1. *4 points, 2 for each part*

Understanding the Floyd-Warshall algorithm

Do Exercise 25.2-7 on page 700 (page 635 in 2/e). To simplify the notation please omit the implicit and unnecessary (k) superscripts in the Floyd-Warshall algorithm, i.e., use the one described in Exercise 25.2-4, which is equivalent to the one presented in class. And describe your algorithm in terms of computing ϕ_{ij} rather than $\phi_{ij}^{(k)}$.

Solution:

$$\phi_{ij}^{(k)} = \begin{cases} \phi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ k & \text{otherwise} \end{cases}$$

$$\phi_{ij}^{(0)} = \begin{cases} \phi_{ij}^{0} = 0 & \text{if } w_{ij} < \infty \\ \text{NIL} & \text{otherwise} \end{cases}$$

To compute $\phi_{ij}^{(k)}$, replace line 7 of the Floyd-Warshall algorithm on page 695 (modified to get rid of the (k) superscripts, which are not needed) with

if $d_{ik} + d_{kj} < d_{ij}$ **then**
    $d_{ij} = d_{ik} + d_{kj}$
    $\phi_{ij} = k$
endif

The recursive algorithm for computing the actual path from i to j is

PATH(i, j) **returns** the list of edges in the shortest path from i to j **is**
    ▷ It is assumed that a path exists
    if $\phi_{ij} = 0$ **then return** an empty list **endif**
    $L =$ PATH(i, ϕ_{ij})
    $L = L +$ PATH(ϕ_{ij}, j) ▷ the operator + denotes append
    return L
end PATH

2. *14 points: 8 for part (a), 4 for (b), and 2 for (c)*

Understanding NP-completeness proofs and conditions under which polynomial-time algorithms exist

The *min-ones satisfiability problem (M1-Sat)* is defined as follows.

Given a formula F in conjunctive normal form and an integer k, does there exist a satisfying assignment A for F such that the number of true variables in A is $\leq k$?

a. Prove that M1-Sat is NP-complete. Be sure to include and clearly mark all parts of the proof, even those that may seem trivial.

b. Let p be the number of variables that have at least one *positive* occurrence – a variable x has a positive occurrence if some clause contains x as a literal (as opposed to $\overline{x}$). Give a $O(|F|)$ algorithm for solving M1-Sat when p is constant, where $|F|$ is the total number of literals in F, the usual measure of the size of a formula.

c. Determine the fastest growth rate $f(n)$ for which you can come up with a polynomial time algorithm for M1-Sat if $p \in O(f(n))$. Describe the algorithm and explain why an even slightly faster growth rate would not work (for your algorithm – whether or not a faster growth rate works in general depends on whether P = NP).

Solution:

(a) Proof that M1-Sat is in NP: Verify a certificate consisting of a truth assignment – check that every clause is satisfied and that the number of variables x with $x =$ true (1) is $\leq k$.

Proof that VC (vertex cover) reduces to M1-Sat: Let $G = (V, E)$ and integer c be an instance of VC. Let $V = \{v_i \mid i = 1, \ldots, n\}$. Now create an M1-Sat formula F in which variable x_i represents v_i and, for each $v_iv_j \in E$, there is a clause $(x_i \vee x_j)$. Let k in the M1-Sat instance be the c of the VC instance.

F can be created in polynomial time: all you need to do is traverse the edges of G, creating a clause for each one.

If G has a cover S of size c, let $x_i =$ true for every $v_i \in S$ and false otherwise. This ensures that the number of true variables is $\leq k = c$. Since every edge v_iv_j has at least one vertex in S, every clause has at least one true variable.

Conversely, if F has a satisfying assignment with $\leq k$ true variables, let X be the set of true variables and let $S = \{v_i \mid x_i =$ true $\}$. Clearly $|S| \leq k = c$. And, since every clause in F has at least one true variable, every edge in E will have at least one endpoint in S.

Note: The reduction from VC shows that M1-Sat is NP-Complete even if all literals are positive and no clause has more than two literals. Also, for the M1-Sat instances created by the reduction, it is possible to obtain a solution with *exactly* k literals rather than merely $\leq k$.

A much simpler reduction from ordinary Sat (or 3-Sat) takes an instance F (formula) of Sat and transforms it into an instance F, k of M1-Sat with $k = n$, the number of variables in F. Clearly F has a satisfying assignment if and only if F has a satisfying assignment with $\leq k = n$ true variables.

