Holistic Twig Joins on Indexed XML Documents

Haifeng Jiang, Wei Wang, Hongjun Lu, Jerey Xu Yu

Presented by Wanxing (Sarah) Xu

2008.10

XML Query process Method

- Structure Join: PAIR at one time
- Path Stack: PATH at one time
- Twig Stack: TWIG at one time

- Indexed-Twig: Using index for nodes to accelerate

How to Accelerate?

- Algorithm:
  - TSGeneric^{+}: giving more opportunity to jump
- Index:
  - TR-tree: jumping faster and further each time

Algorithms

- The Generic Twig Join Algorithm (*TSGeneric*)
- The *TSGeneric*$^{+}$ Algorithm

The *TSGeneric* Algorithm

- Algorithm *getNext*(q)
  - Returns a query node q_x in the subtree q satisfying all the following:
  - q_x has a solution extension.
  - if q_x has siblings, C_{qx} -> start < C_{qs} -> start for all sibling q_s of q_x.
  - If $q_x \neq$ q, $C_{parent(qx)}$ -> start > C_{qx} -> start.

getNext(q)

Algorithm 2 getNext(q)

```
if isLeaf(q) then
    return q;
for q_i in children(q) do
    n_i = getNext(q_i);
    if n_i ≠ q_i then
        return n_i;
end for
n_min = minarg_{n_i} {C_{n_i}→start};
n_max = maxarg_{n_i} {C_{n_i}→start};
while C_q→end < C_{n_max}→start do
    C_q→advance();
end while
if C_q→start < C_{n_min}→start then
    return q;
else
    return n_min;
```

Corollary 1 → (line: if $C_q \rightarrow start < C_{n_{min}} \rightarrow start$ then)

If q is a leaf, return q.

If q is not a leaf, determine if the children of q has solution extension rooted at them, else return descendant of q with solution extension.

Determine descendant of q with minimum and maximum *start* attribute.
Advance cursor of q so so that `Cq -> end >= Cnmax -> start`.
If `Cq -> start < Cnmin -> start` return node q, else return descendant n_{min} of q with minimum *start* attribute.

Cursor Interface

- ***C_q*****`->fwdBeyond(`*****C_p*****`)`** forwards C_q to the first element *e*, such that ***`e.start > `*****C_p*****`->start`**
- ***C_q*****`->fwdToAncestorOf(`*****C_p*****`)`** forwards the cursor to the first ancestor of C_p and returns TRUE. If no such ancestor exists, it stops at the first element *e*, such that ***`e.start > `*****C_p*****`->start`**, and returns FALSE.

TSGeneric with Cursor Interface

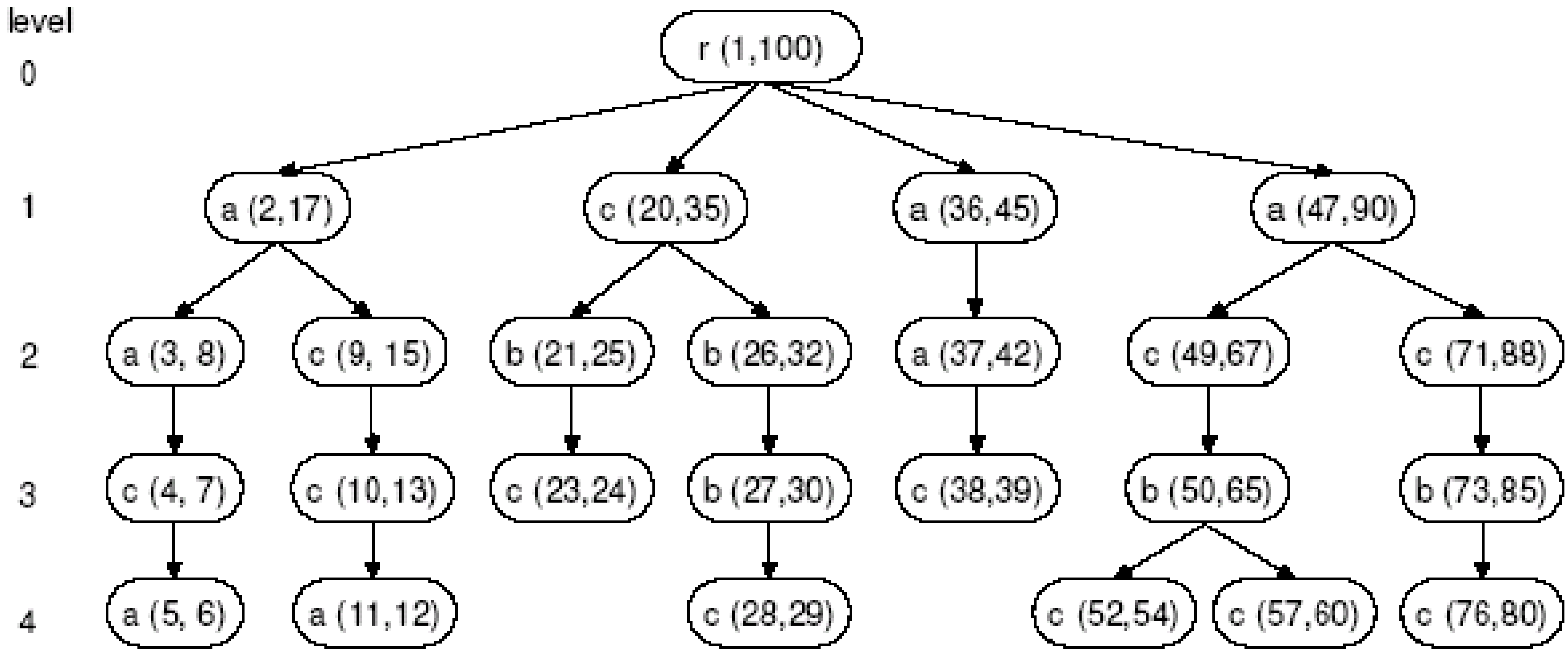
Algorithm 2 getNext(q)

```
if isLeaf(q) then
    return q;
for q_i in children(q) do
    n_i = getNext(q_i);
    if n_i ≠ q_i then
        return n_i;
end for
n_min = minarg_{n_i} {C_{n_i}→start};
n_max = maxarg_{n_i} {C_{n_i}→start};
while C_q→end < C_{n_max}→start do
    C_q→advance();
end while
if C_q→start < C_{n_min}→start then
    return q;
else
    return n_min;
```

Algorithm 3 getNextCursor(q)

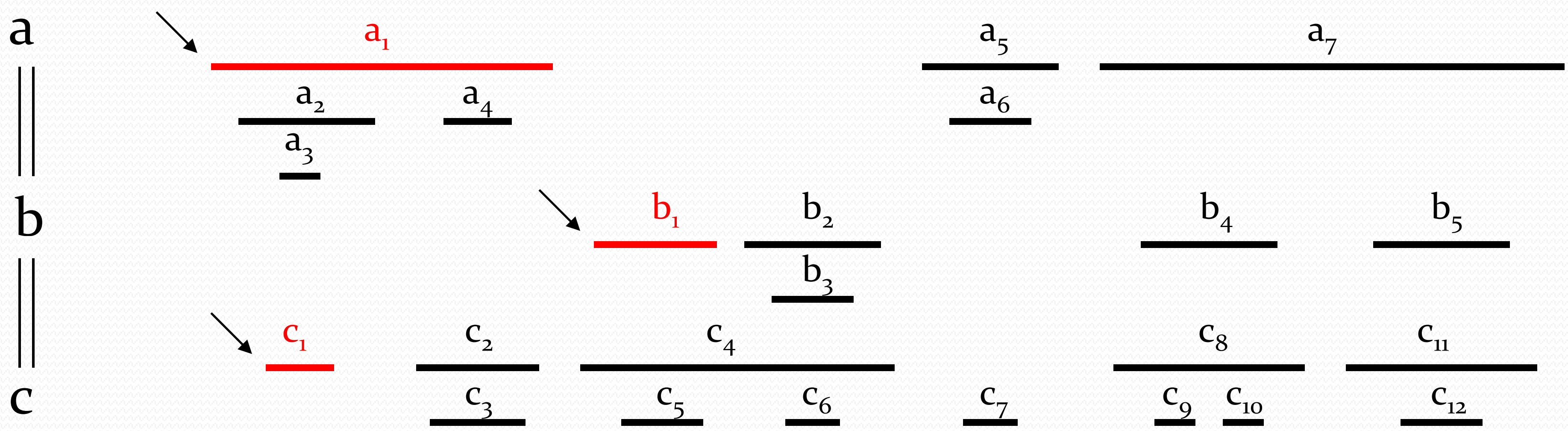
```
if isLeaf(q) then
    return q;
for q_i in children(q) do
    n_i = getNextCursor(q_i);
    if n_i ≠ q_i then
        return n_i;
end for
n_min = minarg_{n_i} {C_{n_i}→start};
n_max = maxarg_{n_i} {C_{n_i}→start};
if C_q→fwdToAncestorOf(C_{n_max}) == TRUE then
    if C_q is an ancestor of C_{n_min} then
        return q;
return n_min;
```

