Vergelijken van modellen voor het aanbieden van tolken

Een wiskundig model voor Capio

Anke Gasseling, Wouter Lardinois en Eloy Stoppels

15 juni 2015

1 Abstract

Capio is een bedrijf dat een applicatie wil starten waardoor het mogelijk is om online tolken beschikbaar te stellen. Klanten kunnen een aanvraag plaatsen, vervolgens worden ze geholpen of geweigerd. Capio wil ervoor zorgen dat er zoveel mogelijk klanten geholpen kunnen worden. Dit systeem hebben we gemodeleerd als een multi skill call center. We hebben een aantal modellen gebruikt om het systeem van Capio te analyseren: Erlang Loss systeem, overflow model, trunk reservation en threshold. Bij alle modellen kan er, gegeven de aankomst van klanten en de beschikbare capaciteit aan tolken, de blokkeringskans bepaald worden. Ook kan er, gegeven de aankomst van klanten en de maximale blokkeringskans, de nodige capaciteit bepaald worden. Capio kan, afhankelijk van welk model ze het beste bij hun systeem vinden passen, de benodige capaciteit bepalen.

2 Voorwoord

Het verslag dat voor u ligt, is het resultaat van ongeveer vijf maanden onderzoek naar modellen voor het voorspellen van de benodigde capaciteit tolken voor het bedrijf Capio Live Interpreters. Deze opdracht is onze afsluiting van de bachelor Technische Wiskunde aan de Universiteit Twente te Enschede.

Wij willen in het bijzonder Richard Boucherie bedanken voor zijn begeleiding vanuit de UT en het meedenken bij het opstellen en vergelijken van de verschillende modellen. Daarnaast gaat ook onze speciale dank uit naar Geert Memelink en Marc Loermans voor hun begeleiding vanuit Capio en voor de benodigde gegevens, die nodig waren om de modellen te testen.

Inhoudsopgave

3 Introductie

Het bedrijf Capio Live Interpreters heeft het idee om een applicatie te maken, zodat een gebruiker overal ter wereld kan worden voorzien van tolkdiensten in beeld en geluid. De gebruiker die een tolk nodig heeft, is dan dus niet meer afhankelijk van de tijd en de plaats waar hij op dat moment is. Een gebruiker kan de Capio dienst zelf kosteloos downloaden. De gebruiker kan een bepaalde tijd van tevoren aangeven dat hij een tolk wil hebben, maar de gebruiker kan ook op elk moment een aanvraag indienen om op datzelfde moment een tolk te krijgen. Een aanvraag kan bestaan uit twee talen waarin de gebruiker graag een tolk wil hebben, maar ook uit bijvoorbeeld een specialisatie, zoals medisch of zakelijk. Daarnaast kan de gebruiker ook nog andere dingen aangeven, zoals geslacht of dialect. Zo kan de gebruiker zijn aanvraag heel specifiek maken. In dit verslag zullen we alleen ingaan op de keuze van de specialisatie. Het is mogelijk dat er geen tolk aan een aanvraag kan voldoen. In dat geval wordt er gekeken of de klant door een andere tolk kan worden geholpen.

Gecertificeerde tolken kunnen de Capio dienst ook kosteloos downloaden. Tolken kunnen van tevoren in hun agenda in de applicatie aangeven wanneer zij beschikbaar zijn. Echter kunnen zij zich ook op elk willekeurig moment weer aan- of afmelden. Op het moment dat er een tekort van bepaalde tolken dreigt, kan de Capio dienst niet-beschikbare tolken een notificatie sturen om aan te geven dat er een tekort dreigt aan zijn soort tolken. De tolk kan op dat moment bepalen of hij alsnog beschikbaar wil zijn.

De Capio dienst zorgt ervoor dat een gebruiker uit drie verschillende tolken kan kiezen, waarbij alle drie de tolken aan zijn aanvraag voldoen. In het geval dat er geen drie tolken aan de aanvraag kunnen voldoen, zal de gebruiker uit minder tolken kunnen kiezen. De gebruiker kan dan de tolk naar zijn wens kiezen en gebruik maken van zijn of haar vertaaldienst. Tijdens een vertaalsessie, kan een tolk zich nooit afmelden. De tolk dient altijd eerst zijn sessie met de gebruiker af te maken.

4 Probleemstelling

Het is voor Capio van belang dat er altijd genoeg tolken beschikbaar zijn om aan alle aanvraag te kunnen voldoen. Daarom hebben is voor hun van belang welke capaciteit aan tolken zij nodig zulllen hebben om aan zoveel mogelijk aanvragen te kunnen voldoen. Zij willen de kans dat een klant geblokkeerd wordt, dus geen klant aan de lijn krijgen, zo klein mogelijk hebben. Daarnaast wil Capio ervoor zorgen dat een klant uit drie tolken kan kiezen. Capio heeft daarom de vraag hoe zij zo optimaal mogelijk de drie tolken kunnen selecteren.

Voor Capio is het belangrijkste dat zij de capaciteit voor tolken goed in kunnen schatten. Vandaar dat onze prioriteit vooral zal liggen bij het voorspellen van de capaciteit, zodat er aan zoveel mogelijk aanvragen kan worden voldaan. Wij zullen verschillende modellen vergelijken, waarbij onze vergelijkingen gebasseerd zijn op het procent klanten dat wordt geblokkeerd.

Zoals al eerder is besproken, kunnen tolken zich op elk moment aan- of afmelden. Capio zal ervoor zorgen dat een tolk de aangegeven beschikbare tijd ook altijd beschikbaar zal blijven. Wij zullen dus verder niet meenemen dat een tolk zich op elk moment af kan melden. Op het moment dat er een tolk bij komt, is er natuurlijk geen probleem, omdat je dan juist meer klanten kan helpen. Wij zullen dit daarom niet meenemen in ons onderzoek.

Allereerst zullen wij in dit verslag bekijken welke onderzoeken al naar vergelijkbare problemen zijn gedaan. Wij zullen een aantal van deze modellen uitlichten en verder uitwerken. Aan de hand van deze modellen zullen wij verschillende gegevens testen en bekijken met welk model de capaciteit van tolken het beste kan worden voorspelt en waarbij het procent klanten dat wordt geblokkeerd zo klein mogelijk is. Aan de hand van deze resultaten zullen wij Capio adviseren hoe zij het beste de capaciteit van de tolken kunnen voorspellen.

5 Literatuur

Het modelleren van de aanvraag van tolken van gebruikers en het toewijzen van tolken aan een bepaalde aanvraag is nog niet eerder gedaan. Het is echter wel te vergelijken met een call center. Bij een call center zijn er klanten, die bellen en worden er bepaalde agenten aan een telefoontje toegewezen. Dit is te vergelijken met ons probleem. De gebruikers van de applicatie zijn de klanten en de tolken zijn de agenten. Het modelleren van een call center is al vaker gedaan. Er zijn wel verschillende soorten call centers. Er zijn call centers waarbij alle agenten alle telefoontjes kunnen beantwoorden, maar ook call centers waarbij er verschillende soorten agenten zijn, die verschillende soorten telefoontjes kunnen beantwoorden. Zo'n call center heeft de naam multiskill call center. Wij hebben te maken met een multiskill call center, omdat er verschillende soorten tolken zijn met verschillende skills en er verschillende soorten aanvragen komen.

O. Zeynep Aksin en Patrick T. Harker hebben een call center gemodelleerd als een specifiek type loss systeem met meerdere typen klanten die dezelfde processor delen. [3] Het model dat zij gebruiken houdt rekening met verschillende soorten telefoonlijnen in een call center. Zij nemen aan dat aankomsten volgens een Poisson proces aankomen, waarbij aankomsten van verschillende typen klanten onafhankelijk van elkaar zijn. Servicetijden van dit model hangen af van het totaal aantal klanten in het systeem. Dit is in ons probleem niet het geval. Doordat de telefoonlijnen niet voorkomen in ons probleem en de servicetijden alleen van het type klant afhangen, is dit model te verschillend van ons probleem en dus niet bruikbaar. Tevgik Aktekin en Refik Soyer hebben call centers gemodelleerd als Bayesian wachtrijmodellen met ongeduldige klanten.[4] Ze beschouwen wachtrijen waarbij tussenaankomsttijden, servicetijden en blokkeringstijden exponentieel verdeeld zijn en het systeem s verschillende servers heeft. Zij beschrijven hier verschillende Erlang modellen. Er wordt aangegeven dat het $M/M/S$ model, ook wel het Erlang C-model genoemd, een model is waarbij klanten oneindig veel geduld hebben. Het $M/M/s/s$ model, ook wel het Erlang B-model genoemd, is een wachtrijmodel waarbij klanten juist geen geduld hebben en het systeem verlaten op het moment dat zij het vol aantreffen. Daarnaast is er ook nog een Erlang A-model ($M/M/s/+M$ model), waarbij de laatste M staat voor een exponentiele verdeling van de tijd dat een klant bereid is te wachten. Zij gaan in hun artikel veel verder in op het Erlang A-model, dan op de andere twee Erlang-modellen. In ons model wordt er aangenomen dat een klant niet bereid is te wachten totdat een tolk vrij komt. Het Erlang A-model is dus niet op ons systeem van toepassing. Het Erlang B-model echter wel. Keith Ross bespreekt dit model uitvoerig in zijn boek[6], waar het onder de naam Erlang Loss model staat. In dit Erlang Loss model zijn alle klanten van hetzelfde type en kunnen alle tolken deze klanten helpen. Dit is een heel erg versimpeld model voor ons probleem. Omdat alle type klanten onafhankelijk van elkaar aankomen, kunnen er ook verschillende Erlang Loss modellen worden gebruikt, die onafhankelijk van elkaar zijn. Hiermee kunnen dan de verschillende groepen gemodelleerd worden. Erlang Loss modellen naast elkaar voor verschillende typen klanten lijkt erg op ons probleem, daarom zullen wij het Erlang Loss model gebruiken. János Sztrik heeft een dictaat uitgebracht met een overzicht van verschillende wachtrijmodellen.[5] Veel van deze wachtrijmodellen zijn niet bruikbaar voor ons systeem. Dit komt doordat er veel modellen zijn waarbij gebruik wordt gemaakt van maar één server of doordat een model een wachtrij heeft, die in ons systeem niet voorkomt. Het Erlang-Loss model, de $M/M/n/n$-wachtrij, staat hier ook in. Dit is het Erlang B-model, zoals deze hierboven ook genoemd is.

