Computational Geometry II

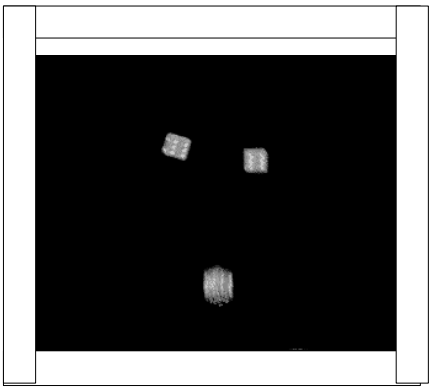
Randomized Algorithms

Computational Geometry II
Lecturer: Gill Barequet

Michal Aharon

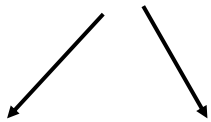
Randomized Algorithms

- A randomized algorithm is an algorithm that makes random choices during its execution.
- Whatever the random choices are, a randomized algorithm always runs in a finite time and outputs the correct solution.



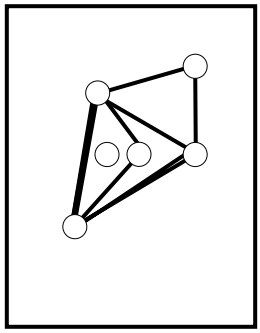
Randomized Incremental method

- Randomized Incremental method:
  - Solve for a small subset of the data.
  - A valid solution is maintained while the rest of the data is processed.



Off-line algorithm: requires a prior knowledge of the whole data.

on-line (semi-dynamic) algorithms: do not look ahead at the objects that remain to be inserted.



Randomized Incremental method

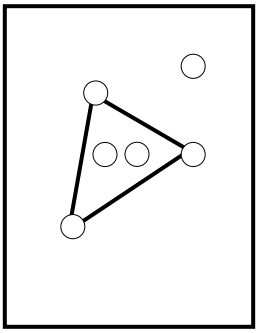
- Sometimes, these algorithms are not random at all, as the order of the inserted data is imposed on the algorithm.
- However, we refer to this order as random, and then perform a **randomized analysis** of the algorithm.

⇩

Assumption: All permutations are expected with equal probability.

Definition - Reminder

- *n* Objects, regions defined by objects, regions conflict with objects
- $F_j(S)$ – all regions defined by *S* which conflict with j objects in *S*.
- $F^i_j(S)$ – all regions defined by *i* objects from S and conflict with *j* objects.
- $F_0(S)$ – all regions defined by *S* which do not conflict with any object in *S*.

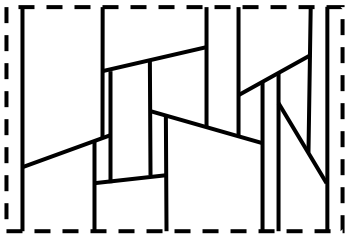


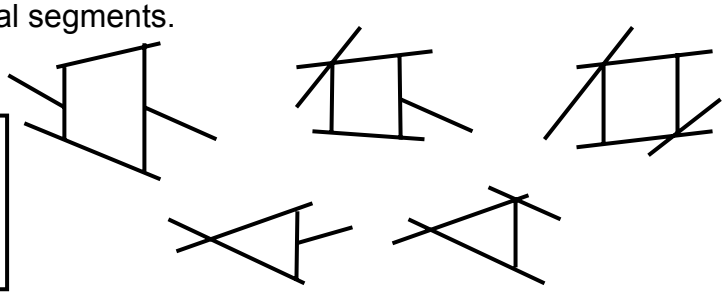
On Our Agenda

- Sample Problem Description –
  - Vertical decomposition of a set of segments
- Off-line solution to the problem
  - Randomized analysis
- General off-line solutions of incremental problems.
- on-line solution to the problem
  - Randomized analysis
- General on-line solutions of incremental problems.
- Extentions…

Vertical decomposition of a set of segments

- Let *S* be a set of n segments in the plane. We are interested in building a planar map $F_0(S)$ that includes a minimal number of trapezoids defined and without conflict over the set *S*, using only vertical segments.

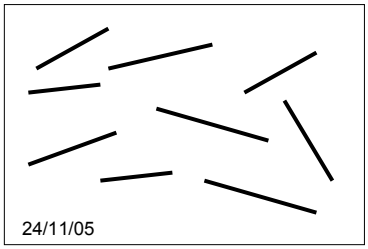


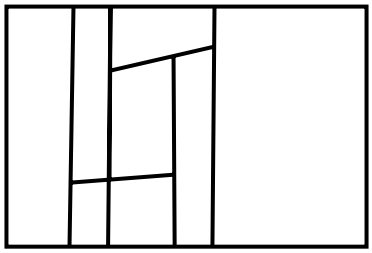


The maximal number of segments that define a region is 4.

Vertical Decomposition – General Approach

- In each step, a new segment is processed.
- The regions that conflict with the new segment are deleted.
- Regions defined by the new segment are created.
- At each step, the set of regions *F* is defined and without conflict with the already processed segments.

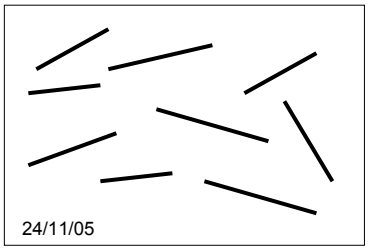


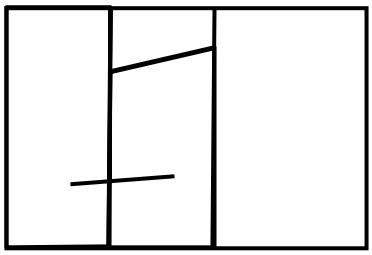


Vertical Decomposition – General Approach

- The main problem is:

How do we find the regions that conflict with the current segment ?



