

CME 305: Discrete Mathematics and Algorithms

Instructor: Professor Amin Saberi (saberi@stanford.edu)

HW#3 – Solutions

1. Kleinberg and Tardos Problem 8.28

One way to solve this is via reduction from IS by placing a vertex in the middle of each edge and connecting all pairs of such vertices with an edge. Another way is to reduce from 3-SAT similarly to the reduction to IS but by introducing an extra vertex for each literal which appears along with its negation in the formula. Below is yet another reduction from IS .

Solution: Given a set S we can easily verify via breadth first search whether each vertex has a path at least three to any other vertex. It is also easy to verify that $|S| = k$, and thus $SIS \in NP$. We now show that $IS \leq_p SIS$ to conclude that SIS is NP-complete.

Construct a graph G' by making k copies of the original graph G_1, G_2, \dots, G_k . For each v in G consider corresponding vertices v_1, v_2, \dots, v_k with v_i in G_i . Add all possible edges between these vertices producing a complete subgraph in G' for each v in G . This mapping f is polynomial-time computable in the input (G, k) .

First, we show that $X \in IS \Rightarrow f(X) \in SIS$. Let S be an IS of size k in G . For each $v \in S$ construct a set S' by adding a copy of v from each G_i . Clearly S' has k elements. The shortest path between $u_j, v_i \in S'$ is of length at least 3 (it takes one edge to travel from v_j to the subgraph G_i where u_i is and since S is an IS there is a path of length at least two between v_i and u_i).

Next we show that $f(X) \in SIS \Rightarrow X \in S$. Let S' be an SIS of size k in G' . We can ensure that each element is in a separate graph copy G_i and S' is still an SIS . Now, construct S by selecting a v in G corresponding to each v_i in G_i . Since there is a path of three or more between any pair of vertices in S' and we contract the path by one by moving all vertices to G .

2. Let A be a $n \times n$ matrix, B a $n \times n$ matrix and C a $n \times n$ matrix. We want to design an algorithm that checks whether $AB = C$ without calculating the product AB . Provide a randomized algorithm that accomplishes this in $O(n^2)$ time with high probability.

Solution: Pick $x \in \{0, 1\}^n$ such that $x_i = 1$ with probability $\frac{1}{2}$.

1. compute $y = Bx, z = Ay, w = Cx$
2. if $z \neq w$ return false
3. return true

First note that it takes $O(n^2)$ time to compute all the above matrix vector multiplications. Also note that we avoid roundoff error. Computing w and y involves no multiplication and there is no error in the computation of z assuming that AB can be computed exactly.

Now, if $AB - C = 0$ the algorithm is always correct. So, assume $AB - C \neq 0$. We compute the probability that the algorithm returns true,

$$P(z = w) = P(ABx = Cx) = P((AB - C)x = 0).$$

Let $D = AB - C$ and let d_i be the i^{th} row of D . We have that that

$$\begin{aligned} P(Dx = 0) &\leq P(d_i^T x = 0) \\ &= P\left(\sum_j d_{ij}x_j = 0\right) \\ &\leq P\left(\sum_j d_{ij}x_j = 0 \mid x_2, \dots, x_n\right) \\ &= \frac{1}{2} \end{aligned}$$

where the last step comes from the fact that we are only free to pick x_1 which has probability of $\frac{1}{2}$ regardless which value we pick. Thus the probability we make a mistake is $P(z = w) \leq \frac{1}{2}$. Repeating the algorithm some constant k times (say 10) we bound the probability of error by $\frac{1}{2^k} \approx 0.001$.

3. A tournament is a complete directed graph i.e. a directed graph which has exactly one edge between each pair of vertices. A Hamiltonian path is a path that traverses each vertex exactly once. A random tournament, is a tournament in which the direction of all edges is selected independently and uniformly at random.

- (a) What is the expected value of the number Hamiltonian paths in a random tournament?

Solution: Let Γ be the set of all permutations of vertices in G . For each permutation $\sigma \in \Gamma$ the probability there is a path from the first vertex of σ to the last vertex of σ is $P(\sigma \text{ is a path}) = \frac{1}{2^{n-1}}$ (i.e. each edge has a probability $\frac{1}{2}$ of being in the direction of the path and we have $n - 1$ path edges).

Let I_σ be the indicator variable of whether σ is a path in G . Let N be the random variable counting the number of such paths. We can compute its expected value as

$$E[N] = \sum_{\sigma \in \Gamma} I_\sigma = \sum_{\sigma \in \Gamma} P(\sigma \text{ is a path}) = \frac{n!}{2^{n-1}}$$

since $|\Gamma| = n!$, the number of permutations of the vertices.

- (b) Use part (a) to show that for every n , there is a tournament with n players and at least

$$\frac{n!}{2^{n-1}}$$

Hamiltonian paths.

Solution: For any random variable X , it is clear from the definition of expected value that the probability that $X \geq E[X]$ is nonzero. Thus $N \geq \frac{n!}{2^{n-1}}$ with positive probability. Therefore, there exists a tournament with at least this many Hamiltonian paths.

