

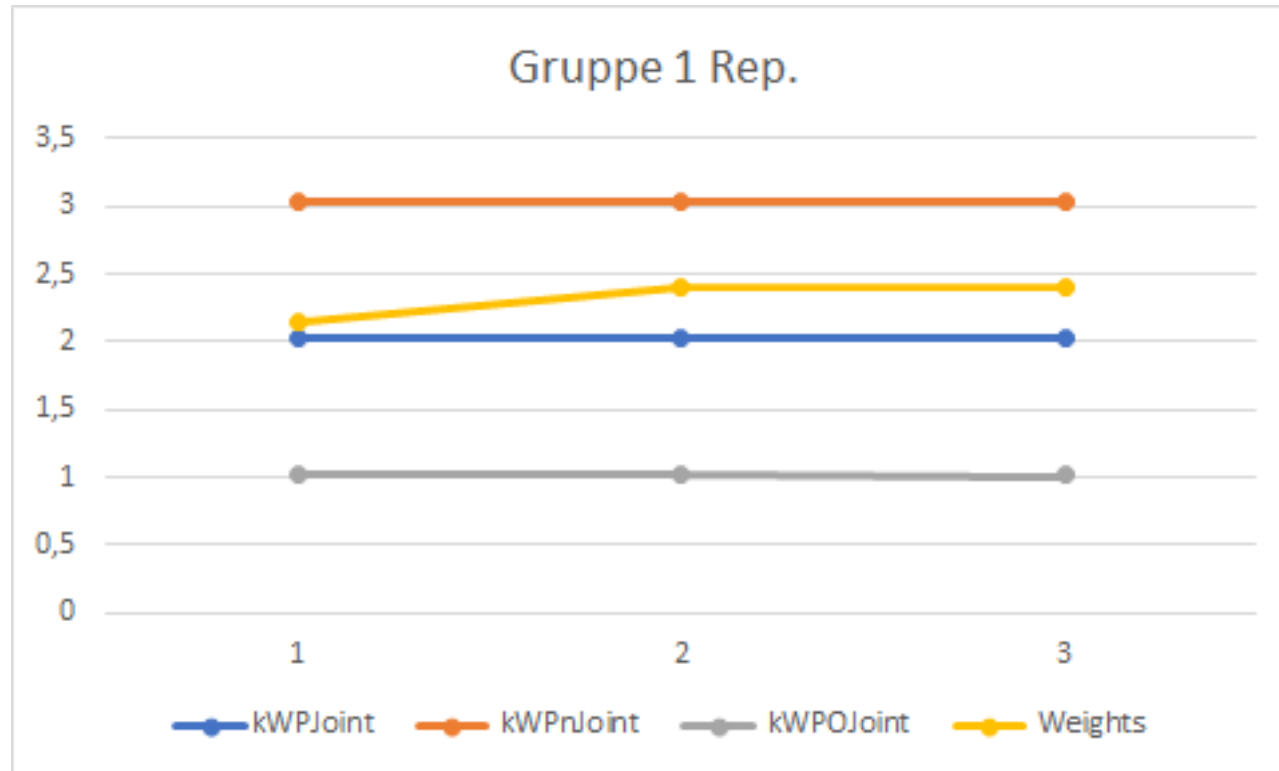
Projekt 2

Sebastian Peters
Marvin Weiler
Georgios

Replikation Gruppe 1

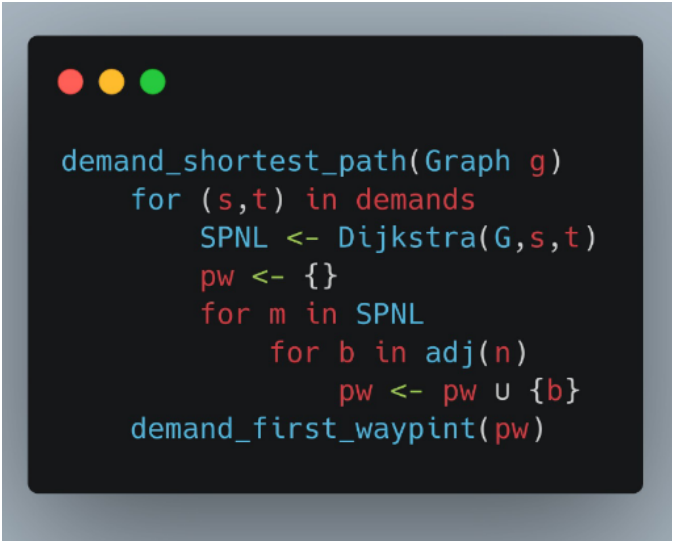
- Repository <https://github.com/gajanns/Fachprojekt2> geklont
- *.topo.py & *.topo.sh schon da
- Nanonet_batch.py ausgeführt

