

Report zum Fachprojekt Routingalgorithmen

Pouria Araghchi
TU Dortmund
Dortmund, Deutschland
pouria.araghchi@tu-dortmund.de

Kai Lukas Ilmenau
TU Dortmund
Dortmund, Deutschland
kai.ilmenau@tu-dortmund.de

Naveed Niazi
TU Dortmund
Dortmund, Deutschland
naveed.niazi@tu-dortmund.de

ABSTRACT

In der heutigen Zeit werden mehrere Millionen Datenpakete von und zu Knotenpunkten verschickt. Daher ist das *Routing* von diesen Datenpaketen in der heutigen Zeit wichtiger denn je. Gutes Routing kann Datenstaus verringern oder sogar verhindern, Verteilerknoten auslasten oder entlasten und sorgt damit fuer einen reibungslosen Datenverkehr ueberall auf der Welt. Wie gut das Routing innerhalb einer Netzwerktopologie ist, wird durch den darunterliegenden Routingalgorithmus bestimmt. Hierfür gibt es verschiedenste Ansätze mit variierender Komplexität und Erfolgswahrscheinlichkeit.

CCS CONCEPTS

• Networks → Traffic engineering algorithms.

KEYWORDS

weight optimization, waypoint optimization

1 EINLEITUNG

fehlt

2 THEORETISCHE GRUNDLAGEN

Hier wird etwas zum Paper stehen[1].

3 UMSETZUNG

In diesem Abschnitt werden unsere Projekte und die dazugehörigen Ideen vorgestellt. Die beiden Projekte sind in Unterabschnitte und die algorithmischen Ideen und Umsetzungen des jeweiligen Gruppenmitglieds in Paragraphen unterteilt.

3.1 Algorithmen zu Projekt 1

In Projekt 1 ging es darum, eine algorithmische Idee auf Basis von [1] herauszuarbeiten und diese mithilfe desselben Git-Projekts¹ mit alternativen Routingalgorithmen zu vergleichen.

3.1.1 Sequential combination aus inverse capacity und demand first waypoints. Der Kerngedanke hier war es, einen schnellen Algorithmus zu finden, welcher hinreichend gute Ergebnisse in Bezug auf die MLU abliefern. Diese Eigenschaft ist insbesondere für Netzwerke sinnvoll, dessen Auslastungen sich regelmäßig während des Betriebs ändern und der Algorithmus reaktiv sehr schnell alle Gewichte und Wegpunkte neu berechnet.

Die Grundlage dieses Algorithmus war die sequentielle Kombination von dem empirisch häufig verwendeten *inverse_capacity* und

¹https://github.com/fruitiesPunch/FaPro_P1, siehe Abschnitt A

Authors' addresses: Pouria Araghchi, TU Dortmund, Otto-Hahn-Straße 14, Dortmund, Deutschland, pouria.araghchi@tu-dortmund.de; Kai Lukas Ilmenau, TU Dortmund, Otto-Hahn-Straße 14, Dortmund, Deutschland, kai.ilmenau@tu-dortmund.de; Naveed Niazi, TU Dortmund, Otto-Hahn-Straße 14, Dortmund, Deutschland, naveed.niazi@tu-dortmund.de.

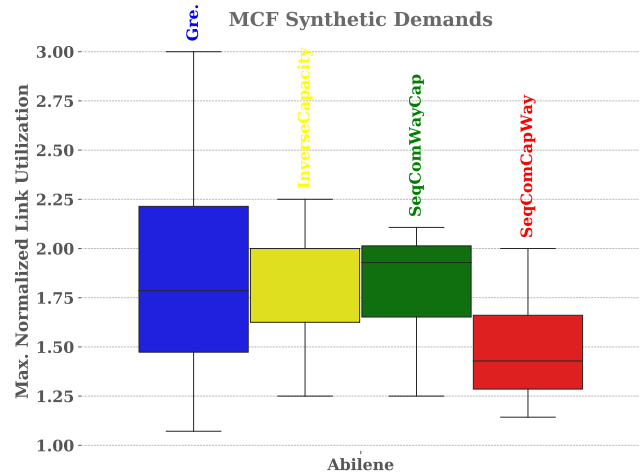


Figure 1: Vergleich von vier Algorithmen mit syntetischen Anforderungen auf der Abilene-Topologie. Legende: greedy waypoints in blau, inverse capacity in gelb, SeqComWayCap in grün, SeqComCapWay in rot.

dem etwas langsameren aber genaueren *demand_first_waypoints*. Anmerkung: Die Algorithmen Namen "demand first waypoints" und "greedy waypoints" sind hier austauschbar.

