

## Report

### Gruppe 3

Simon Buschmann, Yannik Buchner - Least Loaded Link First (LLLF)  
Nikita Podibko, Jan Draeger - Average Link Utilization/Random Load Aware  
Johannes Heinrich, Malek Haoues Rhaïem - 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

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Einleitung</b>   | <b>1</b>  |
| 1.1      | Motivation . . . . .  | 1         |
| 1.2      | Aufbau der Arbeit . . . . .   | 1         |
| <b>2</b> | <b>Least Loaded Link First Algorithmus</b>                          | <b>3</b>  |
| 2.1      | Grundidee . . . . .   | 3         |
| 2.2      | Eigenschaften . . . . .   | 4         |
| 2.3      | Ergebnisse . . . . .  | 5         |
| 2.4      | Mögliche Verbesserungen und Ausblick . . . . .                      | 7         |
| <b>3</b> | <b>Randomized Load Aware Path Selection Algorithmus</b>             | <b>9</b>  |
| 3.1      | Warum wurde der Algorithmus entwickelt? . . . . .                   | 9         |
| 3.2      | Warum sind randomisierte Algorithmen in Routing nützlich? . . . . . | 9         |
| 3.3      | Wie der Algorithmus funktioniert . . . . .                          | 10        |
| 3.4      | Bewertungsfunktion . . . . .  | 10        |
| 3.5      | Pseudocode . . . . .  | 10        |
| 3.6      | Komplexität . . . . .   | 11        |
| 3.7      | Vorteile und Limitationen . . . . .                                 | 11        |
| 3.8      | Mögliche Verbesserungen . . . . .                                   | 11        |
| <b>4</b> | <b>Apl Waypoints Algorithmus</b>                                    | <b>13</b> |
| 4.1      | Zielfunktion und mathematische Grundlagen . . . . .                 | 13        |
| 4.2      | Zentralitätsbasierte Waypoint-Kandidatenauswahl . . . . .           | 13        |
| 4.3      | Kandidatenauswahl und Größenadaptivität . . . . .                   | 14        |
| 4.4      | Demand-spezifische Waypoint-Optimierung . . . . .                   | 14        |
| 4.5      | Globale Flow-Rekalkulation und Ergebnisvalidierung . . . . .        | 15        |
| 4.6      | Experimentelle Evaluation . . . . .                                 | 15        |
| <b>5</b> | <b>Zusammenfassung</b>  | <b>17</b> |
| 5.1      | Reproduktion . . . . .  | 17        |
| 5.1.1    | Projekt 1 . . . . .   | 17        |
| 5.1.2    | Projekt 2 . . . . .   | 17        |
| 5.2      | Fazit . . . . .   | 17        |
|          | <b>Abbildungsverzeichnis</b>  | <b>21</b> |
|          | <b>Literaturverzeichnis</b>   | <b>23</b> |



# 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:*  $G(V, E, c : E \mapsto \mathbb{R})$ ,  $D = \{D_1 = (s_1, t_1, d_1), D_2, \dots, D_n\}$

