Inzet Brandweerauto's bij Calamiteiten

Bacheloropdracht Technische Wiskunde

Maike de Jongh

Begeleiders: Dr. Ir. M. de Graaf en Prof. Dr. M.N.M. van Lieshout

29-06-2018

Samenvatting

Met enige regelmaat vallen er in Twente incidenten voor waar de brandweer aan te pas moet komen. Een goede planning voor de inzet van brandweerauto's is dan ook van groot belang. In dit onderzoek bestudeer ik allereerst gevestigde modellen voor de planning van ambulancevervoer en de verschillen tussen de brandweer en ambulancediensten. Vervolgens ontwikkel ik een algemeen model voor de dynamische herpositionering van brandweerauto's dat gebaseerd is op Markovbeslissingstheorie. Op basis van actuele en toekomstige locaties van de voertuigen en incidenten bepaalt dit model of en op welke manier kazernes herbezet zouden moeten worden. Met behulp van data over onder andere vroegere branden pas ik tenslotte het model toe op de regio Twente.

Inhoudsopgave

1 Voorwoord

Na weken van puzzelen, programmeren en bezoekjes aan de brandweer ligt dan nu het onderzoek voor u waarmee ik mijn bachelor Technische Wiskunde afrond. Ik heb er met plezier aan gewerkt en erg veel van geleerd. Niet alleen zijn mijn modelleer- en programmeervaardigheden aangescherpt, maar ik ben ook nog meer overtuigd geraakt van de bijdrage die wiskunde kan leveren aan de samenleving.

Allereerst wil ik graag mijn begeleiders Maurits de Graaf en Marie-Colette van Lieshout bedanken voor hun waardevolle feedback de afgelopen weken. Daarnaast was dit onderzoek niet geslaagd zonder de medewerking van Veiligheidsregio Twente. Ik wil graag Emiel Sanders en Niels Peters bedanken voor hun behulpzaamheid en enthousiasme en hun uitgebreide toelichting bij de data. Ik hoop met de resultaten van dit onderzoek een wederdienst te hebben bewezen.

Veel leesplezier!

Maike de Jongh

Enschede, 29 juni 2018

2 Inleiding

Op een zekere avond slaat er in een restaurant in het centrum van Hengelo een vlam in de pan. Het vuur verspreidt zich in rap tempo en brandweerauto's uit talloze nabijgelegen kazernes rukken uit. De kazerne in Borne is vanwege dit incident tijdelijk onbemand. Als er op dit moment toevallig ook een brand uitbreekt in Borne moet de brandweer dus zijn toevlucht zoeken tot een andere kazerne. Het duurt dan een stuk langer voordat er een brandweereenheid ter plaatse is. Stel nu dat er een auto vanuit de kazerne in Almelo was verplaatst naar die in Borne zodra deze onbemand raakte, dan was het voorval veel sneller afgehandeld.
In dit project ontwikkel ik een model voor de optimale inzet van brandweerauto's bij calamiteiten. Daarbij laat me leiden door de volgende onderzoeksvraag:

Gegeven de actuele positionering van een verzameling brandweerauto's, data over de locaties en tijdstippen van vroegere branden en data over de bebouwing in Twente, hoe kunnen de brandweerauto's zodanig geherpositioneerd worden dat de fractie van het aantal oproepen dat binnen een zeker tijdslimiet bereikt wordt maximaal is?

Er zijn verschillende soortgelijke onderzoeken uitgevoerd voor de planning van ambulancevervoer. Waar ambulances vroeger vaak gebonden waren aan een bepaald ziekenhuis, worden ze tegenwoordig proactief ingezet. Dat wil zeggen dat de dekking van de regio in real-time wordt verbeterd door dynamische herpositionering van ambulances op basis van actuele en toekomstige locaties van de voertuigen en de incidenten. Bij het ontwikkelen van een model voor de inzet van brandweerauto's bij calamiteiten laat ik me inspireren door deze modellen voor ambulancevervoer. Deze zijn echter niet direct toepasbaar op brandweerauto's. In dit onderzoek bestudeer ik allereerst de gevestigde modellen voor de planning van ambulancevervoer en zoek ik uit in welke opzichten de brandweer verschilt van de ambulancediensten. Vervolgens ontwikkel ik een algemeen model voor de herpositionering van brandweerauto's dat gebaseerd is op Markovbeslissingstheorie. Tenslotte maak ik gebruik van gegevens over vroegere branden, ter beschikking gesteld door de brandweer, om dit model toe te passen op de regio Twente.

3 Literatuuronderzoek

Alvorens een model op te stellen voor de herpositionering van brandweerauto's laat ik me inspireren door gevestigde modellen voor de planning van ambulancevervoer. Er zijn talloze verschillende methoden bedacht om dit probleem aan te pakken. Een helder overzicht van enkele gangbare tactieken is gegeven in [1]. Hierin wordt beschreven wat proactieve planning van ambulancevervoer precies inhoudt en worden onder andere het zogenaamde "Maximum Coverage Location Problem", het "Maximum Availability Location Problem' en het "Maximum Expected Coverage Location Problem"besproken. Deze modellen maximaliseren de dekking van de regio door de ambulances. Ze gaan echter uit van de traditionele ambulancedienstverlening, waarbij iedere ambulance een vaste standplaats heeft. Naast deze modellen wordt in het artikel dynamisch ambulance management geïntroduceerd, waarbij ambulances kunnen verplaatsen naar een nieuwe basis. Er wordt een beknopt overzicht gegeven van een aantal online optimaliseringsalgoritmen, zoals het D-MEXCLP-algoritme, waarbij gebruik wordt gemaakt van de zogenaamde bezettingsgraad. Dit is de totale werklast gedeeld door de totale beschikbare capaciteit. Het D-MEXCLP-algoritme wordt uitgebreid omschreven in [7]. In eerste instantie worden alleen ambulances verplaatst die net klaar zijn met het afhandelen van een incident. Het algoritme berekent de optimale locatie voor deze ambulance op basis van de bestemmingen van andere werkloze ambulances. Dit is de locatie die de grootste toename in dekking oplevert volgens het MEXCLP-model, beschreven in [3]. Dit model is een lineair programmeer probleem, waarbij rekening wordt gehouden met het feit dat er soms niet genoeg ambulances beschikbaar zijn om aan de vraag te voldoen. In [3] wordt bovendien een heuristische oplossingsmethode gepresenteerd. Deze methode is gebaseerd op het feit dat er een unieke plek is vanuit waar het grootste gedeelte van de regio gedekt kan worden. De heuristiek plaatst eerst alle ambulances op deze plek en onderzoekt vervolgens in ieder stadium alternatieve locaties.
Ook in [5] wordt een lineair programmeer probleem beschreven voor het dynamisch herpositioneren van ambulances. De doelfunctie is zodanig gekozen dat het percentage van de ongelukken dat door minstens 2 ambulances binnen een bepaald tijdslimiet kan worden bereikt, wordt gemaximaliseerd.
In [6] wordt een model uiteengezet dat het aantal ambulances dat nodig is voor een bepaalde kwaliteit van dienstverlening berekent. Daarbij wordt rekening gehouden met willekeurigheid van reis- en servicetijden. In het model beschreven in [9] zijn deze tijden afhankelijk van het tijdstip van de dag.
Tenslotte wordt in [10] een dynamisch ambulance management model ontwikkeld dat is toegespitst op landelijke regio's, waar een beperkt aantal ambulances aanwezig zijn. De verdeling van ambulances wordt beschouwd als een systeem dat zich ieder tijdslot in een bepaalde toestand bevindt. Op basis van deze toestand en mogelijke nieuwe incidenten wordt de beste verdeling van ambulances geselecteerd.

Verschillen tussen brandweer en ambulancediensten

De beschreven modellen voor de planning van ambulancevervoer zijn niet direct toepasbaar op brandweerauto's. Er zijn een aantal fundamentele verschillen tussen de brandweer en de ambulancediensten. Ten eerste is het aantal incidenten waar de brandweer aan te pas moet komen significant kleiner dan het aantal incidenten waarvoor een ambulance moet uitrukken. Het is voor de brandweer dan ook een hele realistische aanname dat er niet meerdere incidenten tegelijkertijd plaatsvinden.
Daarnaast maakt de brandweer gebruik van meerdere typen brandweerauto's, zoals tankautospuiten, hoogwerkers, hulpverleningsvoertuigen en schuimblusvoertuigen. In de praktijk stuurt de brandweer er echter vrijwel alleen tankautospuiten op uit.
Tenslotte verschilt de cyclus die een brandweerauto doorloopt van die van een ambulance. Een brandweerauto keert immers na het afhandelen van een incident meteen terug naar een kazerne, terwijl een ambulance vaak eerst met een patiënt naar een ziekenhuis rijdt en pas daarna naar een nieuwe post wordt gestuurd.

4 Het Model

Voorgaande modellen voor de planning van ambulancevervoer zoeken veelal naar de beste nieuwe locatie voor een vrijgekomen ambulance. In dit onderzoek ontwikkel ik een strategie die alle vrije brandweerauto's in ogenschouw neemt. Daarbij maak ik gebruik van Markovbeslissingstheorie. In dit hoofdstuk construeer ik een complex en algemeen model waar in allerlei situaties gebruik van kan worden gemaakt. Later in het onderzoek zal ik dit model toepassen op de regio Twente.
Bij het modelleren maak ik gebruik van de volgende aannames:

1. De tijd die een brandweerauto nodig heeft om van een zeker punt naar een ander punt te rijden ligt vast en hangt niet af van het tijdstip t.
2. De tijd die een auto doorbrengt bij een incident hangt alleen af van het type auto.
3. De kansverdeling van het aantal incidenten in een tijdsinterval is altijd gelijk en niet afhankelijk van het tijdstip.

Allereerst verdeel ik het gebied in kwestie in vakken. De verzameling van deze vakken noem ik V. Laat N het totale aantal vakken voorstellen. De brandweer beschikt over een aantal kazernes, verspreid over de regio. De deelverzameling van V die bestaat uit vakken die een kazerne bevatten, noem ik W. Laat M het aantal kazernes voorstellen. Laat K het aantal typen auto's en B het totale aantal auto's voorstellen.

Toestanden

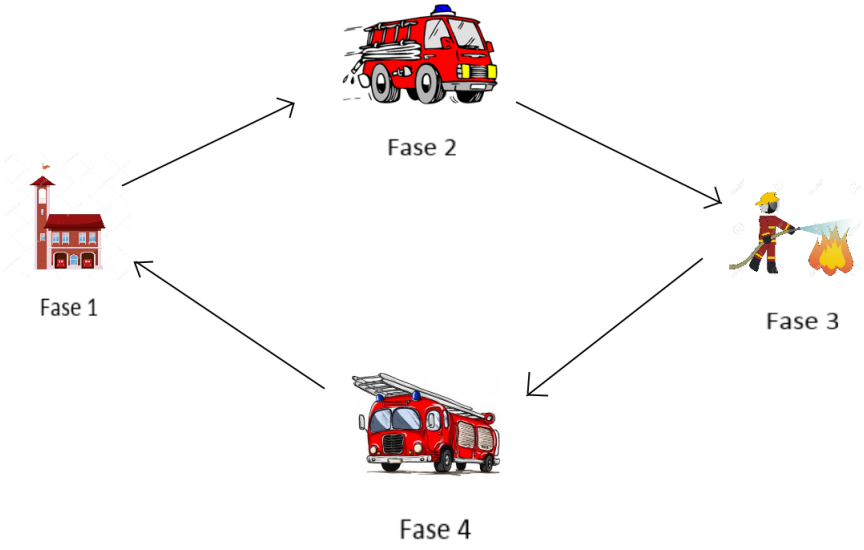
In het Markovbeslissingsprobleem beschouw ik de verdeling van de brandweerauto's over de regio als een systeem dat zich iedere tijdseenheid bevindt in een bepaalde toestand. De verzameling van mogelijke toestanden noem ik de toestandsruimte S. Een toestand s_t op tijdstip t geef ik weer met behulp van een aantal vectoren.

Iedere kazerne heeft slechts beperkte ruimte in zijn garage. Het maximum aantal auto's per kazerne geef ik weer door middel van vectoren $\mathbf{m}_k$ voor $1 \leq k \leq K$, waarbij $\mathbf{m}_k(i)$ het maximum aantal auto's van type k voorstelt dat de kazerne van vak $i \in W$ kan herbergen.
Het type van een auto sla ik op in een vector $\mathbf{n}$, waarbij $\mathbf{n}(i)$ het type van auto i voorstelt voor $1 \leq i \leq B$.
Om bij te houden waar een auto precies mee bezig is op een zeker moment definieer ik vier verschillende fasen:

- **Fase 1**: Auto staat werkloos bij een kazerne.
- **Fase 2**: Auto is op weg naar een incident.

- **Fase 3**: Auto is bezig met het afhandelen van een incident.
- **Fase 4**: Auto is op weg naar een kazerne.

De verschillende fasen zijn geïllustreerd in figuur 1.



Figuur 1: Fasen

De fase waarin een auto zich bevindt op een zeker tijdstip t, geef ik weer door middel van een vector $\mathbf{u}_t$, waarbij $\mathbf{u}_t(i) \in \{1, 2, 3, 4\}$ de fase van auto i voorstelt voor $1 \leq i \leq B$.
Wanneer een auto uitrukt naar een incident, rijdt hij na afloop ofwel door naar een volgend incident of keert hij terug naar een kazerne. Ik neem aan dat de auto als dat mogelijk is, terugkeert naar de kazerne waar hij vandaan is gekomen. Wanneer er bij deze kazerne geen plek meer is, ga ik ervan uit dat hij naar de dichtstbijzijnde kazerne rijdt die nog wel een een extra auto kan herbergen. Iedere auto is dus ieder moment verbonden aan een kazerne. Wanneer een auto zich in fase 1 bevindt, is dit de kazerne waar hij staat. Wanneer de auto zich in fase 2 of 3 bevindt, is dit de kazerne waar hij vandaan is gekomen. Als de auto zich in fase 4 bevindt, is dit kazerne waarnaar hij onderweg is. De kazerne waaraan een auto is verbonden op een zeker tijdstip t sla ik op in een vector $\mathbf{v}_t$, waarbij $\mathbf{v}_t(i)$ de kazerne voorstelt van auto i voor $1 \leq i \leq B$.
Iedere tijdseenheid kan er in ieder van de vakken een incident voorvallen. Deze incidenten zijn er in alle soorten en maten. Verschillende typen incidenten vragen om verschillende typen en verschillende aantallen brandweerauto's. Het aantal auto's van een zeker type k dat nodig is voor een incident dat voorvalt in tijdseenheid t geef ik weer in een vector $\mathbf{x}_{t,k}$, waarbij $\mathbf{x}_{t,k}(i)$ het aantal auto's

van type k voorstelt dat nodig is voor een incident bij vraagpunt $i \in V$.
Verder houd ik bij hoeveel auto's van type k er bij een bepaalde kazerne staan op een zeker tijdstip t. Deze aantallen geef ik weer in vectoren $\mathbf{y}_{t,k}$, waarbij $\mathbf{y}_{t,k}(i)$ het aantal auto's voorstelt bij kazerne $i \in W$.
Het aantal auto's van type k dat op een tijdstip t bezig is met het oplossen van een incident sla ik op in vectoren $\mathbf{b}_{t,k}$, waarbij $\mathbf{b}_{t,k}(i)$ het aantal auto's voorstelt dat bezig is bij vraagpunt $i \in V$.
Het aantal auto's van type k dat op tijdstip t op weg is naar een vraagpunt of naar een kazerne geef ik weer door middel van respectievelijk vectoren $\mathbf{w}_{t,k}$ en $\mathbf{z}_{t,k}$, waarbij $\mathbf{w}_{t,k}(i)$ het aantal auto's voorstelt van type k dat op weg is naar vraagpunt $i \in V$ en $\mathbf{w}_{t,k}(i)$ het aantal auto's voorstelt van type k dat op weg is naar kazerne $i \in W$.

