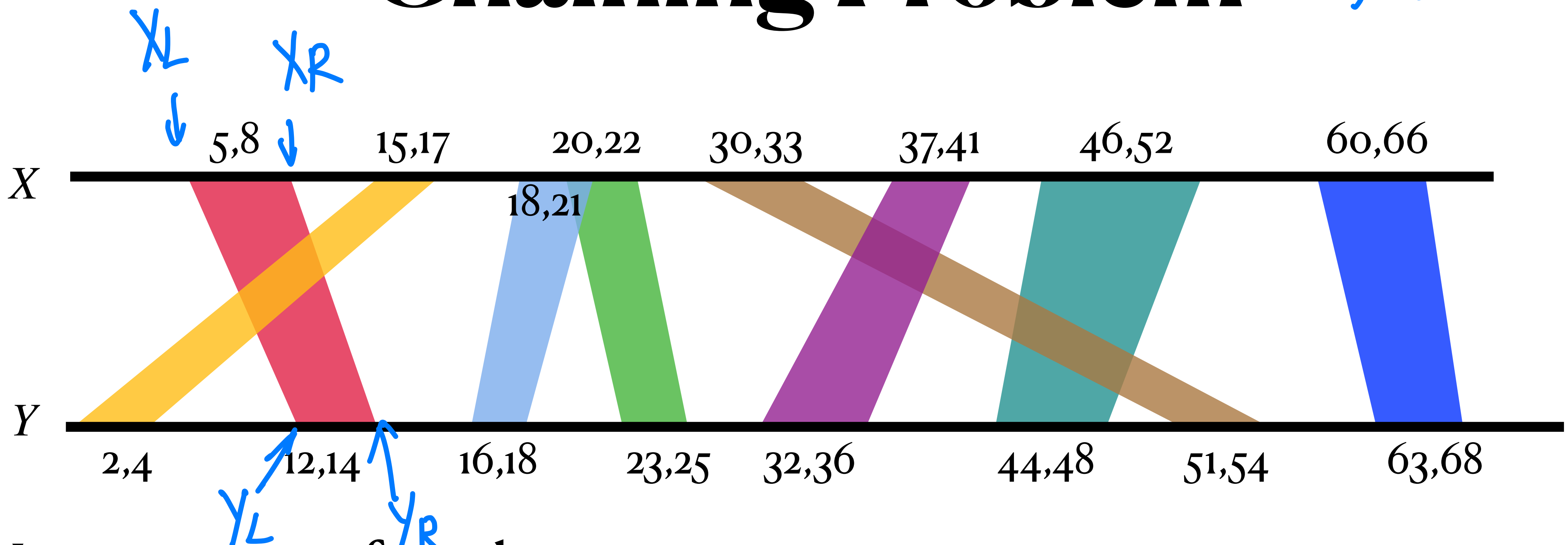
CSE 566 Spring 2023

Chaining for Sequence Alignment

Instructor: Mingfu Shao

Chaining Problem

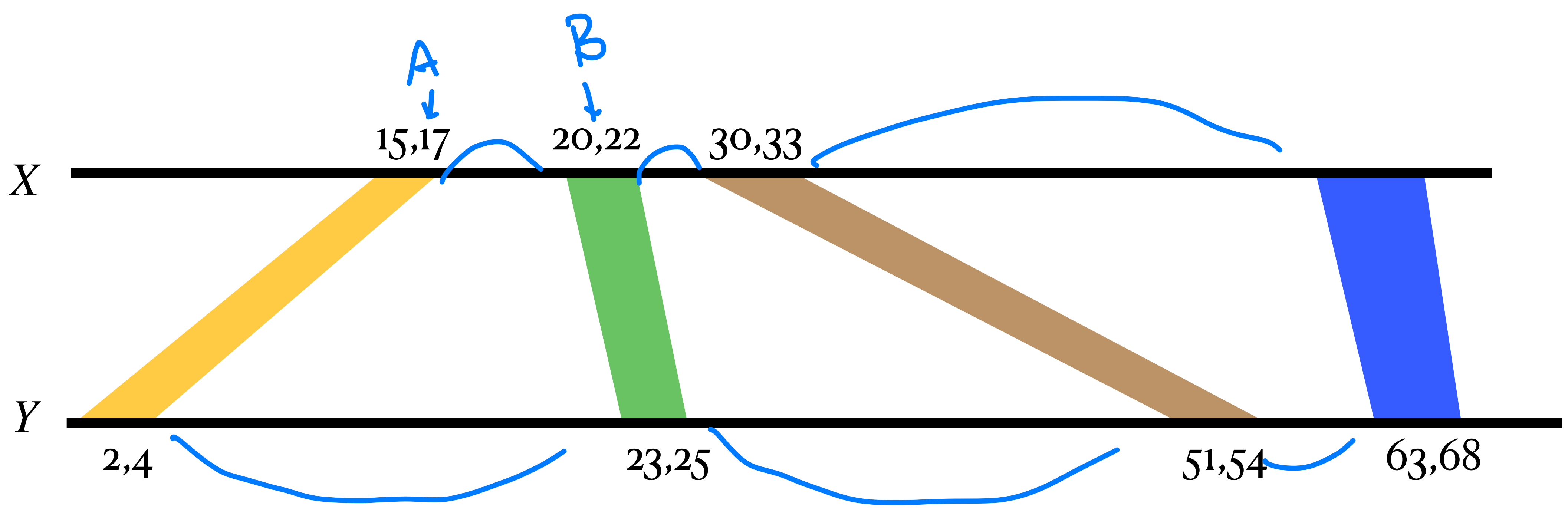
seed-&-extend



- Input: a set of anchors
- Output: a chain of anchors so that its score is maximized
- Define: anchors $A < B$ if $A.x_R < B.x_L$ and $A.y_R < B.y_L$
