NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

IDATT2104 - Nettverksprogrammering

Telekino – Optimalering av mobile mesh nettverk ved bruk av BGP

Av :
Arunan Gnanasekaran og Victor Sebastian I K Bugge

Mai, 2023

Forord

Telekino er en fjerntyringsmaskin oppfunnet av den spanske ingeniøren Leonardo Torres y Quevedo, en pionér innen fjernstyringsteknologi og automasjon. Quevedo er også kjent for å ha utviklet den første automatiske sjakkroboten, som betraktes som det første dataspillet i historien. Vi har hentet inspirasjon fra dette og oppkalt optimeringsalgoritmen presentert i rapporten etter Telekino.

Produktet er primært utviklet som en test, basert på inspirasjon fra Sebastian Lague og hans YouTube-video *"Coding Adventure: Ant and Slime Simulations"*. Vi er takknemlige for den betydelige inspirasjonen vi fikk gjennom denne videoen.

Basert på inspirasjon fra Sebastian Lague, planla vi opprinnelig å utvikle en Unity-simulering. Dessverre, på grunn av tidsbegrensninger, måtte vi redusere ambisjonene og lage en enklere simulering. Utviklingen av algoritmen tok lengre tid enn forventet, noe som førte til begrensninger i simuleringen. Vi ønsker likevel å takke for den fantastiske oppgaven og er vel fornøyd med det vi har fått til.

Table of Contents

List of Figures

1 Introduksjon

Optimal plassering av aksesspunkter er et viktig aspekt innenfor nettverksplanlegging. Mange av dagens løsninger for å oppnå dette er *sentraliserte*, hvor man bruker posisjon og en sentral enhet for å kartlegge den beste dekningen. For eksempel anvender man applikasjoner som utfører bygningskartlegging og identifiserer optimale posisjoner for aksesspunkter når man planlegger signalstyrken i et bygg (Research 2023). For nettverk med mobile aksesspunkter, har man også fordelen av å kunne optimere ytterligere ved å flytte trådløse noder til områder med høyere trafikkbelastning etter plassering.

Imidlertid oppstår det utfordringer når man forsøker å bruke posisjon innenfor et mobilt nettverk, da man ikke alltid har pålitelig tilgang til denne informasjonen. Selv om GPS kan brukes for å finne posisjonen, oppstår det situasjoner der dette blir problematisk, for eksempel når man befinner seg under bakken eller har tykke betongvegger som forstyrrer signalet.

En alternativ tilnærming er å benytte relativ posisjonering ved hjelp av *triangulering*. I stedet for å være avhenge av GPS-signaler, kan man triangulere posisjonen ved hjelp av de mobile nodene i nettverket. Imidlertid kan nøyaktighetsproblemer oppstå. Det kreves også tilstrekkelig antall signaler for at nodene skal kunne triangulere sin egen posisjon, og et begrenset antall noder kan føre til unøyaktige resultater. I tillegg, hvis en node mister all kontakt, blir det umulig å bestemme posisjonen.

De nevnte eksemplene gir innsikt i problemene som kan oppstå med en sentralisert tilnærming. En *desentralisert* løsning kan derimot muliggjøre optimal dekning og hastighet uten bruk av posisjonsdata. I en desentralisert tilnærming opererer hver node individuelt, basert på egne forutsetninger og uten nødvendigheten av å handle som en helhet.

En desentralisert tilnærming åpner dermed opp for en bred rekke muligheter. Ved å frigjøre seg fra behovet for detaljert kartlegging og omfattende planlegging, blir det mulig å skape et optimalt nettverk på en enklere og mer skalerbar måte. Dette fører til økt effektivitet og redusert kompleksitet i implementeringen av mobile nettverksløsninger.

I denne rapporten presenteres det en slik desentralisert optimeringsalgoritme som bygger på prinsipper innen nettverksruting og diskret matematikk. Løsningen baserer seg kun på bruk av nettverksignaler en node har til andre noder, uten bruk av posisjon. Løsningen er inspirert av BGP-protokollen og bygger på den distribuerte Bellman-Ford algoritmen. Dette er metoder og algoritmer som er utviklet med mål om effektivitet og pålitelighet i et distribuert miljø.

Rapporten analyserer de teoretiske fundamentene som underbygger denne løsningen og presenterer praktiske implementeringsaspekter, samt resultater fra simuleringer og eksperimenter. Gjennom en utforskning av denne distribuerte tilnærmingen, blir potensialet for å oppnå en optimal og skalerbar nettverksløsning uten behovet en sentralisert enhet demonstrert.

2 Teori

2.1 Ruter og ruting

En nettverksruter brukes for å koble sammen nettverk og lede datatrafikk i mellom nettverkene. Den fungerer som en digital trafikkdirigent som bestemmer den optimale ruten for å sende data fra en kilde til en destinasjon (Wikipedia 2023).

Ruting er prosessen med å bestemme hvilken rute dataene skal følge i et datanettverk. Når data blir sendt fra en enhet til en annen, brytes de ned i mindre pakker. Disse pakkene inneholder informasjon som kildeadresse, destinasjonsadresse og selve dataene (Wikipedia 2023).

