# *Cat vs dog*

BIPARTITE GRAPHS, network flows, matchings and vertex covers are the topics of the problem [1] in this note.

[1] Spotify. Cat vs dog. 2012. URL https://labs.spotify.com/puzzles/

**Problem**

The latest reality show has hit the TV: "Cat vs. Dog". In this show, a bunch of cats and dogs compete for the very prestigious Best Pet Ever title. In each episode, the cats and dogs get to show themselves off, after which the viewers vote on which pets should stay and which should be forced to leave the show.

Each viewer gets to cast a vote on two things: one pet which should be kept on the show, and one pet which should be thrown out. Also, based on the universal fact that everyone is either a cat lover (i.e. a dog hater) or a dog lover (i.e. a cat hater), it has been decided that each vote must name exactly one cat and exactly one dog.

Ingenious as they are, the producers have decided to use an advancement procedure which guarantees that as many viewers as possible will continue watching the show: the pets that get to stay will be chosen so as to maximize the number of viewers who get both their opinions satisfied. Calculate this maximum number of satisfied viewers.

At first glance this looks similar to a SAT problem [2], something like $(c_1 \wedge \neg d_3)$, $(c_3 \wedge \neg d_1)$, $(d_2 \wedge \neg c_2)$, ... where $c_i$ are the cats and $d_j$ are the dogs. The goal would be to pick the biggest subset of boolean expressions (votes) that are satisfied.

But SAT is about one boolean expression and about assigning values to boolean variables to satisfy it. Seems like SAT is fundamentally different and not a good approach in solving this problem. What if we want to visualize the boolean expressions and see the relationships between them, i.e. which ones are in conflict. Conflict between two boolean expressions means one expression has $c_i$ and the other expression has $\neg c_i$ or one has $d_j$ and the other $\neg d_j$. A good way to do that is with a graph as in Figure 1.1. The nodes in the graph are the boolean expressions and edges connect boolean expressions that are in conflict.

It becomes apparent that the graph is bipartite with cat lovers on one side and dog lovers on the other. That is good because a lot of graph algorithms are much simpler and work faster if the graphs are bipartite. But what algorithm should we use? We need to find the biggest subset of nodes in the graph that are not in conflict.

Sometimes it's easier to compute the complement of what we want:

[2] Boolean satisfiability problem http://en.wikipedia.org/wiki/Boolean_satisfiability_problem

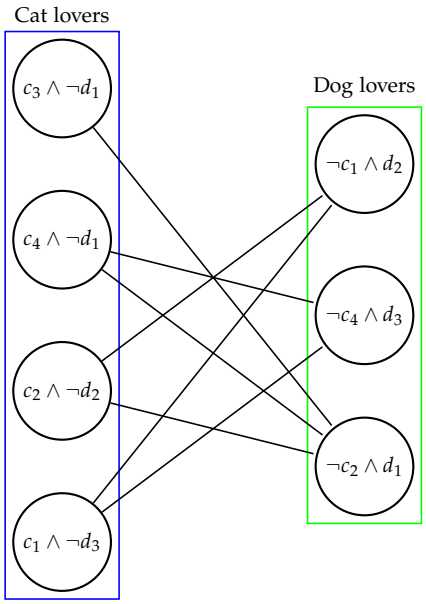



Figure 1.1: Votes form a bipartite graph. A graph is **bipartite** if the vertex set is partitioned into two subsets (blue and green in this case) such that no vertices in a subset are adjacent.

the smallest subset of nodes that are involved in conflicts. Removing these nodes and the edges they touch should leave us with a graph with only nodes and no edges, i.e. only votes without conflicts. Because we strive to remove the smallest subset of conflicting nodes we are left with the biggest subset of votes without conflicts.

The subset of nodes that are involved in conflicts is a vertex cover [3] for our bipartite graph.

We need to compute a minimum vertex cover. This will be a good excuse to learn about network flows in graphs, maximum flows and minimum cuts. This delightful detour will eventually bring us to maximum matchings[4] and then finally to minimum vertex covers.

