Network Flow Problems

Stefano Gualandi
Università di Pavia, Dipartimento di Matematica

email: stefano.gualandi@unipv.it
twitter: @famo2spaghi
blog: http://stegua.github.com

Introduction

The program of this course on **Operations Research** is:

1. Modeling Optimization Problems

   `https://www.informs.org/Recognizing-Excellence/INFORMS-Prizes/Franz-Edelman-Award`

2. Network Flows Models and Algorithms
3. Linear Programming (LP)
4. Integer Linear Programming (ILP)

In this lecture

Overview Network Flow Problems: (1) Shortest Path, (2) Maximum Flow, and (3) Minimum Cost Flow.

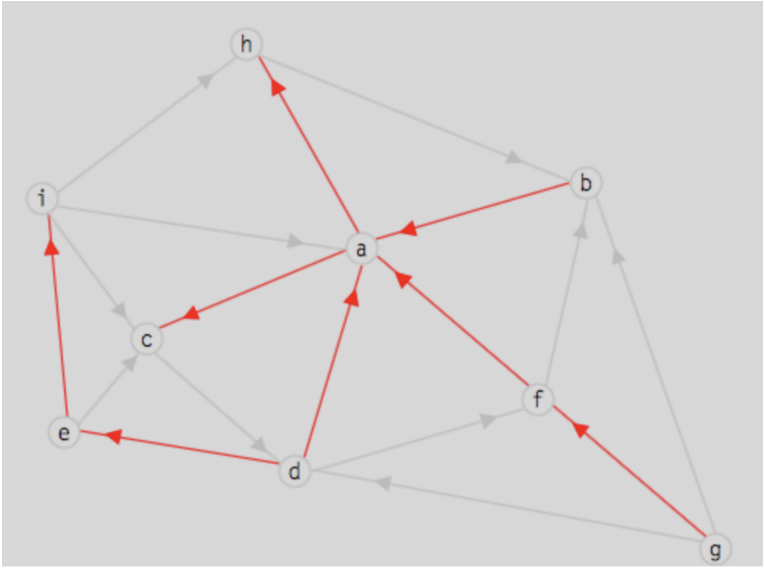
Network Flow Problems

Given $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{x} \in \mathbb{R}^n$, we consider the LP problem:

$$\begin{aligned} \min \quad & \boldsymbol{c}^\intercal \boldsymbol{x} && (1) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (2) \\ & \boldsymbol{x} \geq 0, && (3) \end{aligned}$$

Depending on the structure of the coefficient matrix $\boldsymbol{A}$, the r.h.s. vector $\boldsymbol{b}$, and the cost vector $\boldsymbol{c}$, we can have different type of LP problems, which can be solved by different types of algorithms.

Many LP problems can be viewed as a problem of minimizing the cost of moving quantity of materials along a network, as *street*, *train*, *electrical*, and *communication* networks.
Also other applications can be **modeled** as network flow problems: *facility locations*, *resource management*, and *financial planning*.



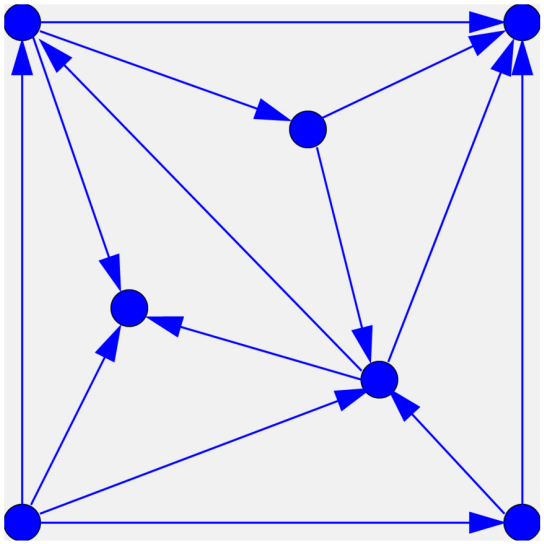
Networks

A network is defined by

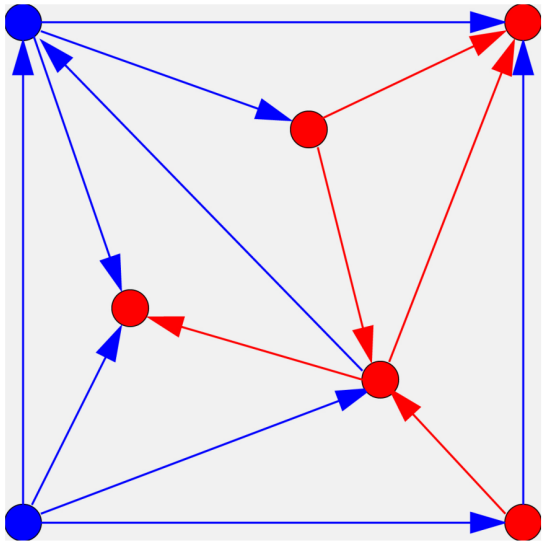
- A set of m **nodes** N
- A set of **arcs** A which is a subset of all possible ordered pairs (i, j) with $i, j \in N$, where i is the **tail** and j is the **head**:

$$A \subseteq \{(i, j) \mid i, j \in N, i \neq j\}.$$

The pair $G = (N, A)$ is called a network (or sometime *digraph* or *graph*).