XML Data Sample



getNext(q) (Examples)

getNext(root) = ?



Data Streams and Cursors:

C_a → $a_1, a_2, a_3, a_4, a_5, a_6, a_7$

C_b → b_1, b_2, b_3, b_4, b_5

C_c → $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}$

When can We Jump?

Lemma 1

Suppose a call of **`getNextCursor(root)`** returns a query node q. If the stack S_{qa} of any ancestor q_a of node q is empty, then the current extension of node q does not contribute to any further results and element C_q can be discarded.

The *TSGenrice*$^+$ Algorithm

- A cursor-based structual join algorithm (*SJCursor*)
- *Broken* Edge (p, c): if elements C_p and C_c do not have an ancestor-descendant relationship
- SJCursor: finds the first ancestor-descendant pair starting from the current cursors of the two nodes connected by the edge.

getNextExt(q)

Algorithm 3 getNextCursor(q)

```
if isLeaf(q) then
    return q;
for q_i in children(q) do
    n_i = getNextCursor(q_i);
    if n_i ≠ q_i then
        return n_i;
end for
n_min = minarg_{n_i} {C_{n_i} → start};
n_max = maxarg_{n_i} {C_{n_i} → start};
if C_q → fwdToAncestorOf(C_{n_max}) == TRUE then
    if C_q is an ancestor of C_{n_min} then
        return q;
return n_min;
```

Algorithm 5 getNextExt (q)

```
if isLeaf(q) then
    return q;
if empty(S_q) then
    LocateExtension (q);
    return q;
for q_i in children(q) do
    n_i = getNextExt (q_i);
    if n_i ≠ q_i then
        return n_i;
end for
n_min = minarg_{n_i} {C_{n_i} → start};
n_max = maxarg_{n_i} {C_{n_i} → start};
if C_q → fwdToAncestorOf(C_{n_max}) == TRUE then
    if C_q is an ancestor of C_{n_min} then
        return q;
return n_min;
```

Algorithm 6 LocateExtension (q)

while (**not** end(q)) **and** (**not** hasExtension(q)) **do**
  (p, c) = PickBrokenEdge (q); {see section 4.1}
  SJCursor (p, c);
end while

`Function hasExtension(`q`)`

for each edge (p, c) in the sub query tree q **do**
  if isBroken(p, c) **then**
    return FALSE;
end for
return TRUE;

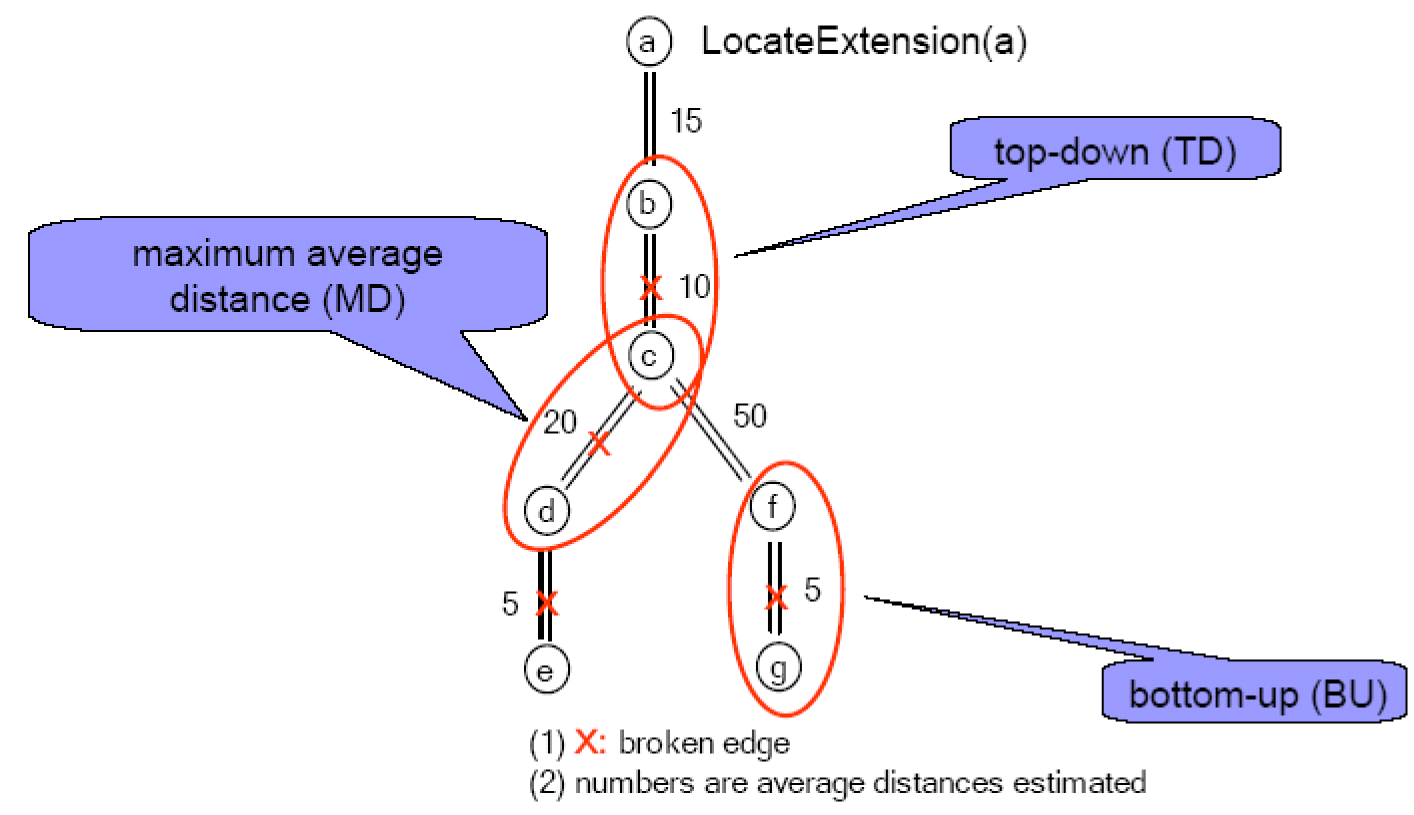
Algorithm 7 PickBrokenEdge (q)

Let Edges[1...K] be the vector containing all K broken edges in q in breadth first order;
if heuristic == MD **then**
  $(p_s, c_s) = \text{maxarg}_{(p_i, c_i)} AvgDist_{p_i \triangleleft c_i}$
else if heuristic == TD **then**
  (p_s, c_s) = Edges[1];
else
  (p_s, c_s) = Edges[K];
return (p_s, c_s);

