EFOP-3.4.3-16-2016-00014

Business Informatics

Handout

Prepared by Tamás Vinkó

Methodological expert: Edit Gyáfrás

This teaching material has been made at the University of Szeged, and supported by the European Union.
Project identity number: EFOP-3.4.3-16-2016-00014

2019

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Contents

Course information

Course title: Business Informatics

Course code: 23C120

Credit: 5

Type: lecture and seminar

Contact hours / week: 2+1

Evaluation: five-grade

Semester: 3rd

Prerequisites: -

Learning Outcomes

a) regarding knowledge, the student

- is familiar with the up-to-date, theoretically sound, mathematics-statistics based and econometric modelling methods (along with their limitations) of recognizing, expressing and solving problems along with data collection and processing.
- Knows and utilizes the decision theories and analyzing methods of economics, international economics and world economics.

b) regarding competencies, the student

- can make independent and new deductions, formulate original thoughts and solution methods, utilize sophisticated analytical and modeling methods. The student is capable of formulating solution strategies for complex problems and decisions within the organizational culture both in a domestic and an international setting;
- after obtaining the necessary practical skills and experience, the student is capable of leading medium sized and major organizational units and performing a general economic function within an economic organization. The student is also capable of planning and directing economic processes along with managing resources;

c) regarding attitude, the student

- is open to new results and achievements of economic research and practical experiments;

d) regarding autonomy and responsibility, the student

- is involved in research and developmental projects; in project groups the student works for the goal of the team in an autonomous but cooperative way actively utilizing his/her practical and theoretical knowledge.

Requirements

In order to meet the objectives of the course, attendance of the lectures is highly recommended, hence, the attendance is registered. In-class performance is assessed and its results form part of the end-term grade. There will be home assignments. In the exam period, in previously defined dates and time, a written exam has to be passed.

The final score (%) of the course is constructed as follows:

- 10% lecture + seminar attendance + lecture/seminar activities
- 40% home assignments
- 10% presentation
- 40% exam

Home assignments: essays on selected topics, 3 times during the semester (worth 10, 15 and 15 points, respectively)

Presentation: 15-20 minutes well-prepared and detailed presentation on a selected topic.

Grades (based on percentages)

80-100: 5 (excellent)

70-79: 4 (good)

60-69: 3 (medium)

50-59: 2 (satisfactory)

0-49: 1 (fail)

As for students with individual schedules, they are obliged to elaborate a project in detail. The project should be consulted with the lecturer of the course.

Course topics

Business Informatics integrates core elements from the disciplines of business administration, information systems and computer science into one field. The aim of the course is to provide insights into some relevant mathematical modeling and algorithmic concepts and to show some recent applications for real-world problems.

Recommended reading

- Chvátal, V. (1983): Linear Programming. Freeman, New York. ISBN-10: 0716715872
- D. Jungnickel (2004): Graphs networks and algorithms. Algorithms and computation in mathematics. Springer, ISBN-10: 3540219056
- Ralph M. Stair, George W. Reynolds (2010): Principles of Information Systems - A Managerial Approach. Course Technology, Cengage Learning, ISBN-13:9780-324665284
- Albert-László Barabási (2016): Network Science, Cambridge University Press, ISBN-13: 978-1107076266.
- Riccardo Mazza (2009), Introduction to Information Visualization, Springer, ISBN: 978-1-84800-218-0

Chapter 1

Graphs and Networks

Learning outcome of the topic *The basic definitions of the simple yet powerful modeling technique of graphs are studied. The important graph classes are discussed together with some type of real-world networks. Through these examples the students will learn how to use this wonderful mathematical model for capturing certain characteristics of real-world phenomena.*

1.1 Definitions

In the most common sense of the term, a *graph* is an ordered pair

$$G = (V, E)$$

comprising

- a set V of *vertices* or *nodes*
- together with a set E of *edges* or *links*, which are 2-element subsets of V

This type of graph may be described precisely as *undirected*.

The vertices belonging to an edge are called the *ends*, *endpoints*, or *end vertices* of the edge. Two nodes, $u \in V$ and $v \in V$ are *adjacent* if $(u, v) \in E$. A vertex may exist in a graph and not belong to an edge.

The *order* of a graph is $|V|$ (the number of vertices). A *graph's size* is $|E|$, the number of edges. The *degree of a vertex* is the number of edges that connect to it, where an edge that connects to the vertex at both ends (a loop) is counted twice.

An *undirected* graph is one in which edges have no orientation. The edge (u, v) is identical to the edge (v, u), i.e., they are not ordered pairs.

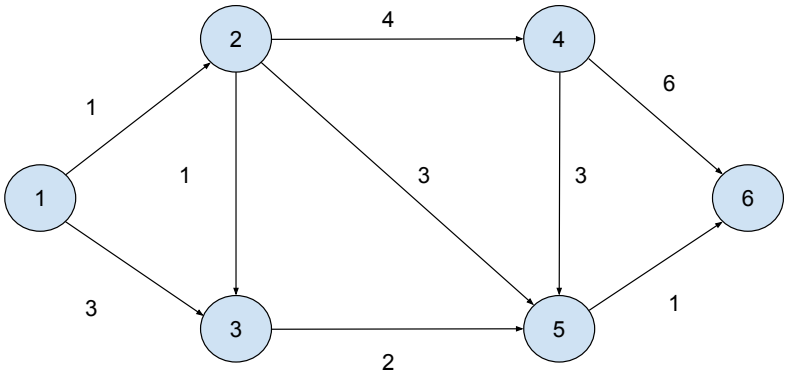
A *directed* graph or *digraph* is an ordered pair $D = (V, A)$ with

- V a set whose elements are called vertices or nodes, and
- A a set of ordered pairs of vertices, called *arcs, directed edges*, or sometimes *arrows*.

An arc $a = (u, v)$ is considered to be directed from u to v; v is called the *head* and u is called the *tail* of the arc.

A graph is a *weighted graph* if a number (weight) is assigned to each edge. Such weights might represent, for example, costs, lengths or capacities, etc. depending on the problem at hand. Some authors call such a graph a *network*.

As an example here is a directed, weighted graph of 6 nodes:



Note that the actual *drawing* of a graph is rather arbitrary.

1.1.1 Important classes

A regular graph is a graph where each vertex has the same number of neighbors, i.e., every vertex has the same degree.
A regular graph with vertices of degree k is called a k-regular graph or regular graph of degree k.

Complete graphs have the feature that each pair of vertices has an edge connecting them. That is, the graph contains all possible edges.

In a bipartite graph the vertex set can be partitioned into two sets, W and X, so that no two vertices in W are adjacent and no two vertices in X are adjacent.

A planar graph is a graph whose vertices and edges can be drawn in a plane such that no two of the edges intersect.

A tree is a connected graph with no cycles.

A forest is a graph with no cycles (i.e. the disjoint union of one or more trees).

Exercise: Give examples for all the definitions above.

1.1.2 Networks

Networks are graphs, basically, so the definition is the same as above. However, we use the term network in case the graph is representing some real world phenomena or procedure or anything. Moreover, the questions one usually asks in networks are different from the questions raised in graph theory.

Usually 4 types of networks are distinguished [8]:

Technological networks: Internet routers, road/train routes, power grid, airplane routes.

Information networks: World-Wide-Web, citation, trade

Social networks: Facebook, LinkedIn, friendship (in general)

Biological networks: food-chain (non-relevant for this course, just as a nice example).

Exercise: in the above examples try to find out what are the nodes and the edges for the different graphs/networks.

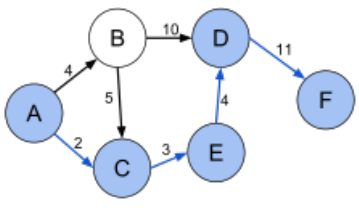
1.2 Graph Algorithms

In the following we give a brief overview of the most important (and easy-to-understand) graph algorithms.

1.2.1 Shortest path

The shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. If the graph is non-weighted, then what is to be minimized: the number of edges.

As an illustration consider the following graph, in which we indicate the shortest path between the nodes A and F.



Dijkstra algorithm What follows is we give an algorithm to find the shortest path [4].

Let us call the node at which we are starting the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.

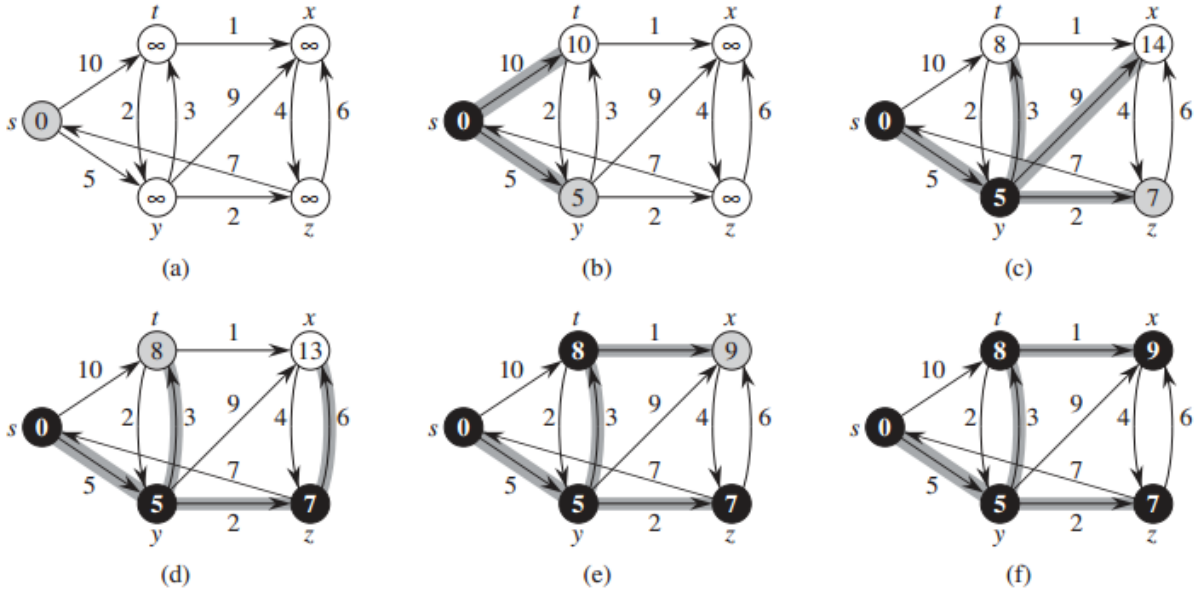
    - For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then STOP. The algorithm has finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Example The following example is taken from [2, 3]



The execution is as follows:

- (a) Initialize the distance of $s = 0$, other distances $= \infty$.
- (b) Choose minimum-distance unknown vertex, s (s now becomes known). Update distances of s's unknown neighbors (t, y).
- (c) Choose minimum-distance unknown vertex, y (y now becomes known). Update distances of y's unknown neighbors (t, x, z).
- (d) Choose minimum-distance unknown vertex, z (z now becomes known). Update distances of z's unknown neighbors (x).
- (e) Choose minimum-distance unknown vertex, t (t now becomes known). Update distances of t's unknown neighbors (x).
- (f) Choose minimum-distance unknown vertex, x (x now becomes known). No distances to update. Finished.

Related problem: traveling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Interestingly, while the Shortest Path problem is easy to solve (as we have seen, greedy algorithm works and gives the optimal solution), the Traveling Salesman problem is difficult[1].

A very nice demo can be seen and played with at

`http://examples.gurobi.com/traveling-salesman-problem/`

1.2.2 Minimum Spanning Tree

Given a connected, undirected graph, a *spanning tree* of that graph is a subgraph that is

- a tree
- and connects all the vertices together.

A single graph can have many different spanning trees. We can also assign a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree.

A *minimum spanning tree* (MST) is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

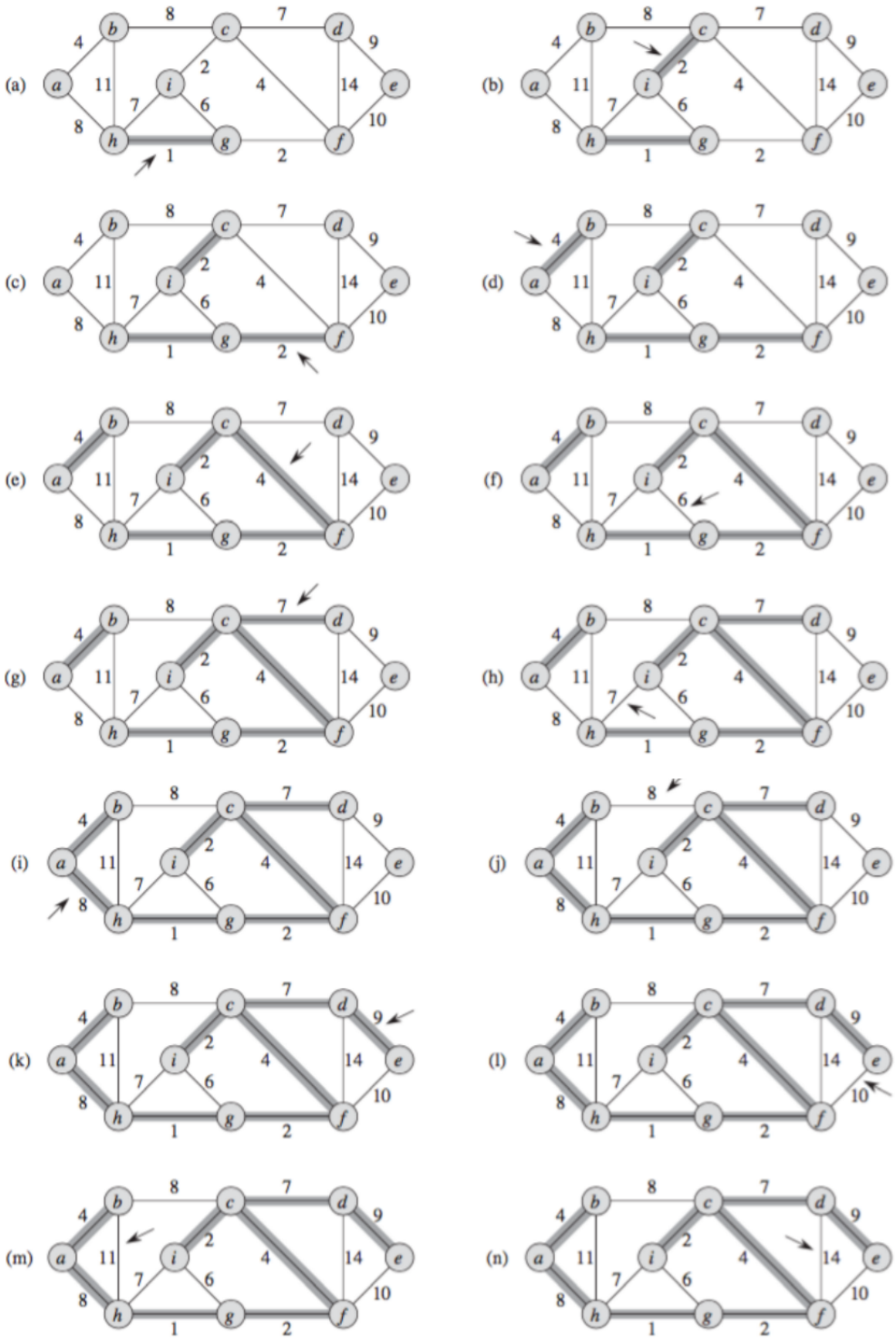
Kruskal's algorithm The following algorithm determines the minimum spanning tree of a graph [6].