Daraufhin bildeten sich zwei Fragen.

- (1) Ist die Kombination genauer als dessen Bestandteile?
- (2) Ist die zusätzliche Rechenzeit gerechtfertigt?

Die Abbildungen 1 und 3 geben eine Antwort auf Frage 1. Sie zeigen, dass die sequentielle Kombination beider Algorithmen in beiden Fällen mindestens genauso gut ist wie eines seiner Bestandteile. Im Fall von *SeqComCapWay* ist dieser im Durchschnitt sogar besser als beide Algorithmen. Dies lässt sich darauf zurückführen, dass die Kantengewichte durch den Algorithmus *inverse_capacity* zu Beginn verbessert werden und anschließend durch den zweiten Algorithmus weiter verbessert werden, falls möglich. Im Folgenden wird daher nur noch auf die sequentielle Kombination *SeqComCapWay* (in rot) eingegangen.

Die Abbildungen 2 und 4 hingegen sind relevant für die Frage 2. Diese zeigen einen etwas genaueren Blick auf die Rechenergebnisse geplottet über die dafür benötigte Zeit. Hier kann auch direkt gesehen werden, dass *inverse_capacity* mit Abstand der schnellste hier verwendete Algorithmus ist. Dies ist besonders vorteilhaft, da es bedeutet, dass die sequentielle Kombination aus *inverse_capacity* und *greedy_waypoints* kaum länger mehr Berechnungszeit benötigt als *greedy_waypoints* alleine. Somit konnte empirisch gezeigt werden, dass in diesem "vereinfachten" Beispiel die Kombination der oben genannten Algorithmen nicht nur im Durchschnitt bessere

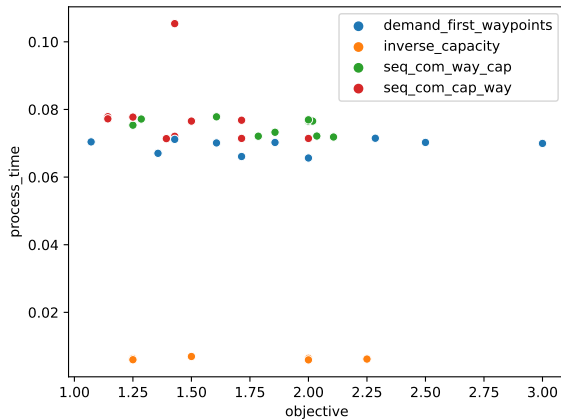


Figure 2: Vergleich von vier Algorithmen mit syntetischen Anforderungen auf der Abilene-Topologie. Legende: greedy waypoints in blau, inverse capacity in gelb, SeqComWayCap in grün, SeqComCapWay in rot.

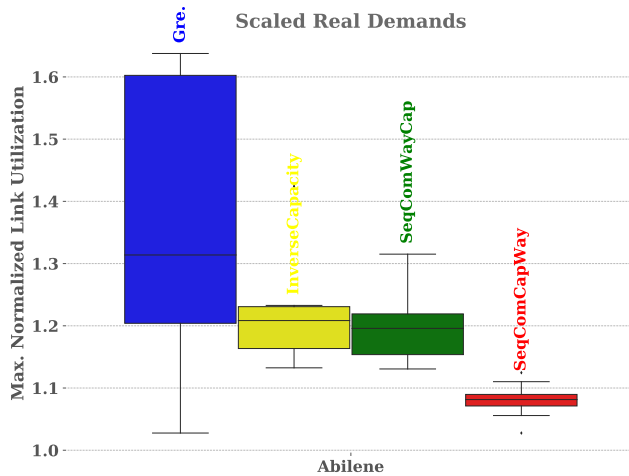


Figure 3: Vergleich von vier Algorithmen mit realen Anforderungen auf der Abilene-Topologie. Legende: greedy waypoints in blau, inverse capacity in gelb, SeqComWayCap in grün, SeqComCapWay in rot.

