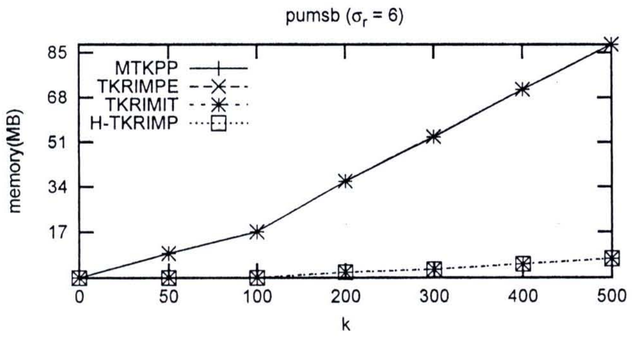


Figure 6.40: Memory usage of H-TKRIMP on *mushroom*



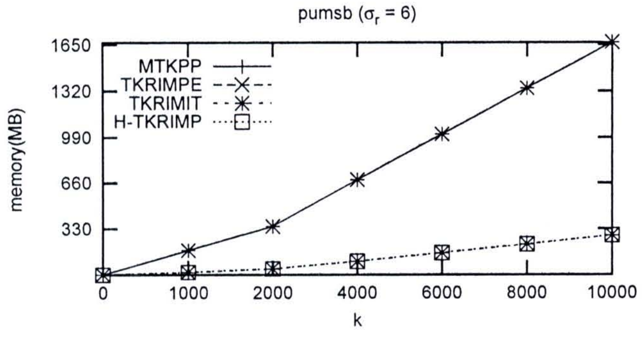
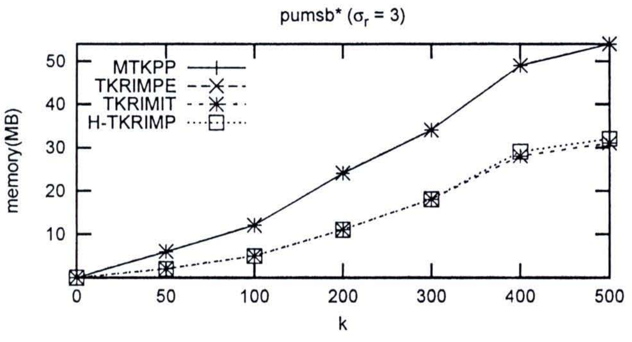


Figure 6.41: Memory usage of H-TKRIMP on *pumsb*



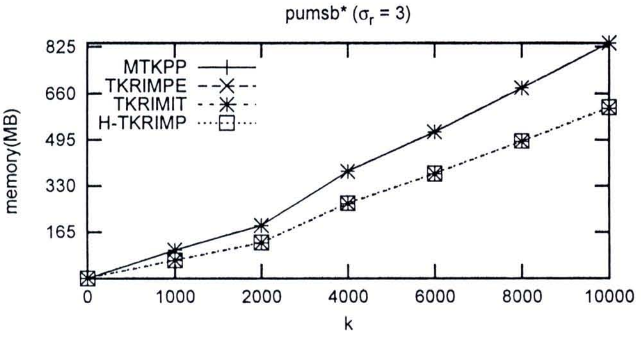
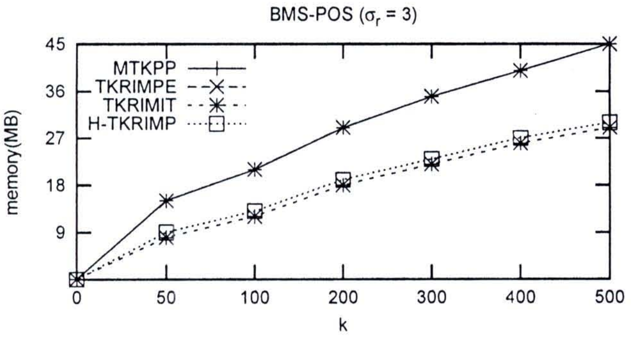


Figure 6.42: Memory usage of H-TKRIMP on *pumsb**



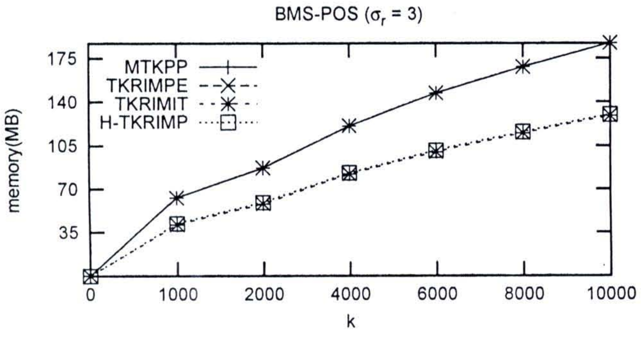
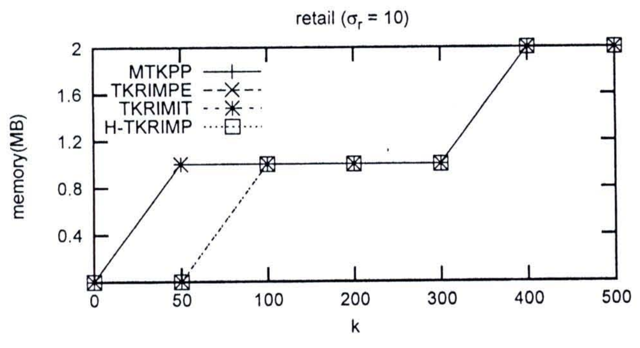