Heuristics for picking an Edge

- Maximum Distance (MD): choose the edge whose next match is the *farthest* from the current cursors of its two nodes, so that we can skip the most number of edges.
- Top Down (TD): choose the first edge according to the breadth first traversal order
- Bottom Up (BU): choose the last edge according to the breadth first traversal order

An example of MD, TD, BU



SJCursor(*p, c*) Algorithm

Algorithm 4 SJCursor (p, c)

```
while (not end(C_p)) and (not end(C_c))
and isBroken(p, c) do
  if C_p→start < C_c→start then
    C_p→fwdToAncestorOf(C_c);
  else
    C_c→fwdBeyond(C_p);
end while
```

Function isBroken$(p,\ c)$

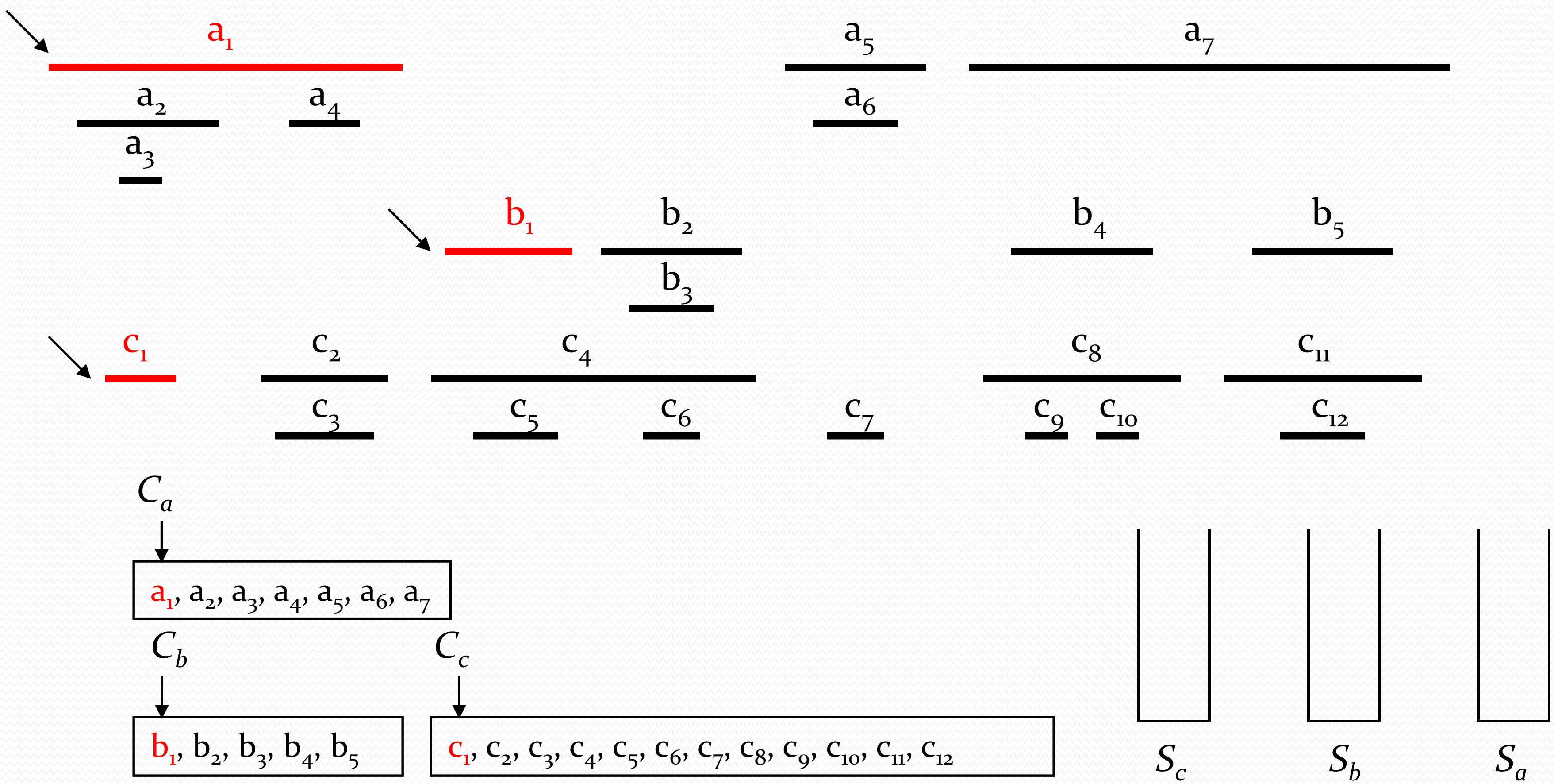
```
return not (C_p→start < C_c→start and C_c→start <
C_p→end);
```

If the edge is not broken, or either C_p or C_c reaches the end, return. Otherwise, proceed below.

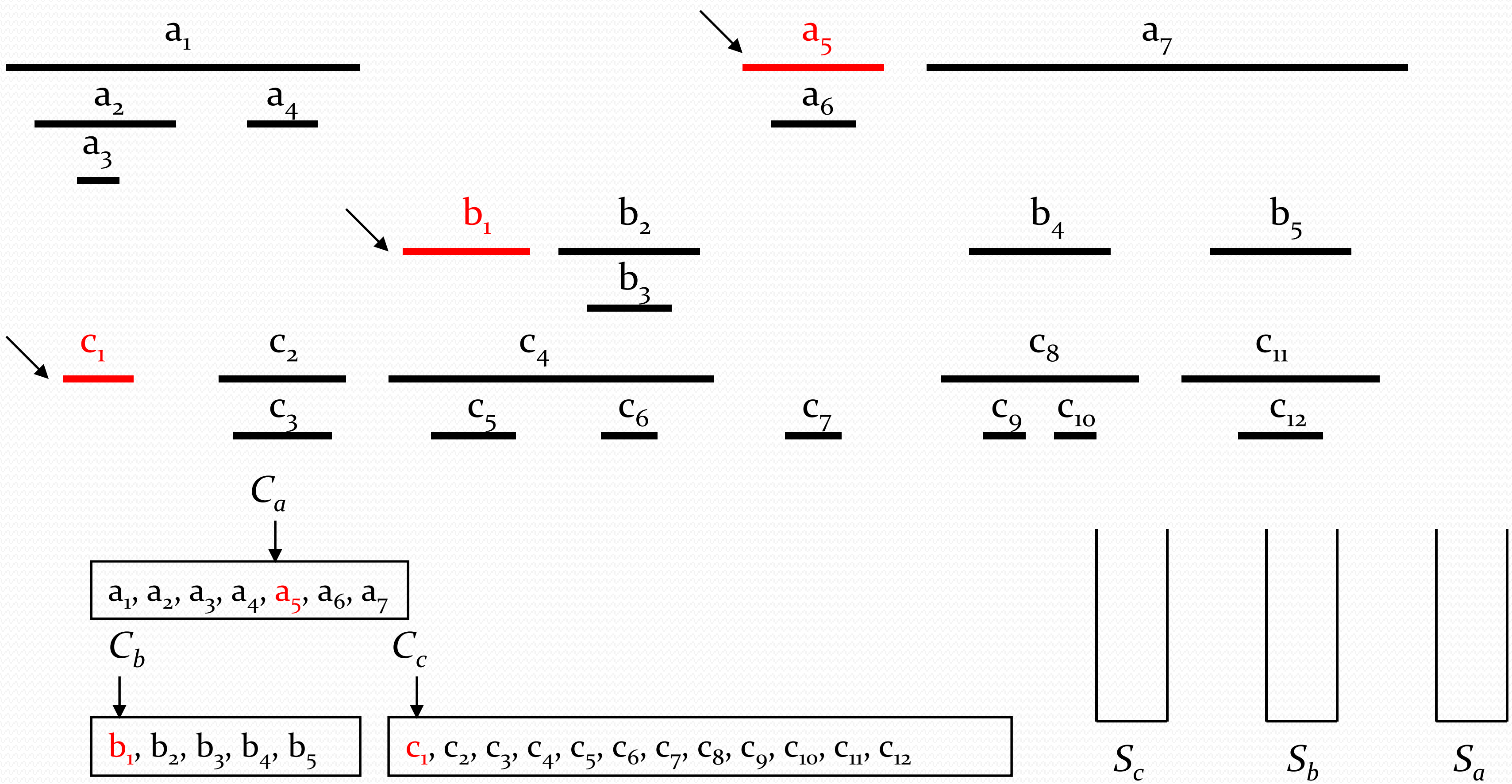
If `C_p->start < C_c->start`, move C_p to the first ancestor element of C_c (or beyond C_c if no such ancestor exists).