*Ausgabe:*  $P = \{P_1, P_2, \dots, P_n\}$

```
1: for  $e \in E$  do
2:    $loads_e \leftarrow 0$ 
3: end for
4:  $P \leftarrow \{\}$ 
5: for  $i \in \{1, \dots, n\}$  do
6:    $P_i \leftarrow PotentialUtilizationDijkstra(G, loads, D_i)$ 
7:   for  $e \in P_i$  do
8:      $loads_e \leftarrow loads_e + d_i$ 
9:      $utilization_e \leftarrow loads_e / c(e)$ 
10:  end for
11:   $P \leftarrow P \cup \{P_i\}$ 
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:    $visited_v \leftarrow \text{False}$ 
5: end for
6:  $utilization_{s_i} \leftarrow 0$ 
7: while  $visited_{t_i} == \text{False}$  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:    $visited_u \leftarrow \text{True}$ 
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:

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

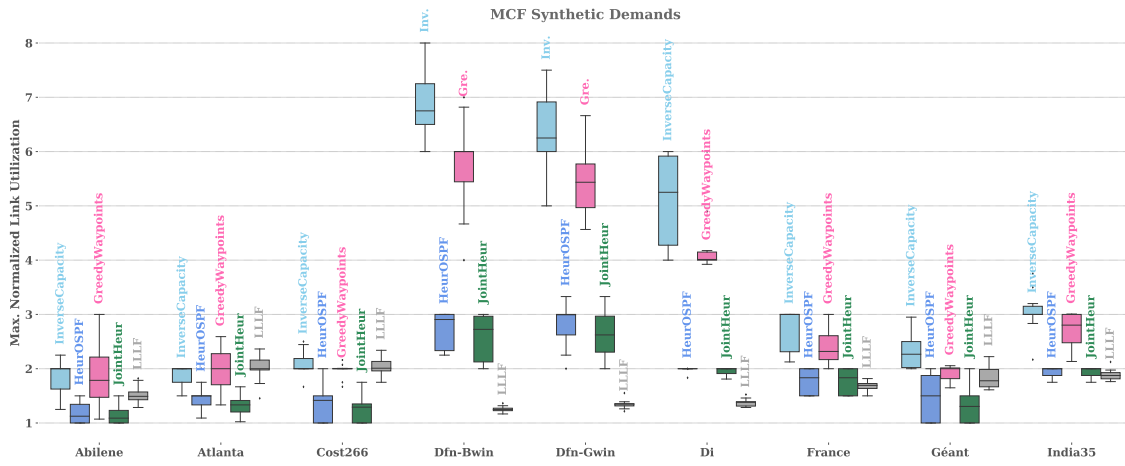
Die theoretische MLU die erreicht werden müsste wäre 0.5.

Man sieht in der Abbildung [2.2b](#), dass die theoretische MLU leider nicht erreicht wurde. Mögliche Probleme, die bei dazu geführt haben könnten, wären:

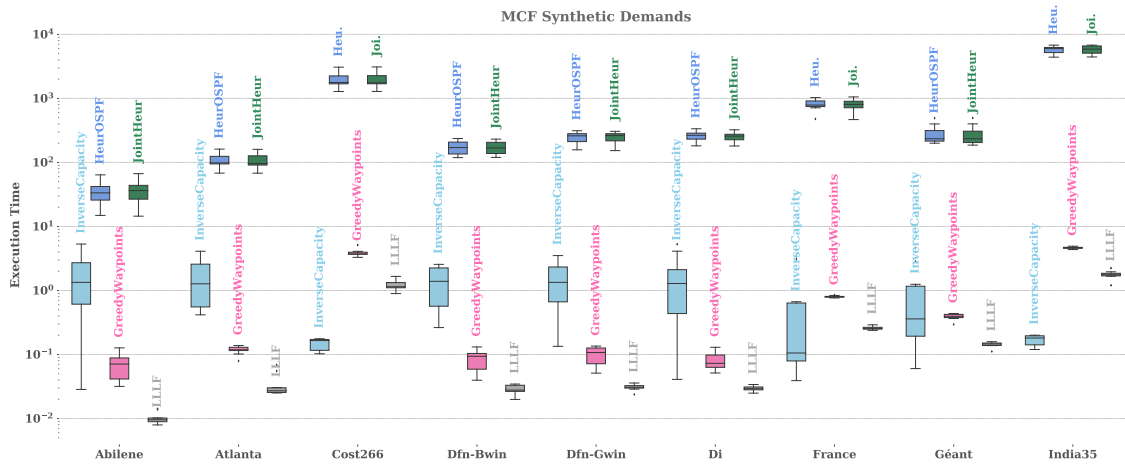
- Die ersten paar Tests waren fehlerhaft:
  - nuttcp-t: v6.1.2: Error: connect: Connection refused
  - nuttcp-r: v6.1.2: Error: bind: Address already in use
- Auch könnte es Probleme bei mehreren Demands mit selben Start und Ziel geben.
- Möglicherweise war die Topologie auch zu komplex für die Simulation.

Abbildung 2.1: Ergebnisse LLLF Projekt 1

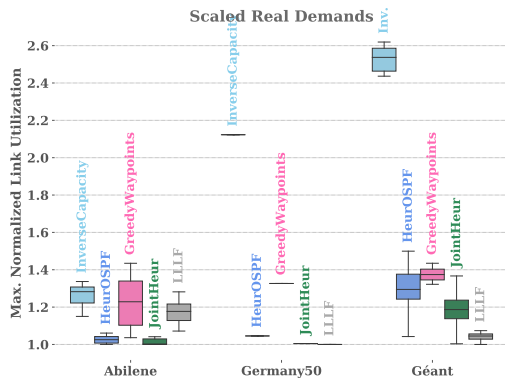
(a) Synthetische Demands MLU



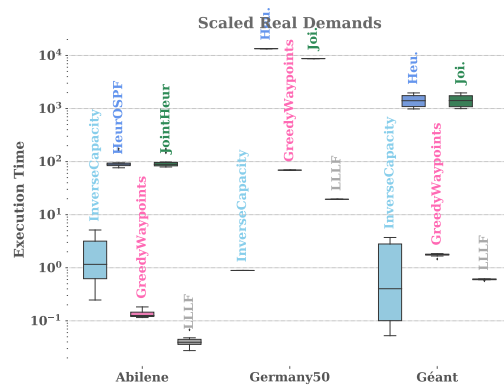
(b) Synthetische Demands Laufzeit

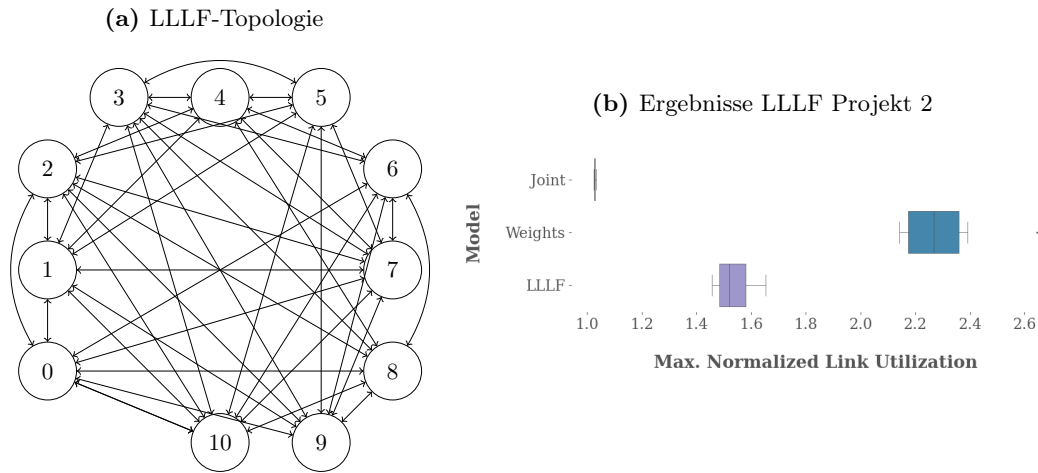


(c) Echte Demands MLU



(d) Echte Demands Laufzeit



**Abbildung 2.2:** Ergebnisse LLLF Projekt 2

## 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 `RandomizedLoadAwarePathSelection`-Algorithmus ist ein heuristischer, teilweise randomisierter Routing-Algorithmus, der entworfen wurde, um Netzwerke gegen lokale Überlastung zu schützen und gleichzeitig kurze Pfade beizubehalten. Er wählt für jede Demand mehrere Kandidatenpfade (in diesem Fall nicht alle mögliche Pfade, nur k-shortest wegen begrenzten Ressourcen), wählt daraus zufällig drei Pfade und bewertet jeden Pfad mit einer lastbasierten Funktion. Danach wird der beste der drei Pfade gewählt und als optimaler Pfad gesetzt. Dort, wo deterministische Shortest-Path-Strategien zu Hotspots führen, erzeugt die kontrollierte Randomisierung Diversität in der Pfadwahl und verteilt so den Datenverkehr im Netz.

### 3.1 Warum wurde der Algorithmus entwickelt?

Ziel war es, eine einfache, praktisch einsetzbare Methode zu entwickeln, die folgende Anforderungen erfüllt:

- **Lastverteilung:** Vermeidung von starken Auslastungsspitzen (hohe MLU) durch bewusstes Umgehen bereits stark ausgelasteter Links und durch Verteilung des Traffics im Netz zur Verbesserung der ALU.
- **Einfachheit:** Der Algorithmus ist verständlich und intuitiv implementierbar.
- **Robustheit gegenüber Eingangsvariationen:** Kleine Änderungen in den Demands oder Topologie sollten nicht sofort zu dramatischen Verschlechterungen führen.

### 3.2 Warum sind randomisierte Algorithmen in Routing nützlich?

Randomisierung ist in Routing-Kontexten aus mehreren Gründen vorteilhaft:

- **Diversität statt Konvergenz auf Hotspots:** Deterministische Algorithmen wählen häufig dieselben minimalen Pfade für viele Demand-Paare. Kleine Zufallsentscheidungen vermeiden Hotspoterstellung und sorgen für bessere Verteilung von Traffic durch das Netz.

- **Reale Lastprognose:** In realen Netzen liegen oft unsichere oder variable Lasten zugrunde; die randomisierte Auswahl modelliert diese Unsicherheit besser.
- **Einfachheit der Implementierung:** Randomisierte Auswahl kann oft leicht zu bestehenden heuristischen Verfahren hinzugefügt werden, ohne auf komplexe globale Optimierer angewiesen zu sein.

### 3.3 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.

### 3.4 Bewertungsfunktion

Sei  $P$  ein Pfad bestehend aus Kanten  $(e_1, \dots, e_m)$  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.

### 3.5 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, ...)
```

