technische universität
dortmund

Report

Gruppe 3

Simon Buschmann, Yannik Buchner - Least Loaded Link First (LLLF)
Nikita Podibko, Jan Draeger - Avarage Link Utilization/Random Load Aware
Johannes Heinrich, Malek Haoues Rhaiem - Average Path Length (APL)
14. September 2025

Gutachter:
Prof. Klaus-Tycho Förster

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für praktische Informatik (Lehrstuhl IV)
https://ms.cs.tu-dortmund.de/

Inhaltsverzeichnis

Kapitel 1
Einleitung

1.1 Motivation

Routing-Algorithmen sind essenziell für die effiziente Datenübertragung in Computernetzwerken und helfen dabei, den optimalen Pfad für den Datenfluss zwischen zwei oder mehr Knotenpunkten zu bestimmen. Zu den bekanntesten Algorithmen gehören Dijkstra's Algorithmus, der auf den kürzesten Weg fokussiert, und der Bellman-Ford-Algorithmus, der bei dynamischen Netzwerken mit wechselnden Kosten funktioniert.
In dieser Arbeit haben wir uns das Paper [5] und die zwei dazugehörigen Repositories [4] und [3] angeschaut. Dabei haben wir versucht, die Ergebnisse zu reproduzieren. Auch haben wir uns drei weitere Algorithmen überlegt und diese in das bestehende Projekt eingearbeitet. Unsere Erweiterungen kann man hier finden [2] und [1].

1.2 Aufbau der Arbeit

Innerhalb dieses Reportes stellen wir unsere Ergebnisse bei der Einarbeitung unserer eigenen drei Algorithmen vor. Dabei ist jedem Algorithmus ein Kapitel gewidmet (2, 3, 4). Wir erläutern dabei die Idee des Algorithmus, die jeweiligen Vor- und Nachteile, die Schwierigkeiten, welche wir bei der Einarbeitung in die beiden Projekte hatten, und die Ergebnisse, die wir mit den Algorithmen erzielt haben. In Kapitel 5 gehen wir noch auf die Reproduktion der Gruppe 4 ein, welche auch zwei eigene Algorithmen entwickelt hat.

Kapitel 2

Least Loaded Link First Algorithmus

Der *Least Loaded Link First* (LLLF) Algorithmus ist ein heuristisches Verfahren zur Pfadwahl in Kommunikationsnetzen. Ziel ist es, Anfragen für Datenübertragungen so durch das Netzwerk zu leiten, dass die Auslastung einzelner Kanten möglichst gleichmäßig verteilt wird (Maximum Link Utilisation MLU). Dadurch sollen Engpässe vermieden und die Gesamteffizienz des Netzwerks gesteigert werden.

2.1 Grundidee

Im Gegensatz zu klassischen Routingverfahren, die z. B. nur auf minimale Pfadlänge oder Kosten achten, berücksichtigt der LLLF-Ansatz die aktuelle Belastung der Netzwerkressourcen. Jeder Übertragungswunsch (Demand) wird nacheinander betrachtet, und für ihn wird ein Pfad gesucht, der die zukünftige maximale Auslastung des Netzes minimal hält. Dies geschieht durch eine Abwandlung des Dijkstra-Algorithmus, die nicht die Länge der Pfade, sondern die relative Auslastung der Links als Kostenfunktion verwendet.
In Algorithmus 2.1 wurde eine sehr simple Umsetzung in Pseudo-Code erstellt.

Algorithmus 2.1 LLLF-Solver

Eingabe: $G(V, E, c : E \mapsto \mathbb{R})$, $D = \{D_1 = (s_1, t_1, d_1), D_2, \ldots, D_n\}$
Ausgabe: $P = \{P_1, P_2, \ldots, P_n\}$

```
1:  for e ∈ E do
2:      loads_e ← 0
3:  end for
4:  P ← {}
5:  for i ∈ {1, ..., n} do
6:      P_i ← PotentialUtilizationDijkstra(G, loads, D_i)
7:      for e ∈ P_i do
8:          loads_e ← loads_e + d_i
9:          utilization_e ← loads_e / c(e)
10:     end for
11:     P ← P ∪ {P_i}
12: end for
13: return P
```

Algorithmus 2.2 PotentialUtilizationDijkstra

Eingabe: $G(V, E, c : E \mapsto \mathbb{R})$, $loads$, $D_i = (s_i, t_i, d_i)$

Ausgabe: P_1

1: **for** $v \in V$ **do**
2: $utilization_v \leftarrow \infty$
3: $predecessor_v \leftarrow -1$
4: $visited_v \leftarrow$ False
5: **end for**
6: $utilization_{s_i} \leftarrow 0$
7: **while** $visited_{t_i} ==$ False **do**
8: $u \leftarrow v$ with minimal $utilization_v$
9: **for** $n \in neighbours(u)$ **do**
10: **if** $utilization_n > \max\{utilization_u, (loads_{(u,n)} + d_i))/c(u, n)\}$ **then**
11: $predecessor_n \leftarrow u$
12: $utilization_n \leftarrow \max\{utilization_u, (loads_{(u,n)} + d_i))/c(u, n)\}$
13: **end if**
14: **end for**
15: $visited_u \leftarrow$ True
16: **end while**
17: $P_1 \leftarrow$ Generate Path with $predecessor$ starting at $predecessor_t$
18: **return** P_1

2.2 Eigenschaften

- Fairness: Der Algorithmus vermeidet es, wiederholt denselben am wenigsten belasteten Link zu verwenden, sondern strebt eine gleichmäßigere Auslastung des Netzwerks an.

- Greedy-Heuristik: LLLF trifft jeweils nur eine lokal optimale Entscheidung für die aktuelle Anfrage, ohne globale Optimierung über alle Demands hinweg. Dadurch ist er rechenökonomisch, erreicht aber nicht immer die theoretisch bestmögliche Gesamtauslastung.

- Laufzeit: Der Algorithmus läuft effizient, wobei die Laufzeit ungefähr der von Dijkstra entspricht.

- Praktische Relevanz: In großen Telekommunikations- und Datennetzen ist LLLF attraktiv, weil er relativ einfach implementierbar ist und dynamisch auf Änderungen im Netz reagieren kann.