De tijd die een brandweerauto nodig heeft om van een zeker punt naar een ander punt te rijden ligt vast en hangt niet af van het tijdstip t (aanname 1). Deze reistijden geef ik weer door middel van een NxN matrix T, waarbij $T(i,j)$ de tijd voorstelt die een auto nodig heeft om van punt i naar punt j te rijden. Daarnaast is de tijd die nodig is om een incident op te lossen alleen afhankelijk van het type auto (aanname 2). Deze tijden sla ik op in een vector $\mathbf{r}$, waarbij $\mathbf{r}(k)$ de tijd voorstelt die een auto van type k nodig heeft om een incident op te lossen. De tijd die een auto op een zeker tijdstip t nog moet besteden aan een incident houd ik bij in een matrix H_t, waarbij $H_t(i,j)$ de resterende tijd voorstelt die auto j, $1 \leq j \leq B$, nog nodig heeft voor een incident bij vraagpunt $i \in V$.
Op soortgelijke wijze houd ik de tijd bij die een auto op een zeker tijdstip t nog nodig heeft om naar een zekere plek te rijden bij in een matrix Q_t, waarbij $Q_t(i,j)$ de resterende reistijd voorstelt voor auto j, $1 \leq j \leq B$ naar vraagpunt $i \in V$.

Een toestand s_t op tijdstip t in het Markovbeslissingsprobleem wordt gegeven door $s_t = (\mathbf{u}_t, \mathbf{v}_t, \mathbf{x}_t, \mathbf{y}_t, \mathbf{b}_t, \mathbf{w}_t, \mathbf{z}_t, H_t, Q_t)$, waarbij $\mathbf{x}_t = (\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, ..., \mathbf{x}_{t,K})$, $\mathbf{y}_t = (\mathbf{y}_{t,1}, \mathbf{y}_{t,2}, ..., \mathbf{y}_{t,K})$, $\mathbf{b}_t = (\mathbf{b}_{t,1}, \mathbf{b}_{t,2}, ..., \mathbf{b}_{t,K})$, $\mathbf{w}_t = (\mathbf{w}_{t,1}, \mathbf{w}_{t,2}, ..., \mathbf{w}_{t,K})$ en $\mathbf{z}_t = (\mathbf{z}_{t,1}, \mathbf{z}_{t,2}, ..., \mathbf{z}_{t,K})$.

Beslissingen

Afhankelijk van de toestand van het systeem op een zeker moment zijn er verschillende acties mogelijk. De verzameling van mogelijke beslissingen wanneer het systeem zich bevindt in een zekere toestand s_t noem ik $D(s_t)$. Een beslissing d wordt gegeven door $d = (A_1, A_2, ..., A_K)$, waarbij A_k voor $1 \leq k \leq K$ een MxM matrix is en $A_k(i,j)$ het aantal auto's voorstelt dat moet worden verplaatst van kazerne $i \in W$ naar kazerne $j \in W$. Vanzelfsprekend kunnen er hooguit $\mathbf{y}_{t,k}(i)$ auto's van type k worden verplaatst vanuit punt $i \in W$. Meer auto's staan er immers niet bij deze kazerne. Bovendien kan een kazerne van punt $j \in W$ niet meer dan $\mathbf{m}_k(j)$ auto's herbergen. Aangezien er op tijdstip

t nog $\mathbf{z}_{t,k}(j)$ auto's van type k onderweg zijn naar vak $j \in W$ kunnen er dus hooguit $\mathbf{m}_k(j) - \mathbf{z}_{t,k}(j) - \mathbf{y}_{t,k}(j)$ auto's van type k naar vak j worden gestuurd. Kortom, voor ieder vak $i \in W$ en ieder type k

$$\sum_{j=0}^{M} A_k(i,j) \leq \mathbf{y}_{t,k}(i) \tag{1}$$

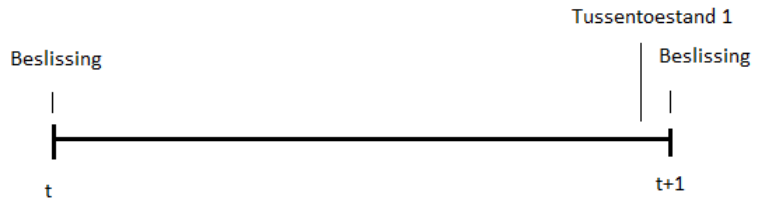
en voor ieder vak $j \in W$ en ieder type k

$$\sum_{i=0}^{M} A_k(i,j) \leq \mathbf{m}_k(j) - \mathbf{z}_k(j) - \mathbf{y}_k(j). \tag{2}$$

Eerste Tussentoestand

Wanneer er een incident voorvalt bij een zeker vraagpunt wil de brandweer er natuurlijk voor zorgen dat er zo snel mogelijk voldoende auto's ter plaatse zijn. Om te achterhalen vanuit welke kazerne de brandweer het beste te hulp kan schieten, definieer ik een functie f die een vraagpunt $i \in V$ afbeeldt op een permutatie van de elementen van W. De volgorde van deze elementen geeft aan welke kazernes worden verkozen boven andere kazernes. Stel bijvoorbeeld dat $f(i) = (j_1, j_2, ..., j_M)$ met $i \in V$ en $j_1, j_2, ..., j_M \in W$. Dit wil zeggen dat er in het geval van een incident bij vraagpunt i bij voorkeur auto's erop uit zullen worden gestuurd vanuit kazerne j_1. Als dit niet meer mogelijk is, zal de brandweer zijn toevlucht zoeken tot kazerne j_2 en zo verder.

Het verplaatsen van auto's die werkloos bij een kazerne staan, gebeurt alleen direct aan het begin van een tijdseenheid.
Laat $s'(t) = (\mathbf{u'}_t, \mathbf{v'}_t, \mathbf{x'}_t, \mathbf{y'}_t, \mathbf{b'}_t, \mathbf{w'}_t, \mathbf{z'}_t, H'_t, Q'_t)$ een tussentoestand voorstellen waarin het systeem zich bevindt vlak voor het begin van tijdseenheid $t+1$. Dit is geïllustreerd in figuur 2.



Figuur 2: Tijdsbalk

Op dit moment zijn er auto's op weg gestuurd naar incidenten die zijn voorgevallen in tijdseenheid t. De functie f bepaalt welke auto's dit zijn. Een auto die uitrukt gaat van fase 1 naar fase 2:

$$\mathbf{u'}_t(i) = \begin{cases} \mathbf{u}_t(i) \text{ als } \mathbf{u}_t(i) \neq 1 \\ 1 \text{ als } \mathbf{u}_t(i) = 1 \text{ en auto } i \text{ niet uitgerukt} \\ 2 \text{ als } \mathbf{u}_t(i) = 1 \text{ en auto } i \text{ uitgerukt} \end{cases} \quad \text{voor } 1 \leq i \leq B. \tag{3}$$

De auto's horen nog bij dezelfde kazerne, dus

$$\mathbf{v'}_t(i) = \mathbf{v}_t(i) \text{ voor } 1 \leq i \leq B. \tag{4}$$

Laat $\mathbf{h}_{t,k}(i,j)$ het aantal auto's van type k voorstellen dat vanuit kazerne $i \in W$ is uitgerukt naar vraagpunt $j \in V$ in tijdseenheid t. Het aantal auto's dat nog vereist is bij een vraagpunt $i \in V$ aan het eind van tijdseenheid t is gelijk aan het aantal auto's benodigd aan het begin van dit tijdsvak minus het totaal aantal auto's dat is uitgerukt naar dit incident. Dus

$$\mathbf{x'}_{t,k}(i) = \mathbf{x}_{t,k}(i) - \sum_{j \in W} \mathbf{h}_{t,k}(j,i) \text{ voor } 1 \leq k \leq K \text{ en } i \in V. \tag{5}$$

Het aantal auto's dat overblijft bij een zekere kazerne $i \in W$ wordt gegeven door

$$\mathbf{y'}_{t,k}(i) = \mathbf{y}_{t,k}(i) - \sum_{j \in V} \mathbf{h}_{t,k}(i,j) \text{ voor } 1 \leq k \leq K \text{ en } i \in W. \tag{6}$$

Tenslotte is het aantal auto's dat op dit moment onderweg is naar vraagpunt $i \in V$ gelijk aan

$$\mathbf{w'}_{t,k}(i) = \mathbf{w}_{t,k}(i) + \sum_{j \in W} \mathbf{h}_{t,k}(j,i) \text{ voor } 1 \leq k \leq K \text{ en } i \in V. \tag{7}$$

Het aantal auto's dat bezig is bij een zeker vraagpunt en het aantal auto's dat op weg is naar een kazerne veranderen niet, dus

$$\mathbf{b'}_{t,k}(i) = \mathbf{b}_{t,k}(i) \text{ voor } 1 \leq k \leq K \text{ en } i \in V \tag{8}$$

en

$$\mathbf{z'}_{t,k}(i) = \mathbf{z}_{t,k}(i) \text{ voor } 1 \leq k \leq K \text{ en } i \in V. \tag{9}$$

De matrix H wordt bijgewerkt aan het begin van een tijdvak, dus

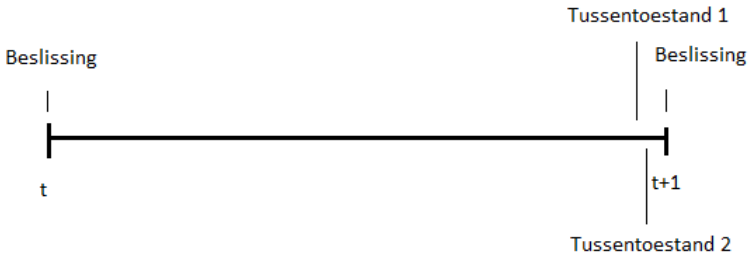
$$H'_t(i,j) = H_t(i,j) \text{ voor } i \in V \text{ en } 1 \leq j \leq B. \tag{10}$$

De tijd die een auto j, $1 \leq j \leq B$ nodig heeft om naar een vraagpunt $i \in V$ te rijden is gelijk aan $T(\mathbf{v}_t(j), i)$. Dit getal rond ik naar boven af naar het eerste gehele aantal tijdseenheden. Bovendien begin ik pas met aftellen aan het begin van het eerstvolgende tijdslot. Aangezien de matrix Q dan ook bijgewerkt wordt, tel ik er een enkele tijdseenheid extra bij op:

$$Q'_t(i,j) = \begin{cases} \lceil T(\mathbf{v}_t(j), i) \rceil + 1 \text{ als auto } j \text{ uitrukt naar vraagpunt } i \\ Q_t(i,j) \text{ anders} \end{cases} \tag{11}$$
$$\text{voor } i \in V \text{ en } 1 \leq j \leq B.$$

Tweede Tussentoestand

Het is mogelijk dat er niet voldoende auto's beschikbaar zijn om aan de vraag te voldoen. Dit is het geval als $\mathbf{x}'_{t,k}(i) > 0$ voor een zeker vraagpunt $i \in V$, tijdstip t en type k. In deze situatie zullen auto's die hun dienst voltooien bij een zeker incident niet terugkeren naar een kazerne, maar direct doorrijden naar een volgend incident. Uiteraard zal er als eerst een beroep gedaan worden op de vrijgekomen auto's die zich het dichtst in de buurt van het incident bevinden. Laat $s"(t) = (\mathbf{u}"_t, \mathbf{v}"_t, \mathbf{x}"_t, \mathbf{y}"_t, \mathbf{b}"_t, \mathbf{w}"_t, \mathbf{z}"_t, P"_t, Q"_t)$ een tussentoestand voorstellen waarin het systeem zich bevindt tussen het moment waarop het zich in de eerste tussentoestand bevindt en het begin van tijdsvak $t+1$. Dit is geïllustreerd in figuur 3.



Figuur 3: Tijdsbalk

Auto's die hun dienst hebben voltooid in tijdsvak t zullen nu indien nodig op weg zijn naar een nieuw incident. Er geldt

$$\mathbf{u}"_t(i) = \begin{cases} 2 \text{ als auto } i \text{ direct doorrijdt na afhandelen incident} \\ \mathbf{u}'_t(i) \text{ anders} \end{cases} \quad \text{voor } 1 \leq i \leq B. \tag{12}$$

Alle auto's blijven verbonden aan dezelfde kazerne, dus

$$\mathbf{v}"_t(i) = \mathbf{v}'_t(i) \text{ voor } 1 \leq i \leq B. \tag{13}$$

Laat $\mathbf{g}_{t,k}(i,j)$ het aantal auto's voorstellen van type k dat aan het einde van tijdvak t direct doorrijdt van een incident bij vraagpunt $i \in V$ naar een incident bij vraagpunt $j \in V$. Het aantal auto's dat nog vereist is bij een vraagpunt $i \in V$ is gelijk aan het aantal auto's benodigd na de eerste tussentoestand minus het totaal aantal auto's dat op dit moment naar het incident vertrekt. We krijgen

$$\mathbf{x}"_{t,k}(i) = \mathbf{x}'_{t,k}(i) - \sum_{j \in V} \mathbf{g}_{t,k}(j,i) \text{ voor } i \in V. \tag{14}$$

Het aantal auto's dat bezig is bij een vraagpunt wordt nu

$$\mathbf{b}"_{t,k}(i) = \mathbf{b}'_{t,k}(i) - \sum_{j \in V} \mathbf{g}_{t,k}(i,j) \text{ voor } i \in V \tag{15}$$

en het aantal auto's dat onderweg is naar een vraagpunt wordt gegeven door

$$\mathbf{w}"_{t,k}(i) = \mathbf{w}'_{t,k}(i) + \sum_{j \in V} \mathbf{g}_{t,k}(j,i) \text{ voor } i \in V. \tag{16}$$

Bovendien geldt

$$Q"_t(i,j) = \begin{cases} \lceil T(l,i) \rceil + 1 \text{ als auto } j \text{ van vraagpunt } l \text{ naar vraagpunt } i \text{ rijdt.} \\ Q'_t(i,j) \text{ anders} \end{cases} \quad \text{voor } i \in V \text{ en } 1 \leq j \leq B. \tag{17}$$

en $\mathbf{y}"_{t,k} = \mathbf{y}'_{t,k}$, $\mathbf{z}"_{t,k} = \mathbf{z}'_{t,k}$ en $H"_t = H'_t$.

Overgangskansen

Aan het begin van tijdseenheid $t+1$ wordt een bepaalde actie $d = (A_1, A_2, ..., A_k)$ uit de beslissingsruimte $D(s_t)$ gekozen. Afhankelijk van de toestand s_t van het systeem op tijdstip t, de genomen beslissing $d \in D(s)$ en mogelijke nieuwe incidenten in het volgende tijdvak komt het systeem op tijdstip $t+1$ in een nieuwe toestand s_{t+1} terecht. Laat $p(s_t|s_{t-1}, d)$ de kans voorstellen dat het systeem zich op een tijdstip t in toestand s_t bevindt gegeven dat het systeem zich de tijdseenheid daarvoor in toestand s_{t-1} bevond en beslissing d is genomen.

