MPC Algorithm for $(1+\epsilon)$ Maximum Matching

Jaehyun Koo

October 11, 2023

Distributed Computation Models

The classic way of circumventing Moore's law is **Parallel Computing**.

Parallel Computing assumes **many** processors and **shared** memory.
Generally, all the information needed is in the right place.

Distributed Computation Models

The classic way of circumventing Moore's law is **Parallel Computing**.

Parallel Computing assumes **many** processors and **shared** memory. Generally, all the information needed is in the right place.

The modern way of circumventing Moore's law is **Cloud Computing**.

(which we will call distributed computing)

Distributed Computing can utilize much more processing powers with freedom. However, the information needed is not in the right place, and the network is inherently expensive.

Distributed Computation Models

In this slide we will assume the **Massively Parallel Computation** model with near-linear memory.

We solve a problem on graph G with n vertices and m edges with max degree Δ. There are $\tilde{\Theta}(\Delta)$ machines with memory size $S = \tilde{\Theta}(n)$.

Each machine has infinite computation power, while we *prefer* to have a near-linear algorithm (and it usually be in such a way).

Distributed Computation Models

In this slide we will assume the **Massively Parallel Computation** model with near-linear memory.

We solve a problem on graph G with n vertices and m edges with max degree Δ. There are $\tilde{\Theta}(\Delta)$ machines with memory size $S = \tilde{\Theta}(n)$.

Each machine has infinite computation power, while we *prefer* to have a near-linear algorithm (and it usually be in such a way).

In each round, each machine can send messages to any other machine, but the sum of sent/received messages should not exceed S.

Distributed Computation Models

In this slide we will assume the **Massively Parallel Computation** model with near-linear memory.

We solve a problem on graph G with n vertices and m edges with max degree Δ. There are $\tilde{\Theta}(\Delta)$ machines with memory size $S = \tilde{\Theta}(n)$.

Each machine has infinite computation power, while we *prefer* to have a near-linear algorithm (and it usually be in such a way).

In each round, each machine can send messages to any other machine, but the sum of sent/received messages should not exceed S.

All vertices and edges are given to a random machine.

With $O(1)$-round preprocessing, we can assume for each vertex, there exists a machine storing all adjacent vertex.

Maximum Matching

Given an undirected graph, a maximum matching is a set of edges that do not share an endpoint.

We can relax this as an LP: We find a vector $x : E \to [0, 1]$ such that for each vertex the sum of x_e is at most 1.

Maximum Matching

Given an undirected graph, a maximum matching is a set of edges that do not share an endpoint.

We can relax this as an LP: We find a vector $x : E \to [0, 1]$ such that for each vertex the sum of x_e is at most 1.

The paper we are presenting claims $O(\log \log n)$ round $(1 + \epsilon)$ approximation to the maximum matching.

The presentation will only deal with finding a fractional solution that is at least $\frac{1}{2+\epsilon}$ of maximum matching. The rest is mostly just rounding and applying previously known results.

Good thing is that it also finds $2 + \epsilon$ approximation min vertex cover.

An $O(\log n)$ round algorithm

An $O(\log n)$ round algorithm

- Initialize $x_e = \frac{1}{\Delta}$ for all $e \in E$.
- The vertex v is **frozen** if $\sum_{e \ni v} x_e \geq 1 - 2\epsilon$.
- The edge e is **frozen** if one of its endpoints are frozen.
- For each round, if there are unfrozen edges, assign $x_e = \frac{x_e}{1-\epsilon}$ for all unfrozen edges.

An $O(\log n)$ round algorithm

An $O(\log n)$ round algorithm

- Initialize $x_e = \frac{1}{\Delta}$ for all $e \in E$.
- The vertex v is **frozen** if $\sum_{e \ni v} x_e \geq 1 - 2\epsilon$.
- The edge e is **frozen** if one of its endpoints are frozen.
- For each round, if there are unfrozen edges, assign $x_e = \frac{x_e}{1-\epsilon}$ for all unfrozen edges.

The algorithm is expressed in centralized way, but it's easy to see that this is an MPC algorithm.

