Course 2.18.1: Distributed Algorithms for Networks Complexity Classes

Mikaël Rabie

1 Path and Cycle Complexities

1.1 Transition Automata - Slides 3-4

We build an automata for each admissible neighborhood (i.e. output of left, node itself and right node). We put an edge $ABC \to BCD$ if a node with output B with output A on its left can have outputs C then D on the right. Sometimes, we can simplify this automata if the knowledge of the output on the left and/or the right is not necessary to know the possible next neighbor.

1.2 How to Decide the Complexity - Slide 6

- $O(1)$: If there is a self loop, it is sufficient to give the corresponding output on each node of the path.

  On the other hand, let have a problem with complexity $o(\log n)$. We have the following Theorem:

  Theorem 1 ([2]) *If a problem on paths can be computed in $o(\log^* n)$ rounds, there exists an algorithm with the same complexity stable under relative order.*

  Stable under relative order means that if we replace identifiers $\{id_u\}_{u\in V}$ with other ones $\{id'_u\}_{u\in V}$ such that, for all $u, v \in V^2$, $id_u < id_v \iff id'_u < id'_v$. In particular, let assume we have an algorithm stable under relative order with complexity $T = o(\log^* n)$. If we consider the path such that the identifiers are increasing by 1, three neighbors in the middle of the path have the same relative order of identifiers in their radius T neighborhood. Hence, they must have the same output O, implying that there is a self loop on the state OOO in the transition automata of the problem.

  We can prove that in any automata, either

  1. There exists a node v and $k > 0$ such as $\forall i \geq k$, there exists a path of length i from v to v.
  2. For all nodes v, either v cannot be reached from v, or there exists $k > 1$ such as any path of length $i > 0$ from v to v is such as k divides i.

- $\Omega(n)$: Let assume we are in case 2, and we have a $o(n)$ algorithm. Let x be the size of the output set, and n such that the complexity is less than $6n$. Let consider $x + 1$ nodes at distance at least $\frac{n}{3x}$ from each other. An output o must appear at least in two of those

segments. By the property 2, those two nodes must be at distance that is a multiple of k. However, the algorithm being too slow, those two nodes did not see each other before outputting. Hence, (by eventually adding one more node in between), this output cannot respect the automata, contradicting the $o(n)$ complexity.

- $\Theta(\log^* n)$: In case 1, we can build a $O(\log^* n)$ algorithm. First, we compute an MIS at distance k, i.e. a set of nodes such that any consecutive pair of nodes are at distance between $k+1$ and $2k$. To do so, we can see that an MIS is a set such any consecutive elements are at distance 2 or 3. If we compute an MIS among those nodes (considering two consecutive ones as neighbors), we get a set of elements at distance between $4 = 2^2$ and $9 = 3^2$. This computation takes 3 times the number of steps an MIS computation would take (as we need at most 3 rounds for two neighboring nodes to communicate), which is $3\alpha \log^* n$ (α being the constant of the $O(\log^* n)$ complexity to compute an MIS).

  If we do it i times, we get elements at distance between 2^i and 3^i, and it is computed in $(1+3+3^2+\ldots+3^{i-1})\alpha \log^* n$ rounds, which is $\frac{3^i-1}{3-1}\alpha \log^* n = O(3^i \log^* n)$. For $i = \log k$, we get elements at distance between k and $3^{\log k}$. After $3^{\log k}$ more rounds, we can add elements in segments of size at least $2k$ to make sure that all segments are now of size between k and $2k$. Note that all this process depends on k that is a constant from the automata, meaning that the complexity is $O(\log^* n)$ to compute this set.

  From this set, we put the output of the node v of the automata on the corresponding nodes in the path. The property ensures that in at most $2k$ rounds, we can figure out how to complete the output in between the nodes.

2 Decidability on Paths with Inputs

2.1 Complexity Separation on Paths with inputs - Slides 9-11

Given a Turing Machine and a tape space k, does the Machine stop on that space? This problem is PSPACE-hard. Given a Turing Machine and some integer $k > 0$, we create a LCL problem with inputs such that its complexity is $\Theta(n)$ if and only if the Machine does not halt on a tape of length k.

The idea of the problem is that it checks whether we have simulated the Turing Machine with the input on the path. The simulation is explained on Slide 10. The path starts whether with state a or b. The output has two possibilities:

- Copy the closest symbol a or b that can be found when we go left in the path. It is indeed LCL, as either you have input a or b and you must have the same output, or you must have the same output that your left neighbor's.

- Have either state E to say that there is an error somewhere in the simulation on your left, or being part of a proof of an error. Locally, to check that your output is accepted, you must have on your left symbol E, or check the validity of the proof of error you are in. In particular, a node with input a or b cannot have output E, hence a proof must be found. Slide 11 gives an example of proof, where the state of the 6th symbol on the tape has not be copied in two consecutive steps of the simulation. E^2 means error of symbol copy, the first 0 means that we should expect a 0 on the right as a 0 was on the left. the numbers from 0 to k are here to ensure that we compare symbols at the same place on the tape.

The deduction of the complexity comes as follows:

- If the Turing Machine does halt on space k, any simulation must be of constant length K (that only depends on the Machine and k that are fixed parameters). Hence, if after the input symbol a or b, the path is of length at least $K+1$, an error must occur. More precisely, form any point on the path, looking K nodes to left must either lead to a or b, or to an error. Hence, the problem can be solved in $K = O(1)$ rounds.
- If the Turing Machine does not halt, we can make an input of any length such that no error can be found on the left. In that input, nodes must find the first node to know if they have to output a or b, implying a $\Theta(n)$ complexity.