4. In class we proved that the presorted Longest Processing Time (LPT) minimum makespan approximation algorithm gives a $3/2$ -approximation, but noted that this is not the best bound possible.

Show that LPT is a $4/3$ -approximation. Give an example to show this bound is tight.

Solution: Let L^* be the optimal makespan and L be the makespan produced by the LPT algorithm. Assume the sorted job are sorted in descending order. From the class notes we have that

$$L \leq L^* + t_n$$

where t_n is the last (smallest) job. Consider the case that $t_n \leq L^*/3$. Then,

$$L \leq L^* + L^*/3 = 4/3L^*.$$

For the case in which $t_n > L^*/3$ we can see that there are at most two jobs per machine, for a total of $2m$ jobs. Three jobs which would yield a load strictly greater than $3L^*/3 = L^*$. We now show that for $n \leq 2m$, LPT produces the optimal assignment. Consider the properties of the optimal assignment L^* for jobs $j = 1, \dots, n$ with processing times $t_1 > t_2 \dots > t_n$.

- (a) Each job $j \leq m$ is assigned its own machine. Otherwise, any two such jobs on a machine would produce a load greater than the makespan of LPT. Thus, w.l.o.g. we can assign each such job j to machine $i = j$ in the optimal assignment.
- (b) By the same reasoning, the next $k \leq m$ jobs have to be placed on machines $m, m-1, \dots, m-k+1$. Otherwise, we again would exceed the makespan of LPT.

Finally, the assignment of the last k jobs has to follow the LPT placement since any interchange of the last job with any other increases the makespan. Thus LPT produces the optimal makespan for $n \leq 2m$.

To produce a tight bound let $n = 2m + 1$ and define two jobs with processing times $t_i = 4m - i + 1$ for $i = 1, 2, \dots, 2m + 1$. After the first $2m$ jobs have been placed, LPT results in an assignment with a load $L_i = 6m + 1$ on each machine i . Placing job n anywhere we obtain a makespan $L = 8m + 1$.

Now consider the assignment which pairs job loads $4m - i + 1$ and $2m + i + 2$ for $i = 1, 2, \dots, m - 1$. And place loads $2m, 2m + 1$ and $2m + 2$ on machine m . The load on all machines is $6m + 3$. The ratio $(8m + 1)/(6m + 3) \rightarrow 4/3$ as m grows large. Therefore, this assignment asymptotically achieves the proved bound showing that it is optimal.

5. Kleinberg and Tardos Problem 11.10

Solution: (a) Let T be an independent set in G and let S be the output of the greedy algorithm. Note that the algorithm terminates when all vertices are either deleted or in set S . Thus for $v \in T$, either $v \in S$ or v was deleted which means it has some neighbor v' of maximal weight moved into set S . This vertex must satisfy $w(v') \geq w(v)$ and we have the desired result.

(b) Let W^* be the weight of the optimal independent set T . Let W be the weight of set S output by the algorithm. For each $v \in T$, define $v' = v$ if $v \in S$ or as its neighbor

in S otherwise. Such a neighbor is guaranteed by part (a) and satisfies $w(v') \geq w(v)$.

$$\begin{aligned}
 W^* &= \sum_{v \in T} w(v) \\
 &\leq \sum_{v', v \in T} w(v') \\
 &\leq 4 \sum_{u \in S} w(u) \\
 &= 4W
 \end{aligned}$$

where we've used the fact that each $u \in S$ can have at most 4 neighbors in T .

6. Extra Credit: Solve Problem 3 from the midterm.

We first prove two claims.

- (a) For each vertex v , its neighborhood $N(v)$ is a bipartite graph (thus 2-colorable).

Proof: Assume $N(v)$ is not bipartite. Then it has an odd cycle with each vertex in the cycle connected to v . It is easy to see that such a graph is not 3-colorable which contradicts the main assumption of this problem.

- (b) Any graph with maximum degree Δ is $\Delta + 1$ -colorable.

Proof: The coloring is produced with a greedy strategy. For each uncolored vertex we assign distinct colors to $v \cup N(v)$ which has a maximum cardinality $\Delta + 1$. This strategy will never run into a conflict since at each step we have at most $\Delta + 1$ vertices to color and that many total colors to choose from.

We now state the algorithm to produce a $O(\sqrt{n})$ -coloring for any 3-colorable graph G .

1) Partition the vertices V of the graph in the set $S = \{v \in V \mid d(v) \geq \sqrt{n}\}$. For each $v \in S$ use colors c_{v1} for v and c_{v2}, c_{v3} for its neighborhood $N(v)$ (by claim (a)).

2) Color the remaining vertices with exactly \sqrt{n} colors. This can be done by a greedy procedure from claim (b) since the maximum degree of the vertices in $V - S$ is $\sqrt{n} - 1$.

Clearly this algorithm runs in polynomial time. There are at most \sqrt{n} vertices in S (since there are n total). And we use three colors at each step when coloring S and their neighbors. The remaining vertices are shown to be \sqrt{n} -colorable. Thus the graph can be colored with $O(\sqrt{n})$ colors.