Optimalisatie metro-aanleg in Utrecht met lokaal zoeken

Ildiko Flesch Arthur van Dam

21 januari 2000

1 Inleiding

Een belangrijke tak in de Computational Science is de klasse van optimalisatieproblemen. Bij dergelijke problemen is een bepaalde situatie gemodelleerd, waarin vaak veel verschillende parameters elkaar beïnvloeden. Daarbij is er een kostenfunctie, die in termen van die parameters uitdrukt, wat de kosten zijn van de huidige toestand van het model.

Tijdens het practicum[1], wat in dit verslag wordt beschreven, is het openbaar-vervoersnetwerk van de stad Utrecht bekeken. Het bestaande buslijnennetwerk is bekend en doel is, om één of meerdere metrolijnen aan te leggen zodanig dat de kostenfunctie van het model daalt.

Zoals vaak gebeurd bij optimalisatie van netwerk- of graaf-problemen, wordt hier een methode van *lokaal zoeken* gebruikt, specifieker: *simulated Annealing*.

2 Probleembeschrijving

2.1 Huidige situatie

Het openbaar-vervoersnetwerk van Utrecht wordt dus bekeken. Hierbij wordt alleen naar de stadsbussen (van het GVU) gekeken. Om het probleem wat globaler te houden, worden alleen de lijnen 1, 2, 3, 4, 5, 6, 7, 8, 11, en 12 bekeken. Hiervan zijn alle haltes bekend en deze huidige toestand van het vervoersnetwerk is in appendix A in kaart gebracht.

Over het aantal reizigers dat vervoerd wordt, is minder bekend; de volgende gegevens zijn beschikbaar:

> *Per dag reizen er circa 121.000 mensen met de bus,*
> *als volgt verdeeld over de lijnen:*
>
> | | | |
> |---|---|---|
> | *CS - De Uithof* | *24%* | *bus 11, 12* |
> | *Zuilen - Kanaleneiland* | *13%* | *bus 7* |
> | *Overvecht Noord - Lunetten* | *12%* | *bus 1* |
> | *Zuilen - Galgenwaard* | *11%* | *bus 3* |
> | *Zuilen - Hoograven* | *9%* | *bus 6* |

[1] Dit practicum is onderdeel van het vak Optimalisering, faculteit Informatica, 1999/2000

2.2 Nieuwe situatie: metro-aanleg

In het kader van een (fictief) project heeft de gemeente Utrecht een bedrag ontvangen voor de aanleg van één of meerdere metro-lijnen. Eis is dat iedere metro-lijn langs Centraal Station loopt. Verdere aannamen en eisen hierover staan in sectie 3.3.

Het voordeel van een metro is dat de reizigers sneller van A naar B kunnen reizen, bovendien kan een metro – door de grotere capaciteit – eventueel een of meer buslijnen (gedeeltelijk) vervangen.
Enkele gegevens over de twee vervoerstypen in Utrecht:

	capaciteit	*gem. snelheid*
bus	*50 - 100*	*12*
metro	*200 - 1000*	*30*

Waar bij de modellering en implementatie op gelet zal worden is:

- hoe de reistijd en traject-/materiaalkosten in één kostenfunctie kunnen worden gebracht,
- hoe aan bepaalde belangrijke haltes (CS, stadion) hogere prioriteit gegeven kan worden,
- hoe er rekening kan worden gehouden met het feit dat er in bepaalde gebieden – zoals bijv. de binnenstad en de Uithof – meer mensen uitstappen dan elders,
- hoe verandering van de verschillende soorten kosten de oplossing kan beïnvloeden.

In de nu volgende secties zal worden beschreven hoe deze situatie flexibel gemodelleerd wordt, en hoe oplossingen op bovenstaande punten zijn gevonden.

3 Model

Om tot een efficiënt en eenvoudig uitbreidbaar programma te komen is gekozen voor een object-georiënteerde aanpak.

3.1 Objecten

In het model worden de volgende objecten onderscheiden:

Network Hierin worden alle losse componenten van het vervoers-netwerk bijeengebracht; een lijst van transportmiddelen en een lijst van haltes. In **Network** staan ook algemene gegevens, waaronder overstaptijden, afstand- en tijd-tabellen.

Transport Dit geeft een type vervoersmiddel aan. In het gebruikte model zijn er twee: de bus en de metro, maar in andere situaties is het heel goed mogelijk om er bijvoorbeeld een tram, of trein aan toe te voegen. Bij een **Transport** horen gegevens over de snelheid, capaciteit en kosten. Ook is er een lijst van **Line**-objecten, welke door dit vervoerstype verreden worden.

Line Dit is een lijn, aangegeven door een lijst van haltes (**Stop**s), waarlangs de route loopt en een frequentie waarmee deze lijn rijdt.

Stop Dit is een halte; aanname is dat een halte door alle vervoerstypen bereikt kan worden. Er wordt ook bijgehouden welke lijnen er langs deze halte rijden.

