Projekt 1

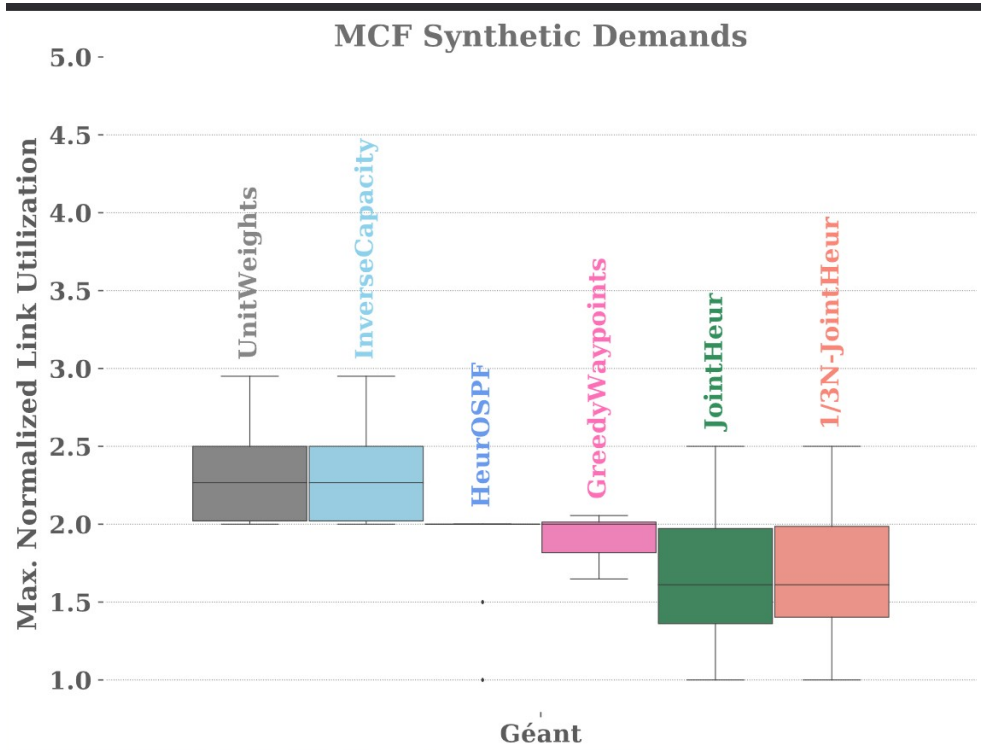
Marvin Weiler

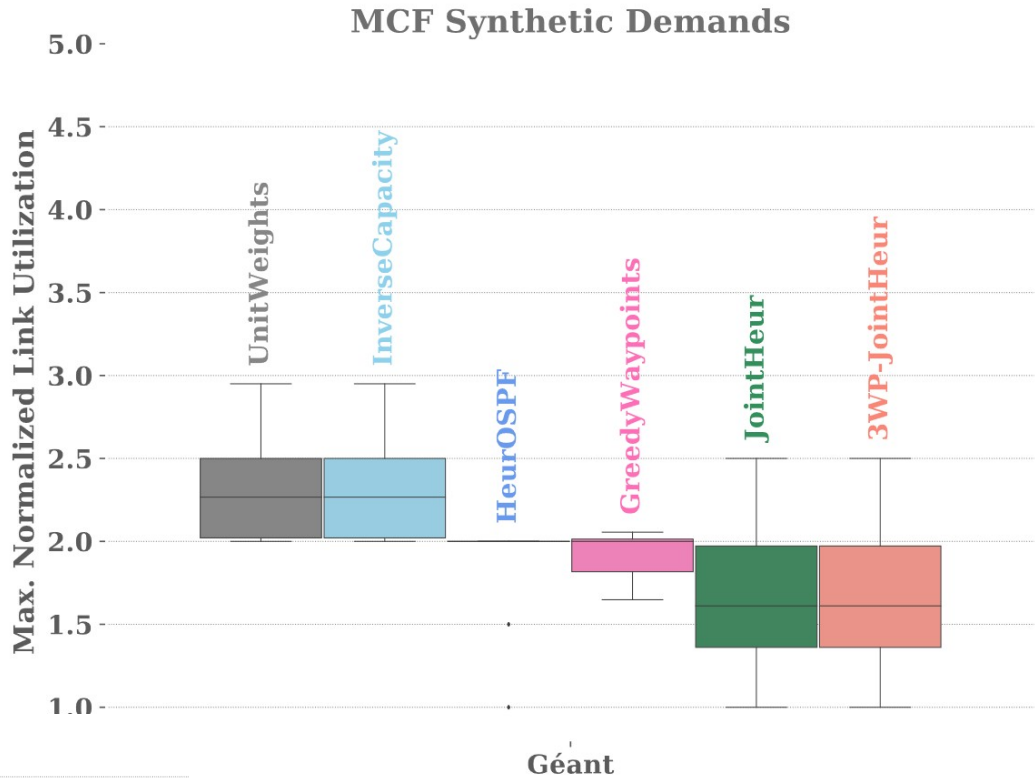
Sebastian Peters

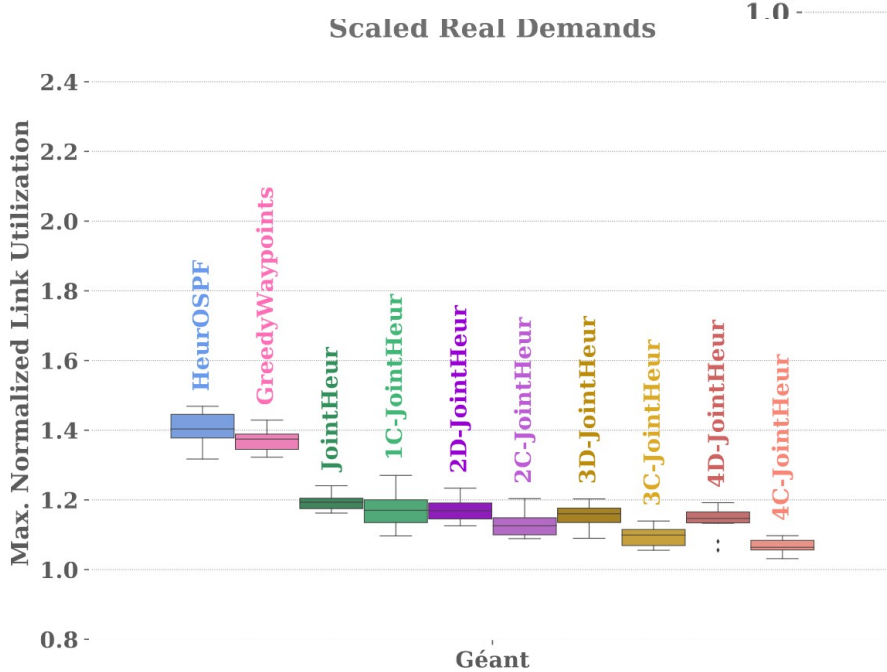
Georgios Karamoussanlis

Replikation Gruppe 1

- Verbindung mit IRB-Server (Debian 11.3)
- Anaconda installiert
- Repository (3124299) geklont
- Conda activate , Dependencies installiert
- Main() ruft Algo Main() auf 3x


MCF Synthetic Demands
Max. Normalized Link Utilization
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
UnitWeights
InverseCapacity
HeurOSPF
GreedyWaypoints
JointHeur
1/3N-JointHeur
Géant


MCF Synthetic Demands
Max. Normalized Link Utilization
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
UnitWeights
InverseCapacity
HeurOSPF
GreedyWaypoints
JointHeur
3WP-JointHeur
Géant


Scaled Real Demands
Max. Normalized Link Utilization
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
HeurOSPF
GreedyWaypoints
JointHeur
1C-JointHeur
2D-JointHeur
2C-JointHeur
3D-JointHeur
3C-JointHeur
4D-JointHeur
4C-JointHeur
Géant

Random Waypoints (1)

