Supplemental Information

This document contains the pseudo-code showing the algorithms described in this manuscript. Figure 1 shows the routines required for top down search in a suffix array. Algorithm 1 shows the main part of the algorithm for finding MEMs in a sparse suffix array. The algorithm must be be started at offset prefixes p_0 in the query string P. In other words, MEM(p_0) must be run for each value of $p_0 = 0 \ldots K-1$. The parallel version runs each call $p_0 = 0 \ldots K-1$ in a separate thread. The MEMs algorithm relies on calls to **traverse** (see Algorithm 2) in order to match up to $\mathbf{L}-(K-1)$ characters and to find the maximum length match. If a match of length $\geq \mathbf{L}-(K-1)$ characters can be obtained, the suffix array interval $d:[s..e]$ corresponding to matches of length $\geq \mathbf{L}-(K-1)$ and the interval $q:[l..r]$ corresponding to the maximum length match is used by **collectMEMs** in Algorithm 3 below to find right maximal matches. Each right maximal match must be verified for left maximality by scanning up to K characters to the left of the match (see **findL** Algorithm 4 below). Suffix links are simulated using **suffixlink** and **expandlink** in Figure 2.

bsearchL(c, $p:[l..r]$)
if $c = S[SA[l]+p]$ return l
while $r-l>1$
  $m=(l+r)/2$
  if $c \leq S[SA[m]+p]$ then $r=m$
  else $l=m$
return r

bsearchR(c, $p:[l..r]$)
if $c = S[SA[r]+p]$ return r
while $r-l>1$
  $m=(l+r)/2$
  if $c < S[SA[m]+p]$ then $r=m$
  else $l=m$
return l

topdown(c, $p:[s..e]$)
if $c < S[SA[s]+p]$ return $\perp:[0..0]$
if $c > S[SA[e]+p]$ return $\perp:[0..0]$
l=**bsearchL**(c, $p:[s..e]$)
r=**bsearchR**(c, $p:[s..e]$)
if $l \leq r$ return $(p+1):[l..r]$
else return $\perp:[0..0]$

search(P)
$p=0$, $s=0$, $e=n-1$
while $p<|P|$
  $p':[s'..e']$=**topdown**($P[p]$, $p:[s..e]$)
  if $p'=\perp$ return $\perp$
  $p:[s..e]=p':[s'..e']$
return $[s..e]$

Figure 1: Elements of top down traversal in suffix array by $O(\log n)$ binary search. **bsearchL** and **bsearchR** use binary search to locate the left and right ends of matched interval respectively. **topdown** uses each binary search to match one character c at a time. **search** shows how **topdown** can be used to locate an exact match of a query string P in the suffix array indexed reference string S.

Algorithm 1 MEM(p_0)

$d : [s..e] = q : [l..r] = 0 : [0..n/K-1]$, $p = p_0$
while $p < |P| - (K - p_0)$
  $d : [s..e]$ = **traverse**(p, $d : [s..e]$, $\mathbf{L} - (K-1)$)
  $q : [l..r]$= **traverse**(p, $q : [l..r]$, $|P|$)
  if $d \leq 1$ then
    $d : [s..e] = q : [l..r] = 0 : [0..n/K-1]$
    $p = p + K$, continue
  if $d \geq \mathbf{L} - (K-1)$ then **collectMEMs**(p, $d : [s..e]$,$q : [l..r]$)
  $p = p + K$
  $d : [s..e]$ = **suffixlink(**$d : [s..e]$**)**
  $q : [l..r]$ = **suffixlink(**$q : [l..r]$**)**
  if $d = \perp$ then $d : [s..e] = q : [l..r] = 0 : [0..n/K-1]$, continue

Algorithm 2 traverse(p, $d : [s..e]$, M)

while $p + d < |P|$
  $d_2 : [s_2..e_2]$ = **topdown**($P[p+d]$, $d : [s..e]$)
  if $d_2 = \perp$ return $d : [s..e]$
  $d : [s \ldots e] = d_2 : [s_2..e_2]$
  if $d \geq M$ break
return $d : [s..e]$

Algorithm 3 collectMEMs(p, $d : [s..e]$,$q : [l..r]$)

for $i = l \ldots r$ do **findL**(p, $SA[i]$, q)
while $q \geq d$
  if $r + 1 < n/K$ then $q = \max(LCP[l], LCP[r+1])$
  else $q = LCP[l]$
  if $q \geq d$ then
    while $LCP[l] \geq q$ do
      $l = l - 1$, **findL**(p, $SA[l]$, q)
    while $r + 1 < n/K$ and $LCP[r+1] \geq q$ do
      $r = r + 1$, **findL**(p, $SA[r]$, q)

Algorithm 4 findL(p, i, q)

for $k = 0 \ldots K-1$ do
  if ($p = 0$ or $i = 0$) and $q \geq \mathbf{L}$
    MEM at i in S, p in P , and length q
    return
  if $P[p-1] \neq S[i-1]$ and $q \geq \mathbf{L}$
    MEM at i in S, p in P, and length q
    return
  $p = p - 1$, $i = i - 1$, $q = q + 1$

suffixlink($q : [l..r]$)
$q = q - K$
if $q \leq 0$ return $\perp : [0 \ldots 0]$
$l = ISA[SA[l]/K + 1]$
$r = ISA[SA[r]/K + 1]$
$q : [l..r]$ = **expandlink**($q : [l..r]$)
return $q : [l..r]$

expandlink($q : [l \ldots r]$)
if $q = 0$ return $0 : [0 \ldots n-1]$
$T = 2q \log n$, $e = 0$
while $l \geq 0$ and $LCP[l] \geq q$
    $e = e + 1$, if $e \geq T$ return $\perp : [0 \ldots 0]$
    $l = l - 1$
while $r \leq n-1$ and $LCP[r+1] \geq q$
    $e = e + 1$, if $e \geq T$ return $\perp : [0 \ldots 0]$
    $r = r + 1$
return $q : [l \ldots r]$

Figure 2: Suffix links are simulated using **suffixlink** and **expandlink**.