CS4445/9544 Analysis of Algorithms II

Second assignment.

1. *A group L of ℓ lecturers in some university need to teach a set $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ of courses. Each course is taught once per day. For each course C_i we are given its starting time t_i and its duration d_i. Two different courses can be taught by the same lecturer if they do not overlap. We want to determine whether all the courses can be taught by the ℓ lecturers.*

   Give in pseudocode an algorithm for solving this problem. If all the courses can be taught by the lecturers, the algorithm must return the value true*; otherwise it must return the value* false*.*

   We model the problem as a maximum flow problem where a unit of flow sent from the source to the sink represents one lecturer and the path that the unit of flow follows in the flow network determines the set of courses that the lecturer is to teach. The algorithm is as follows.

   Algorithm ScheduleCourses (ℓ, C, S, D)
   In: Number ℓ of lecturers, set $C = \{C_1, C_2, \ldots, C_n\}$ of courses and sets $S = \{t_1, t_2, \ldots, t_n\}$ of starting times and $D = \{d_1, d_2, \ldots, d_n\}$ of durations.
   Out: *True* if C can be taught by the ℓ lecturers; *false* otherwise.

   (a) Build the flow network $G = (V, E)$ where $V = C \cup C' \cup \{s, s', t\}$, $C' = \{C'_1, C'_2, \ldots, C'_n\}$ and $E = \{(s, s'),\ c(s, s') = \ell\} \cup \{(s', C_i) \mid C_i \in C,\ c(s', C_i) = 1\} \cup \{(C_i, C'_i) \mid C_i \in C,\ c(C_i, C'_i) = 1$, lower bound $\boldsymbol{\ell}(C_i, C'_i) = 1\} \cup \{(C'_i, t) \mid C'_i \in C',\ c(C'_i, t) = 1\} \cup \{(C'_i, C_j) \mid C'_i \in C', C_j \in C, t_i + d_i \leq t_j,\ c(C'_i, C_j) = 1\}$. Whenever not stated, lower bounds for the edges are zero.

   (b) Compute a flow satisfying the lower bounds as follows:

   - Set $f'(C_i, C'_i) = 1$ for all $C_i \in C$ and $f'(u, v) = 0$ for all other edges.
   - Build a flow network $G' = (V', E')$, where $V' = V \cup \{s^*, t^*\}$ and $E' = \{(s^*, s),\ c(s^*, s) = \ell\} \cup \{(t, t^*),\ c(t, t^*) = \ell\} \cup \{(s^*, C'_i) \mid C'_i \in C',\ c(s^*, C'_i) = 1\} \cup \{C_i, t^*) \mid C_i \in C,\ c(C_i, t^*) = 1\} \cup (E \setminus \{(C_i, C'_i) \mid C_i \in C\})$.
   - Use the algorithm of Ford and Fulkerson to compute a maximum flow f'' for G'.
   - **if** f'' saturates all the edges of the form (s^*, C'_i) and all the edges of the form (C_i, t^*) (this means that the flow function f for G where $f(u, v) = f'(u, v) + f''(u, v)$ for every edge $(u, v) \in E$, satisfies the lower bounds) **then return** *true*
     else return *false*

   Prove that your algorithm is correct.

   Termination. First, we show that the algorithm terminates. Building the flow network G takes finite time as the number of courses is finite. Computing the flow function f' takes finite time as G has a finite number of edges. Building G' also takes finite time as C is finite. Since all capacities are integer the algorithm of Ford and Fulkerson terminates in finite time. Finally, testing whether f'' saturates the edges specified in the algorithm requires finite time.

   Feasibility. The output produced by the algorithm is either *true* or *false*, so the output is feasible.

