Chapter 19

More Network Flow Applications

CS 473: Fundamental Algorithms, Spring 2011
April 5, 2011

19.1 Edge disjoint paths

19.1.1 Edge-disjoint paths in a directed graphs

Question 19.1.1 *Given a graph G (either directed or undirected), two vertices s and t, and a parameter k, the task is to compute k paths from s to t in G, such that they are* **edge disjoint***; namely, these paths do not share an edge.*

To solve this problem, we will convert G (assume G is a directed graph for the time being) into a network flow graph H, such that every edge has capacity 1. Find the maximum flow in G (between s and t). We claim that the value of the maximum flow in the network H, is equal to the number of edge disjoint paths in G.

Lemma 19.1.2 *If there are k edge disjoint paths in G between s and t, then the maximum flow in H is at least k.*

Proof: Given k such edge disjoint paths, push one unit of flow along each such path. The resulting flow is legal in h and it has value k. ■

Definition 19.1.3 (0/1-**flow.**) *A flow f is* 0/1-**flow** *if every edge has either no flow on it, or one unit of flow.*

Lemma 19.1.4 *Let f be a 0/1 flow in a network H with flow value μ. Then there are μ edge disjoint paths between s and t in H.*

Proof: By induction on the number of edges in H that has one unit of flow assigned to them by f. If $\mu = 0$ then there is nothing to prove.

Otherwise, start traversing the graph H from s traveling only along edges with flow 1 assigned to them by f. We mark such an edge as used, and do not allow one to travel on such an edge again. There are two possibilities:

(i) We reached the target vertex t. In this case, we take this path, add it to the set of output paths, and reduce the flow along the edges of the generated path π to 0. Let H' be the resulting flow network and f' the resulting flow. We have $|f'| = \mu - 1$, H' has less edges, and by induction, it has $\mu - 1$ edge disjoint paths in H' between s and t. Together with π this forms μ such paths.

(ii) We visit a vertex v for the second time. In this case, our traversal contains a cycle C, of edges in H that have flow 1 on them. We set the flow along the edges of C to 0 and use induction on the remaining graph (since it has less edges with flow 1 on them). The value of the flow f did not change by removing C, and as such it follows by induction that there are μ edge disjoint paths between s and t in H. ■

Since the graph G is simple, there are at most $n = |V(H)|$ edges that leave s. As such, the maximum flow in H is n. Thus, applying the Ford-Fulkerson algorithm, takes $O(mn)$ time. The extraction of the paths can also be done in linear time by applying the algorithm in the proof of Lemma 19.1.4. As such, we get:

Theorem 19.1.5 *Given a directed graph G with n vertices and m edges, and two vertices s and t, one can compute the maximum number of edge disjoint paths between s and t in H, in $O(mn)$ time.*

As a consequence we get the following cute result:

Lemma 19.1.6 *In a directed graph G with nodes s and t the maximum number of edge disjoint $s-t$ paths is equal to the minimum number of edges whose removal separates s from t.*

Proof: Let U be a collection of edge-disjoint paths from s to t in G. If we remove a set F of edges from G and separate s from t, then it must be that every path in U uses at least one edge of F. Thus, the number of edge-disjoint paths is bounded by the number of edges needed to be removed to separate s and t. Namely, $|U| \leq |F|$.

As for the other direction, let F be a set of edges thats its removal separates s and t. We claim that the set F form a cut in G between s and t. Indeed, let S be the set of all vertices in G that are reachable from s without using an edge of F. Clearly, if F is minimal then it must be all the edges of the cut (S, T) (in particular, if F contains some edge which is not in (S, T) we can remove it and get a smaller separating set of edges). In particular, the smallest set F with this separating property has the same size as the minimum cut between s and t in G, which is by the max-flow mincut theorem, also the maximum flow in the graph G (where every edge has capacity 1).

But then, by Theorem 19.1.5, there are $|F|$ edge disjoint paths in G (since $|F|$ is the amount of the maximum flow). ■

19.1.2 Edge-disjoint paths in undirected graphs

We would like to solve the s-t disjoint path problem for an undirected graph.

