4 Maximum flows in networks

Throughout this chapter we will consider a directed network N with two distinguished vertices x and y. We denote the weight of an arc e of N by $c(e)$ and refer to it as the *capacity* of e. We assume that the capacity of each arc of N is a non-negative integer. Given a vertex v of N, we denote the set of arcs leaving v by $A_N^+(v)$ and the set of arcs entering v by $A_N^-(v)$.

4.1 Definition

Suppose f is a function which associates a non-negative integer with each arc of N, so $f : A(N) \to \mathbb{N}$. For each vertex v of N let $f^+(v) = \sum_{e \in A_N^+(v)} f(e)$ and $f^-(v) = \sum_{e \in A_N^-(v)} f(e)$. We say that f is an *xy-flow in N* if:

- $0 \leq f(e) \leq c(e)$ for all $e \in A(N)$, and
- $f^+(v) = f^-(v)$ for all $v \in V(N) - \{x, y\}$.

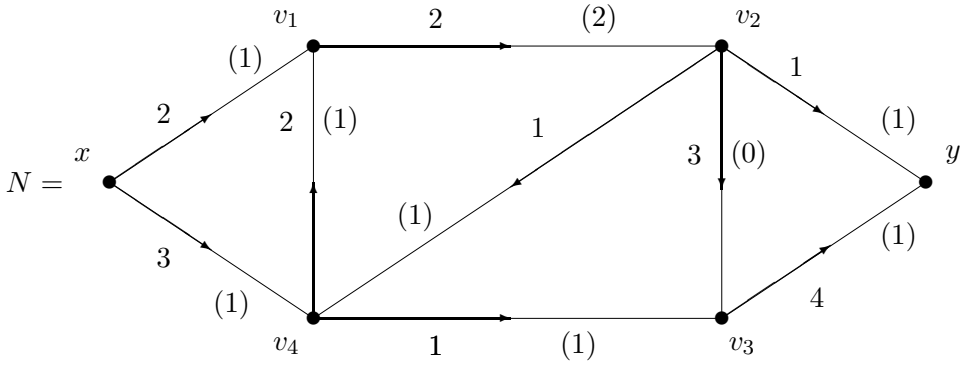
The *value* of the xy-flow f is given by

$$val(f) = f^+(x) - f^-(x).$$

Thus the value of the flow is the net flow out of x.

4.2 Example

The following network represents one way roads in a city centre. The capacity of each arc is the maximum number of cars/second which can travel along the corresponding road. Cars can only enter the network at junction x and leave at junction y. The given xy-flow represents the number of cars/second travelling along each road. The value of a flow is the rate at which cars are passing through the network from x to y.



The flow along an arc is given by the number in brackets and its capacity by the number not in brackets. We have $val(f) = f^+(x) - f^-(x) = (1+1) - 0 = 2$.

We will consider the optimization problem of finding an xy-flow of maximum value in N.

4.3 Notation

For U a proper subset of $V(N)$, let $A_N^+(U)$ be the set of all arcs of N from U to $V(N) - U$. We say that $A_N^+(U)$ is an *arc-cut* of N. When $x \in U$ and $y \in V(N) - U$ we say that $A_N^+(U)$ is an *xy-arc-cut* of N. The capacity of the arc-cut $A_N^+(U)$ is defined to be

$$c^+(U) = \sum_{e \in A_N^+(U)} c(e) \text{ and } c^-(U) = \sum_{e \in A_N^-(U)} c(e).$$

Similarly, given an xy-flow f for N we let

$$f^+(U) = \sum_{e \in A_N^+(U)} f(e) \text{ and } f^-(U) = \sum_{e \in A_N^-(U)} f(e)$$

Thus $f^+(U)$ is the total flow from U to $V(N) - U$ and $f^-(U)$ is the total flow from $V(N) - U$ to U.

4.4 Lemma

Let f be an xy-flow in N and $U \subset V(N)$ with $x \in U$ and $y \in V(N) - U$. Then $val(f) = f^+(U) - f^-(U)$.

