Kinetic Data Structures

Pawel Winter

Content – 4 Hours

- Motivation
- General Framework
- Performance Issues
- Maximum
- Sorted List
- Convex Hull
- Delaunay Triangulation
- Alpha Complexes
- Application to Protein Folding

Motivation

- Maintain a configuration of moving objects.
- Each object has a flight plan.
- Applications:
  - Collision detection in robotics
  - Animation
  - Physical simulation
  - Mobile and wireless networks

Motion

- $p(t) = (x(t),y(t))$: Position of p at time t in [0,1].
- $x(t)$, $y(t)$ are polynomials in t.
- Degree d of motion: max degree of $x()$, $y()$.
- Linear motion: $d = 1$.
- Trajectory of objects can change, for example it can be piecewise linear.
- Objects can be added and/or deleted at any time.
- To keep things simple we will most of the time assume that all trajectories are linear and that all objects exist between $t_{\text{start}} = 0$ and $t_{\text{end}} = 1$.

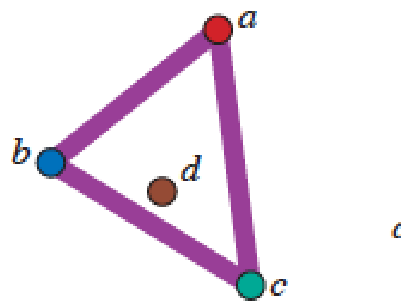
Dealing with Motion - Alternatives

- Brute Force: Fix a time step Δt. Recompute everything from scratch every Δt.
- Dynamic: If only a fraction of objects moves, delete these objects after each time step and insert them at their new positions. Need for dynamic updates (deletions and insertions). Problems if Δt too big or too small.
- Kinetic: Do the updating only when the combinatorial structure changes. Need to bookkeep when such changes may occur.

General Framework

- The task is to maintain a data structure of a continuously moving (subset of) objects.
- A set of certificates validates the combinatorial structure of current configuration.
- A failure of a certificate at a given time indicates an event where the combinatorial structure changes.
- Events are stored in a heap, ordered by time.
- For a given event, data structure and certificate updates are carried out. New events corresponding to failures of new certificates are inserted into the heap.

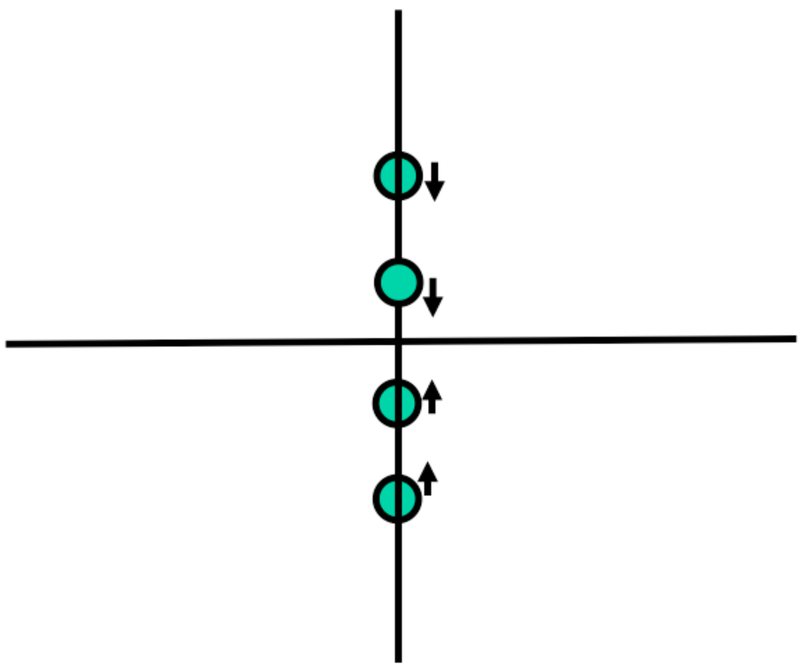
General Framework



- Events occur when one of the certificates seizes to hold. Data structure needs to be updated, new certificates must be added, their failure times need to be determined.

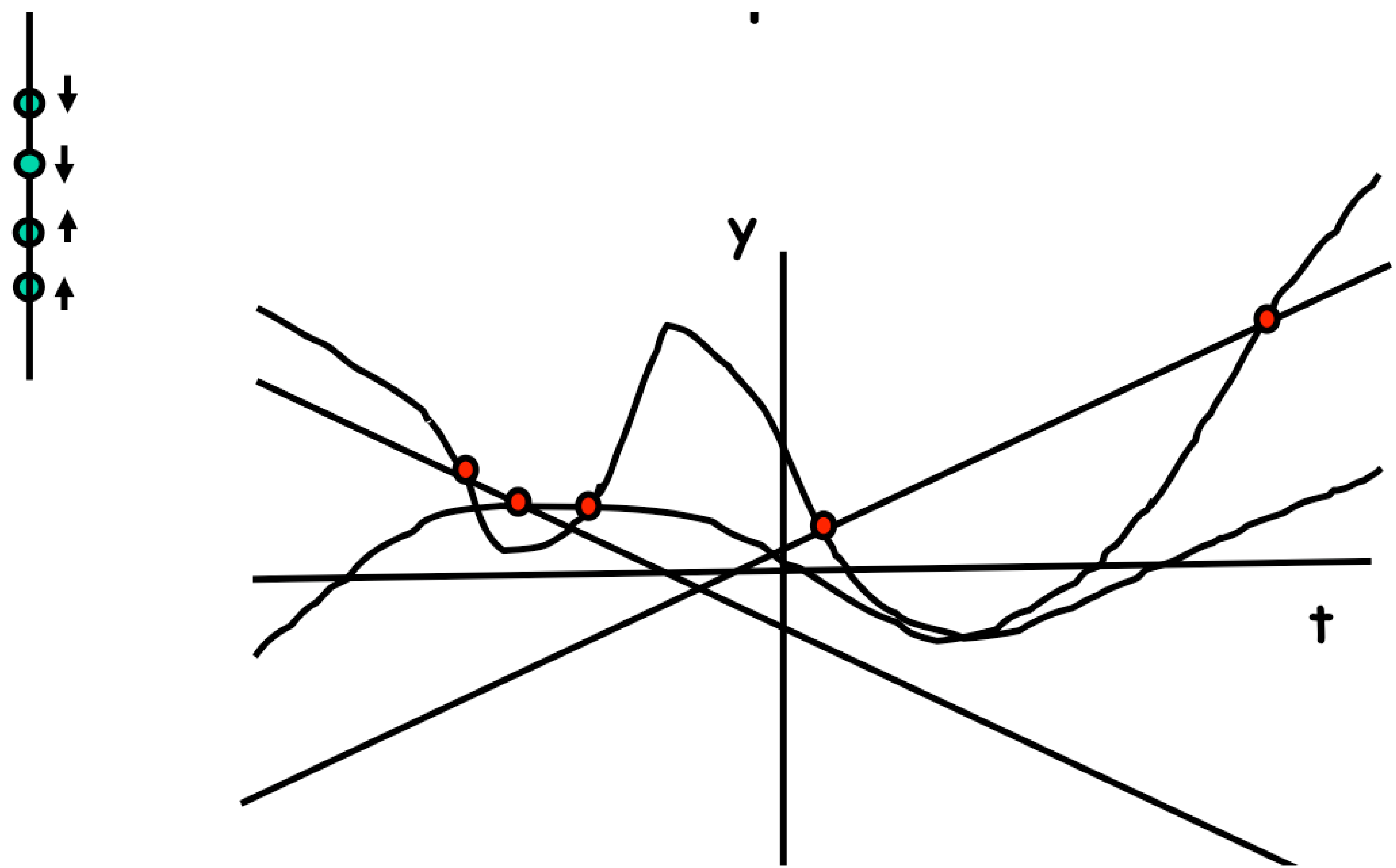
- Certificates:
  - *c* to the left of *ab*
  - *d* to the left of *ab*
  - *a* to the left of bc
  - *d* to the left of *bc*
  - *b* to the left of *ca*
  - *d* to the left of *ca*
- Data structure:
  - Circular double-linked list of CH vertices

