SIEMENS

# Multi-path Link-state Routing

2012-01-16

IEEE 802.1 AVB TG Meeting

January 2012, Munich, Germany

Michael Bahr, Siemens AG

Franz-Josef Götz, Siemens AG

SIEMENS

# Overview

## Single Path Routing

- Link State Routing / Dijkstra‘s Algorithm
- Link State Routing in IS-IS

## Multi Path Routing

- Edge Disjoint vs. Node Disjoint
- Simple 2 Step Approach
- Problems of Simple 2 Step Approach
- Correct Approach
  - Edge Disjoint Shortest Pair Algorithm
  - Node Disjoint Shortest Pair Algorithm

## Single Path Routing

The Task:

- In general:
  Find a path from source S to destination D in a network.
- More specific:
  Find the shortest path (= best path) from source S to destination D in a network with respect to a specific link metric.

Link State:

- two endpoints of link
- link metric (often called „link cost“, „link weight“, or „link length“)
- direction
- different, equivalent ways of representation

# Dijkstra's Algorithm

SIEMENS

| | |
|---|---|
| `N` | set of nodes in network |
| `Nb(i)` | set of neighbor nodes of node `i` |
| `d(i)` | distance from node `i` to source `S` (= sum of the link metrics from `S` to `i`) |
| `l(ij)` | metric of link from node `i` to node `j` (`i` and `j` are neighbors) |
| `P(i)` | predecessor of node `i` on path from `S` to `i` |
| `U` | set of nodes for which the shortest path from `S` has not yet been found |

**1. Initialization**

```
d(S) := 0; d(i) := l(Si) if i∈Nb(S), d(i) := ∞ otherwise;
U := N - {S};
```

**2. Selection of next node**

```
j := j∈U with d(j) = min d(k) ∀k ∈ U;
U := U - { j };
if j == D then STOP; // if U == ∅ then STOP
```

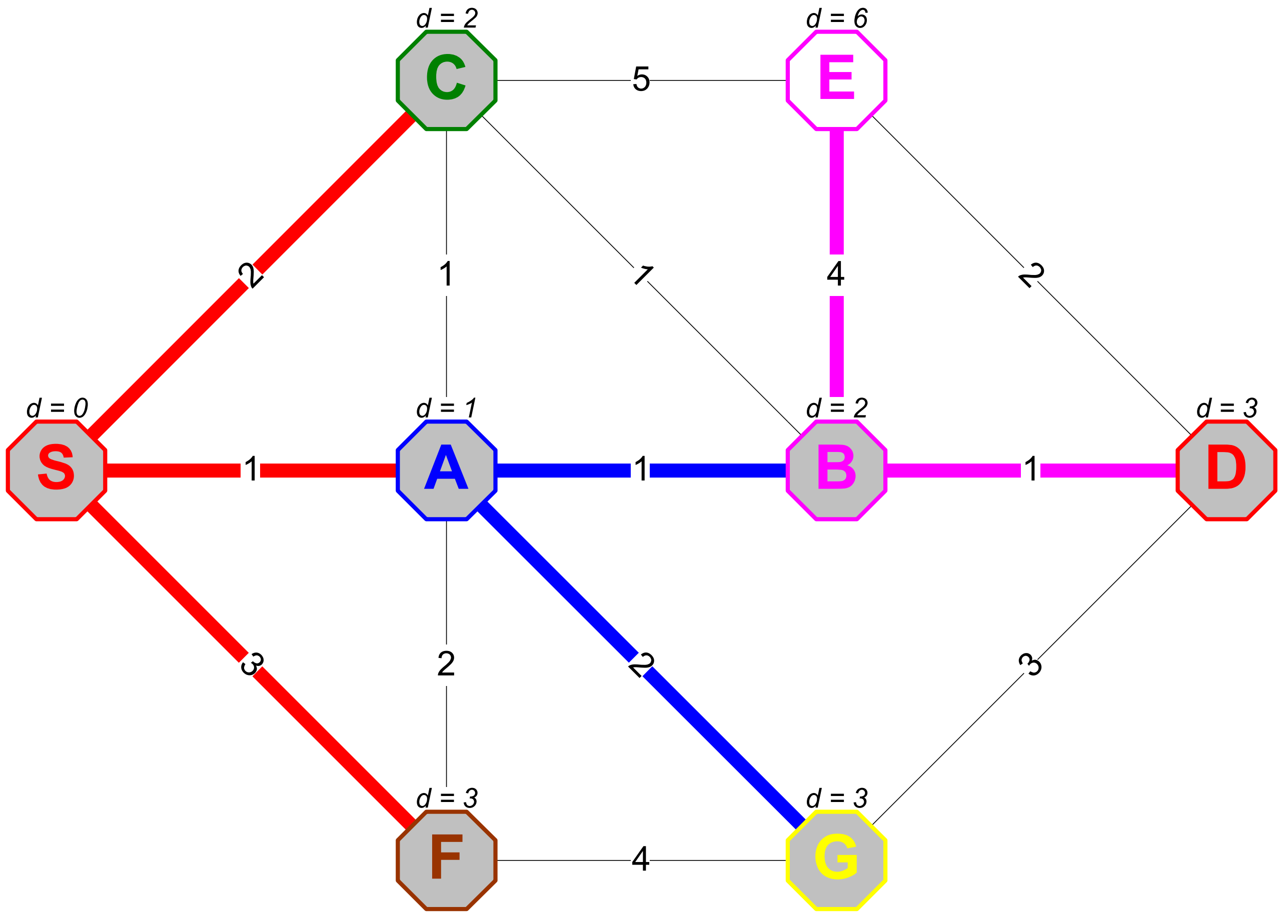
**3. Update of distances and predecessors**

```
∀k ∈ (Nb(j) ∩ U) if d(j) + l(jk) < d(k) then
        d(k) := d(j) + l(jk);
        P(k) := j;
goto 2.
```

Dijkstra's Algorithm first described in [1]
description based on [2]

SIEMENS

# Dijkstra‘s Algoritm – An Example




topology based on [2]

## Link-state Routing Algorithm in IS-IS

Three Lists

- Unknown: all links not processed yet, corresponds to $i \in U$
- Tentative: all links with nodes currently processed, corresponds to $i \in U$ with $d(i) < \infty$
- Paths or Known: all processed nodes and predecessors, corresponds to $i \notin U$

(1) Start with root node as node under consideration

(2) Move all link states containing the considered node from Unknown list to Tentative list, if Tentative list is empty STOP

(3) Select link state with best cost to root from tentative list and add new node to Paths list and make the new node the considered node

(4) Delete link state with worse cost to root for the same new node from Tentative list

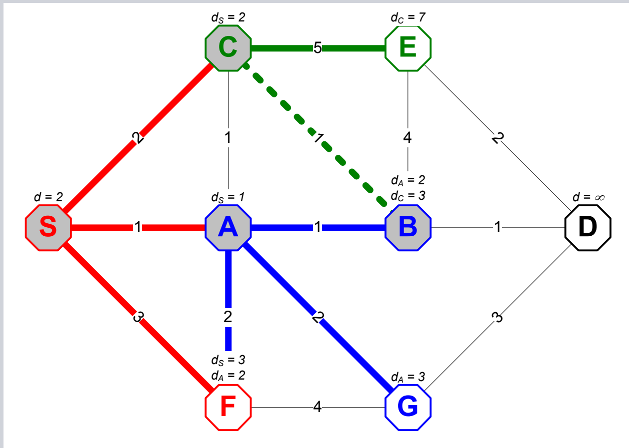
(5) go back to (2)

# Link-State Routing Algorithm in IS-IS – An Example

| UNKNOWN From --> To | Link Metric |
|---|---|
| ~~C --> E~~ | ~~5~~ |
| ~~C --> B~~ | ~~1~~ |
| F --> G | 4 |
| E --> D | 2 |
| E --> B | 4 |
| ~~E --> C~~ | ~~5~~ |
| B --> D | 1 |
| ~~B --> C~~ | ~~1~~ |
| B --> E | 4 |
| G --> F | 4 |
| G --> D | 3 |
| D --> G | 3 |
| D --> B | 1 |
| D --> E | 2 |

| TENTATIVE From --> To | Link Metric | Metric to Root |
|---|---|---|
| S --> F | 3 | 3 |
| ~~A --> B~~ | ~~1~~ | ~~2~~ |
| A --> G | 2 | 3 |
| A --> F | 2 | 3 |
| C --> E | 5 | 7 |
| ~~C --> B~~ | ~~1~~ | ~~3~~ |

| Paths Destination | Next Hop | Path Metric |
|---|---|---|
| A | A | 1 |
| C | C | 2 |
| B | A | 2 |



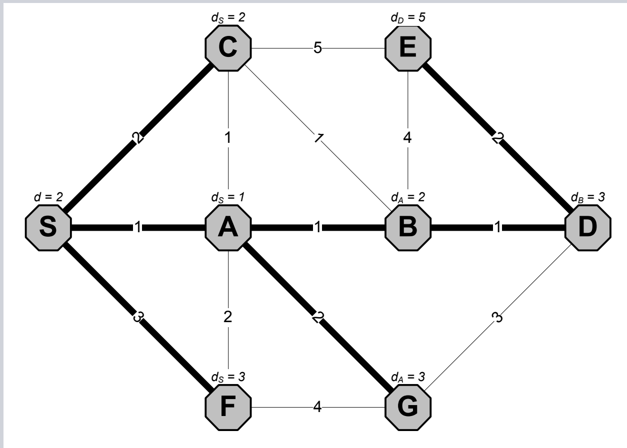

topology based on [2]
lists based on [3]

# Link-State Routing Algorithm in IS-IS – An Example

| UNKNOWN | |
|---|---|
| From --> To | Link Metric |
| | |

| TENTATIVE | | |
|---|---|---|
| From --> To | Link Metric | Path Metric |
| | | |

| Paths | | |
|---|---|---|
| Destination | Next Hop | Path Metric |
| A | A | 1 |
| C | C | 2 |
| B | A | 2 |
| F | F | 3 |
| G | A | 3 |
| D | A | 3 |
| E | A | 5 |




topology based on [2]
lists based on [3]

SIEMENS

# Multi Path Routing

The Task:

- In general:
  Find *m* multiple paths from source S to destination D in a network (if they exist).
- More specific:
  Find the shortest set of *m* paths (= set of *m* paths with best total value) from source S to destination D in a network with respect to a specific link metric (if they exist).
- The favourite: $m = 2$

Edge Disjoint:

- no shared links (edges)
- copes with excavators, spades, digging, link breaks, …

Node Disjoint:

- no shared intermediate nodes (source S and destination D are shared)
- copes with node failures

