Advanced Algorithms - Problem Set Solutions

Louis Holley

May 10, 2017

Problem 1.

To show a language is in P, we need to show that there is a polynomial-time algorithm that decides it. We can simulate a non-deterministic Turing machine in exponential time using a deterministic Turing machine. Hence, if the subset sum problem is in NP, we know unary subset sum (the same problem but with exponentially larger input size) is in P, since we can solve it by first converting the input to binary representation (this takes polynomial-time) and consequently shrinking the input size to log N. Then, we can simulate the non-deterministic Turing machine running on this input in time that is exponential in its input size, which is polynomial for log N.

Problem 2.

2.1

Determining if a graph is 2-colourable is equivalent to determining whether or not the graph is bipartite. It is possible to test whether a graph is bipartite in linear time using depth first search. The main idea is to assign to each vertex the colour that differs from the colour of its parent in the depth first search tree, assigning colours in a preorder traversal of the depth first search tree. This will necessarily provide a 2-colouring of the spanning tree consisting of the edges connecting vertices to their parents.

2.2

- If we can determine wheter a k-colouring for a given graph G exists or not in polynomial time, the colouring graph problem can be solved by assigning k from 1 to $|V|$ and applying at most $|V|$ instances of the decision problem. That means the graph colouring problem can also be solved in polynomial time.
- In contrast, if the graph-colouring problem can be solved and the minimum number n_c of colours needed can be found in polynomial time, the decision problem can also be solved by checking if $k \geq n_c$ in constant time.

We have proved that the decision problem is solvable in polynomial time if and only if the graph-colouring problem is solvable in polynomial time.

2.3

A graph G is 3-colourable if and only if the graph G' is k-colourable, where G' is the disjoint union of G and K_{k-3} with all possible edges between the two parts. In **2.2** we showed that deciding whether a graph is k-colourable is as hard as the graph-colouring problem itself. Thus, if 3-COLOUR is NP-Hard, then k-COLOUR is NP-Hard, and since deciding is as hard as colouring, the decision problem from part 2 is NP-Hard.

Problem 3.

To prove that 0-1 Integer Programming is NP-Complete, we must show

- 0-1 Integer Programming is in NP
- 0-1 Integer Programming is NP-Hard

IP is in NP because the integer solution can be used as a witness and can be verified in polynomial time. We now prove that IP is NP-Hard by reduction from SAT. A SAT instance is described by a set of Boolean variables and clauses. We reduce it to an instance of 0-1 Integer Programming with the same number of variables. In addition, for each variable v_i we have the constraint $0 \leq v_i \leq 1$. For each clause we have a constraint that corresponds to it, e.g. for the clause $(v_1 \vee \bar{v_3} \vee v_7)$ in the SAT instance, we have the constraint

$$v_1 + (1 - v_3) + v_7 \geq 1$$

Clearly, this reduction can be done in polynomial time. Moreover, it is easy to verify that if a given SAT instance has a satisfying assignment then the corresponding IP instance has an integer solution and vice versa.

Problem 4.

We first show that longest-simple-cycle is in NP. An instance x is an undirected graph and an integer k. Let a certificate y be a sequence of at least k vertices from the graph. The verifier goes through the vertices in the certificate one-by-one, marking each as it goes to check that there are no repeats, and verifying that there is an edge between each consecutive node in the certificate. Finally, the verifier checks that the cycle is completed by an edge from the last vertex in the certificate back to the first vertex. If and only if these checks are satisfied, the verified outputs "yes". This verifier takes linear time in the number of vertices; since the verifier is correct and takes polynomial time, the problem is in NP.

To show the problem is NP-Hard, we reduce HAMILTONIAN-CYCLE to it. Given a graph G that is an instance of HAMILTONIAN-CYCLE, define an instance $< G, k >$ of longest-simple-cycle containing the same graph $G = (V, E)$ and $k = |V|$, the number of vertices in G.

If a graph G contains a Hamiltonian cycle, then this cycle is a simple cycle with $|V|$ vertices, so the correct decision on the instance of longest-simple-cycle is "yes". If there is no Hamiltonian cycle, then there is no longest-simple-cycle with $|V|$ or more vertices, so the correct decision is "no".

Since the reduction is correct and takes polynomial-time, longest-simple-cycle is NP-Hard. Since it is also in NP, it is NP-Complete.