2 points for proof that M1-Sat is in NP; 2 points each for spelling out the details of the transformation, proving that a certificate for the VC instance implies one for the M1-Sat instance, and for proving the converse.

(b) Let x be a variable that does not have any positive occurrences. Let $F/(x = 0)$ be the reduced version of F in which x is false. Then all the clauses of F in which x appears are eliminated, the value of p for $F/(x = 0)$ is the same as that for F, and, if there is a satisfying assignment with k true variables in $F/(x = 0)$, there will be one in F as well (since none of the clauses of F can be satisfied by making x true).

So all we need to do is the following:

- Create a set S of variables that have no positive ocurrences; this can be done in one pass over F and requires $O(|F|)$ time and $O(n)$ bits of additional space if the set is maintained as a bit vector.
- Do another pass over F, eliminating any clause that contains a variable in S. Since S is stored as a bit vector, determining whether $x \in S$ takes constant time – total time is therefore $O(|F|)$.
- The remaining formula has p variables. Try out every possible setting of those p variables in $O(2^p|F|)$ time. Since p is constant, the algorithm runs in time $O(|F|)$.

2 for the observation about reducing the formula, 2 for the details of the algorithm.

(c) Based on part (b) we need to find the fastest growth rate $f(n)$ for which $2^{f(n)}$ is polynomial. This means $f(n) \in \mathrm{O}(\lg n)$.

3. *12 points: 2 points for (a), 8 for (b), and 2 for (c)*

Purpose: understanding NP*-completeness and reductions among the four variants of a problem: optimization, evaluation, certificate, and decision.*

Do parts (a)-(c) of Problem 34-3 on page 1103 (page 1019 in 2/e) but with the following twist: define the *graph coloring problem* as one whose input is a graph G and whose output is an actual function $c : V \to \{1, \ldots, k\}$ that minimizes k. In other words, the output has to *give the actual color of each vertex* in the k-coloring.

The most important part of this problem is part (b) and, as I pointed out in class, the hardest reduction is from the decision version to the certificate. Here it is important to note that the actual color of a vertex does not

matter – what matters is which vertices have the same color.

Solution:

(a) Suppose the two colors are red (R) and blue (B). The algorithm involves a simple modification of DFS-VISIT – an almost identical modification works for BFS. Since this is an undirected graph, the only non-tree edges are back edges.

As you create the DFS tree, color each vertex R or B. When traversing the adjacency list, give all previously unvisited vertices the opposite color. If an already visited vertex has the same color, this is a contradiction. In this situation the graph has a cycle of odd length and it's easy to see that no assignment of two colors is possible.

(b) The decision problem can be stated as follows: Given an undirected graph $G = (V, E)$ and an integer k, does there exist a function $c : V \to \{1, \ldots, k\}$ such that, for every $uv \in E$, $c(u) \neq c(v)$? We will refer to c as a k-coloring from here on.

Suppose we were able to solve (in polynomial time) the optimization problem, which is to find a k-coloring with minimum k. Then we can easily solve the *evaluation* problem – output the minimum k without giving c, the *certificate* problem – given k, output a k-coloring or indicate that none is possible, and the decision problem.

The other direction is the hard part. First note that polynomial algorithms for both the evaluation and certificate problem imply one for the optimization problem: Use the evaluation algorithm to find the minimum k and the certificate algorithm to produce the actual coloring. We can use an algorithm for the decision problem to solve the evaluation problem: Simply run the decision algorithm for $k = 1, \ldots, n$ until the answer is "yes".

Now suppose GC-DEC(G, k) is a boolean function that, in polynomial time, solves the decision problem. Define G/xy to be G with vertices x and y identified as in edge contraction, but in this case xy is not an edge. More precisely, if $G = (V, E)$ then $G/xy = (V', E')$ where $V' = V - \{y\}$ and $E' = E - \{yz \in E\} \cup \{xz \mid yz \in E \text{ and } xz \notin E\}$. The following algorithm uses GC-DEC to solve the certificate problem.

```
GC-CERT(G = (V, E), k) returns a k-coloring c if one exists false otherwise
    if |V| ≤ k then return a c that maps each vertex to a different color endif
    for all pairs of vertices x, y do
        if xy ∉ E  and GC-DEC(G/xy, k) then
            c ← GC-CERT(G/xy, k)
            return c modified so that c(y) = c(x)
        endif
    end do
    return false
end GC-CERT
```

The algorithm either returns a trivial coloring – if the number of colors $\geq$ the number of vertices – or uses the decision algorithm to determine whether there's a pair x, y of non-adjacent vertices that can be colored the same color in such a way that no additional colors will be required. If there is such a pair x, y, the instance is reduced to one in which x and y are identified. If not, it's impossible to color the graph with k colors. If it were possible, each color would correspond to a set of mutually non-adjacent vertices that could be identified in the above algorithm.