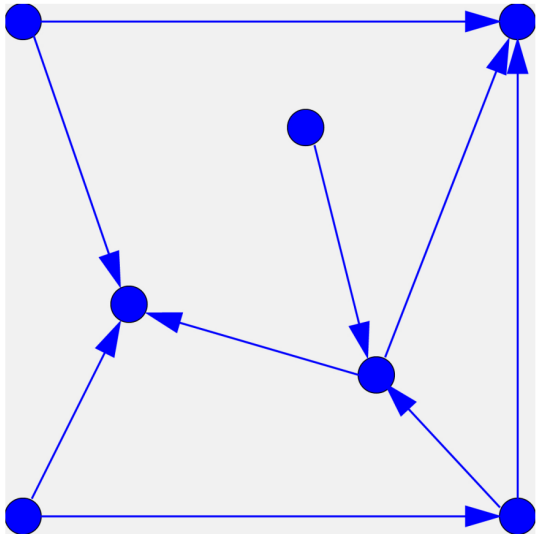
Network



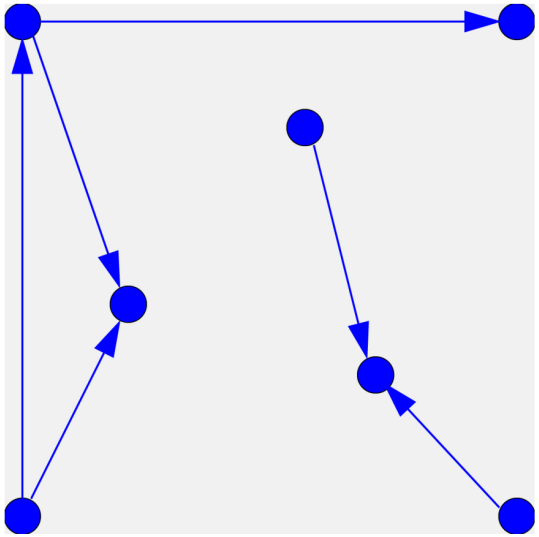
Subnetwork

Subnetwork

Given a network $G = (N, A)$, a network $H = (N', A')$ is a **subnetwork** of G if $N' \subseteq N$ and $A' \subset A$.



Connected



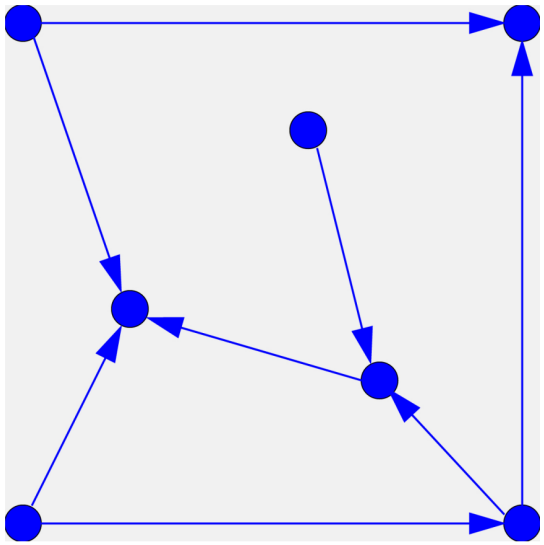
Disconnected

Connected Network

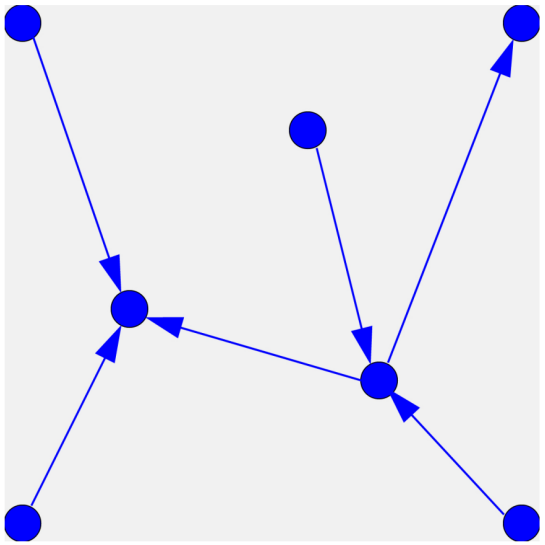
Given a network $G = (N, A)$, an ordered list of nodes $(n_1, n_2, \ldots, n_k)$, with $n_i \in N$, is called a **path** if each adjacent pair of nodes in the list is connected by an arc $(i, j) \in A$. Here, we can have $(n_i, n_{i+1}) \in A$ or $(n_{i+1}, n_i) \in A$ (we do not consider arc orientations), but a pair of nodes $\{i, j\}$ can appear only once in the path.

A network is called **connected** if there is a path connecting every pair of nodes.

REMARK: in this course, we work with connected networks.



Cyclic

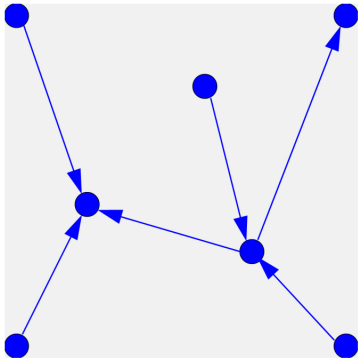


Acyclic

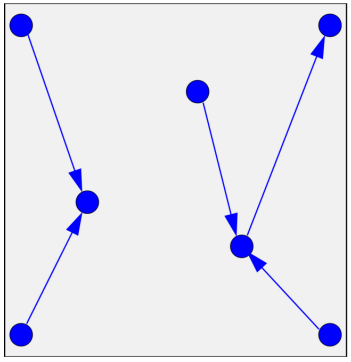
Cyclic and Acyclic Network

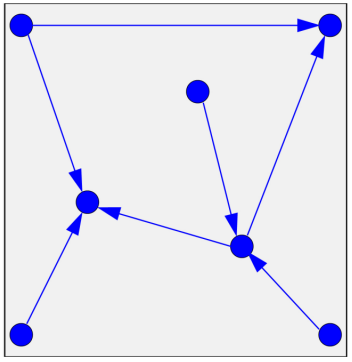
A **cycle** is a path in which the last node coincides with the first node.

A network is said **acyclic** if does not contain any cycles.