Problem 19.1.7 *Given undirected graph G, s and t, find the maximum number of edge-disjoint paths in G between s and t.*

The natural approach is to duplicate every edge in the undirected graph G, and get a (new) directed graph H. Next, apply the algorithm of Section 19.1.1 to H.

So compute for H the maximum flow f (where every edge has capacity 1). The problem is the flow f might use simultaneously the two edges $(u \to v)$ and $(v \to u)$. Observe, however, that in such case we can remove both edges from the flow f. In the resulting flow is legal and has the same value. As such, if we repeatedly remove those "double edges" from the flow f, the resulting flow f' has the same value. Next, we extract the edge disjoint paths from the graph, and the resulting paths are now edge disjoint in the original graph.

Lemma 19.1.8 *There are k edge-disjoint paths in an undirected graph G from s to t if and only if the maximum value of an $s-t$ flow in the directed version H of G is at least k. Furthermore, the Ford-Fulkerson algorithm can be used to find the maximum set of disjoint s-t paths in G in $O(mn)$ time.*

19.2 Applications

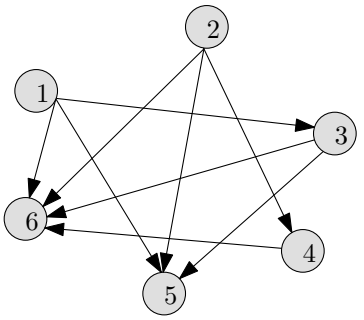
19.2.1 Survey design

We would like to design a survey of products used by consumers (i.e., "Consumer i: what did you think of product j?"). The ith consumer agreed in advance to answer a certain number of questions in the range $[c_i, c_i']$. Similarly, for each product j we would like to have at least p_j opinions about it, but not more than p_j'. Each consumer can be asked about a subset of the products which they consumed. In particular, we assume that we know in advance all the products each consumer used, and the above constraints. The question is how to assign questions to consumers, so that we get all the information we want to get, and every consumer is being asked a valid number of questions.

The idea of our solution is to reduce the design of the survey to the problem of computing a circulation in graph. First, we build a bipartite graph having consumers on one side, and products on the other side. Next, we insert the edge between consumer i and product j if the product was used by this consumer. The capacity of this edge is going to be 1. Intuitively, we are going to compute a flow in this network which is going to be an integer number. As such, every edge would be assigned either 0 or 1, where 1 is interpreted as asking the consumer about this product.

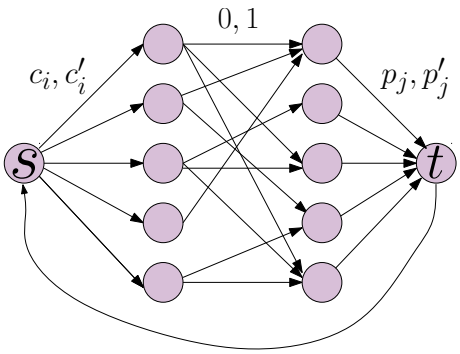
1: Boston (depart 6 A.M.) - Washington DC (arrive 7 A.M,).
2: Urbana (depart 7 A.M.) - Champaign (arrive 8 A.M.)
3: Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
4: Urbana (depart 11 A.M.) - San Francisco (arrive 2 P.M.)
5: San Francisco (depart 2:15 P.M.) - Seattle (arrive 3:15 P.M.)
6: Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.).

(i)



(ii)

Figure 19.1: (i) a set $\mathcal{F}$ of flights that have to be served, and (ii) the corresponding graph G representing these flights.



The next step, is to connect a source to all the consumers, where the edge $(s \to i)$ has lower bound c_i and upper bound c_i'. Similarly, we connect all the products to the destination t, where $(j \to t)$ has lower bound p_j and upper bound p_j'. We would like to compute a flow from s to t in this network that comply with the constraints. However, we only know how to compute a circulation on such a network. To overcome this, we create an edge with infinite capacity between t and s. Now, we are only looking for a valid circulation in the resulting graph G which complies with the aforementioned constraints. See figure on the right for an example of G.

