Supplementary File of MACFP: Maximal Approximate Consecutive Frequent Pattern Mining under Edit Distance

Jingbo Shang Jian Peng Jiawei Han
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA.
{shang7, jianpeng, hanj}@illinois.edu

1 Experiments

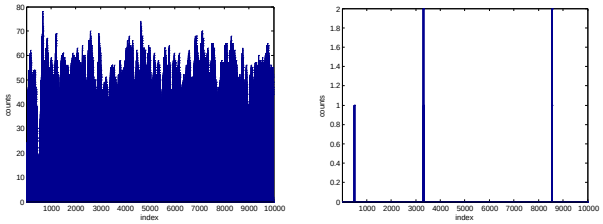
The details of the running time are offered in Table 1.

2 Applications

In this section, we present two more synthetic cases to demonstrate the power of MACFP in repetitive region detection. We mainly study the performance when the length of repetitive region is extremely short and very long. More specifically,

1. A huge number of repeats of a short region, which corresponds to the short tandem repeat problem. More specifically, $n = 10,000$, $m = 10$, and $T = 1,000$.

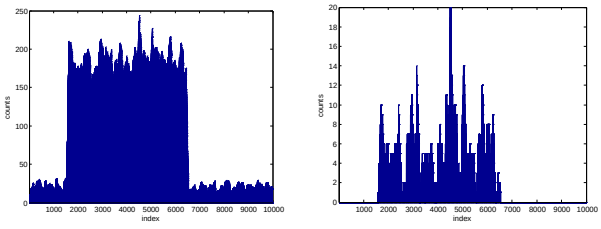
2. Reasonably many repeats of a large chunk, which corresponds to copy number variation. More specifically, $n = 10,000$, $m = 5,000$, and $T = 10$.



(a) RMC (b) MACFP

Figure 1: Real World Application Scenario 3: Extremal short tandem repeats. More specifically, $n = 10,000$, $m = 10$, and $T = 1,000$. In our experiment, the repetitive region is located between 501 and 510.

In the third scenario (i.e., extremal short tandem repeats), the repetitive region is located between 501 and 510. As shown in Figure 1, RMC finds several peaks and the highest one is around 1,000. However, the correct region between 501 and 510 is actually a valley. Therefore, RMC will mislead scientists in this case. On the other hand, MACFP ($k = 1$, $\sigma = 5,000$, and $L = 10$) finds only three candidate positions including the correct one, which is much more efficient and clear.



(a) RMC (b) MACFP

Figure 2: Real World Application Scenario 4: long copy number variation. More specifically, $n = 10,000$, $m = 5,000$, and $T = 10$. In our experiment, the repetitive region is located between 1,534 and 6,533.

In the forth scenario (i.e., the long copy number variation problem), the repetitive region is located between 1,534 and 6,533. As shown in Figure 2, RMC finds an obvious peak and it is the correct one. In this scenario, the length of pattern is longer than the random accessed strings; therefore, we can only mine some short pattern and try to assemble them together. Interestingly, by plotting the short patterns we mined by MACFP ($k = 1$, $\sigma = 10$, and $L = 100$), there is a clear consecutive regions between 1,534 and 6,533, which can guide the scientists correctly.

In summary, the MACFP algorithm can identify the repetitive DNA region more correctly and clearly than the widely-used RMC method which directly matches the short reads onto the reference DNA sequences, especially when the length of repetitive region is shorter than the length of short reads.

Both experiments show that the MACFP algorithm can identify the repetitive DNA region more correctly and clearly than the widely-used RMC method which directly matches the short reads onto the reference DNA sequences.

Table 1: Run time in seconds (average from 10 runs) and the numbers of identified patterns.

