Appendix III

Lemma 4.1 If $i \neq j$, $U(i) = U(j)$ if and only if ep[i]->pe = ep[j] (in which case, ep[j]->pe = ep[i]). Otherwise, U(i) and U(j) are disjoint.

Proof: This is ensured by the compress(N) routine of the Prepare_eppnodes algorithm.

Lemma 4.2 $lc \leq 2n$ and for every $i < lc$, if U(i) is non-empty, then ep[i]->e is a left endpoint i.e. if (b, ']') is a right endpoint in S, then $b \geq$ ep[lc]->e.x.

Proof: In the MIntMiner algorithm, lc is initialized to 0. So initially, $lc \leq 2n$ holds. It can be observed that the value of lc is never decreased by the algorithm. The value of lc is increased only within the locate_left() and slide_to_right() routines and so it suffices to establish that after any increase of lc within these routines, $lc \leq 2n$ holds.

When the locate_left() routine uses the slide_to_right() routine to increase the value of lc, it ensures that when slide_to_right() is invoked, $lc < 2n$ and ep[lc + 1] is a left endpoint. So in slide_to_right(), initially, $lcc < 2n$ and also ep[lcc + 1]->e is a left endpoint. Therefore, after the first increment of lcc in slide_to_right(), $lcc \leq 2n$ and ep[lcc]->e is a left endpoint. Any subsequent increment in the value of lcc in slide_to_right() now occurs in the while loop. It can be observed that prior to any increment of lcc in this while loop, ep[lcc]->e is always a left endpoint and so it's partner endpoint has to be ep[j]->e for some $j > lcc$ and $j \leq 2n$. This implies that $lcc < 2n$ prior to any increment of lcc in the while loop and so, after the increment, $lcc \leq 2n$. Since lc is reset to the value of lcc when slide_to_right() exits, it has thus been proved that after any increase of lcc (and hence of lc) within slide_to_right(), $lc \leq 2n$ holds. The check at the start of each iteration of the repeat-forever loop in locate_left() ensures that if lc is incremented in locate_left() without invoking slide_to_right(), then also, after the increment, $lc \leq 2n$. This completes the proof of the first part of the lemma.

The second part of the lemma will now be established. It can be observed that lc is incremented to an undeleted right endpoint in S i.e. lc is incremented to a value such that ep[lc]->e.b = ']' and ep[lc]->isdeleted = 0 only in locate_left(). However immediately after this increment of lc in locate_left(), U(lc) is made empty. This ensures that for every $i < lc$, if U(i) is non-empty, then ep[i]->e is a left endpoint. Therefore if (b, ']') is a right endpoint in S, then this right endpoint is ep[j]->e for some non-empty U(j) with $j \geq lc$. This implies *(ep[j]) $\geq$ *(ep[lc]) and so $b \geq$ ep[lc]->e.x. This completes the proof of the lemma.

Lemma 4.3 $\rho = \sum_{i=1}^{lc} | U(i) |$

Proof: Initially, $\rho = lc = 0$ and so this is trivially true. It can be observed that the value of lc is never decreased by the algorithm. Hence, to show the invariance of the given equality, all the three cases in which the value of lc may be increased are considered below –

(i) lc may be incremented to a deleted endpoint in the locate_left() routine. After this increment, neither side of the given equality changes and so the given equality is an invariant.

(ii) lc may be increased by the slide_to_right() routine from i to j. In this case, both sides of the given equality are increased by $\sum_{v=i+1}^{j}$ |U(v)|. Hence the given equality is an invariant.

(iii) The value of lc may be incremented immediately before delintv(lc) is called in the locate_left() routine. After delintv(lc) is executed, it can be observed that both sides of the given equality are reduced by |U(lc)|. Hence in this case also, the given equality is an invariant.

This completes the proof of the lemma.

Lemma 4.4 The value of ρ is less than k whenever locate_left() is invoked.

