Shortest paths and trees

Carlo Mannino
(from Lex Schrijver notes)

University of Oslo, INF-MAT5360 - Autumn 2010 (Mathematical optimization)
Combinatorial Optimization
Basic Definition and examples
Combinatorial Optimization Problem

- Finite ground set $E$, weight function $w : E \rightarrow R$. (i.e. $w \in R^E$)
- Feasible solutions $\mathcal{F} = \{F_1, \ldots, F_m\}$, with $F_i \subseteq E$, $i = 1, \ldots, m$
- Combinatorial optimization problem (CO)
  \[ \max \{ w(F) : F \in \mathcal{F} \} , \quad \text{where} \quad w(F) = \sum_{e \in F} w(e) \]
- Let $S \subseteq \{0,1\}^E$ be the set of the incidence vectors of the sets in $\mathcal{F}$
  \[ S = \{ \chi^F : F \in \mathcal{F} \} \]
  \[ w(F) = \sum_{e \in F} w(e) = w^T \chi^F \]
- Combinatorial optimization problem (rewritten)
  \[ \max \{ w^T x : x \in S \} \quad \text{0-1 linear program} \]
- Solving (CO) and 0-1 LP is difficult ($NP$–hard)
Example: project selection

- Projects A e B
- Profits $w_A$ e $w_B$
- Costs $c_A = 5$, $c_B = 7$
- Budget constraint $\leq D = 10$

$E = \{A, B\}$

Feasible Solutions $\mathcal{F} = \{\emptyset, \{A\}, \{B\}\}$

$c(\emptyset) = 0$, $c(\{A\}) = 5$, $c(\{B\}) = 7$,
$c(\{A, B\}) = 12 > D$ \{A, B\} not feasible

Project selection problem:

Find a selection of projects satisfying the budget constraint and maximizing profit.

$$\max w_A x_A + w_B x_B$$

$x \in \mathcal{S} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$
From CO to LP

\[ \max \{ w^T x : x \in S \} \]

- \( P = \text{conv}(S) \) convex hull of the points in \( S \). \( P \) is a polytope.
- Vertices of \( P = \text{ext}(P) = S \).

\[ \max \{ w^T x : x \in S \} = \max \{ w^T x : x \in \text{ext}(P) \} = \max \{ w^T x : x \in P \} \]

linear program!

We can solve (CO) by linear programming!
Basic combinatorial optimization problems: shortest path, minimum spanning tree, maximum flow, minimum cut.

Connections with linear programming and polyhedral theory

Integer Polyhedra

Methods: heuristic algorithms

Methods: exact approaches

A huge number of relevant real-life applications can be modeled as COs
You need to connect a source $s$ (of water, packets, …) to a number of locations (farms, computers).

Each connection (pipe, fiber) has a cost.

WANT: find a minimum cost network connecting all locations to the source.
Example: flow

Expected demand:

<table>
<thead>
<tr>
<th>A</th>
<th>RM</th>
<th>MI</th>
<th>FR</th>
<th>ZU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>-</td>
<td>30</td>
<td>-</td>
<td>80</td>
</tr>
<tr>
<td>MI</td>
<td>-</td>
<td>-</td>
<td>40</td>
<td>14</td>
</tr>
<tr>
<td>FR</td>
<td>-</td>
<td>14</td>
<td>-</td>
<td>14</td>
</tr>
<tr>
<td>ZU</td>
<td>-</td>
<td>-</td>
<td>30</td>
<td>-</td>
</tr>
</tbody>
</table>

- **30+80<120**
- **14+14<30**
- **40+30<80**
- **80+40+14+14<150**

**Passengers should**

- reach their destinations
- be assigned to departing flights
- .... Satisfying capacity constraints

**Flights (available sits):**

- Roma-Milano: 120 sits
- Milano-Zurich: 150 sits
- Frankfurt-Milano: 30 sits
- Zurich-Frankfurt: 80 sits
Example: Project Scheduling

Projects decompose into activities

Activities may require resources, which in turn may be limited

Precedence Relations exist between activities.

WANT: find a schedule of the activities satisfying all precedence constraints and minimizing the project completion time

= “7 must start at least 6 time units after 5 is terminated”
Example: *Job-Shop Scheduling*

- A product (*job*) must be processed on different machines
- Processing a *job* on a machine is called *operation*
- Each machine can process at most $k$ jobs at a time.