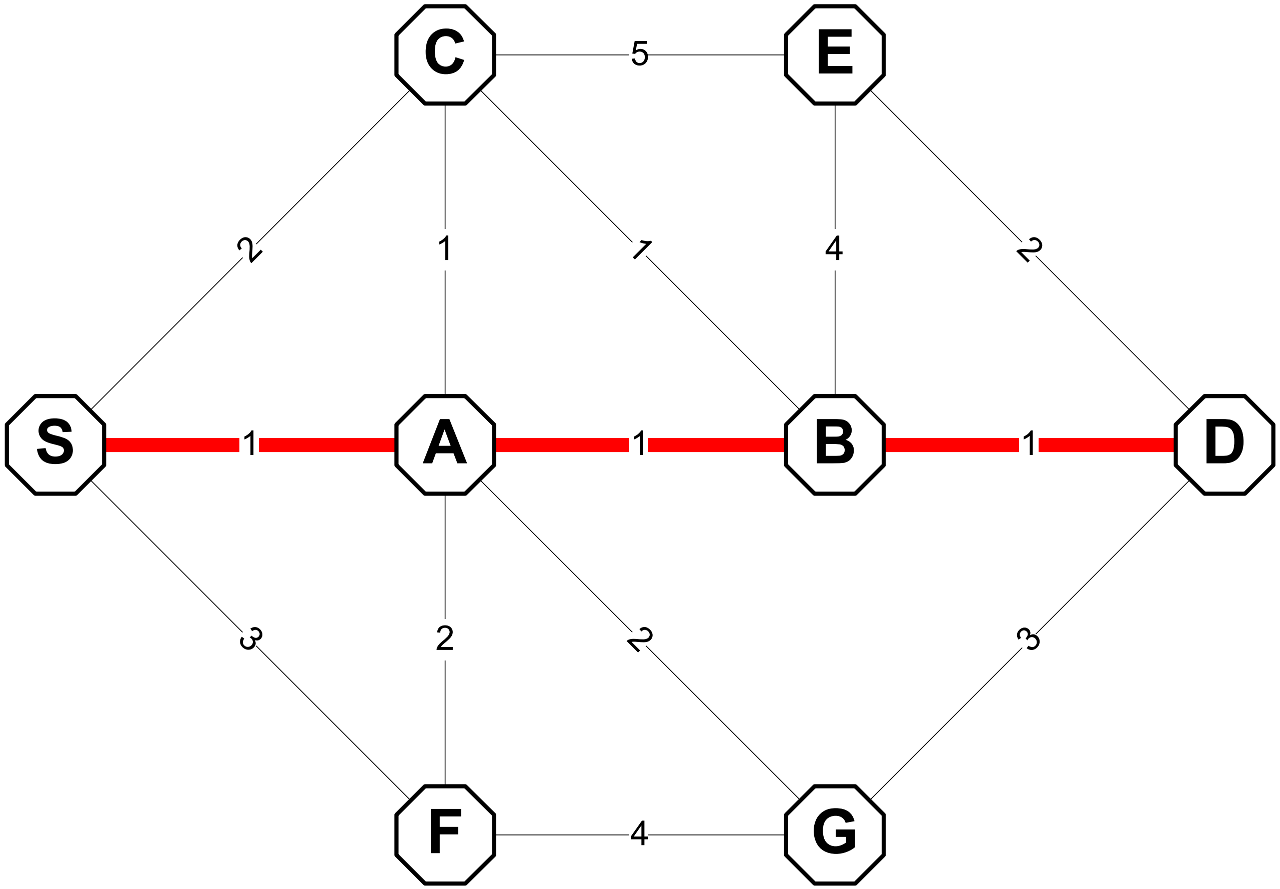
## Simple 2 Step Approach

1. Use Dijkstra‘s Algorithm to find a first shortest path.
2. Edge-disjoint: remove links of found shortest path from network topology.
   Node-disjoint: remove intermediate nodes of found shortest path and links connecting them from network topology.
3. Use Dijkstra‘s Algorithm on pruned network topology to find a second shortest path.

description based on [2]

# Simple 2 Step Approach
# Edge Disjoint – An Example

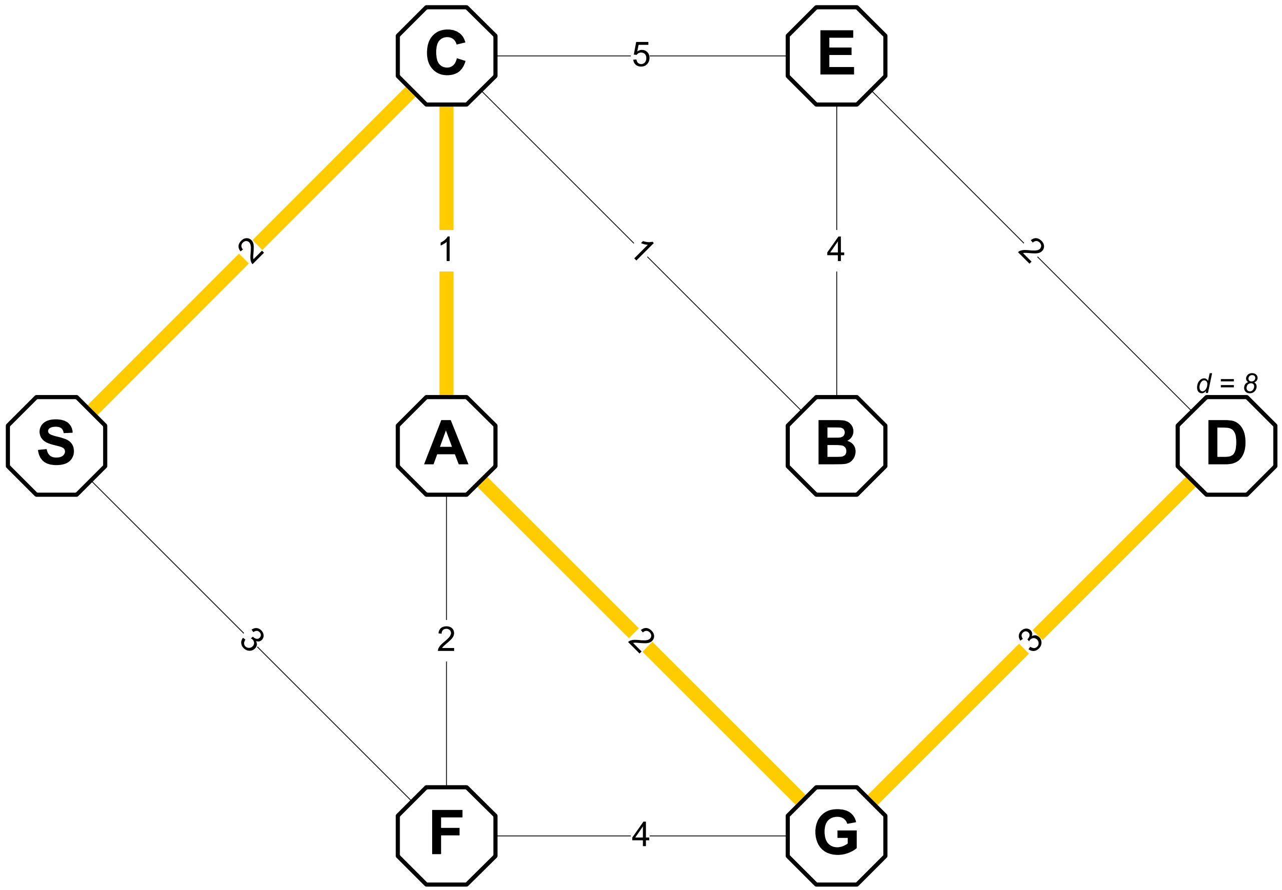
SIEMENS




topology based on [2]

# Simple 2 Step Approach
# Edge Disjoint – An Example

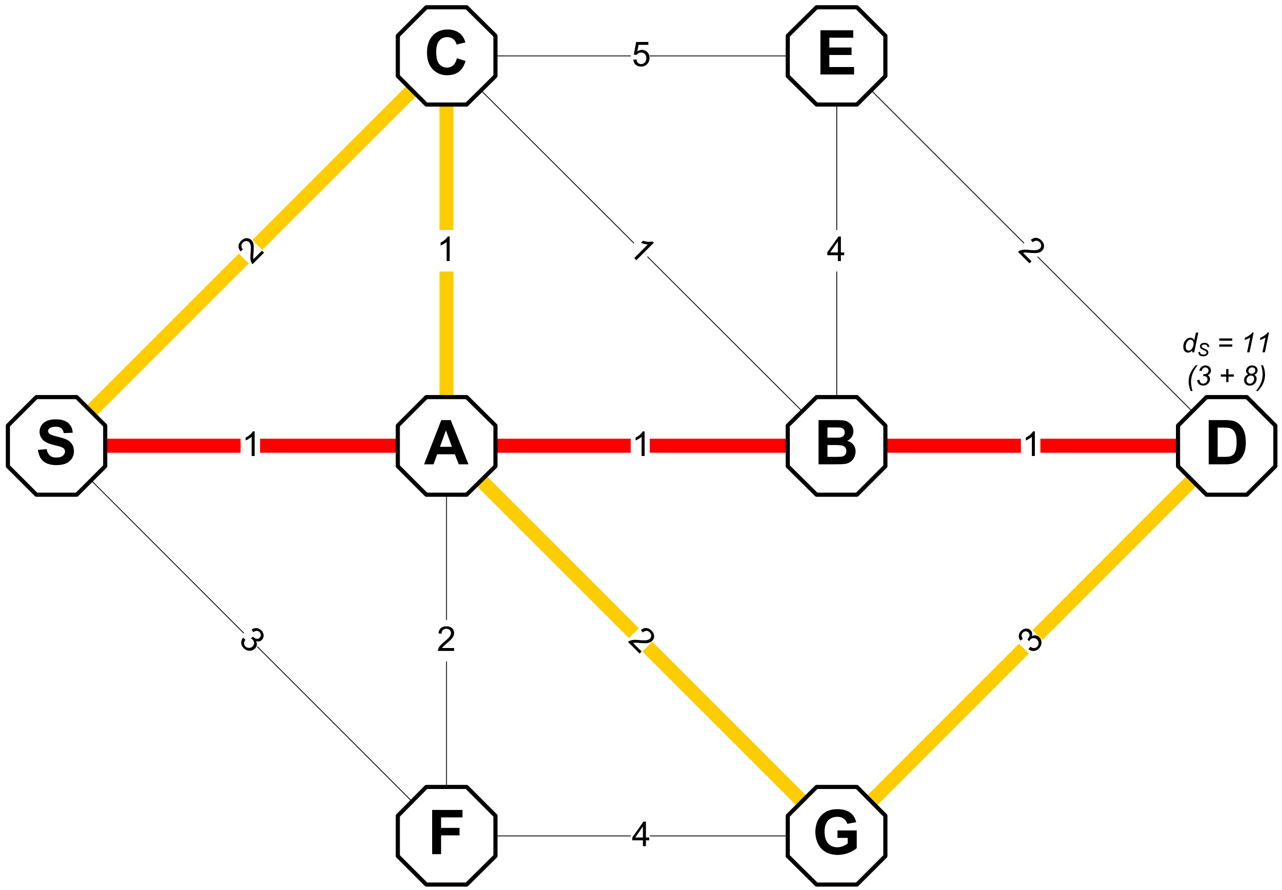
SIEMENS




topology based on [2]