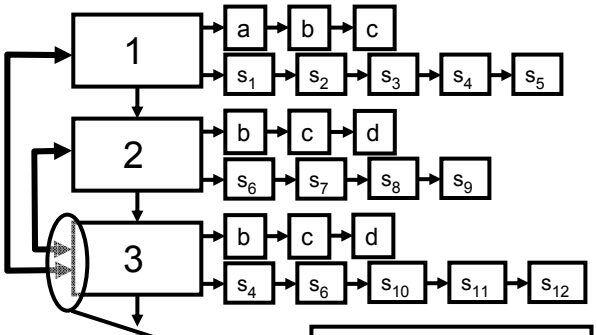


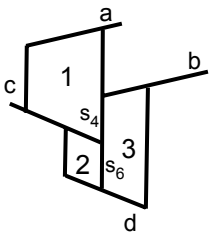
Vertical Decomposition – Off-line Approach

- Two data structures are stored:

$Dec_s(R)$

Conflict Graph

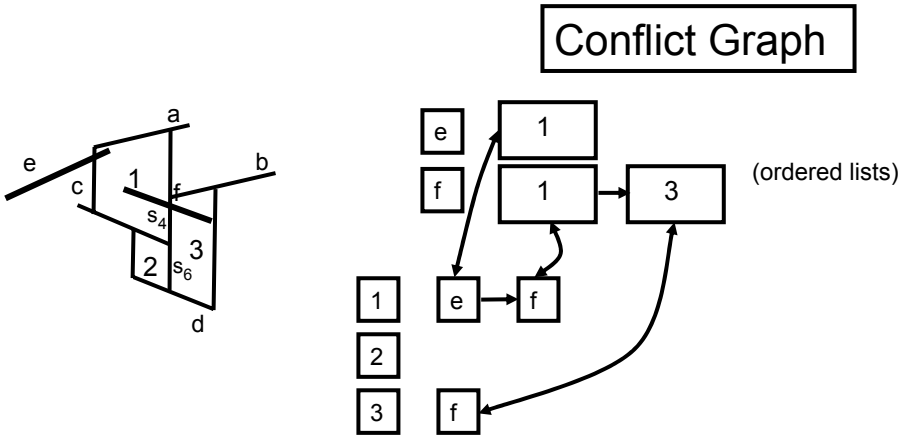




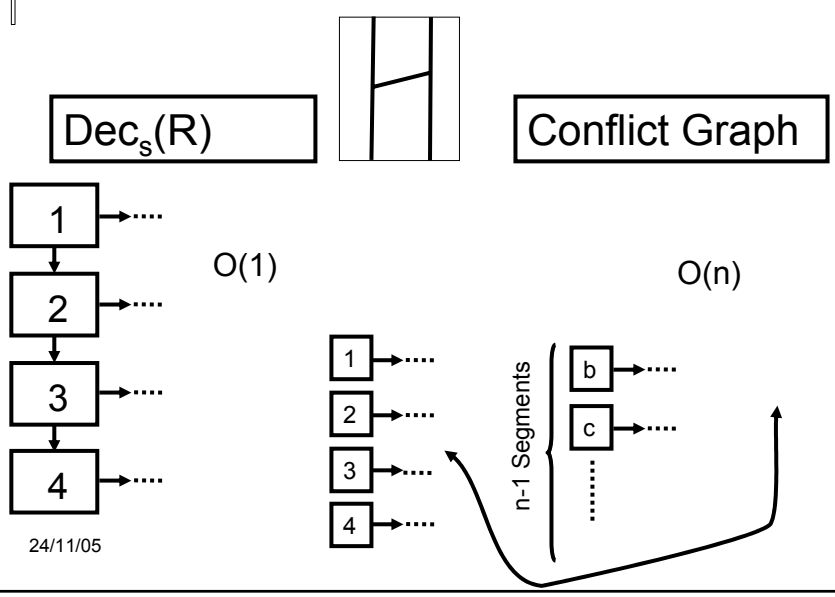
At most 4 regions

Vertical Decomposition – Off-line Approach

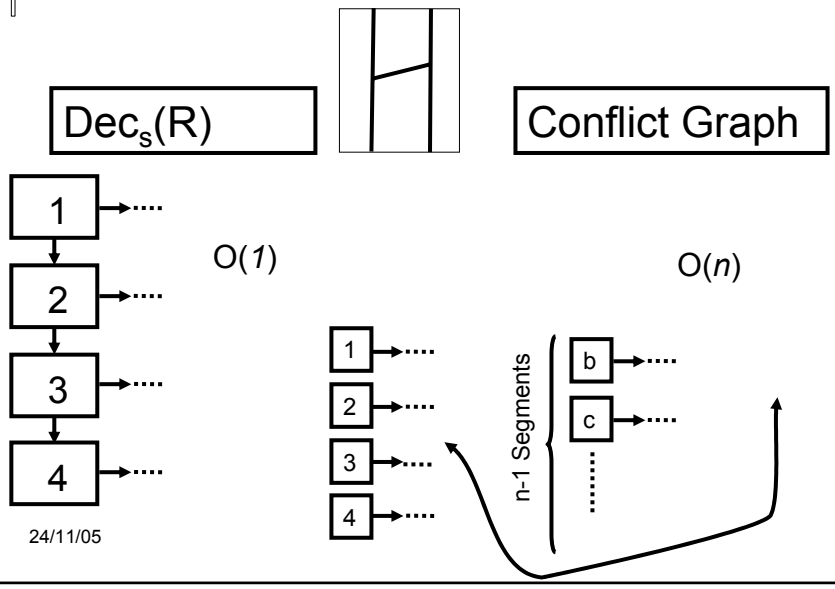
- Two data structures are stored:



