Distributed Disaster Disclosure

Algorithms for Event Detection

Stefan Schmid

Collaborators:
Bernard Mans
Roger Wattenhofer

TECHNISCHE UNIVERSITÄT MÜNCHEN

Wroclaw Information
Technology Initiative (2008)

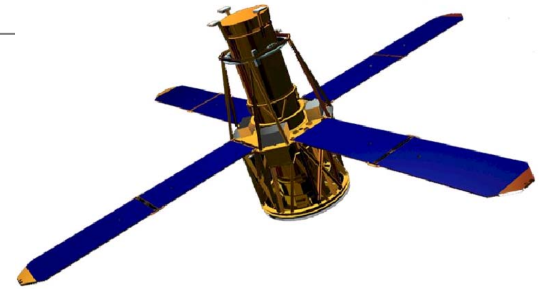
Motivation

- Talk deals with natural disasters
  - Flooding, earthquakes, fires, etc.

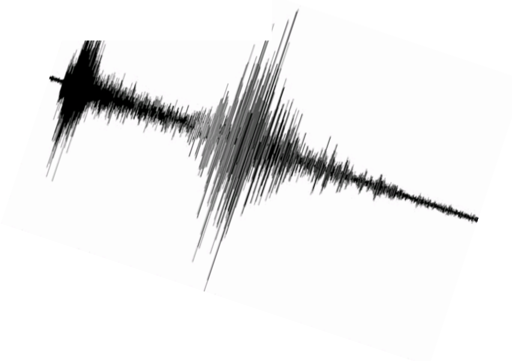
- Need for fast disclosure
  - to warn endangered towns (shelter)
  - to inform helpers (e.g., firemen)

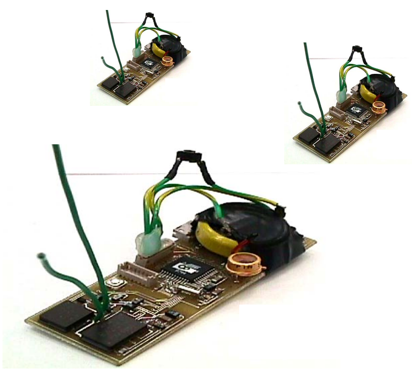
Our focus: environmental monitoring and early warning systems

Today’s Warning System



- Different kinds of warning systems
  - Satellites
  - Seismic sensors
  - Smoke detectors
  - etc.

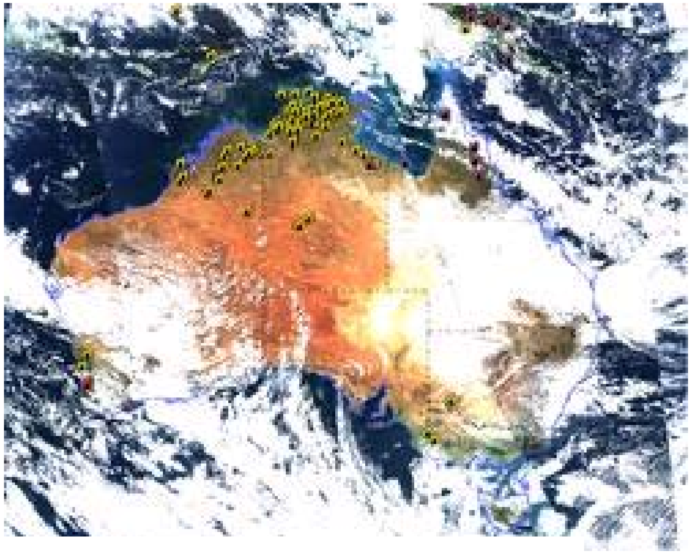




- Focus of this talk: Sensor nodes
  - Simple „computers“ with sensors
  - Sensors measure physical properties (e.g., heat)
  - Basic wireless communication
  - Cheap, can be distributed over a certain area
  - Limited energy supply

Why Sensor Nodes?

