Course 2.18.1: Distributed Algorithms for Networks
Speed Up Simulation

Mikaël Rabie

1 Speed up 3-coloring - Slide 4

Let assume we have an algorithm $\mathcal{A}$ that produces a k-coloring of the path in T rounds. The goal is to create an algorithm $\mathcal{A}'$ that produces a k'-coloring of the path in one less round (with a well-chosen k').

We first consider an algorithm that works from edges: each edge has access to the identifiers of both endpoints. At each round, they communicate to get information of the neighboring edges (for example in one round, they learn the identifiers of the neighboring nodes of their own endpoints). Let consider an edge uv. After $T-1$ rounds, they have the knowledge of the $T-1$ identifiers on the left and on the right of the edge.

Let u_T (reps. v_T) be the node at distance T from u (resp. v), see Figure 1. To produce the output of u with $\mathcal{A}$, the edge uv is missing the identifier of u_T. We define S_L as the set of possible colors that $\mathcal{A}$ could produce depending on the identifier of u_T. This set can be computed by uv after $T-1$ rounds. Symmetrically, it computes S_R the set of possible colors that v could output depending on the identifier of v_T.

Algorithm $\mathcal{A}_1$ outputs the color $S_l \# S_R$ on edge uv. As we have two subsets of $[1,k]$, we get 4^k possible colors. Let's prove that we have indeed a coloring for edges:

We have the fact that $S_L \cap S_R = \emptyset$. Indeed, $c \in S_L \cap S_R$ implies that there exists some identifiers for u_T and v_T such that $\mathcal{A}$ produces c on both u and v, which contradicts the fact that $\mathcal{A}$ is a coloring algorithm.

Let uv and vw be two neighboring edges, with respective colors $S_L \# S_R$ and $S'_L \# S'_R$. We have the following properties:

- $S_L \cap S_R = \emptyset$ & $S'_L \cap S'_R = \emptyset$, thanks to the previous observation.

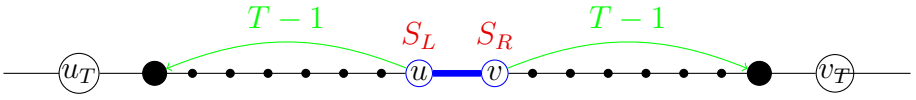

Figure 1: $T-1$ rounds on edge algorithm $\mathcal{A}'$

- $S'_L \cap S_R \neq \emptyset$, as it corresponds to the possible outputs that v can get with $\mathcal{A}$. In particular, given identifiers to the full path, the color of v produced by $\mathcal{A}$ must be both in S_L and S'_R.
- $S_L \# S_R \neq S'_L \# S'_R$. Let $c \in S'_L \cap S_R$. By the first property, c cannot be in S_L as it is in S_R. Hence, S_L and S'_L must be different.

From this, we deduce that we built an algorithm $\mathcal{A}_1$ that 4^k-colors edges in $T-1$ rounds. We do the same construction to build an algorithm $\mathcal{A}_2$ that 4^{4^k}-colors vertices in $T-1$ rounds.

By iterating l times this process, we get to an algorithm $\mathcal{A}_{2l}$ that $4^{4^{\cdot^{\cdot^{\cdot^k}}}}$-colors (with $2l$ 4 in the tower) nodes in $T-l$ rounds.

Let assume that we have an algorithm that 3-colors the path (i.e. $k=3$) in T rounds, with $T < \frac{\log_4^* n}{2}$. By choosing $l = T$, we create $\mathcal{A}_{2T}$, an algorithm that c-colors vertices in 0 rounds, with $c = 4^{4^{\cdot^{\cdot^{\cdot^3}}}} < n$. This is impossible in 0 round (even with access to some randomness).

We deduce from the following lower bound:

Theorem 1 ([3]) *An algorithm which colors the n-cycle with three colors requires at least $\frac{1}{2}(\log^* n - 3)$ communication rounds.*
The same bound holds also for randomized algorithms.

