NATIONAL UNIVERSITY OF SINGAPORE

School of Computing

Final Examination for
Semester 1 AY2013/14

CS5234—Combinatorial and Graph Algorithms

November 19 2013 Time Allowed: 2 Hour

INSTRUCTIONS TO CANDIDATES

1. This exam contains **FOUR (4)** questions and comprises **FOURTEEN (14)** printed pages, including this page and four pages of scratch paper.
2. The exam is closed book. You may bring two double-sided sheet of A4 paper to the quiz. (You may not bring any magnification equipment!) You may not use a calculator, your mobile phone, or any other electronic device.
3. Write your solutions in the space provided. If you need more space, then use the scratch paper at the end of the exam.
4. Read through the problems before starting. Do not spend too much time on any one problem.
5. Show your work. Partial credit will be given.
6. Be neat. If I cannot read your solution, then I cannot give you credit for it.
7. You may use (unchanged) any algorithm or data structure given in class. You do not need to restate the algorithm. If you modify the algorithm, you must explain exactly how your version works.
8. Draw pictures frequently to illustrate your ideas.
9. Good luck!

MATRICULATION NUMBER: _____

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Warehouse Management	20	
2	Linear Programming	30	
3	MaxCut	24	
4	SecretNets	26	
Total:		100	

Problem 1. Warehouse Management [20 points]

You have been placed in charge of the delivery infrastructure for an on-line grocery store. Your job is to decide where to situate warehouses in order to minimize the costs of operating the warehouse and shipping the groceries to the customers. You have information on all your customers, including their location and the amount of goods that they want. Each customer must receive all the required groceries, and may receive shipments from one or more warehouses. In more detail, for this problem, you are given:

- A set of customers $c_1, \ldots, c_n$, where each c_i represents the location of customer c_i. (Each location is a set of coordinates in Euclidean space.)
- The demand d_i for each customer c_i. That is, customer c_i requires d_i kilograms of groceries.
- A set of possible locations to open warehouses $w_1, \ldots, w_k$. Each w_j repesents the location of warehouse site w_j.
- The cost v_j of operating each warehouse w_j. That is, to operate a warehouse at location w_j costs w_j dollars.
- The cost Z for shipping 1 kilogram of groceries 1 kilometer. That is, if you are shipping x kilograms of groceries to a customer that is y kilometers from the warehouse, then it costs xyZ dollars.

Your job is to formulate this problem as an integer linear program. (You do not have to solve the problem. You simply have to write the appropriate integer linear program.) Your integer linear program does not have to be in standard form, however it should only use constructs that we have already shown can be transformed into standard form.

Please write your answer on the next page.

Variables: *(For each variable, explain its intuitive meaning.)*

Solution: There are two sets of variables: $x_1, \ldots, x_k$, one per warehouse, and $f_{j,i}$, one for every pair $j \in \{1, \ldots, k\}$ and $i \in \{1, \ldots, n\}$. The x_j are integral, and variable $x_j = 1$ if you open warehouse w_j, and $x_j = 0$ otherwise. The variable $f_{j,i}$ represents the number of kilgrams of groceries shipped from warehouse w_j to custoemr c_i, and do not have to be integral.

Objective:

Solution: minimize $\sum_{j \in \{1,\ldots m\}} \left(v_j \cdot x_j + \sum_{i \in \{1,\ldots,n\}} f_{j,i} \cdot Z \cdot |c_j - w_i| \right)$

Constraints: *(For each constraint, first describe in English the goal of the constraint. Then provide the mathematically precise constraint.)*

Solution:

- [*All x_j variables are integral 0/1 variables.*]
  $\forall j \in \{1, \ldots, m\} : x_j \in \{0, 1\}$
- [*All flows are at least 0.*]
  $\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}, f_{j,i} \geq 0.$
- [*Each customer receives the desired quantity of groceries.*]
  $\forall i \in \{1, \ldots, n\} : \sum_{j \in \{1,\ldots,m\}} f_{j,i} = d_i.$
- [*Only deliver goods from open warehouses.*]
  $\forall j \in \{1, \ldots, m\}, \forall i \in \{1, \ldots, n\} : f_{j,i} \leq x_j \cdot d_i.$

Notice in the last contraint that $x_j d_i$ is the maximum amount of groceries that warehouse j can deliver to customer i. If $x_j = 1$ and the warehouse is open, then this quantity equals d_i, i.e., the demand by i. If $x_j = 0$ and the warehouse is closed, then this quantity equals 0.