MLU-Ergebnisse liefert, sondern das auch in derselben Zeit erledigt. Und somit lässt sich die Frage der Rechtfertigung auslagern. In allen Fällen, in denen der *greedy_waypoints*-Algorithmus gerechtfertigt ist, ist auch die oben genannte Kombination gerechtfertigt. Da es aus Zeitgründen sowie mangelnder Rechenressourcen nicht so einfach möglich war, diese Algorithmen auf sehr großen Topologien zu testen, kann nicht mit Sicherheit gesagt, wie hoch die Berechnungszeit in Topologien mit 50 oder mehr Knoten wirklich ist. Daher ist es schwer, diese Algorithmen (mit Ausnahme von *inverse_capacity* in solchen Topologien dynamisch zu verwenden.

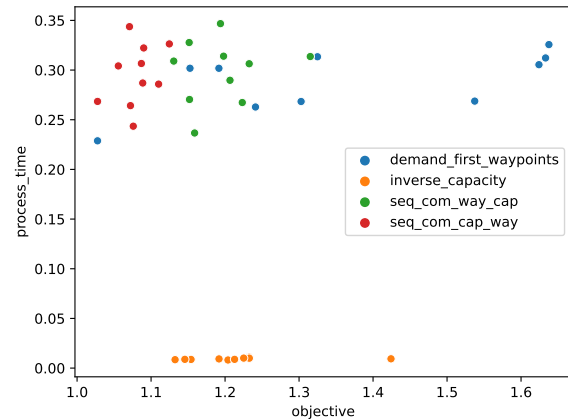


Figure 4: Vergleich von vier Algorithmen mit syntetischen Anforderungen auf der Abilene-Topologie. Legende: greedy waypoints in blau, inverse capacity in gelb, SeqComWayCap in grün, SeqComCapWay in rot.

3.2 Experimente zu Projekt 2

In diesem Projekt ging es darum, die eigene algorithmische Idee zu nehmen und in einem virtuellen Netzwerk² zu testen. Dieses Projekt wurde ebenfalls mithilfe eines bereits existierenden Repositories³ bearbeitet.

3.2.1 Sequential combination aus inverse capacity und demand first waypoints. Zum Testen dieses Algorithmus wurde eine vereinfachte Topologie mit wenigen Anforderungen erstellt. In Abbildung 6 ist die finale Topologie zu sehen, wobei hier bereits der *inverse_capacity*-Algorithmus darauf ausgeführt wurde. In der Tabelle 1 sind die beiden Anforderungen sowie deren Start- und Endknoten angegeben. Die Timeouts in diesen Experimenten haben zu einigen Problemen geführt. Da der Rechner auf dem speziell dieser Algorithmus getestet wurde, relativ schwach war (4GB RAM), wird vermutet, dass die Timeouts^{4 5}

at now+2min

und

sleep(8 * 60)

nicht ausgereicht haben, sodass einige der Prozesse frühzeitig abgebrochen wurden. Abbildung 5 zeigt dabei die MLU-Werte aller drei Algorithmen im Vergleich. Die Tatsache, dass die Boxplots hier nur als Striche dargestellt werden, weist daraufhin, dass es keine Streuung in den finalen Ergebnissen gab, was auf das frühzeitige Abbrechen einiger Rechenprozesse zurückgeführt werden kann.

4 REPLIKATION

Im Folgenden werden die Ergebnisse und Einsichten der Replikationen von Gruppe 2 vorgestellt.

²<https://github.com/nikolaussuess/nanonet>, siehe Abschnitt A

³https://github.com/fruitiesPunch/FaPro_P2, siehe Abschnitt A

⁴<https://github.com/nikolaussuess/nanonet>, siehe Abschnitt A

⁵*nanonet_batch.py* aus https://github.com/fruitiesPunch/FaPro_P2, siehe Abschnitt A

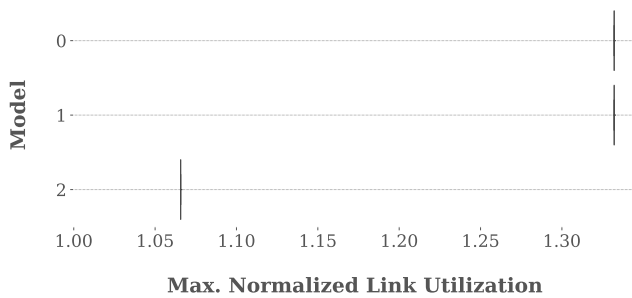


Figure 5: Vergleich von drei Algorithmen als Boxplotdiagramme. Legende: 0 = Joint, 1 = Weights, 2 = Pouria.