Given a circulation f in G it is straightforward to interpret it as a survey design (i.e., all middle edges with flow 1 are questions to be asked in the survey). Similarly, one can verify that given a valid survey, it can be interpreted as a valid circulation in G. Thus, computing circulation in G indeed solves our problem.

We summarize:

Lemma 19.2.1 *Given n consumers and u products with their constraints $c_1, c_1', c_2, c_2', \ldots, c_n, c_n'$, $p_1, p_1', \ldots, p_u, p_u'$ and a list of length m of which products where used by which consumers. An algorithm can compute a valid survey under these constraints, if such a survey exists, in time $O((n+u)m^2)$.*

19.2.2 Airline Scheduling

Problem 19.2.2 *Given information about flights that an airline needs to provide, generate a profitable schedule.*

The input is a detailed information about "legs" of flight that the airline need to serve. We denote this set of flights by $\mathcal{F}$. We would like to find the minimum number of airplanes needed to carry out this schedule. For an example of possible input, see Figure 19.1 (i).

We can use the same airplane for two segments i and j if the destination of i is the origin of the segment j and there is enough time in between the two flights for required maintenance. Alternatively, the airplane can fly from dest(i) to origin(j) (assuming that the time constraints are satisfied).

Example 19.2.3 *As a concrete example, consider the flights:*

1. *Boston (depart 6 A.M.) - Washington D.C. (arrive 7 A.M,).*
2. *Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)*
3. *Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.)*

This schedule can be served by a single airplane by adding the leg "Los-Angeles (depart 12 noon)- Las Vegas (1 P,M.)" to this schedule.

19.2.3 Modeling the problem

The idea is to model the feasibility constraints by a graph. Specifically, G is going to be a directed graph over the flight legs. For i and j, two given flight legs, the edge $(i \to j)$ will be present in the graph G if the same airplane can serve both i and j; namely, the same airplane can perform leg i and afterwards serves the leg j.

Thus, the graph G is acyclic. Indeed, since we can have an edge $(i \to j)$ only if the flight j comes after the flight i (in time), it follows that we can not have cycles.

We need to decide if all the required legs can be served using only k airplanes?

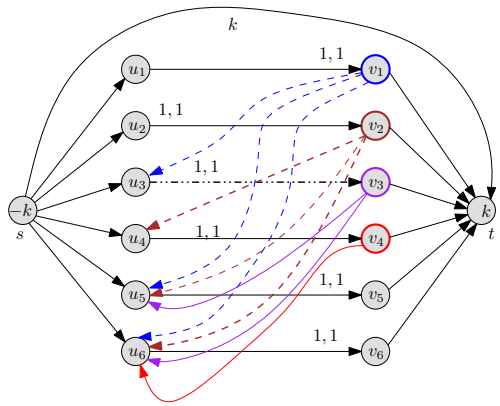


Figure 19.2: The resulting graph H for the instance of airline scheduling from Figure 19.1.

19.2.4 Solution

The idea is to perform a reduction of this problem to the computation of circulation. Specifically, we construct a graph H, as follows:

- For every leg i, we introduce two vertices $u_i, v_i \in V(H)$. We also add a source vertex s and a sink vertex t to H. We set the demand at t to be k, and the demand at s to be $-k$ (i.e., k units of flow are leaving s and need to arrive to T).
- Each flight on the list must be served. This is forced by introducing an edge $e_i = (u_i \to v_i)$, for each leg i. We also set the lower bound on e_i to be 1, and the capacity on e_i to be 1 (i.e., $\ell(e_i) = 1$ and $c(e_i) = 1$).

- If the same plane can perform flight i and j (i.e., $(i \to j) \in E(\mathsf{G})$) then add an edge $(v_i \to u_j)$ with capacity 1 to H (with no lower bound constraint).

- Since any airplane can start the day with flight i, we add an edge $(s \to u_i)$ with capacity 1 to H, for all flights i.