1. create a forest F (a set of trees), where each vertex in the graph is a separate tree
2. create a set S containing all the edges in the graph
3. while S is nonempty and F is not yet spanning do the followings
    - remove an edge with minimum weight from S
    - if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree

As we can see, this is also a greedy-type algorithm.

[1] this means, superficially, that no better algorithm than brute force can be made to solve a general TSP problem

Example The following example is taken from [2]



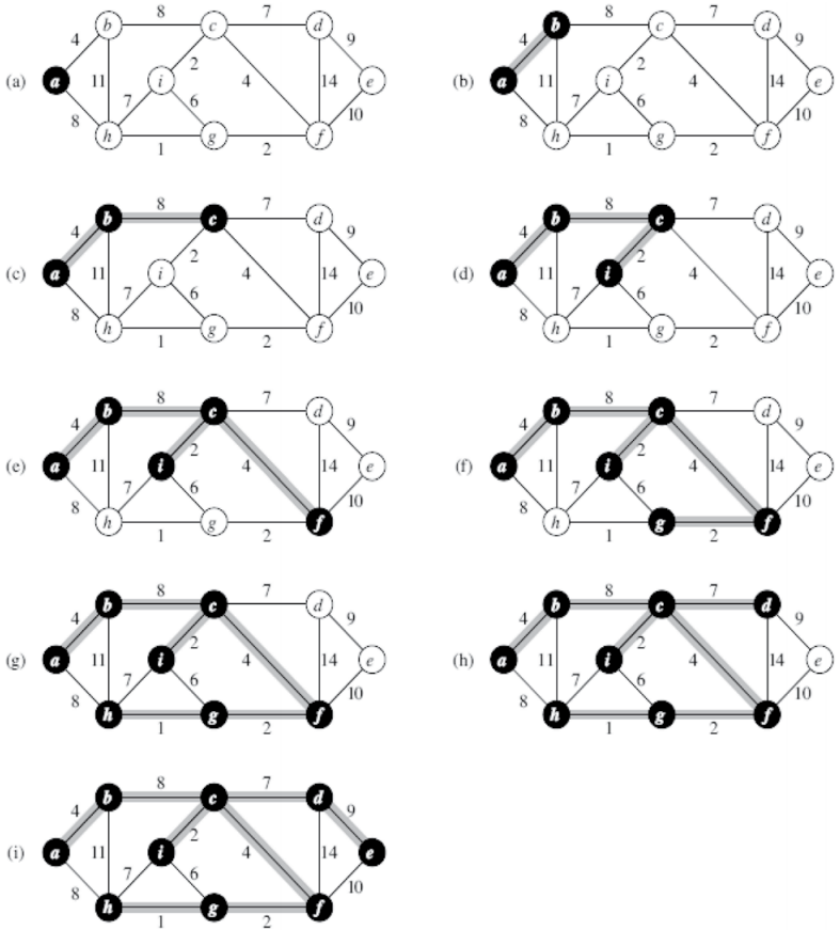
The small arrows on the figures show which edge is currently considered by the algorithm. The shaded edges belong to the spanning tree. Note that in many steps the algorithm merges separated trees, so during the execution we might obtain forests, but upon finishing we have the minimum spanning tree.

Prim's algorithm The other algorithm for obtaining the minimum spanning tree is the following [9]

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

This is also greedy-type algorithm.

Example The following example is taken from [2], it shows the execution of Prim's algorithm on the same graph as for the Kruskal.



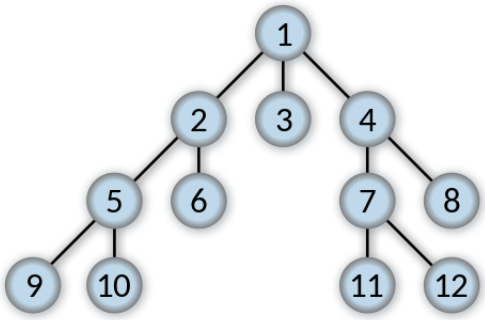
MST example in economics The stock price, as showed in stocks markets, is the reflection of the corresponding company. However, its day-to-day behavior is not merely constrained by the companys own fundamentals; it is influenced by the other companies traded in the market and by the economic factors. Those interrelations among stocks are well-known and quantified in terms of Pearson correlation coefficient. It is then customary to summarize those

interrelationships in the form of a symmetric matrix C of size $n \times n$, called *correlation matrix*, where n is the number of stocks under study. This is nothing else than the previously mentioned *adjacency matrix*, thus it represents a graph. This graph has n^2 edges and n nodes, where not all the edges are really useful. MST can give a representation of a *useful* graph, having only $n - 1$ edges.

1.2.3 Breadth-first search (BFS)

This is an algorithm for traversing a tree (or graph). It starts at the tree root (or some arbitrary node of a graph) and explores the neighbor nodes first, before moving to the next level neighbors.

The concept is shown in the graph below, where the nodes are numbered according to there order of visit. The figure is taken from wikipedia.



BFS application: Testing bipartiteness

BFS can be used to test bipartiteness, by

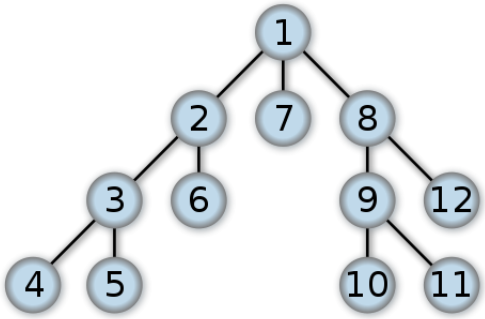
- starting the search at any vertex
- and giving alternating labels to the vertices visited during the search.

That is, give label 0 to the starting vertex, 1 to all its neighbors, 0 to those neighbors' neighbors, and so on. If at any step a vertex has (visited) neighbors with the same label as itself, then the graph is **not bipartite**. If the search ends without such a situation occurring, then the graph is bipartite.

Depth-first search (DFS)

This is another algorithm for traversing a tree (or graph). One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

As an illustration we show the same graph as for the BFS, where the nodes are labeled according to their order in which they are visited by DFS. The figure is taken from wikipedia.



DFS application: finding circle

We do a DFS traversal of the given graph. For every visited vertex v, if there is an adjacent u such that u is already visited and u is not parent of v, then there is a cycle in the graph. If we don't find such an adjacent for any vertex, we say that there is no cycle. The assumption of this approach is that there are no parallel edges between any two vertices.

Chapter 2

Optimization models: linear programming

Learning outcome of the topic *Using small-scale examples the modeling approach with linear programming is shown. We study how to turn an economics-related optimization problem into a mathematical model using the techniques of linear programming. Simple examples demonstrate the geometric meaning of the models. A short introduction is also given to AMPL, which is a mathematical programming language. The students will learn how to turn the write-up of the model into a computer program, which can then be solved easily and can provide the possibility of generalization.*

2.1 Small scale examples

Trader

Assume that there is a small shop selling soft drinks. A child has to go to the shop to buy some light drinks (juice, cola) under the following conditions:

- weights of drinks: 1liter juice = 2kg, 1liter cola = 1kg.
- child can carry maximum 5kg
- father prefers cola, he requests: cola - juice ≥ 1.
- mother requests: at least 3l drink
- trader's profit: 1l juice = 3EUR, 1l cola = 1EUR

Problem to be solved: how much drink should be bought by the child in order to maximize profit, and consider that the child had to carry them, and the requests of the parents are also satisfied.

The mathematical formalism is the following:

- x amount of juice in liter
- y amount of cola in liter
- obviously x and y are non-negative (zero is allowed)
- father's constraint:

$$y \geq x + 1$$

- mother's constraint:

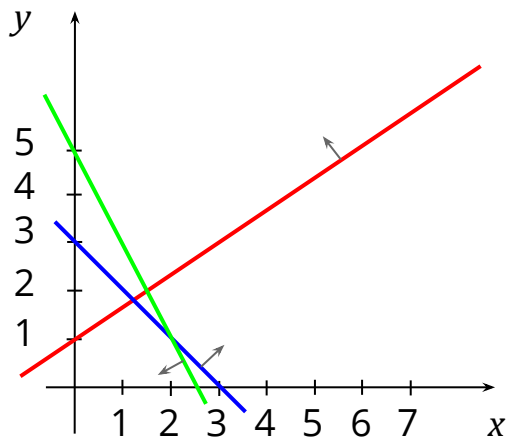
$$x + y \geq 3$$

- child's capacity:

$$2x + y \leq 5$$

- maximize profit: $3x + y$

Given the fact that we have only two variables, it is possible to visualize these constraints using Cartesian coordinate system:

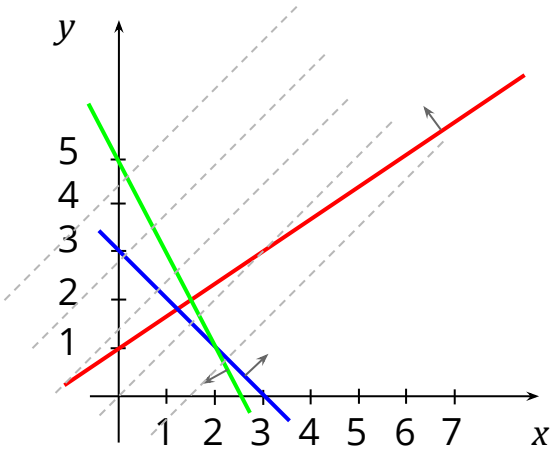


The red line corresponds to the father's constraint, the blue line for the mother's constraint, and the child's capacity is represented by the green line. Together with the x and y axes (so 5 lines altogether) they give the space of the feasible solutions.

The trader's objective function to be maximized: $3x + y$, which can be re-formulated as

$$y = -3x.$$

We need to consider the lines with slope $= -3$:



the possible solutions for the objective function are shown as gray dashed lines. We need to look for the one which crosses the feasible region and the y value is maximized.
That is $(4/3, 7/3)$ for which we obtain $19/3$.

Production planning

The second example is about a company which produces two different products: A and B.

The profits are

- 10 unit/piece for product A,
- 20 unit/piece for product B.

Both products need two machines: M_1 and M_2.

- product A needs 2 hours on M_1 and 2 hours on M_2,
- product B needs 3 hours on M_1 and 1 hour on M_2.

M_1 can work maximum 18 hours/day, and M_2 can work maximum 10 hours/day.

The aim is: maximize profit.

Mathematical formalism is the following.

Variables:

- x: number of product A,
- y: number of product B.

Constraints:

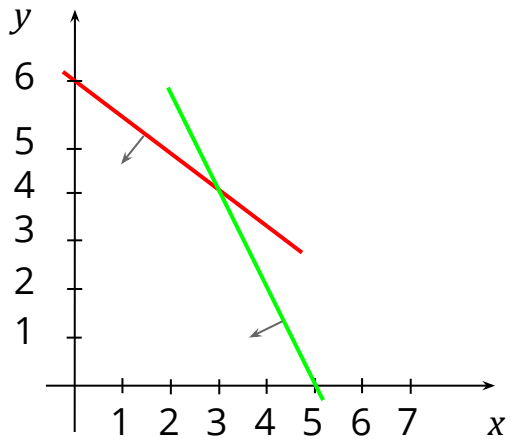
- M_1 capacity: $2x + 3y \leq 18$,

- M_2 capacity: $2x + y \leq 10$.

Objective function:

$$\max 10x + 20y.$$

As in the first example, we still have only 2 variables, so we can use the geometric solution technique:



Rearranging the objective function $10x + 20y$ as an equation we get

$$y = -\frac{1}{2}x$$

so we need to look for such (x, y) pairs where the lines with slope equal to $-1/2$ and the feasible region have common point(s) and y is maximal.
That is $(0, 6)$.

The optimal planning: 0/day for product A, and 6/day for product B. This way we have 120 unit as profit and we can't get better.

2.2 AMPL

In those cases when the number of variables in the linear programming model is larger than 2 we cannot use the geometrical technique. It is not the subject of this course to teach the general methods for solving LPs (such as simplex algorithm, etc).

However, there is a quite general modeling language system in which (among many other, much more involved optimization modeling) we can write down and solve LPs. This is called AMPL, which stands for 'A Mathematical Programming Language'. We give a very brief and rather intuitive introduction to this system.

The simplest way to use AMPL is by its web-based interface, which can be reached at the address:

https://ampl.com/cgi-bin/ampl/amplcgi

We have 2 windows here, in the top one there are already some commands which we should keep as is:

```
solve;
display _varname, _var
```

The first one gives the command to solve the model, while the second one will report the solution: the name of the variables and their optimal value.