Ik neem aan dat incidenten in ieder vak voorvallen volgens een Poisson proces met parameter λ_i voor $i \in V$. Aangezien de kans op het voorvallen van een incident niet verandert (aanname 3), hangt de kansverdeling van de toename van het aantal incidenten in een bepaald tijdsinterval alleen af van de lengte van het tijdsinterval. De toename van het aantal incidenten is dus stationair. Bovendien is het aantal incidenten dat is voorgevallen in een bepaald tijdsinterval niet afhankelijk van het aantal incidenten in een ander niet-overlappend tijdsinterval. Een Poisson proces is dus een geschikt model voor het voorvallen van incidenten. Dit impliceert dat de kans op n incidenten bij punt $i \in V$ in een enkele tijdseenheid wordt gegeven door

$$P(n \text{ incidenten in vak } i) = \frac{\lambda_i^n}{n!} e^{-\lambda_i} \text{ voor } i \in V,\ n = 0, 1, 2, ... \tag{18}$$

Laat $P_k(n)$ de kans voorstellen dat er voor een zeker incident n auto's van type k nodig zijn voor $1 \leq k \leq K$. Hieruit volgt dat voor tijdseenheid t, vraagpunt $i \in V$ en type k, $1 \leq k \leq K$, geldt

$$\begin{aligned} P(\mathbf{x}_{t,k}(i) - \mathbf{x}"_{t-1,k}(i) = n) = \sum_{j=0}^{\infty} P(j \text{ incidenten in vak } i) \cdot \\ P(\mathbf{x}_{t,k}(i) - \mathbf{x}"_{t-1,k}(i) = n \mid j \text{ incidenten in vak } i) = \\ \sum_{j=0}^{\infty} \frac{\lambda_i^j}{j!} e^{-\lambda_i} \sum_{m_1+m_2+...+m_j=n} \prod_{z=1}^{j} P_k(m_z). \end{aligned} \tag{19}$$

Ik neem aan dat bij het herpositioneren van werkloze auto's een auto i eerder zal worden verplaatst dan een auto j wanneer $i < j$. Er geldt

$$\mathbf{u}_{t+1}(i) = \begin{cases} 4 \text{ als } \mathbf{u}"_t(i) = 1 \text{ en auto } i \text{ wordt verplaatst} \\ 3 \text{ als } \mathbf{u}"_t(i) = 2 \text{ en } Q"_t(j,i) = 1 \text{ voor een zekere } j \in V \\ 4 \text{ als } \mathbf{u}"_t(i) = 3 \text{ en } H"_t(j,i) = 1 \text{ voor een zekere } j \in V \\ 1 \text{ als } \mathbf{u}"_t(i) = 4 \text{ en } Q"_t(j,i) = 1 \text{ voor een zekere } j \in V \\ \mathbf{u}"_t(i) \text{ anders} \end{cases} \quad \text{voor } 1 \leq i \leq B \tag{20}$$

en

$$\mathbf{v}_{t+1}(i) = \begin{cases} j \text{ als auto } i \text{ wordt verplaatst naar kazerne } j \\ \mathbf{v}"_t(i) \text{ anders} \end{cases} \quad \text{voor } 1 \leq i \leq B. \tag{21}$$

Gedurende tijdseenheid t zijn er auto's vertrokken van en gearriveerd bij kazerne $i \in W$, dus

$$\mathbf{y}_{t+1,k}(i) = \mathbf{y}"_{t,k}(i) - \sum_{j \in W} A(i,j) + |\{j|Q"_t(i,j) = 1\}| \text{ voor } i \in W. \tag{22}$$

Auto's die arriveren bij een vraagpunt gaan direct aan de slag en auto's die klaar zijn met het afhandelen van een incident vertrekken ofwel naar een ander incident of naar een kazerne. Dit geeft

$$\mathbf{b}_{t+1,k}(i) = \mathbf{b}"_{t,k}(i) - |\{j|H"_t(i,j) = 1\}| + |\{j|Q"_t(i,j) = 1\}| \text{ voor } i \in V. \tag{23}$$

Er worden op het exacte moment van de beslissing geen nieuwe auto's naar incidenten gestuurd, dus

$$\mathbf{w}_{t+1,k}(i) = \mathbf{w}"_{t,k}(i) - |\{j|Q"_t(i,j) = 1\}| \text{ voor } i \in V. \tag{24}$$

Er worden auto's op weg gestuurd naar kazerne i en andere auto's zijn daar inmiddels gearriveerd:

$$\mathbf{z}_{t+1,k}(i) = \mathbf{z}"_{t,k}(i) + \sum_{j \in W} A(j,i) - |\{j|Q"_t(i,j) = 1\}| \text{ voor } i \in W. \tag{25}$$

Tenslotte worden de matrices met de tijden bijgewerkt:

$$H_t(i,j) = \begin{cases} H"_t(i,j) - 1 \text{ als } H"_t(i,j) > 0 \\ 0 \text{ anders} \end{cases} \quad \text{voor } i \in V \text{ en } 1 \leq j \leq B \quad (26)$$

en

$$Q_t(i,j) = \begin{cases} Q"_t(i,j) - 1 \text{ als } Q"_t(i,j) > 0 \\ 0 \text{ anders} \end{cases} \quad \text{voor } i \in V \text{ en } 1 \leq j \leq B. \quad (27)$$

Kosten

Afhankelijk van de toestand van het systeem wordt ieder tijdsvak een zekere hoeveelheid kosten gerekend. Laat $c(s_t)$ de kosten voorstellen voor een enkel tijdslot t wanneer het systeem zich in toestand s_t bevindt. Aangezien het voornaamste doel van de brandweer is om zo snel mogelijk de locatie van een incident te bereiken, dienen deze kosten af te hangen van de responstijden. Laat ϕ een kostenfunctie voorstellen die de reistijd van een brandweerauto naar een zeker incident afbeeldt op de bijbehorende kosten. Wanneer er dus een auto vanuit een kazerne $i \in W$ uitrukt naar een incident bij vraagpunt $j \in V$, wordt er $\phi(T(i,j))$ aan kosten gerekend. De kosten voor een zeker tijdslot t waarin het systeem zich bevindt in toestand $s_t = (\mathbf{u}_t, \mathbf{v}_t, \mathbf{x}_t, \mathbf{y}_t, \mathbf{b}_t, \mathbf{w}_t, \mathbf{z}_t, H_t, Q_t)$ definieer ik als

$$c(s_t) = \sum_{i \in V} \sum_{j=1}^{B} (\phi(Q_t(i,j)) - \phi(\max(Q_t(i,j) - 1, 0))). \quad (28)$$

Een auto j die onderweg is naar vraagpunt $i \in V$ draagt dan gedurende tijdslot t een hoeveelheid $\phi(Q_t(i,j)) - \phi(\max(Q_t(i,j) - 1, 0))$ aan kosten bij. De kosten die toe te schrijven zijn aan een auto die vanuit een kazerne $i \in W$ uitrukt naar een incident bij vraagpunt $j \in V$ zijn dan

$$\sum_{k=1}^{T(i,j)} (\phi(k) - \phi(k-1)) = \phi(T(i,j)). \quad (29)$$

Er zijn vele verschillende opties mogelijk voor de kostenfunctie ϕ. Aangezien de brandweer de responstijden wil minimaliseren, dient $\phi(t)$ in ieder geval een niet-dalende functie te zijn waarvoor bovendien geldt $\phi(0) = 0$. Als het doel is om de gemiddelde responstijd te minimaliseren, zou

$$\phi(t) = t \text{ voor } t \geq 0 \quad (30)$$

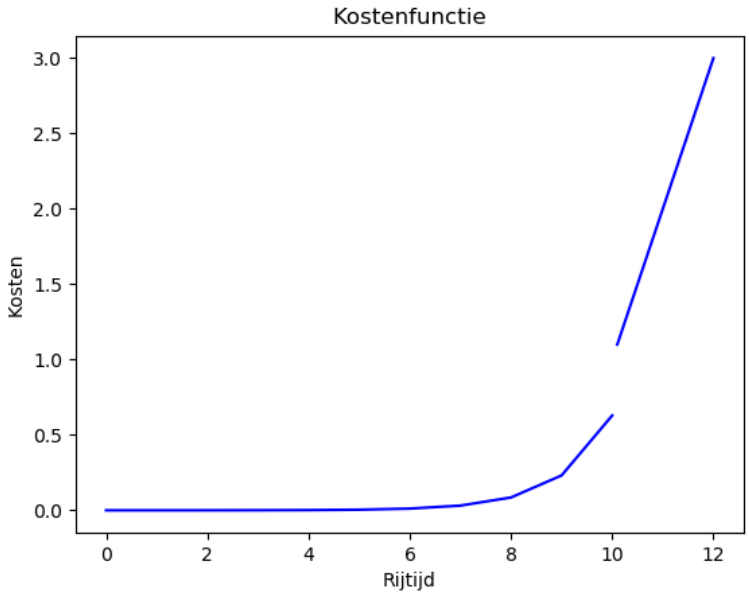
een logische keuze zijn. Deze kostenfunctie zal leiden tot een kleine gemiddelde responstijd. De variantie in de responstijden kan echter nog heel groot zijn. Wanneer de brandweer zich ten doel stelt om incidenten te bereiken binnen een bepaald tijdslimiet T^*, zou de kostenfunctie

$$\phi(t) = \begin{cases} 0 \text{ als } 0 \leq t \leq T^* \\ 1 \text{ als } t > T^* \end{cases} \tag{31}$$

een plausibele keuze zijn. Hierbij wordt echter geen onderscheid gemaakt tussen een hele korte responstijd en een responstijd die net binnen de tijdslimiet valt. Ook responstijden die vlak boven de tijdslimiet vallen en responstijden die veel te lang zijn leveren dezelfde hoeveelheid kosten op. Om deze nadelen te omzeilen, maak ik gebruik van de kostenfunctie geformuleerd door van Barneveld, Bhulai en van der Mei (2015):

$$\phi(t) = \begin{cases} \frac{1}{\gamma}(e^t - 1) \text{ voor } 0 \leq t \leq T^* \\ t - (T^* - 1) \text{ voor } t > T^*, \end{cases} \tag{32}$$

waarin γ een schalende parameter voorstelt. Deze functie, gegeven in figuur 4 voor $\gamma = 35000$ en $T^* = 10$, stijgt geleidelijk, maakt een sprongetje bij T^* en stijgt vervolgens constant verder.



Figuur 4: Kostenfunctie

Voor de tijdslimiet zijn de kosten voor een langere responstijd dus flink hoger dan voor een hele korte responstijd. Het sprongetje bij T^* zorgt voor verschil in kosten tussen responstijden binnen en buiten de tijdslimiet. Tenslotte wordt boven T^* de gemiddelde overschrijding van het tijdslimiet geminimaliseerd.

Het doel van het Markovbeslissingsprobleem is het vinden van een beleid δ in een beleidsruimte Δ dat de verwachte verdisconteerde kosten gedurende een oneindig aantal perioden minimaliseert. Laat $V_\delta(s)$ de verwachte verdisconteerde kosten voorstellen gedurende een oneindig aantal perioden onder beleid δ gegeven dat het systeem zich aan het begin van periode 1 in toestand s bevindt. Laat

$$V(s) = \min_{\delta \in \Delta} V_\delta(s). \tag{33}$$

Ik ga op zoek naar een optimaal beleid δ^* dat dit minimum realiseert, oftewel waarvoor geldt

$$V(s) = V_{\delta^*}(s). \tag{34}$$

Een dergelijk probleem kan worden opgelost met behulp van de optimaliteitsvergelijkingen. Deze zien er als volgt uit:

$$V_\delta(s) = c(s) + \beta \sum_{i \in S} p(i|s, \delta(s)) V_\delta(i) \text{ voor } s \in S, \tag{35}$$

waarbij β een disconteringsfactor voorstelt.

Heuristiek

Het oplossen van een stelsel optimaliteitsvergelijkingen is voor omvangrijke problemen een enorm karwei. De rekentijd zal buiten de perken treden. Daarom zijn we in veel gevallen genoodzaakt onze toevlucht te zoeken tot heuristische methoden. Om een geschikte actie te selecteren gegeven de huidige toestand van het systeem, maak ik gebruik van een soortgelijke heuristische methode als die beschreven door van Barneveld, Bhulai en van der Mei (2015) . Laat $R_i(s_t, d)$ de responstijd voorstellen voor een incident bij vraagpunt $i \in V$ wanneer het systeem zich in toestand s_t bevindt en beslissing d is genomen aan het begin van tijdslot t. Als er gedurende tijdslot t geen incident voorvalt bij vraagpunt i geldt $R_i(s_t, d) = 0$. Voor iedere d in beslissingsruimte $D(s_t)$ bereken ik de verwachte kosten $V(d)$ voor eventuele nieuwe incidenten in tijdslot $t + 1$:

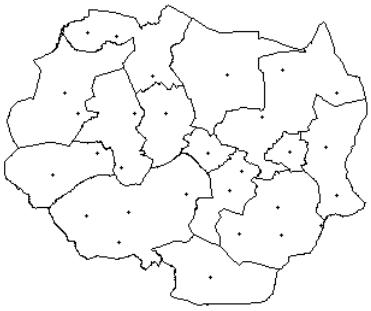
$$V(d) = \sum_{s^* \in S} (\sum_{i \in V} \phi(R_i(s^*, d)) P(s_{t+1} = s^* | s_t). \tag{36}$$

Vervolgens selecteer ik de beslissing d^* die deze verwachte kosten minimaliseert:

$$d^* = \arg\min_{d \in D(s_t)} V(d). \tag{37}$$

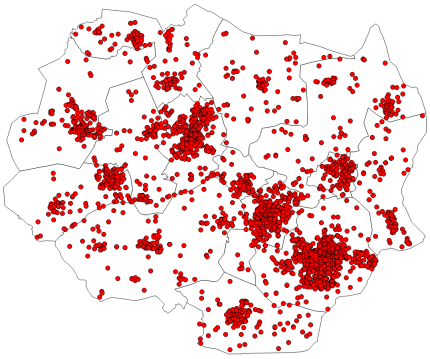
5 Beschikbare Data

Om het model toe te passen maak ik gebruik van data ter beschikking gesteld door Veiligheidsregio Twente. De brandweer opereert in Twente vanuit 29 kazernes, verspreid over de regio. De locaties zijn weergegeven als zwarte puntjes in figuur 5.



Figuur 5: Locaties brandweerkazernes

Er zijn gegevens beschikbaar over incidenten van 2004 tot 2016. Voor ieder incident is het tijdstip en de locatie opgetekend. Daarnaast is voor ieder incident het type brand geregistreerd en zijn de branden onderverdeeld in de categorieën 'geen actie brandweer', 'kleine brand', 'middelbrand', 'grote brand' en 'zeer grote brand'. In figuur 6 zijn alle incidenten die hebben plaatsgevonden in 2008 weergegeven.



Figuur 6: Alle incidenten in 2008

De brandweer heeft bovendien de regio verdeeld in vakken van 500 meter bij 500 meter. Deze vakken gebruik ik voor de verzameling V. Om de verzame-

ling W van kazernes te bepalen, stel ik voor iedere kazerne aan de hand van de coördinaten vast in welk van de vakken deze ligt. De code die ik voor dit doeleinde heb geschreven is te vinden in Appendix A.

Voor incidenten in de buurt van de grenzen van Twente wordt soms de hulp ingeroepen van kazernes buiten Twente. Bijvoorbeeld uit Gelderland of Duitsland. Deze voeg ik niet toe aan de verzameling W. Ik neem dus aan dat er geen auto's verplaatst worden van en naar deze kazernes.

Hoewel de brandweer gebruik maakt van verschillende typen auto's worden voor verreweg de meeste incidenten slechts tankautospuiten ingezet. Iedere kazerne heeft in principe plaats voor één tankautospuit. Een voorbeeld van de verdeling van brandweerauto's over de kazernes is weergegeven in figuur 7.

Cat	ALMELO	BORNE	DELUTT	DELDEN	DENHOV	DENEKP	DIEPHM	ENSCHDBK	ENSCHDGL	ENSCHDHP	ENSCHDN	ENTER	GOOR	HAAKSB	HENGOVC
OVERIG	053111 053131 053151 053171 059095 059097	051331 051361	052541	051541 051562	053431 053462 053479	052341 052362	051731 051779	054431	054331	054131 054152 054171 054181 059036	054231 054251	055631 055662	051431 051461	054541 054561 054563 054579 054581	051081 051111 051131 051151 051171 051181 059094

Cat	HELLDN	HENGOVN	HOLTEN	LOSSER	MARKLO	NIJVDL	OLZAAL	OOTMSM	RIJSSN	TUBBRG	VRIEZV	VROOMH	WEERSL	WIERDN	05REG
OVERIG	055341 055362 055379	051231 051263 051281	055241 055261 055262 055279	052231 052261	051641 051662 051679	055441 055461 055463 055481	052131 052151 052163 052171 052181 053181 059096	052641 052663 052681	055141 055151 055163 055179 055181 059093	052431 052441 052461 052462 052479	053631 053662 053663 053681	053531 053561	052731 052761	055541 055562 055563 055579 055581	05GG01 05GG02 05GG03 05GG04 05GG05 05GG06 05GG07 05GG08 05GG09 05GG10 05GG11 05GG12

Figuur 7: Verdeling van brandweerauto's over kazernes

Iedere kolom geeft weer welke auto's er in de garage van een bepaalde kazerne aanwezig zijn. De auto's zijn weergegeven door middel van een 6-cijferige code. De eerste twee cijfers geven aan bij welke regio de auto in kwestie hoort. De cijfers '05' staan voor de regio Twente. De laatste twee cijfers geven het type van de auto aan. De cijfers '31' en '41' staan voor tankautospuiten. Een auto met code '31' is een gewone tankautospuit en een auto met type '41' is een tankautospuit die in staat is om door het bos te rijden. In mijn berekeningen maak ik geen onderscheid tussen deze twee typen tankautospuiten. De overige typen brandweerauto's laat ik buiten beschouwing.

