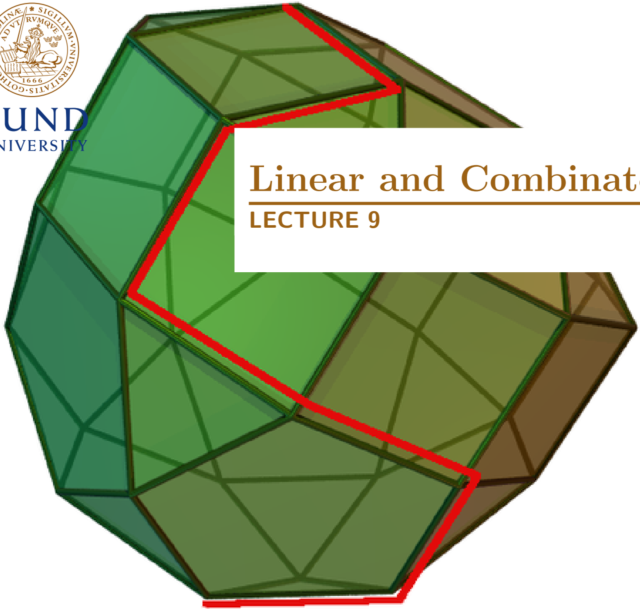


Linear and Combinatorial Optimization 2020

LECTURE 9

Overview

1 Graphs and networks

2 The maximal flow problem

3 The shortest route problem

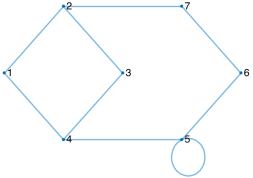
4 The Vigenère cipher – Theory for Lab 2

Graphs and networks

- A graph consists of

  nodes that are points (in the plane)

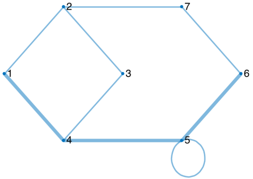
  edges that connect the nodes.
- A *loop* is an edge that connects a node to itself.



A graph with 7 nodes and 9 edges. Note that there is a loop at node number 5.

Graphs and networks

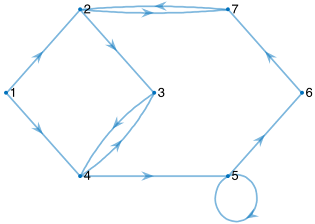
- A *path* is an ordered set of edges joining one node to another.
- A graph is said to be *connected* if there is a path joining any two nodes of the graph.
- A *cycle* is a path joining a a node to itself (possibly passing some other nodes



The path 1 → 4 → 5 → 6, joining node 1 to node 6 is highlighted.

Graphs and networks

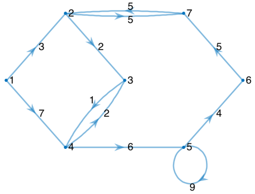
- A *directed graph* or a *digraph*, is a graph where each edge is equipped with a direction (indicated with an arrow).



A digraph

Graphs and networks

- A *network* is a connected, directed graph where each edge has been assigned a non-negative number – a *capacity*.
- The *capacity* represents the maximum amount that may pass along an edge (e.g. water in a pipe, data traffic, etc.). The capacity of an edge from node i to node j is represented by a number c_{ij}, $i, j = 1, \ldots, n$.



A network

Graphs and networks

Definition

A flow in a network can be represented by x_{ij} *such that* $0 \leq x_{ij} \leq c_{ij}$*, where*

- *there is a special note, called the* source S*, such that* $\sum_{i=1}^{n} x_{i1} = 0$*, where we have assumed that the source node has number* 1*. This means that all flow comes from* S*.*
- *there is a special node, called the* sink T*, such that* $\sum_{j=1}^{n} x_{nj} = 0$*, where we have assumed that the sink node has number* n*. This means that all flow goes to* T*.*
- *For all other nodes* $(k = 2, \ldots, n-1)$*,* $\sum_{i=1}^{n} x_{ik} = \sum_{j=1}^{n} x_{kj}$*, i.e. inflow = outflow.*