In the second window we need to type-in our model.
For example, the production problem has the following AMPL coding:

```
var x>=0;
var y>=0;
maximize profit: 10*x + 20*y;
subject to
M1: 2*x + 3*y <= 18;
M2: 2*x + y <= 10;
```

We can see that the variables are given as `var`. The objective function is either minimized or maximized, and we always have to give a name to it (which is arbitrary).
Then we need to give the constraints, if any. Here we also need to name them (which is again arbitrary).

In order to solve this model we need to push the 'Send' button. Before doing so, we can select the solver to be used. By default, the `minos` solver is set up - that is fine as it can solve LPs.

Solving the problem we get:

```
MINOS 5.51: optimal solution found.
1 iterations, objective 120
: _varname _var    :=
1   x        0
2   y        6
;
```

which is the solution we are familiar with.

Chapter 3

Optimization models: integer linear programming

Learning outcome of the topic *The students will learn what to do with linear programming when integer solutions are needed, which leads us to the integer linear programming (ILP). The general scheme of branch-and-bound technique is studied. Moreover, through a problem studied earlier –shortest path on a graph– the students will also learn how to use ILP for solving certain graph-theoretical problems. The examples help the students to gain further knowledge and experience about algorithmic thinking and mathematical modeling.*

3.1 Motivational example

In order to motivate the topic of this chapter let us consider the following optimization problem.

There is a pizzeria with the following characteristics:

- it sells 2 kinds of pizza
- the prices are 600 and 800
- the ingredients needed:

	M	H
cheese	10	5
ham	2	4
pineapple	0	3

- for 1 day we have: 550 cheese, 150 ham and 120 pineapple.

As usual, the pizzeria wants to maximize profit.

LP model The variables:

x_1: number of M pizza,

x_2: number of H pizza,

$x_1, x_2 \geq 0$.

The constraints:

$$\begin{aligned} 10x_1 + 5x_2 &\leq 550 \\ 2x_1 + 4x_2 &\leq 150 \\ 3x_2 &\leq 120 \end{aligned}$$

The objective function:

$$\max 600x_1 + 800x_2$$

Now, let us see the model in AMPL:

```
var x1 >= 0;
var x2 >= 0;
maximize profit: 600*x1 + 800*x2;
subject to
cheese: 10*x_1 + 5*x2 <= 550;
ham: 2*x1 + 4*x2 <= 150;
pineapple: 2*x2 <= 120;
```

If we upload this to the AMPL server and we use MINOS as a solver, we obtain the solution:

```
MINOS 5.51: optimal solution found.
2 iterations, objective 39666.66667
: _varname    _var      :=
1   x1       48.3333
2   x2       13.3333
;
```

This means that we need to sell pizza slices in order to maximize profit.

3.2 Integer solutions?

Now the question arises: is it possible to obtain integer solution for the previous problem?

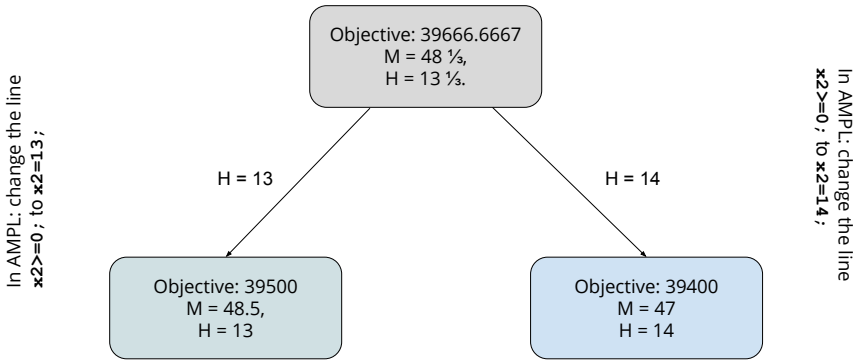
This is the subject of the so-called Integer Linear Programming (ILP), where

- the variables can take only integer variables,

- and a special version where the variables can only be either 0 or 1 (binary LP)

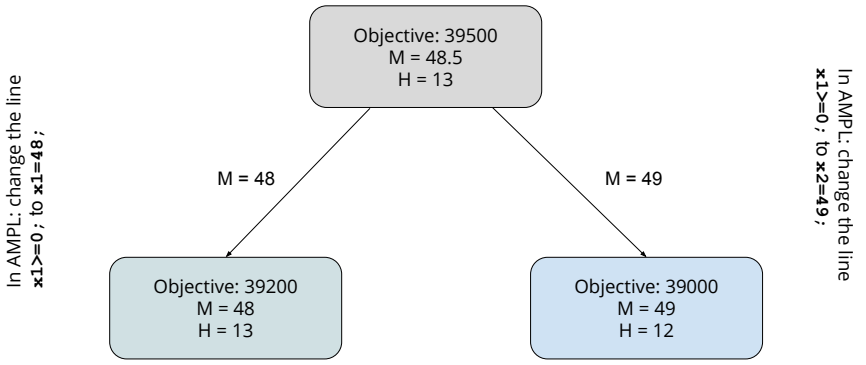
In the previous example we got $481/3$ M and $131/3$ H pizza.

Now, let us consider the following two cases: $H = 13$ and $H = 14$. The following figure illustrates these two choices and it includes the solutions for those (restricted) problems:



Note that in both cases we obtained worse solution than before (when we allowed the selling of slices).

For the case `x2 = 13` we obtained `x1 = 48.5`, so we can consider again two cases:



We obtained lower objective values than before. Hence, the best (integer) solution is

$$M = 47 \quad H = 14,$$

and the objective function (profit) is 39,400.

The approach we just followed is called: Branch-and-Bound. You get the idea why.

3.3 ILP in AMPL

For the pizza selling problem the AMPL model is

```
var x1 >= 0 integer;
var x2 >= 0 integer;
maximize profit: 600*x1 + 800*x2;
subject to
cheese: 10*x_1 + 5*x2 <= 550;
ham: 2*x1 + 4*x2 <= 150;
pineapple: 2*x2 <= 120;
```

If we try to solve this with MINOS then we get

```
MINOS 5.51: ignoring integrality of 2 variables
MINOS 5.51: optimal solution found.
3 iterations, objective 39666.66667
: _varname    _var      :=
1   x1       48.3333
2   x2       13.3333
;
```

which is essentially the same as before and we have the evidence that MINOS is ignoring the fact that we require to keep the variables as integer.

Hence, we need to use another solver, which is capable of solving ILP. One of them is called `lpsolve`. Choosing `lpsolve` we obtain

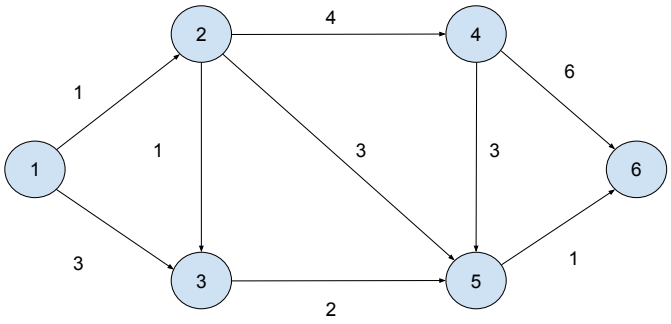
```
LP_SOLVE 4.0.1.0: optimal, objective 39400
7 simplex iterations
5 branch & bound nodes: depth 3
: _varname _var    :=
1   x1       47
2   x2       14
;
```

We can see that `lpsolve` took 3 branch-and-bound steps, as we did.

3.4 Graph problems as ILP

Previously we have studied graph problems and ILP formalism separately. As an example we are going to demonstrate how ILP can be used to solve the shortest path problem on graphs.

Let us consider the following graph:



Very important fact: in the ILP formalism the variables represent the edges.

For example, on the above graph, x_{35} represents the edge between node 3 and 5. Clearly,

- if $x_{35} = 1$ then the edge is on the shortest path,
- if $x_{35} = 0$ then the edge is not on the shortest path.

Moreover, we know that if we are in a node, then we need to leave it (on one of the outgoing edges). The only exceptions are the starting and the target node.

We go straight to the AMPL model as we got quite familiar with the formalism.

```
var x12 binary;
var x13 binary;
var x23 binary;
var x24 binary;
var x25 binary;
var x35 binary;
var x45 binary;
var x46 binary;
var x56 binary;

minimize path: x12 + 3*x13 + x23 + 4*x24 + 3*x25 + 2*x35 +
               3*x45 + 6*x46 + x56 ;

subject to

leaving: x12 + x13 = 1 ;
```

```
at_node2: x12 = x23 + x24 + x25;
at_node3: x13 + x23 = x35;
at_node4: x24 = x46 + x45;
at_node5: x35 + x25 + x45 = x56;
```

We can see that the objective function (to be minimized) is the sum of the binary variables multiplied with their corresponding edge weights.
Solving the AMPL model with `lpsolve` (as we need to use an ILP solver) we obtain

```
LP_SOLVE 4.0.1.0: optimal, objective 5
5 simplex iterations
: _varname _var    :=
1   x12      1
2   x13      0
3   x23      0
4   x24      0
5   x25      1
6   x35      0
7   x45      0
8   x46      0
9   x56      1
;
```

So the shortest path is 1 - 2 - 5 - 6 and its weighted length is 5 (as we have seen earlier).

Exercise

Consider the modified version of the shortest path problem: what do we need to do if we must visit certain nodes? How would you modify the model?
(Do not forget: the variables are representing edges and not nodes!)

Chapter 4

Critical Path Method

Learning outcome of the topic: *After studying the basic concepts of graph theory and some classical algorithms, in this topic we will learn the critical path method (CPM) which is a standard procedure for project scheduling and analysis. The students will learn how to formalize a project into a dependency graph and execute several algorithms to obtain the optimal scheduling. Even further analysis is studied which reveals the slack time, if any, in the optimally planned execution of a project. By all these the students will understand what is going on inside otherwise 'blindly' used project management tools.*

4.1 Definitions

The essential technique for using Critical Path Method (CPM) [5] is to construct a model of the project that includes the following:

- A list of all activities required to complete the project (also known as *Work Breakdown Structure*)
- The time (duration) that each activity will take to be completed
- The dependencies between the activities.

CPM calculates

- The longest path of planned activities to the end of the project
- The earliest and latest that each activity can start and finish without making the project longer

It determines *critical* activities (on the longest path).

Prioritize activities for the effective management and to shorten the planned critical path of a project by:

- Pruning critical path activities
- '*Fast tracking*' (performing more activities in parallel)
- '*Crashing the critical path*' (shortening the duration of critical path activities by adding resources)

Further properties of CPM:

- Represent a project (set of task) as a graph (or network)
- A project consists of a collection of well defined tasks (jobs)
- A project ends when all jobs have been completed
- Jobs may be started and stopped independently of each other within a given sequence (no *continuous-flow* processes)
- Jobs are ordered (*technological sequence*)

4.2 Simple Example: Job List

Two Parts X and Y: Manufacture and Assembly using lathe[1]

Job ID	Description	Predecessor(s)	Duration [min]
A	Start		0
B	Get materials for X	A	10
C	Get materials for Y	A	20
D	Turn X on lathe	B,C	30
E	Turn Y on lathe	B,C	20
F	Polish Y	E	40
G	Assemble X and Y	D,F	20
H	Finish	G	0

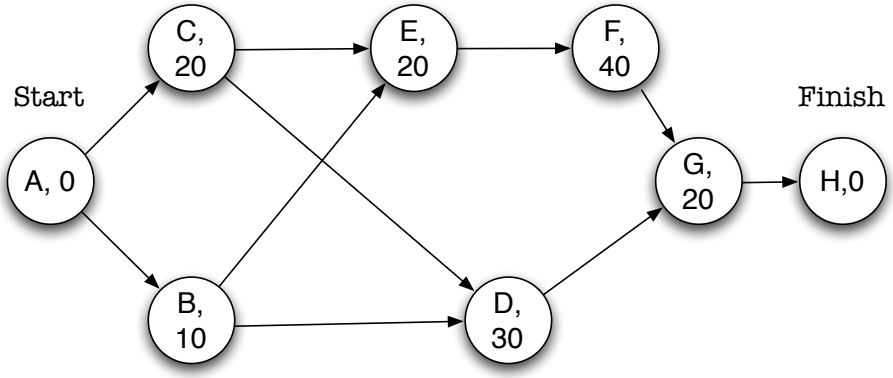
How does the project graph look like?

- Each job is drawn on a graph as a circle
- Connect each job with immediate predecessor(s), directed edges
- Jobs with no predecessor connect to "**Start**"
- Jobs with no successors connect to "**Finish**"
- "**Start**" and "**Finish**" are pseudo-jobs of length 0
- Result: a finite number of "arrow paths" from "**Start**" to "**Finish**"

[1] A lathe is a tool that rotates the work-piece about an axis of rotation to perform various operations such as cutting, sanding, knurling, drilling, deformation, facing, turning, with tools that are applied to the work-piece to create an object with symmetry about that axis.

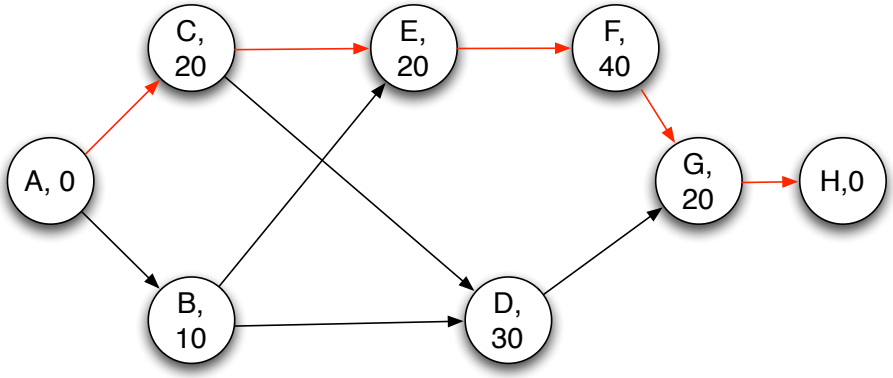
- The total time of each path is the sum of job times
- The path with the longest total time is the *critical path*
- There can be multiple critical paths, that is the minimum time to complete a project

One possible drawing is the following:



There are 4 unique paths: A, C, E, F, G, H; A, C, D, G, H; A, B, D, G, H; and A, B, E, F, G, H.