2.3 Ergebnisse

In den Ergebnissen aus Projekt 1 2.2 sieht man, wie der LLLF-Algorithmus im Vergleich zu den anderen Algorithmen abschneiden. Der LLLF-Algorithmus hat dabei die Farbe Grau. Bei den synthetischen Demands in 2.1a und 2.1b liegt der Algorithmus bei der MLU im Mittelfeld, wobei es ein paar Topologien gibt, bei welchen er sehr gut abscheidet. Bei der Laufzeit kann man den Algorithmus auch bei den anderen schnellen Algorithmen verordnen. Bei den echten Demands in 2.1c und 2.1d liegt der Algorithmus auch wieder im Mittelfeld. Dort schneiden, die zwei komplexen Algorithmen teilweise besser ab, brauchen dafür aber auch 1000-mal so lange. Der LLLF-Algorithmus lässt sich hier also auch wieder bei den schnellen anderen Algorithmen, wie Inverse Capacity und Greedy Waypoints einordnen.

In Projekt zwei wurden der Algorithmus dann auf einem eigenen Topologie simuliert. Da in Projekt 1 schon festgestellt wurde, dass der Algorithmus gut auf dichten Topologien, also welche mit vielen Kanten, funktioniert, haben wir folgende eigene Topologie 2.2a erstellt, wobei alle Kanten Kapazität 2 haben.

Die benutzen Demands lauten wie folgt:

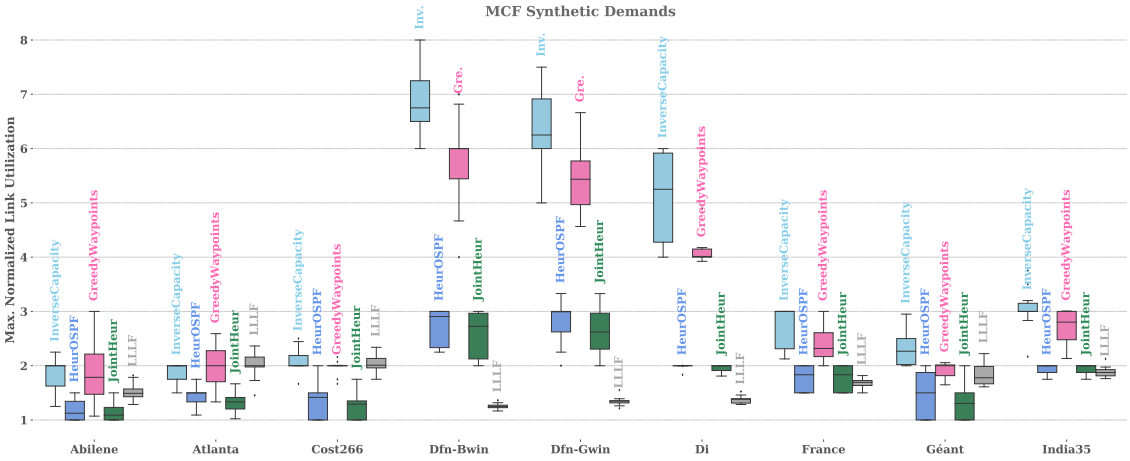
- $s = 8,\ t = 1,\ d = 1.0$ $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$ $s = 8,\ t = 1,\ d = 1.0$
- $s = 8,\ t = 1,\ d = 1.0$ $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$ $s = 6,\ t = 4,\ d = 1.0$
- $s = 6,\ t = 4,\ d = 1.0$ $s = 6,\ t = 4,\ d = 1.0$

Die theoretische MLU die erreicht werden müsste wäre 0.5.
Man sieht in der Abbildung 2.2b, dass die theoretische MLU leider nicht erreicht wurde. Mögliche Probleme, die bei dazu geführt haben könnten, wären:

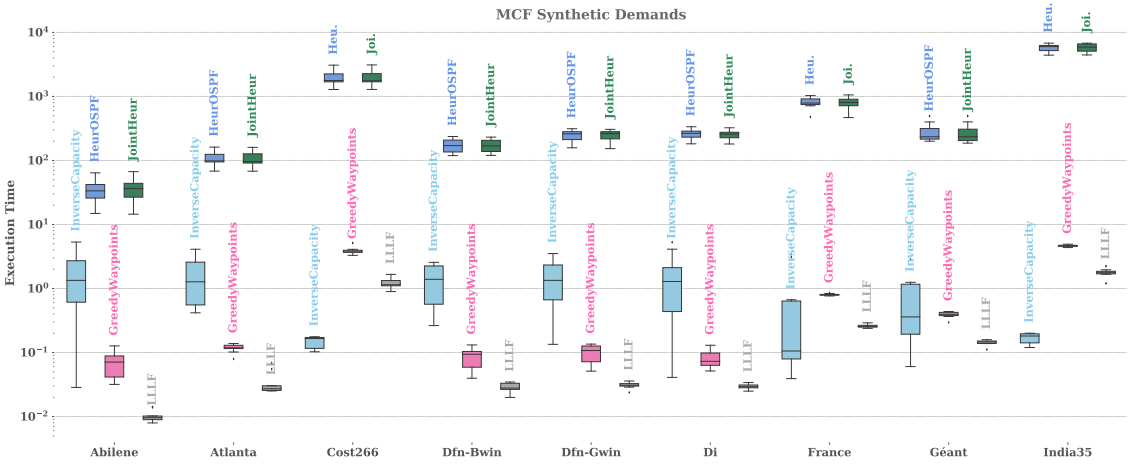
- Die ersten paar Tests waren fehlerhaft:
  - nuttcp-t: v6.1.2: Error: connect: Connection refused
  - nuttcp-r: v6.1.2: Error: bind: Address already in use
- Auch könnte es Probleme bei mehreren Demands mit selben Start und Ziel geben.
- Möglicherweise war die Topologie auch zu komplex für die Simulation.