Problem 2. Linear Programs [30 points]

Problem 2.a. [15 points] Give all the vertices of the polytope associated with the following linear program, and find the point that maximizes the objective function.

$$\begin{array}{lrl} \text{maximize} & x_1 + 2x_2 & \\ \text{subject to} & x_1 + 3x_2 & \leq 6 \\ & -6x_1 - 3x_2 & \leq -3 \\ & x_1 + x_2 & \leq 4 \\ & x_1 & \geq 0 \\ & x_2 & \geq 0 \end{array}$$

Solution: The following points are vertices of the polytope:

- $(0.5, 0)$
- $(0, 1)$
- $(0, 2)$
- $(3, 1)$
- $(4, 0)$

The vertex that maximizes the objective function is $(3, 1)$. (Drawing a picture makes this easier.)

Problem 2.b. [15 points] Find the dual of the following linear program:

$$\begin{array}{rcl} \text{maximize} & 2x_1 + x_2 + 4x_3 + 12x_4 & \\ \text{subject to} & x_1 + 3x_2 & \leq 6 \\ & x_1 + 7x_3 + 4x_4 & \geq 9 \\ & x_1 + x_2 + x_3 + 4x_4 & = 13 \\ & x_1 & \geq 0 \\ & x_2 & \geq 0 \\ & x_3 & \geq 0 \\ & x_4 & \geq 0 \end{array}$$

Solution:

$$\begin{array}{rcl} \text{minimize} & 6y_1 - 9y_2 + 13y_3 - 13y_4 & \\ \text{subject to} & y_1 + y_2 + y_3 - y4 & \geq 2 \\ & 3y_1 + y_3 - y_4 & \geq 1 \\ & 6y_2 + y_3 - y_4 & \geq 4 \\ & 4y_2 + 4y_3 - 4y_4 & \geq 12 \\ & y_1 & \geq 0 \\ & y_2 & \geq 0 \\ & y_3 & \geq 0 \\ & y_4 & \geq 0 \end{array}$$

Notice that the variables $(y_3 - y_4)$ always appear together, and the dual could be simplified:

$$\begin{array}{rcl} \text{minimize} & 6y_1 - 9y_2 + 13y' & \\ \text{subject to} & y_1 + y_2 + y' & \geq 2 \\ & 3y_1 + y' & \geq 1 \\ & 6y_2 + y' & \geq 4 \\ & 4y_2 + 4y' & \geq 12 \\ & y_1 & \geq 0 \\ & y_2 & \geq 0 \\ & y' & \text{unconstrained} \end{array}$$

Problem 3. Approximating MaxCut [24 points]

The problem of finding a minimum cut in a graph $G = (V, E)$ is easy to solve (e.g., using Ford-Fulkerson). By contrast, the problem of finding a maximum cut is NP-hard! In this problem, we will develop a 2-approximation algorithm for this problem. (A famous result by Goemans and Williamson uses a more complicated technique—semidefinite programming—to find a 0.878 approximation.)

Consider an unweighted, undirected graph $G = (V, E)$. The goal is to find a maximum cut, i.e., to partition the vertices V into two (disjoint) sets A and B where the number of edges crossing the cut is maximized.

Assume you are given a cut (A, B) where $A \subseteq V$ and $B \subseteq V$. For every node v, we define: $edges(v, A)$ to be the number of edges connecting v to a node in A, and $edges(v, B)$ to be the number of edges connecting v to a node in B.