Tree = Connected + Acyclic

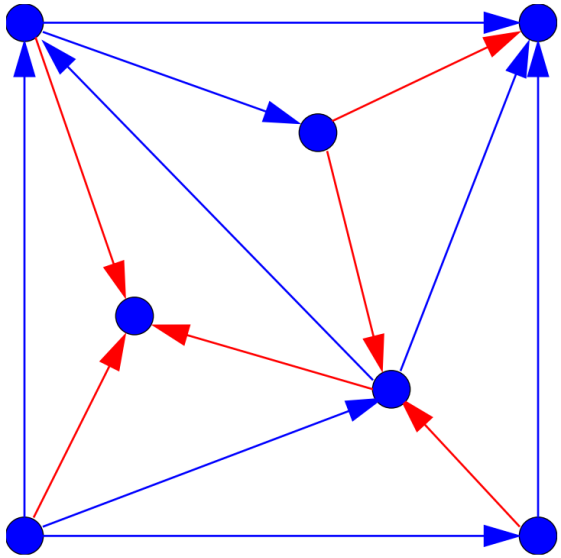




Not Trees

Directed Trees

A network is a **tree** if it is connected and acyclic.

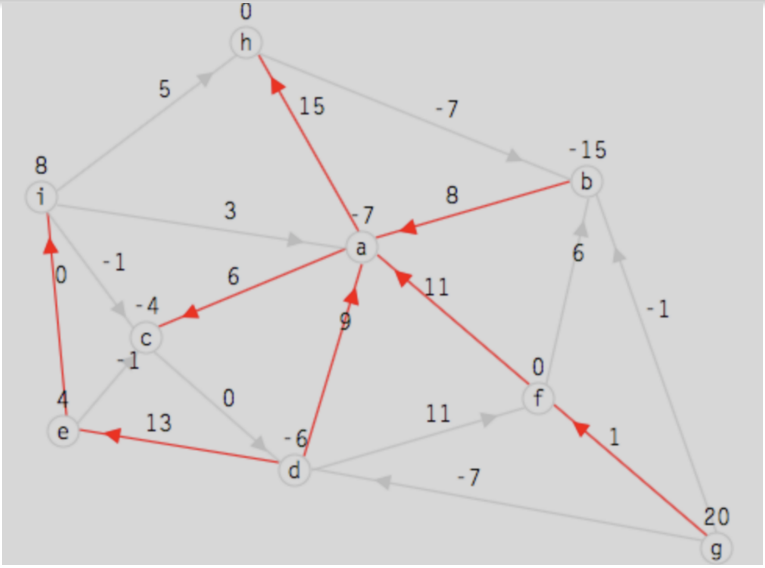


Spanning Tree – A tree touching every node

(No need to recolor the nodes.)

Spanning Tree

Given $G = (N, A)$, a subnetwork $H = (N', A')$ is called a **spanning tree** if it is a tree and $N = N'$.



Network Flow Problem

For each node $i \in N$, let b_i denote the amount of material being **supplied** at node i, with the convention that negative values are **demands**. A Network Flow Problem asks to move materials from supply toward demand nodes. We make the assumption

$$\sum_{i \in N} b_i = 0.$$

For each arc $(i, j) \in A$, we have a cost c_{ij} for shipping a unit of material from i to j.

Decision Variables

For each $(i, j) \in A$, we have a non-negative variable $x_{ij} \geq 0$ that is the amount of material moving from i to j along arc (i, j).

Objective Function

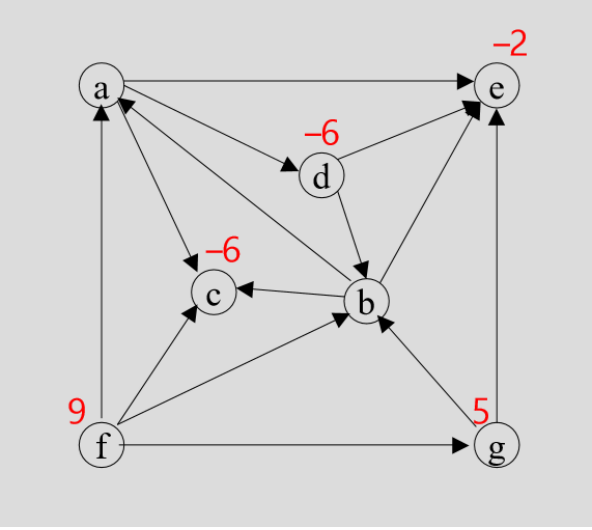
We want to minimize the total cost of moving material

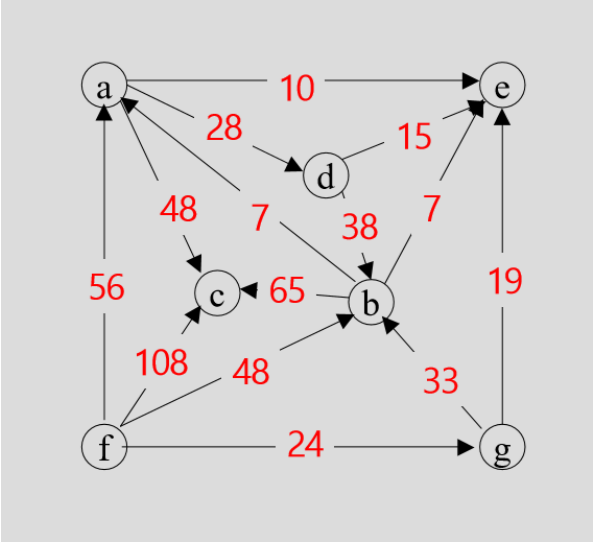
$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