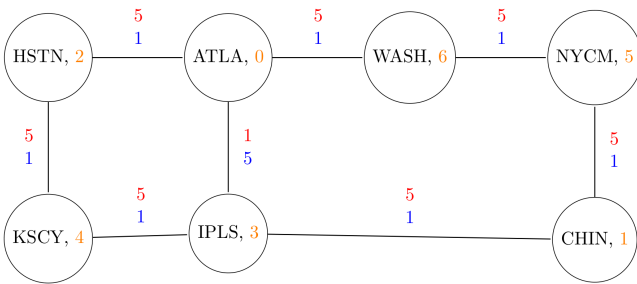


Figure 6: Vereinfachte Netzwerktopologie [Anlehnung an Abilene]. Legende: Kapazitäten in rot, Gewichte in blau.

Table 1: Vereinfachte Anforderungstabelle. Vollständige Tabelle unter https://github.com/fruitiestPunch/FaPro_P2/tree/master/pouria

↓ von, nach →	IPLS	WASH	...
ATLA	5		
IPLS		7	
⋮			

4.1 Replikation zu Projekt 1

Die Replikation von diesem Projekt war zu Beginn leider etwas schwierig, weil in beiden Fällen Abhängigkeiten fehlten. Ein großes Problem beim Replizieren des Deep-Learning-Projekts war, dass sich der Prozess nach einer Weile selbstständig beendet hat. Die Vermutung liegt nahe, dass der Rechenaufwand so hoch war, dass der Rechner ab einem bestimmten Zeitpunkt den Prozess selbst beendet hat. Ein weiteres Problem beim Replizieren waren die Fehlermeldungen und Abbrüche beim Plotten. Diesem und ähnlichen Fehlern sind wir bei unserer Bearbeitung ebenfalls begegnet und die Behebung hat sich als außerordentlich schwierig erwiesen, weil dieser "Plottererror" innerhalb einer Python-Bibliothek liegt. Abbildung 7 zeigt einen solchen Fehler, welcher sich hartnäckig auch nach einigen Anpassungsversuchen weiterhin erhalten hat.

```

python .\src\plot_results.py .\out\
MCF Synthetic Demands - all_algorithms
Mean objective over all topologies:
mean = np.mean(df_x["objective"].values.mean())
ret = ret.dtype.type(ret / rcount)
nan: nan
ZeroDivisionError: division by zero

```

Figure 7: Plotting-Fehlermeldung im Terminal

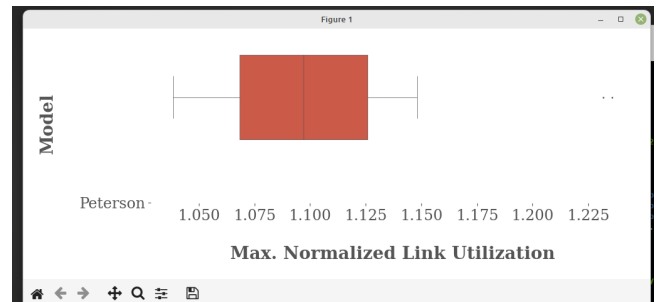


Figure 8: Ergebnisdiagramm des Codes von Gruppe 2

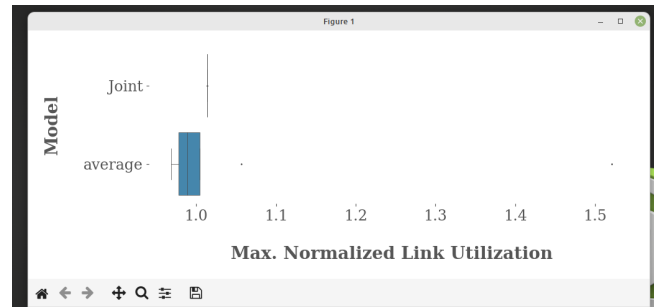


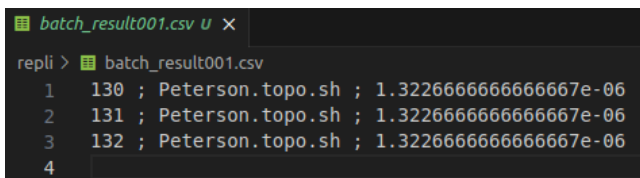
Figure 9: Ergebnisdiagramm des Codes von Gruppe 2