Proof Since f is an xy-flow we have $f^+(v) = f^-(v)$ for all $v \in U - \{x\}$. Hence

$$val(f) = f^+(x) - f^-(x) = \sum_{u \in U} f^+(u) - \sum_{u \in U} f^-(u).$$

The sums on the right hand side of the above equality count $f(e)$ on each arc e incident with vertices of U. There are three alternatives for such an arc e.

- e is incident with two vertices of U. Then $f(e)$ is counted once in $\sum_{u \in U} f^+(u)$ and once in $\sum_{u \in U} f^-(u)$ so its contribution to $\sum_{u \in U} f^+(u) - \sum_{u \in U} f^-(u)$ is zero.
- $e \in A_N^+(U)$. Then $f(e)$ is counted once in $\sum_{u \in U} f^+(u)$ so its contribution to $\sum_{u \in U} f^+(u) - \sum_{u \in U} f^-(u)$ is $f(e)$.
- $e \in A_N^-(U)$. Then $f(e)$ is counted once in $\sum_{u \in U} f^-(u)$ so its contribution to $\sum_{u \in U} f^+(u) - \sum_{u \in U} f^-(u)$ is $-f(e)$.

Thus

$$\sum_{u \in U} f^+(u) - \sum_{u \in U} f^-(u) = \sum_{e \in A_N^+(U)} f(e) - \sum_{e \in A_N^-(U)} f(e) = f^+(U) - f^-(U).$$

Hence $val(f) = f^+(U) - f^-(U)$.

4.5 Corollary

Let f be an xy-flow in N and $U \subset V(N)$ with $x \in U$ and $y \in V(N) - U$. Then $val(f) \leq c^+(U)$.
Proof This follows from Lemma 4.4 since f is a flow and hence $f^+(U) \leq c^+(U)$ and $f^-(U) \geq 0$.

4.6 Corollary

Let f be an xy-flow in B and $U \subset V(N)$ be such that U separates x from y. If $val(f) = c^+(U)$ then f is an xy-flow of maximum value in N and $A_N^+(U)$ is an xy-arc cut of minimum capacity in N.
Proof Follows immediately from Corollary 4.5.

4.7 Definition

Let f be an xy-flow in N. An *f-unsaturated* path in N is a path P (possibly containing both forward and backward arcs) satisfying the following two conditions.

- for each forward directed arc e of P we have $f(e) < c(e)$, and
- for each backward directed arc e of P we have $f(e) > 0$.

Suppose P is an f-unsaturated xy-path P in N. For each arc e of P let: $s(e) = c(e) - f(e)$ if e is a forward arc of P, and $s(e) = f(e)$ if e is a backward arc of P. Put $s(P) = \min\{s(e)\ ;\ e \in A(P)\}$.

Thus, in Example 4.2, the path $P = xv_4v_2v_3y$ is an f-unsaturated xy-path and $s(P) = \min\{2, 1, 3, 3\} = 1$.

4.8 Lemma

Suppose f is an xy-flow in N and P is an f-unsaturated xy-path P in N. Then N has an xy-flow g with $val(g) = val(f) + s(P)$.

Proof Define a function $g : A(N) \to \mathbb{N}$ by

$$g(e) = \begin{cases} f(e) & \text{if } e \notin A(P) \\ f(e) + s(P) & \text{if } e \text{ is a forward arc of } P \\ f(e) - s(P) & \text{if } e \text{ is a backward arc of } P \end{cases}$$

It is straightforward (but tedious) to check that g is an xy-flow in N and $val(g) = val(f) + s(P)$.

The Ford-Fulkerson Algorithm

Ford and Fulkerson gave an algorithm for constructing an xy-flow of maximum value in a directed network N in 1956. Their algorithm begins with a given xy-flow f and searches for an f-unsaturated xy-path in N. It does this by growing a maximal f-unsaturated directed tree T rooted at x i.e. a maximal directed tree T with the property that all paths in T starting at x are f-unsaturated. If T contains y then we may use the f-unsaturated xy-path in T and Lemma 4.8 to construct a new flow g with $val(g) > val(f)$. We then iterate. If T does not contain y then we can use the tree T to construct an xy-arc-cut $A_N^+(U)$ with $val(f) = c^+(U)$. We may then use Corollary 4.6 to deduce that f is a flow of maximum value. The algorithm is more complicated to describe than previous algorithms (since it grows a

tree at every iteration) so we split it up into subroutines.