Correctness of the output. To show that the algorithm outputs the correct value we need to show that all the courses in C can be taught by ℓ lecturers **if and only if** there is a flow function f for G that satisfies all the lower bounds.

i) All the courses in C can be taught by ℓ lecturers $\Rightarrow$ there is a flow function f for $G = (V, E)$ that satisfies all the lower bounds. Assume that there is a schedule S that assigns all the courses in C to the ℓ lecturers. In this schedule lecturer i performs a set of courses $S_i = \{C_{i1}, C_{i2}, \ldots C_{ir_i}\}$ in this order. We show how to build a flow function f that satisfies the lower bounds from this schedule S.

If the total number of lecturers used in S is $\ell' \leq \ell$ then make $f(s, s') = \ell' \leq c(s, s') = \ell$. For each set S_i of courses make $f(s', C_{i1}) = 1$, $f(C'_{ir_i}, t) = 1$, and for all $j = 1, \ldots, r_i - 1$ make $f(C_{ij}, C'_{ij}) = f(C'_{ij}, C_{i(j+1)}) = 1$. Finally, make $f(C_{ir_i}, C_{ir'_i}) = 1$. Note that if lecturer i teaches course C_{ij} followed by course $C_{i(j+1)}$ then the starting time of C_{ij} plus its duration is no larger than the starting time of $C_{i(j+1)}$ and, therefore, $(C_{ij}, C_{i(j+1)}) \in E$. The value of f is zero for all other edges in E.

The above flow function f assigns a value to every edge of G. Since all capacities are 1, except that of edge (s, s'), for which $f(s, s') \leq c(s, s')$, then f satisfies the capacity constraints. Finally, since one unit of flow is sent for each lecturer along a path from s to t flow conservation is satisfied as well.

To show that f satisfies the lower bounds, note that all courses must appear in the sets S_i, so $f(C_i, C'_i) = 1 = \ell(C_i, C'_i)$ for every $C_i \in C$. Therefore, we have shown that f is a valid flow function that satisfies the lower bounds, as required.

ii) There is a flow function f for G that satisfies all the lower bounds $\Rightarrow$ all the courses in C can be taught by ℓ lecturers. Let f be a flow function for G that satisfies all the lower bounds, then $|f| \leq \ell$ as the cut separating s from the rest of the vertices has capacity ℓ. Since f satisfies all the lower bounds, then $f(C_i, C'_i) = 1$ for all courses $C_i \in C$.

To determine from f which courses are taught by each lecturer, we use the following algorithm.

Algorithm AssignCourses(G, f)
In: Flow network $G = (V, E)$ and flow function f.
Out: Assignment of courses to lecturers

$i \leftarrow 0$
while $f(s, s') > 0$ **do** {

1. Build the network $G'' = (V, E'')$ where $E'' = \{(u, v) \mid (u.v) \in E \text{ and } f(u, v) > 0\}$.
2. Find a path p from s to t in G'' by using depth first search traversal (DFS).
3. $f(u, v) \leftarrow f(u, v) - 1$ for all edges $(u, v) \in p$.
4. Assign to lecturer i all courses C_i for which $(C_i, C'_i) \in p$.

}
return Course assignment.

Note that in the above algorithm the **while** loop is repeated $|f|$ times as each iteration decreases the value of f by 1. Since $|f| \leq \ell$, then at most ℓ lecturers are assigned courses in step 4.

Observe also that after decreasing the value of the flow on the edges in path p in Step 3 we get a new valid flow function f. Therefore, all edges (C_i, C'_i) must belong to the set P of paths chosen in Step 2 as otherwise if there is some edge (C_j, C'_j) that does not belong to any path in P then $f(C_j, C'_j) = 1$ after the algorithm terminates. But this cannot happen because when the **while** loop terminates the value of the flow is zero.

Therefore, algorithm AssignCourses assigns all courses to at most ℓ lecturers, as required.

Compute the time complexity of the algorithm.

