Combinatorial Optimization 2019/2020

Week 1: Graph theory and network flows.

---

1 Basics of Graph Theory

Part of these notes are a translation of excerpts from Chapter 1 of the Dutch lecture notes *Grafen: Kleuren en Routeren* by Alexander Schrijver
`http://homepages.cwi.nl/~lex/files/graphs1_3.pdf`
Below, the numbering of the Exercises corresponds to the numbering in Schrijver's lecture notes.

A graph G is defined by a pair (V, E) where V is a finite set of points and E is a set of pairs of two (distinct) points. A graph is drawn by depicting the points in V as (thick) dots and the pairs in E as lines between two points. The points defining a graph are called vertices and the lines are called edges. The number of vertices is usually denoted by $|V| = n$ and the number of edges by $|E| = m$. A graph is called simple is there is at most one edge between every pair of points.

In general, we denote by $|X|$ the number of elements in a set X. For two sets X_1 and X_2 we denote by $X_1 \setminus X_2$ all elements that are in X_1 but not in X_2.

If for two vertices $u, v \in V$ we have the edge $e = \{u, v\} \in E$ then we say that u and v are adjacent. We also say that u and v are incident to e and, conversely, that e is incident to u and v. Two edges that share a vertex are also said to be adjacent.

The number of edges incident to vertex $v \in V$ is called the degree of v. Notation: $d(v)$. A graph is called regular if all vertices have the same degree. A graph is called k-regular if all vertices have degree k.

Exercise 1.7. How many 2-regular graphs exist with $V = \{1, 2, 3, 4, 5\}$?

Exercise 1.8. How many 3-regular graphs exist with $V = \{1, 2, 3, 4, 5\}$?

Exercise 1.11. How many edges has a 5-regular graph on 16 vertices?

Exercise 1.12. How many edges has a k-regular graph on n vertices?

Exercise 1.16. Prove that every graph has an even number of points with odd degree.

Exercise 1.19. Prove that a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ has a vertex with degree $\leq 2m/n$ and a vertex with degree $\geq 2m/n$.

Exercise 1.21. Prove that every graph $G = (V, E)$ with $|V| \geq 2$ has two vertices of the same degree.

A graph $G = (V, E)$ is complete if each pair of points is adjacent (defines an edge). A complete graph on n points is denoted by K_n.

A graph G is bipartite if V can be split into two sets V_1 and V_2 such that for each edge $e = \{u, v\} \in E$, we have $|e \cap V_1| = |e \cap V_2| = 1$. A complete bipartite graph with $V = V_1 \cup V_2$ has edge set $E = \{\{v_1, v_2\} \mid v_1 \in V_1,\ v_2 \in V_2\}$. The complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ is denoted by $K_{m,n}$.

Exercise 1.26. For which values of m and n is $K_{m,n}$ regular?

A walk in a graph $G = (V, E)$ is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that for all $i = 1, \ldots, k$, $\{v_{i-1}, v_i\} \in E$. It is called a walk is *from* v_0 *to* v_k. We say that this walk has (combinatorial) length k. If all vertices on the walk are distinct we call it a path.

Exercise 1.41. How many paths are there from vertex 1 to vertex 3 in K_3?

Exercise 1.42. How many paths are there from vertex 1 to vertex n in K_n?

Exercise 1.44. Prove that that a graph of which each vertex has degree at least k, has a path of length k.

A graph is called connected if there is a path between any two of its vertices.

Exercise 1.47. Prove that every connected graph on n vertices contains at least $n-1$ edges.

Exercise 1.48. Does there exist a non-connected graph on 6 vertices containing 11 edges?

Exercise 1.50. Prove that every non-connected graph on n vertices contains at most $\frac{1}{2}(n-1)(n-2)$ edges.

A graph $G' = (V', E')$ is called a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A component of $G = (V, E)$ is a maximal connected subgraph $G' = (V', E')$. That means, it is a connected subgraph and there is no other connected subgraph $G'' = (V'', E'')$ with $V' \subseteq V''$ and $E' \subseteq E''$.