The maximal flow problem

- In the maximal flow problem, we maximize the value of the flow which is the sum of all x_{1j}, (or equivalently, the sum of all x_{in}). The problem can be formulated as the LP problem

$$
\begin{array}{ll}
\text{maximize} & \sum_{k=1}^{n} x_{1k}, \\
\text{subject to} & \begin{cases} \sum_{i=1}^{n} x_{i1} = 0, \\ \sum_{j=1}^{n} x_{nj} = 0, \\ \sum_{i=1}^{n} x_{ik} = \sum_{j=1}^{n} x_{kj}, \qquad k = 2, \ldots, n-1, \\ \quad 0 \leq x_{ij} \leq c_{ij}, \qquad i, j = 1, \ldots, n. \end{cases}
\end{array}
$$

- The problem can be solved with the simplex method, but there are also other, more efficient, special algorithms.
- We will learn one of these algorithms, the *Ford–Fulkersson algorithm*.

The maximal flow problem

Definition

Given a flow x_{ij} in a network,

- *the net flow from node i to node j is given by $\widehat{x}_{ij} = x_{ij} - x_{ji}$, and*
- *the excess capacities are the numbers*

$$d_{ij} = c_{ij} - x_{ij} + x_{ij} = c_{ij} - \widehat{x}_{ij}.$$

Note that $\widehat{x}_{ij} = -\widehat{x}_{ji}$.

Example

If $c_{ij} = 4$, $c_{ij} = 0$ and $\widehat{x}_{ij} = 1$, then we have $d_{ij} = 3$ and $d_{ji} = 1$. The meaning of $c_{ji} = 0$ is that no flow can go from node j to node i.

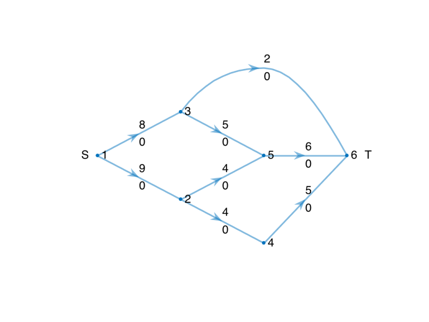
The Ford–Fulkersson algorithm

1. Start with a flow (for example $x_{ij} = 0$ for $i, j = 1, \ldots, n$), and compute all the excess capacities. Then we start labelling the nodes as follows, starting from the source S. Start by letting $N_1 = \{1\}$. For each node that can be reached from node 1 by an edge with positive excess capacity, do the following:
    - Label these nodes by (e_k, p_k), where $e_k = d_{1k}$ (the excess capacity from node $1 = S$ to node k) and $p_k = 1$ (the number of the node that led to node k).
    - Replace N_1 by $N_1 \cup \{k\}$.

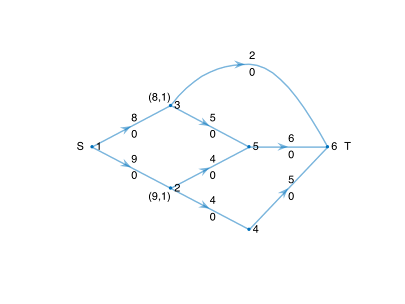
The Ford–Fulkersson algorithm

Example

We assume that we have the following network with capacities c_{ij}. Starting with the flow x_{ij}, the excess capacities are equal to the capacities:



Original network with all $d_{ij} = c_{ij}$



The first step in the algorithm completed. The nodes 2 and 3 are labelled and $N_1 = \{1, 2, 3\}$.