Problem 5.

In order to prove that half 3-CNF is NP-Hard, we consider a reduction from a known NP-Complete problem to half 3-CNF. We reduce from 3-SAT to half 3-CNF.

Consider the following reduction. First, we need to convert an instance of 3-SAT to half 3-CNF. Our approach is to create a ϕ' which contains 4 times as many clauses as ϕ. Suppose ϕ contains m clauses; when creating ϕ', first we take all clauses from ϕ. Next, we create m clauses of the form:

$$(p \vee \bar{p} \vee q)$$

Clearly, these clauses are always true, regardless of the individual truth assignments for p and q. Next, we create $2m$ clauses of the form:

$$(p \vee q \vee r)$$

These clauses are always true or always false. Therefore, we have created a boolean formula ϕ' which contains all the clauses in ϕ and m clauses which are always true and $2m$ clauses which are either all true or all false.

We need to show that there exists a "yes" instance of half 3-SAT if and only if there exists a "yes" instance of half 3-CNF.

- Assume that there exists a truth assignment which causes ϕ to be true. Then, the m clauses which correspond to ϕ in ϕ' are true and there are m clauses which are always true. Thus, simply let p and r be false, and there exists a truth assignment which satisfies half 3-CNF, where half the clauses are true and half are false.

- Assume that there exists a truth assignment which causes half 3-CNF to be satisfied, or a truth assignment that causes half the clauses in ϕ' to be true and half false. But m clauses in ϕ' are always true, which means that the $2m$ clauses cannot be true if half 3-CNF is satisfied. Thus, the $2m$ clauses must be false, which means that ϕ is true, so 3-SAT is also satisfied.

We have shown that a "yes" instance of 3-SAT provides a "yes" instance of half 3-CNF and vice versa, which concludes the proof.

Problem 6.

Consider a bipartite graph with left part L and right part R such that L has 5 vertices of degrees (5,5,5,5,5) and R has 11 vertices of degrees (5,4,4,3,2,2,1,1,1,1,1) - it is easy to draw. Clearly there exists a vertex-cover of size 5 (the left vertices). The idea is to show that the proposed algorithm chooses all the vertices on the right part, resulting in the approximation ratio of $\frac{11}{5} > 2$. After choosing the first vertex in R, the degrees of L decrease to (4,4,4,4,4). After the third, they decrease to (3,3,2,2,2). After the fourth, they decrease to (2,2,2,2,1). After the fifth, they decrease to (2,2,1,1,1). After the sixth, they decrease to (1,1,1,1,1). The algorithm still has to choose 5 more vertices, so regardless of which it selects from those left, 11 vertices in total are selected.

Problem 7.

Every tree has at least two leaves, meaning that there is always an edge which is adjacent to a leaf. The algorithm iterates through each node of the tree, selecting every non-leaf, until the tree has either 0 or 1 edges left. When the tree consists of an isolated edge, pick either vertex. All leaves can be identified and trimmed in linear time using depth-first search.

Problem 8.

Suppose v is an our cycle and we are about to add vertex u because its distance to v is smaller than the distance from any other vertex not on the cycle to any vertex on the cycle. Let w be the vertex that comes after v on the cycle. Since the metric TSP satisfies the triangle inequality, $c(w,u) \leq c(w,v) + c(u,v)$. The net increase in cost by adding vertex u to the path is

$$c(u,v) + c(u,w) - c(v,w) \leq c(u,v) + c(w,v) + c(u,v) - c(v,w) = 2c(v,u)$$

Let x_i denote the distance from v_i to v_{i+1}, where v_i is the i^{th} vertex in the cycle and v_{n+1} is the first vertex. Then the total cost of the algorithm is $2 \cdot \sum_{i=1}^{n} x_i$. Note that Prim's MST algorithm, starting from the same initial vertex v_1, would have added vertices to an MST at the same order $v_2, v_3, \cdots, v_n$ and at cost $\sum_{i=1}^{n} x_i$. The cost of a minimum spanning tree is bounded above by the cost of an optimal tour. This implies that the cost of our approximation algorithm is at most twice the cost of an optimal tour.

Problem 9.

