MAT 160 - Project 3

Sangwon Yoon, Ryan Tjoa and Ahmed Mahmoud

Problem 1:

(a) Our solution consists of computing all possible path from s to t and then maximize and pick the longest path. Let A be the set of vertices in the graph such that $s \in A$ and $t \notin A$. Let $\delta(A)$ be the set of edges that goes from a to b such that $a \in A$ and $b \notin A$. We call $\delta(A)$ thet ***st*-cut.** It is obvious that in order to find a path from s to $t,$ that path must intersection at least one of the *st*-cuts. In other words, at least one of the *st*-cuts edges must exists in the path.
Let P be a set of edges of the graph such that it contains at least one edge from every *st*-cut, then it can be proved that there must exist an st-path inside P.
From these two observations, our models consists of the decision variable x which is an array of length equal to the number of edges in the graph. If $x_i = 1,$ that means that edge i is present in the longest path. Our objective function is

$$Objective\ Function : maximize \sum_{e \in E(G)} c_e x_e$$

where $E(G)$ is the set of edges in graph G and c_e is the cost/weight on the edge e.
The constraints are

$$Constraints \sum_{e \in \delta(A)} x_e \geq 1, \ \forall A \subset V(G)$$

$$x \in \{0, 1\}$$

where $V(G)$ is the set of vertices in the graph G. Note that the first constraint is applied for all *st*-cut in the graph which contains more than just one constraint.

If we change the length of the path to be defined as the length of the largest path, we then have to use another metric for the objective function while the constraints and decision variables will remain the same. The new objective function will be

$$Objective\ Function : maximize\ max(c_e x_e)$$

Note that x_e is evaluated only for $e \in E(G)$

(b) In this formulation of TSP problem, in sequence, the salesperson visits each city x_i until all cities are visited. The tour is composed of legs where we started from a city x_i in the sequence and end at the next city x_j in the sequence. The decision variable $x_{i,j,k}$ takes a value of 1 if the edge from i to j makes up the kth leg of the tour and 0 otherwise. For this new formulation, the objective function is

$$Objective\ Function : maximize \sum_{k=1}^{n} \sum_{i,j \in E(G)} c_{i,j} x_{i,j,k}$$

where n is the number of cities. The constraints for this new formulation as

$$Constraints \sum_{k=1}^{n} \sum_{j}^{n} x_{i,j,k} = 1,\ \forall i \in n,\ j \neq i$$

$$\sum_{k=1}^{n} \sum_{i}^{n} x_{i,j,k} = 1,\ \forall j \in n,\ i \neq j$$

These are two basic constraints that ensure that a city is entered and left exactly once. Additionally we should specify that the number of the legs is equal to the number of the cities such that

$$\sum_{k=1}^{n} \sum_{i,j}^{n} x_{i,j,k} = n$$

We also should make sure that only one arc (city) can be visited during a single leg of the tour

$$\sum_{i,j}^{n} x_{i,j,k} = 1,\ k = 1, \ldots, n$$

The reverse constraint is also important for correct answer i.e., a city is only visited at most only one time (during one leg)

$$\sum_{k}^{n} x_{i,j,k} \leq 1, \quad i,j = 1, \ldots, n$$

The next constraints are concerned with the subtours that might happen. First we need to make sure that the tour ends to the first city it starts at. Such constraint can be written as

$$\sum_{j}^{n} x_{C,j,1} = 1 \qquad \qquad \sum_{i}^{n} x_{i,C,n} = 1$$

where the tour starts at city C. Similar constraints have to be imposed to ensure that the starting of (k+1)-th leg is the ending of k-th leg. These constraints can be written as

$$\sum_{i}^{n} x_{i,j,k} - \sum_{l}^{n} x_{j,l,k+1} = 0, \quad j = 1, \ldots, n,\ \ k = 2, \ldots, n$$

In additional to having $x_{i,j,k}$ as binary variable. This formulation has $O(n^3)$ constraints due to the third constraints.