Proof: Initially, locate_left() is invoked when $\rho = 0$ and so this is trivially true. Subsequently, locate_left() is invoked immediately after extract_first() exits. Note that when the first call to delintv(rc) is made within the extract_first() routine, $\rho < k +$ ep[rc]->freq and the left endpoint of I(rc) is less than or equal to ep[lc]->e. Hence, after the delintv(rc) routine is executed for the first time within extract_first(), the value of ρ reduces by |U(rc)| = ep[rc]->freq and ρ then becomes less than k. There is no increase in the value of ρ in extract_first() and so when it exits, $\rho < k$. This proves that whenever locate_left() is invoked, ρ is less than k.

Lemma 4.5 M(k, S) remains unchanged by locate_left().

Proof: Noting that S is changed (i.e. reduced) in locate_left() only by the delintv(lc) statement, it suffices to establish that M(k, S) remains unchanged after this statement has been executed. Let J = [a, b] be any member of M(k, S) and let I(lc) = [c, d] when delintv(lc) is invoked in locate_left(). Note that when delintv(lc) is invoked in locate_left(), ep[lc]->e.x = d. When locate_left() is invoked, $\rho < k$ (by Lemma 4.4). The value of ρ is increased in locate_left() only by the slide_to_right() routine and as soon as the value of ρ becomes greater than or equal to k, locate_left() exits. Hence, when the delintv(lc) routine is invoked by locate_left(), $\rho < k$ i.e. $\sum_{i=1}^{lc} |U(i)| < k$ (by Lemma 4.3). This implies that at this point, there is some U(j) with $j > lc$ such that ep[j]->e is a left endpoint and $J \subseteq I(j)$ (because J is k-frequent with respect to S). Therefore, it can be concluded that (d, ']') = ep[lc]->e $\leq$ ep[j]->e $\leq$ (a, '['). But (d, ']') $\neq$ (a, '[') and so, (d, ']') < (a, '['). This implies that [c, d] cannot contain It has thus been established that the set of interval transactions viz. U(lc), that is deleted from S after the delintv(lc) statement in locate_left() is executed, does not support any member of M(k, S). By Corr. 4.5.3, the lemma holds.

Lemma 4.6 If ep[lc]->e.x = a when slide_to_right() exits, then $\rho = \sigma([a, a], S)$.

Proof: When slide_to_right() exits, ep[lc]->e is a left endpoint. By Lemma 4.2, for every non-empty U(i) with $i < lc$, ep[i]->e is a left endpoint. Thus, when slide_to_right() exits, for every non-empty U(i) with $i \leq lc$, ep[i]->pe->e is a right endpoint and it is equal to ep[j]->e for some $j > lc$. This implies that when slide_to_right() exits, every non-empty U(i) with $i \leq lc$ supports [a, a]. Also, slide_to_right() ensures that if $m > lc$ and ep[m]->e is a left endpoint, then ep[m]->e > ep[lc]->e. So, U(m) cannot support [a, a]. Therefore, when slide_to_right() exits, $\sigma([a, a], S) = |\cup_{i=1}^{lc} U(i)|$. It follows from Lemma 4.1 that all the non-empty U(i) with $i \leq lc$ are disjoint and so, $|\cup_{i=1}^{lc} U(i)| = \sum_{i=1}^{lc} |U(i)|$. Therefore, using Lemma 4.3, it is clear that $\rho = \sum_{i=1}^{lc} |U(i)| = |\cup_{i=1}^{lc} U(i)| = \sigma([a, a], S)$ when slide_to_right() exits. This proves the lemma.

Lemma 4.7 If locate_left() returns 0, then M(k, S) is empty. If locate_left() returns 1, then M(k, S) is non-empty and locate_left() sets lc to the largest value satisfying ep[lc]->e = (a,'['), where (a,'[') is the left endpoint of first(M(k, S)).

