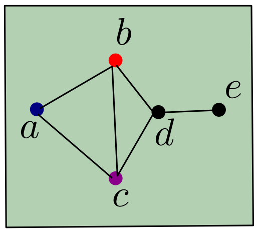
Maximum Flow and its Applications: Part 3

Saladi Rahul, Sept. 12th and 14th 2023

Having done all the work to come up with a polynomial time algorithm to compute maximum flow (and hence, minimum cut), it is time reap the benefits. We will look at few problems some of which at first sight seem to have no connection to a flow network! However, via clever reductions these problems can be solved by mapping to a flow network. We will assume that maximum flow can be computed in $O(|E||V|)$ time via Orlin's algorithm.

1 Application 1: densest subgraph problem



The densest subgraph problem is a well studied problem in the database community and the theoretical computer science community. The input is an *undirected* graph $G = (V, E)$. Define *density* of a subset $S \subseteq V$ as $d(S) = \frac{|E(S)|}{|S|}$, where $E(S)$ is the number of edges among vertices in S. The goal is to report the subset of the vertices with the largest density. In the figure shown on the right, the densest subgraph is $\{a, b, c, d\}$ with density $5/4$.

Densest subgraph problem has a lot of real-world applications.

- Consider the friendship graph in a social media platform such as Facebook or Instagram. Each user is a vertex and there is an edge between two vertices if the corresponding users are friends (or follow each other) in the app. Finding dense subgraphs in the friendship graph can be useful for the app in recommending ads: if 100 users form a dense subgraph and if an ad was clicked by say 10 of them, then it is quite likely that the remaining 90 might have similar interests and hence, show interest in that ad.

- Finding dense subgraphs is also useful in detecting spam or collusion among users. For example, the famous Page Rank algorithm considers a webpage W to be important if many other websites have a link to W. So, lets say all the students in this course create their websites and in order to increase their visibility decide to link them to each other's webpage; such a collusion can be detected by finding dense subgraphs :)

Differences in both the problems. At first look, the maximum flow and the densest subgraph problem do not seem to be much related. The densest subgraph problem deals with an undirected graph, whereas the flow network is a directed graph. Apparently we do not see a source and a sink vertex, nor do we see edge capacities in the graph for the densest subgraph problem.

There are, however, some similarities. Both of them are maximization problems. More importantly, another way to look at the densest subgraph problem is that it is asking for a partition of the vertices into two sets which is similar to an (s,t)-cut in a flow network. Now we show the reduction.

Reduction. Recall that the input to the densest subgraph problem is an undirected graph $G = (V, E)$. Define $E(S, V \setminus S) \subseteq E$ to be the set of edges such that one of the vertex is in S and the other vertex is in $V \setminus S$. Also, define $deg(v)$ as the degree of vertex v in G. Finally, $m = |E|$.

Exercise. Prove that $|E(V \setminus S)| = \sum_{v \in V \setminus S} \frac{deg(v)}{2} - \frac{|E(S,V \setminus S)|}{2}$, where $E(V \setminus S)$ is the set of edges among vertices in $V \setminus S$.

We will first solve a *decision problem*. Given an integer β, is the densest subgraph's density greater than or equal to β or less than β. We will need the follow lemma to start things off. The utility of the lemma will become clear later.

Lemma 1. $d(S) > \beta \iff \frac{|E(S)|}{|S|} > \beta \iff |S|\beta + \sum_{v \in V \setminus S} \frac{deg(v)}{2} - \frac{|E(S,V \setminus S)|}{2} < m.$

Proof.

$$\begin{aligned}
& d(S) < \beta \\
& \iff |S|\beta - |E(S)| < 0 \\
& \iff |S|\beta - (m - |E(S, V \setminus S)| - |E(V \setminus S)|) < 0 \\
& \iff |S|\beta + |E(S, V \setminus S)| + |E(V \setminus S)| < m \\
& \iff |S|\beta + \sum_{v \in V \setminus S} \frac{deg(v)}{2} - \frac{|E(S, V \setminus S)|}{2} < m \qquad \text{(from the above exercise).}
\end{aligned}$$

□

Now construct a flow network G' as follows. The vertices in G' will be $V' = V \cup \{s, t\}$. Place an edge from s directed towards each vertex $v \in V$ with capacity $\frac{deg(v)}{2}$. Place an edge from each vertex $v \in V$ directed towards t with capacity β. For each edge $e = (u, v) \in E$, places two directed edges (u, v) and (v, u) in G' with capacity $1/2$. See Figure 1.