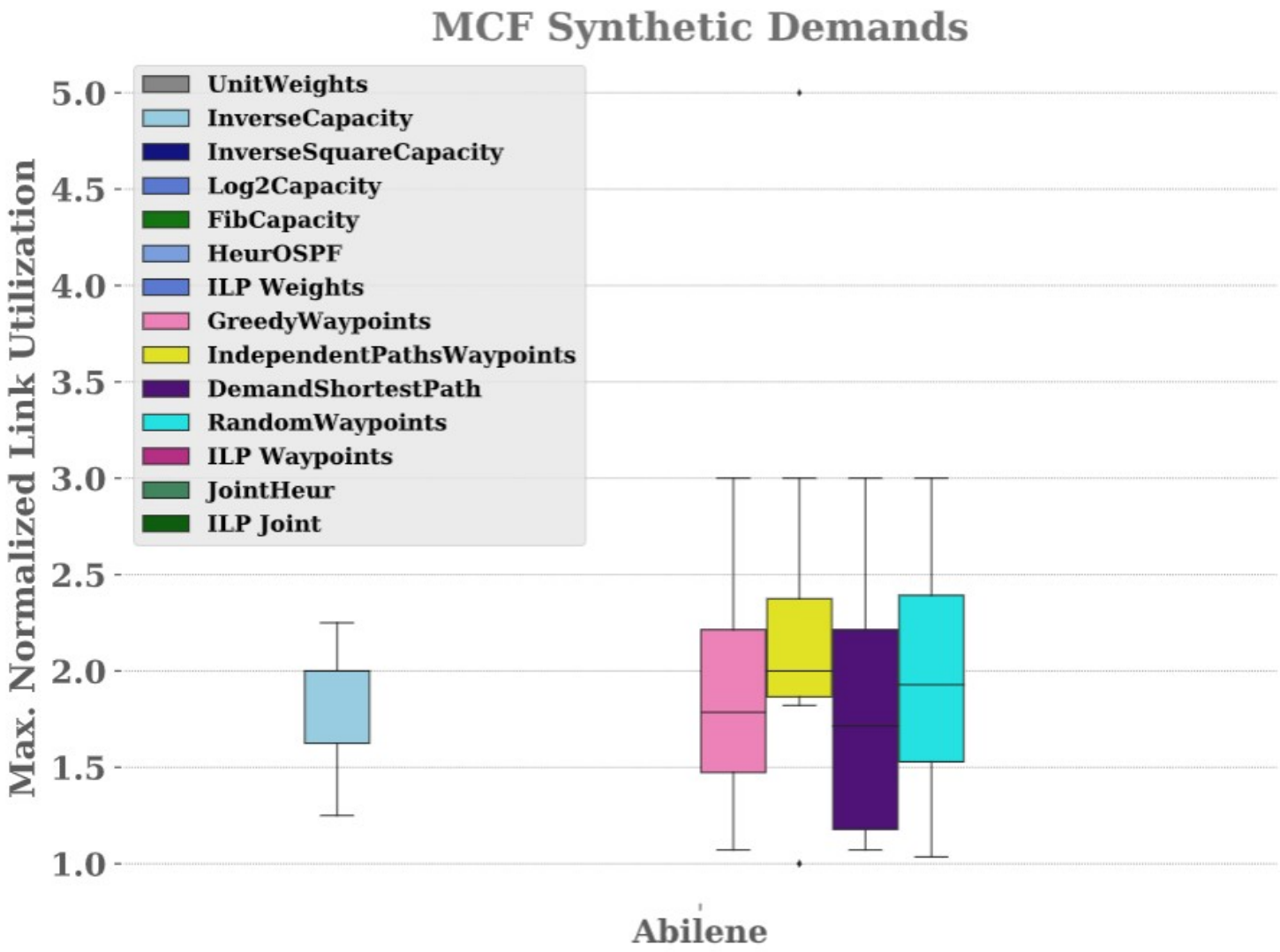
- Aus randomisierten Wegpunkten den Besten speichern

```
function randomWaypoints(){
    for demand in demand_list{

        //Variable für die Größe der randomWaypoints-Liste
        waypointCount = x

        check waypointCount < n

        //Liste für die Wegpunkt Indizes
        randomList = Liste aus "x" zufälligen Zahlen

        while( ein Element aus randomList Source / Target ist ){
            randomList = Liste aus "x" zufälligen Zahlen
        }

        for waypoint in randomList{

            wenn waypoint MLU verbessert dann speichern
        }
    }
}
```