Om het probleem behapbaar te maken, ga ik ervan uit dat er in totaal slechts 1 incident kan voorvallen per tijdvak. Aangezien er zelden meerdere incidenten tegelijkertijd voorvallen waar de brandweer aan te pas moet komen, is dit een hele realistische aanname. Daarnaast neem ik aan dat er voor een incident slechts 1 brandweerauto nodig is. De kans dat er in een zeker tijdvak een incident voorvalt bij vraagpunt $i \in V$ wordt dan gegeven door

$$
\begin{aligned}
&P(\text{incident bij } i|\text{ max. 1 incident per tijdslot}) = \\
&\frac{P(\text{incident bij } i,\text{ max. 1 incident per tijdslot })}{P(\text{max. 1 incident per tijdslot})} = \\
&\frac{\lambda_i e^{-\lambda_i} \prod_{j\neq i} e^{-\lambda_j}}{\prod_j e^{-\lambda_j} + \sum_k \lambda_k e^{-\lambda_k} \prod_{j\neq k} e^{-\lambda_j}} = \frac{\lambda_i}{1+\sum_{j=1}^{N} \lambda_j}.
\end{aligned} \tag{38}
$$

Met behulp van de data over de incidenten bepaal ik de waarden van λ_i, oftewel het gemiddeld aantal incidenten per tijdvak in vak $i \in V$. Ik hanteer tijdvakken van een minuut. De code waarmee ik λ_i bereken is te vinden in Appendix B. Voor ieder vak i tel ik het aantal incidenten dat daar in de periode van 2004 tot 2016 heeft plaatsgevonden en deze getallen deel ik vervolgens door het totaal aantal minuten dat in deze periode is verstreken.

Verder beschikt de brandweer over allerlei informatie over objecten uit de Basis Registraties Adressen en Gebouwen (BAG) in Twente. In figuur 8 is een klein deel van deze gegevens weergegeven.

Postcode	X	Y	Objectsoort	Omschrijving	Positie	Normtijd	Opkomsttijd	KVT_code
7611AZ	6,635099	52,37383	30011	woonfunctie	1	8	6,541943	ALMELO
7611AZ	6,635099	52,37383	30011	woonfunctie	2	0	11,36679	VRIEZV
7611AZ	6,635099	52,37383	30011	woonfunctie	3	0	12,35494	WIERDN
7611AZ	6,636559	52,37457	30011	woonfunctie	1	8	6,408713	ALMELO
7611AZ	6,636559	52,37457	30011	woonfunctie	2	0	11,1612	VRIEZV
7611AZ	6,636559	52,37457	30011	woonfunctie	3	0	12,22171	WIERDN
7611AP	6,628018	52,38059	30007	onderwijsfunctie	1	8	8,507242	ALMELO
7611AP	6,628018	52,38059	30007	onderwijsfunctie	2	0	10,47486	VRIEZV
7611AP	6,628018	52,38059	30007	onderwijsfunctie	3	0	13,36867	WIERDN
7611AR	6,628866	52,3815	30011	woonfunctie	1	8	8,21973	ALMELO
7611AR	6,628866	52,3815	30011	woonfunctie	2	0	10,18734	VRIEZV
7611AR	6,628866	52,3815	30011	woonfunctie	3	0	13,08116	WIERDN
7611AR	6,628791	52,38146	30011	woonfunctie	1	8	8,222733	ALMELO
7611AR	6,628791	52,38146	30011	woonfunctie	2	0	10,19035	VRIEZV
7611AR	6,628791	52,38146	30011	woonfunctie	3	0	13,08416	WIERDN
7611AR	6,628628	52,38139	30011	woonfunctie	1	8	8,327842	ALMELO
7611AR	6,628628	52,38139	30011	woonfunctie	2	0	10,29545	VRIEZV
7611AR	6,628628	52,38139	30011	woonfunctie	3	0	13,18927	WIERDN

Figuur 8: Voorbeeld informatie over BAG objecten

Voor ieder van deze objecten is de precieze locatie en de functie geregistreerd. Daarnaast is aan ieder van de objecten een lijst van de drie kazernes gekoppeld van waaruit de brandweer als eerst auto's zal sturen in het geval van een incident. Bovendien is voor ieder object en iedere kazerne de normtijd en de opkomsttijd geregistreerd. Deze begrippen zijn nader gespecificeerd in het Dekkingsplan brandweer Twente [2]. Volgens dit dekkingsplan schrijft de wet voor dat de brandweer te allen tijde binnen 18 minuten aanwezig dient te zijn bij een incident. De normtijden, oftewel de opkomsttijden waar de brandweer naar streeft, liggen ruim binnen deze limiet. Deze zijn meestal 5, 6, 8 of 10 minuten,

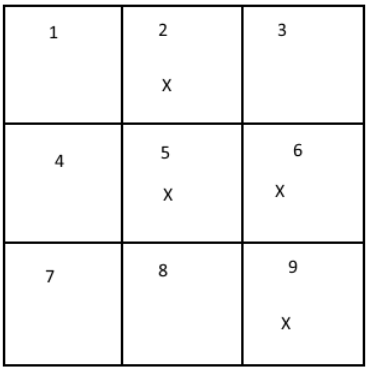
afhankelijk van de functie van het object. In vele gevallen zijn deze tijden niet haalbaar en het bestuur van de veiligheidsregio heeft de bevoegdheid om van deze normtijden af te wijken. De opkomsttijd is in het dekkingsplan gedefinieerd als de tijd tussen de aankomst van een melding en de aankomst van de eerste brandweerauto bij het incident. Deze tijd is opgebouwd uit drie delen: de verwerkingstijd, de uitruktijd en de aanrijdtijd. De verwerkingstijd is de tijd die nodig is om de melding te ontvangen, de ernst van het incident in te schatten en een geschikte kazerne op de hoogte te stellen. Vervolgens verstrijkt er enige tijd totdat er daadwerkelijk een auto de kazerne verlaat. Brandweerauto's worden meestal bemand door vrijwilligers. Het duurt even voordat deze gekleed en wel gereed zijn om te vertrekken. De tijd die verstrijkt tussen het alarm in de kazerne en het moment waarop er daadwerkelijk een auto vertrekt, wordt de uitruktijd genoemd. De aanrijdtijd tenslotte is de tijd die een brandweerauto nodig heeft om naar de plek van het incident te rijden.

Met behulp van deze data bepaal ik voor ieder punt in V de drie meest geschikte kazernes. De code is te vinden in Appendix C. Allereerst verzamel ik voor ieder punt i in V alle BAG objecten in het corresponderende vakje. Vervolgens maak ik op basis van de kazerne ranglijsten voor deze objecten een ranglijst voor het vakje. Hiervoor maak ik gebruik van de zogenaamde Borda count methode [4]: voor ieder BAG object in het vakje ken ik de kazerne die op positie 3 staat 0 punten toe, de kazerne die op positie 2 staat 1 punt en de kazerne die op positie 1 staat 3 punten. Uiteindelijk plaats ik de kazerne met het hoogste aantal punten op positie 1, de kazerne met het op een na hoogste aantal punten belandt op positie 2, etc.

Voor de opkomsttijd van een brandweerauto vanaf een zekere kazerne naar een bepaald vakje neem ik het gemiddelde van de opkomsttijden vanaf deze kazerne naar de objecten in het vakje. De normtijd voor een vakje bereken ik op dezelfde manier. Om de beslissingsruimte in te perken, neem ik aan dat er in totaal slechts 1 auto per tijdseenheid naar een andere kazerne kan worden verplaatst en wel alleen naar 1 van de 2 dichtstbijzijnde kazernes.
Om de rijtijd tussen een paar kazernes te bepalen maak ik gebruik van de lijst met BAG objecten. Voor iedere kazerne zoek ik allereerst het bijbehorende BAG object. De kazerne die bij dit object op positie 1 staat is als het goed is deze kazerne zelf. Aangezien de aanrijdtijd vanaf een kazerne naar dezelfde kazerne gelijk is aan 0, is de bijbehorende opkomsttijd dus gelijk aan de verwerkingstijd en de uitruktijd voor een incident. Uit gegevens van de brandweer blijkt dat de uitruktijd voor auto's die geherpositioneerd worden ongeveer gelijk is aan 90 seconden. De rijtijd van een kazerne i naar een kazerne j wordt dan gegeven door deze uitruktijd plus de opkomsttijd van kazerne j naar kazerne i minus de verwerkingstijd en uitruktijd van kazerne j. De code die ik heb geschreven om deze tijden te bepalen is gegeven in Appendix D.

6 Resultaten

Om te onderzoeken of het model wenselijke resultaten oplevert, laat ik het allereerst los op een eenvoudig scenario. Ik beschouw een rooster van 9 vakken, zoals weergegeven in figuur 9. De vakken zijn genummerd en de kruisjes stellen brandweerkazernes voor.



Figuur 9: Rooster van 9 vakken

Ik ga ervan uit dat er bij iedere kazerne slechts 1 auto kan staan, dat de zijde van een vierkant 500 meter lang is en dat de auto's rijden met een snelheid van 30 km/u. Daarnaast hanteer ik hemelsbrede afstanden tussen de punten, tijdsloten van een kwartier en een normtijd van 7,5 minuten. Ik neem aan dat een auto 15 minuten besteedt bij een incident. Laat λ_i het gemiddelde aantal ongelukken per kwartier voorstellen voor vak i. Ik neem $\lambda_1 = 10$ en $\lambda_i = 0.1$ voor $i \in \{2, 3, 4, 5, 6, 7, 8, 9\}$. In vak 1 gebeuren dus significant meer ongelukken dan op de overige plekken. Stel dat er op een zeker moment een incident voorvalt in vak 3 dat 2 brandweerauto's vereist. Als het goed is zullen dan de auto van de kazerne in vak 2 en de auto van de kazerne in vak 6 uitrukken. Er zijn dan nog maar twee werkloze auto's over. Aangezien vak 1 nogal incidentgevoelig is in verhouding tot de overige vakken, zou de auto van kazerne 9 dus beter op kunnen schuiven naar een dichterbij gelegen kazerne.

Om de optimale actie te bepalen maak ik gebruik van de heuristische methode. Zelfs een dergelijk eenvoudig scenario is te complex om een exacte oplossing te vinden met behulp van de optimaliteitsvergelijkingen. Er zijn immers vele verschillende toestanden mogelijk. Allereerst kan een auto zich in 4 verschillende fasen bevinden. Aangezien er 4 auto's in het spel zijn, levert dit $4^4 = 256$ mogelijkheden op. Daarnaast kan een auto toebehoren aan 4 verschillende kazernes. Dit levert opnieuw 256 mogelijkheden op. Verder kan er in ieder vakje een incident plaatsvinden en kunnen er bovendien nog auto's nodig zijn voor

incidenten uit eerdere tijdsvakken. Dit levert minimaal 10 mogelijkheden op voor de vector $\mathbf{x}$. Daarnaast kan er bij iedere kazerne ofwel geen of 1 werkloze auto staan. Dit levert $2^4 = 16$ mogelijkheden op voor de vector $\mathbf{y}$. Wanneer alle auto's bezig zijn met het oplossen van een incident zijn er verder nog $9^4 = 6561$ mogelijkheden voor de vector $\mathbf{b}$. Wanneer alle auto's op weg zijn naar incidenten zijn er bovendien 6561 mogelijkheden voor de vector $\mathbf{w}$. Wanneer alle auto's op weg zijn naar een kazerne zijn er $4^4 = 256$ mogelijkheden voor de vector $\mathbf{z}$. Als alle auto's bezig zijn met het afhandelen van een incident zijn er in totaal $9^4 = 6561$ mogelijkheden voor de matrix P en als alle auto's onderweg zijn naar een nieuw punt zijn er in totaal minimaal $9^4 = 6561$ mogelijkheden voor de matrix Q. Het totaal aantal mogelijke toestanden is dus in de orde van grootte van $256 \cdot 256 \cdot 10 \cdot 16 \cdot 6561 \cdot 6561 \cdot 256 \cdot 6561 \cdot 6561 \approx 4,97 \cdot 10^{24}$. Het oplossen van het stelsel optimaliteitsvergelijkingen voor een dergelijk systeem kost enorm veel rekentijd. Terugvallen op de heuristische methode is dan ook noodzakelijk.

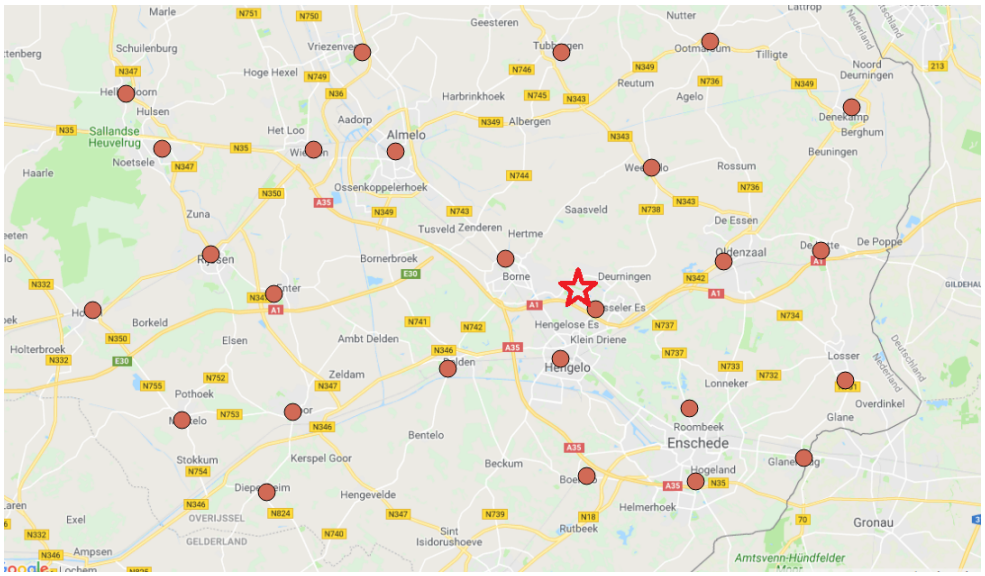
Om de optimale actie te bepalen, maak ik gebruik van de kostenfunctie in (26) met $\gamma = 0.8$. Deze waarde van γ bleek na enig uitproberen een realistische functie op te leveren. In tabel 4 zijn alle mogelijke acties weergegeven met de bijbehorende kosten. Deze kosten zijn berekend met behulp van de heuristische methode weergegeven in (36). De code is gegeven in Appendix E.