Subroutine: Maximal Unsaturated Tree We are given an xy-flow f in N. We construct a maximal f-unsaturated directed tree T rooted at x.

Initial Step Let T_1 be the directed tree with $V(T_1) = \{x\}$ and $A(T_1) = \emptyset$.

Iterative Step Suppose we have constructed a tree T_i with for some $i \geq 1$.

- If there exists an arc $e = uv$ of N from T_i to $N - T_i$ with $f(e) < c(e)$ then put $T_{i+1} = T_i + v + e$.
- If there exists an arc $e = vu$ of N from $N - T_i$ to T_i with $f(e) > 0$ then put $T_{i+1} = T_i + v + e$.
- If all arcs of N from T_i to $N - T_i$ satisfy $f(e) = c(e)$ and all arcs of N from $N - T_i$ to T_i satisfy $f(e) = 0$ then STOP. Put $T = T_i$ and output T.

Subroutine: Augment Flow We are given an xy-flow f in N and an f-unsaturated xy-path P. We construct a new xy-flow g with $val(g) = val(f) + s(P)$ as in the proof of Lemma 4.8.

Main Algorithm We are given an xy-flow f_1 in N. (If no xy-flow is given then we take f_1 to be the *zero-flow* by putting $f_1(e) = 0$ for all $e \in A(N)$.) We iteratively construct a sequence of xy-flows $f_1, f_2, \ldots$ with $val(f_{i+1}) > val(f_i)$. The algorithm terminates when it finds an xy-flow f and a proper subset U of $V(N)$ such that $x \in U$, $y \in V(N) - U$ and $val(f) = c^+(U)$.

Iterative Step Suppose we have an xy-flow f_i for some $i \geq 1$. Grow a maximal f_i-unsaturated tree T rooted at x using Subroutine: Maximal Unsaturated Tree.

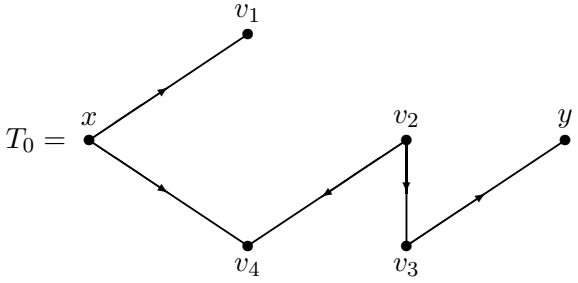
- If y is a vertex of T then the path P from x to y in T is f_i-unsaturated. Construct a new xy-flow f_{i+1} with $val(f_{i+1}) = val(f_i) + s(P)$ using Subroutine: Augment Flow.
- If y is not a vertex of T then STOP. Put $f = f_i$, and $U = V(T)$, and output f and U.

4.9 Example

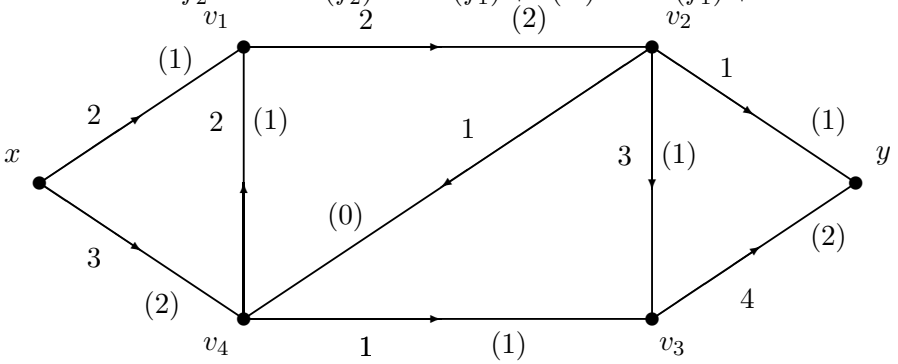
We apply the algorithm to the flow f_1 in the network N of Example 4.2 as follows.

