2.3

Definition

A **weighted graph** G is a graph alongside a function $w : E(G) \to \mathbb{R}^+$.

If G is a weighted graph and $H \subseteq G$, then, $w(H) := \sum_{e \in E(H)} w(e)$.

A **minimum weight spanning tree** (or MWST) is a spanning tree T such that $w(T)$ is minimized among all possible spanning trees. In other words, $w(T) \leq w(T')$ $\forall T' \subseteq G$ where T' is a spanning tree.

Kruskal's Algorithm

INPUT Weighted graph G with n vertices

OUTPUT A MWST, T^* if G is connected, otherwise a message "G is not connected"

STEP 1 Create a list of edges, L_E, in order from smallest weight to largest weight. Start T^* with no edges but all vertices of G.

STEP 2 If the number of edges in T^* is strictly less than $n-1$ AND if there are still edges in L_E, examine the first edge in L_E (i.e., the edge with smallest weight), $e = \{a, b\}$

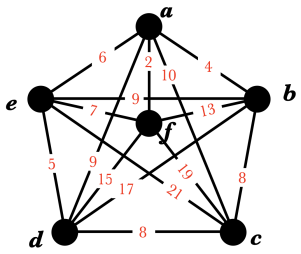
SUBSTEP 2.1 If a and b are in different components of T^*, add e to T^* and remove e from L_E. Return to STEP 2.

SUBSTEP 2.2 If a and b are in the same component, remove e from L_E and do not add to T^*. Return to STEP 2.

STEP 3 If the number of edges in T^* is $n-1$, then output T^*, which is the MWST for G. Otherwise, the number of edges in T^* is strictly less than $n-1$ and G was not connected.

Example

We will find a MWST for the following graph:



First, we create the following table of all the edges in G.

Edge	Cost
af	2
ab	4
de	5
ae	6
ef	7
bc	8
cd	8
ad	9
be	9
ac	10
bf	13
df	15
bd	17
cf	19
ce	21

We can read the following table describing the steps in Kruskal's algorithm from left to right (i.e., we check the edge of lowest weight, then we check the components, then we select our substep).

Edge	Components Of T′	Substep to be used
af	$\{a\},\{b\},\{c\},\{d\},\{e\},\{f\}$	2.1
ab	$\{a,f\},\{b\},\{c\},\{d\},\{e\}$	2.1
de	$\{a,b,f\},\{c\},\{d\},\{e\}$	2.1
ae	$\{a,b,f\},\{c\},\{d,e\}$	2.1
ef	$\{a,b,d,e,f\},\{c\}$	2.2
bc	$\{a,b,d,e,f\},\{c\}$	2.1
—	$\{a,b,c,d,e,f\}$	—

Proof of Kruskal's Algorithm

If G is connected, then Kruskal's Algorithm produces a minimum weight spanning tree.

Let G be a connected graph, and let T_K be the output from Kruskal's algorithm on G. It is easy to check that T_K is a spanning tree, since it is acyclic and has $n(G) - 1$ edges.

Suppose T^* is a MWST with largest edge intersection with T_K — in other words, $|E(T_K) \cap E(T^*)| \geq |E(T_K) \cap E(T')|$ for any other MWST T'.

If $T^* = T_K$, then we are done, since T_K is assumed to be a minimum weight spanning tree. Otherwise, assume toward contradiction that $T^* \neq T_K$. Let e be the first edge chosen by Kruskal's algorithm that is not in T^*. Then, by a previous result, $\exists e' \in E(T^*) - E(T_K)$ such that $T' = T^* + e - e'$ is a spanning tree.

We are assuming, however, that Kruskal's algorithm would choose e over e'. Let $e_1, \ldots, e_k$ be edges in $E(T_K)$ before e. Since e_i was selected before e, we know that $e_i \in E(T^*)$ for each $i \in [k]$.

Let $G_k = (V, \{e_1, \ldots, e_k\})$. we are assuming that Kruskal's algorithm would not choose e' for two reasons:

- If e' shows up before e, then e' would connect two vertices in $G_j = (V, \{e_1, \ldots, e_j\})$. Since $G_j + e' \subseteq T^*$, then T^* contains a cycle and isn't a spanning tree.
- If e' shows up after e in L_E, then $w(e') \geq w(e)$, meaning $w(T') \leq w(T^*)$, meaning that $w(T_K) \leq w(T^*)$, implying that T_K is of a lower weight than T^*.

Dijkstra's Algorithm

We want to find an algorithm to find the shortest path between two vertices in a weighted graph — Dijkstra's Algorithm can solve this.

We know that if P is the shortest u, v path, and $x \in P$, then following P from u to x will yield the shortest u, x path.

Input A weighted graph, G, and $u \in V(G)$

Output For each $z \in V(G)$, the distance $d(u, z)$.

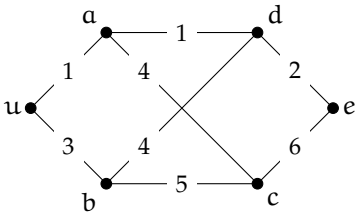
Initialization Extend the weight function such that if $xy \notin E(G)$, then $w(xy) = \infty$. Create S that contains all vertices whose distances from u are known. Let $S := \{u\}$. Let $t : V \to \mathbb{R}^+ \cup \{0, \infty\}$ which will keep track of the tentative distance between u and z. Let $t(z) := w(uz)$ for all $z \neq u$, and $t(u) := 0$.

Condition to Terminate Loop If $t(z) = \infty$ for all $z \notin S$ **or** $S = V$, then go to end.

Loop Else, pick $v \in V - S$ such that $t(v) = \min_{z \notin S} t(z)$. Add v to S. Explore the edges from v to update tentative distances; for each edge vz with $z \notin S$, $t(z) := \min\{t(z), t(v) + w(vz)\}$.

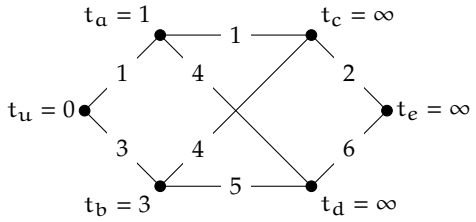
End Set $d(u, v) = t(v)\ \forall v \in V$.

On the following weighted graph, we can do Dijkstra's algorithm as follows:

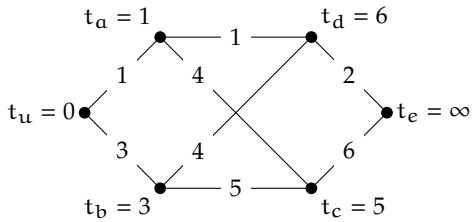


Dijkstra's Algorithm: Worked Example

We let $S = \{u\}$, and include our tentative distances for the first step.

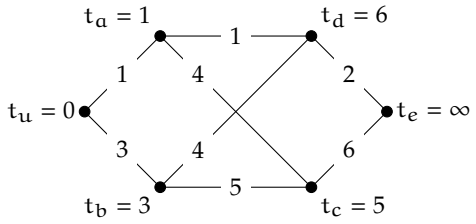


Now, $S = \{u, a\}$ because $t_a \leq t_b$. We now start from a and include our tentative distances.

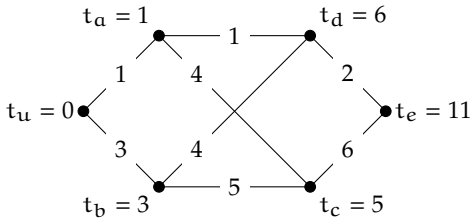


Next, we include b into S, making it $\{u, a, b\}$. We then check our tentative distances from b, where

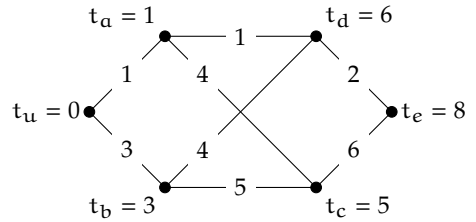
we find that they are no better than tentative distances from a, so we keep the tentative distances on c and d as what they were with a.



Next, we include c into S, making it $\{u, a, b, c\}$, and we update our tentative distances.



Finally, we include d and update tentative distances:



In order to find the direct paths, we can add arrows along the edges that were selected by Dijkstra's algorithm.

