CE 272
Traffic Network Equilibrium

Lecture 5
Notation and Shortest Path Algorithms

Previously on Traffic Network Equilibrium...

Optimization problems typically have the following three components:

1. Objective function
2. Decision variables
3. Constraints

While the first two are present in all optimization models, it is not necessary to have constraints. Problems without constraints are called *unconstrained problems*.

Previously on Traffic Network Equilibrium...

Proposition (Sufficient KKT Conditions)

Suppose f, g_i, and h_i are all differentiable and convex. Then, any $\bar{\mathbf{x}}$ and $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}})$ that satisfy the following KKT conditions are optimal to the primal and dual and the duality gap is 0.

$$g_i(\bar{\mathbf{x}}) \leq 0 \, \forall \, i = 1, \ldots, l$$

$$h_i(\bar{\mathbf{x}}) = 0 \, \forall \, i = 1, \ldots, m$$

$$\bar{\boldsymbol{\lambda}} \geq \mathbf{0}$$

$$\bar{\lambda}_i g_i(\bar{\mathbf{x}}) = 0 \, \forall \, i = 1, \ldots, m$$

$$\nabla_{\mathbf{x}} f(\bar{\mathbf{x}}) + \sum_{i=1}^{l} \bar{\lambda}_i \nabla_{\mathbf{x}} g_i(\bar{\mathbf{x}}) + \sum_{i=1}^{m} \bar{\mu}_i \nabla_{\mathbf{x}} h_i(\bar{\mathbf{x}}) = \mathbf{0}$$

Lecture Outline

1. Notation for Representing Networks
2. Optimality Conditions
3. Label Correcting Algorithms
4. Label Setting Algorithms

Notation for Representing Networks

Notation for Representing Networks

Introduction

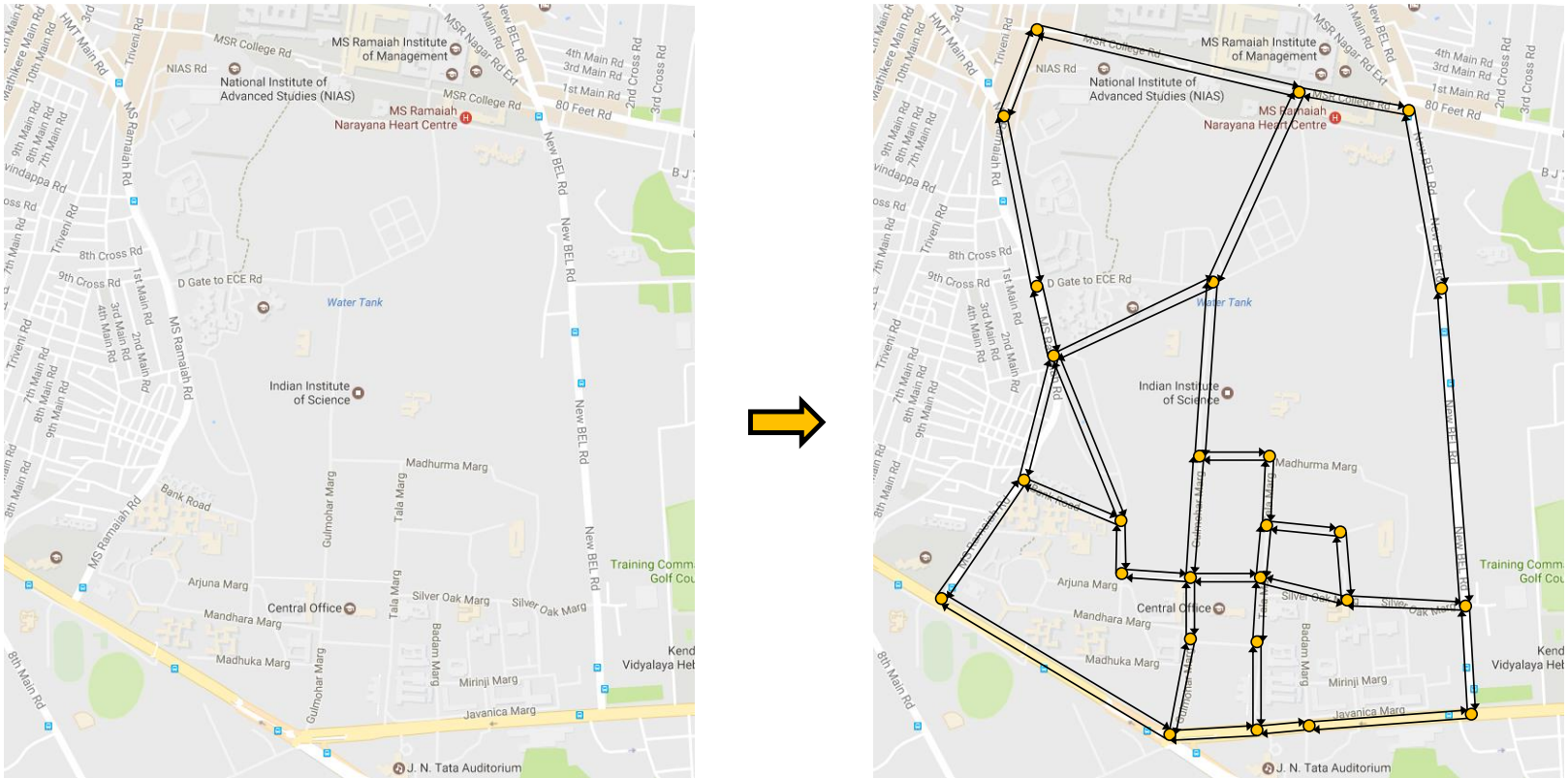
Before attempting to find equilibrium solutions, we need to represent a physical traffic network in a form that can be handled mathematically and by a computer.

We will also need methods to compute shortest paths efficiently because the equilibrium principle assumes that users always try to select shortest paths.

Notation for Representing Networks

Graphs

A graph is a collection of nodes and arcs.



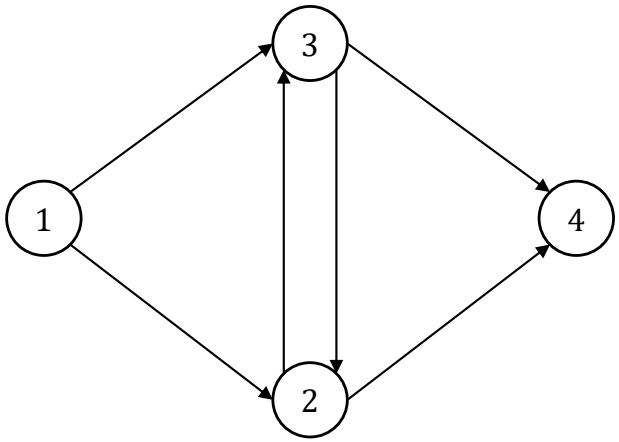
Nodes in a traffic network are junctions or intersections. Arcs are the roadway links connecting adjacent junctions.

Notation for Representing Networks

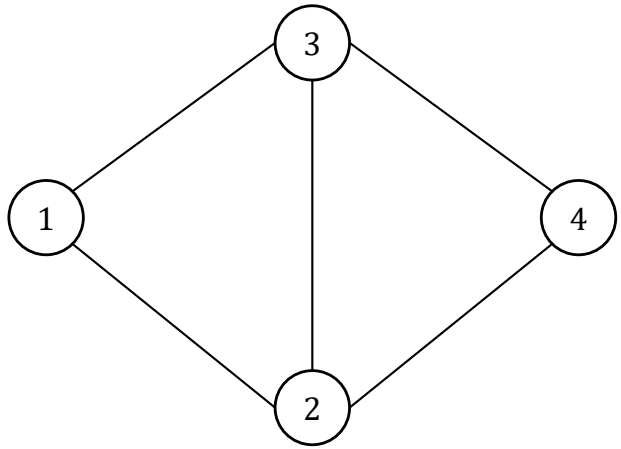
Graphs

Nodes are sometimes referred to as vertices. Arcs are also called links or edges.

Graphs can either be directed or undirected.



Directed Graph



Undirected Graph

Transportation networks are almost always represented using directed graphs. Undirected graphs are used to represent other networks such as social networks.

Notation for Representing Networks

Graphs

Denote a graph using $G = (N, A)$, where N is the set of nodes and A is the set of arcs. An arc $(i, j) \in A$ connects a tail node i to a head node j.

We'll use n and m to denote the number of nodes and arcs respectively.

Given a node i, the set of nodes $\{j : (i, j) \in A\}$ are called downstream nodes and the arcs connecting i to them are called downstream arcs. We can similarly define upstream nodes and arcs.