Replikation Gruppe 1



kWPJoint.topo.sh	2,01893296
kWPJoint.topo.sh	2,01893296
kWPJoint.topo.sh	2,01893296
kWPnJoint.topo.sh	3,028402939
kWPnJoint.topo.sh	3,028402709
kWPnJoint.topo.sh	3,028402939
kWPOJoint.topo.sh	1,013865253
kWPOJoint.topo.sh	1,013801627
kWPOJoint.topo.sh	1,013800955
Weights.topo.sh	2,144658736
Weights.topo.sh	2,396762245
Weights.topo.sh	2,396866853

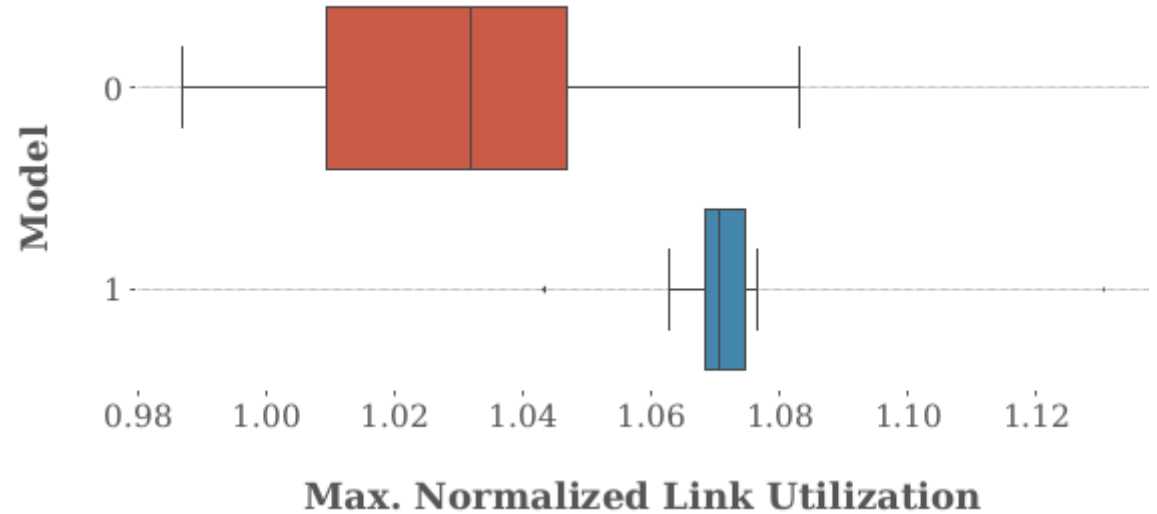
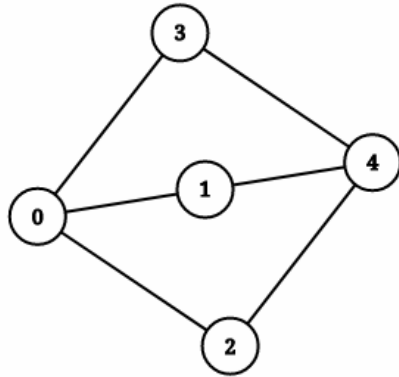
Algorithmus 1: DemandShortestPath (1)



