Graph Connectivity and Minimum Cuts

1 Basic Concepts

Often we want to find a *minimum cut* in the *entire graph,* not just between a specified source and destination. The size of this minimum cut characterizes the *connectivity* of the graph. It is important to study it, as it allows to decide how vulnerable the network is to link or node failures.

Exercise: Create an example where the (edge-)connectivity of the graph is 3 times smaller than the connectivity between a given source-destination pair.

Let us consider an undirected graph, which is also unweighted, that is, no capacities assigned to the edges. If it models the topology of a network, then the size of the minimum cut tells us that minimum how many links have to fail to disconnect the network (assuming, of course, that it was originally connected). This is the *edge-connectivity*. We can similarly define *node connectivity*, as well. These connectivity values characterize the *vulnerability* of the network, that is, how easy it is to disconnect it. We mainly focus on the (simpler) edge-connectivity. Therefore, often we just call it connectivity, for short.

Definitions

- **Edge-connectivity between two nodes:** $\lambda(x, y)$ = minimum number of edges that need to be deleted to disconnect nodes $x \neq y$.

  $\lambda(x, y)$ is the size of a *minimum cut* between x and y.

- **Edge-connectivity (of the graph):** $\lambda(G)$ = minimum number of edges that need to be deleted to disconnect G.

  $$\lambda(G) = \min_{x \neq y} \lambda(x, y).$$

  $\lambda(G)$ is the size of a *minimum cut.*

- **Node-connectivity of the graph:** $\kappa(G)$ = minimum number of nodes (vertices) that need to be deleted, such that the remaining graph is either disconnected, or it has no edge at all. $\kappa(G)$ is the size of a *minimum vertex-cut*, also called *node-cut*.

Naming convention: If a graph has $\lambda(G) = k$, we call it k-connected. Similarly, if $\kappa(G) = k$, then the graph is called k-node-connected. Occasionally we also use the phrase that the graph is $\geq k$-connected, to express that $\lambda(G) \geq k$.

Exercises

1. Construct a graph with $\kappa(G) = 2$ and $\lambda(G) = 3$.

2. Let $\delta(G)$ denote the minimum node degree in a graph G. A basic result is that the following holds for every graph G:

$$\kappa(G) \leq \lambda(G) \leq \delta(G). \tag{1}$$

   (Sometimes it is referred to as *Whitney's Theorem*.) Prove it!

   Solution: Regarding the minimum degree, it is clear that if we remove all the $\delta(G)$ edges that are adjacent to a node of minimum degree, then it gets disconnected from the rest of the graph. Thus, we obtain

$$\lambda(G) \leq \delta(G). \tag{2}$$

   To prove $\kappa(G) \leq \lambda(G)$, assume $\lambda(G) = k$ (note that we must have $k \geq 1$, as we deal with connected graphs). Then the nodes of G can be partitioned into 2 sets, say A and B, such that precisely k edges go between the two sets. Let us call this set of k edges E, and let $e = \{x, y\}$ be one of them. For every other edge in E, if any, pick one of its endpoints, which is different from x and y. Clearly, each edge must have such an endpoint, otherwise it would coincide with e. Let W be the set of the points we picked (possibly $W = \emptyset$, if $k = 1$).

   Since the cut has $k - 1$ edges other than e, we have $|W| \leq k - 1$ (the reason for possibly not having exact equality is that some edges may share an end node, and that is picked only once).

   Now set $C = W \cup \{x\}$. Clearly, $|C| \leq k$, due to $|W| \leq k-1$. We claim that C is a node-cut. Why?

- C contains at least one endpoint from each cut edge, so the removal of C removes all cut edges. If there are no more edges at all in the graph, then C satisfies the definition of a node-cut.
- If, after the removal of C, at least one edge remains, then such an edge must be fully either in A or B, since it cannot be in the cut, as the cut edges have all been removed. Let $e' = \{u, v\}$ be such an edge. Pick the endpoint of the cut edge $e = \{x, y\}$ that is not in the same set as u. Let us say, this endpoint of e is y (since we can name the nodes arbitrarily). Then u and y are separated by C, since they are on different sides of the cut, and the removal of C removes all edges in the cut. Therefore C is a node-cut, as it separates (at least) two nodes.