- Similarly, any airplane can end the day by serving the flight j. Thus, we add edge $(v_j \to t)$ with capacity 1 to G, for all flights j.

- If we have extra planes, we do not have to use them. As such, we introduce a "overflow" edge $(s \to t)$ with capacity k, that can carry over all the unneeded airplanes from s directly to t.

Let H denote the resulting graph. See Figure 19.2 for an example.

Lemma 19.2.4 *There is a way to perform all flights of $\mathcal{F}$ using at most k planes if and only if there is a feasible circulation in the network H.*

Proof: Assume there is a way to perform the flights using $k' \leq k$ flights. Consider such a feasible schedule. The schedule of an airplane in this schedule defines a path π in the network H that starts at s and ends at t, and we send one unit of flow on each such path. We also send $k - k'$ units of flow on the edge $(s \to t)$. Note, that since the schedule is feasible, all legs are being served by some airplane. As such, all the "middle" edges with lower-bound 1 are being satisfied. Thus, this results is a valid circulation in H that satisfies all the given constraints.

As for the other direction, consider a feasible circulation in H. This is an integer valued circulation by the Integrality theorem. Suppose that k' units of flow are sent between s and t (ignoring the flow on the edge $(s \to t)$). All the edges of H (except $(s \to t)$) have capacity 1, and as such the circulation on all other edges is either zero or one (by the Integrality theorem). We convert this into k' paths by repeatedly traversing from the vertex s to the destination t, removing the edges we are using in each such path after extracting it (as we did for the k disjoint paths problem). Since we never use an edge twice, and H is acyclic, it follows that we would extract k' paths. Each of those paths correspond to one airplane, and the overall schedule for the airplanes is valid, since all required legs are being served (by the lower-bound constraint). ■

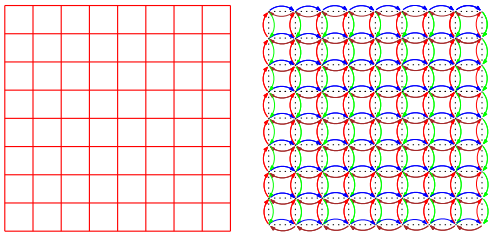
Extensions and limitations. There are a lot of other considerations that we ignored in the above problem: (i) airplanes have to undergo long term maintenance treatments every once in awhile, (ii) one needs to allocate crew to these flights, (iii) schedule differ between days, and (iv) ultimately we interested in maximizing revenue (a much more fluffy concept and much harder to explicitly describe).

In particular, while network flow is used in practice, real world problems are complicated, and network flow can capture only a few aspects. More than undermining the usefulness of network flow, this emphasize the complexity of real-world problems.

(i) (ii)

Figure 19.3: The (i) input image, and (ii) a possible segmentation of the image.

19.2.5 Image Segmentation



In the ***image segmentation problem***, the input is an image, and we would like to partition it into background and foreground. For an example, see Figure 19.3.

The input is a bitmap on a grid where every grid node represents a pixel. We covert this grid into a directed graph G, by interpreting every edge of the grid as two directed edges. See the figure on the right to see how the resulting graph looks like.

Specifically, the input for out problem is as follows:

- A bitmap of size $N \times N$, with an associated directed graph $\mathsf{G} = (V, E)$.
- For every pixel i, we have a value $f_i \geq 0$, which is an estimate of the likelihood of this pixel to be in foreground (i.e., the larger f_i is the more probable that it is in the foreground)
- For every pixel i, we have (similarly) an estimate b_i of the likelihood of pixel i to be in background.
- For every two adjacent pixels i and j we have a separation penalty p_{ij}, which is the "price" of separating i from j. This quantity is defined only for adjacent pixels in the bitmap. (For the sake of simplicity of exposition we assume that $p_{ij} = p_{ji}$. Note, however, that this assumption is not necessary for our discussion.)

