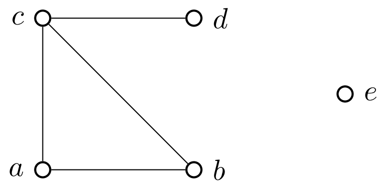
1 Paths and cycles

A *path* in a graph is a sequence of distinct vertices $v_0, \ldots, v_\ell$ such that $\{v_i, v_{i+1}\}$ is an edge in the graph for all $0 \le i < \ell$. We can also describe a path as a sequence of the $k-1$ edges $\{v_0, v_1\}, \ldots, \{v_{\ell-1}, v_\ell\}$. The vertices v_0 and v_ℓ are the *endpoints* of the path. The number ℓ is the *length* of the path. (The definition allows for paths of length zero.) If there exists a path with endpoints v and w we say that v is *connected to w*.

For example, in the graph given by the following diagram, a, c, d is a path of length 2 and a, b, c, d is a path of length 3. The sequence a, b, e is not a path because $\{b, e\}$ is not an edge, and a, b, c, a is not a path because vertex a occurs more than once.



A *cycle* in a graph is a cyclic sequence[1] of distinct vertices $v_1, \ldots, v_\ell$ such that $\{v_1, v_2\}, \ldots, \{v_{\ell-1}, v_\ell\}$, $\{v_\ell, v_1\}$ are edges and they are all distinct. The number ℓ is called the *length* of the cycle. For example, the above graph has only one cycle — the cycle a, b, c — and this cycle has length 3. (The definition does not allow cycles of length zero.)

If we do not impose the requirement that $v_1, \ldots, v_\ell$ are all distinct, then we call $v_1, \ldots, v_\ell$ a *closed walk*. Any graph with at least one edge has an infinite number of closed walks: If $\{u, v\}$ is an edge, then u, v; u, v, u, v; u, v, u, v, u, v, and so on are all closed walks.

2 Connectivity, trees, and forests

A graph is *connected* if for every pair of vertices u, v, u is connected to v. The above graph is not connected because there exists no path from a to e.

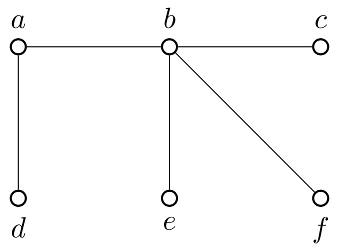
A *connected component* of a graph G is a subgraph of G consisting of all vertices that are connected to a given vertex and all edges incident to them. This graph has the following two connected components:

$$\begin{aligned}&(V_1, E_1), \quad \text{where } V_1 = \{a, b, c, d\} \text{ and } E_1 = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}\}\\ &(V_2, E_2), \quad \text{where } V_2 = \{e\} \text{ and } E_2 = \varnothing.\end{aligned}$$

It should be clear that there can be no edges between vertices that belong to different connected components of the same graph.

A *forest* is a graph that has no cycles. A *tree* is a connected forest. A *leaf* in a forest is a node of degree 1. Here is a diagram of a tree with four leaves:

[1] Cyclic means that sequences (a, b, c); (b, c, a), (c, b, a) etc. all represent the same cycle.



Theorem 1. *Every forest with n vertices and m edges has $n - m$ connected components.*

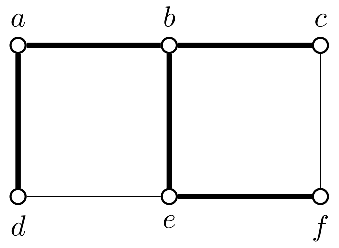
Proof. We prove the theorem by induction on m.
Base case $m = 0$: If a forest has no edges, every vertex is its own connected component, so there are exactly n of them.
Inductive step: Assume that every forest with m edges has $n - m$ connected components, where n is the number of vertices. Let G be a forest with n vertices and $m + 1$ edges. Remove an arbitrary edge e from G (but do not modify its vertices). The resulting graph, call it G', is a forest with n vertices and m edges, so by our inductive assumption it has $n - m$ connected components.

Both vertices of e cannot belong to the same connected component of G', for then the path between the endpoints of e together with e would form a cycle in G.