An $O(\log n)$ round algorithm

An $O(\log n)$ round algorithm

- Initialize $x_e = \frac{1}{\Delta}$ for all $e \in E$.
- The vertex v is **frozen** if $\sum_{e \ni v} x_e \geq 1 - 2\epsilon$.
- The edge e is **frozen** if one of its endpoints are frozen.
- For each round, if there are unfrozen edges, assign $x_e = \frac{x_e}{1-\epsilon}$ for all unfrozen edges.

The algorithm is expressed in centralized way, but it's easy to see that this is an MPC algorithm.

We need to prove three things: Algorithm requires $O(\log n)$ rounds, **frozen** vertices form a small vertex cover, and x_e is a large fractional matching.

An $O(\log n)$ round algorithm: Proof

Frozen vertices obviously form a vertex cover, since otherwise there should be unfrozen edges.

Num of frozen vertices are at most $(2+5\epsilon)$ of Min Vertex Cover

- Let S be the frozen vertices. We prove that $|S| \leq \frac{2}{1-2\epsilon}\sum x_e$.
- Multiply x_e by $\frac{2}{1-2\epsilon}$.
- Say that each edges have x_e dollars - we distribute half of them to each two vertices.
- Frozen vertices have at least 1 dollars by definition, proof is complete.
- For sufficiently small ϵ we have $|S| \leq \frac{2}{1-2\epsilon}\sum x_e \leq (2+5\epsilon)\sum x_e$.

An $O(\log n)$ round algorithm: Proof

Num of frozen vertices is at most $(2+5\epsilon)$ of Min Vertex Cover

- We have $|S| \leq (2+5\epsilon)\sum x_e$.
- This is at most the maximum fractional matching.
- Maximum fractional matching coincides with minimum fractional vertex cover by LP Duality.
- Which in turn is at most the minimum integral vertex cover.

An $O(\log n)$ round algorithm: Proof

Num of frozen vertices is at most $(2+5\epsilon)$ of Min Vertex Cover

- We have $|S| \leq (2+5\epsilon)\sum x_e$.
- This is at most the maximum fractional matching.
- Maximum fractional matching coincides with minimum fractional vertex cover by LP Duality.
- Which in turn is at most the minimum integral vertex cover.

This itself gives an $(2+\epsilon)$ approx for minimum vertex cover.

$2 - O(1)$ approximation is hard under Unique Games Conjecture.

An $O(\log n)$ round algorithm: Proof

Algorithm requires $O(\log n)$ rounds

- After each $\frac{1}{\epsilon} = O(1)$ rounds, x_e is doubled. x_e can not exceed 1 as it will be freezed. Hence we have $\frac{\log \Delta}{\epsilon} = O(\log n)$ rounds.

An $O(\log n)$ round algorithm: Proof

Algorithm requires $O(\log n)$ rounds

- After each $\frac{1}{\epsilon} = O(1)$ rounds, x_e is doubled. x_e can not exceed 1 as it will be freezed. Hence we have $\frac{\log \Delta}{\epsilon} = O(\log n)$ rounds.

$\sum x_e$ is at least $\frac{1}{2+5\epsilon}$ of Integral Maximum Matching

- Any vertex cover is at least large as any maximum matching.
- (MM) $\leq |S| \leq (2+5\epsilon)x_e$

An $O(\log \log n)$ round algorithm

The $O(\log \log n)$ round algorithm performs similar computation compared to the $O(\log n)$ ones. However, it simulates the first $\frac{1}{100}O(\log n)$ rounds **without communication**.

Repeating this will give $O(\log \log n)$ rounds, since each communication *almost* halves the number of necessary rounds.

How?

An $O(\log\log n)$ round algorithm: Attempt without recursion

We randomly partition the set of vertices into $m = \sqrt{\Delta}$ sets. For each vertices, it selects one of m sets uniformly independently at random.

We use m machines, and each machine receives the induced subgraph of vertex partition.

Each machines runs the first $\frac{1}{100}O(\log n)$ rounds, assuming the edges outside of machines *will have roughly the same shape*.

After the emulation, we record the time where each vertex were frozen, and the edges not in the subgraph will be frozen accordingly.