$\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$. The biggest variance with values $\in \{0, 1\}$ occurs when $\mathbf{P}[X = 0] = \frac{1}{2}$ and $\mathbf{P}[X = 1] = \frac{1}{2}$. By simple calculation:

$$\mathbf{E}[X^2] = 0^2 \times \frac{1}{2} + 1^2 \times \frac{1}{2} = \frac{1}{2}$$

$$\mathbf{E}[X]^2 = (0 \times \frac{1}{2} + 1 \times \frac{1}{2})^2 = \frac{1}{4}$$

$$\mathbf{E}[X^2] - \mathbf{E}[X]^2 = \frac{1}{4}$$

Since this value is for the distribution under which the variance of X is largest, we conclude that for any discrete random variable X that takes on values in the range $\{0, 1\}$, $\mathbf{Var}[X] \leq \frac{1}{4}$.

Problem 10.

We know that for all $s \in S, X'(s) \leq X(s)$. Let A be the set of points in S such that $X'(s) \geq t$. Let B be the set of points such that $X(s) \geq t$. B contains A as a subset, so $\mathbf{P}[B] \geq \mathbf{P}[A]$.

Problem 11.

X is the random variable denoting the number of monochromatic copies of K_4. The probability that a certain 4-subset forms a monochromatic K_4 is 2×2^{-6} (2 for two different colours):

$$\mathbf{E}[X] = \binom{n}{4} \times 2 \times 2^{-6} = \binom{n}{4} 2^{-5}$$

(Here, the $\binom{n}{4}$ comes from choosing 4 vertices from n vertices).

For the algorithm, colour each edge uniformly and independently. Let p = $\mathbf{P}[X \leq \binom{n}{4}2^{-5}]$. Then we have

$$\binom{n}{4} 2^{-5} = \mathbf{E}[X]$$

$$= \sum_{i \leq \binom{n}{4}2^{-5}} i\mathbf{P}[X = i] + \sum_{i \geq \binom{n}{4}2^{-5}+1} i\mathbf{P}[X = i]$$

$$\geq p + (1 - p)(\binom{n}{4} 2^{-5} + 1)$$

which implies that

$$\frac{1}{p} \leq \binom{n}{4} 2^{-5}$$

Thus, the expected number of samples is at most $\binom{n}{4}2^{-5}$. Testing to see if $X \leq \binom{n}{4}2^{-5}$ can be done in $O(n^4)$ time, so the algorithm is polynomial.

Problem 12.

12.1

The number of edges across the cut is a random variable $Q = \sum_{\{i,j\}\in E} \mathbf{I}[Q_i \neq Q_j]$, where $\mathbf{I}$ denotes the indicator function on binary events and for each $i \in V$, Q_i is a Bernoulli random variable with parameter $\frac{1}{2}$. Thus,

$$\mathbf{E}[Q] = \mathbf{E}[\sum_{\{i,j\}\in E} \mathbf{I}[Q_i \neq Q_j]] = \sum_{\{i,j\}\in E} \mathbf{E}[\mathbf{I}[Q_i \neq Q_j]] = \sum_{\{i,j\}\in E} \mathbf{P}[Q_i \neq Q_j] = \frac{|E|}{2}$$

Since the combined weight of all edges in the graph is an obvious upper bound on the weight of any cut, this shows that the expected weight of the cut produced by the algorithm is at least half of the weight of the maximum cut.

12.2

For question 2, I believe it is the case that if you were to increase or decrease the value of p, the sets would become uneven in size and hence more edges would be between vertices in either set A or set B rather than between them, and hence the weight of the cut would be decreased, and the approximation ratio would become worse than 2. Not sure how to prove this formally but still thinking about it.

12.3

No clue on this one.

Problem 13.

Since we are told that the any k of the m vectors are linearly independent modulo 2, and our operation for computing X_i is taking one of these and XORing it with a random vector, the resulting output is still a random vector. K-wise independence requires

$$\mathbf{P}[\cap_{i\in I}(X_i = x_i)] = \Pi_{i\in I} P[X_i = x_i]$$

and because each X_i is a random vector, the knowledge of any one of $X_i = x_i$ does not change the probability of, for instance, $X_j = x_j$.

Problem 14.