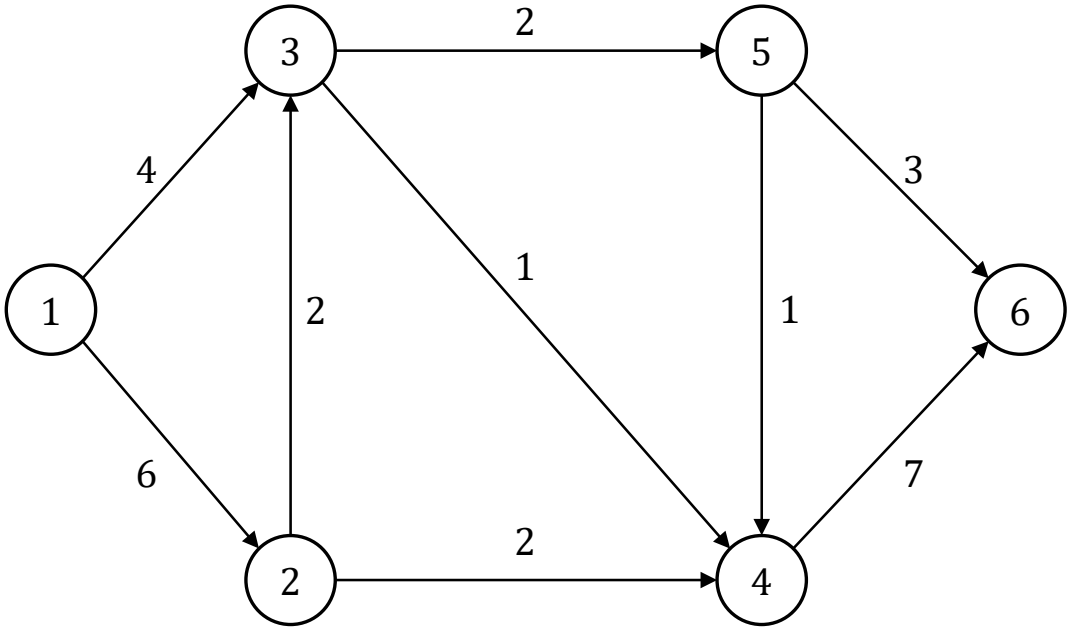
In addition, the nodes and arcs in a graph can have attributes

- Cost
- Travel time
- Demand between pairs of nodes
- Fixed costs at junctions (signal delays)

Notation for Representing Networks

Graphs

Consider the following 6 node and 9 arc graph. Suppose the values next to the arcs indicate travel times.



How do we store this information mathematically or on a computer?

Notation for Representing Networks

Matrix Notation

Node-node adjacency matrix:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 6 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

- Pros: Easy to check if there is an arc between two nodes
- Cons: Many entries are 0s and the storage requirements are high.

Notation for Representing Networks

Matrix Notation

Node-arc incidence matrix:

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}
\begin{array}{c}
\begin{array}{ccccccccc} (1,2) & (1,3) & (2,3) & (2,4) & (3,4) & (3,5) & (4,6) & (5,4) & (5,6) \end{array} \\
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1
\end{pmatrix}
\end{array}
$$

The travel times can be stored in another vector of size equal to the number of arcs.

- Pros: Has a special structure (exactly one $+1$ and -1 in each column). We'll see why this is useful later.
- Cons: Storage is again an issue.

Notation for Representing Networks

Adjacency List/Set

Adjacency List:

1: 2, 3
2: 3, 4
3: 4, 5
4: 6
5: 4, 6
6:

1: (2,6), (3,4)
2: (3,2), (4,2)
3: (4,1), (5,2)
4: (6,7)
5: (4,1), (6,3)
6:

- Pros: Compact way to handle data.
- Cons: Retrieving data of a particular arc requires scanning the adjacency list. Implementation is not trivial.

Optimality Conditions

Optimality Conditions

The Shortest Path Problem

The shortest path problem is to find the optimal distance/time/cost (and the path) to a node from an origin r.

Let μ_i denote the distance label which denotes the cost of a path from the source r to node i. By construction, $\mu_r = 0$.

Think of these as our decision variables. We can then try to find the necessary and sufficient conditions for the **optimality of the labels**.

Optimality Conditions

Necessary and Sufficient Conditions for Labels

Proposition (Necessary conditions)

If a vector of labels $\boldsymbol{\mu}$ are the shortest path distances

$$\mu_j \leq \mu_i + t_{ij} \, \forall \, (i,j) \in A$$

Proof.

Trivial (by contradiction). ■

Optimality Conditions

Necessary and Sufficient Conditions for Labels

Proposition (Sufficient conditions)

Labels $\boldsymbol{\mu}$ that denote the lengths of paths from r to different nodes and satisfy $\mu_j \leq \mu_i + t_{ij}\, \forall\, (i,j) \in A$ are the shortest path distances.

Proof.

Let $P = r = i_1 - i_2 - \ldots - i_k = s$ be **any** path from r to s. Let its length be μ_s. Since the $\boldsymbol{\mu}$s satisfy the above inequality,

$$\mu_{i_k} \leq \mu_{i_{k-1}} + t_{i_{k-1} i_k}$$
$$\mu_{i_{k-1}} \leq \mu_{i_{k-2}} + t_{i_{k-2} i_{k-1}}$$
$$\vdots$$
$$\mu_{i_2} \leq \mu_{i_1} + t_{i_1 i_2}$$

Adding the above inequalities, $\mu_s \leq \mu_r + \sum_{(i,j)\in P} t_{ij} = \sum_{(i,j)\in P} t_{ij}$. Therefore, μ_s is a lower bound for the cost of a path from r to s. Since, it is the length of some path from r to s, it is also an upper bound. ■

Optimality Conditions

Necessary and Sufficient Conditions for Paths

The necessary and sufficient conditions are also commonly referred to as Bellman's Optimality Conditions.

The earlier proposition deals with the optimality of the distance labels. What are the **optimality conditions for a path**?

Optimality Conditions

Necessary and Sufficient Conditions for Paths

Proposition (Necessary and Sufficient Conditions)

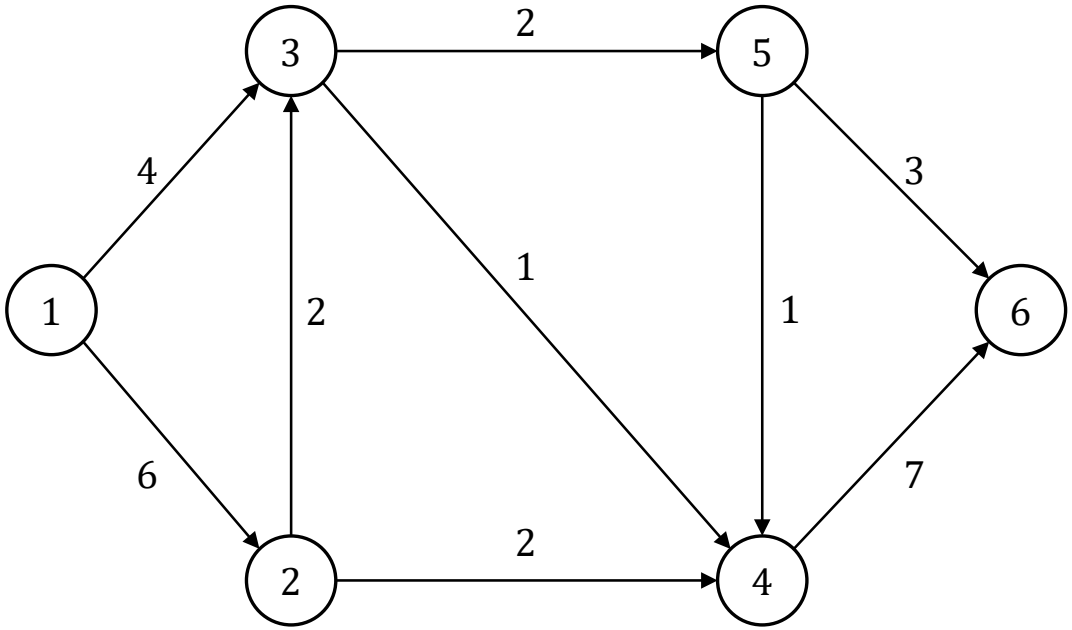
Let $\boldsymbol{\mu}$ represent a vector of shortest path distances. A path P from r to s is optimal iff $\mu_j = \mu_i + t_{ij} \, \forall \, (i,j) \in P$.

The proof is very similar and uses the **subpath optimality property**.

Optimality Conditions

Linear Programming Formulation

Let us try to find the shortest path between nodes 1 and 6 using an optimization framework.



Identify the

- Objective
- Decision Variables
- Constraints

Optimality Conditions

Linear Programming Formulation

- **Objective**
  Clearly, the objective is to reduce the travel time.