The *main punchline* is the following: If there is a subset $S \subseteq V$ such that $d(S) > \beta$, then the minimum cut in G' will be less than or equal to m. Such a cut can be obtained by placing the vertices in S on the side of s and the remaining vertices $V \setminus S$ on the side of t. The following lemma proves it precisely.

Lemma 2. *If there is a subset $S \subseteq V$ such that $d(S) > \beta$, then the minimum cut in G' is less than m. On the other hand, if for every subset $S \subseteq V$, we have $d(S) \leq \beta$, then the minimum cut in G' will be equal to m.*

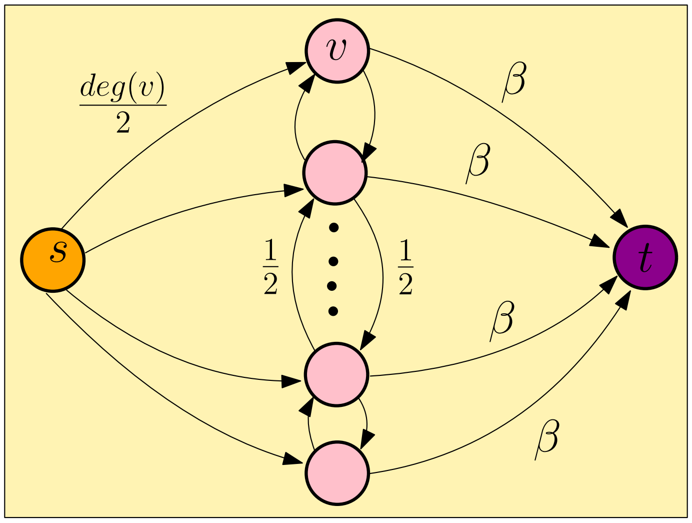


Figure 1: The flow network constructed for the densest subgraph problem.

Proof. Consider any subset $S \subseteq V$. Then consider a partition of vertices in G' into two sets $S' = S \cap \{v\}$ and $T' = V' \setminus S'$. This is a valid (s,t)-cut since $s \in S'$ and $t \in T'$. The capacity of this (s,t) cut will be:

$$c(S',T') = |S|\beta + \sum_{v \in V \setminus S} \frac{deg(v)}{2} - \frac{|E(S, V \setminus S)|}{2}$$

(Note how the edges to t helped us capture the first term in $c(S',T')$ and how the edges from s helped us capture the second term in $c(S',T')$.)

Assume that there is a subset $S \subseteq V$ such that $d(S) > \beta$. Then by applying Lemma 1 on S, we have $c(S',T') < m$. Therefore, the minimum cut will be less than m.

Now assume that for every subset $S \subseteq V$, we have $d(S) \leq \beta$. Consider the following two cases:

- If $S' = \{s\}$, then the cut value is equal to $\sum_{v \in V} \frac{deg(v)}{2} = m$.
- If $S' = S \cup \{s\}$, for any $S \subseteq V$ with $|S| \geq 1$, then by applying Lemma 1 on S, we have $c(S',T') \geq m$.

Therefore, the minimum cut will be equal to m. □

Back to the original problem. How do we turn the decision problem into solving the original problem? The standard technique in computer science is binary search! But how do we do binary search on β which can take *real values* in the range $[0, \frac{n-1}{2}]$. Actually, if you think about it, there are only $O(n^3)$ different values β can have.

Lemma 3. *If* $d(S_1) \neq d(S_2)$, *then* $|d(S_1) - d(S_2)| \geq \frac{1}{n^2}$.

