CS 516 Final Exam

KEY

This is a "take home" exam and is thus open to books and notes, but you must do your own work and not consult other people. You are free to assume the results of any theorem or solved problems from the textbook, course discussions, or homework problems. **Solve each problem on separate sheets of paper.** Neatness, conciseness, clarity, and correctness count. Give concrete example of constructions if they make the solution clearer.

1. Show that the following set F is recognizable but not co-recognizable:

$$F = \{\langle M \rangle \mid M \text{ is a TM that accepts at least 481 strings}\}.$$

Proof. We show that F is recognizable by constructing the following TM R that accepts all $\langle M \rangle \in F$ but loops forever otherwise:

$R =$ "On input $\langle M \rangle$ where M is a TM:
1. Initialize counter $c = 0$.
2. For $L = 0 \ldots \infty$
3. For all strings $x \in \Sigma^*$ where $|x| \leq L$
4. Simulate M on input x for at most L steps.
5. If M accepts x
6. Increment counter $c \leftarrow c + 1$.
7. If $c \geq 481$ then halt and accept."

Note R's careful construction – in particular step 4 limits the number of steps used to simulate M. L is used to cap both the length of all strings x and the number of simulation steps. If M indeed accepts at least 481 strings, then eventually M will be allowed to execute a sufficient number of steps to accept 481 of these strings.

By Rice's Theorem, clearly F is not decidable. Therefore, by Theorem 4.22 it is not co-recognizable.

2. You are given a 3-CNF boolean formula ϕ with v variables and c clauses.

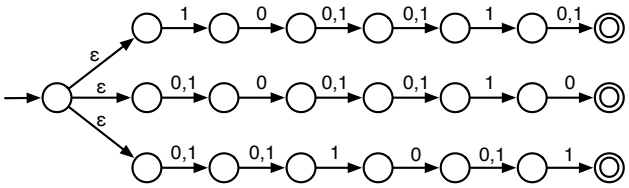
   (a) Show how to contruct, in polynomial time, an NFA N with $O(v \cdot c)$ states that accepts all non-satisfying assignments which are represented as a boolean string of length v.

   (b) Argue why we can not build an DFA for $\overline{L(N)}$ in polynomial time unless P = NP.

Solution:

(a) As an example, the NFA N below accepts non-satisfying boolean strings for the formula ($v = 6$, $c = 3$):

$$\phi = (\overline{x_1} \vee x_2 \vee \overline{x_5}) \wedge (x_2 \vee \overline{x_5} \vee x_6) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_6})$$



N begins by nondeterministically choosing one of the $c = 3$ rows containing $v + 1 = 7$ states which corresponds to one of the c clauses. The first row accepts strings of the form $10(0|1)(0|1)1(0|1)$ which corresponds to $x_1 = T$, $x_2 = F$, $x_5 = T$ which does not satisfy the first clause. This construction can be generalized for any 3-CNF formula ϕ with $c \times (v + 1) + 1$ states.

(b) Every string accepted by N corresponds to an assignment of variables that does not satisfy ϕ. If we can find a string $s \notin L(N)$ of length v, then we know ϕ is satisfiable. Thus if we can build a DFA for $\overline{L(N)}$ we could search for an accepting path of length v to determine if ϕ where satisfiable. If we could build such a DFA in polynomial time (we certainly can look for a path in polynomial time) then we could decide 3SAT in polynomial time. Of course we already know that 3SAT is NP-complete, so this would only be possible if P = NP.

3. A zoo keeper wants to place as many different animals in a large pen as possible. The zoo keeper wants to avoid putting animals in the same pen that do not get along. If there are n different animals to choose from (numbered 1 through n), we can construct a $n \times n$ matrix whose entries measure how much any pair of animals *clash*. A clash value of 0 means the animals can live in perfect harmony, whereas a clash value of 1 means one will likely kill the other (*e.g.,* a lion and zebra would have a clash value of 1). Here is sample matrix for $n = 5$ example animals:

	1	2	3	4	5
1	0.0	0.2	0.4	0.9	1.0
2	0.2	0.0	0.6	0.1	0.3
3	0.4	0.6	0.0	0.4	0.5
4	0.9	0.1	0.4	0.0	0.1
5	1.0	0.3	0.5	0.1	0.0

Animals 2 and 4 will mostly get along, but putting animals 1 and 5 together is a bad idea. Any set of animals incurs a *penalty* which is the sum of all pairs of clash values; *e.g.;* for example, the set $\{1, 2, 4\}$ incurs a penalty of $0.2 + 0.9 + 0.1 = 1.2$.

$$\text{ZOO-KEEPER} = \{\langle n, M, p, k\rangle \quad | \quad n = \text{number of animals, } M \text{ is } n \times n \text{ clash matrix,} \\ \text{we can place } k \text{ animals in a pen with penalty} \leq p\}$$

Show that ZOO-KEEPER is NP-complete.