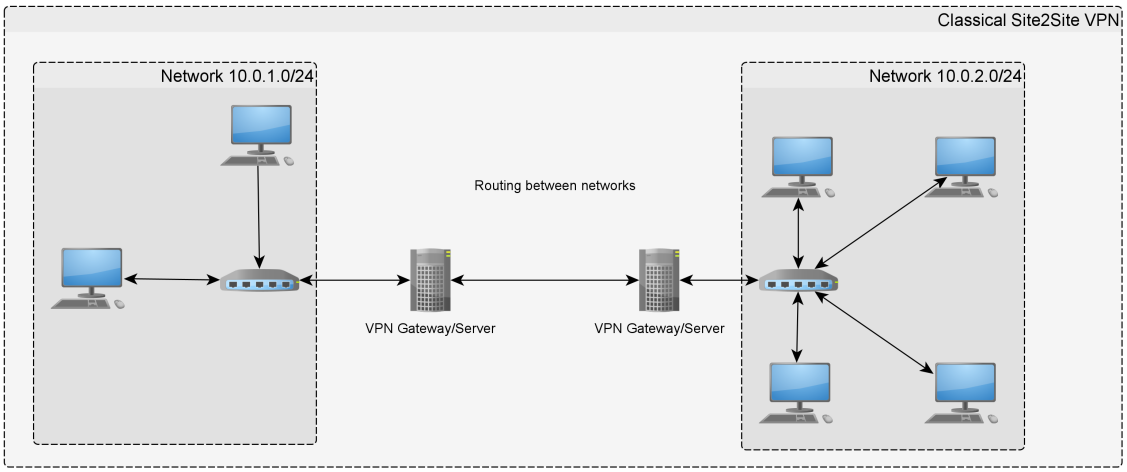

Figure 1: Ruting mellom to nettverk

2.2 Border Gateway Protocol

Border Gateway Protocol tabell, også kjent som BGP-rutingtabell, er en datastruktur som inneholder informasjon om tilgjengelige ruter for en ruter. BGP gjør det mulig for rutere å utveklse informasjon om ruter. Dette gjør det mulig for en ruter å velge beste rute basert på faktorer som lengde og kostnad (Cloudflare 2023).

BGP er fullstendig desentralisert, der hver ruter må finne den beste rutingen på egenhånd. Det er ingen sentralisert enhet som styrer hvordan pakker skal rutes. Det er ingen som eier BGP. (Cloudflare 2023).

2.3 Distrubert Bellman-Ford

Distribuert Bellman-Ford er en viktig algoritme som brukes for å finne korteste vei i et nettverk, der hver node tar selvstendige valg. Denne algoritmen spiller en sentral rolle innen nettverksruting, da den gir muligheten til å finne den optimale ruten mellom forskjellige punkter i et nettverk. Et konkret eksempel på bruken av Bellman-Ford-algoritmen er innen BGP (Border Gateway Protocol), som er en protokoll som brukes i internett-ruting. Ved å anvende Bellman-Ford-algoritmen kan BGP finne den mest effektive ruten mellom to noder.

Algoritmen fungerer ved at hver node i nettverket oppdaterer informasjon om den raskeste veien til alle andre noder den kjenner til. I starten har hver node en initiell oppfatning av den raskeste veien til hver nabonode. Deretter sender hver node sin oppfatning om den raskeste veien til naboene sine. Dette gjentas flere ganger til informasjonen har blitt spredt gjennom hele nettverket.

Underveis oppdaterer hver node sin egen oppfatning av hva den raskeste veien er, basert på den informasjonen den mottar fra naboene. Denne informasjonen lagres i en rutetabell. Rutetabellen inneholder informasjon om hvilken nabo som gir den korteste veien til hver enkelt node. Hvis en node mottar informasjon om en raskere vei til en node, oppdaterer den rutetabellen sin. Når alle nodene har bygget en fullstendig rutetabell, kan den raskeste veien traverseres ved å hoppe fra node til node i henhold til informasjonen i rutetabellen.

3 Metode

3.1 Oversikt

For å optimalt plassere nodene i nettverket brukes en variant av den distribuerte Bellman-Ford algoritmen. Nodene utveksler informasjon om avstanden til endepunktene i nettverket. Basert på denne informasjonen genererer hver node en "rutetabell" over kostnadder for å komme seg til endepunktene basert på informasjon fra nabonoder. Utifra dette generer noden en verdi som reflekterer dens nytteverdi for nettverket i nåværende posisjon. Deretter foretar noden en liten forflytning for å oppdatere signalstyrken til nabonodene og beregne dens bidrag til forbindelsen. Basert på disse beregningene flyttes noden i den retningen som gir størst økning av dette bidraget.

Python er et kraftig verktøy for å testing og lage prototyper. Python har biblioteker som *matplotlib* og *numpy* som gjør det lett å teste og debugge. Python ble derfor tatt i bruk for å utforme algoritmen. Deretter kunne man lage en mer effektiv simulering i et lavere nivå språk som C++. Dette kutter ned på simuleringstiden og la støtte til samtidighet, slik at simuleringen reflekterer hvordan det er i virkeligheten.