- Define: a list of anchors $(A_1, A_2, \cdots, A_k)$ forms a chain if $A_1 < A_2 < \ldots < A_k$

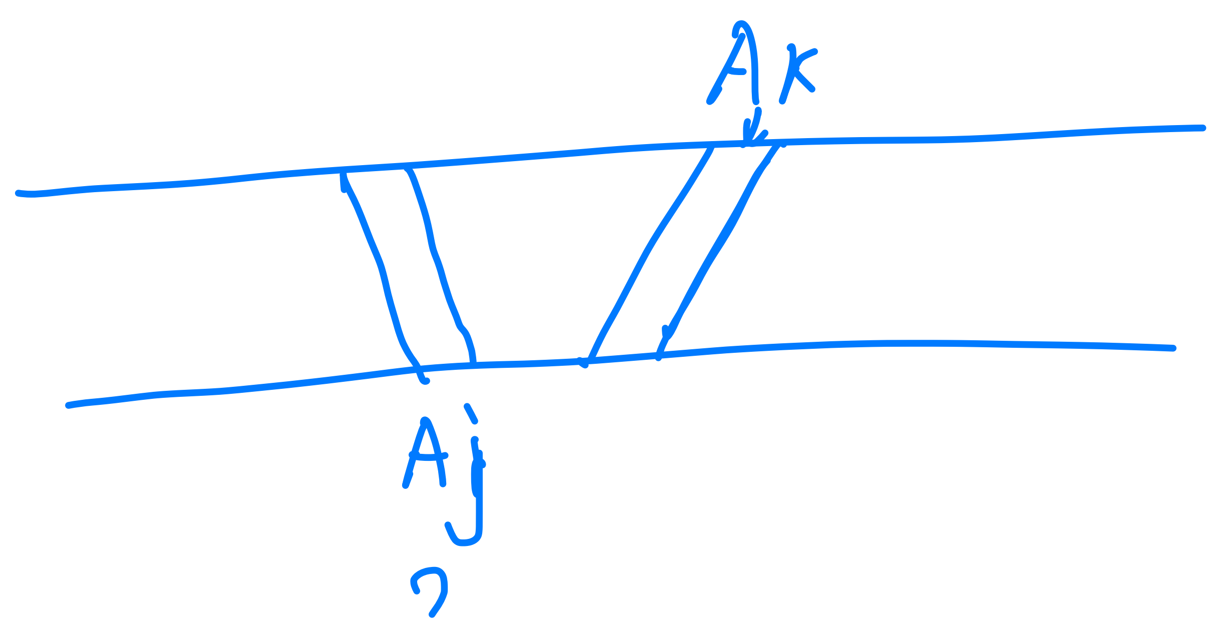
Scoring Function

- Let $C = (A_1, A_2, \cdots, A_k)$ be a chain
- $f(C) = \sum_{i=1}^{k} \text{score}(A_i) + \sum_{i=1}^{k-1} \text{gap-cost}(A_i, A_{i+1})$
- The gap-cost is letter independent:
  $\text{gap-cost}(A, B) = \lambda(B.x_L - A.x_R + B.y_L - A.y_R)$