Therefore the vertices of e are in two different connected components of G'. All other connected components of G' except for these two stay the same in G, so G has one fewer connected component than G'. Therefore G has exactly $(n - m) - 1 = n - (m + 1)$ connected components. □

Corollary 2. *In a tree, the number of vertices equals the number of edges plus one.*

A *spanning tree* of a connected graph G is a subgraph of G that includes all the vertices and is a tree. Here is a diagram of a graph and one of its spanning trees. The edges of the spanning tree are marked by thick lines:



Theorem 3. *Every connected graph has a spanning tree.*

Proof. The proof is by induction on the number of edges. If the graph has no edges, then for it to be connected it must consist of a single vertex. This vertex is then a spanning tree.

Now assume that every connected graph with m edges has a spanning tree. Let G be a graph with $m + 1$ edges. We consider two cases. If G is a tree, then G is its own spanning tree. Otherwise, G has a cycle. Take any edge e on this cycle and remove it. The remaining graph G' is connected: If any path between two vertices of G uses the edge e, it can be rerouted along the other edges of the cycle. Since G' has m edges, by the inductive hypothesis it has a spanning tree. Therefore G also has a spanning tree. □

The proof of this theorem tells us how to find a spanning tree in a connected graph: If the graph is a tree already, we are done. Otherwise, it must contain a cycle. Remove one of the edges in this cycle and recursively find a spanning tree in this subgraph.

Here is a corollary of Theorem 3 and Theorem 1:

Corollary 4. *Every graph with n vertices and m edges has at least $n - m$ connected components.*

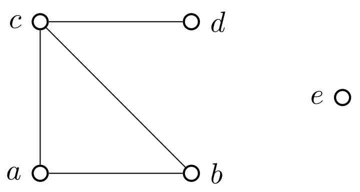
Proof. Let G be a graph with n vertices and m edges. By Theorem 3, every connected component of G has a spanning tree. Let F be the union of all these spanning trees. Then F is a forest with n vertices, $m' \leq m$ edges, and the same number of connected components as G. By Theorem 1, F has $n - m'$ connected components, so G also has $n - m'$ connected components. Since $m' \leq m$, the number of connected components of G is at least as large as $n - m$. □

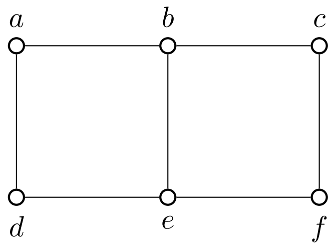
3 Bipartite graphs revisited

Recall that a graph is bipartite if its vertices can be partitioned into two sets L, R so that all edges have one vertex in L and one vertex in R.

Theorem 5. *A graph is bipartite if and only if it has no cycle of odd length.*

Before we prove this theorem, let us look at two familiar examples.





The first graph is not bipartite and the cycle a, b, c has odd length. The "reason" this graph is not bipartite is precisely the existence of this cycle: If we put vertex a on one side of the partition, b goes on the other side and c is on the same side, so the edge $\{a, c\}$ is inconsistent with the partition. We will generalize this argument to rule out the possibility that a graph with a cycle of odd length can be bipartite.

In contrast, the second graph has no cycles of odd length and it is bipartite. One way to find the partition L, R is to start at an arbitrary vertex — say a — and put it in one set of the partition — say L. Then its neighbours b and d must be in R, their neighbours c and e must be in L, and their neighbour f must be in R again. We obtain the partition $L = \{a, c, e\}$, $R = \{b, d, f\}$. In the proof, we will argue that as long as there are no odd length cycles, it is always possible to obtain a valid partition in a similar manner.

The following simple lemma will be useful to have:

Lemma 6. *If a graph has a closed walk of odd length, then it has a cycle of odd length.*

Proof. We prove the lemma by strong induction of then length ℓ of the closed walk.
Base case $\ell = 1$: There are no closed walks of length one, so the base case holds trivially.
Inductive step: Assume that if a graph has a closed walk of odd length up to ℓ, then it has an odd length cycle. Let $v_1, \ldots, v_{\ell+1}$ be a closed walk of length $\ell + 1$ and assume $\ell + 1$ is odd. If all the vertices in the closed walk are distinct, then $v_1, \ldots, v_{\ell+1}$ is a cycle of odd length. Otherwise, two of them must be the same, say v_i and v_j where $1 \leq i < j \leq \ell + 1$. Consider the closed walks:

$$v_i, v_{i+1}, \ldots, v_{j-1} \quad \text{and} \quad v_j, \ldots, v_{\ell+1}, v_1, \ldots, v_{i-1}$$

