Distributed computation vs. sublinear time algorithms

Today: Connection between sublinear time algorithms and local distributed algorithms (local = constant number of communication rounds) on sparse graphs

[Parnas Ron]
Sparse Graphs

- **Representation:**
  - Max degree $d$
  - Adjacency list format

- **Problems of interest:**
  - Vertex cover
  - Matching
  - Dominating set
  - Set cover (on sparse set systems)
Vertex Cover (VC)

Def. $V' \subseteq V$ is a **vertex cover** if $\forall e=(u,v) \in E$, either $u \in V'$ or $v \in V'$. 

Recall: In a degree-d graph: $\text{VC} \leq \frac{m}{d+1}$
Approximation for VC

- Multiplicative?
  - VC of graph with no edges vs. graph with 1 edge

- Additive?
  - Need to allow some multiplicative error: Computationally hard to approximate to better than 1.36 factor (maybe even 2…)

- Combination?
  - Def. $y'$ is $(\gamma, \delta)$-estimate of $y$ if
    $$y \cdot y' \cdot \gamma y + \delta$$

Good for minimization problems
Distributed Algorithms (simple version)

- Network
  - Processors
  - Links
  - (assume maximum degree is known to all)
- Communication round
  - Each node sends message to each neighbor

- Vertex Cover Problem:
  - Network graph = input graph
  - After $k$ rounds, each node knows if it is in VC
    (no guarantees that any other node knows about you!)
Main insight

in k-local (k-round) algorithm for vertex cover, output of node v can depend only on nodes of distance at most k from v

number of such nodes $\leq d^k$ (ok if $d,k$ constants)

(note: this works even for randomized algorithms if you can reproduce their random bits!)
Connection for Vertex Cover

Thm [Parnas Ron]: $t$-round distributed algorithm for vertex cover yields $d_{\text{max}}^{O(t)}$ sequential query approximation algorithm for vertex cover.

Reduction idea:
• Sample vertices of graph
• For each sampled vertex $v$, simulate distributed algorithm to see if $v$ is in VC
• Output $(\text{fraction in VC}) \cdot n$
A “local algorithm” for vertex cover

- **Vertex Cover algorithm**: (max degree $d$)
  - $i \leftarrow 1$
  - While edges remain:
    - Remove vertices of degree $> d / 2^i$ and adjacent edges
    - Update degrees of remaining nodes
  - Increment $i$
  - Output *all removed vertices* as VC

- **How many rounds?**
  \[ \log d \]
Example run of Parnas-Ron

Remove vertices of degree $\geq 8$
Remove vertices of degree $\geq 4$
Remove vertices of degree $\leq 1$
Why a Vertex Cover?

- Vertex Cover algorithm:
  - $i \leftarrow 1$
  - While edges remain:
    - Remove vertices of degree $> \frac{d_{\text{max}}}{2^i}$ and adjacent edges
    - Update degrees of remaining nodes
    - Increment $i$
  - Output *all removed vertices* as VC

- No edges remain at end – all removed along with adjacent vertex
Why a good approximation?

Let $\text{VC}_G = \text{size of min vertex cover of } G$

Theorem: $\text{VC}_G \cdot |C| \cdot (2\log d + 1) \text{VC}_G$

Proof:

$\text{VC}_G \cdot |C|:$
- Algorithm removes edges only if at least one endpoint placed in cover
- All edges gone at end
Why a good approximation? (cont.)

Theorem: \( VC_G \cdot |C| \cdot (2\log d + 1) VC_G \)

Need to prove:
\( |C| \cdot (2\log d + 1) VC_G \)
Approximation algorithm for vertex cover

On input G, with max degree d, there is an $O(d^{O\left(\log d\right)} / \varepsilon^2)$ (sequential) time algorithm which outputs $\bar{\mathcal{V}}$ such that

$$VC_G \cdot \bar{\mathcal{V}} \cdot (2\log d + 1) \cdot VC_G + \varepsilon^2 \cdot n$$

- Proof: $O(\log d)$ round distributed algorithm + Parnas-Ron theorem
- No dependence on $n$
- Can get $O(1)$ multiplicative estimates and faster runtimes in terms of $d, \varepsilon$
Constant time approximation algorithms for sparse graphs