	Small Dataset				Medium Dataset			Large Dataset		
parameter		TDP	MACFP	# Patterns		MACFP	# Patterns		MACFP	# Patterns
k	1	527.35	0.05	0	1	0.53	22	1	6.22	316
	2	682.90	0.14	16	2	1.75	145	2	24.02	2,438
	3	857.63	0.61	67	3	8.87	573	3	107.49	5,797
	4	1059.92	3.52	189	4	43.08	1,264	4	532.19	10,340
	5	1267.16	24.38	340	5	197.72	2,149	5	3631.55	15,313
L	10	1112.62	54.07	6143	30	38.86	3,754	30	991.06	22,061
	20	878.41	3.82	379	40	18.07	1,727	40	202.05	12,072
	30	857.63	0.61	67	50	8.87	573	50	106.06	5,797
	40	853.84	0.24	5	60	4.29	103	60	59.12	2,355
	50	853.43	0.15	0	70	2.493	32	70	34.61	831
σ	2	1053.02	0.90	311	2	12.67	1,059	2	103.53	10,208
	3	921.51	0.74	145	3	9.61	783	3	106.56	7,004
	4	857.63	0.61	67	4	8.87	573	4	107.07	5,797
	5	820.90	0.55	31	5	7.50	457	5	108.93	4,983
	6	798.27	0.50	15	6	7.35	308	6	108.88	4,179

3 Proofs

LEMMA 1. ***Pattern Anti-Monotonicity***. *If $S_{i,j}$ is a frequent approximate consecutive pattern, $S_{i+1,j}$ and $S_{i,j-1}$ are also frequent approximate consecutive patterns.*

Proof. Suppose $sup(S_{i,j}) = p \geq \sigma$ is computed from the following sequence of non-overlapped approximate equivalent substrings

$$l_1 \leq r_1 < l_2 \leq r_2 < \ldots < l_x = i \leq r_x = j < \ldots < l_p \leq r_p$$

where, $\forall q(1 \leq q \leq p), d(S_{i,j}, S_{l_q,r_q}) \leq k$.

Considering a specific approximate equivalent substrings S_{l_q,r_q}, where

$$d(S_{i,j}, S_{l_q,r_q}) \leq k$$

Because edit operations include the insertions and deletions, at least one of the followings should be true depending on whether S_i matches S_{l_q} in the distance calculation.

- $d(S_{i+1,j}, S_{l_q,r_q}) \leq k$, if S_i does not match S_{l_q}.
- $d(S_{i+1,j}, S_{l_q+1,r_q}) \leq k$, if S_i matches S_{l_q}.

Therefore, we can use either the same (l_q, r_q) or the shrunk $(l_q + 1, r)$ substring for $S_{i+1,j}$, and thus we have the following sequence of non-overlapped approximate equivalent substrings for $S_{i+1,j}$

$$l'_1 \leq r_1 < l'_2 \leq r_2 < \ldots < l'_x = i+1 \leq r_x = j < \ldots < l'_p \leq r_p$$

where, $\forall q(1 \leq q \leq p), d(S_{i,j}, S_{l'_q,r_q}) \leq k$, and l'_q is either l_q or $l_q + 1$, as we demonstrated before. This sequence proves that $sup(S_{i+1,j}) \geq p \geq \sigma$.

Similar arguments could be made for $sup(S_{i,j-1}) \geq \sigma$.

LEMMA 3. *Suppose $S_{u+1,v}$ or $S_{u,v-1}$ is an equivalent neighbor of $S_{i,j}$, the approximate support of $S_{i,j}$ is same even if we ignore $S_{u,v}$.*

Proof. If there is no solution, which achieves the maximum number of non-overlapping equivalent neighbors of $S_{i,j}$, involves $S_{u,v}$, the lemma holds obviously since less equivalent neighbors considered will also lead to no solution.

Otherwise, assume in a solution which achieves the maximum number of non-overlapping equivalent neighbors of $S_{i,j}$, there exists at least one substring $S(u, v)$, which is not extremely small. That is, at least one of $S_{u+1,v}$ and $S_{u,v-1}$ is also an equivalent neighbor of $S_{i,j}$. Therefore, we can replace $S_{u,v}$ using its substring while not violating the disjoint condition. Therefore, the maximum number of non-overlapping equivalent neighbors keeps the same if we ignore $S_{u,v}$.

This process could be applied recursively until the non-overlapping equivalent neighbors are all extremely small. In conclusion, only extremely small equivalent neighbors matter.

4 Algorithms

4.1 Tailored Dynamic Programming (TDP)

Considering the lower bound of edit distance, we can ignore the pairs of substrings which are too different on the lengths. This helps us prune many unnecessary states in the classical dynamic programming for edit distance [3, 1], as illustrated in Figure 3.