4.2 Replikation zu Projekt 2

Die Replikation des Codes aus Gruppe 2 lief im Allgemeinen problemlos, wie die Abbildungen 8 und 9 zeigen. Ein Problem, welches auch innerhalb unserer Gruppe aufgetaucht ist, waren die identischen Werte nach Beendigung der `nanonet_batch.py`-Datei. Wie bereits in Abschnitt 3.2.1 erwähnt, liegt dieses Problem vermutlich an mangelnder Rechenleistung während des Berechnungsprozesses. Nach wiederholtem Ausführen des Codes konnte dieses Problem jedoch behoben werden.

5 ZUSAMMENFASSUNG

Es kann gesagt werden, dass wir während des Fachprojekts vieles lernen konnten, insbesondere Dinge, die nicht notwendigerweise im Stundenplan stehen, aber gerade auf der akademischen Laufbahn als Standard angesehen werden. Das Nachstellen einer wissenschaftlichen Arbeit hat sich als nicht-trivial erwiesen und das



```

349  batch_result001.csv u x
350  repli > batch_result001.csv
351  1 130 ; Peterson.topo.sh ; 1.3226666666666667e-06
352  2 131 ; Peterson.topo.sh ; 1.3226666666666667e-06
353  3 132 ; Peterson.topo.sh ; 1.3226666666666667e-06
354  4

```

Figure 10: Ergebnis-CSV-Datei mit identischen MLU-Werten in jeder Zeile

obwohl diese Arbeit für ihre hohe Reproduzierbarkeit ausgezeichnet wurde. Dies lässt leider nur auf eine Folgerung schließen; viele andere wissenschaftlichen Arbeiten lassen sich viel schwerer oder gar nicht nachstellen. Während des Replizierens der Projekte der anderen Gruppen war es ebenfalls überraschend, das selbst wenn unsere Gruppen alle an demselben Hauptprojekt gearbeitet haben und theoretisch alle dieselben Pakete verwendeten, es manchmal dennoch zu Replikationsschwierigkeiten kam. Allerdings war es auch eine schöne Erfahrung in einer Gruppe an so einem Projekt zu sitzen und die Probleme intern sowie mit den anderen Gruppen besprechen zu können. Das hat nicht nur zu einem größeren Zusammengehörigkeitsgefühl geführt, sondern wir haben auch gelernt, kollaborativ gemeinsame Schwierigkeiten zu lösen.

6 AUSBLICK

Aufgrund der relativ kurzen Bearbeitungszeit und den begrenzten Rechenkapazitäten unserer eigenen Computer war es etwas schwer, unsere Algorithmen, insbesondere in Projekt 2, auf großen Topologien zu testen. Dies kann dazu führen, dass bestimmte Effekte wie

Datenstaus, welche gerade in großen Topologien auftauchen, in kleinen kaum oder gar nicht vorkommen.

ACKNOWLEDGMENTS

Vielen Dank an Marvin für seine Hilfe und Geduld mit unseren Problemen. Ohne seine Hilfe wäre das alles in der kurzen Zeit sehr viel schwieriger gewesen.

REFERENCES

- [1] Mahmoud Parham, Thomas Fenz, Nikolaus Süß, Klaus-Tycho Foerster, and Stefan Schmid. 2021. Traffic Engineering with Joint Link Weight and Segment Optimization. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies* (Virtual Event, Germany) (CoNEXT '21). Association for Computing Machinery, New York, NY, USA, 313–327. <https://doi.org/10.1145/3485983.3494846>

A ONLINERESSOURCEN

Im Rahmen dieses Fachprojekts wurden drei öffentliche Repositories als Hauptquellen verwendet. Die ersten beiden Repositories https://github.com/fruitiesPunch/FaPro_P1 und https://github.com/fruitiesPunch/FaPro_P2, welche *forks* von https://github.com/tfenz/TE_SR_WAN_simulation und https://github.com/nikolaussuess/TE_SR_experiments_2021 sind. Die dritte Hauptquelle stellt <https://github.com/nikolaussuess/nanonet> dar, welche zur Erstellung von vom Projekt les- und interpretierbaren Netzwerktopologiedaten verwendet wurde.

Received 6. August 2023; überarbeitet **ausstehend**; akzeptiert 15. August 2023