- **WANT**: find a schedule of the operations satisfying machine capacities and additional precedence constraints.
Example: vehicle routing

Transfer goods from origins to destinations
• Minimizing transportation costs
• Satisfying:
  • - constraints on vehicle capacities
  • - connectivity constraints
  • …

Several parameters are involved

1. Origin and destination position
2. Demand level
3. Fleet size
…
Example: vehicle routing

- Each vehicle visits a subset of customers and returns to depot

$\begin{align*}
  d_1 &= 3 \\
  d_2 &= 2 \\
  Q_A &= 10 \\
  d_3 &= 2 \\
  d_4 &= 1 \\
  d_5 &= 1
\end{align*}$
A famous instance

Standard test instance G-n262-k25 (Gillett & Johnson 1976)
"The world record" for G-n262-k25: 5685 vs. 6119
(SINTEF 2003)
Shortest paths and trees
Walks and paths

\[ D = (V,A) \] directed graph, \( s,t \in V \)

- \( V \) vertices
- \( A \) arcs

**Walk:** alternating sequence of vertices and arcs:

\[ P = (v_0, a_1, v_1, \ldots, a_m, v_m) : a_i = (v_{i-1}, v_i) \quad i = 1, \ldots, m \]

**Path:** walk with no repeated vertices
Length and distance

- **s-t walk (path):** walk (path) with starting vertex \( s \) and end vertex \( t \)
- **Length of walk \( P \):** number of arcs
- **\( v \) reachable from \( u \):** there exists and \( u-v \) path in \( D \)
- **Distance from \( s \) to \( t \):** minimum length of any \( s-t \) path \((+\infty \text{ if } t \text{ is not reachable from } s)\)
- **\( V_i \):** set of vertices at distance \( i \) from \( s \)

\[
D = (V, A)
\]
Finding shortest paths

- \( V_i \): set of vertices at distance \( i \) from \( s \)

**Recursive Rule:**

\( V_{i+1} \): set of vertices \( v \in V \setminus (V_0 \cup V_1 \cup \ldots \cup V_i) \) for which \( (u,v) \in A \) for some \( u \in V_i \)

**Shortest Path Algorithm:**

1. Set \( V_0 = \{s\} \), \( i = 0 \).
2. While \( V_i \neq \emptyset \)
   
   3. Compute \( V_{i+1} \) from \( V_i \)
   
   4. Set \( i = i + 1 \)

EndWhile

- Running Time: \( O(|A|) \):

- Finds the distance from \( s \) to all vertices reachable from \( s \)
- Finds \( T = (V,A') \) shortest path tree
- At each iteration explores new arcs; at the end every arc is visited at most once
Graphs with non-negative arc lengths

- Length (weight) function $l: A \rightarrow Q_+$
- Given walk $P = (v_0, a_1, v_1, \ldots, a_m, v_m)$
- Length of $P$: $l(P) = \sum_{i=1}^{m} l(a_i)$
- Distance from $s$ to $v$ (w.r.t. $l$): $\text{dist}(v)$ length of a minimum length $s$-$v$ path in $D$ ($+\infty$ if no $s$-$v$ path exists)

On a shortest $s$-$v$ path $s = v_0, v_1, \ldots, v_k = v$

$\text{dist}(v_i) = \text{dist}(v_{i-1}) + l(v_{i-1}, v_i)$

(every sub-path is a shortest path)
Graphs with non-negative arc lengths

\[ \delta^-(v) = \{ e \in A: e = (u,v) \text{ for some } u \in V \} \]

\[ \delta^+(v) = \{ e \in A: e = (v,u) \text{ for some } u \in V \} \]

\[ N(v) = \{ u \in V: uv \in A \} \]

Trivial Facts:

(i) \( \text{dist}(s) = 0 \)

(ii) \( \text{dist}(v) = \min \{ \text{dist}(u) + l(uv): uv \in \delta^-(v) \} \)

(iii) \( \text{dist}(v) \leq \text{dist}(u) + l(uv), uv \in \delta^-(v) \)
Dijkstra Shortest Path Algorithm:

1. Set $U := V$, $f(s) := 0$, $f(v) := +\infty$ for $v \in U \setminus \{s\}$

2. While $U \neq \{\}$

3. Select $u \in U$ minimizing $f(u)$. Set $U := U \setminus \{u\}$.

4. For each $uv \in \delta^+(u)$

5. If $f(v) > f(u) + l(uv)$ Reset $f(v) := f(u) + l(uv)$

EndFor

EndWhile

The final function $f$ gives the distance from $s$. 

Theorem 1.3
Proof of Theorem 1.3

Proof.

