Concurrent greedy with cleanup

For certain families of graphs, concurrent greedy with cleanup runs in $O(n)$ time

Requirements:

(i) All graphs in the family are sparse: $m = O(n)$

(ii) The family is closed under edge contraction: combine both ends of the edge into a single vertex, with an edge to any vertex that was adjacent to either end
If $m \leq cn$, concurrent greedy with cleanup takes
\[ O(c(n + n/2 + n/ 4 +...)) = O(n) \] time

**Trees**: $m < n$, closed under contraction, no cleanups needed

**Planar graphs**: $m < 3n$, closed under contraction

For concurrent greedy (with or without cleanups) to run in $O(m)$ time on a arbitrary graph, we need a way to “thin” a graph: using red rule, color red all but $O(n)$ edges, in $O(m)$ time.
Faster algorithms for general graphs

$O(m\lg\lg n)$ Yao 1975, packets

Run concurrent greedy algorithm, but with only $m/\lg n$ edges. To do this, group edges incident to each vertex into packets, each of size $\lg n$ (with at most one small packet per vertex). Give the main algorithm only the minimum-weight edge in each packet.

Time to find packet minima is $O(m\lg\lg n)$. 
O(mlg*n) Fredman & Tarjan 1984, F-heaps

Store the set of edges incident to each blue tree in an F-heap (or rank-pairing heap). Run single-source greedy until blue tree is big enough; then choose an unconnected source and run single-source from it. Repeat until all blue trees are big enough. Clean up. Repeat. Algorithm is a hybrid of single-source and concurrent greedy with cleanup. Each round takes O(m) time. If blue trees have size at least $k$ before a round, they have size at least $2^k$ after $\rightarrow \lg^*n$ rounds.
$O(m \lg \lg^* n)$ Gabow et al. 1986, F-heaps + packets

$O(m)$ Karger et al. 1995, thinning + random sampling

$O(m \alpha(n))$ Chazelle 1998, soft heaps + complicated hybrid algorithm

$O(\text{minimum})$ Pettie & Ramachandran 2002, Chazelle’s algorithm with fixed-depth recursion + brute force for small subproblems
The power of random sampling

Concurrent greedy + thinning

How to thin?
A related question: MST verification

*Given a spanning tree $T$, is it an MST?*

**Yes,** if and only if every non-tree edge $(v, w)$ has maximum weight on the cycle formed with the path in $T$ joining $v$ and $w$.

**Proof:** Red rule

**Use the same idea to thin:** given any forest (set of vertex-disjoint trees), can color red any non-tree edge whose ends are in the same tree and whose weight is maximum on the cycle formed with tree edges.
Thinning using a forest
Thinning using a forest
How to find maxima on tree paths?

For now, assume $O(m)$

How to find a good forest? Best is an MST, but too expensive to compute

**Good enough**: an MSF (minimum spanning forest) of a random sample of the edges. (The sample subgraph may not be connected)
Randomized minimum spanning tree algorithm

Concurrent greedy with occasional thinning

Let $b = \text{#blue trees, initially } n$, 
$e = \text{#uncolored edges, initially } m$
$c = \text{a constant to be chosen later}$

while $b > 1$ do

if $e < cb$ then one pass of concurrent greedy
else thinning step
Thinning step

Sample the uncolored edges by selecting each edge independently with probability $\frac{1}{2}$

Find an MSF of the sample by applying the MST algorithm recursively to each connected component of the sample

Color **red** all sampled edges not in the MSF and all non-sampled edges maximum on a cycle with MSF edges

After thinning, expected #uncolored edges $\leq 2b$
Expected running time

\[ R(e) \leq O(e) + R(e - b/2) \text{ if sparse} \]
\[ \leq O(e) + R(e/2) + R(2b) \text{ if dense} \]

Sparse: \( e < cb \rightarrow b/2 > e/(2c) \)
\[ \rightarrow e - b/2 < e(1 - 1/(2c)) \]

Dense: \( e \geq cb \rightarrow 2b \leq 2e/c \)
\[ \rightarrow e/2 + 2b \leq e(1/2 + 2/c) \]

\[ c = 5 \rightarrow R(e) \leq O(e) + R(9e/10) = O(e) \]
After thinning, expected #uncolored edges $\leq 2b$

**Proof**: Think of building the MSF $F$ of the sample in the following way: Process the edges in increasing order by weight. To process $(v, w)$, flip coin. If heads, put $(v, w)$ in sample: if ends in same tree, color red; otherwise, add to $F$. If tails, not in sample: if ends in same tree, color red; otherwise, not in sample, not colored.
Proof (cont.): Do coin flip after testing whether ends are in same tree: if ends in same tree, color red; otherwise, flip coin, add to $F$ if heads. This change has no effect on the outcome: $F$ is the same, as is the set of red edges. (The outcomes of the coin flips on the red edges have no effect.)

Expected #uncolored edges = expected #coin flips = expected #flips until $b-1$ heads. Each flip increases expected #heads by $\frac{1}{2} \rightarrow$ expected #flips = $2(b-1)$. 
Finding maxima on tree paths

*Convert to a problem on a shallow tree*

Given tree $T$ with edge weights, the *Borůvka tree* $B(T)$ is formed from $T$ by running the concurrent greedy algorithm on $T$. Tree $B$ contains one node for each blue tree formed. Each leaf of $B$ is a vertex of $T$; each non-leaf is a blue tree containing $>1$ vertex; the root is the final blue tree. Node $x$ is the parent of node $y$ in $B$ if $y$ is a blue tree before some pass $k$ and $x$ is the blue tree containing the vertices of $y$ after pass $k$. The weight of edge $(x, y)$ is the weight of the edge incident to $y$ selected during pass $k$. 

Minimum spanning tree

Diagram showing a network with labeled nodes and edges indicating connections with weights.
Borůvka tree
If $T$ has $n$ vertices, $B$ has $<2n$ nodes, $\leq n/2^k$ of depth $k$

The concurrent greedy algorithm can build the Borůvka tree of the MST as it builds the MST

$T(v, w) =$ path joining vertices $v$ and $w$ in $T$
$B(v, w) =$ path joining nodes $v$ and $w$ in $B$
$p(v) =$ parent of $v$ in $B$
For any \( v, w \) in \( T \),
\[
\max\{c(x, y) \mid (x, y) \text{ on } T(v, w)\} = \max\{c(x, y) \mid (x, y) \text{ on } B(v, w)\}
\]

**Proof:**

\((\leq)\): Let \((x, y)\) have maximum weight on \( T(v, w) \). Let \( U \) be a blue tree that selects \((x, y)\). Deleting \((x, y)\) from \( T \) forms a cut \( X, Y \) with one of \( v \) and \( w \) in \( X \) and the other in \( Y \). Let \( x \) and \( v \) be in \( X \), so \( y \) and \( w \) are in \( Y \). Since \((x, y)\) has maximum weight on \( T(x, y) \), \( v \) but not \( w \) is in \( U \). The edge \((U, p(U))\) has weight \( c(x, y) \), and this edge is on \( B(v, w) \).

\((\geq)\): Let \((U, p(U))\) be any edge on \( B(v, w) \). Let \( v \) be in \( U \), so \( w \) is not in \( U \). Let \((x, y)\) be the edge on \( T(v, w) \) with exactly one end in \( U \). Then \( c(x, y) \geq c(U, p(U)) \).