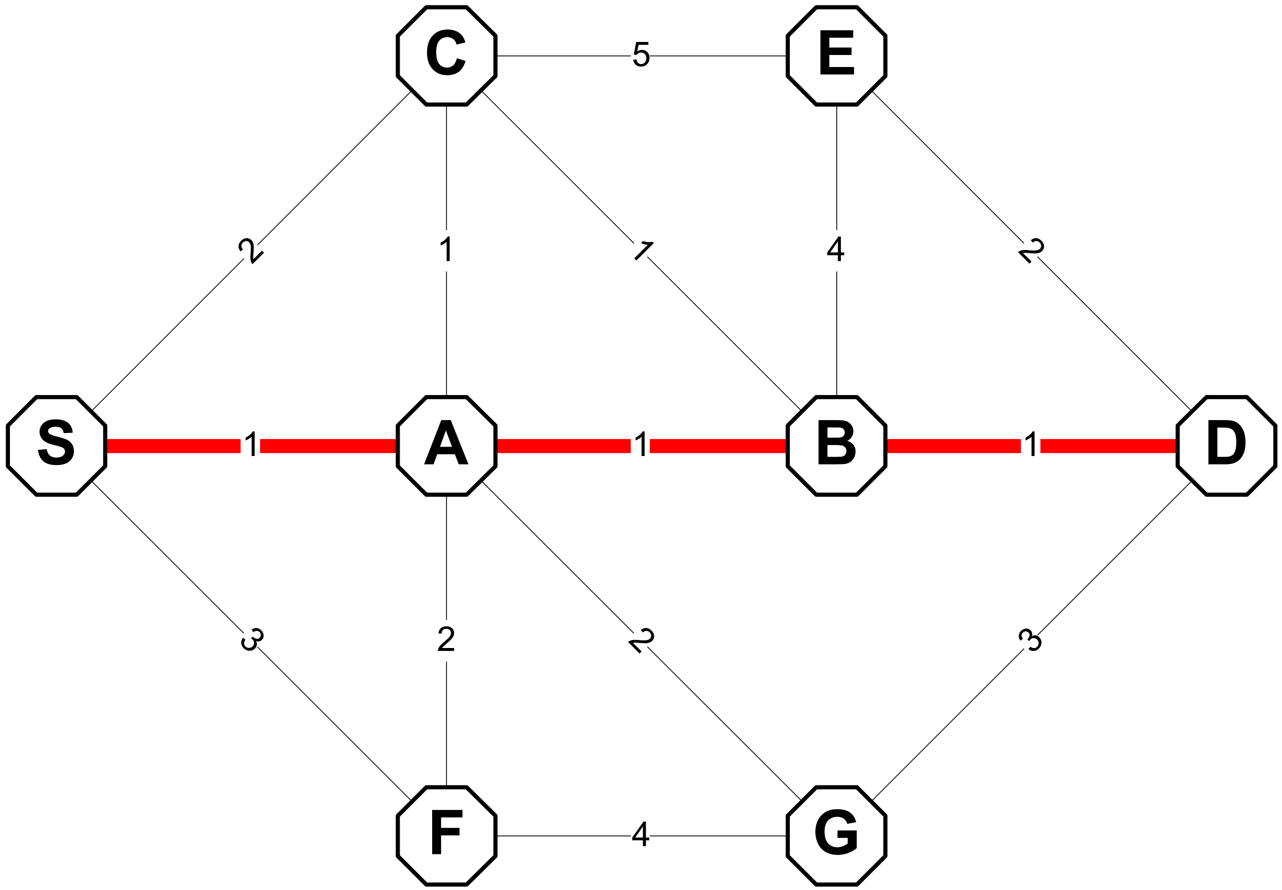
# Simple 2 Step Approach
# Edge Disjoint – An Example

SIEMENS




topology based on [2]

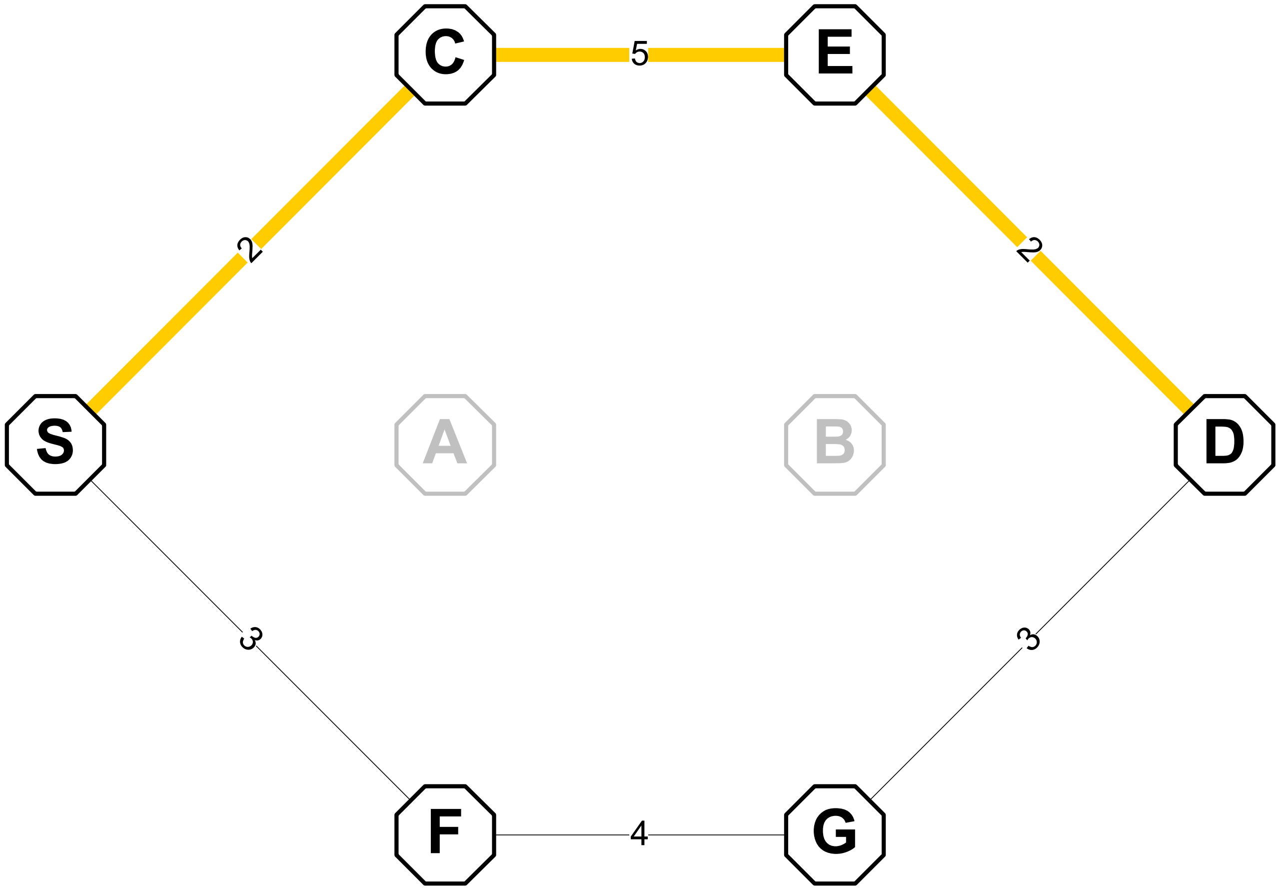
# Simple 2 Step Approach
# Node Disjoint – An Example




topology based on [2]

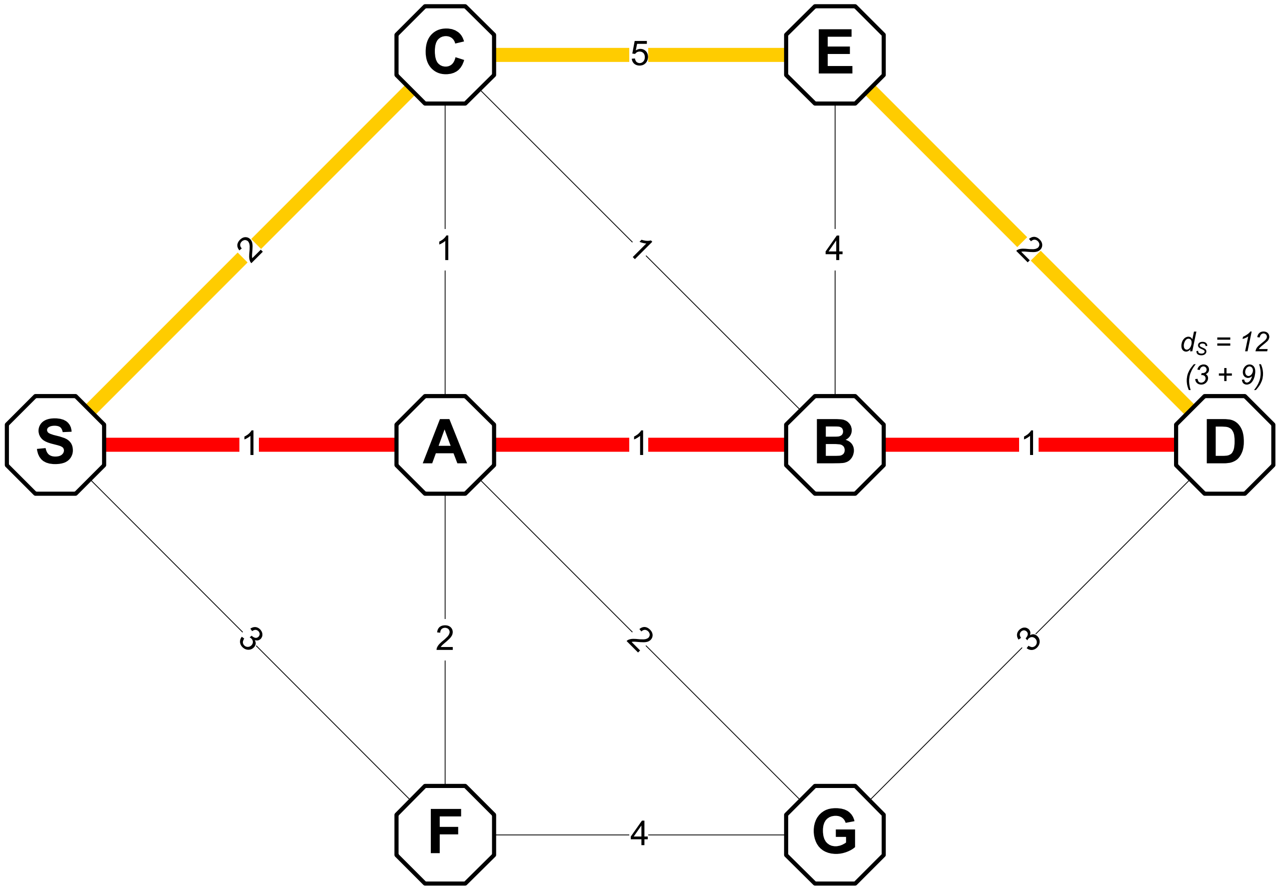
# Simple 2 Step Approach
# Node Disjoint – An Example

SIEMENS




topology based on [2]

# Simple 2 Step Approach
# Node Disjoint – An Example




topology based on [2]

# Problems with Simple 2 Step Method

Sub-Optimality:

- more than one set of $m$>1 edge-disjoint / node-disjoint paths might exist in a network
- Simple 2 Step Method might *NOT* find the set of $m$>1 paths with least total length

False Alarms:

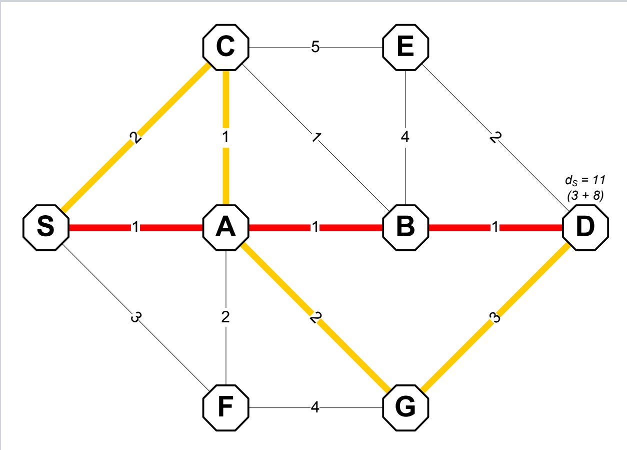
- Simple 2 Step Method might *NOT* find a set of $m$>1 edge-disjoint / node-disjoint paths even if it exists
- failure to find existing $m$>1 edge-disjoint / node-disjoint paths

Observation: The „1 → n“ problem:

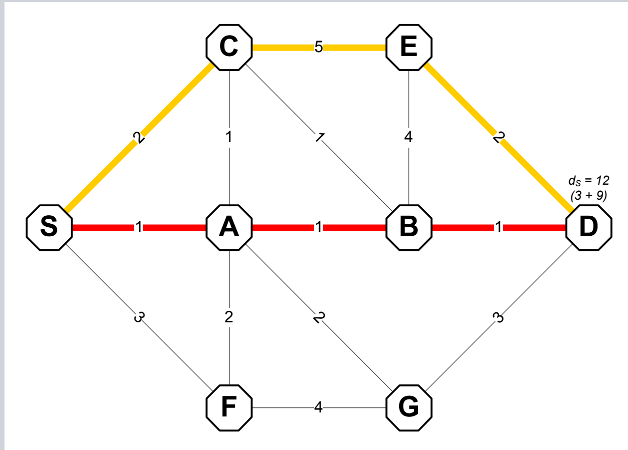
  - $m$ = 1: shortest path $\subseteq$ set of $m$=1 paths with least total length
  - m > 1: shortest path not necessarily part of set of $m$>1 paths with least total length

# Sub-Optimality of Simple 2 Step Method

Best found pair of edge-disjoint paths is $d_S = 11$

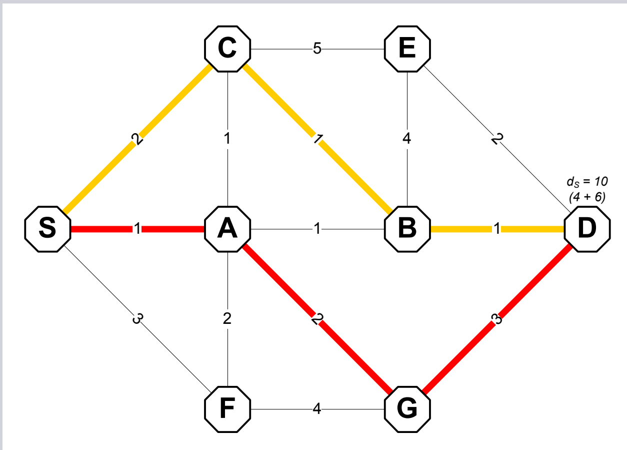



Best found pair of node-disjoint paths is $d_S = 12$




**But:**

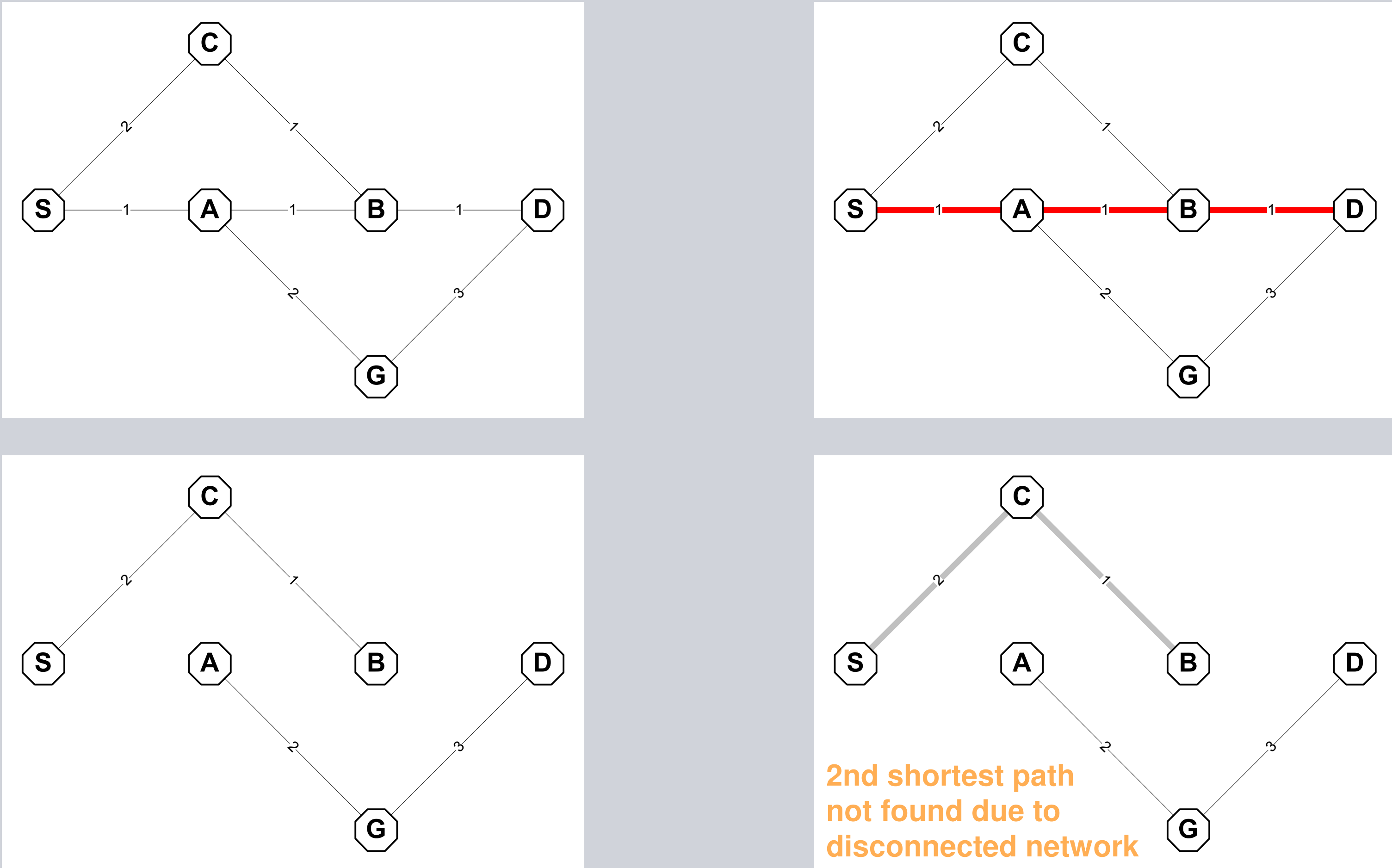
**The really best pair of edge-disjoint and node-disjoint paths is $d_S = 10$.**




topology based on [2]

# Failure to Find Existing Multiple Paths With Simple 2 Step Method

SIEMENS



## Edge-Disjoint Shortest Pair Algorithm

1. Find a first shortest path.
2. Replace each edge of the shortest path with a unidirectional edge directed towards the source and with its metric made negative.
3. Find a second shortest path on this modified network topology
   - need a modified Dijkstra‘s Algorithm that can handle loop-free directed negative edges
4. Transform edges to original network topology (bidirectional and positive metric)
5. Delete edges common to both found shortest paths and regroup remaining edges to shortest pair of edge-disjoint paths.

description based on [2]

# Modified Dijkstra‘s Algorithm

SIEMENS

| | |
|---|---|
| `N` | set of nodes in network |
| `Nb(i)` | set of neighbor nodes of node `i` |
| `d(i)` | distance from node `i` to source `S` (= sum of the link metrics from `S` to `i`) |
| `l(ij)` | metric of link from node `i` to node `j` (`i` and `j` are neighbors) |
| `P(i)` | predecessor of node `i` on path from `S` to `i` |
| `U` | set of nodes for which the shortest path from `S` has not yet been found |

**1. Initialization**

```
d(S) := 0; d(i) := l(Si) if i∈Nb(S), d(i) := ∞ otherwise;
U := N – {S};
```

**2. Selection of next node**

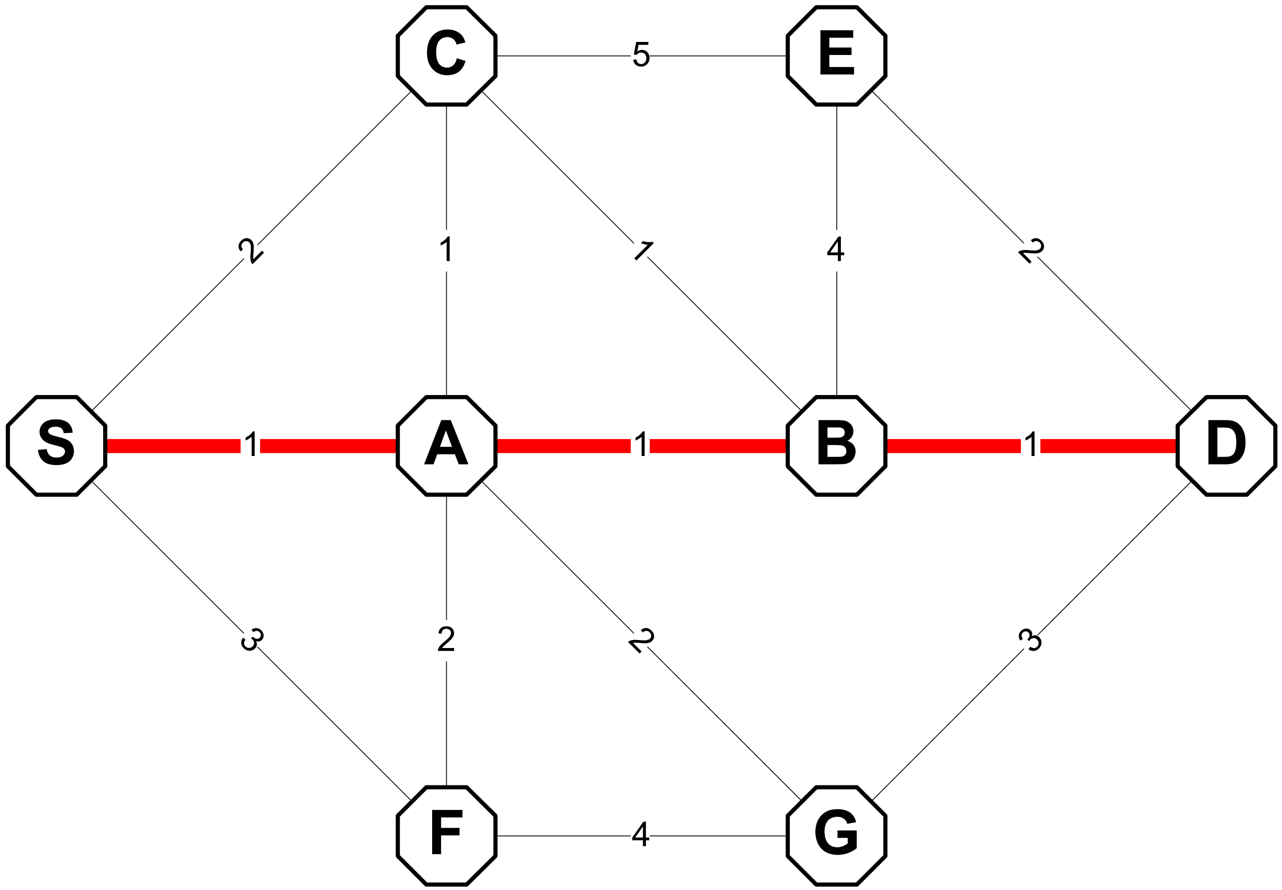
```
j := j∈U with d(j) = min d(k) ∀k ∈ U;
U := U – { j };
if j == D then STOP; // if U == ∅ then STOP
```

**3. Update of distances and predecessors**

```
∀k ∈ Nb(j) if d(j) + l(jk) < d(k) then
     d(k) := d(j) + l(jk);
     P(k) := j;
     U := U ∪ { k };
goto 2.
```