<u>Problem 2</u>:

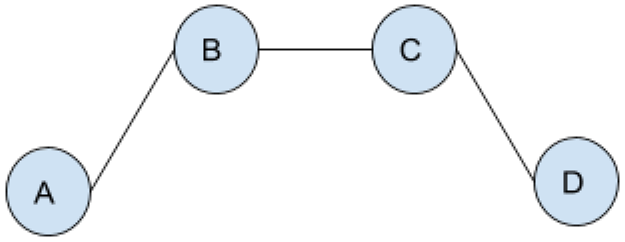
(a) A matching of a graph is a set of edges no two of which share an endpoint. It is maximal if it is not contained in another bigger matching. A vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge. Such a set is said to cover the edge of the graph.

For the sake of contradiction, assume that the maximal matching M is not a vertex cover. This means that there is a edge in $E \notin M$ such that E has two endpoints u and v and these two endpoints are not in the vertex cover. Moreover, no other edge connected to u or v that is in the matching (because otherwise u or v will be in the vertex cover and thus E will be covered). That means we can add E to the matching and it would still be valid which means it is not maximal and hence a contradiction.

(b) Let the size of the maximum matching be E. Then since the maximum matching shares no endpoints, the largest vertex cover for the maximum matching is $2E$. Thus, since any maximal matching has the same number of edges or less than the maximum matching, and any maximal matching has vertex cover of maximum size $2E$, it holds true that $V(M) \leq 2E$.

(d) The running time complexity is $O(m)$. This is because our algorithm will have to go through all the edges of the graph in order to decide if this edge should be in the match or not. The algorithm should also keep track of the vertices incident to edges in the matching such that a vertex is *marked* if it is incident to an edge in the matching. That way, when processing an edge, there will be two constant-time operations ($O(1)$) that check if the endpoints of the edge is marked or not. If not marked, the edge is added to the match and the two endpoints are marked. Otherwise the edge is discarded.

(e) Consider the graph on the right with four vertices and three edges. Using the greedy algorithm, say we start with the edge BC. Then this is a maximal matching because we cannot add any more edges that would not share endpoints with edge BC. However, we see that the maximum matching would be the matching AB and CD because we get two edges that do not share any endpoints.



Let M^* be the maximum matching of the graph. Let M be the matching returned from the greedy algorithm. If there is an edge e such that $e \in M^*,\ e \notin M,$ then there must have been another edge f that prevented adding e during the greedy algorithm such that f and e share a common endpoint. Note that there could be at most two of such conflicting f edges at the two differents ends of e. From that we can conclude that the solution of the greedy algorithm should have at least half as many edges as the true maximum matching M^*.

(f) Let G be a graph that contains V vertices and E edges. We denote the edges that are incident to a vertex v as $N(v)$. Let x_e be the binary decision variable for each edge $e \in E$ such that $x_e = 1$ if edge e is in the match and $x_e = 0$ otherwise. We get the model

$$Maximize \sum_{e \in E} x_e$$

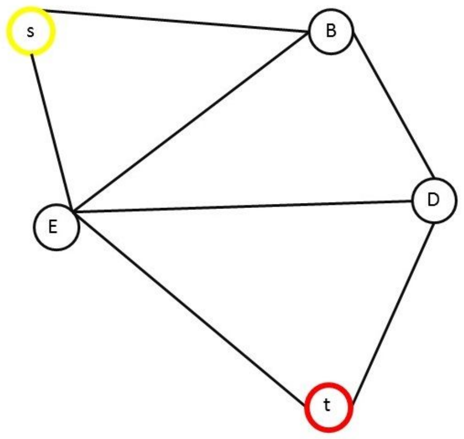
$$Subject\ to: \sum_{e \in N(v)} x_e \le 1, \ \forall v \in V$$

$$x_e \in \{0, 1\}, \ \forall e \in E$$

<u>Problem 3</u>:

First, we start with the hint given in the problem and use the augmented graph H. The augmented graph H contains two sets of vertices; first $G-t$ where the vertex t is deleted along with its incident edges, second a disjoint comput of $G-s$. We then connect the corresponding vertices of the two sets with edges of weight 0. Figure 1 shows an example of graph G and its augmented graph H. Not that H has an even number of vertices. We can see that there is an even path from s to t in G if and only if H has a perfect matching.