Vertical Decomposition – Initialization

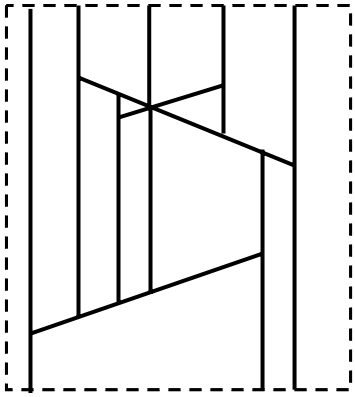


Vertical Decomposition – Initialization



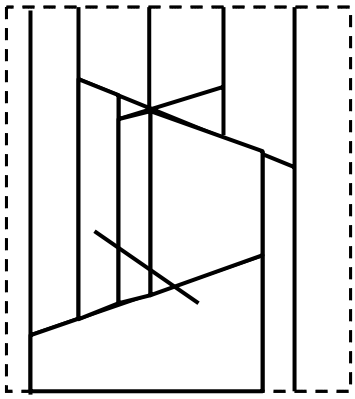
Vertical Decomposition

- Updating the decomposition:
  - The conflict graph reports all the trapezoids that conflict with the current processed segment.
  - Each such trapezoid is split into at most 4 subregions.



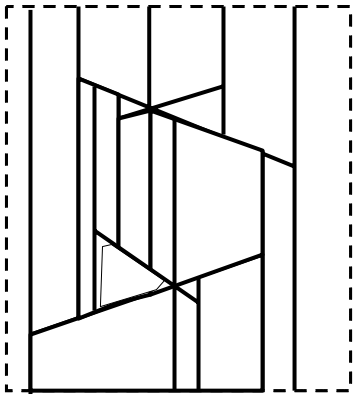
Vertical Decomposition

- Updating the decomposition:
  - The conflict graph reports all the trapezoids that conflict with the current processed segment.
  - Each such trapezoid is split into at most 4 subregions.
  - Not all of these subregions will be included in *F(R∪{S}).*



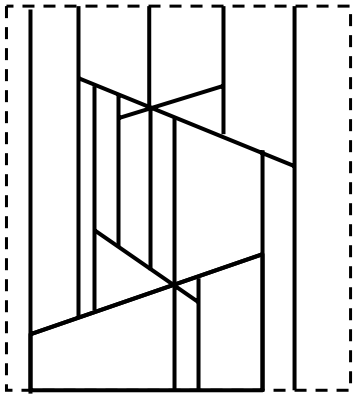
Vertical Decomposition

- Updating the decomposition:
  - The conflict graph reports all the trapezoids that conflict with the current processed segment.
  - Each such trapezoid is split into at most 4 subregions.
  - Not all of these subregions will be included in F(R∪{S}).
  - *S* may intersect a vertical wall. A part of the wall might be deleted and the two subregions that join this part should be joined.
  - The update of the decomposition can be done in time linear in the number of conflicting regions.



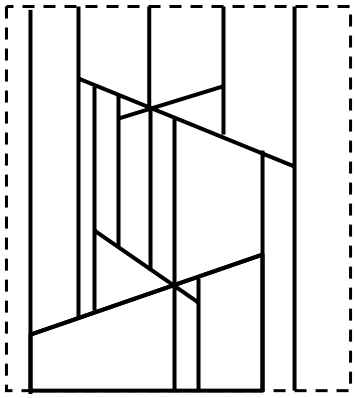
Vertical Decomposition

- Updating the conflict graph:
  - When a trapezoid *F* is split into sub-regions, the list of segments that conflict with F is traversed linearly, and up to 4 new lists are created.
  - When joining regions to create a new one, their lists of conflict segments should be merged.



Vertical Decomposition – Algorithm Analysis

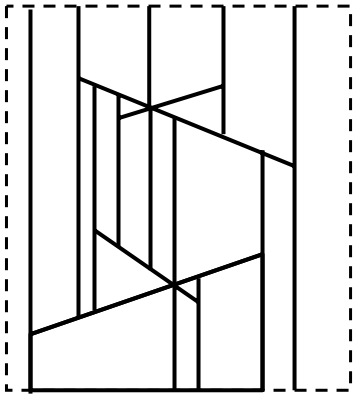
- Each update stage obeys:
  - Each update stage requires time proportional to the number of killed or created regions in this step.
  - The update of the conflict graph can be carried out in time proportional to the number of arcs added or removed during this step.



Vertical Decomposition – Algorithm Analysis

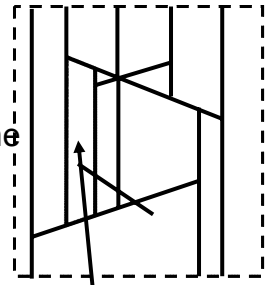
- Each update stage obeys:
  - Each update stage requires time [illegible] the number of [illegible]ions in this s[illegible]
  - The [illegible]ph can be c[illegible] proportional t[illegible] arcs added or remo[illegible] this step.

The Update Condition



Vertical Decomposition – Algorithm Analysis

- Let be a region determined by *i* segments that has *j* conflicts with all segments.
- The probability that *F* be one of the regions created by the algorithm is:

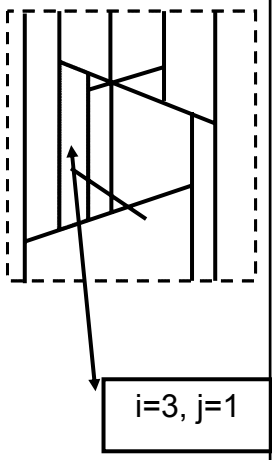


i=3, *j*=1

Proof: this equals the probability that the i creating segments will be processed before the j conflicting segments.