Ger Koole and Jérôme Talim hebben een multiskill call center gemodelleerd als een netwerk van wachtrijen.[1] Het systeem wordt gemodelleerd als een wachtrijsysteem waarbij geen wachtenden worden toegelaten, deze gaan direct het systeem weer uit. Zij hebben de agenten in verschillende groepen ingedeeld en dat gemodelleerd als een $G/G/R_j$-model per groep, waarbij R_j het aantal agenten in groep j is. Het $G/G/R_j$-model geeft aan dat de tussenaankomsttijden en de bedieningsduur exponentieel verdeeld zijn en dat er in een groep R_j agenten zijn. Als er een klant binnenkomt en de agentgroep waar hij normaal gesproken naar toe wordt gestuurd is vol, wordt hij naar een andere groep gestuurd. Dit wordt het overflow proces genoemd. In het artikel van Nelly Litvak, Marleen van Rijsbergen, Richard J. Boucherie en Mark van Houdenhoven zijn intensive care patiënten gemodelleerd als een overflow systeem.[9] De groep waar klanten naartoe worden doorgestuurd als hun eerste groep vol zit, wordt de overflow groep genoemd. Als er gebruikt wordt gemaakt van deze overflow methode moet er allereerst worden gekeken naar de Equivalent Random Method (ERM). De reden waarom we dit artikel gebruiken is, omdat ons model heel erg op een overflow model lijkt en dat wordt in dit artikel duidelijk uitgelegd. In het boek van Robert Cooper[8] wordt deze zogenaamde Equivalent Random Method uitgelegd. Deze methode wordt gebruikt om van een systeem zijn blokkeringskans te bepalen. Het wordt gebruikt bij een systeem waarbij de blokkeringskans niet direct kan worden uitgerekend, dit komt door het feit dat er meerdere aankomsten in een bepaalde groep zijn. Doormiddel van de Equivalent Random Method kan er van dit systeem een equivalent systeem gemaakt worden, waarbij de blokkeringskans wel uitgerekend kan worden. Dit komt omdat er dan maar van twee servers gebruik wordt gemaakt en hiervoor zijn de blokkeringsformules bekend. Het boek van Robert Cooper is gebasseerd op een boek van R.I. Wilkinson[12], hierin legt Wilkinson uit hoe hij op de Equivalent Random Method komt. In ons model is er ook nog sprake van een directe aankomst bij een overflow groep, hier hebben Maartje Zonderland, Richard Boucherie, Michael Carter en David Stanford een artikel over geschreven.[10] Hierdoor veranderen de formules die gebruikt moeten worden om de blokkeringskans uit te rekenen. Deze formules kunnen we gebruiken op het moment dat er een directe aankomst is in de overflow groep. Om het overflow model goed te kunnen benaderen, hebben we nog artikelen van Cooper [8] en Rapp [13]geraadpleegd. Hierin worden verschillende benaderingen gegeven die van belang zijn om de blokkeringskans uit te kunnen rekenen. Rapp [13] geeft in zijn artikel namelijk een methode hoe je het gemiddelde en de variantie van een systeem kunt benaderen. Cooper [8] geeft in zijn artikel een betere benadering hiervoor die gebasseerd is op het artikel van Rapp. Daarnaast geeft Wilkinson in zijn boek ook grafieken waaruit je het gemiddelde en de variantie kunt afleiden.

Het is mogelijk om het syteem dat we hebben als een knapsack-model te bekijken. Hierin hebben we een groep van agenten die allemaal verschillende skills hebben en er komen een aantal soorten aanvragen binnen. Dit komt overeen met het idee van tolken die specifieke aanvragen krijgen. In het boek van Keith Ross [6] wordt het knapsack model uitvoerig behandeld. Er zijn verschillende manieren om dit knapsack model te bekijken en te berekenen. Wij hebben gekozen om trunk reservation en threshold te gebruiken. Beide zullen uitvoerig behandeld worden in het hoofdstuk model. Het idee van beide politieken is dat niet de complete capaciteit voor alle soorten call types kan worden ingezet, maar dat een deel gereserveerd is voor specifieke call types. Alles wat we nodig hebben voor het uitrekenen van trunk reservation staat in het boek van Keith Ross [6]. Voor het uitrekenen van alle waarden van threshold wordt gebruik gemaakt van Monte Carlo Sommatie. De Monte Carlo Sommatie wordt we-

derom behandeld in het boek van Keith Ross in [6]. Bij de Monte Carlo Sommatie wordt er geprobeerd de blokkeringskans en andere waarden te schatten en hiervoor een betrouwbaarheidsinterval te bepalen. Voor deze Monte Carlo methode is het noodzakelijk om een aantal random toestanden te genereren die een bepaalde kansdichtheid hebben. Hiervoor hebben we de alias methode gebruikt, deze wordt behandeld in het artikel van J. Cole Smith en Sheldon H. Jacobson [11].

Wyean Chan, Ger Koole en Pierre L'Ecuyer hebben een artikel geschreven waarbij zij ingaan op het zogenaamde 'routing problem' en de bijbehorende 'routing policies', waarbij het aantal agenten en hun bijbehorende skills van tevoren bekend zijn en vast blijven voor de gehele periode.[2] Een aantal politieken zou nuttig kunnen zijn voor ons model, een aantal ook totaal niet. De policies die in het artikel behandeld worden, werken vooral goed bij kleine tot middelmatig grootte van systemen. Ons systeem is ook klein tot middelmatig. In het systeem zijn er drie belangrijke 'performance measures', namelijk 'service level' (wachtijd van klanten, niet belangrijk voor ons systeem), 'abandonment ratio' (fractie klanten die geblokkerd worden) en 'occupancy ratio (bezettingsgraad van de agenten). Dit levert een optimalisatie functie, namelijk de sommatie van alle drie de 'performance measures' (in ons geval dus twee). In het artikel wordt er gewerkt met een penalty systeem en zal de optimalisatie functie geminimaliseerd moeten worden. Er worden in het artikel verschillende mogelijke policies genoemd, bijvoorbeeld 'FCFS' (First-come, first-served) of 'priority policies'. Een aantal van deze policies maken gebruik van een 'delay', hierbij wordt er soms even gewacht totdat een agent aan een vraag gekoppeld wordt, hierdoor kan het zo zijn dat een geschiktere agent de vraag kan beantwoorden. In ons systeem zullen klanten niet wachten en werken dit soort 'delay policies' niet. In ons systeem zijn er wel aanvragen die een hogere prioriteit hebben dan andere aanvragen en zullen sommige van deze policies goed van pas kunnen komen. Mocht er nog worden gesimuleerd voor ons systeem, is dit artikel erg handig. Het is verder uitgewerkt in de appendix.

6 Model

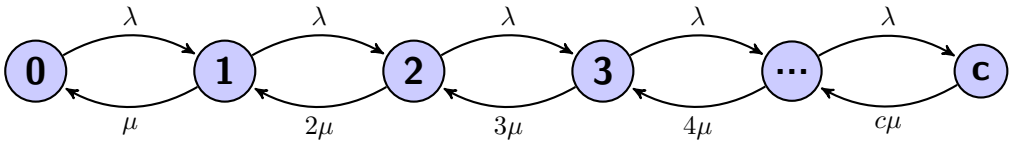
Aan de hand van de literatuur hebben we verschillende modellen gekozen om verder uit te werken. Allereest zal het Erlang loss model besproken worden. Dit model zal worden bekeken voor een systeem waarbij alle agenten alle verschillende klanten kunnen helpen. Voor verschillende soorten agentgroepen, kunnen vervolgens verschillende Erlang loss modellen naast elkaar worden gebruikt. Als tweede zal het overflow model verder worden uitgewerkt. Het overflow model maakt gebruik van een zogenaamde overflow groep. Dit model gaat verder op het Erlang loss model door extra groepen toe te voegen. Vervolgens gaan we kijken naar trunk reservation. Als laatste zal er worden gekeken naar het knapsack model met threshold politieken.