The Ford–Fulkersson algorithm

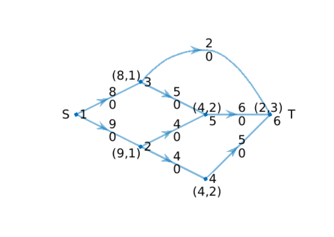
2. For each node in N_1, we label the nodes $k \notin N_1$ that can be reached from a node m such that $m \in N_1$ and $d_{mk} > 0$ as follows:
   - Label node k by (e_k, p_k), where $e_k = \min(d_{mk}, e_m)$ (the minimum of the excess capacities of the edges in the path from S to node k via node m) and $p_k = m$ (the node that led to node k).
   - Replace N_1 by $N_1 \cup \{k\}$.
3. Continue to add nodes as long as possible. When no more nodes can be added, we are in one of the following situations:
   a) The sink is unlabelled, or
   b) The sink is labelled.

Let us do these steps for our example before proceeding with the general algorithm.

Ford–Fulkersson algorithm

Example (Cont.)

- *From step 1 we have $N_1 = \{1, 2, 3\}$, and from N_1 we can reach all the remaining nodes (Node 4 and 5 can be reached from node 2 and node 6 can be reached from node 3.) We get new labels as in the figure below, and $N_1 = \{1, 2, 3, 4, 5, 6\}$, i.e. we are in case b).*



All the nodes have been labelled.

Ford–Fulkersson algorithm

Now we continue with the general algorithm.

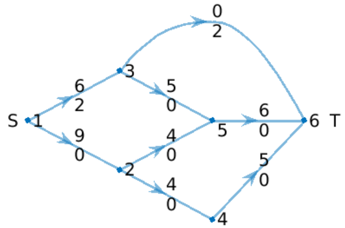
3. (Cont.) If case a) holds, then the flow is optimal, as we will prove soon. If case b) holds, then there is a path from source to sink with excess capacity e_n (where n is the number of the sink). We can find the path using p_n and backtracking through the whole path. Add e_n to all the x_{ij} in this path and compute new excess capacities using this new flow. Start again from 1.

Example (Cont.)

- *In our example,* $n = 6$*, and* $e_6 = 2$ *which shows that there is a path from* S *to* T *whose edges all have excess capacity* ≥ 2*. As* $p_6 = 3$ *and* $p_3 = 1$*, we find that the path is* $1 \to 3 \to 6$*.*
- *Adding 2 to all the edges of that path, we obtain the network with excess capacities as on the next slide.*

The Ford–Fulkersson algorithm

Example (Cont.)

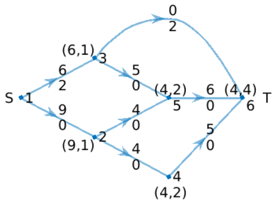


The network with updated excess capacities

- *Note that the excess capacities corresponding to the edges which has the same direction as the arrow are written to the left of (or above) the arrow, and the excess capacities corresponding to edges in the opposite direction is written to the right of (or below) the arrow.*
- *We'll start labelling the nodes on the next slide.*

The Ford–Fulkersson algorithm

Example (Cont.)

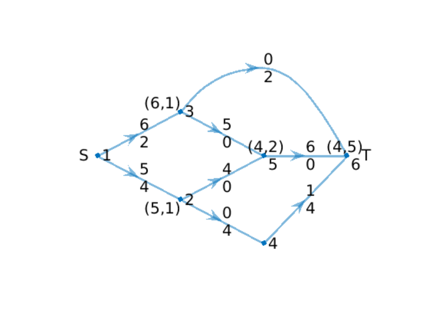


New labels are put on the nodes

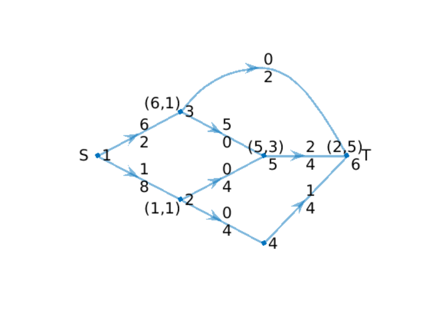
- *The nodes 2 and 3 are labelled first, and then 4 and 5 (which both can be reached from node 2) and then 6 (which can be reached from node 4).*
- *We are in case b), and a path from S to T that we have found is $1 \to 2 \to 4 \to 6$.*
- *Since $e_6 = 4$, the flow can be increased by 4 along that path, and new excess capacities can be computed. We do this in the next slide as well as labelling the nodes.*

The Ford–Fulkersson algorithm

Example (Cont.)