Otherwise, forward C_c to the first element whose start value is larger than `C_p -> start`.

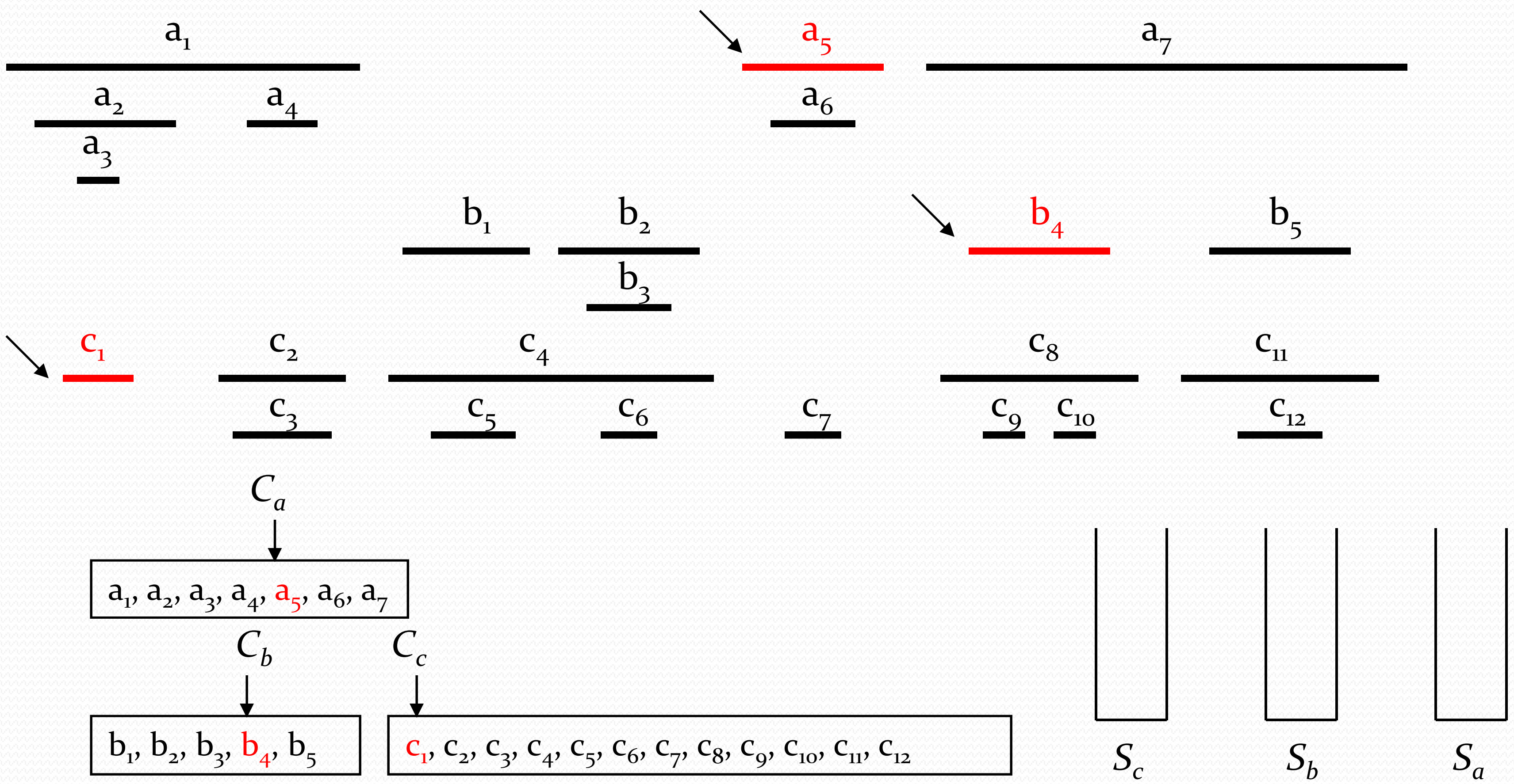
Calling *SJCursor*(*a*, *b*)



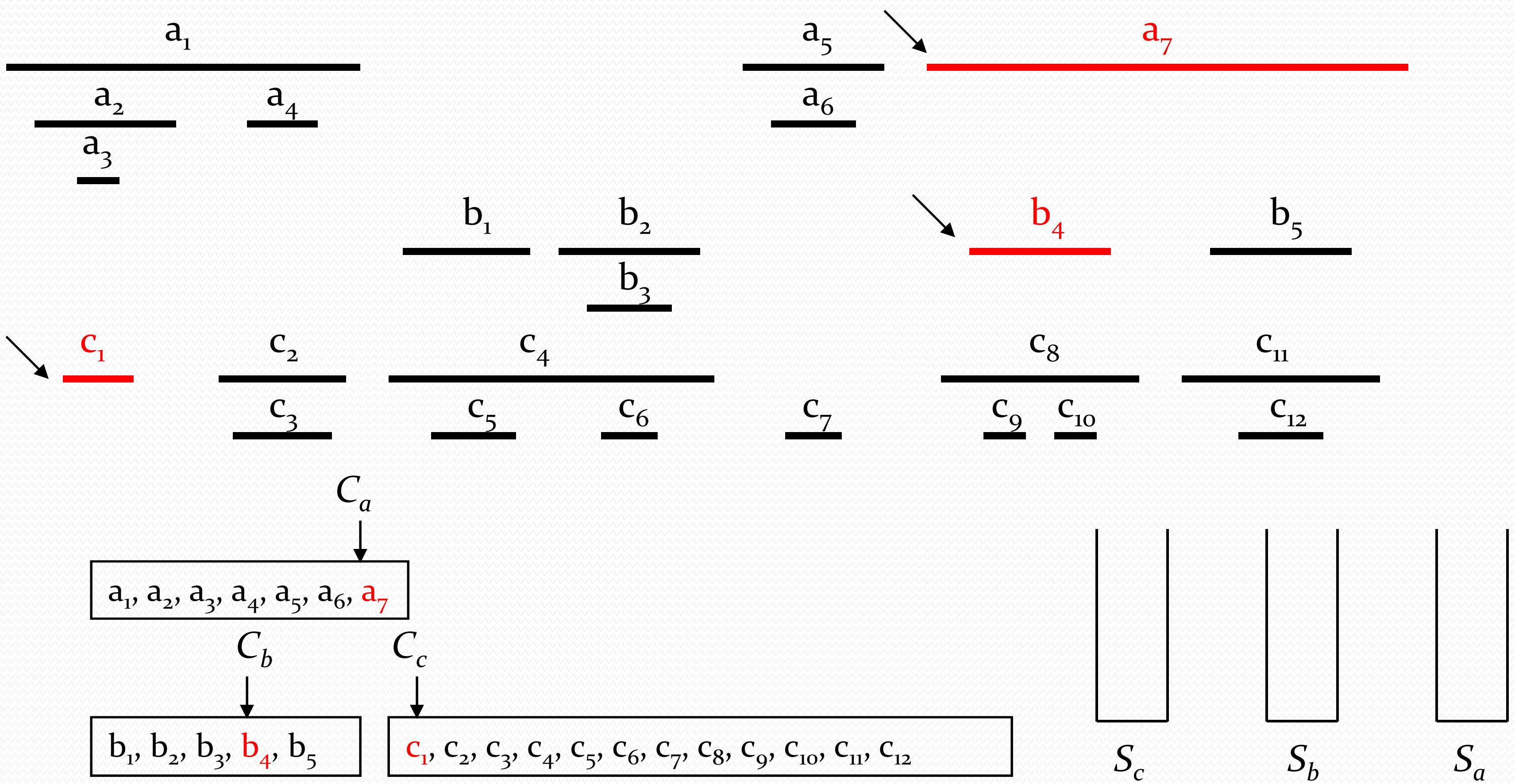
Calling *SJCursor*(*a*, *b*) (2)



