15-859FF: Coping with Intractability **CMU, Fall 2019**
HW #1 (out Wednesday Sep 11, 2019) **Due:** Thursday Sep 26, 2019

You may discuss all problems on this HW with others, but groups of size at most 3, please. Submission details (and corrections) will appear on Piazza, please check it regularly.

Exercises

Exercises are for fun and edification, please do not submit. **But do solve them, we may need ideas from there later in the course!**

1. **(Bounded-Depth Search)** Consider the algorithm for vertex cover mentioned in lecture:

   (a) drop all isolated vertices.

   (b) if there is a vertex u of degree 1, it has a single neighbor v, so add v to the vertex cover, and recurse on $G - \{u, v\}$ with parameter $k - 1$.

   (c) else all degrees are ≥ 2, so pick some vertex, and recurse on both $(G - \{u\}, k - 1)$, and $(G - N_G(u), k - |N_G(u)|)$.

   Here $N_G(u)$ is the set of neighbors of u. And always, when we drop a set of vertices, we drop all edges incident to those vertices. Show this algorithm correctly finds an optimal vertex cover (e.g., why is step (b) valid). Show that the runtime of this algorithm is $T(n) \leq O(\phi^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

2. **(Searching or Deciding?)** Show that the following are equivalent for the Vertex Cover problem (i.e., the existence of one implies the existence of the others).

   (a) An algorithm that answers the "decision" version: "given (G, k), does there exist a vertex cover in G of size k?" in time $f(k)\, n^{O(1)}$ for some (computable) function f.

   (b) An algorithm for the "value" version of Vertex Cover "Given G, what is $vc(G)$, the size of the smallest vertex cover of G" in time $g(vc(G))\, n^{O(1)}$, for some (computable) function g.

   (c) An algorithm for the "search" version of the problem: "Given G, output an vertex cover of G with smallest size" in time $h(vc(G))\, n^{O(1)}$, for some (computable) function h.

3. **(Reduce, Reduce...)** The exponential time hypothesis (ETH) says: *There exists an absolute constant $\delta > 0$ such that 3SAT does not have an algorithm that runs in time $2^{\delta n}$*, where n is the number of variables in the 3SAT instance.

   Consider the classical reduction from 3SAT to Clique (451 notes, Section 7). This reduction takes a 3SAT instance φ with n variables and $m \leq O(n^3)$ clauses, and produces a graph G with $N \leq 8m$ vertices, and $M = \Theta(N^2)$ edges, such that φ is satisfiable if and only if G has a clique of size $K := m$.

   Assuming the ETH, show that Clique does not have algorithms that run time $2^{o(|V(G)|^{1/3})}$ time. Similarly, show that Clique does not have algorithms that run time $2^{o(|E(G)|^{1/6})}$ time.

4. **(Hit the Set!)** Give a $O(d^k \text{poly}(n))$ algorithm for the d-Hitting Set, the hitting set problem on d-hypergraphs. Recall that a d-hypergraph is a set system $(V, \mathcal{F})$ with n elements

V, and $\mathcal{F}$ is a collection of subsets of V each with size at most d. A set $C \subseteq V$ is a *hitting set* if it hits all sets in $\mathcal{F}$, i.e., $C \cap S \neq \varnothing$ for all $S \in \mathcal{F}$.

Moreover, use the Sunflower lemma to show a kernel of size $d!k^d\text{poly}(d) = c_d k^d$ for d-Hitting Set, that can be constructed in polynomial time $\text{poly}(|V| + |\mathcal{F}|)$. Observe that this immediately implies a $f(c_d k^d) + \text{poly}(|V| + |\mathcal{F}|)$-time algorithm for d-Hitting Set.

5. **(Explicit Bias)** Suppose a coin has bias p, i.e., it comes up heads with probability p. Show that the expected number of flips before we see a heads is $1/p$. Use Markov's inequality to show that if we flip it $(2/p)\log_2 n$ times, the probability that we see all tails is at most $1/n$. Use a direct calculation to show that even with $1/p \ln n$ flips, the probability that we see all tails is at most $1/n$. (You may use the fact that $1 + x \leq e^x$ for all x.)

6. **(Better-than-Bruce Force)** Graph 3-colorability is one of the classic NP-hard problems and thus is unlikely to admit efficient worst-case algorithms. But this does not mean that we can't improve upon the naive algorithm of trying out all possible 3-colorings, which takes $\approx 3^n$ time on an n-vertex graph.

   (a) Give an algorithm with running time $2^n \cdot \text{poly}(n)$ that, when given an n-vertex graph G as input, determines if G has a valid 3-coloring, and if so, outputs one. You may assume that G is connected.

   (b) Give an algorithm with running time $\left(\binom{n}{n/3} \cdot n^3\right) \cdot \text{poly}(n)$ that, when given an n-vertex graph G as input, determines if G has a valid 3-coloring, and if so, outputs one. (One can show that $\binom{n}{n/3} < (1.9)^n$ for large n, so this is an improvement over the first part.) Hint: Determining if a graph is 2-colorable is easy.

Problems

Please solve problem 6, and any 4 of the other 5 problems. The Cygan et al book provides hints to its problems. Please try to solve these problems yourself before looking at these hints!

1. Give an FPT algorithm that takes (G, k) and outputs the *number* of (inclusion-wise) *minimal* vertex covers of G that have size at most k. (So your output is an integer between 0 and $\binom{n}{k}$. A vertex cover is *minimal* if dropping any vertex from it makes it no longer be a vertex cover.)

2. (2.6)

3. (2.9)

4. (3.14)

5. (13.28)

6. Give an $(3/2)^n\text{poly}(n)$-time algorithm for 3Coloring as follows: the input is a graph G, and we want to color each vertex using colors in $\{r, g, b\}$.

   Repeat the following steps at most $(3/2)^n \log n$ times:

   (a) For each vertex u, choose a set $L_u \subseteq \{r, g, b\}$ of size exactly 2, uniformly at random.

(b) Try to find a coloring of the vertices of G where each vertex u is colored using one of the two colors in L_u.

If any of these repetitions output a coloring in step (b), output it. If not, output "No: G is not 3-colorable".

(Hint: recall that the 2SAT problem can be solved in polynomial time.)

Show that the randomized algorithm above is correct with high probability. I.e., it has the following property: (i) if G is not-3-colorable, the algorithm always says "No", and (ii) if G is 3-colorable, then the algorithm finds this coloring with probability at least $1 - 1/n$.