Proof: The routine locate_left() returns 0 if lc = 2n. When locate_left() returns 0, if ep[2n]->e is a right endpoint, then U(2n) is obviously empty (because U(2n) would have been deleted in the previous iteration of the repeat-forever loop in locate_left() or in the extract_first() routine). Also, at the point of return of locate_left(), it follows from Lemma 4.2 that for every non-empty U(i) with $i < 2n$, ep[i]->e is a left endpoint. Hence, when locate_left() returns 0, there does not exist any non-empty U(i) with $i \leq 2n$ in which ep[i]->e is a right endpoint. This implies that S and hence M(k, S) is empty.

The routine locate_left() returns 1 as soon as the slide_to_right() routine exits with $\rho \geq k$. If ep[lc]->e = (a, '[') at the point of return of slide_to_right() just before locate_left() exits with return value 1, then by Lemma 4.6 it follows that [a, a] is k-frequent. Thus M(k, S) is non-empty (by Theorem 4.1). Let first(M(k, S)) = [c, d]. By Theorem 4.1 and Corr. 4.3.1, $c \leq a$. Suppose $c < a$. By Theorem 4.2, (c, '[') is an endpoint of I(i) for some non-empty U(i) and since $c < a$, therefore $i < lc$. Because the exit condition of locate_left() was not satisfied when the endpoint (c, '[') was encountered, it follows from Lemma 4.6 that $\sigma([c, c], S) < k$. This is a contradiction because $\sigma([c, c], S) \geq \sigma([c, d], S) \geq k$.

Therefore, it can be concluded that $c = a$. This proves that when locate_left() exits with return value 1, it sets lc to a value such that ep[lc]->e is the left endpoint of first(M(k, S)). The slide_to_right() routine that is invoked by locate_left() just before it exits with return value 1, ensures that for $lc < i \leq 2n$, ep[lc]->e < ep[i]->e. This implies that when locate_left() returns 1, lc is set to the largest value satisfying ep[lc]->e = (c, '['). This completes the proof of the lemma.

Lemma 4.8 At the beginning of an iteration of the repeat-forever loop in locate_right(), if the following conditions are satisfied:

a) M(k, S) is not empty and ep[lc]->e is the left endpoint of first(M(k, S))

b) $rc \leq 2n$

c) ep[i]->e is a left endpoint for every non-empty U(i) with $i \leq rc$

then

(i) condition b) holds good throughout the iteration (and hence at the start of the next iteration, if any)

(ii) condition c) holds good at the beginning of the next iteration, if any

(iii) M(k, S) is not changed by the iteration

(iv) condition a) holds good throughout the iteration (and hence at the start of the next iteration, if any)

Proof: Condition a) of the hypothesis implies that at the beginning of an iteration of the repeat-forever loop in locate_right(), there is a non-empty U(i) in S in which ep[i]->e is a right endpoint and from condition c) of the hypothesis, it is clear that $i > rc$. Since $i \leq 2n$, it can be concluded that $rc < 2n$ at the beginning of the iteration of the repeat-forever loop in locate_right(). Within the iteration, the value of rc is incremented only once. This proves (i).

After the value of rc is incremented within the iteration of the repeat-forever loop in locate_right(), if ep[rc]->e is a right endpoint, then within that iteration itself, either U(rc) is deleted from S (i.e. U(rc) is made empty) by delintv(rc) or locate_right() exits. This proves (ii).

Before proving (iii), it is first observed that within an iteration of the repeat-forever loop in locate_right(), S is changed only by the delintv(rc) routine. Next, it is noted that the delintv(rc) routine is invoked if after the increment of rc within that iteration, ep[rc]->e is found to be a right endpoint and if at least one of the following conditions hold –