2 Automatic Round Reduction

2.1 Principle of Round Elimination - Slide 8

We consider the port-numbering model (i.e. no access to unique identifiers).

The goal of Round elimination is to generalize the previous technique. From a given algorithm $\mathcal{A}$ that solves some problem Π in T rounds, we want an automatic way to build a new problem Π' and some new algorithm $\mathcal{A}'$ in $T-1$ rounds. If we iterate this process T times, we should end up with a problem that can be solved in 0 round. If that problem cannot be solved trivially, we deduce that the initial problem could not be solved in T rounds, providing a lower bound.

2.2 Weak 3-labeling - Slide 9

When we build Σ_1, by default, we have all the non-empty subset of $Sigma_0$. A_1 must be pairs of sets such that for any choice for each set, we end up with a pair of P_0. However, $P_0 = \{[1,1],[2,2],[3,3]\}$ implies that each pairs of sets must be singletons (as if 1 is in a set, 1 must be the only element of the other set, and vice versa). As Σ_1 becomes simplified to $\{\{1\},\{2\},\{3\}\}$, P_1 are directly the same as P_0 by replacing elements by the singleton of that element.

2.3 Solving Π_1 - Slide 10-11

We perform a generalized version of the technique on paths: for each edge, we compute the set of outputs the corresponding vertex would have computed, looking at all possible configurations on the missing part of the graph (i.e. the nodes in the red rectangle for v and in the yellow rectangle for w).

Notice that there exists actually an equivalence, $\mathcal{A}_0$ solves Π_0 in T rounds if and only if $\mathcal{A}_1$ solves Π_1 in $T-1$ rounds.

2.4 Back to weak 3-labeling - Slide 12

We first simplify notations by identifying each singleton to the element itself. Note that P_1 is not directly P_0: we have switched which nodes are active and which are passive. We can easily check that the problem is not trivial: active nodes (of degree 2) must produce the same output everywhere (as we consider a deterministic algorithm), meaning that passive nodes will have the same color on each edge, which is not allowed.

Again, Σ_2 is the set of non-empty subsets of Σ_1. For A_2, we need to compute triplets of sets such any choice ensure that at least 2 different colors were chosen. A way to look at all possibilities is to order the elements from Σ_2 by decreasing size, and check lexicographically each triplets (discarding each prefix that can be completed, and ignoring triplets included in previous ones).

For example, here, we start with set $\{1,2,3\}$. If we have a second $\{1,2,3\}$, we cannot have a third set that matched, as we can take 3 times the same color. However, if for a second set we just take 2 colors, the last set can contain the 3rd color and we ensure that we cannot pick 3 times the same color. We could also consider the case $\{1,2,3\}$ and two singletons, but it would be included in a previous case (by adding the 3rd color to one of the singletons).

If we take $\{1,2\}$, taking again $\{1,2\}$ would force the last element to be $\{3\}$, that is a case included in the case where we add 3 to the first set, we do not need to consider it. However, if we add $\{2,3\}$ to the first set, we realise that with $\{1,3\}$ as a third set, any choice includes at least 2 colors.

To compute P_2, we take any pair of Σ_2 such as there is a possible choice to get a pair of A_1. This directly corresponds to choosing any pairs X, Y such as $X \cap Y \neq \emptyset$.

This new problem is trivial: if each node of degree 3 puts on its port 1 set $\{2,3\}$, port 2 set $\{1,3\}$ and port 1 set $\{1,2\}$ (that option is in A_2), there will always be a non-empty intersection between the two sets the degree 2 nodes are given.

We can deduce a 2-rounds algorithm: Black nodes put the 3 pairs, one on each of its edges in the first round. On the next round, White nodes choose in the two pairs it got a common color, and outputs it on both edges.

3 Sinkless Orientation

Sinkless orientation consists in orienting edges such that each node of degree at least 2 as an outgoing edge. If we are given a black and white graph, we orient O edges from Black to White nodes and I fro White to Black nodes. It implies that a Black node must have a O edge in its neighborhood, and a White node needs an I.