3.2 Systeemparameters

Om tot een flexibel programma te komen, kunnen vrijwel alle parameters van buitenaf opgegeven worden, dit gebeurt door middel van een aantal data-files die door het programma ingelezen worden. Zodoende kunnen de volgende zaken ingesteld worden:

- Network

  Overstaptijd Geeft in uren aan, hoeveeltijd het kost om bij een halte van lijn te wisselen.

- Transport

  Capaciteit Aantal passagiers dat in één voertuig kunnen.

  Snelheid Gemiddelde snelheid in km/u.

  Constante kosten Basiskosten voor het laten rijden van één voertuig.

  Variabele kosten Kosten per km. voor het laten rijden van één voertuig.

- Line

  Transporttype Transporttype waarmee deze lijn wordt gereden.

  Haltes Lijst van haltes waarlangs deze lijn rijdt.

  Frequentie Frequentie waarmee de lijn wordt gereden (keren per uur).

- Stop

  x- en y-coordinaat Locatie van de halte.

- Simulated Annealing

  Start temperatuur Temperatuur waarbij de optimalisatie gestart wordt.

  Limiet temperatuur Temperatuur waarbij de optimalisatie gestopt wordt (ondergrens).

  Koel-factor Factor waarmee de temperatuur wordt vermenigvuldigd na een koelperiode.

  Koel-periode Aantal bekeken buurruimten, voordat de temperatuur wordt aangepast.

  Output-periode Aantal bekeken buurruimten, voordat het systeem in *GNU-Plot*-formaat wordt geëxporteerd.

- Passagiersverdeling

  % 'one-liners' Percentage mensen, dat op één lijn rijdt.

  % 'important-places-visitors' Percentage mensen dat de belangrijke locaties langs die lijn bezoekt.

Zodra bovenstaande gegevens zijn ingeladen, wordt een aantal matrices met gegevens gevuld, zoals tijd- en afstand-tabellen; dit bespaart een hoop overbodig herhaald rekenwerk in de optimalisatie-iteratie.

3.3 Aannamen en eisen

Voordat er geoptimaliseerd wordt, worden eerst nog een aantal aannamen en eisen gedaan:

- Voertuigen hebben een constante snelheid v_{ij}, onafhankelijk van op welk traject ij gereden wordt.
- Het aantal passagiers c_{ij} dat op een traject ij reist, is constant. Dit is een redelijke aanname, omdat het gebruikte model op uurbasis is; voor alle uren van de dag kan een andere optimalisatie worden uitgevoerd.
- Er zijn voldoende bussen en metro's beschikbaar.
- De frequentie van een lijn moet minstens 6 maal per uur zijn; dit stelt een bovengrens aan de wachttijd.
- De frequentie van een lijn blijft na aanleg ongewijzigd (zie ook sectie 3.5).
- De overstaptijd is voor alle haltes en lijnen gelijk.
- Metrohaltes moeten minstens 1 km uit elkaar liggen, om te profiteren van de snelheid van de metro.
- De metro moet langs Centraal Station gaan, om daar grote capaciteit te hebben.
- De totale lengte van alle metro-trajecten mag niet groter zijn dan 10 km, omdat de aanleg ervan zeer kostbaar is.

3.4 Passagiers-verdeling

Zoals gebleken, zijn er vrij weinig gegevens bekend over de gevraagde capaciteiten op alle trajecten, alleen het tabelletje in sectie 2.1 is beschikbaar.

Als naar de kaart in appendix A wordt gekeken, blijkt dat alle buslijnen (en ook alle toekomstige metro-lijnen) langs Centraal Station rijden. Zoals welbekend stappen hier verreweg de meeste mensen in en uit de bus. CS is dus erg belangrijk.

Een wat algemenere aanpak is echter wenselijk; er wordt een lijst bijgehouden van haltes die belangrijk zijn[2]. Bij het bepalen van de aantallen reizigers op alle deeltrajecten ij, wordt onderscheid gemaakt tussen mensen die met één lijn reizen en mensen die met meerdere reizen.

Van de getallen die in eerdergenoemd tabelletje staan, reizen $l\%$ mensen met die ene lijn, de rest moet daarna nog met een andere lijn verder, of kwam al van een andere. Van dit percentage reist in totaal $p\%$ naar één van de belangrijke plaatsen die langs die lijn ligt en die reizigers zijn op een willekeurige halte van die lijn opgestapt. De andere $100 - p\%$ reist tussen twee willekeurige haltes van die lijn. Tot slot wordt de overgebleven $100 - l\%$ van het totaal aantal reizigers over alle $n^2 - n$ deeltrajecten verdeeld.