Maximum Maintenance

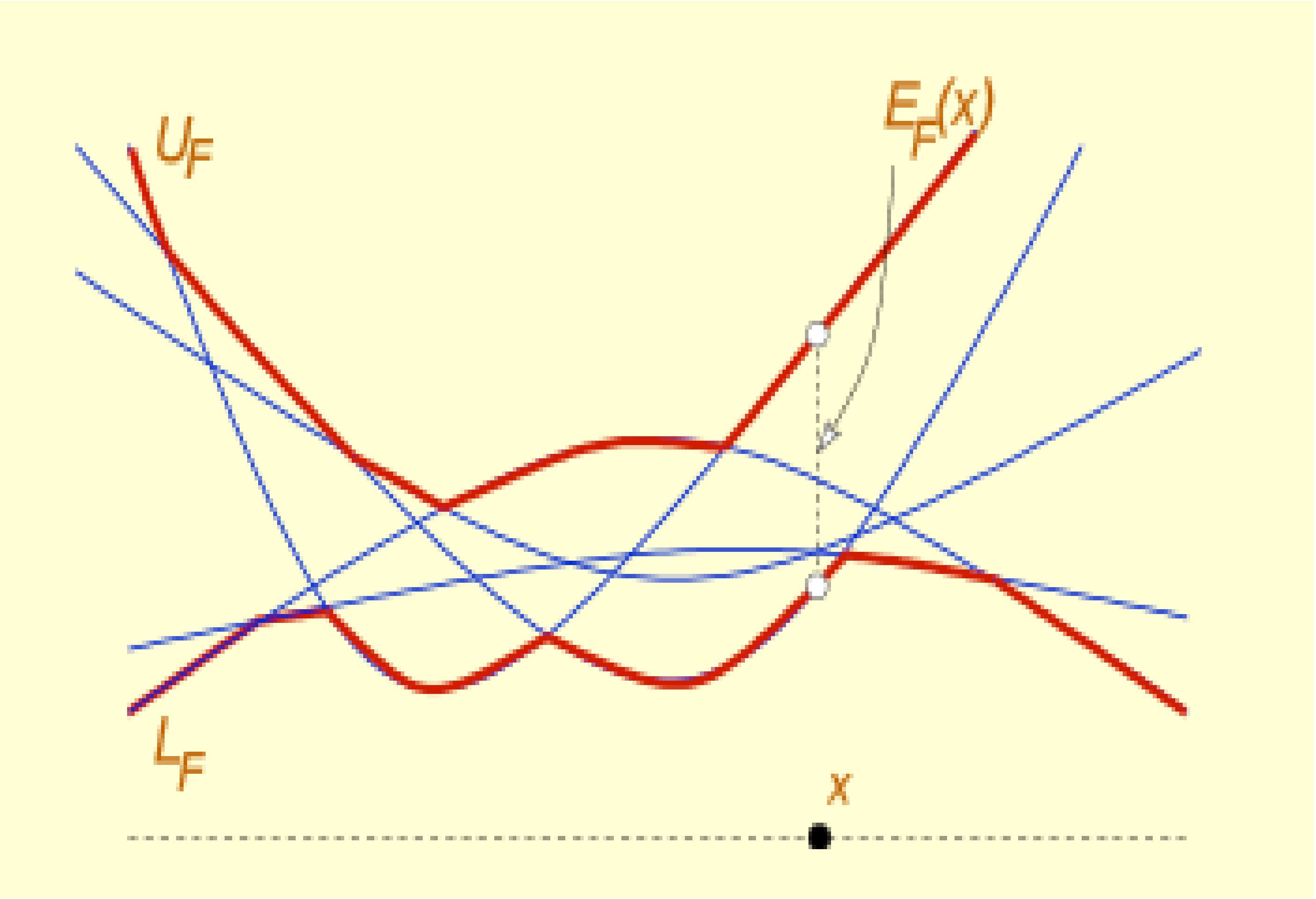


- Points move up and down in some more or less complicated manner.
- Manner of movement (e.g., direction, speed, acceleration) can change. The time of next change is known.
- We want to keep track of the top point.

Upper Envelope



Upper and Lower Envelopes



Maximum
Upper Envelope Approach?

- If trajectories are linear, then the size of the upper envelope is O(n).
- What certificates are needed? Events?
- Upper envelopes can be determined in O($n \log n$) time. How?
- Similar bounds can be derived if every pair of trajectories intersects at most s times, for some fixed integer s.
- Problem: Change of a single trajectory requires the recomputation of the entire upper envelope.

Upper Envelope in O($n$log$n$) time

- Use divide-and-conquer with merging requiring O(n) time.
- Use plane sweep.

Maximum
Maintain a Sorted List?