3.2 Implementasjon

Hver node er koblet sammen i en Connection. Denne representerer tilkoblingen mellom to noder. Hver tilkobling er tildelt en kostnad. Det er denne kost-funksjonen som summeres opp når man forsøker å finne raskeste vei. Kostnadden til en tilkobling er gitt ved funksjonen $C(d) = d^2$, der d er avstanden mellom de to nodene i tilkoblingen. Denne funksjonen ble valgt på bakgrunn av to kriterier:

$$C(d_1) < C(d_2) \quad \text{for} \quad 0 < d_1 < d_2 \tag{1}$$

$$2 \cdot C(d) < C(d+e) + C(d-e) \quad \text{for} \quad |e| < d \tag{2}$$

Disse begrensningene ble valgt slik at lengre avstander førte med seg større kostnad, og slik at den optimale plasseringen av en node tilkoblet til to andre stasjonære noder ligger midt i mellom de.

```
@dataclass
class Connection:

    nodes: tuple[Node, Node]
    cost: int

    def update_cost(self) -> float:
        self.cost = self.calculate_cost()

    def calculate_cost(self) -> float:
        return \
            (self.nodes[0].pos.x - self.nodes[1].pos.x) ** 2 + \
            (self.nodes[0].pos.y - self.nodes[1].pos.y) ** 2
```

Listing 1: Implementasjon av Connection klassen

Nodene styres etter en funksjon som forsøker å anslå verdien til tilkoblingene til en node. Hvis en node har tilkobling til flere endepunkter, kan dette skje gjennom en eller flere av den naboer. Funksjonen gir summen av kostnadene til alle unike tilkoblede naboer. Dette er implementert slikt.

```
def get_value(node: Node):
    connected_nodes = {route.source for route in
        node.endpoint_routes.values()}
    return - len(connected_nodes) * sum(
        node.connections[connected_node].calculate_cost()
        for connected_node in connected_nodes
```

```
    )
```

Listing 2: Implementasjonen av en funksjon som anslår verdien til en node sine tilkoblinger.

Summen er negert slik at tilkoblinger med høyre kostnad gir lavere verdi, og skalert med antall unike tilkoblinger for å straffe tilkoblinger til flere noder.

For å finne retning å bevege seg i, deriveres denne funksjonen numerisk med hensyn på x og y. På denne måten finner man retningen som øker resultatet av denne verdifunksjonen raskest.

```
def find_move_direction(node: Node, wiggle: float, move_strength: float,
    max_speed: float):
    # Do not move if the node only knows about one route to an endpoint.
    if len(node.endpoint_routes) < 2:
        return Point(0, 0)

    current_value = get_value(node)

    # How does the value change if we move the node a little bit in the x
        or y direction?
    node.pos.x += wiggle
    new_value = get_value(node)
    dx = new_value - current_value

    node.pos.x -= wiggle
    node.pos.y += wiggle
    new_value = get_value(node)
    dy = new_value - current_value

    node.pos.y -= wiggle
    speed_x = dx / wiggle * move_strength
    speed_y = dy / wiggle * move_strength

    # Limit the speed of the node.
    speed_x = min(max_speed, max(-max_speed, speed_x))
    speed_y = min(max_speed, max(-max_speed, speed_y))

    return Point(speed_x, speed_y)
```

Listing 3: Metode for å finne hvilken retning en node skal bevege seg

Hastigheten er begrenset for å stoppe at droner får ekstremt stor hastighet der den deriverte blir veldig stor.

Et CLI ble laget for å enkelt kunne kjøre simulering og eksportere resultatene. Hvordan man bruker CLI-et kan bli funnet i README.md som medfølger i mappen.

4 Resultater

4.1 Simulering

Før man leser resultatene er det anbefalt å lese om hvordan simuleringen fungerer. Dette står i README filen som medfølger. Alle figurer følger med i mappen images, dersom man vil de nærmere på dem.

Algoritmen forsøker å optimere plasseringen av noder i et åpent rom med to endepunkter. For å få best forbindlese, er den beste plasseringene av nodene en rett linje med noder som er plassert likt unna hverandre. Under er et utklipp av en simulering med 3 noder og 2 endepunkter. Denne kjøringen ble gjort med følgende kommando:

```
python main.py -n 5 -e 2 --seed 42 -d --export
```