Aangezien het huidige bus-netwerk een goede bereikbaarheid geeft, is het vaak zo dat mensen met één buslijn al op hun plaats van bestemming kunnen komen. Dit betekent dat l vrij hoog kan worden genomen (bijvoorbeeld 85%).
De waarde van p is wat minder eenvoudig te schatten; de *important places* zijn zo verschillend

[2]Dit zijn de 12 haltes die in Appendix A een naam anders dan 'Stop...' hebben

van aard, dat op hele verschillende momenten het naar de ene plaats veel drukker is dan naar de andere. Omdat deze plaatsen toch gemiddeld erg veel reizigers trekken, kan p op zo'n 75% worden gezet. Natuurlijk zijn deze parameters allemaal van buitenaf aan te passen in het programma.

3.5 Frequentieverdeling

Iedere lijn heeft een eigen frequentie, met een ondergrens van 6 per uur. Natuurlijk kan bij inzet van een metro de frequentie van een buslijn die gedeeltelijk langs dat metrotraject loopt, verlaagd worden. Op het gedeelte van die buslijn waar de metro niet rijdt, moet de bus echter nog wel altijd de gevraagde capaciteit leveren. En dit hangt ook weer samen met, en kan aangevuld worden door andere aanliggende bus- of metro-lijnen.
Dit levert een groot samenhangend systeem op, dat vertaald kan worden in een matrix-vector-stelsel met dimensie n (aantal haltes; hier 66). Omdat het oplossen cq. benaderen van de oplossing buiten de scope van dit practicum ligt, is er een aanname gedaan:
Bij het aanmaken van een lijn wordt deze standaard op een bepaalde frequentie groter/gelijk de ondergrens gezet, en deze wordt tijdens SA niet meer veranderd. Zodra SA een oplossing levert, kan (eventueel met hulp van computer-output) handmatig worden vastgesteld of de gevraagde capaciteiten c_{ij} geleverd worden, en of van bepaalde lijnen de frequenties eventueel nog omlaag kunnen.

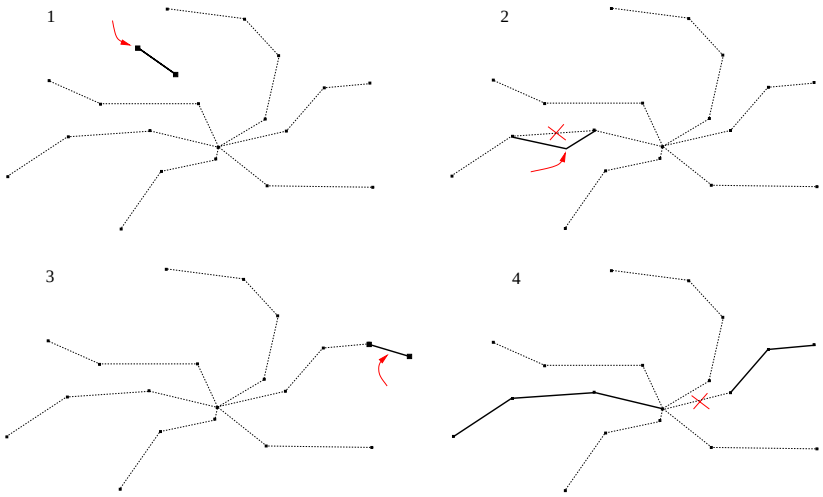
4 Optimalisatie

Nu alle gegevens in een bruikbare structuur zijn gezet, en de voorwaarden voor het systeem bekend zijn, kan er geoptimaliseerd worden. Zoals gezegd wordt er een methode van *lokaal zoeken* gebruikt; Simulated Annealing (SA).

SA bekijkt – gegeven een bepaalde toestand van het systeem – een bepaalde buurruimte. Een buurruimte is weer de zelfde toestand van het systeem, met kleine verandering(en) erin aangebracht. In dit specifieke geval komt dat neer op één van onderstaande aanpassingen (zie ook fig. 1):

1. Toevoegen van een lijn met 2 haltes; deze kan later uitgroeien tot een grotere lijn.
2. Toevoegen van een halte aan een bestaande lijn, om de bereikbaarheid van die halte te vergroten. Als eis wordt wel gesteld dat de halte in de buurt van de lijn ligt, omdat deze anders te ver moet omrijden.
3. Toevoegen van een halte aan het begin of einde van een lijn.
4. Verwijderen van een traject tussen twee aanliggende haltes uit een lijn, waardoor de lijn wordt opgesplitst (handig als daartussen bijvoorbeeld een metro komt), of 1 deeltraject korter wordt. Bij de metro mag Centraal Station niet verwijderd worden.

Bij de eerste drie punten moet, alvorens toe te voegen, worden gecontroleerd of de maximale lengte van het metro-traject niet overschreden wordt. In werkelijkheid wordt de toevoeging altijd gedaan. Zodra bij het afsluiten van de SA-slag dan de kosten worden bepaald, wordt in geval van te grote lengte, kosten oneindig geretourneerd, en wordt de toelatingskans in SA gelijk aan 0. Bij het eerste punt, moet in geval van een metro-lijn Centraal Station in de



Figuur 1: De vier gebruikte buursituaties in een vervoersnetwerk

lijn komen. Bij het laatste punt wordt gekeken of Centraal Station niet uit een metro-lijn verwijderd wordt.

