James V. Jalwan Homework 3 CSE 202: B00

Problem 1: Graph cohesiveness (KT 7.46)

In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume for purposes of this question that friendship is symmetric, so we can consider an undirected graph.

Now suppose we want to study this graph G, looking for a "close-knit" group of people. One way to formalize this notion would be as follows. For a non-empty subset S of nodes, let $e(S)$ denote the number of edges in S-that is, the number of edges that have both ends in S. We define the *cohesiveness* of S as $e(S)/|S|$. A natural thing to search for would be a set S of people achieving the maximum cohesiveness.

1. Give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness greater than α.
2. Give a polynomial-time algorithm to find a set S of nodes with maximum cohesiveness.

Problem 1.1

Goal:

Determine whether $\exists$ a "close-knit" group of people in G

s.t.

$$\frac{\text{the \# of edges w/ both endpoints in a subset of nodes } (S) \text{ in } G}{\text{The \# of nodes in } S} > \alpha$$

Definitions:

S = a non-empty subset of nodes in G

$||S||$ = The number of nodes in S

$e(S)$ = The number of edges in S (that have both their endpoints in S)

$$\text{Cohesiveness} = \frac{e(S)}{||S||}$$

α = rational number for which want to check if $\exists$ a set S in G with cohesiveness greater than this

- As α is a rational number we can rewrite it as an integer fraction $\rightarrow \alpha = \frac{a}{b}$

a = Numerator for the reformulated α parameter

b = Denominator for the reformulated α parameter.

Key Idea:

Given an undirected friend graph G, we can create an equivalent bipartite flow graph G' with edges E and vertices V from G as vertices in G'. Moreover, this can be reformulated as a flow graph by adding a source node s and a sink node t.

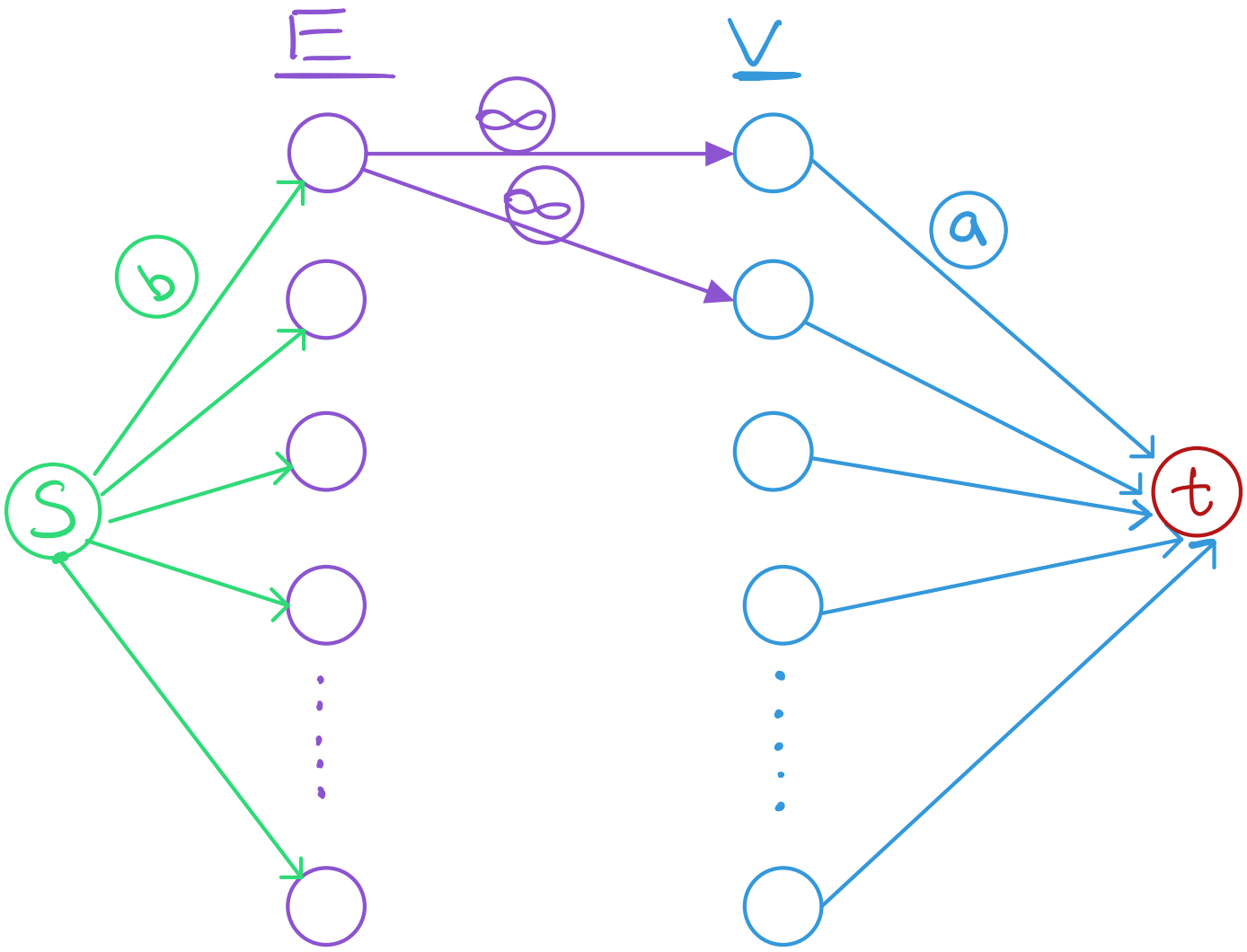
To create a problem equivalency (i.e., valid reduction) we can use the α reformulation defined above to check α cohesiveness. The existence of α cohesiveness will relate to the min cut/max flow, and the exact subset S yielding $>\alpha$ cohesiveness (if one exists) can be extracted from the residual graph R_G after the completion of the max flow algorithm.

Specifically the quantity that is being evaluated / maximized in the context of this problem is:

$$\frac{e(s)}{||s||} > \alpha = \frac{a}{b} \Rightarrow b \cdot e(s) - a||s|| > 0$$

Thus if there exist a $b \cdot e(s) - a||s|| > 0$, then there exists a set S in G with a cohesiveness $> \alpha$.

Graph Construction:



Define the above flow graph G' with the following:

$V \rightarrow$ the set of vertices in G'

$E \rightarrow$ the set of edges in G'

$V = \{s, t\} \cup \{E \mid \forall$ undirected edges e_{ij} in $G\}$

$\cup \{V \mid \forall$ vertices v_i in $G\}$

$E = \{(s, e_{ij}) \mid$ edges between the source and edge vertices in $G'\}$

with a capacity of b.

$\cup \{(e_{ij}, v_i), (e_{ij}, v_j)$ w/ $i \neq j\}$ with a conceptual capacity of ∞.

$\cup \{(v_i, t) \mid$ edges between all G vertices and the sink in $G'\}$ with a capacity of a.

Algorithm:

Create a bipartite flow graph G' from G as described above in Graph Construction. Run one of the four max flow algorithms (for selection see Time Complexity and Max Flow Algorithm Selection below). If the max flow/min cut is $< b \cdot$ the # of edges from S to E, conclude that there exists a set S in G with a cohesiveness $> \alpha$. The valid set S with cohesiveness $> \alpha$ can be extracted by finding all nodes reachable from S in $R_{G'}$ at the termination of the max flow algorithm (i.e., there are no augmenting paths in $R_{G'}$ from S to t). Specifically this can be extracted by running breadth-first search (BFS) on $R_{G'}$ from S and returning all reachable vertices in V and edges in E. In the event the max flow/min cut is not $< b \cdot$ the # of edges from S to E, return that there does not exist a set S in G with a cohesiveness $> \alpha$.