An $O(\log\log n)$ round algorithm: Attempt without recursion

An almost correct $0.99 \times O(\log n)$ round algorithm

- Initialize $x_e = \frac{1}{\Delta}$ for all $e \in E$.
- Randomly partition the set of vertices into $m = \sqrt{\Delta}$ sets.
- Each machine $1, 2, \ldots, m$ holds the induced subgraph $G_1, G_2, \ldots, G_m$ of such partition.
- Let $y_v = \sum_{e \ni v} x_e$. We freeze vertex v when this is at least $1 - 2\epsilon$.
- We use the estimate $\tilde{y}_v = m \sum_{e \ni v, e \in G_i} x_e$.
- Using this we emulate $0.01 \times O(\log n)$ times locally and record the time when each vertex got frozen.

An $O(\log \log n)$ round algorithm: Attempt without recursion

An almost correct $0.99 \times O(\log n)$ round algorithm

- Using this we emulate $0.01 \times O(\log n)$ times locally and record the time when each vertex got frozen.
- For each edge not in any induced subgraph, adjust x_e accordingly per the minimum frozen time of two vertex. We call this value x_e^{MPC}, which is an estimate to the lesser extent.
- Using this we can compute the resulting $y_v^{MPC} = \sum_{e \ni v} x_e^{MPC}$.
- If $y_v^{MPC} > 1$, the vertex is bad and completely purged from G. If $y_v^{MPC} > 1 - 2\epsilon$, it is just freezed.
- Simulate remaining $0.99 \times O(\log n)$ rounds naively.

An $O(\log \log n)$ round algorithm: Attempt without recursion

The previous algorithm is **almost** correct. There is some technical issue, but we will fix it later. We will instead see how to recursively repeat this procedure.

Roughly, in the early stage the estimate is correct. But in the later stage, the relevant x_e are large and thus inconcentrated, while the number of sample is fixed. making m samples insufficient for estimation.

An $O(\log \log n)$ round algorithm: Attempt without recursion

The previous algorithm is **almost** correct. There is some technical issue, but we will fix it later. We will instead see how to recursively repeat this procedure.

Roughly, in the early stage the estimate is correct. But in the later stage, the relevant x_e are large and thus inconcentrated, while the number of sample is fixed. making m samples insufficient for estimation.

Fortunately, in the later stage, there are less relevant edges. After t rounds, the non-frozen edges form a graph with degree at most $\Delta(1-\epsilon)^t$.

Hence, after doing the first $\frac{1}{100}$ fraction of iteration, we reorganize the graph into smaller number of subgraphs.

An $O(\log \log n)$ round algorithm: Attempt without recursion

The previous algorithm is **almost** correct. There is some technical issue, but we will fix it later. We will instead see how to recursively repeat this procedure.

Roughly, in the early stage the estimate is correct. But in the later stage, the relevant x_e are large and thus inconcentrated, while the number of sample is fixed. making m samples insufficient for estimation.

Fortunately, in the later stage, there are less relevant edges. After t rounds, the non-frozen edges form a graph with degree at most $\Delta(1-\epsilon)^t$.

Hence, after doing the first $\frac{1}{100}$ fraction of iteration, we reorganize the graph into smaller number of subgraphs.

In some sense, each round **reduces the max degree** of relevant graph, so we can just repeat the same procedure according to the new max degree.

An $O(\log \log n)$ round algorithm

An almost correct $O(\log \log n)$ round algorithm

- If $\Delta \leq \log^{9999} n$, simulate $O(\log \Delta)$-round algorithm instead.
- Initialize $x_e = \frac{1}{\Delta}$ for all $e \in E$.
- Randomly partition the set of vertices into $m = \sqrt{\Delta}$ sets.
- Each machine $1, 2, \ldots, m$ holds the induced subgraph $G_1, G_2, \ldots, G_m$ of such partition. Let $G_0 = G_1 \cup G_2 \cup \ldots \cup G_m$.
- We use the estimate $\tilde{y}_v = m \sum_{e \ni v, e \in G_i} x_e + \sum_{e \ni v, e \in G \setminus G_0} x_e$.
- Using this we emulate $0.01 \times O(\log n)$ times locally and record the time when each vertex got frozen.
- Compute x_e^{MPC} and y_v^{MPC} as previously, freezing or deleting vertices if necessary.
- Reduce Δ and continue.