By brute-force (complete enumeration of all the possible solutions) we obtain the critical path: A, C, E, F, G, H. The value of the solution is 100 (time unit). On the graph it is colored red:



4.3 Critical Path (CP)

CP is the *bottleneck route* with the following properties:

- Shortening or lengthening tasks on the critical path directly affects project finish
- Duration of *non-critical* tasks is irrelevant
- *Crashing* all jobs is ineffective, focus on the few percentage of jobs that are on the CP

- *Crashing* tasks can shift the CP to a different task
- Shortening tasks – technical and economical challenge
- Previously non-critical tasks can become critical
- Lengthening of non-critical tasks can also shift the critical path

For large projects there are many paths, so we need an algorithm to identify the CP efficiently.

4.4 CP Algorithm

Consider the following notations:

- Times
  - Start time (S)
  - For each job: Earliest Start (ES)
    - Earliest start time of a job if all its predecessors start at ES
  - Job duration: t
  - Earliest Finish: $EF = ES + t$
- Finish time (F): earliest finish time of the overall project

The following description is a quite standard one and it is taken from [7].

CP Algoritm

1. Mark the value of S to left and right of **Start**
2. Consider any new unmarked job, all of whose predecessors have been marked.
   Mark to the left of the new job the largest number to the right of its immediate predecessors: (ES)
3. Add to ES the job time t and mark result to the right (EF)
4. Stop when **Finish** node has been reached

4.5 Latest Start and Finish Times

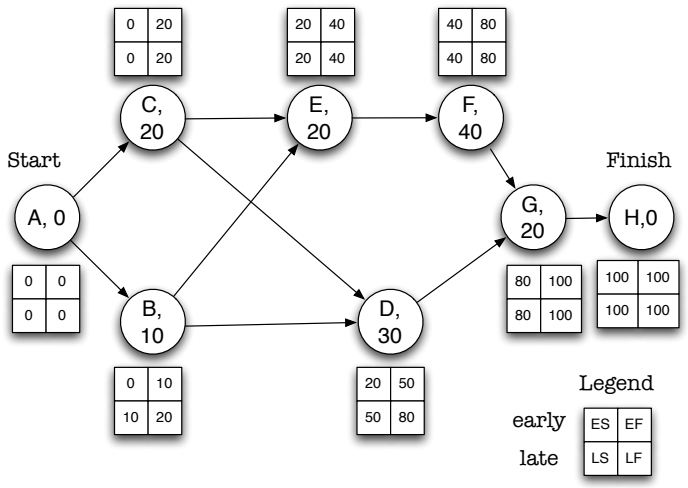
Set target finish time for project: $T \geq F$. Usually target is a specific calendar date, e.g. November 7, 2018. When is the latest date the project can be started?

- Late Finish (LF): latest time a job can be finished, without delaying the project beyond its target time (T)
- Late Start: $LS = LF - t$

Work from the end of the project: T

1. Mark value of T to left and right of **Finish**
2. Consider any new unmarked job, all of whose successors have been marked – mark to the right the smallest LS time marked to the left of any of its immediate successors
3. Subtract from this number, LF, the job time t and mark result to the left of the job: LS
4. Continue upstream until **Start** has been reached, then STOP

4.5.1 Illustration of LS and FT



Note: ES and EF values are given by the original execution of the algorithm.

4.6 Slack

Some tasks have $ES = LS$, we have no slack. Total Slack of a task

$$TS = LS - ES.$$

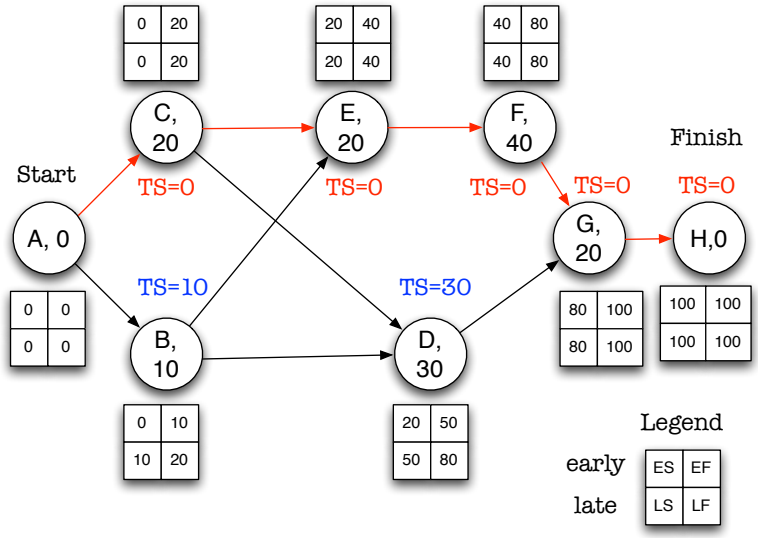
Maximum amount of time a task may be delayed beyond its early start without delaying project completion. Slack time is precious, managerial freedom, don't squander it unnecessarily

- e.g. resource, work load smoothing

When $T = F$ then all critical tasks have $TS = 0$. At least one path from **Start** to **Finish** with critical jobs only. When $T > F$, then all critical jobs have $TS = T - F$.

4.6.1 Illustration of slack

On the same graph as we used earlier:



4.7 Complete example – event organization

Consider the following problem (taken from [1]):

Task	dependency	duration
license (L)	–	10
security (S)	–	5
sponsors (Sp)	L	6
equipments (E)	Sp	7
musicians (M)	Sp	14
ticket (T)	M	5
media (N)	S, M	8
poster (P)	S, M	4
rehearsals (R)	N	8
printing (Pr)	T, P	7

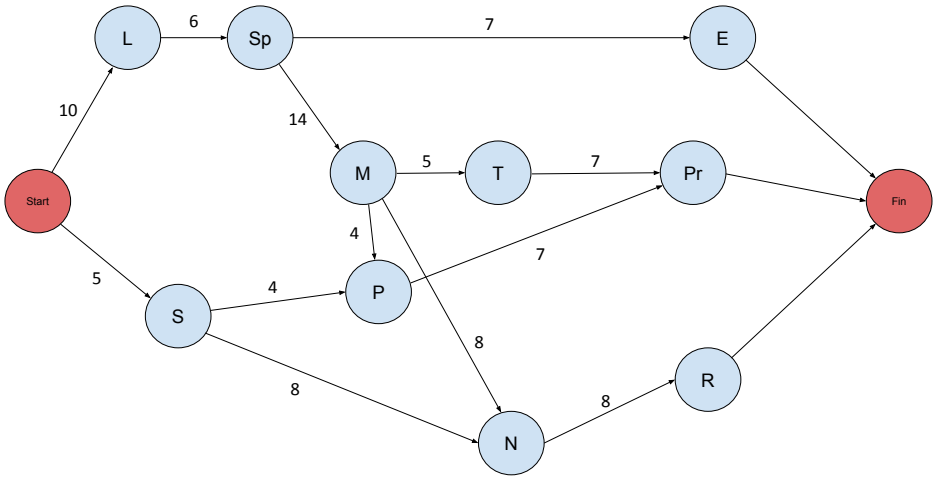
What is the CPM solution?

CPM solution

First step: let's draw the graph based on the above table in order to visualize the dependencies.

- Note that we are drawing the graph in such a way that we put the duration of a task on the incoming edge(s). This formalism is closer to the AMPL model which we saw during the lecture.

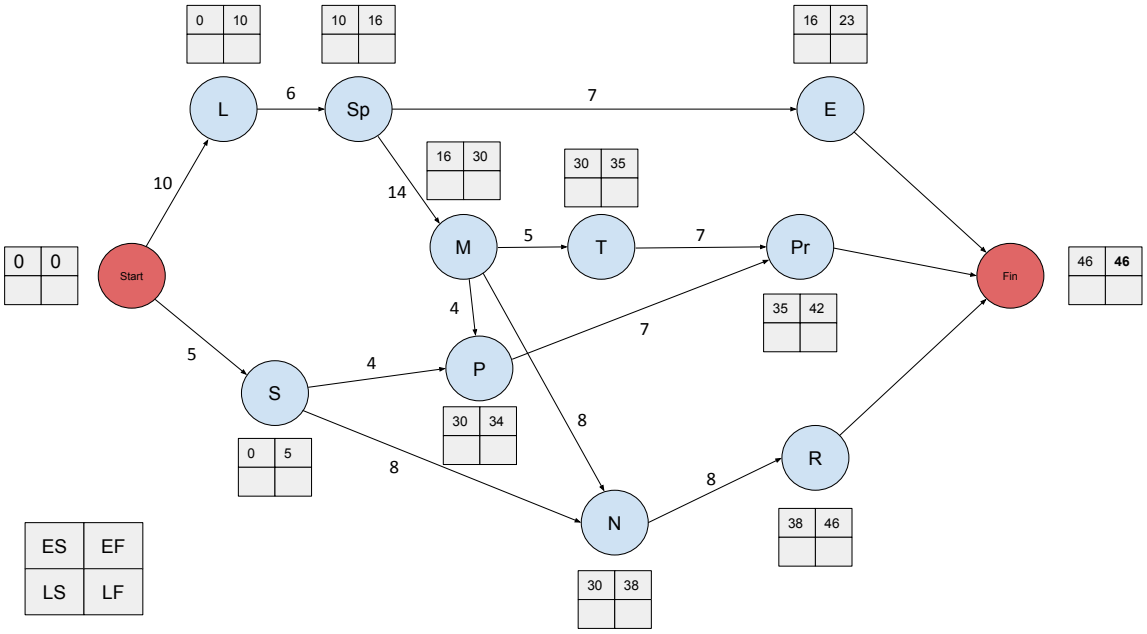
- As usual, we added 2 artificial nodes (Start and Fin). This is also needed in the AMPL model.



Second step: We execute now the algorithm and obtain the time of the longest path.

- For this, we introduce that small 2-by-2 table, which we used at the lecture, containing: Early Start, Early Finish.

- (the other two elements of the table will be filled in the next step).

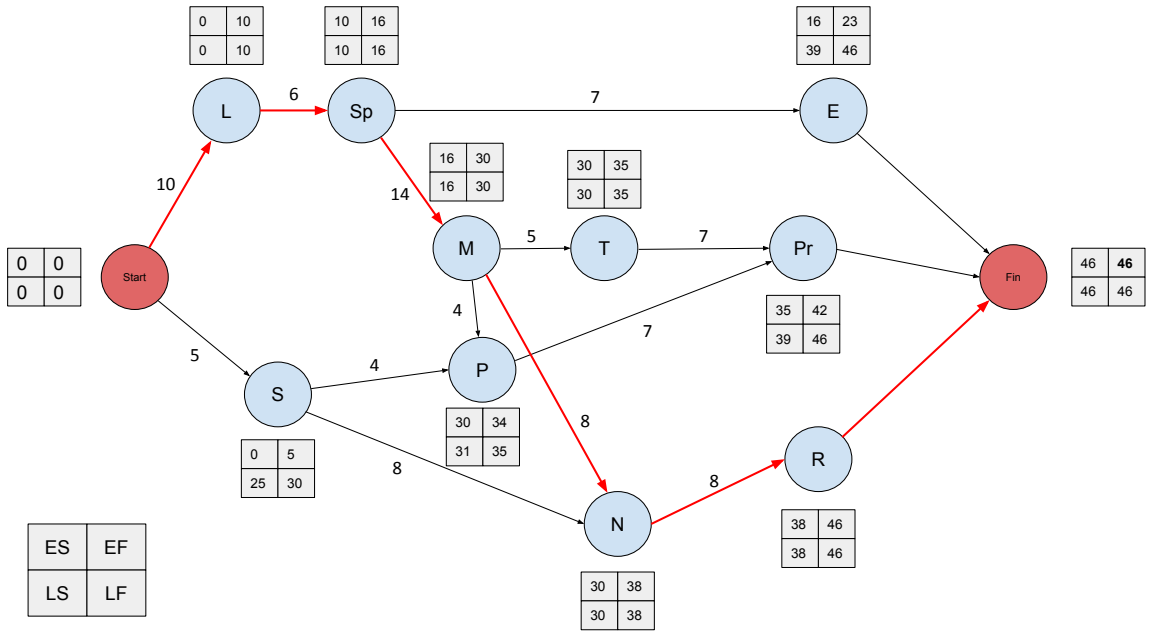


As a result, we can see that the length of the longest path is 46 (unit of time): that is the ES (and EF) numbers we obtained in the 'Fin' node.

There are some trivial results (such as the numbers at nodes L, Sp and E), as well as some non-trivial ones:

- *Why did we get 30 and 34 at node P?* Node P has 2 incoming edge:
  - on the edge coming from node S we get 5, to which we need to add 4 (the duration of node P) which makes 9.
  - on the edge coming from node M we get 30, to which we need to add 4 (the duration of node P) which makes 34.
  - now we need to take the maximum of 9 and 34, which makes 34 (that is the ES value of node P), hence EF = 34.
- So the general rule is: if we have multiple incoming edges we need to take the maximum of the previous nodes' EF values.