### 3.6 Komplexität

Sei  $D$  die Anzahl der Demands,  $K$  die Anzahl der ausgewählten Kandidaten pro Demand,  $L$  die durchschnittliche Pfadlänge und  $T_{ksp}$  die Kosten, um  $k$ -kürzeste Pfade zu berechnen. Dann ist die Laufzeit ungefähr:

$$O(D \cdot (T_{ksp} + K \cdot L))$$

### 3.7 Vorteile und Limitationen

#### Vorteile

- **Einfache Realisierbarkeit:** Es sind nur lokale Lastinformationen nötig.
- **Verbesserte Lastverteilung:** gezielte Vermeidung bereits ausgelasteter Links reduziert MLU in vielen Szenarien. Die Methode verbessert die ALU durch randomisierte Pfadverteilung.
- **Parametersteuerung:**  $K$ , Oversample und die Gewichtung erlauben eine Steuerung zwischen Exploration und Exploitation.

#### Limitationen

- **Kein globales Optimum:** Algorithmische Heuristik - keine Garantie auf optimale MLU/ALU.
- **Reihenfolgensensitiv:** die Reihenfolge, in der Demands abgearbeitet werden, beeinflusst das Ergebnis (Greedy-Verhalten).
- **Determinismus/Statistik:** Randomisierte Entscheidungen erfordern statistische Auswertung (mehrere Durchläufe zur Mittelwertermittlung).
- **Skalierung bei sehr großen Netzen:**  $k$ -shortest-Berechnung teuer.

### 3.8 Mögliche Verbesserungen

1. **Probabilistische Wahl statt argmin:** Statt strikt das Minimum zu wählen, könnte ein Softmax-/Boltzmann-Verfahren verwendet werden, das schlechtere Pfade mit niedriger Wahrscheinlichkeit auswählt - es erhöht die Exploration und reduziert die Varianz.
2. **Adaptive  $K$  und Oversample:** Wähle  $K$  abhängig vom Knotengrad, von der Nachfragegröße, von der Menge verfügbarer Ressourcen oder von der aktuellen Netzbelastung.

3. **Reihenfolge-Randomisierung/Priorisierung:** Zuerst große Demands oder latenzkritische Flows behandeln; alternativ die Reihenfolge pro Durchlauf randomisieren und die beste Lösung über mehrere Durchläufe wählen.
4. **Berücksichtigung von Latenz- oder Kosten-Tradeoffs:** Kombiniere den Last-Score mit der Pfadlänge in einer gewichteten Zielfunktion.



# 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 Gewichtungssparameter 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 Gewichtungssparameter 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  $s \rightarrow v \rightarrow t$  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  $s \rightarrow t$
  - Addition des Flusses auf dem Waypoint-Pfad  $s \rightarrow v \rightarrow t$
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.

