Powerful and Efficient Bulk Shortest-Path Queries:

Cypher language extension & Giraph implementation

Peter Rutgers, Claudio Martella, **Spyros Voulgaris**, Peter Boncz

VU University Amsterdam

Goal and Contributions

- Context: Shortest-path queries in Giraph
- Desired functionality
  - Edge weights (monotonic cost function!)
  - Multiple sources and destinations (“bulk” queries)
  - Top-N shortest paths for each pair
  - Filters on path edges and vertices
  - Provide both paths and their costs
- Our contributions are twofold:
  - Cypher language extension
  - Efficient top-N shortest path algorithm design & implementation on Giraph

vrije Universiteit amsterdam

Outline

Cypher Extension

Algorithms and Implementation

Evaluation

Conclusions

Shortest Paths in Cypher [1/2]

```
MATCH path=shortestPath( (a)-[*]->(b) )
WHERE <condition>
RETURN path, length(path);
```

- No weighted paths!
- No top-N shortest paths!
- Conditions in WHERE applied *after* finding path
  - Could result in empty answer!

Shortest Paths in Cypher [1/2]

```
MATCH path=shortestPath( (a)-[*]->(b) )
WHERE none(x in nodes(path) WHERE x.danger)
RETURN path, length(path);
```

- No weighted paths!
- No top-N shortest paths!
- Conditions in WHERE applied *after* finding path
  - Could result in empty answer!

Shortest Paths in Cypher [2/2]

```
MATCH path=(a)-[r*]->(b)
WHERE none(x in nodes(path) WHERE x.danger)
RETURN path,
       reduce(sum=0, x IN r | sum=sum+x.dist*x.speed)
       AS len
ORDER BY len DESC
LIMIT 5
```

- Matches *all* paths! Expensive!
- Orders all paths that remain after the WHERE condition
- Complex query for humans
- Complex query for the query planner
  - Hard to detect and optimize

vrije Universiteit amsterdam

Proposed language extension

```
MATCH path=(src)-[e* | sel(e)]->(dst)
CHEAPEST n SUM cost(e) AS d
```

- Selector applied *before* WHERE condition (optional)
- Multiple paths (top-N) for each pair
- Custom cost function
- AS keyword to bind cost to variable
- Supports bulk queries (multiple sources / multiple destinations)

Example

- Suppose you are building a navigation system
  - Some nodes are of type Src, some of type Dst
  - Some nodes have the property danger
  - The cost of each segment is the distance times the speed limit

- You can get the top-3 cheapest routes by the following simple query:

```
MATCH path=(a:Src)-[e* | not(endNode(e).danger)]->(b.Dst)
CHEAPEST 3 SUM e.dist * e.speed AS len
RETURN a, b, path, len
```

vrije Universiteit amsterdam

Outline

Cypher Extension

Algorithms and Implementation

Evaluation

Conclusions

The Lighthouse Project

- Cypher-based declarative language, query planning and execution, for Apache Giraph.

- **Parser**
  - Turns Cypher query into query graph

- **Planner**
  - Builds query plan (tree of operators)

- **Execution engine**
  - Runs query plan on Giraph`

Top-N Shortest Path

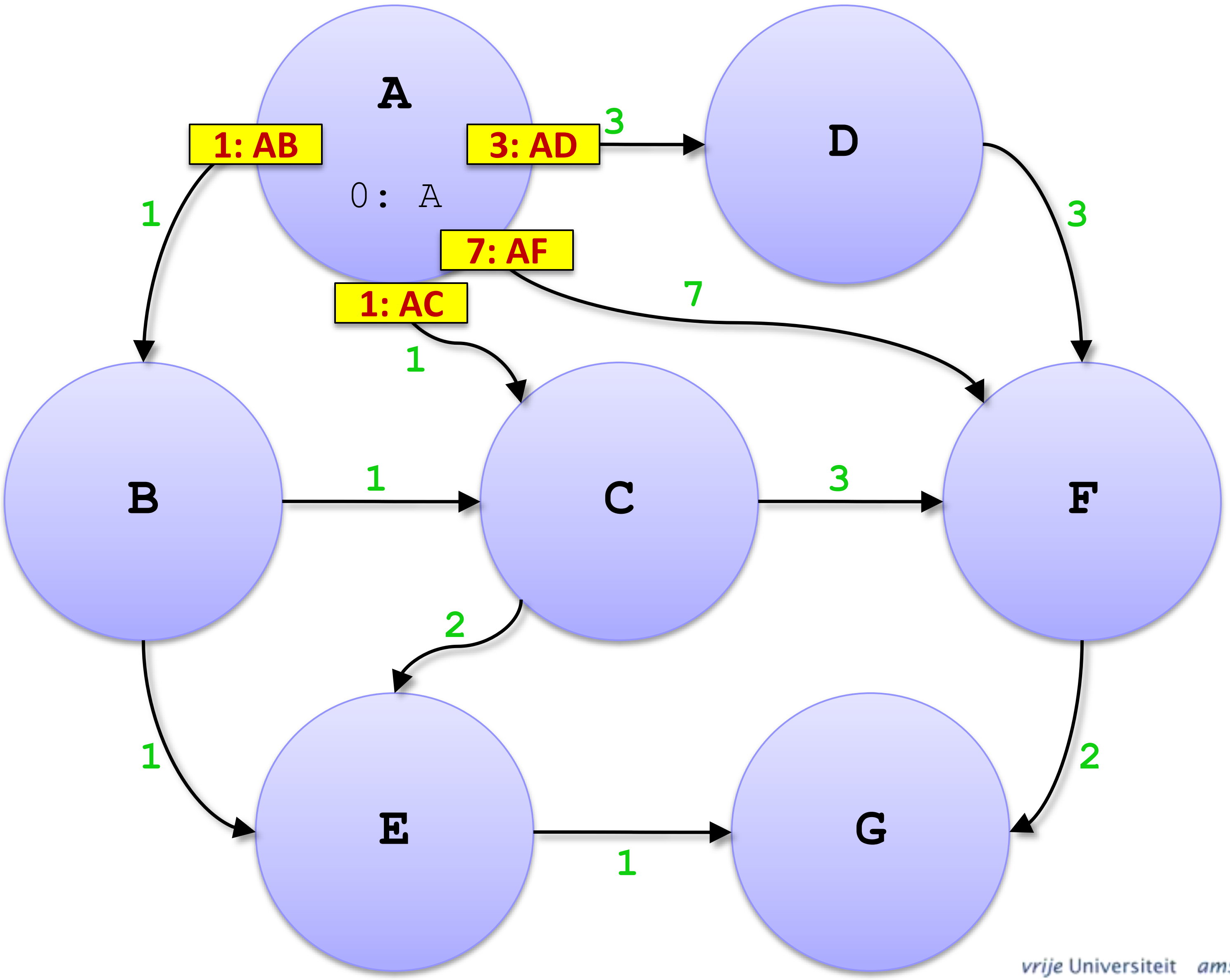
- We need to compute both the **cost** and the **path** itself