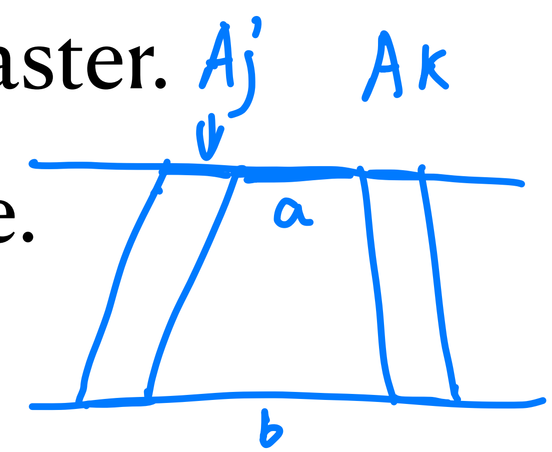
Dynamic Programming Algorithm

- Sort all anchors according to x_L: $(A_1, A_2, \cdots, A_n)$
- Define $OPT(k)$ as the score of the optimal chain in the first k anchors, where A_k must appear in the chain.
- $OPT(k) = \text{score}(A_k) + \max\limits_{j: A_j < A_k} (OPT(j) + \text{gap-cost}(A_j, A_k))$
- Running time: $O(n^2)$