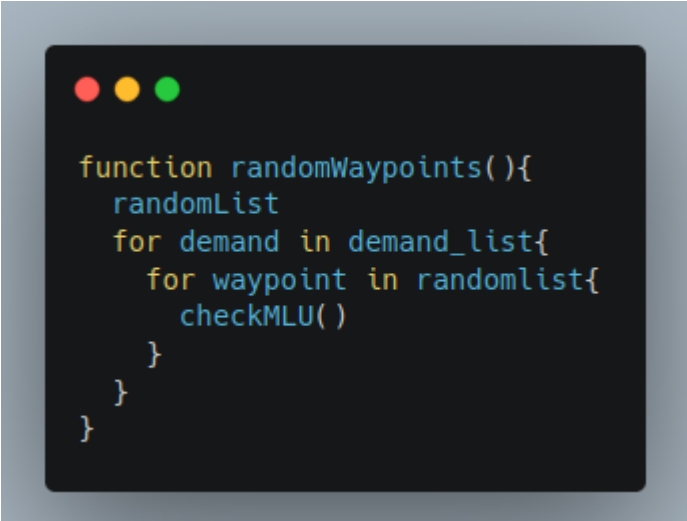
```
demand_shortest_path(Graph g)
  for (s,t) in demands
    SPNL <- Dijkstra(G,s,t)
    pw <- {}
    for m in SPNL
      for b in adj(n)
        pw <- pw u {b}
    demand_first_waypint(pw)
```

- Shortest Path für einen Demand bestimmen
- Mögliche Wege müssen entlang des SP liegen

Algorithmus 1: DemandShortestPath (2)



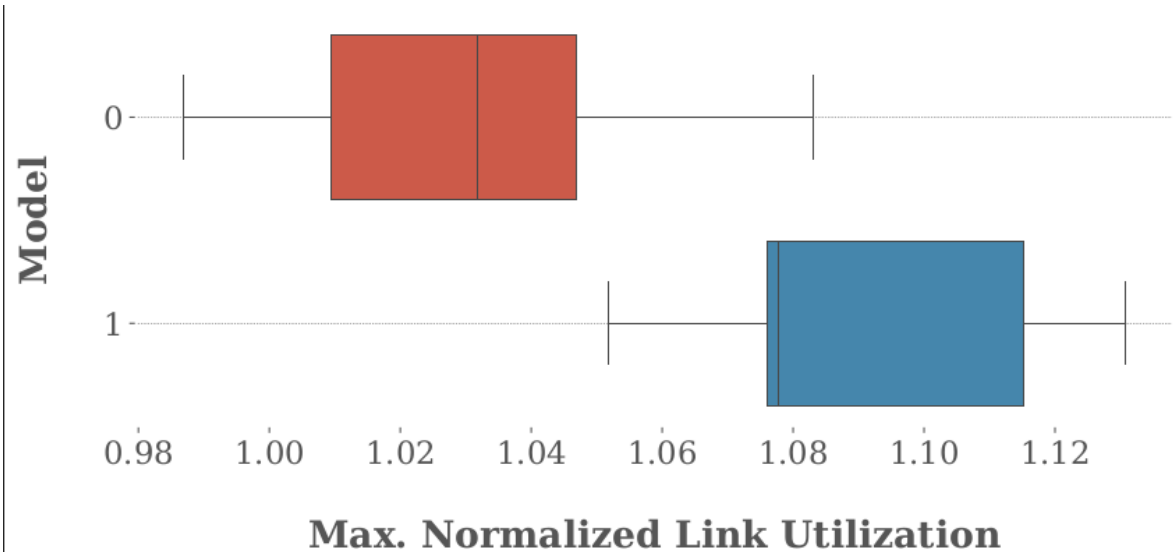
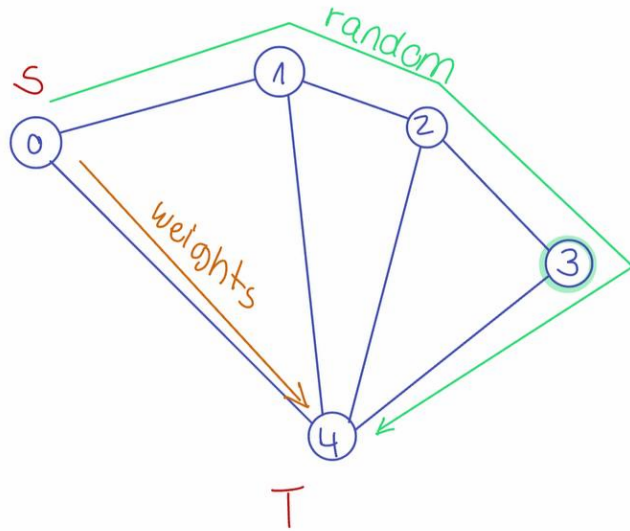
Algorithmus 2: RandomWaypoints (1)



