Lecture 5: finding integer solutions for IPs

- Illustration of Dijkstra’s shortest path algorithm
- Possible implementation of Dijkstra’s algorithm
- Combinatorial optimization algorithms
- Polynomial time algorithms: illustration
- LP rounding
- Illustration of LP rounding: scheduling

Start from the origin node
Assign infinity to non-connected nodes
- Compute the distance of each node to the set of all considered nodes
- Pick node with lowest computed distance not picked already; this becomes part of the set of considered nodes
- Update shortest paths
- Loop
Start from the origin node
Assign infinity to non connected nodes
- Compute the distance of each node to the set of all considered nodes
- Pick node with lowest computed distance not picked already; this becomes part of the set of considered nodes
- Update shortest paths
- Loop

Start from the desired node, called s.
Set shortest path value at this node equal to zero.
For every other node, set shortest path value to:
- distance between this node and s if connected
- \(-\infty\) distance if not connected
Set of considered nodes := s
While set of considered nodes is not equal to graph, loop:
find closest node to set of considered nodes
add it to set of considered nodes
update shortest path for all nodes not in set of considered nodes
Stop when all nodes are in set of considered nodes.
Possible implementation of Dijkstra's algorithm

begin
S := Ø
\( d[u] := \infty \) for each node \( i \) in the graph
\( d[s] = 0 \) and \( pred[s] = 0 \)
S := \{ s \}
while \( \text{not done} \) do
begin
let \( u \in S^c \) for which \( d[u] = \min(d[v]) \), \( v \in S \)
S := S U \{ u \}
S^c := S \backslash \{ u \}
for each \( v \) in the graph do
if \( d[u] + c_{uv} < d[v] \) then
\( d[v] := d[u] + c_{uv} \)
\( pred[v] := u \)
end
end

Possible implementation of Dijkstra's algorithm

begin
S := Ø
\( d[u] := \infty \) for each node \( i \) in the graph
\( d[s] = 0 \) and \( pred[s] = 0 \)
S := \{ s \}
while \( \text{not done} \) do
begin
let \( u \in S^c \) for which \( d[u] = \min(d[v]) \), \( v \in S \)
S := S U \{ u \}
S^c := S \backslash \{ u \}
for each \( v \) in the graph do
if \( d[u] + c_{uv} < d[v] \) then
\( d[v] := d[u] + c_{uv} \)
\( pred[v] := u \)
end
end

Possible implementation of Dijkstra's algorithm

begin
S := Ø
\( d[u] := \infty \) for each node \( i \) in the graph
\( d[s] = 0 \) and \( pred[s] = 0 \)
S := \{ s \}
while \( \text{not done} \) do
begin
let \( u \in S^c \) for which \( d[u] = \min(d[v]) \), \( v \in S \)
S := S U \{ u \}
S^c := S \backslash \{ u \}
for each \( v \) in the graph do
if \( d[u] + c_{uv} < d[v] \) then
\( d[v] := d[u] + c_{uv} \)
\( pred[v] := u \)
end
end

Possible implementation of Dijkstra's algorithm

begin
S := Ø
\( d[u] := \infty \) for each node \( i \) in the graph
\( d[s] = 0 \) and \( pred[s] = 0 \)
S := \{ s \}
while \( \text{not done} \) do
begin
let \( u \in S^c \) for which \( d[u] = \min(d[v]) \), \( v \in S \)
S := S U \{ u \}
S^c := S \backslash \{ u \}
for each \( v \) in the graph do
if \( d[u] + c_{uv} < d[v] \) then
\( d[v] := d[u] + c_{uv} \)
\( pred[v] := u \)
end
end
Features of Dijkstra’s algorithm

Simplest Dijkstra: labels the nodes as the set S is increased, and assigns a cost \( d(i) \) to node \( i \) for all \( i \) in S.

Dijkstra with predecessor: also keeps track of the predecessor of node \( i \), called \( \text{pred}(i) \), as the set S is grown.

No particular need for \( t \): starts from \( s \), and grows: finds the shortest path from \( s \) to any node in the set S at a particular time.

Once S spans the whole graph, the algorithm returns the shortest path from \( s \) to any node.

Once a node \( n \) is in the set S, the value \( d(n) \) is the shortest distance from \( s \) to \( n \). Therefore, if one is only interested in finding the shortest path from \( s \) to \( n \), the algorithm can be stopped as soon as \( n \) is contained in S.

Example of LP rounding: aircraft scheduling

Use it to reconstruct a physical solution