Proof. ZOO-KEEPER is in NP since we can verify the that penalty for a given k animals does not exceed p in polynomial time.

We show that ZOO-KEEPER is NP-hard via the reduction 3SAT $\leq_p$ ZOO-KEEPER. Given a 3CNF formula ϕ with V variables and C clauses we show how to construct an instance $\langle n, M, p, k\rangle$ of ZOO-KEEPER with $n = 2V + 3C$ animals in polynomial time.

- All the entries of the (symmetric) clash matrix M are assumed to be zero unless specified otherwise.
- We create $2V$ "variable animals" corresponding to each variable x_i being true and each variable x_i being false; contradicting variable settings result in a clash of 1:

$$M(x_i \text{ is true}, x_i \text{ is false}) \quad = \quad 1 \quad \text{(and vice versa)}$$

- There are 3 animals (c_1, c_2, c_3) for every clause for a total of $3C$ "clause animals." The following clause

$$(\overline{x_2} \vee x_5 \vee \overline{x_7})$$

will generate the following clashes between clause and animal variables:

$$\begin{aligned} M(c_1, x_2 \text{ is true}) &= 1 \text{ (and vice versa)} \\ M(c_2, x_5 \text{ is false}) &= 1 \\ M(c_3, x_7 \text{ is true}) &= 1 \end{aligned}$$

We also force animals from the same clause to clash:

$$\begin{aligned} M(c_1, c_2) &= 1 \\ M(c_1, c_3) &= 1 \\ M(c_2, c_3) &= 1 \end{aligned}$$

Follow this example for all clauses in ϕ.

- We set $k = V + C$ and $p = 0$ which means that the ZOO-KEEPER solver must find k animals out of n that can live in perfect harmony.

This construction can obviously be be done in polynomial time.

If $\phi \in$ 3SAT then we can use the satisfying setting of the V variables and pick the corresponding V variable animals and choose a satisfied term from each clause for the remaining C clause animals.

Conversely, if our constructed instance $\langle n = 2V+3C, M, p = 0, k = V+C \rangle \in$ ZOO-KEEPER then we can satisfy ϕ by simply setting ϕ's variables according to the "variable animals" settings. $\square$

4. (7.29) Consider the following scheduling problem. You are given a list of final exams $F_1, \ldots, F_k$ to be scheduled, and a list of students $S_1, \ldots S_l$. Each student is taking some specified subset of these exams. You must schedule these exams into slots so that no student is required to take two exams in the same slot. The problem is to determine if such a schedule exists and only uses h slots. The language is

   $$\text{SCHEDULE} = \{\langle F, S, h\rangle \mid F \text{ is a list of finals, } S \text{ is a list of subsets specifying the finals each student is taking, and finals can be scheduled in to } h \text{ slots so that no student is taking two exams in the same slot}\}.$$

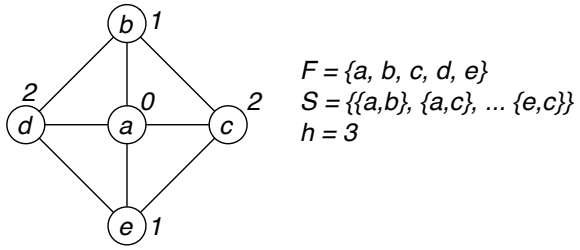
   For example, if $F = \{1, 2, 3, 4\}$ and $S = \{\{1, 3\}, \{1, 2\}, \{2, 3, 4\}\}$, then $\langle F, S, 2\rangle \notin$ SCHEDULE, but $\langle F, S, 3\rangle \in$ SCHEDULE. Show that SCHEDULE is NP-complete.

   Proof. SCHEDULE is in NP because we can guess a schedule with h slots and verify in polynomial time that no student is taking two exams in the same slot. We prove that SCHEDULE is NP-hard by showing that 3COLOR $\leq_P$ SCHEDULE. Given a graph $G = (V, E)$ we simply create an instance $\langle F, S, h\rangle$ where $F = V$, $S = E$, and $h = 3$ (which can obviously be done in polynomial time).

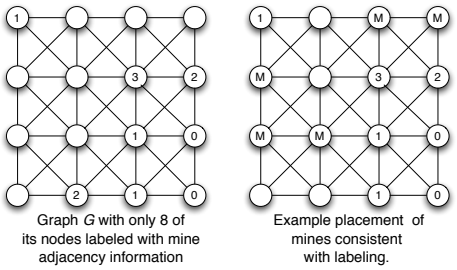
   Given a 3-coloring for G, we can get a schedule for $\langle V, E, 3\rangle$ by assigning finals that correspond to nodes colored with color i to slot i for $i = 0, 1, 2$. Adjacent nodes are colored by different colors, so corresponding exams taken by the same student are assigned to different slots.