Problem 19.2.5 *Given input as above, partition V (the set of pixels) into two disjoint subsets F and B, such that*

$$q(F, B) = \sum_{i \in F} f_i + \sum_{i \in B} b_i - \sum_{(i,j) \in E, |F \cap \{i,j\}| = 1} p_{ij}.$$

is maximized.

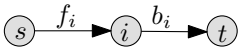
We can rewrite $q(F, B)$ as:

$$q(F,B) \;=\; \sum_{i\in F} f_i + \sum_{j\in B} b_j - \sum_{(i,j)\in E, |F\cap\{i,j\}|=1} p_{ij} = \sum_{i\in v}(f_i+b_i) - \sum_{i\in B} f_i - \sum_{j\in F} b_j - \sum_{(i,j)\in E, |F\cap\{i,j\}|=1} p_{ij}.$$

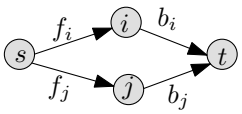
Since the term $\sum_{i\in v}(f_i + b_i)$ is a constant, maximizing $q(F, B)$ is equivalent to minimizing $u(F, B)$, where

$$u(F,B) \;=\; \sum_{i\in B} f_i + \sum_{j\in F} b_j + \sum_{(i,j)\in E, |F\cap\{i,j\}|=1} p_{ij}. \tag{19.1}$$

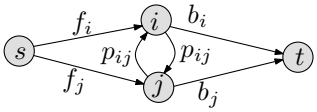
How do we compute this partition. Well, the basic idea is to compute a minimum cut in a graph such thats its price would correspond to $u(F, B)$. Before dwelling into the exact details, it is useful to play around with some toy examples to get some intuition. Note, that we are using the max-flow algorithm as an algorithm for computing minimum directed cut.



To begin with, consider a graph having a source s, a vertex i, and a sink t. We set the price of $(s \to i)$ to be f_i and the price of the edge $(i \to t)$ to be b_i. Clearly, there are two possible cuts in the graph, either $(\{s, i\}, \{t\})$ (with a price b_i) or $(\{s\}, \{i, t\})$ (with a price f_i). In particular, every path of length 2 in the graph between s and t forces the algorithm computing the minimum-cut (via network flow) to choose one of the edges, to the cut, where the algorithm "prefers" the edge with lower price.



Next, consider a bitmap with two vertices i an j that are adjacent. Clearly, minimizing the first two terms in Eq. (19.1) is easy, by generating length two parallel paths between s and t through i and j. See figure on the right. Clearly, the price of a cut in this graph is exactly the price of the partition of $\{i, j\}$ into background and foreground sets. However, this ignores the separation penalty p_{ij}.



To this end, we introduce two new edges $(i \to j)$ and $(j \to i)$ into the graph and set their price to be p_{ij}. Clearly, a price of a cut in the graph can be interpreted as the value of $u(F, B)$ of the corresponding sets F and B, since all the edges in the segmentation from nodes of F to nodes of B are contributing their separation price to the cut price. Thus, if we extend this idea to the directed graph G, the minimum-cut in the resulting graph would corresponds to the required segmentation.

Let us recap: Given the directed grid graph $\mathsf{G} = (V, E)$ we add two special source and sink vertices, denoted by s and t respectively. Next, for all the pixels $i \in V$, we add an edge $e_i = (s \to i)$ to the graph, setting its capacity to be $c(e_i) = f_i$. Similarly, we add the edge $e'_i = (j \to t)$ with capacity $c(e'_i) = b_i$. Similarly, for every pair of vertices $i.j$ in that grid that are adjacent, we assign the cost p_{ij} to the edges $(i \to j)$ and $(j \to i)$. Let H denote the resulting graph.

The following lemma, follows by the above discussion.

Lemma 19.2.6 *A minimum cut (F, B) in H minimizes $u(F, B)$.*

Using the minimum-cut max-flow theorem, we have:

Theorem 19.2.7 *One can solve the segmentation problem, in polynomial time, by computing the max flow in the graph H.*

19.2.6 Project Selection

You have a small company which can carry out some projects out of a set of projects P. Associated with each project $i \in P$ is a revenue p_i, where $p_i > 0$ is a profitable project and $p_i < 0$ is a losing project. To make things interesting, there is dependency between projects. Namely, one has to complete some "infrastructure" projects before one is able to do other projects. Namely, you are provided with a graph $\mathsf{G} = (P, E)$ such that $(i \to j) \in E$ if and only j is a prerequisite for i.