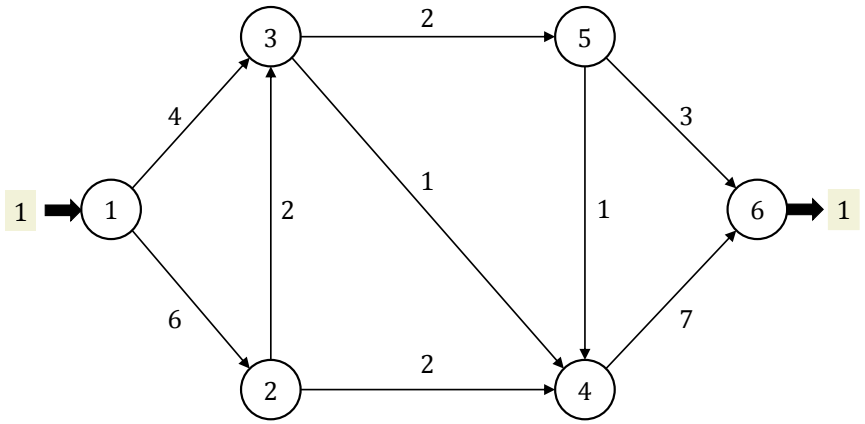
- **Decision Variables**
  Let us define a binary variable x_{ij} which is 1 if link (i, j) belongs to the shortest path and is 0 otherwise.

- **Constraints**
  We need 1 unit of flow (or 1 vehicle) to enter node 1 and leave node 6. When it passes through intermediate nodes 'what goes in must come out'.

Optimality Conditions

Linear Programming Formulation

The **objective** is $4x_{13} + 6x_{12} + 2x_{23} + \ldots + 7x_{46} + 3x_{56}$



Flow conservation constraints:

- Node 1: $x_{12} + x_{13} = 1$
- Node 2: $x_{12} = x_{23} + x_{24}$
- Node 3: $x_{13} + x_{23} = x_{34} + x_{35}$

$\vdots$

- Node 6: $x_{46} + x_{56} = 1$

In addition, we have integrality constraints, i.e., $x_{ij} \in \{0, 1\} \, \forall \, (i, j) \in A$.

Optimality Conditions

Linear Programming Formulation

Let's write the equality constraints in the following form.

$$
\begin{array}{rrrrrrrrrcr}
x_{12} & + x_{13} & & & & & & & & = & 1 \\
-x_{12} & & + x_{23} & + x_{24} & & & & & & = & 0 \\
& - x_{13} & - x_{23} & & + x_{34} & + x_{35} & & & & = & 0 \\
& & & - x_{24} & - x_{34} & & + x_{46} & - x_{54} & & = & 0 \\
& & & & & - x_{35} & & + x_{54} & + x_{56} & = & 0 \\
& & & & & & - x_{46} & & - x_{56} & = & -1
\end{array}
$$

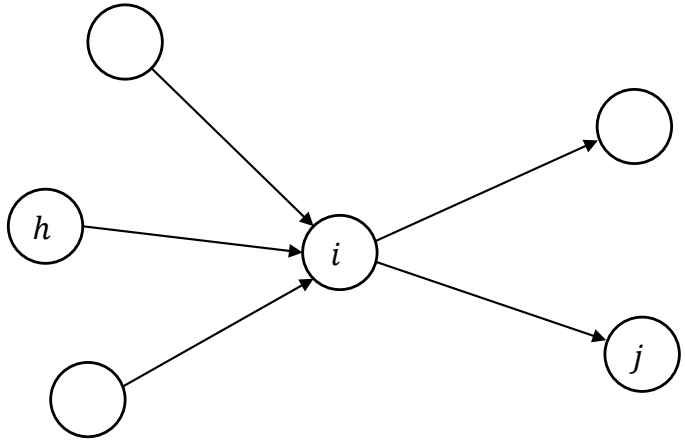
Do you notice any structure in these constraints?

$$
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1
\end{pmatrix}
\begin{pmatrix}
x_{12} \\ x_{13} \\ x_{23} \\ x_{24} \\ x_{34} \\ x_{35} \\ x_{46} \\ x_{54} \\ x_{56}
\end{pmatrix}
$$

Optimality Conditions

Linear Programming Formulation

Generalizing this,



The equality constraints can be written as

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{h:(h,i)\in A} x_{hi} = \begin{cases} 1 & \text{if } i = r \\ -1 & \text{if } i = s \\ 0 & \text{otherwise} \end{cases}$$

Hence, the shortest path problem takes the form

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

$$\text{s.t.} \sum_{j:(i,j)\in A} x_{ij} - \sum_{h:(h,i)\in A} x_{hi} = \begin{cases} 1 & \text{if } i = r \\ -1 & \text{if } i = s \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} \in \{0,1\} \, \forall \, (i,j) \in A$$

Recall that the equality constraints can be written as $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is an $n \times m$ matrix, $\mathbf{x}$ is a $m \times 1$ vector, and $\mathbf{b}$ is a $n \times 1$ vector.

Optimality Conditions

Linear Programming Formulation

Luckily, the node-arc incidence matrix **A** satisfies a property called total unimodularity because it has exactly one $+1$ and one -1 in each column.

If the constraint matrix has this property, one can replace the integer constraints with inequalities, solve the problem as an LP, and get an integer optimal solution!

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

$$\text{s.t.} \sum_{j:(i,j)\in A} x_{ij} - \sum_{h:(h,i)\in A} x_{hi} = \begin{cases} 1 & \text{if } i = r \\ -1 & \text{if } i = s \\ 0 & \text{otherwise} \end{cases}$$

$$0 \leq x_{ij} \leq 1 \, \forall \, (i,j) \in A$$

Optimality Conditions

Linear Programming Formulation

Convert the LP into standard form and write the KKT Conditions for the shortest path problem.

Notice that the upper bounds $x_{ij} \leq 1 \, \forall \, (i,j) \in A$ are redundant.

Rope Model for Shortest Paths

Let's start with the last set of KKT conditions. **Gradient of the Lagrangian vanishes:**

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{(i,j)\in A} t_{ij} x_{ij} + \sum_{(i,j)\in A} \lambda_{ij}(-x_{ij}) + \sum_{i\in N} \mu_i \left(\sum_{j:(i,j)\in A} x_{ij} - \sum_{h:(h,i)\in A} x_{hi} - b_i \right)$$

$$\frac{\partial \mathcal{L}}{\partial x_{ij}} = t_{ij} - \lambda_{ij} + \mu_i - \mu_j = 0$$

$$\Rightarrow \lambda_{ij} = t_{ij} + \mu_i - \mu_j$$

$\boldsymbol{\lambda}$s are also called *reduced costs*.

Optimality Conditions

Linear Programming Formulation

Primal feasibility:

$$\mathbf{Ax} = \mathbf{b}$$
$$0 \leq x_{ij} \leq 1 \forall (i,j) \in A$$

Dual feasibility:

$$\lambda_{ij} \geq 0 \forall (i,j) \in A$$

Complementary Slackness:

$$\lambda_{ij} x_{ij} = 0 \forall (i,j) \in A$$

Gradient of the Lagrangian vanishes:

$$\lambda_{ij} = t_{ij} + \mu_i - \mu_j \forall, (i,j) \in A$$

From the above conditions, interpreting $\boldsymbol{\mu}$s as the distance labels, we get the Bellman's conditions: $\mu_j \leq t_{ij} + \mu_i$ and if $x_{ij} = 1 \Rightarrow \mu_j = \mu_i + t_{ij}$!!!

There are efficient algorithms to solve LPs. But for network problems, we can almost always construct algorithms which are much faster.

Label Correcting Algorithms

Label Correcting Algorithms

Introduction

Recall the optimal conditions for shortest path distances:

$$\mu_j \leq t_{ij} + \mu_i \, \forall \, (i,j) \in A$$

The general approach in shortest path algorithms:

1. Initialize the labels of all nodes except the origin to ∞. The label of the origin is set to 0.
2. The labels are upper bounds to the shortest distance. Iteratively reduce them until the above optimality conditions are satisfied.

Label Correcting Algorithms

Introduction

In most algorithms, while finding the shortest distance from an origin r to some node in the network, we get the shortest distances to all the other nodes for free.

Such methods in which we fan out from the origin are called *one-to-all* algorithms. We also keep track of a *predecessor labels* π, which for each node i gives its upstream node in a path of length μ_i.

We almost always never store paths. Upon termination, the predecessor labels are used to re-construct the optimal path.

Alternately, we can construct *all-to-one* algorithms which find the shortest paths from all nodes to a destination s.

Label Correcting Algorithms

Generic Label Correcting Algorithm

In the most naive version of a label correcting algorithm,

- Scan each arc in the network and if the optimality condition is violated, update the label of the head node.
- Repeat until no arc violates the optimality condition.

GENERIC LABEL CORRECTING(G, r)

Step 1: Initialize

$\mu_r \leftarrow 0, \pi_r \leftarrow r$

$\mu_i \leftarrow \infty, \pi_i \leftarrow -1 \, \forall \, i \in N \backslash \{r\}$

Step 2:

while Some arc (i, j) satisfies $\mu_j > \mu_i + t_{ij}$ **do**

$\mu_j \leftarrow \mu_i + t_{ij}$

$\pi_j \leftarrow i$

end while

Label Correcting Algorithms

Generic Label Correcting Algorithm

We can formally prove that the generic label correcting method works.