Knowing $|C| \leq k$, and that C is a node-cut, we have

$$\kappa(G) \leq |C| \leq k = \lambda(G). \tag{3}$$

Inequalities (3) and (2) together imply (1) that we wanted to prove.

Comment. The graph parameters $\kappa(G)$, $\lambda(G)$ and $\delta(G)$ are positive integers for any connected graph G, with at least two nodes. Furthermore, as we have seen, they must always satisfy the condition $\kappa(G) \leq \lambda(G) \leq \delta(G)$.

At this point it is natural to ask: is there any other condition they must satisfy, so that a graph exists with the given parameters? Interestingly, the (nontrivial) answer is that nothing else is imposed:

Theorem. *For any three positive integers k, ℓ, d, if they satisfy*

$$k \leq \ell \leq d,$$

then there exists a graph G, such that

$$\kappa(G) = k, \qquad \lambda(G) = \ell, \qquad \delta(G) = d.$$

3. We want to design a network topology, represented by an undirected graph, such that it satisfies the following conditions:

   - If any 3 nodes fail, the network remains connected.
   - If any 4 links fail, the network remains connected.
   - Every node has at most 9 neighbors.
   - The *average* number of neighbors the nodes have is at most 6.
   - Any two nodes are reachable from each other in at most 3 hops.
   - The network has exactly 30 nodes.

   Solution. Let us denote the maximum degree by $\Delta(G)$, the average degree by $\overline{d}(G)$, and the diameter by $\mathrm{diam}(G)$. (The diameter is defined as the smallest integer k, such that any two nodes can reach each other on a path of length at most k.)

   Then the conditions mean that we are looking for a graph with the following parameters:

   $$\kappa(G) \geq 4$$
   $$\lambda(G) \geq 5$$
   $$\Delta(G) \leq 9$$
   $$\overline{d}(G) \leq 6$$
   $$\mathrm{diam}(G) \leq 3$$
   $$n = 30.$$

   How to find such a graph?... See one on the next page.

How was this graph found? Designing it by hand may be a daunting task...

Answer: Check out the website *House of Graphs*.

Comment: The above exercises suggest the following general *graph synthesis* problem:

> *Given a list of graph parameter values, find a graph with these parameters.*

Unfortunately, no efficient algorithm is known to solve this for an *arbitrary* set of parameters.

4. Consider a connected network with undirected links. We have two pieces of information about the network:

   - The network has 37 nodes.
   - It is impossible to disconnect the network by removing 2 or fewer links.

   Justify wether the following statement is true or not:

   > There *must* be a node with at least 4 links adjacent to it.

   Solution: If it is impossible to disconnect the network by removing 2 links, then each node must have degree ≥ 3. Let us assume that the degree of each node is *exactly* 3. Then, if we sum up the degrees, we get $3 \times 37 = 111$. On the other hand, the sum of the degrees is twice the number of edges, since each edge contributes 2 to the sum (1 at each of its endpoints). Thus, we get $111 = 2m$, where m is the number of edges. Since m must be an integer, the equality

   $$111 = 2m$$

   cannot hold, as the right-hand side is even, the left-hand side is odd. Thus, the assumption that all degrees are 3 leads to a contradiction. Therefore, there must be a node with degree $d \neq 3$. Since we cannot disconnect the network by removing 2 or fewer links, $d \leq 2$ is impossible. This yields that this node must have degree at least 4.

5. Consider a network, represented by a k-connected graph G. We want to extend the network by adding a new node u. We decide to connect the new node u to ℓ different old nodes $v_1, \dots v_\ell$ that we can chose from G. What should be the value of ℓ, and how should we choose the ℓ neighbors $v_1, \dots v_\ell$ of u, if we want to guarantee that the extended network preserves k-connectivity?