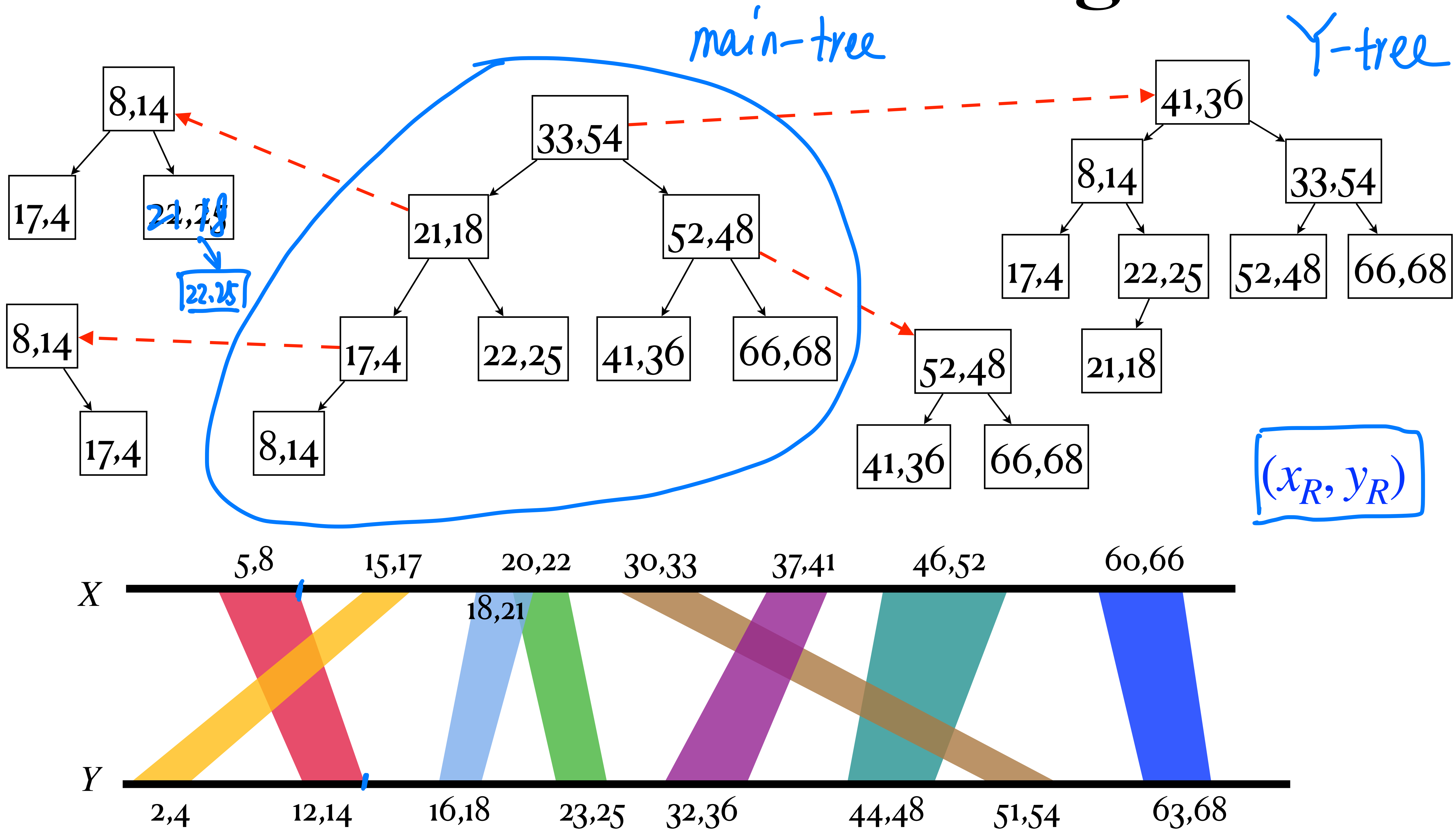
Improved Algorithm

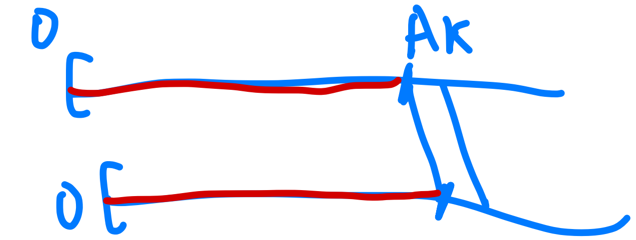
- The same framework: find $OPT(k)$, for $k = 1,2,\cdots,n$
- Key: find $\max\limits_{j:A_j<A_k} \left(OPT(j) - \lambda(A_j\,.\,x_R + A_j\,.\,y_R)\right) := \max\limits_{j:A_j<A_k} OPT_\lambda(j)$
- Idea #1: use a 2D range tree to fetch $\{j : A_j < A_k\}$ faster.
- Idea #2: store/update the max-value in each subtree.



$$
\begin{aligned}
OPT(k) &= \text{score}(A_k) + \max_{j:A_j<A_k}(OPT(j) + \text{gap-cost}(A_j, A_k)) \\
&= \text{score}(A_k) + \max_{j:A_j<A_k}(OPT(j) + \lambda(A_k\,.\,x_L - A_j\,.\,x_R + A_k\,.\,y_L - A_j\,.\,y_R)) \\
&= \text{score}(A_k) + \lambda(A_k\,.\,x_L + A_k\,.\,y_L) + \max_{j:A_j<A_k}(OPT(j) - \lambda(A_j\,.\,x_R + A_j\,.\,y_R)) \\
&= \text{score}(A_k) + \lambda(A_k\,.\,x_L + A_k\,.\,y_L) + \max_{j:A_j<A_k} OPT_\lambda(j)
\end{aligned}
$$