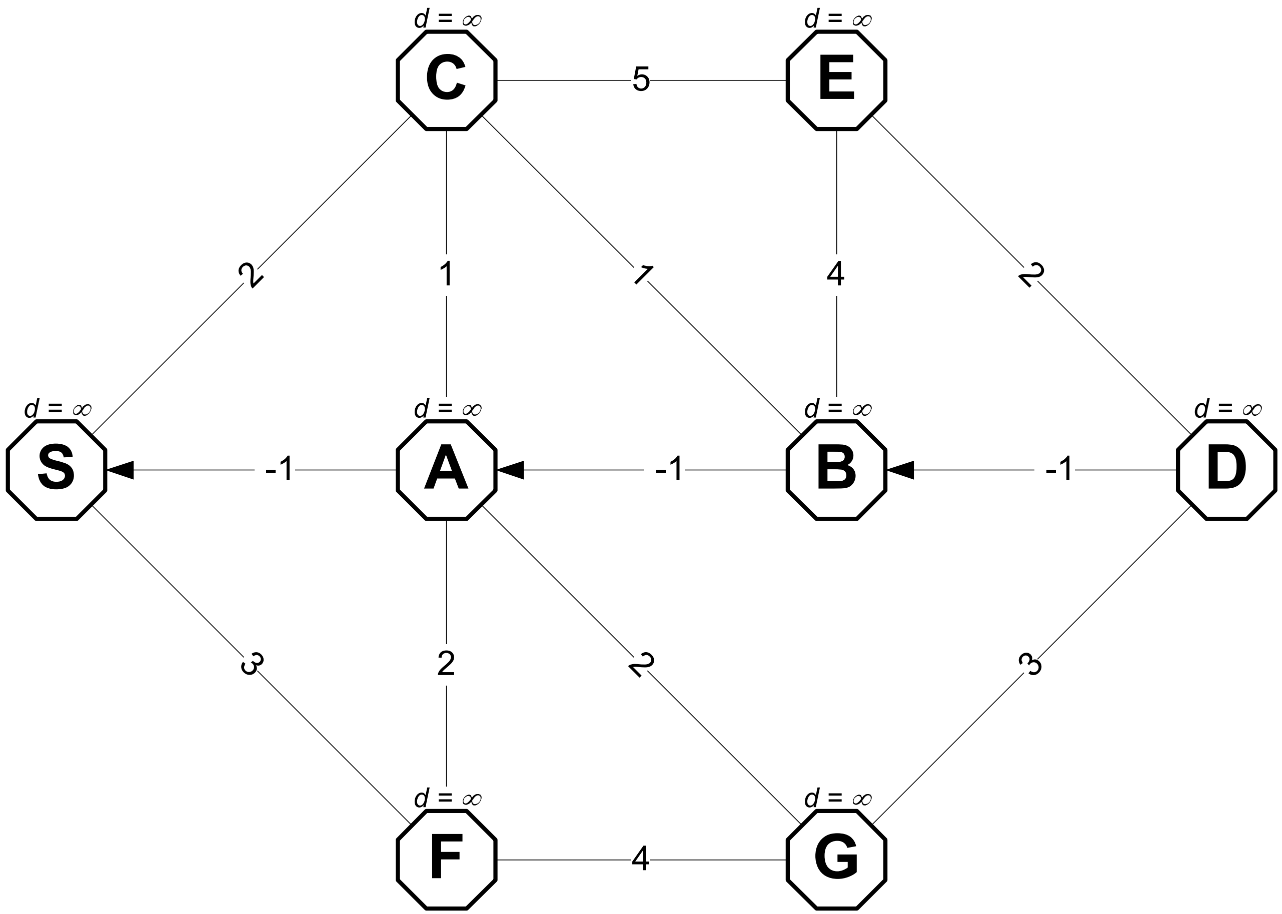
from [2]
description based on [2]

# Edge Disjoint Shortest Pair Algorithm – An Example



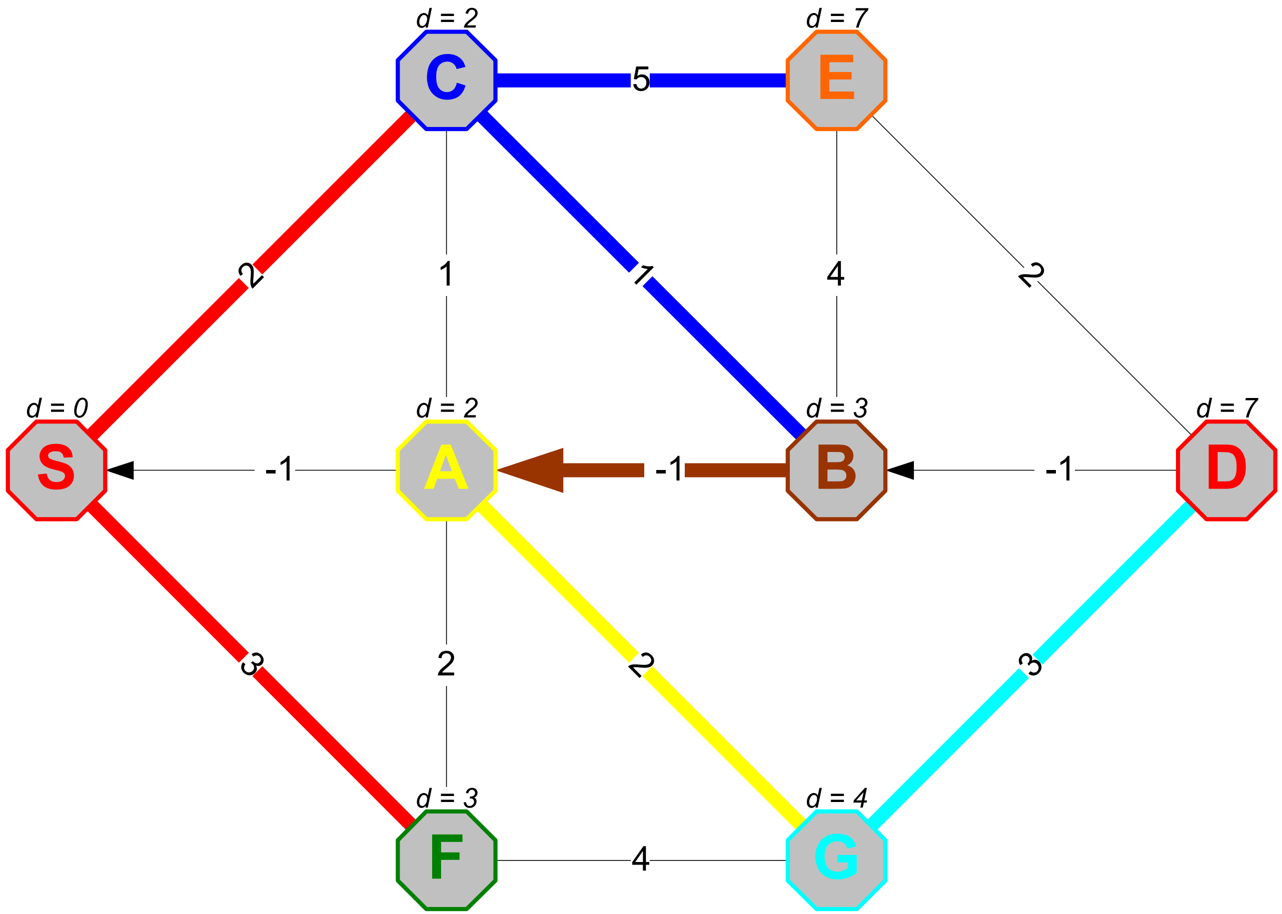

topology based on [2]

# Edge Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

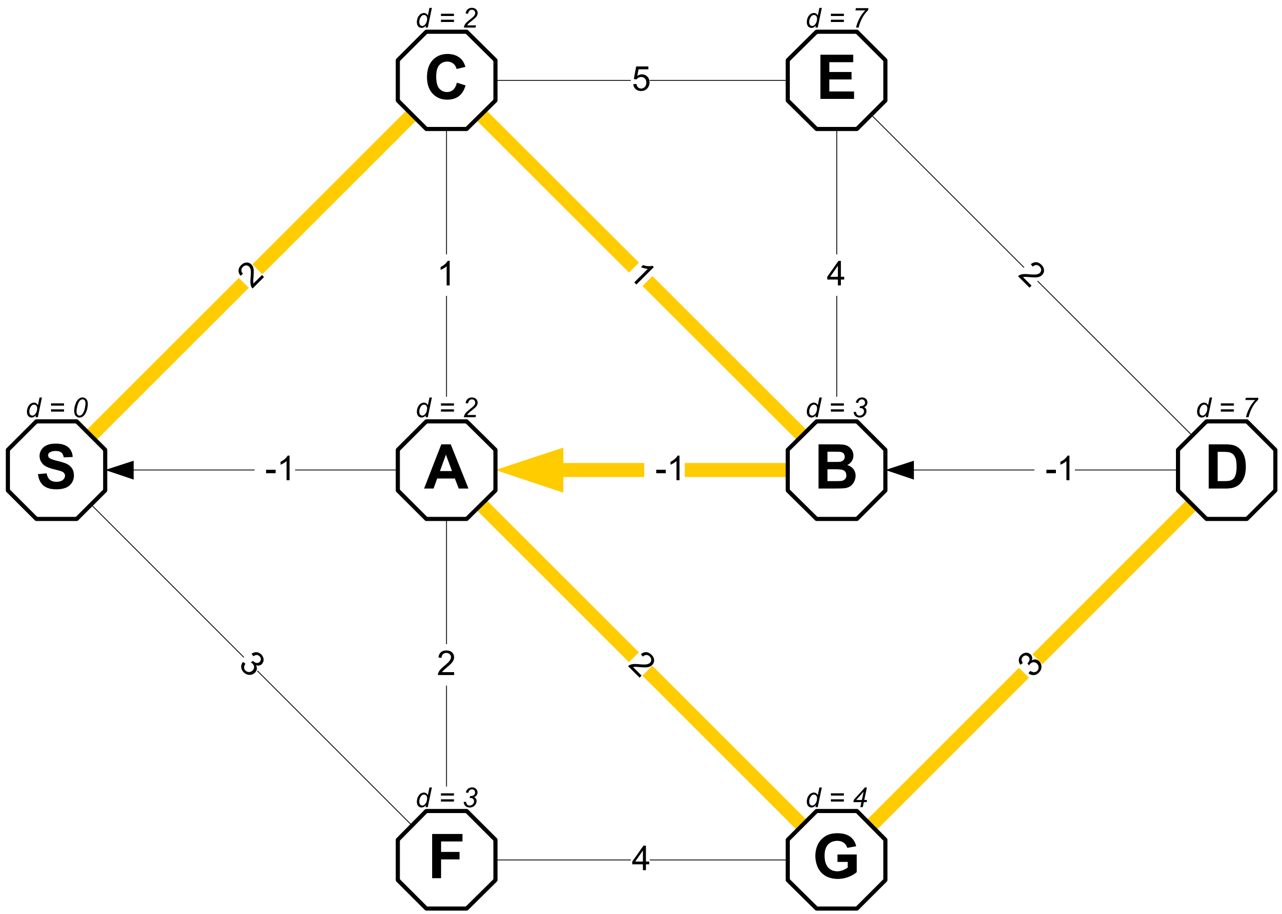
# Edge Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