[3] A **vertex cover** is a subset of nodes such that each edge in a graph is incident to at least one vertex in the subset.

[4] A **matching** is a subset of edges such that no two edges in the subset share a vertex.

We begin with **network flows** in graphs. We work with a directed graph $G = (V, E)$ that has two special vertices $s$ and $t$ called **source** and **target**. No edge goes into *source* and no edge comes out of *target*. We also have a function $c : E \to \mathbb{R}_{\geq 0}$ that assigns a non-negative capacity to each edge. The graph $G$ together with source $s$ and target $t$ and capacity function $c$ form a **network** $(G = (V, E), s \in V, t \in V, c)$.

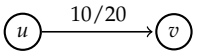



Figure 1.2: In figures we annotate an edge with flow and capacity as shown here. In this case $f(u \to v) = 10$ and $c(u \to v) = 20$. If only one number is annotating the edge then it's the capacity.

**Definition 1.1.** A function $f : E \to \mathbb{R}_{\geq 0}$ is a **flow** through network $(G, s, t, c)$ if $f$ satisfies the following constraints:

- *capacity constraint*: flow along an edge cannot exceed the capacity of the edge
$$\forall e \in E : f(e) \leq c(e)$$
- *conservation constraint*: incoming flow into a vertex (except for source and target) equals outgoing flow from the vertex
$$\forall v \in V \setminus \{s, t\} : \sum_u f(u \to v) = \sum_w f(v \to w)$$

Source $s$ generates flow and target $t$ consumes flow. The **value** of flow $f$, denoted $|f|$, is defined as

$$|f| = \sum_w f(s \to w) = \sum_v f(v \to t)$$

Given a network $(G, s, t, c)$ what is the maximum flow value that can be pumped through it? Figure 1.3 shows a flow of value 20 through an example network. It saturates the flow along one particular path and avoids the other edges. Is 20 the maximum flow value that can be achieved for this example network? Figure 1.4 shows the same network but now with a flow of value 30. Can we do better than 30? The answer is no, because that would exceed the outgoing capacity of source $s$ or the incoming capacity of $t$.

Our goal is to device an algorithm that constructs a flow with maximum value through a given network. To gauge the progress of our

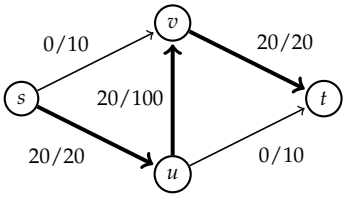



Figure 1.3: Example network flow. Here $|f| = 20$ and the whole flow is pumped along the path $s \to u \to v \to t$. In this exampe $f$ **saturates** $s \to u$ and $v \to t$ and **avoids** $s \to v$ and $u \to t$.

For notational simplicity we assume functions $f$ and $c$ are defined on $V \times V$ and $f(u \to v) = c(u \to v) = 0$ if $u \to v$ is not an edge in $G = (V, E)$.

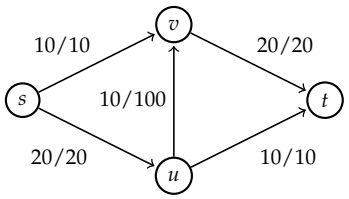



Figure 1.4: Same example network with a flow of value $|f| = 30$.

algorithm we need an upper bound for the maximum flow value. As said before the maximum value clearly cannot exceed the outgoing capacity of source $s$ or the incoming capacity of $t$. But more generally if we sever the ties between source and target along some subset of edges such that there are no more paths from source to target then the maximum flow value cannot exceed the capacity of the cut. This seems like a useful concept to formalize.