Vertical Decomposition – Algorithm Analysis

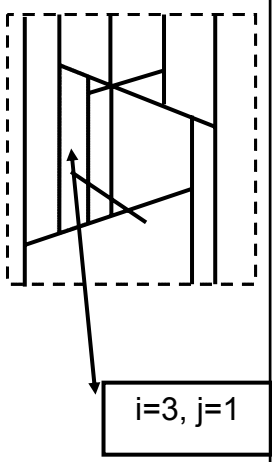
- Let *F* be a region determined by *i* segments that has j conflicts with all segments.
- The probability that *F* be one of the regions created by the algorithm during step r,
- Let be the probability that a region F of be defined and without conflict over a random r-sample of S (4.2.1).



Vertical Decomposition – Algorithm Analysis

- Let *F* be a region determined by *i* segments that has *j* conflicts with all se
- Th the du
- Let F c con (4.

Proof: For a region *F* to exists after step r, we require that the R already processed segments do not conflict with F (P=). For *F* to be created in step r, we require that one of the segments creating *F* is processed, and this happens in P = *i/r*.

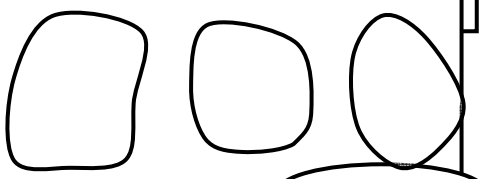


Vertical Decomposition – Algorithm Analysis

- Let S be a set of n objects.
- The expected total number of regions created by the algorithm is:

denotes the expected number of regions defined and without conflict over a random r-

Proof:



4.2.2

Algorithm Analysis – Reminder

- Lemma 4.2.2:

Vertical Decomposition – Algorithm Analysis

- Let *S* be a set of n objects.
- The expected total number of regions created by the algorithm is:
- The expected total number of conflict arcs added to the conflict graph by the algorithm is:

denotes the expected number of regions defined and without conflict over a random r-sample of *S*

Vertical Decomposition – Algorithm Analysis

- Let *S* be a set of n objects.

denotes the

Proof:

4.2.7

4.2.5

Algorithm Analysis – Reminder

- Lemma 4.2.5:

Algorithm Analysis – Reminder

- Lemma 4.2.7:

(for a constant)

Vertical Decomposition – Algorithm Analysis

- Let *S* be a set of *n* objects.
- The expected total number of conflict arcs added to the conflict graph by the algorithm is:
- If the algorithm satisfies the update condition, then its complexity is, on the average:

denotes the expected number of regions defined and without conflict over a random r-sample of *S*

Vertical Decomposition – Algorithm Analysis

- Let *S* be a set of n objects.

Proof:

Update condition

ed

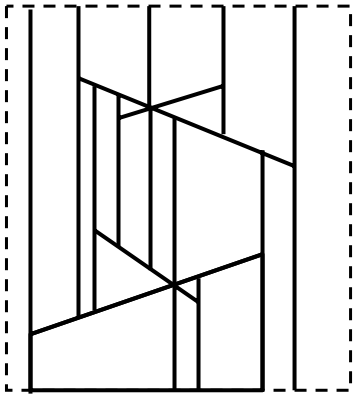
conflict over a

random r-sample of *S*

- If the algorithm satisfies the update condition, then its complexity is, on the average:

Vertical Decomposition – **The Update Condition**

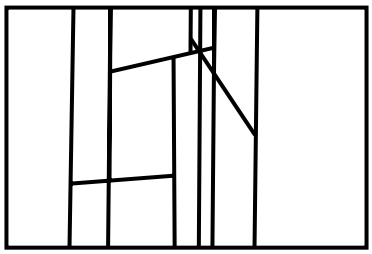
- Each update stage obeys:
  - Each update stage requires time proportional to the number of killed or created regions in this step.
  - the update of the conflict graph can be carried out in time proportional to the number of arcs added or removed during this step.



Vertical Decomposition – Algorithm Analysis

- Let S be a set of n segments, *a* pairs of which intersect. Then,

Let a(R) be the number of intersecting pairs in R. The number of regions in the vertical decomposition is: O(r+a(R)).



Vertical Decomposition – Algorithm Analysis

- Let S be a set of n segments, *a* pairs of which intersect. Then,

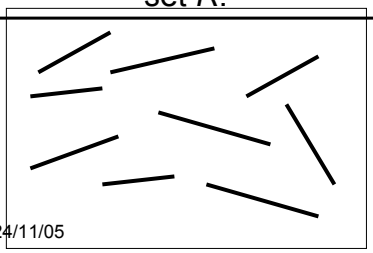
Proof: For a subset *R* of r segments, denote *a(R)* the number of intersecting pairs. The number of regions is *O(r+a(R)).* An intersection point P in S is also an intersection point in *R* with probability

Total complexity of the vertical decomposition Algorithm

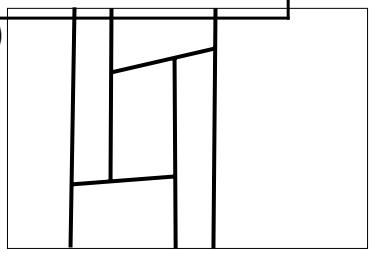
General off-line solutions - Randomized Incremental Method

Trapezoids	⟷	Regions
Segments	⟷	Objects
Planar map of trapezoids that are defined and without conflict with the set *R*.	⟷	$F_0(R)$

S



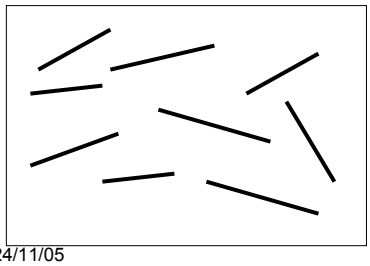
F(R)



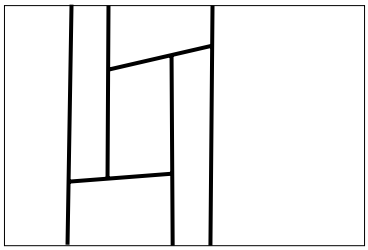
General off-line solutions - Randomized Incremental Method

- **<u>Regions Killed By O:</u>** The regions of $F_0(R)$ that do not belong to $F_0(R \cup \{O\})$ are the ones the conflict with O
- **<u>Regions Created By O:</u>** The regions of $F_0(R \cup \{O\})$ that do not belong to $F_0(R)$ are the ones determined by a subset of $R \cup \{O\}$ that contains O.