- Example: SENTINEL
  - Australian bushfire monitoring system
  - Based on satellites
  - Provides timely information about hotspots
  - Satellites may miss certain heat sources, e.g., if there is smoke!
  - Sensor nodes can be a good alternative



SENTINEL Disclaimer Agreement

Please note the limitations of the Sentinel Hotspots mapping system:

1. Under ideal conditions, the hotspots shown will have been detected 1-24 hours ago, depending on regional information received from the last satellite overpass.
2. The hotspot location on any map (no matter how detailed) is only accurate to at best 1.5 km.
3. The symbol used for the hotspot on the maps does not indicate the size of the fire.
4. Not all hotspots are detected by the satellites. Some heat sources may be too small, not hot enough, or obscured by thick smoke or cloud.
5. The satellites detect any heat source that is hotter than normal. As well as fires these may include industrial operations such as furnaces.

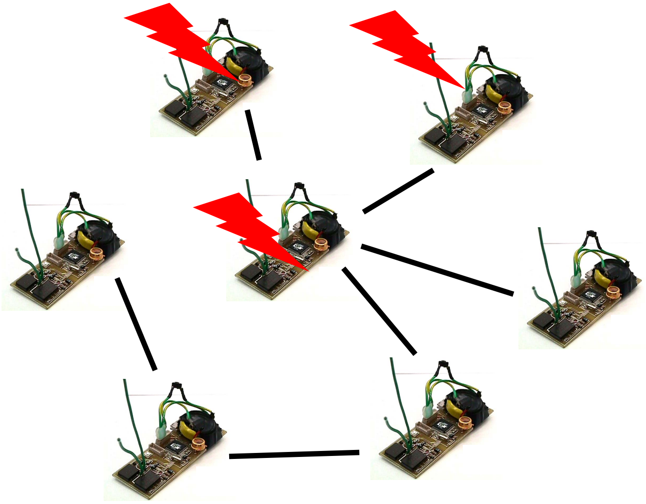
Example: A Distributed Sensor System

- Based on laptops
  - not a classic sensor network

- Earthquake network
  - Jesse Lawrence (Stanford), Elizabeth Cochran (Riverside)
  - E.g., Apple laptops since 2005 are outfitted with accelometers, to protect harddrive when falling; or USB shake sensors
  - Fill the „gaps“ between seismometers already in place in California.

- Goal: early warning of quakes based on gentle waves before the more brutal ones come. (E.g., stop high-speed trains)

Algorithmic Perspective



- Given a sensor network
  - Local event: connected subset of nodes senses event (simultaneously)
  - „connected event component“

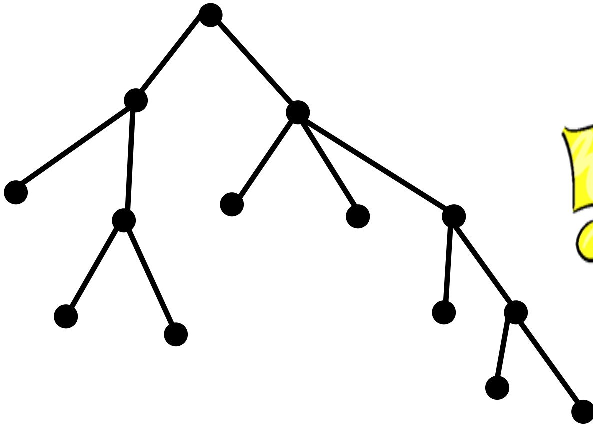
- Goal of distributed algorithm
  - Determine total number of nodes which sensed the event (size of event component)
  - Algorithm should be fast
  - Output sensitive: In case of „small disasters“, only a small number of messages is transmitted.
  - In case of large disasters, an alarm can be raised (e.g., priority = depends on event component size)

Model

- Preprocessing of graph is allowed
  - Only unknown: subset of nodes where event will happen
- Evaluation
  - Time complexity: time needed until *at least one node* knows event component size s
  - Communication complexity: total number of messages sent
- Assumptions
  - All nodes sense event simultaneously
  - „Synchronous“ environment (upper bound on message transmission time)
  - Nodes which did not sense event can also help to disclose the disaster by forwarding messages (on-duty model)
  - Only one event (can easily be generalized)

Appetizer: What about the Tree?