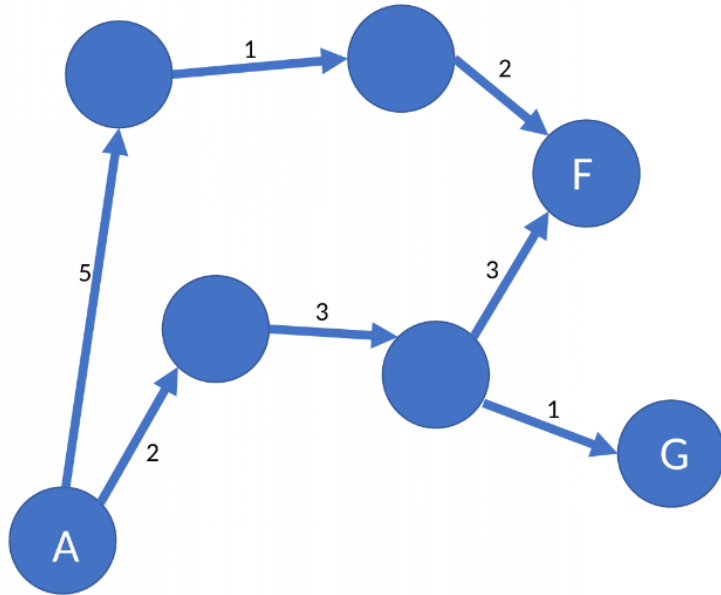
```
function randomWaypoints(){  
  randomList  
  for demand in demand_list{  
    for waypoint in randomlist{  
      checkMLU()  
    }  
  }  
}
```

- Aus randomisierten Wegpunkten den besten speichern

Algorithmus 2: RandomWaypoints (2)

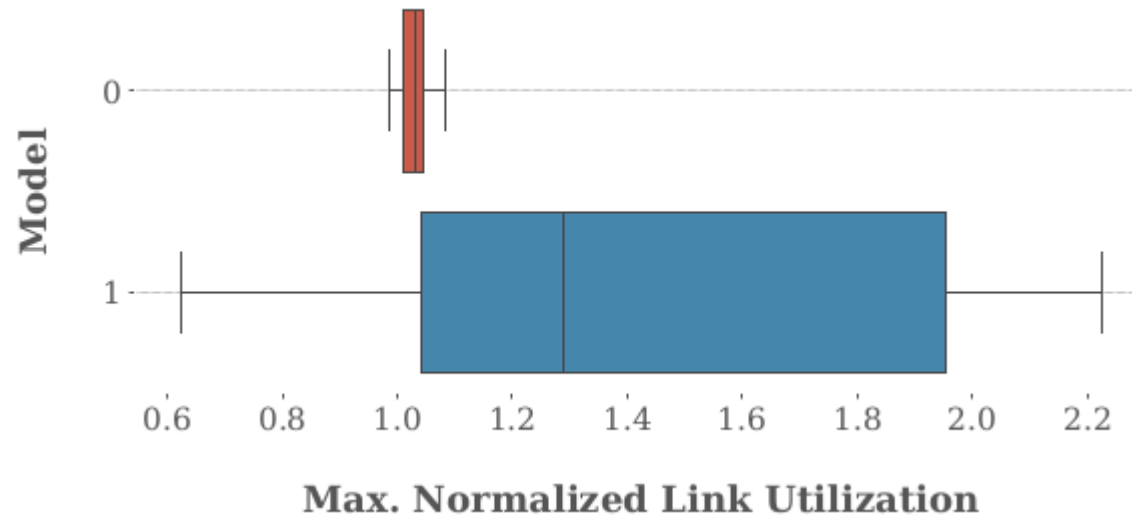


Algorithmus 3: IndependentPathsWaypoints (1)