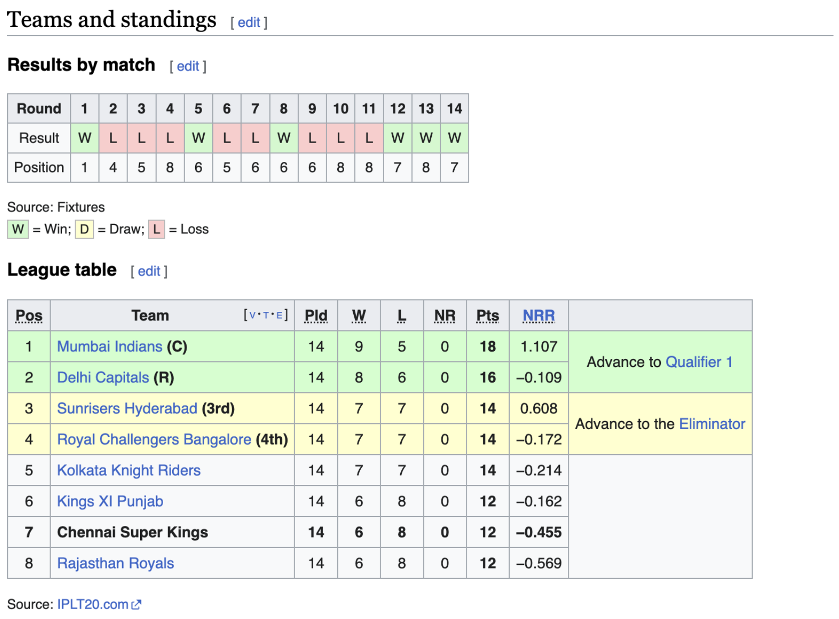
Proof. $|d(S_1) - d(S_2)| = \left|\frac{|E(S_1)||V(S_2)|-|E(S_2)||V(S_1)|}{|S_1||S_2|}\right|$. Since $d(S_1) \neq d(S_2)$, the numerator will be at least one and the denominator will be at most n^2. □

Therefore, the number of distinct values β can take is $O(n^3)$. As such, the number of binary searches performed will be $O(\log n^3) = O(\log n) = O(\log |V|)$. The overall running time to compute the densest subgraph will be $O(|E||V| \log |V|)$.

2 Application 2: Baseball elimination and CSK's IPL elimination

2.1 CSK in 2020

In 2020 CSK did not start well in the IPL. As shown in Figure 2, by the end of its first seven games it had won merely two games. As a cricket enthusiast and given the excellent track record of CSK in the past, I was wondering if there still a chance for CSK to finish at the top of the table. Later I got to know that a similar question has been asked and analyzed in the context of baseball which is discussed next.



Teams and standings [edit]

Results by match [edit]

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Result	W	L	L	L	W	L	L	W	L	L	L	W	W	W
Position	1	4	5	8	6	5	6	6	6	8	8	7	8	7

Source: Fixtures
W = Win; D = Draw; L = Loss

League table [edit]

Pos	Team [V·T·E]	Pld	W	L	NR	Pts	NRR	
1	Mumbai Indians **(C)**	14	9	5	0	**18**	1.107	Advance to Qualifier 1
2	Delhi Capitals **(R)**	14	8	6	0	**16**	−0.109	
3	Sunrisers Hyderabad **(3rd)**	14	7	7	0	**14**	0.608	Advance to the Eliminator
4	Royal Challengers Bangalore **(4th)**	14	7	7	0	**14**	−0.172	
5	Kolkata Knight Riders	14	7	7	0	**14**	−0.214	
6	Kings XI Punjab	14	6	8	0	**12**	−0.162	
7	**Chennai Super Kings**	**14**	**6**	**8**	**0**	**12**	**−0.455**	
8	Rajasthan Royals	14	6	8	0	**12**	−0.569	

Source: IPLT20.com

Figure 2: Table taken from Wikipedia.

2.2 Baseball elimination

Major League Baseball is quite popular in USA. Here is the table standings from the American League East on August 30, 1996.

Team	Won–Lost	Left	NYY	BAL	BOS	TOR	DET
New York Yankees	75–59	28		3	8	7	3
Baltimore Orioles	71–63	28	3		2	7	4
Boston Red Sox	69–66	27	8	2		0	0
Toronto Blue Jays	63–72	27	7	7	0		0
Detroit Tigers	49–86	27	3	4	0	0	

Figure 3: Table taken from Wikipedia.