Constraints

We must guarantee that total flow entering an arc is equal to the total flow leaving the node, i.e., we have flow balance constraints for each node

$$\sum_{(j,i)\in A} x_{ji} - \sum_{(i,j)\in A} x_{ij} = -b_i, \quad \forall i \in N.$$





$$x^T = \begin{bmatrix} x_{ac} & x_{ad} & x_{ae} & x_{ba} & x_{bc} & x_{be} & x_{db} & x_{de} & x_{fa} & x_{fb} & x_{fc} & x_{fg} & x_{gb} & x_{ge} \end{bmatrix},$$

$$A = \begin{bmatrix} -1 & -1 & -1 & 1 & & & & & 1 & & & & & \\ & & & -1 & -1 & -1 & 1 & & & 1 & & & 1 & \\ 1 & & & & 1 & & & & & & 1 & & & \\ & 1 & & & & & -1 & -1 & & & & & & \\ & & 1 & & & 1 & & 1 & & & & & & 1 \\ & & & & & & & & -1 & -1 & -1 & -1 & & \\ & & & & & & & & & & & 1 & -1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ -6 \\ -6 \\ -2 \\ 9 \\ 5 \end{bmatrix},$$

$$c^T = \begin{bmatrix} 48 & 28 & 10 & 7 & 65 & 7 & 38 & 15 & 56 & 48 & 108 & 24 & 33 & 19 \end{bmatrix}.$$

Network Flow Problems

The matrix $\boldsymbol{A}$ in this case is called the node-arc incidence matrix.

The Transportation Problem

When we transport goods along a network to model *shipment* problem, we have a Transshipment Problem. A special case is when the nodes in N are partitioned into two sets S and D such that

$$N = S \cup D, \qquad S \cap D = \emptyset,$$

that is every arc in A has its tail in S and head in D, and we have:

$$\begin{aligned} b_i &\geq 0, \quad \forall i \in S, \\ b_i &\leq 0, \quad \forall i \in D. \end{aligned}$$

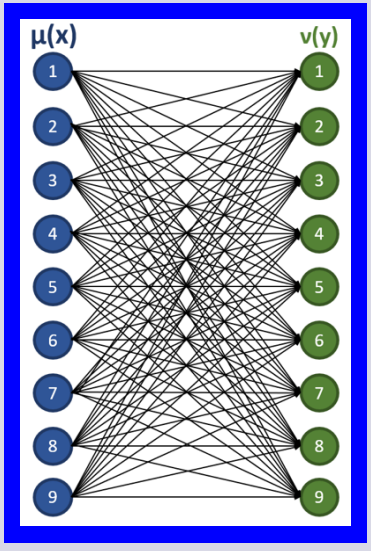
The nodes in S are the supply nodes, those in D the demands nodes. The network is called a Bipartite Graph.

A Network Flow Problem defined on a bipartite graph is called a Transportation Problem.

The Hitchcock Transportation Problem

If every supply node is connected to every demand node, the problem is called a Hitchcock Transportation Problem.

If denote by r_i the supply node, and by s_j the demand nodes, we can write the problem as



$$\min \quad \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij} \tag{4}$$

$$\text{s.t.} \quad \sum_{j \in D} x_{ij} = r_i, \qquad \forall i \in S \tag{5}$$

$$\sum_{i \in S} x_{ij} = s_j, \qquad \forall j \in D \tag{6}$$

$$x_{ij} \geq 0. \tag{7}$$

.

In this formulation, we still have the requirement on the input data: $\sum_{i \in S} r_i = \sum_{j \in D} s_j$.

The Hitchcock Transportation Problem

If every supply node is connected to every demand node, the problem is called a Hitchcock Transportation Problem.

If denote by r_i the supply node, and by s_j the demand nodes, we can write the problem as

$$\min \quad \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij} \tag{8}$$

$$\text{s.t.} \quad \sum_{j \in D} x_{ij} \leq r_i, \qquad \forall i \in S \tag{9}$$

$$\sum_{i \in S} x_{ij} \geq s_j, \qquad \forall j \in D \tag{10}$$

$$x_{ij} \geq 0. \tag{11}$$

In this formulation, we can relax the requirement to $\sum_{i \in S} r_i \geq \sum_{j \in D} s_j$.

Tolstoj's Transportation Problem

	Arkhangelsk	Yaroslavl'	Murom	Balakhonikha	Dzerzhinsk	Kishert'	Sverdlovsk	Artemovsk	Iledzhk	Dekonskaya	*Demand*
Agryz				709	1064	693					2
Aleksandrov					397			1180			4
Almaznaya								81		65	1.5
Alchevskaya								106		114	4
Baku								1554		1563	10
Barybino								985		968	2
Berendeevo		135			430						10
Bilimbai						200	59				1
Bobrinskaya								655		663	10
Bologoe		389						1398			1
Verkhov'e								678		661	1
Volovo								757		740	3
Vologda	634					1236					2
Voskresensk				427				1022		1005	1
V.Volochek		434						1353		1343	5
Galich	815	224				1056					0.5
Goroblagodatskaya						434	196				0.5
Zhlobin								882		890	8
Zverevo								227		235	5
Ivanovo					259						6
Inza				380	735					1272	2
Kagan								2445	2379		0.5
Kasimov			0								1
Kinel'				752		1208			454	1447	2
Kovylkino				355						1213	2
Kyshtym						421	159				3
Leningrad	1237	709						1667		1675	55
Likino			223		328						15
Liski								443		426	1
Lyuberdzhy			268		411					1074	1
Magnitogorskaya						932	678		818		1
Mauk						398	136				5
Moskva			288	378	405			1030		1022	141
Navashino			12	78							2
Nizhegol'								333		316	1
Nerekhta		50			349						5
Nechaevskaya			92								0.5
N.-Novgorod					32						25
Omsk						1159	904		1746		5
Orenburg									76		1.5
Penza				411				1040	883	1023	7
Perm'	1749					121					1
Petrozavodsk	1394										1
Poltoradzhk								1739	3085	1748	4
Pskov								1497		1505	10
Rostov/Don								287		296	20
Rostov/Yarosl		56			454						2
Rtishchevo								880		863	1
Savelovo		325						1206		1196	5
Samara				711					495	1406	7
San-Donato						416	157				1
Saratov								1072		1055	15
Sasovo				504				1096		1079	1
Slavyanoserbsk								119		115	1.1
Sonkovo		193						1337			0.5
Stalingrad								624		607	15.4
St.Russa		558						1507		1515	5
Tambov								783		766	4
Tashkent								3051	1775		3
Tula								840		848	8
Tyumen'						584	329				6
Khar'kov								251		259	60
Chelyabinsk						511	257		949		2
Chishmy				1123		773			889		0.5
Shchigry								566		549	4
Yudino				403	757	999					0.5
Yama								44		52	5
Yasinovataya								85		93	6
Supply	5	11.5	8.5	12	100	12	15	314	10	55	543