**Definition 1.2.** In a network $(G, s, t, c)$ a **cut** is a partition of the vertex set $V$ into two subsets $S$ and $T$, such that $V = S \cup T$, $S \cap T = \emptyset$ and $s \in S, t \in T$. The **capacity** of the cut $(S, T)$, denoted $\|S, T\|$, is defined as

$$\|S, T\| = \sum_{v \in S} \sum_{w \in T} c(v \to w)$$

**Theorem 1.3.** *With network $(G, s, t, c)$, for any flow $f$ and any cut $(S, T)$ we have*

$$|f| \leq \|S, T\|$$

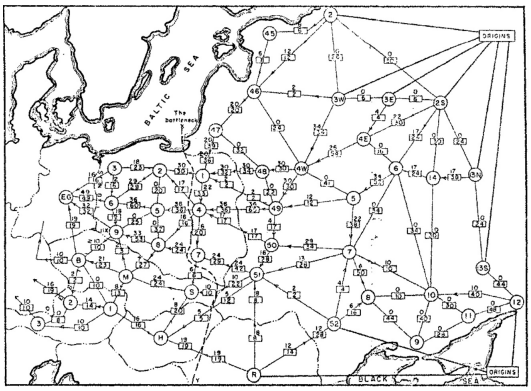
*Furthermore equality holds if and only if $f$ saturates every edge from $S$ to $T$ and avoids every edge from $T$ to $S$.*

*Proof.*

$$\begin{aligned}
|f| &= \sum_w f(s \to w) && \text{(by definition)} \\
&= \sum_w f(s \to w) - \sum_v f(v \to s) && \text{(second sum terms are all zero)} \\
&= \sum_{u \in S} (\sum_w f(u \to w) - \sum_v f(v \to u)) && \text{(flow conservation constraint)} \\
&= \sum_{u \in S} (\sum_{w \in T} f(u \to w) - \sum_{v \in T} f(v \to u)) && \text{(edges in } S \text{ cancel each other out)} \\
&\leq \sum_{u \in S} \sum_{w \in T} f(u \to w) && \text{(because } f(v \to u) \geq 0) \\
&\leq \sum_{u \in S} \sum_{w \in T} c(u \to w) && \text{(flow capacity constraint)} \\
&= \|S, T\| && \text{(by definition)}
\end{aligned}$$

□

Theorem 1.3 tells us that if we keep increasing a flow and/or decreasing a cut we should eventually meet at a maximum flow that equals a minimum cut. But given a network how do we start? A first valid flow is $\forall e \in E : f(e) = 0$. We could then try a greedy strategy. Starting with source $s$ find the path to $t$ with the biggest capacity[5] and pump as much flow as we can through it as illustrated in Figure 1.3. Unfortunately we are stuck at that point. We cannot pump more flow out of $s$ on $s \to v$ because that would violate flow conservation at $v$




Alexander Schrijver. On the history of the transportation and maximum flow problems. 2002. URL http://homepages.cwi.nl/~lex/files/histtrpclean.pdf:
Network flows and minimum cuts played a role in the Cold War. The figure is a schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as "The bottleneck".

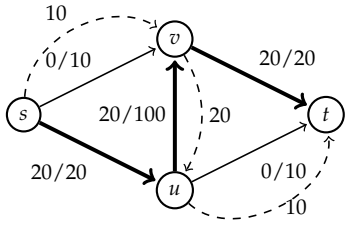



Figure 1.5: Dashed edges show how the greedy $s - t$ path flow can be augmented and reversed in order to increase overall flow.

[5] The capacity of a path is the minimum over the capacities of the edges forming the path.

(we are at the maximum outgoing flow at $v$). The dashed edges in Figure 1.5 show some of our options. On edges where the current flow leaves residual capacity we can pump more and on edges where there is existing flow we can reverse it. Again, this concept seems worth formalizing.