The sum of the lengths of these two closed walks is $\ell + 1$, so each one has length strictly less than $\ell + 1$, Moreover, $\ell + 1$ is an odd number, so one of them must have odd length. By the inductive assumption, the existence of this walk implies the existence of an odd cycle. □

Proof of Theorem 5. Assume the graph is bipartite with its vertices partitioned into L and R. Suppose $v_1, \ldots, v_\ell$ is a cycle for some $\ell \geq 2$. We will show that ℓ is even. Without loss of generality, we may assume v_1 is in L; if not, we look at the sequence $v_2, \ldots, v_\ell, v_1$, which describes the same cycle. Since v_1 is in L,

v_2 must be in R, v_3 must be in L again, and so on; so v_ℓ is in L if ℓ is odd and in R if ℓ is even. Since $\{v_\ell, v_1\}$ is an edge, v_ℓ must be in R, so ℓ is even.

Now assume the graph, which we call G, has no cycles of odd length. We will prove that it is bipartite.

Suppose that we can show every *connected* graph that has no cycles of odd length is bipartite. Now take an arbitrary, not necessarily connected, graph G with no cycles of odd length. Then each connected component C of G has no cycles of odd length, so its vertices can be partitioned into two sets L_C and R_C so that there are no edges of C within either of L_C and R_C. Let L be the union of all such L_C and R be the union of all such R_C. Then L, R is a partition of the vertices of G. There are no edges within L and no edges within R, so G is bipartite.

Therefore, we can now assume *without loss of generality* that the graph G is connected in addition to having no cycles of odd length. We will show that G is bipartite. Let v_0 be an arbitrary vertex and L and R be the following sets:

$$L = \{w\colon \text{there exists a path of even length with endpoints } v_0, w\}$$
$$R = \{w\colon \text{there exists a path of odd length with endpoints } v_0, w\}$$

Since G is connected, every vertex must be connected to v_0, so it belongs to one of L or R. We will now show that $L \cap R = \varnothing$. The proof is by contradiction. Suppose there was a vertex $u \in L \cap R$. Since u is in L, there is an even length path from v_0 to u:

$$v_0, v_1, \dots, v_s = u, \quad s \text{ is even}$$

Since u is in R, there is an odd length path from v to v_0. The reverse path is an odd length path from u to v:

$$u = v_s, v_{s+1}, \dots, v_{s+t} = v_0, \quad t \text{ is odd.}$$

Let us look at the sequence $v_0, v_1, \dots, v_{s+t-1}$. This is a closed walk of length $s+t$, which is an odd number. By Lemma 6, G has a cycle of odd length, which is a contradiction.

Therefore L, R is a partition of the vertices of G. It remains to show that there can be no edges within L and no edges within R. We first show there are no edges within L. For contradiction, suppose that there was an edge $\{u, w\}$ such that $u \in L$ and $w \in L$. By the definition of L, there must exist even length paths from w to v_0 and from u to v_0. If we take the first path, then the second path in reverse, and finally follow the edge u, w, we obtain a closed walk of even $+$ even $+ 1 =$ odd length. By Lemma 6, this contradicts our assumption. We now show there are no edges within R. The proof is analogous; the only difference is that the two paths now have odd length. The length of the resulting closed walk is now odd $+$ odd $+ 1$, which is still an odd number. □

We state a corollary of this theorem that will be useful for us shortly.

Corollary 7. *Let G be a graph whose edges are the union of two matchings. Then G is bipartite.*

Proof. We show that G has no cycles of odd length. By Theorem 5, G must be bipartite.

For contradiction, assume that there exists an odd length cycle $v_1, v_2, \dots, v_\ell$ in G. Then the edge $\{v_1, v_2\}$ must belong to at least one of the two matchings whose union forms the edges of G. Call this matching Ξ_1. The edge $\{v_2, v_3\}$ cannot belong to Ξ_1 for otherwise Ξ_1 would not be a matching, so it must belong to Ξ_2. Continuing the reasoning in this way, we conclude that for every $1 \le i < \ell$, $\{v_i, v_{i+1}\}$ must be in Ξ_1 if i is odd and in Ξ_2 if i is even, so the edge $\{v_{\ell-1}, v_\ell\}$ must be in Ξ_2. But then the edge $\{v_\ell, v_1\}$ is in Ξ_1, and so v_1 has two neighbours in Ξ_1 (v_2 and v_ℓ). This contradicts our assumption that Ξ_1 is a matching. □

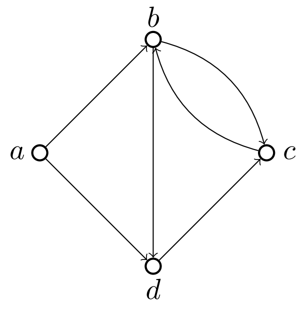
4 Directed graphs

A *simple directed graph* (or *digraph*) G consists of a nonempty set of vertices V and a set of directed edges E, each of which is an ordered pair of vertices (u, v) with $u \neq v$.