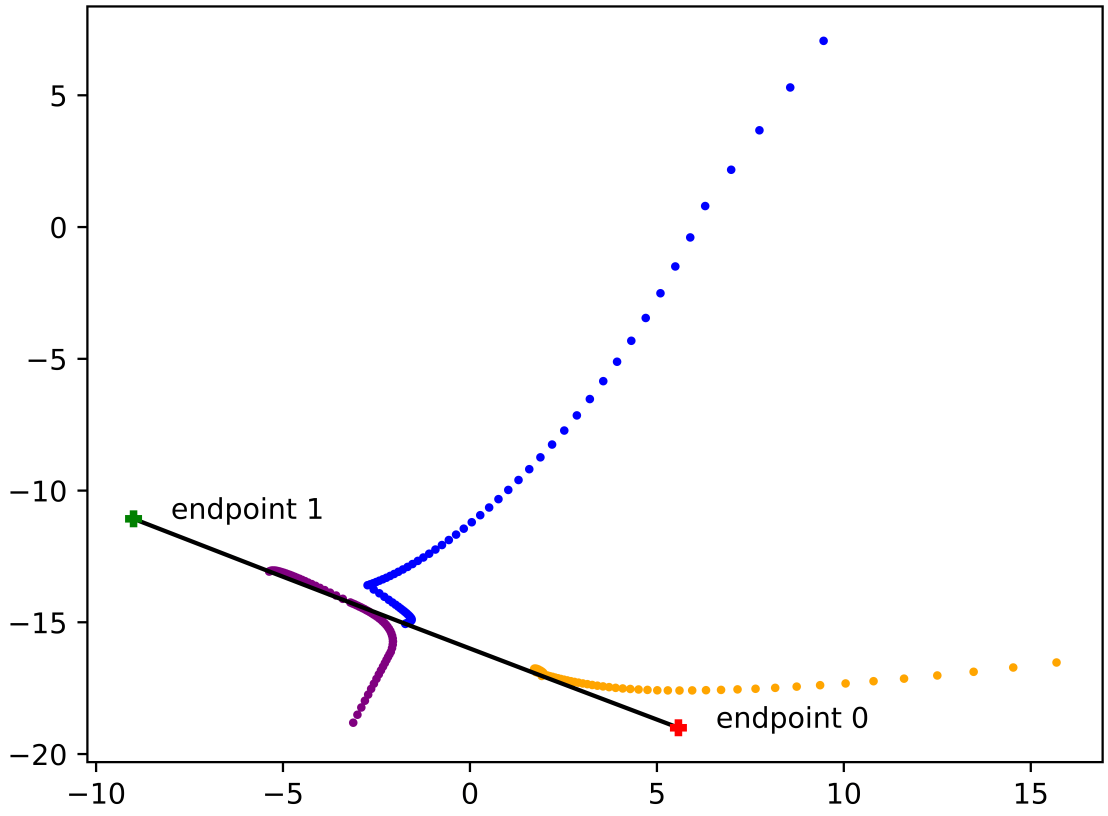

Figure 2: Optimalisering av nettverk med tre noder og to endepunkter

Med flere endepunkter, ser man at algoritmen resulterer i et stjernemønster. Simuleringen under viser til hvordan algoritmen håndterer 20 noder og 3 endepunkter. Denne simuleringen ble gjort med følgende kommando:

```
python main.py -n 20 -e 3 --seed 42 -d --export
```

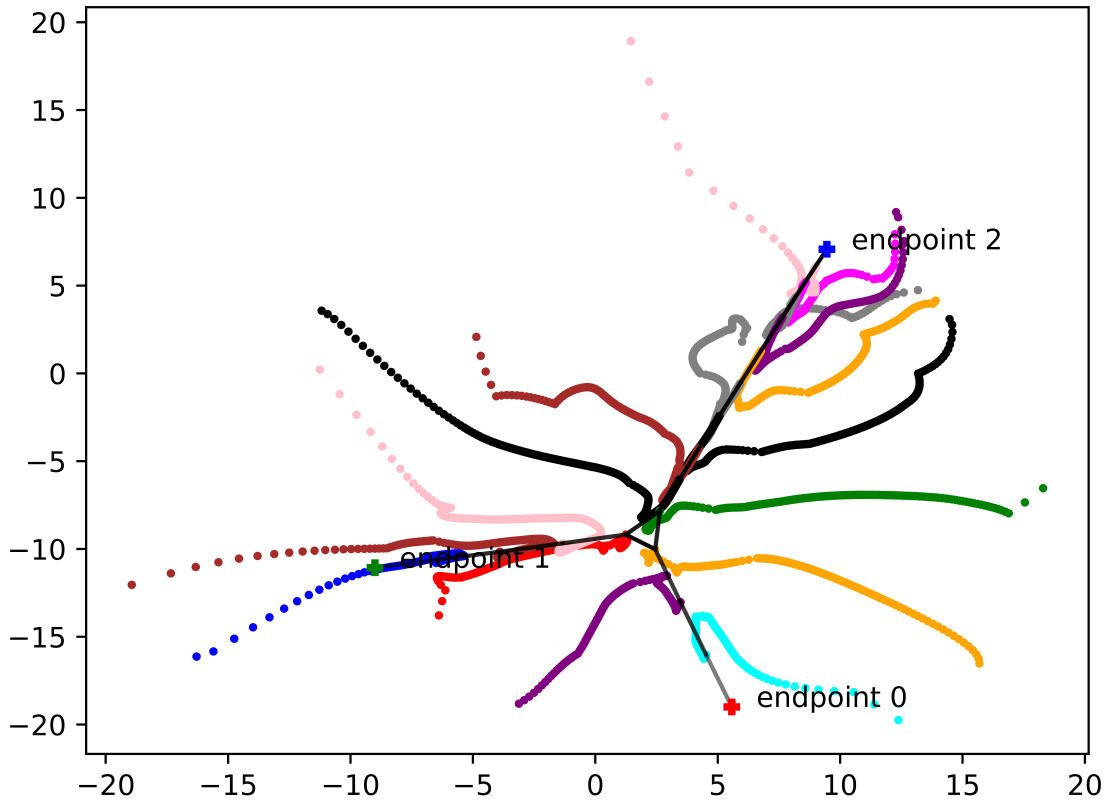

Figure 3: Optimalisering av nettverk med 20 noder og 3 endepunkter

Ved å endre nodedomenet til et lite domene, kan man simulere hvordan nodene optimaliserer koblingen når alle starter samme sted. Bildet under viser til en simulering som har blitt kjørt med følgende kommando:

```
python main.py -n 24 -e 5 --seed 42 -d --node-domain -0.1 0.1
    --endpoint-domain -20 20 --simulation-steps 3000 --export
```