- Show that it terminates
- When it terminates, prove that optimality conditions are met. (Trivial)

From an implementation standpoint, one option to find an arc that violates the optimality conditions is to scan all the arcs in a fixed order. But this technique isn't intelligent.

Can we do better?

Label Correcting Algorithms

Modified Label Correcting Algorithms

Instead of scanning all arcs within each iteration, let's keep a *list of nodes whose downstream arcs may violate the optimality criteria*. Call this the scan eligible list (SEL).

Pick a node i from this list and update the label of its downstream nodes $j : (i, j) \in A$ if the optimality conditions aren't met. When μ_j is reduced what happens to the labels of

- Downstream nodes of j
- Upstream nodes of j

Therefore, if there are any updates, add j to SEL. (Always?)

Label Correcting Algorithms

Modified Label Correcting Algorithms

MODIFIED LABEL CORRECTING(G, r)

Step 1: Initialize
$\mu_r \leftarrow 0, \pi_r \leftarrow r$
$\mu_i \leftarrow \infty, \pi_i \leftarrow -1 \, \forall \, i \in N \backslash \{r\}$
SEL $\leftarrow \{r\}$

Step 2:
while SEL$\neq \emptyset$ **do**
    Remove i from SEL
    for $j : (i, j) \in A$ **do**
        if $\mu_j > \mu_i + t_{ij}$ **then**
            $\mu_j \leftarrow \mu_i + t_{ij}$
            $\pi_j \leftarrow i$
            if $j \notin$ SEL **then** add j to SEL
        end if
    end for
end while