The proof can be outlined as follows:

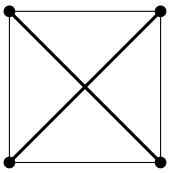
If $z \in S$: then, $t(z) = d(u, z)$.

Else if $z \notin S$: then, $t(z)$ is the length of a shortest u, z path P such that $V(P - z) \subseteq S$.

3.1

Matchings

In a simple graph, a **matching** M is a set of pairwise disjoint edges. In other words, $\forall e_i, e_j \in M$, then $e_i \cap e_j = \emptyset$. In an arbitrary graph, a matching is a set of non-loop edges with no shared endpoints. For example, the thick edges are a matching.



If v is incident to an edge, then v is **saturated** by M, otherwise it is unsatured by M.

A **perfect matching** is a matching that saturates every vertex. We know that all graphs with perfect matchings have even number of vertices, but the alternative case may not be true (for example, we might consider a graph with an isolated vertex).

A **maximal matching** is a matching that cannot be enlarged by adding any other edges. A **maximum** matching is a matching of maximum size among all matchings. In other words, if M^* is a maximum matching, then $|M^*| \geq |M| \; \forall M \in G$.

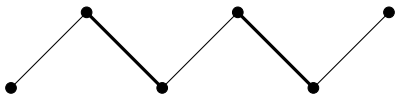
- Not every maximal matching is a maximum matching.
- However, every maximum matching is a maximal matching (because you cannot extend a maximum matching by definition).

Alternating and Augmenting Paths

Given a matching M, an M-**alternating path** is a path that alternates between edges in M and edges not in M. An M-alternating path whose endpoints are unsaturated by M is an M-**augmenting** path.

An M-alternating path:

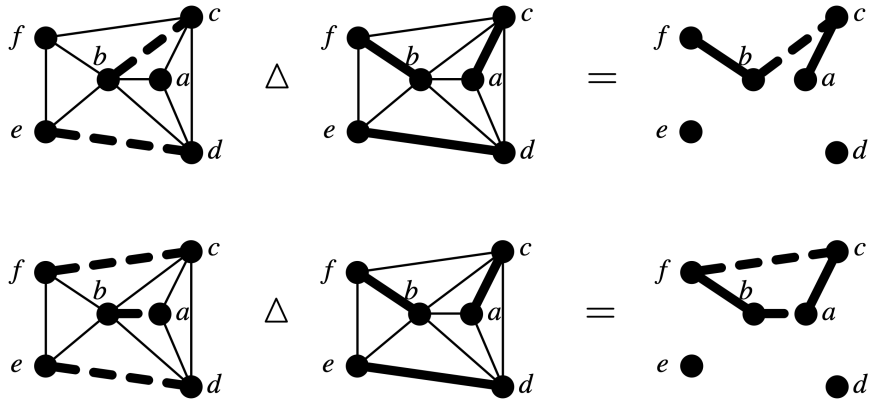
An M-augmenting path:



If M is a matching and there exists an M-augmenting path in G, then M is not a maximum matching (as in the path P that contains M, you can switch the matching edges as follows)

Symmetric Difference

The **symmetric difference** $G \triangle H$ is the subgraph of $G \cup H$ whose edges are the edges of $G \cup H$ that appear in exactly one of G and H. A picture of an example of a symmetric difference between matchings is shown below.



Theorems and Lemmas

Lemma 3.1.9

Every component of the symmetric difference of two matchings is a path or an even cycle.

Let M and M′ be two matchings. Each vertex is incident to at most 1 edge in M and at most one edge in M′. Thus, $d(v) \leq 2$ for each vertex in M and M′. Therefore, each component is either a path or a cycle.

If a component is a cycle, then it must be even because the edges of the cycle must alternate between M and M′.

Theorem 3.1.10

A matching M in a graph G is a *maximum* matching if and only if G has no M-augmenting path.

($\Rightarrow$) Suppose G has an M-augmenting path, P. Exchange the edges of P in M with the edges of P not in M. This action increases the size of the matching by 1, meaning that M was not a maximum matching initially.

($\Leftarrow$) Suppose M is not a maximum matching. Let M′ be a matching with more edges than M (i.e., $|M'| > |M|$). Let $H = (V(G), M \triangle M')$. By Lemma 3.1.9, we know that each component of H is either a path or an even cycle. Since $|M'| > |M|$, there is a component of H that contains more edges from M′ than edges from M, which means it cannot be an even cycle — therefore, this component must be a path that alternates between M′ and M. Because there are more edges from M′ than M in this path, meaning this path is an M-augmenting path.

Perfect Matchings

Let $G = (X, E, Y)$ be a bipartite graph. A matching that saturates X is an X-**perfect matching**.

If there exists a set of vertices $S \subseteq X$ such that $|S| > |N(S)|$, then it is impossible for G to have an

X-perfect matching.

Theorem 3.1.11 (Hall, 1935)

A bipartite graph $G = (X, E, Y)$ has an X-perfect matching if and only if $|S| \leq |N(S)|$ for all $S \subseteq X$.

($\Rightarrow$) If G has an X-perfect matching, then each vertex in X is matched to a distinct vertex in $N(S)$. Thus, $|S| \leq |N(S)|$.

($\Leftarrow$) We will prove via the contrapositive. Suppose G does not have an X-perfect matching. Let M be a maximum matching in G, and let $u \in X$ be an M-unsaturated vertex. Let $S = \{x \in X : \exists\, M\text{-alternating } u, x \text{ path}\}$, and $T = \{y \in Y : \exists M\text{-alternating } u, y \text{ path}\}$.

We can partition S into $\{u\}, X_1, \ldots, X_k$ and T into $Y_1, \ldots, Y_k$, where $|X_i| = |Y_i|$ for $1 \leq i \leq k$. Our partition works as follows: $Y_1 = N(u), X_1 = N_M(Y_1), Y_2 = N(X_1) - Y_1, X_2 = N_M(Y_2)$, and the general form is $Y_i = N(X_{i-1}) - (Y_1 \cup \cdots \cup Y_{i-1})$ and $X_i = N_M(Y_i)$.

Since M is a maximum matching, we know that there are no M-augmenting paths, meaning $|X_i| = |Y_i|$ (otherwise, if $|X_i| < |Y_i|$, we would be able to find a M-augmenting u, y path). Therefore, $|S| = |\{u\}| + |X_1| + \cdots + |X_k|$ while $|T| = |Y_1| + \cdots + |Y_k|$, so $|S| = |T| + 1$.

Corollary 3.1.13

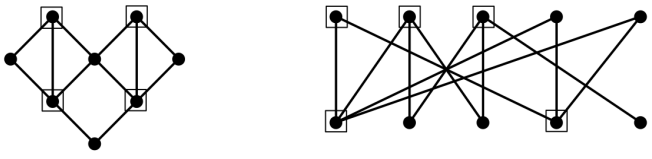
For $k > 0$, every k-regular bipartite graph has a perfect matching.

Let $k > 0$ and let $G = (X, E, Y)$ be a k-regular bipartite graph. We will show that G satisfies Hall's condition. Let $S \subseteq X$ and $[S, N(S)] = \{xy \in E(G) : x \in S \text{ and } y \in N(S)\}$, and $[X, N(S)] = \{xy \in E(G) : x \in X \text{ and } y \in N(S)\}$. Since $[S, N(S)] \subseteq [X, N(S)]$. Additionally, $\sum_{x \in S} d(X) = |[S, N(S)]|$ and $\sum_{y \in X} d(y) = |[X, N(S)]|$.

Since G is k=regular, we find that $|[S, N(S)]| = k|S|$ and $|[X, N(S)]| = k|N(S)|$, meaning that $|S| \leq |N(S)|$

Vertex Covers

A **vertex cover** is a set $Q \subseteq V(G)$ such that Q contains at least one endpoint of every edge. A picture of two vertex covers is shown below:



Of course, a very simple way to find a vertex cover is to select every vertex into Q. However, that isn't particularly useful, so we are focused on *small* vertex covers.

- The size of the **minimum vertex cover** is $\beta(G)$.

- The size of the **maximum matching** is $\alpha'(G)$ (the edge correspondent to $\alpha(G)$ for the size of the maximum independent set).