Detroit is currently at the bottom of the table. A die-hard fan of Detroit Tigers would claim that their team can still finish at the top of the table by winning all the remaining 27 games and ending up with $49 + 27 = 76$ wins which is more than Yankees. However, this simple argument misses the point that whenever two teams play at least *one* of them will win! The remaining four teams cannot loose all their games :) The following is a more rigorous way to answer this question.

Formally, for all $1 \leq i \leq n$, let W_i be the number of wins for team i. Also, let G_{ij} be the number of games left to be played between team i and j, for all $1 \leq i, j \leq n$. We want to know whether it is possible for *team* n to finish with the most number of wins at the end of the season.

Let $R_i = \sum_{j=1}^{n} G_{ij}$ be the total number of remaining games for team i. We will consider the scenario where team n will win all its remaining games and hence, finish the season with $W_n + R_n$ wins. Therefore, for all $1 \leq i \leq n-1$, team i must win at most $W_n + R_n - W_i$ of its remaining matches.

We will construct the graph as follows (see Figure 4):

- First, construct a bipartite graph. The left side consists of $\binom{n-1}{2}$ nodes, where node g_{ij} corresponds to the pair of team i and team j, for all $1 \leq i, j \leq n - 1$. The right side consists of $n-1$ nodes where node t_i corresponds to team i. From each node g_{ij}, there are two directed edges to nodes t_i and t_j with capacity $+\infty$.
- Next, construct a source node s. There will be $\binom{n-1}{2}$ edges (s, g_{ij}) with capacity G_{ij}.
- Finally, construct a target node t. There will be $n - 1$ edges (t_i, t) with capacity $W_n + R_n - W_i$.

Lemma 4. *The following two statements are equivalent:*

1. *Team n wins the most number of games at the end of the season.*
2. *The maximum flow in the graph saturates all the edges leaving source s.*

Proof. $(1) \implies (2)$: Suppose team n wins the most number of games at the end of the season. Then we construct the following flow in the graph. In each of the G_{ij} games between team i and team j, if team i wins, then send a unit flow on the path s–g_{ij}–t_i–t;

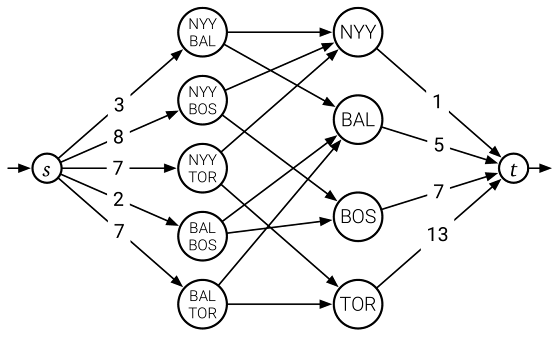


Figure 11.7. Cubs win! Cubs win!

Figure 4: Table taken from Wikipedia.

else, send a unit flow on the path s–g_{ij}–t_j–t. Do this for each pair (i, j) for all $1 \leq i, j \leq n-1$. Since team i wins at most $W_n + R_n - W_i$ remaining games, for all $1 \leq i \leq n-1$, the flow on the edge (t_i, t) satisfies the capacity constraint. Since *all* the G_{ij} are played between team i and team j, for all $1 \leq i, j \leq n - 1$, the edge (s, g_{ij}) is saturated. Note that this is indeed the maximum flow in the graph since each edge leaving s is saturated.

$(2) \implies (1)$: Suppose the maximum flow in the graph saturates all the edges leaving source s. Now we will use the flow to create an appropriate scenario. For all $1 \leq i, j \leq n-1$, among the games between team i and team j, consider a scenario where team i wins $f_{(g_{ij},t_i)}$ games and team j wins $f_{(g_{ij},t_j)}$ games. (Recall that $f_{(u,v)}$ is the flow on edge (u, v).) By the flow conservation constraint, $f_{(g_{ij},t_i)} + f_{(g_{ij},t_j)} = f_{(s,g_{ij})}$, and $f_{(s,g_{ij}} = G_{ij}$ since each edge leaving s is saturated. Therefore, $f_{(g_{ij},t_i)} + f_{(g_{ij},t_j)} = G_{ij}$, which implies that the scenario considered ensures that *all* the G_{ij} games between team i and team j are played.