- $\Omega(n^2)$ events but only $O(n)$ configuration changes (new top element).
- A kinetic data structure is said to be efficient if the ratio between certificate failures (internal and external events) and the number of configuration changes (external events) is $O(\log^c n)$ where c is a constant.
- Sorted list is not an efficient kinetic data structure for Maximum Maintenance.

What Else Can Be Measured?

- Responsive: Failing certificates can be processed in $O(\log^c n)$ where c is a constant.
- Compact: Number of certificates should be $O(n\log^c n)$ where c is a constant.
- Local: Each object is in $O(\log^c n)$ certificates where c is a constant. So when an object changes its trajectory, not too many certificates are affected.

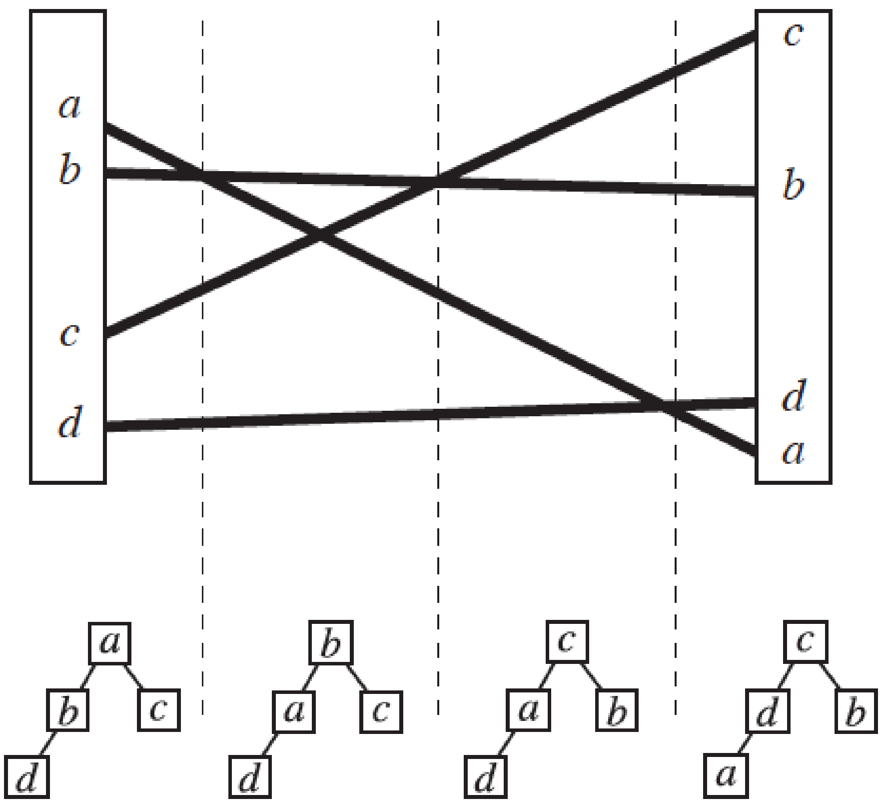
Maximum Maintenance Using Upper Envelope

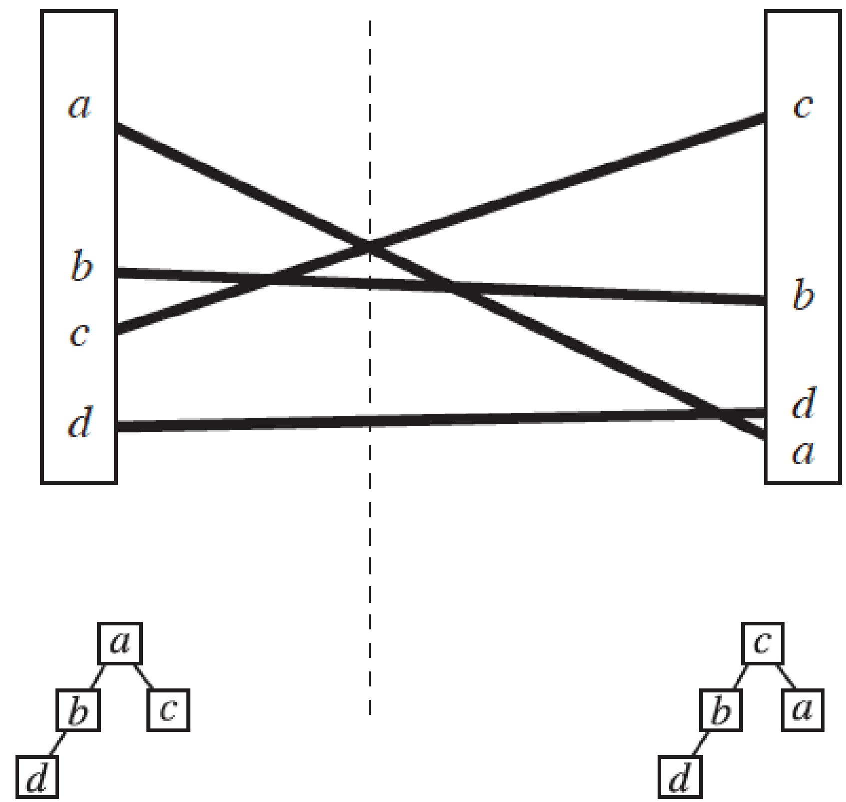
- Keep upper envelope with the information which point is maximum explicitly given
  - Responsive: Yes
  - Compact: Yes
  - Local: No
  - Efficient: Yes

Maximum Maintenance Using Sorted List

- Keep a certificate for each pair of neighbouring points. Events occur when points swap.
- Responsive: Yes,
  - swap in O(1) time,
  - (lazy) heap deletions and insertions in O(log*n*).
- Compact: Yes, O(*n*), *n*-1 certificates at any time.
- Local: Yes, O(1), each point in at most 2 certificates.
- Efficient: No.