Alternatively, the minimum vertex cover is denoted $\tau(G)$ and the maximum matching is denoted $\nu(G)$.

An example of minimum vertex covers is shown below:

For any graph G, we can see that $\alpha'(G) \leq \beta(G)$ because to cover the edge of the maximum matching, we will need at least one vertex per edge in the maximum matching. Therefore, the minimum vertex cover must contain within it every vertex corresponding to an edge in the maximum matching, so $\beta(G) \geq \alpha'(G)$.

Theorem 3.1.16 (König-Egervary Theorem)

If G is a bipartite graph, then $\beta(G) = \alpha'(G)$.

Let $G = (X, E, Y)$ be a bipartite graph, and let M be a maximum matching in G. We are going to show that there exists a Q such that $|Q| = |M|$.

For each e in M, with endpoints $x \in X$ and $y \in Y$, select either x or y to be in Q as follows:

Rule: If there exists an M-alternating path that starts at an unsaturated vertex $u \in X$ and ends at y, then $y \in Q$ — else, $x \in Q$.

Claim: Q is a vertex cover. We will use cases to show that every edge has at least one endpoint in Q.

Case 1: Suppose $xy \in M$. Then, $x \in Q \veebar y \in Q$.

Case 2: Suppose $xy \notin M$. If x is M-unsaturated, then y must be M-saturated (or else we would add the edge to the matching). We can create an M-alternating path that starts at x and ends at y, meaning that $y \in Q$.

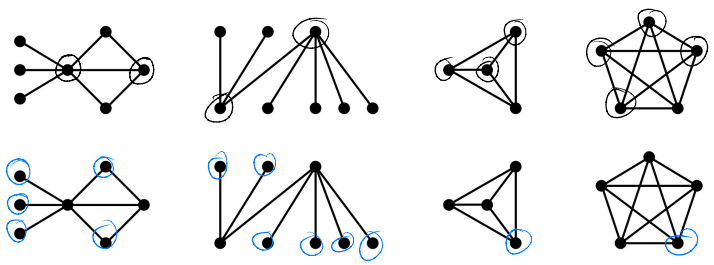
Otherwise, if x is M-saturated, let $xv \in M$ — then, we can find that $v \in Q$ because we create an M-alternating path that starts at $u \in X$ and ends at v. If P does not contain y, then we extend by going from v to x to y.

Edge Covers

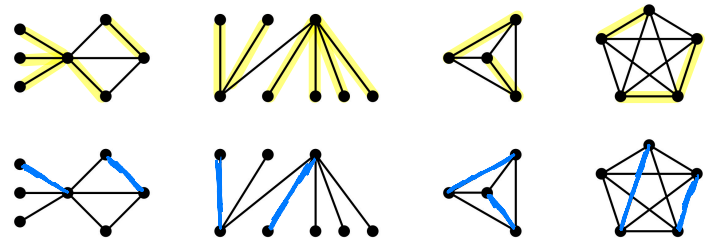
An **edge cover** is a set of edges A such that every vertex is incident to some edge in A. If A is an edge cover, then $\bigcup_{e \in A} e = V(G)$.

The **edge cover number** is the size of a minimum edge cover of G. It is denoted $\beta'(G)$.

We can see a relationship between vertex covers and independent sets as follows, where the top graph shows the minimum vertex cover and the bottom graph shows the maximum independent set.



Similarly, we can see a relationship between minimum edge covers and maximum matchings.



Theorems and Lemmas

Following from the previous result regarding independent sets and vertex covers, we might be able to state the following propostions:

Lemma 3.1.21

In a graph G, $S \subseteq V(G)$ is an independent set if and only if $\overline{S}$ is a vertex cover. Therefore, we get the following result:

$$\alpha(G) + \beta(G) = n(G)$$

S is an independent set.

$\Leftrightarrow \forall u, v \in S, u \nleftrightarrow v$.

$\Leftrightarrow \forall e \in e(G)$, both endpoints of e are not in S.

$\Leftrightarrow$ Both endpoints of e are in $\overline{S}$.

$\Leftrightarrow \overline{S}$ is a vertex cover.

Thus, if S is a maximum independent set, then $\overline{S}$ is a minimum vertex cover, and since $|S| + |\overline{S}| = n(G)$, this means $\alpha(G) + \beta(G) = S$.

Theorem 3.1.22 (Gallai's Theorem)

If G is a graph without isolated vertices, then $\alpha'(G) + \beta'(G) = n(G)$. (i.e., if G is a graph without isolated vertices, then the size of a minimum edge cover summed with the size of the maximum matching equals the number of vertices in G).

We will prove the forward direction first, as follows:

Let M be a maximum matching, and let $A = V(M)$ and $B = \overline{A}$. Since M is a maximum matching, $\nexists e \in E(G)$ such that e connects two edges in B (or else you could simply add it to the matching). Additionally, since there are no isolated vertices in G, $\forall v \in B, \exists w \in A$ such that $v \leftrightarrow w$.

Select one edge such that $e = vw$ for each $v \in B$, and let the set of these edges be N. Then, $C = M \cup N$ is an edge cover, as every vertex in A is covered by M, and every vertex in B is covered by N. So, we have the following:

$$\begin{aligned} |M| + |C| &= |M| + (|M| + |N|) && \text{Since } M \text{ and } N \text{ are disjoint} \\ &= \left(\frac{|A|}{2}\right) + \left(\frac{|A|}{2} + |B|\right) && \text{By the definition of } M \text{ and } N \\ &= n(G) \end{aligned}$$

So, we have that $|M| + |C| = n(G) \geq \alpha'(G) + \beta'(G)$.

Now, we will prove the reverse direction:

Let C be a minimum edge cover. The components of C are stars (there are no cycles or copies of P_4). Let M be a matching formed by taking one edge from each edge in each component of C. Assign each "center" vertex in with an edge from M and assign each "leaf" edge with an edge from C.

Each vertex in $V(G)$ appears exactly once in each star, we have that $n(G) = |M| + |C|$. So, since $|C| = \beta'(G)$ and $|M| \leq \alpha'(G)$, we have that $|M| + |C| = n(G) \leq \alpha'(G) + \beta'(G)$.

Since $n(G) \geq \alpha'(G) + \beta'(G)$ and $n(G) \leq \alpha'(G) + \beta'(G)$, we have that $n(G) = \alpha'(G) + \beta'(G)$.

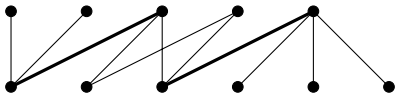
Corollary 3.1.24

If G is a bipartite graph with no isolated vertices, then $\alpha(G) = \beta'(G)$.

We previously showed that $\alpha(G) + \beta(G) = n(G)$ and $\alpha'(G) + \beta'(G) = n(G)$, and by the König-Egerváry theorem, we know that $\alpha'(G) = \beta(G)$ in bipartite graphs with non-isolated vertices, so $\alpha(G) + \beta(G) - \beta(G) = \alpha'(G) - \alpha'(G) + \beta'(G)$, so $\alpha(G) = \beta'(G)$.

Our Hungarian Method for Maximum Matchings

This is a method for finding a maximum matching in a bipartite graph that builds off the augmenting path algorithm. Consider the graph below:



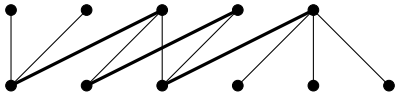
We will find the maximum matching by creating a "Hungarian Forest" as follows:

Initialization In our graph, we set $X_0 = \{x_1, x_2, x_4\}$ and $Y_0 = \{y_2, y_4, y_5, y_6\}$, where $X_0 \cup Y_0$ are the set of M-unsaturated vertices in G.

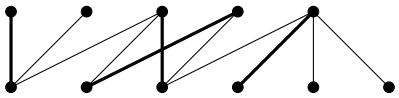
Iteration Set $X_1 = X_0$ and $Y_1 = N(X_1)$. Then, we do $X_2 = N_M(X_1)$, and $Y_2 = N(X_2) - Y_1$, and so on. The general case is $X_n = N_M(Y_{n-1})$, and $Y_n = N(X_n) - Y_{n-1}$.

End condition If Y_n is empty, then we stop the algorithm.