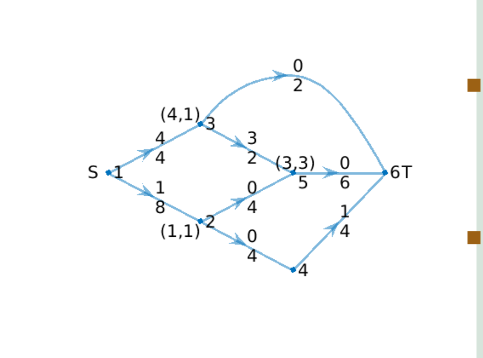
We have found the path $1 \to 2 \to 5 \to 6$ *with* $e_6 = 4$*. Increase the flow along that path by* 4 *and compute new excess capacities:*



We have found the path $1 \to 3 \to 5 \to 6$ *with* $e_6 = 2$*. Increase the flow along that path by* 2 *and compute new excess capacities:*

The Ford–Fulkersson algorithm

Example (Cont.)



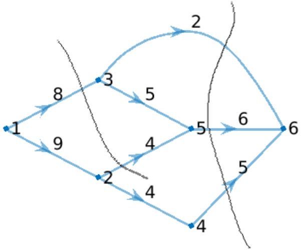
We did not manage to label the T node, so we are in case a).

- *Since we cannot find a path from S to T, we suspect that we have found the maximal flow with value* $2 + 4 + 4 + 2 = 12$.
- *That the flow is indeed optimal follows from the Max flow–Min cut theorem, which we will see after some definitions that will be needed for stating the theorem.*

The max flow–min cut theorem

Definition

- *A cut in a network is a set of directed edges such that every path from source to sink contains at least one edge from the set.*
- *The capacity of a cut is the sum of the capacities of the edges in that cut.*



$\{3 \to 6, 4 \to 6, 5 \to 6\}$ is a cut with capacity $2 + 6 + 5 = 13$, but $\{1 \to 3, 2 \to 5\}$ is not a cut.

The max flow–min cut theorem

Theorem (Max flow–min cut)

The maximum value of a flow in a network is the minimum of the capacities of all cuts in the network.

Proof 1 (Sketch).

Write the problem as an LP problem and check that the min cut problem is the dual of the max flow problem. The result follows from the strong duality theorem. □

The max flow–min cut theorem

Direct proof.

1. The value of a flow in a network is always $\leq$ than the capacity of any cut.
   - For a given path, the flow along this path cannot exceed the capacity of an arc of the path that belongs to the cut.
   - As a flow is the sum of such paths from S to T, its value cannot exceed the capacity of the cut.
2. In particular, the value of the maximal flow is less than the value of the minimum cut.
3. We will show that they are in fact equal, by constructing a flow and a cut with whose value/capacity is equal. This will be done by using the set N_1 in the final step of the Ford–Fulkersson algorithm.

The max flow–min cut theorem

Proof (Cont.)

4. Suppose that we found a flow such that the Ford–Fulkersson algorithm terminates without the sink being labelled. As in the algorithm, we have

$$N_1 = \{S\} \cup \{\text{labelled nodes}\}, N_2 = \{\text{unlabelled nodes}\}.$$

Note that $T \in N_2$. Let

$$A = \{\text{edges from } N_1 \text{ to } N_2\}.$$

Then A is a cut, since $S \in N_1$ and $T \in N_2$, and so it is clear that any path from S to T has to pass A.

The max flow–min cut theorem

Proof (Cont.)