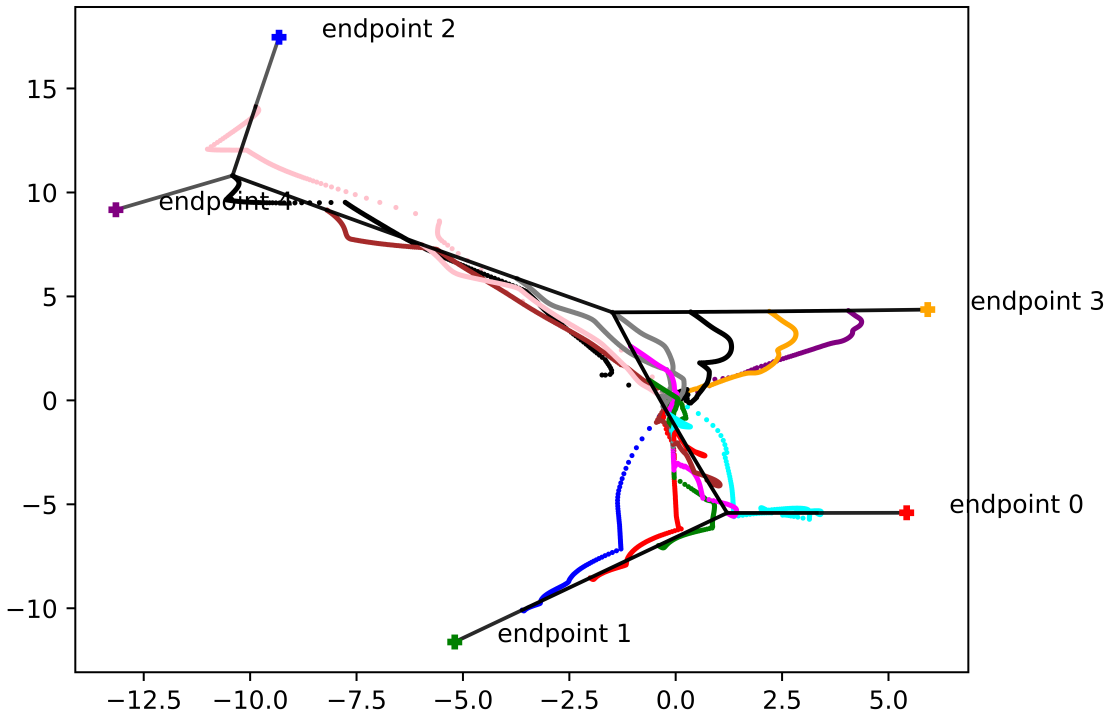

Figure 4: Danne et nettverk med 24 noder og 5 endepunkter

En simulering av et stort område ble utført ved å øke domenet i simuleringen. To kjøringer ble gjennomført: en for å optimalisere algoritmen og en annen der alle nodene startet på samme sted og dannet et nettverk.

For å utføre simuleringen som optimaliserte nettverket, ble følgende kommando brukt:

```
python main.py -n 100 -e 5 --seed 42 --node-domain -100 100 --export
```

Simuleringen der nodene starter på samme sted og danner et nettverk, ble utført med følgende kommando:

```
python main.py -n 100 -e 5 --seed 42 --endpoint-domain -100 100
    --node-domain -0.2 0.2 --export --simulation-steps 10000
```