Let $Y^* = \bigcup Y_i$ (the vertices of Y that are in our Hungarian Forest). If $Y_0 \cap Y^* = \emptyset$, then M is a maximum matching. Otherwise, select $y \in Y_0 \cap Y^*$, and follow the path from y to the root of the tree to create an M-augmenting path, and "flip" the path (exchange matching edges with non-matching edges). In this case, after flipping the path starting between x_4 and y_2, yielding the following graph.



After building our Hungarian forest in the same process, we find that $Y_0 \cap Y^* \neq \emptyset$, so we pick an M-augmenting path starting at y_4, and flip the path starting at y_4 following up to x_1. Afterwards, we find the graph as follows:



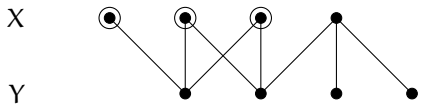
3.3

Preliminary

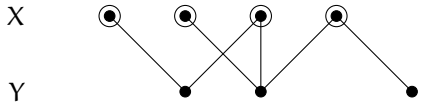
Q: Suppose $G = (X, E, Y)$ is a bipartite graph that *does not* have a X-perfect matching. What does Hall's Theorem imply about G?

A: There must exist a subset $A \subseteq X$ such that $|A| > |N(A)|$.

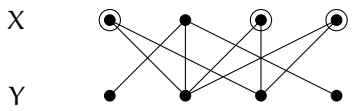
If you have a bipartite graph, you can *either* find an X-perfect matching, or you can find $A \subseteq X$ such that $|A| > |N(A)|$. We can test this proposition on the following set of bipartite graphs:



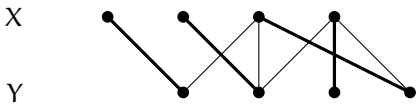
In the above graph, we can see that there is a set of vertices X such that the set of neighbors is of smaller cardinality.



In the above graph, we can see that the entirety of X is of a larger cardinality than the entirety of N(X), so there cannot be an X-perfect matching.



In the above graph, we can see that there is a subset of X such that N(S) is of smaller cardinality than S.

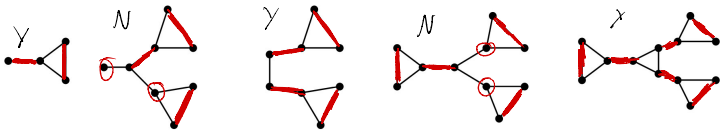


In the above graph, we have found an X-perfect matching.

We can infer from the previous exercise that if $G = (X, E, Y)$ and $|X| = |Y|$, and G does *not* have a perfect matching, then we can find both $A \subseteq X$ and $B \subseteq Y$ such that $|A| > |N(A)|$ and $|B| > |N(B)|$.

Conditions for a Perfect Matching

We can find a set of perfect matchings in the following graphs as follows:



From this, we can find some conditions for a perfect matching as follows:

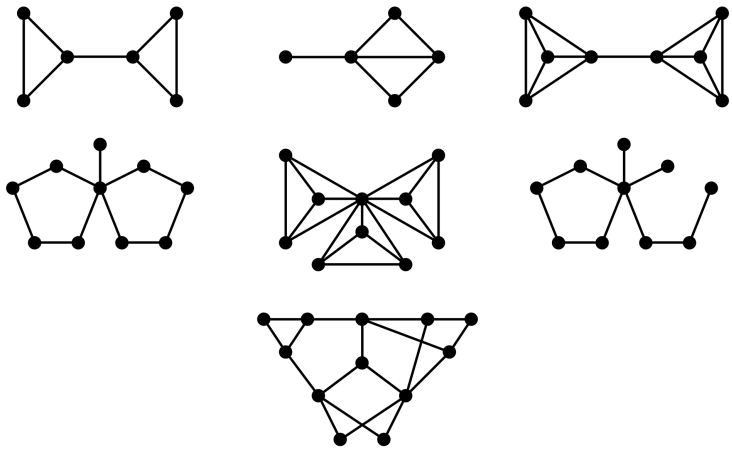
- $n(G)$ is even
- For each component H in G, $n(H)$ is even
- $\nexists v \in G$ such that $G - v$ has 2 or more odd components

We will let $q(G)$ denote the number of *odd* components in G. Our necessary condition is as follows:

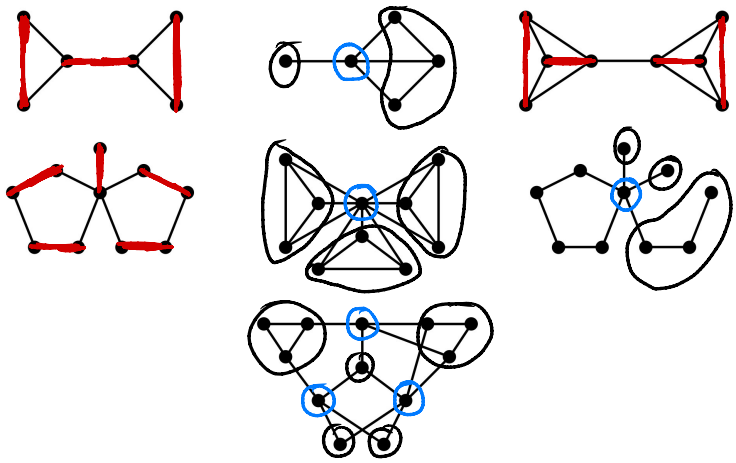
$$\forall S \subseteq V, q(G - S) \leq |S|$$

Exercise

For each of the following graphs, find a perfect matching or find S such that $q(G - S) > |S|$.



The solutions are as follows — matchings are denoted in red, deleted vertices in blue, and components after deletion in black.



Tutte's Theorem

A **factor** of a graph is a spanning subgraph of G, and a k-factor is a spanning, k-regular subgraph of G. Note that a 1-factor is essentially a perfect matching, but a matching is a set of edges while a factor is a graph. If M is a matching in G, then $H = (V, M)$ is a 1-factor.

A graph G has a 1-factor if and only if $q(G - S) \leq |S|$ for all $S \subseteq V$. A graph satisfies Tutte's Condition if $\forall S \subseteq V, q(G - S) \leq |S|$.

($\Rightarrow$) To prove the forward direction, we know that the odd components of $G - S$ must have vertices matched to distinct vertices in S — this is because in any maximum matching within an odd component of $G - S$, there must be one vertex remaining, and since G has a 1-factor, each of

these spare vertices must be matched to a vertex in S.

($\Leftarrow$) To prove the reverse direction, we will do so by contradiction. When we add an edge between two components of $G - S$, the number of *odd* components does not increase. So, Tutte's condition is preserved by the addition of edges, meaning that if $G' = G + e$, then $q(G' - S) \leq q(G - S) \leq |S|$, because we are assuming $q(G - S) \leq |S|$, and $q(G' - S) \leq q(G - S)$.

Additionally, if $G' = G + e$ has no 1-factor, then G has no 1-factor (since $q(G - S) \geq q(G' - S) > |S|$ in that scenario).

So, if there exists a simple graph G does not have a 1-factor and satisfies Tutte's condition, the theorem does not hold, nor does it hold if G would have a 1 factor upon adding an edge (or else G would have a 1-factor). Assume towards contradiction that G satisfies the Tutte condition or G' has a 1-factor while G does not. We will show that G must contain a 1-factor.

Let U be the set of vertices of degree $|V| - 1$.

Case 1 Suppose $G - U$ consists of disjoint complete graphs. In this case, the vertices in each component of $G - U$ can be paired away, with one extra in the odd components. Since $q(G - U) \leq |U|$ by assumption, and each vertex of U is adjacent to every element of $G - U$, we can match the leftover vertices to U.

The remaining vertices are in U, which is a clique (since, by our assumption, every vertex in U is adjacent to every other vertex in G), meaning we only need to show that there are an even number of vertices in U. Seeing as we have matched an even number of vertices in G so far, it suffices to show that $|V|$ is even, which we can see if we apply the Tutte condition for $S = \emptyset$, which would mean that $q(G) \leq 0$, meaning $q(G) = 0$. Therefore, for the case where $G - U$ is a set of disjoint cliques, we have proved sufficiency.