The idea of our algorithm is to start with an arbitrary cut, and incrementally (greedily) improve it: (i) we search for a node in A with more edges connected to nodes in A than nodes in B, and move it to B; or (ii) we search for a node in B with more edges connected to nodes in B than nodes in A, and move it to A. We continue until we cannot improve the cut any more, and then return the resulting sets. The algorithm is defined as follows:

```
FindCut(V, E)
  A ← V                          ▷ Assign all the nodes to set A.
  B ← ∅                          ▷ B is empty.

  ▷ Search for a node v to move across the cut.
  while ∃v ∈ A where edges(v, A) > edges(v, B) or
        ∃v ∈ B where edges(v, B) > edges(v, A)
    do
       if v ∈ A                  ▷ Then move v to B.
          then  A ← A \ {v}
                B ← B ∪ {v}
       if v ∈ B                  ▷ Then move v to A.
          then  B ← B \ {v}
                A ← A ∪ {v}
  ▷ Return the cut.
  return (A, B)
```

The problem continues on the next page.

Problem 3.a. [12 points] Prove that the algorithm FindCut, described above, terminates in polynomial time.

(Hint: what is the largest possible cut that the algorithm discovers?)

Solution: In every iteration of the while loop, the cost of the cut (A, B) increases by at least one. To see this, consider a node v that is moved from A to B. Let $x = \mathit{edges}(v, A)$ and $y = \mathit{edges}(v, B)$. By moving v from A to B, the cost of the cut is increased by x (as now all the edges to A cross the cut) and decreased by y (as the edges to B used to cross the cut but do not anymore). That is, the cost of the cut increases by $(x - y)$. We know, however, that $x > y$, since we chose node v where $\mathit{edges}(v, A) > \mathit{edges}(v, B)$. Hence, the cost of the cut increases by at least 1. (The same holds true for a node moving from B to A.)
The largest possible cut that the algorithm discovers is m. Thus, the while loop can run for at most m iterations. All the remaining steps are polynomial time, and hence the algorithm runs in polynomial time.

Problem 3.b. [12 points] Prove that the algorithm FindCut, described above, finds a 2-approximation of the maximum cut.

(Hint: For each node, what is the fraction of edges that cross the cut?)

Solution: For each node, at least 1/2 the adjacent edges cross the cut, as otherwise the while loop would not terminate. This implies (by a simple counting argument) that at least half the edges cross the cut, i.e., the cost of the cut is at least $m/2$. On the other hand, optimal algorithm returns a cut with value at most m, and hence the algorithm is a 2-approximation.

Problem 4. SecretNets Corporation Incorporated, Ltd. [26 points]

SecretNets Corp. has built an overlay network for tranporting data across the internet. The SecretNet consists of a directed graph $G = (V, E)$ with a set of entry nodes $e_1, e_2, \ldots, e_k$ and a set of exit nodes $x_1, x_2, \ldots, x_k$. SecretNets makes the following guarantee to its clients: "Your packets will be routed securely through the network, and they will never share a node or an edge in the network with a competitor." This ensures that there is no way that anyone can spy on their packets!

Problem 4.a. [8 points] The engineer who originally designed the SecretNet just quit. Your job, as the replacement, is to find the maximum number of clients that the SecretNet can support.

- Each client is assigned some entrypoint e_i and some exit point x_j. (Any combination of entry point and exit point is allowed.)
- Each client is assigned a path through the network from e_i to x_j.
- No two client paths intersect at any **nodes or edges**.

Give an efficient algorithm for finding the maximum number of clients that the SecretNetwork can support.