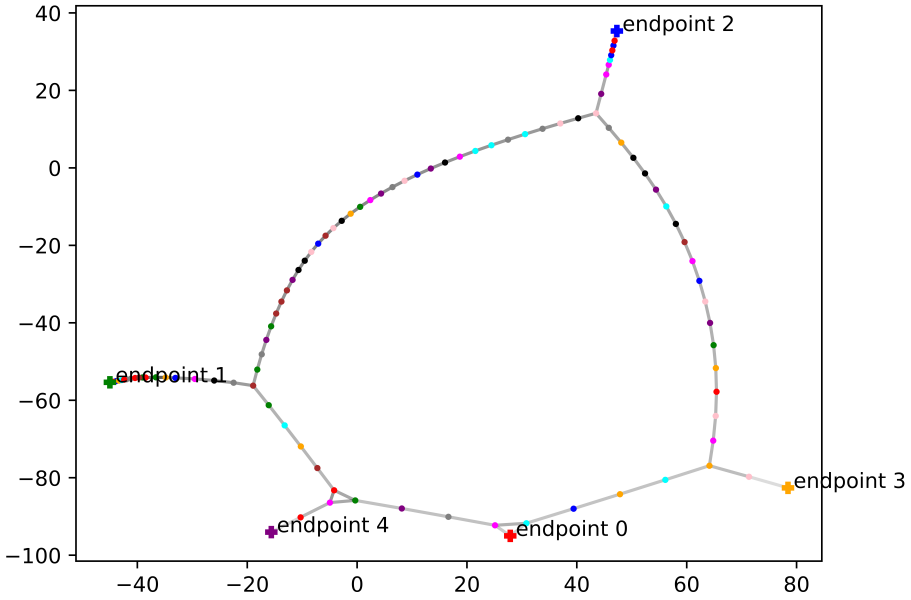

Figure 5: Optimalisering av nettverk med 100 noder og 5 endepunkter

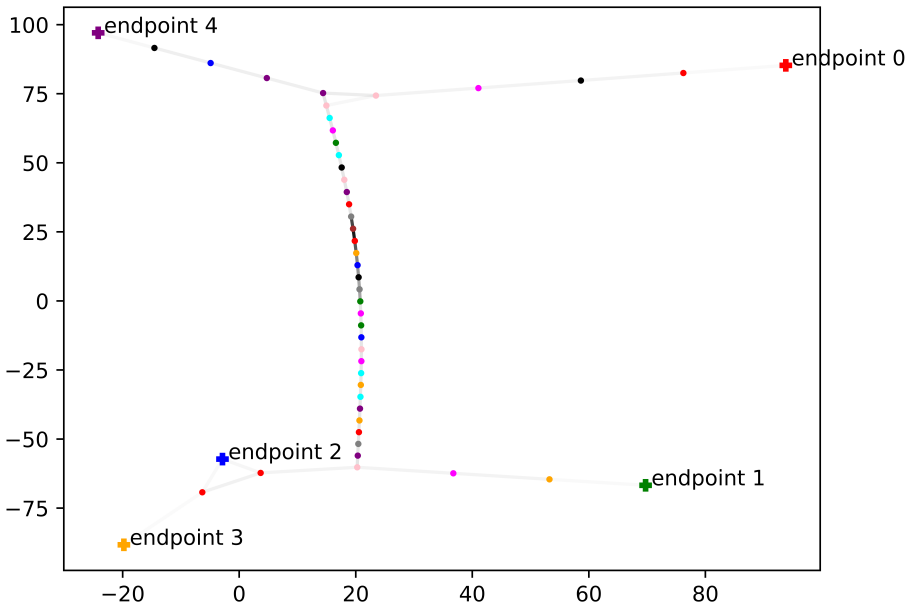

Figure 6: Danne et nettverk med 100 noder og 5 endepunkter

Under er et bilde over et kompleks og stort nettverk. Nettet har 70 endepunkter med 600 noder. For å effektivisere tidsbruk under simuleringen ble koblinger per node redusert til 20. Simuleringen ble kjørt med følgende kommando:

```
python main.py -n 600 -e 70 --node-domain -700 700 --simulation-steps 5000
    --max-connections 70 --seed 42 --export
```

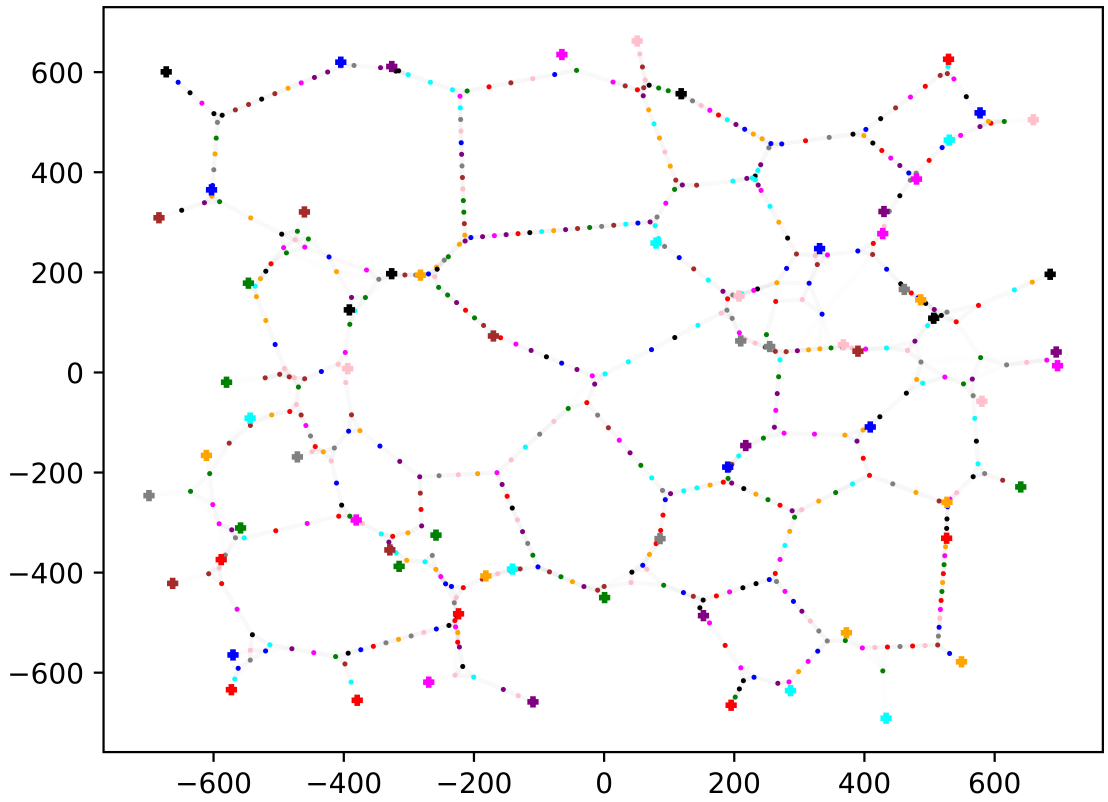

Figure 7: Optimering av et nettverk med 600 noder og 70 endepunkter

4.2 Tidskompleksitet

Simuleringens tidskompleksitet er $O(k \cdot n^2)$, der k er antall simuleringsskritt, og n er antall noder. Dette fordi n antall droner med n-1 kanter som blir sjekket sekvensielt i k simuleringssteg. Om hver node skulle kjørt sin egen versjon av algoritmen, hadde den kun trengt å prosessere informasjon fra alle nodene den er tilkoblet til. En node er koblet sammen med $n - 1$ andre noder. Dermed har algoritmen som kjører på hver node en kompleksitet på $O(n)$.