Abbildung 2.1: Ergebnisse LLLF Projekt 1

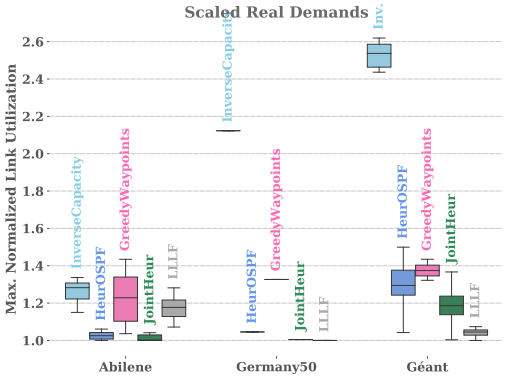
(a) Synthetische Demands MLU



(b) Synthetische Demands Laufzeit



(c) Echte Demands MLU



(d) Echte Demands Laufzeit

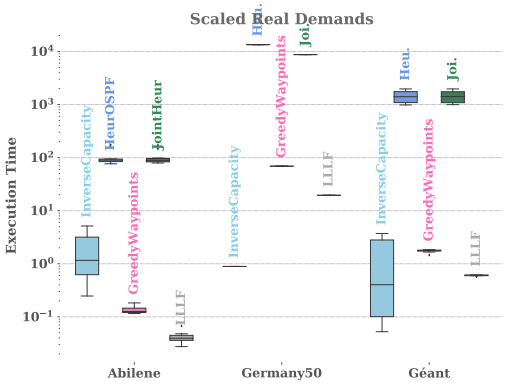
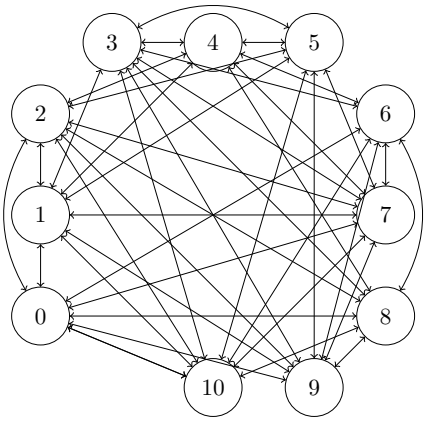
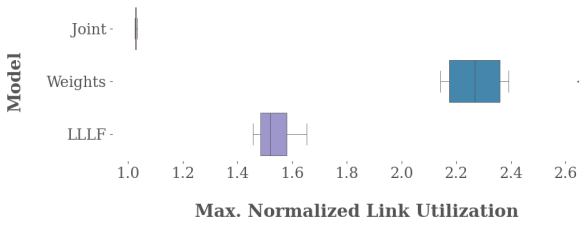


Abbildung 2.2: Ergebnisse LLLF Projekt 2

(a) LLLF-Topologie



(b) Ergebnisse LLLF Projekt 2



2.4 Mögliche Verbesserungen und Ausblick

Um den Algorithmus noch besser zu machen, gibt es noch mögliche Verbesserungen, welche für die Ergebnisse aus den Projekten 1 und 2 teilweise schon implementiert wurden. Die erste Idee war, die Demands vorher zu sortieren, um die großen Demands auf die Topologie zu verteilen und danach mit kleinen aufzufüllen. Die zweite Idee war, bei mehreren besten Pfaden (geringste MLU), erst den kürzesten Pfad zu betrachten. Dadurch wird zu Topologie nicht unnötig belastet. Als dritte Idee, haben wir den Algorithmus nach initialer Verteilung der Demands auf Pfade mehrmals wiederholt ausgeführt, wobei bei jedem Durchlauf für jeden Demand geschaut wurde, ob ein neuer optimaler Pfad existiert.

Abschließend kann man sagen, dass der LLLF-Algorithmus eine positive Bilanz hatte und sich gut in den schon existierenden Algorithmen angeschlossen hat. Der Algorithmus ist ein Beispiel dafür, wie klassische Graphenalgorithmen (hier: Dijkstra) anwendungsorientiert modifiziert werden können, um Lastverteilungen in Netzwerken effizient zu steuern. Trotz seiner heuristischen Natur stellt er in vielen praxisnahen Szenarien eine effektive Methode dar, um Netzwerkengpässe zu vermeiden und die Ressourcenauslastung zu balancieren.

Kapitel 3
Randomized Load Aware Path Selection Algorithmus

Der Algorithmus Randomized Load-Aware Path Selection (RLAPS) ist ein heuristisches Routingverfahren, das entwickelt wurde, um Netzwerke gegen lokale Überlastung abzusichern und dabei dennoch kurze Pfade zu berücksichtigen. Anstelle deterministischer Shortest-Path-Strategien, die häufig zur Ausbildung von Hotspots führen, nutzt der Ansatz kontrollierte Randomisierung, um Diversität in der Pfadwahl zu erzeugen und dadurch den Datenverkehr gleichmäßiger im Netz zu verteilen. Ziel des Verfahrens ist es, eine einfach implementierbare und robuste Methode zur Lastverteilung bereitzustellen, die starke Auslastungsspitzen reduziert und die durchschnittliche Auslastung verbessert, ohne dabei anfällig für kleine Eingangsvariationen in Demands oder Topologie zu sein.

3.1 Grundidee

Die Grundidee besteht darin, für jede Demand zunächst eine Menge von k-kürzesten Pfaden zu berechnen. Aus dieser Menge werden anschließend drei Kandidaten zufällig ausgewählt und anhand einer lastbasierten Bewertungsfunktion beurteilt. Diese Funktion berücksichtigt die bereits auf den Links geplante Last sowie die jeweilige Kapazität und bestimmt so den Pfad, der die geringste zusätzliche Belastung erzeugt. Der beste der drei Kandidaten wird als endgültiger Pfad gewählt, die entsprechende Last auf den Links eingetragen und die lokalen Informationen über die Auslastung aktualisiert.

3.2 Wie der Algorithmus funktioniert

Kurzüberblick:

1. Baue einen Graphen mit Kantenkapazitäten und optionalen Kantengewichtungen auf.

2. Für jede Demand berechne eine Liste von k-shortest paths und wähle daraus 3 Kandidaten randomisiert (Oversampling).

3. Bewerte jeden Kandidatenpfad anhand einer Lastfunktion, die die derzeit erwartete Linkauslastung berücksichtigt.

4. Wähle den besten Pfad (minimaler Score). Trage die Last ein und aktualisiere die lokalen Lastinformationen.

3.3 Bewertungsfunktion

Sei P ein Pfad bestehend aus Kanten $(e_1, \ldots, e_m)$ und d die Demand-Größe. Mit f_e der bisher auf e geplanten Last und c_e der Kapazität gilt die Pfadbewertungsfunktion

$$\text{score}(P, d) = \sum_{e \in P} \frac{f_e + d}{c_e}$$

Diese Wahl zielt direkt darauf ab, die entstehende Auslastung auf den von P genutzten Links zu minimieren.

3.4 Pseudocode

```
for each demand (s,t,d):
candidates = k_shortest_paths(G,s,t,K,oversample)
sampled = random_sample(candidates, K)
best = argmin_{p in sampled} score(p; d, flow\_map)
assign best: update flow_map and G loads
return waypoints, metrics (MLU, ALU, ...)
```

3.5 Experimente 1

Im Rahmen der Experimente wurde der Algorithmus Randomized Load-Aware Path Selection (RLAPS) in die bestehende Testumgebung integriert. Die Implementierung selbst erwies sich vergleichsweise unkompliziert, da der Algorithmus im Kern nur auf der Berechnung von k-kürzesten Pfaden sowie einer einfachen Bewertungsfunktion basiert. Das Ziel der Experimente bestand nicht allein darin, die maximale Linkauslastung (MLU) zu reduzieren, sondern auch die durchschnittliche Linkauslastung (ALU) zu verbessern. Durch die randomisierte Auswahl und die lastbewusste Pfadbewertung sollte der Algorithmus eine gleichmäßigere Lastverteilung im Netzwerk ermöglichen.