First Iteration

Step 1 Grow a maximal f_1-unsaturated tree T from x.

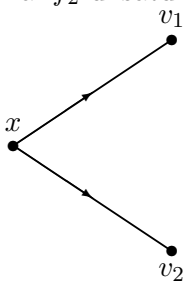


Step 2 T contains y. Use the f_1-unsaturated xy-path $P = xv_4v_2v_3y$ in T to define a new flow f_2 with $val(f_2) = val(f_1) + s(P) = val(f_1) + 1$.



Second Iteration

Step 1 Grow a maximal f_2-unsaturated tree T from x.



Step 2 T does not contain y. Let $f = f_2$ and $U = V(T) = \{x, v_1, v_2\}$.

Note that we have $val(f) = 3 = c^+(U)$. Hence by Lemma 4.6, f is an xy-flow of maximum value in N, and $A_N^+(U)$ is an xy-arc-cut of minimum capacity in N.

4.10 Theorem

Let N be a directed network and x, y be vertices of N. Let f be an xy-flow and $U \subset V(N)$ constructed by the Ford-Fulkerson algorithm. Then $val(f) = c^+(U)$. Hence f is an xy-flow of maximum value in N, and $A_N^+(U)$ is an xy-arc-cut of minimum capacity in N.

Proof We have $U = V(T)$, where T is the maximal f-unsaturated tree rooted at x constructed in the last iteration of the Ford-Fulkerson algorithm and $y \in V(N) - V(T)$. Since T was constructed by Subroutine: Maximal Unsaturated Tree, we have $f(e) = c(e)$ for all arcs from U to $V(N) - U$, and $f(e) = 0$ for all arcs from $V(N) - U$ to U. Using Lemma 4.4, we now have

$$val(f) = f^+(U) - f^-(U) = \sum_{e \in A_N^+(U)} f(e) - \sum_{e \in A_N^-(U)} f(e) = \sum_{e \in A_N^+(U)} c(e) - 0 = c^+(U).$$

Corollary 4.6 now implies that f is an xy-flow of maximum value and that $A_N^+(U)$ is an xy-arc cut of minimum capacity.

4.11 Corollary (The Max-flow Min-cut Theorem)

The maximum value of an xy-flow in N is equal to the minimum capacity of an xy-arc cut in N.

Proof Follows immediately from Theorem 4.10.

4.12 Complexity of network algorithms

Suppose we have an algorithm which will solve a particular problem for any given network N, for example finding a minimum weight spanning tree in N. We would expect the time that the algorithm takes to solve the problem to be dependent on the numbers of vertices and edges in N. However we would also expect the time taken to depend on the weights of the edges, since we could make the algorithm run longer by increasing the size of the edge weights, so that in particular just reading the edge weights could take an arbitrarily long time. There are two ways we can take account of this:

- We say that the algorithm is *polynomial* if its running time on a network N is a polynomial function of $|V(N)|$, $|E(N)|$ and $\log_2 k$, where $k = \max\{|w(e)| \ : \ e \in E(N)\}$. (We use $\log_2 k$ rather than k itself since it takes only $\log_2 k$ binary bits to input k into a computer.)

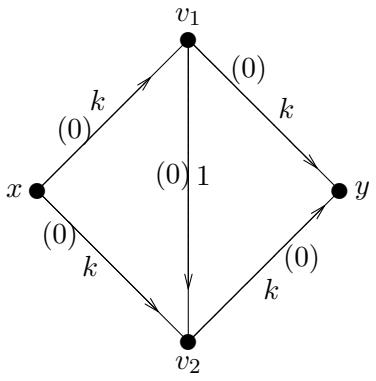
- We say that the algorithm is *strongly polynomial* if its running time on a network N is a polynomial function of $|V(N)|$ and $|E(N)|$), assuming that all elementary arithmetic operations (such as reading a number, comparing two numbers and choosing the largest, or adding or multiplying two numbers) take constant time, *no matter how large the numbers involved are.*

Since elementary arithmetic operations on numbers of size at most k can be performed in $O(log_2 k)$ time, all strongly polynomial network algorithms are polynomial. Our remarks in Chapter 3 imply that the network algorithms of Prim, Kruskal, Dijkstra, and Moravék are all strongly polynomial.