In a digraph, (u, v) and (v, u) represent different edges. One of them could be present in the graph, or both, or neither. In a diagram, we represent them as arrows from u to v and from v to u, respectively.

Here is a diagram of the digraph (V, E) where

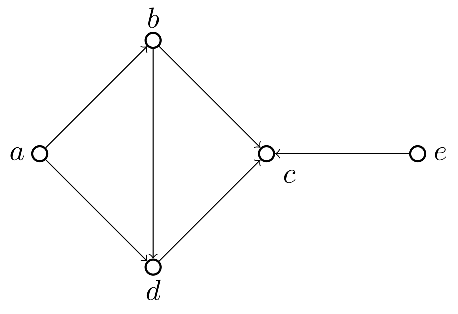
$$V = \{a, b, c, d\} \quad \text{and} \quad E = \{(a, b), (a, d), (b, c), (b, d), (c, b), (d, c)\} :$$



The definitions of "path", "closed walk", and "cycle" in a digraph are identical to those for graphs. For example, a, b, d, c is a path of length 3 in G and b, d, c is a cycle of length 3. G has also a cycle of length 2, namely b, c. In contrast, a graph cannot have a cycle of length 2.

A *topological sort* of a digraph G is an ordering of all its vertices in a sequence so that for every edge (u, v), vertex u comes before vertex v in the sequence. If the vertices represent tasks and edge (u, v) means that task u must be completed before task v then a topological sort represents a possible schedule in which the tasks can be completed one at a time.

The above digraph does not have a topological sort. The vertices b and c cannot be ordered because both b must come before c and c must come before b in the sequence. This is impossible. In contrast, one topological sort of the digraph in the following diagram is the sequence a, b, d, e, c:



The sequence e, a, b, d, c is another topological sort of the same digraph, so topological sorts may not be unique.

Theorem 8. *A digraph has a topological sort if and only if has no cycles.*

We will need the following lemma. A *sink* is a vertex with no outgoing edge. Similarly, a *source* is a vertex with no incoming edge, and a vertex is *internal* if it is neither a source nor a sink.

Lemma 9. *A digraph with no cycles has a sink.*

Proof. We prove the contrapositive: A digraph with no sink has a cycle. Assume that G has no sink. Then every vertex in G has at least one outgoing edge. Visit vertices according to the following procedure, starting from an arbitrary vertex v_0: Keep moving to an adjacent vertex via an outgoing edge. When you reach the first previously visited vertex, stop. Then the sequence of visited vertices will have the following form:

$$v_0, v_1, \dots, v_{k-1}, v_k$$

where $v_0, \dots, v_{k-1}$ are all distinct and $v_k = v_i$ for some i between 0 and $k-1$. Then $v_i, \dots, v_{k-1}$ is a cycle in G, so G has a cycle. □

Proof of Theorem 8. First we show that if a digraph has a topological sort, it cannot have a cycle. For contradiction, suppose $v_1, \dots, v_\ell$ is a cycle. Then v_2 must come after v_1 in the sequence, v_3 must come after v_2, and so on until v_ℓ. So v_ℓ must come after v_1. Since (v_ℓ, v_1) is an edge, v_1 must come after v_ℓ. Contradiction.

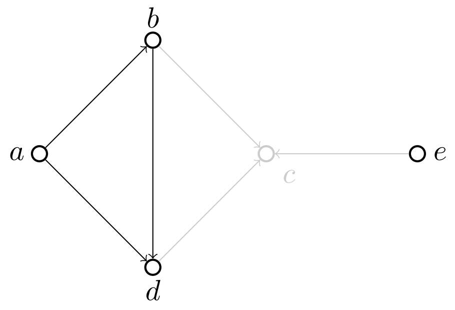
Now we show that if a digraph G has no cycles, then it must have a topological sort. We prove this by induction on the number of vertices n.
Base case $n = 1$: If G has one vertex v, then the sequence v is a topological sort of G.
Inductive step: Assume every digraph with no cycles on n vertices has a topological sort. Let G be a digraph with no cycles on $n+1$ vertices. By Lemma 9, G has a sink t. Remove t and all incoming edges from G. By the inductive assumption, the remaining graph has a topological sort sequence $v_1, \dots, v_n$. Then $v_1, \dots, v_n, s$ is a topological sort of G. □

A digraph with no cycles is called an *acyclic digraph* or a DAG (directed acyclic graph).