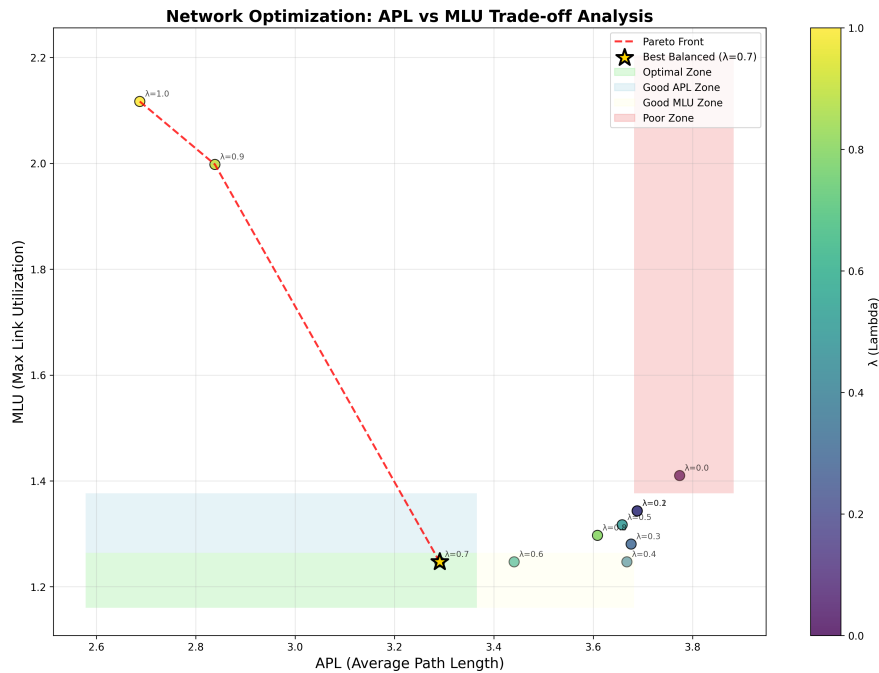
## 4.6 Experimentelle Evaluation

Zur Bewertung unseres Algorithmus testeten wir verschiedene Konfigurationen des  $\lambda$ -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  $\lambda$  die Gewichtung zwischen beiden Metriken steuert.

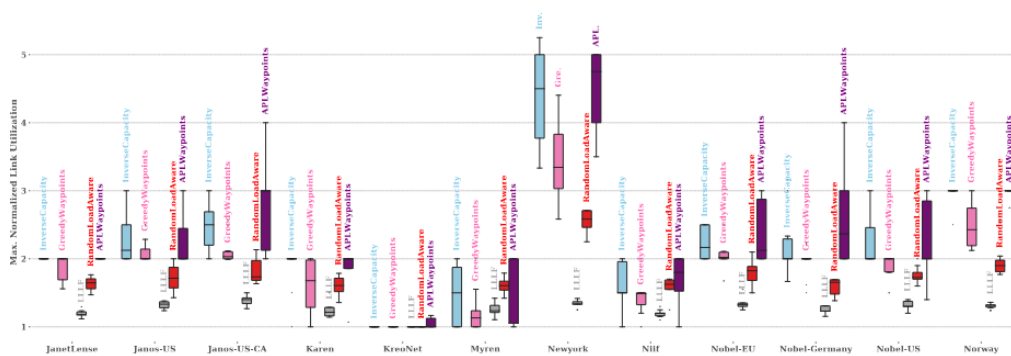
Abbildung 4.1 zeigt die resultierende Pareto-Front für verschiedene  $\lambda$ -Werte. Die Ergebnisse demonstrieren den erwarteten Zielkonflikt: Mit steigendem  $\lambda$  sinkt die APL, während die MLU zunimmt.

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.



Simulation executed on the Germany50 network using randomly generated traffic demands.

**Abbildung 4.1:** Pareto-Front der APL-MLU-Optimierung für verschiedene  $\lambda$ -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