2.2 Complexity Separation on Trees - Slides 12-14

In that scenario, we no longer consider inputs. However, we see that we can encode inputs on trees (Slide 13 shows how to encode binary numbers). By taking a path, and adding to each node an hanging tree corresponding to an input, we are going back to the case of paths with inputs. Hence, deciding the complexity of problems on trees without inputs is at least as hard as deciding complexities on paths with inputs.

3 Toroidal Grids

Toroidal Grids are graphs where each node has degree 4, and knows which neighbor is in which direction (North, South, East or West). The graph wraps up in each direction consistently, i.e. if you keep going North, you come back to the beginning (same if you go West).

3.1 Speed Up Algorithm - Slide 16

If we have a problem Π and an algorithm $\mathcal{A}$ that solves it on Toroidal Grids of dimensions $n \times n$ with sublinear complexity $T(n) = o(\log n)$, we can create another algorithm that solves the same problem with complexity $O(\log^* n)$. The construction of the new algorithm is as follows:

- Take k such as $T(k) < k/4 - 4$.
- Compute a maximal independent set I of $G^{k/2}$ in time $O(\log^* n)$ (i.e. a maximal independent set at distance $k/2$). This can be done with complexity $O(\log^* n + f(\Delta))$, with $\Delta = 4^{k/2} = O(1)$, hence with complexity $O(\log^* n)$.
- Each node u chooses as anchor $i_u \in I$ the closest node from the independent set (breaking ties arbitrarily). They compute new identifiers $id'_u = (x_{i_u} - x_u, y_{i_u} - y_u) \in [-k/2, k/2]^2$. The new identifiers are no longer unique, but they form a distance-$k/2$ coloring (i.e. a coloring where no pair of nodes at distance at most $k/2$ have the same color).

  Indeed, if we have $u \neq v$ such that $id'_u = (x_{i_u} - x_u, y_{i_u} - y_u) = id'_v = (x_{i_v} - x_v, y_{i_v} - y_v)$, we must have $i_u \neq i_v$ (otherwise, as they are at the same coordinate distance from i_u, we would have $u = v$). As $i_u \neq i_v$ and those two nodes are from the independent set, they must be at distance at least $k/2$ from each other. Moreover, the distance between u and v must be the same as the distance from each other than the one between i_u and i_v (as their relative coordinates are the same).

- Now, the nodes simulate $\mathcal{A}$ with the new identifiers, presuming that the dimension is k instead of n. Nodes cannot detect that this is false in $T(k)$ rounds, as they see less than $(k+1)^2$ nodes. Hence, in time $T(k)$, they solve Π.
- The complexity of the new algorithm is $O(\log^* n)$.

3.2 Turing Machine Simulation - Slide 18

We are given a Turing Machine that starts on an empty input, and use half a tape (i.e. the head cannot go left from the starting point). From it, we make the following LCL problem on toroidal grids:

- Each node either computes a 3-coloring of the grid. In that case, it outputs 1, 2 or 3 and its neighbors must also compute a color that is different from its own.

  Otherwise, nodes solve the following problem:

- Each node is either an anchor, or has an orientation (N, NE, E...). The orientation must be towards an anchor. For example, a NE node must have NE nodes on its top and on its right. Its left neighbor can either be N or NE, and is bottom neighbor must either be E or NE. A node in state S must either have an anchor or a node in state S on top.
- If you have output E, either your left neighbor is an anchor, and you must also store q_0 the starting state of the Turing Machine, and the blank symbol $\_$ (i.e. the head of the TM is on you, in state q_0). Otherwise, you must just store $\_$.
- If you have output NE, have a tape symbol, and are not holding the head of the TM, your up neighbor must have the same tape symbol. It can (and must) have the Turing Machine head (with state q' and tape symbol b) only if your left (resp. right) neighbor also has the head (with symbol q and tape symbol a), and the transition of the TM is $\delta(q,a) = (q', b, R)$ (resp. $\delta(q,a) = (q', b, L)$).

  The exception is when you have a final state q_f on your left, right or bottom. In that case, you can take state q_f. Moreover, if you have state q_f, you no longer care about orientation consistency and state copy on top, left and right.

With these rules, we ensure that from any anchor, a simulation of the Turing Machine will occur on its top right part. We have two possibilities:

1. The Turing Machine halts. We know that only a constant space (that depends on the Machine) will be used to simulate the Machine.

   A $O(\log^* n)$ algorithm to solve this problem consists in computing a maximal independent set for the anchors. This set will be at distance large enough to make sure to have room to put the TM simulation. All nodes not in the simulation selects its closest anchor and orients itself accordingly.

2. The Turing Machine does not halt. No simulation can be put, as the grid is finite. Nodes must compute a 3-coloring of the grid, which has complexity $O(n)$ [1].

The small issue here is that, if every node outputs an orientation, like SW, the LCL problem will be satisfied. To prevent that, we add the fact that nodes must be 2-colored in their direction, i.e. they must also output 0 or 1 and their next neighbor in their direction must be different (top right for SW nodes, for example). In the case of anchors computed at constant distance, this 2-coloring can be computed by knowing our distance to the anchor.

References

[1] Sebastian Brandt, Juho Hirvonen, Janne H Korhonen, Tuomo Lempiäinen, Patric RJ Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemysław Uznański. Lcl problems on grids. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 101–110, 2017.

[2] Moni Naor and Larry Stockmeyer. What can be computed locally? In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 184–193, 1993.