Hence, a graph G is connected if and only if it consists of exactly one component.

Exercise 1.59. Prove (from the definition above) that if $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two distinct components of G then $V_1 \cap V_2 = \emptyset$.

Exercise 1.63. Prove that a graph $G = (V, E)$ with each vertex having degree at least $\frac{1}{2}(n-1)$ is connected.

Exercise 1.64. Prove that a graph $G = (V, E)$ has at least $|V| - |E|$ components.

Exercise 1.65. Prove that a graph with exactly two vertices with odd degree must contain a path between these two vertices.

A walk $(v_0, v_1, \ldots, v_k)$ is called a closed walk or a cycle if $v_0 = v_k$. A cycle with all vertices distinct is called a circuit. A circuit of length 3 is called a triangle.

Exercise 1.67. Let G be a graph for which every vertex has a degree of at least 2. Prove that G contains a circuit.

$G = (V, E)$ is called a forest if G does not contain any circuit. A connected forest is called a tree.

Exercise 1.75. Prove that between any pair of vertices in a tree there is exactly one path.

A vertex of degree 1 in a tree is called a leaf of the tree.

Exercise 1.76. Prove that every tree with at least two vertices contains a leaf (cf. Exercise 1.67).

Exercise 1.77. Derive from the previous exercise that every tree on n vertices has exactly $n - 1$ edges.

Exercise 1.79. Prove that a forest on n vertices consisting of k components contains exactly $n - k$ edges.

Exercise 1.80. Prove that every tree with at least two vertices contains at least two leaves.

An Euler cycle in a graph $G = (V, E)$ is a cycle $C = (v_0, v_1, \ldots, v_k)$ (remember that $v_0 = v_k$), with the property that every edge $e \in E$ is traversed exactly once; i.e., for each edge $e \in E$ there exists exactly one

$i \in \{1, \ldots, k\}$ such that $e = \{v_{i-1}, v_i\}$. A graph is called an Euler graph if it contains an Euler cycle. An Euler path in a graph $G = (V, E)$ is a path $C = (v_0, v_1, \ldots, v_k)$ with the property that every edge $e \in E$ is traversed exactly once, but it is not required that $v_0 = v_k$.

Theorem 1.1 (Euler's Theorem). A graph $G = (V, E)$ is an Euler graph if and only if G is connected and each of its vertices has even degree.

Exercise 1.86 Let G be an Euler graph with an even number of edges. Let $d_1, d_2, \ldots, d_n$ be the degrees of the points. Show that there exists a subgraph with degrees $d_1/2, d_2/2, \ldots, d_n/2$.

Exercise 1.88. Let G be a connected graph with exactly two points of odd degree. Use Euler's Theorem to prove that G contains a walk that traverses each edge exactly once.

A circuit C of G is called a Hamilton circuit (or Hamilton cycle) if each vertex of G appears on C. A graph G is called a Hamilton graph if it contains a Hamilton circuit. Hamiltonian cycles / graphs are also called Rudrata cycles / graphs (after a 9th century Indian mathematician)

Exercise 1.90 Let n be an odd number. Show that on an $n \times n$ chess board, it is not possible for a knight (horse) to move over the board, hitting each square exactly once, while starting and ending in the same square.

Exercise 1.91. Show that for each n there exists a graph on n vertices such that each vertex has degree at least $\frac{1}{2}n - 1$ and such that it is not a Hamilton graph.

A matching M is a subset of the edges ($M \subseteq E$) such that no the edges in M have an end point in common. The size of the matching M is the number of edges in it: $|M|$. A maximum macthing in a graph is a matching of maximum size. A perfect matching is a matching for which each vertex is the end point of some edge in the matching, i.e., $2|M| = |V|$. Clearly, a graph with an odd number of vertices can not have a perfect matching.

Exercise X.1 Give an example of a connected graph with an even number of vertices that does not have a perfect matching.