Maximum Maintenance Using Kinetic Swapping Heap



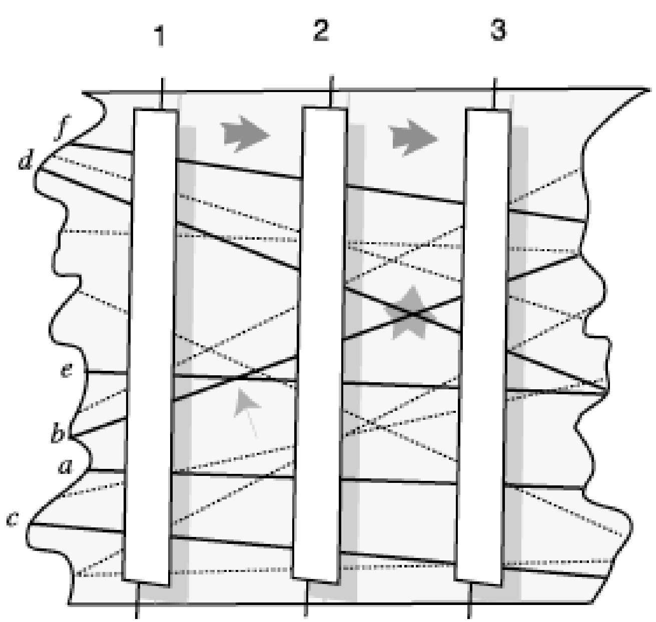


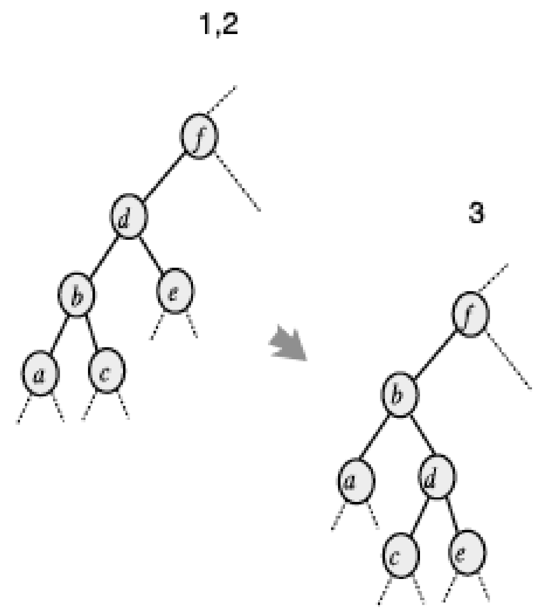
Maximum Maintenance Using Kinetic Swapping Heap

- Keep a certificate for each father-child pair. Events occur when swap is needed.
  - Responsive: Yes, swap in O(1) time and (lazy) deletions and insertions in the certificate heap take O(log*n*).
  - Compact: Yes, O(*n*) certificates
  - Local: Yes, O(1), each point is in at most 3 certificates.
  - Efficient: Difficult to control since the number of internal and external events can vary even if the start and end configurations match.

Maximum Maintenance Using Kinetic Heater

- Insert as in a BST using random key. Use rotations to obtain a heap w.r.t. priorities.

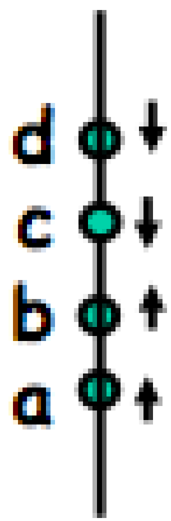


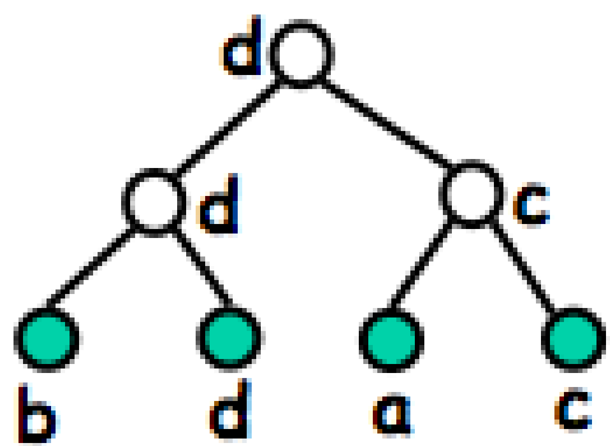


Maximum Maintenance Using Kinetic Heater

- Keep a certificate for each father-child pair. Events occur when rotation is needed.
  - Responsive: Yes, $O(\log n)$
  - Compact: Yes, $O(n)$
  - Local: Yes, $O(1)$
  - Efficient: somewhat complicated, see Basch

Tournament Tree





Tournament Tree

- Certificates: For each internal node, keep track of when its two children flip.
- Responsive: Replace the winner and update the events up the tournament tree. Each of these events needs to be (de)scheduled on the event heap. In total requires $O(\log^2 n)$ time.
- Local: Each point is in $O(\log n)$ certificates.
- Compact: Number of certificates is $O(n)$.

Tournament Tree

- Efficient?
  - External events: The configuration changes when the winner at the root changes. The root can change O(n) times.
  - Internal events: The children of the root can change O(n/2) times, the grandchildren of the root can change O(n/4) times, etc. The total number of internal events is therefore O($n$log$n$).

Kinetic Sorted List

- Maintain a list of the elements in sorted order, with the certificates enforcing the order between adjacent elements.
- When a certificate fails, two elements are swapped.
- At most three certificates must be updated, the certificate of the swapped pair, and the two certificates involving the swapped elements and the elements before and after the swapped pair.