Here we prove that the perfect match of H will give an even path. Suppose P is an even lpath from s to t in G. We start with the edge e_1 of P incident with s and label the edge in $G-t$ graph corresponding to e_1. Then we take the edge e_2 of P following e_1 and label the edge in $G-s$ corresponding to e_2. We continue this way by alternately labeling edge in $G-t$ and $G-s$. For every node v of G not on the path P we label the new edge in H linking the two copies of v. From the construction, the labeled edges in H form a perfect matching with the same weight as P. Thus, from every perfect matching in H we can obtain an even length path in G from s to t. having the same weight by simply skipping all new edge contained in the matching. Thus, G contains an even path from s to t if and only if H contains a perfect matching.



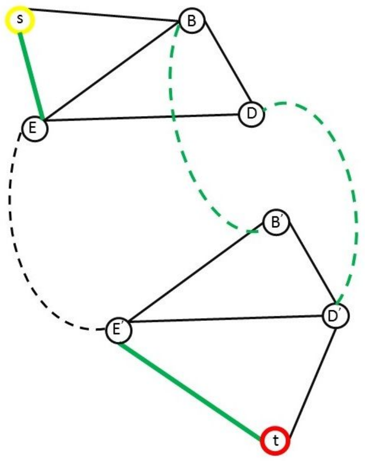


Figure (1): An example for construction of the even path. The input graph G (left) is first augmented (right) to the graph H. Then a complete match is found.

Problem 4:

- In the previous problems we dealt with 1) longest path problem, 2) TSP, 3) Matching and perfect matching 4) even path problem.

1) The longest path problem is NP-hard as it can be reduced to Hamiltonian path problem (https://en.wikipedia.org/wiki/Longest_path_problem)
2) TSP is NP-hard problem. Held–Karp algorithm can solve TSP problem in $O(N^2 2^N)$ (https://en.wikipedia.org/wiki/Travelling_salesman_problem#Computing_a_solution)
3) Perfect matching problem is an NP-complete [1]. It can be solved in $O(V^2 E)$ for general graphs via Edmonds' algorithm or in $O(V^{2.376})$ with Mucha and Sankowski algorithm. (https://en.wikipedia.org/wiki/Matching_(graph_theory))
4) Finding the even shortest path can be done in polynomial time, however for directed graphs it is NP-complete [2].

==

- The set of half-plans defining S is shown in the Figure (2). Two of the half-plans actually pass by a lattice points (shown as black dots in the figure). Only one half-plan needs to be snapped to the closest lattice point in order to define the convex hull appropriately. Thus the convex hull of S can be described by the following set of inequalities
  - $y_1 - y_2 \leq 2$
  - $3y_1 + y_2 \leq 20$
  - $y_1 + 5y_2 \leq 34$

  The extreme points of $conv(S)$ at
  - $y_1 = 5.5,\ y_2 = 3.5$
  - $y_1 = 4.71,\ y_2 = 5.85$