Case 2 Suppose $G - U$ is not a disjoint union of cliques. Then, there must be at least one component in $G - U$ that is not a clique, meaning that there is one pair of vertices $x, z \in V(G - U)$ such that $x \nleftrightarrow z$, while $\exists y \in V(G - U)$ such that $x \leftrightarrow y$ and $y \leftrightarrow z$. Furthermore, $\exists w \in V(G - U)$ such that w and y are not adjacent to each other — if $y \in U$, then y would be adjacent to every vertex in $G - U$, but $y \notin U$.

By our choice of G, adding an edge to G creates a 1-factor. Let M_1 and M_2 be 1-factors defined in $G + xz$ and $G + yw$ respectively. All we need to show is that $M_1 \triangle M_2 \cup \{xy, yz\}$ contains a 1-factor avoiding xz and yw, implying that G contains a 1-factor. Let $F = M_1 \triangle M_2$. Then, since $xz \in M_1 - M_2$ and $yw \in M_2 - M_1$, we know that xz and yw are in F. Additionally, every edge in M_1 and M_2 is of degree 1, so if $ab \in F$, then $ac \in F$, implying that every vertex is of degree 0 or 2.

The components of F are either even cycles or isolated vertices. Let C be the cycle of F containing xz. If C does not contain yw, then we can create a 1-factor by taking all the edges of M_2 in C (as M_2 does not contain xz) and all the edges of M_1 not in C (as M_1 does not contain yw). Because this set contains every edge either in M_1 or M_2, it is a 1-factor.

If C contains both yw and xz, then we use yx and xz to avoid them while constructing our 1-factor. We use M_1 to avoid yw, and then when we reach x or z, we use xy or zy respectively. In the remainder, we use the edges of C in M_2. Seeing as we have avoided both xz and yw in this arrangement, and as seen previously we created a 1-factor by using the edges of M_2 in C, we know that this contains neither xz or yw. Finally, when we use either M_1 or M_2 outside of C, we have finished our 1-factor of G.

Corollary 3.3.7

The Berge-Tutte formula for the largest number of vertices saturated by a matching in G is as follows.

$$\min_{S \subseteq V(G)} \{n(G) - d(S)\}$$

where $d(S) = q(G - S) - |S|$.

If G has a perfect matching, then every unmatched vertex in a component of $G - S$ is matched to a vertex in S, meaning that $d(S) = 0$. If G does not have a perfect matching, then the meaning of minimization of $q(G - S) - |S|$ finds the excess number of vertices in S that cannot be matched to the odd components in $G - S$.

Let $d = \max\{q(G - S) - |S|\}$. If $S = \emptyset$, then $d \geq 0$, so d is always greater than or equal to 0. Let $H \cong K_d$ where $V(H) \cap V(G) = \emptyset$. Let $G' = G \vee H$, where $G \vee H = \{x \in G, y \in H : x \leftrightarrow y\}$.

We will show that $n(G')$ is even. Let S^* be a subset of G such that $q(G - S^*) - |S^*| = d$. Observe that $n(G) \equiv n(G - S^*) + |S^*|$ modulo 2, by definition. Since even components contribute 0 to the parity, and even components contribute 1 to the parity, and $a \equiv -a$ modulo 2, we have that $n(G) \equiv q(G - S^*) - |S^*|$ modulo 2. Finally, we add both sides to yield $n(G) + d \equiv 0$ modulo 2, so $n(G') \equiv 0$ modulo 2.

We will show that G′ satisfies Tutte's condition, $\forall S' \subseteq V(G'), q(G' - S') \leq |S'|$.

Case 0 Let $S' = \emptyset$. Since $n(G')$ is even, and G′ is connected through H, $q(G') = 0 \leq |S'|$.

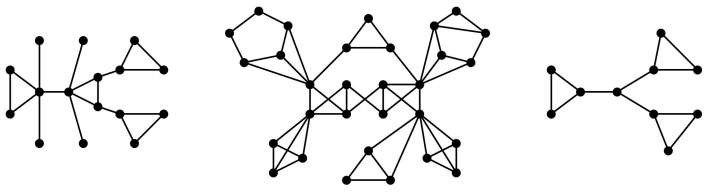
Case 1 Let S′ satisfy $S' \neq \emptyset$ and $V(H) \subseteq S'$. Let $S = S' \cap V(G) = S' - V(H)$. Notice that $q(G' - S') = q(G - S)$, because $G' - S' = G - S$. By the definition of d, the number of odd components in $q(G - S) - |S| \leq d$. So, $q(G' - S') - |S| \leq d$, so $q(G' - S') \leq |S| + d$, meaning $q(G' - S') \leq |S'|$.

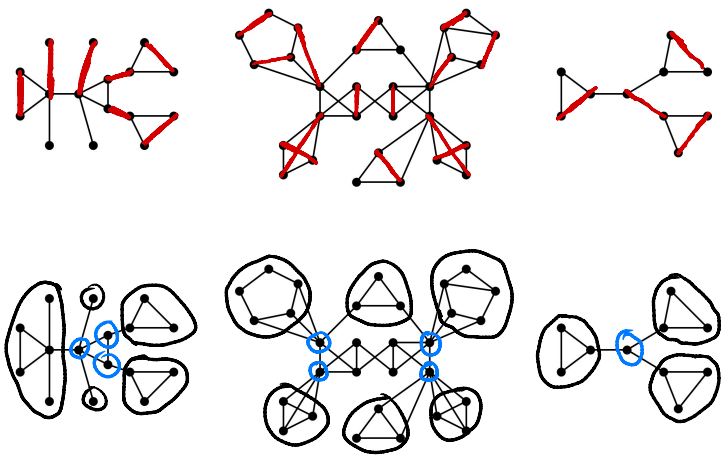
Case 2 Let S′ be such that $S' \neq \emptyset$ and $V(H) \not\subseteq S'$. Then, $V(H) - S' \neq \emptyset$. Let $v \in V(H) - S'$. Then, v is adjacent to every vertex in $G' - S'$. Thus, there is exactly one component in $G' - S'$, so $q(G' - S') \leq 1$, and since S′ was nonempty, we have that $q(G' - S') \leq |S'|$.

Thus, G′ has a 1-factor with $n(G') = n(G) + d$ vertices, so G has a matching of at least $n(G) - d$ vertices of G. Let $S^* \subseteq V(G)$ satisfy $q(G - S^*) - |S^*| = d$. There are at least d odd components of $G = S^*$ that have one vertex unsaturated by any matching. Thus, every matching saturates at *most* $n(G) - d$ vertices of G.

Maximum Matchings and Berge-Tutte Formula

For each of the following graphs, find a maximum matching and show that it is a maximum matching using the Berge-Tutte examples





Corollary 3.3.8

Every 3-regular graph with no cut-edge has a 1-factor.

For $A, B \subseteq V$, let $[A, B]$ denote the set of edges with one endpoint in A and one endpoint in B. We will show that $\forall S \subseteq V, 3q(G - S) \leq [S, \overline{S}] \leq 3|S|$. This implies that $3q(G - S) \leq 3|S|$, meaning $q(G - S) \leq |S|$ for all $S \subseteq V$

Let $S \subseteq V$ be an arbitrary set. Notice that $|[S, \overline{S}]| \leq \sum_{x \in S} d(x) = 3|S|$, because G is 3-regular. Let H_i be an arbitrary odd component in $G - S$. Notice that $3n(H_i) = \sum_{x \in V(H_i)} d(x) = [S, V(H_i)] + 2e(H_i)$. Therefore, we know that $3n(H_i) - 2e(H_i) = [S, V(H_i)]$, so $[S, V(H_i)]$ is of odd cardinality. However, we know that $|[S, V(H_i)]| > 1$ because G has no cut-edges, so $|[S, V(H_i)]| \geq 3$. Thus, summing over all odd components, we have that $\sum_{V(H_i)} |[S, V(H_i)]| \geq \sum_{H_i} 3 = 3q(G - S)$.

Therefore, $3q(G - S) \leq [S, \overline{S}] \leq 3|S|$, so $3q(G - S) \leq 3|S|$, so $q(G - S) \leq |S|$. Since G satisfies Tutte's condition, G must have a 1-factor.

4.1

Definitions

A **vertex cut** of a graph is a set $S \subseteq V$ such that $G - S$ has more than one component. The **connectivity** of a graph, $\kappa(G)$, is the *minimum* size of a vertex set S such that $G - S$ is disconnected or has only one vertex.