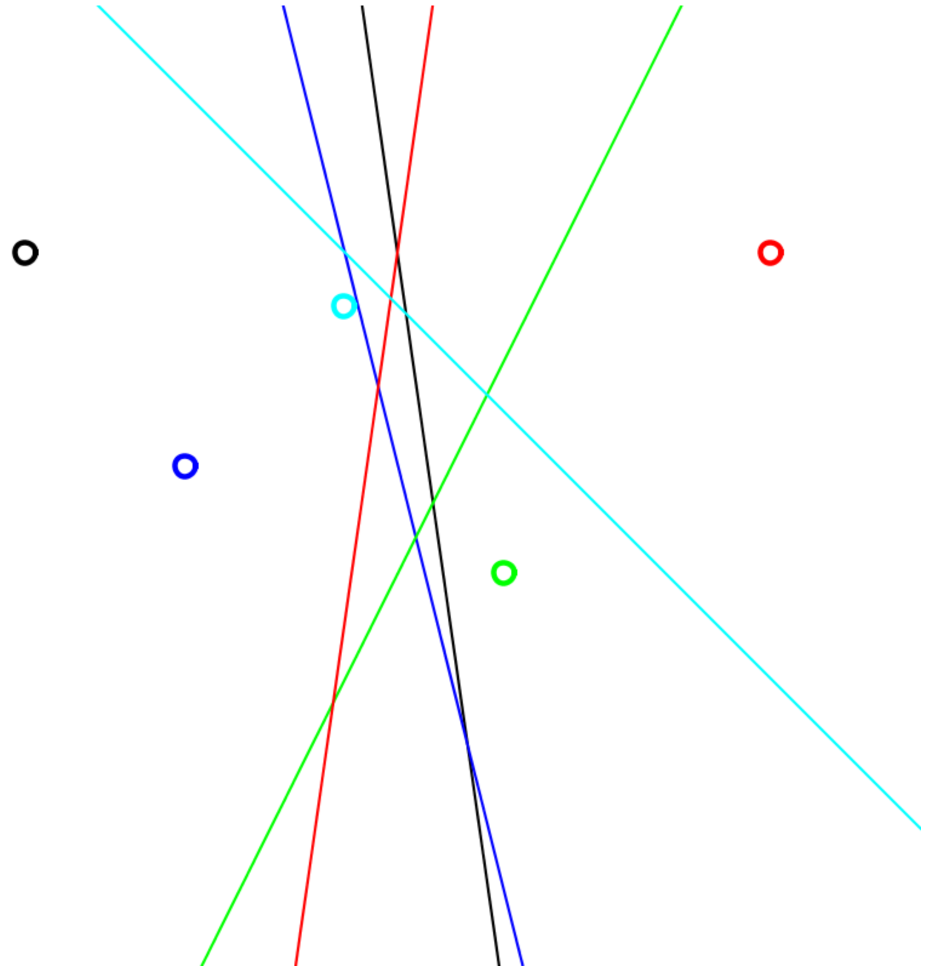
Kinetic Sorted List

- Responsive: a certificate failure causes one swap (which takes O(1) time) and O(1) certificate changes which take O(log *n*) time to reschedule (heap operations).
- Local: every element is involved in at most 2 certificates.
- Compact: there are exactly *n*-1 certificates for a list of *n* elements
- Efficient: no extraneous internal events, every change in the ordering of the elements causes exactly one certificate failure.

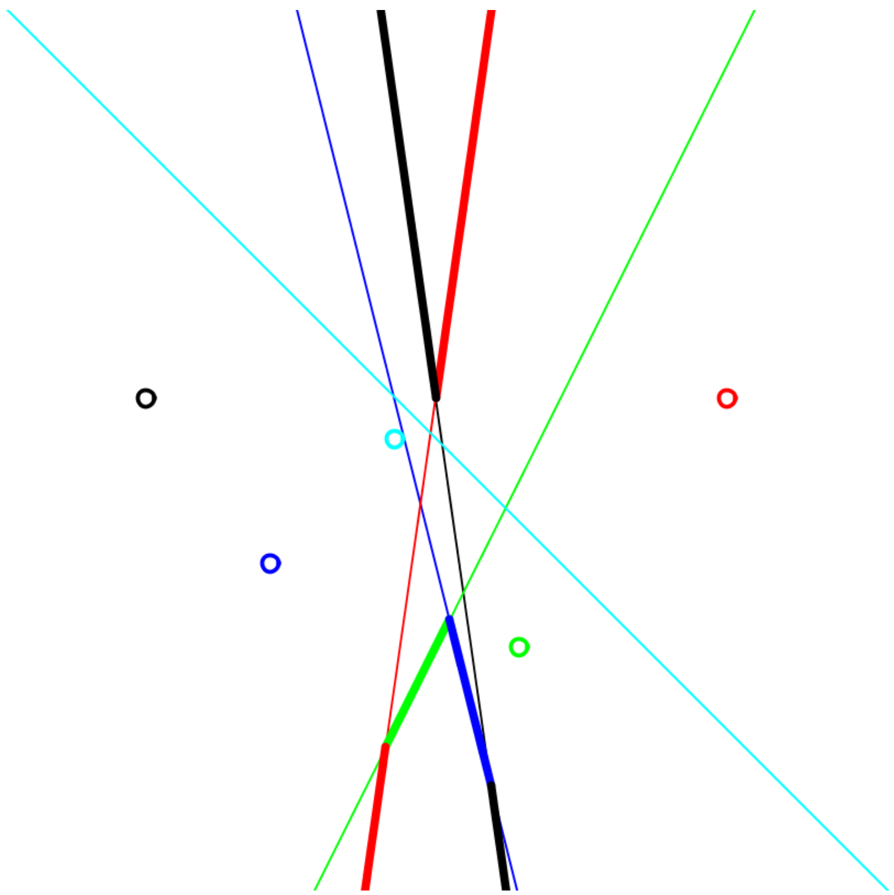
Convex Hull

- Maintain the convex hull of points moving in the plane.
- Compute upper and lower convex hull separately.

$$(a, b) \Leftrightarrow y = ax + b$$



Upper and Lower Envelopes



- Maintain the lower and upper envelopes of a set of continuously changing lines (translations and slope changes).