Label Correcting Algorithms

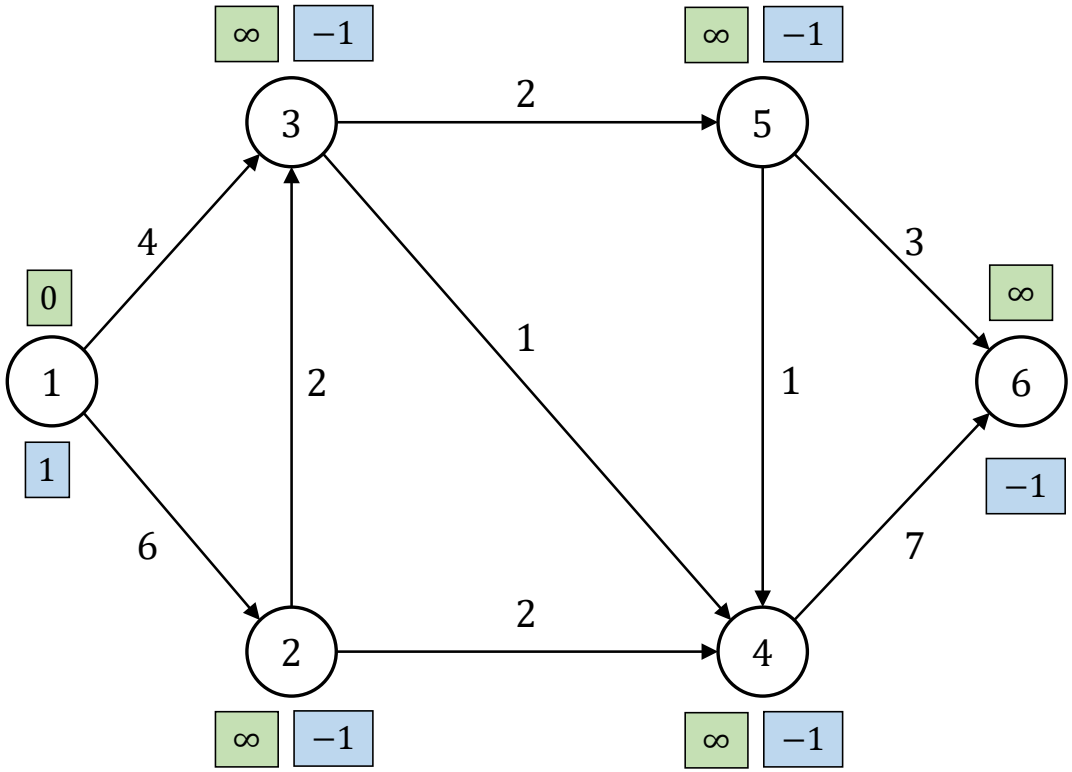
Modified Label Correcting Algorithms

How do we identify

1. Optimal paths
2. Nodes that are not reachable

Label Correcting Algorithms

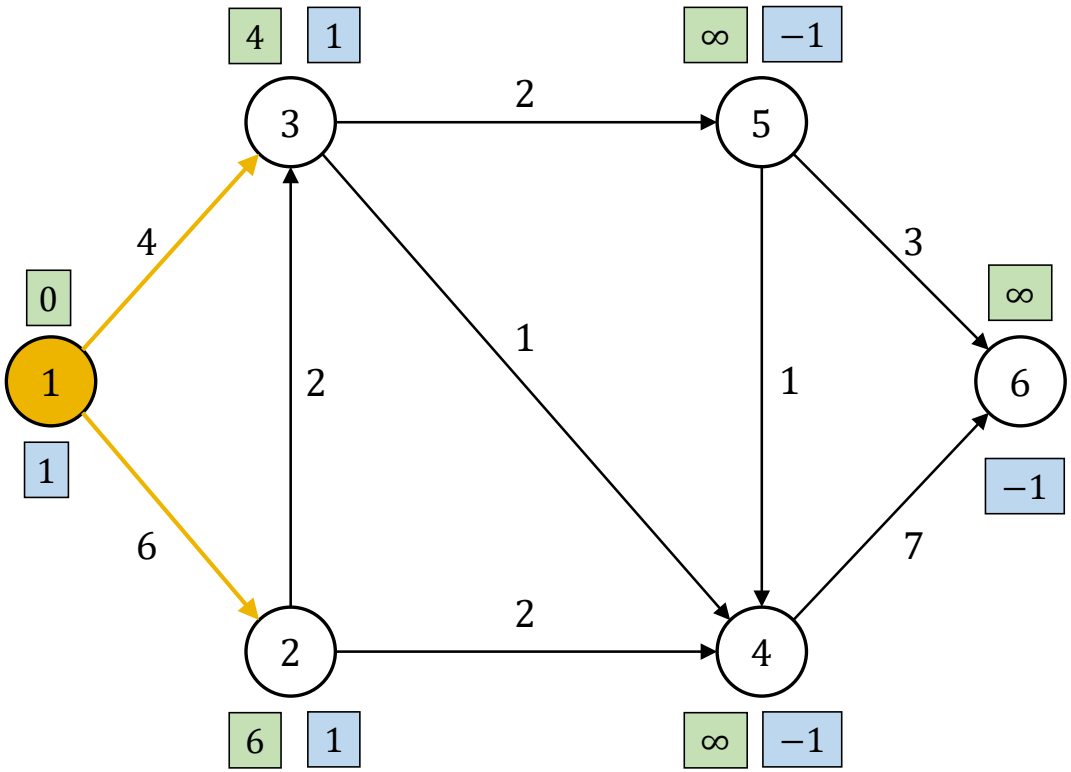
Example



$$SEL = \{1\}$$

Label Correcting Algorithms

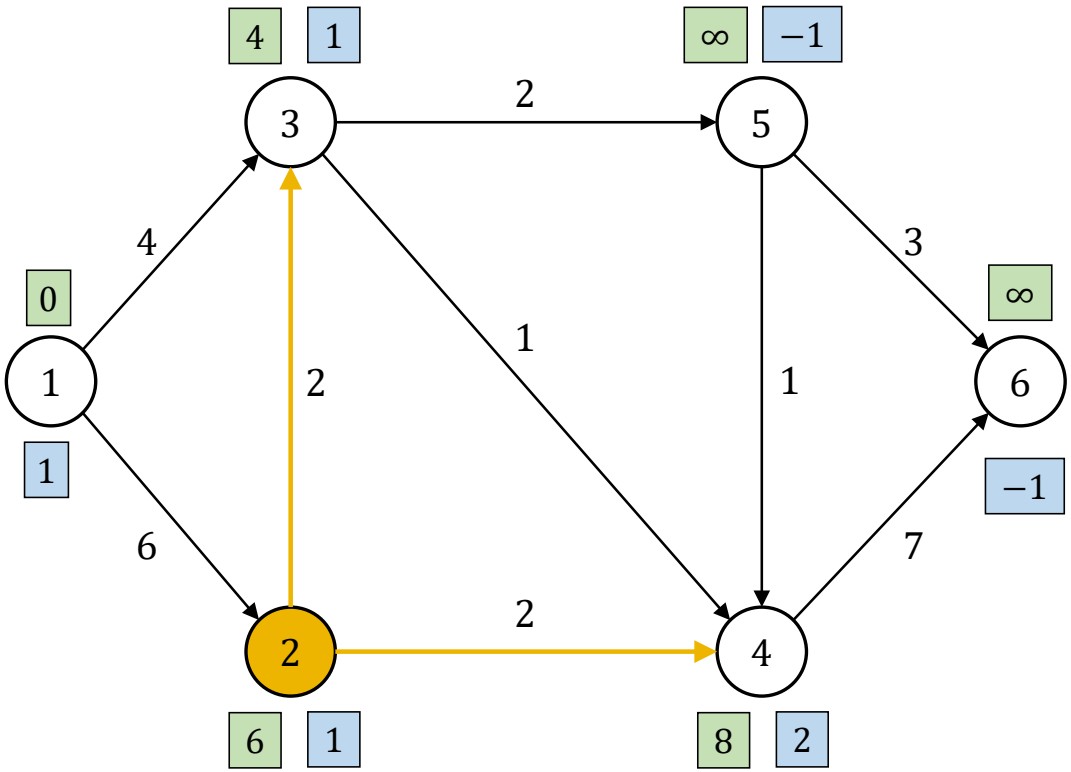
Example



$$SEL = \{2, 3\}$$

Label Correcting Algorithms

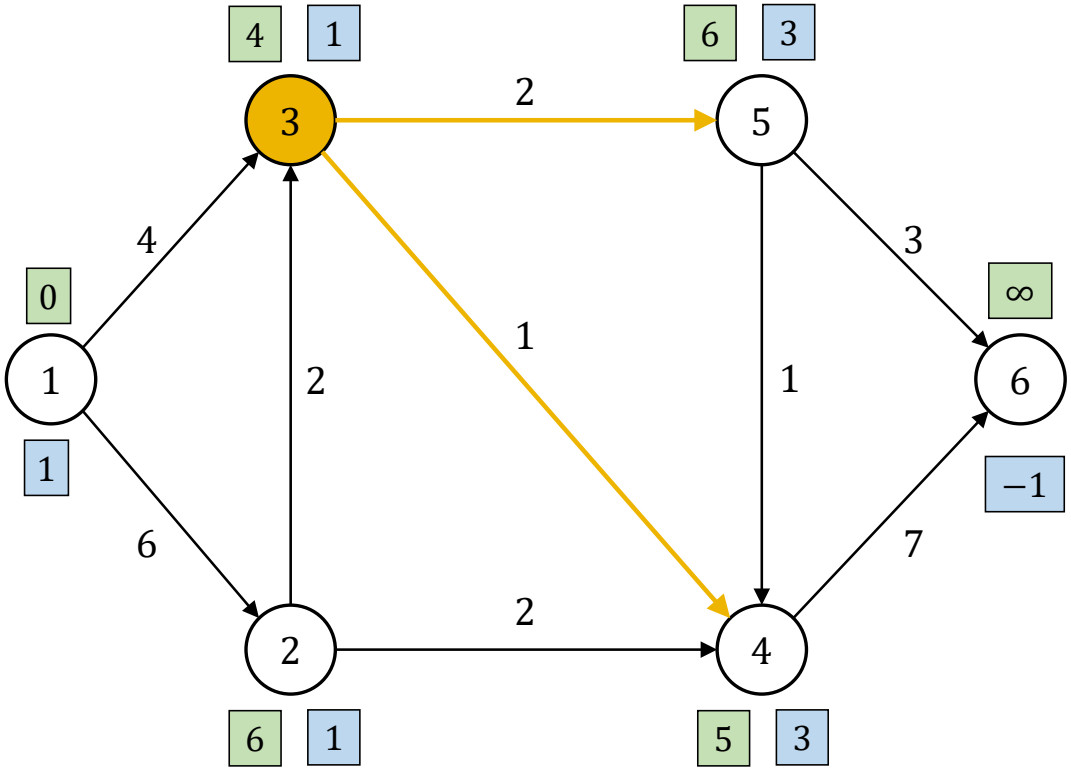
Example



$$SEL = \{3, 4\}$$

Label Correcting Algorithms

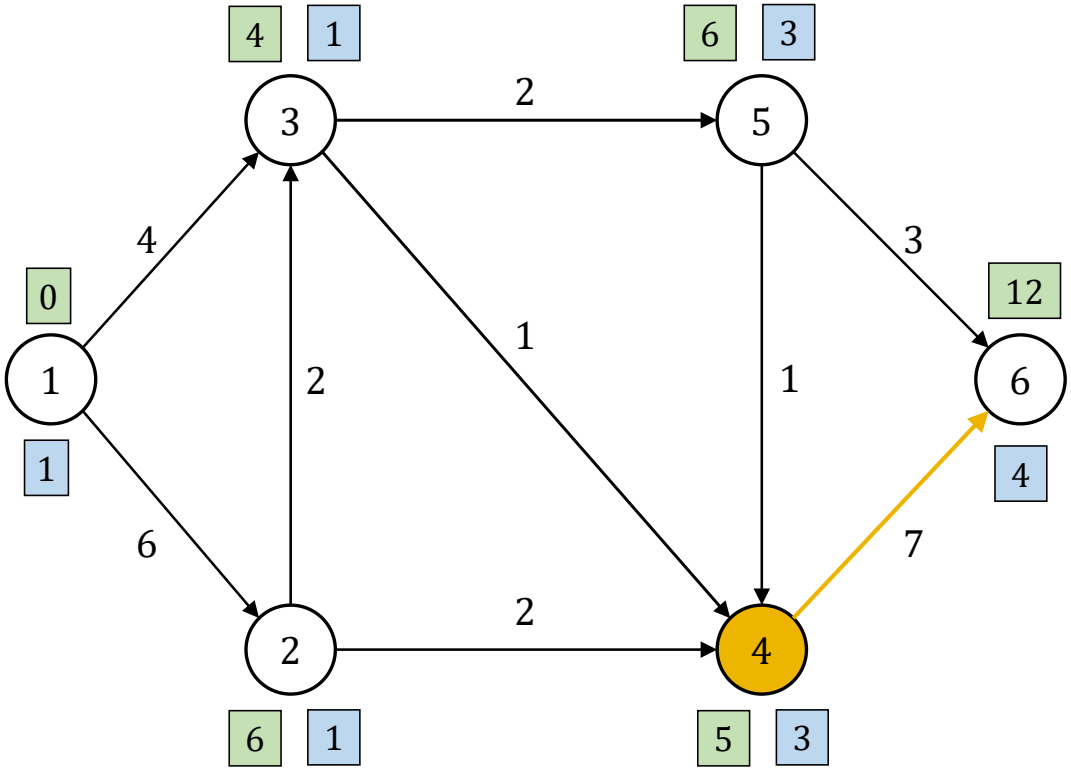
Example



$$SEL = \{4, 5\}$$

Label Correcting Algorithms

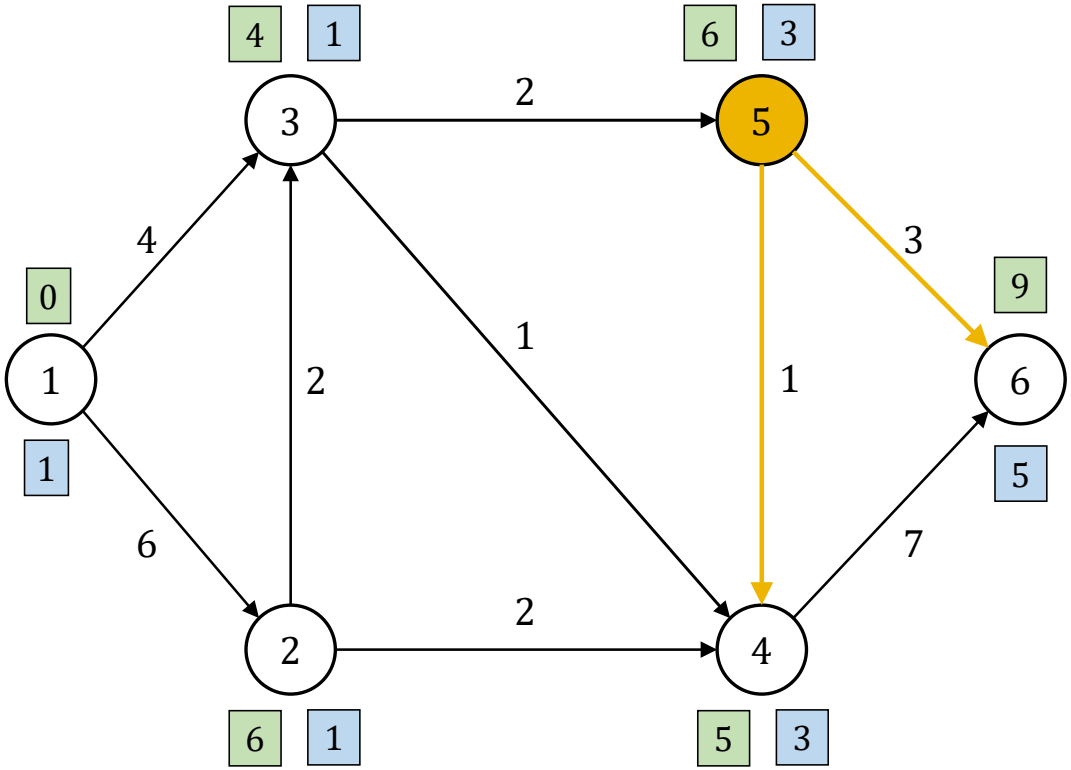
Example



$$SEL = \{5, 6\}$$

Label Correcting Algorithms

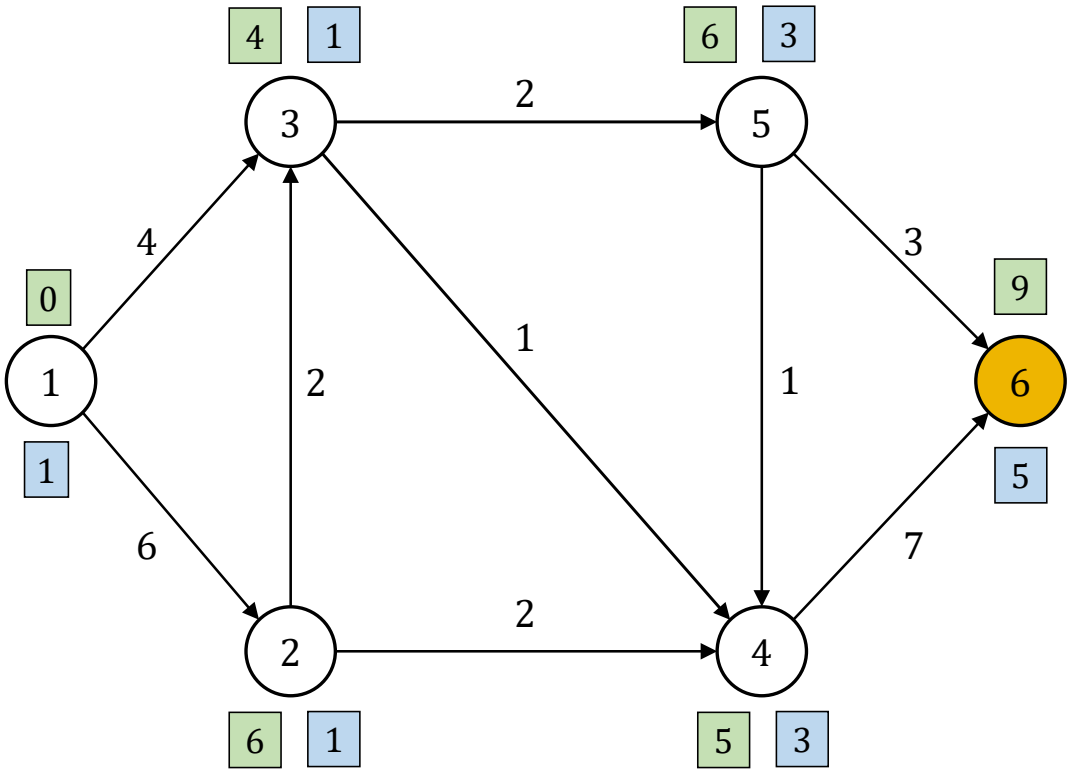
Example



$$SEL = \{6\}$$

Label Correcting Algorithms

Example



$$SEL = \emptyset$$

Label Correcting Algorithms

Example

This method of using a SEL is much faster than the generic label correcting algorithm.

Can we do better?

Label Setting Algorithms

Label Setting Algorithms

Introduction

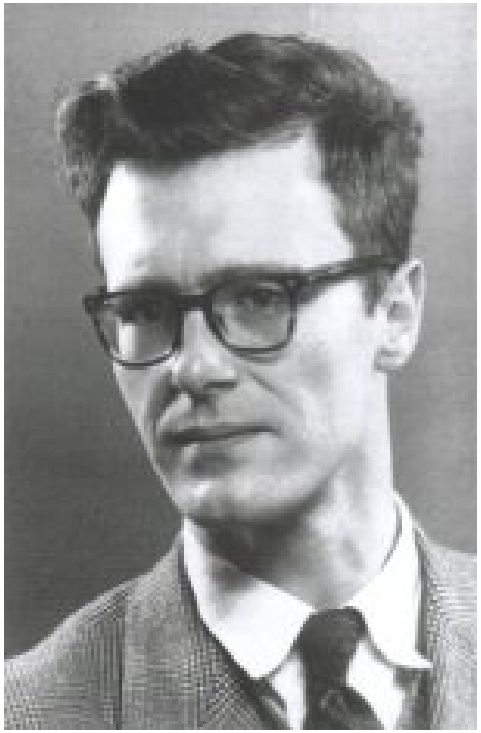
It so happens that when arc costs are non-negative, we can find the shortest paths in a shorter way!

These methods, also called label setting algorithms operate similarly but at every iteration, the label of one node is **set** (i.e., it is optimal).

