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

Illustration of Dijkstra’s algorithm

Start from the origin node
Assign infinity to non connected nodes

- Compute the distance of each node to the set of all considered nodes

Illustration of Dijkstra’s algorithm

Start from the origin node
Assign infinity to non connected nodes

- Compute the distance of each node to the set of all considered nodes

Illustration of Dijkstra’s algorithm

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
Illustration of Dijkstra’s algorithm

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

Illustration of Dijkstra’s algorithm

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
Illustration of Dijkstra’s algorithm

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
Illustration of Dijkstra's algorithm

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

!!! Upper and right corner values changes !!!
Illustration of Dijkstra’s algorithm

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
Illustration of Dijkstra’s algorithm

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

Summary of Dijkstra’s algorithm

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(i):=∞ for each node i
    d(s):=0 and pred(s)=0
    S:= { s }
    while |S|<n do
        begin
            let i in S* for which d(i)=min{d(j), j in S*}
            S = S U {i}
            S* = S* \ {i}
            for each (i,j) in the graph do
                if d(j)>d(i)+cij
                    then
                        d(j):=d(i)+cij
                        pred(j):=i
            end
        end
    end
end

Possible implementation of Dijkstra’s algorithm

begin
    S:=Ø
    d(i):=∞ for each node i
    d(s):=0 and pred(s)=0
    S:= { s }
    while |S|<n do
        begin
            let i in S* for which d(i)=min{d(j), j in S*}
            S = S U {i}
            S* = S* \ {i}
            for each (i,j) in the graph do
                if d(j)>d(i)+cij
                    then
                        d(j):=d(i)+cij
                        pred(j):=i
            end
        end
    end
end
Possible implementation of Dijkstra’s algorithm

\[
\begin{align*}
\text{begin} \\
S &= \emptyset \\
d(i) &= +\infty \text{ for each node } i \\
d(s) &= 0 \text{ and } \text{pred}(s) = 0 \quad \text{Assign zero length to source node} \\
S &= \{ s \} \\
\text{while } |S| < n \text{ do} \\
& \quad \text{begin} \\
& \quad \quad \text{let } i \text{ in } S^* \text{ for which } d(i) = \min\{d(j), \text{ in } S^*\} \\
& \quad \quad S = S \cup \{i\} \\
& \quad \quad S^* = S^* \setminus \{i\} \\
& \quad \quad \text{for each } (i, j) \text{ in the graph do} \\
& \quad \quad \quad \text{if } d(j) > d(i) + c_{ij} \\
& \quad \quad \quad \quad \text{then} \\
& \quad \quad \quad \quad d(j) = d(i) + c_{ij} \\
& \quad \quad \quad \quad \quad \text{pred}(j) = i \\
& \quad \quad \text{end} \\
& \quad \text{end} \\
& \text{end}
\end{align*}
\]

Possible implementation of Dijkstra’s algorithm

\[
\begin{align*}
\text{begin} \\
S &= \emptyset \\
d(i) &= +\infty \text{ for each node } i \\
d(s) &= 0 \text{ and } \text{pred}(s) = 0 \\
S &= \{ s \} \quad \text{Initialize set to source node} \\
\text{while } |S| < n \text{ do} \\
& \quad \text{begin} \\
& \quad \quad \text{let } i \text{ in } S^* \text{ for which } d(i) = \min\{d(j), \text{ in } S^*\} \\
& \quad \quad S = S \cup \{i\} \\
& \quad \quad S^* = S^* \setminus \{i\} \\
& \quad \quad \text{for each } (i, j) \text{ in the graph do} \\
& \quad \quad \quad \text{if } d(j) > d(i) + c_{ij} \\
& \quad \quad \quad \quad \text{then} \\
& \quad \quad \quad \quad d(j) = d(i) + c_{ij} \\
& \quad \quad \quad \quad \quad \text{pred}(j) = i \\
& \quad \quad \text{end} \\
& \quad \text{end} \\
& \text{end}
\end{align*}
\]
Possible implementation of Dijkstra’s algorithm