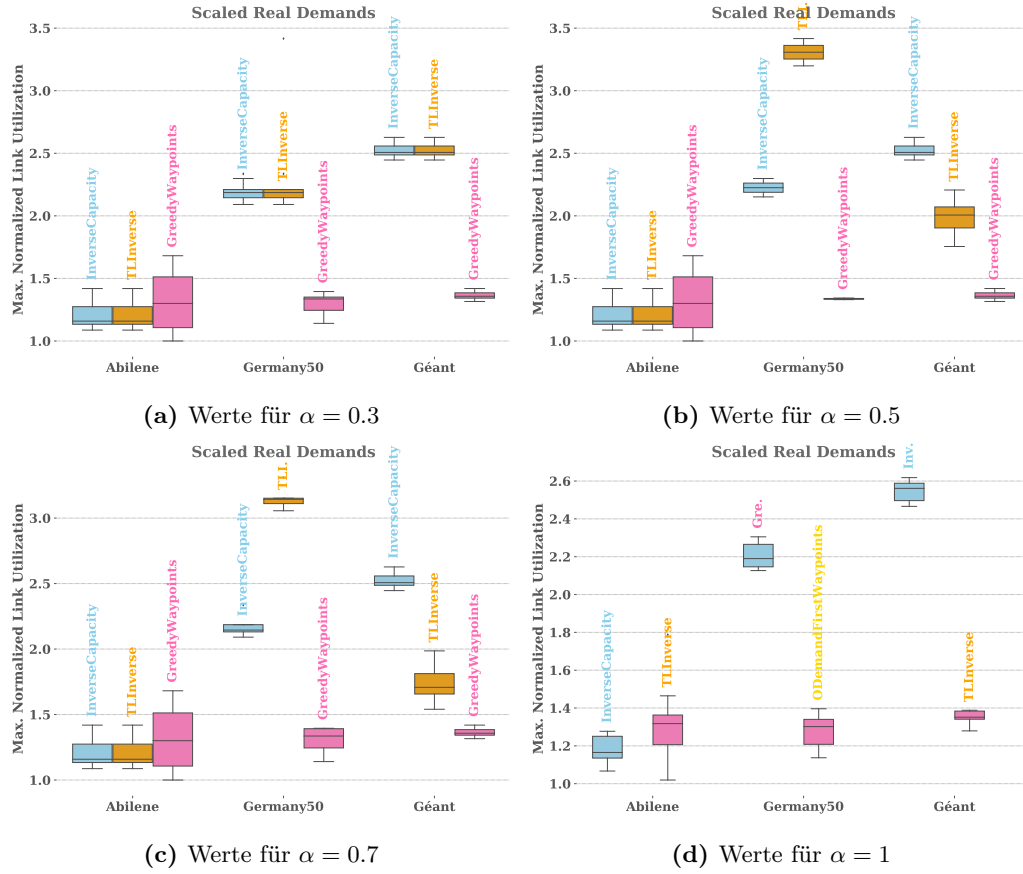


Abbildung 5.1: Reproduktion Projekt 1

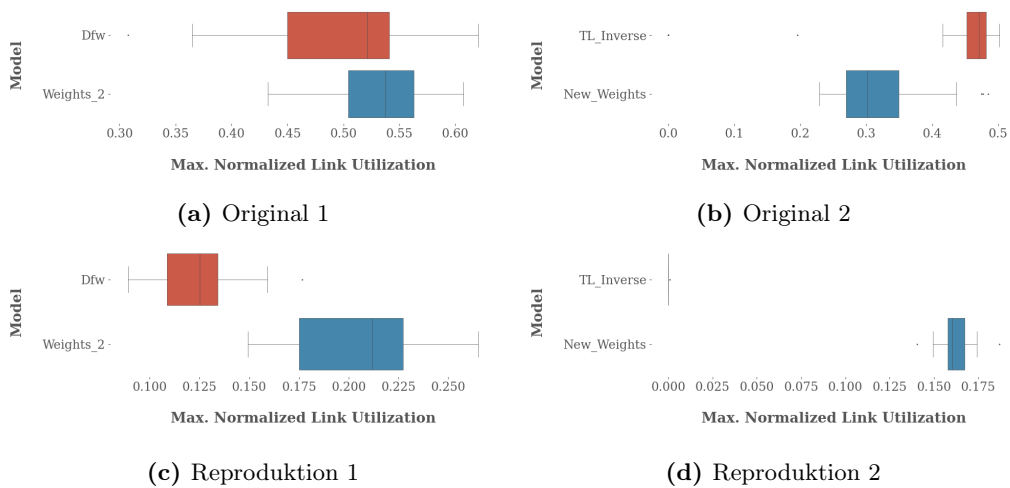


Abbildung 5.2: Reproduktion Projekt 2

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

|     |   |    |
|-----|---|----|
| 2.1 | Ergebnisse LLLF Projekt 1 . . . . .   | 6  |
| 2.2 | Ergebnisse LLLF Projekt 2 . . . . .   | 7  |
| 4.1 | Pareto-Front der APL-MLU-Optimierung für verschiedene $\lambda$ -Werte. Der Punkt $\lambda = 0.7$ (markiert mit Stern) stellt einen ausbalancierten Kompromiss dar. . . . . | 16 |
| 4.2 | Vergleich der Maximum Link Utilization (MLU) verschiedener Routing-Algorithmen über mehrere Netzwerktopologien . . . . .  | 16 |
| 5.1 | Reproduktion Projekt 1 . . . . .  | 18 |
| 5.2 | Reproduktion Projekt 2 . . . . .  | 18 |



# Literaturverzeichnis

- [1] *Erweiterung von TE\_SR\_experiments\_2021.* [https://github.com/Makeandbreak09/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](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](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](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.