- Basic algorithm
  - Each node maintains the top-N paths (and costs) found so far
  - In each step, each node propagates all its updates along all its outgoing edges
  - When a node has received no updates in a step, it votes to halt
  - The algorithm terminates when they all vote to halt

vrije Universiteit amsterdam

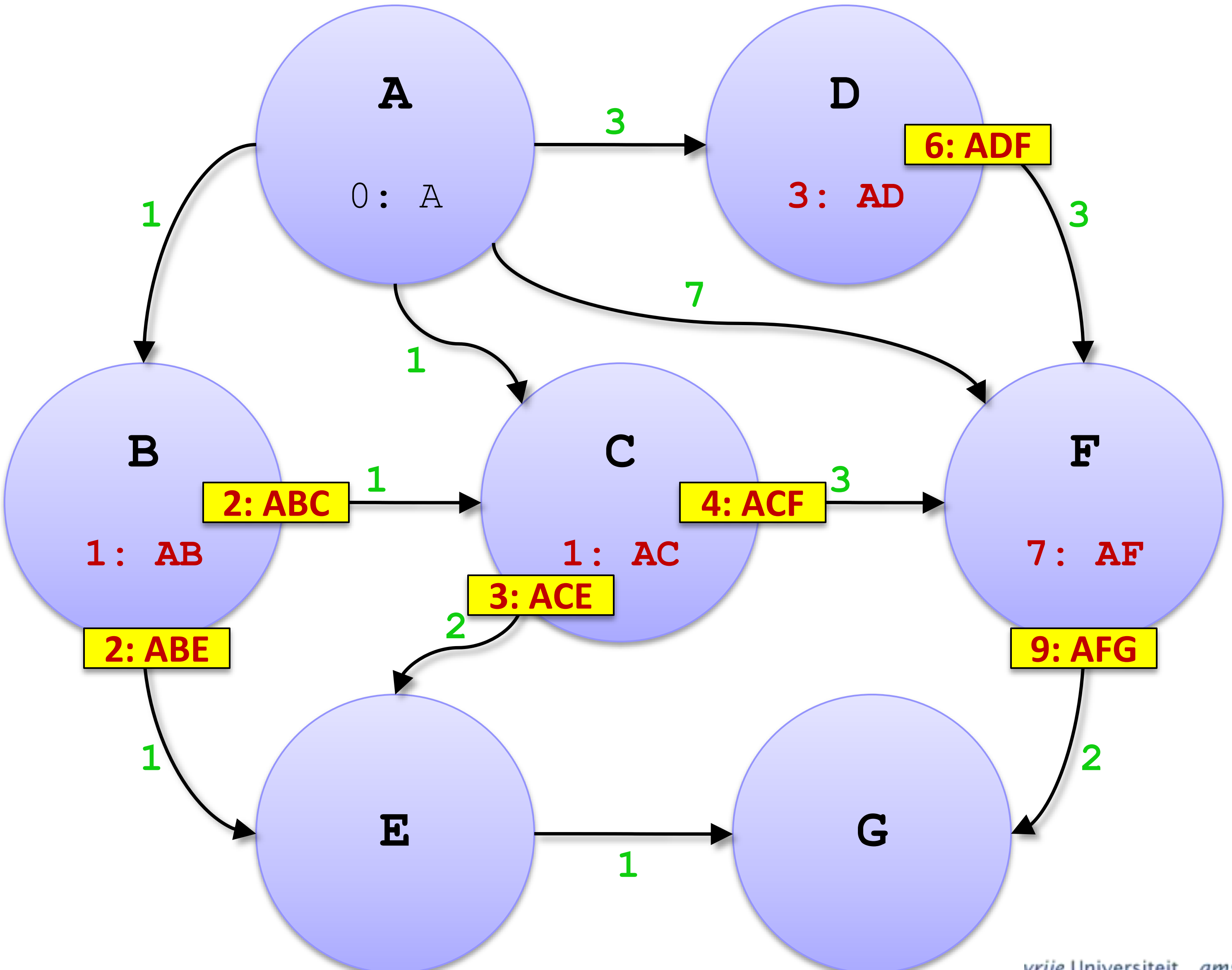
Top-N Shortest Path

N=5



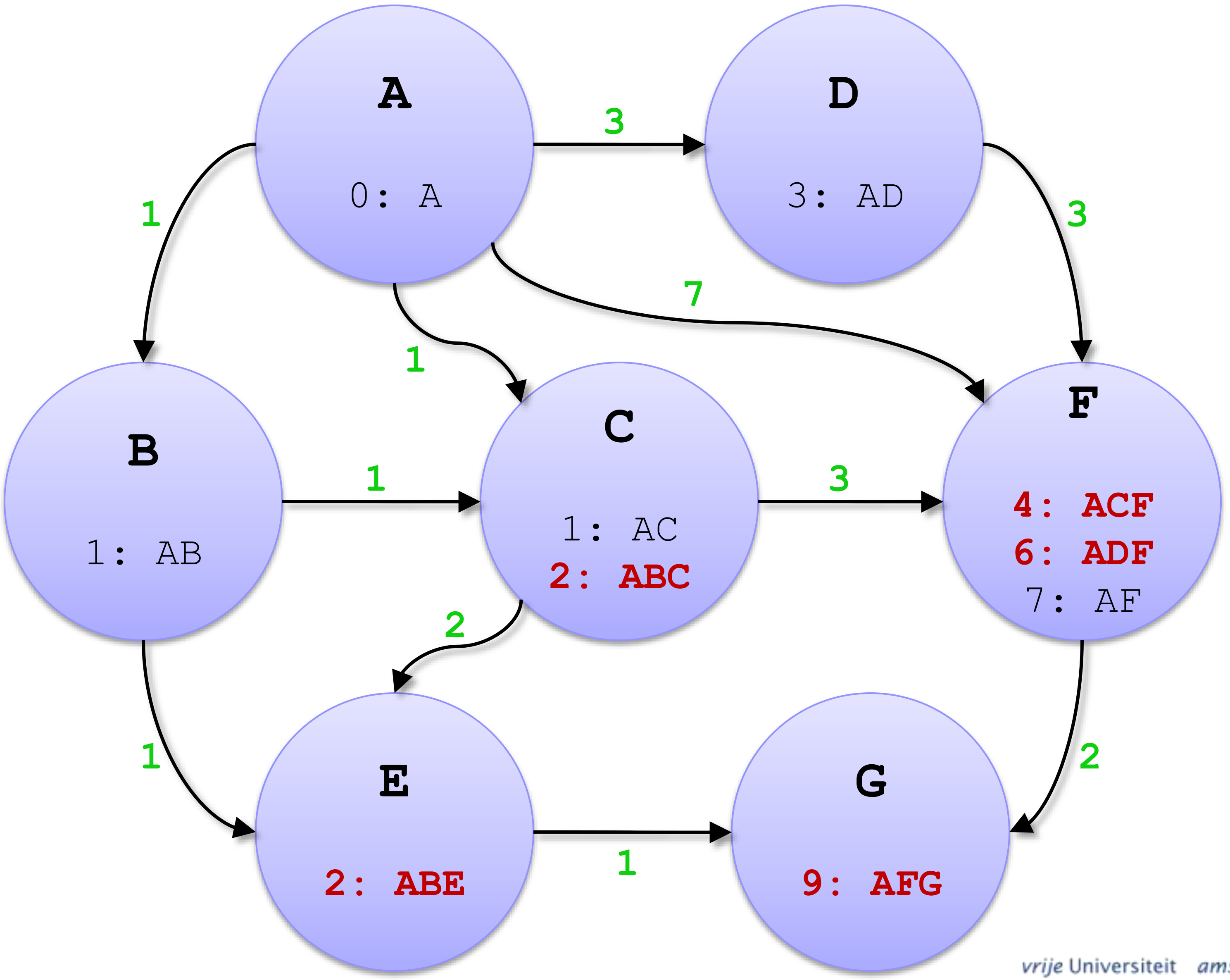
Top-N Shortest Path

N=5



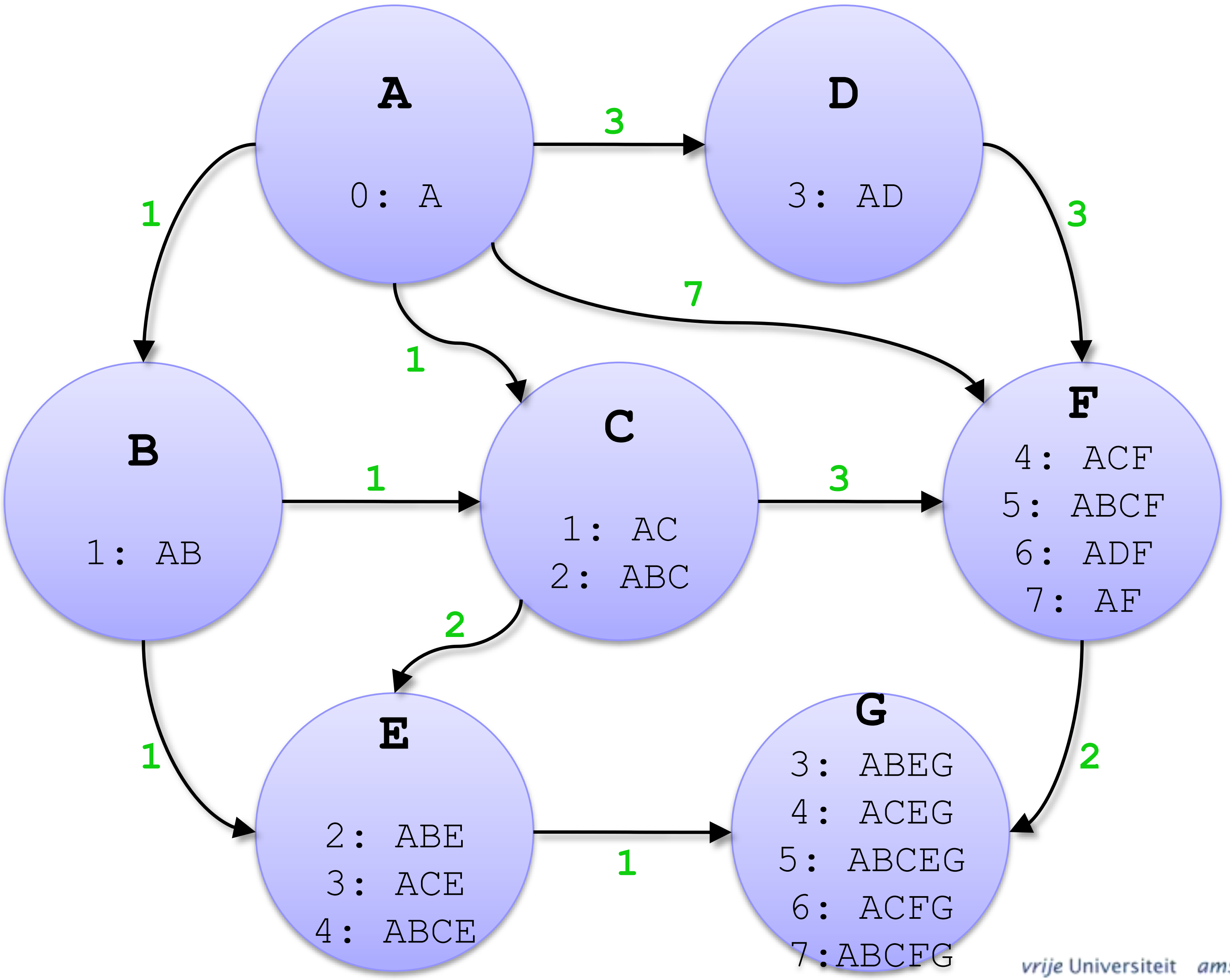
Top-N Shortest Path

N=5



Top-N Shortest Path

N=5



vrije Universiteit amsterdam

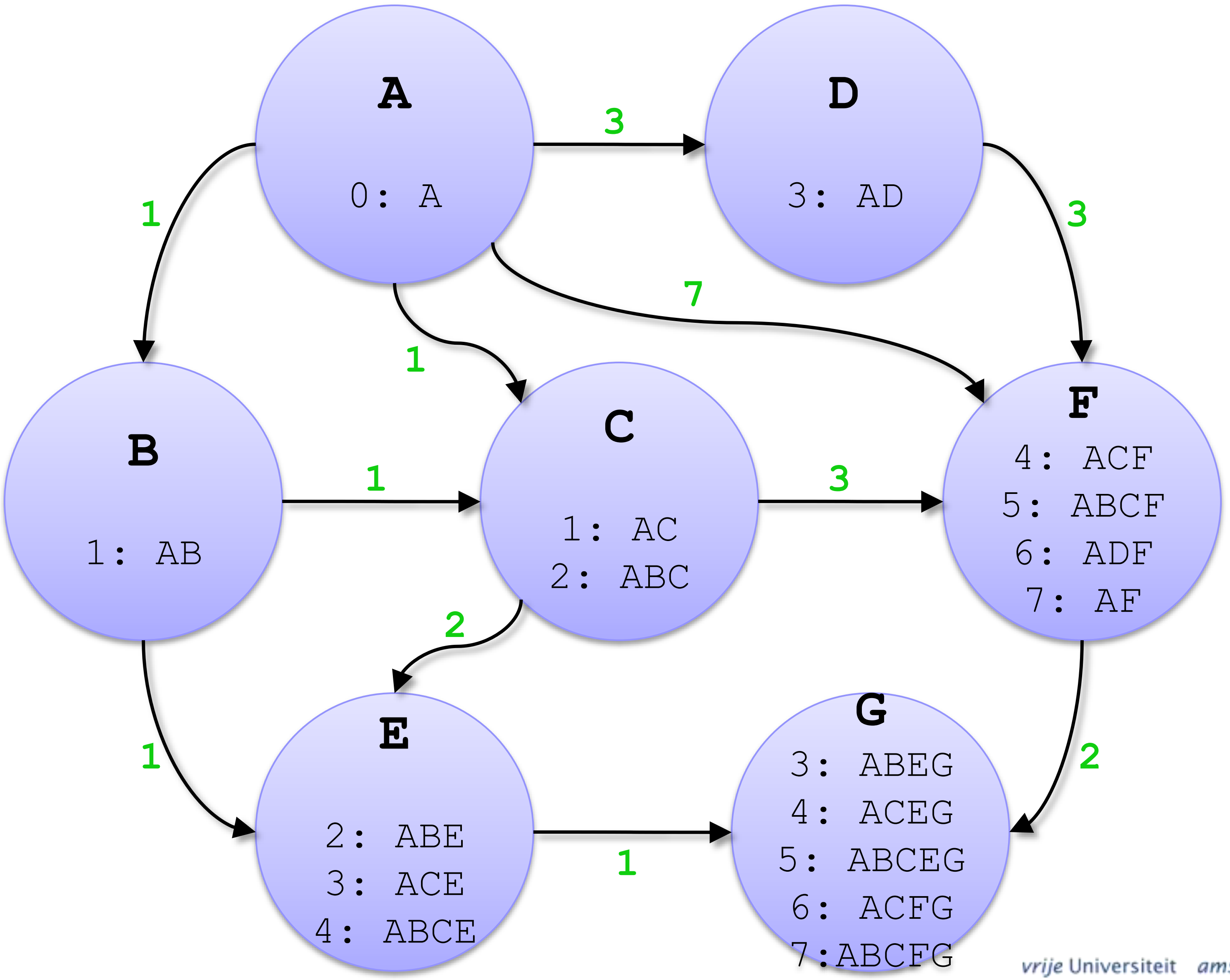
Can we do better?!

- One problem:
  - Memory footprint is too high
  - Paths passed around are too long
- The solution:
  - No need to pass and store the entire path
  - Store only **predecessor node ID** and **cost to date** per path
  - Less communication, lower runtime!
- The price to pay?
  - An extra phase for path reconstruction