- Implementation Intricacy: The maximum flow can be checked by summing the flow out of the source node S, along its outgoing edges. Also note the # of edges from S to E are static in G' & can be computed easily as this is the bounding capacity out of S. Thus $b \cdot$ bounding capacity of S can be computed before running the max flow algorithm & can be stored to compare against the calculated max flow.

Proof of Correctness:

CLAIM 1:

If the minimum cut (/max flow) is $< \sum_{i \in E} b_i$ from s to E then $\exists$ a set S in G with cohesiveness $> \alpha$.

CLAIM 2:

The minimum cut must necessarily omit conceptual ∞ capacity edges, and thus the cut capacity will be determined by edges from s to E and V to t.

CLAIM 3:

If there exists a set S in G with cohesiveness $> \alpha$, then the set of reachable edges in $R_{G'}$ from s corresponds to a valid S satisfying the problem constraints.

- The validity of CLAIM 2 can be easily shown. Specifically there always exists a finite capacity cut that omits ∞ capacity edges (such as the cut of s from the rest of G' or t from the rest of G'). Thus no infinite edges can be in a min cut.
- CLAIM 1 and CLAIM 3 are related and relate to the problem reduction to a flow graph, and the extraction of the problem answer from running the max flow/min cut algorithm corresponds to the problem answer.

- Let A be the set of vertices in $R_{G'}$ s.t. they can be reached from the source node s.
- Let B be the complement of A, that is $B = V - A$
- Know that at the completion of max flow algorithm

$CAP(A,B) = \text{max flow}$

$$CAP(A,B) = \sum_{i \in A} b + \sum_{j \in A} a$$

Define K to be $\sum_{i \notin A} b$ if i from s to E → This is the cut capacity of $A=S$ and thus is a constant value

Observe that we can manipulate $CAP(A,B)$ by adding & subtracting $\sum_{i \in A} b$:

$$CAP(A,B) = \underbrace{\sum_{i \notin A} b + \sum_{i \in A} b}_{= K} - \sum_{i \in A} b + \sum_{j \in A} a$$

$$CAP(A,B) = K - \sum_{i \in A} b + \sum_{j \in A} a$$

$$= K - \left[\sum_{i \in A} b - \sum_{j \in A} a \right]$$

Note that by the construction of G' that

$$\sum_{i \in A} b - \sum_{j \in A} a = b \cdot e(s) - a \cdot \|s\|$$

as all edges from E to V in G were maintained and capacity unbounded (i.e., conceptual capacities of ∞)

Thus here we have:

$$CAP(A,B) = k - [b \cdot e(s) - a \cdot \|s\|]$$

- The minimum cut then occurs when the problem objective is maximized!
- Thus if the min cut $< k$, then there exists a set S in G because necessarily $b \cdot e(s) - a \cdot \|s\| > 0$!
- Finally be definition of $R_{G'}$ the set of all reachable vertices from s in this case corresponds to the edges and vertices in G that exist in S (excluding the source node s).

Time Complexity and Max Flow Algorithm Selection:

Creation of G' and the individual steps of the max flow algorithm are linear in the size of G'. Running BFS on $R_{G'}$ is also linear in the size of G'. Thus the time complexity bottleneck is the max flow algorithm selected.

Given G' as defined in Graph Construction observe the following size of G':

$$||V|| = ||E|| + ||V|| + 2$$

Define $||V|| = n$ for clarity

$$||E|| = \underbrace{||E||}_{\text{layer 1 edges}} + \underbrace{2 \cdot ||E||}_{\text{layer 2 edges}} + \underbrace{||V||}_{\text{layer 3 } (v \to t) \text{ edges}}$$

Note in the worst case G is fully connected and thus there are $\binom{n}{2}$ undirected edges in G. This then gives

$$||V|| = \frac{n \cdot (n-1)}{2} + n + 2 \approx n^2$$

$$||E|| = \frac{n \cdot (n-1)}{2} + n \cdot (n-1) + n \approx n^2$$

Using the highest order terms for both $||V||$ and $||E||$ get the following time complexities for each of the max flow algorithms.

Ford-Fulkerson: $(||V|| + ||E||) \cdot$ Max flow (or C_{Total} if unknown)

- Here this equates to $2n^2 \cdot$ Max flow
  - In the worst case we know max flow is bounded by k (defined above → occurs when there does not exist an S in G with cohesiveness $> \alpha$).
- Thus in the worst case this approach yields a runtime of $2n^2k$

Scaling: $(||V|| + ||E||)^2 \cdot \log C_{max}$

- Here this equates to $4n^4 \cdot \log(\max(b,a))$

Karp-edmunts-dinitz: $(||V|| + ||E||)^2 \cdot ||V||$

- Here this equates to $4n^6$

Preflow Push: $(||V||)^3$

- Here this equates to n^6

Given this we can discard Karp-edmunts-dinitz as it is always worse than Preflow Push by a constant factor. Thus the choice of algorithm is dependent on the connectivity of G and the given α parameter.

Know there is an upper bound to α for feasibility as

$$\frac{\binom{n}{2}}{n} = \frac{a}{b} \Rightarrow \frac{(n-1)}{2} \approx n$$

Thus we can bound scaling and Ford-Fulkerson in terms of n

Scaling $\rightarrow 4n^4 \cdot \log(\max(b,a))$ ← a is the max in worst case w/ $\binom{n}{2}$ $\approx n^2$

- Goes to $4n^4 \cdot \log n^2$

$= 8n^4 \log n$

Ford-Fulkerson $\rightarrow 2n^2 k$

$k = \sum_{i \in \text{if } i \text{ from } s \text{ to } E} b$ with $b \sim n$

- Recall in the worst case there are $\binom{n}{2}$ edges in G
- Thus $k \approx n^3$
- This gives: $2n^2(n^3) = 2n^5$

Given these time complexities, observe Scaling has the lowest order run time of the above 4 approaches. Thus select Scaling and conclude a time complexity of $O(8n^4 \log n) \rightarrow O(n^4 \log n)$.

Problem 1.2:

Key Idea:

We can use the approach developed in Problem 1.1 above to find the maximum cohesiveness in G. Specifically we can perform a binary search across α values. There are between 0 and $\binom{n}{2}$ possibilities for $e(s)$ and n possibilities for S yielding a search space of n^3.

Algorithm:

Start from the α midpoint $\left(\text{i.e., } \frac{\binom{n}{2}}{\frac{n}{2}} = \frac{n-1}{4}\right)$ and run the algorithm described above in Problem 1.1. If there exists an S with that cohesiveness, store the current S and go "right" by changing α to the midpoint between my current α and the current max right (which is initialized at the upper bound $\frac{n-1}{2}$). Set the min left here to be the previously searched α. If there does not exist an S with cohesiveness α, go "left" by changing α to the midpoint between my current α and the min left (initialized to 0). Set the max right to my previously searched α.

Continue this process until min left = max right and return the stored S (or null in the case of a completely disconnected graph G).

Time Complexity:

Performing binary search across n^3 values gives a time complexity of $O(\log n^3) \rightarrow O(3 \log n) \rightarrow O(\log n)$. Combining this with the scaling selection algorithm selected for the algorithm in Problem 1.1 yields the following time complexity:

$$O((8n^4 \cdot \log n) \cdot 3 \log n) \rightarrow O(24 n^4 \log^2 n)$$

Removing constant factors yields the final time complexity of $O(n^4 \log^2 n)$.