**Definition 1.4.** A flow $f$ in a network $(G, s, t, c)$ induces a **residual network** $(G_f, s, t, c_f)$ with **residual graph** $G_f$ and **residual capacity** $c_f$ in the following way:

- all vertices from $G$ are vertices in $G_f$, also source $s$ and target $t$ are the same in $G$ and $G_f$
- if $f(u \to v) > 0$ then $G_f$ has an edge $(v \to u)$ with capacity
$$c_f(v \to u) = f(u \to v)$$
- if $f(u \to v) < c(u \to v)$ then $G_f$ has an edge $(u \to v)$ with capacity
$$c_f(u \to v) = c(u \to v) - f(u \to v)$$

Figure 1.6 shows the residual network of our example network and flow. We observe that there is a simple path[6] $s \to v \to u \to t$ with capacity 10 from source $s$ to target $t$ in the residual graph. This path shows that there still is unused capacity for flow to be pushed from $s$ to $t$. A simple path from $s$ to $t$ in $G_f$ is called an **augmenting path**.

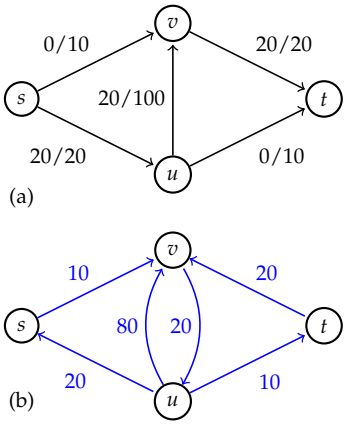



Figure 1.6:
(a) Example network with flow from Figure 1.3.
(b) Residual network (in blue) with edges annotated with their residual capacity.

[6] A simple path is a path where every vertex on the path is visited only once.

**Theorem 1.5.** *Given is a flow $f$ in network $(G, s, t, c)$. If there is an augmenting path in $G_f$ with capacity $F$ then the function $f' : V \times V \to \mathbb{R}_{\geq 0}$ defined as:*

$$f'(u \to v) = \begin{cases} f(u \to v) + F, & \text{if } u \to v \text{ is on the augmenting path} \\ f(u \to v) - F, & \text{if } v \to u \text{ is on the augmenting path} \\ f(u \to v), & \text{otherwise} \end{cases}$$

*is a valid flow in network $(G, s, t, c)$ with $|f'| = |f| + F$.*

*Proof.* We need to check the capacity constraint and the conservation constraint.

Let's start with the capacity constraint. The definition of $f'$ has three cases, so we check all three:

- Edge $u \to v$ is on the augmenting path:

$$\begin{aligned} f'(u \to v) &= f(u \to v) + F && \text{(by definition)} \\ &\leq f(u \to v) + c_f(u \to v) && \text{(by definition of } F\text{)} \\ &= f(u \to v) + c(u \to v) - f(u \to v) && \text{(by definition of } c_f\text{)} \\ &= c(u \to v) \end{aligned}$$

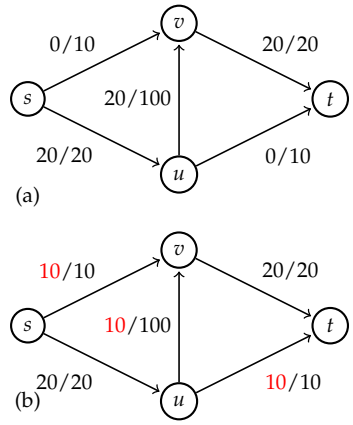



Figure 1.7:
(a) Example network with flow from Figure 1.3.
(b) Augmented flow (changed values in red) from augmenting path $s \to v \to u \to t$.

- Edge $v \to u$ is on the augmenting path:

$$\begin{aligned} f'(u \to v) &= f(u \to v) - F && \text{(by definition)} \\ &\geq f(u \to v) - c_f(u \to v) && \text{(by definition of } F\text{)} \\ &= f(u \to v) - f(u \to v) && \text{(by definition of } c_f\text{)} \\ &= 0 \end{aligned}$$

- Otherwise: In this case the flow of the edge hasn't changed so capacity constraint is satisfied.