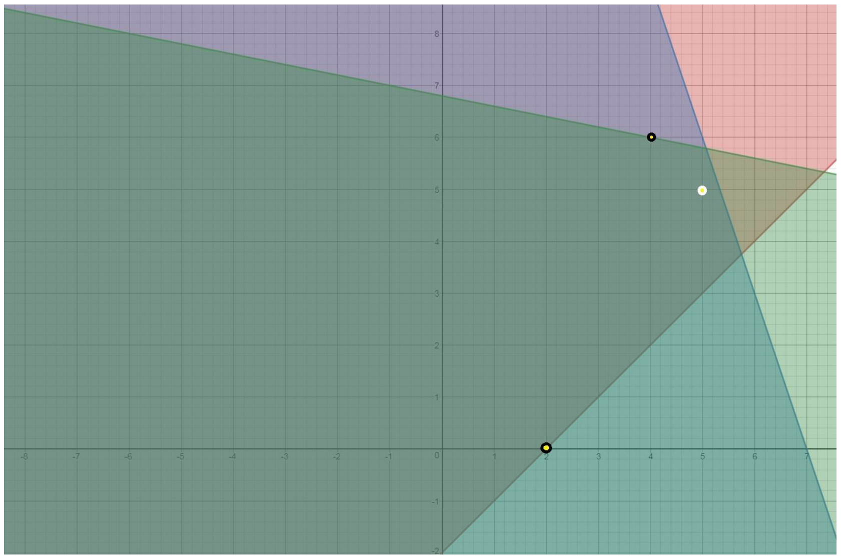
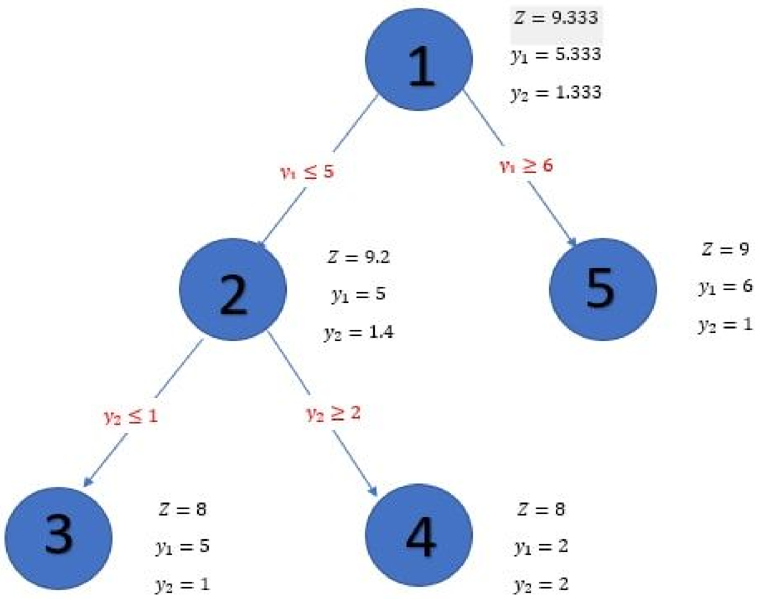


Figure (2): The inequalities of S as described in problem 4

==

- Solve the following equation using branch and bound, geographically

$$\begin{aligned} & max\ \ y_1 + 3y_2 \\ & s.t.\ \ y_1 + 5y_2 \le 12 \\ & \qquad y_1 + 2y_2 \le 8 \\ & y_1, y_2 \ge 0\ integers \end{aligned}$$



To solve the branch and bound problem. We first solve the relaxed Linear programming problem first, record the objective value and the values of each variable at the initial node in the branch and bound diagram. The optimal integer solution will be between the upper bound of the relaxed solution and a lower bound of the rounded down integer solution. By branching off, you create two constraints (from node 1 branching off to node 2 and 3 with the added constraint in red) to eliminate the fractional part of the solution value. You repeat this process until you find your optimal integer solution, with all the variables and the objective function being an integer which in this case, the graph stops branching at node 3,4 and 5 and clearly, by comparing the objective value, node 5 is the optimal integer solution for this problem. Thus, the ILP solution is $y_1 = 6,\ y_2 = 1,\ Z = 9$.

==

- Generate a valid inequality using Chvátal-Gomory cut procedure for the following problem.

$$\begin{aligned} &max\ y_1 + y_2 + y_3 = Z \\ &s.t.\ 3y_1 + 5y_2 - y_3 \le 12 \\ &\quad y_1 + 0y_2 + y_3 \le 7 \\ &\quad y_1 - y_2 + 2y_3 \le 9 \\ &y_1, y_2, y_3 \ge 0 \ integers \end{aligned}$$

We solve the following integer linear programming with simplex method which involves Tableau.