6.1 Erlang Loss model

Het Erlang Loss system is een $M/M/c/c$-wachtrij. Hierbij geeft de eerste M aan dat er aangenomen wordt dat klanten met een Poisson proces aankomen. De tweede M geeft aan dat de service tijd exponentieel verdeeld is. Er zijn c servers beschikbaar en het aantal klanten kan maximaal c zijn. Het model heeft geen wachtrij, waardoor een klant geblokkeerd wordt als er al c klanten worden geholpen. Het is belangrijk het aantal geblokkeerde klanten zo klein mogelijk te houden. Daarom is het belangrijk te kijken naar het deel van de klanten dat wordt geblokkeerd. Hier wordt echter geen verschil gemaakt tussen de verschillende soorten servers. Echter kun je wel verschillende Erlang Loss modellen naast elkaar zetten. Het is dan niet mogelijk dat een server meerdere skills heeft. Dit proces, waarbij het proces in staat n is als er n klanten in het systeem zijn, is een geboorte-sterfte proces met intensiteiten:

$$\lambda_n = \begin{cases} \lambda, & \text{als } n < c, \\ 0, & \text{als } n \geq c, \end{cases}$$

$$\mu_n = n\mu, \quad k = 1, 2, ..., n.$$



De toestandsruimte van dit geboorte-sterfte proces is eindig en dus bestaat de evenwichtsverdeling. Hieruit kan dan de evenwichtsverdeling gehaald worden. De evenwichtverdeling ziet er dan als volgt uit:

$$P_n = \begin{cases} P_0(\frac{\lambda}{\mu})^n \frac{1}{n!}, & \text{als } n \leq c, \\ 0, & \text{als } n > c. \end{cases}$$

met

$$P_0 = \frac{1}{\sum_{n=0}^{c}(\frac{\lambda}{\mu})^n \frac{1}{n!}}.$$

Deze vergelijkingen kunnen bij elkaar ingevuld worden. Dit leidt dan tot de zogenaamde Erlang-B formule, ook wel genoteerd als $B(n, \frac{\lambda}{\mu})$:

$$P_n = \frac{(\frac{\lambda}{\mu})^n \frac{1}{n!}}{\sum_{n=0}^{c} (\frac{\lambda}{\mu})^n \frac{1}{n!}} = \frac{\frac{\rho^n}{n!}}{\sum_{n=0}^{c} \frac{\rho^n}{n!}}, \quad n \leq c$$

Deze formule geeft dus de fractie van de tijd dat het systeem in staat n is. Hiermee kan dus ook de fractie van de tijd dat het systeem in staat c is, worden berekend. Dat is dan gelijk aan de kans dat een klant geblokkeerd wordt. Deze formule is dus erg belangrijk, omdat je deze blokkeringskans zo klein mogelijk wil houden.

Aan dit systeem kan vervolgens nog een stuurparameter worden toegevoegd, waardoor je op een eenvoudige markovbeslissingsproces komt. Hierbij kunnen verschillende soorten acties worden toegevoegd, zeg actie 0 en actie 1. Zo kun je bijvoorbeeld een actie maken dat er een agent bij wordt gehaald. De actie hangt af in welke toestand het systeem zit. Daarnaast kun je aan de verschillende acties kosten hangen, waardoor je de ene actie 'duurder' maakt dan de andere.

Dit model is een zeer vereenvoudigd model van ons probleem. Het voordeel hiervan is wel dat hier bepaalde strategieën op getest kunnen worden, zodat er gekeken kan worden wat die strategiën doen en of ze eventueel van waarde kunnen zijn. Bij dit model zijn de blokkeringskansen eenvoudig te berekenen en zijn strategiën dus gemakkelijker te testen. Vervolgens kunnen dan in één van de andere modellen deze strategiën toegepast worden.

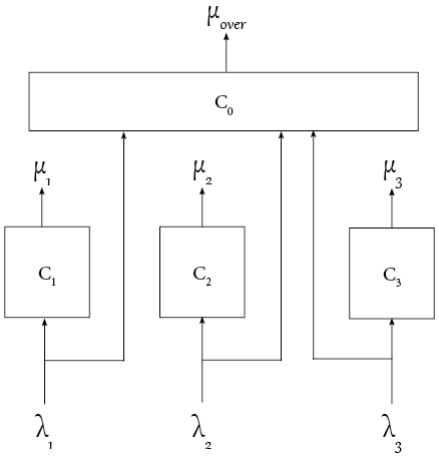
6.2 Overflow model

In deze sectie gaan we kijken naar het zogenaamde overflow model. Bij het overflow model wordt er gebruikt gemaakt van een zogenaamde overflow groep. Als klanten bij hun eigen groep geblokkeerd worden, kunnen ze nog behandeld worden bij de overflow groep. Om dit model goed te kunnen behandelen introduceren we eerst de Equivalent Random Method. Dit is ook een overflow model alleen is dit model niet uitgebreid genoeg voor ons. Maar deze hebben we wel nodig om daarna verder te gaan met het overflow model wat bij ons model voor Capio hoort.

6.2.1 Equivalent Random Method (ERM)

Een 'primary' groep is een eerste groep waar een klant naartoe wordt gestuurd. Wanneer er sprake is van n primary groepen waarvoor geldt dat als een groep vol zit, de klant deze groep als bezet ziet, kan gebruik worden gemaakt van een zogenaamd overflow systeem. De klanten die dus niet geholpen kunnen worden door hun eigen groep, worden dan geholpen door de overflow groep. Klanten die de overflow groep ook bezet vinden verlaten het systeem en worden gezien als 'verloren'. Een bepaalde manier om met deze overflow om te gaan is door gebruik te maken van de 'Equivalent Random Method'.[12]
Ons uiteindelijke doel is om te weten te komen hoeveel klanten verloren gaan. Een manier hiervoor is om de evenwichtsvergelijkingen van dit systeem op te stellen. Het systeem waar

we de evenwichtsvergelijkingen van opstellen is die van figuur 1.



Figuur 1: Een eenvoudig overflow model

Allereerst wordt de toestand weergegeven en de restricties die hierbij gelden:

$$S := (\mathbf{n} = n_{01}, ..., n_{0I}, n_1, ..., n_I), \quad n_i \leq c_i \; \forall i$$
$$\sum_{i=1}^{I} n_{0i} \leq c_0 \; \forall i$$

Waarbij n_{0i} staat voor het aantal klanten in de overflow groep, die afkomstig zijn van groep i en n_i het aantal klanten in groep i is. Daarnaast wordt de capaciteit van iedere groep weergegeven door c_i.

In de evenwichtsvergelijking komen λ_i en μ_i voor. Dit zijn respectievelijk de aankomstintensiteit en bedieningsduur van groep i, waarbij geldt dat a_i gelijk is aan $\frac{\lambda_i}{\mu_i}$. e_i stelt een gebeurtenis met klant type i voor.

$$\Pi(\mathbf{n})[\sum_{i=1}^{I} \lambda_i \mathbb{1}_{(n_i \leq c_i, \sum_{i=1}^{I} n_{0i} \leq c_0)} + \sum_{i=1}^{I} n_i \cdot \mu_i + \sum_{i=1}^{I} n_{0i} \cdot \mu_{over}] =$$
$$[\sum_{i=1}^{I} \lambda_i (\Pi(\mathbf{n} - e_i)\mathbb{1}_{(n_i > 0)} + \Pi(\mathbf{n} - e_{i0})\mathbb{1}_{(n_{0i} > 0,\, n_i = c_i)})] +$$
$$[\sum_{i=1}^{I} (n_i + 1) \cdot \mu_i \cdot \Pi(\mathbf{n} + e_i)\mathbb{1}_{(n_i+1) \leq c_i} + \sum_{i=1}^{I} (n_{0i} + 1) \cdot \mu_{over} \cdot \Pi(\mathbf{n} + e_{0i})\mathbb{1}_{(n_{0i}+1) \leq c_o}]$$