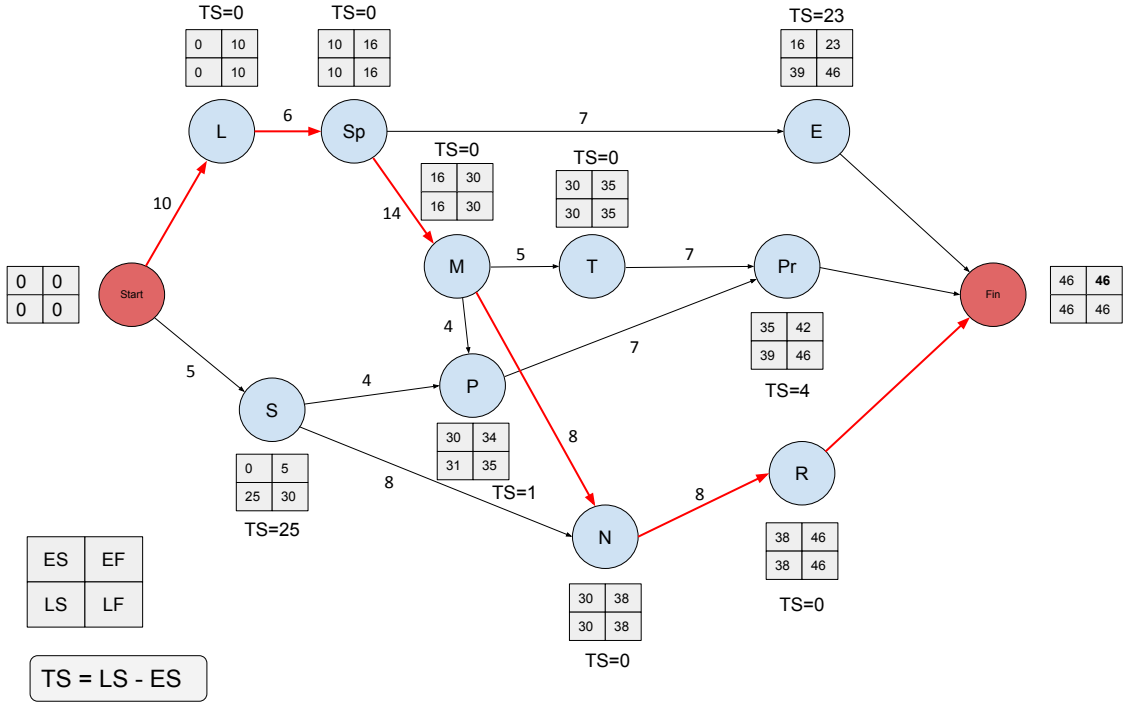
Third step: We execute now the algorithm backwards in order to obtain the LF (late finish) and LS (late start) values. Here it is:



The rules here are also very simple:

- a node with 1 outgoing edge takes the ES value of its neighbor as its own LF value
  - node E is connected to Fin, so LF = 46 (that is the ES of Fin)
  - node T is connected to Pr, so LF = 35 (that is the ES of Pr)
- a node with more than 1 outgoing edge takes the *minimum* of the ES values of its neighbors as its own LF value
  - node M is connected to P, T, and N; the minimum of 30, 30, 30 is apparently 30, so LF = 30 for node M.
- Finally, we get the LS values by subtracting the duration time of a node from the LF value
  - node Pr has LS = 39: 46 - 7.

Final step: We can obtain the longest path and the slack times (TS), as it is shown in the final figure:



- example: node Pr has TS = 46 - 39 = 4.
- In general: those nodes where we have TS = 0 are part of the longest path.

Exercise

Give the CPM graph of the following project and solve it with the algorithm.

The project starts with $(A, 5)$. Task $(B, 10)$ can start after A is completed. This is also true for task $(E, 5)$. Task $(C, 8)$ depends only on $(B, 10)$, while task $(F, 10)$ depends on both $(B, 10)$ and $(E, 5)$. Task $(D, 5)$ is the last task in the project and it can start once $(C, 8)$ and $(F, 10)$ have been finished.

What is the Earliest Finish (EF) time for the whole project?
(Possible answers: 20, 22, 25, 27, 30, 35, 40)

Chapter 5

Program Evaluation and Review Technique

Learning outcome of the topic: *In this lecture the students will learn another project scheduling algorithm, called program evaluation and review technique (PERT), which was developed around the same time as the previously studied CPM. Similarities and differences between the two methods are discussed. The students will also learn how to incorporate probabilities into project scheduling and management.*

5.1 Definitions

PERT = Program (or Project) Evaluation and Review Technique

PERT is a method to analyze the involved tasks in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project.

PERT was developed primarily to simplify the planning and scheduling of large and complex projects. It is able to incorporate uncertainty by making it possible to schedule a project while not knowing precisely the details and durations of all the activities.

It is more of an event-oriented technique rather than start- and completion-oriented, used more in projects where time is the major factor rather than cost.

Critical Path Method (previous lecture) was invented at roughly the same time as PERT.

PERT Terminology

PERT event a point that marks the start or completion of one or more activities. It consumes no time and uses no resources.

predecessor event an event that immediately precedes some other event without any other events intervening. An event can have multiple predecessor events and can be the predecessor of multiple events.

successor event an event that immediately follows some other event without any other intervening events. An event can have multiple successor events and can be the successor of multiple events.

PERT activity the actual performance of a task which consumes time and requires resources (such as labor, materials, space, machinery). It can be understood as representing the time, effort, and resources required to move from one event to another. A PERT activity cannot be performed until the predecessor event has occurred.

PERT sub-activity a PERT activity can be further decomposed into a set of sub-activities. For example, activity A1 can be decomposed into A1.1, A1.2 and A1.3. Sub-activities have all the properties of activities, in particular a sub-activity has predecessor or successor events just like an activity. A sub-activity can be decomposed again into finer-grained sub-activities.

optimistic time (O) the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected

pessimistic time (P) the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).

most likely time (M) the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.

expected time (t_e) the best estimate of the time required to accomplish a task, accounting for the fact that things don't always proceed as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

$$t_e = (O + 4M + P)/6$$

standard deviation and variation

$$\sigma = \frac{P - O}{6} \qquad \sigma^2 = \left(\frac{P - O}{6}\right)^2$$

float or slack is a measure of the excess time and resources available to complete a task. It is the amount of time that a project task can be delayed without causing a delay in any subsequent tasks (free float) or the whole project (total float). Positive slack would indicate

ahead of schedule; negative slack would indicate behind schedule; and zero slack would indicate on schedule.

critical path the longest possible continuous pathway taken from the initial event to the terminal event. It determines the total calendar time required for the project; and, therefore, any time delays along the critical path will delay the reaching of the terminal event by at least the same amount.

critical activity An activity that has total float equal to zero. An activity with zero float is not necessarily on the critical path since its path may not be the longest.

Lead time the time by which a predecessor event must be completed in order to allow sufficient time for the activities that must elapse before a specific PERT event reaches completion.

lag time the earliest time by which a successor event can follow a specific PERT event.

fast tracking performing more critical activities in parallel

crashing critical path Shortening duration of critical activities

5.2 PERT versus CPM

Difference how *task duration* is treated

CPM assumes time estimates are deterministic

- Obtain task duration from previous projects
- Suitable for ”implementation”-type projects

PERT treats duration as probabilistic

- PERT = CPM + probabilistic task times
- Better for ”uncertain” and new projects
- Limited previous data to estimate time duration
- Captures schedule (and implicitly some cost) risk

5.3 Example

Id	Predecessor	Time estimates			Expected time
		Opt. (O)	Normal (M)	Pess. (P)	
A	–	2	4	6	4.00
B	–	3	5	9	5.33
C	A	4	5	7	5.17
D	A	4	6	10	6.33
E	B, C	4	5	7	5.17
F	D	3	4	8	4.50
G	E	3	5	8	5.17

Expected time: $t_e = (O + 4M + P)/6$
std deviation: $\sigma = (P - O)/6$
variation: $\sigma^2 = (P - O)^2/36$

This table contains typical values for calculating the standard deviation and variance:

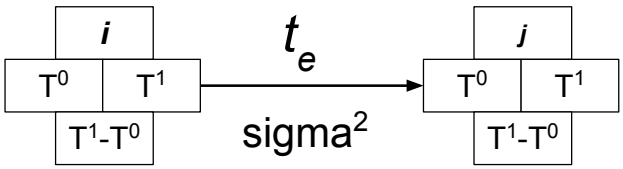
$P - O$	σ	σ^2
1	0.1666666667	0.0277777778
2	0.3333333333	0.1111111111
3	0.5	0.25
4	0.6666666667	0.4444444444
5	0.8333333333	0.6944444444
6	1	1
7	1.1666666667	1.3611111111
8	1.3333333333	1.7777777778
9	1.5	2.25
10	1.6666666667	2.7777777778
11	1.8333333333	3.3611111111
12	2	4
13	2.1666666667	4.6944444444
14	2.3333333333	5.4444444444
15	2.5	6.25
16	2.6666666667	7.1111111111
17	2.8333333333	8.0277777778
18	3	9
19	3.1666666667	10.0277777778
20	3.3333333333	11.1111111111
21	3.5	12.25
22	3.6666666667	13.4444444444
23	3.8333333333	14.6944444444
24	4	16
25	4.1666666667	17.3611111111

Back to the example:

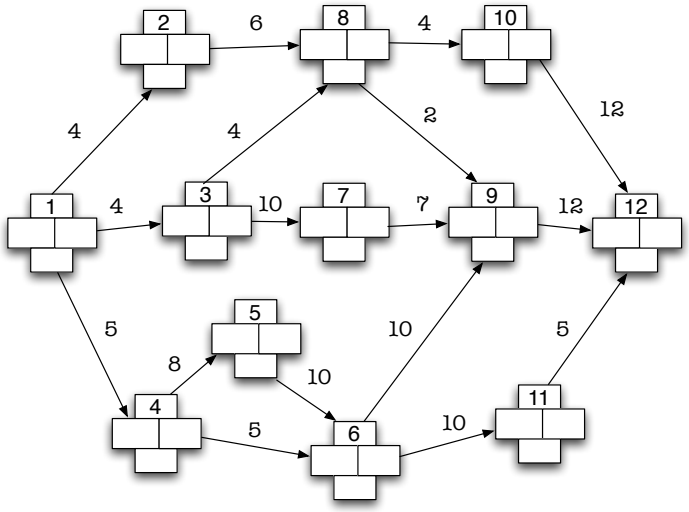
Event	optimistic	normal	pessimistic
1 - 2	2	4	6
1 - 3	1	4	7
1 - 4	3	5	7
2 - 8	3	6	9
3 - 7	5	10	15
3 - 8	2	4	6
4 - 5	4	8	12
4 - 6	1	5	9
5 - 6	6	10	14
6 - 9	4	10	18
6 - 11	2	10	18
7 - 9	3	7	11
8 - 9	1	2	3
8 - 10	1	4	7
9 - 12	5	12	19
10 - 12	4	12	20
11 - 12	4	5	6

5.4 PERT graph

The general scheme of the network:



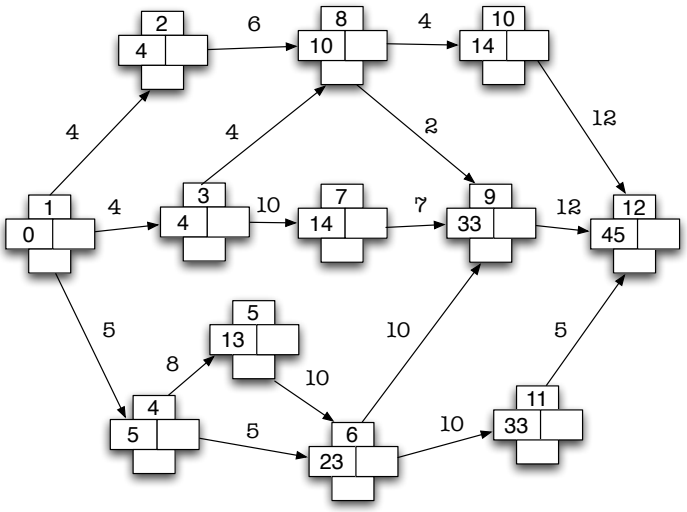
PERT graph - 1



$$
\begin{aligned}
T_2^0 &= 0+4=4 \\
T_3^0 &= 0+4=4 \\
T_4^0 &= 0+5=5 \\
T_5^0 &= 5+8=13 \\
T_6^0 &= \max\{5+5, 13+10\}=23 \\
T_7^0 &= 4+10=14 \\
T_8^0 &= \max\{4+6, 4+4\}=10 \\
T_9^0 &= \max\{14+7, 10+2, 23+10\}=33 \\
T_{10}^0 &= 10+4=14 \\
T_{11}^0 &= 23+10=33 \\
T_{12}^0 &= \max\{33+12, 14+12, 33+5\}=45
\end{aligned}
$$

In general: $T_j^0 = T_i^0 + t_e$ or $T_j^0 = \max\{T_i^0 + t_c\}$

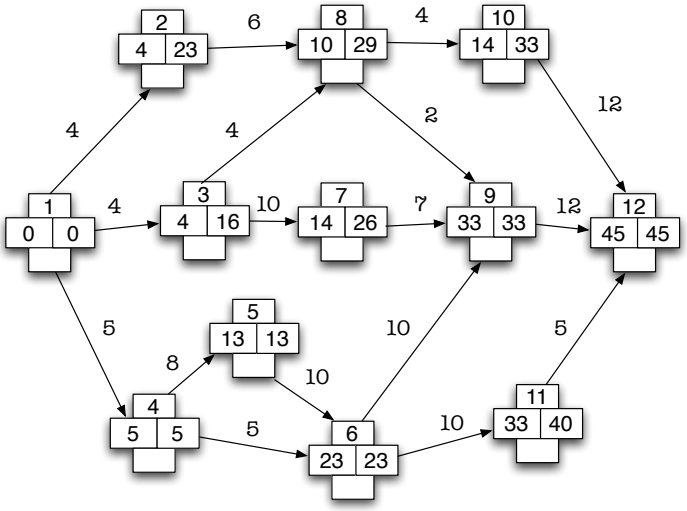
PERT graph - 2



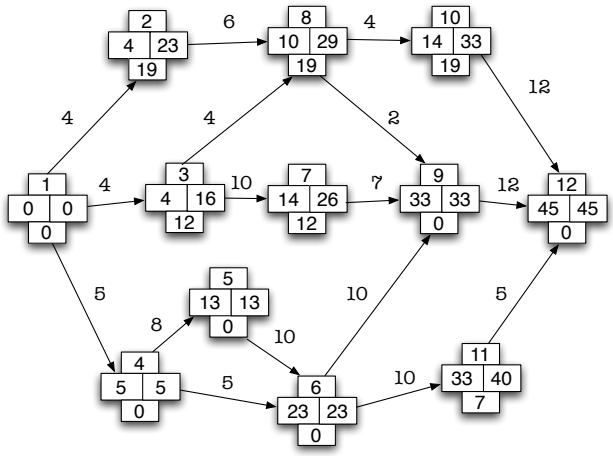
$$
\begin{aligned}
T_{11}^1 &= 45 - 5 = 40 \\
T_{10}^1 &= 45 - 12 = 33 \\
T_9^1 &= 45 - 12 = 33 \\
T_8^1 &= \min\{33 - 2; 33 - 4\} = 29 \\
T_7^1 &= 33 - 7 = 26 \\
T_6^1 &= \min\{33 - 10; 40 - 10\} = 23 \\
T_5^1 &= 23 - 10 = 13 \\
T_4^1 &= \min\{13 - 8; 23 - 5\} = 5 \\
T_3^1 &= \min\{26 - 10; 29 - 4\} = 16 \\
T_2^1 &= 29 - 6 = 23 \\
T_1^1 &= \min\{23 - 4; 16 - 4; 5 - 5\} = 0
\end{aligned}
$$