A directed graph G is defined by a pair (V, A) where V is a finite set of points and A is an ordered set of pairs (arcs) of two (distinct) points. For arc (u, v) we call u the tail and v the head of the arc. We use the following notation for directed graphs:

$\delta^-(v)$: the set of arcs with head v (the arcs towards v)
$\delta^+(v)$: the set of arcs with tail v (the arcs leaving v)

$d^-(v)$: the number of arcs with head v (so $d^-(v) = |\delta^-(v)|$)
$d^+(v)$: the number of arcs with tail v (so $d^+(v) = |\delta^+(v)|$)

2 Network flow algorithms

2.1 Flows

The networks we consider consists of a directed graph $G = (V, A)$, two special nodes $s, t \in V$ which are, respectively, a source and sink of G, and capacities $c_a > 0$ on each arc $a \in A$. (Notation: we also write c_{uv} for $a = (u, v)$.)

A flow f consists of a number $f_a \geq 0$ for each $a \in A$ such that

1. the capacity constraints are met: $0 \leq f_a \leq c_a$ for each $a \in A$,
2. flow conservation is met, that means, the amount of flow leaving a point u (other than the source or sink) is equal to the amount of flow entering u:

$$\sum_{(u,v)\in A} f_{uv} = \sum_{(v,u)\in A} f_{vu} \quad \text{for all } u \in V \setminus \{s, t\}.$$

The value of the flow is the nett flow that leaves s:

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(v,s)\in A} f_{vs}.$$

To simplify notation, we define from now on $f_{uv} = 0$ if $(u, v) \notin A$. Then we can write the value of the flow as

$$\sum_{v\in V} (f_{sv} - f_{vs}).$$

2.2 Cuts

For a set $U \subset V$ define
$\delta^-(U)$: the set of arcs with head in U and tail in $V \setminus U$ (arcs towards U)
$\delta^+(U)$: the set of arcs with tail in U and head in $V \setminus U$ (arcs leaving U)

If $U \subset V$ with $s \in U$ and $t \in V \setminus U$ then $\delta^+(U)$ is called an s-t cut. We call this the cut defined by U. The next theorem says that the value of the flow is equal to the nett flow across any s-t cut.

Theorem 1 *Let f be a flow and $\delta^+(U)$ an s-t cut then*

$$value(f) = \sum_{a \in \delta^+(U)} f_a - \sum_{a \in \delta^-(U)} f_a.$$

Equivalently, when we define $f_{uv} = 0$ if $(u, v) \notin A$, then we can write this theorem as

$$value(f) = \sum_{u \in U} \sum_{v \in V \setminus U} f_{uv} - f_{vu}.$$

Proof: By flow conservation we have that for any $u \in U$ with $u \neq s$

$$\sum_{v \in V} f_{uv} - f_{vu} = 0.$$

Also,

$$\sum_{u \in U} \sum_{v \in U} f_{uv} - f_{vu} = 0,$$

since each arc (u, v) with $u, v \in U$ appears once positive and once negative in the summation. Using both these equation we get

$$\text{value}(f) := \sum_{v \in V} (f_{sv} - f_{vs}) = \sum_{u \in U} \sum_{v \in V} f_{uv} - f_{vu} = \sum_{u \in U} \sum_{v \in V \setminus U} f_{uv} - f_{vu}.$$

□

The capacity of a cut $\delta^+(U)$ is the total capacity of the arcs in the cut:

$$\text{cap}(\delta^+(U)) = \sum_{a \in \delta^+(U)} c_a = \sum_{\substack{(u,v) \in A: \\ u \in U, v \in V \setminus U}} c_{uv}$$

To simplify notation we write $\text{cap}(\delta^+(U))$ as $\text{cap}(U)$ and define $c_{uv} = 0$ if $(u, v) \notin A$. Then we can write the capacity of the cut defined by U as

$$\text{cap}(U) = \sum_{u \in U} \sum_{v \in V \setminus U} c_{uv}. \tag{1}$$

Theorem 2 *For any s-t flow f and s-t cut $\delta^+(U)$:*

$$value(f) \leq cap(U).$$

Proof: By Theorem 1

$$\text{value}(f) = \sum_{u\in U}\sum_{v\in V\setminus U} f_{uv} - f_{vu} \leq \sum_{u\in U}\sum_{v\in V\setminus U} f_{uv} \leq \sum_{u\in U}\sum_{v\in V\setminus U} c_{uv} = \text{cap}(U).$$

The theorm implies that the maximum value of a flow is no more than the minimum capacity of a cut. In fact, equality holds and the proof follows from the algorithm to find a maximum flow.