Runtime of the algorithm is dominated by the number of loop iterations, $O(n^2)$, times the runtime of operations in the loop body: forming G/xy can easily be done in $O(n^2)$ by scanning all the edges and the call to GC-DEC requires polynomial time. So the total is $O(n^4 + n^2 D(n))$, where $D(n)$ is the time for the decision algorithm. This is clearly polynomial.

Another option is to pick a pair x, y of vertices that are not adjacent and consider two cases: either they have the same color or they don't. If they do have the same color the smaller instance is as described above. If they don't have the same color, the "smaller" instance is the original graph with

an edge added between x and y. Why is this smaller? Because it decreases the ratio $|V|/|E|$. When that ratio reaches $2/(|V|-1)$ the graph is a clique and requires $|V|$ colors.

2 points each for general observation about the relationships among the problems, 1 point for the evaluation algorithm, 3 points for figuring out how to reduce to a smaller instance, and 2 points for algorithm details.

(c) The k-coloring decision problem, k-COLOR, is in NP: for a certificate, i.e., a mapping c, we can, in polynomial time, verify that, for every $xy \in E$, $c(x) \neq c(y)$.

Now reduce 3-COLOR to k-COLOR: This is trivial. The reduction maps the graph G of the 3-COLOR instance to the same graph G and lets $k = 3$. Clearly G can be 3-colored iff G can be k-colored with $k = 3$.

4. *10 points: 2 points for (a), 8 for (b)*

Understanding NP-completeness proofs

Do Problem 34-4, parts (a) and (b) on page 1104. The problem statement is identical to that of Problem 3 of Homework 3, but there is no restriction on the t_j's. You already did parts (c) and (d) in Homework 3.

Solution:

(a) Given a set of n tasks $a_1, \ldots, a_n$; corresponding $t_1, \ldots, t_n$ and $p_1, \ldots, p_n$ and $d_1, \ldots, d_n$; and a nonnegative number k, does there exist a subset A of $a_1, \ldots, a_n$ such that for each $a_i \in A$, the sum of the t_j's for all $a_j \in A$ with $d_j \leq d_i$ is $\leq d_i$ and $\sum_{a_i \in A} p_i \geq k$?

Note that the order in which the tasks are scheduled does not matter as much as which ones finish before their deadlines.

(b) The above decision problem is in NP. Given a certificate A – the subset referred to in part (a), we can sort the tasks of A by increasing deadline and maintain the total time T so far. As each new task a_i in A is added, we check that $T + t_i \leq d_i$. If not the certificate is not valid. If so, we let $T = T + t_i$ and move on.

We can show that scheduling with profits and deadlines (SPD) is NP-hard via a transformation from subset sum (SS). Let $S = \{s_1, \ldots, s_n\}$ with target value t be an instance of SS. For each s_i create a task a_i with $t_i = s_i$, $p_i = s_i$, and $d_i = t$. Finally, we let $k = t$. This defines an instance of SPD.

If there is a subset S' of S with $\sum_{s_i \in S'} s_i = t$ then we can schedule the tasks corresponding to the integers in S' before deadline t – their total time is exactly t – and earn a profit $\geq k = t$ – their total profit will exactly t. So simply put a_i into A if $s_i \in S'$.

Conversely, if the tasks in a certificate set A are completed before their deadlines in the SPD instance, they must all finish before time t. So their total time must be $\leq t$. The total profit of the tasks must be $\geq k = t$. In other words $\sum_{a_i \in A} t_i \leq t$ and $\sum_{a_i \in A} p_i \geq t$. Since $s_i = t_i = p_i$ we know $\sum_{a_i \in A} s_i = t$. So simply put s_i into S' if $a_i \in A$.

In part (b), 2 points for each of the four parts: proof that it's in NP – not as trivial as usual, details of construction, proof of certificate mapping in each direction.