First Tableau

y_1	y_2	y_3	s_1	s_2	s_3	Z	b
0	5	-1	1	0	0	0	12
1	0	1	0	1	0	0	7
1	-1	2	0	0	1	0	9
-1	-1	-1	0	0	0	1	0

Second Tableau

y_1	y_2	y_3	s_1	s_2	s_3	Z	b
1	5/3	-1/3	1/3	0	0	0	4

0	-5/3	4/3	-1/3	1	0	0	3
0	-8/3	7/3	-1/3	0	1	0	5
0	2/3	-4/3	1/3	0	0	1	4

Third Tableau

y_1	y_2	y_3	s_1	s_2	s_3	Z	b
1	9/7	0	2/7	0	1/7	0	33/7
0	-1/7	0	-1/7	1	-4/7	0	1/7
0	-8/7	1	-1/7	0	3/7	0	15/7
0	-6/7	0	1/7	0	4/7	1	48/7

Final Tableau

y_1	y_2	y_3	s_1	s_2	s_3	Z	b
7/9	1	0	2/9	0	1/9	0	11/3
1/9	0	0	-1/9	1	-5/9	0	2/3
8/9	0	1	1/9	0	5/9	0	19/3
2/3	0	0	1/3	0	2/3	1	10

Using the final tableau, we know that the solution is, $Z = 10,\ y_1 = 0,\ y_2 = \frac{11}{3}\ , y_3 = \frac{19}{3}$

We get this result by using the simplex method, but we need to make this into an integer solution, in order to do that, let us use the row that has functions in the solutions in the final tableau to create an inequality constraint.

$\frac{7}{9}y_1 + y_2 + \frac{2}{9}s_1 + \frac{1}{9}s_3 = \frac{11}{3}$

We know that $\frac{7}{9}y_1 = 0$ *since* $y_1 = 0$ *from the solution of the simplex method*

We can manipulate this to put all of the integer parts on the left side, and all the fractional parts on the right to get

$y_2 - 3 = \frac{6}{9} - \frac{2}{9}s_1 - \frac{1}{9}s_3$

The left side of the equation is already an integer, The right hand side needs to be an integer thus, the cutting plane (the inequality constraint that would be added) is

$\frac{6}{9} - \frac{2}{9}s_1 - \frac{1}{9}s_3 \leq 0$

<u>Problem 5</u>:

(a) We start with two sequences of length one i.e, $m = 1, n = 1$ (the base of the recursive function). For such sequences, we have three possibilities: 1) the two strings matches exactly (each contain the same letter), 2) they contain different letters but we allow them to have a mismatch, 3) they contain different letters but we introduce a gap. Note that we do not align gap to a gap. From that, we can compute $c(s[0:i],\ t[0:j])$ as minimum of the three possibilities. For this two length-one strings, if they are not matching, then have a mismatch will give us the minimum; this has lower cost then introducing a gap.
For sequences of length 2, the minimal cost can be computed recursively on each of the length-one sub-sequences and add up them together. We can do the same recursive computation for longer sequences.

To give a formula for such computation, we define function $score(i,j)$ which gives zero if $s[i]$ and $t[j]$ matches, otherwise it gives 1. To better visualize the formula, we create a matrix D of size $(n+1) \times (m+1)$. We shall compute the cost recursively such that the cost of the alignment of minimal cost will be given in entry $D(n+1, m+1)$. The entry $D(i,j)$ is the maximum score of the sub-sequence $s[0:i]$ and $t[0:j]$. In order to do this, we will store the minimum score for all possible local alignments; to avoid recomputation. Thus, we have