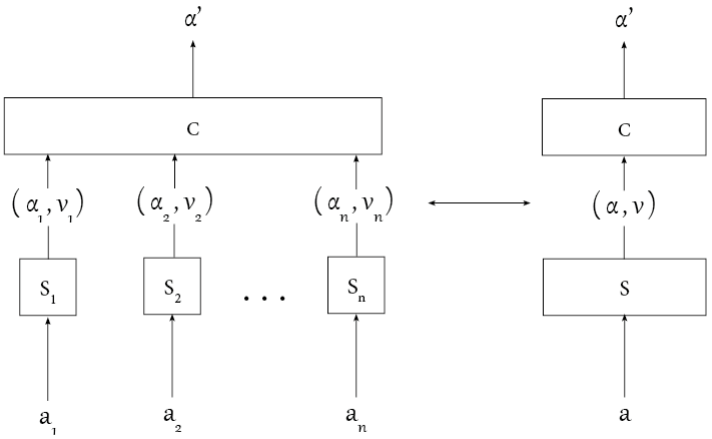
Deze evenwichtsvergelijking, in de vorm zoals die nu staat, is numeriek lastig op te lossen. Het is makkelijker om er een systeem van te maken waarbij er maar één primary groep is en één overflow groep.
Voor een dergelijk overflow systeem, die opgebouwd wordt uit een 'single s-server primary'

groep en een single c-server overflow groep geldt[8]:
1. De overflow stroom is niet Poisson.
2. De fractie van de klanten, waarbij gegeven is dat ze de overflow zijn van de primary groep, die geblokkeerd worden bij de overflow groep wordt gegeven door:

$$\Pi_c = \frac{aB(s+c,a)}{aB(s,a)} \tag{1}$$

Waarbij $a = \frac{\lambda}{\mu}$ en B staat voor de zogenaamde Erlang B-formule, zoals deze al eerder is besproken. De Erlang formule geeft de blokkeringskans in een bepaalde toestand weer. Als je dit keer de aankomstintensiteit doet krijg je als resultaat het totaal aantal klanten dat gemiddeld geblokkeerd wordt in een bepaalde toestand. In figuur 2 is weergegeven waar s,c en a zich bevinden in het systeem. Dit wordt later nader uitgelegd.

In het boek van Cooper[8] wordt er gebruik gemaakt van figuur 2. Dit figuur lijkt op figuur 1 alleen is er hier gekozen voor a_i en geeft het weer wat de bedoeling is van de ERM.



Figuur 2: De bedoeling van ERM

In figuur 2 zijn er n primary groepen, waarvan de overflow van die groepen doorstroomt naar de overflow groep. Deze overflow groep bestaat uit c servers. In de figuur wordt uitgegaan van Poisson aankomstintensiteiten en de bedieningsduren zijn onafhankelijk en identiek exponentieel verdeeld. Dit wordt samen aangegeven met a_i Erl. Dit wordt ook wel de 'load' van de groep genoemd. Het doel van ERM is dit systeem zo goed mogelijk te benaderen met een systeem waarmee wel gerekend kan worden. Hiervoor is α_i het gemiddeld aantal klanten dat geblokkeerd wordt in deze groep en v_i de variantie die daarbij hoort, nodig. Deze worden op de volgende manier bepaald:

$$\alpha_i = a_i B(s_i, a_i), \quad (i = 1, 2, ..., n), \tag{2}$$
$$v_i = \alpha_i(1 - \alpha_i + \frac{a_i}{s_i + 1 + \alpha_i - a_i}), \quad (i = 1, 2, ...n). \tag{3}$$

Een afleiding van deze formules is door Wilkinson[12] gegeven.

De gemiddelde waarde van de overflow van de overflow groep, dit zijn dus alle klanten die de overflow groep bezet vinden, wordt dan gegeven door:

$$\frac{\alpha^{'}}{\sum_{i=1}^{n} \alpha_i} \tag{4}$$

Waarbij $\alpha^{'}$ staat voor het gemiddeld aantal klanten dat het systeem verlaat. Deze formule geeft dus eigenlijk aan wat de fractie klanten is die de overflow groep bezet vindt, gegeven dat hij zijn primary groep ook bezet vindt. Dit is een benadering voor de kans Π_c.

Nu rest ons alleen nog deze $\alpha^{'}$ te vinden. Om dit te doen worden de n primary groepen door een enkele equivalent random primary groep, met bijpassende 'load' vervangen. Dit is ook te zien in het rechterdeel van figuur 2 van een ERM model.

Omdat alle primary groepen onafhankelijk van elkaar zijn, kunnen we voor het totale gemiddelde en variantie, de individuele waarden bij elkaar optellen[8]:

$$\alpha = \alpha_1 + ... + \alpha_n \tag{5}$$
$$v = v_i + ... + v_n \tag{6}$$

Omdat men α en v weet, moeten a en s op de volgende manier gekozen worden. Ze moeten namelijk voldoen aan de vergelijkingen:

$$\alpha = aB(s, a) \tag{7}$$
$$v = \alpha(1 - \alpha + \frac{a}{s + 1 + \alpha - a}) \tag{8}$$

Waarbij a de load is van de equivalent random primary groep en s de capaciteit van de equivalent random primary groep is.
Met deze formules kunnen we een versimpelde formule voor $\alpha^{'}$ opstellen, namelijk:

$$\alpha^{'} = aB(s+c,a) \tag{9}$$

Voor ons nieuwe systeem geldt dan:

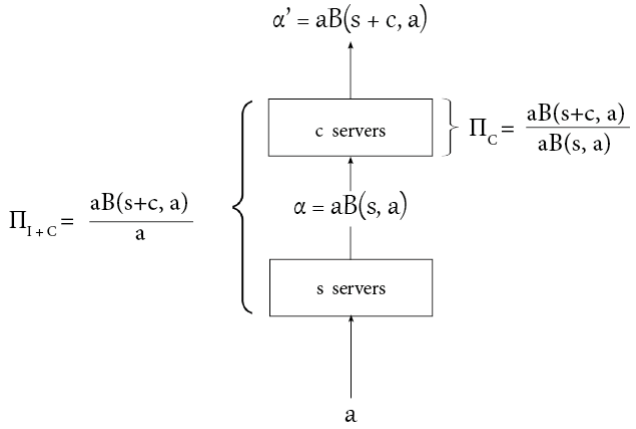
$$\Pi_c = \frac{aB(s+c,a)}{aB(s,a)} = \frac{\alpha^{'}}{\alpha} \tag{10}$$

Deze formule is hetzelfde als die van (4) waarbij de noemer van (4) vervangen is door (5). Daarnaast is er gebruik gemaakt van (7) en (9).

Tevens kan het deel van de klanten dat geblokkeerd wordt van het hele systeem uitgerekend worden. Dit is dus de fractie klanten die geblokkeerd wordt bij de primary groep en bij de overflow groep. De formule hiervoor is als volgt:

$$\Pi = \frac{aB(s+c,a)}{\sum_{i=1}^{n} a_i} \tag{11}$$

Figuur 3 geeft aan waar deze formules zich bevinden in het equivalente systeem.



Figuur 3: De formules weergegeven in een figuur

Nu rest ons alleen nog de vraag hoe formule (7) en (8) te berekenen, hiervoor zijn verschillende manieren, namelijk Wilkinson en Rapp. Wilkinson [12] heeft allerlei grafieken gegeven waarbij je het gemiddelde en de variantie kunt aflezen gebruik makend van de oude capaciteit en de aankomst load. Rapp [13]geeft een schatting hoe de a en s te benaderen zijn, dusdanig

dat het een redelijk goede benadering is van (7) en (8). De schatting voor a is gegeven door:

$$a = v + 3\frac{v}{\alpha}(\frac{v}{\alpha} - 1) \tag{12}$$

De schatting voor s wordt gegeven door:

$$s = \frac{a(\alpha + \frac{v}{\alpha})}{\alpha} + \frac{v}{\alpha - 1} - \alpha - 1 \tag{13}$$

Cooper[8] zegt dat de benaderingen die je nu gemaakt hebt redelijk aan de hoge kant liggen, daarom is het beter om s naar beneden af te ronden naar een geheel getal. Cooper geeft dan een nieuwe formule om de load a te benaderen:

$$a = \frac{(\lfloor s \rfloor + \alpha + 1)(\alpha + \frac{v}{\alpha} - 1)}{\alpha + \frac{v}{\alpha}} \tag{14}$$

Nu heb je redelijk goede benaderingen voor a en s. Met deze waarden kun je verder werken aan de ERM. Tevens kun je nu $\alpha^{'}$ van (9) bepalen en hiermee ook de fractie van geblokkeerde klanten van de overflow groep, zie formule (10).