Solution: First, we build a new graph G' as follows: (i) For every node v in graph G create two nodes v_{in} and v_{out} in the new graph G'. (ii) Add a directed edge (v_{in}, v_{out}) connecting the pairs of nodes. (iii) For every edge (u, v) in the original graph G, add an edge (u_{out}, v_{in}) in the new graph G'. (iv) Create a new source node s, and connect it to every entry point e_i with a directed edge $(s, (e_i)_{in})$. (v) Create a new destination node t and connect it to ever exit point x_j with a directed edge $((x_j)_{out}, d)$.
Notice that every path in G' from an entrypoint e_i to an exitpoint x_J can be mapped to a path in G connecting the same entry and exit points.
Assign a capacity of 1 to every edge in the graph. Find the maximum flow, using Push-Relabel, in polynomial time. The value of the maximum flow is eactly the number of clients that the SecretNetwork can support.
First, notice that if the flow is f, then by the flow decomposition theorem, this can decomposed into k flows $f_1, f_2, \ldots, f_k$ that add up to $|f|$. However, since each outgoing edge from the source has capacity one, each of these flows f_j has capacity one, and hence $k = |f|$. Moreover, each of these flows has to be disjoint: they cannot intersect at one of the *in* nodes, since each such node has only one outgoing edge with capacity one; they cannot intersect at one of the *out* nodes, since each such node has only one incoming edge with capacity one. Thus, we have found $|f|$ disjoint paths through the network G'; each of these can be mapped to a disjoint path in G.
Second, assume for the sake of contradiction that there exist more than $|f|$ disjoint paths from s to t in G. Each of these can be mapped to a disjoint path in G', and hence can lead to a flow of size $> |f|$. However, we have already posited that f is the maximum flow in G', so this is a contradiction.

Problem 4.b. [18 points] It turns out that there are only d clients, at the moment, fewer than the maximum number that your network can support. Each link in the network has a cost to use it, and you want to find the d disjoint paths that are cheapest. That is:

- Assume each edge in the SecretNetwork $e \in E$ has a bandwidth cost $b(e)$.
- Your algorithm should output d paths $p_1, p_2, \ldots, p_d$ that are entirely disjoint, that is, they do not intersect at either the nodes or edges.
- The cost of a path p_j is the sum of the edge costs, i.e., $\sum_{(v,w)\in p_j} b(v,w)$.
- Your algorithm should minimize the total cost, i.e., $\sum_{j=1,\ldots,d} cost(p_j)$.

Give an efficient algorithm, and prove that it satisfies the requirements.

(For partial credit, you may instead give an algorithm that finds client paths that do not intersect at any ***edge*** *in the SecretNetwork.)*

Solution. There are a few different possible solutions here, but the most straightforward is to express this problem as an integer linear program, relax it to a linear program, and rely on an LP solver. While at least one solution to the LP is integral, it is not true that all solutions are integral. We can then give a rounding procedure for transforming a non-integral solution into an integral solution of the same cost.

To develop the (integer) linear program, we first model the problem as a min-cost network flow. (This is not strictly necessary, but simplifies the development of the LP.) Construct a network flow graph G' as in part (a) such that any set of edge-disjoint flows are also node disjoint. Each edge in the graph has capacity $c(e) = 1$, and each edge e has a cost $b(e)$ as specified; new edges added in the transformation from G to G' have cost $b(e) = 0$.

The specified linear program has variables $f_1, f_2, \ldots, f_m$, i.e., one variable for each edge. Intuitively, $f_j = 1$ if one of the d paths runs through edge e in the specified direction, and $f_j = 0$ otherwise. The goal is to minimize the cost of the selected edges, subject to the constraint that we find d disjoint paths. We the construct the following LP (dropping the integrality constraints):

$$
\begin{array}{rcl}
\text{minimize} & \sum_{j=1}^{m} f_j b(j) & \\
\text{subject to} & \sum_{e:e=(s,v)\in E, v\in V} f_e & = -p \\
& \sum_{e:e=(v,t)\in E, v\in V} f_e & = p \\
\forall u \in V \setminus \{s,t\} : & \sum_{e:e=(v,u), v\in V} f_e - \sum_{e:e=(u,w), w\in V} f_e & = 0 \\
& \forall e \in E : f_e & \leq b(e) \\
& \forall e \in E : f_e & \geq 0 \\
& \forall e \in E : f_e & \leq 1
\end{array}
$$