An $O(\log \log n)$ round algorithm: Proof

We need to argue three things: Two about $\tilde{y}_v, y_v^{MPC}$ being a correct estimate, and another about the induced subgraph fitting each machines.

The last fact is much easier to argue, so let's start with that.

Each subgraph have at most $O(n)$ edges

- We bound the probability where one subgraph gets very large.
- Let x_i be the probability where vertex i belongs to G_1. We will prove $\sum_{(i,j)\in E} x_i x_j$ is bounded whp.
- For a fixed x_i, x_j is independent. Let's say it has exactly Δ incident j's by adding dummy variable.
- $Pr[\sum x_j > 11E[\sum x_j]] \leq exp(-\frac{10}{3}E[\sum x_j]) = exp(-\frac{10m}{3})$
- WHP $\sum_{(i,j)\in E} x_i x_j \leq (\sum_{i \in G_1} x_i) \times 11m$.
- Apply Chernoff again, WHP we have $O(n/m)O(m) = O(n)$ edges in a subgraph.

An $O(\log \log n)$ round algorithm: Proof

We now argue that $\tilde{y}_v, y_v^{MPC}$ are good estimates.

An $O(\log \log n)$ round algorithm: Proof

We now argue that $\tilde{y}_v, y_v^{MPC}$ are good estimates.

Note that $\tilde{y}_v, y_v^{MPC}$ is exactly y_v if all freeze times are perfect. Vertex v gets freezed when $y_v \geq 1 - 2\epsilon$. All that really matters is whether if this binary predicate is false or true.

An $O(\log \log n)$ round algorithm: Proof

We now argue that $\tilde{y}_v, y_v^{MPC}$ are good estimates.

Note that $\tilde{y}_v, y_v^{MPC}$ is exactly y_v if all freeze times are perfect. Vertex v gets freezed when $y_v \geq 1 - 2\epsilon$. All that really matters is whether if this binary predicate is false or true.

Say that we obtained some upper bound for $|y_v - \tilde{y}_v|$. Even if the upper bound is small, it can go across the fixed value $1 - 2\epsilon$. How can we use this small upper bound to imply predicate whp?

An $O(\log \log n)$ round algorithm: Proof

We now argue that $\tilde{y}_v, y_v^{MPC}$ are good estimates.

Note that $\tilde{y}_v, y_v^{MPC}$ is exactly y_v if all freeze times are perfect. Vertex v gets freezed when $y_v \geq 1 - 2\epsilon$. All that really matters is whether if this binary predicate is false or true.

Say that we obtained some upper bound for $|y_v - \tilde{y}_v|$. Even if the upper bound is small, it can go across the fixed value $1 - 2\epsilon$. How can we use this small upper bound to imply predicate whp?

Workaround: Let $T_{v,t}$ be a function which returns a random real in $[1 - 4\epsilon, 1 - 2\epsilon]$. If we replace all $1 - 2\epsilon$ into $T_{v,t}$, we can argue that the predicate stays true whp.

This makes the algorithm correct. Now all we need is to prove it.

An $O(\log \log n)$ round algorithm: Proof

The algorithm have $O(\log n)$ **iterations** and $O(\log \log n)$ **phases**.

Def: Bad and good vertex

The vertex is bad if for any previous **iteration** it was frozen in the exact $O(\log n)$ algorithm but not in $O(\log \log n)$ algorithm, or the other way around. A vertex is good if not bad.

Lemma: Using upper bound to make predicate true

Consider the t-th iteration. If $|y_{v,t} - \tilde{y}_{v,t}| \leq \sigma$ for all active good vertex v, then v becomes bad in the t-th iteration with probability at most σ/ϵ and independently of other vertices.

An $O(\log \log n)$ round algorithm: Proof

The algorithm have $O(\log n)$ **iterations** and $O(\log \log n)$ **phases**.

Def: Bad and good vertex

The vertex is bad if for any previous **iteration** it was frozen in the exact $O(\log n)$ algorithm but not in $O(\log \log n)$ algorithm, or the other way around. A vertex is good if not bad.