Definition 19.2.8 *A set $A \subset P$ is* **feasible** *if for all $i \in A$, all the prerequisites of i are also in A. Formally, for all $i \in A$, with an edge $(i \to j) \in E$, we have $j \in A$.*

The **profit** *associated with a set of projects $A \subseteq P$ is* $\text{profit}(A) = \sum_{i \in A} p_i$.

Problem 19.2.9 (Project Selection Problem.) *Select a feasible set of projects maximizing the overall profit.*

The idea of the solution is to reduce the problem to a minimum-cut in a graph, in a similar fashion to what we did in the image segmentation problem.

19.2.7 The reduction

The reduction works by adding two vertices s and t to the graph G, we also perform the following modifications:

- For all projects $i \in P$ with positive revenue (i.e., $p_i > 0$) add the $e_i = (s \to i)$ to G and set the capacity of the edge to be $c(e_i) = p_i$.
- Similarly, for all projects $j \in P$, with negative revenue (k.e., $p_j < 0$) add the edge $e'_j = (j \to t)$ to G and set the edge capacity to $c(e'_j) = -p_j$.
- Compute a bound on the max flow (and thus also profit) in this network: $C = \sum_{i \in P, p_i > 0} p_i$.
- Set capacity of all other edges in G to $4C$ (these are the dependency edges in the project, and intuitively they are too expensive to be "broken" by a cut).

Let H denote the resulting network.

Let $A \subseteq P$ Be a set of feasible projects, and let $A' = A \cup \{s\}$ and $B' = (P \setminus A) \cup \{t\}$. Consider the s-t cut (A', B') in H. Note, that no edge in $E(\mathsf{G})$ is of (A', B') since A is a feasible set.

Lemma 19.2.10 *The capacity of the cut (A', B'), as defined by a feasible project set A, is $c(A', B') = C - \sum_{i \in A} p_i = C - \text{profit}(A)$.*

Proof: The edges of H are either: (i) Edges of G, (ii) edges emanating from s, and (iii) edges entering t. Since A is feasible, it follows that no edges of type (i) contribute to the cut. The edges entering t contribute to the cut the value

$$X = \sum_{i \in A \text{ and } p_i < 0} -p_i.$$

The edges leaving the source s contribute

$$Y = \sum_{i \notin A \text{ and } p_i > 0} p_i = \sum_{i \in P, p_i > 0} p_i - \sum_{i \in A \text{ and } p_i > 0} p_i = C - \sum_{i \in A \text{ and } p_i > 0} p_i,$$

by the definition of C.

The capacity of the cut (A', B') is

$$X + Y = \sum_{i \in A \text{ and } p_i < 0} (-p_i) + C - \sum_{i \in A \text{ and } p_i > 0} p_i. = C - \sum_{i \in A} p_i = C - \text{profit}(A),$$

as claimed. ∎

Lemma 19.2.11 *If (A', B') is a cut with capacity at most C in G, then the set $A = A' \setminus \{s\}$ is a feasible set of projects.*

Namely, cuts (A', B') of capacity $\leq C$ in H corresponds one-to-one to feasible sets which are profitable.

Proof: Since $c(A', B') \leq C$ it must not cut any of the edges of G, since the price of such an edge is $4C$. As such, A must be a feasible set. ∎

Putting everything together, we are looking for a feasible set that maximizes $\sum_{i \in A} p_i$. This corresponds to a set $A' = A \cup \{S\}$ of vertices in H that minimizes $C - \sum_{i \in A} p_i$, which is also the cut capacity (A', B'). Thus, computing a minimum-cut in H corresponds to computing the most profitable feasible set of projects.