S



F(R)



On Our Agenda

- Sample Problem Description –
  - Vertical decomposition of a set of segments
- Off-line solution to the problem
  - Randomized analysis
- General off-line solutions of incremental problems.
- on-line solution to the problem
  - Randomized analysis
- General on-line solutions of incremental problems.
- Extentions…

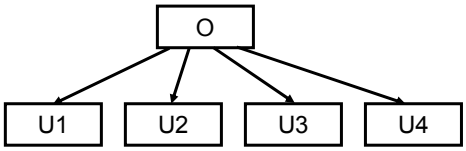
Vertical Decomposition – On-line Approach

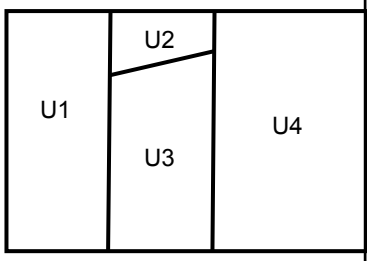
- Two data structures are stored:

$Dec_s(R)$	Influence Graph	flict Graph

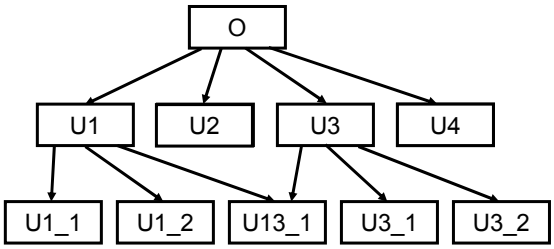
- The Influence Graph is directed and acyclic.
- The nodes correspond to the regions created by the algorithm.
- A node in the graph might have several parents.

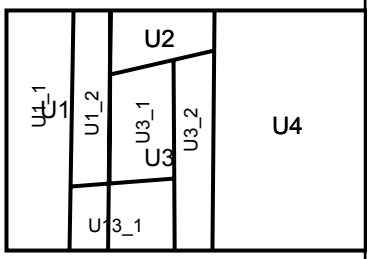
The Influence Graph





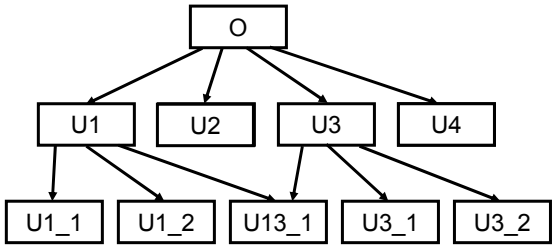
The Influence Graph

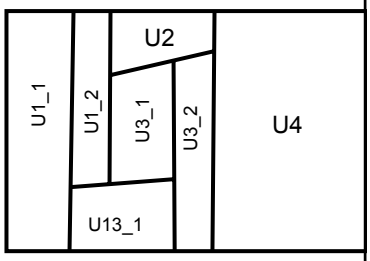




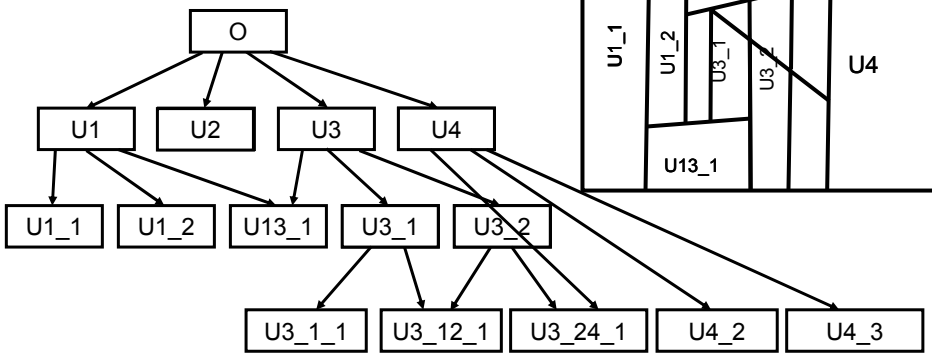
- When a new object is inserted, it should first be located, searching from the root *O*.