- Efficient disaster disclosure on undirected tree?



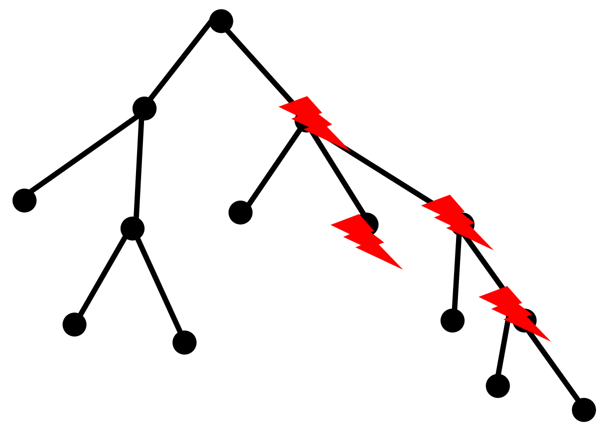
Time O(d), Messages O(s)

d ... Diameter of component

s ... Size of component

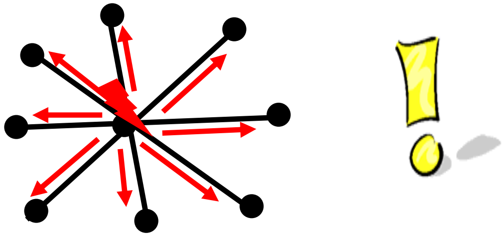
=> asymptotically optimal!

- Idea: in preprocessing phase, make the tree directed!
- At runtime: each node v immediately informs its parent in case of an event; subsequently, wait until all event-children counted the total number of event nodes in their subtrees



The Neighborhood Problem (1)

- A first challenge for general graphs: how can a node find out which of its neighbors also sensed the event? Called the neighborhood problem.
- Asking all neighbors is expensive: e.g., star graph where only center has event:



event component size s=1,
but requires n-1 messages!

- Better idea: only ask neighbors with higher degree? Works for this example! But what about the complete graph? Lower bound n??

The Neighborhood Problem (2)

- Idea: construct a sparse neighborhood cover in preprocessing phase!

  - A set of node sets with certain properties

- Concretely: cover ensures small diameter („local"), where at least one set includes t-neighborhood of each node (for parameter t), and where nodes are in not too many sets (small membership count)

Definition 2 (Sparse (k,t)-neighborhood Cover). *[1] A (k,t)-neighborhood cover is a collection of sets (or clusters) of nodes $S_1, ..., S_r$ with the following properties: (1) $\forall\ v$, $\exists\ i$ such that $N_t(v) \subseteq S_i$, where $N_t(v) = \{u | dist_G(u,v) \leq t\}$, and (2) $\forall i$, $Diam(S_i) \leq O(kt)$.*

A (k,t)-neighborhood cover is said to be sparse if each node is in at most $kn^{1/k}$ sets.

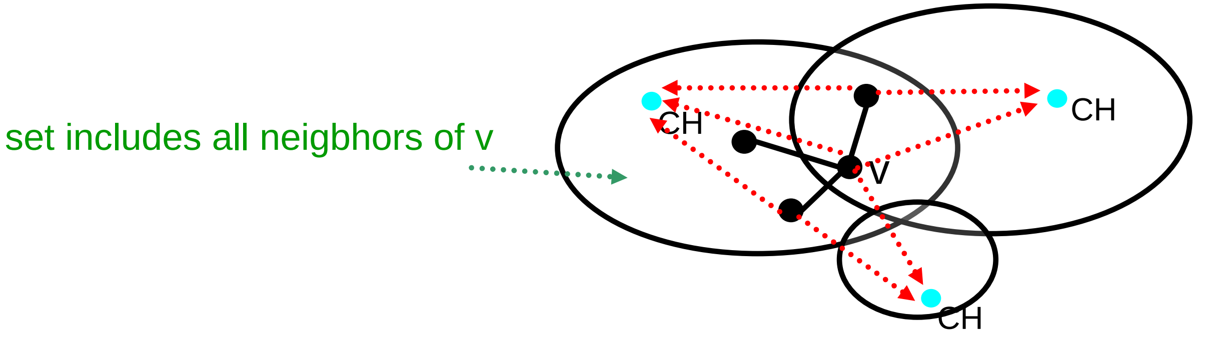
Theorem 1. *[1] Given a graph $G = (V, E)$, $|V| = n$, and integers $k, t \geq 1$, there is a deterministic (and distributed) algorithm which constructs a t-neighborhod cover in G where each node is in at most $O(kn^{1/k})$ clusters and the maximum cluster diameter is $O(kt)$.*