Figure 6.43: Memory usage of H-TKRIMP on *BMS-POS*



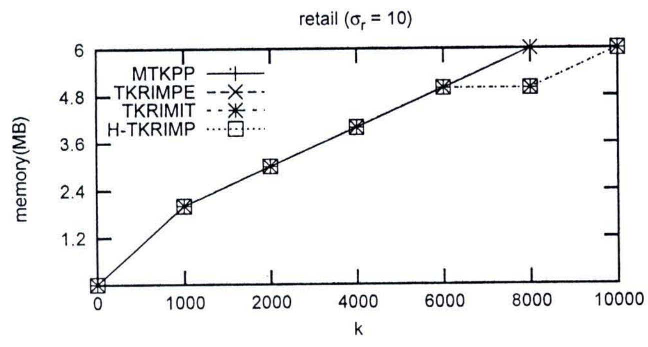
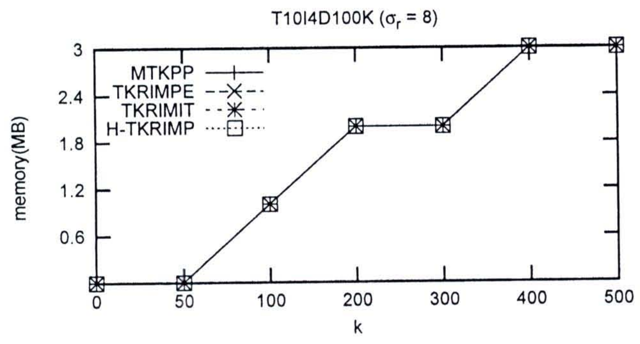


Figure 6.44: Memory usage of H-TKRIMP on *retail*



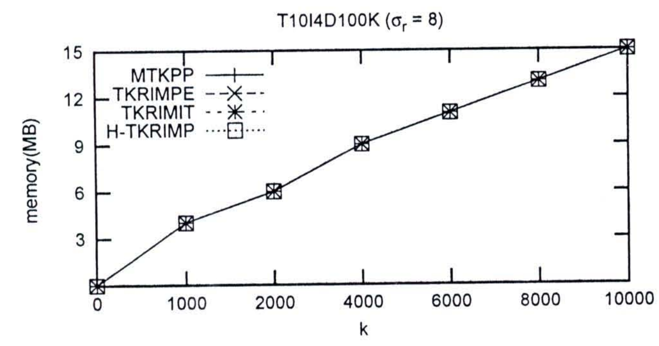
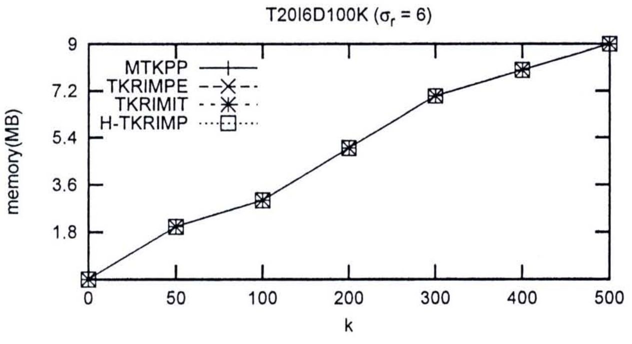


Figure 6.45: Memory usage of H-TKRIMP on *T10I4D100K*



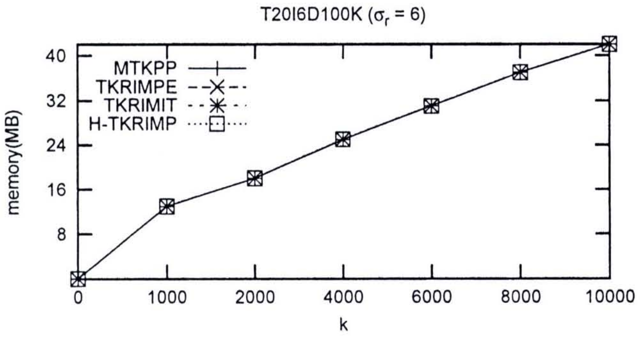
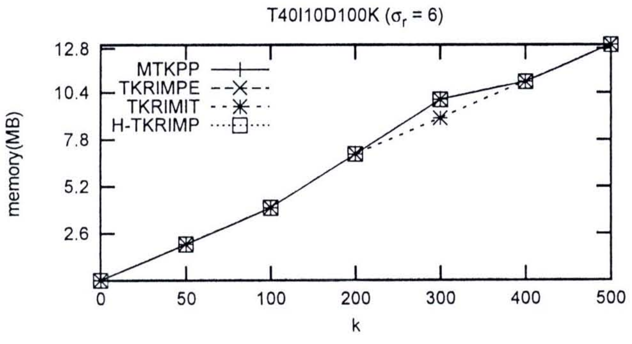