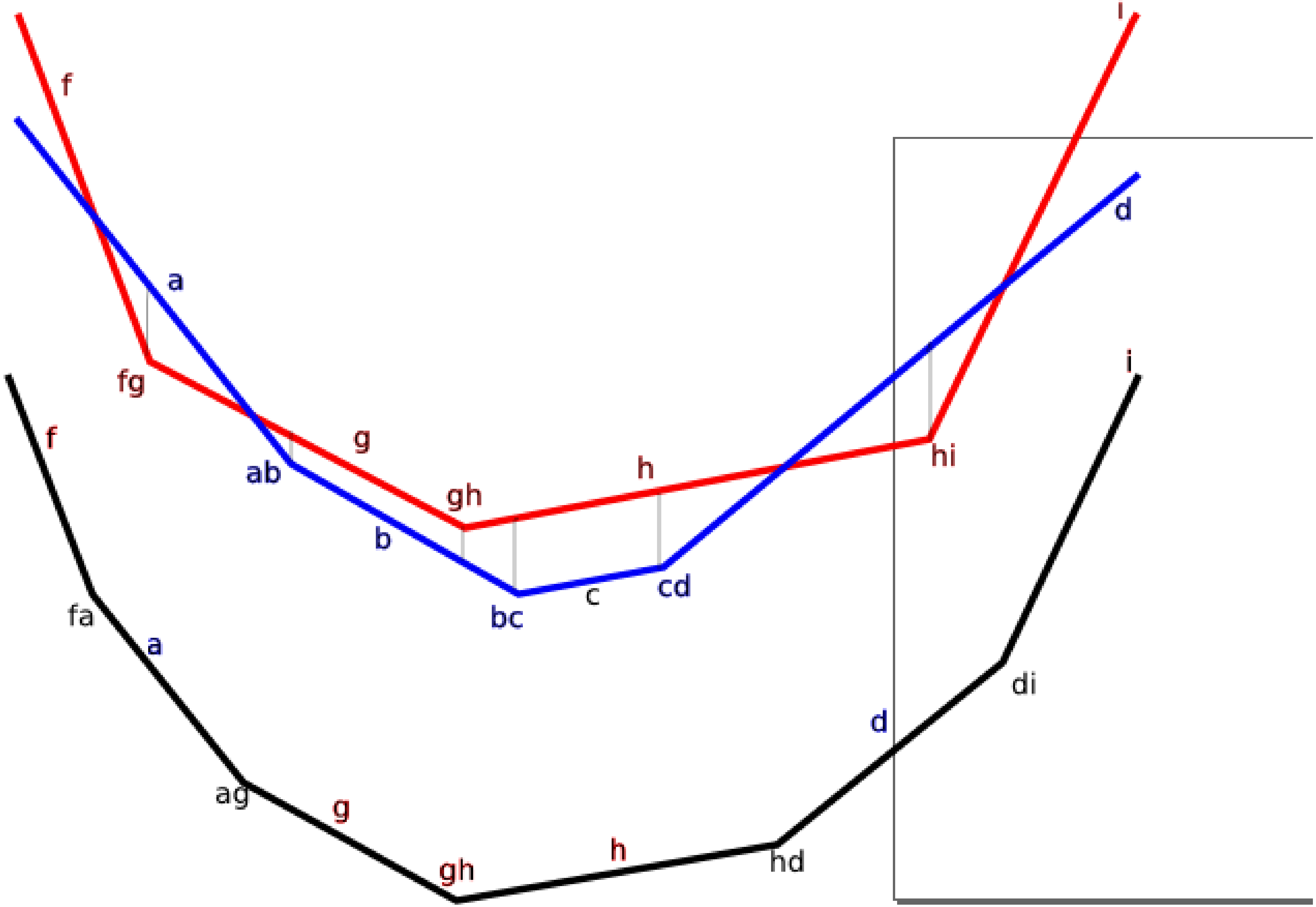
Static Divide-and-Conquer

- Suppose that we know a red upper envelope for $n/2$ lines and a blue upper envelope for the remaining $n/2$ envelopes.
- Upper envelopes are represented as chains of vertices and edges using doubly connected lists.
- Each red vertex knows its contender: the blue edge above or below it.
- Similarly for the blue vertices.
- We also need to know the slopes of half-lines on the left and half-lines on the right.

Merging

- Scan red and blue chains from left to write.
- Assume: Previous vertex added to the new chain was blue.
- Scanned vertex is blue:
  - Contender is below, add the blue vertex.
  - Contender is above, add the intersection as red.
- Scanned vertex is red:
  - Contender is below, add the intersection as red.
  - Contender is above, continue scanning.

Merging Blue and Red Chains

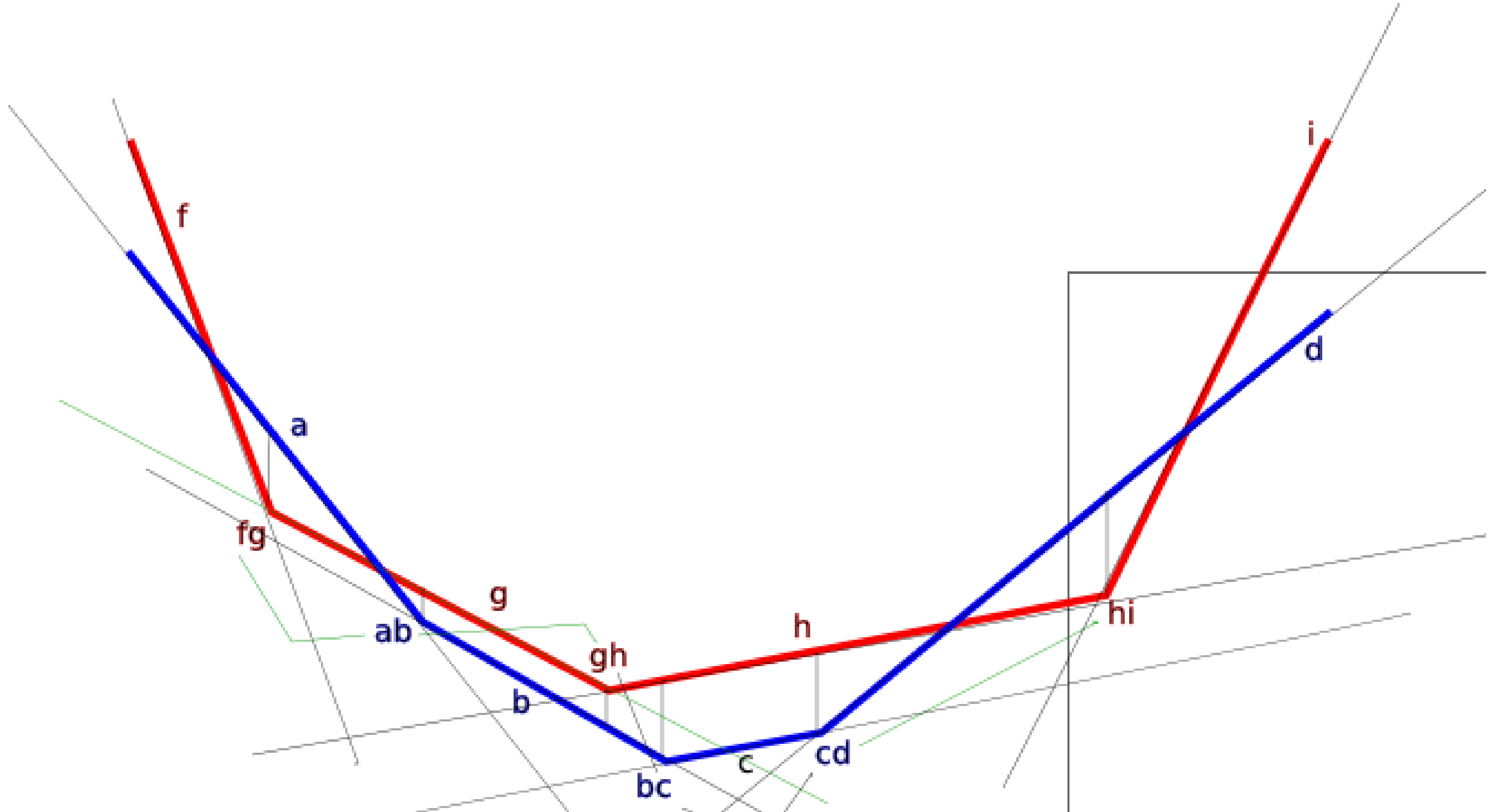


Kinetization

- Keep a record of entire computation in a balanced binary tree.
- Each node is in charge of maintaining the upper envelope of two upper envelopes computed by its children.
- If an event creates a change, the event is processed through the tree.
- Certificates?