Theorem 19.2.12 *If (A', B') is a minimum cut in H then $A = A' \setminus \{s\}$ is an optimum solution to the project selection problem. In particular, using network flow the optimal solution can be computed in polynomial time.*

19.2.8 Baseball elimination

There is a baseball league taking place and it is nearing the end of the season. One would like to know which teams are still candidates to winning the season.

Example 19.2.13 *There* 4 *teams that have the following number of wins:*

NEW YORK: 92, BALTIMORE: 91, TORONTO: 91, BOSTON: 90,

and there are 5 games remaining (all pairs except New York and Boston).

We would like to decide if Boston can still win the season? Namely, can Boston finish the season with as many point as anybody else? (We are assuming here that at every game the winning team gets one point and the losing team gets nada.[1]*)*

First analysis. *Observe, that Boston can get at most 92 wins. In particular, if New York wins any game then it is over since New-York would have 93 points.*

Thus, to Boston to have any hope it must be that both Baltimore wins against New York and Toronto wins against New York. At this point in time, both teams have 92 points. But now, they play against each other, and one of them would get 93 wins. So Boston is eliminated!

Second analysis. *As before, Boston can get at most 92 wins. All three other teams gets* $X = 92 + 91 + 91 + (5 - 2)$ *points together by the end of the league. As such, one of these three teams will get* $\geq \lceil X/3 \rceil = 93$ *points, and as such Boston is eliminated.*

While the analysis of the above example is very cute, it is too tedious to be done each time we want to solve this problem. Not to mention that it is unclear how to extend these analyses to other cases.

19.2.9 Problem definition

Problem 19.2.14 *The input is a set* S *of teams, where for every team* $x \in S$*, the team has* w_x *points accumulated so far. For every pair of teams* $x, y \in S$ *we know that there are* g_{xy} *games remaining between* x *and* y*. Given a specific team* z*, we would like to decide if* z *is eliminated?*

Alternatively, is there away such z *would get as many wins as anybody else by the end of the season?*

19.2.10 Solution

First, we can assume that z wins all its remaining games, and let m be the number of points z has in this case. Our purpose now is to build a network flow so we can verify that no other team *must* get more than m points.

[1] nada = nothing.

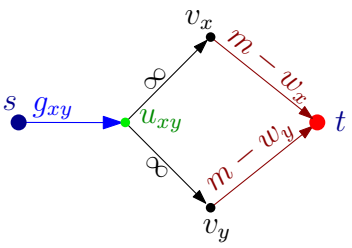
To this end, let s be the source (i.e., the source of wins). For every remaining game, a flow of one unit would go from s to one of the teams playing it. Every team can have at most $m - w_x$ flow from it to the target. If the max flow in this network has value

$$\alpha = \sum_{x,y \neq z, x<y} g_{xy}$$

(which is the maximum flow possible) then there is a scenario such that all other teams gets at most m points and z can win the season. Negating this statement, we have that if the maximum flow is smaller than α then z is eliminated, since there must be a team that gets more than m points.

Construction. Let $S' = S \setminus \{z\}$ be the set of teams, and let $\alpha = \sum_{\{x,y\} \subseteq S'} g_{xy}$. We create a network flow G. For every team $x \in S'$ we add a vertex v_x to the network G. We also add the source and sink vertices, s and t, respectively, to G.

For every pair of teams $x, y \in S'$, such that $g_{xy} > 0$ we create a node u_{xy}, and add an edge $(s \to u_{xy})$ with capacity g_{xy} to G. We also add the edge $(u_{xy} \to v_x)$ and $(u_{xy} \to v_y)$ with infinite capacity to G. Finally, for each team x we add the edge $(v_x \to t)$ with capacity $m - w_x$ to G. How the relevant edges look like for a pair of teams x and y is depicted on the right.



Analysis. If there is a flow of value α in G then there is a way that all teams get at most m wins. Similarly, if there exists a scenario such that z ties or gets first place then we can translate this into a flow in G of value α. This implies the following result.