A graph is k-connected if its connectivity is at least k (i.e., $\kappa(G) \geq k$). A graph is k-connected if any two of its vertices can be joined by at least k independent paths.

For a complete graph K_n, we can see that $\kappa(K_n) = n - 1$, and $\kappa(K_{m,n}) = \min\{m, n\}$. It's important to note that if G is not complete, then G is k-connected if and only if every vertex cut in G has size *at least* k.

Suppose $G \ncong K_n$ and G is simple. We can show that $\kappa(G) \leq k - 2$, since $\exists u, v \in V(G)$ such that $u \nleftrightarrow v$. By deleting $V(G) - \{u, v\}$, we are left with u and v disconneted. Therefore, we know that $\kappa(G) \leq k - 2$.

Disconnections and Examples

We can disconnect G by deleting the neighborhoods of every vertex of minimum degree.

The connectivity is bounded from above by $\delta(G)$. Similarly, for every $k \in \mathbb{N}$, we can find G such that $\kappa(G) = \delta(G) = k$ by using K_{k+1}.

If G is a n-vertex graph and $\kappa(G) \geq k$, then $|E(G)| \geq \lceil nk/2 \rceil$.

Ex. Show that Q_k satisfies $\kappa(G) = k$ and $e(G) = \frac{n(G)k}{2}$.

Proof We know that Q_k is k-regular, so $e(Q_k) = \frac{n(Q_k)k}{2}$. Since $\delta(Q_k) = k$, we know that $\kappa(Q_k) \leq k$. We will show that $\kappa(Q_k)$ must be greater than or equal to k.

Base Case For $k = 1$, the 1-dimensional hypercube, the connectivity is 1 as it is isomorphic to K_2.

Inductive Hypothesis Assume $\kappa(Q_{k-1}) = k - 1$. Observe that Q_k can be partitioned into two copies of Q_{k-1}, Q' and Q'', connected to each other by a perfect matching. Let $V(Q') = \{(a_1, \ldots, a_{k-1}, 0)\}$ and $V(Q'') = \{(a_1, \ldots, a_{k-1}, 1)\}$. These vertices can be perfectly matched to each other, and each of Q' and Q''. Let $S \subseteq V(Q_k)$ satisfy $|S| \leq k - 1$. We will show that $Q_k - S$ is connected.

Case 1 Suppose $S \cap V(Q') \neq \emptyset$ and $S \cap V(Q'') \neq \emptyset$. In this case, $|S \cap V(Q')| < |S|$ and $|S \cap V(Q'')| < |S|$. Since Q' and Q'' are both copies of Q_{k-1}, and Q_{k-1} has connectivity $k - 1$ by the inductive hypothesis, $Q' - S$ is connected and $Q'' - S$ is connected, since $|S| \leq k - 1$, and there are 2^{k-1} disjoint edges joining $V(Q')$ and $V(Q'')$, there exists an edge $\{v', v''\}$ such that $v \in V(Q')$ and $v'' \in V(Q'')$, and $S \cap \{v', v''\} = \emptyset$, so $Q_k - S$ is connected, so $\kappa(Q_k) \geq k$.

Case 2 WLOG, suppose $S \cap V(Q'') = \emptyset$. In this case, any two vertices are connected via Q''. For example, for $v_1, v_2 \in Q'$, follow a path from v_1 to its neighbor $v_1'' \in Q''$, then follow a path to $v_2' \in Q''$, then to v_2. Thus, $Q_k - S$ is connected, and $\kappa(Q_k) \geq k$

Harary Graphs

A Harary Graph, $H_{k,n}$, is defined via the following conditions:

- $\kappa(H_{k,n}) = k$
- $n(H_{k,n}) = n$
- $e(H_{k,n}) = \left\lceil \frac{kn}{2} \right\rceil$

Disconnecting Set

A disconnecting set of edges is a set $F \subseteq E(G)$ such that $G - F$ has more than one component. A graph is k-edge-connected if every disconnecting set has at least k edges. The *edge connectivity* of a graph, $\kappa'(G)$, is the minimum size of a disconnecting set.

Given $S, T \subseteq V(G)$ such that $S \cap T = \emptyset$, we write $[S, T]$ to denote the set of edges with one endpoint in S and the other endpoint in T.

An *edge cut* is a set of the form $[S, \overline{S}]$ where $S \neq \emptyset$ and $\overline{S} \neq \emptyset$.

We can show that $\kappa'(G) \leq \delta(G)$ because we can delete the edges incident on the graph with minimum degree, and then create a disconnected graph.

Theorem 4.1.9: Whitney's Theorem

If G is a simple graph, then the following is true:

$$\kappa(G) \leq \kappa'(G) \leq \delta(G)$$

Since we showed that $\kappa'(G) \leq \delta(G)$, all we need to show is that $\kappa(G) \leq \kappa'(G)$. Let $[S, \overline{S}]$ denote the minimum edge cut.

Case 1, $\forall x \in S, \forall y \in \overline{S}, x \leftrightarrow y$: This is a complete bipartite graph. Let $|S| = \ell$, meaning $|\overline{S}| = n - \ell$. We know, then, that $\kappa'(G) = |[S, \overline{S}]| = |S||\overline{S}| = \ell(n - \ell) \geq n - 1 \geq \kappa(G)$.

Case 2, $\exists x \in S, y \in \overline{S} \ni x \nleftrightarrow y$: In this scenario, we construct our vertex cut T by deleting $N(x)$ and $N(y)$, but not x and y, as well as a vertex in every endpoint of an edge of our edge cut. Thus, we haven't deleted either x or y, so $[S, \overline{S}] \geq T$.

If $\kappa(G) = \delta(G)$, then $\kappa(G) = \kappa'(G) = \delta(G)$.

Examples

Graph such that $\kappa(G) = \kappa'(G) = \delta(G)$: K_n, Q_k, $K_{n,n}$, $H_{k,n}$.

Graph such that $\kappa(G) < \kappa'(G) < \delta(G)$:

Show $\kappa'(G)$ can be arbitrarily smaller than $\delta(G)$: Let G be $K_m + K_n$. Then, $\kappa'(G) = 0$ and $\delta(G) = \min\{m - 1, n - 1\}$.

Theorem 4.1.11

If G is 3-regular, then $\kappa(G) = \kappa'(G)$.

We know that $\kappa(G) \leq \kappa'(G)$. We will show that $\kappa(G) \geq \kappa'(G)$ to prove equality.

Let S be a minimum vertex cut, $|S| = \kappa(G)$. Let H_1 and H_2 be two components of $G - S$. Since S is minimum, $\forall v \in S$, v has a neighbor in H_1 and one in H_2. To construct our minimum edge cut, we do the following:

For each $x \in S$, if $|[x, V(H_1)]| = 1$, then delete one edge between x and $V(H_1)$. If $|[x, V(H_2)] = 2$, then delete an edge between x and $V(H_2)$.

Proposition 4.1.12

If S is a set of vertices in G, then the following are true:

$$\sum_{v \in S} d(v) = 2e(G[S]) + |[S, \overline{S}]|$$

and

$$|[S, \overline{S}]| = \sum_{v \in S} d(v) - 2e(G[S])$$

Corollary 4.1.13

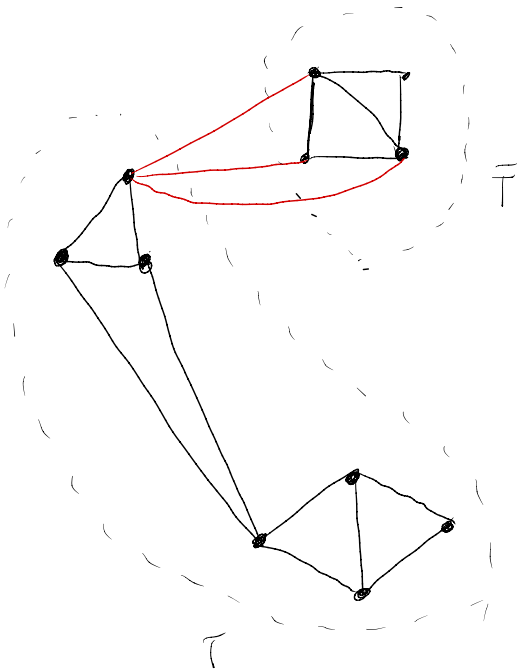
If G is simple and S is a nonempty proper subset of V such that $|[S, \overline{S}]| < \delta(G)$, then $|S| > \delta(G)$.