In general: $T_i^1 = T_j^1 - t_e$ or $T_i^1 = \min\{T_j^1 - t_c\}$

PERT graph - 3



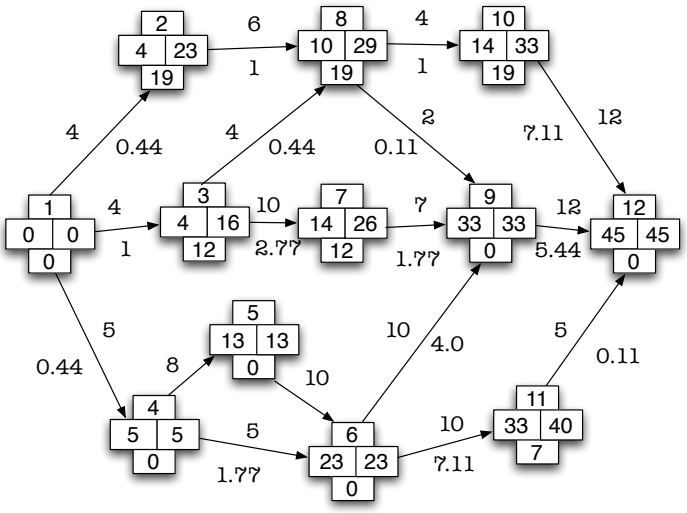
PERT graph - 4, slack



Critical path: $1 \to 4 \to 5 \to 6 \to 9 \to 12$, i.e., where slack$= 0$

$$
\begin{aligned}
\sigma^2_{T^0_1} &= 0 \\
\sigma^2_{T^0_2} &= 0 + 0.44 = 0.44 \\
\sigma^2_{T^0_3} &= 0 + 1 = 1 \\
\sigma^2_{T^0_4} &= 0 + 0.44 = 0.44 \\
\sigma^2_{T^0_5} &= 0.44 + 1.77 = 2.21 \\
\sigma^2_{T^0_6} &= \max\{0.44 + 1.77; 2.21 + 1.77\} = 3.98 \\
\sigma^2_{T^0_7} &= 1 + 2.77 = 3.77 \\
\sigma^2_{T^0_8} &= \max\{0.44 + 1; 1 + 0.44\} = 1.44 \\
\sigma^2_{T^0_9} &= \max\{3.98 + 5.44; 3.77 + 1.77; 1.44 + 0.11\} = 9.42 \\
\sigma^2_{T^0_{10}} &= 1.44 + 1 = 2.44 \\
\sigma^2_{T^0_{11}} &= 3.98 + 7.11 = 11.09 \\
\sigma^2_{T^0_{12}} &= \max\{9.42 + 5.44; 2.44 + 7.11; 11.09 + 0.11) = 14.86
\end{aligned}
$$

PERT graph - 5, variance of starting times



Variance of finishing times

$$
\begin{aligned}
\sigma^2_{T^1_{12}} &= 0 \\
\sigma^2_{T^1_{11}} &= 0 + 0.11 = 0.11 \\
\sigma^2_{T^1_{10}} &= 0 + 7.11 = 7.11 \\
\sigma^2_{T^1_{9}} &= 0 + 5.44 = 5.44 \\
\sigma^2_{T^1_{8}} &= \max\{5.44 + 0.11; 7.11 + 1\} = 8.11 \\
\sigma^2_{T^1_{7}} &= 5.44 + 1.77 = 7.21 \\
\sigma^2_{T^1_{6}} &= \max\{5.44 + 5.44; 0.11 + 7.11) = 10.88 \\
\sigma^2_{T^1_{5}} &= 10.88 + 1.77 = 12.65 \\
\sigma^2_{T^1_{4}} &= \max\{12.56 + 1.77; 10.88 + 1.77) = 14.42 \\
\sigma^2_{T^1_{3}} &= \max\{7.21 + 2.77; 5.55 + 0.44; 8.11 + 0.44) = 9.98 \\
\sigma^2_{T^1_{2}} &= 8.11 + 1 = 9.11 \\
\sigma^2_{T^1_{1}} &= \max\{9.11 + 0.44; 9.98 + 1; 13.42 + 0.44) = 14.86
\end{aligned}
$$

Probability of an event becoming critical

We know that those events which are on the critical path have 0 slack. What is the probability that an event with non-zero slack becomes critical?
That can be calculated:

$$Z = -\frac{T^1 - T^0}{\sigma^2_{T^1} - \sigma^2_{T^0}}$$

Note that Z is **not** a probability!

The *smaller*[1] this number the *smaller* the probability of the event becoming critical.

[1] or *larger* in absolute value

$$
\begin{aligned}
Z_1 &= \frac{-(0-0)}{\sqrt{14.68+0}} = 0 \\
Z_2 &= \frac{-(23-4)}{\sqrt{9.11+0.44}} = -6.1 \\
Z_3 &= \frac{-(16-4)}{\sqrt{9.98+1}} = -3.6 \\
Z_4 &= \frac{-(5-5)}{\sqrt{14.42+0.44}} = 0 \\
Z_5 &= \frac{-(13-13)}{\sqrt{12.65+2.21}} = 0 \\
Z_6 &= \frac{-(23-23)}{\sqrt{10.88+3.98}} = 0 \\
Z_7 &= \frac{-(26-14)}{\sqrt{7.21+3.77}} = -3.6 \\
Z_8 &= \frac{-(29-10)}{\sqrt{8.11+1.44}} = -6.1 \\
Z_9 &= \frac{-(33-33)}{\sqrt{5.44+9.42}} = 0 \\
Z_{10} &= \frac{-(33-14)}{\sqrt{7.11+2.44}} = -6.1 \\
Z_{11} &= \frac{-(40-33)}{\sqrt{0+11.09}} = -2.1 \\
Z_{12} &= \frac{-(45-45)}{\sqrt{0+14.86}} = 0
\end{aligned}
$$

Z	$P\,\%$	Z	$P\,\%$
−3,0	0,13	−1,4	8,08
−2,9	0,19	−1,3	9,68
−2,8	0,26	−1,2	11,51
−2,7	0,35	−1,1	13,57
−2,6	0,47	−1,0	15,87
−2,5	0,62	−0,9	18,41
−2,4	0,82	−0,8	21,19
−2,3	1,07	−0,7	24,20
−2,2	1,39	−0,6	27,43
−2,1	1,79	−0,5	30,58
−2,0	2,28	−0,4	34,46
−1,9	2,87	−0,3	38,21
−1,8	3,59	−0,2	42,07
−1,7	4,46	−0,1	46,02
−1,6	5,48	0,0	50,00
−1,5	6,68		

We can conclude that those events which are not on the critical path are having small probabilities to become critical, the reason for this is that those events have lots of slack.

Probability of keeping the schedule

What is the probability that we are able to keep the deadline which we derived by the PERT model (based on the expected time)?

This can be calculated using

$$Z = \frac{T^H - T^0}{\sqrt{\sigma^2_{T^0}}},$$

where T^H is the time we are giving, and then checking the table from the previous slide (on page 28) for the actual probability.

In the example: what is the probability that we can finish in 43 days, instead of 45? We have

$$Z = \frac{43 - 45}{\sqrt{14.86}} = -0.52,$$

so that the probability is between 27.43% and 30.58%, which can be considered as high.

Chapter 6

Basic concepts in Network Science

Learning outcome of the topic: *Some of the basic concepts of graphs are re-visited and extended. This lecture aims to show how graph analytics can reveal interesting properties of graphs modeling real-life scenarios. The students will learn about a nice tool called Gephi, which is capable of doing analytics on graphs, and visualize them.*

6.1 Networks and their representations

As we have studied in Chapter 1 network (graph) is a set of vertices and edges.

In the following we use the notations for the number of nodes: n, and for the number of edges: m.

Adjacency matrix:

$$A_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Undirected simple graphs: symmetric with all zeros in the diagonal.

6.1.1 Degree

The degree of a node is the number of edges connected to it:

$$k_i = \sum_{j=1}^{n} A_{ij}$$

An undirected network has m edges, so

$$2m = \sum_{i=1}^{n} k_i$$

and thus

$$m = \frac{1}{2}\sum_{i=1}^{n} k_i = \frac{1}{2}\sum_{i,j} A_{ij}$$

The average degree of a network:

$$c = \frac{1}{n}\sum_{i=1}^{n} k_i$$

so we obtain

$$c = \frac{2m}{n},$$

where m is the number of edges and n is the number of vertices.

6.1.2 Density

$$\rho = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)} = \frac{c}{n-1}$$

6.1.3 Shortest Paths and Diameter

Also called as geodesic distance, it is the shortest existing path between two nodes.

Shortest distance: the length of the shortest path. In case of a disconnected network: can have nodes with no paths between them.

Shortest paths are always self-avoiding, and not necessarily unique.

Diameter is the length of the longest geodesic path between any pair of vertices in the network for which a path actually exists.

For a disconnected network diameter is infinite (or we can calculate the diameter for each component).

Gephi – 1

We are experimenting with the software called `gephi`.

This is a shiny tool for analyzing (small) networks (graphs). This software is installed on the computers located in the room where the course takes place.

If you want to install it for yourself, then you need to visit the website:

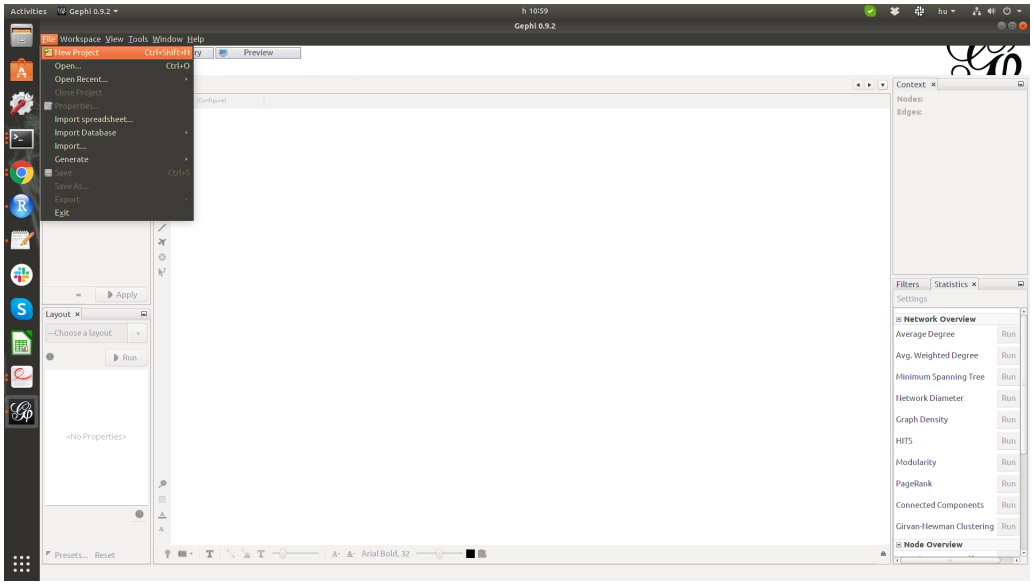
`http://www.gephi.org`

This is a free software; note that it needs Java to be installed.

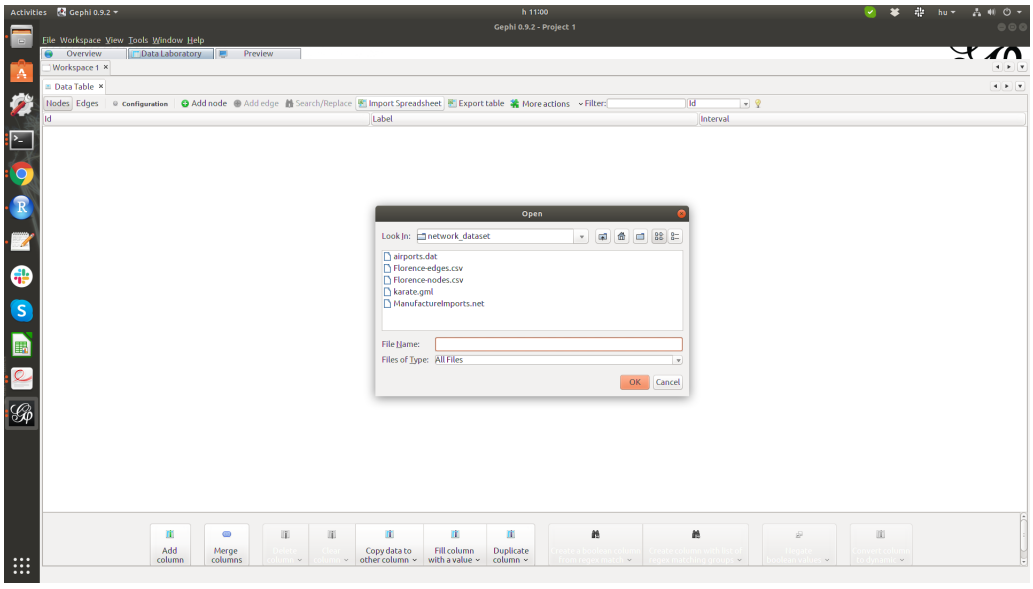
For a start, we are doing some small experiments.