Random Waypoints (2)

- Allgemeiner Vergleich

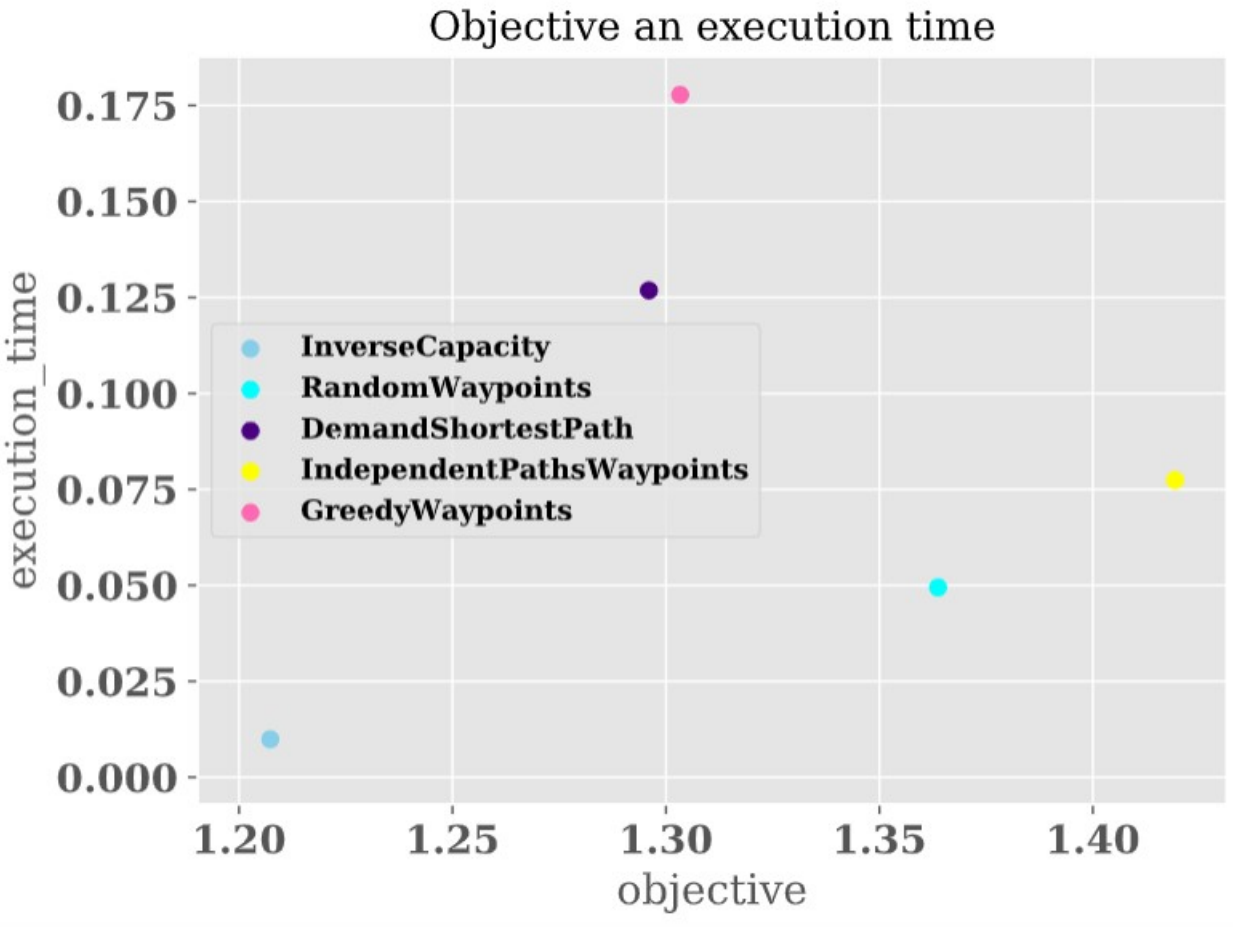