5 Diskusjon

5.1 Simulering

Testene viser at algoritmen forsøker å optimere koblingen mellom nodene, uten behov for posisjonsdata. Nodene er i stand til å ta selvstendige beslutninger basert på nettverkskoblingen mellom dem. Dermed er løsningen desentralisert. Testene viser derimot visse svakheter med algoritmen og implementasjonen.

Algoritmen er grådig og fokuserer kun på å velge den korteste hoppen i hvert simuleringstrinn, uten å ta hensyn til helheten i nettverket. Som et resultat er algoritmen ikke egnet for å oppnå

en optimal helhetsløsning, og kan resultere i en ujevn fordeling av noder rundt endepunktene. Et godt eksempel på dette kan sees i Figur 6. Her starter alle nodene i et lite domene på (-0.1, 0.1), og som et resultat havner de fleste nodene på den nærmeste koblingen. Dette kan skape flaskehalser og et ikke-optimalt nettverk. I tillegg medfører det unødvendig bruk av noder.

Testene viser at algoritmen skalerer godt og er i stand til å danne et nettverk med høy grad av redundans. Dette betyr at hvis en node fjernes fra nettverket, vil de gjenværende nodene kunne fylle tomrommet og opprettholde tilkoblingen. Dette er en viktig egenskap da det bidrar til økt pålitelighet og robusthet i nettverket. Algoritmen demonstrerer dermed evnen til å tilpasse seg endringer i nettverket og opprettholde kontinuerlig kommunikasjon, selv i tilfelle feil eller tap av enkelte noder. Denne redundansen bidrar til å sikre stabilitet og pålitelighet i nettverket som helhet.

5.2 Nettverkstopologi

Optimeringen av hver enkelt node tar utgangspunkt i to ting.

1. Minimer avstanden til noder som gir raskeste vei til endepunkter.
2. Plasser noden midt i mellom disse nodene.

Håpet var at den lokale optimeringen etter disse prinsippene skulle føre til en global optimering av nettverket. Dette skjer til en viss grad. Nodene har en tendens til å danne nettverk med rette streker mellom endepunkter og knutepunkter. Dette ser man eksempelvis i *figur 4*. Algoritmen eliminerer unødige parallelle koblinger, men det gjør ingen ting for homogenisere tettheten av noder globalt. Dersom en strekning har veldig høy tetthet av noder "vet" ikke hver enkelt node om det bare er veldig mange noder, eller om nodene er skjevfordelt. Dette gjelder spesielt for store nettverk, og er eksemplifisert i *figur 6*. Altså: Nodene danner nettverk uten sideliggende tilkoblinger, og minker avstanden mellom endepunkter og knytepunkter, men den sørger ikke for at den globale fordelingen av noder er homogen.

5.3 Tidskompleksitet

Som nevnt i resultatene, er tidskompleksiteten for simuleringen $O(k \cdot n^2)$ og tidskompleksiteten for en gjennomgang av algoritmen på en node er $O(n)$. I praksis vil kompleksiteten være mindre. I virkeligheten vil ikke enhver node ha en tilkobling til hver eneste node i nettverket, blant annet på grunn av rekkevidden til signalene. Om hver node hadde hatt en begrensning på m kanter, hadde tidskomplekisteten for hver node vært $O(m)$. Kompleksiteten for hele simuleringen hadde dermed vært $O(k \cdot n \cdot m)$. Kompleksiteten blir dermed avhengig av hvor stort domenet og hvor mange droner og endepunkter det er, og hvordan disse er koblet sammen. Signalfrekvensen spiller også en viktig rolle i algoritmens tidskompleksitet, da høyere signalfrekvens har kortere rekkevidde, men høyere hastighet.

For å få en nøyaktig vurdering av kompleksiteten, er det nødvendig med mer nøyaktig simulering eller testing i virkeligheten. Det er imidlertid verdt å merke seg at en tilsvarende algoritme brukes i BGP (Border Gateway Protocol), som har en tidskompleksitet på $O(E)$, der E er antall ruter som en ruter har kontakt med (Bremler-Barr et al. 2023). BGP er i stand til å skalere til hele internettet i dag, og derfor kan det antas at en lignende kompleksitet også kan være passende for et sub-nettverk.

Hvis kompleksiteten skulle bli et problem, kan man begrense antall noder som mottar ruter ved å sortere rutene og sende dem til et begrenset antall noder. På denne måten kan man potensielt redusere beregninger en node må gjøre. Imidlertid bør en grundig analyse og ytterligere testing utføres for å evaluere effekten av en slik begrensning og for å sikre optimal ytelse i nettverket.