2.3 MaxFlow: The Ford-Fulkerson (FF) algorithm

The algorithm starts with zero flow: $f_e = 0$ for all $e \in A$. Then, it repeatedly chooses an appropriate path from s to t and increases the flow along the arcs of this path as much as possible.
It is important to note that it may be *necessary to reduce flow* along an arc in order to increase the flow over the path. In the example below, the first number is the flow and the second number is the capacity. The maximum flow is 2. However, the initial flow f_1 of value 1 as shown in the figure cannot be extended without reducing the flow on the arc in the middle.

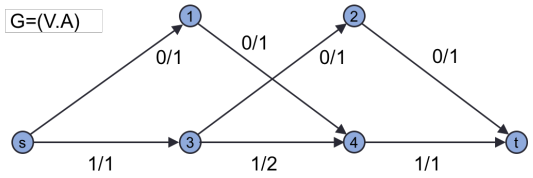


Figure 1: The initial flow f_1 has value 1.

Given a flow f, the residual graph (or residual network) gives for any arc the amount by which the current flow on the arc can be altered. For any arc $(u, v) \in A$, there is an arc (u, v) in the residual graph if there is still capacity left: $f_{uv} < c_{uv}$. But also, there is a reversed arc (v, u) in the residual graph if $f_{uv} > 0$. The residual graph for our example is as follows. The numbers show the amount by which the current flow can be extended, or, if the arc is reversed, the amount by which the current flow can be reduced. The graph has an st-path, namely $s, 1, 4, 3, 2, t$. The minimum capacity on the path is 1. So we add 1 over this path. The new flow has value 2 which is the maximum. A more complex example of the FF algorithm is given in the slides.

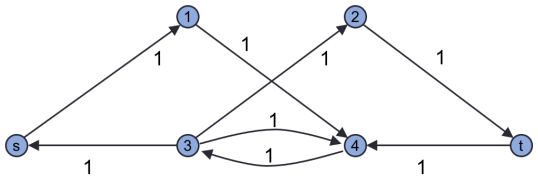


Figure 2: The residual network for f_1.

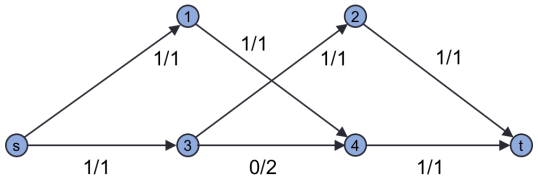


Figure 3: Flow f_2 of value 2 is optimal.

Theorem 3 *If all capacities are integer then the FF algorithm terminates and the flow f_a is integer for each arc $a \in A$.*

Proof: The flow is increased by an integer value in each iteration, so the number of iterations is no more than, for example, $\sum_v c_{sv}$, which is an upper bound on the flow that can leave s. Also, since the increase is integer in each iteration, the final flow is integer on each arc.

Proof of optimality Consider the residual graph when the FF-algorithm terminates. Let U be the set of point that can be reached from s by a directed path. Clearly, $s \in U$ and $t \notin U$ since otherwise there would be a flow augmenting path. So U defines an s-t cut. Since there is no arc from U to $V \setminus U$ we have that:

- for any arc $(u, v) \in A$ with $u \in U$ and $v \in V \setminus U$ we have $f_{uv} = c_{uv}$,
- for any arc $(v, u) \in A$ with $u \in U$ and $v \in V \setminus U$ we have $f_{vu} = 0$.