   Solution: We prove that if we chose *any* k different nodes for the neighbors of u, the resulting graph G' is always k-connected. That is, we can use $\ell = k$, and the neighbors can be chosen arbitrarily.

   Proof. Assume that $\lambda(G') < k$. Then there is a cut C containing $\leq k-1$ edges, such that their removal separates the nodes of G' into two sets A, B with no connection to each other. Let us call A the set that contains u. We observe that u cannot be alone in A, since then all its neighbors would be in B, which means all the k edges adjacent to u would be in the cut, contradicting to $|C| \leq k-1$. If, however, u is not alone in A, then after removing u, we get a decomposition of the original nodes into two non-empty sets ($A - \{u\}$ and B), such that they are separated by the cut C. Since $|C| \leq k-1$, this would contradict to $\lambda(G) = k$. Therefore, we obtain that no cut of size $\leq k-1$ can exist in G', so $\lambda(G') \geq k$ must hold.

6. Based on the idea of Exercise 5, one can also prove the following result:

> *Let G_1, G_2 be two k-connected graphs, such that their node sets are disjoint. Connect the two graphs arbitrarily by k new edges, to obtain a larger graph G. Then the resulting graph G remains k-connected.*

Proof. Assume indirectly that the resulting graph has $\lambda(G) < k$. Then its nodes can be partitioned into two non-empty sets, A and B, such that they are separated by a cut C containing at most $k-1$ edges. Now a key observation is that G_1 must be either fully in A or fully in B. The reason is that if G_1 has nodes in both sets, then its nodes in A would be separated from its nodes in B by the cut C. Since $|C| \leq k-1$, this would contradict to $\lambda(G_1) = k$. As the naming of the sets is irrelevant, let us say that G_1 falls fully in A.

The same argument applies to G_2, as well, so it must also fall fully in one of the sets. Since A, B are non-empty, and they contain no other nodes than the original ones, therefore, G_2 must fall fully in the other set B. This means, the subgraph induced by A is equal to G_1, and the subgraph induced by B is equal to G_2. The sets A, B, however, are connected to each other by at most $k-1$ edges, according to the indirect assumption. This contradicts to the construction, in which we connected G_1 and G_2 by k edges.

This result can be used, for example, to *optimally* solve the following network design problem, which would otherwise look quite hard:

> Given two k-connected networks, connect them into a single k-connected network, by adding new links. Assume that every added new link incurs a given (possibly different) cost, and we want to achieve the goal with minimum cost.

This could appear as a hard optimization problem, since one can choose the connecting links in exponentially many different ways, and we have to find the least expensive solution under the constraint that the resulting larger network remains k-connected.

However, in view of the above result, we can simply choose *any* k links to connect the two networks, the obtained larger network will always

be k connected. Therefore, the solution that attains the minimum cost is simply the one in which we select the k least expensive links.

2 Some Fundamental Theorems

A fundamental question is that how many disjoint paths (routes) can be guaranteed between any two nodes. We focus here on the link-disjoint case. In a network topology this is important to ensure backup routes in case of link failures. A fundamental charaterization is provided by the following theorem.

Theorem (Menger's Theorem) *For any connected graph G the following hold:*

- *For any two different nodes x, y the value of $\lambda(x, y)$ is equal to the maximum number of edge-disjoing paths connecting x and y.*
- *$\lambda(G)$ is equal to the smallest value k with the property that between* any *two nodes there are at least k edge-disjoint paths.*
- *Analogous statements hold for node connectivity, as well.*

Exercises

1. Consider an optical network. Assume we know that this network cannot be disconnected by two link failures. We want to select for any two nodes a primary route to carry the traffic between them. Furthermore, we also want to select two backup routes for the same node pair, such that the backup routes are link disjoint from the primary route, and also from each other, to provide adequate protection whenever at most two links fail simultaneously. Is it true that such a route system can always be selected under the given conditions?