(A) ep[rc]->pe->e > ep[lc]->e

(B) $\rho \geq k + |U(rc)|$

Hence to prove (iii), it has to be established that M(k, S) remains unchanged after delintv(rc) is invoked within the iteration of the repeat-forever loop in locate_right() under either of these two conditions. Let ep[rc]->e = (d, ']') after the increment of rc within the iteration of the repeat-forever loop in locate_right(). Also, let I(rc) = [c, d]. Condition a) of the hypothesis implies that before delintv(rc) is invoked within the iteration of the repeat-forever loop in locate_right(), M(k, S) is not empty and if first(M(k, S)) = [a, b], then ep[lc]->e = (a, '['). From condition c) of the hypothesis, it follows that $b \geq d$. Now, suppose condition (A) holds. Under condition (A), $c > a$. So, $[c, d] \subset [a, b]$. Hence, after delintv(rc) is invoked (under condition (A)) within the iteration of the repeat-forever loop in locate_right() to remove U(rc) from S, by Corr. 4.6.1, M(k, S) remains unchanged. Next, suppose condition (B) holds. Let [f, g] be some member of M(k, S). First, it is observed that by Lemma 4.2 and condition a) of the hypothesis, ep[i]->e is a left endpoint for every non-empty U(i) with $i \le lc$ and all these left endpoints are less than or equal to (f,'['). Next, it is observed that by condition c) of the hypothesis, $g \ge d$. If $g = d$, then each non-empty U(i) with $i \le lc$ will support [f, g] and hence $\sigma([f, g], S) \ge |\cup_{i=1}^{lc} U(i)|$. It follows from Lemma 4.1 that all the non-empty U(i) with $i \le lc$ are disjoint and so, $|\cup_{i=1}^{lc} U(i)| = \sum_{i=1}^{lc} |U(i)|$. Thus if $g = d$, then $\sigma([f, g], S) \ge \rho$ (by Lemma 4.3). It is now observed that $\rho \ge k$ after delintv(rc) is invoked under condition (B). Therefore, it can be concluded that if $g = d$, then after delintv(rc) is invoked under condition (B) to remove U(rc) from S, $\sigma([f, g], S) \ge k$. On the other hand, if $g > d$, then no interval transaction in U(rc) can support [f, g] and so in this case also, [f, g] remains k-frequent with respect to S after delintv(rc) is invoked to remove U(rc) from S. By Corr. 4.5.2, it can therefore be concluded that after delintv(rc) is invoked (under condition (B)) within the iteration of the repeat-forever loop in locate_right() to remove U(rc) from S, M(k, S) remains unchanged. This completes the proof of (iii).

From condition a) of the hypothesis and (iii), it can be concluded that M(k, S) remains non-empty and first(M(k, S)) does not change within the iteration of the repeat-forever loop in locate_right(). Since lc is also not changed within the iteration of the repeat-forever loop in locate_right(), (iv) is hence proved. This completes the proof of the lemma.

Lemma 4.9 At the beginning of the first iteration of the repeat-forever loop in locate_right(), all the conditions of Lemma 4.8 – viz. conditions a), b) and c) hold.

Proof: To prove the lemma, it has to be established that

(i) When locate_right() is invoked for the first time in the MIntMiner algorithm, conditions a), b) and c) of Lemma 4.8 hold at the beginning of the first iteration of the repeat-forever loop.

(ii) If in the i^{th} call to locate_right(), conditions a), b) and c) of Lemma 4.8 hold at the beginning of the first iteration of the repeat-forever loop, then the same will also be true in the $i+1^{th}$ call to locate_right().

When locate_right() is invoked for the first time in the MIntMiner algorithm, then at the beginning of the first iteration of the repeat-forever loop,

- condition a) of Lemma 4.8 is ensured because of the locate_left() routine (Lemma 4.7).
- condition b) and condition c) of Lemma 4.8 hold because of Lemma 4.2 and the fact that ep[lc]->e is a left endpoint (by Lemma 4.7). Note that when locate_right() is invoked for the first time in the MIntMiner algorithm, the value of rc is set to lc.

This establishes (i).