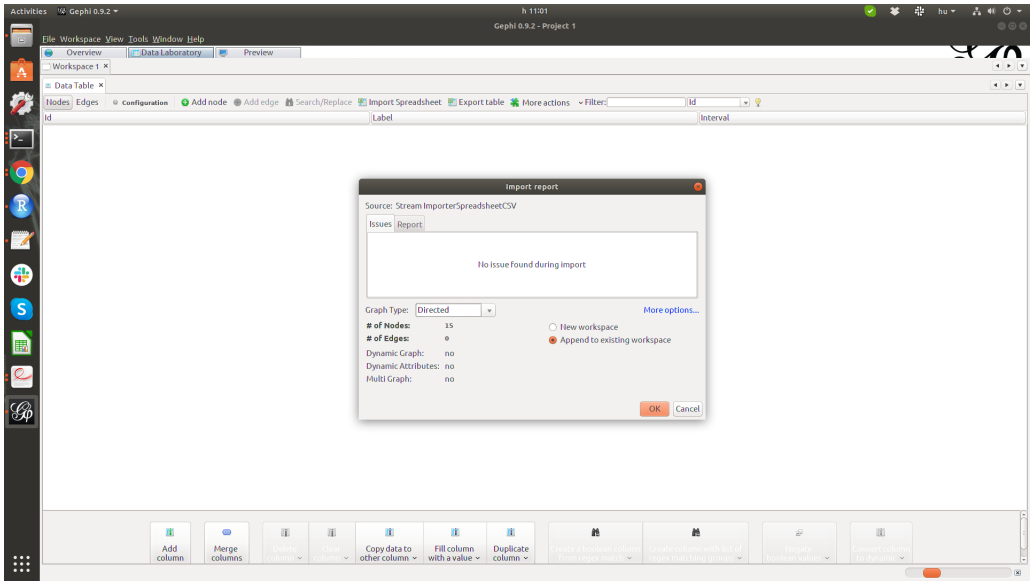
Download the 'Florence' dataset from CooSpace. This dataset represents intermarriages between prominent families in early 15th century Florence (Italy, renaissance).

Start gephi, then File/New Project, then Push the button: Data Laboratory and finally: Import spreadsheet

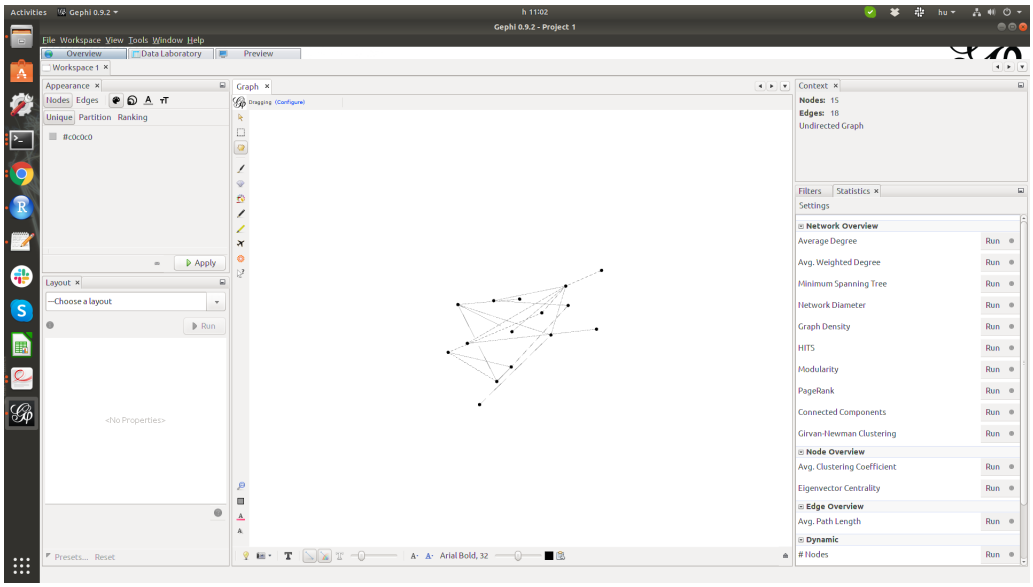


Select first the `Florence-nodes.csv` file and try to load it (selecting in 'Nodes table' in the 'As table' list) Import also the `Florence-edges.csv` file (selecting now the 'Edges table' in the 'As table' list).
Do not forget to select the 'Append to existing workspace' choice button before hitting the OK button.





Now that the graph has been loaded into gephi, we can visualize it by pushing the Overview button.



This should look better, so let's see what we can do for it.

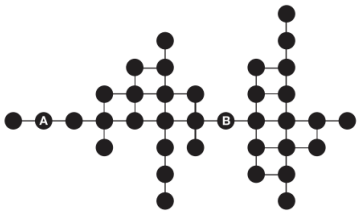
6.2 Centrality

One of the most important concepts in Network Science

"*Which are the most important vertices in the network?*"

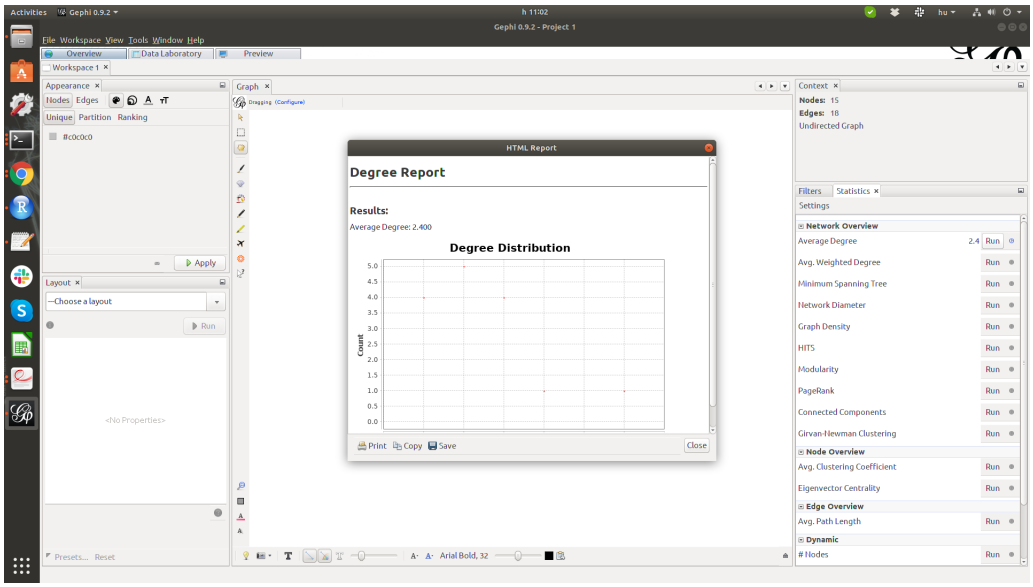
Degree centrality We have several definitions, one of the simplest ones is *degree centrality*, which simply assigns the degree of the nodes to themselves.

On this graph:

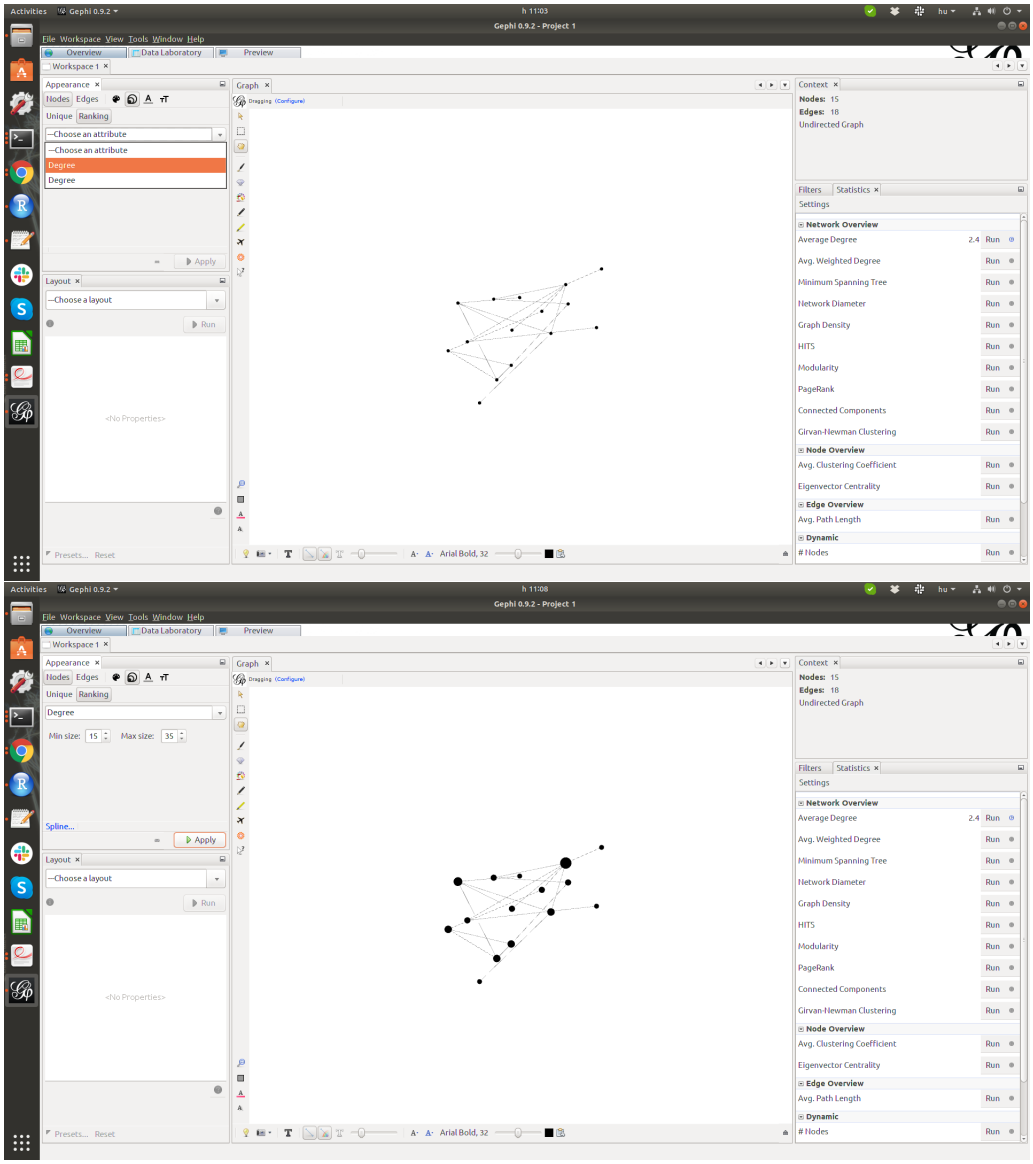


degree of node A and B are the same. We feel that they are not equally important, though.

In gephi it can be calculated by pushing the 'Average Degree' button on the right side of the window:



Once the degree centrality is calculated for the nodes, we can modify on the visualization by assigning these values to the size of the nodes:

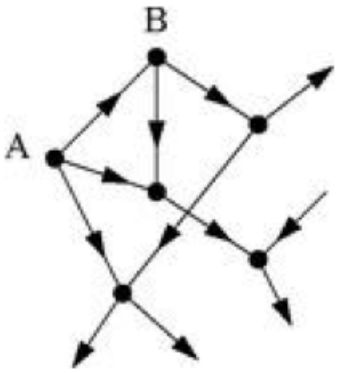


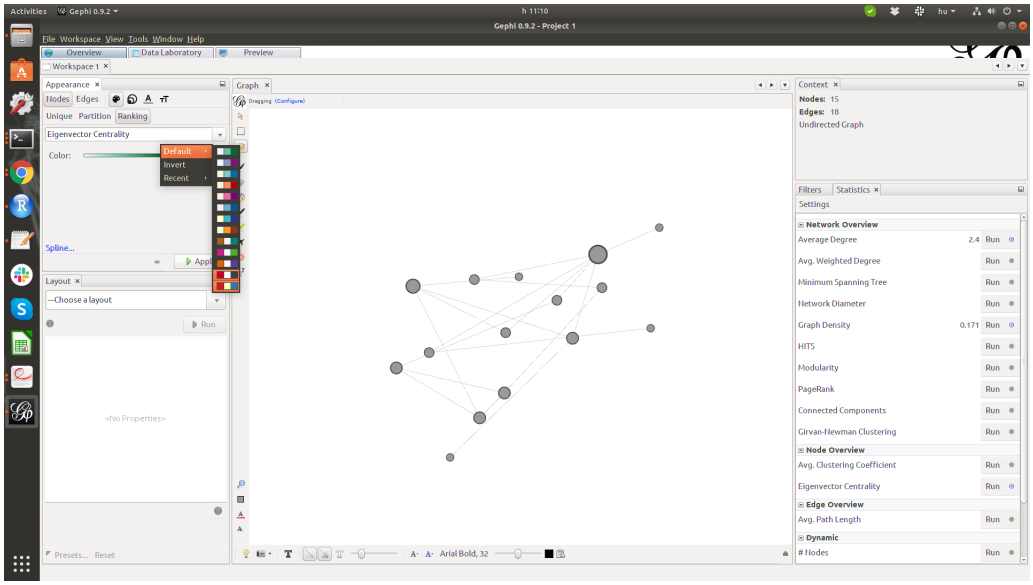
Eigenvector centrality: Importance of a vertex gets increased by having connections to other vertices that are *themselves* important.