Claim 1: at any iteration $f(v) \geq \text{dist}(v)$, for each $v \in V$.

Suppose not. True at initialization. Then there is a first Reset and a vertex $w$ such that:

(i) $f(w) < \text{dist}(w)$; (ii) $f(w) = f(u) + l(uw)$; (iii) $f(u) \geq \text{dist}(u)$

$$\text{dist}(w) \leq \text{dist}(u) + l(uw) \rightarrow \text{dist}(w) \leq f(u) + l(uw) = f(w),$$ contradiction
Proof of Theorem 1.3

- **Claim 2:** at any iteration \( f(v) = \text{dist}(v) \), for each \( v \in V \setminus U \)

We show: when the algorithm *Selects* \( u \in V \setminus U \) then \( f(u) = \text{dist}(u) \).

Suppose not. Then, \( f(u) > \text{dist}(u) \) for some \( u \) (when selected)

\( s = v_0, v_1, \ldots, v_k = u \) shortest \( s-u \) path.

Let \( i \) smallest such that \( v_i \in U, v_{i-1} \notin U \).

- \( s \in U \rightarrow i = 0, f(s) = 0 = \text{dist}(s) \), contradiction.
- \( s \notin U \rightarrow i > 0, v_i \in U \rightarrow i \leq k \)

\( i > 0 \rightarrow f(v_{i-1}) = \text{dist}(v_{i-1}) \) (by induction, since \( v_{i-1} \in V \setminus U \))

\[
f(v_i) \leq f(v_{i-1}) + l(v_{i-1}, v_i) \quad (v_{i-1} \in V \setminus U)
\]

\[
= \text{dist}(v_{i-1}) + l(v_{i-1}, v_i) = \text{dist}(v_i) \quad (\text{shortest path})
\]

\( \rightarrow f(v_i) = \text{dist}(v_i) \leq \text{dist}(u) < f(u) \) contradicting the choice of \( u \).
Complexity of Dijkstra Algorithm

- The *While* iteration is repeated $|V|$
- The *Select* operation requires at most $|V|$ checks
- The contribution to overall complexity is then $O(|V|^2)$
- Every arc is visited exactly once
- Overall complexity $O(|V|^2) + O(|A|)$. This complexity can be improved when $|A| < |V|^2$
- Improve the *Select* by using heaps to store $f(u)$, $u \in U$

Heap: routed forest $(U,F)$, $uv \in F \rightarrow f(u) \leq f(v)$

Routed Forest: every vertex has indegree at most 1.
Dijkstra algorithm can be applied only when arcs have non-negative lengths (*conservative*).
Otherwise a *shortest walk* may not exist (if $D$ contains a negative length di-cycle).
Observe that if $D$ contains a path from $s$ to $v$, then it contains a *shortest path* from $s$ to $v$.

Finding the *shortest path* with arbitrary arc lengths is difficult (*NP-hard*)
Easy if $D$ contains no negative length di-cycles, but we need a different algorithm (e.g. Bellman-Ford, or Floyd-Warshall)
The s-t path polyhedron

\[ x^P \in \{0,1\}^A \]

*incidence vector of an s-t path* \( P \)

\[
\begin{cases}
    x^P_{uv} = 1 & \text{if } uv \in P \\
    x^P_{uv} = 0 & \text{if } uv \notin P
\end{cases}
\]

\[ P = (s,(s,2),2,(2,4),4,(4,t),t) \]

\[ x^P = \begin{pmatrix}
    s_1 & 0 \\
    s_2 & 1 \\
    s_3 & 0 \\
    1_t & 0 \\
    2_1 & 0 \\
    2_4 & 1 \\
    3_2 & 0 \\
    3_4 & 0 \\
    4_3 & 0 \\
    4_t & 1
\end{pmatrix} \]

- A vector \( x^P \in \{0,1\}^A \) is the incidence vector of an *s-t path* of \( D \) if and only if it satisfies a number of equalities.
**The s-t path polyhedron**

\[ x^P \in \{0,1\}^A \]

*incidence vector of an s-t path* \( P \)

\[
\begin{align*}
 x^P_{uv} & = 1 & \text{if } \ uv \in P \\
 x^P_{uv} & = 0 & \text{if } \ uv \notin P \\
\end{align*}
\]

No arc incoming \( s \). One arc outgoing from \( s \)

\[
\sum_{us \in \delta^-_D(s)} x^P_{us} = 0 \\
\sum_{su \in \delta^+_D(s)} x^P_{su} = 1
\]