vrije Universiteit amsterdam

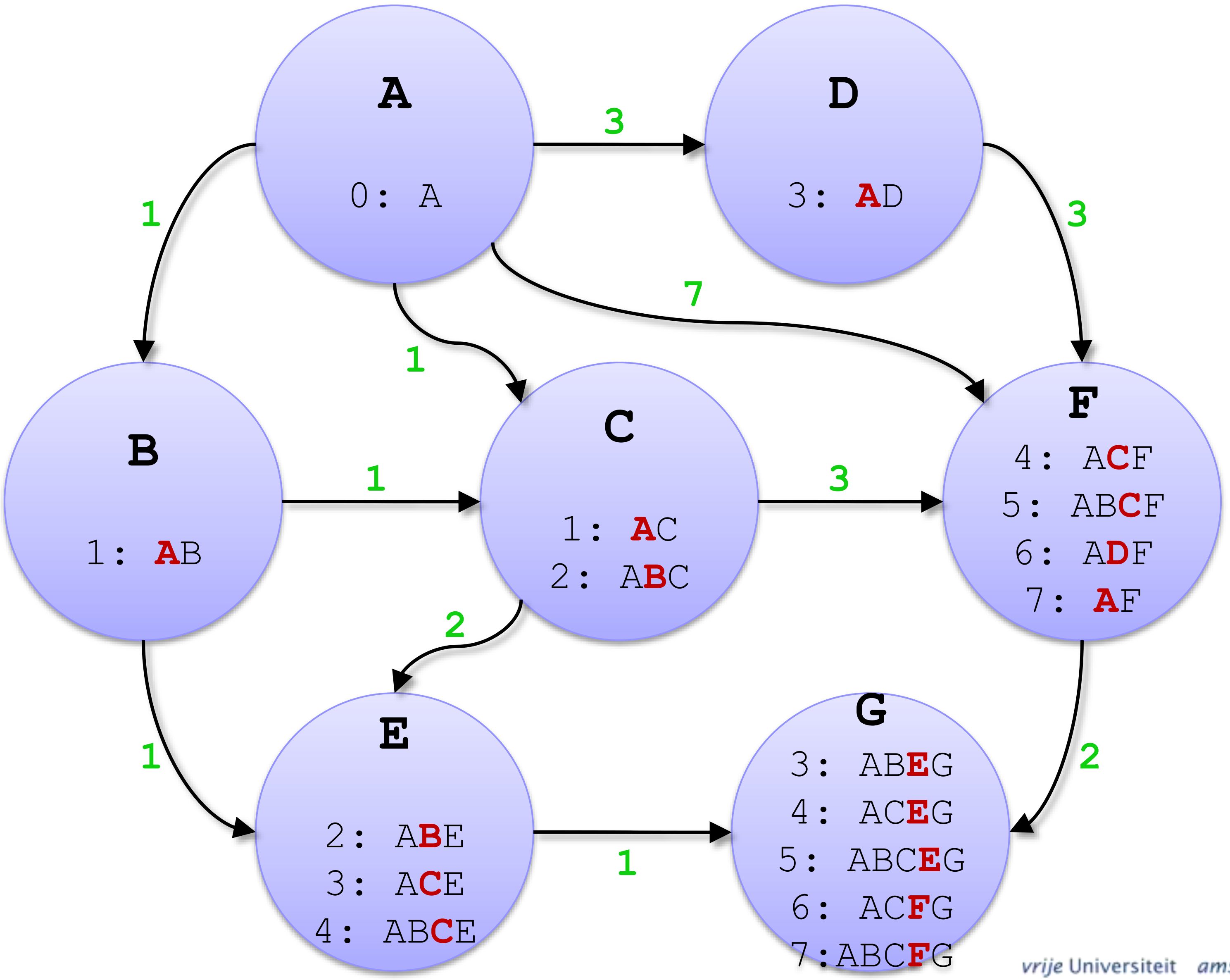
Top-N Shortest Path

N=5



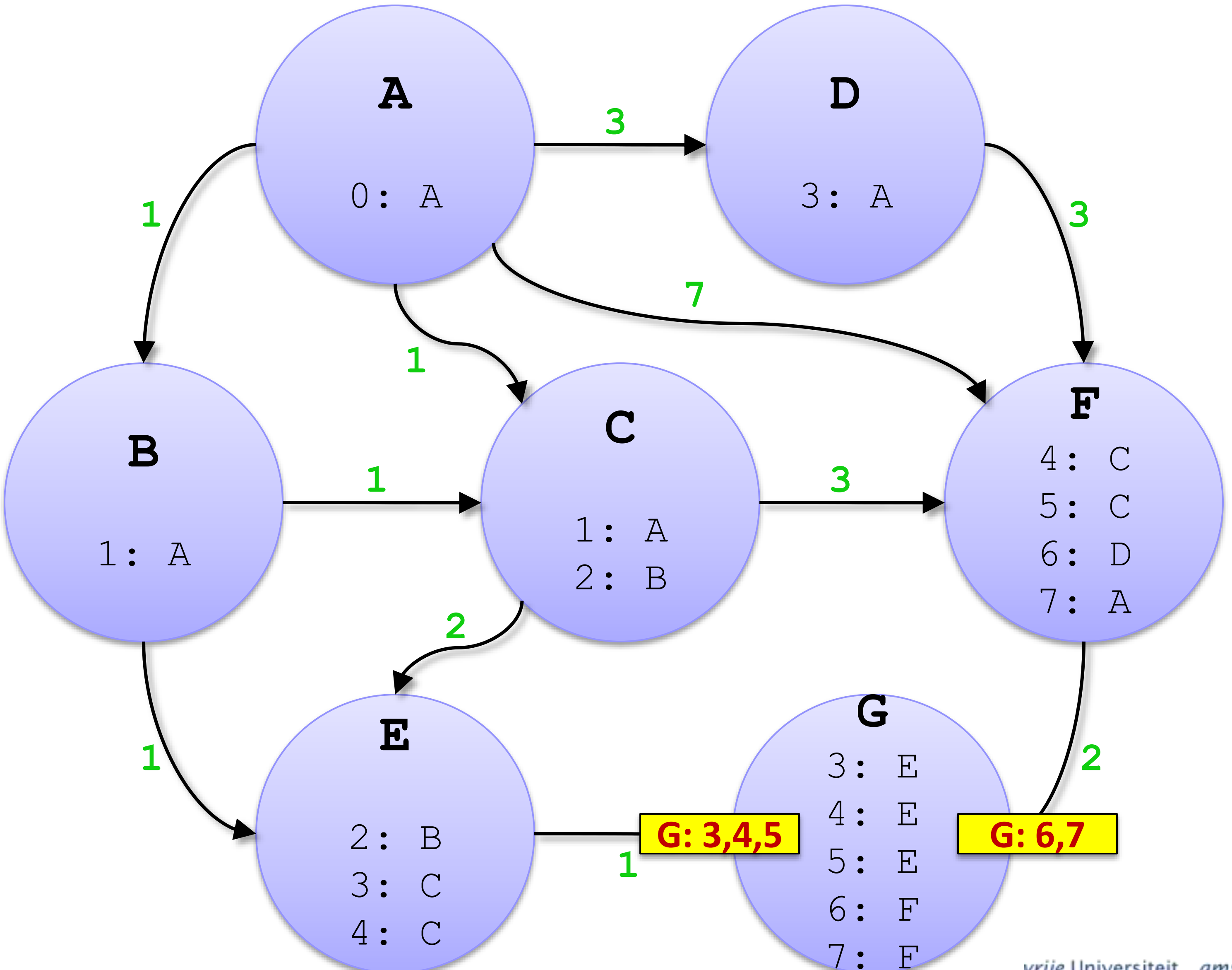
Top-N Shortest Path

N=5

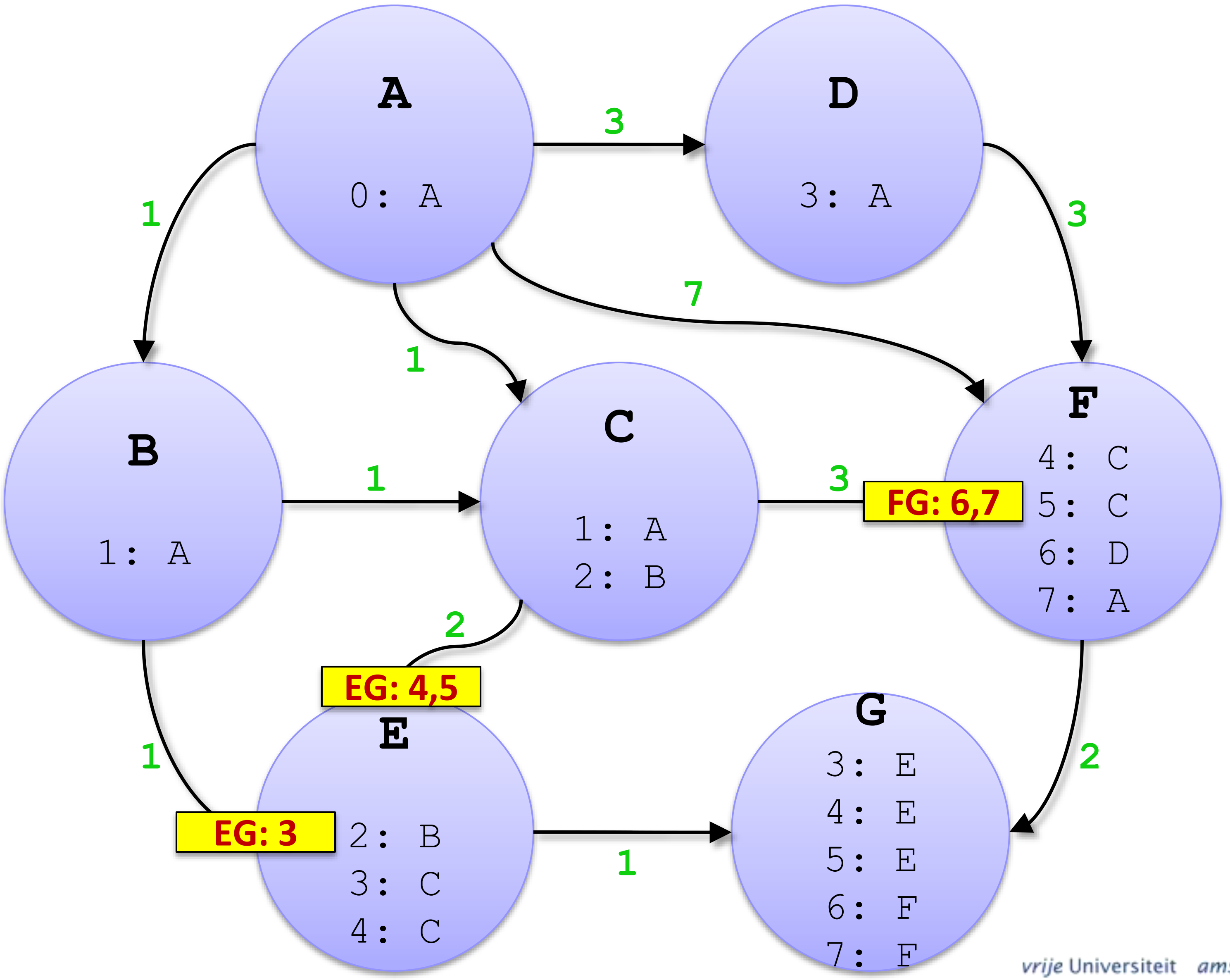


vrije Universiteit amsterdam

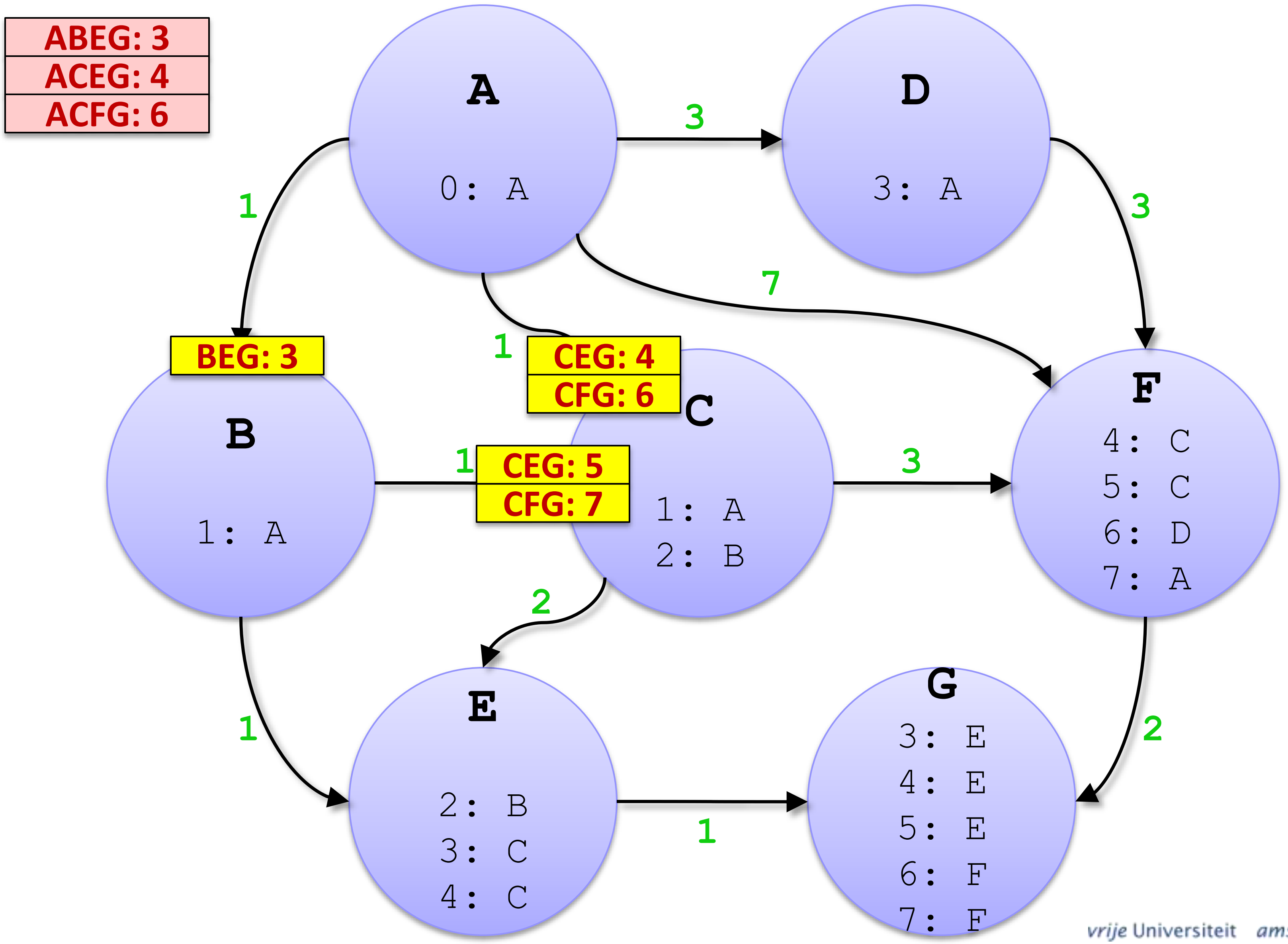
Top-N Shortest Path Reconstruction



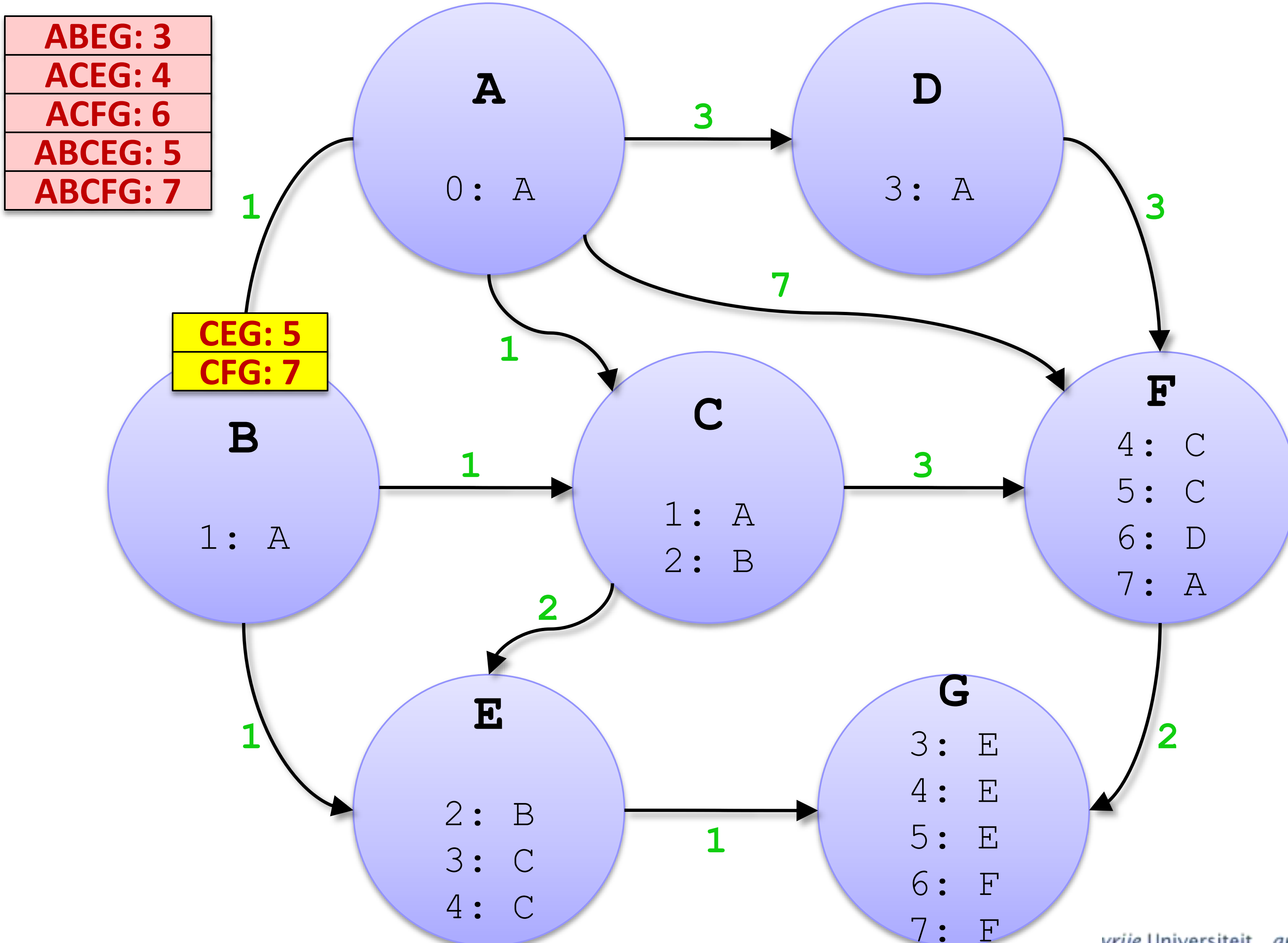
Top-N Shortest Path Reconstruction



Top-N Shortest Path Reconstruction



Top-N Shortest Path Reconstruction



Can we do even better???

- The problem:
  - In the first few supersteps, some **expensive, yet short, paths** are propagated aggressively.
  - Unnecessary resource consumption

- Solution:
  - **Postpone exploration!**
  - Reduce the exponential growth of exploration in the first supersteps.
  - Delay propagating paths that “appear” to be not-too-cheap.

- How?
  - Place paths in buckets [0,Δ], [Δ,2Δ], … and suppress the propagation of paths of bucket *i* until superstep *i.*

vrije Universiteit amsterdam

Pruning via Landmarks

- To further confine unnecessary exploration, we prune based on upper cost bounds.

- We use landmarks:
  - Selected nodes X_i ,
  - For each src/dst pair AB, we compute $|AX_i|$ and $|X_iB|$.
  - $|AX_i| + |X_iB|$ forms an upper bound for $|AB|$.