3.6 Resultate 1

Die Auswertung der Experimente erfolgte anhand der Metriken Average Normalized Link Utilization (ALU), Maximal Normalized Link Utilization (MLU) sowie der Ausführungszeit. Die Ergebnisse sind in den Abbildungen aus den Experimenten dargestellt.

In Bezug auf die durchschnittliche Linkauslastung (ALU) zeigt Abbildung 3.1 dass RLAPS im Vergleich im Mittelfeld liegt. Dennoch wird sichtbar, dass RLAPS einen Beitrag zur Reduzierung der durchschnittlichen Last im Netzwerk leistet und damit das angestrebte Ziel, neben der MLU auch die ALU zu verbessern, grundsätzlich erfüllt.

Betrachtet man die maximale Linkauslastung (MLU), so wird in Abbildung 3.2 deutlich, dass RLAPS in nahezu allen getesteten Topologien eine Verringerung von Hotspots erreicht. Während einfache Verfahren wie Inverse Capacity oder Greedy Waypoints oft hohe Spitzenlasten verursachen, gelingt es RLAPS durch die randomisierte Pfadwahl und lastbasierte Bewertung, die maximale Auslastung einzelner Links abzufedern.

Ein weiterer Aspekt betrifft die Ausführungszeit, wie in Abbildung 3.3 dargestellt. Hier zeigt sich, dass RLAPS zwar deutlich langsamer arbeitet als einfache Verfahren wie Unit Weights, jedoch im Vergleich zu exakten den anderen Ansätzen erheblich schneller ist. Die Implementierung des Algorithmus ist zwar sehr simpel und benötigt lediglich lokale Lastinformationen, die Berechnung von k-kürzesten Pfaden und die wiederholte Bewertung mehrerer Kandidatenpfade führen jedoch zu einem deutlichen Mehraufwand. Besonders auffällig ist, dass die Rechenzeit mit wachsender Topologiegröße und komplexität exponentiell angestiegen ist, was die praktische Anwendbarkeit in sehr großen Netzwerken einschränken kann. Diese eingeschränkte Skalierbarkeit stellt damit eine der wesentlichen Limitationen von RLAPS dar.