Flow network G has $2n+3$ vertices and at most $m = 2n+1+n(n+1)/2$ edges, so building G requires $O(n^2)$ time. Computing the flow f' requires $O(n^2)$ time. Building the flow network G' also needs $O(n^2)$ time as it just has $2n+1$ more edges than G. The cut separating s^* from the rest of the vertices in G' has capacity $n + \ell \leq 2n$, as we can assume without loss of generality that $\ell \leq n$. Therefore, the algorithm of Ford and Fulkerson needs time $O(m|f^*|) = O(n^3)$. Finally, checking whether f'' saturates the required edges takes $O(n)$ time. The time complexity of the algorithm, then is $O(n^3)$.

2. *A set $F = \{f_1, f_2, \ldots, f_n\}$ of files with integer sizes $s_1, s_2, \ldots, s_n$ needs to be stored in a hard disk of capacity K. We wish to find a subset of files of maximum total size* **but not larger** *than K to be stored in the disk. For example, if we have 4 files with sizes 3, 5, 8, and 6 and $K = 15$, an optimum solution is to store in the hard drive the files of size 6 and 8. The above problem is NP-hard.*

   Consider the following algorithm for the problem. The following two questions refer to this algorithm.

   Algorithm files(F, S, n, K)
   In: Set F of files, set $S = \{s_1, s_2, \ldots, s_n\}$ of n file sizes, and hard disk capacity K
   Out: Set of files of total size at most K

   $A \leftarrow \emptyset$
   $total \leftarrow 0$
   for $i \leftarrow 1$ **to** n **do**
   if $total + s_i \leq K$ **then** {
   Add file f_i to A
   $total \leftarrow total + s_i$
   }
   return A

   (*i*) (*10 marks*) *For the above algorithm the value SOL of the solution computed by the algorithm is equal to the total size of the files in set A, i.e. $SOL = total$. Show that the approximation ratio, OPT/SOL, of this algorithm is arbitrarily large by giving an instance in which the total size of the files in the set A returned by this algorithm is very small compared to the value of an optimum solution. Note that the files are not sorted in any particular manner.*

   Consider a set $F = \{f_1, f_2\}$ of two files with sizes 1 and K. The algorithm will only put file f_1 of size 1 in the hard disk, so $SOL = 1$. The optimum solution stores f_2 in the disk, so $OPT = K$. The ratio $SOL/OPT = K/1 = K$ is arbitrarily large as K can be arbitrarily large.

(*ii*) (*20 marks*) *Assume now that the files are sorted in non-increasing order of size. Compute the approximation ratio OPT/SOL of the algorithm.*

Since the files are sorted in non-increasing order of size, then $s_1 \geq s_2 \geq \cdots \geq s_n$. Let $A = \{f_1, \ldots, f_r\}$ be the set of files selected by the algorithm. If $A = F$, i.e. all the files are stored in the hard disk, then the algorithm finds an optimum solution; otherwise there is at least one file from F not in A. Let f_t be the largest file not in A. Then,

$$\sum_{f_i \in A} s_i + s_t > K.$$

Without loss of generality we can assume that $s_1 \leq K$. Since $s_t \leq s_1$ and $f_1 \in A$ then $s_t \leq s_1 \leq \sum_{f_i \in A} s_i$ and so

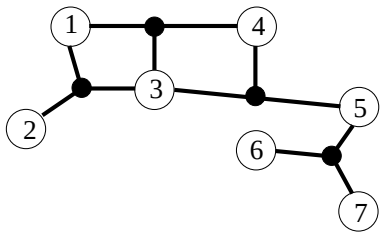
$$SOL = \sum_{f_i \in A} s_i > K - s_t \geq K - \sum_{f_i \in A} s_i.$$

Hence,

$$2 \sum_{f_i \in A} s_i > K \text{ and so } SOL = \sum_{f_i \in A} s_i > \frac{K}{2}.$$

Note that $OPT \leq K$ so $OPT/SOL < \frac{K}{K/2} = 2$.