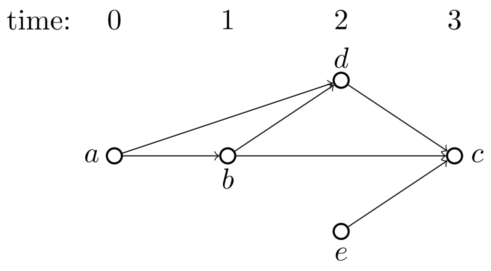
5 Parallel scheduling

The proof of Theorem 8 gives a method for topologically sorting the vertices of a DAG: Find a sink, put it at the end of the sequence, remove it from the graph, then recursively sort the rest.

In the above example, c is the only sink so it must go at the end of the list. After removing it and its incident edges we are left with the following digraph to sort.



There are now two sinks d and e. If these represent tasks to be scheduled, they can be completed in arbitrary order, even at the same time. If we perform d and e at the same time and continue recursively, we obtain the following parallel schedule of the five tasks:



In general, a *parallel schedule* of a DAG G is a partition of the vertices of G into sets $V_0, V_1, \dots, V_\ell$ such for every edge (u, v), if u is in V_i and v is in V_j then $i < j$. The *duration* of the schedule is $\ell + 1$.

In this example, $V_0 = \{a\}, V_1 = \{b\}, V_2 = \{d, e\}, V_3 = \{c\}$ is a parallel schedule of duration 3. Can the duration be reduced to 2? Clearly not because the vertices a, b, d, c form a path, so those tasks will take duration 3 to complete no matter how the other ones are scheduled.

Theorem 10. *For every ℓ, if every path in a DAG G has length at most ℓ then G has a parallel schedule of duration at most $\ell + 1$.*

Proof. The proof is by induction on the length ℓ of the longest path in G.
Base case $\ell = 0$: G consists of isolated vertices and they can all be scheduled at the same time, giving a parallel schedule of duration one.
Inductive step: Assume the proposition is true for ℓ and let G be a DAG whose longest path has length $\ell + 1$. Assign all sinks of G to set $V_{\ell+1}$ in the partition. Every longest path in G must contain a sink; if it didn't it could have been extended by taking an extra edge out of its last vertex. Therefore all paths in the graph G' obtained by removing the vertices V_ℓ and their incident edges from G have length at most ℓ. By inductive hypothesis, G' has a parallel schedule $V_0, \ldots, V_\ell$. Then $V_0, \ldots, V_{\ell+1}$ is a parallel schedule for G. □

An *antichain* in a DAG is a set A of vertices so that there is *no path* between any pair of vertices in A.

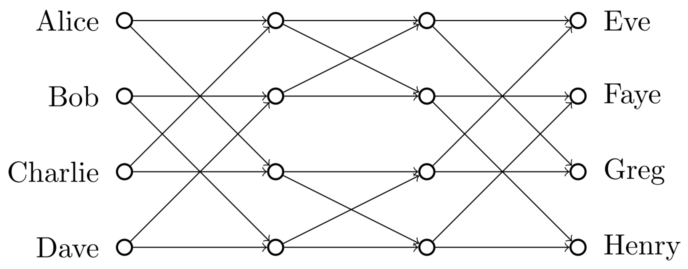
Corollary 11. *For every k, a DAG on n vertices has a path of length at least k or an antichain of size at least n/k.*

Proof. Suppose for contradiction that every path has length less than k and every antichain has size less than n/k. By Theorem 10 the graph has a parallel schedule $V_0, V_1, \ldots, V_\ell$ with $\ell < k$. The sets V_i are all antichains so they all have size less than n/k. Since they partition the vertices, it follows that the graph has less than $k \cdot n/k = n$ vertices, a contradiction. □

6 Switching networks

A collection of paths is *vertex disjoint* if all vertices in all paths in the collection are distinct. A *switching network* for N packets is a DAG with N sources and N sinks (and possibly some internal vertices) so that for every possible pairing of sources and sinks there exists a vertex-disjoint collection of paths from each source to the corresponding sink.