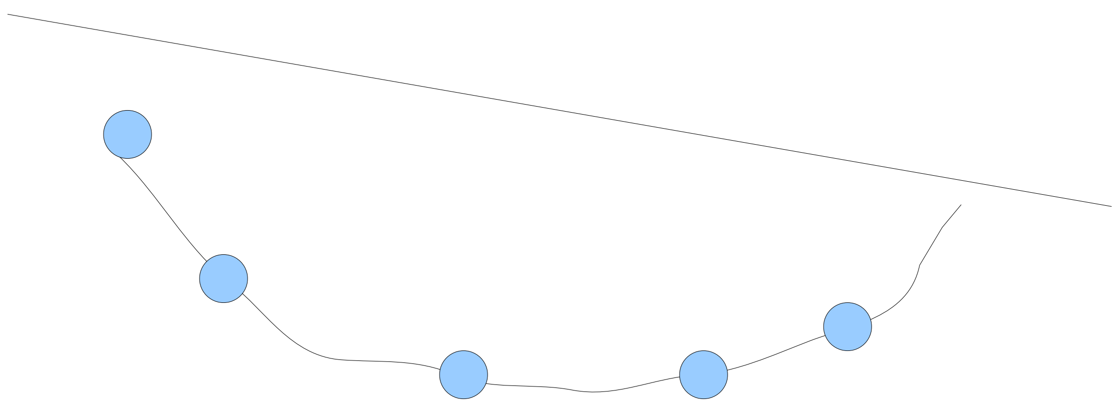
Certificates

- *x*-certificates: exist for pairs of *x*-consecutive vertices. Remain true until *x*-order changes or vertices seize to be *x*-consecutive.
- *y*-certificate: *y*-position (above, below) of contender edge w.r.t. its vertex. Remains true until *y*-position changes or the edge is no longer a contender of the vertex.



Problems?

- An edge could be a contender of O(n) vertices in the other chain.
- This KDS is not local.



Name	Comparison	Condition(s)
$\mathtt{x}[ab]$	$[ab <_x cd]$	$cd = ab.\mathtt{next}$ $\chi(ab) \neq \chi(cd)$
$\mathtt{yli}[ab]$	$[ab <_y \text{ or } >_y \mathtt{ce}(ab)]$	$b \cap \mathtt{ce}(ab) \neq \emptyset$
$\mathtt{yri}[ab]$	$[ab <_y \text{ or } >_y \mathtt{ce}(ab)]$	$a \cap \mathtt{ce}(ab) \neq \emptyset$
$\mathtt{yt}[ab]$	$[\mathtt{ce}(ab) <_y ab]$	$a <_s \mathtt{ce}(ab) <_s b$ $\mathtt{ce}(ab) <_y ab$
$\mathtt{slt}[ab]$	$[a <_s \mathtt{ce}(ab)]$	
$\mathtt{srt}[ab]$	$[\mathtt{ce}(ab) <_s b]$	
$\mathtt{sl}[ab]$	$[b <_s \mathtt{ce}(ab)]$	$b <_s \mathtt{ce}(ab)$ $ab <_y \mathtt{ce}(ab)$ $\chi(ab) \neq \chi(ab.\mathtt{next})$
$\mathtt{sr}[ab]$	$[\mathtt{ce}(ab) <_s a]$	$\mathtt{ce}(ab) <_s a$ $ab <_y \mathtt{ce}(ab)$ $\chi(ab) \neq \chi(ab.\mathtt{prev})$

x-successive vertices, each from different chain

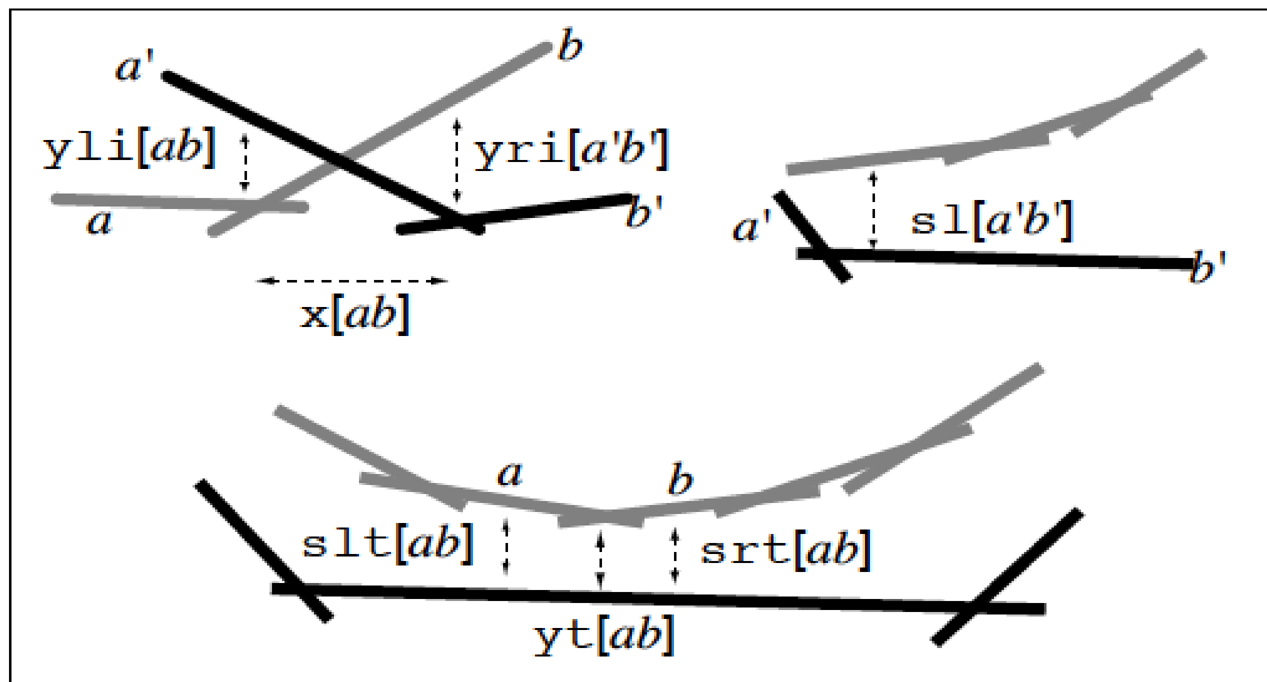
contender of a vertex preceeding intersection, contender defines this intersection