Outline

Cypher Extension

Algorithms and Implementation

Evaluation

Conclusions

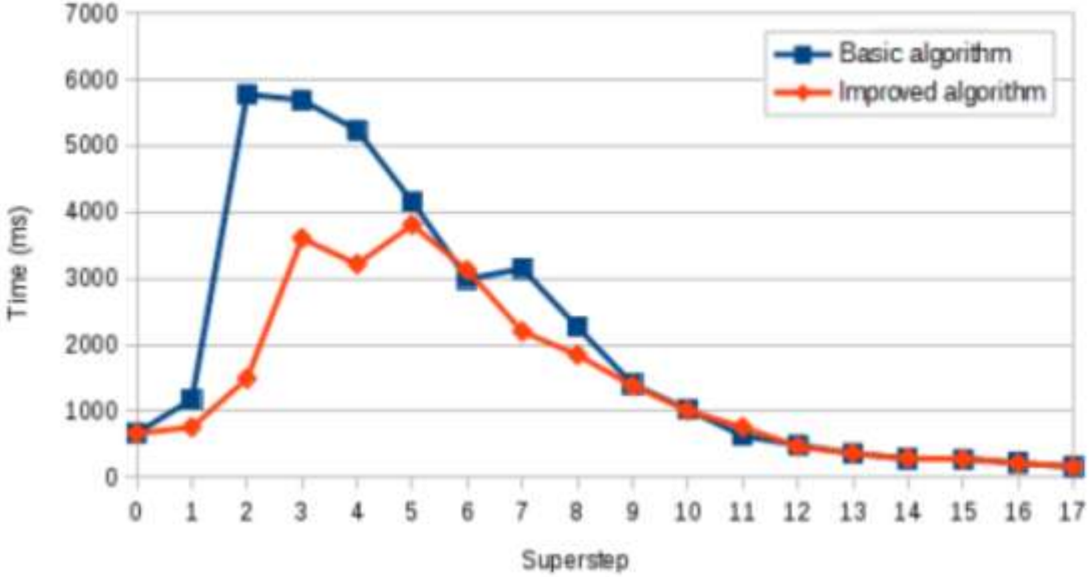
Overall scalability

- LDBC - SF10 trace
  - Scale factor 10, with 72,949 vertices and 4,641,430 edges

#workers	1	2	4	8	16	32
Runtime (sec)	>1000	492	222	126	89	72

vrije Universiteit amsterdam

Postponing Path Exploration (Delta stepping)



- Rnd1K trace: Erdos-Renyi, 1000 vertices, 50K edges
- One-to-all, top-5 shortest paths

- Total runtime drops from 35sec to 25sec
- Total #bytes sent drops by 49%

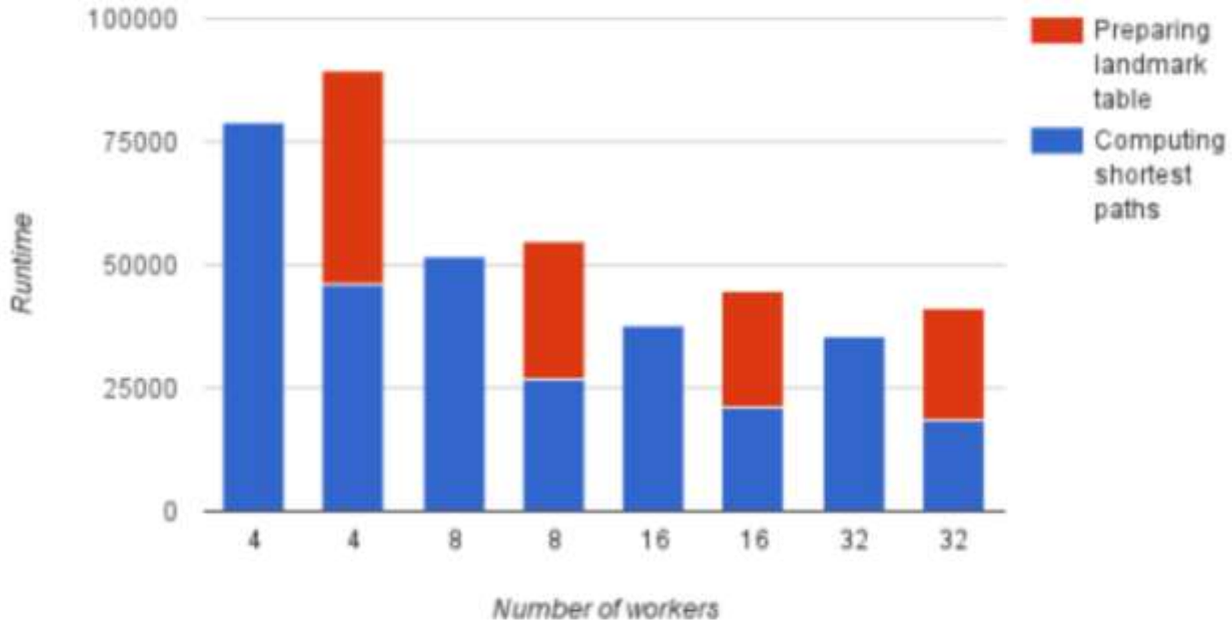
Effect of Multiphase Approach

- Rnd1K trace: 1K nodes, 50K edges

	bytes	messages	supersteps	time
Basic	182,204,626	402628	18	35.92 sec
Multiphase	83,926,097	402749	28 (18+10)	27.132 sec

vrije Universiteit amsterdam

Effect of Landmark Pruning



- LDBC - SF1 trace: 10,993 vertices, 451K edges
- 25 random sources, all nodes as destinations
- Top-5 shortest paths
- 2 landmarks (the highest degree nodes)

- Actual computation drops by ~40%
- Landmark estimation takes too long

vrije Universiteit amsterdam

Outline

Cypher Extension

Algorithms and Implementation

Evaluation

Conclusions

Conclusions

- We proposed new Cypher syntax that allows
  - Flexible edge weights
  - Flexible filter conditions over these
  - Top-N queries
- This syntax is concise, and guarantees that efficient (pruning) algorithms can be employed by the query planner
- We proposed efficient shortest path algorithms
  - Number of messages and data transferred are substantially reduced
  - Much improved memory footprint
  - However, they do not necessarily reduce runtime
  - Landmarks do not always improve runtime