Vervolgens worden de kosten K_{new} van de buurruimte vergeleken met de kosten K_{old} van de oorspronkelijke toestand en wordt aan de hand daarvan beslist of de buurruimte wordt toegelaten of niet. Simulated Annealing gebruikt hierbij een een controle-parameter, de temperatuur T. In het begin van de simulatie is T nog groot, en worden nog vrij vaak verhogingen van de kosten toegelaten. Dit voorkomt dat de optimalisatie al snel vast zou komen in een lokaal minimum. Naarmate de optimalisatie vordert, wordt deze temperatuur verlaagd, door te vermenigvuldigen met een koelfactor ν ($0 < \nu < 1$). Door de dalende temperatuur, worden steeds minder verslechteringen toegestaan, waardoor steeds lagere kosten worden bereikt.

Dit alles komt tot uiting in een toelatingsskans p, voor een zekere buurruimte:

- Als $K_{new} < K_{old}$: $p_{accept} = 1$
- Als $K_{new} > K_{old}$: $p_{accept} = e^{-(K_{new} - K_{old})/T}$

Wanneer een buurruimte niet wordt toegelaten, wordt de oorspronkelijke toestand van het systeem teruggezet; hier komt dat neer op het tijdelijk back-uppen van alle **Line**-objecten, en deze later eventueel weer terug te zetten. De kans p wordt dus beïnvloed door de kosten, maar deze wordt nog genormeerd door de waarde van T. T wordt zo gekozen, dat bij de start van de optimalisatie een verslechtering met kans van ongeveer 0.5 toegestaan wordt.

4.1 Kostenbepaling

Zoals gebleken, hangt het verloop van SA volledig af van de kosten van het systeem. Hoe worden de kosten nu bepaald?
In het model is het wenselijk om twee dingen te optimaliseren:

- Minimaliseren van de reistijden
- Minimaliseren van de vervoerskosten voor het GVU.

Hierbij moge het duidelijk zijn dat minimalisering van reistijd op een traject met veel reizigers hoge prioriteit heeft. De vervoerskosten zijn simpelweg een sommatie van de constante en variabele kosten van alle vervoerstypen. Dit alles komt neer op het minimaliseren van de volgende kostenfunctie:

$$K = \alpha \sum_{i,j} t_{ij} c_{ij} + \beta \sum_{l} (cc + length(l) \cdot vc) \tag{1}$$

waarbij i, j een tweetal haltes zijn, t_{ij} snelste tijd om van i naar j te reizen, c_{ij} het aantal passagiers dat van i naar j reist, l een lijn, $length(l)$ lengte in km van een lijn l. α en β zijn wegings-factoren die aangeven hoe zwaar de twee kostensoorten meewegen in de totale kosten ($\alpha + \beta = 1$). Eigenlijk zijn er nog extra kosten voor de metro, de hele infrastructuur moet hiervoor nog aangelegd worden, dit is beraamd op 500.000.000 Euro. Deze kosten staan echter in geen enkel verband tot de reis- en vervoerskosten. Ook eenheden kunnen hier niet vergeleken worden. Omdat het hier bovendien een eenmalige uitgave betreft, zijn deze kosten niet in de kostenfunctie meegenomen. Wel is de al eerder genoemde restrictie opgelegd, dat de metro niet langer dan 10 km mag worden.

Het bepalen van de vervoerskosten is dus een eenvoudige sommatie, maar het bepalen van de snelste tijden behoeft nog enige uitleg.

4.1.1 Snelste routes met Dijkstra

Het Dijkstra-algoritme is een zogenaamd 'kortste-paden-algoritme'. De definitie van kortste paden, wordt hier volledig gebaseerd op de wensen van de passagiers. Dit betekent dat de *snelste* routes bepaald moeten worden.

Het Dijkstra algoritme lost een zoekprobleem op, van het kortste pad van één punt (*source*) naar alle andere punten in een gerichte graaf, waarbij elk knooppunt een positief gewicht heeft. In dit geval zijn de knooppunten de haltes, en de gewichten van de knooppunten de tijd die nodig is om van de source naar dat punt te reizen.

Er worden twee verzamelingen van knopen, S en Q gebruikt. S bevat knopen waarbij het kortste pad vanaf de source al is uitgerekend. In Q zitten de knopen, waarvoor nog een kortste pad bepaald moet worden. Bij de initialisatie wordt het gewicht van iedere knoop op oneindig gezet, behalve de bronhalte, die de waarde 0 krijgt. Vervolgens wordt in een lus de halte uit de verzameling Q gekozen, die het kleinste gewicht heeft. Deze wordt vanuit Q in S gezet. Bij de start van Dijkstra kan deze uitgekozen halte alleen de bronhalte zijn. Voor de gekozen halte u worden de haltes bekeken die via een directe verbinding bereikbaar zijn vanuit u. Deze staan in de *adjacency-lijst* van u. Voor iedere buurhalte wordt een aanroep gedaan naar de zogenaamde RELAX-routine.