Next is the conservation constraint. For vertices not on the augmenting path flow in and out of them hasn't changed, so conservation constraint is satisfied there. For a vertex $v$ on the augmenting path we have four cases (since the augmenting path is simple and $v \neq s, v \neq t$):

- $u \to v$ on augmenting path and $v \to w$ on augmenting path: in this case one incoming edge into $v$ changed by $F$ and one outgoing changed by $F$, so conservation constraint holds for $v$
- $u \to v$ on augmenting path and $w \to v$ on augmenting path: in this case two incoming edges into $v$ changed, one by $F$ and the other by $-F$, so conservation constraint holds for $v$
- $v \to u$ on augmenting path and $w \to v$ on augmenting path: in this case one incoming edge into $v$ changed by $-F$ and one outgoing changed by $-F$, so conservation constraint holds for $v$
- $v \to u$ on augmenting path and $v \to w$ on augmenting path: in this case two outgoing edges from $v$ changed, one by $-F$ and the other by $F$, so conservation constraint holds for $v$

□

What happens when there is no augmenting path in $G_f$? As the Figure 1.8 hints we then have a maximum flow (in our example $|f| = 30$). The next theorem proves it.

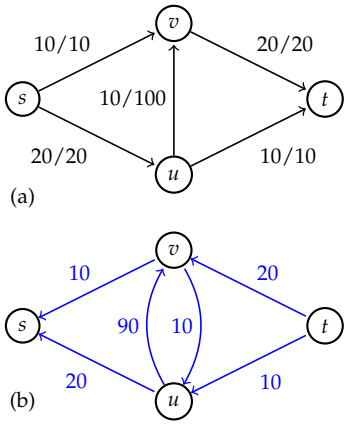



Figure 1.8:
(a) Example network with flow from Figure 1.4.
(b) Residual network (in blue) with edges annotated with their residual capacity.

**Theorem 1.6.** *Given is a flow $f$ in network $(G, s, t, c)$. If there is no augmenting path in $G_f$ then $f$ is a flow with maximum value.*

*Proof.* We define two subsets of $V$. The set $S$ holds all the vertices of $V$ that are reachable from $s$ in $G_f$. Since there is no augmenting path in $G_f$ we have $t \notin S$. We also define $T = V \setminus S$. Clearly $(S, T)$ is a cut of our network. Also there is no $G_f$ edge $u \to v$ with $u \in S$ and $v \in T$ because otherwise $v$ would be reachable from somewhere in $S$ but $v \notin S$, contradicting the definition of $S$. This means (by definition

of $G_f$) that $f$ saturates every edge from $S$ to $T$ and avoids every edge from $T$ to $S$. According to Theorem 1.3 we then have $|f| = \|S,T\|$ which means we have a maximum flow and minimum cut.

□

We can now piece together the following algorithm known as the **Ford-Fulkerson algorithm**:

Listing 1.1: Ford-Fulkerson algorithm

```
f = zero flow;
Gf = residual graph of f in G;

while (exists augmenting path in Gf):
   pa = choose any augmenting path;
   f = augment f with pa;
   Gf = residual graph of f in G;

return f
```

Delbert Ray Fulkerson was an American mathematician who co-developed the Ford–Fulkerson algorithm. https://en.wikipedia.org/wiki/D._R._Fulkerson

**Theorem 1.7.** *If the network has capacities in $\mathbb{N}_{\geq 0}$ then the Ford-Fulkerson algorithm terminates and returns the maximum flow in the network.*

*Proof.* We prove by induction that $f : V \times V \to \mathbb{N}_{\geq 0}$: The base case is the zero flow which is in $\mathbb{N}_{\geq 0}$. Assume the current flow values are in $\mathbb{N}_{\geq 0}$. The augmenting operation adds or subtracts a positive integer value from the current flow values and conforms to capacity constraints, so it keeps the augmented flow values in $\mathbb{N}_{\geq 0}$ which completes the induction.