One arc incoming \( t \). No arc outgoing from \( t \)

\[
\sum_{ut \in \delta^-_D(t)} x^P_{ut} = 1 \\
\sum_{tu \in \delta^+_D(t)} x^P_{tu} = 0
\]

In every \( v \notin \{s,t\} \)

number of incoming arcs = number of outgoing arcs

\[
\sum_{uv \in \delta^-_D(v)} x^P_{uv} = \sum_{vu \in \delta^+_D(v)} x^P_{vu}
\]
The s-t path polyhedron

\[
\sum_{us \in \delta^-(s)} x^P_{us} - \sum_{su \in \delta^+(s)} x^P_{su} = -1 \quad s
\]

\[
\sum_{ut \in \delta^-(t)} x^P_{ut} - \sum_{tu \in \delta^+(t)} x^P_{tu} = 1 \quad t
\]

\[
\sum_{uv \in \delta^-(v)} x^P_{uv} - \sum_{vu \in \delta^+(v)} x^P_{vu} = 0 \quad \forall v \in V \setminus \{s, t\}
\]

\[M \in \{-1, 0, 1\}^{V \times A}\] be the vertex-arc incidence matrix of \(D\)

\[b = (-1, 1, 0, \ldots, 0)^T\]

The s-t path polyhedron: \(Q_{st} = \{x \in \mathbb{R}^A : Mx = b, \ x \geq 0\}\)
The \( s-t \) path polyhedron

The \( s-t \) path polyhedron: \( Q_{st} = \{ x \in \mathbb{R}^A : Mx=b, \; x \geq 0 \} \)

**Theorem**

The vertices of \( Q_{st} \) are precisely the incidence vectors of the \( s-t \) paths in \( D \).

Consider the following LP:

\[
\begin{align*}
\text{(SP)} \\
& \min l^T x \\
& Mx=b \\
& x \geq 0
\end{align*}
\]

If \( (SP) \) has an optimal solution, then it has an optimal solution which is the incidence vector of an \( s-t \) path.
Dual to the \( s-t \) path problem

\[(SP) \quad \min l^T x \\
\quad Mx = b \\
\quad x \geq 0\]

- if \( D \) has an \( s-t \) path \((SP)\) is non-empty

**Assumption**

\( D \) contains an \( s-v \) path for every \( v \in V \)

- Associate to \((SP)\) its dual problem, by introducing \( y \in \mathbb{R}^V \):

\[(DSP) \quad \max y_t - y_s \\
\quad y_v - y_u \leq l_{uv} \quad \text{for all} \quad uv \in A\]
Since \( SP \) is non-empty, \( SP \) has an optimal solution if and only if it is not unbounded

\( SP \) is not unbounded if and only if \( DSP \) is non-empty.

**Theorem**

\( DSP \) is non-empty iff \( D \) does not contain a negative length directed cycle.
Proof of existence theorem

- **Proof:** If part \((D\) does not contain a negative length dicycle\)

Let \(P^*_u\) be a shortest path from \(s\) to \(u\) in \(D\), \(u \in V\).

Let \(y'_u = l(P^*_u)\), for \(u \in V\). Then \(y'\) is dual feasible. **Suppose not.**

Let \(uv\) such that \(y'_v - y'_u > l_{uv}\)

\[l(P^*_v) - l(P^*_u) > l_{uv}\]

\[l(P^*_v) > l_{uv} + l(P^*_u)\]

If \(v\) does not belong to \(P^*_u = (s, \ldots, u)\)

\(P'=(s, \ldots, u, uv, v)\) is \(s-v\) path with

\[l(P') = l(P^*_u) + l_{uv} < l(P^*_v), \quad \text{contradiction}\]

\(v\) belongs to \(P^*_u = (s, \ldots, v, \ldots, u)\). Let \(P^*_v\) \(s-v\) subpath, \(P'\) \(u-v\) subpath

\[C = P' \cup \{uv\}\] is a cycle

\[l(P^*_v) > l(P^*_v) + l(P') + l_{uv}\]

\[0 > l(P') + l_{uv} = l(C)\]

\(C\) **Negative dicycle ! contradiction**
Proof of existence theorem

**Proof: Only-If part** (if \( y' \) feasible, no negative dicycles in \( D \))

Let \( y' = \) be a feasible dual solution

Let \( C=(1,(1,2),2,\ldots,k,(k,1),1) \) be a negative length dicycle: \( l(C) < 0 \)