First, assuming that if this solution is integral, we observe that it finds p disjoint paths. Since the flows are integers between zero and one, we conclude that each $f_e \in \{0, 1\}$, i.e., each edge is either included or not included. Moreover, due to the flow constraints, each node (aside from s and t) must have the same number of incoming edges selected as outgoing edges selected. Finally, by construction, each node (aside from s and t) has either at most one incoming edge or at most one outgoing edge in the graph, and hence the graph consists of disjoint paths from s to t. Finally, the source has outgoing flow of p, and hence there are p disjoint paths.

Second, since the objective function minimizes the sum of the included edge costs, the linear program successfully finds the k disjoint paths with the smallest cost, as long as it finds an integral solution.

Third, we observe that if there are p disjoint paths, then there is a feasible solution. Notably, for each edge e on the p disjoint paths, set $f_e = 1$, and all the constraints are satisfied.

The trickiest part of this problem is constructing an integral solution from the output, which may not be integral. Assume that we have found a non-integral set of flows f_e that are feasible and minimize the objective. We show how to construct (in polynomial time) an integral version.

Start with some node v_0 that has non-integral outgoing flow. Then v_0 must have either at least one outgoing edge with non-integral flow, or at least one incoming edge with non-integral flow. (Notice this is true even of s and t, since each of these nodes has integral flow p outgoing/incoming.) Follow this outgoing/incoming edge to node v_1. Again, v_1 must have some other adjacent edge with non-integral flow; follow this outgoing/incoming edge to node v_2. Continue in this way to define a sequence $v_0, v_1, v_2, \ldots, v_t$ until we form an undirected cycle. (Notice that we must eventually form a cycle.)

We have now constructed a cycle in which every edge has non-integral flow > 0. Since each edge has integral capacity, we can also conclude that each edge on the cycle has residual capacity > 0. Let ϵ be the largest value > 0 such that every edge can have its flow increased or decreased by ϵ without exceeding its maximum capacity and without falling below zero.

Next, we construct two new solutions f^+ and f^-. For solution f^+ we proceed clockwise around the cycle from v_0 adding ϵ to every flow across a forward edge and decreasing by ϵ every flow across a reverse edge. Similarly, we construct f^- by proceeding counterclockwise around the cycle from v_0 adding ϵ to every flow across a forward edge and decreasing by ϵ every flow across a reverse edge. Notice that in constructing these two new flows, we preserve all the flow-balance constraints and the capacity constraints.

Finally, we argue that either f^+ or f^- is a better solution than the original, contradicting our assumption that we had minimized the objective function, or one of them is a flow of identical cost but with fewer non-integral edges. Let B_f be the sum of the bandwidth costs across the forward edges proceeding clockwise from v_0 around the cycle, and let B_r be the sum of the bandwidth costs across the reverse edges proceeding clockwise from v_0 around the cycle. We conclude that solution f^+ increases the objective function by $\epsilon(B_f - B_r)$. Similarly, the solution f^+ increases the objective function by $\epsilon(B_r - B_f)$ (since every forward edge going clockwise is a reverse edge going

counterclockwise, and vice versa).

If $B_f \neq B_r$, then either $(B_f - B_r)$ or $(B_r - B_f)$ is negative, and hence either f^+ or f^- is a better solution, contradicting our assumption that we had minimized the objective function. We can thus assume that $B_f = B_r$. In that case, however, we have found a new solution of the exact same cost that has (at least) one fewer non-integral edges (since by adding/subtracting ϵ from each edge, we have reached the capacity on at least one edge, or dropped one edge to zero flow, as we chose ϵ to be as large as possible.) We can thus repeat the process until we have removed all the non-integral edges.

Repeating this procedure at most m times yields an integral solution to the linear program that minimizes the objective function.

Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper

End of Exam