Now, (ii) will be established. In the $i + 1^{th}$ call to locate_right(), condition a) of Lemma 4.8 holds at the beginning of the first iteration of the repeat-forever loop because of the locate_left() routine (Lemma 4.7). It can be observed that either the value of rc is set to lc within the $i + 1^{th}$ call to locate_right() or when the $i + 1^{th}$ call to locate_right() starts, rc is equal to the value last set by the extract_first()

routine. If rc is set to lc within the $i + 1^{th}$ call to locate_right(), then it can be concluded that conditions b) and c) hold at the beginning of the first iteration of the repeat-forever loop. The argument behind this claim is similar to the one made in (i). Now, suppose the $i + 1^{th}$ call to locate_right() starts with rc equal to the value last set by extract_first(). It has to be proved that in this case also, conditions b) and c) of Lemma 4.8 hold at the beginning of the first iteration of the repeat-forever loop. This will complete the proof of (ii). By Lemma 4.8, it follows that in the i^{th} call to locate_right(), conditions a), b) and c) of Lemma 4.8 hold at the beginning of the last iteration of the repeat-forever loop. Within this last iteration of the repeat-forever loop i.e. just before the i^{th} call to locate_right() ends, rc is set to a value such that U(rc) is non-empty and ep[rc]->e is a right endpoint. This U(rc) gets deleted in the extract_first() routine that is invoked immediately after the i^{th} call to locate_right() ends. Now, if there is any further increment in the value of rc in the extract_first() routine, then immediately after each increment, U(rc) is deleted within extract_first() itself. Moreover, using the argument behind (i) of Lemma 4.8, it can be concluded that in the i^{th} call to locate_right(), rc < 2n at the beginning of the last iteration of the repeat-forever loop. Within an iteration of the repeat-forever loop in locate_right(), rc is incremented just once and so, when the i^{th} call to locate_right() ends and the subsequent call to extract_first() is made, rc ≤ 2n. The while loop in extract_first() ensures that whenever rc is incremented within extract_first(), after the increment, rc ≤ 2n holds. Thus, it has been established that when the extract_first() routine (that is invoked immediately after the i^{th} call to locate_right() ends) exits and the

$i+1^{th}$ call to locate_right() starts with rc at the value set by the extract_first() routine, both the conditions b) and c) of Lemma 4.8 hold at the beginning of the first iteration of the repeat-forever loop. This completes the proof of (ii). By (i) and (ii), the lemma holds.

Lemma 4.10 M(k, S) is not changed by locate_right()

Proof. Follows from Lemma 4.8 and Lemma 4.9

Lemma 4.11 Whenever ep[m] is accessed, m is within bounds i.e. $1 \le m \le 2n$

Proof. The array ep[] is accessed through

(a) lc in locate_right(), delintv(i) and extract_first()

(b) $lc + 1$ in locate_left()

(c) rc in locate_right() and extract_first()

(d) $rc + 1$ in extract_first()

(e) integer variable lcc in slide_to_right()

(f) $lcc + 1$ in slide_to_right()

(g) parameter i in delintv(i)