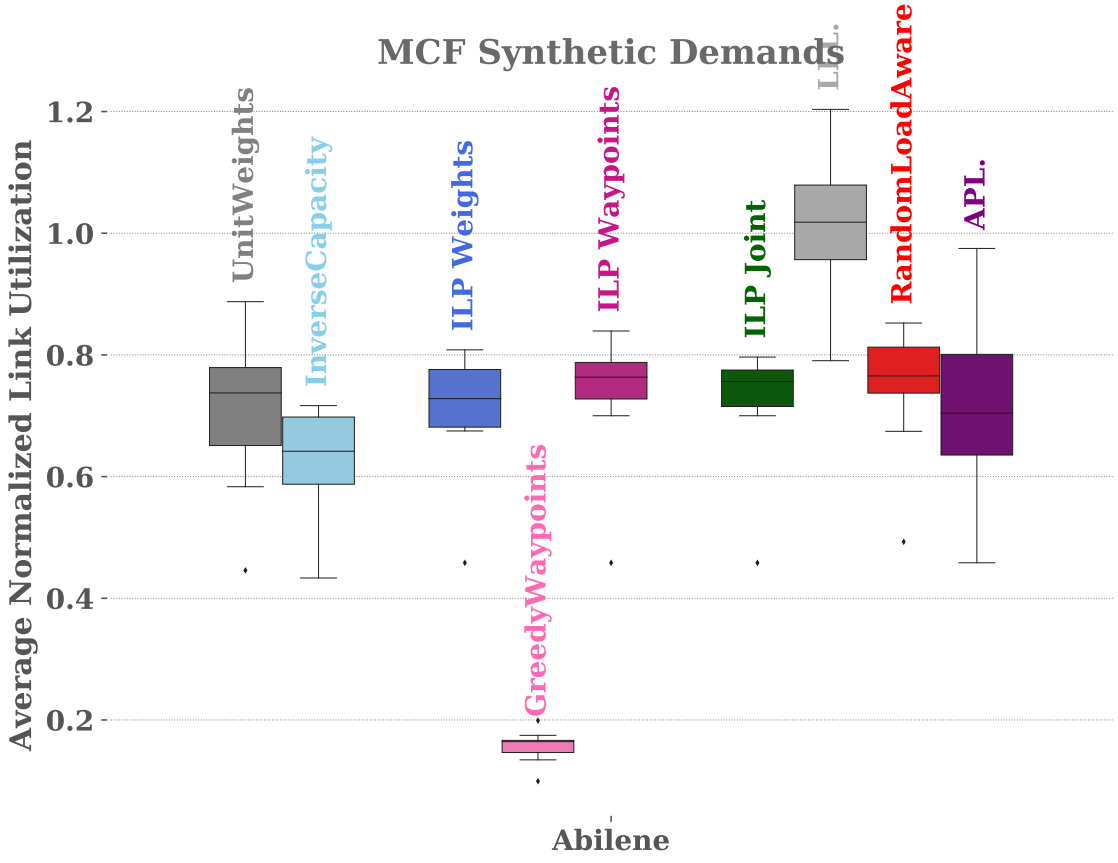


Abb. 3.1: Synthetische Demands ALU

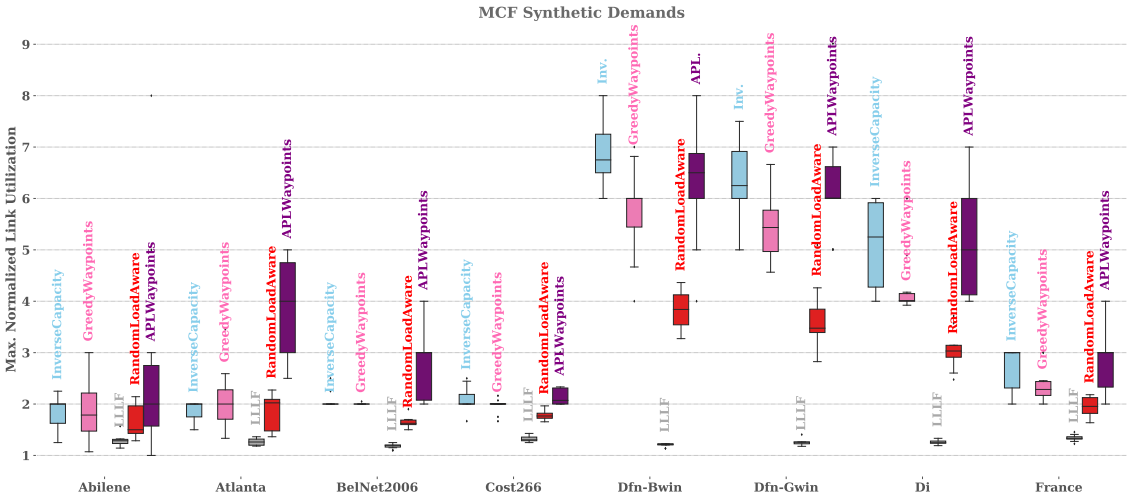


Abb. 3.2: Alle Algorithmen MLU

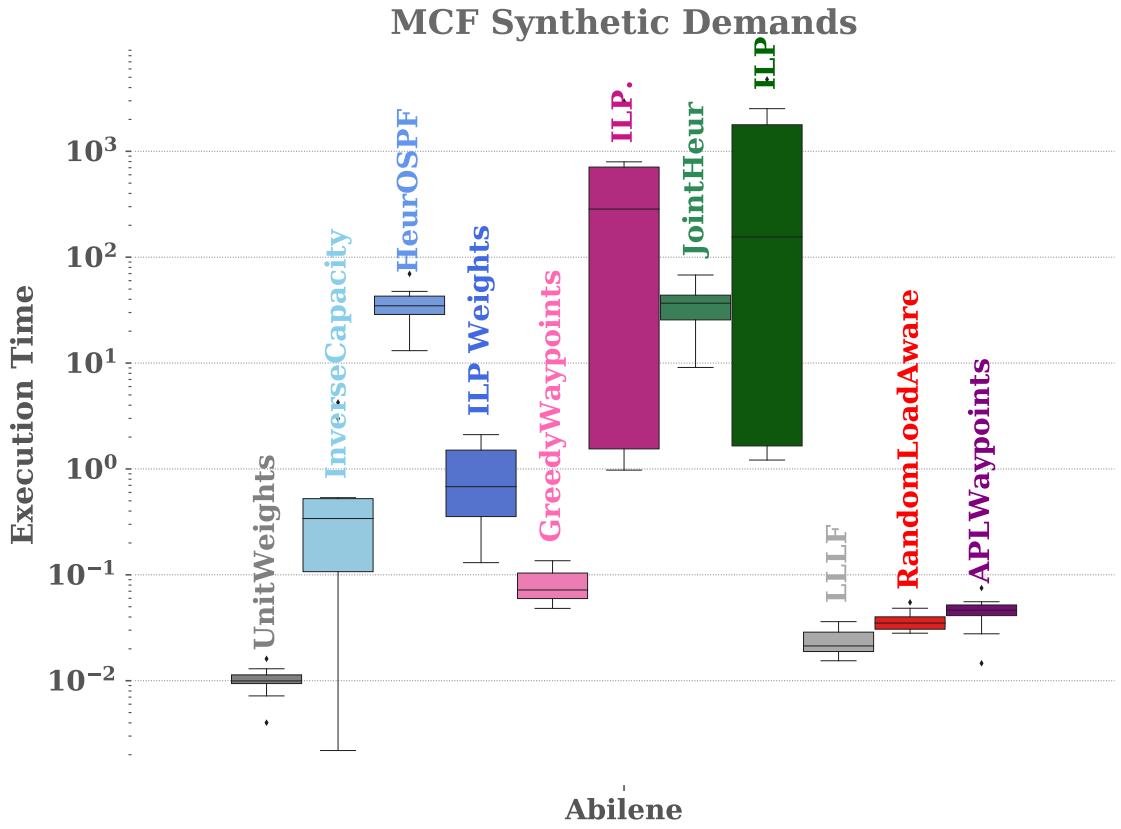


Abb. 3.2: Alle Algorithmen Ausführungszeit

3.7 Experimente 2

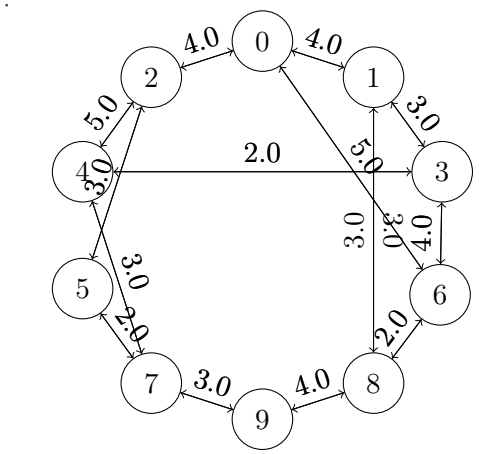


Abb. 4: Topologie

Im zweiten Experiment wurde die Topologie aus Abbildung 4 entworfen, die gezielt die gut geeignet für Randomized Load-Aware Path Selection (RLAPS) ist. Sie bietet eine hohe Pfadvielfalt, wodurch die randomisierte Vorauswahl tatsächlich verschiedene Alternativen berücksichtigen kann. Zentrale Engpässe wurden vermieden, sodass Lasten besser verteilt werden können. Die Kapazitäten wurden konsistent modelliert, um die Bewertungsfunktion

nicht zu verzerren, und die Netzgröße wurde moderat gehalten, damit die Berechnung der k-kürzesten Pfade handhabbar bleibt. Zusätzlich wurden für die Reproduzierbarkeit der Ergebnisse Anpassungen an den Bitraten vorgenommen, sodass die Experimente zügig wiederholt werden können.

3.8 Resultate 2

RLAPS wurde direkt mit der Weights-Methode auf der erstellten Topologie verglichen (vgl. Abbildung 4). Während Weights eine konsistent niedrigere maximale Linkauslastung erzielt und nur geringe Varianz aufweist, zeigt RLAPS ein deutlich breiteres Ergebnisintervall. Der Median der MLU liegt bei RLAPS höher, was auf tendenziell schlechtere Ergebnisse hindeutet. Gleichzeitig belegen die Ausreißer nach unten, dass RLAPS in einzelnen Durchläufen durchaus konkurrenzfähige Resultate liefern kann. Diese Beobachtung verdeutlicht eine zentrale Eigenschaft des Ansatzes: Die Randomisierung schafft Diversität in der Pfadwahl und kann Hotspots wirksam vermeiden, führt aber auch zu höherer Ergebnisvarianz. Während deterministische Verfahren wie Weights eine stabile, aber weniger flexible Lösung bieten, bewegt sich RLAPS zwischen sehr guten und weniger guten Resultaten. Für praktische Anwendungen bedeutet dies, dass RLAPS eher als explorativer Ansatz geeignet ist, dessen Leistungsfähigkeit durch mehrere Läufe und Mittelwertbildung abgesichert werden sollte.