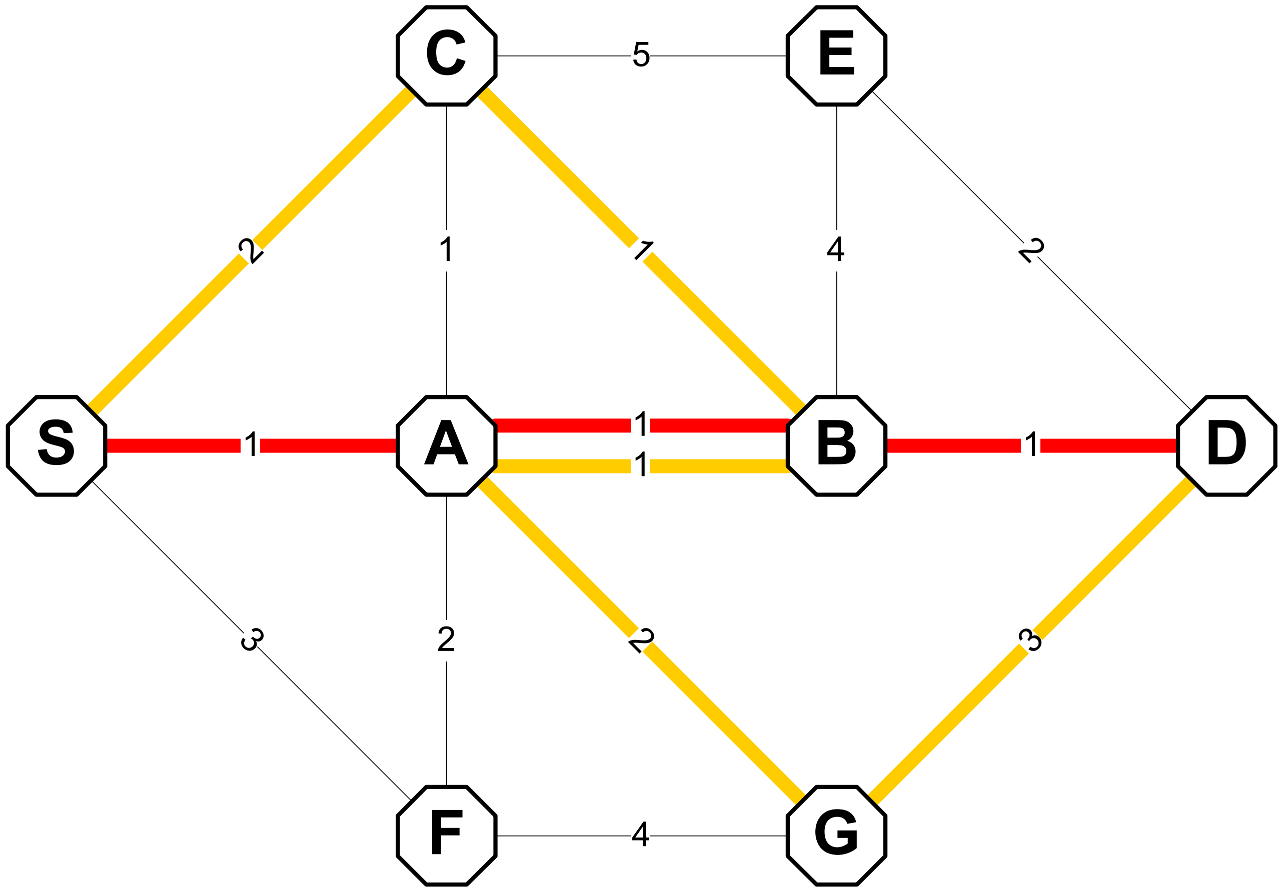
SIEMENS

# Edge Disjoint Shortest Pair Algorithm – An Example



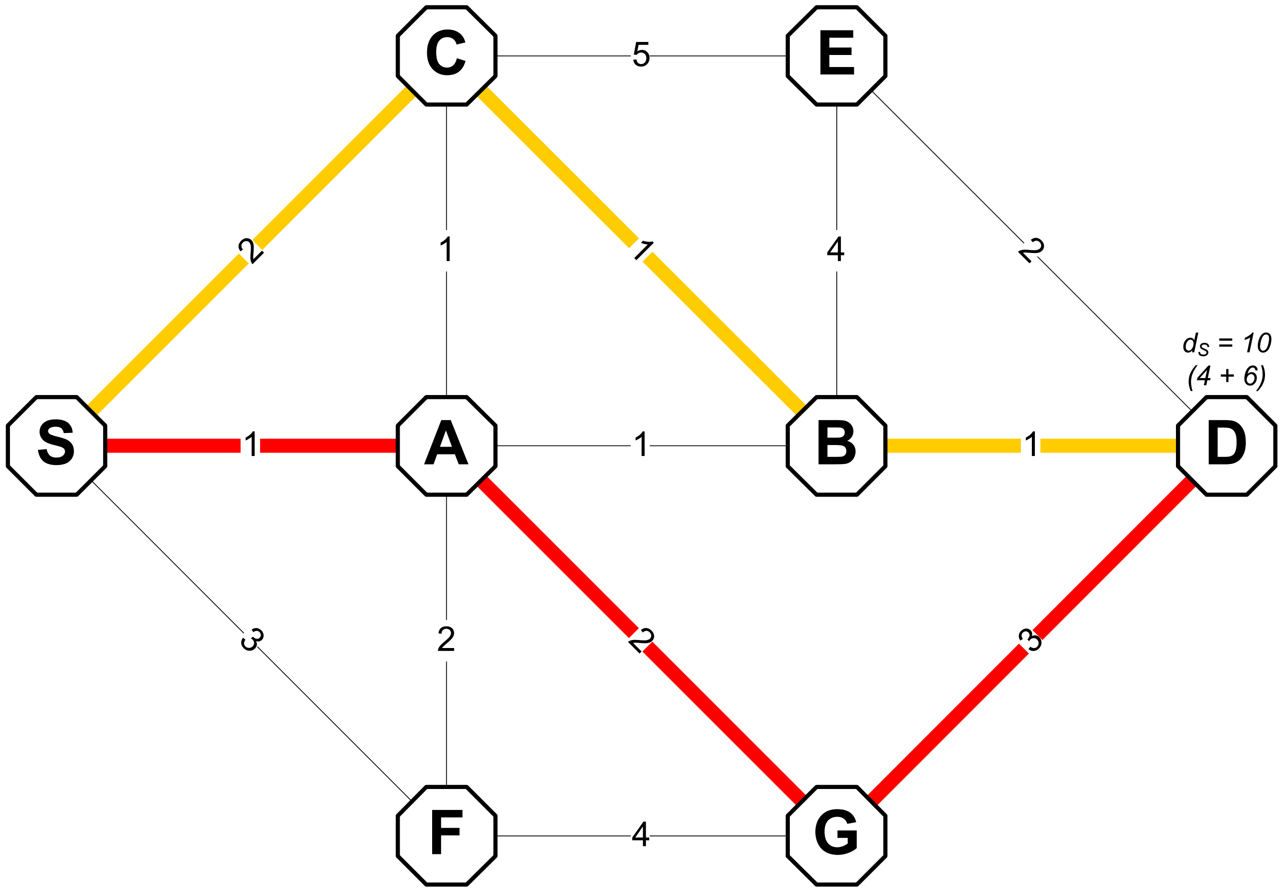

topology based on [2]

# Edge Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

# Edge Disjoint Shortest Pair Algorithm – An Example



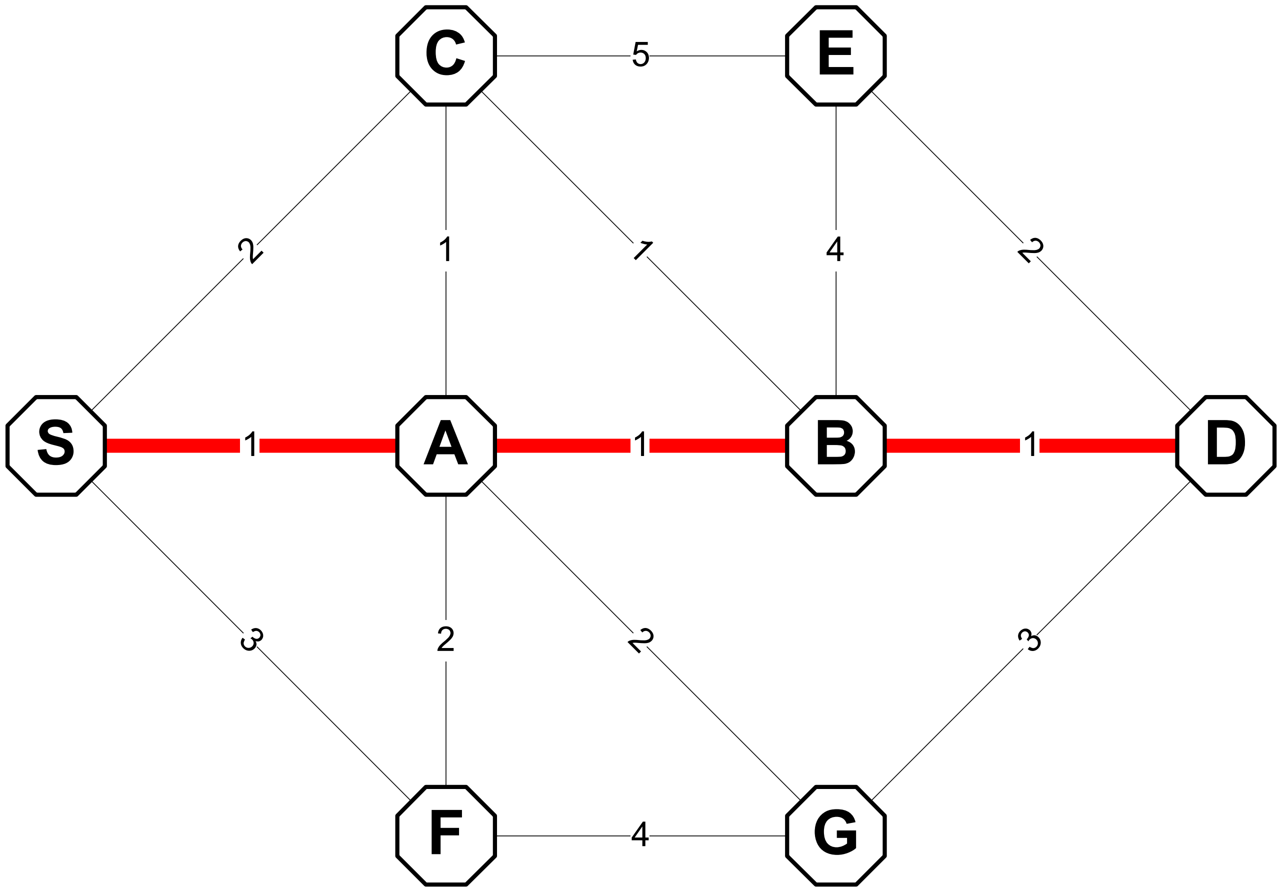

topology based on [2]

## Node-Disjoint Shortest Pair Algorithm

1. Find a first shortest path.
2. Replace each edge of the shortest path with a unidirectional edge directed towards the source and with its metric made negative.
3. Split the intermediate nodes N of the shortest path into two nodes N‘ and N“ with the following edges:
   - connect N‘ with the (outgoing) unidirectional edge towards the source
   - connect N“ with the (incoming) unidirectional edge directed from the destination towards N
   - connect N‘ and N“ with a unidirectional edge with metric = 0 directed from N“ towards N‘
4. Split the edges between the intermediate nodes N of the shortest path and their neighbors into two unidirectional edges with corresponding metric:
   - connect N‘ with the (incoming) unidirectional edge from the neighbor towards N
   - connect N“ with the (outgoing) unidirectional edge towards the neighbor
5. Find a second shortest path on this modified network topology.
6. Transform edges and split nodes back to original network topology.
7. Delete edges common to both found shortest paths and regroup remaining edges to shortest pair of node-disjoint paths.