Now, by Theorem 1,

$$\text{value}(f) = \sum_{u \in U} \sum_{v \in V \setminus U} f_{uv} - f_{vu} = \sum_{u \in U} \sum_{v \in V \setminus U} c_{uv} = \text{cap}(U).$$

So the FF-algorithm returns a flow with value equal to the capacity of cut U. Let MaxFlow denote the maximum flow value and MinCut denote the smallest capacity of an s,-t cut. Then,

$$\text{MaxFlow} \geq \text{value}(f) = \text{cap}(U) \geq \text{MinCut}.$$

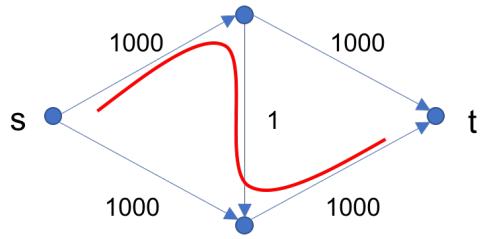
On the other hand, Theorem 2 states that

$$\text{MaxFlow} \leq \text{MinCut}.$$

Combining both we get the following theorem

Theorem 4 (MaxFlow=MinCut) *The maximum value of an s-t flow is equal to the minimum capacity of s-t cut and FF-algorithm returns both a maximum flow and a minimum cut.*

Running time The number of iterations of the FF algorithm may be as large as the value of the flow. If the algorithm always chooses the arc in the



middle on the path then the flow is extended by 1 in each iteration and it will take 2000 iterations to find the maximum flow. Clearly, two iterations would be enough here. With a small adjustment to the algorithm, we can guarantee that the running time only depends on the size of the network and not on the capacities as in the example above.

2.4 MaxFlow: Edmonds-Karp-Dinitz (EKD) algorithm

The algorithm applies the FF algorithm but in each iteartion it chooses the s-t path in the residual graph with the *minimum number* of arcs.

Theorem 5 *The number of iterations of the EKD algorithm is* $O(nm)$

Proof: See slides.

2.5 Mincost flow

In the minimum cost flow problem we are given a network $G = (V, A)$ with $s, t \in V$ and a capacity c_a for any arcs a and, in addition, a cost that we denote by cost_a. The cost of a flow f is

$$\sum_{a \in A} \text{cost}_a f_a.$$

By this definition, the minimum cost (zero) is attained by sending no flow. In the minimum cost flow problem we want to find, *for a given flow value* v, a flow of minimum cost among the flows of value v.

A mincost flow can be computed as follows. First, find any flow f of value v, which can be computed by, for example, the FF algorithm. Next, make the residual network as in FF but now for each reversed arc also make the cost negative. (Note that sending a flow over a reversed arc in the residual corresponds with reducing the flow.) Let C be a cycle in the residual graph. If we augment f by sending flow over C then the value of the flow remains v. However the cost may change: If the sum of the cost of the arcs in C is negative then the cost of the flow will decrease.

Mincost flow algorithm: (The cycle cancelling algorithm)

- Step 1: Find a feasible flow of value v. Make the residual graph.
- Step 2: While there is a negative-cost cycle C in the residual graph:
  - add the largest possible flow over C,
  - update the residual graph.

Theorem 6 *The mincost flow algorithm returns a minimum cost flow.*

Proof:(Only a sketch) Let f be the flow returned by the algorithm. Assume that f' is a cheaper flow of value v. We shall prove that this is not possible. Define $f' - f$ as the flow that sends $f'_{uv} - f_{uv}$ from u to v for each pair u, v. Note that $f' - f$ is a feasible flow in the residual graph since $f + (f' - f) = f'$ is a feasible flow in the original network G. The value of $f' - f$ is $v - v = 0$. A flow of value zero is called a circulation. By assumption, the cost of $f' - f$ is strictly less than zero. Any circulation is the sum of flows over cycles.

Since the cost is strictly less than zero, at least one of these cycles, say C, must have a negative cost. Since, $f' - f$ is feasible in the residual, cycle C is feasible in the residual. However, by definition of the algorithm, there are no negative cost cycles in the residual.