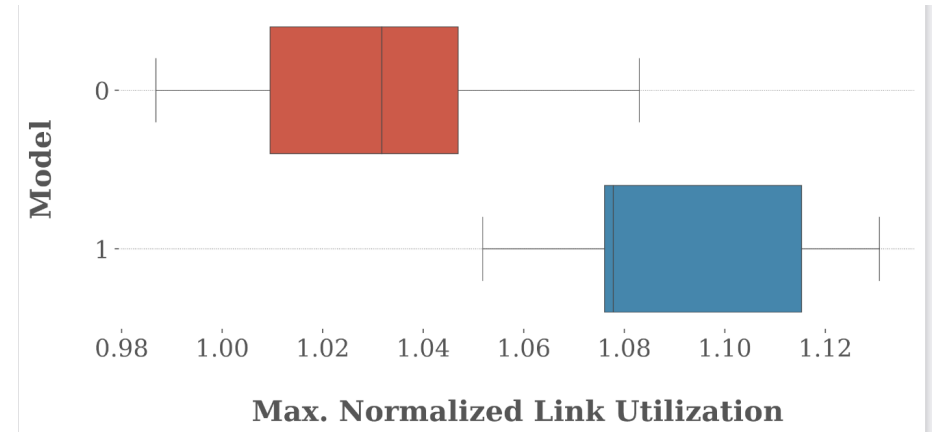
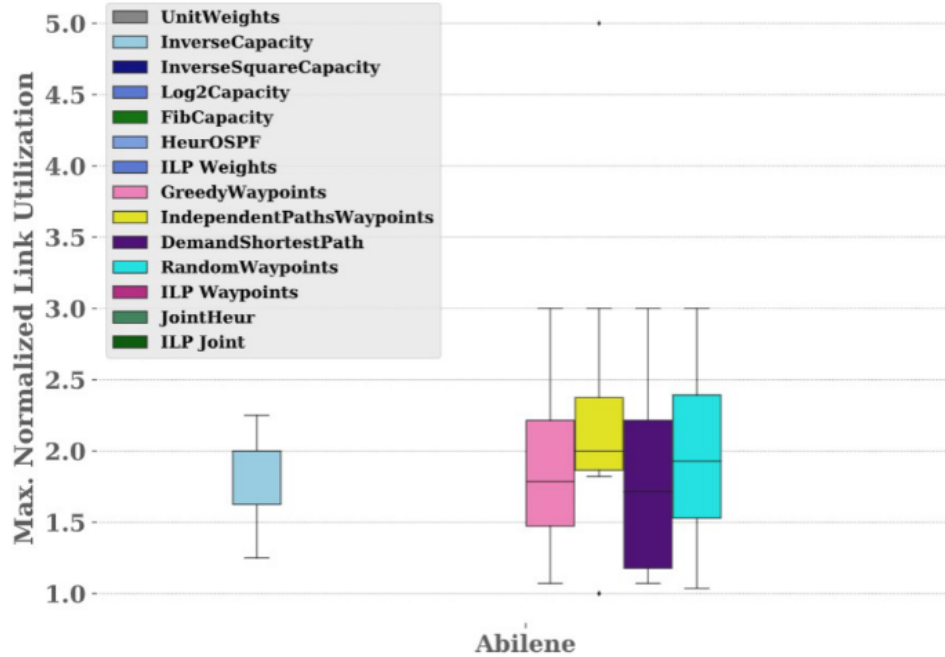
- Greedy besten Weg finden
- Keine Limitierung der Wegpunkte
- Demand entlang eines kurzen Pfades leiten
- Möglichst wenig Überschneidungen der Wege

Algorithmus 3: IndependentPathsWaypoints (2)



Realität vs. Python

MCF Synthetic Demands



Ausblick

- Probleme mit independentPathsWaypoints
- Laufzeitberechnung fehlt
- Algorithmus in Nanonet nicht möglich
- Simulation in viel größeren Netzwerken
- Simulation in realen Netzwerken
- Random Algo in Hardware umsetzbar ?