Actie	Kosten
Geen verplaatsing	1.57
Auto van kazerne 5 naar kazerne 2	1.11
Auto van kazerne 5 naar kazerne 6	2.61
Auto van kazerne 9 naar kazerne 6	1.56
Auto van kazerne 9 naar kazerne 2	1.10

Tabel 1: Acties en bijbehorende kosten

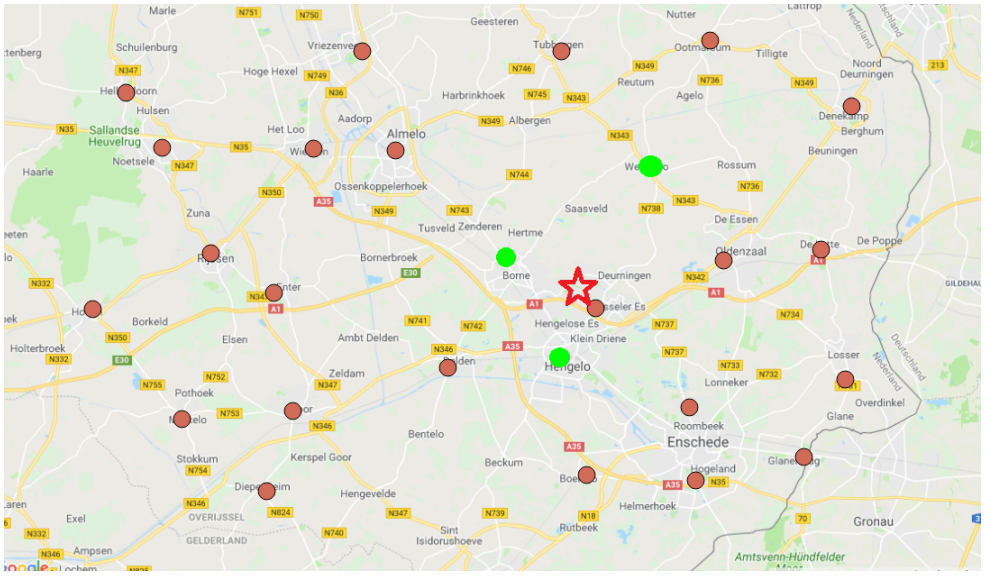
Uit de tabel blijkt dat de brandweer dus inderdaad het beste de auto van kazerne 9 kan verplaatsen naar kazerne 2, die dichter in de buurt ligt van vak 1.

Aangezien het model aannemelijke resultaten op lijkt te leveren voor een simpel scenario, laat ik het los op de regio Twente. Ik laat een groot incident voorvallen aan Trondheimstraat 11 in Hengelo, waar maar liefst 3 tankautospuiten aan te pas moeten komen. De code is te vinden in Appendix F. In figuur 10 is een kaart van Twente weergegeven. De rode stippen stellen brandweerkazernes voor en de rode ster geeft de plaats van het incident aan.



Figuur 10: Kaart van Twente met locatie van het incident

Uit de resultaten blijkt dat er brandweerauto's uitrukken vanuit de kazernes in Hengelo Centrum, Borne en Weerselo. In figuur 11 zijn deze kazernes weergegeven met behulp van groene stippen.



Figuur 11: Kaart van Twente met locatie van het incident en onbemande kazernes

Er zijn 8 beslissingen mogelijk. Deze zijn weergegeven in tabel 2 met de bijbehorende kosten.

Actie	Kosten
Geen verplaatsing	0.40
Auto van Tubbergen naar Weerselo	0.41
Auto van Oldenzaal naar Hengelo centrum	0.52
Auto van Oldenzaal naar Weerselo	0.52
Auto van Hengelo Noord naar Borne	0.40
Auto van Delden naar Hengelo centrum	0.60
Auto van Delden naar Borne	0.60
Auto van Enschede Noord naar Hengelo centrum	0.46

Tabel 2: Acties en bijbehorende kosten

Uit de tabel volgt dat er 2 acties zijn die de minste kosten opleveren: ofwel geen auto's verplaatsen ofwel een auto verplaatsen van de kazerne in Hengelo Noord naar de kazerne in Borne. Dit is een plausibele uitkomst. Uit de lijst met BAG objecten blijkt namelijk dat er vaker een beroep wordt gedaan op de kazerne in Borne dan op die in Hengelo Noord. Bovendien is de verwerkingstijd en de uitruktijd voor de kazerne in Borne korter, waardoor het intuïtief dus gunstiger is voor de brandweer om uit te rukken vanuit Borne dan vanuit Hengelo Noord.

Naast deze situatie heb ik verschillende andere scenario's onder de loep genomen. In vele gevallen blijkt het voordelig te zijn om geen auto's te verplaatsen. Wanneer er echter een auto vrij is in de buurt van een onbemande kazerne waar regelmatig beroep op wordt gedaan, blijkt het vaak gunstig om deze auto te verplaatsen.

Tenslotte onderzoek ik wat er gebeurt wanneer er onvoldoende brandweerauto's zijn om aan de vraag te voldoen. Ik laat een groot incident voorvallen in het gebouw Carré aan Drienerlolaan 5, waar maar liefst 28 tankautospuiten aan te pas moeten komen. Bovendien laat ik tegelijkertijd één tankautospuit bezig zijn bij een kleine brand aan Witbreuksweg 401-303 en één tankautospuit bij een incident aan Westeinde 1 in Vriezenveen. Aangezien er in totaal 29 tankautospuiten beschikbaar zijn in Twente, is er dus één auto te weinig voor het incident in Carré. Ik neem aan dat beide kleine incidenten zijn afgehandeld aan het einde van het tijdvak. Als het goed is, zal de auto die zich het dichtst in de buurt bevindt direct doorrijden naar Carré. Het programma dat deze situatie modelleert is gegeven in Appendix G. Zoals verwacht rukken eerst alle beschikbare auto's uit naar het incident in Carré. Vervolgens rijdt de auto die bezig was bij de Witbreuksweg inderdaad ook door naar dit incident en keert de auto die bezig was in Vriezenveen terug naar de kazerne waar hij vandaan kwam. Aangezien er geen werkloze auto's zijn, is er slechts één mogelijke beslissing in de beslissingsruimte, namelijk geen auto's verplaatsen.

7 Conclusie

Met enige regelmaat valt er in Twente een incident voor waar de brandweer aan te pas moet komen. Een goede planning van de inzet van brandweerauto's is dan ook van groot belang. In dit onderzoek heb ik een model ontwikkeld voor de optimale inzet van brandweerauto's bij calamiteiten.
Allereerst heb ik de gevestigde modellen voor de proactieve planning van ambulancevervoer bestudeerd en de verschillen tussen brandweer en ambulancediensten onderzocht. In tegenstelling tot de ambulancediensten, blijkt de brandweer gebruik te maken van meerdere typen auto's. In de praktijk worden echter vrijwel alleen maar tankautospuiten ingezet. De overige typen auto's heb ik dan ook buiten beschouwing gelaten. Daarnaast is het aantal incidenten waarvoor de brandweer moet uitrukken significant kleiner dan het aantal incidenten waar een ambulance aan te pas moet komen. Tenslotte doorloopt een brandweerauto een andere cyclus dan een ambulance: laatstgenoemde rijdt vaak eerst met een patiënt naar een ziekenhuis alvorens terug te keren naar een post, terwijl een vrijgekomen brandweerauto direct doorrijdt naar een kazerne of een nieuw incident.
Vervolgens heb ik een uitgebreid en algemeen model ontwikkeld voor de optimale inzet van brandweerauto's bij calamiteiten, gebaseerd op Markovbeslissingstheorie. Ik beschouw de verdeling van de brandweerauto's over de regio als een systeem dat zich op ieder tijdstip in een bepaalde toestand bevindt. Afhankelijk van deze toestand zijn er verschillende acties mogelijk. Om een actie te selecteren, maak ik gebruik van een heuristische methode: voor iedere mogelijke beslissing in een zeker tijdslot bereken ik de verwachte kosten voor eventuele nieuwe incidenten in het volgende tijdslot. Vervolgens selecteer ik de beslissing die de minste kosten oplevert.

Om te onderzoeken of het model wenselijke resultaten oplevert, heb ik het getest op een eenvoudig scenario. Het bleek dat er inderdaad een auto verplaatst zou moeten worden naar een kazerne dichterbij een incidentgevoelige plek. Vervolgens heb ik het losgelaten op de regio Twente. Daarbij heb ik gebruik gemaakt van gegevens ter beschikking gesteld door de brandweer. Ik heb een aantal mogelijke scenario's bedacht en de uitkomsten van het model onder de loep genomen. In vele gevallen blijkt het het voordeligst te zijn om geen auto's te verplaatsen. Wanneer er echter een kazerne waar vaak een beroep op wordt gedaan onbemand is, terwijl een nabijgelegen kazerne wel een auto tot zijn beschikking heeft, blijkt het gunstig te zijn om deze auto te verplaatsen.

8 Discussie

Het resultaat van dit onderzoek is een model dat de brandweerauto's in de goede richting lijkt te sturen. De uitkomsten die het model oplevert zijn naar verwachting: Wanneer er een auto vrij is in de buurt van een onbemande kazerne waar regelmatig een beroep op wordt gedaan, blijkt het gunstig te zijn om deze auto te verplaatsen. In vele gevallen blijkt het het voordeligst te zijn om geen kazernes her te bezetten. Dit resultaat is plausibel, aangezien er in Twente geen plaatsen zijn waar regelmatig enorme rampen voorvallen. De brandweer heeft meestal simpelweg voldoende middelen om incidenten op te lossen zonder auto's te herpositioneren.

Hoewel het uiteindelijke model aannemelijke resultaten oplevert, is er nog voldoende ruimte voor uitbreiding. Zo ben ik er bij het modelleren vanuit gegaan dat de rijtijd van een brandweerauto vastligt en niet afhangt van het tijdstip t. In werkelijkheid zullen deze rijtijden echter stochastisch zijn en bovendien afhangen van het tijdstip. Ook de tijden die een auto nodig heeft om een incident op te lossen zullen in werkelijkheid geen vaste waarde hebben.
Bovendien maak ik bij het selecteren van een actie met behulp van de heuristische methode alleen gebruik van de mogelijke situaties in het volgende tijdslot. Een methode die ook rekening houdt met tijdsloten verder in de toekomst zou nauwkeurigere resultaten opleveren. Dit kost echter veel meer rekentijd.

Tenslotte heb ik alleen oplossingen gevonden met behulp van een heuristische methode. Het systeem is immers te complex om exacte oplossingen te vinden. Zelfs voor het simpele scenario beschreven in sectie 4 blijkt de rekentijd uit de hand te lopen. Voor hele eenvoudige situaties zou het wel mogelijk zijn om exacte oplossingen te vinden met behulp van de optimaliteitsvergelijkingen. Voor deze gevallen zou uitgezocht kunnen worden in hoeverre de oplossing die de heuristische methode oplevert afwijkt van de optimale oplossing.

9 Symbolenlijst

Symbool	Betekenis
V	Verzameling vakken
W	Verzameling vakken met kazerne
N	Aantal vakken
M	Aantal kazernes
K	Aantal typen auto's
s_t	Toestand systeem op tijdstip t
$\mathbf{m}_k(i)$	Maximum aantal auto's van type k dat kazerne van punt i kan herbergen
B	Totaal aantal auto's
$\mathbf{n}(i)$	Type van auto i
$\mathbf{u}_t(i)$	Fase van auto i op tijdstip t
$\mathbf{v}_t(i)$	Kazerne waar auto i aan toebehoort op tijdstip t
$\mathbf{x}_{t,k}(i)$	Aantal auto's van type k dat nodig is voor een incident bij vraagpunt $i \in V$ in tijdslot t
$\mathbf{y}_{t,k}(i)$	Aantal auto's van type k dat werkloos bij kazerne $i \in W$ staat op tijdstip t
$\mathbf{b}_{t,k}(i)$	Aantal auto's van type k dat bezig is bij vraagpunt $i \in V$ op tijdstip t
$\mathbf{w}_{t,k}(i)$	Aantal auto's van type k dat op weg is naar vraagpunt $i \in V$ op tijdstip t
$\mathbf{z}_{t,k}(i)$	Aantal auto's van type k dat op weg is naar kazerne $i \in W$ op tijdstip t
$T(i,j)$	Tijd die een auto nodig heeft om van punt $i \in V$ naar punt $j \in V$ te rijden
$\mathbf{r}(k)$	Tijd die een auto van type k nodig heeft om een incident op te lossen.
$H_t(i,j)$	Resterende tijd die auto j nog nodig heeft voor een incident bij vraagpunt $i \in V$ op tijdstip t
$Q_t(i,j)$	Resterende reistijd voor auto j naar vraagpunt $i \in V$ op tijdstip t
f	Functie de een vraagpunt $i \in V$ afbeeldt op een vector bestaande uit elementen van W
S	Toestandsruimte
$\mathbf{h}_{t,k}(i,j)$	Aantal auto's van type k dat vanuit kazerne $i \in W$ is uitgerukt naar vraagpunt $j \in V$ in tijdsvak t
$\mathbf{g}_{t,k}(i,j)$	Aantal auto's van type k dat van een incident bij vraagpunt $i \in V$ direct doorrijdt naar een incident bij vraagpunt $j \in V$.
$D(s)$	Verzameling mogelijke beslissingen wanneer het systeem zich bevindt in toestand $s \in S$
$A_k(i,j)$	Aantal auto's dat moet worden verplaatst van kazerne $i \in W$ naar kazerne $j \in W$
$p(s_t \mid s_{t-1}, d)$	Kans dat het systeem zich op een bepaald tijdstip in toestand s_t bevindt als het zich in de tijdseenheid daarvoor in s_{t-1} bevond en beslissing d is genomen
λ_i	Gemiddeld aantal incidenten per tijdseenheid voor vak $i \in V$
$P_k(n)$	Kans dat er voor een zeker incident n auto's van type k nodig zijn
ϕ	Kostenfunctie
T^*	Tijdslimiet
Δ	Beleidsruimte

Tabel 3: Acties en bijbehorende kosten

Symbool	Betekenis
$V_\delta(s)$	Verwachte kosten gedurende een oneindig aantal perioden onder beleid δ gegeven dat het systeem zich aan het begin van periode 1 in toestand s bevindt
$V(s)$	Verwachte kosten onder optimaal beleid
δ^*	Optimaal beleid
$c(s)$	Kosten voor een enkel tijdslot wanneer het systeem zich in toestand s bevindt
$R_i(s_t, d)$	Responstijd voor een incident bij vraagpunt i wanneer het systeem zich in toestand s_t bevindt en beslissing d is genomen aan het begin van tijdslot t

Tabel 4: Acties en bijbehorende kosten

Referenties

[1] Aardal, K., Van Barneveld, T., Van den Berg, P., Bhulai, S., Van Buuren, M., Van Essen, T., . . . Van der Mei, R. (2015). Van reactieve naar proactieve planning van ambulancediensten. Nieuw archief voor wiskunde, 5(3), 183-192.

[2] Brandweer Twente. (2012). Dekkingsplan Brandweer Twente. Retrieved from http://www.vrtwente.nl/media/1173/e03g-dekkingsplan-brandweer-twente.pdf

[3] Daskin, M.S. (1983). A maximum expected covering location model: formulation, properties and heuristic solution. Transportation Science 17(1), 48-70.

[4] Easley, D., Kleinberg, J. (2010). Networks, Crowds and Markets: Reasoning about a Highly Connected World (2nd ed.). Cambridge, England: Cambridge University Press.

[5] Gendreau, M., Laporte, G., Semet, F. (2001). A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. Parallel Computing 27, 1641-1653.

[6] Ingolfsson, A., Budge, S., Erkut, E. (2008). Optimal ambulance location with random delays and travel times. Health Care Management Science, 11(3), 262-274.

[7] Jagtenberg, C.J., Bhulai, S., van der Mei, R.D. (2015). An efficient heuristic for real-time ambulance redeployment. Operations Research for Health Care 4, 27-35.

[8] Li, X., Zhao, Z., Zhu, X., Wyatt, T. (2011). Covering models and optimization techniques for emergency response facility location and planning: a review. Mathematical Methods of Operations Research, 74(3), 281-310.

[9] Schmid, V., Doerner, K. F. (2010). Ambulance location and relocation problems with time-dependent travel times. European Journal of Operational Research, 207(3), 1293-1303.