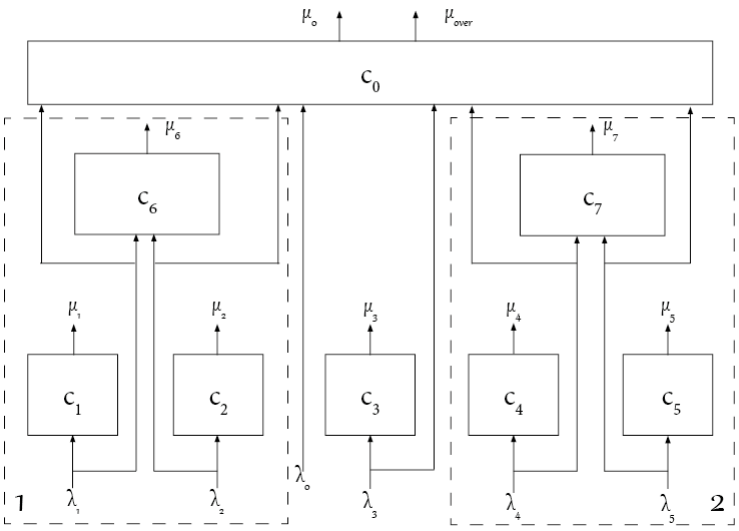
Het fractie van geblokkeerde klanten van iedere individuele groep kan ook nog uitgerekend worden, namelijk met de Katz approximation[10]. De Katz approximation geeft de blokkeringskans van de overflow van primary groep i bij de overflow groep. Je kunt dus per soort 'primary' groep kijken hoeveel klanten van deze groep geblokkeerd worden in de overflow groep. Stel een groep heeft een hele hoge blokkeringskans dan betekent dit dat de capaciteit van deze groep niet groot genoeg is. De Katz approximation maakt gebruik van de zogenaamde 'peakness' dit is; $\zeta_i = \frac{v_i}{\alpha_i}$ en $\zeta = \frac{v}{\alpha}$. Dan wordt de fractie van geblokkeerde klanten van primary groep i gegeven door:

$$k_i = \frac{aB(s+c,a)}{aB(s,a)} \cdot (V(s,c)^{-1} + \frac{\zeta_i - 1}{\zeta - 1} \cdot (1 - V(s,c)^{-1}) \tag{15}$$

Waarbij $V(s,c)$ recursief berekend kan worden door:

$$V(s,j) = \frac{a \cdot j}{aB(s,a) \cdot (s + j - a - aB(s,a) \cdot V(s,j-1))}, \qquad j = 1, 2, ..$$
$$V(s,0) = 1$$

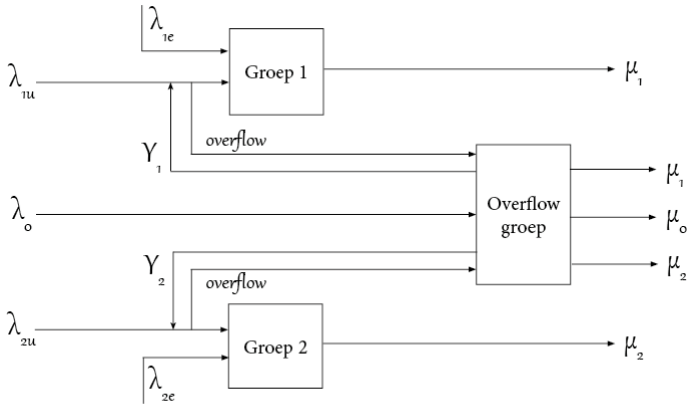
Op het moment dat we het systeem van figuur 1 uitbreiden tot figuur 4 verandert er weinig aan de te volgen methode, zoals hierboven is uitgelegd. Het enige dat verandert, is dat het systeem in stukjes moet worden opgedeeld, dusdanig dat ieder nieuw stukje overheen komt met het algemene ERM model. Dit betekent dat we figuur 4 opdelen in twee gedeelten, aangegeven met de cijfers 1 en 2. Allereerst passen we ERM toe op deze gedeelten. Dit levert precies twee nieuwe servers op. Met deze nieuwe servers en de bovenste server kan weer een ERM uitgevoerd worden. Het komt er dus op neer dat er meerdere keren ERM wordt uitgevoerd dusdanig dat er een equivalent nieuw systeem ontstaat waarbij weer het algemene ERM model kan worden gebruikt en dus ook de bijbehorende fractie van geblokkeerde klanten kan worden uitgerekend.



Figuur 4: ERM vaker toepassen

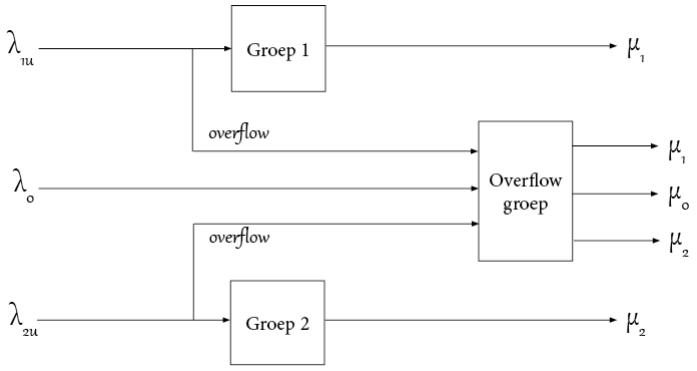
6.2.2 Het uiteindelijke overflow model

Dit model is het zelfde opgebouwd als in de sectie hiervoor alleen het verschil zit hem erin dat er een directe aankomst is in de overflow groep. Een bepaalde manier om deze soort overflow modellen te behandelen is met de Equivalent random method en deze dan aan te passen naar een model waarbij ook een directe aankomst is. In [10] wordt dit uitgelegd voor het soort systeem dat weergegeven is in figuur 5:



Figuur 5: Overflow model met directe aankomst

Als we een versimpelde versie van ons model pakken, komen we uit op een model, zoals in figuur 6. Dit lijkt heel erg op het model van 5 alleen is er geen sprake van terugkoppeling en zijn er in een bepaalde groep niet twee, maar maar één aankomst. Aan de hand van de formules die gegeven zijn in [10] kan er aan ons model gewerkt worden. Voor dit model kan de Markovketen worden gemaakt. Vanuit deze Markovketen kan er gewerkt worden naar de



Figuur 6: Versimpelde versie van ons model

evenwichtsvergelijkingen en de andere belangrijke formules.

Het model wordt dus opgedeeld in drie groepen, twee zogenaamde 'belangrijke' groepen en de overflow groep. In de belangrijke groepen komen alleen de calls aan die geschikt zijn voor deze groepen. De overflow van deze groepen, de calls die niet behandeld kunnen worden gaan naar de overflow groep. In de overflow groep komen ook de standaard calls aan. De calls die niet meer de overflow groep in kunnen, omdat alle tolken bezet zijn, gaan verloren. In [10] gaan ze ervan uit dat de overflow groep een oneindige grootte heeft, dit doen ze omdat dan met de ERM gerekend kan worden. Om later wel de blokkeringskans uit te rekenen wordt wel weer gebruikt gemaakt van een eindige groepsgrootte.

We gaan nu alle formules die we gevonden hebben in [10] en [9] aanpassen naar ons versimpelde model. Laat λ_i de aankomstintensiteit zijn van calltype i en μ_i^{-1} het gemiddelde zijn van de exponentieële bedieningsduur. Laat R_j de maximale capaciteit zijn van groep j zijn. Om onze toestand goed te kunnen definiëren gaan we er vanaf nu vanuit dat call i als eerste wordt behandeld door de bijhorende groep j. Dus als i staat voor type i, dan is groep j de groep die skill i heeft. We zeggen dus dat i=j. De toestandruimte voor dit overflow proces wordt dan:

$$S := (\mathbf{n} = n_{00}, n_{01}, ..., n_{0I}, n_1, ..., n_I), \quad n_i \leq R_i \ \forall i$$

$$\sum_{i=0}^{I} n_{0i} \leq R_0 \ \forall i \tag{16}$$

Waarbij n_{00} staat voor het aantal 'normale' klanten in de overflow groep, n_{0i} staat voor het aantal klanten van groep i die zich bevinden in de overflow groep en n_i is het aantal klanten in groep i. De evenwichtsvergelijking komt er dan als volgt uit te zien:

$$\Pi(\mathbf{n})[\sum_{i=1}^{I}\lambda_i \mathbb{1}_{(n_i \leq R_i, \sum_{i=0}^{I} n_{0i} < R_0)} + \lambda_0 \mathbb{1}_{(\sum_{i=0}^{I} n_{0i} < R_0)} + \sum_{i=1}^{I} n_i \cdot \mu_i + \sum_{i=1}^{I} n_{0i} \cdot \mu_{over} + n_{00} \cdot \mu_0] =$$
$$[\sum_{i=1}^{I}\lambda_i \cdot (\Pi(\mathbf{n} - e_i)\mathbb{1}_{n_i>0} + \Pi(\mathbf{n} - e_{0i})\mathbb{1}_{n_{0i}>0, n_i=R_i}) + \lambda_0 \cdot \Pi(\mathbf{n} - e_{00})\mathbb{1}_{n_{00}>0}]$$
$$+[\sum_{i=1}^{I}(n_i+1) \cdot \mu_i \cdot \Pi(\mathbf{n} + e_i)\mathbb{1}_{(n_i+1)\leq R_i} + \sum_{i=1}^{I}(n_{0i}+1) \cdot \mu_{over} \cdot \Pi(\mathbf{n} + e_{0i})\mathbb{1}_{(n_{0i}+1)\leq R_0} +$$
$$(n_{00} \cdot \mu_0)\Pi(\mathbf{n} + e_{00})\mathbb{1}_{(n_{00}+1)\leq R_0}]$$

Waarbij de stapfunctie de waarde 1 aanneemt als aan het argument is voldaan en de waarde 0 aanneemt als er niet aan is voldaan.

e_{00} is een gebeurtenis die plaatsvindt met een normale klant, e_i is een gebeurtenis van een klant in groep i en e_{0i} staat voor een gebeurtenis waarbij er iets gebeurt met een klant van type i in de overflow groep. Een gebeurtenis kan zijn een aankomst dan wel een vertrek. Deze formule is alleen op te lossen voor speciefieke gevallen. Maar deze restricties gelden voor ons probleem niet. Om ons probleem op te lossen moet de formule verder uitgewerkt worden.