Data:

- $I = \{1, \ldots, n\}$ set of sources
- $J = \{1, \ldots, m\}$ set of destinations
- $C \in \mathbb{R}^{m \times n}$ with c_{ij} distance in km from source i to destination j
- $\boldsymbol{d} \in \mathbb{R}^m$ demands in tons
- $\boldsymbol{s} \in \mathbb{R}^n$ supplies in tons

Decision Variables:

$x_{ij} \geq 0$ quantity of material from i to j

Objective Function:

$$\min \sum_{ij \in I \times J} c_{ij} x_{ij}$$

Constraints:

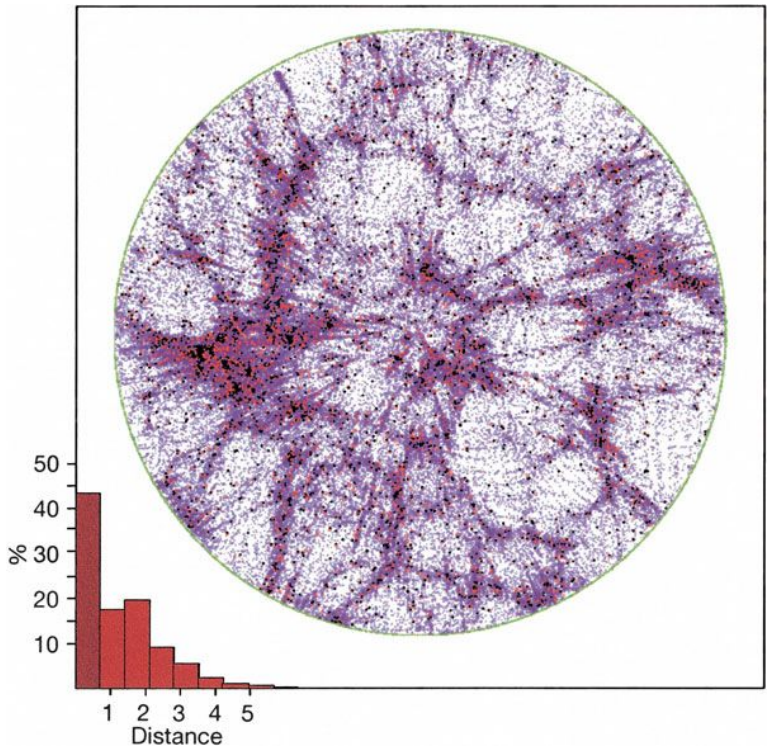
$$\sum_{j \in J} x_{ij} = d_i, \forall i \in I$$
$$\sum_{i \in I} x_{ij} = s_j, \forall j \in J$$

Can we reconstruct the initial condition of the Universe?



Question: "Can one follow back in time to initial locations the highly structured present distribution of mass in the Universe, as mapped by redshift catalogues of galaxies?"

Can we reconstruct the initial condition of the Universe?



Considering that the initial mass distribution was quasi-uniform, about 14 billion years ago, the initial density field makes reconstruction a well-posed instance of what is called the Optimal Mass Transportation Problem.

Frisch, U., Matarrese, S., Mohayaee, R. and Sobolevski, A., 2002. *A reconstruction of the initial conditions of the universe by optimal mass transportation*. Nature, 417(6886), pp. 260–262.

Can we reconstruct the initial condition of the Universe?



Bruno Levy @BrunoLevy01 · 9h

Replying to @Patapom2

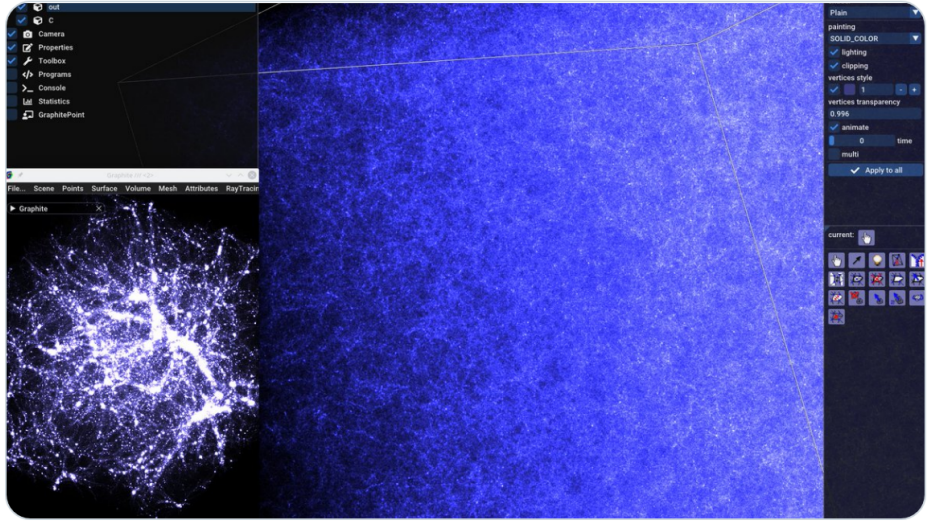
What it is doing is *inverting* the evolution, from current state to initial condition. To test the method, I'm using simulations made by astrophys friends, for which both are known.
The inverted eqn corresponds to OT:
arxiv.org/abs/astro-ph/0...
Here we use more efficient numerics.

