CIS 677

Homework 4

Daniel Sparber

Problem 1

Let $G(V, E)$ be an arbitrary undirected graph.

Without loss of generality, let $V = [1..n]$, let $\pi \in S_n$ be a random permutation.

We now construct the directed acyclic graph $G'(V, E')$:

$\forall \{u, v\} \in E$: if $\pi(u) < \pi(v)$ then $(u, v) \in E'$ otherwise $(v, u) \in E'$

G' has the same vertices as G and a topological order determined by π. I.e. the topological order is equivalent to sorting the nodes $v \in V$ by $\pi(v)$ in ascending order. For every edge in the original graph, G' contains an directed edge with the same nodes, that respects the topological order induced by π.

By construction G' is a DAG. We now run the deterministic poly-time algorithm on G'.

- Case I: G does not contain simple path of length k

  Since G' does not add any additional edges, the longest path of G' is at most as long as the longest path in G. Therefore, G' contains no path of length k. Thus, the deterministic poly-time algorithm always rejects.

- Case II: G contains a simple path of length at lest k

  Let $p = p_1, p_2, .., p_k$ be a path of length k (not necessarily unique).

  G' also contains p if $\pi(p_1) > \pi(p_2) > \cdots > \pi(k)$. I.e. if the topological order induced by the random permutation π does not destroy the path p. Note: the reverse order also preserves the path.

  Any permutation $\pi \in S_n$ is called good, if it preserves the path.

  We now analyze the probability of picking a good permutation at random.

  First, for every good permutation, there are $k! - 2$ bad permutations. This follows from the fact that taking a good permutation and only permuting the nodes in p results in $k! - 1$ bad permutations (there is $k!$ permutations which only permute the nodes in p, the original and reverse permutations are good).

  Thus, the probability of picking a good permutation is at least $\frac{1}{k!-2}$.

  Since k is bounded by $\frac{\log n}{\log \log n}$:

$$k! - 2 < k! \leq k^k \leq \left(\frac{\log n}{\log\log n}\right)^{\frac{\log n}{\log\log n}} \leq (\log n)^{\frac{\log n}{\log\log n}} = \left(2^{\log\log n}\right)^{\frac{\log n}{\log\log n}} = 2^{\log n} = n$$

This implies a probability of at least $\frac{1}{n}$ for picking a good permutation.

Consider the new algorithm:

Repeat $2n$ times:

1. Pick random $\pi \in S_n$
2. Construct G'
3. Run deterministic algorithm:
   - If accepts: output YES

The algorithm runs in poly-time since every step only takes poly-time and the number of repetitions is also polynomial.

The algorithm only fails, if every round fails. By the previous analysis the probability of failure for one round less than $1 - 1/n$.

$\Pr[\textit{success}] = 1 - \Pr[\textit{failure}] \geq 1 - (1 - 1/n)^{2n} \geq 1 - e^{-2} \geq 1 - 1/4 = 3/4$ □

Problem 2

Consider the following high level algorithm:

- If $A = B$: return $N_A = N_B$
- compare(A, B):
  - If $|A| = 1$: compare N_A and N_B and return result
  - Let A_1 be the first $\lceil |A|/2 \rceil$ bits of the bitstring A, A_2 the second half. Same for B_1 and B_2
  - If $A_1 \neq B_1$: return compare(A_1, B_1)
  - Else: return compare(A_2, B_2)

The algorithm is interactive and compares A and B in a recursive way. Since the algorithm always halfs the length of the bitstring, the recursion depth is $\mathcal{O}(\log n)$. Additionally, the recursion does not branch, since the recursion is only called on one of the two halfs. Therefore, there are at most $\mathcal{O}(\log n)$ rounds.

First, consider the algorithm on a single machine. If $A = B$ the algorithm returns the correct result. Otherwise, without loss of generality, let $N_A > N_B$. The base case with just one bit to compare returns the correct result. In the other cases, we check where A and B differ. If they differ in the first half, i.e. $N_{A_1} > N_{B_1}$, $N_A > N_B$ since the first half contains the higher order bits. If the first half is the same, they must differ in the second half, i.e. $N_{A_2} > N_{B_2}$. $N_A > N_B$ follows directly. Thus, the algorithm is correct on a single machine.

Now consider the distributed case. We show that the algorithm still works if A and B are not on the same machine. Alice and Bob both run the algorithm on their own and cooperate to perform the (in)equality tests. For the base case they deterministically compare one bit. Thus, the algorithm succeeds if every (in)equality test succeeds.

We use the following algorithm to compare two bitstrings A and B:

- Alice chooses a random prime number $p \in [2..n^{\mathbf{5}}]$
- Alice computes $N_A \bmod p$ and sends the the result and p to Bob
- Bob computes $N_B \bmod p$ and sends one bit to Alice indicating whether $N_A \bmod p = N_B \bmod p$

The equality check needs to send $\mathcal{O}(\log n)$ bits. By performing an analysis analog to the lecture, we can bound the failure probability by $1/n^3$:

$$\Pr[\textit{failure}] = \frac{\textit{Prime divisors}}{\textit{Primes in } [2..n^5]} \leq \frac{n}{\frac{n^5}{\log n^5}} \leq \frac{n}{n^4} = \frac{1}{n^3}$$

In total there are at most $\log n$ equality tests. By union bound, the probability of any equality test failing is bounded by $\frac{\log n}{n^3} \leq \frac{1}{n^2}$. Thus, the correct result is returned with probability at least $1 - 1/n^2$.

Furthermore, the communication is bounded by $\mathcal{O}(\log n)$ rounds times $\mathcal{O}(\log n)$ bits per rounds. Therefore, the total communication is bound by $\mathcal{O}(\log^2 n)$.