   Solution: The answer is yes. Since 2 link failures cannot disconnect the network, we have $\lambda(G) \geq 3$. Then, by Menger's Theorem, between

any two nodes there is a route system that contains (at least) 3 link-disjoint routes.

2. In a network two disjoint sets of nodes, A and B are selected, each containing k nodes. We know that the network is k-connected. Is it always possible to connect each node in A with a node in B, such that the connecting routes are link-disjoint, and, moreover, they do not share any end-nodes?

   Solution: Yes. Let us add a new node u and connect it to all nodes in A. Similarly, add another new node v and connect it to all nodes in B. By Exercise 5 in Section 1, the extended graph remains k-connected. Then, by Menger's Theorem, there are k edge-disjoint paths connecting u and v. The parts of these paths that fall in the original graph satisfy the requirements.

Another important question for network topology design is this: what is the maximum connectivity that can be achieved with a given number of nodes and edges? Or, turning the question around: for a given number of nodes we can look for the minimum number of edges that makes it possible to satisfy a given connectivity requirement (node or edge-connectivity). Interestingly, there is a surprisingly simple formula answering the question:

Theorem (Harary) *The achievable maximum node and edge connectivites that a simple graph on n nodes with m edges can have are given by*

$$\max \kappa(G) = \max \lambda(G) = \left\lfloor \frac{2m}{n} \right\rfloor$$

assuming $n - 1 \leq m \leq n(n-1)/2$, which is necessary for a connected graph to exist. Then the optimal connectivity can be achieved by the following construction: if $m = n - 1$, then take any tree on n nodes; otherwise follow the following steps:

- *Take a circle on n nodes.*
- *Set $\delta = \lfloor 2m/n \rfloor$.*
- *Connect any two nodes that are within distance $\lfloor \delta/2 \rfloor$ along the circle.*

- *If not all nodes have degree $\geq \delta$, then pick a node with degree $< \delta$, connect it to a node that is at distance $\lfloor n/2 \rfloor$ away along the circle. Repeat this until all nodes have degree δ, possibly with a single node that can have degree $\delta + 1$.*

Exercise

Design a minimum cost network topology on 11 nodes, if each link costs 1 unit of cost, and the network is required to have node connectivity at least 5. Justify that the solution indeed has minimum cost.

3 Algorithms

The minimum cut between two specified nodes can be obtained as a by-product of the maximum flow computation. If, however, we want an overall minimum cut in the whole graph, then a single maximum flow computation does not suffice.

Interestingly, one can find an overall minimum cut *directly,* without using anything about maximum flows. Below we present two algorithms for this problem.

3.1 The Contraction Algorithm

This algorithm uses *randomization*, so it will guarantee the result only with a certain probability, less than 1, but it can be made arbitrarily close to 1.

A single run of the algorithm works as follows (we will have to repeat these runs sufficiently many times, with independent random choices).

Step 1 Pick an edge (v, w) uniformly at random among all edges from the current graph G.

Step 2 Contract the selected edge (i.e., merge its endpoints). Keep the arising parallel edges, but remove the self-loops, if any. The contracted graph, denoted by $G/(v,w)$, will be the new value of G.

Step 3 Are there still more than 2 nodes? If yes, repeat from Step 1. If no, then STOP, the set of edges between the 2 remaining nodes form a cut.

Note that the obtained cut is not necessarily minimum. On the other hand, as shown in the next (optional) lecture note, it will be a minimum cut with probability at least $1/n^2$ in a graph on n nodes. Therefore, if we repeat the algorithm K times independently and choose the smallest of the found cuts, say C, then the following holds:

$$\Pr(C \text{ is not a minimum cut}) \leq \left(1 - \frac{1}{n^2}\right)^K$$

If K is chosen as $K = \ell n^2$, then the above probability will be approximately $\mathrm{e}^{-\ell}$, which is very small already for moderate values of ℓ.