The augmented flow $f'$ modifies one outgoing edge from $s$ by $F > 0$, so by the definition of the value of a flow we have $|f'| = |f| + F$. This means that augmenting strictly increases the value of the flow. We also know that flow values have an upper bound (by Theorem 1.3 any cut capacity is an upper bound). This means the algorithm has to eventually reach the maximum flow and terminate. □

This concludes our detour into network flows[7].
We should bring it back to our problem and the associated bipartite graph of conflicting votes. We want a minimal vertex cover and we would like to use the just derived Ford-Fulkerson algorithm to compute it. So we first have to transform our undirected bipartite graph into a network.

We have an undirected bipartite graph $G(V = X \cup Y, E)$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$ ($X$ could be the votes of cat lovers and $Y$ the votes of dog lovers in our problem or vice versa). We add a source $s$ and a target $t$ and construct a network $(G', s, t, c)$ in the following way:

- vertex set of $G'$ is $X \cup Y \cup \{s, t\}$

[7] We have just scratched the surface of the topic on network flows and algorithms computing maximum flows (an area of active research). For example by making smart choices when choosing the augmenting path we can improve the runtime of the algorithm (also we haven't analyzed the runtime). What happens when the capacities are not in $\mathbb{N}_{\geq 0}$. For details on all this and more see:
Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358
Jeff Erickson. *Algorithms, Etc.* 2015. URL http://jeffe.cs.illinois.edu/teaching/algorithms/.

- $\forall u \in X$ add a directed edge $s \to u$ into edge set of $G'$
- $\forall v \in Y$ add a directed edge $v \to t$ into edge set of $G'$
- $\forall \{u, v\}$ undirected edge in $G$ with $u \in X$ and $v \in Y$ add a directed edge $u \to v$ into edge set of $G'$
- unit capacity[8]: $\forall (u \to v) \in$ edge set of $G' : c(u \to v) = 1$

[8] Unit capacity has the advantage that a flow either saturates the edge or avoids it.

With a network $(G', s, t, c)$ constructed from a bipartite graph $G$ as described above (an example is shown in Figure 1.9) we have an equivalence between a matching in the bipartite graph and a flow in the network. The next theorem states this.

**Theorem 1.8.** *A matching $M$ in $G$ induces a flow $f$ in $G'$ such that $|f| = |M|$. Conversely a flow in $G'$ induces a matching $M$ in $G$ such that $|M| = |f|$.*

*Proof.* ($\Rightarrow$) We have a matching $M$ in $G$, i.e. a subset of edges that don't share a vertex. From the construction of $G'$ it follows that each of the edges in $M$ can be extended to paths from $s$ to $t$ which will only meet in $s$ and $t$. We define a function $f$ that gives unit values to the edges along these paths and zero value to all other edges. We claim that $f$ is a valid flow. It only assigns zero or unit values so it does satisfy the capacity constraint in $G'$. The paths don't intersect except in $s$ and $t$ (because $M$ is a matching), so for any vertex along the path there is exactly one incoming edge with unit value and one outgoing edge with unit value. The rest of the edges have value zero so don't play a role in conservation. This then means that the conservation constraint is satisfied also and $f$ is a flow. Each edge in $M$ corresponds to one of the paths, so there are $|M|$ edges outgoing from $s$ that have unit value. Hence $|f| = |M|$.
($\Leftarrow$) We have a flow $f$ in $G'$. The flow either saturates or avoids an edge. We define the subset $M$ of edges that are saturated by $f$ and are between $X$ and $Y$. We claim that $M$ is a matching in $G$. Suppose it is not a matching. Then there exists a vertex that is shared by two edges in $M$. If this vertex is in $X$ then it means it has two outgoing edges of unit value but only one incoming edge of unit value (from $s$). If this vertex is in $Y$ then it means it has two incoming edges of unit value and only one outgoing edge of unit value (to $t$). In either case this is a contradiction to the conservation constraint of $f$. So $M$ has to be a matching. The size of $M$ is by its definition equal to the number of saturated edges from $X$ to $Y$. But this number has to be equal to the number of edges of unit value going out of $s$ (conservation constraint). Hence $|M| = |f|$. □