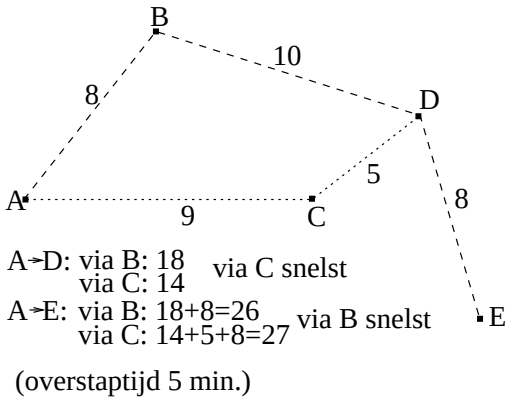
RELAX(U,V) bekijkt of v sneller bereikbaar is via u dan hij op dit moment is, in formulevorm:

$$w(u) + t(u, v) \stackrel{?}{<} w(v) \tag{2}$$

Hierin is $t(u, v)$ de lengte van de directe weg van u naar v. Als dit geldt, dan wordt het gewicht van v aangepast.

In het vervoersprobleem moet dit algoritme echter iets aangepast worden, omdat er geen rekening wordt gehouden met de verschillende snelheden van vervoersmiddelen en met de overstaptijden.

Het is heel belangrijk te realiseren , dat als een passagier optimaal vanaf de source naar een andere halte met bepaalde trajecten en lijnen gaat, het niet zeker is, dat hij naar de



Figuur 2: Kortste paden variëren met de destination-stop

volgende buurhalte ook met de tot nu toe gebruikte trajecten en lijnen moet reizen. Dit is geïllustreerd in figuur 2

Om Dijkstra deze nieuwe vorm van 'kortste paden' te laten vinden, moeten bij een halte alle lijnen opgeslagen worden, waarmee die halte bereikt kan worden, inclusief de bijbehorende reistijden vanaf de source. Als voor een ander punt v een kortste route wordt gezocht, wordt bij de buur u gekeken, hoe er bij die buur gekomen kon worden, vervolgens worden die aankomstlijnen van u vergeleken met de beschikbare lijnen tussen u en v. Voor alle combinaties worden de tijden bepaald, eventueel verhoogd met een overstaptijd. Voor iedere lijn tussen u en v wordt de beste combinatie met één van de aankomende lijnen in u gekozen (liefst zonder overstap) en deze wordt opgeslagen in een verzameling bij het punt v.

In het programma worden de gegevens bij iedere knoop opgeslagen in een struct. In deze struct staat een array van lijnen, waarmee je (eventueel met eerdere overstappen) vanuit de source dat punt kunt bereiken. Ook is er een array die bij de zojuist opgeslagen inkomende lijnen de tijd aangeeft die het kostte om vanuit de source te komen. Verschillende haltes kunnen natuurlijk verschillende lengte van deze arrays nodig hebben en dit is dus variabel gehouden door gebruik van vectoren.
Tot slot is er nog een andere variabele, die de index van de voor het punt zelf optimale aankomende lijn en tijd in de vectoren aangeeft.

Zoals reeds gezegd in sectie 3.2 worden een aantal matrix-datastructuren ingevuld met gegevens over afstand en tijd tussen punten i en j en welke lijnen er tussen die twee punten lopen. Ook wordt bij ieder punt een *adjacency-lijst* bijgehouden, die aangeeft welke buren dat punt heeft.

Om de kortste paden tussen alle paren punten te bepalen, wordt in een lus over alle punten (haltes) dijkstra aangeroepen met die halte als source. Dijkstra retourneert de gegevens voor ieder punt, waarmee bepaald kan worden hoe ieder punt het snelst bereikt kan worden. Hieruit worden de kosten bepaald en deze datastructuren worden in de volgende slag van de lus weer hergebruikt voor een nieuwe source, om zo snelheid te winnen en geheugen te besparen.

Uiteindelijk worden de totale reiskosten geretourneerd, en ligt de controle weer bij SA.

Mochten er punten zijn, waarnaar geen weg mogelijk is, dan wordt dit in de SHORTESTPATHS methode gedetecteerd en worden er ook weer kosten oneindig geretourneerd. De nieuwe

toestand van het model zal dan dus niet geaccepteerd worden, doordat $P_{accept} = e^{-\infty} = 0$.

5 Experimenten

Omdat er bijzonder veel veranderd kan worden aan het vervoersnetwerk, is de verwachting dat SA een nogal rommelig netwerk kan opleveren. Om dit een beetje beperkt te houden, is er een bovengrens gesteld aan het aantal lijnen dat toegevoegd mag worden, daarnaast mogen niet extreem lange lijnstukken gemaakt worden (bijv. Westplein - AZU). Als eerste een experiment, waarbij de reiskosten en vervoerskosten even zwaar wegen ($\alpha = \beta = 0.5$). Hierbij werd een afname van 5% gevonden en dat valt dus wat tegen. Zie figuur 3.