The Neighborhood Problem (3)

- Solution with neighborhood cover:

  - Preprocessing: compute (log n, 1)-neighborhood cover (clusters with log diameter, nodes in at most log sets, 1-neighborhoods included); for each set, define a cluster head (CH) (e.g., node with smallest ID), and compute shortest paths to CH

  - Runtime: Event node informs all its cluster heads, which will reply with corresponding neighbor list

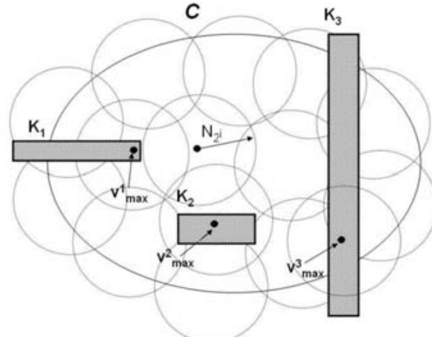


- Analysis (of neighborhood problem only):
  - Time O(log n) and O(s polylog(n)) messages
  - Small cluster diameter ensures fast termination
  - Small membership count / sparseness ensures low message complexity

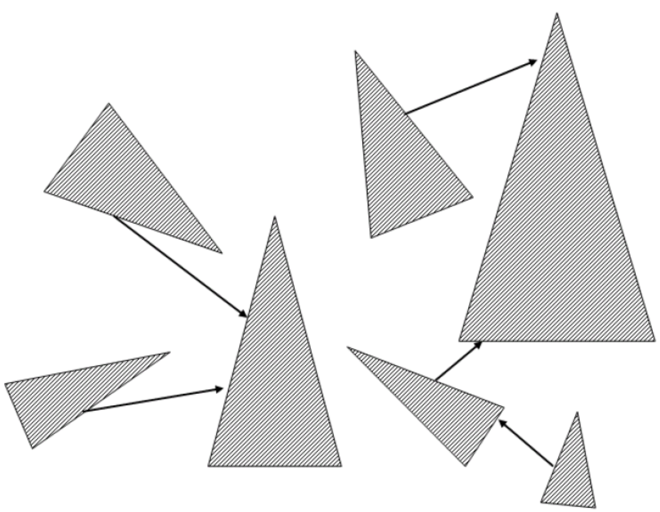
Disaster Disclosure on General Graphs

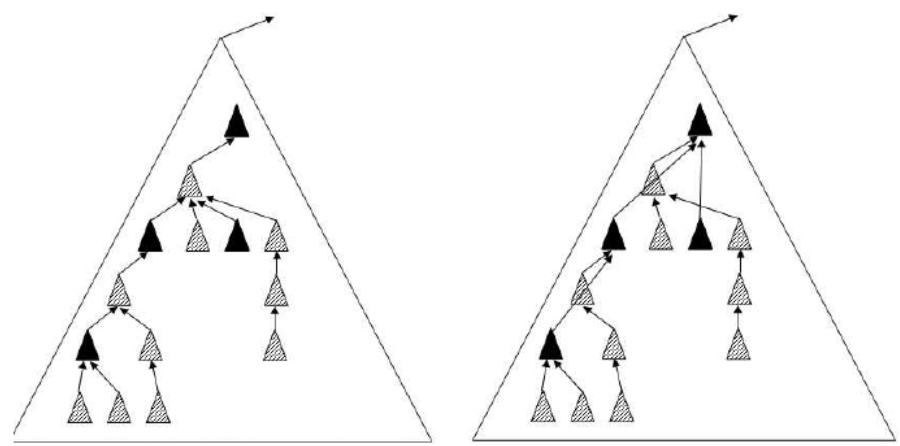
- How to compute the event component size in general graphs?