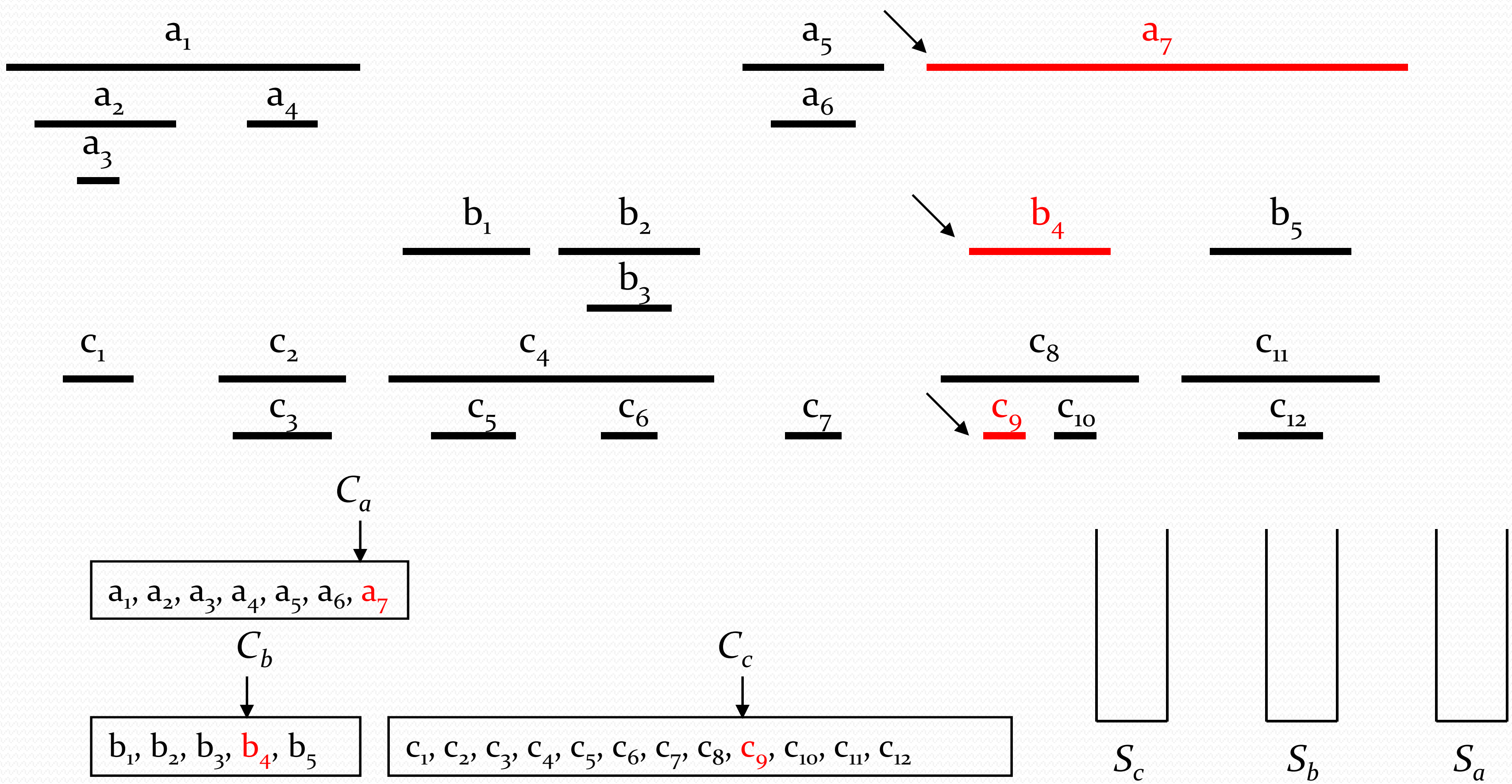
Calling *SJCursor*(*a*, *b*) (3)



Calling *SJCursor*(*a*, *b*) (4)



Calling *SJCursor*(*b*, *c*)



How to Accelerate?

- Algorithm:
  - TSGeneric^{+}: giving more opportunity to jump
- Index:
  - TR-tree: jumping faster and further each time

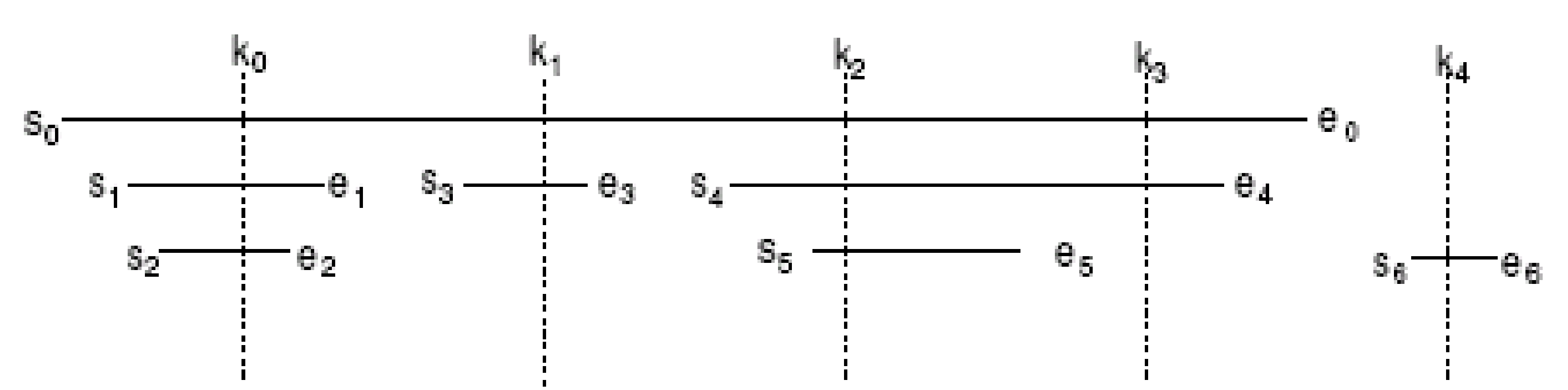
XR-tree

- XML Region Tree (Jiang *et al.*, *ICDE* 2002)
- Based on B^+-Tree (based on the start position of each element $E_i(s_i, e_i)$, i.e. s_i).
- Extended internal nodes with **stab lists** and bookkeeping information.
- Nice property: given an element, all its ancestors and descendents can be identified very efficiently.

Stab

- Element with region $E_i(s_i, e_i)$; Key k
- E_i is said to be **stabbed** by k, or k **stabs** E_i ⇔ $s_i \le k \le e_i$
- A set of ordered keys $k_j (0 \le j < n)$ where $k_x < k_y$ if $x < y$.
- E_i is said to be **primarily stabbed** by k_j, or k_j **primarily tabs** E_i: k_j is the smallest key that stabs E_i among a set of ordered keys.
- The **(primary) stab list** of a key k_j is the list of elements that are (primarily) stabbed by k_j, denoted as $(P)SL_j$ or $(P)SL_{(k_j)}$.

