Principles of Distributed Computing 03/04, 2020

Exercise 3

Lecturer: Mohsen Ghaffari

1 Color Reduction in Vertex-Coloring

Exercise

(1a) Design a single-round algorithm that transforms any given k-coloring of a graph with maximum degree Δ into a k'-coloring for $k' = k - \lfloor \frac{k}{(\Delta+2)} \rfloor$, assuming $k' \geq \Delta + 1$.

We assume $k \geq \Delta + 2$ (as otherwise we cannot in general reduce the number of colors). We put the colors of the given k-coloring into $\lfloor \frac{k}{\Delta+2} \rfloor$ buckets, each of size $\Delta + 2$ except for the last one which may have size between $\Delta + 2$ to $2\Delta + 3$. Within each bucket we can in one round reduce the number of colors by 1, using the method of Lemma 1.23. Since this can be done for all buckets in parallel, in total the number of colors can be reduced by $\lfloor \frac{k}{\Delta+2} \rfloor$ in 1 round.

(1b) Use repetitions of this single-round algorithm, in combination with the $O(\log^* n)$-round $O(\Delta^2 \log \Delta)$-vertex-coloring that we saw in class, to obtain an $O(\Delta \log \Delta + \log^* n)$-round $(\Delta + 1)$-coloring algorithm.

First, we run the $O(\log^* n)$ algorithm that gives us a $(C\Delta^2 \log \Delta)$-coloring for some constant C. Then we repeat the algorithm from (a) until $k \leq \Delta+1$. To bound the number of necessary rounds we need until the number of used colors drops to $\Delta+1$, we first bound how the number of used colors drops in each iteration. We have

$$\lfloor \frac{k}{\Delta + 2} \rfloor \geq \frac{1}{2} \frac{k}{(\Delta + 2)} \geq \frac{k}{6\Delta},$$

where the first inequality holds for any $k \geq \Delta+2$ and for the second inequality we used $\Delta \geq 1$. Hence, the number of used colors drops in each iteration by factor of at least $1 - \frac{1}{6\Delta}$, unless it is already $\Delta + 1$. But this means that after $18\Delta \log(C\Delta)$ iterations the number of colors either drops to $\Delta + 1$ or to

$$C\Delta^2 \log \Delta \cdot \left(1 - \frac{1}{6\Delta}\right)^{18\Delta \log(C\Delta)} \leq$$

$$\leq C\Delta^2 \log \Delta \cdot \mathrm{e}^{-18\Delta \log(C\Delta)/(6\Delta)} = C\Delta^2 \log \Delta (C\Delta)^{-3} < \Delta + 1$$

where we used that for any x we have $1 + x \leq \mathrm{e}^x$. Hence, we achieve a coloring with $\Delta + 1$ colors in $O(\Delta \log \Delta)$ steps, as needed.

2 SuperImposed Codes

Here, we use the concept of cover free families to obtain an encoding that allows us to recover information after superimposition. That is, we will be able to decode even if k of the codewords are *superimposed* and we only have the resulting bit-wise OR.

Exercise

(2a) Concretely, we want a function $Enc : \{0,1\}^{\log n} \to \{0,1\}^{\log m}$ — that encodes n possibilities using $\log m$-bit strings — such that the following property is satisfied: $\forall S \neq S' \subseteq \{1, ..., n\}$ such that $|S| \leq k$ and $|S'| \leq k$, we have that $\vee_{i \in S} Enc(i) \neq \vee_{i \in S'} Enc(i)$. Here $\vee$ denotes the bit-wise OR operation. Present such an encoding function, with a small m, that depends on n and k.

The solution to this exercise is obtained by bending the description of the task into the terms of Lemma 1.19 from the lecture notes. We use a k-cover-free family for the encryption, where the bitstring of length $\log m$ is interpreted as being a set where the i^{th} bit is 1 if element i is in the set. Note that the bitwise OR is the same as the union of the corresponding sets. Since $S \neq S'$, there is an element j that is contained in exactly one of S and S' . W.l.o.g., assume that $j \in S$ (and therefore $j \notin S'$). Due to using a k-cover-free family in our construction, and since $|S'| \leq k$, we know that $Enc(j) \not\subseteq \bigcup_{i \in S'} Enc(i)$, which implies $\bigcup_{i \in S} Enc(i) \neq \bigcup_{i \in S'} Enc(i)$ (or, in terms of bit strings, $\vee_{i \in S} Enc(i) \neq \vee_{i \in S'} Enc(i)$). Applying Lemma 1.19, we get that there exists such a k-cover-free family with ground set of size $\log m = O(k^2 \log n)$, hence $m = 2^{O(k^2 \log n)}$.

3 Yet Another Coloring

Here, we see yet another deterministic method for computing a $(\Delta+1)$-coloring in $O(\Delta \log \Delta + \log^* n)$ rounds. First, using what we saw in the class, we compute an $O(\Delta^2 \log \Delta)$-coloring ϕ_{old} in $O(\log^* n)$ rounds. What remains is to transform this into a $(\Delta + 1)$-coloring, in $O(\Delta \log \Delta)$ additional rounds.

Exercise The current $O(\Delta^2 \log \Delta)$-coloring ϕ_{old} can be written using $C \log \Delta$ bits, assuming a sufficiently large constant C. This bit complexity will be the parameter of our recursion. Partition G into two vertex-disjoint subgraphs G_0 and G_1, based on the most significant bit in the color ϕ_{old}. Notice that each of G_0 and G_1 inherits a coloring with $C \log \Delta - 1$ bits. Solve the $\Delta + 1$ coloring problem in each of these independently and recursively. Then, we need to merge these colors into a $\Delta + 1$ coloring for the whole graph.

(A) Explain an $O(\Delta)$-round algorithm, as well its correctness proof, that once the independent $(\Delta+1)$-colorings of G_0 and G_1 are finished, updates only the colors of G_1 vertices to ensure that the overall coloring is a proper $(\Delta + 1)$-coloring of $G = G_0 \cup G_1$.

In each of $\Delta + 1 = O(\Delta)$ rounds, we choose a distinct one of the $\Delta + 1$ colors used to color G_1, and let each vertex in G_1 with this color pick a new color from the color set used for G_0 such that no neighbor in G has the same color. Since there are $\Delta + 1$ colors to pick from and at most Δ neighbors per node, such a new color always exists. Note that no two neighboring nodes pick a new color in the same round. Since the above algorithm ensures that we always have a proper coloring in the subgraph induced by the nodes from G_0 and the already recolored nodes from G_1, we have a proper $(\Delta + 1)$-coloring after recoloring all nodes from G_1, i.e., after $\Delta + 1$ rounds.

(B) Provide a recursive time-complexity analysis that proves that overall, the recursive method takes $O(\Delta \log \Delta)$ rounds.

After $C \log \Delta - 1$ iterations of splitting each of the vertex-disjoint subgraphs obtained in the previous iteration into two smaller subgraphs, we are left with subgraphs of size 1 (or 0). Note that this splitting process does not require any communication rounds; so far, each node v only has to know its own subgraph, which is simply v itself. After trivially finding a proper $(\Delta+1)$-coloring in each size-1 subgraph, we recursively glue the split subgraphs back together

and transform the $(\Delta+1)$-colorings of any two joined subgraphs into a $(\Delta+1)$-coloring of the obtained combined subgraph, using the process described in (A). Since the depth of our recursion is $C\log\Delta-1$ and the process from (A) runs in $O(\Delta)$ rounds, we obtain a total round complexity of $(C\log\Delta-1)\cdot O(\Delta)=O(\Delta\log\Delta)$.