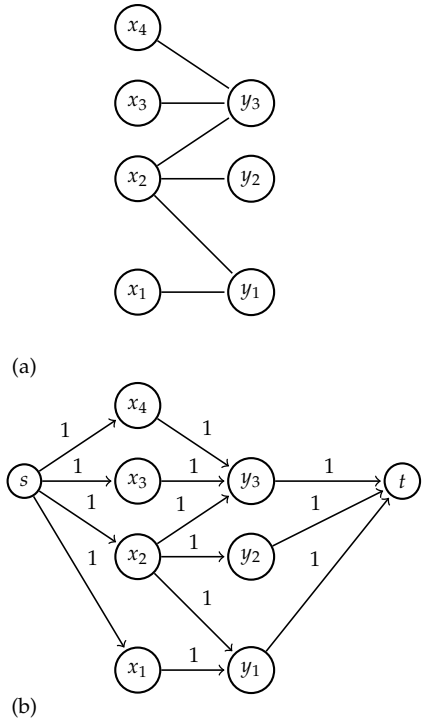



Figure 1.9:
(a) Bipartite graph
(b) Network constructed from it.

Theorem 1.8 let's us use the Ford-Fulkerson algorithm to compute a maximum matching in our bipartite graph $G$. Once we have a max-

imum matching we get the size of a minimum vertex cover with the following theorem, known as **König's theorem**[9]:

**Theorem 1.9.** *In a bipartite graph $G$ the size of a minimum vertex cover $C$ equals the size of a maximum matching $M$.*

*Proof.* $C$ is a vertex cover, so it covers all edges, which means it certainly covers a subset $M$ of all edges. But $M$ is a matching, so no two edges share a vertex. It follows that $|C| \geq |M|$.

From the maximum matching $M$ we get the associated maximum flow $f$ as described in Theorem 1.8. The residual graph $G'_f$ of the associated network cannot have any augmenting paths.

We consider the minimum cut $(S, T)$ associated with the maximum flow $f$. We define the following sets:

- $X_S = X \cap S, X_T = X \cap T$
- $Y_S = Y \cap S, Y_T = Y \cap T$
- $H = \{(u, v)$ edge in $G : u \in S, v \in T\}$
- $B = \{v \in Y_T : \exists u \in X_S$ with $(u, v)$ edge in $G\}$
- $D = X_T \cup Y_S \cup B$

$D$ is a vertex cover: $X_T \subseteq D$ and $Y_S \subseteq D$, so $D$ covers all edges that have endpoints in $X_T$ or $Y_S$. The set $B$ provides cover for $H$.

A vertex $u \in X_T$ is not reachable from $s$ in $G'_f$. It means that $f$ saturates $s \to u$ in $G'$, so the saturated edge $s \to u$ crosses the $(S, T)$ cut and counts towards $\|S, T\|$.

A vertex $v \in Y_S$ is reachable from $s$ in $G'_f$. It means that $f$ saturates $v \to t$ in $G'$ (otherwise some vertex from $T$ would be reachable from $v$ and also from $s$ in $G'_f$ which is a contradiction). The saturated edge $v \to t$ crosses the $(S, T)$ cut and counts towards $\|S, T\|$.

$f$ is a maximum flow so any edge from $X_S$ to $Y_T$ is saturated and counts towards $\|S, T\|$.

$$\|S, T\| = |X_T| + |Y_S| + |H|$$

Figure 1.10 shows this. In (b) there are three saturated (thick) arrows crossing the cut. The first two (counting from left to right) are due to $X_T$ and the last one due to $Y_S$. In this example $H$ is the empty set.