[10] Van Barneveld, T. C., Bhulai, S., Van der Mei, R. D. (2015). A dynamic ambulance management model for rural areas. Health Care Management Science, 1, 1-22. doi: 10.1007/s10729-015-9341-3

10 Appendix

Appendix A

```
import pandas as pd
import os


os.chdir("C:/Users/Maike/Documents/Bacheloropdracht/Data/Brandweer/ \\
Incidenten/4 - Nuttige data/Nuttige data  - alle incidenten/Nuttige data\\
 - excel files")
df1 = pd.read_excel('Twente_incidenten_brw_0415_365.xlsx', \\
 sheet_name='Page1_1')

os.chdir("C:/Users/Maike/Documents/Bacheloropdracht/Data/Brandweer/ \\
CBS 500m vierkanten Twente")
df2 = pd.read_excel('Twente_CBS_covariaten.xlsx', sheet_name='Sheet1')

os.chdir("C:/Users/Maike/Documents/Bacheloropdracht/Data/\\
Covariaten20180307T104019Z001/Covariaten/Gebouwen Twente/Brandweerkazernes \\
Twente")
df3 = pd.read_excel('Twente_brandweerkazernes_RD.xlsx', sheet_name='Blad1')

x1 = df2["X1"]
y1 = df2["Y1"]
x2 = df2["X2"]
y2 = df2["Y2"]
xmid = df2["Xmid"]
ymid = df2["Ymid"]
lat1 = df2["lat1"]
lat2 = df2["lat2"]
lon1 = df2["lon1"]
lon2 = df2["lon2"]

nvakken = len(x1)

#Grenzen en middelpunten van 500m bij 500m vierkanten:
vakken = dict()
vakkenmid = dict()

for i in range(nvakken):
    vakken[i] = [x1[i], x2[i], y1[i], y2[i]]
    vakkenmid[i] = [xmid[i], ymid[i]]
```

```
vakkenlatlon = dict()

for i in range(nvakken):
    vakkenlatlon[i] = [lon1[i], lon2[i], lat1[i], lat2[i]]




xbk = df3["x,N,20,0"]
ybk = df3["y,N,20,0"]
namen = df3["loc_naam,C,254"]

V = [i for i in range(nvakken)]

print(V)

#Kazernes en dictionaries met namen, nummers en codes:

W = []
naamnummer = dict()
nummernaam = dict()

for j in range(nvakken):
    vak = vakken[j]
    for i in range(len(xbk)):
        naam = namen[i]
        x,y = xbk[i],ybk[i]
        if vak[0] <= x <= vak[1] and vak[2] <= y <= vak[3]:
            W.append(j)
            naamnummer[naam] = j
            nummernaam[j] = naam

print(W)

codenaam = dict()

codenaam['DENHOV'] = 'Den Ham'
codenaam['VROOMH'] = 'Vroomshoop'
codenaam['OOTMSM'] = 'Ootmarsum'
codenaam['VRIEZV'] = 'Vriezenveen'
codenaam['TUBBRG'] = 'Tubbergen'
codenaam['HELLDN'] = 'Hellendoorn'
codenaam['DENEKP'] = 'Denekamp'
codenaam['NIJVDL'] = 'Nijverdal'
codenaam['WIERDN'] = 'Wierden'
codenaam['ALMELO'] = 'Almelo-c'
```

```
codenaam['WEERSL'] = 'Weerselo'
codenaam['DELUTT'] = 'De Lutte'
codenaam['RIJSSN'] = 'Rijssen'
codenaam['BORNE'] = 'Borne'
codenaam['OLZAAL'] = 'Oldenzaal'
codenaam['ENTER'] = 'Enter'
codenaam['HENGOVN'] = 'Hengelo-n'
codenaam['HOLTEN'] = 'Holten'
codenaam['HENGOVC'] = 'Hengelo-c'
codenaam['DELDEN'] = 'Delden'
codenaam['LOSSER'] = 'Losser'
codenaam['GOOR'] = 'Goor'
codenaam['ENSCHDN'] = 'Enschede-noord'
codenaam['MARKLO'] = 'Markelo'
codenaam['ENSCHDGL'] = 'Glanerbrug'
codenaam['ENSCHDBK'] = 'Boekelo'
codenaam['ENSCHDHP'] = 'Enschede-centrum'
codenaam['DIEPHM'] = 'Diepenheim'
codenaam['HAAKSB'] = 'Haaksbergen'

Wcodes = dict()

Wcodes['LEMEVD'] = 'G'
Wcodes['LUTTBG'] = 'G'
Wcodes['RAALTE'] = 'G'
Wcodes['OMMEN'] = 'G'
Wcodes['HEETEN'] = 'G'
Wcodes['HARDBG'] = 'G'
Wcodes['BERGTH'] = 'G'
Wcodes['DEVENTB'] = 'G'
Wcodes['LARNGE'] = 'G'
Wcodes['BORCLO'] = 'G'
Wcodes['NEEDE'] = 'G'
Wcodes['EIBERG'] = 'G'
Wcodes['NORDHO'] = 'G'
Wcodes['UELSEN'] = 'G'
Wcodes['GILDEH'] = 'G'
Wcodes['BATHNM'] = 'G'
Wcodes['BADBEN'] = 'G'
Wcodes['GRONAU'] = 'G'
Wcodes['AHAUS_AL'] = 'G'
Wcodes['LOCHEM'] = 'G'



for code in codenaam:
```

$$\text{naam} = \text{codenaam}[\,\text{code}\,]$$
$$\text{nummer} = \text{naamnummer}[\,\text{naam}\,]$$
$$\text{Wcodes}[\,\text{code}\,] = \text{nummer}$$

Appendix B

```
import pandas as pd
import os
from Locaties import vakken, nvakken



os.chdir("C:/Users/Maike/Documents/Bacheloropdracht/Data/Brandweer/ \\
Incidenten/4 - Nuttige data/Nuttige data \\
 - alle incidenten/Nuttige data - excel files")
df = pd.read_excel('Twente_incidenten_brw_0415_365.xlsx',\\
 sheet_name='Page1_1')

xcors = df["X coordinaat"]
ycors = df["Y coordinaat"]



#Totaal aantal incidenten per vakje:

incidents = [0 for i in range(nvakken)]

for i in range(len(xcors)):
    x = xcors[i]
    y = ycors[i]
    for j in range(nvakken):
        vak = vakken[j]
        if vakken[j][0] <= x:
            if vakken[j][1] > x:
                if vakken[j][2] <= y:
                    if vakken[j][3] > y:
                        incidents[j] += 1

#Totaal aantal verstreken minuten:
minutes = (9*365 + 3*366)*24*60

#Gemiddeld aantal ongelukken per vakje per minuut:
labda = [i/minutes for i in incidents]
print(labda)
```

Appendix C

```
from Locaties import V, W, vakkenlatlon, Wcodes, vakkenmid
from math import sqrt
import pandas as pd
import os


os.chdir("C:/Users/Maike/Documents/Bacheloropdracht/Data/Brandweer/ \\
Incidenten/4 - Nuttige data/Nuttige data \\
 - alle incidenten/Nuttige data - excel files")
df = pd.read_excel('BAG.xlsx', sheet_name='Blad1')

#Hemelsbrede afstand tussen 2 vakjes:
def afstandhb(i,j):
    return sqrt((vakkenmid[i][0]-vakkenmid[j][0])**2 + (vakkenmid[i][1] \\
    -vakkenmid[j][1])**2)

#Benodigde tijd om hemelsbreed van vakje 1 naar vakje j te rijden
def T(i,j,snelheid):
    return afstandhb(i,j)/snelheid

snelheid = 30000/60 #m/min

xcors = df["X"]
ycors = df["Y"]
kazernes = df["KVT_code"]
opkomsttijden = df["Opkomsttijd"]
posities = df["Positie"]
normtijden = df["Normtijd"]


#Coordinaten, voorkeurskazernes en bijbehorende opkomsttijden \\
voor ieder BAG object:
BAGxcors = dict()
BAGycors = dict()
BAGkazernes = dict()
BAGopkomsttijden = dict()
BAGnormtijden = dict()


i = 0
j = 0
print("len(xcors)")
print(len(xcors))
```

```
while i < len(xcors):
    BAGxcors[j] = xcors[i]
    BAGycors[j] = ycors[i]
    BAGnormtijden[j] = normtijden[i]
    BAGkazernes[j] = []
    BAGopkomsttijden[j] = []
    BAGkazernes[j].append(kazernes[i])
    BAGopkomsttijden[j].append(opkomsttijden[i])
    while posities[i] == 1:
        i += 1
    BAGkazernes[j].append(kazernes[i])
    BAGopkomsttijden[j].append(opkomsttijden[i])
    while posities[i] == 2:
        i += 1
    BAGkazernes[j].append(kazernes[i])
    BAGopkomsttijden[j].append(opkomsttijden[i])
    while i < len(xcors) and posities[i] == 3:
        i += 1
    j += 1
    print(i)


#BAG objecten voor ieder vakje:
BAG = dict()
print(len(V))

for i in V:
    print(i)
    BAG[i] = []
    vak = vakkenlatlon[i]
    for j in range(len(BAGxcors)):
        x = BAGxcors[j]
        y = BAGycors[j]
        if vak[0] <= x and x < vak[1] and vak[2] <= y and y < vak[3]:
            BAG[i].append(j)

print(BAG)


#Voorkeurskazernes voor ieder vakje bepaald door middel van Borda count
f = dict()

for i in V:
    Borda = dict()
    for j in BAG[i]:
```

```
            pos1 = BAGkazernes[j][0]
            pos2 = BAGkazernes[j][1]
            pos3 = BAGkazernes[j][2]
            if pos1 in Borda.keys():
                Borda[pos1] += 2
            else:
                Borda[pos1] = 2
            if pos2 in Borda.keys():
                Borda[pos2] += 1
            else:
                Borda[pos2] = 1
            if pos3 not in Borda.keys():
                Borda[pos3] = 0
        Bordasort = sorted(Borda.items(), key=lambda x: x[1])
        f[i] = [Bordasort[len(Bordasort)-i][0] for i in range(1,len(Bordasort)+1)]
        print(f[i])
        for j in range(len(f[i])):
            f[i][j] = Wcodes[f[i][j]]


g = dict()


for i in V:
    rijtijden = dict()
    for j in W:
        if j not in f[i]:
            rijtijden[j] = T(i,j,snelheid)
    rijtijdens = sorted(rijtijden.items(), key=lambda x: x[1])
    g[i] =[rijtijdens[i][0] for i in range(len(rijtijdens))]


print(g)
```

Appendix D

```
from Locaties import V, W, Wcodes, vakkenmid
from math import sqrt
from BAG import BAG, f, BAGkazernes, BAGopkomsttijden, BAGnormtijden, g

#Hemelsbrede afstand tussen 2 vakjes:
def afstandhb(i,j):
    return sqrt((vakkenmid[i][0]-vakkenmid[j][0])**2 + (vakkenmid[i][1] \\
    -vakkenmid[j][1])**2)

#Benodigde tijd om hemelsbreed van vakje 1 naar vakje j te rijden
def T(i,j,snelheid):
    return afstandhb(i,j)/snelheid

snelheid = 30000/60 #m/min

#Opkomsttijd voor ieder vakje en iedere kazerne:
opkomsttijden = [[1000 for j in range(len(W)+1)] for i in V]

for i in V:
    kazernetijd = dict()
    for kazerne in f[i]:
        kazernetijd[kazerne] = [0,0]
    for j in BAG[i]:
        for k in range(3):
            kazerne = Wcodes[BAGkazernes[j][k]]
            opkomsttijd = BAGopkomsttijden[j][k]
            kazernetijd[kazerne][0] += opkomsttijd
            kazernetijd[kazerne][1] += 1
    for kazerne in kazernetijd:
        kazernetijd[kazerne] = kazernetijd[kazerne][0]/kazernetijd[kazerne][1]
    for kazerne in f[i]:
        if kazerne == 'G':
            opkomsttijden[i][-1] = kazernetijd[kazerne]
        else:
            opkomsttijden[i][W.index(kazerne)] = kazernetijd[kazerne]

print(opkomsttijden)

#Normtijd voor ieder vakje:
normtijden = [0 for i in V]

for i in V:
    normtijdsom = 0
```

```
    normtijdtot = 0
    for obj in BAG[i]:
        normtijdsom += BAGnormtijden[obj]
        normtijdtot += 1
    if normtijdtot > 0:
        normtijden[i] = normtijdsom/normtijdtot
    else:
        normtijden[i] = 18

print(normtijden)


#Rijtijden tussen kazernes:
Tkazernes = [[0 for j in W] for i in W]
vutijd = dict()
kazernes = dict()
tijden = dict()
for i in W:
    kazernes[i] = []
    tijden[i] = []

    #Adressen kazernes:
    #DENHOV: Dorpsstraat 49
nummer = Wcodes['DENHOV']
vutijd[nummer] = 4.692711
kazernes[nummer].append(Wcodes['VROOMH'])
kazernes[nummer].append(Wcodes['HELLDN'])
tijden[nummer].append(10.35755)
tijden[nummer].append(17.10387)
    #VROOMH: Plantsoen 8
nummer = Wcodes['VROOMH']
vutijd[nummer] = 5.303232
kazernes[nummer].append(Wcodes['DENHOV'])
kazernes[nummer].append(Wcodes['VRIEZV'])
tijden[nummer].append(9.711151)
tijden[nummer].append(18.97246)
    #OOTMSM: Laagsestraat 4
nummer = Wcodes['OOTMSM']
vutijd[nummer] = 0
kazernes[nummer].append(f[nummer][0])
kazernes[nummer].append(f[nummer][1])
tijden[nummer].append(T(nummer,f[nummer][0],30))
tijden[nummer].append(T(nummer,f[nummer][1],30))
    #VRIEZV: De Zuivering 30
nummer = Wcodes['VRIEZV']
vutijd[nummer] = 5.11952
```

```
kazernes[nummer].append(Wcodes['ALMELO'])
kazernes[nummer].append(Wcodes['WIERDN'])
tijden[nummer].append(10.58462)
tijden[nummer].append(14.87527)
    #TUBBRG: Reutummerweg 6
nummer = Wcodes['TUBBRG']
vutijd[nummer] = 4.320176
kazernes[nummer].append(Wcodes['WEERSL'])
kazernes[nummer].append(Wcodes['OOTMSM'])
tijden[nummer].append(13.54713)
tijden[nummer].append(14.46432)
    #HELLDN: Ninaberlaan 80
nummer = Wcodes['HELLDN']
vutijd[nummer] = 4.455963
kazernes[nummer].append(Wcodes['NIJVDL'])
kazernes[nummer].append(Wcodes['LUTTBG'])
tijden[nummer].append(10.81315)
tijden[nummer].append(14.00146)
    #DENEKP: Nordhornsestraat 66
nummer = Wcodes['DENEKP']
vutijd[nummer] = 6.292545
kazernes[nummer].append(Wcodes['NORDHO'])
kazernes[nummer].append(Wcodes['OOTMSM'])
tijden[nummer].append(14.26808)
tijden[nummer].append(15.95175)
    #NIJVDL: Kerkstraat 138 B
nummer = Wcodes['NIJVDL']
vutijd[nummer] = 5.337078
kazernes[nummer].append(Wcodes['HELLDN'])
kazernes[nummer].append(Wcodes['WIERDN'])
tijden[nummer].append(9.821511)
tijden[nummer].append(15.59732)
    #WIERDN: Tulpenstraat 1
nummer = Wcodes['WIERDN']
vutijd[nummer] = 5.418854
kazernes[nummer].append(Wcodes['ALMELO'])
kazernes[nummer].append(Wcodes['VRIEZV'])
tijden[nummer].append(10.64662)
tijden[nummer].append(14.38036)
    #ALMELO: Brugstraat 6
nummer = Wcodes['ALMELO']
vutijd[nummer] = 2.71114
kazernes[nummer].append(Wcodes['VRIEZV'])
kazernes[nummer].append(Wcodes['WIERDN'])
tijden[nummer].append(13.1841)
tijden[nummer].append(13.917)
```

```
    #WEERSL: Legtenbergerstraat 3-5
nummer = Wcodes['WEERSL']
vutijd[nummer] = 4.118008
kazernes[nummer].append(Wcodes['TUBBRG'])
kazernes[nummer].append(Wcodes['OLZAAL'])
tijden[nummer].append(13.78185)
tijden[nummer].append(14.96943)
    #DELUTT: Lossersestraat 16
nummer = Wcodes['DELUTT']
vutijd[nummer] = 7.15037
kazernes[nummer].append(Wcodes['OLZAAL'])
kazernes[nummer].append(Wcodes['LOSSER'])
tijden[nummer].append(12.65664)
tijden[nummer].append(16.22845)
    #RIJSSN: Molenstalweg 17
nummer = Wcodes['RIJSSN']
vutijd[nummer] = 5.626116
kazernes[nummer].append(Wcodes['ENTER'])
kazernes[nummer].append(Wcodes['HOLTEN'])
tijden[nummer].append(11.00583)
tijden[nummer].append(15.13008)
    #BORNE: De Bieffel 11
nummer = Wcodes['BORNE']
vutijd[nummer] = 5.482643
kazernes[nummer].append(Wcodes['HENGOVC'])
kazernes[nummer].append(Wcodes['ALMELO'])
tijden[nummer].append(11.29905)
tijden[nummer].append(12.7833)
    #OLZAAL: Ossenmaatstraat 1
nummer = Wcodes['OLZAAL']
vutijd[nummer] = 5.986021
kazernes[nummer].append(Wcodes['HENGOVC'])
kazernes[nummer].append(Wcodes['WEERSL'])
tijden[nummer].append(13.59896)
tijden[nummer].append(13.65761)
    #ENTER: Vonderweg 2
nummer = Wcodes['ENTER']
vutijd[nummer] = 4.813159
kazernes[nummer].append(Wcodes['RIJSSN'])
kazernes[nummer].append(Wcodes['GOOR'])
tijden[nummer].append(12.29278)
tijden[nummer].append(15.79629)
    #HENGOVN: Torenlaan 2a
nummer = Wcodes['HENGOVN']
vutijd[nummer] = 8.476582
kazernes[nummer].append(Wcodes['OLZAAL'])
```

```
kazernes[nummer].append(Wcodes['BORNE'])
tijden[nummer].append(14.4639)
tijden[nummer].append(14.63198)
    #HOLTEN: Eg 15
nummer = Wcodes['HOLTEN']
vutijd[nummer] = 5.990411
kazernes[nummer].append(Wcodes['MARKLO'])
kazernes[nummer].append(Wcodes['RIJSSN'])
tijden[nummer].append(13.92199)
tijden[nummer].append(14.77849)
    #HENGOVC: Lansinkseweg 59
nummer = Wcodes['HENGOVC']
vutijd[nummer] = 2.654896
kazernes[nummer].append(Wcodes['ENSCHDN'])
kazernes[nummer].append(Wcodes['BORNE'])
tijden[nummer].append(13.07496)
tijden[nummer].append(13.98006)
    #DELDEN: De Berken 2B
nummer = Wcodes['DELDEN']
vutijd[nummer] = 5.861839
kazernes[nummer].append(Wcodes['HENGOVC'])
kazernes[nummer].append(Wcodes['BORNE'])
tijden[nummer].append(11.80315)
tijden[nummer].append(16.27986)
    #LOSSER: Broekhoekweg 40
nummer = Wcodes['LOSSER']
vutijd[nummer] = 6.195663
kazernes[nummer].append(Wcodes['ENSCHDGL'])
kazernes[nummer].append(Wcodes['GRONAU'])
tijden[nummer].append(12.45946)
tijden[nummer].append(14.98517)
    #GOOR: Van Kollaan 19
nummer = Wcodes['GOOR']
vutijd[nummer] = 5.050576
kazernes[nummer].append(Wcodes['DIEPHM'])
kazernes[nummer].append(Wcodes['MARKLO'])
tijden[nummer].append(10.53203)
tijden[nummer].append(12.16796)
    #ENSCHDN: Lijsterstraat 15
nummer = Wcodes['ENSCHDN']
vutijd[nummer] = 2.612671
kazernes[nummer].append(Wcodes['ENSCHDHP'])
kazernes[nummer].append(Wcodes['HENGOVC'])
tijden[nummer].append(9.158512)
tijden[nummer].append(13.21087)
    #MARKLO: Schoolstraat 13a
```

```
nummer = Wcodes['MARKLO']
vutijd[nummer] = 5.253374
kazernes[nummer].append(Wcodes['GOOR'])
kazernes[nummer].append(Wcodes['DIEPHM'])
tijden[nummer].append(12.59203)
tijden[nummer].append(14.91039)
    #ENSCHDGL: Tolstraat 5
nummer = Wcodes['ENSCHDGL']
vutijd[nummer] = 5.907366
kazernes[nummer].append(Wcodes['ENSCHDHP'])
kazernes[nummer].append(Wcodes['GRONAU'])
tijden[nummer].append(11.06119)
tijden[nummer].append(12.38933)
    #ENSCHDBK: Verzetslaan 21
nummer = Wcodes['ENSCHDBK']
vutijd[nummer] = 0
kazernes[nummer].append(f[nummer][0])
kazernes[nummer].append(f[nummer][1])
tijden[nummer].append(T(nummer,f[nummer][0],snelheid))
tijden[nummer].append(T(nummer,f[nummer][1],snelheid))
    #ENSCHDHP: Spaansland 20
nummer = Wcodes['ENSCHDHP']
vutijd[nummer] = 3.098141
kazernes[nummer].append(Wcodes['ENSCHDN'])
kazernes[nummer].append(Wcodes['ENSCHDGL'])
tijden[nummer].append(9.326859)
tijden[nummer].append(14.98367)
    #DIEPHM: Wilsonweg 6
nummer = Wcodes['DIEPHM']
vutijd[nummer] = 4.856943
kazernes[nummer].append(Wcodes['GOOR'])
kazernes[nummer].append(Wcodes['MARKLO'])
tijden[nummer].append(10.93867)
tijden[nummer].append(15.50475)
    #HAAKSB: Parallelweg 2
nummer = Wcodes['HAAKSB']
vutijd[nummer] = 7.133388
kazernes[nummer].append(Wcodes['DIEPHM'])
kazernes[nummer].append(Wcodes['MARKLO'])
tijden[nummer].append(10.23876)
tijden[nummer].append(12.69184)

for i in range(len(W)):
    if kazernes[W[i]][0] != 'G':
        Tkazernes[i][W.index(kazernes[W[i]][0])] = tijden[W[i]][0] \\
        - vutijd[kazernes[W[i]][0]] + 1.5
```

```
if kazernes[W[i]][1] != 'G':
    Tkazernes[i][W.index(kazernes[W[i]][1])] = tijden[W[i]][1] - \\
    vutijd[kazernes[W[i]][1]] + 1.5
```