In contrast, label correcting methods are guaranteed to give optimal labels only after termination.

Label Setting Algorithms

Dijkstra's Method



Edsger W. Dijkstra

Label Setting Algorithms

Dijkstra's Method

The set of nodes can be divided into two groups. S is used to represent the nodes whose labels are set. The remaining nodes are denoted using $\bar{S}$.

The labels of nodes in S are optimal and those in $\bar{S}$ are upper bounds.

In each iteration, we pick a node in $\bar{S}$ with the lowest label and move it to S and scan its downstream arcs.

Label Setting Algorithms

Dijkstra's Method

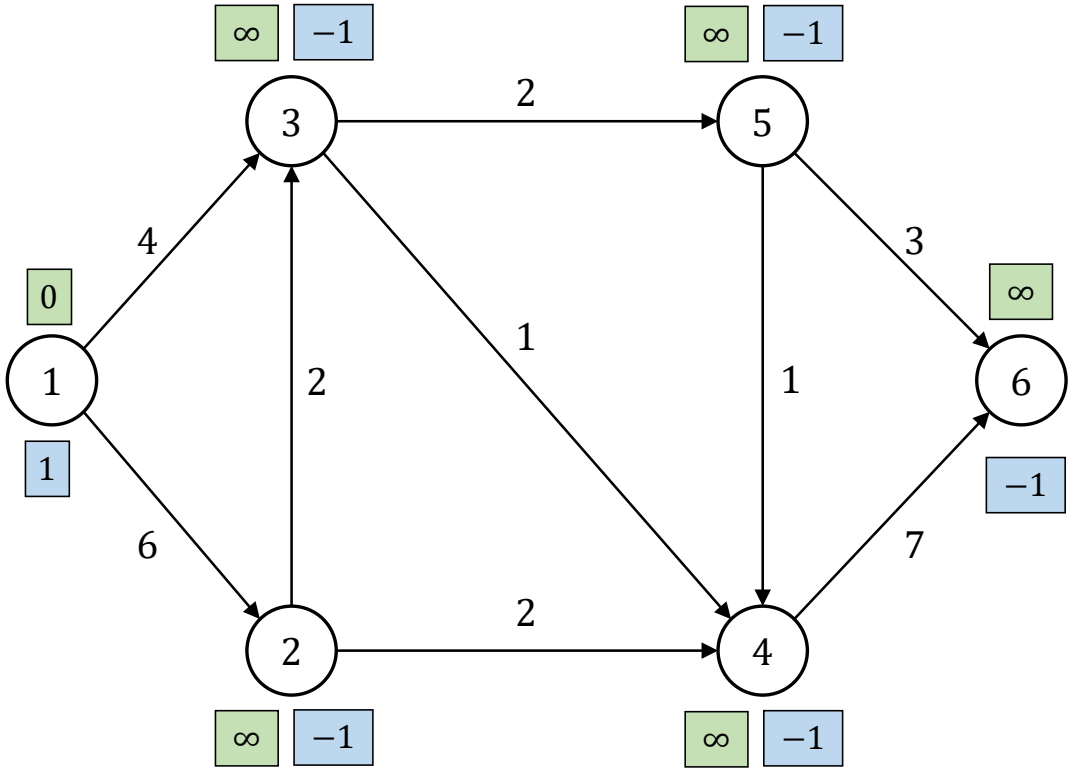
DIJKSTRA'S ALGORITHM(G, r)

Step 1: Initialize
$S \leftarrow \emptyset$, $\bar{S} \leftarrow N$
$\mu_r \leftarrow 0, \pi_r \leftarrow r$
$\mu_i \leftarrow \infty, \pi_i \leftarrow -1 \, \forall \, i \in N \backslash \{r\}$

Step 2:
while $\bar{S} \neq \emptyset$ **do**
    $i \leftarrow \arg\min_{j \in \bar{S}} \mu_j$
    $S \leftarrow S \cup \{i\}$, $\bar{S} \leftarrow \bar{S} \backslash \{i\}$
    for $j : (i, j) \in A$ **do**
        if $\mu_j > \mu_i + t_{ij}$ **then**
            $\mu_j \leftarrow \mu_i + t_{ij}$
            $\pi_j \leftarrow i$
        end if
    end for
end while