contender of a vertex succeeding intersection, contender defines this intersection

contender of vertex ab is below ab, slope of the contender is between slopes of a and b

contender of vertex ab is above ab, slope of b is less than the slope of the contender, successor of ab must be in the other chain

contender of vertex ab is above ab, slope of a is greater than the slope of the contender, predecessor of ab must be in the other chain.



Locality

- Claim: An edge *e* appears in O(1) certificates.
  - *e* appears in a certificate because one of its end-vertices appears in a certificate.
  - *e* appears in a certificate because it is a contender edge.

Locality - Vertices

- Vertex of *e* can appear in at most *2 x*-certificates.
- Vertex of *e* can appear in all other certificates at most once – these certificates involve uniquely defined contenders of such vertices.

Locality - Contenders

- Let *e* be a contender edge of some vertex.
  - Assume that *e* is intersected by the other chain.
    - Only yli[...] and yri[...] certificates can involve *e*.
    - *e* is intersected by the other chain at most twice. Hence *e* occurs in O(1) certificates.
  - Assume that *e* is not intersected by the other chain.
    - If *e* is below the other chain, it can be involved in only one triplet of certificates: yt[...], slt[...] and srt[...].
    - If *e* is above the other chain, it can be involved in at most one sl[...] certificate and at most one sr[...] certificate.

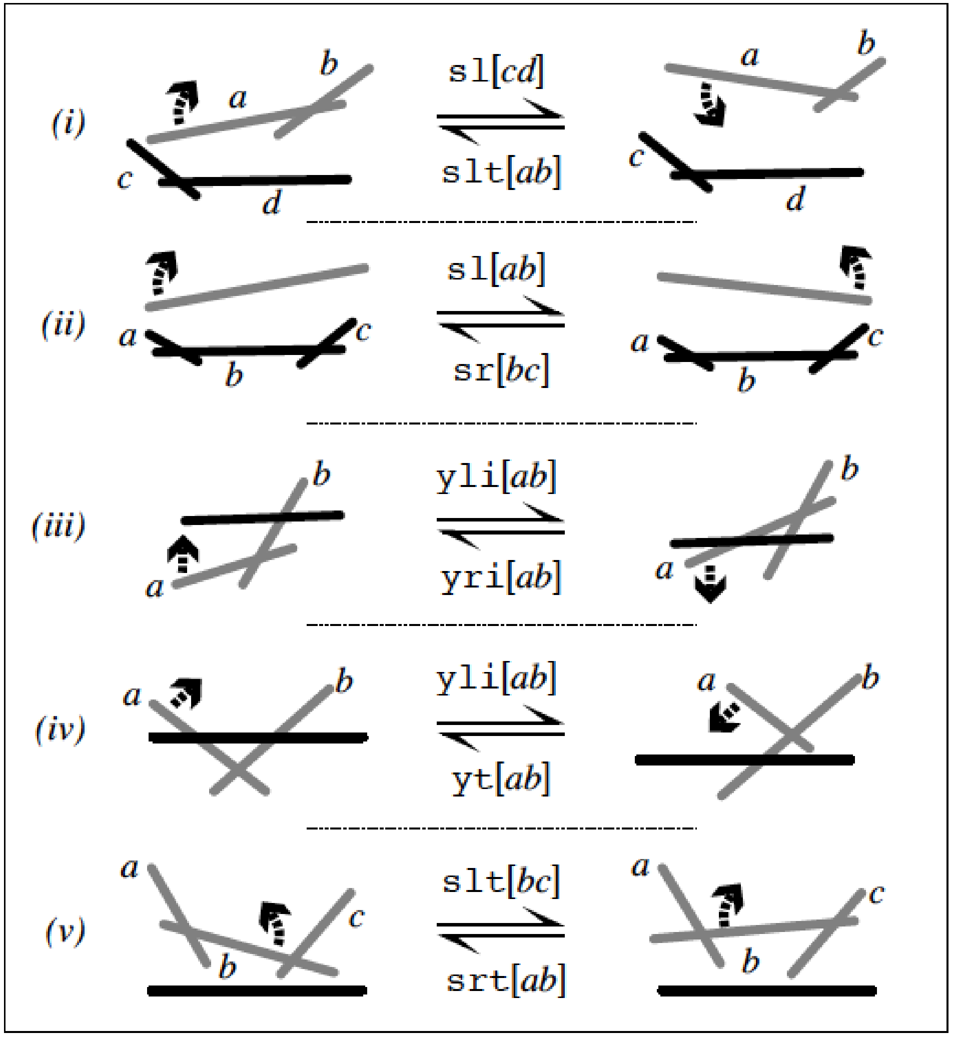
Locality - Contenders

- Suppose that *e* is not intersected but is a contender of many vertices.
  - This can only involve sl[...] and sr[...] certificates.
  - The vertices that have *e* as their contenders are of the same color.
  - Only the rightmost vertex with *e* as its contender can be involved in a sl[...] certificate.
  - Only the leftmost vertex with *e* as its contender can be involved in a sr[...] certificate.

Certificates - Correctness

- Let *L* denote the set of certificates and assume that they are valid for 2 different configurations.
- Have fun getting a contradiction :-)

Maintenance



Kinetization

- Keep a record of entire computation in a balanced binary tree.
- Each node is in charge of maintaining the upper envelope of two upper envelopes computed by its children.
- If an event creates a change, the event is processed through the tree.

Kinetic Convex Hulls

- Responsive (processing of failing certificates)? Yes, in $O(\log^2 n)$
- Compact (number of certificates)? Yes, $O(n \log n)$
- Local (number of certificates involving any object)? Yes, $O(\log n)$
- Efficient: Yes, see Bash for details.