5



Bruno Levy @BrunoLevy01 · 18h

Stuck at home, but tonight I'm flying far far away !
Early universe reconstruction with optimal transport,
16M points in 1h15min, using a V100 for linear algebra, and 86M points in 7h20min (my V100 only has 16 Gb, did not fit in there, I should buy a bigger one).



3 5 51



The Assignment Problem

Given a set S of m people, a set D of m tasks, and for each $i \in S, j \in D$a cost c_{ij} associated with assigning person i to task j, the assignment problem is to assign each person to one and only one task, so that all tasks are covered and the overall cost of the assignment in minimized.

Decision Variables

We introduce 0-1 variables for pair $(i, j) \in S \times D$:

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned to task } j. \\ 0 & \textit{otherwise} \end{cases}$$

Objective Function

$$\min \quad \sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij}.$$

Constraints

The constraint that each person is assigned to one task

$$\sum_{j \in D} x_{ij} = 1, \quad \forall i \in S.$$

The constraint that each task is covered by one person

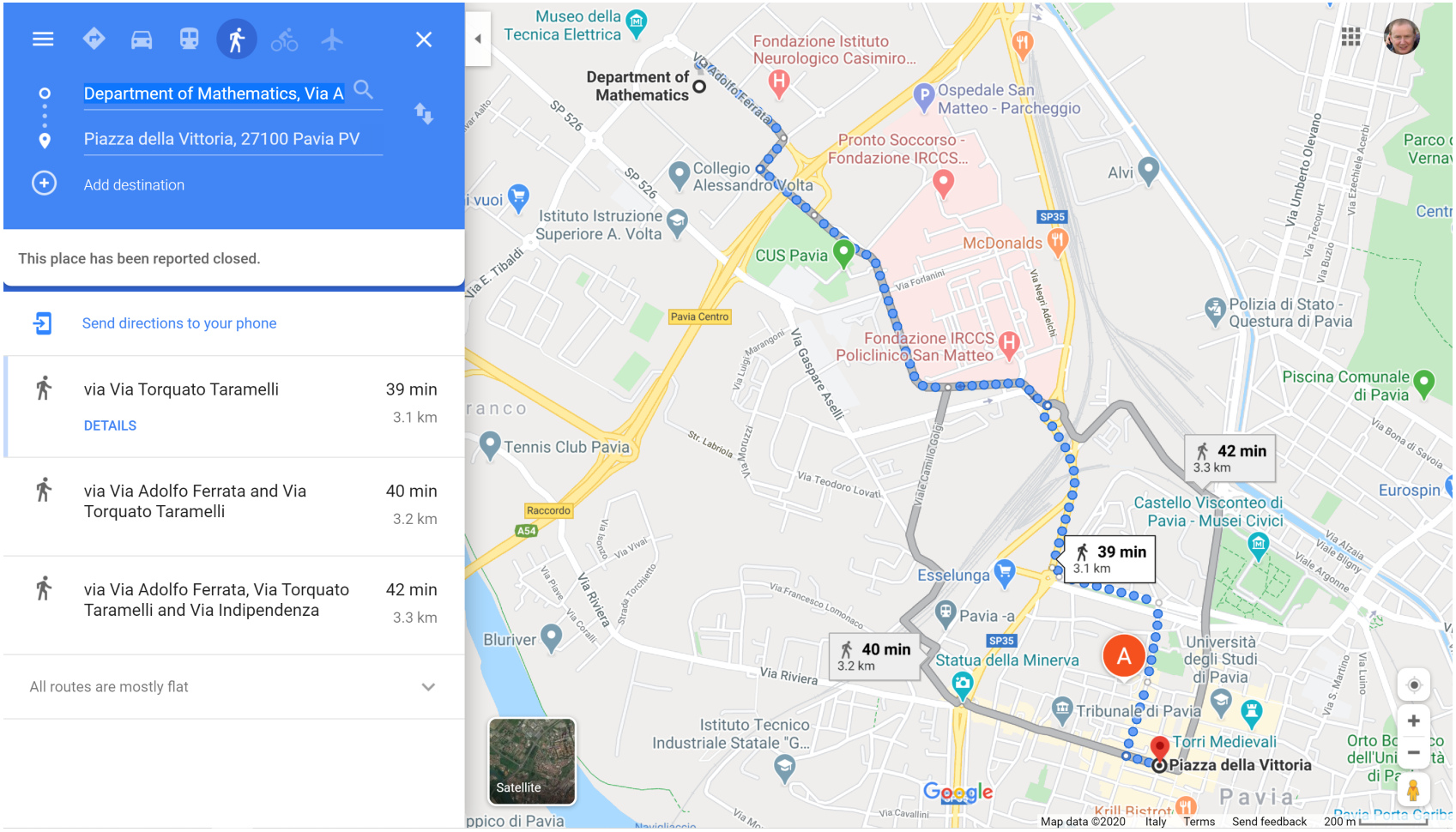
$$\sum_{i \in S} x_{ij} = 1, \quad \forall j \in D.$$

QUESTION: Which is the relation between the Hitchcock Transportation problem and the Assignment Problem?

Except for the integrality constraint, the assignment problem is a special case of the transportation problem.

The Hitchcock transportation problem is the LP relaxation of the assignment problem:

- Any feasible solution of the assignment problem is a feasible solution of the transportation.
- Any integral feasible solution to the LP relaxation is a feasible solution to the assignment problem.
- If we use the simplex algorithm to solve the LP relaxation, we get an integral solution (but not if we use an interior point algorithm)
- Remember: $z_{LP} \leq z_{IP}$


Department of Mathematics, Via A
Piazza della Vittoria, 27100 Pavia PV
Add destination
This place has been reported closed.
Send directions to your phone
via Via Torquato Taramelli
39 min
3.1 km
DETAILS
via Via Adolfo Ferrata and Via Torquato Taramelli
40 min
3.2 km
via Via Adolfo Ferrata, Via Torquato Taramelli and Via Indipendenza
42 min
3.3 km
All routes are mostly flat
Department of Mathematics
Piazza della Vittoria
42 min
3.3 km
39 min
3.1 km
40 min
3.2 km
Satellite
Map data ©2020
Italy
Terms
Send feedback
200 m

The Shortest Path Problem

Given a network $G = (N, A)$, the shortest path problem is to find the shortest directed path from one specific node (the source) to another node (the destination). Indeed, we assume that is given the length c_{ij} of every arc in A.