  MLU



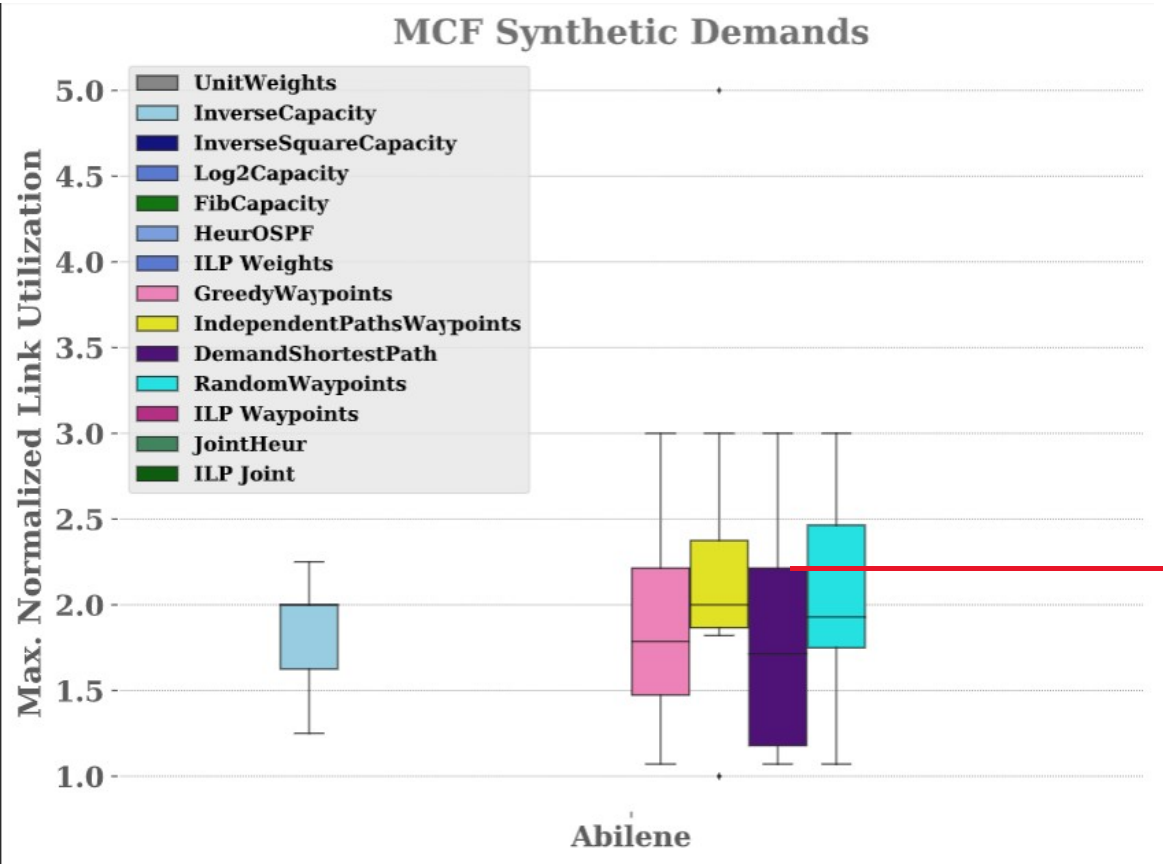
Random Waypoints (3)