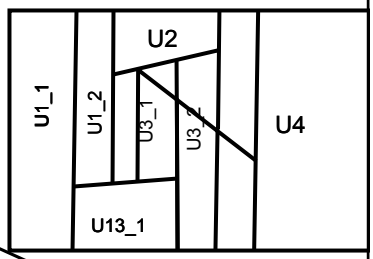
The Influence Graph



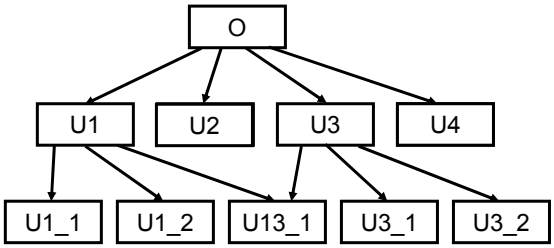


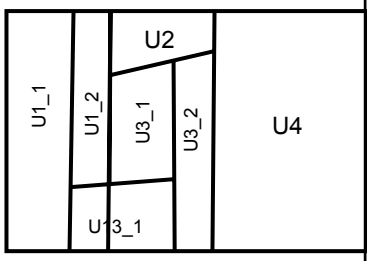
The Influence Graph





The Influence Graph





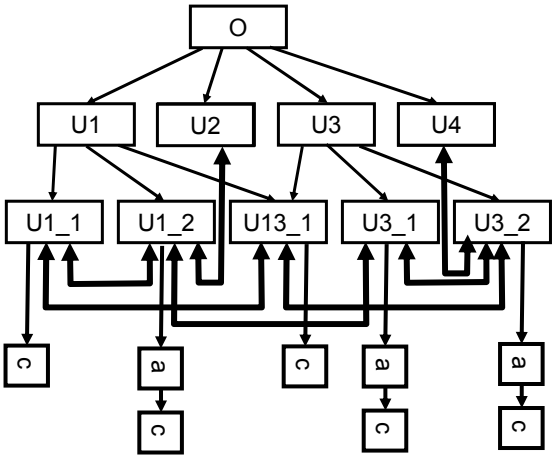
Vertical Decomposition – On-line Approach

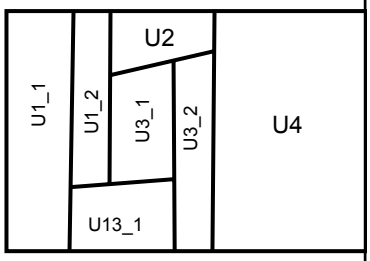
- Two data structures are stored:

Influence Graph

- The Influence Graph is directed and acyclic.
- The nodes correspond to the regions created by the algorithm.
- A node in the graph might have several parents.
- It holds that:
  - Each leaf in the influence graph represents a region which is defined and without conflict over the current subset of objects.
  - The domain of influence of a region associated with a node is contained in the union of the domains of influence of the regions associated with the parents of that node.

The Influence Graph

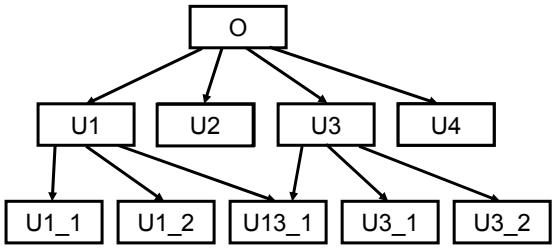




The decomposition can be omitted here, as all information about the planar map can be retrieved from the influence graph.

Analysis of the On-line Algorithm

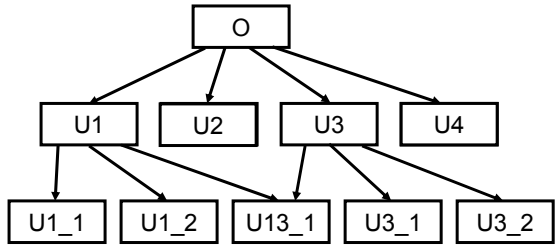
- The algorithm must satisfy:
  - A conflict between a given region and object can be detected in constant time.
  - The number of children of each node in the graph is bounded by a constant (4).
  - The parents of a node created by O are nodes that are killed by O, and the update takes time linear in the number of nodes killed or created at each step.



Analysis of the On-line Algorithm

- The algorithm must satisfy:
  - A confli[illegible]n region and object can be detected in consta[illegible]
  - The numbe[illegible]e in the graph is bounded by a [illegible]
  - The parents of a no[illegible]des that are killed by O, and the update tak[illegible] number of nodes killed or created at each [illegible]

The Update Condition



Analysis of the On-line Algorithm

- If an algorithm satisfies the update condition, then:
- The expected storage used by the algorithm to process the *n* objects is

denotes the expected number of regions defined and without conflict over a random r-

Proof:This is exactly the bound proved for the expected number of created regions.

Analysis of the On-line Algorithm

- If an algorithm satisfies the update condition, then:
- The expected complexity of the algorithm is

denotes the expected number of regions defined and without conflict over a random r-sample of *S*

Analysis of the On-line Algorithm

Proof:

1. Complexity for locating the object: if a region of is created at some step, this node will be visited *j* times. That is equal to the number of conflict arcs created in the off-line version (with the same permutation). This was proved to be as above.
2. Complexity of the all update phases – proportional to the number of regions created (previous slide).
3. Total complexity is therefore the sum of the two terms, which equals the above.

Notice that the complexity of the algorithm is dominated by the cost of the locating phases.
This will later be changed in an off-line accelerated version.

Algorithm

Proo

1. Co if a region of is vill be visited *j* tim of conflict arcs cre the same permutation). This was proved to be as above.
2. Complexity of the all update phases – proportional to the number of regions created (previous slide).
3. Total complexity is therefore the sum of the two terms, which equals the above.

Analysis of the On-line Algorithm

- If an algorithm satisfies the update condition, then:
- The expected time complexity of the locating phase at step k is

denotes the expected number of regions defined and without conflict over a random r-sample of S

Analysis of the On-line Algorithm

Proof: a region F in is created at step r with probability . The probability that this region conflicts with the object in step k, knowing *F* is created prior to step *k*, is . Therefore, the expected number of nodes that conflicts with *O* is:

4.2.7 4.2.5

Algorithm Analysis – Reminder

- Lemma 4.2.5:

Algorithm Analysis – Reminder

- Lemma 4.2.7:

Analysis of the On-line Algorithm

- If an algorithm satisfies the update condition, then:
- The expected time complexity of the update phase at step *k* is:

denotes the expected number of regions defined and without conflict over a random r-sample of *S*