An interesting by-product of this algorithm is the theorem that the number of minimum cuts in a graph is at most n^2, since each arises with at least $1/n^2$ probability in a run and these are mutually exclusive events. Note that the total number of different cuts may well be exponentially large, but only at most n^2 of them can be minimum (actually, n^2 is only an upper bound, the precise tight bound is $n(n-1)/2$).

Exercise

Show that the bound $n(n-1)/2$ on the number of minimum cuts is tight (that is, it can hold with equality).

Solution: Consider a graph on n nodes which is a simple circle (ring). It cannot be disconnected by removing a single edge, but it can be disconnected by removing 2 edges, so the minimum cut size is 2. If we remove *any* 2 edges from the graph, it gets disconnected. Therefore, every pair of edges form a minimum cut. Since there are n edges in a circle on n nodes, we can choose $n(n-1)/2$ different minimum cuts, which is exactly the bound on the number

of minimum cuts. Thus, in this case the bound holds with equality, which shows that it is tight.

3.2 The Nagamochi-Ibaraki Algorithm for Minimum Cut

A clever deterministic algorithm for minimum cuts was discovered by Nagamochi and Ibaraki. Recall that the (edge-)connectivity of a graph G is denoted by $\lambda(G)$; and $\lambda(x, y)$ denotes the connectivity between two different nodes x, y.

Let G_{xy} be the graph obtained from G by contracting (merging) nodes x, y, whether or not they are originally connected. In this operation we omit the possibly arising self-loops (if x, y are connected in G), but keep the parallel edges.

We can now use the following result that is proved among the exercises:

For any two nodes x, y

$$\lambda(G) = \min\{\lambda(x, y),\ \lambda(G_{xy})\} \tag{4}$$

Thus, if we can find two nodes x, y for which $\lambda(x, y)$ can be computed easily (without a flow computation), then the above formula yields a recursive algorithm.

One can indeed easily find such a pair x, y of nodes. This is done via the so-called *Maximum Adjacency (MA) ordering.* An MA ordering $v_1, \ldots, v_n$ of the nodes is generated recursively the by the following algorithm:

- Take any of the nodes for v_1.
- Once $v_1, \ldots, v_i$ is already chosen, take a node for v_{i+1} that has the maximum number of edges connecting it with the set $\{v_1, \ldots, v_i\}$.

Nagamochi and Ibaraki proved that this ordering has the following nice property:

Theorem *In any MA ordering $v_1, \ldots, v_n$ of the nodes*

$$\lambda(v_{n-1}, v_n) = d(v_n)$$

holds, where $d(.)$ denotes the degree of the node.

Therefore, it is enough to create an MA ordering, as described above, and take the last 2 nodes in this ordering for x, y. Then the connectivity between them will be equal to the degree of the last node in the MA ordering. Then one can apply formula (4) recursively to get the minimum cut. Here we make use of the fact that it reduces the problem to computing $\lambda(G_{xy})$ and G_{xy} is already a smaller graph.

Exercises

1. Prove that $\lambda(G) \leq \lambda(G_{xy})$ always holds, assuming G_{xy} still has at least two nodes. In other words, merging two nodes in a graph G that has at least 3 nodes, can never decrease the edge-connectivity.

   Solution: Let u denote the node obtained by merging x and y. Let C be an edge set that forms a minimum cut in G_{xy}. Then $|C| = \lambda(G_{xy})$, and C cuts the node set of G_{xy} into two disjoint, non-empty subsets. Let us call these sets A and B, such that the one that contains the node u is called A. Let us now undo the merging, such that u is turned back into the original two nodes x, y, still keeping them in the set A. Then C will still separate A and B, so it is a cut in the resulting graph. However, the resulting graph is just the original graph G, since undoing the merging gives back the original graph. Thus, C remains a cut in G, but not necessarily a minimum cut. Therefore, $|C| \geq \lambda(G)$. A the same time, we had $|C| = \lambda(G_{xy})$, yielding $\lambda(G_{xy}) \geq \lambda(G)$, which is what we wanted to prove.