begin
S:=∅
d(i):=+∞ for each node i
d(s):=0 and pred(s)=0
S:= { s }

while |S|<n do
    While there is still some nodes left
        begin
            let i in S* for which d(i)=min(d(j), j in S*)
            S = S ∪ {i}
            S* = S* \ {i}

            for each (i,j) in the graph do
                if d(j)>d(i)+c_{ij}
                    then
                        d(j):=d(i)+c_{ij}
                        pred(j):=i

        end

    end

end

Possible implementation of Dijkstra’s algorithm

begin
S:=∅
d(i):=+∞ for each node i
d(s):=0 and pred(s)=0
S:= { s }

while |S|<n do
    begin
        let i in S* for which d(i)=min(d(j), j in S*)
        S = S ∪ {i}
        S* = S* \ {i}

        for each (i,j) in the graph do
            if d(j)>d(i)+c_{ij}
                then
                    d(j):=d(i)+c_{ij}
                    pred(j):=i

    end

end
Possible implementation of Dijkstra’s algorithm

begin
S:=∅
d(i):=+∞ for each node i
d(s):=0 and pred(s)=0
S:= { s }
while |S|<n do
begin
  let i in S* for which d(i)=min{d(j), j in S*}
  S = S U {i}
  S* = S* \ {i}
  for each (i,j) in the graph do
    if d(j)>d(i)+c_{ij}
      then
        d(j):=d(i)+c_{ij}
        pred(j):=i
  end
end
end

Possible implementation of Dijkstra’s algorithm

begin
S:=∅
d(i):=+∞ for each node i
d(s):=0 and pred(s)=0
S:= { s }
while |S|<n do
begin
  let i in S* for which d(i)=min{d(j), j in S*}
  S = S U {i}
  S* = S* \ {i}
  for each (i,j) in the graph do Loop to relabel nodes around the pink set
    if d(j)>d(i)+c_{ij}
      then
        d(j):=d(i)+c_{ij}
        pred(j):=i
  end
end
end
Possible implementation of Dijkstra's algorithm

begin
    S := Ø
    d(i) := +∞ for each node i
    d(s) := 0 and pred(s) := 0
    S := { s }
    while |S| < n do
        begin
            let i in S for which d(i) = min{d(j) j in S'}
            S := S U {i}
            S' := S' \ {i}
            for each (i,j) in the graph do
                if d(j) > d(i) + c_{ij}
                then
                    d(j) := d(i) + c_{ij}
                    pred(j) := i
            end
        end
    end

Possible implementation of Dijkstra's algorithm

begin
    S := Ø
    d(i) := +∞ for each node i
    d(s) := 0 and pred(s) := 0
    S := { s }
    while |S| < n do
        begin
            let i in S for which d(i) = min{d(j) j in S'}
            S := S U {i}
            S' := S' \ {i}
            for each (i,j) in the graph do
                if d(j) > d(i) + c_{ij}
                then
                    d(j) := d(i) + c_{ij}
                    pred(j) := i
            end
        end
    end
Possible implementation of Dijkstra’s algorithm

begin
    S := ∅
    d(i) := +∞ for each node i
    d(s) := 0 and pred(s) = 0
    S := { s }
    while |S| < n do
        let i in S* for which d(i) = min{d(j), j in S*}
        S := S U {i}
        S* := S* \ {i}
        for each (i, j) in the graph do
            if d(j) > d(i) + c_{ij} then
                d(j) := d(i) + c_{ij}
                pred(j) := i
        if it is shorter to go to node j through link (i, j) than computed previously then set this as the shortest path to node j.
        Set the predecessor of j to be i; if you want to go to j, the shortest path is through i
    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 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

\[
\min: \sum_{i,j} \theta_i x_{ij}
\]
\[
s.t.: \sum_i x_{ij} = 1
\]
\[
x_{ij} \geq 0
\]
\[
\sum_{i' \in I(i)} \sum_j x_{i'j} \leq 1
\]

Use it to reconstruct a physical solution