Expected number of regions created at step k

Expected number of regions killed at step k

Proof: the expected number of regions created at step k is

A region is killed if it is a region before step k, and if it conflicts with O. This happens with probability

and the expected number of regions killed in step k is

4.2.7

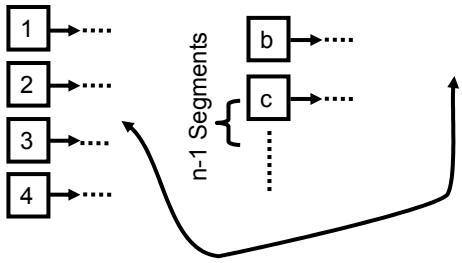
4.2.5

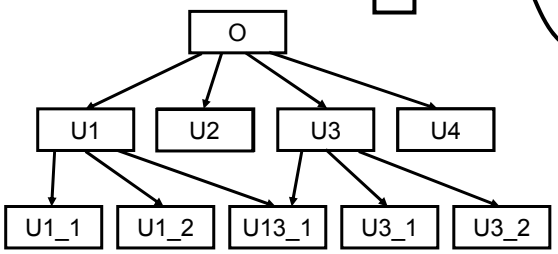
Vertical Decomposition - Analysis

- The expected number of trapezoids is:
- The expected time complexity of an on-line algorithm is
- and its expected storage is
- The average time complexity of the *n*th insertion is

Summary of Previous Lecture

- We defined random incremental algorithms, and talked about off-line and on-line versions.
- Conflict graph.
- Influence graph.



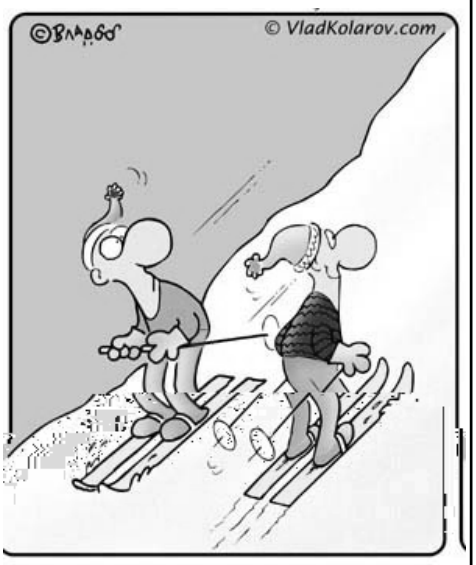


Accelerated incremental algorithms

Locating phase

Updating phase

Total

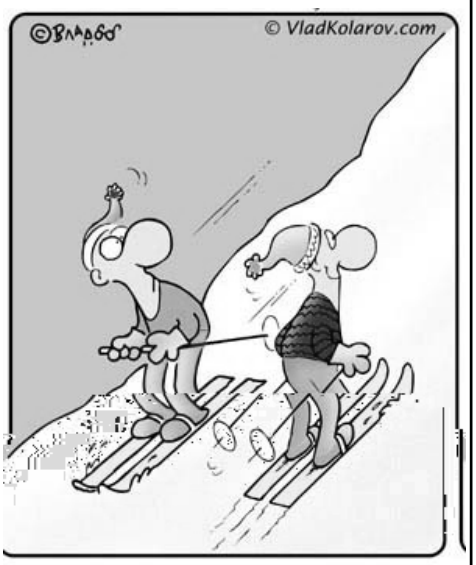


Accelerated incremental algorithms

Locating phase

Updating phase

Total



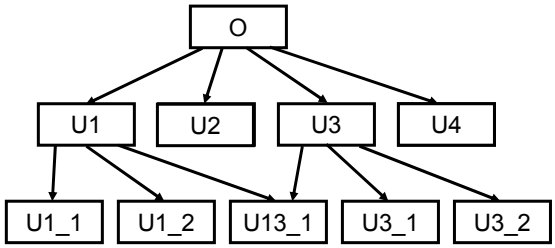
Accelerated incremental algorithms

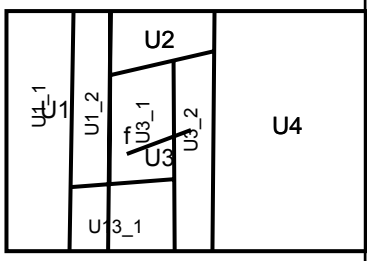
Process in the On-line approach

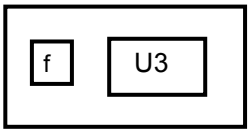
Compute the conflict graph

Process in the On-line approach, while start each search from the lowest node in the conflict graph