Since to determine the shortest path for a pair of nodes, we need to find the shortest path o possible every node to the destination, the shortest path problem is usually defined as:

> *The shortest path problem is the problem of finding the shortest path from every node in N to a specific node $r \in N$. The destination node r is called the root node.*

The Shortest Path Problem

Set a unit of supply to every node $i \in N$ except the root node r, which has a demand equal to $m - 1$:

$$b_i = \begin{cases} +1 & \text{if } i \neq r, \\ -m+1 & \text{if } i = r. \end{cases}$$

The cost c_{ij} is equal to the length of the arc (i, j).

CONSEQUENCE: Any Network Flow algorithm can solve the shortest path problem.

REMARK: The shortest path problem structure permits to develop more efficient algorithms.

Labeling Algorithms

Let us denote the distance from node i to the root r by the *label* v_i.
There exists two main type of algorithms:

1. Label-Correcting: the labels represent *approximate* distance to the root and they are iteratively improved until they reached the optimal distances.
2. Label-Setting: the labels are set in systematically order in such a way that at they they all represent the shortest distance to the root.

Labeling-Correcting Algorithm

For a labeling correcting algorithm, we need a system of equations that characterize the optimal solution.

First, clearly $v_r = 0$.

From a node i, we should select an arc (i, j) that minimizes the distance to r, that is

$$v_i = \min\{c_{ij} + v_j \mid (i,j) \in A\}, \quad \forall i \in N \setminus \{r\}.$$

This is a *Bellman's equation* and the labels v_i plays the role of *value functions*. The argument used to derive this formula is called principle of dynamic programming.

From Bellman's equation, we can identify the arcs on a shortest path to the root:

$$\mathcal{T} = \{(i,j) \in A \mid v_i = c_{ij} + v_j\}.$$

QUESTION: Is this set a tree? **TODO: Example**

Method of Successive Approximation

Initialize the labels:

$$v_i^{(0)} = \begin{cases} 0 & i = r, \\ \infty & i \neq r. \end{cases}$$

Apply the Bellman's equation at iteration k:

$$v_i^{(k+1)} = \begin{cases} 0 & i = r, \\ \min\{c_{ij} + v_j^{(k)} \mid (i,j) \in A\} & i \neq r. \end{cases}$$

Algorithm 1: Labeling-Correcting Algorithm

Data: Network $G = (N, A)$, cost vector $\boldsymbol{c}$, root r
Result: v_i optimal labels (function value)

$v_r \leftarrow 0$;
$p_r \leftarrow r$;
for $i \in N \setminus \{r\}$ **do**
    $v_i \leftarrow \infty$;
while $\exists (i,j) \in A \mid v_i > c_{ij} + v_j$ **do**
    $v_i \leftarrow c_{ij} + v_j$;
    $p_j \leftarrow i$;

Efficiency

The Labeling-Correcting algorithm is guaranteed to stop in no more than m iterations (once per node).

Note that $v_i^{(k)}$ it is the length of the shortest path from i to r that has k or fewer arcs in the path.

Hence, the algorithm cannot take more than m iterations, since every shortest path can visit every node at most once.

At each iteration we have to visit every arc, than the worst-case complexity of the algorithm is $O(mn)$.

Algorithm 2: Modified Labeling-Correcting Algorithm

Data: Network $G = (N, A)$, cost vector $\boldsymbol{c}$, root r
Result: v_i optimal labels (function value)

```
1  v_r ← 0;
2  p_r ← r;
3  LIST := {r};
4  for i ∈ N \ {r} do
5      v_i ← ∞;
6  while LIST ≠ ∅ do
7      remove an element i from LIST;
8      for (i, j) ∈ A do
9          if v_i > c_ij + v_j then
10             v_i ← c_ij + v_j;
11             p_j ← i;
```

Labeling-Setting Algorithm

An example of label-setting algorithm is the Dijkstra's algorithm.

The algorithm use a data structure $\mathcal{F}$ of *finished* nodes, which at the begging is empty. Later, the algorithm add once at the time the node to $\mathcal{F}$: it *sets* the correct label v_i one at the time.

Algorithm 3: Labeling-Setting: Dijkstra's Algorithm

Data: Network $G = (N, A)$, cost vector $\boldsymbol{c}$, root r
Result: v_i optimal labels (function value)

```
1  F ← ∅;
2  v_r ← 0;
3  while F^c ≠ ∅ do
4      j ← arg min{v_k | k ∉ F};
5      F ← F ∪ {j};
6      for (i, j) ∈ A and i ∉ F do
7          if v_i > c_ij + v_j then
8              v_i ← c_ij + v_j;
9              p_j ← i;
```

DNA Sequence Alignment

The DNA can be viewed as a string based on the alphabet $\Sigma = \{A, C, G, T\}$. Given two sequences of DNA denoted by $B = b_1 b_2 \dots b_p$ and $D = d_1 d_2 \dots d_q$, we are interested in measuring their similarity.

A way to measure the (dis-)similarity is to determine the minimum cost required to transform the sequence B into D.

To transform B into D we can

1. Insert an element in B at a cost of α units.
2. Delete an element from B at a cost of β units.
3. Mutate an element b_i into an element d_j at a cost $g(b_i, d_j)$.

There many ways to transform B into D and it is not trivial to identify the minimum cost transformation. However, we can applied dynamic programming and formulated this problem as a shortest path problem on an appropriately defined network.