- Allgemeiner Vergleich

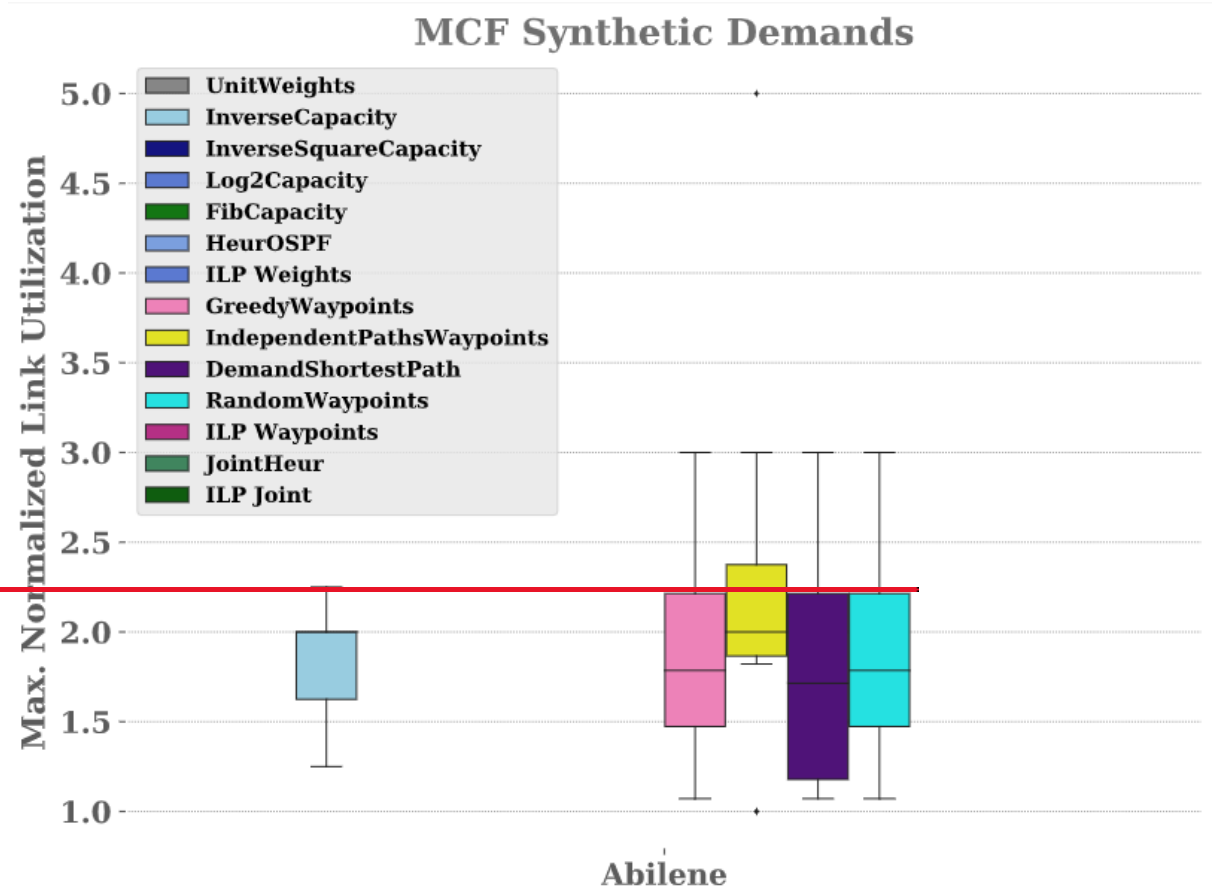
  Process-Time



Random Waypoints (4)