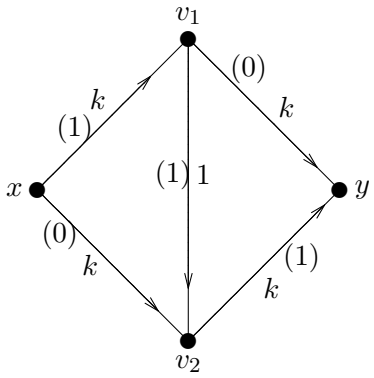
4.13 Lemma

The implementation of the Ford-Fulkerson algorithm given above is neither polynomial nor strongly polynomial.

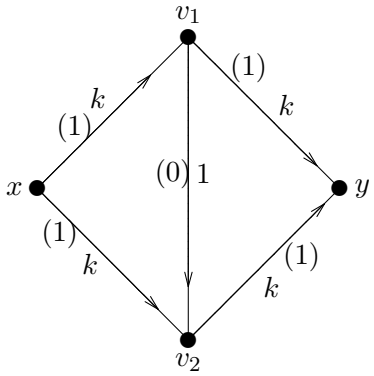
Proof Consider the following network N.



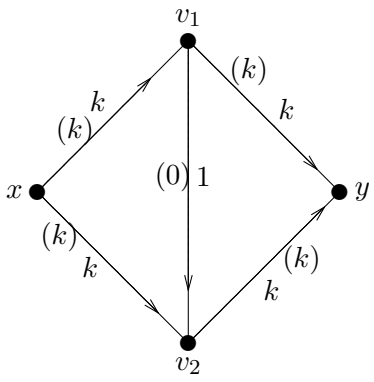
The capacities of the arcs xv_1, v_1y, xv_2, v_2y are all equal to k where k is a large positive integer. Suppose we run the Ford-Fulkerson algorithm on N. We start with the 'zero flow' f_1, by putting $f_1(e) = 0$ for all $e \in A(N)$. The first iteration of the algorithm could find the f_1-unsaturated xy-path $P_1 = xv_1v_2y$. The maximum amount of flow we can send along this path is $s(P_1) = 1$. This gives rise to the new xy-flow f_2 shown below which has value 1.



The second iteration of the algorithm could find the f_2-unsaturated xy-path $P_2 = xv_2v_1y$. The maximum amount of flow we can send along this path is $s(P_2) = 1$. This gives rise to the new xy-flow f_3 shown below which has value 2.



The third iteration of the algorithm could find the f_3-unsaturated xy-path $P_3 = xv_1v_2y$ and produce an xy-flow f_4 of value 3. We could continue in this way for $2k + 1$ iterations before we find the xy-flow of maximum value shown below.



Since k can be arbitrarily large, the running time is not bounded above by any (polynomial) function of $|V(N)|$ and $|E(N)|$, even if we assume that all elementary arithmetic operations take constant time. Hence the algorithm is not strongly polynomial. It is not polynomial since $2k+1$ is not a polynomial function of $|V(N)|$, $|E(N)|$ and $\log_2 k$.

4.14 Note

Dinits (1970), and Edmonds and Karp (1972) suggested a refinement of the Ford-Fulkerson algorithm which makes it strongly polynomial. They showed that if the Subroutine: Maximum Unsaturated Tree is implemented in such a way that it grows a breadth first search unsaturated tree, then the running time of the algorithm is $O(|V(N)|^2 \times |A(N)|)$ (assuming all elementary arithmetic operations take constant time).