Appendix E

```
from math import sqrt
from math import ceil
from math import exp

V = [i for i in range(9)]
W = [1, 4, 5, 8]
vakkenmid = dict()
vakkenmid[0] = [500, 2500]
vakkenmid[1] = [1500, 2500]
vakkenmid[2] = [2500, 2500]
vakkenmid[3] = [500, 1500]
vakkenmid[4] = [1500, 1500]
vakkenmid[5] = [2500, 1500]
vakkenmid[6] = [500, 500]
vakkenmid[7] = [1500, 1500]
vakkenmid[8] = [2500, 500]


def afstandhb(i,j):
    return sqrt((vakkenmid[i][0]-vakkenmid[j][0])**2 + (vakkenmid[i][1] \\
    -vakkenmid[j][1])**2)

def T(i,j,snelheid):
    return afstandhb(i,j)/snelheid




B = 4

labda = [10, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

#Begintoestand
u = [1 for i in range(B)]
v = []
for i in W:
    v.append(i)
x = [0, 0, 2, 0, 0, 0, 0, 0, 0]
y = dict()
for i in V:
    if i in W:
```

```
        y[i] = 1
    else:
        y[i] = 0

b = [0 for i in V]
w = [0 for i in V]
z = dict()
for i in V:
    z[i] = 0
r = 1
snelheid = 30000/4 #m/kwartier
P = [[0 for j in range(B)] for i in V]
Q = [[0 for j in range(B)] for i in V]
m = dict()
for i in W:
    m[i] = 1
Tlim = 0.5
tau = 1

f = dict()

for i in V:
    rijtijden = dict()
    for j in W:
        rijtijden[j] = T(i,j,snelheid)
    rijtijdens = sorted(rijtijden.items(), key=lambda x: x[1])
    f[i] =[rijtijdens[i][0] for i in range(len(rijtijdens))]
    print(rijtijden)



#print(f)
g = dict()

for i in W:
    rijtijden = dict()
    for j in W:
        rijtijden[j] = T(i, j, snelheid)
    rijtijdens = sorted(rijtijden.items(), key=lambda x: x[1])
    g[i] = [rijtijdens[i][0] for i in range(1,4)]


#Tussentoestand

for i in range(len(x)):
    j = 0
```

```
    while x[i] > 0 and j < len(W):
        if y[f[i][j]] > 0:
            y[f[i][j]] -= 1
            x[i] -= 1
            w[i] += 1
            for k in range(B):
                if v[k] == f[i][j] and u[k] == 1:
                    u[k] = 2
                    Q[i][k] = int(ceil(T(v[k],i,snelheid))) + 1

        else:
            j += 1


#print(y)

#Bepaal actieruimte

def actieruimte(u,v,w,x,y,z,b,P,Q):
    result = []
    A = [[0 for j in V] for i in V]
    result.append(A)
    Amax = [[0 for j in V] for i in V]
    for i in V:
        for j in V:
            if i in W and j in W:
                Amax[i][j] = min(y[i], m[j]-z[j]-y[j])
    for i in W:
        for j in g[i]:
            for k in range(1,Amax[i][j]+1):
                A = [[0 for j in V] for i in V]
                A[i][j] += k
                result.append(A)
    return result



def costfun(t):
    gamma = 0.8
    if t <= Tlim:
        return (1/gamma)*(exp(t)-1)
    else:
        return t - (Tlim - 1)
```

```
xprimes = []

xprime = [0 for i in V]
p = 1
for i in range(len(V)):
    p *= exp(-1*labda[i])
xprime.append(p)
xprimes.append(xprime)

for i in range(len(V)):
    xprime = [0 for j in V]
    xprime[i] += 1
    p = labda[i]*exp(-1*labda[i])
    for j in range(len(V)):
        if j != i:
            p *= exp(-1*labda[j])
    denominator1 = 1
    for k in range(len(V)):
        denominator1 *= exp(-1*labda[k])
    denominator2 = 0

    for l in range(len(V)):
        prod = labda[l]*exp(-1*labda[l])
        for n in range(len(V)):
            if n != m:
                prod *= exp(-labda[n])
        denominator2 += prod
    denominator = denominator1 + denominator2
    p =  p/denominator
    xprime.append(p)
    xprimes.append(xprime)

print(xprimes)
Vdstar = 100000000


print(len(actieruimte(u,v,w,x,y,z,b,P,Q)))

for A in actieruimte(u,v,w,x,y,z,b,P,Q):
    unew = u[:]
    vnew = v[:]
    wnew = w[:]
    xnew = x[:]
    ynew = dict(y)
    znew = dict(z)
    bnew = b[:]
```

```
Pnew = []
for i in P:
    Pnew.append(i[:])
Qnew = []
for i in Q:
    Qnew.append(i[:])
freecars = dict()
for i in V:
    for j in range(B):
        if Pnew[i][j] > 1:
            Pnew[i][j] -= 1
        elif Pnew[i][j] == 1:
            Pnew[i][j] -= 1
            bnew[i] -= 1
            freecars[j] = i



        if Qnew[i][j] > 1:
            Qnew[i][j] -= 1
        elif Qnew[i][j] == 1:
            if unew[j] == 4:
                Qnew[i][j] -= 1
                unew[j] = 1
                ynew[i] += 1
                znew[i] -= 1
            elif unew[j] == 2:
                Qnew[i][j] -= 1
                unew[j] = 3
                bnew[i] += 1
                wnew[i] -= 1
                Pnew[i][j] = tau

for i in range(len(xnew)):
    while xnew[i] > 0 and len(freecars.keys()) > 0:
        afstand = 10000000
        for j in freecars.keys():
            afstandtemp = afstandhb(freecars[j],i)
            if afstandtemp < afstand:
                afstand = afstandtemp
                car = j
        xnew[i] -= 1
        wnew[i] += 1
        unew[car] = 2
        Qnew[i][car] = int(ceil(T(freecars[car], i, snelheid)))
        freecars.pop(car)
```

```
for i in freecars.keys():
    if m[vnew[i]] > ynew[vnew[i]] + znew[vnew[i]]:
        unew[i] = 4
        znew[v[i]] += 1
        Qnew[v[i]][i] = int(ceil(T(freecars[i],v[i],snelheid)))+1
    else:
        j = 0
        found = False
        while not found:
            if m[f[freecars[i]][j]] > ynew[f[freecars[i][j]]] + \\
            znew[f[freecars[i][j]]]:
                found = True
                znew[f[freecars[i]][j]] += 1
                unew[i] = 4
                Qnew[f[freecars[i]][j]][i] = int(ceil(T(freecars[i], \\
                 f[freecars[i]][j], snelheid)))
            else:
                j += 1




for i in V:
    for j in V:
        n = A[i][j]
        if n > 0:
            cars = []
            for k in range(len(v)):
                if vnew[k] == i and unew[k] == 1:
                    cars.append(k)
            for k in cars[0:n]:
                unew[k] = 1
                vnew[k] = j
            ynew[i] -= len(cars)
            ynew[j] += len(cars)
print("Action")
print(A)

Vd = 0


for xprime in xprimes:
    xprimecop = xprime[:]
    p = xprimecop.pop()
```

```
        phiR = 0

        for i in range(len(xprimecop)):
            j = 0
            while xprimecop[i] == 1 and j < len(W):
                if ynew[f[i][j]] > 0:
                    R = T(f[i][j], i, snelheid)
                    phiR += costfun(R)
                    xprimecop[i] -= 1
                else:
                    j += 1
        Vd += phiR*p
    print(Vd)

    if Vd < Vdstar:
        Vdstar = Vd
        dstar = A

print(Vdstar)
print(dstar)
```