5.4 Videre arbeid

Algoritmen egner seg til å optimalisere et allerede laget nettverk, men bør ikke brukes til å danne et nettverk. I simuleringen har alle noder kontakt med endepunktet, men i virkeligheten er det ikke slik. Dette kan føre til at noen endepunkt kan bli glemt, fordi ingen noder har kontakt med dem. En annen algoritme bør derfor brukes dersom man skal danne et nettverk. Man kan også prøve å optimere algoritmen ved å endre på parametere slik at den kan danne bedre nettverk. Som et eksempel, kan det være interessant å se på om andre funksjoner for kostnadden for å traversere mellom to noder, eller andre funksjoner for å beregne verdien til en node, kan hjelpe mot noen av svakhetene med algoritmen (for eksempel at den har en tendens til å ikke fordele nodene homogent om enkelte strekninger har en høyere tetthet).

Det er viktig å erkjenne at den grådige algoritmen kanskje ikke er egnet for å danne et nettverk på en optimal måte. Derfor kan det være hensiktsmessig å vurdere av annen løsning som innebærer to faser: en for optimalisering og en for nettverksdannelse. Det er viktig å sikre at alle dronene bytter fase samtidig for å opprettholde koordineringen. Dette kan oppnås ved å implementere en tidsstyring, bruk av signaler eller andre metoder. Det er likevel behov for ytterligere forskning på dette området for å finne den beste tilnærmingen. Ved å utforske alternativer kan man muligens utvikle en mer effektiv og tilpasset løsning for å oppnå både optimalisering og nettverksdannelse i dronenettverk.

For øyeblikket tar ikke algoritmen hensyn til hindringer i miljøet. Dette begrenser dens evne til å håndtere komplekse og varierte landskap eller områder med fysiske hindringer, noe som begrenser mulighetene til å bruke algoritmen i virkelige scenerier. Videre forskning og utvikling må derfor utføres for å forbedre algoritmen på dette området.

6 Konklusjon

Målet med løsningen er å optimere nettverkstilkoblingen ved en distribuert måte, uten bruk av posisjonsdata. Algoritmen presentert i rapporten demonstrerer at dette er mulig, selv om det fortsatt gjenstår betydelig arbeid for å identifisere potensielle svakheter og optimere ytterligere. Det er nødvendig med en mer realistisk simulering som for eksempel inkluderer hindringer for å teste algoritmen i en mer realistisk kontekst.

Det er imidlertid spennende å utforske potensielle bruksområder for denne algoritmen. Algoritmen fungerer grådig. Den vil dermed alltid forsøke å forbedre sin nåværende posisjon, og aldri gå tilbake etter "dårlige" valg. En svakhet med dette er at det er svært vanskelig å oppnå den mest optimale konfigurasjonen av aksesspunkter. Om man kjører simuleringen med mange noder, vil konfigurasjonen mest sannsynlig stoppe når den når et lokalt minima, en konfigurasjon der alle mulige endringer vil gjøre konfigurasjonen dårligere. Likevel kan den fungere som et sekundærsystem for nettverk som krever høy redundans når posisjonsdata ikke er tilgjengelig, eller som et sekundærsystem for et sentralstyrt system. Det kan også være mulig å kombinere denne algoritmen med andre algoritmer for å kompensere for dens grådighet.

I tillegg til å være et potensielt sekundærsystem for nettverk med høy redundans, kan denne algoritmen også være verdifull i situasjoner der rask implementering og fleksibilitet er avgjørende. For eksempel kan den brukes i nød- eller katastrofeområder der infrastrukturen er begrenset eller skadet. Ved å utnytte algoritmens evne til å optimere nettverkstilkoblingen uten posisjonsdata, kan man raskt etablere midlertidige nettverk for å muliggjøre kommunikasjon og informasjonsdeling.

Det er viktig å merke seg at selv om denne algoritmen viser lovende resultater, er det fortsatt behov for grundig testing og ytterligere forskning for å validere dens pålitelighet og ytelse i ulike scenarier. En iterativ tilnærming som tar hensyn til tilbakemeldinger og erfaringer fra praktiske implementeringer vil være avgjørende for å forbedre algoritmens effektivitet og adressere eventuelle begrensninger.

Samlet sett representerer denne rapporten et viktig skritt mot å forstå potensialet til den presenterte algoritmen og dens anvendelser. Videre arbeid vil bidra til å styrke dens ytelse, pålitelighet og

bredere anvendbarhet i ulike kontekster der nettverkstilkobling uten posisjonsdata er avgjørende.

Bibliography

Bremler-Barr, Anta et al. (2023). *Bringing order to BGP: Decreasing time and message complexity.* URL: https://www.sciencedirect.com/science/article/abs/pii/S1389128609001522 (visited on 20th May 2023).

Cloudflare (2023). *What is BGP?* URL: https://www.cloudflare.com/learning/security/glossary/what-is-bgp/ (visited on 18th May 2023).

Research, Tarlogic (2023). *Wifi planning, Design and locate access points in simulation.* URL: https://www.acrylicwifi.com/en/blog/wifi-planning-design-place-locate-access-point-simulation/ (visited on 21st May 2023).

Wikipedia (2023). *Router (computing).* URL: https://en.wikipedia.org/wiki/Router_(computing) (visited on 18th May 2023).