- Paradigm + local algorithms yield \textit{constant time} approximation algorithms for bounded degree graph problems \cite{Parnas Ron} \cite{Kuhn Moscibroda Wattenhofer} \cite{Marko Ron}
  
  - Applies to vertex cover, maximum matching, dominating set, sparse set cover, sparse positive linear programs, …
Change of context....
Large inputs
Large outputs
When we don’t need to see all the output…

do we need to see all the input?
Local Computation Algorithms

Input $x$

$LCA$

Output $y$

Difficulty: consistency with a single legal answer

$i_1, i_2, \ldots \rightarrow LCA \rightarrow y_{i_1}, y_{i_2}, \ldots$

Output $y$
A first example:

Maximal Independent Set
Maximal independent set

- Sparse undirected graph $G=(V,E)$, degree at most $d$ (constant)
- Independent set: subset $V'$ of $V$ such that no two vertices connected by an edge
Maximal Independent Set

- Subset $I$ of vertices $V$ of undirected graph $G = (V,E)$ is **independent** if no two neighbors are in $I$.

- Independent set $I$ is **maximal** if no strict superset of $I$ is independent.
  - But might not be maximum!
Maximal independent set

- Central problem:
  - Important optimization tool: e.g., task scheduling
  - Distributed computing
    - “Can I broadcast without conflicting with neighbors?”
    - Communication in wireless sensor nets
    - Construct backbone in ad-hoc wireless network
  - Drosophila brain development (choice of leader cells)
- …
Maximal independent set

Is node $u$ in the maximal independent set?
A fast (but not space efficient) local computation algorithm

- Lazy Greedy Algorithm: (initially, MIS is empty)
  - Query: “Is node $u$ in the MIS?”
  - Answer: if neighbors of $u$ not in MIS, then put $u$ into it (and remember this decision!)

- Requirements: $O(d)$ time, $O(n)$ space

- Note:
  - Answer depends on query order
  - Must remember past choices
  - Can’t allow parallel copies of algorithm
A new challenge: Consistency

- Many possible MIS solutions –
  - Which one? Must be consistent with past answers!
  - What if questions asked in parallel?
A hope

- Can we find MIS algorithm $A$ for which output for node $v$ depends only on few inputs? Then simulate $A$’s behavior for $v$!
  - E.g., [a la Parnas Ron] if there is a $k$ round distributed algorithm for MIS, then:
    - $v$’s output depends only on inputs and computations of $k$-ball around $v$
    - Can read/simulate in $d^k$ time!

- But how big is $k$?

Big Graph

k neighborhood of $v$
How fast can MIS be computed?

- Lexicographically-first-MIS is P-complete [Cook]
- Randomized $O(d \log n)$ rounds [Luby]
  - Yields $d^{\mathcal{O}(\log n)} = n^{\mathcal{O}(\log d)}$ time LCA
- $O(d \log d + \log^* n)$ rounds [Barenboim Elkin]
  - Yields $(d^{O((d + \log^* n) \log d)})$ time LCA
  - yields best LCA!
Here – a nonoptimal algorithm, but not bad!

A distributed algorithm for MIS (simplification of [Luby]):

- repeat $O(\log d)$ times in parallel
  - each vertex selects itself with probability $1/2d$
  - if $v$ selects itself and no neighbor selected
    - add $v$ to MIS
    - remove $v$ and $N(v)$ from the graph
Distributed algorithm for MIS
Distributed algorithm for MIS
How many rounds?

Theorem: The number of phases is \( \leq 8d \log n \) with probability at least \( 1 - \frac{1}{n} \)

Corollary: \( E[number \ of \ phases] \) is \( O(d \log n) \)
Global behavior?
After running $O(d \log d)$ rounds of Luby’s distributed algorithm ...

Remaining live components are all small!
New “two-phased” plan

On input $v$:

- Get partial solution which answers most queries and breaks up the graph into small components:
  - Simulate $O(d \log d)$ rounds of Luby’s algorithm from $v$’s perspective
  - LCA’s simulation runtime: $O(d^d \log d)$

- If $v$ not determined yet, solve the whole surviving component that contains it:
  - If $v$ not yet in “MIS” or neighbor of “MIS” then:
    - Let $C_v$ be the “still live” component of vertices containing $v$
    - Solve MIS in $C_v$ to determine if $v$ is in MIS
  - LCA’s Runtime: $O(\text{size of } C_v \cdot d^d \log d)$
Two remaining details:

- How big is \( v \)'s component?
- How do we simulate the random bits consistently?
How big is v’s component?