We have

$$|M| = |f| = \|S, T\| = |X_T| + |Y_S| + |H| \geq |X_T| + |Y_S| + |B| \geq |D|$$

$D$ is a vertex cover and $C$ is a minimum vertex cover, so $|D| \geq |C|$. It follows that $|C| \geq |M| \geq |D| \geq |C|$ which means $|C| = |M|$. □

[9] For a short and elegant proof see: Romeo Rizzi. A short proof of König's matching theorem. *Journal of Graph Theory*, 33(3):138–139, 2000. URL https://math.dartmouth.edu/archive/m38s12/public_html/sources/Rizzi2000.pdf

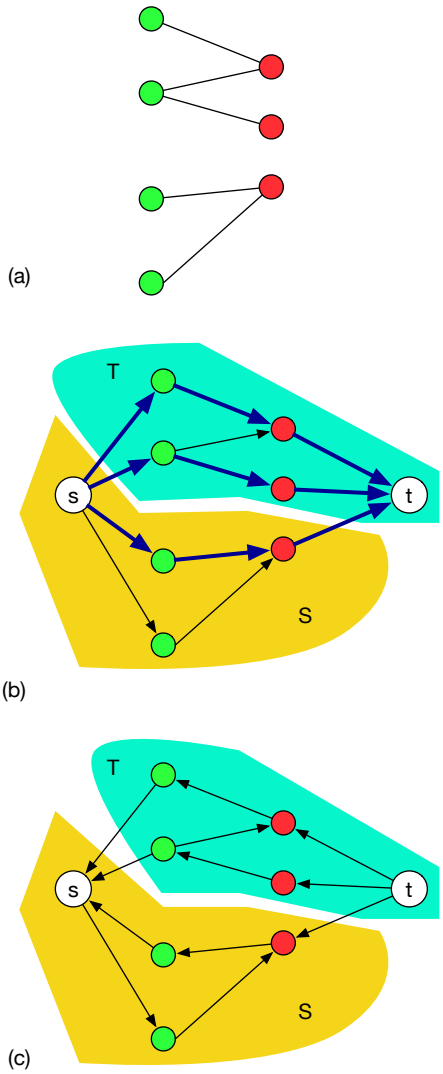



Figure 1.10:
(a) A bipartite graph $G(X \cup Y, E)$. $X$ are green vertices, $Y$ are red vertices.
(b) Maximum flow $f$ (thicker arrows) and minimum cut $(S, T)$ in the corresponding network $G'$. Thicker arrows between green and red vertices form the maximum matching.
(c) Same cut displayed with the corresponding residual graph $G'_f$.

This solves the problem in this note. The number of satisfied viewers is $|V| - |M|$, where $V$ is the set of vertices in the bipartite graph $G$ of votes with their conflicts as edges and $M$ is a maximum matching in $G$ computed with the Ford-Fulkerson algorithm taking advantage of the min-max duality shown in Figure 1.11.

Here is a similar problem: There are bridges running from one side of a river to the other side. The bridges are laid out in a random fashion. Given the coordinates of all the bridges, find the largest set of non-intersecting bridges.

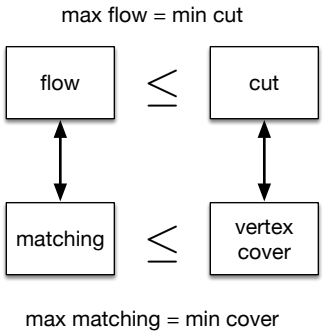



Figure 1.11: Min-max duality in bipartite graphs and corresponding networks.

# *Bibliography*


Jeff Erickson. *Algorithms, Etc*. 2015. URL http://jeffe.cs.illinois.edu/teaching/algorithms/.

Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358.

Romeo Rizzi. A short proof of König's matching theorem. *Journal of Graph Theory*, 33(3):138–139, 2000. URL https://math.dartmouth.edu/archive/m38s12/public_html/sources/Rizzi2000.pdf.

Alexander Schrijver. On the history of the transportation and maximum flow problems. 2002. URL http://homepages.cwi.nl/~lex/files/histtrpclean.pdf.

Spotify. Cat vs dog. 2012. URL https://labs.spotify.com/puzzles/.