We gaan gebruik maken van de kansgenerende functie, deze functie gebruiken we omdat we dan ERM kunnen toepassen op ons systeem. Hiervoor bepalen we het gemiddelde en de variantie, hoe men dit moet doen wordt later uitgelegd. Omdat we er eerst van uitgaan dat de capaciteit van de overflow groep oneindig is, kan de evenwichtsvergelijking versimpeld worden. Hieronder staat deze versimpelde formule, dit is de globale evenwichtsvergelijking:

$$\Pi(n_{0i}, n_i)(\lambda_i + n_i \cdot \mu_i + n_{0i} \cdot \mu_{over}) =$$
$$\lambda_i \Pi(n_{0i}, n_i - 1) + (n_{0i} + 1) \cdot \mu_{over} \Pi(n_{0i} + 1, n_i) + (n_i + 1) \cdot \mu_i \Pi(n_{0i}, n_i + 1)$$
$$\text{voor } n_i \leq R_i$$
$$\Pi(n_{0i}, n_i)(\lambda_i + n_i \cdot \mu_i + n_{0i} \cdot \mu_{over}) =$$
$$\lambda_i \Pi(n_{0i}, n_i - 1) + (n_{0i} + 1) \cdot \mu_{over} \Pi(n_{0i} + 1, n_i) + \lambda_i \Pi(n_{0i} - 1, n_i)$$
$$\text{voor } n_i = R_i$$

Nu introduceren we de kansgenerende functie, deze is als volgt gedefinieerd:

$$G_{i,n_i(z)} = \sum_{n_{0i}=0}^{\infty} \Pi(n_{0i}, n_i) z^{n_{0i}}, |z| \leq 1 \tag{17}$$

De kansgenerende functie geeft het aantal klanten van groep i die aanwezig is in de overflow groep. Voor deze functie geldt dat$G_{i,n_i(z)} = E(z^{n_{0i}})$. De kansgenerende functie wordt gebruikt

om het gemiddelde, E_i, en de variantie, V_i, van de overflow van primary groep i te bepalen. Dit in het geval dat we uitgaan van een oneindige overflow groep. Deze bepalen we omdat we dan uiteindelijk, analoog aan de ERM methode, een nieuw systeem kunnen opstellen. Het gemiddelde en de variantie kunnen op de volgende manier berekend worden:

$$E_i = \sum_{n_i=0}^{R_i} \frac{d}{dz} G_{i,n_i(z)}|_{z=1} \tag{18}$$

$$V_i = \sum_{n_i=0}^{R_i} \frac{d^2}{dz^2} G_{i,n_i(z)}|_{z=1} + E_i - (E_i)^2 \tag{19}$$

Als we nu de globale evenwichtsvergelijking vermenigvuldigen met $z^{n_{0i}}$ en daarna sommeren over alle mogelijke waarden van i, krijgen we de volgende formules:

$$[\lambda_i + n_i \cdot \mu_i] G_{i,n_i}(z) + \mu_{over} \cdot (z-1) \cdot \frac{d}{dz} G_{i,n_i}(z) =$$
$$\lambda_i G_{i,n_{i+1}}(z) + (n_i+1) \cdot \mu_i G_{i,n_{i+1}}(z) \text{ voor } 0 \leq n_i < R_i \tag{20}$$
$$[\lambda_i \cdot (1-z) + n_i \cdot \mu_i] G_{i,n_i}(z) + \mu_{over} \cdot (z-1) \cdot \frac{d}{dz} G_{i,n_i}(z) =$$
$$\lambda_i G_{i,n_i-1}(z) \text{ voor } n_i = R_i \tag{21}$$

Omdat we weten hoe we E_i en V_i moeten uitrekenen nemen we de afgeleide over z van de formules hierboven en evalueren we deze bij $z = 1$, we krijgen nu:

$$[\lambda_i + n_i \cdot \mu_i + \mu_{over}] \frac{d}{dz} G_{i,n_i}(z)|_{z=1} =$$
$$\lambda_i \frac{d}{dz} G_{i,n_i-1}(z)|_{z=1} + (n_i+1) \cdot \mu_i \frac{d}{dz} G_{i,n_i+1}(z)|_{z=1} \text{ voor } 0 \leq n_i < R_i \tag{22}$$
$$[\mu_{over} + n_i \cdot \mu_i] \frac{d}{dz} G_{i,n_i}(z)|_{z=1} - \lambda_i \cdot B(R_i, \rho_i) =$$
$$\lambda_i \frac{d}{dz} G_{i,n_i-1}(z)|_{z=1} \text{ voor } n_i = R_i \tag{23}$$

Waarbij $\rho_i = \frac{\lambda_i}{\mu_i}$ en B weer voor de Erlang formule staat.

Daarnaast zijn we ook geïnteresseerd in de tweede afgeleide over z, deze evalueren we weer bij $z = 1$.

$$[\lambda_i + n_i \cdot \mu_i + 2 \cdot \mu_{over}] \frac{d^2}{dz^2} G_{i,n_i}(z)|_{z=1} =$$
$$\lambda_i \frac{d^2}{dz^2} G_{i,n_i-1}(z)|_{z=1} + (n_i+1) \cdot \mu_i \frac{d^2}{dz^2} G_{i,n_i+1}(z)|_{z=1} \text{ voor } 0 \leq n_i < R_i \tag{24}$$
$$[2 \cdot \mu_{over} + n_i \cdot \mu_i] \frac{d^2}{dz^2} G_{i,n_i}(z)|_{z=1} - 2 \cdot \lambda_i \frac{d}{dz} G_{i,n_i}(z) =$$
$$\lambda_i \frac{d^2}{dz^2} G_{i,n_i-1}(z)|_{z=1} \text{ voor } n_i = R_i \tag{25}$$

Met:

$$\frac{d}{dz}G_{i,n_i}(z)|_{z=1} = \sum_{n_{0i}=0}^{\infty} n_{0i} \cdot \Pi(n_{0i}, n_i) z^{n_{0i}-1},\ z = 1 \tag{26}$$

Dit noteren we ook wel als $g_i(n_i)$. Waarbij geldt dat $g_i(-1) = 0\ \forall i$, we bepalen $g_i(R_i)$ dan doormiddel van recursie op de formules (22) en (23). Nu volgt dat E_i en V_i verkregen kunnen worden door te sommeren over alle mogelijke waarden van n_i. Er geldt dan het volgende:

$$E_i = \frac{\lambda_i}{\mu_{over}} B(R_i, \rho_i) \tag{27}$$

$$V_i = \frac{\lambda_i}{\mu_{over}} \cdot \frac{d}{dz}G_{i,n_i}(z)|_{z=1} + E_i - (E_i)^2 \tag{28}$$

Tevens geldt:

$$E_0 = \frac{\lambda_0}{\mu_0} \tag{29}$$

$$V_0 = \frac{\lambda_0}{\mu_0} \tag{30}$$

Nu kunnen we ERM toe gaan passen. Want we hebben het gemiddelde en de variantie van het complete systeem. We generen dus één server die over dezelfde eigenschappen beschikt als alle servers samen. Dit gaat precies op dezelfde manier als uitgelegd in sectie 'Equivalent Random Method'. De nieuwe equivalente server heeft dan load a en capaciteit R dusdanig dat:

$$aB(R, a) = E \tag{31}$$

$$E(1 - E + \frac{a}{R + 1 + E - a}) = V \tag{32}$$

Met: ($i = 0$, want we hebben een groep n_{00})

$$E = \sum_{i=0}^{I} E_i \tag{33}$$

$$V = \sum_{i=0}^{I} V_i \tag{34}$$

Deze formules zijn zo omdat de servers onafhankelijk van elkaar opereren. Nu kan de fractie geblokkeerde klanten van de overflow groep worden uitgerekend, dit kan met formule (10). Als we hierin onze variabele invullen krijgen we:

$$\Pi_c = \frac{EB(R + c, E)}{EB(R, E)} \tag{35}$$

Het gemiddelde aantal klanten dat geblokkeerd wordt, W, kan analoog aan (9) worden bepaald:

$$W = E \cdot B(R + c, E) \tag{36}$$

Of er kan gebruik worden gemaakt van de 'Katz approximation' om te weten te komen wat de blokkeringskans van de individuele groepen is en omdat we met deze blokkeringskans de verwachting van het aantal klanten van soort n_{0i} makkelijk kunnen berekenen. Dit zijn de klanten die de overflow zijn van groep i en behandeld worden door de overflow groep. De formule hiervoor is[10]:

$$E[n_{0i}] = \frac{\lambda_i}{\mu_{over}} \cdot B(R_i, \rho_i) \cdot (1 - k_i) \tag{37}$$