(h) loop counter j in Step 2 of the MIntMiner algorithm

Though initially $lc = 0$, before the first access of ep[lc] takes place in the algorithm, the slide_to_right() routine increases the value of lc and hence $lc \ge 1$ whenever ep[lc] is accessed. By Lemma 4.2, $lc \le 2n$. Thus in case of (a), the result holds. The check at the beginning of the repeat-forever loop in locate_left() ensures that when ep[lc + 1] is accessed in locate_left(), $lc < 2n$ and hence $lc + 1 \le 2n$. Thus in case of (b), the result holds. Though initially $rc = 0$, before the first access of ep[rc] takes place in the algorithm, the value of rc is set to lc. At this point, lc ≥ 1 and hence rc ≥ 1 whenever ep[rc] is accessed. By Lemma 4.8 and Lemma 4.9, it follows that rc $\leq 2n$ whenever ep[rc] is accessed in locate_right(). Also, because of this, the access of ep[rc] in extract_first() is within bounds. Thus in case of (c), the result holds. In extract_first(), the check in the while loop ensures that when ep[rc + 1] is accessed, rc $< 2n$ and hence rc $+ 1 \leq 2n$. Thus in case of (d), the result holds. Before the first access of ep[lcc] takes place in slide_to_right(), lcc is incremented once. Hence lcc ≥ 1 whenever ep[lcc] is accessed in slide_to_right(). Also, it can be observed that in slide_to_right(), when ep[lcc] and ep[lcc + 1] are accessed, ep[lcc]->e is a left endpoint. This implies that the other endpoint of I(lcc) is at *(ep[i]) for some i > lcc. Clearly, i $\leq 2n$. Hence in slide_to_right(), when ep[lcc] is accessed, lcc $< 2n$ and when ep[lcc + 1] is accessed, lcc + 1$\leq 2n$. Thus in case of (e) and (f), the result holds. When ep[i] is accessed in delintv(i), the value of the parameter i is either lc or rc. Thus in case of (g), the result holds (because it holds in case of (a) and (c)). Finally, in case of (h), the result is ensured by the for loop of Step 2 of the MIntMIner algorithm. This completes the proof of the lemma.

Lemma 4.12 When locate_right() exits, ep[rc]->e is the right endpoint of first(M(k, S))

Proof: By Lemma 4.8 and Lemma 4.9, in the beginning of the last iteration of the repeat-forever loop in locate_right(), M(k, S) is not empty and ep[lc]->e = (a, '['), where [a, b] = first(M(k, S)). By Lemma 4.2, it now follows that in the beginning of the last iteration of the repeat-forever loop in locate_right(), ep[i]->e is a left endpoint for every non-empty U(i) with $i \le lc$. Let ep[rc]->e = (d, ']') after the increment of rc within the last iteration of the repeat-forever loop in locate_right(). By Lemma 4.8 and Lemma 4.9, it follows that in the beginning of the last iteration of the repeat-forever loop in locate_right(), for every non-empty U(i) with $i \le lc$, ep[i]->pe->e $\ge$ (d, ']'). Also, by Lemma 4.7, it follows that lc has been set to largest value satisfying ep[lc]->e = (a,'['). So, $\sigma([a, d], S) = |\cup_{i=1}^{lc} U(i)|$. It follows from Lemma 4.1 that in the beginning of the last iteration of the repeat-forever loop in locate_right(), the non-empty U(i) with $i \le lc$ are all disjoint and so, $|\cup_{i=1}^{lc} U(i)| = \sum_{i=1}^{lc} |U(i)|$. Thus by Lemma 4.3, it follows that $\sigma([a, d], S) = |\cup_{i=1}^{lc} U(i)| = \sum_{i=1}^{lc} |U(i)| = \rho$. Again, by Lemma 4.8 and Lemma 4.9, it follows that $b \ge d$. Suppose $b > d$. Then, after the increment of rc within the last iteration of the repeat-forever loop in locate_right(), U(rc) will not support [a, b]. The exit condition of locate_right() ensures that ep[rc]->pe->e $\le$ ep[lc]->e and so, at this point, U(rc) = U(j) for some $j \le lc$. Clearly, if $b > d$, then U(j) will also not support [a, b] and hence $\sigma([a, b], S) \le \sum_{i=1}^{lc}|U(i)| - |U(j)| = \rho - |U(rc)|$. Also, by the exit condition of locate_right(), $\rho - |U(rc)| < k$. Hence, it can be concluded that if $b > d$ when locate_right() exits, then $\sigma([a, b], S) < k$. This is however a contradiction because [a, b] is k-frequent with respect to S. Therefore, it is established that $b = d$ when locate_right() exits. Thus the lemma holds.

Lemma 4.13 If find_first() returns 0, then M(k, S) is empty. If find_first() returns 1, then M(k, S) is non-empty and find_first() sets lc and rc such that lc is the