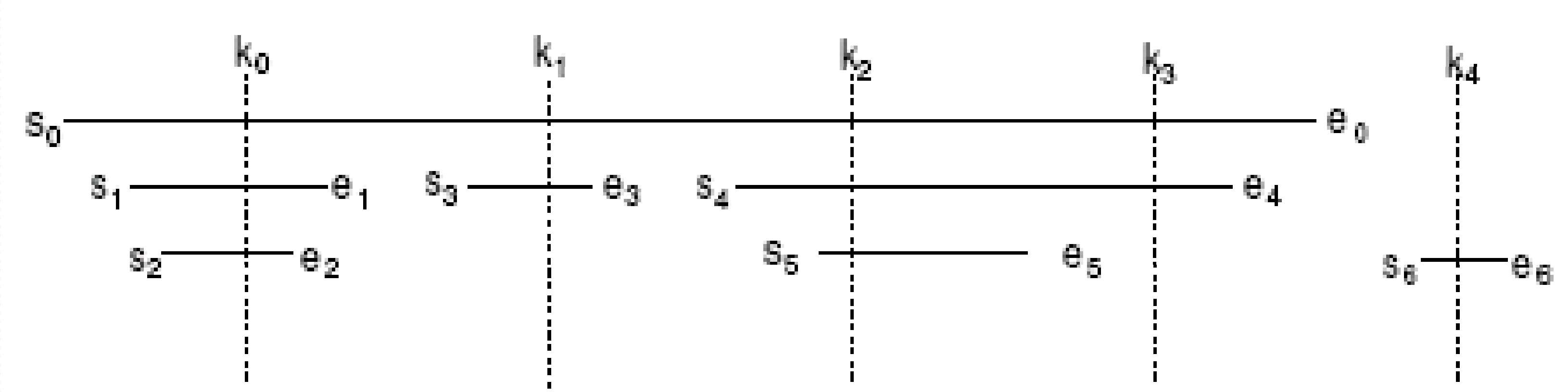
Example of Stab, Stab Lists



- $SL_0=\{E_0, E_1, E_2\}$; $PSL_0=\{E_0, E_1, E_2\}$
- $SL_2=\{E_0, E_4, E_5\}$; $PSL_2=\{E_4, E_5\}$
- $SL_3=\{E_0, E_4\}$; $PSL_3=\phi$

Internal Nodes

- The start and end position ps_j, pe_j, of a key k_j, are defined as the start and end position of the first element in the primary stab list of k_j, if not empty; or (*nil*, *nil*) if empty.



- (k_0, s_0, e_0), (k_1, s_3, e_3), (k_2, s_4, e_4),
- (k_3, nil, nil), (k_4, s_6, e_6).

- An element *e* is included in the **stab list of an index page *I*** if:
- (1) there exists some key *k* in *I* such that *e.start* <= *k* <= *e.end* (or *k* stabs the region of element *e*); and
- (2) no ancestor page of *I* has a key that stabs *e*, i.e. *I* is the highest index page that stabs *e*.

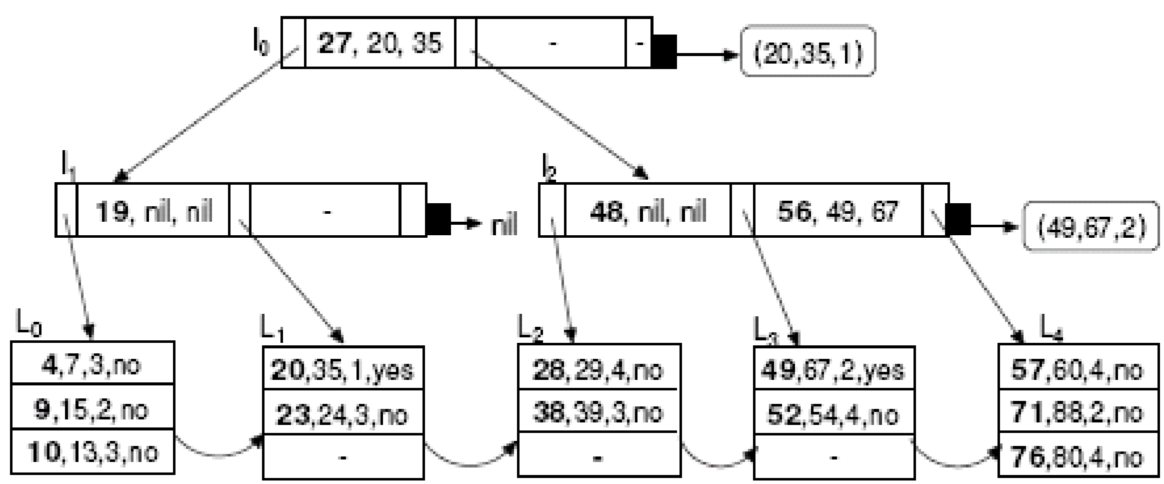


Figure 4: The XR-tree for *c* elements in Figure 1

Search for all descendants

- B^+-tree is based on the start position of each element.
- Equivalent to B^+-tree range search for *e.start<R.start<e.end* (elements do not have overlaps).

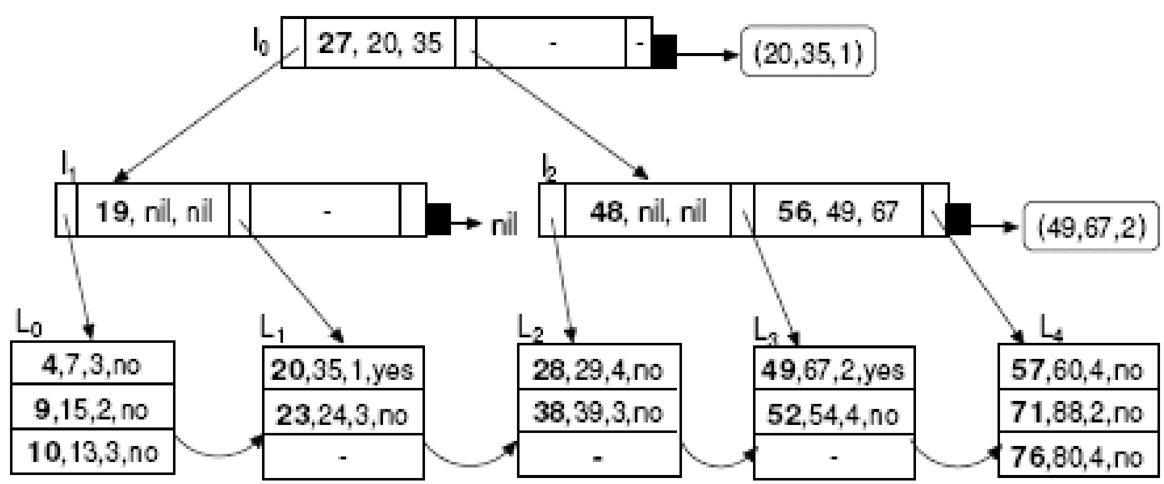


Figure 4: The XR-tree for c elements in Figure 1

Search for all ancestors

- All the ancestors of *e* can be collected from the stab lists of index pages and the leaf page when we navigate down the XR-tree using *e.start*

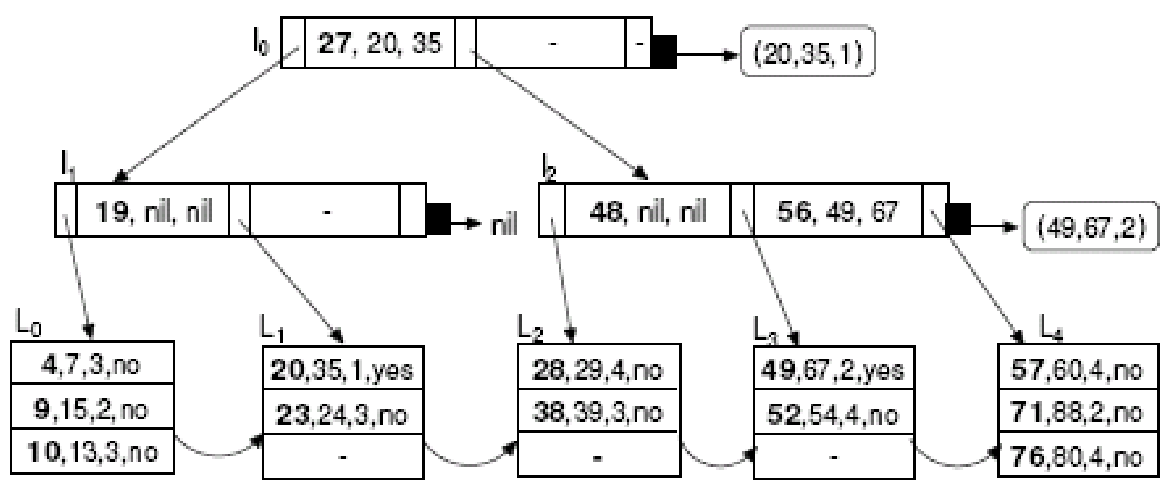


Figure 4: The XR-tree for *c* elements in Figure 1

Cursor interfaces

- C_q->advance()
- C_q->fwdBeyond(C_p)
- C_q->fwdToAncestorOf(C_p)