Lemma: Using upper bound to make predicate true

Consider the t-th iteration. If $|y_{v,t} - \tilde{y}_{v,t}| \leq \sigma$ for all active good vertex v, then v becomes bad in the t-th iteration with probability at most σ/ϵ and independently of other vertices.

Proof. If $|\tilde{y}_{v,t} - T_{v,t}| > \sigma$ the vertex can not become bad. Let's say, the vertex always becomes bad otherwise. Then, among the choice from 2ϵ length interval, we can have at most 2σ length of bad choices, which can happen with probability at most σ/ϵ, and this is independent of other vertices.

An $O(\log \log n)$ round algorithm: Proof

On ground truths

We assume that after the end of each **phase**, the assignment $x_{e,t}$ is taken from the $x_{e,t}^{MPC}$. This doesn't really matter, since the algorithm doesn't care about the assignment as long as the freeze condition is sound.

What about the deleted vertex ($y_v^{MPC} > 1$)? We just pretend that they never existed at all. It's fine as long as there isn't too much of them (and there is not, we will not prove this here)

An $O(\log \log n)$ round algorithm: Proof

Defining $diff(v,t)$ to bound $|y_{v,t} - y_{v,t}^{MPC}|$

Let $x_{e,t}^{MPC}$ be the value of $x_{e,t}$ derived by the $O(\log \log n)$ algorithm.
Define $diff(v,t) = \sum_{e \in N(v)} |x_{e,t} - x_{e,t}^{MPC}|$.
As $y_{v,t} = \sum_{e \in N(v)} x_{e,t}$ and $y_{v,t}^{MPC} = \sum_{e \in N(v)} x_{e,t}^{MPC}$, we can see $|y_{v,t} - y_{v,t}^{MPC}| \leq diff(v,t)$.
Since $x_{e,t} < x_{e,t}^{MPC}$ and $x_{e,t} > x_{e,t}^{MPC}$ can both hold, the equality may not hold.

An $O(\log \log n)$ round algorithm: Proof

Defining $diff(v,t)$ to bound $|y_{v,t} - y_{v,t}^{MPC}|$

Let $x_{e,t}^{MPC}$ be the value of $x_{e,t}$ derived by the $O(\log \log n)$ algorithm.
Define $diff(v,t) = \sum_{e \in N(v)} |x_{e,t} - x_{e,t}^{MPC}|$.
As $y_{v,t} = \sum_{e \in N(v)} x_{e,t}$ and $y_{v,t}^{MPC} = \sum_{e \in N(v)} x_{e,t}^{MPC}$, we can see $|y_{v,t} - y_{v,t}^{MPC}| \leq diff(v,t)$.
Since $x_{e,t} < x_{e,t}^{MPC}$ and $x_{e,t} > x_{e,t}^{MPC}$ can both hold, the equality may not hold.

Bounding $diff(v,t)$ in base case

Let t^* be the first iteration of some phase, then $diff(v,t^*) = 0$. This is immediate by the way we set ground truths.

An $O(\log \log n)$ round algorithm: Proof

Defining $diff^{\text{local}}(v,t)$ to bound $|y_{v,t} - \tilde{y}_{v,t}|$

Let t^* be the first iteration of some phase, $N_{v,t}^{\text{local}}$ be the active neighbor in the sampled subgraph, $N_{v,t}^{\text{central}}$ be the active neighbor in the ground truth, $B_{v,t}^{\text{local}}$ the bad vertex in the neighbor of sampled subgraph, for vertex v in time t.

Define $diff_{v,t}^{\text{local}} = w_{t^*} \cdot |m \cdot |N_{v,t^*}^{\text{local}} \cap N_{v,t^*}^{\text{central}}| - |N_{v,t^*}^{\text{central}}||$
$+ \sum_{\hat{t}=t^*+1}^{t} \epsilon w_{\hat{t}} \cdot |m \cdot |N_{v,\hat{t}}^{\text{local}} \cap N_{v,\hat{t}}^{\text{central}}| - |N_{v,\hat{t}}^{\text{central}}||$
$+ \text{m w}_t \cdot |B_{v,t}^{\text{local}}|$
We have $|y_{v,t} - \tilde{y}_{v,t}| \leq diff_{v,t}^{\text{local}}$

An $O(\log\log n)$ round algorithm: Proof

Defining $diff^{\text{local}}(v,t)$ to bound $|y_{v,t} - \tilde{y}_{v,t}|$

Let t^* be the first iteration of some phase, $N_{v,t}^{\text{local}}$ be the active neighbor in the sampled subgraph, $N_{v,t}^{\text{central}}$ be the active neighbor in the ground truth, $B_{v,t}^{\text{local}}$ the bad vertex in the neighbor of sampled subgraph, for vertex v in time t.