Our sketch is that $\delta(G) > |[S, \overline{S}]| = \sum_{x \in S} d(x) - 2e(G[S]) \geq \sum_{x \in S} \delta(G) - 2\binom{|S|}{2} = |S|\delta(G) - |S|(|S| - 1)$.

Thus, $\delta(G) > |S|\delta(G) - |S|(|S| - 1)$

Bonds and Blocks

A *bond* is a minimal nonempty edge cut. For example:



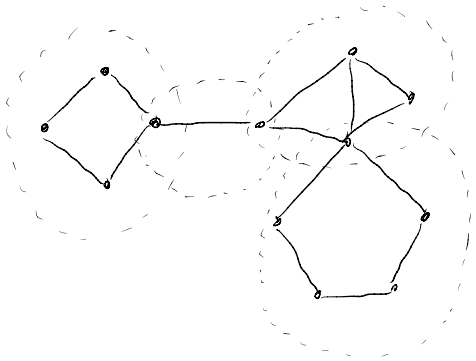
Proposition 4.1.15

If G is a connected graph, then an edge cut F is a bond if and only if $G - F$ has exactly two components.

($\Leftarrow$) Suppose $G - F$ has exactly two components, H_1 and H_2, where $F = [V(H_1), V(H_2)]$. If we add back any edge from F, then the resulting graph will be connected. If we take any $F' \subset F$, we can see that $G - F'$ is connected. Therefore, F is the minimal edge cut.

($\Rightarrow$) Let F be a bond. We will prove via contrapositive. If $G - F$ has more than two components, then for the edge cut $F = [S, \overline{S}]$, we find that $G - F$ will contain components H_1, H_2, and H_3.

A *block* is a maximal connected subgraph of G that has no cut-vertex within itself. In other words, if H is a block in G, then H *may* contain a cut-vertex in G, but will not contain a cut-vertex in H.



In G, the blocks of G are:

- isolated vertices

- cut-edges
- maximal 2-connected subgraphs

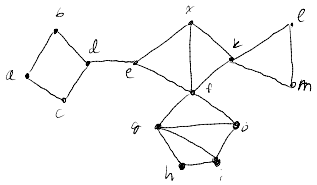
Proposition 4.1.19

Two blocks in a graph share at most one vertex.

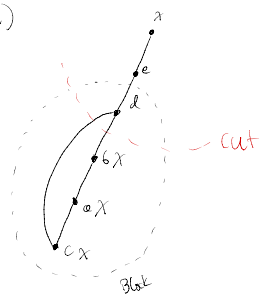
Suppose toward contradiction that there are two blocks B_1 and B_2 such that $|V(B_1) \cap V(B_2)| = 2$. We will show that $B_1 \cup B_2$ contains no cut-vertex, meaning neither B_1 nor B_2 is maximal. Delete an arbitrary vertex from $B_1 \cup B_2$. Then, $B_1 - v$ and $B_2 - v$ are still connected, as they still share at least one vertex in common.

The *block graph* of G is a bipartite graph H with bipartition $A \cup \mathcal{B}$, where A is the set of cut-vertices and $\mathcal{B}$ is the set of all blocks in G. The edges are defined as $a \in A$ is adjacent to $B_i \in \mathcal{B}$ if and only if $a \in B_i$.

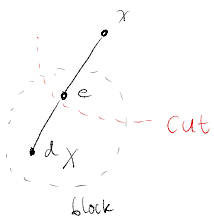
To find the blocks (and the resulting block graph), we start at a vertex and travel along the neighbors until no edges are above the "parent" vertex. Including the parent, this creates a block, and we delete the block save for the parent vertex.


a
b
c
d
e
f
g
h
i
k
l
m
x

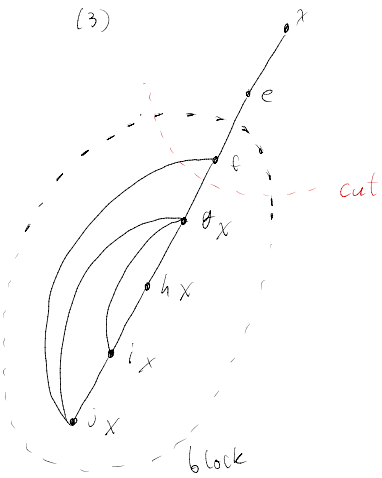
(1)


x
e
d
b
a
c
cut
block

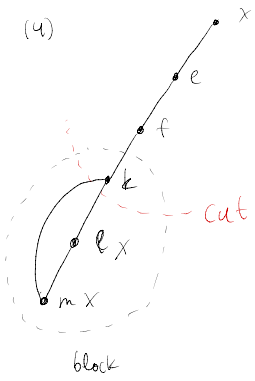
(2)


x
e
d
cut
block

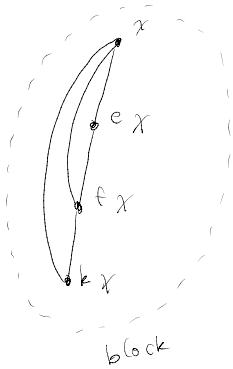
(3)


x
e
f
g
h
i
j
cut
block

(4)


x
e
f
k
l
m
cut
block

(5)


x
e
f
k
block

4.3

Network Flow

A *network* is a directed graph with nonnegative capacity $c(e)$ on each edge e, a source vertex s, and a sink vertex t. A flow $f(e)$ assigns a value to each edge. We write $f^+(v)$ for the total flow on edges leaving v and $f^-(v)$ for the total flow on edges entering v.

A flow if feasible if it satisfies the following constraints:

Capacity constraint: $0 \leq f(e) \leq c(e)$

Conservation constraint: $f^+(v) = f^-(v)$ for each vertex $v \notin \{s, t\}$

The value, $\text{val}(f)$, is the net flow $f^-(t) - f^+(t)$ into the sink. The maximum flow is a feasible flow of maximum value.

When f is feasible, a f-augmenting path is a source to sink path P in G such that for each $e \in E(P)$:

- If P follows e in the forward direction, then $f(e) < c(e)$.
- If P follows e in the backward direction, then $f(e) > 0$.

Let $\epsilon(e) = c(e) - f(e)$ when e is forward on P and $\epsilon(e) = f(e)$ when e is backward on P. The tolerance of P is $\min_{e \in E(P)} \epsilon(e)$.

Lemma 4.3.5

If P is a f-augmenting path with tolerance z, then changing the flow by $+z$ on foward edges in P and by $-z$ on backward edges in P, then we yield a new flow f' with $\text{val}(f') = \text{val}(f) + z$.

The definition of tolerance ensures that $0 \leq f'(e) \leq c(e)$ for every edge e, meaning that the capacity constraint holds.

The edges of P incident to an internal vertex occur either as $(+,+), (-,+), (+,-), (-,-)$. In each case, the change to the flow into and out of v remains the same, so the net flow remains 0.

Finally, the net flow into t increases by z.

A source/sink cut, $[S, T]$ consists of the edges from a source set S and a sink set T, where $s \in S$ and $t \in T$. The capacity of the cut is denoted $\text{cap}(S, T)$ and sum of the capacities of the edges of $[S, T]$.

Given a set of vertices U, let $f^+(U)$ denote the total flow on edges leaving U and $f^-(U)$ denote the total flow on edges entering U. The net flow out of U is $f^+(U) - f^-(U)$.

Lemma 4.3.7

If U is a set of vertices in a network, then the net flow out of U is the sum of net flows out of the nodes of U. In particular, if f is a feasible flow and $[S, T]$ is a source/sink cut, then the net flow out of S is equal to the net flow into T, which is equal to $\text{val}(f)$.

Our claim is equivalent to the following:

$$f^+(U) - f^-(U) = \sum_{v \in U} [f^+(v) - f^-(v)]$$

We will work in three cases:

$x, y \in U$: In this case, $f(xy)$ is not counted on the left side. On the right side, $f(xy)$ contributes positively via $f^+(x)$ and contributes negatively via $f^-(y)$, which are equivalent, meaning that, in this scenario, they are equivalent.