Label Setting Algorithms

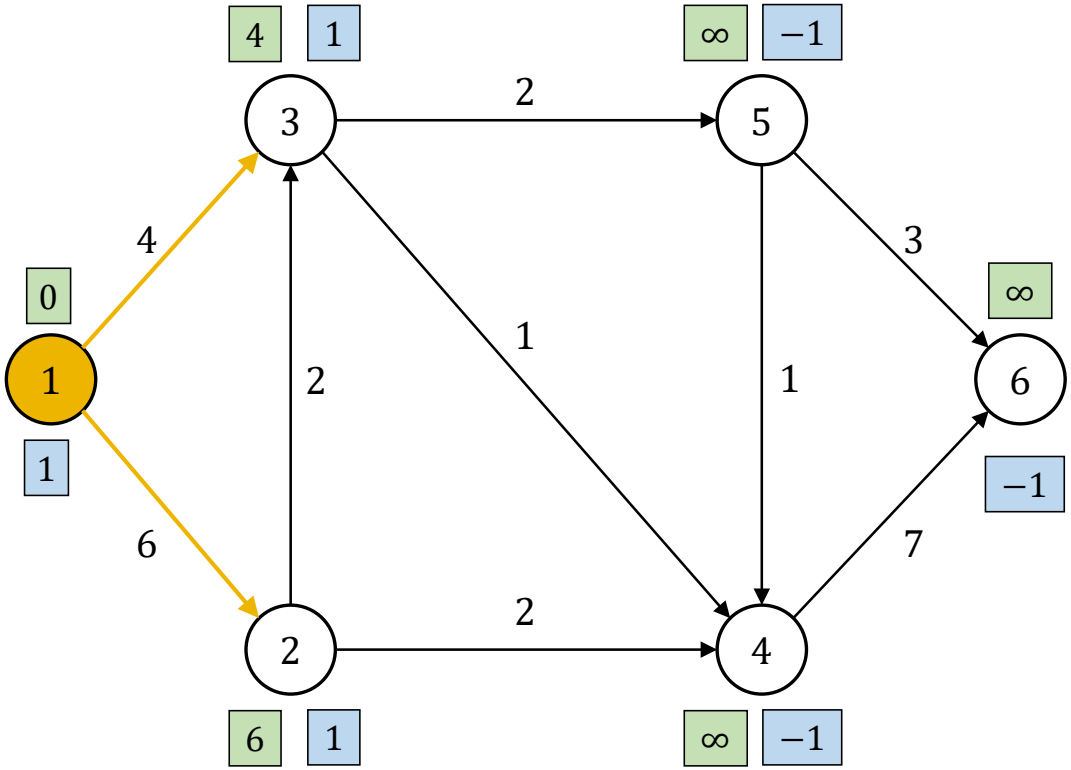
Example



$$S = \emptyset,\ \bar{S} = \{1, 2, \ldots, 6\}$$

Label Setting Algorithms

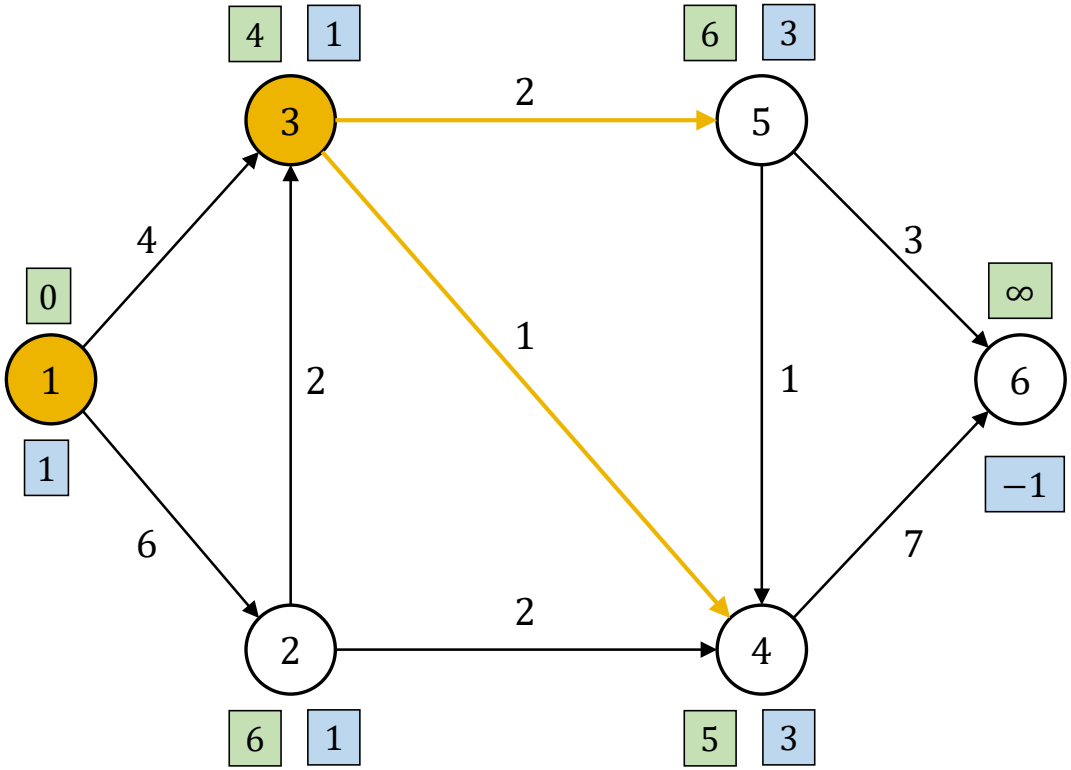
Example



$$S = \{1\},\ \bar{S} = \{2, 3, 4, 5, 6\}$$

Label Setting Algorithms

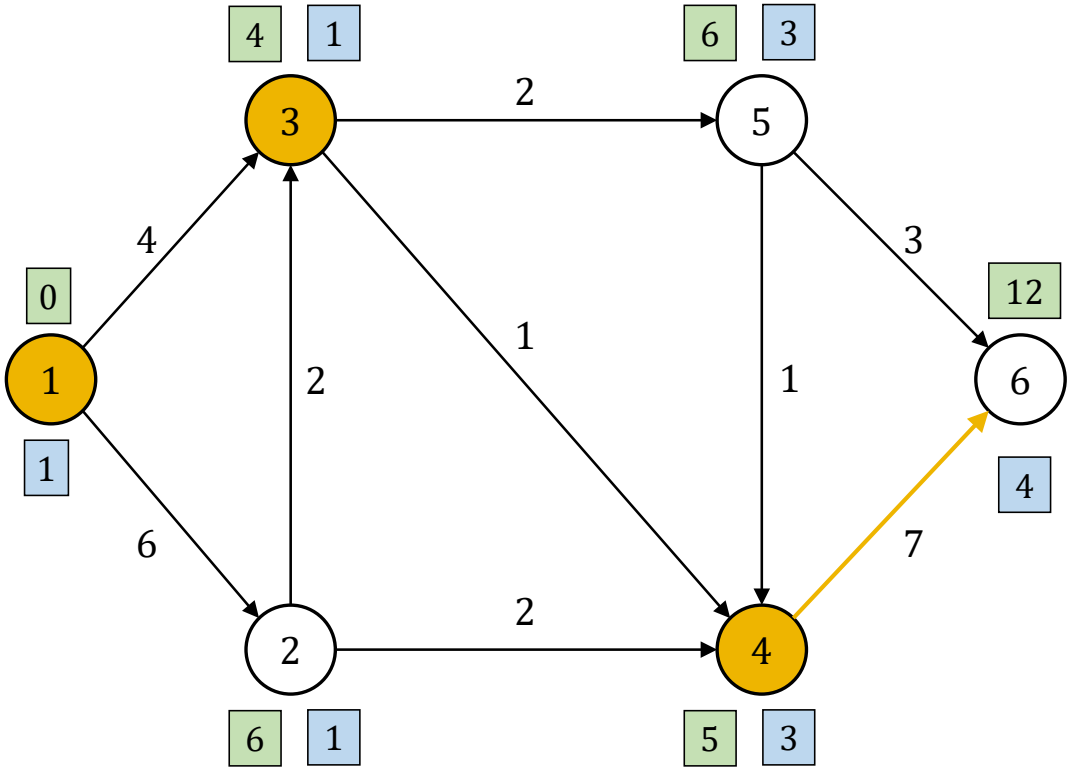
Example



$$S = \{1, 3\},\ \bar{S} = \{2, 4, 5, 6\}$$

Label Setting Algorithms

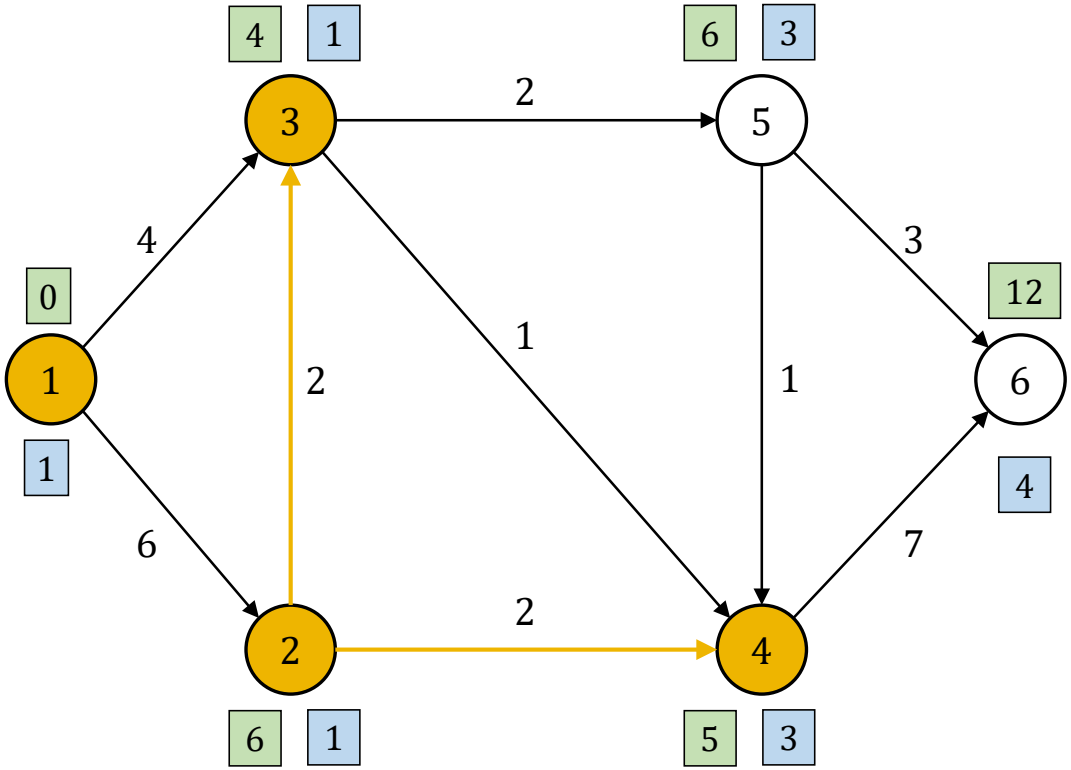
Example



$$S = \{1, 3, 4\},\ \bar{S} = \{2, 5, 6\}$$

Label Setting Algorithms

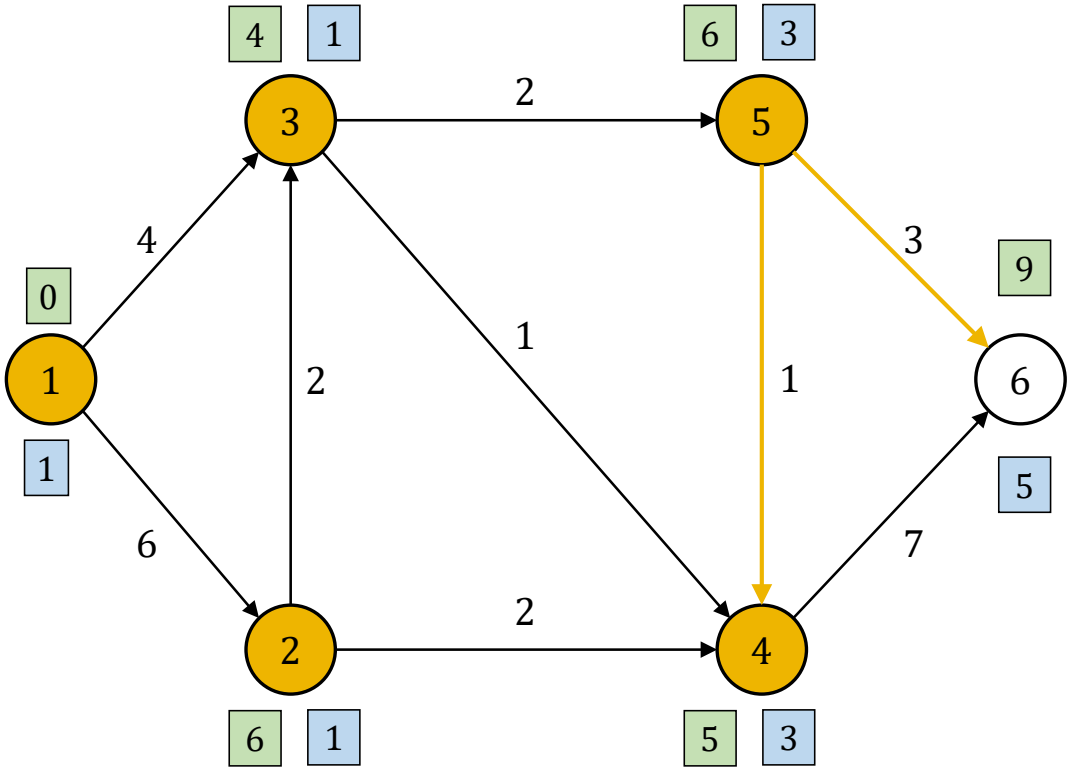
Example



$$S = \{1, 3, 4, 2\},\ \bar{S} = \{5, 6\}$$

Label Setting Algorithms

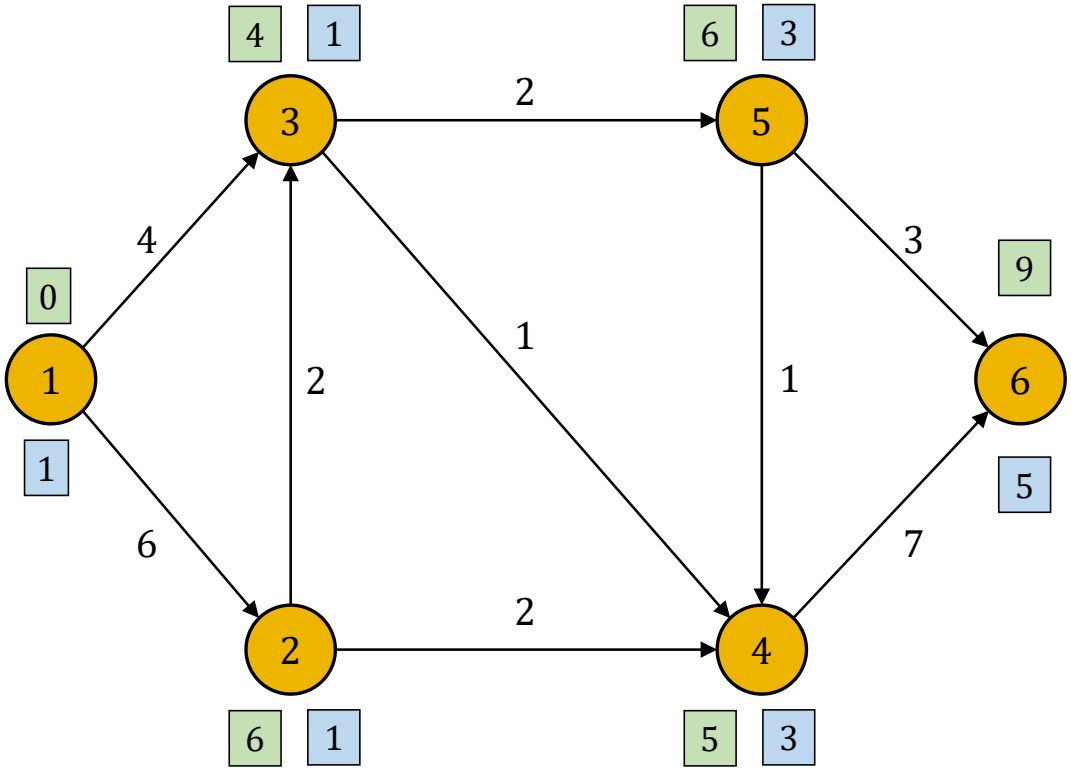
Example



$$S = \{1, 3, 4, 2, 5\},\ \bar{S} = \{6\}$$

Label Setting Algorithms

Example



$$S = \{1, 3, 4, 2, 5, 6\},\ \bar{S} = \emptyset$$

Label Setting Algorithms

Computational Performance

Label setting algorithms terminate after n iterations (assuming all nodes are reachable) because in each step, we move a node from $\bar{S}$ to S.

Label correcting algorithms can take more iterations since nodes can re-enter the SEL. But within each iteration label setting algorithms take more time. (Why?)

It is not so difficult to derive worst case complexity bounds for both algorithms for different implementations.

Supplementary Reading

- Boyles, Chapter 2
- Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). Network flows: theory, algorithms, and applications.

Your Moment of Zen

Sometimes we don't need an algorithm to find the shortest path...