Er geldt dus ook:

$$E[n_{00}] = \frac{\lambda_0}{\mu_0} \cdot (1 - k_0) \tag{38}$$

In de meest optimale situatie worden calls uit de overflow groep weer teruggestuurd naar hun eigen groep als hier weer ruimte beschikbaar is. Dit is echter niet wenselijk, omdat een klant niet opeens overgeschakeld wil worden naar een andere agent, maar wiskundig is het wel verstandig om hier naar te kijken, want het geeft de meest optimale situatie weer. Dan kan worden gekeken hoever de oplossing van deze optimale oplossing af zit. Hiervoor moet het model aangepast worden. Er komt een soort terugkoppeling bij. Een deel van de klanten die zich in de overflow groep bevindt, wordt met een bepaalde intensiteit terug gestuurd naar zijn desbetreffende primary groep. De terugkoppeling heeft een intensiteit van γ_i, waarbij geldt: $\mu_{over} = \gamma_i + \mu_i$. Een logische restrictie die hieruit volgt is dat $\mu_{over} \geq \mu_i$. Onze evenwichtsvergelijking krijgt hierdoor ook een deel dat bij deze γ_i hoort.
Er komen de volgende factoren bij:

$$\sum_{i=1}^{I} (n_{0i} \cdot \gamma_i) \mathbb{1}_{n_i \leq c_i}$$

$$\sum_{i=1}^{I} (n_{0i} + 1) \cdot \gamma_i \cdot \Pi(\mathbf{n} + e_{0i} - e_i) \mathbb{1}_{(n_i + 1 \leq c_i)}$$

Waarbij de eerste factor voor het = teken erbij komt en de tweede factor er na het = teken bij komt.

De aankomstintensiteit bij een primary groep verandert door deze terugkoppeling ook. Laten we de nieuwe aankomstintensiteit ω_i noemen. Dan geldt hiervoor:

$$\omega_i = \lambda_i + \gamma_i \cdot E[n_{0i}] \tag{39}$$

Om het model te analyseren, kunnen we gebruik maken van deze terugkoppeling, dan moeten we wel in de vergelijkingen hierboven λ_i vervangen door ω_i. Er geldt dan voor het gemiddeld

aantal klanten in de overflow groep dat dit een sommatie is van alle individuele groepen die aanwezig zijn in de overflow groep. Noem dit $E[n_0]$. De formule die daarbij hoort is:

$$E[n_0] = \sum_{i=0}^{k} E[n_{0i}] \tag{40}$$

Daarnaast geldt dat voor het gemiddeld aantal klanten in de primary groep i geldt dat:

$$E[n_i] = \frac{\omega_i}{\mu_i}(1 - B(R_i, \frac{\omega_i}{\mu_i})) \tag{41}$$

De gemiddelde bezettingsgraad (τ_i) van een groep wordt dan gegeven door:

$$\tau_i = \frac{E[n_{0i}]}{R_i} \tag{42}$$

Om te kijken hoeveel klanten er in totaal geblokkeerd worden, kan er gekeken worden naar de formules die eerder uitgelegd zijn. De formules (40) en (42) kunnen ook gebruikt worden voor het model zonder terugkoppeling, net als formule (41) alleen moet hier dan de ω_i weer terugveranderd worden in λ_i.

6.2.3 Trunk Reservation

Het kan handig zijn om capaciteit binnen bepaalde skillgroepen te 'reserveren' voor belangrijkere call types. Dit zorgt ervoor dat klanten vaker geholpen worden door tolken die aan hun specifieke vraag voldoet. Voor het bepalen van de blokkeringskans en de bezettingsgraad van de tolken gebruiken we de globale evenwichtsvergelijking. Voor de globale evenwichtsvergelijking hebben we

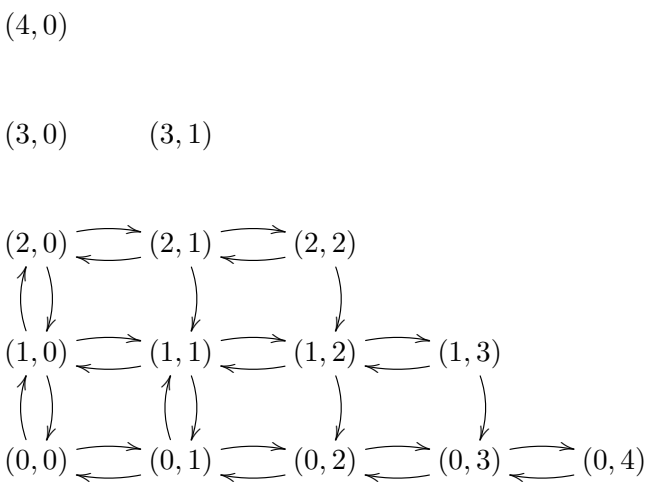
$$\sum_{i=1}^{I} \pi_f(\mathbf{n} - e_i)\lambda_i f_i(\mathbf{n} - e_i) + \sum_{i=1}^{l} \pi_f(\mathbf{n} + e_i)\mu_i(n_i + 1) = \sum_{i=1}^{l} [\lambda_i f_i(\mathbf{n}) + \mu_i n_i]\pi_f(\mathbf{n}). \tag{43}$$

Hierbij geeft I het totaal aantal call types, λ_i de intensiteit bij call type i, μ_i geeft de bedieningsintensiteit bij call type i, $\mathbf{n} = (n_1, ..., n_I)$ de toestand, n_i het aantal klanten van call type i in toestand $\mathbf{n}$, $f_i(\mathbf{n})$ geeft de toelatingsfunctie in toestand $\mathbf{n}$ voor call type i en $\pi_f(\mathbf{n})$ geeft de kans om in toestand $\mathbf{n}$ te zijn onder toelatingsfunctie f. [6] De toelatingsfunctie f_i bij trunk reservation ziet er als volgt uit:

$$f_i(\mathbf{n}) = \begin{cases} 1 & \text{Als } \mathbf{n} + e_i \leq C - t_i \\ 0 & \text{anders.} \end{cases}$$

De toelatingsfunctie f_i geeft dus 1 of 0, 1 als er nog een klant van type i geaccepteerd kan worden, 0 indien dit niet het geval is. Hierbij geeft t_i de trunk reservation aan van call type i, dus het aantal call types die gereserveerd worden. Een eenvoudig voorbeeld om te laten zien

hoe het in zijn werking gaat.



We hebben $C = 4$, $I = 2$, dus alle tolken kunnen beide call soorten beantwoorden. In het figuur hierboven is de situatie getekend waarbij $t_1 = 0$ en $t_2 = 2$, met op de x-as het aantal tolken bezig met call type 1 en op de y-as call type 2. We kunnen zien dat deze gekozen trunk reservation politiek niet convex is. Convex betekent dat als je van een toestand $\mathbf{n}$ naar toestand $\mathbf{m}$ kan gaan, je ook van toestand $\mathbf{m}$ naar toestand $\mathbf{n}$ moet kunnen gaan. We kunnen duidelijk zien in het figuur dat dit niet het geval is en dus is onze politiek niet convex. De vergelijking (43) geldt onder elke politiek, dus ook niet convexe politieken.
Voor het oplossen van (43) moeten we het stelsel vergelijkingen oplossen van de vorm $Ax = 0$. De matrix A bevat de coefficiënten van $\pi(n)$ in

$$\sum_{i=1}^{I} \pi_f(\mathbf{n} - e_i)\lambda_i f_i(\mathbf{n} - e_i) + \sum_{i=1}^{I} \pi_f(\mathbf{n} + e_i)\mu_i(n_i + 1) - \sum_{i=1}^{I} [\lambda_i f_i(\mathbf{n}) + \mu_i n_i]\pi_f(\mathbf{n}) = 0$$

en x is hierbij $\pi(\mathbf{n})$. Hieruit kunnen we $\pi(\mathbf{n})$ bepalen, oftewel de kans dat we in een toestand n zitten. De blokkeringskans B kunnen we vervolgens bepalen door

$$B = \sum_{\mathbf{n} \in S} \pi_f(\mathbf{n}) \sum_{i=1}^{I} f_i(\mathbf{n} + e_i) \frac{\lambda_i}{c}$$

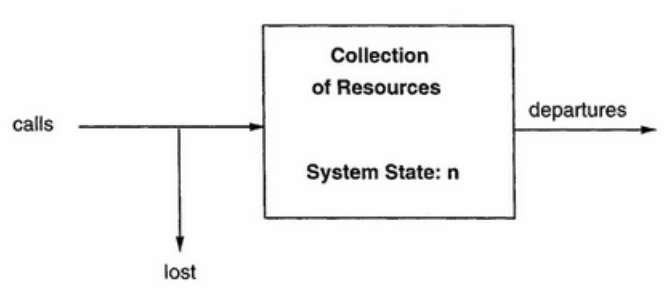
met

$$c = \sum_{i=1}^{I} [n_i \mu_i + \lambda_i f_i(\mathbf{n} + e_i)],$$

met S alle mogelijke toestanden van $\mathbf{n}$.