5. Now we will show that the capacity of A is equal to the value of the flow. We have

$$\textstyle\sum_{i=1}^{n} x_{ik} = \sum_{j=1}^{n} x_{kj}, \qquad j = 2, \ldots, n-1.$$

Since $d_{ij} = c_{ij} - x_{ij} + x_{ji}$, it follows that

$$\textstyle\sum_{j=1}^{n} d_{ij} = \sum_{j=1}^{n} c_{ij} \underbrace{- \sum_{j=1}^{n} x_{ij} + \sum_{j=1}^{n} x_{ji}}_{=0 \text{ for } i=2,\ldots,n-1}.$$

For the source node ($i = 1$), we have

$$\textstyle\sum_{j=2}^{n} (c_{1j} - d_{1j}) = \sum_{j=2}^{n} x_{1j},$$

since $x_{j1} = 0$ for the source.

The max flow–min cut theorem

Proof (Cont.)

6. Summing the two last equalities for $i \in N_1$, we obtain

$$\sum_{i \in N_1} \sum_{j=1}^{n} (c_{ij} - d_{ij}) = \sum_{j=2}^{n} x_{1j},$$

which is the value of the flow. If both i and j belong to N_1, then $c_{ij} - d_{ij} = \widehat{x}_{ij}$ and $c_{ji} - d_{ji} = \widehat{x}_{ji}$ cancel out, while if $i \in N_1$ and $j \in N_2$, then $d_{ij} = 0$, and so the flow can be expressed as

$$\sum_{j=2}^{n} x_{1j} = \sum_{i \in N_1} \sum_{j \in N_2} (c_{ij} - d_{ij}) = \sum_{i \in N_1} \sum_{j \in N_2} c_{ij},$$

i.e. the capacity of the cut A.

Proof (Cont.)

Proof (Cont.)

7. We have proved that our flow is equal to the capacity of our cut. Since all flows have value which is less than or equal to the capacity of any cut, this shows that these are the max flow and min cut, respectively, and that they are equal.

□

The shortest route problem

The problem.

Given a graph or a directed graph with distances on each edge, what is the shortest distance between two particular nodes?

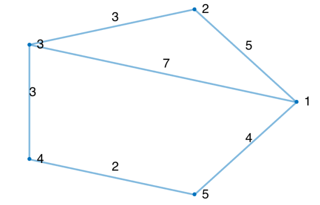
We fix the starting node, which we will call the origin. We give the idea for a simple algorithm, in which we will need the following procedure, depending on a parameter a (the maximum allowed distance):

- Form the set N_r consisting of the origin and all nodes that can be reached from the origin by a shortest route of length at most a, and let the set N_u consist of all other nodes.

Initially (when $a = 0$), N_r consists only of the origin. Then, as we increase a, we transfer nodes from N_u to N_r until $N_u = \emptyset$ or the destination node belongs to N_r.

The shortest route problem

Example



What is the shortest distance between node 1 and node 4?

- *We choose node 1 as the origin, and let $a = 0$. Initially, $N_r = \{1\}$ and $N_u = \{2, 3, 4, 5\}$.*
- *a is increased to 4. Then $N_r = \{1, 5\}$ and $N_u = \{2, 3, 4\}$.*
- *a is increased to 5. Then $N_r = \{1, 2, 5\}$ and $N_u = \{3, 4\}$.*
- *a is increased to 6. Then $N_r = \{1, 2, 4, 5\}$ and $N_u = \{3\}$.*

The shortest route problem

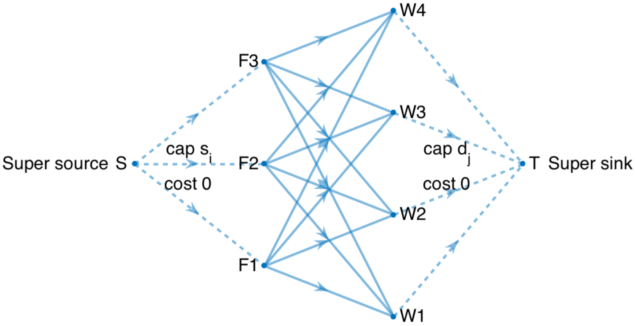
Example (Cont.)