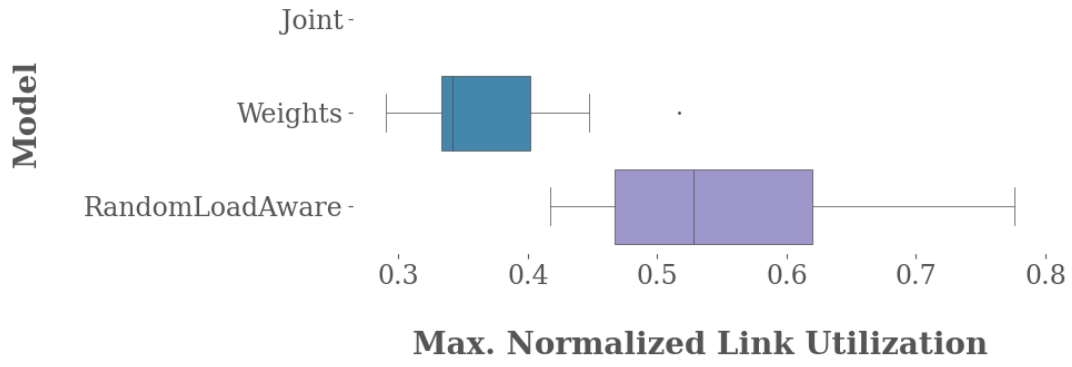


Abb. 5: Projekt 2 MLU

Kapitel 4
Apl Waypoints Algorithmus

Der AplWaypoints-Algorithmus wurde entwickelt, um eine optimale Balance zwischen der durchschnittlichen Pfadlänge (Average Path Length, APL) und der maximalen Linkauslastung (Maximum Link Utilization, MLU) durch strategische Waypoint-Auswahl zu erreichen. Das Ziel ist ein robustes Netzwerk mit verteilter Auslastung bei gleichzeitig kurzen Pfaden für geringe Latenz.

4.1 Zielfunktion und mathematische Grundlagen

Die gewichtete durchschnittliche Pfadlänge wird berechnet als:

$$\text{APL} = \frac{\sum_{(s,t,d)\in\mathcal{D}} d \cdot \text{dist}(s,t)}{\sum_{(s,t,d)\in\mathcal{D}} d}$$

wobei $\mathcal{D}$ die Menge aller Demands (s,t,d) mit Quelle s, Ziel t und Demand-Größe d darstellt, und $\text{dist}(s,t)$ die Hopanzahl des kürzesten Pfades zwischen s und t bezeichnet.

Der Algorithmus optimiert eine kombinierte Zielfunktion:

$$\text{Objective} = \lambda \cdot \text{APL} + (1-\lambda) \cdot \text{MLU}$$

wobei $\lambda \in [0,1]$ den Gewichtungsparameter darstellt. Für $\lambda = 0.5$ werden APL und MLU gleichgewichtet, für $\lambda < 0.5$ wird MLU priorisiert, und für $\lambda > 0.5$ wird APL stärker gewichtet.

4.2 Zentralitätsbasierte Waypoint-Kandidatenauswahl

Die Auswahl der Waypoint-Kandidaten basiert auf einem mehrdimensionalen Zentralitätsscore, der drei wesentliche Netzwerkmetriken kombiniert:

$$\text{Score}(v) = \alpha \cdot \text{BC}(v) + \beta \cdot \text{DC}(v) + \gamma \cdot \text{DV}(v)$$

wobei:

- $\text{BC}(v)$ die normalisierte Betweenness Centrality von Knoten v bezeichnet
- $\text{DC}(v)$ die Degree Centrality (Knotengrad) von Knoten v darstellt
- $\text{DV}(v)$ das Demand-Volumen von Knoten v repräsentiert
- $\alpha = 0.5$, $\beta = 0.3$, $\gamma = 0.15$ die empirisch bestimmten Gewichtungsparameter sind

Die Betweenness Centrality misst, wie häufig ein Knoten auf kürzesten Pfaden zwischen anderen Knotenpaaren liegt und identifiziert somit kritische Verbindungsknoten. Die Degree Centrality berücksichtigt die direkte Konnektivität eines Knotens, während das Demand-Volumen traffic-intensive Knoten ("Demand Hotspots") identifiziert.

4.3 Kandidatenauswahl und Größenadaptivität

Der Algorithmus wählt die k besten Knoten basierend auf ihrem Zentralitätsscore als Waypoint-Kandidaten aus:

$$k = \max(10, \lfloor 0.2 \cdot |V| \rfloor)$$

Diese adaptive Größenbestimmung ist bewusst gewählt: Für kleinere Topologien ($|V| < 50$) wird eine Mindestanzahl von $k = 10$ Kandidaten sichergestellt, um ausreichende Diversität zu gewährleisten. Bei größeren Topologien wird proportional 20% der Knoten als Kandidaten betrachtet, um die Rechenkomplexität zu begrenzen und gleichzeitig eine repräsentative Auswahl zu erhalten.

4.4 Demand-spezifische Waypoint-Optimierung

Der Kernalgorithmus durchläuft alle Demands in absteigender Reihenfolge ihrer Demand-Größe. Für jede Demand (s, t, d) wird systematisch evaluiert, welcher Waypoint-Kandidat v den optimalen Pfad $s \to v \to t$ erzeugt.

Der Optimierungsprozess umfasst folgende Schritte:

1. **Pfadberechnung**: Für jeden Kandidaten $v \in \mathcal{C}$ wird der kombinierte Pfad aus $\text{dist}(s, v) + \text{dist}(v, t)$ berechnet

2. **Flow-Update**: Die Flüsse auf allen Links werden entsprechend der neuen Pfadführung aktualisiert:

   - Entfernung des ursprünglichen Flusses auf dem direkten Pfad $s \to t$
   - Addition des Flusses auf dem Waypoint-Pfad $s \to v \to t$

3. **Zielfunktionsberechnung**: Für jede Waypoint-Option wird die resultierende APL und MLU berechnet und die kombinierte Zielfunktion evaluiert

4. **Beste Waypoint-Auswahl**: Der Waypoint mit der minimalen Zielfunktion wird für diese Demand ausgewählt

4.5 Globale Flow-Rekalkulation und Ergebnisvalidierung

Nach der waypoint-spezifischen Optimierung aller Demands führt der Algorithmus eine globale Neuberechnung der Flows durch. Dies gewährleistet Konsistenz, da die sequenzielle Optimierung einzelner Demands zu suboptimalen Gesamtflüssen führen kann.