Figure 6.46: Memory usage of H-TKRIMP on *T20I6D100K*



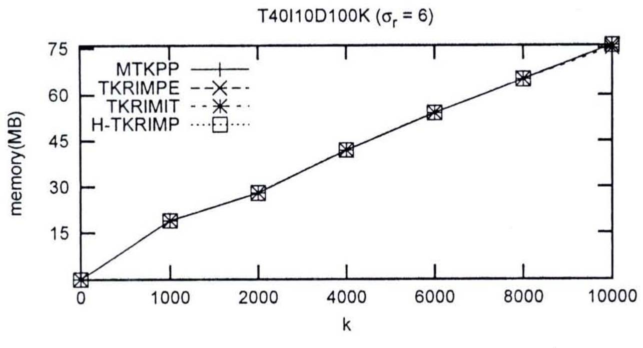


Figure 6.47: Memory usage of H-TKRIMP on *T40I10D100K*

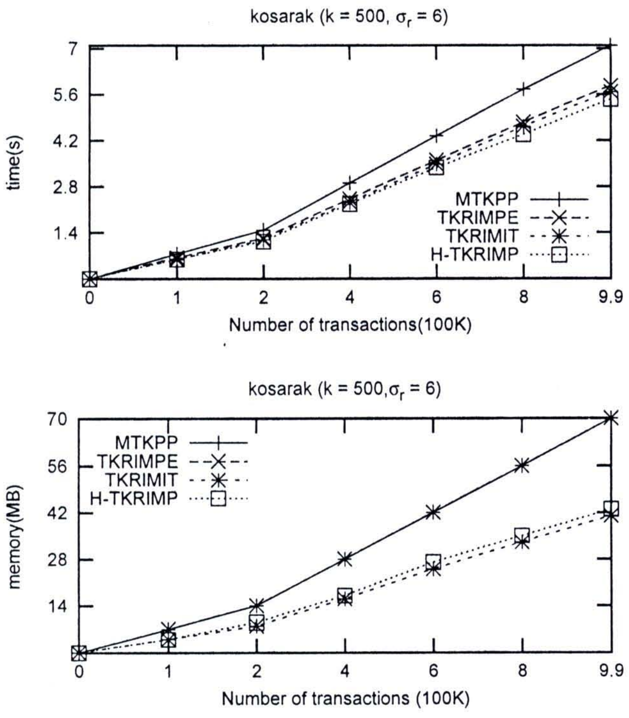


Figure 6.48: Scalability of H-TKRIMP (k : $500, \sigma_r = 6$)

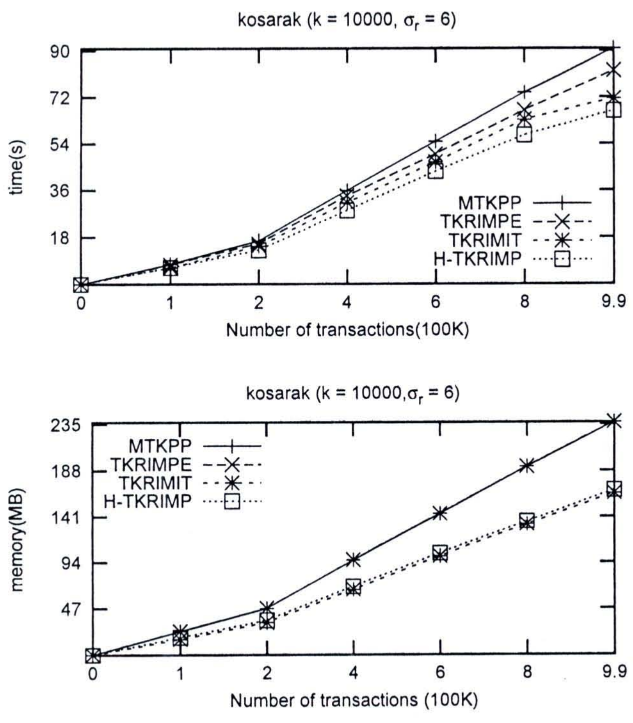


Figure 6.49: Scalability of H-TKRIMP (k : $10,000, \sigma_r = 6$)