- Algorithm 1: Hierarchical network decomposition



- Algorithm 2: Merging trees and pointer jumping





Hierarchical Network Decomposition (1)

- Use exponential hierarchy of covers: $D_1 = (\log n, 1)$, $D_2 = (\log n, 2)$, $D_3 = (\log n, 4)$, ..., $D_i = (\log n, 2^i)$, neighborhood increases exponentially

  - diameter also increases, sparseness remains logarithmic

  - then: CHs and shortest paths

- Runtime:

  - First all event nodes in active state

  - Contact CHs to learn 1-neighborhood (cover log n, 1)

  - Then, go to larger decompositions iteratively

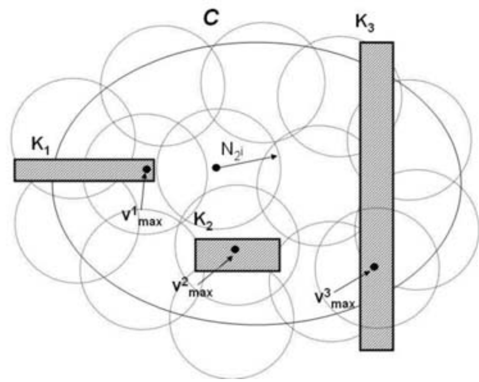
  - Active nodes inform CHs about event component K part they already know

  - Cluster head does the following:

    (1) if component entirely contained in cluster => output size, done.

    (2) if component hits boundary of cluster, determine node with largest ID in component K; if this node‘s entire 2^i neighborhood is contained in C, make this the only remaining active node, otherwise set all nodes to passive (=> not too many nodes continue exploration, low message complexity).

Hierarchical Network Decomposition (2)



- Observation:
  - largest node in component always survives (until entire component included)
  - in phase i, at least 2^i nodes have to be passive for an active node
  - number of active nodes decreases geometrically

Algorithm 1 ALG_{DC}

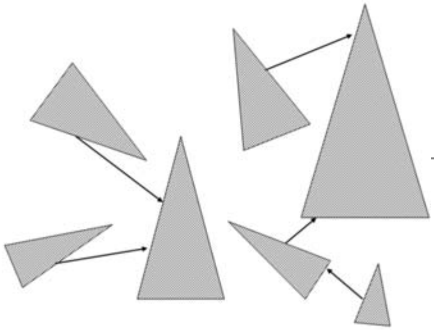
```
(* Global Preprocessing *)
for i from 0 to log d:
    D_i := (log n, 2^i)-NC;
(* Runtime *)
i := 1;
∀v ∈ V': v.active := true;
while (∃v : v.active = true)
    ∀ active v: notify v's cluster heads in D_i;
    for all clusters C
        let K := {K_1, ..., K_r} be C's components;
        ∀K ∈ K:
        if (K ⊆ C): output(size(K));
        else
            v_max := max{i | i ∈ (K ∩ C)};
            ∀v ∈ K: v.active := false;
            if (N_{2^i}(v_max) ⊆ C)
                v_max.active := true;
    i++;
```

Runtime O(d log n) and at most O(s log d log n) messages needed.

d... weak diameter
of component
(ess. last cluster diameter)

in each phase s log n messages are sent,
and there are log d many phases

Merging Forests and Pointer Jumping



- Idea:
  - Solve neighborhood problem with (log n, 1)-cover
  - Each event node selects parent = neighboring event node with larger ID (if any)
  - Start merge forest: learn about root („pointer jumping“) and join the largest neighboring tree
  - Hence, in phase i, minimal tree is of size at least 2^i