   Given a schedule for $\langle V, E, 3\rangle$, we can get a 3-coloring for the original graph by assigning colors according to the slots of corresponding nodes. Since exams taken by the same student are scheduled in different slots, adjacent nodes are assigned different colors. □

   Below is an example 3-colored graph which divide the 5 finals into 3 slots for 8 students. Final a is scheduled into slot 0, b and e into slot 1, and c and d into slot 2.



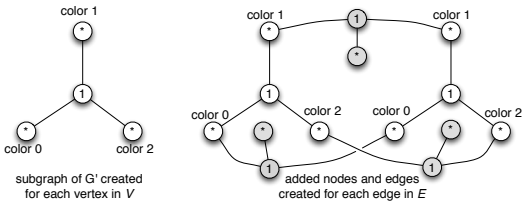
5. (7.30) In the *mine consistency problem* (MCP) you are given an undirected graph G along with a labeling of some of G's nodes. These labels are numbers which reveal how many *mines* are adjacent to the corresponding node. You must determine whether there is a placement of mines on the remaining unlabeled nodes that is consistent with the labeling. The partially labeled graph on the left of the figure below does have a consistent labeling as revealed by the mine placement on the right:



Formulate this problem as a language and show that it is NP-complete.

MCP = $\{\langle G, N\rangle \mid G = (V, E)$ is an undirected graph and $N : V \to \{*, 0, 1, 2 \ldots\}$ is a function that assigns either a * or a number to each vertex. There exists some placement of mines on *-nodes that is consistent with the adjacency counts specified by N.$\}$.

Proof. Clearly MCP in is NP since we can guess a placement of mines in *-nodes and verify consistency in polynomial time. To show that MCP is NP-hard we prove 3COLOR $\leq_P$ MCP. Given an input graph $G = (V, E)$ to the 3COLOR problem, we construct a corresponding graph $G' = (V', E')$ and a labeling N. First, for each vertex in V, we create 4 vertices and 3 edges connected and labeled as illustrated on the left of the following figure:



The center node is labeled with a 1 and therefore exactly one of its three adjacent *-nodes must contain a mine – this corresponds to the color chosen for the input vertex. Next, for each E, we add 6 nodes and 9 edges to G' that are labeled and connected with the corresponding vertices as shown on the right of the figure. The new (gray) nodes labeled with a 1 prevent mines from being placed in both adjacent *-nodes corresponding to the same color. The new (gray) *-nodes allow for a consistent labeling when neither corresponding color is selected.

The new graph G' has $4|V| + 6|E|$ nodes and $3|V| + 9|E|$ edges and thus can obviously be constructed from G in polynomial time.

Any consistent placement of mines in G' will tell us exactly how to color the input graph G. Each vertex of V is assigned exactly one color and our construction makes it impossible for adjacent vertices of V to have the same color. Therefore $\langle G, N\rangle \in$ MCP $\Rightarrow \langle G\rangle \in$ 3COLOR.

Given a 3-coloring of G we can determine exactly where to place mines. Place one mine based on each vertex's color in V as implied by the left figure. Add any needed mines to the extra (gray) *-nodes so that all of the (gray) 1-nodes are satisfied. Therefore $\langle G\rangle \in$ 3COLOR $\Rightarrow$ $\langle G, N\rangle \in$ MCP.$\square$

6. (8.9) A *ladder* is a sequence of strings $s_1, s_2, \ldots s_k$, wherein every string differs from the preceding one in exactly one character. For example the following is a ladder of English words, starting with "`head`" and ending with "`free`":

   `head, hear, near, fear, bear, beer, deer, deed, feed, feet, fret, free.`

   Let

   $$\text{LADDER}_{DFA} = \{\langle M, s, r\rangle \mid M \text{ is a DFA and } L(M) \text{ contains a ladder of strings, starting with } s \text{ and ending with } t\}.$$

   Show that LADDER_{DFA} is in PSPACE.

   Proof. We construct a nondeterministic TM N that decides LADDER using polynomial space:

   N = "On input $\langle M, s, r\rangle$ where M is a DFA and s and r are strings:
   1. If $r \notin L(M)$ reject. (Decidable since M is a DFA).
   2. For $i = 1 \ldots |\Sigma^{|s|}|$
   3. If $s \notin L(M)$ reject.
   4. Mutate one character in s at random (pick random position and random character from Σ).
   5. If $s = r$ then accept.
   6. Reject.

   The maximum number of iterations $|\Sigma^{|s|}|$ guarantees that N will have a chance to examine all ladders beginning with s. The counter i can be encoded in polynomial space. This shows that LADDER $\in$ NSPACE, but NSPACE = PSPACE by Savitch's Theorem. $\square$.