Next, in this scenario the number of remaining games won by each team i will be $\sum_{j=1}^{n-1} f_{(g_{ij},t_i)}$ which is less than or equal to $W_n + R_n - R_i$ by conservation of flow constraint at node t_i. Therefore, by the end of the season the total number of games won by each team i will be at most $W_n + R_n$. □

Final algorithm. Construct the graph described above, run the maximum flow algorithm, and check if all the edges leaving the source are saturated or not.

3 Maximum bipartite matching

In the *maximum bipartite matching* problem, the input is a bipartite graph. The goal is to select the largest subset of the edges such that no two edges share a common vertex. One can think of many real-world applications of this problem. Consider a dating website in which one side represents men and other other side represents women. An edge represents mutual interest between a man and a woman. The maximum matching will pair up as many couples as possible. Matching will also ensure that no one is paired up with two or more! Other applications can include workers on one side and jobs on the other side.

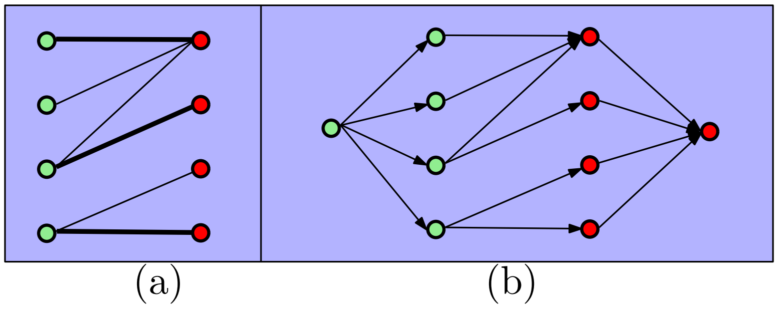


Figure 5: (a) Bipartite graph $L \cup R$ with the three edges in the maximum matching shown in bold, (b) The corresponding flow network G. The capacity of each edge is one. The maximum flow in G is three.

Graph construction. Let $L \cup R$ be the set of vertices in the bipartite graph. We will construct a graph G with vertex set $L \cup R \cup \{s, t\}$. For each edge (u, v) in the bipartite graph, create a directed edge (u, v) in G. For the source vertex s, create edges (s, u) for all $u \in L$. For the sink vertex t, create edges (v, t) for all $v \in R$. The capacity of all the edges will be one. See Figure 5. Can you think about the intuition behind capacity being one?

Lemma 5. *The size of the maximum bipartite matching in $L \cup R$ is equal to the value of the maximum flow in G.*

Proof. We will first prove that maximum flow is greater than or equal to the maximum matching. Next we will prove that maximum matching is greater than or equal to maximum flow. Both these statements can be simultaneously true only when they are equal to each other.

Let the size of the maximum matching be F. Then for each edge (u, v) in the matching, we will route one unit of flow on the path $s{-}u{-}v{-}t$. Since the edges form a matching, each vertex in L and R will be used at most once to route a flow. Hence, the flow will satisfy the capacity constraints. This implies that a flow of value F can be routed in G. Therefore, maximum flow in G is greater than or equal to the maximum matching.

Now assume that the maximum flow in the graph is F. Note that the flow on any edge will either be zero or one. Using this fact, let $L' = \{u_1, u_2, \ldots, u_F\}$ be the nodes in L receiving one unit of flow from s and let $R' = \{v_1, v_2, \ldots, v_F\}$ be the nodes in R sending one unit of flow to t. Also, for any node $u \in L'$, since the flow on the edge (s, u) is one unit, at most *one* outgoing edge from u can have flow equal to one. By a symmetric argument, for any node $v \in R'$, at most one incoming edge to v can have flow equal to one. This implies a matching of size F among L' and R': for each $u \in L'$, if (u, v) is the outgoing edge with flow one unit, then add (u, v) to the matching. Therefore, maximum bipartite matching is greater than or equal to the maximum flow. □

Algorithm. Run the maximum flow algorithm on G. Report all the edges (u, v) with $u \in L$ and $v \in R$ such that the flow on the edge (u, v) in G is equal to one. Via Orlin's algorithm this can be done in $O(|V||E|)$ time.

Exercise. In fact, Ford-Fulkerson algorithm will also run in $O(|V||E|)$ time in this setting. Why?

Exercise. Can the capacity of the edges from L to R be made $+\infty$?