Example of DNA Alignment

Consider $B = AGTT$ and $D = CTAGC$. One possible way to transform B into D is to delete one T from B, and adding two elements at the begging, giving the sequence:

$$B = AGTT \quad \Rightarrow \quad B' = \circledast \circledast AGT$$

We then align B' with D and mutate the element T into C, so that the two sequence become identical.

$$\begin{array}{rccc} B' = & \circledast \circledast AGT & & CTAGC \\ D = & CTAGC & & CTAGC \end{array}$$

The overall cost is

$$\beta + 2\alpha + g(T, C)$$

DNA Alignment: Dynamic Programming

Let $f(i,j)$ denote the minimum cost of transforming the sub-sequence $b_1 b_2 \ldots b_i$ into the sub-sequence $d_1 d_2 \ldots d_j$.

The value $f(p,q)$ is the minimum cost of transforming B into D, and it is computed by first computing $f(i,j)$ for all $i = 0, 1, \ldots, p$ and forall $j = 0, 1, \ldots, q$.

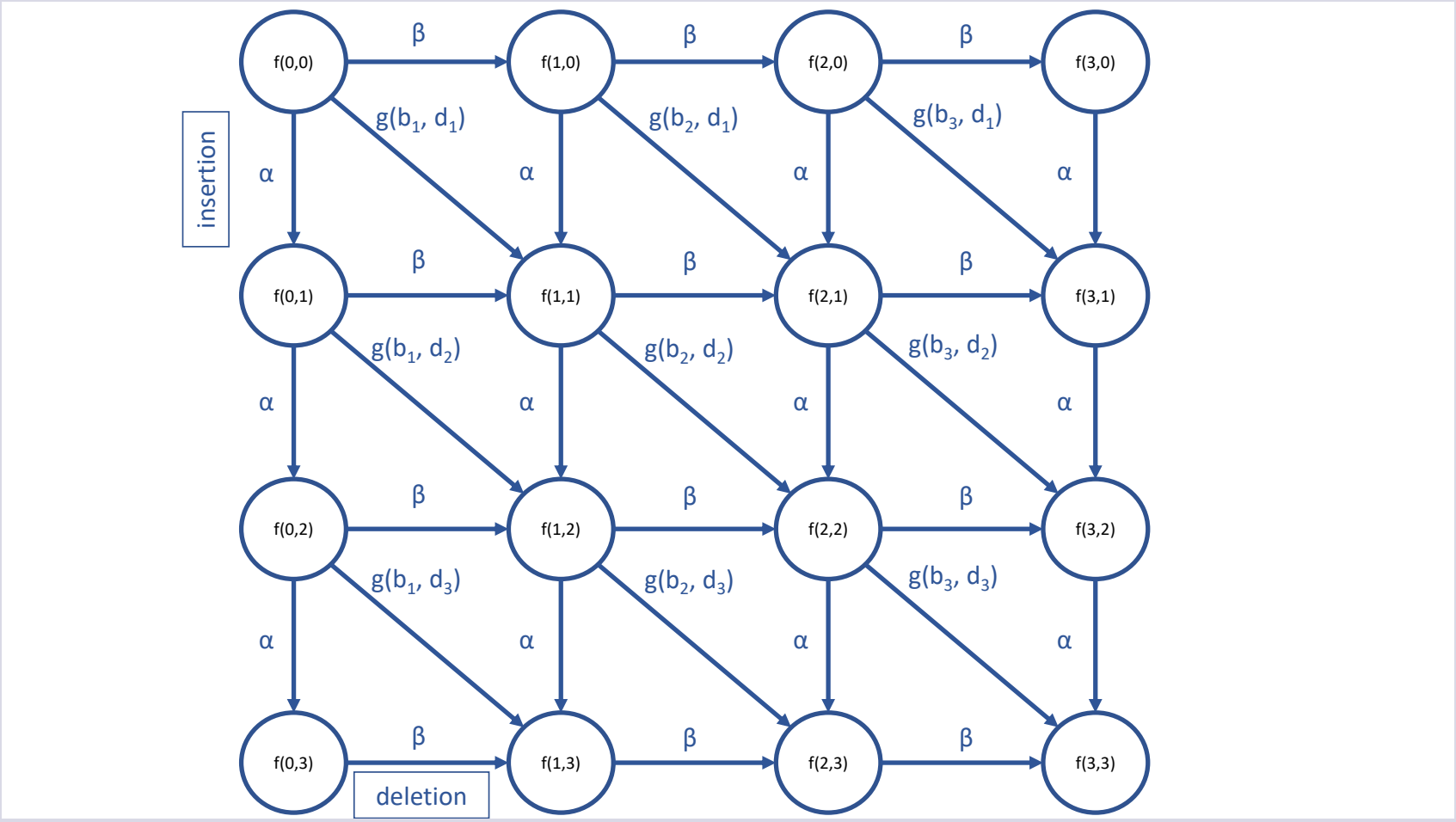
The intermediate $f(i,j)$ are computed as:

- $f(i,0) = \beta i$ for all i
- $f(0,j) = \alpha j$ for all j
- $f(i,j) =$

$$\min \begin{cases} f(i-1,j-1) + g(b_i, d_j) & \text{case: } b_i, d_j \text{ aligned} \\ f(i,j-1) + \alpha & \text{case: } b_i \text{ is not aligned with } d_j \\ f(i-1,j) + \beta & \text{case: } B' \text{ do not contain } b_i. \end{cases}$$

The shortest-path formulation is given in the next slide.

DNA Alignment: Shortest Path



Let $f(i,j)$ denote the minimum cost of transforming the sub-sequence $b_1 b_2 \ldots b_i$ into the sub-sequence $d_1 d_2 \ldots d_j$.

References for Mathematical Modeling

- Chap. 14.1 and 15 in Vanderbei, R.J., 2015. Linear programming. Heidelberg: Springer.
- Chap. 4, 5, 19.4 in Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., 1988. Network flows.