Store Anchors with 2D Range Tree



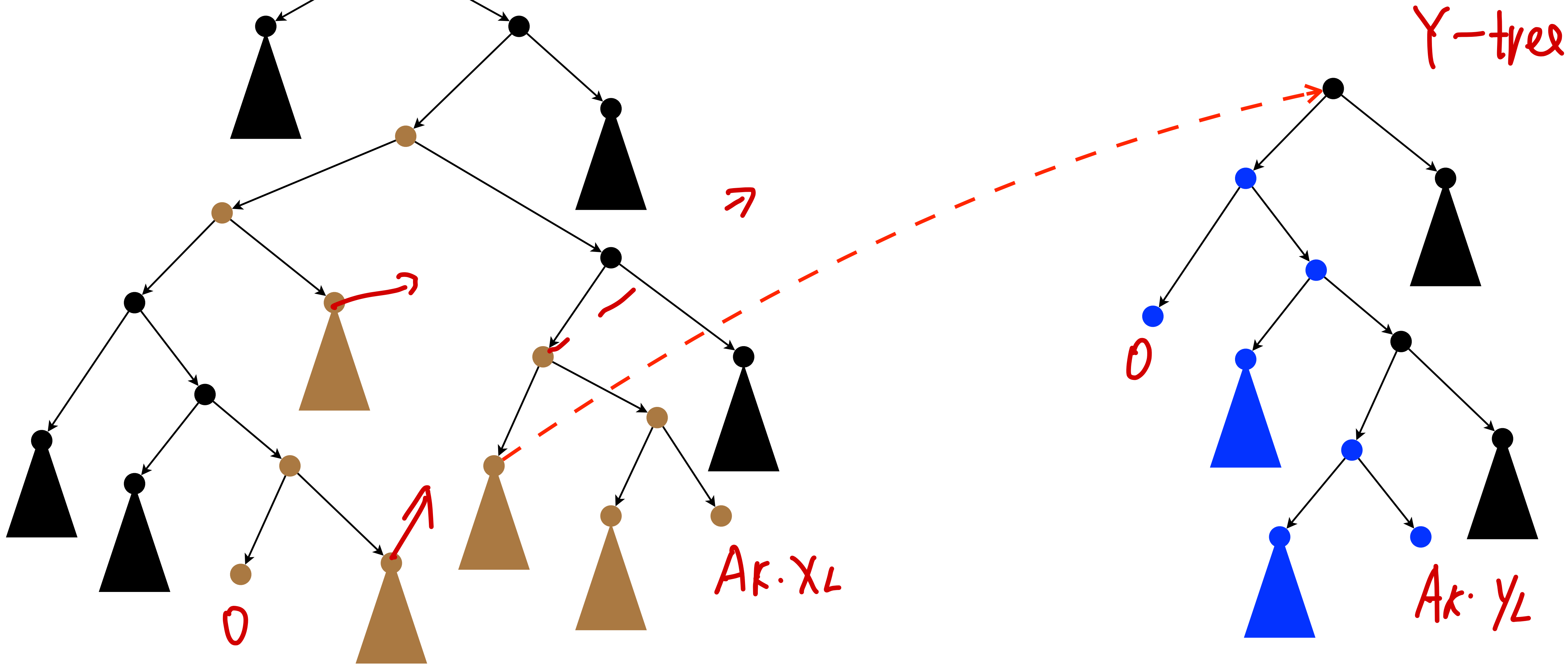


Query Optimal Previous Anchor

- For the current anchor A_k, feasible previous anchor, i,e., $\{A_j \mid A_j < A_k\}$, can be queried by range $[0, A_k . x_L)$, $[0, A_k . y_L)$
- Cannot afford calculating $OPT_\lambda(j)$ for every $\{A_j \mid A_j < A_k\}$.
- Note that $\{A_j \mid A_j < A_k\}$ are represented by a set of nodes, $O(\log^2 n)$ of them, and a set of subtrees, $O(\log^2 n)$ of them.
- Idea: for each node v store $M(v)$, defined as the maximum OPT_λ for all nodes in the subtree rooted at v.
- Then, $\max_{j : A_j < A_k} OPT_\lambda(j)$ can be found in $O(\log^2 n)$ time!