Proof of Correctness:

Know by the proof outlined for Problem 1.1 above that for a particular α the approach is valid. Here observe that binary search allows us to narrow the search space. Specifically the key point is that if there exists no set S with a given α, then there can exist no set S with an even greater α. Thus we can validly narrow the search space by a factor of 2 at each iteration.

Problem 2: Remote Sensors

Devise as efficient as possible algorithm for the following problem. You have n remote sensors s_i and $m < n$ base stations B_j. For $1 \leq j \leq m$, base station B_j is located at (x_j, y_j) in the two-dimensional plane. You are given that no two base-stations are less than 1 km apart (in standard Euclidean distance, $\sqrt{((x_j - x_k)^2 + (y_j - y_k)^2)}$). All base stations have the same integer bandwidth capacity C.

For $1 \leq i \leq n$, sensor s_i is located at (x_i, y_i) in the two-dimensional plane and has an integer bandwidth requirement of r_i, which can be met by assigning bandwidth on multiple base stations. Let $b_{i,j}$ be the amount of bandwidth assigned to sensor s_i on base station B_j. The assignment must meet the following constraints:

- No sensor may be assigned any bandwidth on a base station more than 2 km distance from it, i.e., if the distance from s_i to B_j is greater than 2, $b_{i,j} = 0$.
- The sum of all the bandwidth assigned to any remote sensor s_i must be at least r_i: for each $1 \leq i \leq n$, $\sum_j b_{i,j} \geq r_i$.
- The sum of all bandwidth assigned on base station B_j must be at most C: for each $1 \leq j \leq m$, $\sum_i b_{i,j} \leq C$.

Your algorithm should find a solution meeting the above constraints if possible, and otherwise output a message saying "No solution exists". Prove the correctness of your algorithm and discuss its time compplexity.

Definitions:

Let m = the number of base stations

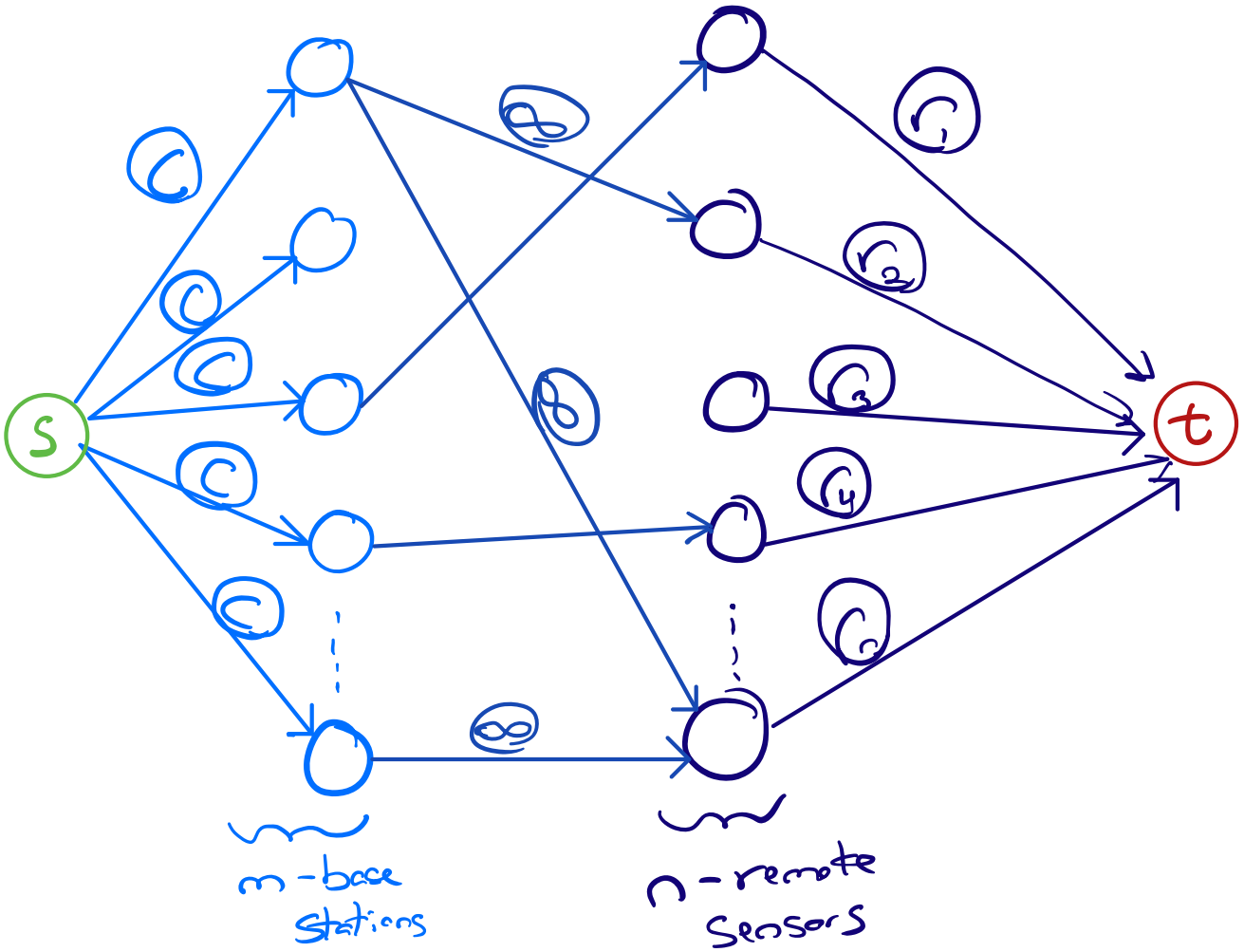
Let n = the number of remote sensors

Let C = the integer bandwidth capacity of a base station

Key Idea:

Each sensor has an integer bandwidth requirement r_i, which can be satisfied by assigning bandwidth ($\leq C$) from all sensors within a valid radius (i.e., 2 Km). The radius of base stations, coupled with the assignment nature of the problem, lends itself to the formulation of a bipartite graph, and maximizing flow from base stations to sensors.

Graph Construction:



Define the above graph G with the following:

The set of vertices V:

$$V = \{s\} \cup \{B_j \mid 1 \le j \le m\} \cup \{S_i \mid 1 \le i \le n\} \cup \{t\}$$

Edges E:

$E = \{(S, B_j) \mid 1 \leq j \leq m\}$ with capacity of C

the bandwidth integer capacity of the base stations.

$\cup \{(B_j, S_i) \mid \forall B_j \text{ w/i } 2\text{Km of a sensor } S_i\}$

with a conceptual capacity of ∞.

$\cup \{(S_i, t) \mid 1 \leq i \leq n\}$ with a capacity of r_i the integer bandwidth requirement for sensor i

Algorithm:

Create a bipartite graph G as defined above in Graph Construction. Run the Ford-Fulkerson algorithm on G to obtain the maximum flow. If the maximum flow equals $\sum_{i=1}^{n} r_i$ (i.e., the bandwidth requirements for each remote sensor S_i) then the solution of all b_{ij} that satisfies the problem constraints is the flow along the (B_j, S_i) edges. If the max flow is $< \sum_{i=1}^{n} r_i$ then return the message "No solution exists" as specified by the problem.

- Implementation Intricacy: The maximum flow can be checked by summing the flow out of the source node S, along its outgoing edges.

Proof of Correctness:

CLAIM 1:

$\exists$ an assignment of bandwidth from stations to sensors satisfying the problem constraints if max flow $= \sum_{i=1}^{n} r_i$, otherwise no valid assignment exists.