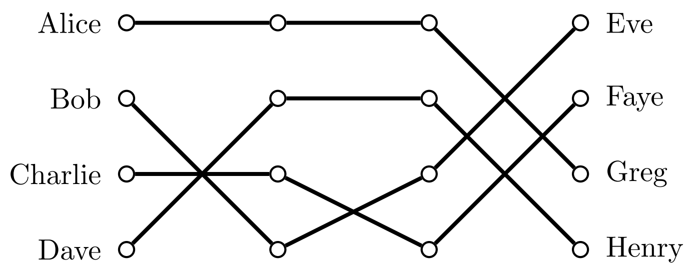
Here is a diagram of a switching network, which we'll call B_2, for 4 packets:



You can think of the sources and sinks as representing people living in Hong Kong and New York, resepectively, and of the edges as connecting wires. Now suppose each person in Hong Kong wants to connect up to their friend in New York according to this pairing:

$$\{\text{Alice}, \text{Greg}\}, \{\text{Bob}, \text{Eve}\}, \{\text{Charlie}, \text{Faye}\}, \{\text{Dave}, \text{Henry}\}$$

Your job is to route their calls along vertex-disjoint paths; we wouldn't want their lines to cross! Here is one choice of vertex-disjoint paths from every source to the corresponding sink:



Here is how I found these paths. The subgraph connecting the middle nodes splits into a "top component" and a "bottom component". Among Alice's and Charlie's calls, one must travel through the top component and the other one through the bottom component, for otherwise they would clash at a middle node. The same is true for Bob's and Dave's calls.

Now let's look at the calls that are supposed to reach Eve and Greg. These two must not clash either. Since Alice wants to talk to Greg and Bob wants to talk to Eve, Alice's and Bob's calls cannot clash in the middle. The same reasoning imposes a constraint on Charlie's and Dave's calls.

We can represent this information in a *constraint graph* whose vertices are callers and whose edges connect pairs that cannot be routed through the same middle component:

$$\begin{tikzcd}
\text{A} \arrow[r, dash] \arrow[d, dash] & \text{B} \arrow[d, dash] \\
\text{C} \arrow[r, dash] & \text{D}
\end{tikzcd}$$

The constraint graph is bipartite: The vertices can be partitioned into the sets $T = \{00, 11\}$ and $B = \{10, 01\}$. We route the packets in T through the top component and the packets in B through the bottom component. This uniquely specifies the first and last edge in each path (as in the above example). The middle edges are then easy to figure out.

How can we know for sure that B_2 is a switching network? We need to verify that all possible source-sink pairings admit edge-disjoint paths. It turns out that B_2 has a special, recursive structure that guarantees it. To make this structure more apparent, we will show how to construct, for every $n \geq 1$, a switching network B_n with 2^n sources and 2^n sinks, provided we have already constructed B_{n-1}. The digraph B_n is called the Beneš network (pronounced *Benesh*) after its inventor.

The "base" digraph B_1 looks like this:

$$\begin{tikzcd}
s_0 \arrow[r] \arrow[dr] & t_0 \\
s_1 \arrow[r] \arrow[ur] & t_1
\end{tikzcd}$$

Now we show how to construct the digraph B_{n+1}, assuming we have already constructed B_n. Recall that B_n has 2^n sources and 2^n sinks:

Step 1: Take *two* disjoint (no shared vertices or edges) copies of B_n, which we will call the top copy and the bottom copy.

Step 2: Introduce 2^{n+1} sources and 2^{n+1} sinks in B_{n+1}. Pair up the sources, and connect each pair with a unique source in the top and bottom copies of B_n. Do the same for the sinks.

The digraph B_2 was indeed obtained from B_1 according to this specification.

Theorem 12. *For every $n \geq 1$, the digraph B_n is a switching network.*

Proof. We prove the theorem by induction on n. In the base case $n = 1$, every source is connected to every sink, so there are vertex-disjoint paths for any pairing.

For the inductive step, we assume that B_n is a switching network and argue that so is B_{n+1}. Take any pairing Π of the sources and sinks in B_{n+1} (this represents the calls to be routed). We need to show that all pairs in Π can be connected by edge-disjoint paths.