3 Wegpunkte



10 Wegpunkte

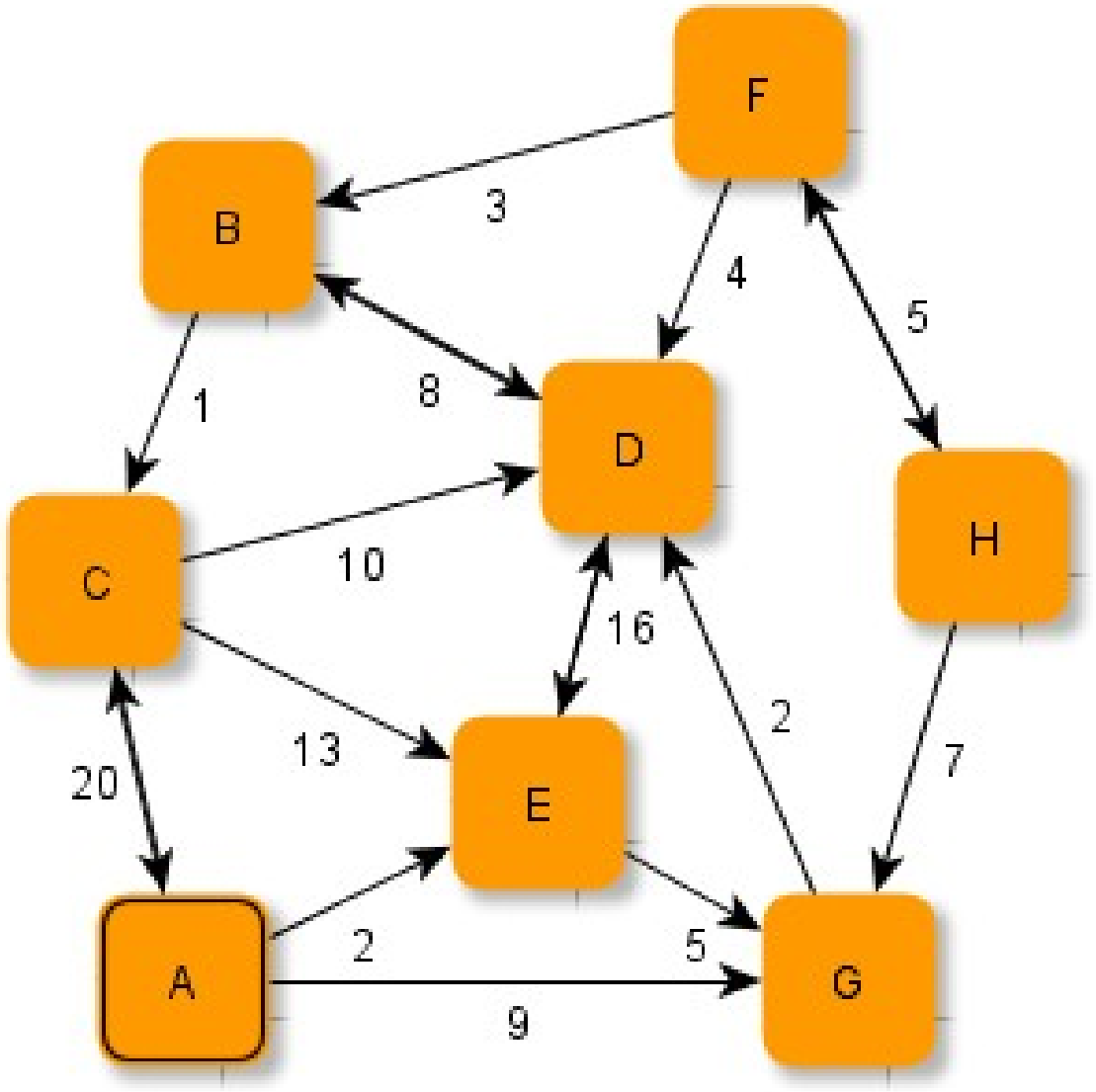
Demand Shortest Path (1)

- Shortest Path S für einen Demand bestimmen
- Mögliche Wege müssen entlang des Weges S liegen.