- Note the capacities of the edges (S, B_j) and (S_i, t) satisfy the upper & lower bounds of the problem constraints. Specifically, a base station can push no more than C bandwidth and a sensor must receive at least r_i bandwidth from base stations w/i 2Km from itself.
- These constraints are satisfied by the edge capacities in G & the (B_j, S_i) edges which are only constructed between base stations and a sensor w/i 2 Km of one another.
- Note that when max flow $< \sum_{i=1}^{n} r_i$, it means that the limitations occur either through bandwidth upper limits of a base station, or the lack of an edge to push bandwidth to a sensor (i.e., a base station is "out of range").
- Also note that max flow $= \sum_{i=1}^{n} r_i$ cannot occur erroneously. By setting the lower bound of a sensors bandwidth requirement as the edge capacity from (S_i, t) this ensures no

superfluous flow $> r_i$ can occur for all n sensors.
Thus only when the max flow $= \sum_{i=1}^{n} r_i$ can an assignment of bandwidth from base stations to sensors occur.

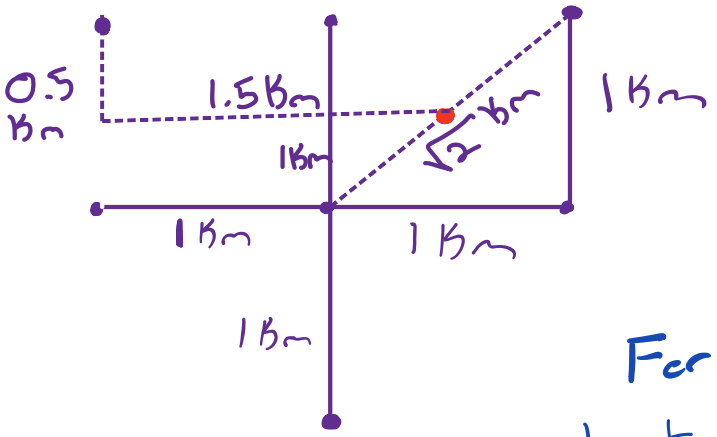
CLAIM 2:

If the max flow $= \sum_{i=1}^{n} r_i$, then the flow along (B_j, S_i) edges corresponds to the bandwidth assignment b_{ij} from base stations to sensors.

- The flow bottleneck for a base station B_j is C + given the construction of G. all the sensors to which B_j is connected can have at most flow $= C$.
- All the sensors to which B_j is connected then can have a flow that is $\leq \min(C, r_i)$ given the capacities of the edges.
- Edges (B_j, S_i) will only have flow if $\exists$ an edge (i.e., B_j is within $2K_m$ of s_i) and the Ford-Fulkerson algorithm assigns flow to an edge.
- Thus by the edge capacities, the construction of edges (B_j, S_i) and CLAIM 1, the flow along these edges corresponds to a valid assignment.

Time Complexity and Max Flow Algorithm Selection:

Note that the problem states that "no two base-stations are less than 1 Km apart". This equates to saying that for any base station, ∃ a radius of 1 w/i which no base-station can also exist. In the worst case scenario then we observe that base-stations could establish a lattice-like pattern, with a sensor placed equidistant between all sensors in a "square".



• = sensor

• = base-station

For adjacent squares note the distance equates to: $\sqrt{1.5^2+0.5^2} = \sqrt{2.5}\,Km < 2\,Km$

- For subsequent corners, observe the distance is $\frac{3\sqrt{2}}{2} > 2\,Km$

- Thus in the worst case a sensor can have an edge between itself and 12 base-stations:
  - 12 occurs from the 4 corners of my current square + 2 corners in the left, right, above, and below squares.
- Given this we know that $||V|| = n+n+2$ and $||E|| = n+n+12n = n+13n$ in the worst case.

- As we know the max flow is bounded by $\sum_{i=1}^{n} c_i$, we observe

Ford Fulkerson has a time complexity of:

$$(|V|+|E|)\cdot \text{Max Flow} = (2m+14n+2)\cdot \sum_{i=1}^{n} c_i$$

Preflow-push, Scaling, and KED are all polynomial in m and n + thus select the Ford-Fulkerson algorithm for $\sum_{i=1}^{n} c_i$

$$< \min\left((m+n+2)^3, (2m+14n+2)\cdot \min(\log C_{max}, m+n+2)\right).$$

Thus conclude that the time complexity of this algorithm is $O((2m+14n+2)\cdot \sum_{i=1}^{n} c_i) \rightarrow O((m+n)\sum_{i=1}^{n} c_i)$ by selection of Ford-Fulkerson.

Note:

Graph creation + individual steps of the algorithm are linear in the size of the graph, + thus conclude the flow algorithm above is the time complexity bottlenecks.

Problem 3: Scheduling in a medical consulting firm (KT 7.19)

You've periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they've just come to you with a new problem. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that exactly p_i doctors be present.

There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor j provides a set L_j of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists and try to return to each doctor j a list L'_j with the following properties.

(A) L'_j is a subset of L_j, so that doctor j only works on days he or she finds acceptable.

(B) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

1. Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2 \ldots, p_n$, and the lists $L_1, \ldots, L_k$, and does one of the following two things.

   - Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A) and (B); or
   - Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A) and (B).

2. The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists $L'_1, L'_2, \ldots, L'_k$ exists.

   Thus the hospital relaxes the requirements as follows. They add a new parameter $c > 0$, and the system now should try to return to each doctor j a list L'_j with the following properties.

   (A^*) L'_j contains at most c days that do not appear on the list L_j.

   (B) (Same as before) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

   Describe a polynomial-time algorithm that implements this revised system. It should take the numbers $p_1, p_2, \ldots, p_n$, the lists $L_1, \ldots, L_k$, and the parameter $c > 0$, and do one of the following two things.

   - Return lists $L'_1, L'_2, \ldots, L'_k$ satisfying properties (A^*) and (B); or
   - Report (correctly) that there is no set of lists $L'_1, L'_2, \ldots, L'_k$ that satisfies both properties (A^*) and (B).

Problem 3.1

Definitions:

P_i = The # of doctors required to work on day i

K = The # of doctors

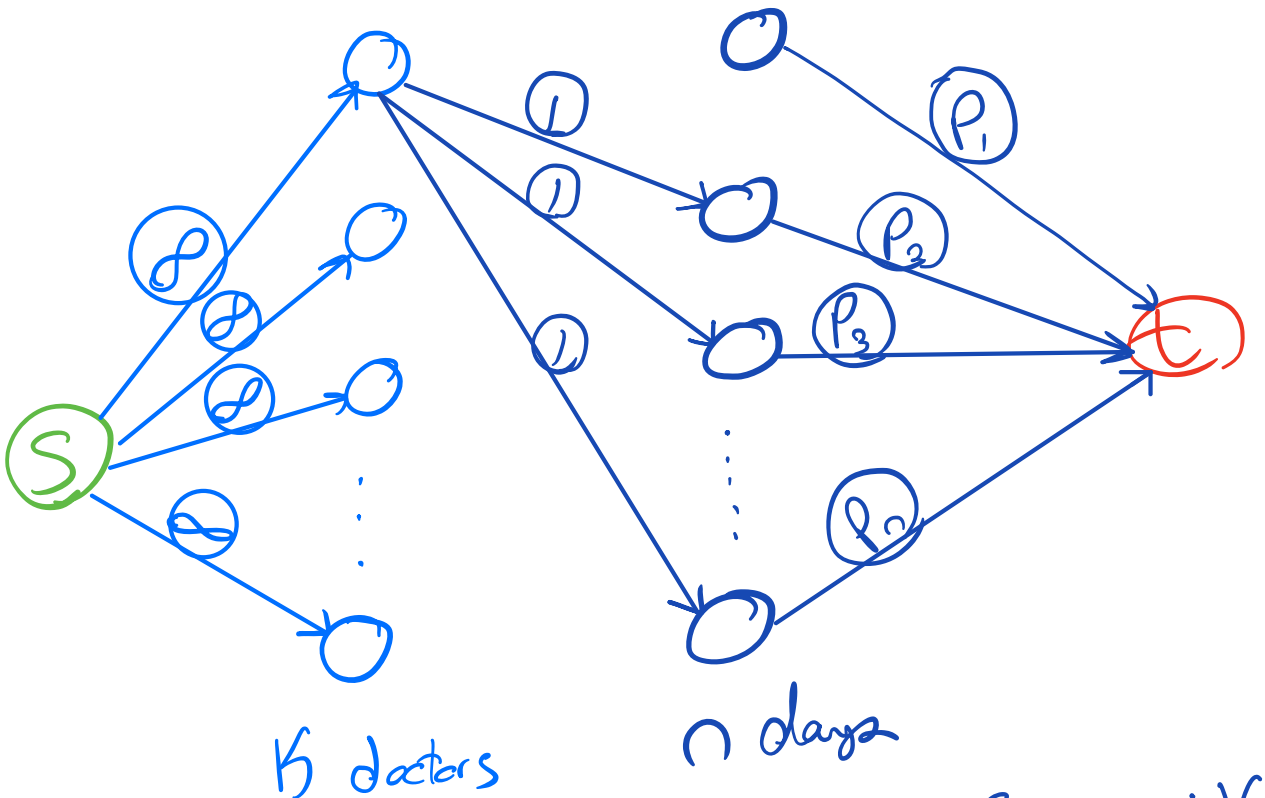
L_j = List of days doctor $j \leq K$ is willing to work

L'_j = Subset of L_j if given all $L_{1 \to K}$ the hospital can obtain P_i doctors for the next n days ($1 \leq i \leq n$)

KeyIdea:

Create a bipartite graph mapping K-doctors to the n days, with edges assigned according to each doctors L_j. If we can bound edges by capacities according to the problem constraints, we can create a flow-graph and create an assignment via flow.

Graph Construction:



Define the above graph G with the following: $V \rightarrow$ the set of vertices, $E \rightarrow$ the set of edges.

$$V = \{s, t\} \cup \{j \mid 1 \leq j \leq K\} \cup \{i \mid 1 \leq i \leq n\}$$

$E = \{(s,j) \mid \forall j \in K\text{-doctors}\}$ with a conceptual capacity of ∞.

$\cup \{(j,i) \mid \forall j \in K\text{-doctors and } i \in n\text{-days}\}$ with a capacity of 1. an edge only created if day i is in a doctors L_j

$\cup \{(i,t) \mid \forall i \in n\text{-days}\}$ with a capacity of P_i, the number of doctors required to work on day i.

Algorithm:

Create a bipartite graph G as defined above in Graph Construction. Compute the max flow using 1 of the 4 max flow algorithms (for selection see Time Complexity and Max Flow Algorithm Selection below). If the max flow $= \sum_{i=1}^{n} P_i$, then create $L'_j \; \forall j \in K$ doctors. Specifically these L_j equate to the edges from doctor j to all days in L_j with non-negative flow (which here will be 1). If the max flow $< \sum_{i=1}^{n} P_i$, then report that there is no set of lists $L_1', L_2', \ldots L_K'$ that satisfy the problem constraints.

- Implementation Intricacy: The maximum flow can be checked by summing the flow out of the source node S, along its outgoing edges.

Proof of Correctness:

CLAIM 1:

$\exists$ an assignment $L_1', L_2', \ldots L_k'$ from the lists $L_1, L_2, \ldots L_k$, if the maximum flow through $G = \sum_{i=1}^{n} p_i$. Otherwise no assignment exists.

- There are 2 capacity bottlenecks in G:
  1) Capacity (p_i) for all $i \in n$-days on (i, t) edges
  2) Capacity (1) $\forall$ (j, i) edges, which were created only if doctor j is willing to work on day i (i.e., day i is in L_j)
- Given this, (j, i) edges establish whether or not a doctor can work on day i, contributing at most 1 unit of flow.
- For a day i, there needs to be p_i units of flow = $\sum_{(j,i)} 1$ the incoming flow. This ensures that to every (j, i) edge with flow (note the flow along these edges can only be 0 or 1 based on these edge capacities) equates to 1 doctor working on day i. Thus all edges with flow must = p_i for all n days for there be to a valid assignment $L_1', L_2', \ldots L_k'$.
- The capacities on (i, t) edges ensure we cannot overflow a particular (i, t) edge ensuring that when maxflow = $\sum_{i=1}^{n} p_i$ every (i, t) edge is saturated with exactly p_i flow.

- Note: This also effectively constrains the maximum flow in G to be $\sum_{i=1}^{n} p_i$

- When maxflow $< \sum_{i=1}^{n} p_i$, that equates to the fact that there is at least one day where p_i doctors cannot work.

CLAIM 2:

If the maximum flow $= \sum_{i=1}^{n} p_i$, then all (j, i) edges with flow correspond to a valid assignment. Specifically, for any doctor j, these edges to days i, with flow correspond to $L'j$.

- By CLAIM 1, know that flow can only $= \sum_{i=1}^{n} p_i$ when there is an assignment.
- Since flow is bounded to be either 0 or 1 on these edges, by the capacity (assuming p_i is an integer... hard to have half a doctor work on a day), all edges with flow (i.e., flow = 1 since flow will be pushed along edge (i, t) in discrete units of 1.

- Note: If p_i is not an integer, assume hospital needs a partial shift. In such a case all individual (j, i) edges w/ flow = 1 equate to full-shifts and all (j, i) edges with flow > 0 & < 1 equating to partial shifts (i.e., 0.2 flow equates to a 20% shift)
  - This would necessitate obtaining a least common multiple of all capacities to convert all capacities in G to integer, before scaling the flow back by such a multiple.

- Note that when flow = 0 in the existence of a valid assignment, this corresponds to a day i that is saturated w/ P_i doctors, & thus no more doctors need to be assigned to day i. Thus doctors only work on flow (i, i) edges.

Time Complexity and Max Flow Algorithm Selection:

Creation of G and the individual steps of the max flow algorithm are linear in the size of the graph. Thus the max flow algorithm selection is the time complexity bottleneck.

Note in the worst case, every doctor could work every day creating the following size of G:

$$||V|| = K + n + 2$$

$$||E|| = K + n + Kn$$

- Given the dense-ness of $||E||$ in this case, omit algorithms non-linear in E (i.e., Scaling and KED).
- Preflow-push gives: $O((K+n+2)^3)$ which contains not only Kn, but also $Kn^2 + K^2n$
- Ford-Fulkerson gives $O((2K+2n+Kn+2)\sum_{i=1}^{n} P_i)$, where $\sum_{i=1}^{n} P_i$, the max possible flow is upper bounded by Kn (i.e., the number of doctors · the number of days; if any $P_i > K$ can terminate early as know we cannot assign more doctors than we have available.)

max possible flow

Substituting Kn to $\sum_{i=1}^{n} P_i = Kn$ to the Ford-Fulkerson algorithm get a time complexity of:

$$O((2K+2n+Kn+2)\underbrace{\sum_{i=1}^{n} P_i}_{Kn}) = O(2K^2n + 2Kn^2 + K^2n^2 + 2Kn)$$

Compare this to pre-flow push which has K^3, n^3, Kn^2, K^2n terms, but its exponents are bounded by 3. Thus in the worst case where $K=n$ the highest order term in Ford-Fulkerson is n^4 vs. in pre-flow push it is n^3.