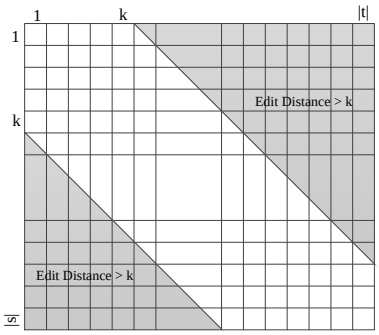


Figure 3: Illustration for Tailored Dynamic Programming

By utilizing this idea, we can have Algorithm 1.

Algorithm 1: Tailored Dynamic Programming

Require: a short string s, a long string t, and distance threshold k
Return: $\forall 0 \leq i \leq k$, $q_i = \min\{j | d(s, t_{0,j}) \leq i\}$
Time Complexity: $O(|s|k)$
$f_{*,*} \leftarrow k+1$
$f_{1,0} \leftarrow 0$
for $i = 1$ ***to*** $|s|$ **do**
  for $\delta = -k$ ***to*** k **do**
    $j \leftarrow i + \delta$
    `// insert` s_i `before` t_j `/ delete` s_i
    $f_{i+1,\delta-1} \leftarrow \min\{f_{i+1,\delta}, f_{i,\delta} + 1\}$
    `// insert` t_j `before` s_i `/ delete` t_j
    $f_{i,\delta+1} \leftarrow \min\{f_{i,\delta+1}, f_{i,\delta} + 1\}$
    `// match/replace` s_i `and` t_j
    $f_{i+1,\delta} \leftarrow \min\{f_{i,\delta+1}, f_{i,\delta} + (s_i \neq t_j)\}$
vector $q \leftarrow +\infty$
for $\delta = k$ ***downto*** $-k$ **do**
  $j \leftarrow i + \delta$
  if $f_{|s|+1,\delta} \leq k$ **then**
    $q_{f_{|s|+1,\delta}} \leftarrow j$
return vector q

4.2 Lower Bound Pruning We present the details of applying lower bound pruning in Algorithm 2.

4.3 From Neighbors to Non-overlapping Neighbors As mentioned in the problem formulation, we define the maximum number of disjoint (non-overlapping) equivalent neighbors as our approximate support. We propose a post-processing algorithm as shown in Algorithm 3 to help us compute the approximate support. Because it is a classical interval scheduling problem [2], the correctness is proved in previous work. Thus we omit the proof here.

Algorithm 2: Lower Bound Pruning

Require: two substrings $S_{l,r}$ and $S_{u,*}$, edit distance threshold k
Return: the lower bound of $\forall v, d(S_{l,r}, S_{u,v})$
Time Complexity: $O(|\Sigma|)$ after $O(|S||\Sigma|)$ pre-processing
$v_1, v_2 \leftarrow 0$
for $c \in \Sigma$ **do**
  $c_1 \leftarrow$ counts of c in $S_{l,r}$
  $c_{upper} \leftarrow$ counts of c in $S_{u,u+r-l+k}$
  $c_{lower} \leftarrow$ counts of c in $S_{u,u+r-l-k}$
  if $c_1 > c_{upper}$ **then**
    $v_1 \leftarrow v_1 + c_1 - c_{upper}$
  if $c_1 < c_{lower}$ **then**
    $v_2 \leftarrow v_2 + c_{lower} - c_1$
return $\max\{v_1, v_2\}$

Algorithm 3: Maximum Non-overlapping Intervals

Require: A set of substrings C
Return: A maximum set of non-overlapped substrings
Time Complexity: $O(|C|log|C|)$, using comparison-based sorting
Sort C increasingly by right ends
$M \leftarrow \emptyset$
$r_{max} \leftarrow -\infty$
for $S_{l,r}$ *in* C **do**
  if $l > r_{max}$ **then**
    $r_{max} \leftarrow r$
    $M \leftarrow M \cup \{S_{l,r}\}$
return M

References

[1] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.
[2] J. Kleinberg and É. Tardos. *Algorithm design*. Pearson Education India, 2006.
[3] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684. IEEE, 2002.