We define an *undirected* "constraint" graph G as follows. The vertices of G are the sources of B_{n+1} (the callers in Hong Kong). The edges are a union of two (perfect) matchings. The first matching consists of those pairs of sources that were paired up when constructing B_{n+1} in step 2. The second matching is constructed like this. For those sinks that were paired up in step 2 of the construction of B_{n+1} we find the sources that were matched to them in Π (these are the callers that want to talk to them) and put an edge between them.

By Corollary 7, the graph G is bipartite. Let T, B be a partition of its vertices so there are no edges within T or within B. We now route every path out of a source according to its label in the partition (the paths out of sources in T go through the top component, while those in B go through the bottom component). A path arriving at a given sink is also routed according to the labels: If sink t is paired up with source s, we route the arriving path into t through the top component if s is in T and through the bottom one if s is in B. Now every pair in Π has been assigned consistently to the same middle component. By inductive hypothesis, both the top and bottom component are switching networks, so the relevant sources and sinks can be connected by vertex-disjoint paths. We have now connected all pairs in Π by vertex-disjoint paths, completing the inductive step. □

The vertex-disjoint paths in our example were obtained by following the procedure described in the proof. To check your understanding, it is a good idea to walk though the steps of the proof keeping this example in mind.

7 Expanders and superconcentrators*

The Beneš networks B_n is for $N = 2^n$ packets with $4n2^n - 2^{n+1} = \Theta(N \log N)$ edges (as was shown in Lecture 7). If we measure the "cost" of a switching network by the number of edges, this is a large improvement over the complete bipartite network in which each source-sink pair is connected for a total of N^2 edges when N grows large. Can we do even better and design a switching network for N packets with $o(N \log N)$ edges when N is large? It turns out that this is impossible. It is, however, possible to construct a related type of digraph that we describe next.

Definition 13. A *superconcentrator* for N packets is a digraph with N sources and N sinks in which for every set S of sources and every set T of sinks of the same size, which we call k, there exists a collection of k vertex-disjoint paths from the vertices in S to the vertices in T.

This sounds a lot like our definition of a switching network. What is the difference? A switching network can realize a collection of vertex-disjoint paths for *every possible* pairing of sources and sinks, while the superconcentrator merely guarantees the *existence* of a pairing for which vertex-disjoint paths exist. For example, if the sources are Alice, Bob, and Charlie and the sinks are Dave, Eve, and Faye, a superconcentrator guarantees that there exist vertex-disjoint paths from Alice and Bob to Dave and Faye, but not necessarily that Alice can be paired up with Dave and Bob with Faye. If we think of Alice, Bob, and Charlie as customers and Dave, Eve, and Faye as service representatives, then any k customers can get to talk to any k service representatives that happen to be at work, but they do not get to choose who talks to whom.

The distinction between switching networks and superconcentrators sounds technical. However, superconcentrators can have substantially fewer edges when the number of packets is large.

Theorem 14. *There exist a superconcentrator S_N for N packets with $O(N)$ edges for every N.*

The proof of Theorem 14 makes use of an important type of graph called an expander. We will call a bipartite graph with vertex partition (L, R) an *expander* if for every subset S of L of size at most $|L|/2$, the set of neighbors of S is at least as large as S, namely $|N(S)| \geq |S|$.[2]

Lemma 15. *For every $N > 64$ there exists an expander E_N with $|L| = N$, $|R| = \lfloor 3N/4 \rfloor$ in which all vertices in L have degree (at most) 32.*

In particular, the expander E_N has (at most) $32N$ edges. The proof of Lemma 15 is not too difficult but is beyond the scope of this course. It turns out that if every vertex in L is connected with 32 vertices in R independently at random the resulting graph is an expander with good probability.

Armed with Lemma 15 we can now prove Theorem 14. If $N \leq 64$ then we can take the superconcentrator S_N to be the Beneš network B_6, discarding some sources and sinks if necessary. This is clearly a superconcentrator.

If $N > 64$, we construct the digraphs S_N recursively as follows:

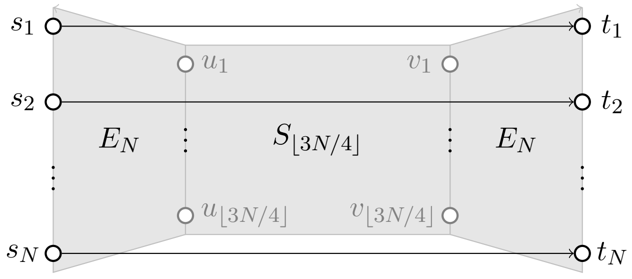
1. Take N sources $s_1, \dots, s_N$, N sinks $t_1, \dots, t_N$, and two sets of internal vertices $u_1, \dots, u_{\lfloor 3N/4 \rfloor}$ and $v_1, \dots, v_{\lfloor 3N/4 \rfloor}$.