- Einschränkung der möglichen Wegpunkte
- Bessere Laufzeit

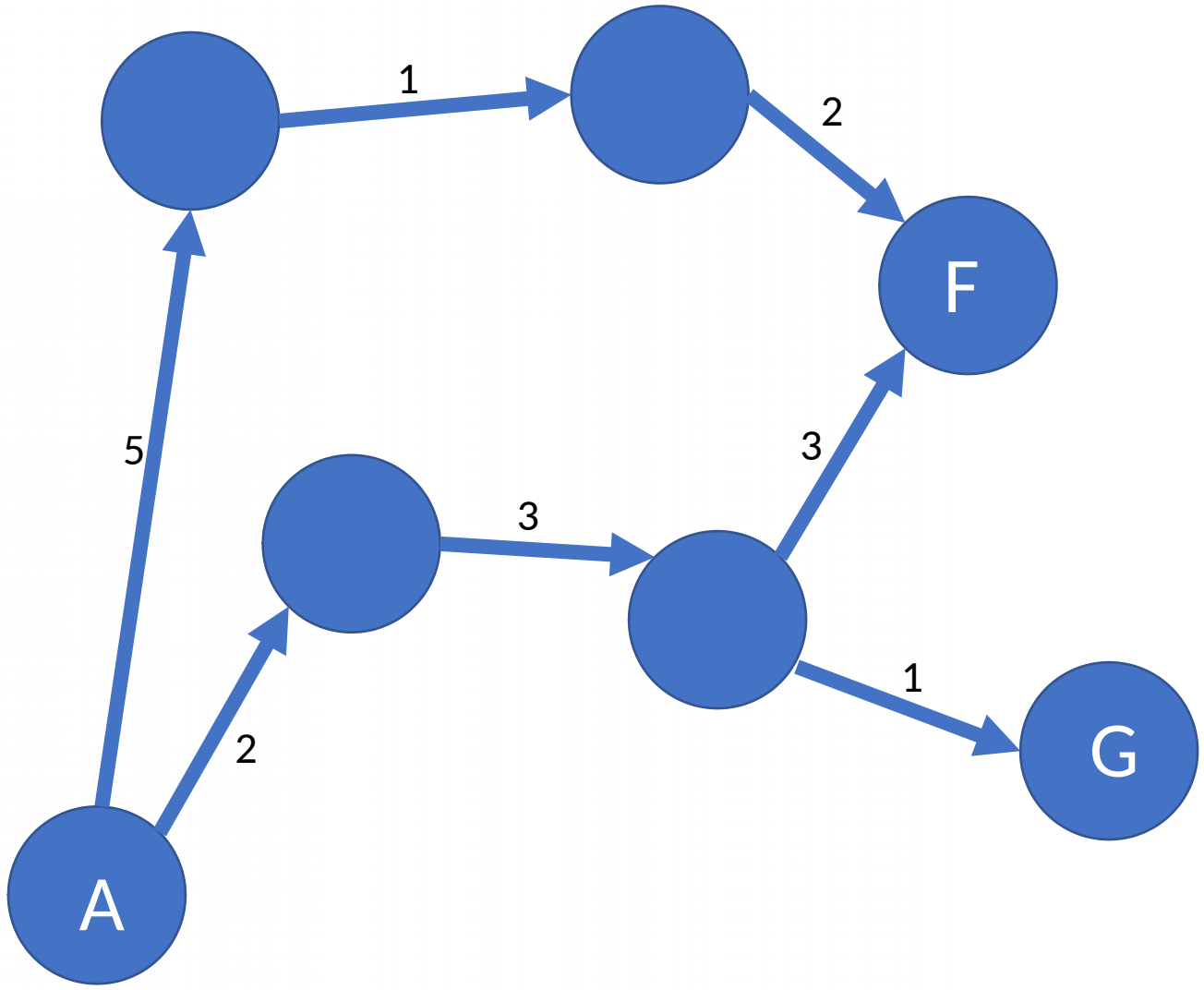
Demand Shortest Path (2)

```
demand_shortest_path(Graph g)
    for (s,t) in demands
        SPNL <- Dijkstra(G,s,t)
        pw <- {}
        for m in SPNL
            for b in adj(n)
                pw <- pw ∪ {b}
    demand_first_waypint(pw)
```



Independent Paths Waypoints(1)

- Greedy besten Weg finden
- Keine Limitierung der Wegpunkte
- Demand entlang eines kurzen Pfades leiten
- Möglichst wenig Überschneidungen der Wege



Independent Paths Waypoints(2)

```
IndependentPathsWaypoint(Graph G, Demands D, limit)

best_objective <- utilization()
paths = []
for demand in Demands:
    source, target <- demand
    best_path <- None
    i <- 0
    best_path = None
    for path in all_shortest_paths(G, source, target):
        if i > limit
            break
        else
            i++

        if len(path) == 2 // This is the best possible Path
            best_path = path
            break
        else
            add all Nodes on path as waypoints
            objective <- utilization()

            if objective > best_objective
                best_path <- path
                best_objective <- objective
            endif
        endif
    paths.append(best_path)
    endfor
endfor
return best_objective, paths
```

Vergleich aller Algorithmen