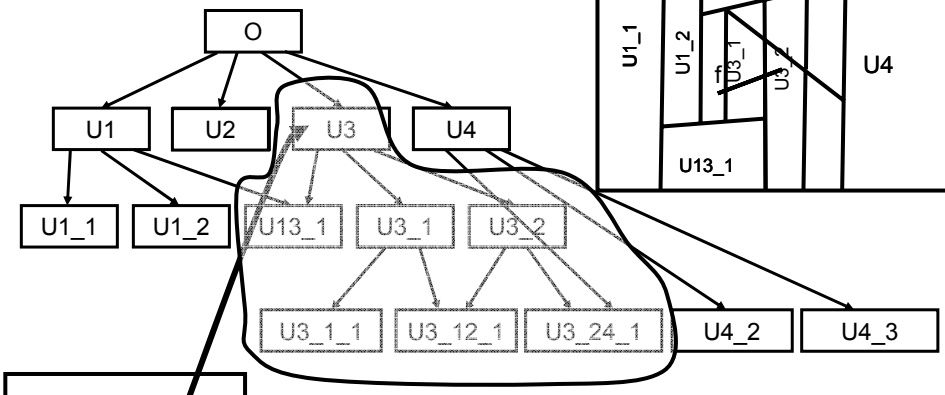
The Influence Graph

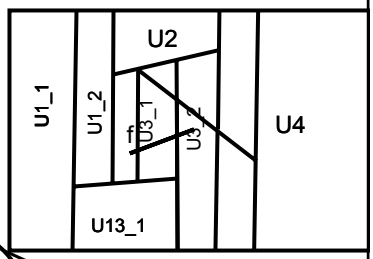


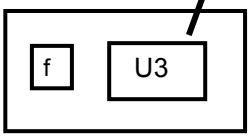




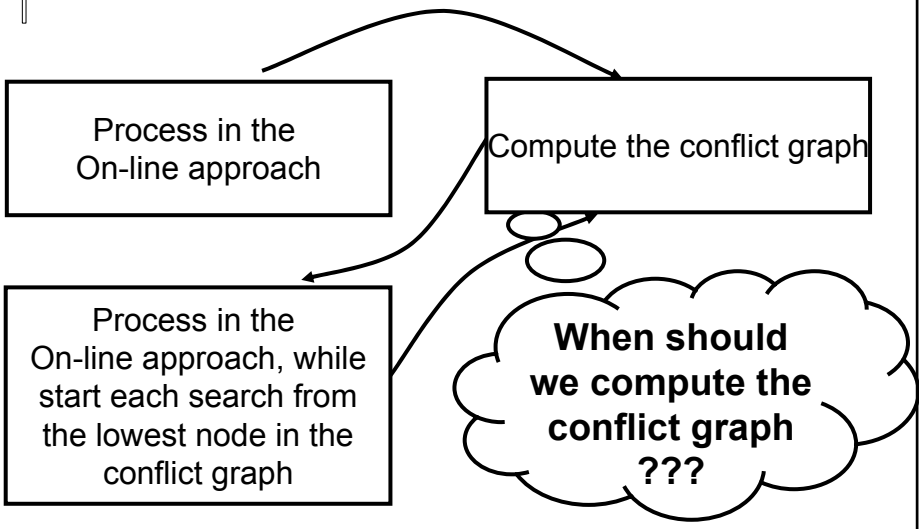
The Influence Graph







Accelerated incremental algorithms



Accelerated incremental algorithms

- We try to combine the conflict graph and the influence graph so to achieve a static algorithm (off-line) that has a lower average complexity.
- We don't maintain the conflict graph at every step, but update it only at certain steps.
- If an on-line algorithm satisfies the update condition, knowledge of the conflict graph at step *k* can be used to perform the locating phase in step *m* with an average complexity of

Accelerated incremental algorithms

Proof: The conflict graph at step k can be augmented, for each object O in $S \backslash S_k$, by a list of pointers to the nodes of the influence graph which correspond to a region of $F_0(S_k)$ that conflict with O. In order to locate O_m at step m, the algorithm may start to traverse the influence graph not from the root, but from the nodes of the influence graph which correspond to a region of $f_0(S_k)$ that conflicts with O_m. The expected number of regions that conflicts with the object O_m is,

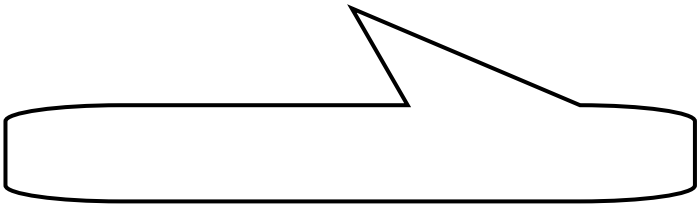
Accelerated incremental algorithms

- If the conflict graph is computed at steps

log*n remains smaller than 5 for all numbers n from 1 up to $2^{65,535}$

Accelerated incremental algorithms

- If the conflict graph is computed at steps



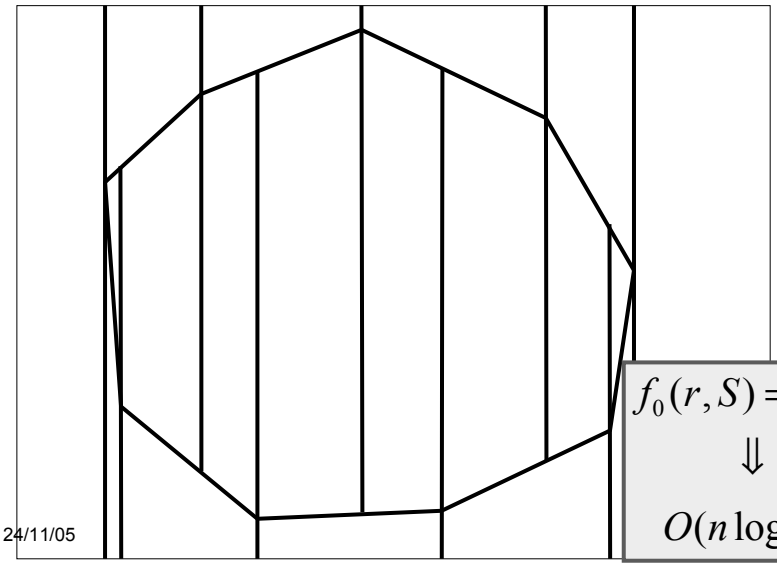
Accelerated incremental algorithms

Given that: $f_0(r,S) = O(r)$, and that the conflict graph can be built in expected time $O(n)$

- Then the conflict graph is computed log^*n times, with complexity $O(n \log^*n)$. The locating phases, between step n_k and step n_{k+1}, have a total average complexity of

Total complexity:

Vertical Decomposition of a Polygon



$$f_0(r,S) = O(r)$$
$$\Downarrow$$
$$O(n \log^* n)$$

To summarize

Randomized Incremental Algorithms

Off-line approach | On-line approach

Complexity Bounds

Conflict graph | Influence graph

Update Condition

Accelerated Algorithm

Thank you!