Die finale Lösung umfasst:

- Eine Waypoint-Zuordnung für jede Demand in der Form $d_{idx} : [(s, v), (v, t)]$ für Waypoint-Pfade oder $d_{idx} : [(s, t)]$ für direkte Pfade
- Die resultierende MLU basierend auf den finalen Link-Auslastungen
- Die gewichtete APL unter Berücksichtigung aller gewählten Pfade
- Detaillierte Load-Informationen für jedes Link im Netzwerk

Der AplWaypoints-Algorithmus erzielt durch diese mehrstufige, zentralitätsbasierte Optimierung eine effektive Balance zwischen Pfadlängenminimierung und Lastverteilung, wodurch sowohl die Netzwerklatenz als auch die Auslastungseffizienz verbessert werden.

4.6 Experimentelle Evaluation

Zur Bewertung unseres Algorithmus testeten wir verschiedene Konfigurationen des λ-Parameters im Bereich $[0.0, 1.0]$ und analysierten den Trade-off zwischen der durchschnittlichen Pfadlänge (APL) und der maximalen Linkauslastung (MLU).

Der entwickelte Algorithmus optimiert eine kombinierte Zielfunktion der Form $(\lambda \cdot \text{APL}) + (1 - \lambda) \cdot \text{MLU}$, wobei λ die Gewichtung zwischen beiden Metriken steuert.

Abbildung 4.1 zeigt die resultierende Pareto-Front für verschiedene λ-Werte. Die Ergebnisse demonstrieren den erwarteten Zielkonflikt: Mit steigendem λ sinkt die APL, während die MLU zunimmt.

Basierend auf dieser Analyse erweist sich insbesondere $\lambda = 0.3$ als praktikabler Wert, da er eine ausgewogene Lastverteilung bei akzeptabler Pfadlänge ermöglicht. Höhere Werte wie $\lambda = 0.7$ verbessern zwar die Latenz weiter, führen jedoch zu erhöhter Auslastung auf einzelnen Links. Die konkrete Wahl des optimalen Parameters sollte daher in Abhängigkeit der jeweiligen Netzwerkanforderungen (z.B. Verzögerung vs. Überlastresistenz) erfolgen.

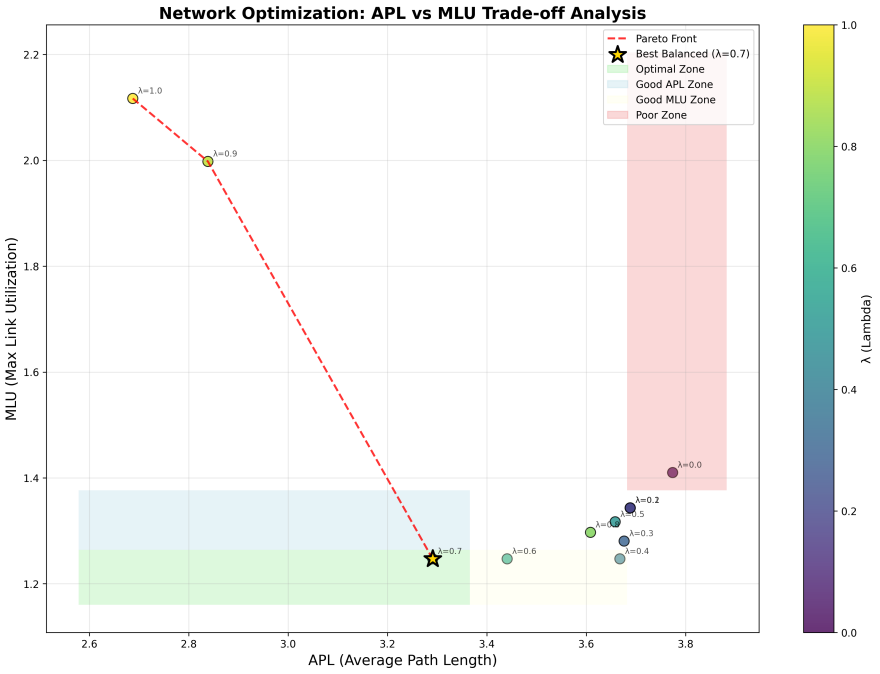


Abbildung 4.1: Pareto-Front der APL-MLU-Optimierung für verschiedene λ-Werte. Der Punkt $\lambda = 0.7$ (markiert mit Stern) stellt einen ausbalancierten Kompromiss dar.

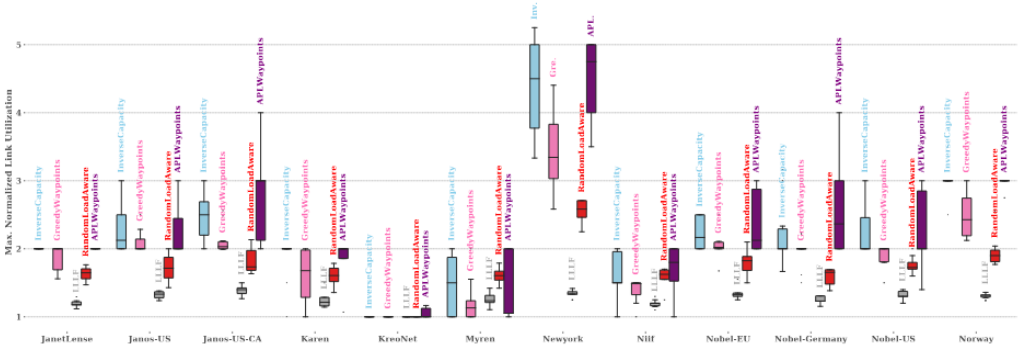


Abbildung: MLU-Value for each topologie(MCF Synthetic Demands)

Abbildung 4.2: Vergleich der Maximum Link Utilization (MLU) verschiedener Routing-Algorithmen über mehrere Netzwerktopologien

Kapitel 5
Zusammenfassung

5.1 Reproduktion

Die Reproduktion von Gruppe 4 lief bei beiden Projekten gut.

5.1.1 Projekt 1

- Projekt war ausreichend kommentiert, strukturiert und Änderungen/Additionen klar sichtbar
- Code sowie grafische Darstellung ohne Probleme ausführbar
- Algorithmische Ideen auf Basis der Zwischenpräsentation gut umgesetzt
- Ergebnisse eigener Ausführung identisch mit originalen Ergebnissen

In Abbildung 5.1 sind die Ergebnisse unserer Reproduktion. Diese waren dieselben, wie bei denen.