2. Add edges between the sets $L = \{s_1, \dots, s_n\}$ and $R = \{u_1, \dots, u_{\lfloor 3N/4 \rfloor}\}$ as in the expander E_N from Lemma 15. Direct all edges from L to R.

3. Add edges between the sets $L' = \{t_1, \dots, t_n\}$ and $R' = \{v_1, \dots, v_{\lfloor 3N/4 \rfloor}\}$ as in the expander E_N from Lemma 15. Direct all edges from R' to L'.

4. Add an arbitrary perfect matching between L and L'.

5. Include a copy of $S_{\lfloor 3N/4 \rfloor}$, identifying its sources and sinks with the vertices in R and R', respectively.


s_1
t_1
u_1
v_1
s_2
t_2
E_N
$S_{\lfloor 3N/4 \rfloor}$
E_N
$u_{\lfloor 3N/4 \rfloor}$
$v_{\lfloor 3N/4 \rfloor}$
s_N
t_N

The next claim says that the construction is indeed correct:

Claim 16. *For every N, the graph S_N is a superconcentrator.*

Proof. The proof is by strong induction on N. For the base case(s) $N \leq 64$, S_N is a switching network, so it is in particular a superconcentrator. For the inductive step, we will assume that $S_{\lfloor 3N/4 \rfloor}$ is a superconcentrator and prove that S_N must also be one.

The proof is by cases depending on the size k of the sets S and T in the definition of superconcentrators. Let us first consider the case $k \leq N/2$. By the definition of expanders, every subset S' of S has at least $|S'|$ neighbors in R. By Hall's theorem, there exists a matching Ξ that matches the vertices in S to some subset U consisting of k vertices in R using edges from the first copy of E_N. For the same reason, the vertices in T can be matched to some subset V consisting of k vertices in R' in the second copy of E_N via a matching Ξ'. Because $S_{\lfloor 3N/4 \rfloor}$ is a superconcentrator there must exist a collection Π of k vertex-disjoint paths from U to V in $S_{\lfloor 3N/4 \rfloor}$. The vertex-disjoint paths from S to T can then be obtained by taking the union of Ξ, Π, and Ξ'. This concludes the proof for the case $k \leq N/2$.

[2]The actual definition is in fact more general and involves several parameters; if you are interested see the reference.

If $k > N/2$, there must exist at least $2k - N$ indices i for which both s_i is in S and t_i is in T. (This can be proved easily using the inclusion-exclusion principle, which we will introduce in Lecture 10.) The matching from step 5 in the construction of S_N gives disjoint paths of length 1 between these $2k - N$ sources and sinks. After discarding these from the digraph S_N, there remain at most $k - (2k - N) = N - k$ sources and sinks to match. This number is at most $N/2$, so by the analysis of the case $k \leq N/2$ there exist a vertex-disjoint collection of paths in S_N among the remaining sources and sinks. This concludes the proof for the case $k > N/2$ and the inductive step. $\square$

To finish the proof of Theorem 14 it remains to upper bound the number of edges $E(N)$ in the digraph S_N. By construction, when $N \geq 64$ the number of edges in S_N equals the number of edges in $S_{\lfloor 3N/4 \rfloor}$, plusthe number of edges in each of the two copies of E_N ($32N$ each), plus N edges for the matching in step 5. Adding the contribution of all these terms, we obtain the recurrence

$$E(N) = E(\lfloor 3N/4 \rfloor) + 65N \qquad \text{when } N > 64$$

with $E(N) \leq 3328$ for $N \leq 64$. We can therefore bound $E(N)$ by

$$E(N) \leq (65 \cdot N + 65 \cdot (3/4)N + 65 \cdot (3/4)^2 N + \cdots) + 3328 = 260N + 3328$$

for all N.

References

This lecture is based on Chapters 9, 10, and 11 of the text *Mathematics for Computer Science* by E. Lehman, T. Leighton, and A. Meyer. Section 7 is based on Chapter 1 of the survey *Expander graphs and their applications* by Hoory, Linial, and Wigderson.