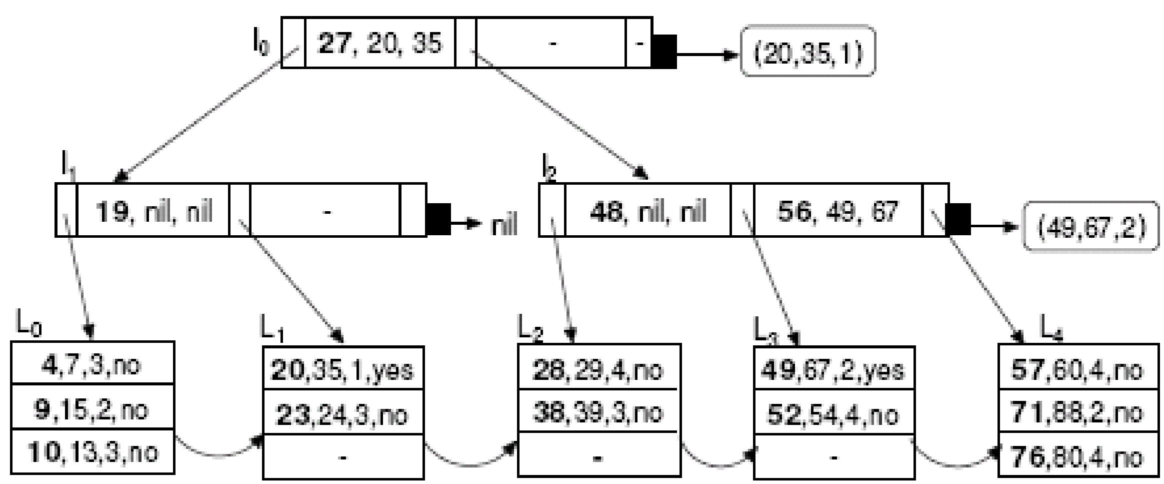


Figure 4: The XR-tree for c elements in Figure 1

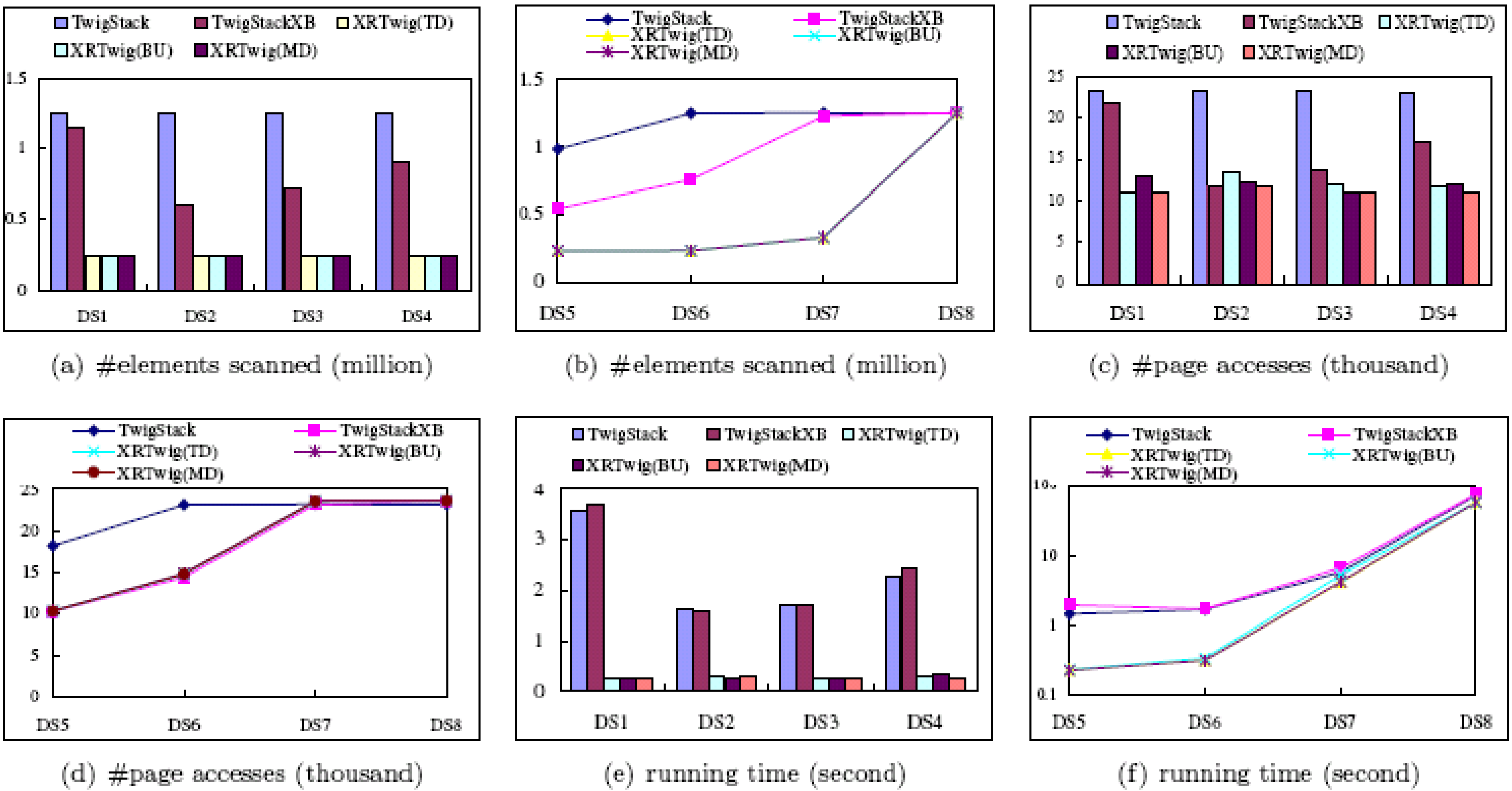
Performance of XR-tree

- Space: linear in the size of the XML document
- Time
  - h: B^+-tree heights; R: result size; B: block size
  - Search for all descendants: $O(h+R/B)$ in the worst case
  - Search for all ancestors: $O(h+R)$ in the worst case
  - Insert/delete: $O(h+c)$, amortized

Performance Study

- TwigStack, using TSGeneric
  - TwigStack (with no Index)
  - TwigStackXB (TwigStack with XB-tree index)
- XRTwig, using TSGeneric+ and XR-tree index
  - XRTwig(TD)
  - XRTwig(BU)
  - XRTwig(MD)

	TSGeneric	TSGeneric+
No Index	TwigStack	
XB-tree	TwigStackXB	
XR-tree		XRTwig (TD) XRTwig (BU) XRTwig (MD)



(a) #elements scanned (million)
(b) #elements scanned (million)
(c) #page accesses (thousand)
(d) #page accesses (thousand)
(e) running time (second)
(f) running time (second)