3.1 Sinkless Orientation on Paths - Slides 13

Let assume there is an algorithm in $T \leq (n-5)/4$ rounds. Let's take a path of length $(n-5)/2$. The middle node must have an outgoing edge, that forces the orientation along that direction in

the path. If we duplicate that path, and join the two copies in that direction, the node in the middle must have only ongoing edges, contradicting the Sinkless Orientation of the path.

3.2 Sinkless Orientation on Trees - Slides 14-17

Now only nodes of degree at least 3 must have an outgoing edge. We see that we get $\Pi_1 = \Pi_2 = \ldots = \Pi_k$ for any $k \geq 1$. One could think that it implies that there is a lower bound for any $T \leq D$, D being the diameter of the graph, which contradicts the $O(\log n)$ algorithm of slide 14.

The key is that the round reduction techniques relies on the fact that after k reduction, we have a regular (d, δ)-regular tree of depth k from some node, to be able to do one more reduction. However, if d and δ are at least 3, any tree of depth $\log_3 n$ must have at least a node of degree 2 or a leaf (otherwise, we would have more than n nodes in the graph). Hence, the round reduction technique only works for $T = o(\log n)$ if d or δ is at least 3.

4 Maximal Matching Lower Bound - Slide 18

The M edges form the Maximal Matching. Black nodes either have a M edge and they provide an edge O to the other neighbors to "prove" that they are matched. Otherwise, Black nodes need to have only P on its edges, that point to a matched White node. White nodes accept if they are matched (and all other edges can have any of P or O, i.e. the provide the pointer P to Black nodes who need it) or, if they are not matched, they only accept if all edges are O, that are proofs that all of their Black neighbors are matched.

Note that $[PO]$ means state P or O. This allows us to simplify the notation of P_0 that would have been $P_0 = [O^\Delta] \cup_{i<\Delta} [MP^iO^{\Delta-1-i}]$.

4.1 Parametrized Matching - Slide 19

We add a joker state X (the intuition being that X can end up being a matched or not matched edge). In [1], it was proved that if there exists an algorithm that solved $\Pi_\Delta(x, y)$ in T rounds, there exists an algorithm that solves $\Pi_\Delta(x+1, y+x)$ in $T-1$ rounds.

We can see that from $Pi_\Delta(0, 0)$, after k simplifications, we get $Pi_\Delta(k, k(k-1)/2)$. By assuming that the problem is not trivial if $x + y = o(\Delta)$, we get that the problem is not trivial if $k^2 = o(\Delta)$, i.e. $k = o(\sqrt{\Delta})$.

4.2 Weak $\sqrt{\Delta}$-matching - Slides 20-22

Suppose that we can solve Maximal Matching in $o(\Delta)$ rounds. The three steps to get to the $\Omega(\Delta)$ lower bound:

1. We solve weak "$\sqrt{\Delta}$-Matching" in $o(\sqrt{\Delta})$ rounds, by splitting each node in $\sqrt{\Delta}$ copies, each of degree $\sqrt{\Delta}$, and running the Maximal Matching algorithm on the new graph with bounded degree $\sqrt{\Delta}$.

2. A solution of weak $\sqrt{\Delta}$-matching gives directly a solution of $\Pi_\Delta(\sqrt{\Delta} - 1, 0)$ (by giving X to each matched edges). We deduce that we have a $o(\sqrt{\Delta})$ algorithm to solve weak $\sqrt{\Delta}$-matching.

3. By iterating the reduction of Π_Δ, we solve $\Pi_\Delta(\sqrt{\Delta}+o(\sqrt{\Delta}), o(\Delta))$ in 0 rounds, which is not a trivial problem.

Note that a better use of the round elimination technique for Maximal Matching has been used since then in [2] to provide an easier round reduction for a direct $\Omega(\Delta)$ lower bound.

References

[1] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, **Mikaël Rabie**, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 481–497. IEEE Computer Society, 2019.

[2] Sebastian Brandt and Dennis Olivetti. Truly tight-in-Δ bounds for bipartite maximal matching and variants. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 69–78. ACM, 2020.

[3] Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.