Essentially here we are multiplying x by $a \bmod 2^u$ and then bit shifting right $u - m$ times, leaving the highest order m bits as the hash code. To understand the intuition behind this algorithm, notice that if $ax \bmod 2^u$ and $ay \bmod 2^u$ have the same highest order m bits, then $a(x - y) \bmod 2^u$ has either all 1s or all 0s as its highest order m bits (depending on which is larger, $ax \bmod 2^u$ or $ay \bmod 2^u$ respectively). Assume that the least significant set bit of $x - y$ appears at position $u - c$. We need to prove the probability of a collision for each possible placement of the least set significant bit:

- The case where $c > m$. Since a is a random odd integer and odd integers have inverses in the ring Z_{2^u}, it follows that $a(x - y) \bmod 2^u$ will be uniformly distributed among u-bit integers with the least significant set bit at position $u - c$. The probability that these bits are all 0s or all 1s is therefore at most $\frac{2}{2^m} = \frac{2}{M}$.
- The case where $c < m$. If $c < m$, then higher order m bits of $a(x - y) \bmod 2^u$ contain both 0s and 1s, so it is certain that $h(x) \neq h(y)$.
- The case where $c = m$. Finally, if $c = m$ then bit $u - m$ of $a(x - y) \bmod 2^u$ is 1 and $h_a(x) = h_a(y)$ if and only if bits $u - 1, \cdots, u - m + 1$ are also 1, which happens with probability $\frac{1}{2^{m-1}} = \frac{2}{M}$.

We have shown that for any position of the least significant set bit of $x - y$, at most the probability of collision is $\frac{2}{M}$ with this construction.

Problem 15.

- The running time of the algorithm is $O(n)$. For each of the two sets S_1 and S_2, we run n hashing operations and increment counters n times. Finally, we make n comparisons.
- If S_1 and S_2 are identical, then clearly the i^{th} counters for both tables will match for all i, regardless of the choice of hash function. So, if S_1 and S_2 are identical, the algorithm is correct with probability 1.
- Suppose S_1 and S_2 are not identical, and furthermore S_1 and S_2 are disjoint (we can assume they are disjoint without loss of generality since we can remove elements common to S_1 and S_2 in the analysis). Fix some $x \in S_1$ and note from our assumptions that $x \notin S_2$. For each $y \in S_2$, we know $y \neq x$ and we have by the property of universal hash functions that $P_h[h(x) = h(y)] \leq \frac{1}{cn}$. Taking a union bound over all $y \in S_2$, we have

$$\mathbf{P}_h[\exists y \in S_2 : h(x) = h(y)] \leq \frac{1}{c}$$

Hence, with probability $1 - \frac{1}{c}$, we have that $h(x) \neq h(y)$ for all $y \in S_2$. In that case, the $h(x)th$ counter for S_2 is 0, whereas that for S_1 is at least 1, so the algorithm outputs "no". Thus, if S_1 and S_2 are not identical, the algorithm outputs "no" with probability at least $1 - \frac{1}{c}$.

Problem 16.

For every matrix A there exists a non singular matrix P such that $PAP^{-1} = J$ where J has Jordan normal form[1]. Now using $tr(ABC) = tr(CAB) = tr(BCA)$ (trace is invariant under cyclic permutations[2]), we obtain:

$$tr(A) = tr(P^{-1}JP) = tr(PP^{-1}J) = tr(J) = \sum_i \lambda_i$$

where λ_i are the eigenvalues of A.

Problem 17.

$$Px = \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} P_{11}x_1 + P_{12}x_2 + \cdots + P_{1n}x_n \\ P_{21}x_1 + P_{22}x_2 + \cdots + P_{2n}x_n \\ \vdots \\ P_{n1}x_1 + P_{n2}x_2 + \cdots + P_{nn}x_n \end{bmatrix}$$

$$\sum_i (Px)_i = x_1(P_{11}+P_{21}+\cdots+P_{n1})+x_2(P_{12}+P_{22}+\cdots+P_{n2})+\cdots+x_n(P_{1n}+P_{2n}+\cdots+P_{nn})$$

$$\sum_i (Px)_i = \sum_i x_i = 1$$

Problem 18.

To find out what the i^{th} column of P is, we can use the i^{th} basis vector e_i (basis vectors are probability vectors since there is only one nonzero entry which is 1). Since $P \cdot e_i$ is exactly the i^{th} column of P, and we know that Px is a probability vector for any probability vector x, we conclude that every i^{th} column of P is a probability vector, and as such P is a stochastic matrix.

[1] `https://en.wikipedia.org/wiki/Jordan_normal_form`
[2] `https://en.wikipedia.org/wiki/Trace_(linear_algebra)`

Problem 19.