Problem 3

Part (a)

Let Φ be arbitrary. Let $C = 34$.

Assume 1% of all possible assignments for Φ are satisfying.

Assume there exists a set S of $C + 1 = 35$ disjoint clauses in Φ.

Each clause in S has three variables. There are $2^3 = 8$ possible assignments for these three variables. Only one of these assignments does not satisfy the clause. I.e. 7 assignments are satisfying for the given clause.

There are $3(C + 1)$ distinct variables in S and therefore $2^{3(C+1)} = 8^{(C+1)}$ possible assignments for these variables. Out of those assignments, $7^{(C+1)}$ are satisfying. For the remaining $k = n - 3(C + 1)$ variables, at most 2^k out of 2^k assignments are satisfying.

This yields the following percentage p of satisfying assignments:

$p = \frac{7^{(C+1)}2^k}{8^{(C+1)}2^k} = \left(\frac{7}{8}\right)^{(C+1)} = \left(\frac{7}{8}\right)^{35} < 0.01 = 1\%$

This contradicts is a contradiction. Thus, a constant C exists. C is at most 34.

Part (b)

Let Φ be arbitrary. Let S be the largest set of disjoint clauses in Φ. Let V be the set of all variables that occur in the clauses of S.

We know S has at most C clauses. Therefore, V has at most $3C$ variables.

Claim: Every clause in Φ has at least one variable in V.

Proof: Assume there exists a clause $X \in \Phi$ with no variables in V. Therefore, X is disjoint to all clauses in S. Thus, S is not the largest set of disjoint clauses ($S \cup \{X\}$ is larger). This is a contradiction.

Part (c)

We continue using the set V form part (b). Let $k = |V|$.

Algorithm

For all possible assignments A for the variables in V:

- Modify Φ as following:
    1. Replace all variables of Φ which are part of in V with their corresponding assignment.
    2. Remove all clauses that now contain a 1, since they are trivially satisfied.

3. Remove all zeros, since they do not change the logic value of a clause.

- Run a 2-SAT solver on the modified Φ.

Each clause in Φ has at least one variable in V. Therefore, after modifying Φ as described above, there are at most 2 variables remaining in every clause. The number of clauses is at most as high as the number of clauses in the original Φ. The modified Φ is always therefore always a valid 2-SAT formula.

There are at most 2^{3C} possible assignments for variables in V. Thus, the algorithm solves at most 2^{3C} 2-SAT instances.

Problem 4

We define an arbitrary node $c \in V$ as center node.

$E' = \{\{c, v\} \mid v \in V\}$

E' contains exactly n edges by construction, thus fulfilling the requirement of the task.

The diameter of $G'(V, E \cup E')$ is bounded by 2 since, for any $u, v \in V$, $u \to c \to v$ is a valid path in G' of length 2.

G is connected, therefore $m \geq n - 1$. Furthermore, $|E'| \in \mathcal{O}(n) \subseteq \mathcal{O}(m)$ and $|E| = m$. The total number of edges $|E \cup E'|$ is therefore in $\mathcal{O}(m)$.

Corollary form lecture: In an undirected graph with diameter D, the cover time is bound by $\mathcal{O}(mD \log n)$.

By invoking the corollary we get a cover time of $\mathcal{O}(m2 \log n) = \mathcal{O}(m \log n)$.

Problem 5

High level outline:

- Use some computation, such that the expected value is equal to the total amount of annual sales.
- Use Chernoff bounds to analyse the probability, use the formulas for relative error with $\delta = 0.01 = 1\%$.
- Do the computation $c \in \log(N+M))^{\mathcal{O}(1)}$ times with some random parameters and send the concatenation of all results to the headquarter. By repeating c times the probability of getting the correct result is at least 99%.

If the result of each computation each server does is of order $(\log(N+M))^{\mathcal{O}(1)}$, then the total size of the single message each server sends is also bound by $(\log(N+M))^{\mathcal{O}(1)}$.

Computation:

Idea 1:

Each server picks some entries at random, computes the sum of sales and sends the result to the headquarter. The headquarter sums up all received values.

We adjust the way elements are picked at random such that on expectation no value is missing or duplicated amongst all messages.

To have no duplicates we could use the shared source of randomness to assign stores to servers. If the server has the requested entry, it includes it into the sum.

The servers can also send the count of tuples that are part in their sum to the headquarter. The count can be sent with $\log M$ bits. The server can sum up the counts as well to check if all stores were included.

Analysis

Idea 1:

Let X be a random variable. X is consists of independent Bernoulli random variables has the following property:

$\mathbb{E}[X] = \mu =$ total amount of annual sales.

$1 \cdot N \leq \mu \leq M \cdot N$

We now use Chernoff bounds ($\delta = 0.01$):

$\Pr[X > (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{\mu\delta^2}{4}} \leq e^{-\frac{N}{40{,}000}}$

$\Pr[X < (1-\delta)\mathbb{E}[X]] \leq e^{-\frac{\mu\delta^2}{2}} \leq e^{-\frac{N}{20{,}000}}$

By repeating $c = 6 \cdot 40,000$ times, the probability of failure can be reduced to:

$\Pr[X > (1+\delta)\mu] \leq e^{-\frac{c \cdot N}{40{,}000}} = e^{-6N} \leq e^{-6} < 0.005$

$\Pr[X < (1-\delta)\mu] \leq e^{-\frac{c \cdot N}{20{,}000}} = e^{-12N} \leq e^{-6} < 0.005$

Therefore, the success probability is greater than 99%.