We can calculate the Eigenvector centrality in `gephi`.

Eigenvector centrality - directed networks: Think about WWW: given a webpage P which links to another 1000 pages, does this make page P important? Simpler example: EC scores of node A and B are zero: the score of A is zero, no vertices pointing to it (cool) score of B is zero, there is only one node (A) which links to it, and $\text{EC}(A) = 0$ (that is not OK).

In gephi, we can assign the eigenvector centrality (or all the other centrality measures discussed below) to the nodes' coloring:





PageRank This is/was the central part of the search engine of Google. It returns useful answers to queries not because it is better at finding relevant pages, but because it is better at deciding what order to present its findings in

- its perceived accuracy arises because the results at the top of the list of answers it returns are often highly relevant to the query,
- but it is possible and indeed likely that many irrelevant answers also appear on the list, lower down.

Hubs and Authorities (HITS) So far we have considered measures that accord a vertex high centrality if those that point to it have high centrality. However, in some networks it is appropriate also to accord a vertex high centrality if it points to others with high centrality.

- think about a survey article containing lots of references

authorities are nodes that contain useful information on a topic of interest

hubs are nodes that tell us where the best authorities are to be found.

A node can be both, which makes sense only for directed networks.

Closeness centrality: Differs from the previous ones, it measures the mean distance of a vertex to other vertices.

If d_{ij} is the shortest path between node i and j, then

$$l_i = \frac{1}{n} \sum_j d_{ij}$$

is the mean geodesic distance of node i from the other nodes

In social networks: nodes with low l_i value might find that their opinion reach others in the community more quickly than the opinions of someone with higher l_i value.

Taking the inverse of mean geodesic distance leads us to the definition of *closeness centrality*:

$$C_i = \frac{1}{l_i} = \frac{n}{\sum_j d_{ij}}$$

Very natural measure of centrality and used often in social networks

Betweenness centrality: Measures the extent to which a vertex lies on paths between other vertices. More precisely: only the shortest paths count.

Intuition: only the shortest paths play role in the spreading of information

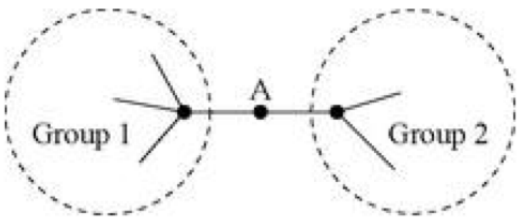
Vertices with high BC value have impact in the network:

- lots of messages pass throughout them,
- their removal would disrupt the communication between the other nodes most.

Let $n_{st}^i = 1$ if vertex i is on the shortest path from node s to node t; and $g_{st} =$ total number of shortest paths between s and t. The general definition of BC:

$$x_i = \sum_{s,t} \frac{n_{st}^i}{g_{st}}$$

Vertex A is called *broker*:



Note that vertex A is on the periphery of both groups and in other respects may not be well-connected

- probably A would not have particularly impressive values for eigenvector or closeness centrality, and its degree centrality is only 2
- nonetheless it might have a lot of influence on the network as a result of its control over the flow of information between others.

Transitivity: Friend of a friend is also my friend.

Partial transitivity can be very useful:

- In many networks, particularly social networks, the fact that u knows v and v knows w does not guarantee that u knows w, but makes it much more likely.
- The friend of my friend is not necessarily my friend, but is far more likely to be my friend than some randomly chosen member of the population.

Clustering coefficient:

$$C = \frac{\text{number of closed paths of length two}}{\text{number of paths of length two}}$$

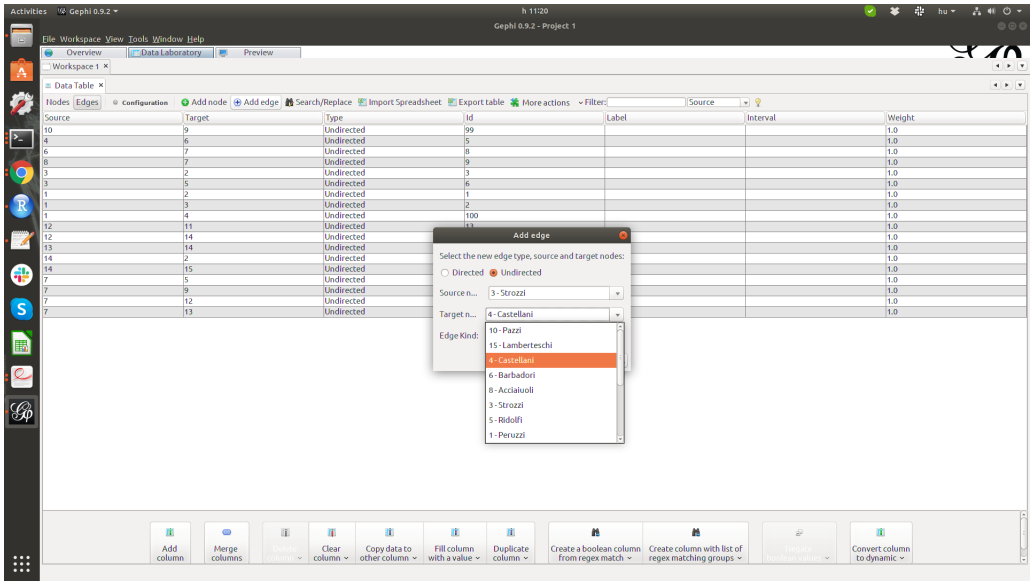
Homophily We love those who love us.

- People have, it appears, a strong tendency to associate with others whom they perceive as being similar to themselves in some way.

Using some math trick, one can find *community structure* in the graph. In `gephi` this can be done with Modularity.

Gephi – 3

The dataset in fact does not contain an edge between the Strozzi and Castellani families. We can add it in the Laboratory page:



Once the network gets changed we need to re-compute all the metrics.

Gephi – 4

The other dataset we are playing with is the file `karate.gml` (download it from CooSpace).

This dataset represents the famous Karate-club analyzed by Zachary for social interactions

- Two nodes are connected if they are friends in the club

Load the file into gephi (Select New Project and then File/Load).

This dataset is interesting because its community structure. It can be revealed by running the Modularity computation and then using the visualization where we assign colors (or size) to the different groups.

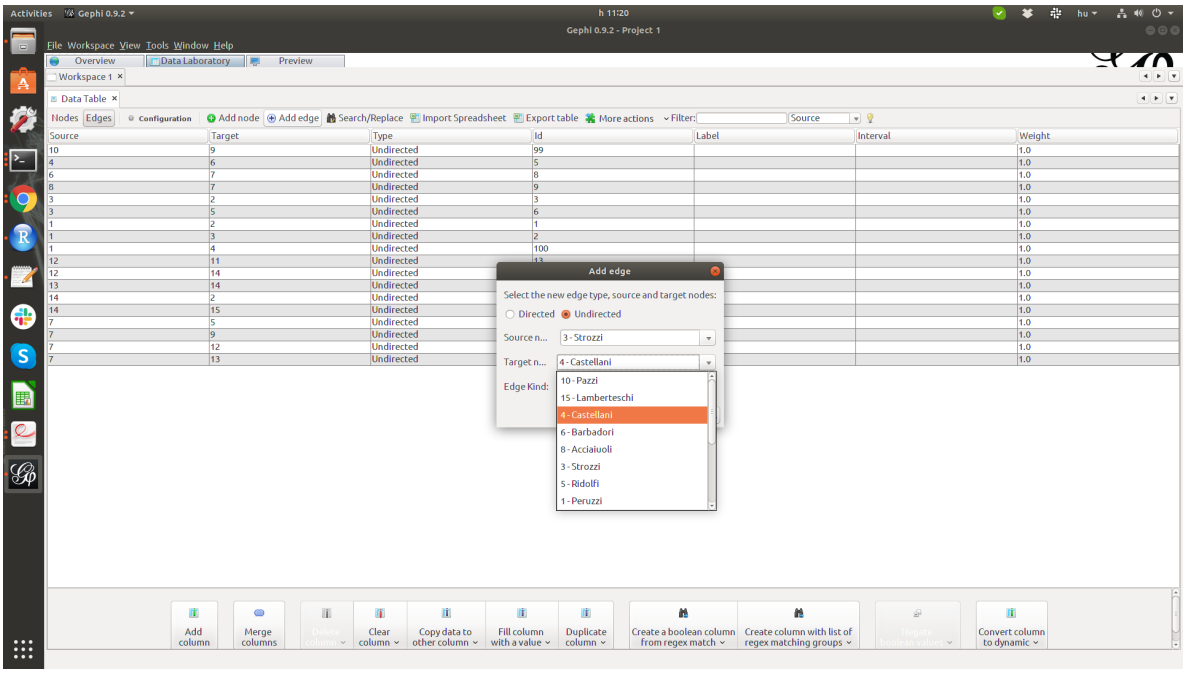
Note that you need to use not the default 1.0 parameter for the Modularity computations, but a bigger number, say 1.5, to obtain two groups.

Gephi – 5

The 3rd dataset is named `intl.gexf` (download it from CooSpace). This dataset is much bigger than the other two. The network represents airports as nodes and flight connections as edges. After loading the dataset we calculate several measures like degree and diameter. For the diameter calculation gephi computes all shortest paths, hence we obtain also the betweenness centrality measure. Computing the number of components, it turns out that there is a large component and several small ones.

- Keep only the biggest one by deleting nodes in the Laboratory part.

Node removal can be done, for example by sorting the nodes according to their component ID. It is also useful to delete low-degree nodes.



Exercises

Karate club

The other dataset we are playing with is the file `karate.gml` (download it from CooSpace).

1. This dataset is representing the famous Karate-club analyzed by Zachary for social interactions

2. Two nodes are connected if they are friends in the club

3. Load the file into gephi (Select New Project and then File/Load).

4. This dataset is interesting because of its community structure. This can be revealed by running the Modularity computation and then using the visualization where we assign colors (or size) to the different groups.

5. Note that you need to use not the default 1.0 parameter for the Modularity computations, but a bigger number, say 1.5, to obtain 2 groups.

Airports and connections

The 3rd dataset is named `intl.gexf` (download it from CooSpace).

1. This dataset is much bigger than the other two.
2. The network represents airports as nodes and flight connections as edges.
3. After loading the dataset we calculate several measures like degree and diameter.
4. For the diameter calculation gephi computes all shortest paths, hence we obtain also the betweenness centrality measure.
5. Computing the number of components, it turns out that there is a large component and several small ones. Keep only the biggest one by deleting nodes in the Laboratory part.
6. Node removal can be done, for example by sorting the nodes according to their component ID.
7. It is also useful to delete low-degree nodes.

Exams

The Business Informatics exam has two parts:

- Closed book part, 55 minutes
- Open book part, 40 minutes

Closed book part

During the closed book part students receive 10 selected questions from the list of possible questions, which are the following[1]:

– What is a model in economics?

– What are the two main approaches we mentioned for (economic) models?

– Give an example for the scientific modelling.

– Information concepts: data, information, knowledge. What are these?

– Give at least 3 characteristics of valuable information (with some details about each).

– Give an example of omitted details in modelling.

- How do we define a graph?
- What is the size of a graph?
- What is the degree of a vertex in a graph?
- What is a weighted graph?
- What is a regular graph?
- What is a complete graph?
- What is a bipartite graph?

[1] the study material for the questions marked with '–' sign are not covered in this handout

- What is a planar graph?
- What is a tree?
- How do you define the shortest path problem in a graph?
- How do you define the travelling salesman problem in a graph?
- What is the minimum spanning tree problem?
- What is the Breadth-first search?
- How can you test bipartiteness of a graph using Breadth-first search?
- What is the Depth-first algorithm?
- How can you test with Depth-first algorithm if a graph has circle?
- What is Linear Programming (LP)?
- What is Integer Linear Programming (ILP)? What is the difference between LP and ILP?
- How can we formalize the shortest path problem on a graph using Linear Programming?
- What is the Critical Path Method (CPM)?
- What does CPM calculate?
- How does CPM represent a project (set of tasks)?
- What are the 4 classes of networks/graphs representing real-world phenomena?
- Give a representative example of Technological Networks.
- Give a representative example of Information Networks.
- Give a representative example of Social Networks.
- Give two examples of network centrality measures.

– Information visualization: what are the so-called Tuftes rules?
– How do you define the lying factor?
– What is data-ink ratio and how can you increase it?
– What is the equal dimensions rule (by Tufte)?
– What is the definition of preattentive properties?
– What are the 4 categories of preattentive properties?
– What is the Gestalt theory?

Open book part

During the open book part students receive (maximum) 3 problems to be solved using the algorithms we studied. They are allowed to use notes, the lecture slides or any other online source. It is however strictly forbidden to use the help of anyone inside or outside the classroom.

The set of problems:

- Simple Linear Programming (formalization and solution with AMPL)
- Shortest paths in a graph
- Minimum spanning tree of a graph
- CPM
- (PERT)

Bibliography

[1] Blázsik, Zoltán Gazdasági folyamatok modellezése Typotech, ISBN 978 963 279 493 8, 2011

[2] Cormen, Thomas H., *et al.* Introduction to algorithms. MIT press, 2009.

[3] Michael Fried. CS313: Data structures Course. `https://venus.cs.qc.cuny.edu/~mfried/cs313/`

[4] Gass, Saul and Fu, Michael Dijkstra's Algorithm. Encyclopedia of Operations Research and Management Science. Springer, 2013.

[5] Jesse Santiago and Desirae Magallon. CRITICAL PATH METHOD Princeton CEE 320 VDC SEMINAR, 2009

[6] Kruskal, J.B. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the AMS 7 (February 1956) 48–50

[7] Levy, Ferdinand K., Gerald L. Thompson, and Jerome D. Wiest. The ABCs of the critical path method. Harvard University, Graduate School of Business Administration, 1963

[8] Newman, Mark. Networks. Oxford University Press, 2018

[9] Prim, R.C. Shortest connection networks and some generalizations. Bell Syst. Tech. J. 36, 1389–1401. 1957.