Om betere service te leveren aan de reizigers, kan ook de reiskosten ook volledige prioriteit worden gegeven ($\alpha = 1$). Zie figuur 4.
Duidelijk is te zien, dat nu erg veel lijnen worden aangelegd en er een erg rommelig netwerk ontstaat. De kosten zijn zelfs met 15% toegenomen. De vervoerskosten zijn dus wel degelijk belangrijk.

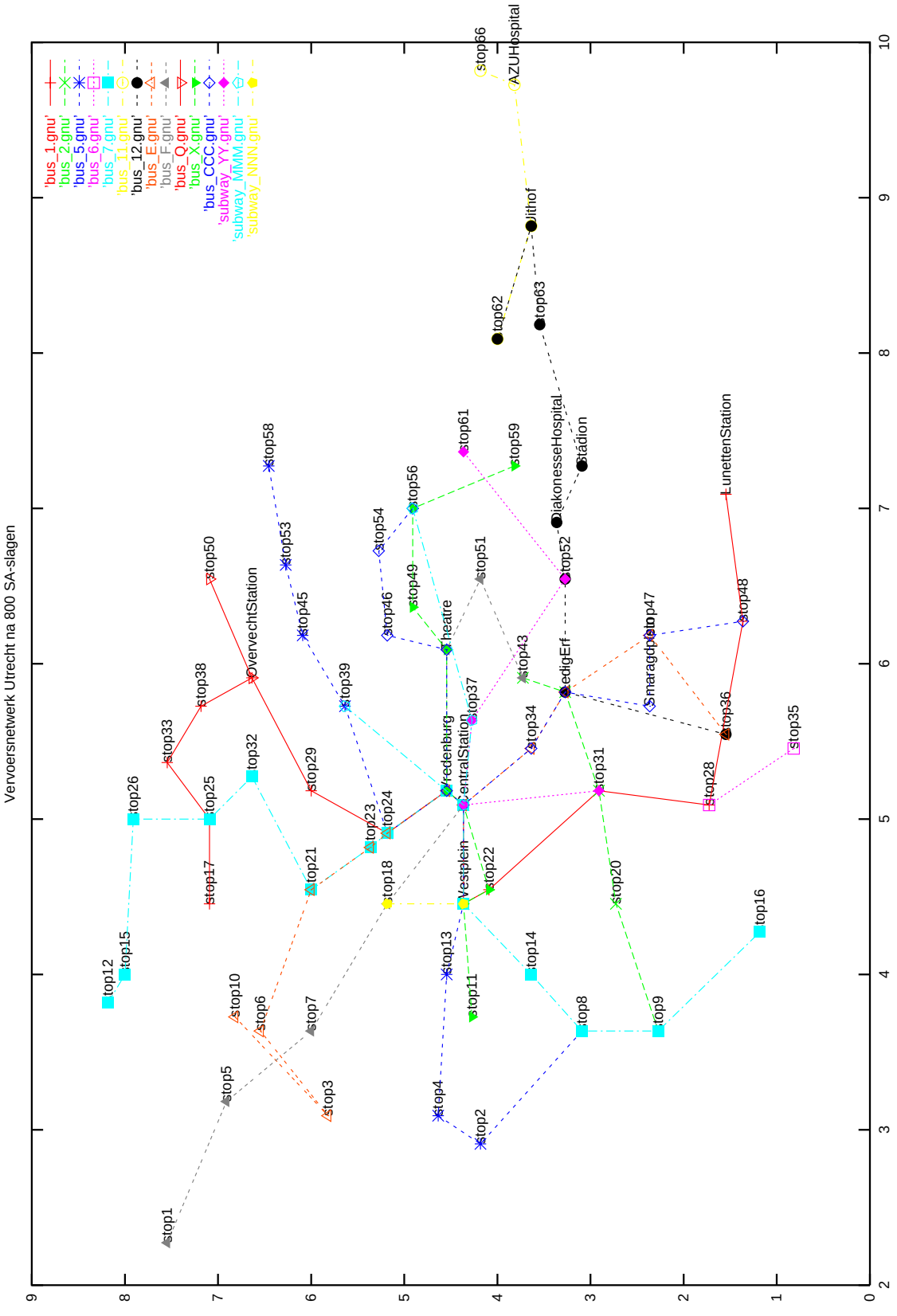
Tenslotte kunnen we nog een bepaalde aanname doen. Het huidige busnetwerk van de GVU is bijzonder functioneel en door professionals gepland. Er mag dus vanuit gegaan worden dat dit al (bijna) optimaal is. Het model wordt nu zó veranderd dat de buslijnen onaangetast blijven en er alleen met de metro geëxperimenteerd mag worden. Zie figuur 5. Hierbij werd een afname van 22% geboekt.