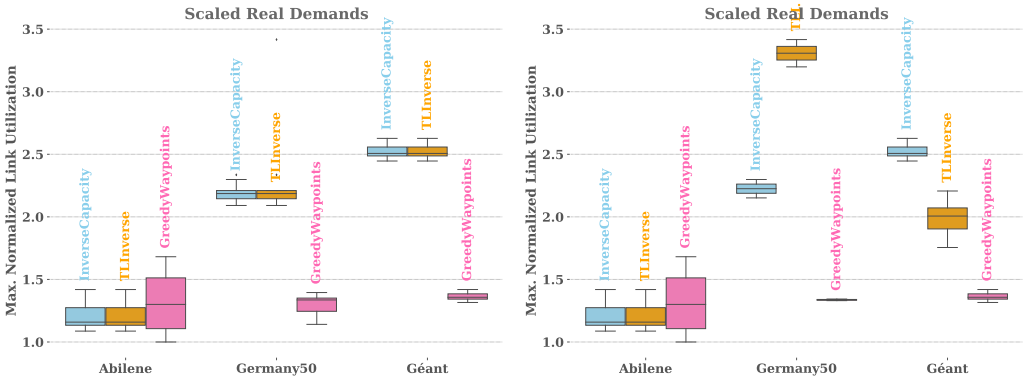
5.1.2 Projekt 2

- Der Code lief ohne Probleme durch
- Unsere Ergebnisse sind aber aus irgendeinem Grund besser
  - Vielleicht, weil mein Computer besser ist

In Abbildung 5.2 sind unsere Reproduktionsergebnisse von Projekt zwei. Dabei sieht, dass das unsere Ergebnisse dabei besser abschneiden, als die von Gruppe 4. Zwischen den Ergebnissen aus 5.2a und 5.2c sind unsere Ergebnisse dabei 2- bis 5-mal so gut. Die Ergebnisse von 5.2b und 5.2d sind auch 2-mal so gut. Wobei scheint bei TL-Inverse aber ein Fehler aufgetreten.

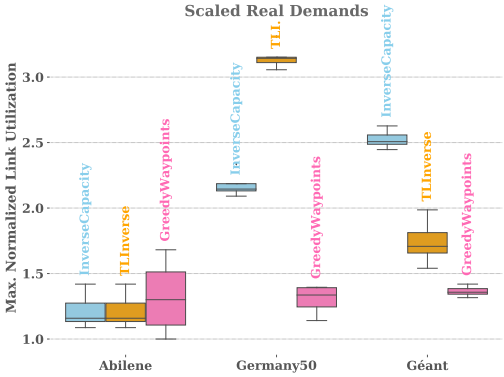
5.2 Fazit

Im Gesamten lief die Einarbeitung unserer Algorithmen gut. Es gab zwar an einigen Stellen Schwierigkeiten, sich in die Projekte einzuarbeiten und unsere Algorithmen passend zu implementieren, aber diese Schwierigkeiten konnten überwunden werden. Die Ergebnisse

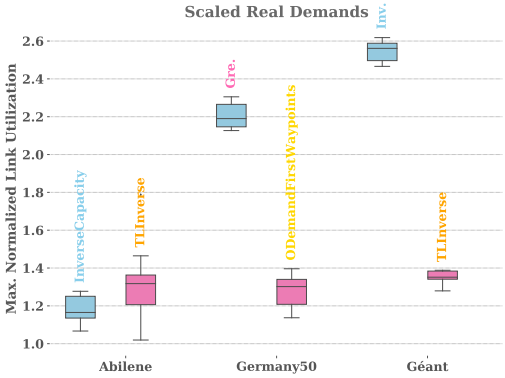


(a) Werte für $\alpha = 0.3$

(b) Werte für $\alpha = 0.5$

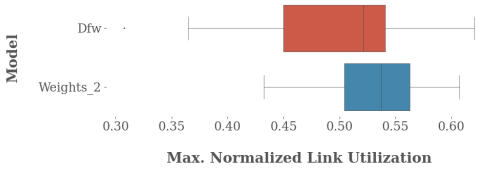


(c) Werte für $\alpha = 0.7$

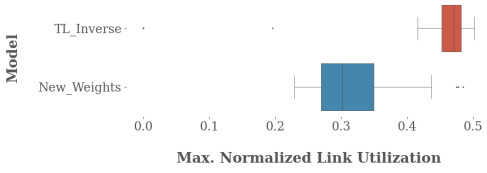


(d) Werte für $\alpha = 1$

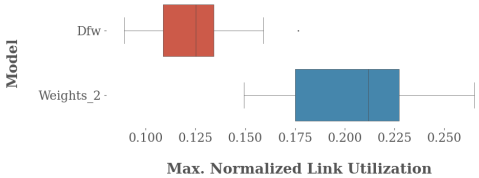
Abbildung 5.1: Reproduktion Projekt 1



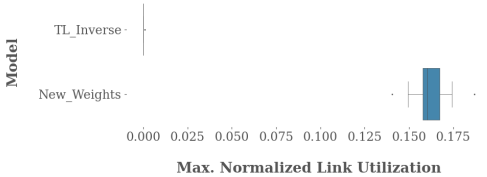
(a) Original 1



(b) Original 2



(c) Reproduktion 1



(d) Reproduktion 2

Abbildung 5.2: Reproduktion Projekt 2

der Algorithmen waren dabei gemischt. Teilweise waren sie schneller als die schon existierenden Algorithmen, teilweise aber auch langsamer. Unsere Algorithmen haben sich dabei also gut in die schon existierenden aufgereiht.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *Erweiterung von TE_SR_experiments_2021.* https://github.com/Makeandbreak09/TE_SR_experiments_2021, 2025. [Online; accessed 11-09-2025].

[2] *Erweiterung von TE_SR_WAN_simulation.* https://github.com/Makeandbreak09/TE_SR_WAN_simulation, 2025. [Online; accessed 11-09-2025].

[3] Parham, Mahmoud, Thomas Fenz, Nikolaus Süss, Klaus-Tycho Foerster und Stefan Schmid: *TE_SR_experiments_2021.* https://github.com/nikolaussuess/TE_SR_experiments_2021, 2021. [Online; accessed 11-09-2025].

[4] Parham, Mahmoud, Thomas Fenz, Nikolaus Süss, Klaus-Tycho Foerster und Stefan Schmid: *TE_SR_WAN_simulation.* https://github.com/tfenz/TE_SR_WAN_simulation, 2021. [Online; accessed 11-09-2025].

[5] Parham, Mahmoud, Thomas Fenz, Nikolaus Süss, Klaus-Tycho Foerster und Stefan Schmid: *Traffic engineering with joint link weight and segment optimization.* In: *Proceedings of the 17th international conference on emerging networking experiments and technologies*, Seiten 313–327, 2021.