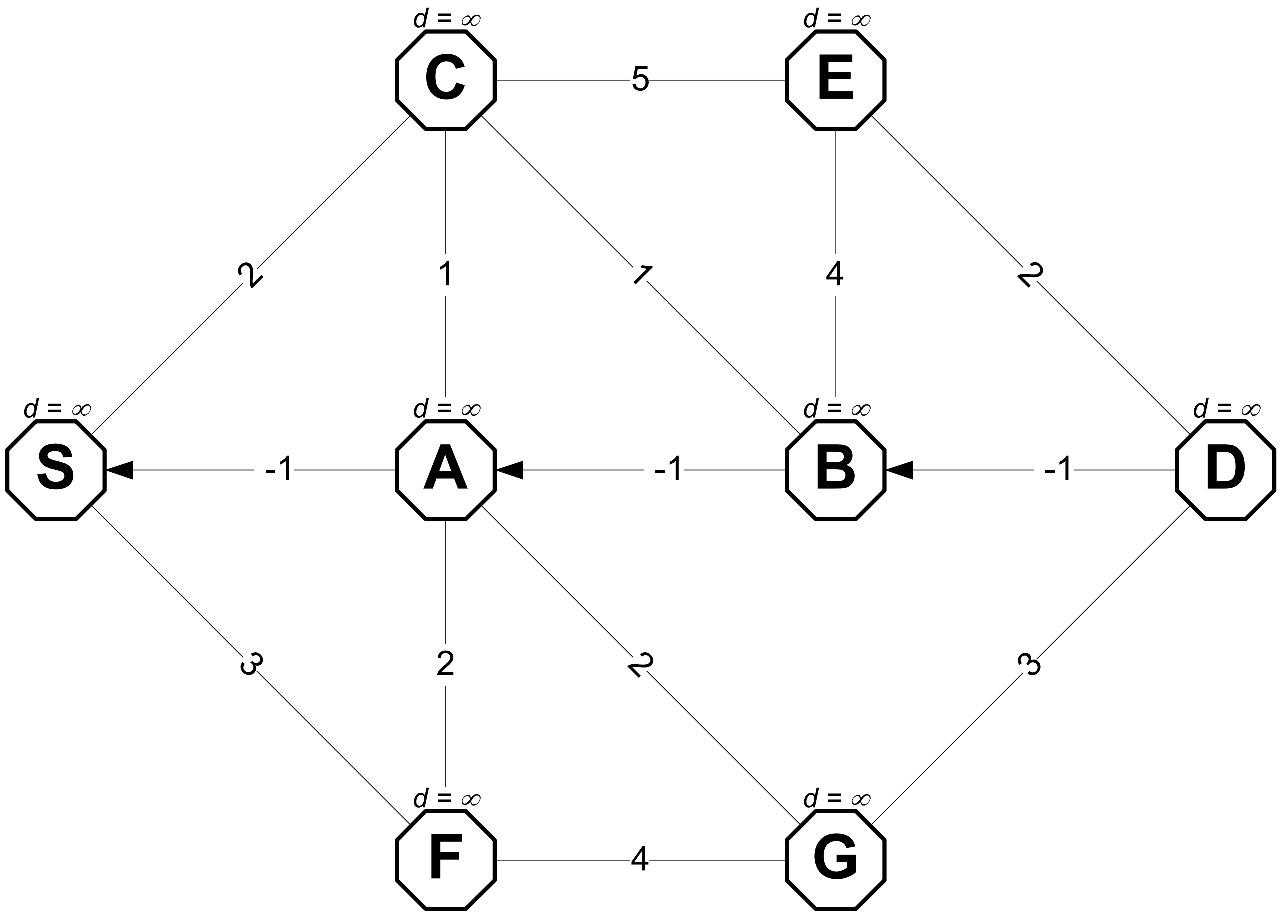
description based on [2]

# Node Disjoint Shortest Pair Algorithm – An Example



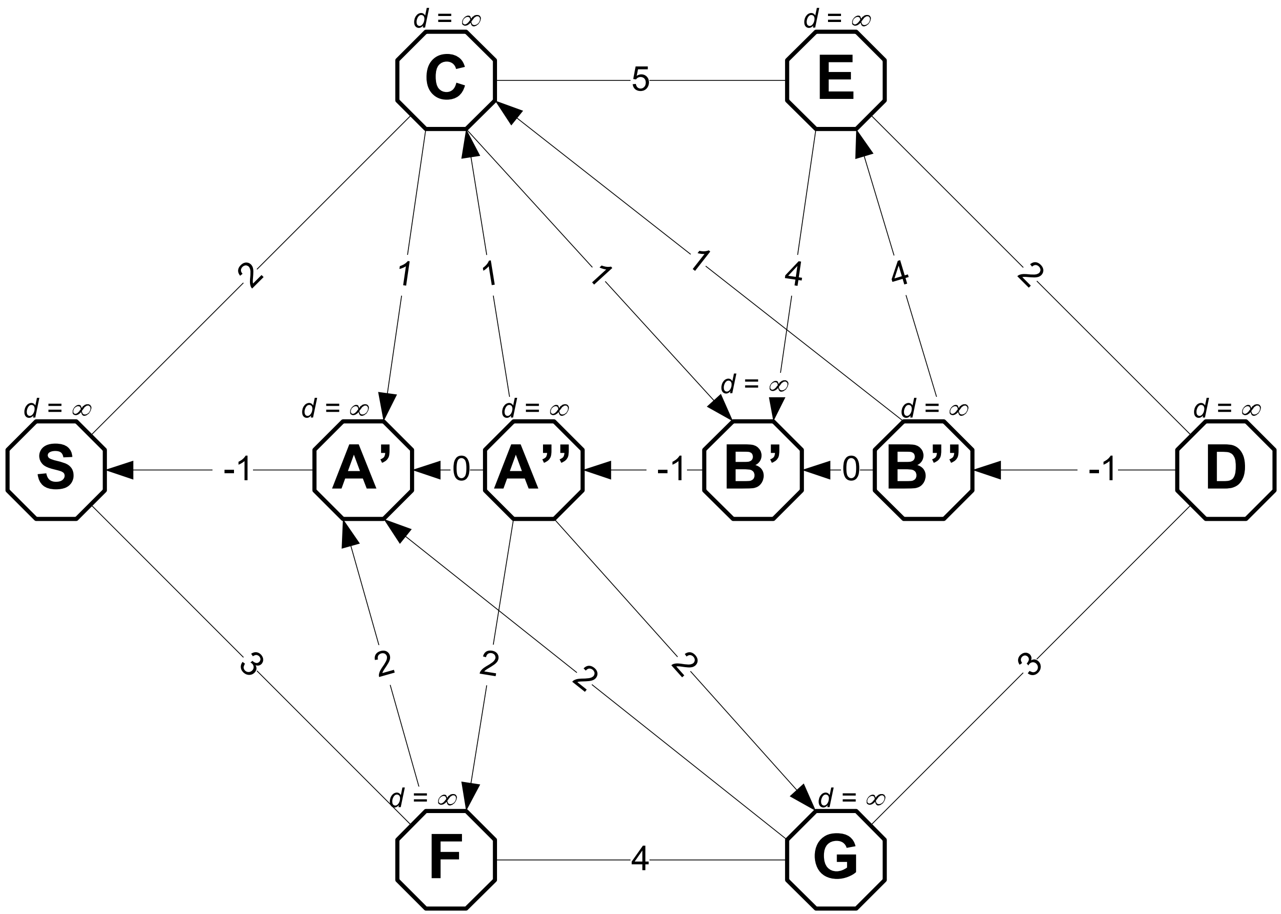

topology based on [2]

# Node Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

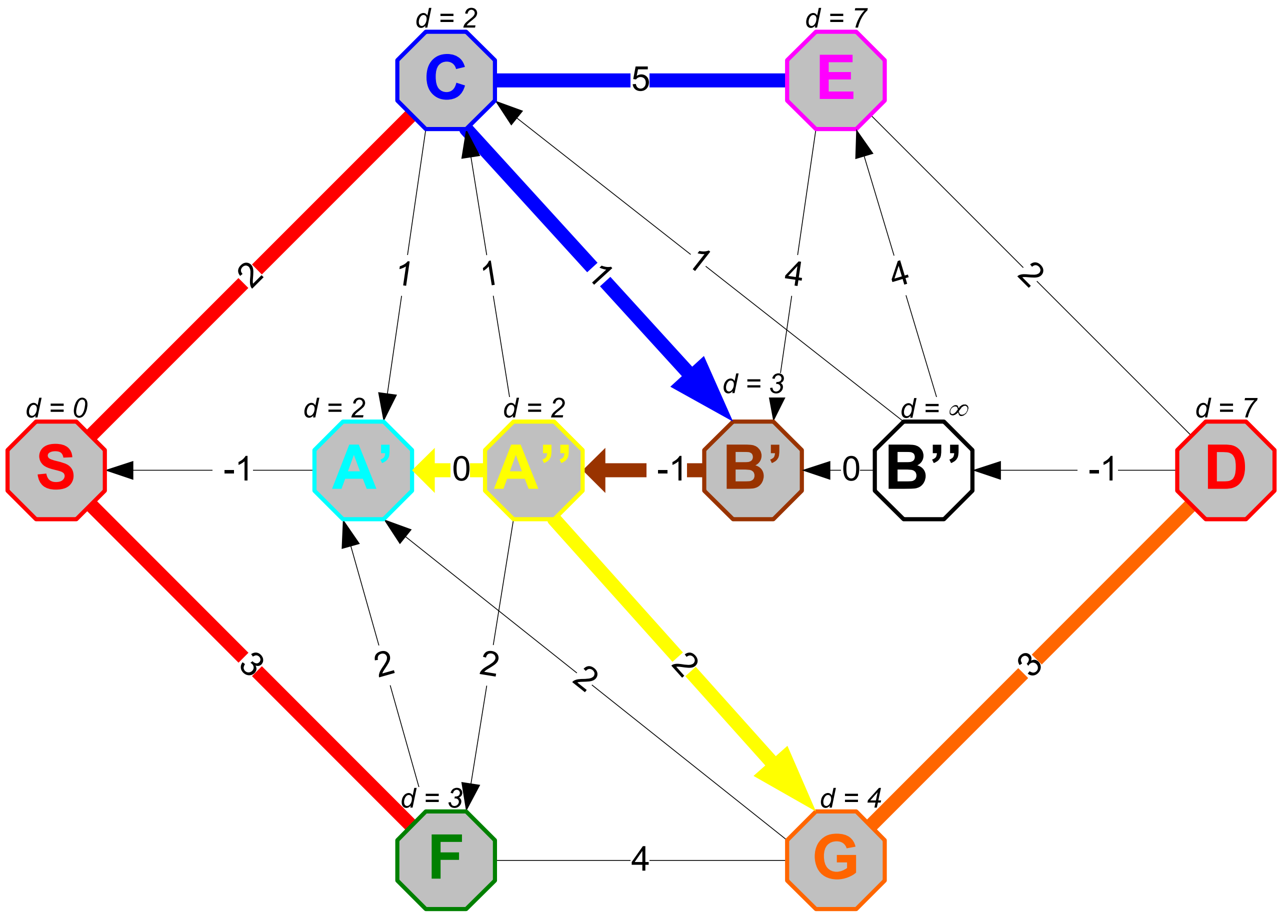
# Node Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