3 Flow exercises

Exercise 1 Show that for any flow, the total (nett) flow leaving s is equal to the total (nett) flow entering t:

$$\sum_{v \in V} (f_{sv} - f_{vs}) = \sum_{v \in V} (f_{vt} - f_{tv}).$$

Exercise 2 Consider the following problem: There are p families going out for dinner and together they use q tables. No two members of a family should sit at the same table. Let a_i be the number of people in family i ($i = 1, 2, \ldots, p$) and table j has place for b_j persons ($j = 1, \ldots, q$). Formulate this problem as a maximum flow problem. (Make a sketch of the network including the capacities on the arcs.)

Exercise 3 In this exercise we will see the relation between the maxflow-mincut theorem and LP-duality.

Consider a network $G = (V, A)$ with source s and sink t. The arcs are labeled $1, 2, \ldots, m$ ($|A| = m$). Let c_j be the capacity of arc j. Let p be the number of directed s-t paths and label these paths $1, 2, \ldots, p$. Let $q_{ij} = 1$ if arc i is on path j and let $q_{ij} = 0$ otherwise. Then the next LP is an exact formulation of the maximum flow problem.

$$\begin{array}{lll} \max & \sum_{i=1}^{p} x_i & \\ s.t. & \sum_{i=1}^{p} q_{ij} x_i \leqslant c_j, & \text{for all } j = 1, \ldots, m \\ & x_i \geqslant 0 & \text{for all } i = 1, \ldots, p. \end{array} \tag{2}$$

(a) Give the corresponding LP-formulation for the network below.

(b) Give a maximum flow and its corresponding LP-solution.

(c) Give the dual of this LP.

(d) Find a solution to the dual with value equal to the primal solution found in (b).

(e) Give an interpretation of this dual solution in terms of the network.

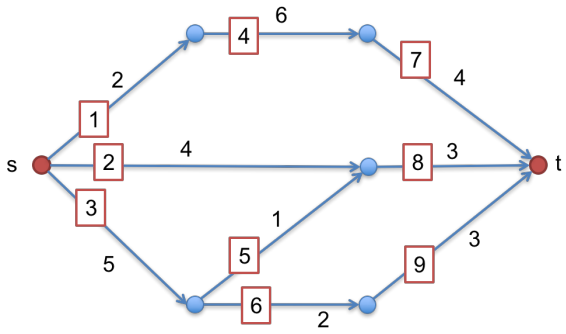
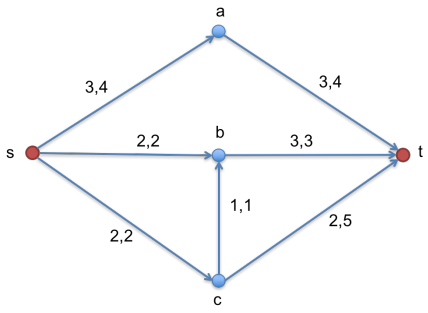


Figure 4: The labels in squares give the numbers of the arcs.

Exercise 4 Explain how you can find a maximum matching in a bipartite graph by using a maxflow algorithm. That means, formulate the matching problem as a flow problem and explain how you can deduce the maximum matching from the maximum flow.

Exercise 5 In the network below, each first number is the arc's capacity and the second number is its cost per unit flow. Find a minimum cost flow of value 3 by following the steps of the cycle cancelling algorithm. Start with a flow of value 3 over the path s, a, t.



Exercise 6 Consider a flow network with integer capacities. Prove or disprove the following statements.

(a) If all capacities are even, then there is maximum flow in which f_a is even for all $a \in A$.
(b) If all capacities are odd, then there is maximum flow in which f_a is odd for all $a \in A$.

Exercise 7 (a) An arc a in the flow network is called *upwards critical* if increasing the capacity of a increases the value of the maximum flow. Does every network possess an upwards critical arc?
(b) An arc a in the flow network is called *downwards critical* if decreasing the capacity of a decreases the value of the maximum flow. Does every network possess a downwards critical arc?