Adding edges to a graph may either increase or decrease the mixing time of a random walk depending on the particular situation. Adding an edge can shorten the distance from x to y thereby decreasing the mixing time, or the edge could increase the probability of a random walk going to some far off portion of the graph, thereby increasing the mixing time. Counter examples in each direction:

- For a counterexample in decrease direction, consider the "lollipop graph", which consists of a clique on $\frac{n}{2}$ vertices with a "tail" of length $\frac{n}{2}$ (edges) attached (so the total number of vertices is n). Take G to be the lollipop graph, and G' to be the complete graph on n vertices. Then $\Theta(n \log n)$ is the mixing time for G', while $\Theta(n^3)$ is the mixing time for G. You can find proofs for the bounds here[3], but it is easy to intuit from the shape of the graph.
- For a counterexample in the increase direction, take G to be the line graph on n vertices, and G' to be the lollipop graph. Then $\Theta(n^3)$ is the mixing time for G' and $\Theta(n^2)$ is the mixing time for G.

[3] http://people.eecs.berkeley.edu/~sinclair/cs174/sol9.pdf

Problem 20.

Let G be a d-regular bipartite graph such that half of the vectors V are in set A and the other half are in set B, and there are no edges between vectors in the same set. Take x such that:

- $x_i = 1$ if $V_i \in A$, and
- $x_i = -1$ if $V_i \in B$

From the Rayleigh Quotient, we know that

$$x^T L x = \sum_{(u,v)\in E} \frac{(x_u - x_v)^2}{d}$$

We can substitute this into the Courant-Fisher formula ($\lambda_n = \max_{x \in R^n \setminus \{0\}} \frac{x^T L x}{x^T x}$) as follows:

$$\lambda_n = \frac{\sum_{(u,v)\in E} \frac{(x_u - x_v)^2}{d}}{x^T x}$$

Since for every edge $(u, v) \in E$, u and v are in different sets, $(x_u - x_v)^2 = 4$. We can replace the summation since we know the number of edges in a d-regular graph is $|E| = \frac{|V|}{2} \cdot d$, giving

$$\lambda_n = \frac{\frac{|V|}{2} \cdot d \cdot \frac{4}{d}}{x^T x}$$

$$\lambda_n = \frac{2 \cdot |V|}{x^T x}$$

Since $x^T x = |V|$, we get

$$\lambda_n = 2$$

as required. Now for the other direction. Let $\lambda_n = 2$ and we'll see what we can find out about the graph. Again substituting the Rayleigh Quotient into the Courant-Fisher formula gives:

$$\lambda_n = \frac{\sum_{(u,v)\in E} \frac{(x_u - x_v)^2}{d}}{x^T x} = 2$$

We can split the nodes into two sets A and B arbitrarily without assuming that the graph is bipartite, so let's do that. Regardless of how we assign nodes to set A or set B, $x^T x = |V|$ and $|E| = \frac{|V|}{2} \cdot d$. Denote each edge between nodes in set A and each edge between nodes in set B as $E_A B$. For these nodes, the value of $x_u - x_v$ will be 0. This makes the whole term 0, so we can just ignore these from the summation. Assume there are k such nodes, so expanding the rest of it out, we sum over $(u, v) \in E \setminus E_{AB}$, and get

$$\lambda_n = \frac{\sum_{(u,v)\in E\setminus E_{AB}} \frac{(x_u - x_v)^2}{d}}{x^T x} = 2$$

$$\frac{(\frac{|V|\cdot d}{2} - k)\cdot \frac{4}{d} + 0 \cdot k}{|V|} = 2$$

$$\frac{(\frac{|V|\cdot d}{2} - k)\cdot \frac{4}{d}}{|V|} = 2$$

$$(\frac{|V|\cdot d}{2} - k)\cdot \frac{4}{d} = 2|V|$$

$$\frac{|V|\cdot d}{2} - k = \frac{2|V|\cdot d}{4}$$

$$\frac{|V|\cdot d}{2} - k = \frac{|V|\cdot d}{2}$$

$$k = 0$$

So if $\lambda_n = 2$, there are no vertices in set A or B that connect to other vertices from the same set - which means that the graph is bipartite.

We have thus shown that a graph is bipartite if and only if $\lambda_n = 2$.