Thus we select the pre-flow push algorithm here with a time complexity of $O((K+n+2)^3) \simeq O((K+n)^3)$.

Problem 3.2:

Definitions:

P_i, K, L_j are defined in the same manner as in Problem 3.1 above.

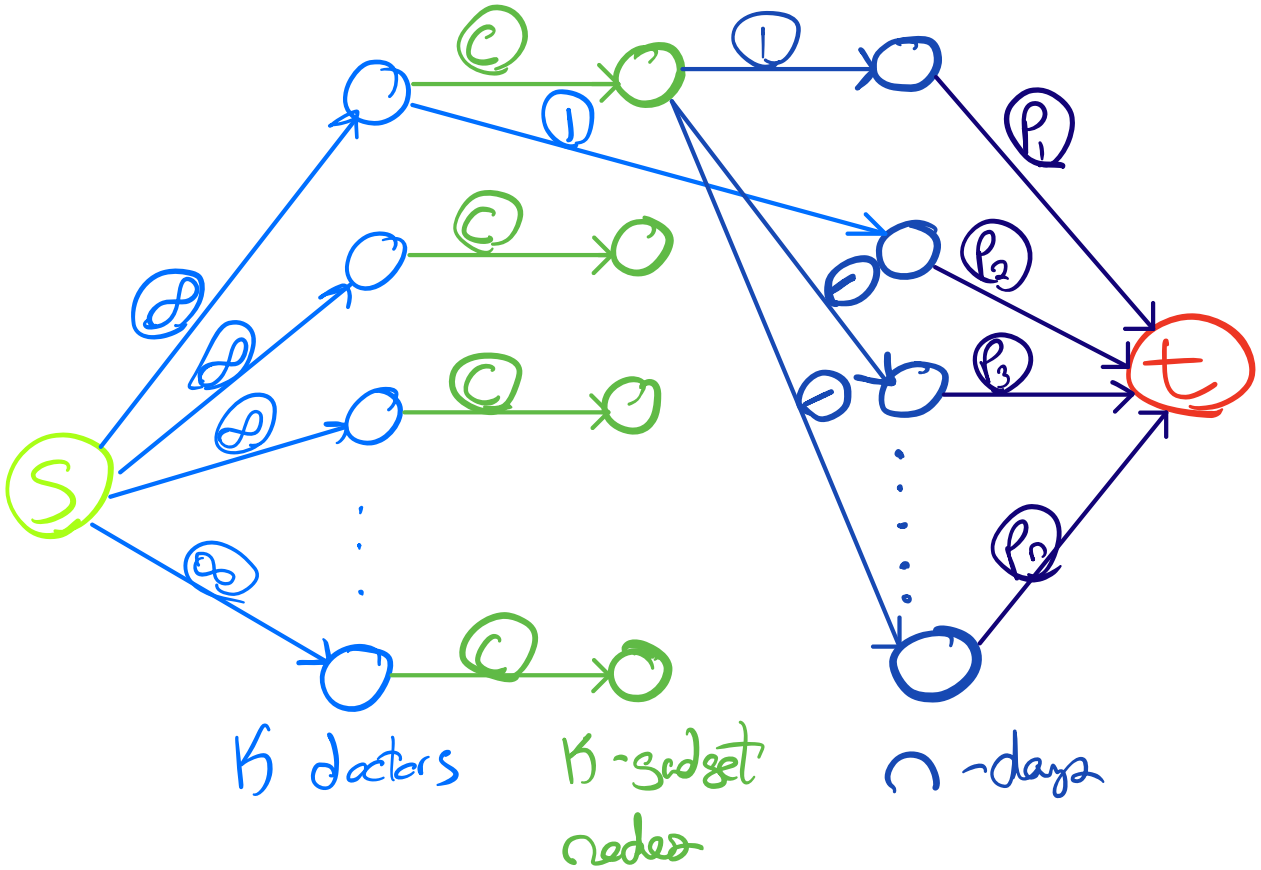
C = flexibility parameter > 0 s.t. given a doctors list L_j, doctor j can be assigned to work no more than C days outside of his/her L_j.

L_j' = Subset of L_j + at most c days not in L_j if given all $L_1 \to K$ the hospital can obtain P_i doctors for the next n days.

Key Idea:

Introducing the flexibility parameter C introduces a new constraint that needs to be accounted for if attempts to build on flow-approach from Problem 3.1 above. To account for this constraint we can introduce another layer of k "gadget" nodes with an inbound capacity of C.

Graph Construction:



Define the above graph G with the following: $V \rightarrow$ the set of vertices, $E \rightarrow$ the set of edges.

$V = \{s, t\} \cup \{j \mid j \in K\text{-doctors}\}$ ← doctor nodes

$\cup \{g_j \mid j \in K\text{-doctors}\}$ ← gadget nodes

$\cup \{i \mid i \in n\text{-days}\}$

$E = \{(s, j) \mid \forall j \in K\text{-doctors}\}$ with a conceptual capacity of ∞.

$\cup \{(j, g_j)$ for all K-doctors, s.t. there is only one edge between doctor j and gadget j (the indegree of g_j will only be 1 with the lone edge coming from doctor j)$\}$ with a capacity of C.

$\cup \{(j, i) \mid \forall j \in K\text{-doctors}$ and $i \in n\text{-days}$ with an edge only created if day i is in a doctors $L_j\}$ with a capacity of 1.

$\cup \{(g_j, i) \mid \forall i\text{-days}$ for doctor j for which no (j, i) edge exists$\}$ with a capacity of 1.

$\cup \{(i, t) \mid \forall i \in n\text{-days}\}$ with a capacity of P_i, the number of doctors required to work on day i.

Algorithm:

Create a bipartite graph G as defined above in Graph Construction. Compute the max flow using 1 of the 4 max flow algorithms (for selection see Time Complexity and Max Flow Algorithm Selection below). If the max flow $= \sum_{i=1}^{n} p_i$, then create L_j' $\forall$ $j \in k$ doctors. Specifically these L_j' equate to the edges from doctor j to all days in L_j with non-negative flow (which here will be 1) plus all the edges from doctor j's corresponding gadget node to all i days w/ which it is connected with non-zero flow. If the maximum flow $< \sum_{i=1}^{n} p_i$, then report that there is no set of lists $L_1', L_2', \ldots L_k'$ that satisfy the problem Constraints A* and B.

- Implementation Intricacy: The maximum flow can be checked by summing the flow out of the source node S, along its outgoing edges.

Proof of Correctness:

CLAIM 1:

$\exists$ an assignment $L_1', L_2', \ldots L_k'$ from the lists $L_1, L_2, \ldots L_k$, if the maximum flow through $G = \sum_{i=1}^{n} p_i$. Otherwise no assignment exists.

- There are 3 capacity bottlenecks in G:

1) Capacity (p_i) for all $i \in n$-days on (i, t) edges

2) Capacity (1) $\forall$ (j, i) edges, which were created only if doctor j is willing to work on day i (i.e., day i is in L_j). Similarly there are capacities of (1) $\forall$ (g_j, i) edges, which were created only on days for which doctor j was unwilling to work.

3) Capacity (c) $\forall$ (j, g_j) edges, the flexibility parameter for doctors which cannot be exceeded.