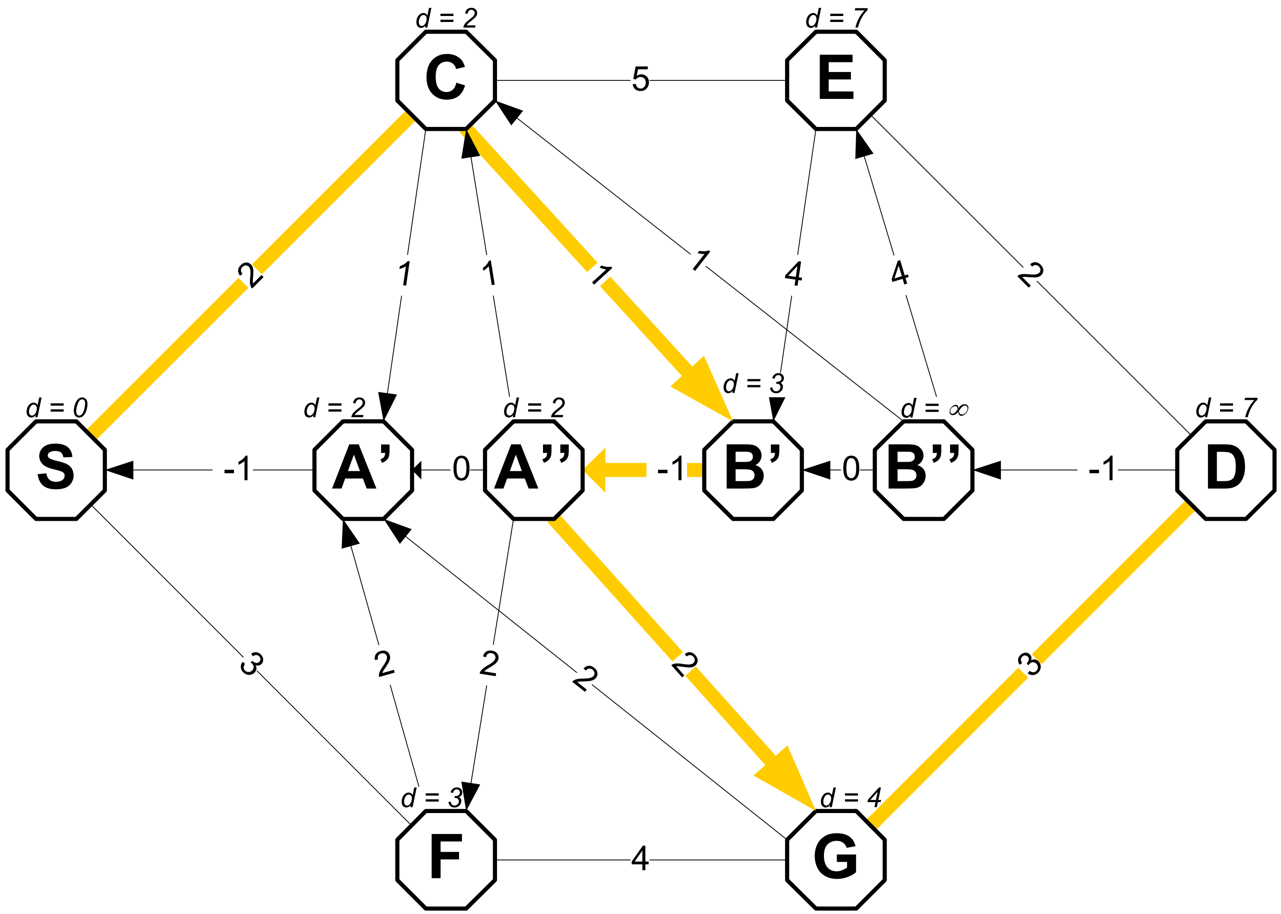
SIEMENS

# Node Disjoint Shortest Pair Algorithm – An Example



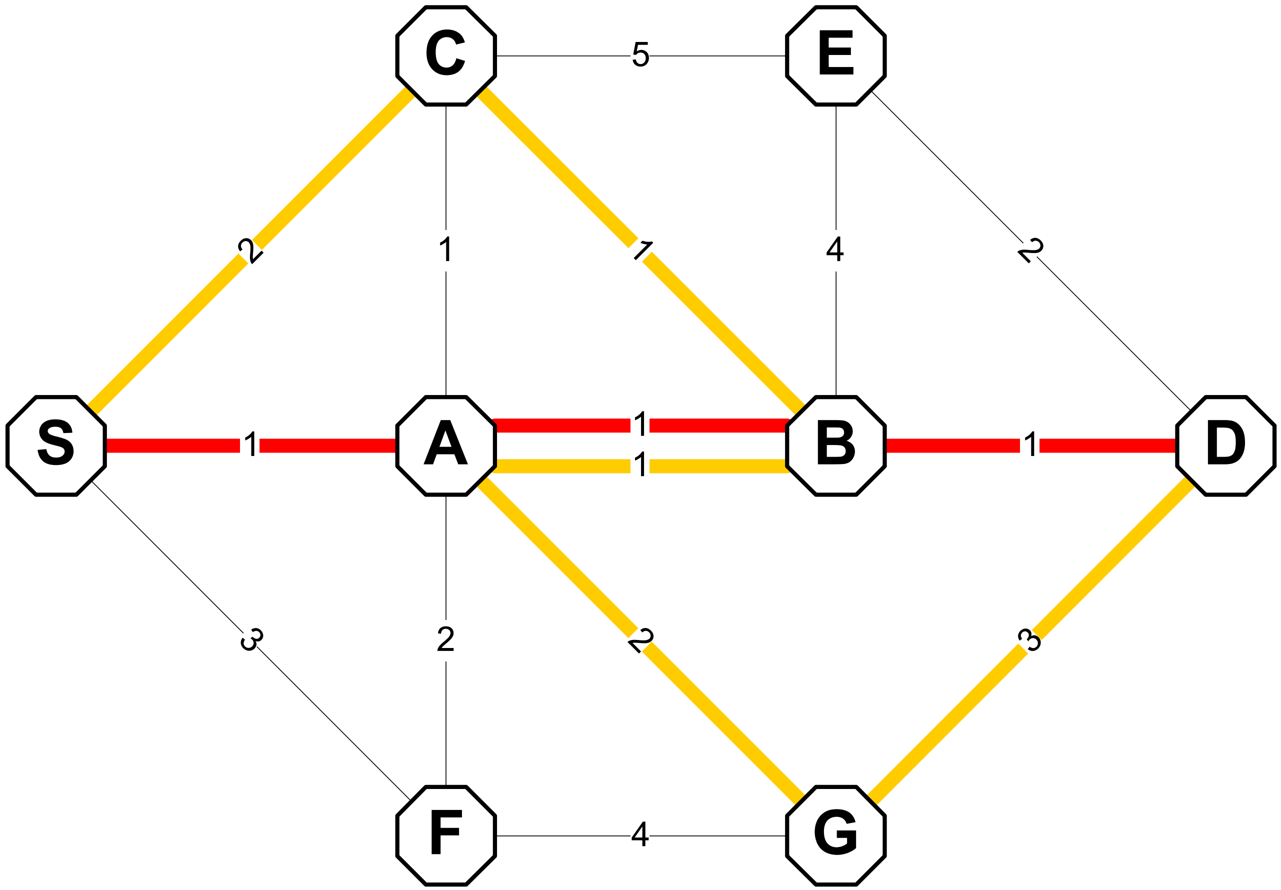

topology based on [2]

# Node Disjoint Shortest Pair Algorithm – An Example



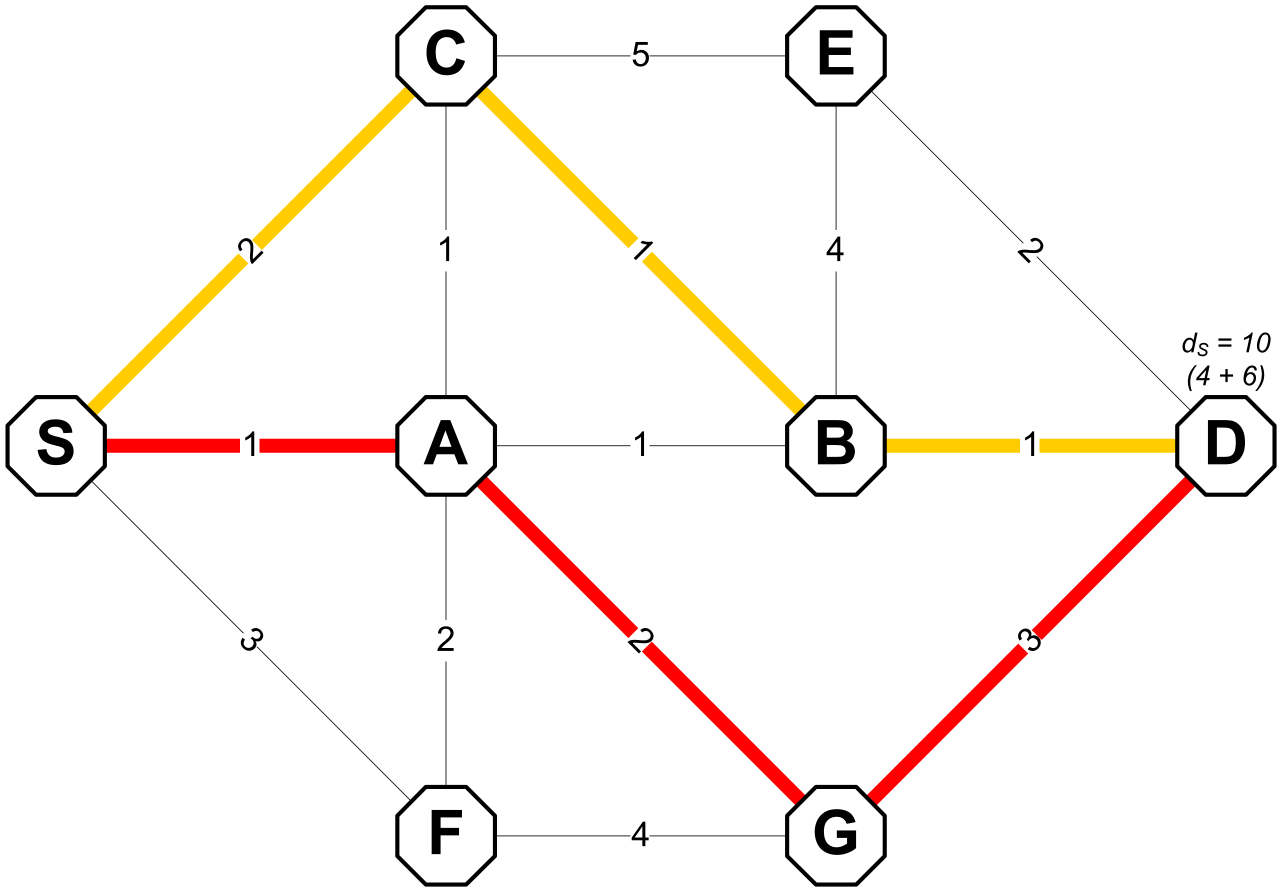

topology based on [2]

# Node Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

# Node Disjoint Shortest Pair Algorithm – An Example




topology based on [2]

SIEMENS

## Conclusion

- multiple paths can be easily computed with already existing link-state information.
- edge-disjoint as well as node-disjoint multiple paths can be computed.
- same computational complexity as traditional shortest path link state routing.
  - traditional shortest path link state routing:
    - decentralized O(N), centralized $O(N^2)$
  - shortest pair algorithms $O(2N+L_{SP}) \rightarrow O(N)$
    - two iterative runs of a shortest path algorithm are needed
    - needs to handle loop-free unidirectional negative edges
- finds the best (= globally optimal) set of m paths with best total metric if it exists
- well-suited for stream-oriented traffic

SIEMENS

# References

[1] E. W. Dijkstra: „A Note on Two Problems in Connexion with Networks“, Numerische Mathematik 1, pages 269-271, 1959

[2] Ramesh Bhandari: „Survivable Networks – Algorithms for Diverse Routing“, Kluwer Academic Publishers, 1999

[3] Hannes Gredler and Walter Goralski: „The Complete IS-IS Routing Protocol“, Springer, 2005

# Thank you
# for your attention!