\( y' \) feasible implies

\[
\begin{align*}
y'_2 - y'_1 & \leq l_{12} \\
y'_3 - y'_2 & \leq l_{23} \\
\vdots \\
y'_1 - y'_k & \leq l_{k1}
\end{align*}
\]

\[
0 \leq l(C) < 0 \quad \text{contradiction!}
\]
Trees and spanning trees

\[ G = (V, E) \] undirected graph

- \( G \) is a forest if it does not contain a cycle

\[ u, v \in V \] connected if \( G \) contains an \( u-v \) path

\( G \) connected every pair \( u, v \in V \) is connected

- Tree: connected forest

- Every pair of vertices in a tree is connected by a unique path (prove it).
Spanning trees

- $G = (V, E)$ connected undirected graph.
- A tree $H = (W, T)$ is **spanning** $G = (V, E)$ iff $W = V$ and $T \subseteq E$
- Length (weight) function $l: E \rightarrow R$
- Length of $H = (W, T)$: $l(T) := \sum_{e \in T} l(e)$

**Minimum Spanning Tree Problem**

Given a connected undirected graph $G$, and length function $l$, find a spanning tree in $G$ of minimum length

When no confusion arises, forests and trees will be represented by sets of edges.
Dijkstra-Prim spanning tree algorithm

Maintains a tree on a subset of vertices and grows it at each iteration until it becomes spanning

$U$-cut: $U \subseteq V \quad \delta(U): \{uv \in E: u \in U, v \in V/U\}$

**Dijkstra-Prim minimum spanning tree algorithm:**

1. Choose $v_1 \in V$. Set $U_1 = \{v_1\}$. Set $T_1 = \{}$.
2. While $U_k \neq V$
   3. Chose $e_{k+1} \in \delta(U_k)$ with minimum length
   4. Reset $T_{k+1} = T_k \cup \{e_{k+1}\}$; Reset $U_{k+1} = U_k \cup e_{k+1}$
   5. Reset $k = k+1$

EndWhile
Greedy spanning tree algorithm (Kruskal)

- Maintains a forest and grows it at each iteration until it becomes a (spanning) tree

**Greedy minimum spanning tree algorithm (Kruskal):**

1. Set $T_0 = \{\}$.
2. For $k=1,\ldots,|V|-1$
   3. Choose $e_k$ such that:
      
      $T_{k-1} \cup \{e_k\}$ is a forest and $l(e_k)$ is minimum
   4. Reset $T_k = T_{k-1} \cup \{e_{k+1}\}$;

EndFor

The proof of correctness for both algorithms is based on the properties of the *greedy forests*. 
A forest $F$ is greedy if there exists a minimum-length spanning tree $T$ of $G$ that contains $F$.

Let $F$ be a greedy forest, let $U$ be one of its components, and let $e \in \delta(U)$. If $e$ has minimum length among all edges in $\delta(U)$, then $F \cup \{e\}$ is again a greedy forest.
Proof of Theorem 1.11

- $T$ minimum-length spanning tree that contains $F$.
  - $P$ unique path between end vertices of $e \in \delta(U)$
  - $P$ contains at least an edge $f \in \delta(U)$
  - $T' = T \setminus \{f\} \cup \{e\}$ is a spanning tree (prove it).
  - $l(e) \leq l(f) \rightarrow l(T') \leq l(T)$ and $T'$ is a minimum length spanning tree
  - $F \cup \{e\}$ does not contain cycles (forest)
  - $F \cup \{e\} \subseteq T'$ implies $F \cup \{e\}$ greedy forest
The Dijkstra-Prim method and the Kruskal method yield a spanning tree of minimum length.

- At the first stage of the algorithms $T_0$ is a greedy forest.
- At each subsequent stage $k$, $T_k$ is a greedy forest.
- After $|V|-1$ steps the algorithms terminate with a spanning tree.
Exercises

- Show that every pair of vertices in a tree is connected by a unique path.

- Show that if \( G = (V, T) \) is a tree, then \(|T| = |V| - 1\).

- Show that if \( T \) is a spanning tree of \( G = (V, E) \) and \( f \in E \setminus T \), then \( T \cup f \) contains a unique cycle \( C \) (called fundamental). Show that if \( e \in C \), then \( T \setminus \{e\} \cup f \) is a spanning tree.

- Let \( D \) be a directed graph, and \( M \) be the corresponding vertex-arc incidence matrix. Show that a set of independent columns of \( M \) corresponds to the edges of a forest.

- Show formally that the 0,1 solutions of the \( s-t \) path polyhedron are precisely the incidence vectors of the \( s-t \) paths.