6 Conclusie

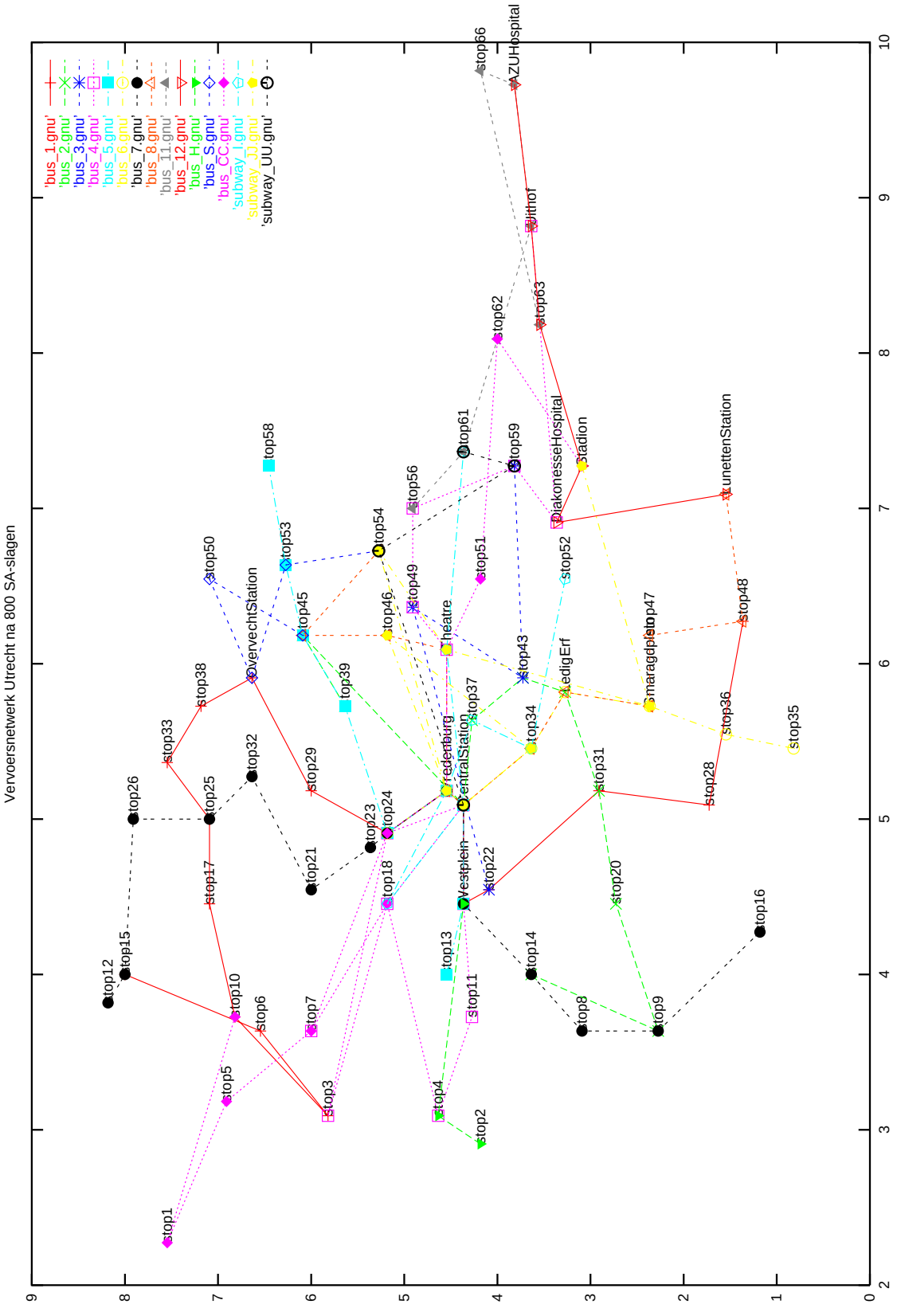
Het modelleren van een vervoersnetwerk blijkt niet zo'n groot probleem te zijn. Om echter een SA-probleem algemeen te formuleren, valt niet mee. Het geproduceerde programma, is dus heel goed uit te breiden voor nieuwe netwerken, maar de toepassing van de SA-methode hierop zal in dat geval moeten worden herzien.

Verder wat de optimalisering zelf betreft; er moet opgepast worden dat er niet te veel gekke dingen mogelijk zijn. Door de onduidelijkheid van de verschillende kosten, kan SA nog behoorlijk 'vreemde' vervoersnetwerken produceren. Het bestaande buslijnen-netwerk bleek moeilijk te verbeteren; SA stuurde het eerder in de war. Bij de metro bleek SA echter een effectieve methode voor het vinden van een traject.