- *We have reached the destination node* 4*, and the distance from the origin is* 6*.*

- Both the shortest route problem and the transportation problem can be seen as special cases of a *minimal cost flow problem:*
- Minimize the cost from source to sink of a fixed total amount of transported goods, when in addition to the distances between nodes, there are also maximum capacities that each edge can take.

The shortest route problem

The figure below shows the resulting graph when the transportation problem is viewed as a minimal cost flow problem.



The Vigenère cipher

- In Lab session 2, one of the topics will be the Vigenère cipher. Here is some background information.
- The Vigenère cipher is a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers based on the letters of a keyword. I will explain how it works, but first a bit of history.
- The Vigenère cipher was originally described by Giovan Battista Bellaso in 1553, but the scheme was later misattributed to Blaise de Vigenère (another 16th century cryptographer) in the 19th century, and thus acquired its present name.

Blaise de Vigenère, 1523 – 1596

The Vigenère cipher

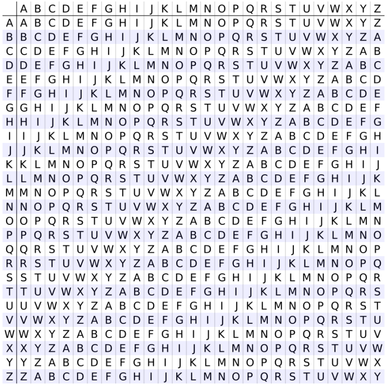
- Though the cipher is easy to understand and implement, for three centuries it resisted all attempts to break it, and it was considered unbreakable.
- Many people have tried to implement encryption schemes that are essentially Vigenère ciphers.
- Friedrich Kasiski was the first to publish a general method of deciphering a Vigenère cipher in 1863, although versions of it had been solved before that.

The Vigenère cipher

- The key is a short string, for example 'HELLO'.
- At encryption and decryption, the key is written periodically below the text. Each letter in the key gives the number of (cyclic) shifts for the above letter in the alphabet:

$$a \to b \to c \to \cdots \to y \to z \to a \dots$$

- A means no shift, B one step, C two steps, etc.
- A Vigenère table (see figure) can be used for doing this more easily.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

A Vigenère table or a Tabula recta can be of use for encryption/decryption when the key is known

The Vigenère cipher

Example

Alphabet:	abcdefghijklmnopqrstuvwxyz
Original text:	thistextshouldbeencrypted
Secret key:	HELLOHELLOHELLOHELLOHELLO
Encrypted text:	altdhlbedvvywopliynfftepr

- When sending a message, only the submitter and the receiver both have access to the secret key beforehand, while the message is accessible to everybody.
- Now, imagine that a third party wants to get access to the secret message. They then have to find the secret keyword (assuming that they know that a Vigenère cipher has been used), since they already have access to the encrypted message.

The Vigenère cipher

- Let us put ourselves in the shoes of the third party that wants to solve the cipher.
- The first thing to do is to try to find the length of the key word. There are methods for this, e.g. *Kasiski examination* or the *Friedman test (kappa test)*. We will not cover this, since it has nothing to do with optimization, but you can google for information if you are interested.
- When the length of the key word has been found (or guessed), we can use optimization and frequency analysis to find the key word.

The Vigenère cipher

- We assume that the length of the key word is n. We will now try to find the key word. Note that the number of possible keywords with an alphabet of length 26 is 26^n. This is the number of feasible points in the optimization problem.
- If the first letter of the key word is a, then letter number $1 + nk$ for every $k = 0, 1, \dots$ is the same in the encrypted text as in the original text.
- If the first letter of the key word is b, then letter number $1 + nk$ for every $k = 0, 1, \dots$ is shifted one step, etc. (Similar relationships for all letters in the alphabet and all letters in the keyword.)
- This gives a relationship between the probabilities of letter number i in the key word and products of probabilities of certain letters in the unencrypted text.