- Know that Capacity bottleneck 1) above ensures a prevention of flow oversaturation. Specifically the capacity of p_i here ensures at most p_i units can flow along a particular $i \rightarrow t$ path. Thus if flow equals $\sum_{i=1}^{n} p_i$ then for each day i, the flow along its corresponding edge in G is exactly p_i corresponding to the problem constraints.

- The capacities on (i, t) edges ensure we cannot overflow a particular (i, t) edge ensuring that when maxflow $= \sum_{i=1}^{n} p_i$ every (i, t) edge is saturated with exactly p_i flow.

- Similarly note that capacity bottleneck 2) above ensures a 1-1 doctor to day amount of flow. From any gadget or doctor node the flow is upper bounded by 1, and thus every edge here with flow contributes 1 unit to p_i, which equates to the problem, every doctor counts once to the sum p_i (assuming no fractional requirements; if desired this can be accounted for → see note in Problem 3.1 above).
- Capacity Constraint 3) is needed to satisfy the flexibility requirement of the problem & is justified in the reasoning for CLAIM 2 & below. However note that all paths that flow through these gadget nodes can have at most one-unit of flow as all outgoing edges from g_i have a capacity of 1 as stated in the above point.
- When $\text{maxflow} < \sum_{i=1}^{n} p_i$, that equates to the fact that there is at least one day where p_i doctors cannot work. This occurs when all incoming edges to i are capacity saturated by C. Specifically since there are k edges incident on all n days, all non-gadget edges will be assigned flow to day i, & thus this necessarily means that capacity bottleneck 3) caps the flow to day i as all g_i → i edges are possible, but no flow runs through any of them, & thus no assignment exists that satisfies the problem constraints A^* & B.

CLAIM 2:

If the maximum flow $= \sum_{i=1}^{n} p_i$, then all (j, i) and (g_i, i) edges with flow correspond to a valid assignment of doctor j to i days, corresponding to their L_j's.

- For a valid assignment to exist all i on-days, must have p_i doctors assigned on day i
  - By CLAIM 1, know that flow can only $= \sum_{i=1}^{n} p_i$ when there is an assignment.
- For a valid assignment be exist, doctors should only be assigned to days in their corresponding list, L_j, and at most c-days outside that list.
  - By the capacity constraint 2) know that the flow through edges outlined in capacity constraint 2) account to an assignment as flow is upper-bounded by 1 along these paths (for more detailed reasoning see above).
  - While this is valid, need to ensure an excess of flow cannot run through the gadget nodes. If this were to occur this would violate the flexibility constraint imposed by the problem.
    - Capacity Constraint 3) ensures that at most c units of flow can run through a gadget node. This ensures the flexibility constraint is adhered to and thus by conservation of flow the exiting flow cannot exceed the incoming flow which is bounded by c.
- Thus a valid assignment can be obtained as outlined by this Claim.

Time Complexity and Max Flow Algorithm Selection:

Creation of G and the individual steps of the max flow algorithm are linear in the size of the graph. Thus the max flow algorithm selection is the time complexity bottleneck.

Given the creation of the graph observe the following:

$||V|| = 2k + n + 2$ ← gadget + doctors + days + source + sink

$||E|| = \underbrace{k}_{\text{source to doctors}} + \underbrace{kn}_{\text{doctors to gadget}} + \underbrace{kn}_{\text{gadgets to days + doctors to days}} + \underbrace{n}_{\text{days to sink}} = k(2+n) + n = 2k + n + kn$

Note this is within constant bounds of the number of vertices + edges in Problem 3.1 above with the following:

$$||V|| = k + n + 2$$

$$||E|| = k + n + kn$$

Thus by the same reasoning outlined in the justification stated in Problem 3.1, select pre-flow push here yielding a time complexity of: $O((2k + n + 2)^3) \approx O((k+n)^3)$

Problem 4: Cellular network

Consider the problem of selecting nodes for a cellular network. Any number of nodes can be chosen from a finite set of potential locations. We know the cost $c_i \geq 0$ of establishing site i. If sites i and j are selected as nodes, then we derive the benefit b_{ij}, which is the revenue generated by the traffic between the two nodes. Both the benefits and costs are non-negative integers. Find an efficient algorithm to determine the subset of sites as the nodes for the cellular network such that the sum of the benefits provided by the edges between the selected nodes less the selected node costs is as large as possible.

Design an efficient polynomial-time algorithm.

Provide a high-level description of your algorithm, prove its correctness, and analyze its time complexity.

Goal:

Find the subset of nodes to maximize profits. Specifically

$$\max_{S \subseteq \text{Cellular nodes}} \left(\sum_{i,j \in S} b_{i,j} - \sum_{i \in S} c_{is} \right)$$

Definitions:

c_i = the costs of establishing site i with an integer cost ≥ 0.

b_{ij} = benefit derived by the traffic between nodes i & j

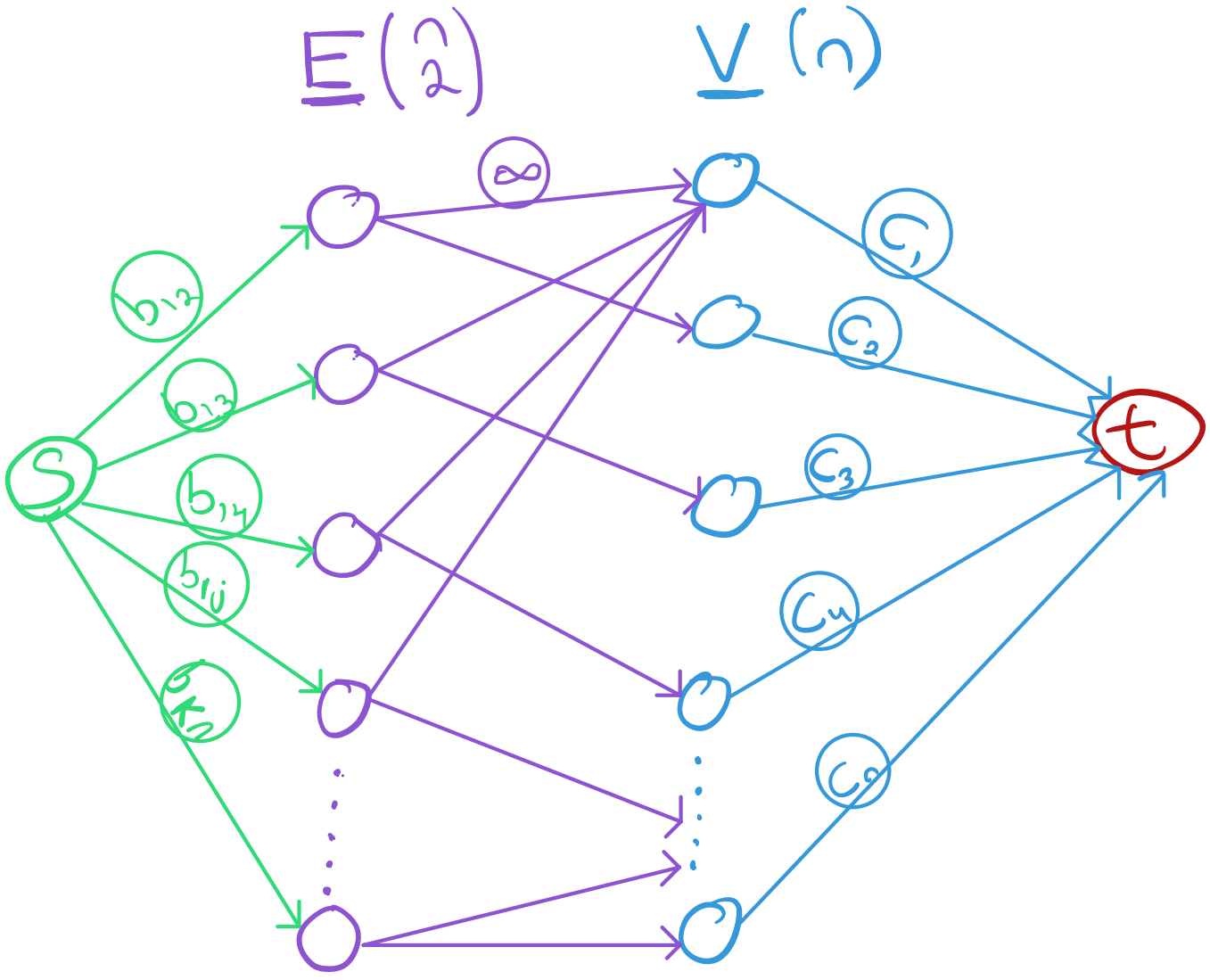
- This benefit is a non-negative integer