Theorem 19.2.15 *Team z has been eliminated if and only if the maximum flow in G has value strictly smaller than α. Thus, we can test in polynomial time if z has been eliminated.*

19.2.11 A compact proof of a team being eliminated

Interestingly, once z is eliminated, we can generate a compact proof of this fact.

Theorem 19.2.16 *Suppose that team z has been eliminated. Then there exists a "proof" of this fact of the following form:*

1. *The team z can finish with at most m wins.*
2. *There is a set of teams $\widehat{S} \subset S$ so that $\sum_{s \in \widehat{S}} w_x + \sum_{\{x,y\} \subseteq \widehat{S}} g_{xy} > m \left|\widehat{S}\right|$.*

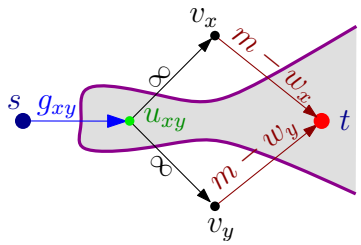
(And hence one of the teams in $\widehat{S}$ must end with strictly more than m wins.)

Proof: If z is eliminated then the max flow in G has value γ, which is smaller than α. By the minimum-cut max-flow theorem, there exists a minimum cut (S,T) of capacity γ in G, and let $\widehat{S} = \left\{ x \;\middle|\; v_x \in S \right\}$

Claim 19.2.17 *For any two teams x and y for which the vertex u_{xy} exists, we have that $u_{xy} \in S$ if and only if both x and y are in $\widehat{S}$.*

Proof: $\underline{\left(x \notin \widehat{S} \text{ or } y \notin \widehat{S}\right) \implies u_{xy} \notin S}$: If x is not in $\widehat{S}$ then v_x is in T. But then, if u_{xy} is in S the edge $(u_{xy} \to v_x)$ is in the cut. However, this edge has infinite capacity, which implies this cut is not a minimum cut (in particular, (S,T) is a cut with capacity smaller than α). As such, in such a case u_{xy} must be in T. This implies that if either x or y are *not* in $\widehat{S}$ then it must be that $u_{xy} \in T$. (And as such $u_{xy} \notin S$.)

$\underline{x \in \widehat{S} \text{ and } y \in \widehat{S} \implies u_{xy} \in S}$: Assume that both x and y are in $\widehat{S}$, then v_x and v_y are in S. We need to prove that $u_{xy} \in S$. If $u_{xy} \in T$ then consider the new cut formed by moving u_{xy} to S. For the new cut (S', T') we have



$$c(S', T') = c(S,T) - c\big((s \to u_{xy})\big) .$$

Namely, the cut (S', T') has a lower capacity than the minimum cut (S,T), which is a contradiction. See figure on the right for this *impossible* cut. We conclude that $u_{xy} \in S$. ■

The above argumentation implies that edges of the type $(u_{xy} \to v_x)$ can not be in the cut (S,T). As such, there are two type of edges in the cut (S,T): (i) $(v_x \to t)$, for $x \in \widehat{S}$, and (ii) $(s \to u_{xy})$ where at least one of x or y is not in $\widehat{S}$. As such, the capacity of the cut (S,T) is

$$c(S,T) = \sum_{x \in \widehat{S}} (m - w_x) + \sum_{\{x,y\} \not\subset \widehat{S}} g_{xy} = m \left|\widehat{S}\right| - \sum_{x \in \widehat{S}} w_x + \left(\alpha - \sum_{\{x,y\} \subseteq \widehat{S}} g_{xy} \right) .$$

However, $c(S,T) = \gamma < \alpha$, and it follows that

$$m \left|\widehat{S}\right| - \sum_{x \in \widehat{S}} w_x - \sum_{\{x,y\} \subseteq \widehat{S}} g_{xy} < \alpha - \alpha = 0.$$

Namely, $\displaystyle\sum_{x \in \widehat{S}} w_x + \sum_{\{x,y\} \subseteq \widehat{S}} g_{xy} > m \left|\widehat{S}\right|$, as claimed. ■