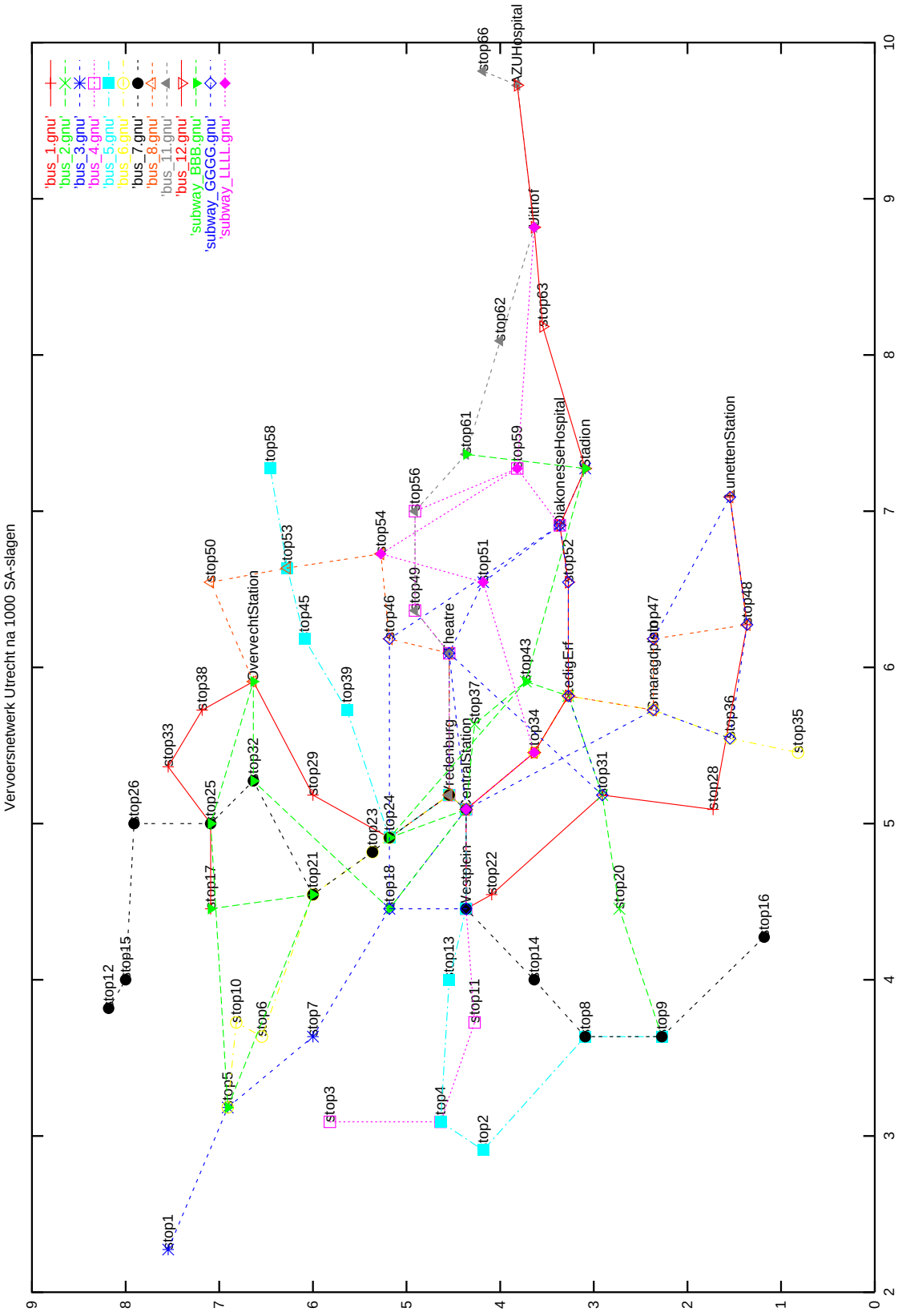
Tot slot nog over toekomstige onderzoeksmogelijkheden: er zijn in dit model nogal wat aannamen gedaan. Een aantal hiervan zou wat beter uitgewerkt kunnen worden. Zo zou in een nieuw programma de frequenties van de lijnen zo laag mogelijk kunnen worden gezet, om overcapaciteit te voorkomen. Ook kan de verdeling van de passagiers nog wel wat genuanceerder worden gedaan, alhoewel hierbij wel meer gegevens van het GVU nodig zijn.
Ook kan dit programma binnen een groter geheel gebruikt worden. Hierbij valt te denken aan meerdere **Network**-objecten, zoals de grote randstad-steden. Deze worden weer samengebracht door een overkoepelend netwerk, wat het streekvervoer tussen deze steden in kaart brengt.



Figuur 3: Vervoersnetwerk bij $\alpha = \beta = 0.5$



Figuur 4: Vervoersnetwerk bij $\alpha = 1, \beta = 0$



Figuur 5: Vervoersnetwerk bij $\alpha = 1, \beta = 0$, en busnetwerk onaangepast

A Bestaand busnetwerk van Utrecht