Define $diff_{v,t}^{\text{local}} = w_{t^*} \cdot |m \cdot |N_{v,t^*}^{\text{local}} \cap N_{v,t^*}^{\text{central}}| - |N_{v,t^*}^{\text{central}}||$

$+ \sum_{\hat{t}=t^*+1}^{t} \epsilon w_{\hat{t}} \cdot |m \cdot |N_{v,\hat{t}}^{\text{local}} \cap N_{v,\hat{t}}^{\text{central}}| - |N_{v,\hat{t}}^{\text{central}}||$

$+$ m w$_t \cdot |B_{v,t}^{\text{local}}|$

We have $|y_{v,t} - \tilde{y}_{v,t}| \leq diff_{v,t}^{\text{local}}$

The first term is the initial error from random partition, second term sums the cumulated error from random partition, last term accounts for bad vertices.

An $O(\log \log n)$ round algorithm: Proof

Defining $diff^{\textbf{local}}(v,t)$ to bound $|y_{v,t} - \tilde{y}_{v,t}|$

Let t^* be the first iteration of some phase, $N^{\text{local}}_{v,t}$ be the active neighbor in the sampled subgraph, $N^{\text{central}}_{v,t}$ be the active neighbor in the ground truth, $B^{\text{local}}_{v,t}$ the bad vertex in the neighbor of sampled subgraph, for vertex v in time t.

Define $diff^{\text{local}}_{v,t} = w_{t^*} \cdot |m \cdot |N^{\text{local}}_{v,t^*} \cap N^{\text{central}}_{v,t^*}| - |N^{\text{central}}_{v,t^*}||$
$+ \sum_{\hat{t}=t^*+1}^{t} \epsilon w_{\hat{t}} \cdot |m \cdot |N^{\text{local}}_{v,\hat{t}} \cap N^{\text{central}}_{v,\hat{t}}| - |N^{\text{central}}_{v,\hat{t}}||$
$+ \text{m w}_t \cdot |B^{\text{local}}_{v,t}|$

We have $|y_{v,t} - \tilde{y}_{v,t}| \leq diff^{\text{local}}_{v,t}$

An $O(\log \log n)$ round algorithm: Proof

Defining $diff^{\text{local}}(v,t)$ to bound $|y_{v,t} - \tilde{y}_{v,t}|$

Let t^* be the first iteration of some phase, $N_{v,t}^{\text{local}}$ be the active neighbor in the sampled subgraph, $N_{v,t}^{\text{central}}$ be the active neighbor in the ground truth, $B_{v,t}^{\text{local}}$ the bad vertex in the neighbor of sampled subgraph, for vertex v in time t.

Define $diff_{v,t}^{\text{local}} = w_{t^*} \cdot |m \cdot |N_{v,t^*}^{\text{local}} \cap N_{v,t^*}^{\text{central}}| - |N_{v,t^*}^{\text{central}}||$
$+ \sum_{\hat{t}=t^*+1}^{t} \epsilon w_{\hat{t}} \cdot |m \cdot |N_{v,\hat{t}}^{\text{local}} \cap N_{v,\hat{t}}^{\text{central}}| - |N_{v,\hat{t}}^{\text{central}}||$
$+$ m $\mathrm{w}_t \cdot |B_{v,t}^{\text{local}}|$
We have $|y_{v,t} - \tilde{y}_{v,t}| \leq diff_{v,t}^{\text{local}}$

Specifically, for the second term, if there are no bad vertices, $\tilde{y}_{v,\hat{t}} - \tilde{y}_{v,\hat{t}-1} = \epsilon w_{\hat{t}} m \cdot |N_{v,\hat{t}}^{\text{local}} \cap N_{v,\hat{t}}^{\text{central}}|$ and $y_{v,\hat{t}} - y_{v,\hat{t}-1} = \epsilon w_{\hat{t}} |N_{v,\hat{t}}^{\text{central}}|$.