$x, y \notin U$: In this case, $f(xy)$ contributes to neither side, so they are equivalent once more.

$xy \in U, \overline{U}$: In this case, $f(xy)$ contributes positively on both sides.

$xy \in \overline{U}, U$: In this case, $f(xy)$ contributes negatively on both sides.

Summing over all edges, we get equality.

When $[S, T]$ is a source/sink cut, and f is a feasible flow, then the net flow from vertices in S sums to $f^+(s) - f^-(s)$ (as every interior node has net flow of 0), and net flow of nodes from T sums to $f^+(t) - f^-(t)$, which equals $-\text{val}(f)$. Hence, the net flow across any source/sink cut equals both the net flow out of s and the net flow into t.

Corollary 4.3.8

If f is feasible and $[S, T]$ is a source/sink cut, then $\text{val}(f) \leq \text{cap}(S, T)$.

By the previous lemma, we know that $\text{val}(f) = f^+(S) - f^-(S) \leq f^+(S)$, since the flow into S is no less than 0. Since the capacity constraint requires $f^+(S) \leq \text{cap}(S, T)$, we get that $\text{val}(f) \leq \text{cap}(S, T)$.

Ford-Fulkerson Algorithm

Among all source-sink cuts, the one with minimum capacity yields the maximum bound on flow. This is the *minimum cut* problem. The max flow and min cut problems are dual optimization problems.

Given a flow with value α and a cut with value α, corollary 4.3.8 proves that the cut is a minimum cut and the flow is a maximum flow. The Ford-Fulkerson algorithm finds augmenting paths to increase the flow value. If it does not find any larger flow, then it finds a cut with the same capacity as this flow.

Input A feasible flow f into a network.

Output An f-augmenting path or a cut with capacity $\text{val}(f)$.

Idea Find the nodes reachable from s by paths with positive tolerance. Reaching t completes an f-augmenting path. During the search, R is the set of nodes reached and $S \subseteq R$ is the set of nodes searched.

Initialization $R = \{s\}$, $S = \emptyset$

Iteration Choose $v \in R - S$. For each exiting edge vw with $f(vw) < c(vw)$, and $w \notin R$, add w to R. For each entering edge uv with $f(uv) > 0$, and $u \notin R$, add u to R.

Label each vertex added to R as "reached" and record v as the vertex reaching it. After exploring all edges at v, add v to S.

If t is reached, then trace the path reaching t to report an f-augmenting path, and terminate. If $R = S$, then return $[S, \overline{S}]$ and terminate. Otherwise, iterate.

Crucially, the Ford-Fulkerson algorithm works best with rational capacities and flows, or else the algorithm will iterate forever.

Theorem 4.3.11: Max Flow-Min Cut

In every network, the maximum value of a feasible flow equals the minimum capacity of a source/sink cut.

In the max flow problem, the zero flow ($\forall e, f(e) = 0$) is always a feasible flow. Afterwards, we apply the Ford-Fulkerson algorithm, adding vertices to S at most once, and terminating either when $t \in R$ or when $S = R$.

If $t \in R$, we have an f-augmenting path, and increase the flow value. After repeating the Ford-Fulkerson algorithm, when the capacities are rational, each augmentation increases the flow by a multiple of $1/a$ where a is the LCM of the denominators. After finitely many repetitions of the algorithm, the capacity of the cut is reached, and our algorithm terminates with $S = R$.

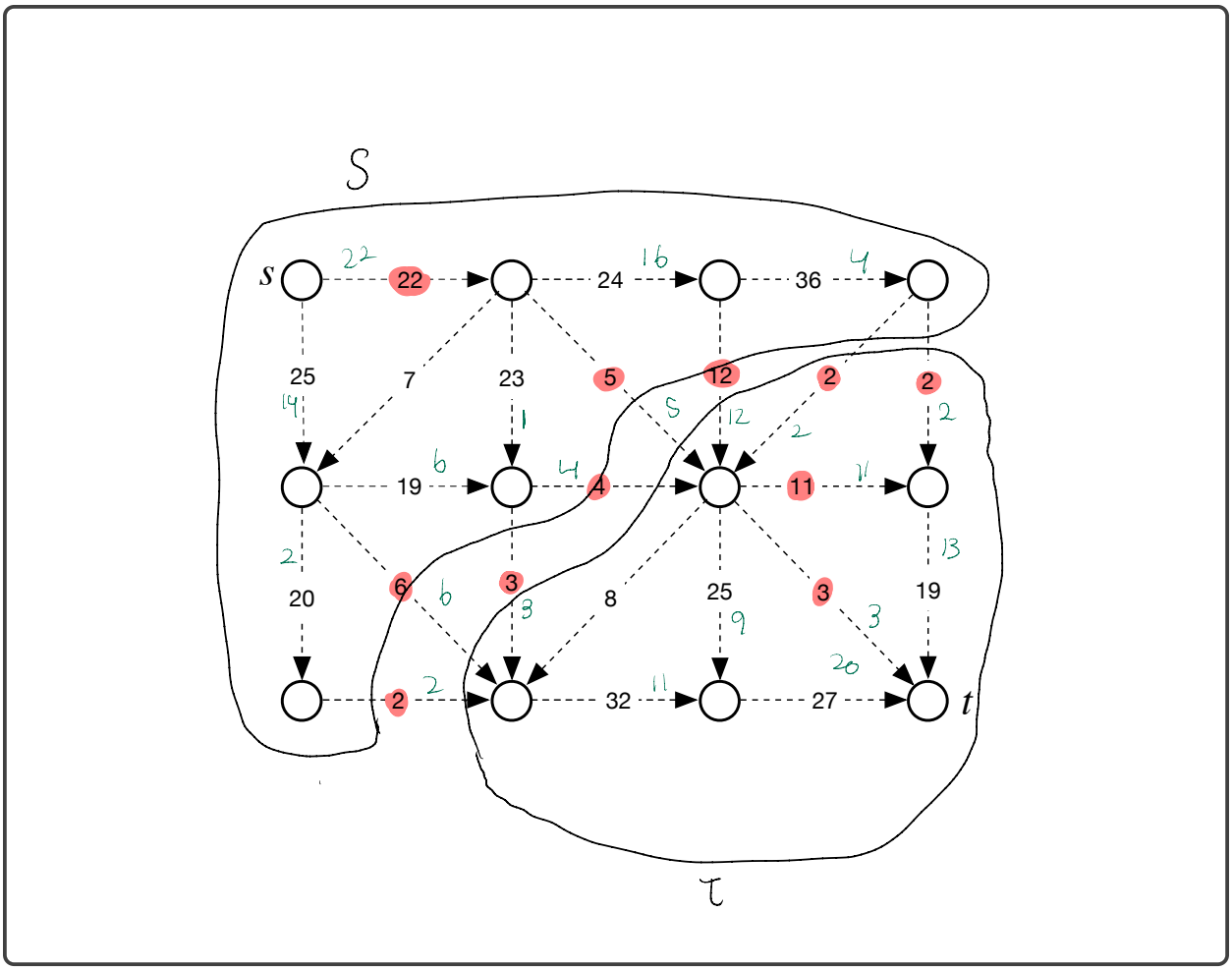
When we terminate with $S = R$, we claim that, for $T = \overline{S}$, this is a $[S, T]$ cut with capacity $\text{val}(f)$ for f as the present flow. Because $s \in S$ and $t \notin S = R$, the algorithm termination, this termination step is a cut. Additionally, since applying the labeling algorithm introduces no vertex in T to R, no edge from S to T has excess capacity, and no edge from T to S has nonzero flow in f. So, $f^+(S) = \text{cap}(S, T)$ and $f^-(S) = 0$, meaning $\text{val}(f) = \text{cap}(S, T)$.

Ford-Fulkerson Algorithm Example

To use the Ford-Fulkerson algorithm on paper, we do the following:

- Identify an f-augmenting path from s to t.
- Add the tolerance along each edge of the path.
- Mark any edges where the flow is equal to the capacity as "full."
- Continue until you cannot find any f-augmenting paths from s to t.

We can see an example of a completed Ford-Fulkerson algorithm:


S
s
22
24
36
25
7
23
5
12
2
2
19
4
11
20
6
3
8
25
3
19
2
32
27
t
τ