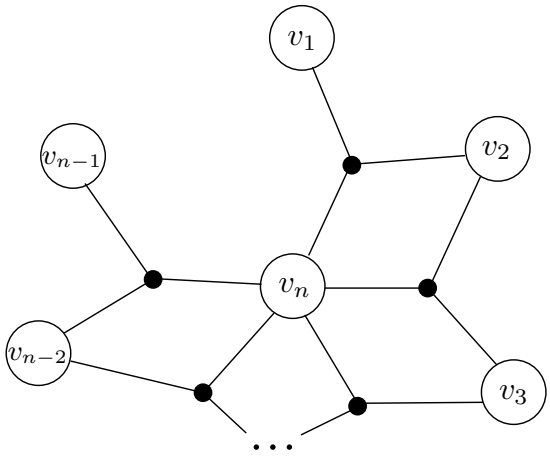
3. *In a graph an edge (u, v) is specified by its two endpoints. A* 3-hypergraph *$H = (V_H, E_H)$ consists of a set of vertices and a set of hyperedges, where a hyperedge is specified by three endpoints (u, w, v), not by two as in graphs. The minimum vertex cover problem on 3-hypergraphs is to find the smallest set S if vertices, such that every hyperedge has at least one endpoint in S. For example, for the following 3-hypergraph with hyperedges $(1, 2, 3)$, $(1, 3, 4)$, $(3, 4, 5)$, and $(5, 6, 7)$ a minimum vertex cover is $S = \{3, 5\}$.*



The minimum vertex cover problem on 3-hypergraphs is NP-hard. Consider the following generalization of the algorithm that we studied in class for the minimum vertex cover problem.

Algorithm VertexCover($H = (V_H, E_H)$)
Input: 3-hypergraph H
Output: Vertex cover of H of small size.

$S \leftarrow \emptyset$
while $E_H \neq \emptyset$ **do** {
  Choose a hyperedge $(u, v, w) \in E_H$
  $S \leftarrow S \cup \{u, v, w\}$
  Remove from E_H every hyperedge incident on u, v, or w



}
Output S

- (*20 marks*) *Compute the approximation ratio of this algorithm.*

  Let $(u_1, v_1, w_1), (u_2, v_2, w_2), \ldots, (u_k, v_k, w_k)$ be the set of hyperedges chosen by the algorithm, so $SOL = 3k$. Note that the endpoints of these hyperedges are disjoint because after choosing a hyperedge (u, v, w) the algorithm removes all hyperedges incident on u, v, or w. Hence, OPT needs to include at least one of the vertices u_i, v_i, or w_i for each hyperedge (u_i, v_i, w_i) selected by the algorithm, and so $OPT \geq k$. Therefore,

  $$\frac{SOL}{OPT} \leq \frac{3k}{k} = 3.$$

- (*10 marks*) *Suppose that in every iteration of the* **while** *loop instead of adding to S the three vertices u, v, and w we only add u to S. The idea, is that then we might be able to produce a smaller solution as we add only one vertex to S, not three of them. Compute the approximation ratio of this modified algorithm.*

  If the algorithm only puts in set S one vertex of each hyperedge selected, the set S might contain up to $n-2$ vertices. Note that after selecting $n-2$ vertices and removing all hyperedges incident on them there cannot be any hyperedges remaining as the two vertices not selected might not have a hyperedge connecting them (as each hyperedge connects 3 vertices).

  The algorithm will select $n-2$ vertices for the hypergraph $H = (V_H, E_H)$ shown in the figure, where $V_H = \{v_1, v_2, \ldots, v_n\}$ and $E_H = \{(v_n, v_i, v_{i+1}) \mid \text{ for all } i = 1, 2, \ldots, n-2\}$. The algorithm might choose vertex v_1 and remove hyperedge (v_n, v_1, v_2), then it might choose vertex v_2 and remove hyperedge (v_n, v_2, v_3), and so on, until it chooses vertex v_{n-2} and removes the last hyperedge (v_n, v_{n-2}, v_{n-1}). Therefore, $SOL = n - 2$.

  The optimum solution will choose only one vertex, v_n, so $OPT = 1$ and so $SOL/OPT \leq n - 2$.