Profit = Sum of all benefits across all pairs of established nodes − the sum of the costs of establishing all the selected nodes

Key Idea:

Cellular nodes create an undirected graph, which we can convert to a bipartite graph by conceptualizing undirected edges as vertices. Moreover, by assigning benefits and costs correctly in the context of a flow graph & running a max flow algorithm, the subset of nodes to select that maximizes profits equate to the nodes reachable from S in $R_G \Rightarrow$ min cut.

Graph Construction:



Define the above graph G with the following: V → the set of vertices, E → the set of edges in the bipartite graph

$V = \{ S, t \} \cup \{$ all (finite) potential locations for cellular node site$\}$ ← define the number of sites as n
← this is V in the above figure

$\cup \{$ all (i,j) edges that exist in the undirected graph$\}$

- Note that there are $\binom{n}{2}$ edges in the undirected graph thus this term contributes $\binom{n}{2}$ vertices

- This is E in the above figure

$E = \{(s, e_{ij})$ for all edges that exist in the undirected graph$\}$ with a capacity of b_{ij} → the benefit derived from sites i and j

$\cup \{(e_{ij}, v_i), (e_{ij}, v_j)\}$ with a conceptual capacity of ∞.

- This states that each e_{ij} only connects to its corresponding vertices v_i and v_j. Note $i \neq j$.

$\cup \{(v_i, t)\ \forall\ n$ potential locations$\}$ with a capacity of c_i, the cost of establishing site i.

Algorithm:

Create a bipartite graph G as defined above in Graph Construction above. Run one of the four max flow algorithms (for selection see Time Complexity and Max Flow Algorithm Selection section below). The maximum profit that can be obtained equates to the min cut (which equates to the max flow by duality) on G. The subset of sites as nodes for the cellular network equate to the subset of nodes reachable from s in the Residual graph R_G, at the time max flow is reached in G (i.e., when there are no more Augmenting Paths in R_G). In R_G, these sites can be determined by running BFS from s and returning all reachable sites (i.e., v_i).

Time Complexity and Max Flow Algorithm Selection:

Creation of G and the individual steps of the max flow algorithm are linear in the size of G. Running BFS on R_G is also linear in the size of G. Thus the time complexity bottleneck is the max flow algorithm selected.

Given G as defined in Graph Construction observe the following size of G:

$$||V|| = \binom{n}{2} + n + 2 = \frac{n \cdot (n-1)}{2} + n + 2 \approx n^2$$

$$||E|| = \binom{n}{2} + 2n + n = \frac{n \cdot (n-1)}{2} + 3n \approx n^2$$

Using the approximation of both $||V||$ and $||E||$ as their highest order terms n^2 we observe the following time complexities for each of the max flow algorithms

Ford-Fulkerson: $2n^2 \cdot$ max flow (or min cut)

- Without a clear upper bound on max flow here we can use the less tight bound $C_{total} = \sum_{e \in E} C_e$ for all non-infinite capacity edges.

Scaling: $(n^2 + n^2)^2 \log C_{max} = 4n^4 \log C_{max}$

- C_{max} is the maximum non-infinite capacity in the graph G here

Karp-Edmunds-Dinitz: $(n^2+n^2)^2 n^2 = 4n^6$

Preflow push: $(n^2)^3 = n^6$

Thus the choice of algorithm here depends on C_{max} and C_{total}. Note we can discard KDE as it has a worse time complexity than preflow push by a constant factor. For the 3 remaining algorithms, observe they all share an n^2 term, so need to look at the remaining defining characteristics. Specifically the minimum among C_{total}, $4n^2 \log C_{max}$, and n^4 determine which method is selected. If C_{total} is the minimum, select Ford-Fulkerson. If $4n^2 \log C_{max}$ is the minimum, select Scaling. If n^4 is the minimum select preflow push.

Given the dependency on the relationship between the number of sites, costs, and benefits, default to the lowest order polynomial algorithm: Ford-Fulkerson. This gives a time complexity of $O(2n^2 \cdot C_{total}) \rightarrow O(n^2 C_{total})$.

Proof of Correctness:

CLAIM 1:

Given G, the subset of sites extracted from the minimum cut in R_G, equates to the subset of sites that maximize profit.

- Let A be the set of vertices in R_G s.t. they can be reached from the source node s.
- Let B be the complement of A, that is $B = V - A$
- Know that at the completion of max flow algorithm $CAP(A,B) = \text{max flow}$
- $CAP(A,B) = \sum_{ij \notin A} b_{ij} + \sum_{i \in A} c_i$ ⟸ note omission of ∞ capacity edges. See CLAIM 2 below.

Define $b = \sum_{\forall ij} b_{ij}$ ← this equates to the cut capacity of just $A = S$, and is a constant value.

Rearrange $CAP(A,B)$ by adding and subtracting $\sum_{ij \in A} b_{ij}$:

$$CAP(A,B) = \underbrace{\sum_{ij \notin A} b_{ij} + \sum_{ij \in A} b_{ij}}_{b} - \sum_{i,j \in A} b_{ij} + \sum_{i \in A} c_i$$

$$CAP(A,B) = K - \sum_{i,j \in A} b_{ij} + \sum_{i \in A} c_i$$

$$= K - \underbrace{\left[\sum_{i,j \in A} b_{ij} - \sum_{i \in A} c_i \right]}_{\text{profit}}$$

Thus $CAP(A,B)$ is minimized when profit is maximized, and thus the min cut and all nodes in A correspond to the subset of sites that maximize the profit.

CLAIM 2:

For any min cut that occurs, there must necessarily be no cut edges with ∞ capacity.

- In this case a cut will either occur across edges from s or inband to t.
- This can be shown by observing that there always exists a finite capacity cut in G by selecting from these edges + always omitting ∞ capacity edges.

- This Claim has important implications for the cellular site selection. Specifically that all projects selected in A have at least one path from s in R_G + thus their cost is the flow bottlenecks.
  - Compare this to the case where benefit is a flow bottleneck $\rightarrow$ Cost must necessarily $\geq$ benefit.

CLAIM 3:

Given the cellular node site selection project, it can be reduced to a flow graph G, that satisfies the constraints of the problem.

- The capacity bottlenecks equating to pairwise node benefits and costs, coupled with unbounded capacity edges between E and V yield the maximum profit after running the selected max flow.
- The detailed reasoning for this is shown above in CLAIM 1.