We have
$|y_{v,t} - \tilde{y}_{v,t}| \leq |y_{v,t^*} - \tilde{y}_{v,t^*}| + \sum_{\hat{t}=t^*+1}^{t} |(y_{v,\hat{t}} - y_{v,\hat{t}-1}) - (\tilde{y}_{v,\hat{t}} - \tilde{y}_{v,\hat{t}-1})|$,
which is just first and second term.

An $O(\log\log n)$ round algorithm: Proof

The following lemma establishes a good recurrence for error bounds.

Main Technical Lemma

Let v be an active vertex in iteration $t-1$. If we have

- $diff^{\text{local}}_{v,t-1} \leq \sigma$
- $diff_{v,t-1} \leq \sigma$

Then we have

- $diff^{\text{local}}_{v,t} \leq 5(\sigma + \epsilon m^{-0.2})$
- $diff_{v,t} \leq 5(\sigma + \epsilon m^{-0.2})$

An $O(\log \log n)$ round algorithm: Proof

Things that distort $\tilde{y}_v$ from y_v is the set of bad vertices, and the effect of random partitioning. It is similar for y_v^{MPC}, but here random partitioning is not an issue.

An $O(\log \log n)$ round algorithm: Proof

Things that distort $\tilde{y}_v$ from y_v is the set of bad vertices, and the effect of random partitioning. It is similar for y_v^{MPC}, but here random partitioning is not an issue.

For the bad vertices created **before** iteration t, their weight increases by $w_t - w_{t-1} = \frac{\epsilon}{1-\epsilon} w_{t-1} \leq 2\epsilon w_{t-1}$. The new error we can get is $2\epsilon\sigma$, since that's the proportion of error increasing.

An $O(\log \log n)$ round algorithm: Proof

Things that distort $\tilde{y}_v$ from y_v is the set of bad vertices, and the effect of random partitioning. It is similar for y_v^{MPC}, but here random partitioning is not an issue.

For the bad vertices created **before** iteration t, their weight increases by $w_t - w_{t-1} = \frac{\epsilon}{1-\epsilon} w_{t-1} \leq 2\epsilon w_{t-1}$. The new error we can get is $2\epsilon\sigma$, since that's the proportion of error increasing.

For the bad vertices created **in** iteration t, we have $n_{v,t-1}^{\text{local}} = |N_{v,t-1}^{\text{local}} \cap N_{v,t-1}^{\text{central}}|$ candidates of new bad vertices. They can be bad with probability σ/ϵ, so the expectation is $n_{v,t-1}^{\text{local}}\sigma/\epsilon$. We will prove a concentration around this expected value.

An $O(\log \log n)$ round algorithm: Proof

Note that the local neighbors are sampled from the actual neighbor with probability $\frac{1}{m}$. We use usual Chernoff to derive WHP result below.

If $\mu \geq m^{0.6}$, we have
$P(|X - \mu| \geq m^{-0.2}\mu)$
$\leq 2\exp(-m^{-0.4}\mu/3) \leq 2\exp(-m^{0.2})/3$

If $\mu \leq m^{0.6}$, we have
$P(|X - \mu| \geq m^{0.6})$
$=$ P(X - $\mu \geq m^{0.6}) \leq 2\exp(-m^{-0.4}\mu/3) \leq \exp(-m^{0.6})/3$

An $O(\log \log n)$ round algorithm: Proof

Note that the local neighbors are sampled from the actual neighbor with probability $\frac{1}{m}$. We use usual Chernoff to derive WHP result below.

If $\mu \geq m^{0.6}$, we have
$P(|X - \mu| \geq m^{-0.2}\mu)$
$\leq 2\exp(-m^{-0.4}\mu/3) \leq 2\exp(-m^{0.2})/3$

If $\mu \leq m^{0.6}$, we have
$P(|X - \mu| \geq m^{0.6})$
$=$ P(X - $\mu \geq m^{0.6}) \leq 2\exp(-m^{-0.4}\mu/3) \leq \exp(-m^{0.6})/3$

Using this, we can upper bound the number of local neighbors, and the number of bad local neighbors WHP.