2. Building on Exercise 1, prove

   $$\lambda(G) = \min\{\lambda(x, y),\ \lambda(G_{xy})\}.$$

   Solution: Let C be an edge set that forms a minimum cut of G. Then $|C| = \lambda(G)$ holds. Let A, B be the two node sets into which C cuts the nodes of G. Now we can consider two cases:

a.) The nodes x, y are on different sides of the cut C. In this case C is also a cut between x and y. Since C is a minimum cut in the whole graph, there cannot be a smaller cut between x and y, so C is a minimum cut between x and y. Therefore, in this case, $\lambda(G) = \lambda(x, y)$ holds.

b.) The nodes x, y are on the same side of the cut C. Then, after merging x and y, C still remains a cut in the new graph G_{xy}. Since C cannot be smaller than a minimum cut in G_{xy}, therefore, $|C| \geq \lambda(G_{xy})$. We chose C, such that $|C| = \lambda(G)$, yielding $\lambda(G) \geq \lambda(G_{xy})$. On the other hand, we already know from Exercise 1 that $\lambda(G) \leq \lambda(G_{xy})$. Therefore, in this case, the only possibility is $\lambda(G) = \lambda(G_{xy})$.

Thus, we obtain that in case a) $\lambda(G) = \lambda(x, y)$, while in case b) $\lambda(G) = \lambda(G_{xy})$. We also know that $\lambda(G) \leq \lambda(x, y)$ (by definition), and $\lambda(G) \leq \lambda(G_{xy})$ (by Exercise 1). Since precisely one of cases a), b) must occur, therefore, by combining the above facts, we obtain that $\lambda(G)$ must be equal to the smaller of $\lambda(x, y)$ and $\lambda(G_{xy})$. This means, $\lambda(G) = \min\{\lambda(x, y), \lambda(G_{xy})\}$, as desired.

3. Create a pseudo-code for the Nagamochi-Ibaraki minimum cut algorithm.

Comments on Minimum Cut Algorithms

1. The Contraction Algorithm is much simpler, but it is randomized, so it does not give a 100% guarantee that the found cut is indeed minimum. There exist de-randomized (i.e., deterministic) versions of the Contraction Algorithm, but they are more complicated than the simple randomized version.

2. An interesting, already mentioned, by-product of the Contraction Algorithm is that any n-node graph can have at most $n(n-1)/2 = O(n^2)$ different minimum cuts. Furthermore, they can all be listed by a polynomial-time algorithm.

3. The above can be generalized to *approximately* minimum cuts. Let us say that a cut is α-minimum, if the number of edges in it is within α times the minimum, where $\alpha > 1$ is a constant. Then one can prove that the number of α-minimum cuts in any n-node graph is at most $O(n^{2\alpha})$. They can be all listed by a polynomial-time algorithm, as well.

Exercise. Given a k-connected network, create a polynomial-time algorithm that can decide whether it can be disconnected by at most $3k$ isolated link failures. (Isolated link failures: the failing links do not share any node.)

Solution (sketch): This kind of disconnection can happen precisely when there is a cut in the graph that contains at most $3k$ edges, and they are all disjoint. In other words, they form a matching, so such a cut is called a *matching cut.*

Generally, the problem whether a graph contains a matching cut is known to be NP-complete. We, however, want to decide only if there is a matching cut of size $\leq 3k$, where $k = \lambda(G)$.

By the above discussion, all α-minimum cuts can be listed in polynomial time, and their number is bounded by $O(n^{2\alpha})$. In our case $\alpha = 3$, so we can list all 3-minimum cuts; there will be at most $O(n^6)$ of them. we can directly check for each one whether it is a matching cut or not, i.e., it consists only of disjoint edges or not. If at least one of them does, then $\leq 3k$ isolated link failures can disconect the network, otherwise not.