- Claim: whp, after phase 1, for all v, size of $C_v$ is $O(poly(d) \cdot \log n)$

- Why?
  - What you want to say (but can’t):
    - the probability that any node survives is small, say $p$
    - assume for a minute (this is far from true) that the survival of a node is independent of the survival of all other nodes.
    - consider any connected component of size $k$, then the probability it survives is at most $p^k$
    - so the probability that a connected component of size $k$ survives is at most (number of connected components of size $k$) * $p^k$
How big is v’s component?

- Claim: whp, after phase 1, for all v, size of $C_v$ is $O(poly(d) \cdot \log n)$

- Why?
  - What you want to say (but can’t):
    - the probability that any node survives is small, say $p$
    - assume for a minute (this is far from true) that the survival of a node is independent of the survival of all other nodes.
    - consider any connected component of size $k$, then the probability it survives is at most $p^k$
    - so the probability that a connected component of size $k$ survives is at most (number of connected components of size $k$) $\times p^k$

- Problem: in a general graph, too many connected components, and the survival probabilities not independent.

  - Take advantage of the fact that this is degree $d$ bounded
  - show that in any large connected component, there are many nodes whose survival in any round is independent
    - (i.e. those that are at least distance 3)
How big is \(v\)'s component?

- **Claim:** w.h.p., after phase 1, for all \(v\), size of \(C_v\) is \(O(\text{poly}(d) \cdot \log n)\)

- **Why?**
  - (As in Beck’s analysis for algorithmic Lovasz Local Lemma)
    - Any large surviving component must have a large tree as a subgraph
    - Degree bound implies that any large tree has a large subset of nodes that are at least distance 3 from each other
      - hence their presence/nonpresence in MIS is independent (requires slight modification of algorithm)
    - The probability that any such subset survives is at most \((\text{number of trees of size } k) \cdot (\text{probability that } k \text{ independent nodes survive})\)
How do we simulate random bits?

- Naïve implementation:
  - Write down all $O(dn \log n \log d)$ random bits?

- Fewer random bits:
  - $O(\text{polylog } n)$-independent bits suffice
  - Write down polylog $n$ size seed
Put everything together...

Theorem: For any degree $d$ bounded graph, there is an $O(d \log n)$ space, $O(d^{O(d \log d)} \log^2 n)$ time local computation algorithm for MIS.

Recent improvement: $O(d^{O(d \log d)} \log n)$ time

[Mansour Rubinstein Vardi Xie]
Local Computation Algorithms: A model
Local Computation Algorithms (LCAs)

- **F**: a computation problem
  - input \( x (|x| = n) \)
  - set of legal solutions \( F(x) = \{y_1, y_2, \ldots, y_s\} \)

- **LCA** implements oracle access to some \( y_k \)
  - For any sequence of queries \( i_1, i_2, \ldots, i_q \), LCA replies with \( (y_k)_{i_1} (y_k)_{i_2} \ldots (y_k)_{i_q} \) with
    - at most \( t(n) \) time per query
    - at most \( s(n) \) space per query
    - correct for all queries with probability \( 1 - \delta(n) \)
Local computation algorithms

Input: x (RAM)

LCA

random string

work space

$y_1$

$y_2$

$y_k(i_2)$

$y_k(i_q)$

$y_{k(i_1)}$

$y_{k(i_q)}$
Wish list for LCAs

- Fast: $t(n)$ at most polylog
- Space efficient: $s(n)$ at most polylog
- Parallel queries ok
- Query order oblivious

Cloud computation: can coordinate without too much communication
One takeaway message

- Kill several birds with one stone:
  - Space, parallelism, history independence and consistency are related!
Open questions

- Better techniques?
  - dependency on $d$ is exponential: is polynomial possible? (some hope: [Yoshida, Yamamoto, Ito, Onak, Ron, Rosen, Rubinfeld])
  - Hypergraph coloring with LLL parameters? (e.g. based on [Moser, Tardos]?)

- Other problems?
  - Resource management (when is my tennis court time?), managing coalitions, MSTs, spanners, string matching

- Dynamic LCAs?