With some calculation we have $2(\sigma + \epsilon m^{-0.2})$ errors.

An $O(\log\log n)$ round algorithm: Proof

Then we have the distortion from random partitioning, which similar to the previous argument, we can yield $\epsilon m^{-0.2}$ upper bounds.

So, the total error is $(1+2\epsilon)\sigma + 2(\sigma + \epsilon m^{-0.2}) + \epsilon m^{0.2} \leq 5(\sigma + \epsilon m^{-0.2})$. QED.

An $O(\log \log n)$ round algorithm: Proof

Then we have the distortion from random partitioning, which similar to the previous argument, we can yield $\epsilon m^{-0.2}$ upper bounds.

So, the total error is
$(1+2\epsilon)\sigma + 2(\sigma + \epsilon m^{-0.2}) + \epsilon m^{0.2} \leq 5(\sigma + \epsilon m^{-0.2})$. QED.

Lemma: Base case for induction

If t is the first iteration of a phase, we have $diff_{v,t}^{\text{local}} \leq m^{-0.2}$ WHP, and $diff_{v,t} = 0$ certainly.

First result is the Chernoff identical to the previous arguments, second result is immediate by definition.

An $O(\log \log n)$ round algorithm: Proof

Combining all previous results, we can obtain the following:

Lemma: Each phase has good values

If a phase consists of at most $I = (\log m)/(10 \log 10)$ iterations, then it holds $|y_{v,t} - \tilde{y}_{v,t}| \leq m^{-0.1}$ and $diff(v,t) \leq m^{-0.1}$ whp.

This error is smaller than ϵ (we assumed that $\Delta > \log^{9999} n$).

An $O(\log \log n)$ round algorithm: Proof

Combining all previous results, we can obtain the following:

Lemma: Each phase has good values

If a phase consists of at most $I = (\log m)/(10 \log 10)$ iterations, then it holds $|y_{v,t} - \tilde{y}_{v,t}| \leq m^{-0.1}$ and $diff(v,t) \leq m^{-0.1}$ whp.

This error is smaller than ϵ (we assumed that $\Delta > \log^{9999} n$).

Consider a vertex that should not be frozen but was. In the ground truth, it still has high enough $y_{v,t}$ which makes the situation OK.

An $O(\log \log n)$ round algorithm: Proof

Combining all previous results, we can obtain the following:

Lemma: Each phase has good values

If a phase consists of at most $I = (\log m)/(10 \log 10)$ iterations, then it holds $|y_{v,t} - \tilde{y}_{v,t}| \leq m^{-0.1}$ and $diff(v,t) \leq m^{-0.1}$ whp.

This error is smaller than ϵ (we assumed that $\Delta > \log^{9999} n$).

Consider a vertex that should not be frozen but was. In the ground truth, it still has high enough $y_{v,t}$ which makes the situation OK.

Consider a vertex that should be frozen but was not. Actually, there are not a lot of such vertices, and the number can be bounded. This bounds the number of vertices with $y_v > 1$ as well.

An $O(\log \log n)$ round algorithm: Proof

Combining all previous results, we can obtain the following:

Lemma: Each phase has good values

If a phase consists of at most $I = (\log m)/(10 \log 10)$ iterations, then it holds $|y_{v,t} - \tilde{y}_{v,t}| \leq m^{-0.1}$ and $diff(v,t) \leq m^{-0.1}$ whp.

This error is smaller than ϵ (we assumed that $\Delta > \log^{9999} n$).

Consider a vertex that should not be frozen but was. In the ground truth, it still has high enough $y_{v,t}$ which makes the situation OK.

Consider a vertex that should be frozen but was not. Actually, there are not a lot of such vertices, and the number can be bounded. This bounds the number of vertices with $y_v > 1$ as well.

There are a lot of technicalities omitted, but that is the high-level idea to prove that the algorithm yields $2 + \epsilon$ approx VC and maximum matching.