Algorithm 2 ALG_{PJ}

while $(\exists v$ s.t. $v.parent \neq root)$
 $\forall v \in \mathcal{T}$:
  with $prob = 1/2$ $v.bit := 0$, **else** $v.bit := 1$;
 $\forall v \in \mathcal{T}$:
  if $(v.bit = 0 \wedge v.parent.bit = 1)$
   $IS := IS \cup \{v\}$;
 $\forall v \in IS$:
  $v.parent = v.parent.parent$;

Algorithm 3 ALG_{FOREST}

$\forall v \in V$: **define** $v.parent$;
let $\mathcal{T} := \{T_1, ..., T_f\}$ be set of resulting trees;
while $(|\mathcal{T}| > 1)$ **do**
 $\forall T \in \mathcal{T}$: $ALG_{PJ}(T)$;
 $\forall T \in \mathcal{T}$:
  $T_m := max\{X | X \in \mathcal{T}, adjacent(X, T)\}$;
  if $(T < T_m)$: **merge** $T \triangleright T_m$;
 update $\mathcal{T}$: set of resulting trees;

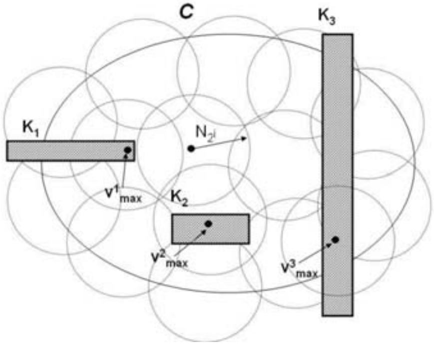
Runtime O(d log s + log s log n) and at most O(s log s (d+log n)) messages needed.

[Time: log n for neighborhood problem. Single tree after log s phases, as tree size doubles. Star conversion with PJ in log s time, each hop taking time at most d. Amortized over all phases also d log s. Convergecasts take time d. Asking children about size of neighboring trees is neighborhood problem, time log n, for each of log s many rounds.]

Summary

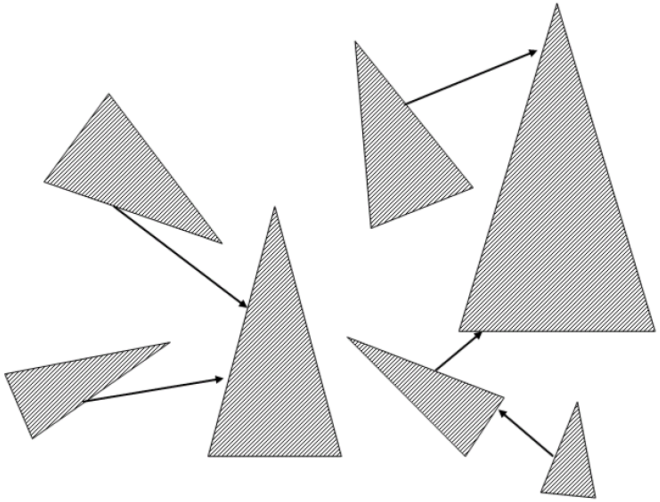
- Easy in special graphs, e.g., on trees

- Algorithm 1: Hierarchical network decomposition



Runtime O(d log n) and at most
O(s log d log n) messages needed.

- Algorithm 2: Merging trees and pointer jumping



Runtime O(d log s + log s log n) and at most O(s log s (d+log n)) messages needed.

Conclusion

- Distributed event detection and alarming

- Two first algorithms
  - Network decomposition
  - Merging trees

- Open problems
  - Alternative algorithms? Distributed MST construction on general graphs?
  - Off-duty model: Non-events node are in sleep mode
  - Lower bounds
  - Smaller messages?
  - Faulty environments when components are not necessarily connected?
  - Dynamic case: e.g., detection of large wave fronts?
  - etc.

Dziekowac!

Slides and papers at

http://www14.informatik.tu-muenchen.de/personen/schmiste/