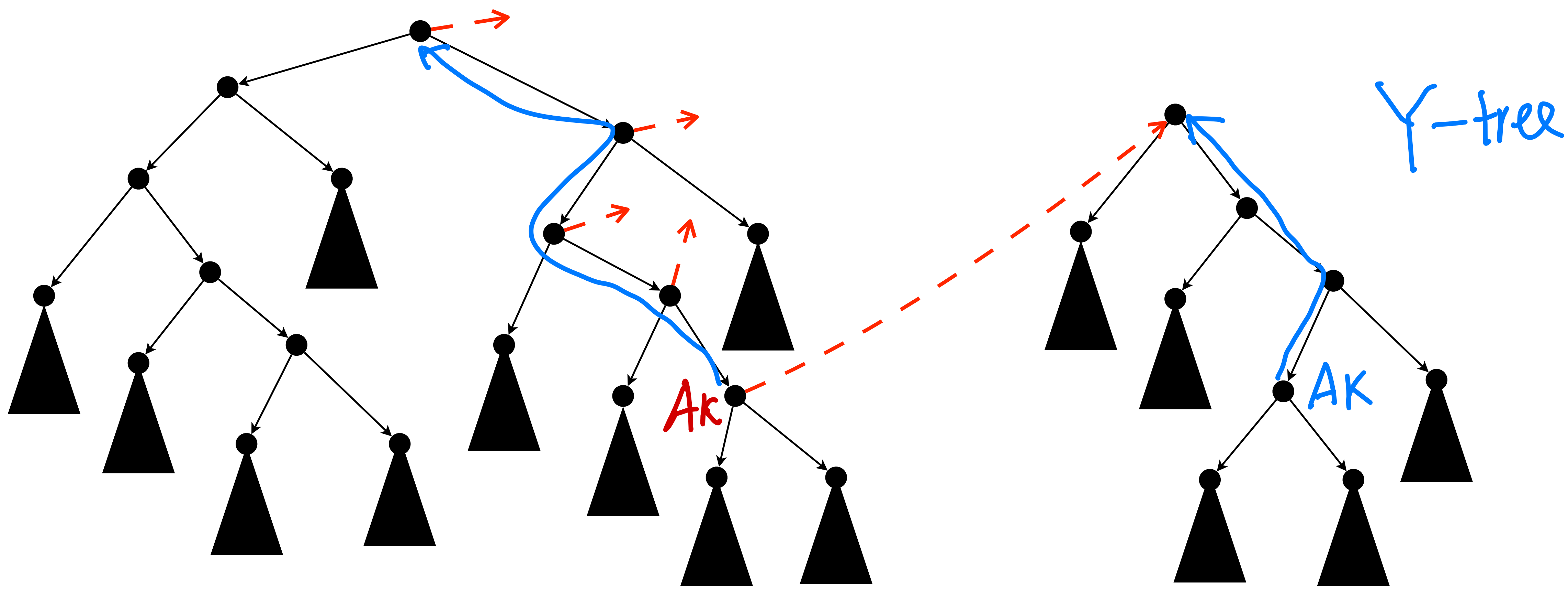
An Example

$M(v) = \max_{u \in T_v} OPT_\lambda(u)$



Update $M(v)$

- After getting $OPT(k)$ and $OPT_\lambda(k)$, we need to update $M(v)$, for every node in each Y-tree that involves A_k.
- #nodes need to update: $O(\log^2 n)$.



Complete Algorithm

Build 2D range tree for all anchors using x_R and y_R $O(n \cdot \log n)$

For every node v in every Y-tree, init $M(v) = -\infty$

For each A_k in ascending order of x_L

  Query $[0, A_k.x_L), [0, A_k.y_L)$ -> a list nodes and subtrees $O(\log^2 n)$

  Scan the nodes to find $\max_j OPT_\lambda(j)$

  Scan the roots of the subtrees to find $\max_v M(v)$

  Take the ~~minimum~~ maximum of above two which gives $\max_{j:A_j<A_k} OPT_\lambda(j)$

  Calculate $OPT(k) = \text{score}(A_k) + \lambda(A_k.x_L + A_k.y_L) + \max_{j:A_j<A_k} OPT_\lambda(j)$

  Calculate $OPT_\lambda(k) := OPT(k) - \lambda(A_k.x_R + A_k.y_R)$

  Find A_k in the main tree

  For each node on the path from A_k to the root in the main tree:

    follow the link to reach Y-tree and find A_k in the Y-tree

    Update $M(v)$ for each v on the path from A_k to the root

end For   if $M(v) < OPT_\lambda(k)$: $M(v) \leftarrow OPT_\lambda(k)$.

Analysis

- Running time: $O(n \log^2 n)$
- Space complexity: $O(n \log n)$
- Can be generalized to chaining ~~k~~ d sequences:
    - Time complexity $O(n \log^d n)$
    - Space complexity

$$O(n \log^{d-1} n).$$

More Chaining Algorithms

- Sparse dynamic programming. I: Linear cost functions; II: Convex and concave cost functions. (1992)
- Chaining multiple-alignment fragments in subquadratic time (1995)
- Chaining algorithms for multiple genome comparison (2004)
- Algorithms for Colinear Chaining with Overlaps and Gap Costs (2022)

RMQ data structure