Appendix F

```
from Locaties import V,W, vakkenmid, nummernaam
from Opkomsttijden import g, f,opkomsttijden, normtijden, kazernes, Tkazernes
from Labdaminuutvakje import labda
from math import sqrt
from math import ceil
from math import exp



def afstandhb(i,j):
    return sqrt((vakkenmid[i][0]-vakkenmid[j][0])**2 + (vakkenmid[i][1] \\
    -vakkenmid[j][1])**2)

def T(i,j,snelheid):
    return afstandhb(i,j)/snelheid


B = 29



#Begintoestand:
u = [1 for i in range(B)]
v = []
for i in W:
    v.append(i)
x = [0 for i in V]
x[3653] = 3
y = dict()
for i in V:
    if i in W:
        y[i] = 1
    else:
        y[i] = 0
b = [0 for i in V]
w = [0 for i in V]
z = dict()
for i in W:
    z[i] = 0
r = 15
snelheid = 30000/60 #m/min
P = [[0 for j in range(B)] for i in V]
```

```
Q = [[0 for j in range(B)] for i in V]
m = dict()
for i in W:
    m[i] = 1
Tlim = normtijden
tau = 10
transcost = 0



#Tussentoestand:

for i in range(len(x)):
    j = 0
    while x[i] > 0 and j < len(f[i]):
        if f[i][j] == 'G':
            x[i] -= 1
        else:
            if y[f[i][j]] > 0:
                print(nummernaam[f[i][j]])
                y[f[i][j]] -= 1
                x[i] -= 1
                w[i] += 1
                for k in range(B):
                    if v[k] == f[i][j] and u[k] == 1:
                        u[k] = 2
                        Q[i][k] = int(ceil(opkomsttijden[i][W.index(f[i][j])]))

            else:
                j += 1
    j = 0
    while x[i] > 0 and j < len(g[i]):
        if y[g[i][j]] > 0:
            y[g[i][j]] -= 1
            print(nummernaam[g[i][j]])
            x[i] -= 1
            w[i] += 1
            for k in range(B):
                if v[k] == g[i][j] and u[k] == 1:
                    u[k] = 2
                    Q[i][k] = int(ceil(opkomsttijden[i][W.index(g[i][j])])) + 1

        else:
            j += 1
```

```
#Actieruimte

def actieruimte(u,v,w,x,y,z,b,P,Q):
    result = []
    A = [[0 for j in W] for i in W]
    result.append(A)
    Amax = [[0 for j in W] for i in W]
    for i in range(len(W)):
        for j in range(len(W)):
            Amax[i][j] = min(y[W[i]], m[W[j]]-z[W[j]]-y[W[j]])
    for i in range(len(W)):
        for j in range(2):
            if kazernes[W[i]][j] != 'G':
                for k in range(1,Amax[i][W.index(kazernes[W[i]][j])]+1):
                    A = [[0 for j in W] for i in W]
                    A[i][W.index(kazernes[W[i]][j])] += k
                    result.append(A)
    print(len(result))
    return result


#Kostenfunctie:
def costfun(t,i):
    gamma = 35
    if t <= Tlim[i]:
        return (1/gamma)*(exp(t)-1)
    else:
        return t - (Tlim[i] - 1)


#Mogelijke nieuwe incidenten met bijbehorende kansen:
xprimes = []

xprime = [0 for i in V]
p = 1
for i in range(len(V)):
    p *= exp(-1*labda[i])
xprime.append(p)
xprimes.append(xprime)

for i in range(len(V)):
    xprime = [0 for j in V]
    xprime[i] += 1
    p = labda[i]
    denominator = 1 + sum(labda)
    p =  p/denominator
```

```
        xprime.append(p)
        xprimes.append(xprime)




#Beste actie selecteren:
Vdstar = 100000000
dstar =  []
it = 0

for A in actieruimte(u,v,w,x,y,z,b,P,Q):
    #Kosten voor verplaatsen auto's:
    transcosts = 0
    for i in range(len(W)):
        for j in range(len(W)):
            if A[i][j] > 0:
                transcosts += transcost * A[i][j]

    #Nieuwe toestand:
    it += 1
    unew = u[:]
    vnew = v[:]
    wnew = w[:]
    xnew = x[:]
    ynew = dict(y)
    znew = dict(z)
    bnew = b[:]
    Pnew = []
    for i in P:
        Pnew.append(i[:])
    Qnew = []
    for i in Q:
        Qnew.append(i[:])
    freecars = dict()
    for i in V:
        for j in range(B):
            if Pnew[i][j] > 1:
                Pnew[i][j] -= 1
            elif Pnew[i][j] == 1:
                Pnew[i][j] -= 1
                bnew[i] -= 1
                freecars[j] = i

            if Qnew[i][j] > 1:
```

```
                Qnew[i][j] -= 1
            elif Qnew[i][j] == 1:
                if unew[j] == 4:
                    Qnew[i][j] -= 1
                    unew[j] = 1
                    ynew[i] += 1
                    znew[i] -= 1
                elif unew[j] == 2:
                    Qnew[i][j] -= 1
                    unew[j] = 3
                    bnew[i] += 1
                    wnew[i] -= 1
                    Pnew[i][j] = tau

    #Vrijgekomen auto's eventueel naar ander incident sturen:
    for i in range(len(xnew)):
        while xnew[i] > 0 and len(freecars.keys()) > 0:
            afstand = 10000000
            for j in freecars.keys():
                afstandtemp = afstandhb(freecars[j],i)
                if afstandtemp < afstand:
                    afstand = afstandtemp
                    car = j
            xnew[i] -= 1
            wnew[i] += 1
            unew[car] = 2
            Qnew[i][car] = int(ceil(T(freecars[car], i, snelheid)))
            freecars.pop(car)



    #Vrijgekomen auto's naar kazerne sturen:
    for i in freecars.keys():
        if m[vnew[i]] > ynew[vnew[i]] + znew[vnew[i]]:
            unew[i] = 4
            znew[vnew[i]] += 1
            Qnew[vnew[i]][i] = int(ceil(opkomsttijden[freecars[i]]\\
            [W.index(vnew[i])]))+1
        else:
            j = 0
            found = False
            while j < len(f[freecars[i]]) and not found:
                if m[f[freecars[i]][j]] > ynew[f[freecars[i][j]]] + \\
                znew[f[freecars[i][j]]]:
                    found = True
                    znew[f[freecars[i]][j]] += 1
```

```
                unew[i] = 4
                Qnew[f[freecars[i]][j]][i] = int(ceil(opkomsttijden\\
                [freecars[i]][W.index(f[freecars[i]][j])]))
            else:
                j += 1
        j = 0
        while j < len(g[freecars[i]]) and not found:
            if m[g[freecars[i]][j]] > ynew[g[freecars[i][j]]] +\\
             znew[g[freecars[i][j]]]:
                found = True
                znew[g[freecars[i]][j]] += 1
                unew[i] = 4
                Qnew[g[freecars[i]][j]][i] = int(ceil(opkomsttijden \\
                [freecars[i]][W.index(g[freecars[i]][j])]))
            else:
                j += 1


#Auto's verplaatsen van kazerne naar kazerne
for i in range(len(W)):
    for j in range(len(W)):
        n = A[i][j]
        if n > 0:
            cars = []
            for k in range(len(v)):
                if vnew[k] == W[i] and unew[k] == 1:
                    cars.append(k)
            for k in cars:
                unew[k] = 4
                vnew[k] = W[j]
                Qnew[W[j]][k] = int(ceil(Tkazernes[i][j]))
            ynew[W[i]] -= len(cars)
            znew[W[j]] += len(cars)
print(it)
#print(u, v, w, y, z, b, P, Q)

Vd = 0

#Verwachte kosten:
for xprime in xprimes:
    xprimecop = xprime[:]
    p = xprimecop.pop()
    phiR = transcosts
    #print("p")
   # print(p)
```

```
            for i in range(len(xprimecop)):
                j = 0
                while xprimecop[i] > 0 and j < len(f[i]):
                    if f[i][j] == 'G':
                        xprimecop[i] -= 1
                    else:
                        if ynew[f[i][j]] > 0:
                            R = int(ceil(opkomsttijden[i][W.index(f[i][j])]))
                            phiR += costfun(R,i)
                            xprimecop[i] -= 1
                        else:
                            j += 1
                j = 0
                while xprimecop[i] > 0 and j < len(g[i]):
                    if ynew[g[i][j]] > 0:
                        R = int(ceil(opkomsttijden[i][W.index(g[i][j])]))
                        phiR += costfun(R, i)
                        xprimecop[i] -= 1
                    else:
                        j += 1
            Vd += phiR*p

        print('Beslissing:')
        print('van')
        for j in range(len(W)):
            for k in range(len(W)):
                if A[j][k] > 0:
                    print(nummernaam[W[j]])
                    print('naar')
                    print(nummernaam[W[k]])
        print('kosten:')
        print(Vd)

        if Vd <= Vdstar:
            Vdstar = Vd
            dstar.append(A)


print(Vdstar)
print(dstar)
print(len(dstar))

i = 1

#Beste beslissingen:
print("Optimale kosten:")
```

```
print(Vdstar)
for A in dstar:
    print("beslissing")
    print(i)
    print(":")
    for j in range(len(W)):
        for k in range(len(W)):
            if A[j][k] > 0:
                print(A[j][k])
                print("cars from")
                print(nummernaam[W[j]])
                print("to")
                print(nummernaam[W[k]])
    i += 1
```

Appendix G

```
from Locaties import V,W, vakkenmid, nummernaam
from Opkomsttijden import g, f,opkomsttijden, normtijden, kazernes, Tkazernes
from Labdaminuutvakje import labda
from math import sqrt
from math import ceil
from math import exp



def afstandhb(i,j):
    return sqrt((vakkenmid[i][0]-vakkenmid[j][0])**2 + (vakkenmid[i][1] \\
    -vakkenmid[j][1])**2)

def T(i,j,snelheid):
    return afstandhb(i,j)/snelheid


B = 29



#Begintoestand:
u = [1 for i in range(B)]
u[0] = 3
u[1] = 3
v = []
for i in W:
    v.append(i)
x = [0 for i in V]
x[4870] = 28
y = dict()
for i in V:
    if i in W and i != v[0] and i != v[1]:
        y[i] = 1
    else:
        y[i] = 0
b = [0 for i in V]
b[4524] = 1
b[1103] = 1
w = [0 for i in V]
z = dict()
for i in W:
    z[i] = 0
```

```
r = 15
snelheid = 30000/60 #m/min
P = [[0 for j in range(B)] for i in V]
P[4524][0] = 1
P[1103][1] = 1
Q = [[0 for j in range(B)] for i in V]
m = dict()
for i in W:
    m[i] = 1
Tlim = normtijden
tau = 10
transcost = 0



#Tussentoestand:

for i in range(len(x)):
    j = 0
    while x[i] > 0 and j < len(f[i]):
        if f[i][j] == 'G':
            x[i] -= 1
        else:
            if y[f[i][j]] > 0:
                print(nummernaam[f[i][j]])
                y[f[i][j]] -= 1
                x[i] -= 1
                w[i] += 1
                for k in range(B):
                    if v[k] == f[i][j] and u[k] == 1:
                        u[k] = 2
                        Q[i][k] = int(ceil(opkomsttijden[i][W.index(f[i][j])]))

            else:
                j += 1
    j = 0
    while x[i] > 0 and j < len(g[i]):
        if y[g[i][j]] > 0:
            y[g[i][j]] -= 1
            print(nummernaam[g[i][j]])
            x[i] -= 1
            w[i] += 1
            for k in range(B):
                if v[k] == g[i][j] and u[k] == 1:
                    u[k] = 2
                    Q[i][k] = int(ceil(opkomsttijden[i][W.index(g[i][j])])) + 1
```

```
        else:
            j += 1


#Actieruimte

def actieruimte(u,v,w,x,y,z,b,P,Q):
    result = []
    A = [[0 for j in W] for i in W]
    result.append(A)
    Amax = [[0 for j in W] for i in W]
    for i in range(len(W)):
        for j in range(len(W)):
            Amax[i][j] = min(y[W[i]], m[W[j]]-z[W[j]]-y[W[j]])
    for i in range(len(W)):
        for j in range(2):
            if kazernes[W[i]][j] != 'G':
                for k in range(1,Amax[i][W.index(kazernes[W[i]][j])]+1):
                    A = [[0 for j in W] for i in W]
                    A[i][W.index(kazernes[W[i]][j])] += k
                    result.append(A)
    print(len(result))
    return result


#Kostenfunctie:
def costfun(t,i):
    gamma = 35
    if t <= Tlim[i]:
        return (1/gamma)*(exp(t)-1)
    else:
        return t - (Tlim[i] - 1)


#Mogelijke nieuwe incidenten met bijbehorende kansen:
xprimes = []

xprime = [0 for i in V]
p = 1
for i in range(len(V)):
    p *= exp(-1*labda[i])
xprime.append(p)
xprimes.append(xprime)

for i in range(len(V)):
```

```
        xprime = [0 for j in V]
        xprime[i] += 1
        p = labda[i]
        denominator = 1 + sum(labda)
        p =  p/denominator
        xprime.append(p)
        xprimes.append(xprime)




#Beste actie selecteren:
Vdstar = 100000000
dstar =  []
it = 0

for A in actieruimte(u,v,w,x,y,z,b,P,Q):
    #Kosten voor verplaatsen auto's:
    transcosts = 0
    for i in range(len(W)):
        for j in range(len(W)):
            if A[i][j] > 0:
                transcosts += transcost * A[i][j]

    #Nieuwe toestand:
    it += 1
    unew = u[:]
    vnew = v[:]
    wnew = w[:]
    xnew = x[:]
    ynew = dict(y)
    znew = dict(z)
    bnew = b[:]
    Pnew = []
    for i in P:
        Pnew.append(i[:])
    Qnew = []
    for i in Q:
        Qnew.append(i[:])
    freecars = dict()
    for i in V:
        for j in range(B):
            if Pnew[i][j] > 1:
                Pnew[i][j] -= 1
            elif Pnew[i][j] == 1:
```

```
                Pnew[i][j] -= 1
                bnew[i] -= 1
                freecars[j] = i

            if Qnew[i][j] > 1:
                Qnew[i][j] -= 1
            elif Qnew[i][j] == 1:
                if unew[j] == 4:
                    Qnew[i][j] -= 1
                    unew[j] = 1
                    ynew[i] += 1
                    znew[i] -= 1
                elif unew[j] == 2:
                    Qnew[i][j] -= 1
                    unew[j] = 3
                    bnew[i] += 1
                    wnew[i] -= 1
                    Pnew[i][j] = tau
print("freecars:")
print(freecars)

#Vrijgekomen auto's eventueel naar ander incident sturen:
for i in range(len(xnew)):
    while xnew[i] > 0 and len(freecars.keys()) > 0:
        afstand = 10000000
        for j in freecars.keys():
            afstandtemp = afstandhb(freecars[j],i)
            if afstandtemp < afstand:
                afstand = afstandtemp
                car = j
        xnew[i] -= 1
        wnew[i] += 1
        unew[car] = 2
        Qnew[i][car] = int(ceil(T(freecars[car], i, snelheid)))
        print("auto van")
        print(freecars[car])
        print("naar")
        print(i)
        freecars.pop(car)



#Vrijgekomen auto's naar kazerne sturen:
for i in freecars.keys():
    if m[vnew[i]] > ynew[vnew[i]] + znew[vnew[i]]:
        unew[i] = 4
```

```
            znew[vnew[i]] += 1
            Qnew[vnew[i]][i] = int(ceil(opkomsttijden[freecars[i]] \\
            [W.index(vnew[i])]))+1
        else:
            j = 0
            found = False
            while j < len(f[freecars[i]]) and not found:
                if m[f[freecars[i]][j]] > ynew[f[freecars[i][j]]] + \\
                znew[f[freecars[i][j]]]:
                    found = True
                    znew[f[freecars[i]][j]] += 1
                    unew[i] = 4
                    Qnew[f[freecars[i]][j]][i] = int(ceil(opkomsttijden\\
                    [freecars[i]][W.index(f[freecars[i]][j])]))
                else:
                    j += 1
            j = 0
            while j < len(g[freecars[i]]) and not found:
                if m[g[freecars[i]][j]] > ynew[g[freecars[i][j]]] + \\
                 znew[g[freecars[i][j]]]:
                    found = True
                    znew[g[freecars[i]][j]] += 1
                    unew[i] = 4
                    Qnew[g[freecars[i]][j]][i] = int(ceil(opkomsttijden \\
                    [freecars[i]][W.index(g[freecars[i]][j])]))
                else:
                    j += 1


#Auto's verplaatsen van kazerne naar kazerne
for i in range(len(W)):
    for j in range(len(W)):
        n = A[i][j]
        if n > 0:
            cars = []
            for k in range(len(v)):
                if vnew[k] == W[i] and unew[k] == 1:
                    cars.append(k)
            for k in cars:
                unew[k] = 4
                vnew[k] = W[j]
                Qnew[W[j]][k] = int(ceil(Tkazernes[i][j]))
            ynew[W[i]] -= len(cars)
            znew[W[j]] += len(cars)
print(it)
#print(u, v, w, y, z, b, P, Q)
```

```
Vd = 0

#Verwachte kosten:
for xprime in xprimes:
    xprimecop = xprime[:]
    p = xprimecop.pop()
    phiR = transcosts
    #print("p")
   # print(p)

    for i in range(len(xprimecop)):
        j = 0
        while xprimecop[i] > 0 and j < len(f[i]):
            if f[i][j] == 'G':
                xprimecop[i] -= 1
            else:
                if ynew[f[i][j]] > 0:
                    R = int(ceil(opkomsttijden[i][W.index(f[i][j])]))
                    phiR += costfun(R,i)
                    xprimecop[i] -= 1
                else:
                    j += 1
        j = 0
        while xprimecop[i] > 0 and j < len(g[i]):
            if ynew[g[i][j]] > 0:
                R = int(ceil(opkomsttijden[i][W.index(g[i][j])]))
                phiR += costfun(R, i)
                xprimecop[i] -= 1
            else:
                j += 1
    Vd += phiR*p

print('Beslissing:')
print('van')
for j in range(len(W)):
    for k in range(len(W)):
        if A[j][k] > 0:
            print(nummernaam[W[j]])
            print('naar')
            print(nummernaam[W[k]])
print('kosten:')
print(Vd)

if Vd <= Vdstar:
    Vdstar = Vd
```

```
        dstar.append(A)


print(Vdstar)
print(dstar)
print(len(dstar))

i = 1

#Beste beslissingen:
print("Optimale kosten:")
print(Vdstar)
for A in dstar:
    print("beslissing")
    print(i)
    print(":")
    for j in range(len(W)):
        for k in range(len(W)):
            if A[j][k] > 0:
                print(A[j][k])
                print("cars from")
                print(nummernaam[W[j]])
                print("to")
                print(nummernaam[W[k]])
    i += 1
```