Figure 6: Experimental results for query Q1

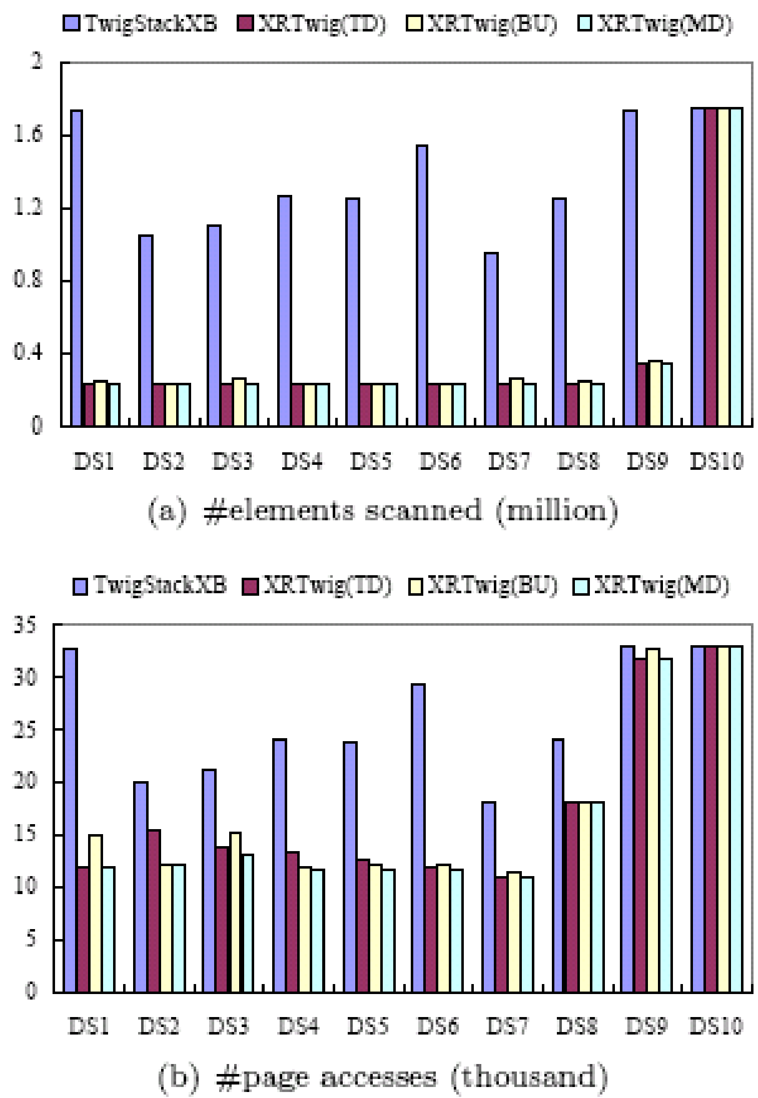


Figure 7: Experimental results for query Q2

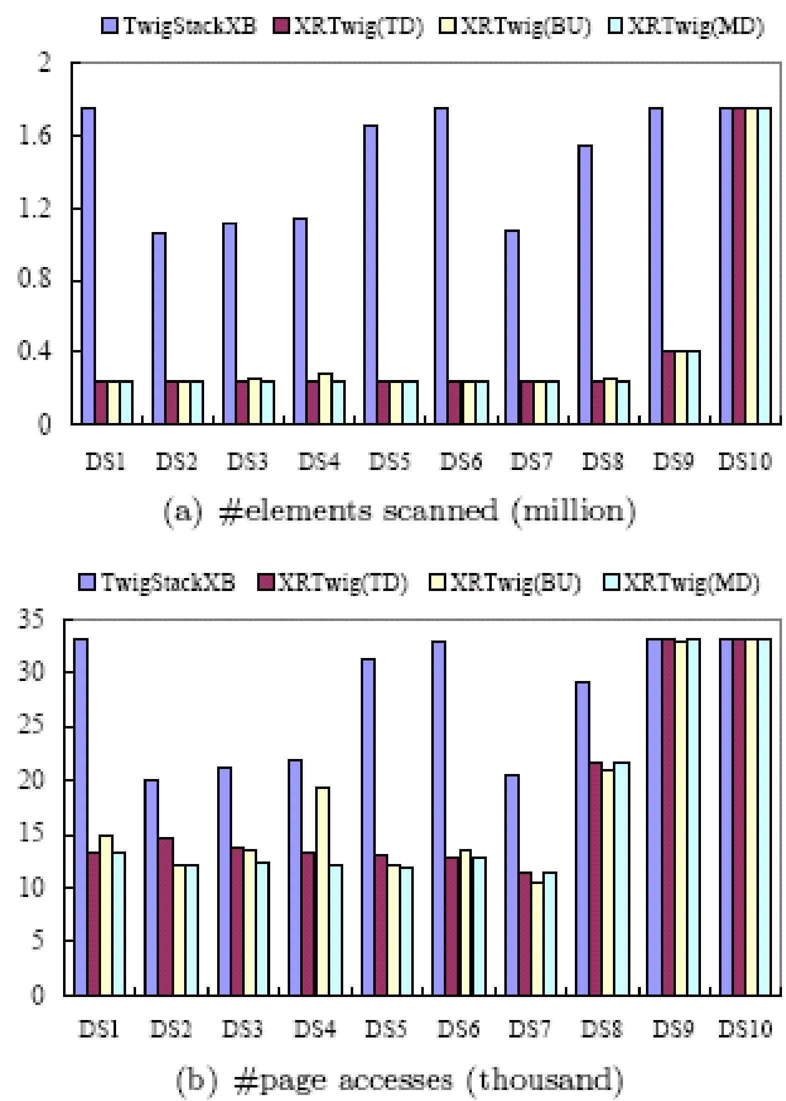


Figure 8: Experimental results for query Q3

Comparison of heuristics

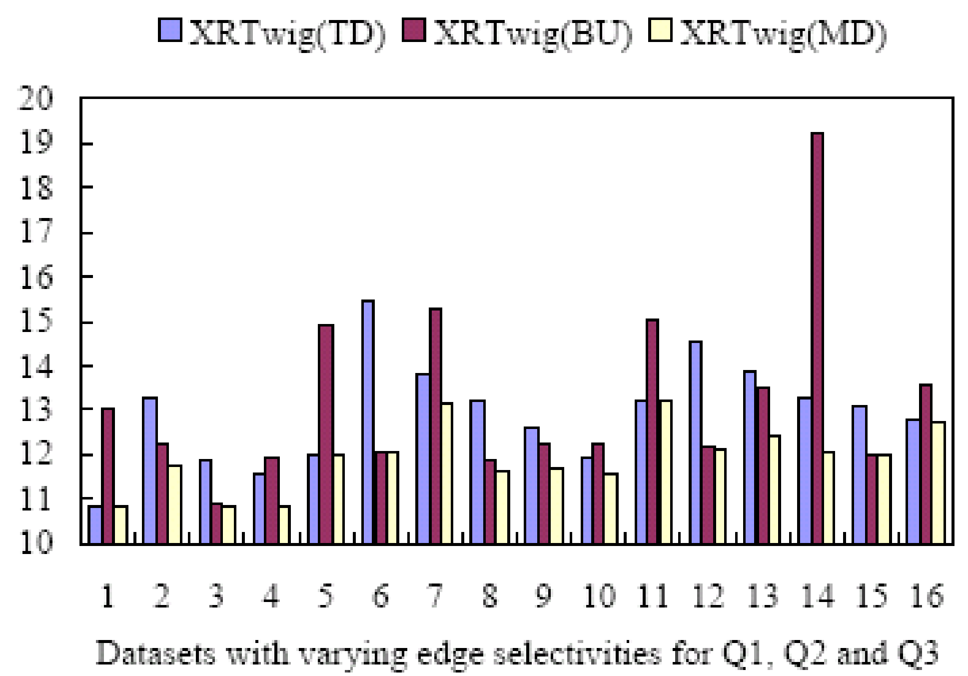


Figure 9: #Page accesses under different edge-picking heuristics (thousand)

Thank you!

Questions?