The Vigenère cipher

Example

For our example with a 25 letter long text and with a password of length 5, we can compute a probability for each feasible keyword, like for example

$$P(key = \texttt{'abcde'}) = P(text = \texttt{'akradlacarvxulllhwkbfscmn'})$$

etc. Note that there are 26^5 *possible keywords since this alphabet has 26 letters, and we don't want to check all of these. We use frequency analysis on the encrypted text to estimate probabilities like the ones on the right hand side.*

The Vigenère cipher

Example (Cont.)

By using trigram statistics which are frequences of three consecutive letters in a text (of a given language), we estimate the probability of the right hand side according to:

$$P(text = \texttt{'akradlacarvxulllhwkbfscmn'})$$

Let us first estimate the probability that the original text starts with the three letters 'akr'*:*

$$P(text = \texttt{'akr...'}) \approx \textit{frequency of the trigram } \texttt{'akr'} \textit{ in English text}$$

We denote the frequency of the trigram 'akr' *by* $T(\texttt{'akr'})$, *etc.*

The Vigenère cipher

Example (Cont.)

Next, we estimate the probability that the first four letters of the original text are 'akra'*:*

$$
\begin{aligned}
P(\mathit{text} = \texttt{'akra...'}) \approx{}& P(\mathit{text} = \texttt{'akr...'})\cdot \\
&\cdot P(\mathit{text}(4) = \texttt{'a'} \,|\, \mathit{text}(2) = \texttt{'k'} \text{ and } \mathit{text}(3) = \texttt{'r'}) \\
&\approx T(\texttt{'akr'}) \cdot \frac{T(\texttt{'kra'})}{P(\mathit{text}(2) = \texttt{'k'} \text{ and } \mathit{text}(3) = \texttt{'r'})} \\
&\approx T(\texttt{'akr'}) \cdot \frac{T(\texttt{'kra'})}{B(\texttt{'kr'})},
\end{aligned}
$$

using conditional probability and only considering letters one or two steps back. In the last row, we have used the notation $B(\texttt{'kr'})$ *for the frequency of the bigram* 'kr' *in English text.*

The Vigenère cipher

Example (Cont.)

In the same way, we can compute the probability that the text starts with 'akrad' *by computing*

$$\begin{aligned} P(text = \texttt{'akrad...'}) &\approx P(text = \texttt{'akra...'})\cdot \\ &\cdot P(text(5) = \texttt{'d'} \mid text(3) = \texttt{'r'} \textit{ and } text(4) = \texttt{'a'}) \\ &\approx \frac{T(\texttt{'akr'}) \cdot T(\texttt{'kra'})}{B(\texttt{'kr'})} \cdot \frac{T(\texttt{'rad'})}{B(\texttt{'ra'})} \\ &= \frac{T(\texttt{'akr'})T(\texttt{'kra'})T(\texttt{'rad'})}{B(\texttt{'kr'})B(\texttt{'ra'})} \end{aligned}$$

We can continue and get a probability for the whole text. We will get the product of all the trigram frequencies of the (candidate for the) original text in the numerator and the product of all the bigrams except the first one in the denominator.

The Vigenère cipher

- We will work with logarithms of the probabilities rather than the probabilities themselves, and so we take logarithms of both sides of the final formula. The multiplication/division turns into addition/subtraction. This gives us the objective function of the *maximization* problem.
- The reason for using logarithms is that it is computationally easier to add than to multiply and also because the probabilities involved will be very small, and so if many such probabilities were multiplied together, the computational error would be too large.
- The *logarithms of the* trigram and digram frequencies are stored in a table that the Matlab functions used in the lab have access to when computing the value of the objective function.