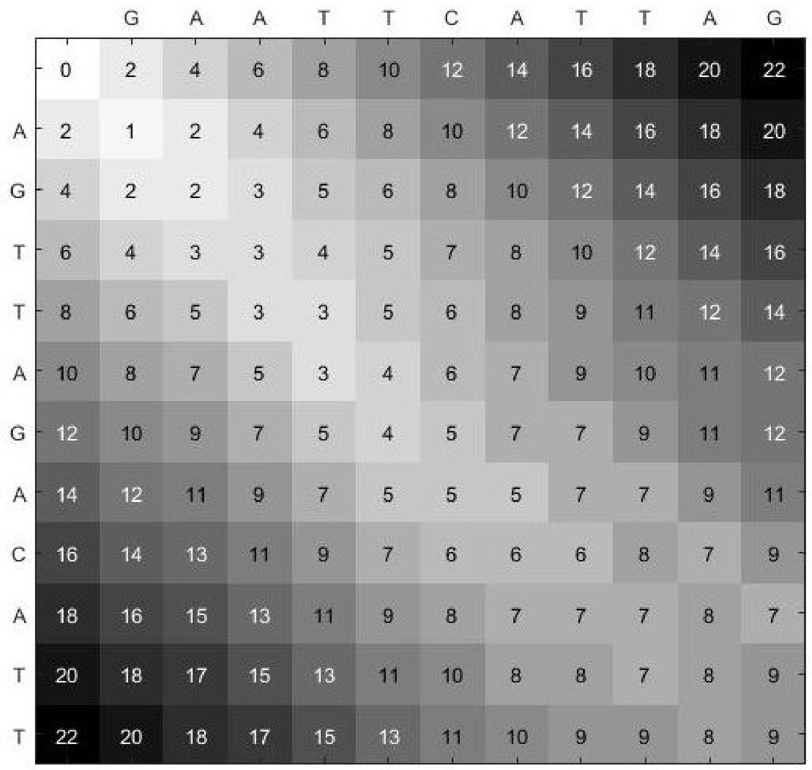
$$D(i,j) \ = \ min\{D(i-1,j-1) + score(i,j), \quad D(i-1,j) + 2, \quad D(j,j-1) \ + 2 \ \} \qquad (1)$$

The figure below shows the matrix D for an example of two short sequences. The sequences s is shown on the top of the matrix while the sequence t is shown to the left.

The equation above can be read as we take the minimum of 1) adding the score of the previous sub-sequence to $score$ 2) introducing a gap either in s or t. In order to be able to compute $D(i,1)$ and $D(1,j)$, we store in $D(0,j)$ the cost of having the sub-sequence $s[i:0]$ as gaps. Thus, the first row in the matrix would look like
$[0,\ 2,\ 4\ ,\ 6,\ 8,\ 10......,2n]$. The first column will be the same (transpose of first row).

(b) The algorithm is based on first filling in the matrix D entries and second tracing back to find the best alignment. Since the question asks for only computing $c(s,\ t)$, we ignore the trace back step. Thus, the algorithm start with the matrix D empty except for the first row and column as described in **(a)**. Starting with the second row, we compute the second entry in this row as given in equation (1). Note that for this matrix entry, all the input to equation (1) are readily available. We then move on to finish this row and move on to the third row. The process goes on until all rows has been processed. By the end of the process, the cost of the alignment of minimal cost will reside at entry $D(n+1,m+1)$.

(c/d) We implemented the algorithm above using MATLAB and the given two strings. The cost of alignment of minimal cost we get is **<u>223</u>**.
(e) The two DNA sequences have the same biological function since the cost of the optimal alignment divided by the length of the sequence is **<u>2.23%</u>** (<5%).



		G	A	A	T	T	C	A	T	T	A	G
	0	2	4	6	8	10	12	14	16	18	20	22
A	2	1	2	4	6	8	10	12	14	16	18	20
G	4	2	2	3	5	6	8	10	12	14	16	18
T	6	4	3	3	4	5	7	8	10	12	14	16
T	8	6	5	3	3	5	6	8	9	11	12	14
A	10	8	7	5	3	4	6	7	9	10	11	12
G	12	10	9	7	5	4	5	7	7	9	11	12
A	14	12	11	9	7	5	5	5	7	7	9	11
C	16	14	13	11	9	7	6	6	6	8	7	9
A	18	16	15	13	11	9	8	7	7	7	8	7
T	20	18	17	15	13	11	10	8	8	7	8	9
T	22	20	18	17	15	13	11	10	9	9	8	9

Figure (3): Example of the D matrix created to calculate the minimal cost of aligning two sequences s,t

<u>References</u>:

[1] *Lacroix, Mathieu, A. Ridha Mahjoub, Sébastien Martin, and Christophe Picouleau. "On the NP-completeness of the perfect matching free subgraph problem." Theoretical Computer Science 423 (2012): 25-29.*

[2] *Nedev, Zhivko Prodanov. "Finding an even simple path in a directed planar graph." SIAM Journal on Computing 29, no. 2 (1999): 685-695.*