6.2.4 Threshold politieken

Bij de threshold politieken veranderd het model in een knapsack model met een strengere toelatingsfunctie. Het idee van het knapsackmodel staat gegeven in figuur 7, dit figuur komt uit het boek van Keith Ross [6]. We hebben een capaciteit aan servers en alle soorten klanten kunnen geholpen worden door iedere server. De toestand van het systeem is $\mathbf{n}$, waarbij



Figuur 7: Knapsackmodel

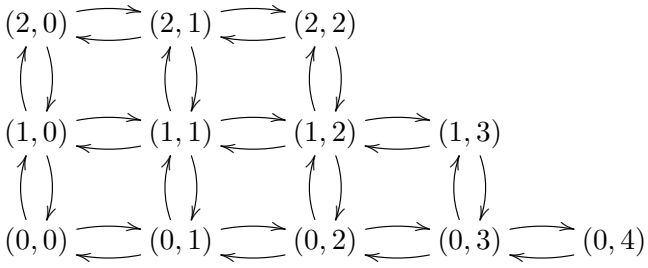
$\mathbf{n} = (n_1, ..., n_I)$. Hierbij geeft n_i aan hoeveel klanten van type i in het systeem is. De toelatingsfunctie is simpel, indien $\sum_i n_i + 1 <= C$ wordt de klant toegelaten, anders niet. C geeft hierbij de capaciteit aan. Threshold politieken lijken erg veel op trunk reservation politieken, echter zijn threshold politieken wel convex. In plaats van dat er een soort reservering zit op een bepaalde call type, zit er nu een reservering binnen een groep. Hierdoor kan het werk beter verdeeld worden tussen de verschillende skillgroepen en wederom kan er capaciteit achter gehouden worden voor belangrijkere call types. Het model blijft exact hetzelfde, zo ook de evenwichtsvergelijking 43, echter verandert de toelatingsfunctie f_i.[6] De toelatingsfunctie wordt:

$$f_i(n) = \begin{cases} 1 & \text{Als } n_i + 1 \leq C_i \text{ en } \mathbf{n} + 1 \leq C \\ 0 & \text{anders.} \end{cases}$$

Hierbij geeft C_i de capaciteit van call type i aan. De constraint $\mathbf{n} + 1 \leq C$ zorgt ervoor dat er niet meer agenten ingezet kunnen worden dan dat er zijn. Er hoeft namelijk niet te gelden dat $C_1 + C_2 + ... + C_I \leq C$, oftewel alle C_i samen kunnen meer zijn dan de totale capaciteit. Hieronder een voorbeeld die illustreert hoe threshold politieken werken.

$(4,0)$

$(3,0) \qquad (3,1)$



We kunnen zien dat dit voorbeeld wel een convexe politiek is. Sterker zelfs, alle threshold politieken zijn convex. Dit betekent dat we snellere en makkelijkere methodes kunnen toepassen. De toestanden die bij de threshold politieken hoort, zien er als volgt uit:

$$S = \{(n_1, ..., n_I) : n \leq C; n_i \leq C_i, i = 1, ..., I\}.$$

Aangezien S convex is onder de toelatingsfunctie f_i van (44), kunnen we een snellere en handigere methode hanteren dan bij trunk reservation. De kans dat je in een specifieke toestand $\mathbf{n}$ zit is makkelijker te bepalen, namelijk door

$$\pi(n) = \frac{1}{G} \prod_{k=1}^{K} \frac{\rho^{n_k}}{n_k!},$$

met

$$G = \sum_{\mathbf{n} \in S} \prod_{k=1}^{K} \frac{\rho^{n_k}}{n_k!}. \tag{44}$$

Echter is het bepalen van G vrij lastig als je naar grote systemen gaat kijken. Er zijn een aantal methodes bedacht om deze G te schatten. Wij zullen naar Monte Carlo sommatie gaan kijken om G te schatten.

6.2.5 Monte Carlo Sommatie

In het boek van Ross[6] word de Monte Carlo sommatie uitgewerkt. Het idee van Monte Carlo Sommatie is om G met de vorm (44) te bepalen. We kunnen deze vergelijking (44) omschrijven naar

$$G = \sum_{n_1=0}^{N_1} ... \sum_{n_K=0}^{N_K} g(\mathbf{n}) \mathbb{1}(\mathbf{n} \in S)$$

met

$$g(\mathbf{n}) := \prod_{k=1}^{K} \frac{\rho_k^{n_k}}{n_k!}$$

en

$$N_k := \max(n_k : (n_1, ..., n_K) \in S).$$

Hierbij geeft N_k het maximale aantal type k klanten die in het systeem kunnen zijn. Verder geeft $\mathbb{1}$ de stapfunctie weer. De toestand $(N_1, ..., N_K)$ hoeft dus niet een toestand te zijn die in S zit. Definieer

$$\tilde{S} := \{0, ..., N_1\} \times ... \times \{0, ..., N_k\}.$$

En laat $p(n)$ een discrete kansverdeling zijn voor $\tilde{S}$. We hebben nodig dat $p(n) > 0$ voor alle $\mathbf{n} \in S$. Laat $Y_r = (Y_{1r}, ..., Y_{Kr})$, $r = 1, 2, ..., R$ een reeks van onderling onafhankelijke random vectoren zijn, waarbij Y_i de discrete kansverdeling functie $p(\mathbf{n})$ heeft. Elke Y_i stelt een toestand voor die zich altijd in $\tilde{S}$ bevind. Laat

$$\Phi_r(Y_r) := \frac{g(Y_r)\mathbb{1}(Y_r \in S)}{p(Y_r)}.$$

Dan is

$$\bar{\Phi}_r := \frac{1}{R} \sum_{r=1}^{R} \Phi_r(Y_r) \tag{45}$$

een schatter voor G. De variantie $\sigma_R^2(\Phi)$ wordt geschat door:

$$\sigma_R^2(\Phi) := \frac{1}{R-1} \sum_{r=1}^{R} (\Phi_r(Y_r) - \bar{\Phi}_R)^2. \tag{46}$$

Dit is de standaard steekproefvariantie. Met behulp van het boek van Ross [6] kunnen we een betrouwbaarheidsinterval opstellen, namelijk $\bar{\Phi}_R \pm c(\alpha)\sigma_R(\Phi)/\sqrt{R}$ met $c(\alpha)$ de kritieke waarde van de standaard normaal verdeling. Voor het bepalen van Y_r hebben we echter een kansfunctie $p(n)$ nodig. In het boek van Ross [6] suggereren ze de volgende kansfunctie:

$$p_\gamma(\mathbf{n}) = \frac{1}{c}\prod_{k=1}^{K}\frac{\gamma^{n_k}}{n_k!}, \mathbf{n} \in \tilde{S}$$

met

$$c := \prod_{k=1}^{K}\sum_{l=0}^{N_k}\frac{\gamma_k^l}{l!}.$$

Hierbij zijn γ_k positieve reeële getallen. Dit is niet de beste schatter, omdat voor de beste schatter kennis nodig is van de blokkeringskans. Dit is niet handig, aangezien we juist de blokkeringskans willen uitrekenen. Voor het bepalen van γ_k gebruiken we

$$\gamma_k = [1 + .15(1 - z)]\rho_k, \qquad k = 1, ..., K$$

met

$$z := \max_{1\leq k\leq K}\frac{\sum_{k=1}^{K}\rho_k}{C_k}.$$

Met behulp van deze γ_k hebben we dus een kansfunctie $p_\gamma(n)$ waarmee we de Y_r vast hebben gesteld. Voor het bepalen van deze Y_r gebruiken we het alias algoritme, meer hierover staat in de appendix. Als we de Y_r bepaald hebben, neemt de schatter $\bar{\Phi}_R$ de volgende vorm aan:

$$\bar{\Phi}_R = \frac{1}{R}\sum_{r=1}^{R}\frac{g(Y_r)\mathbb{1}(Y_r \in S)}{p_\gamma(Y_r)}$$

Nu we $\bar{\Phi}_R$ bepaalt hebben, kunnen we een betrouwbaarheidsinterval opstellen voor de blokkerinskans voor de verschillende soorten klanten. Deze blokkerinskans B_i is

$$1 - B_i = \bar{\Gamma}_R \pm \frac{c(\alpha)s_R}{\bar{\Phi}\sqrt{R}}$$

met

$$\bar{\Gamma}_R := \frac{\sum_{r=1}^{R}\Phi_r^{(1)}}{\sum_{r=1}^{R}\Phi_r},$$

hierbij geldt dat $\Phi_r^{(1)} = \Phi_r\mathbb{1}(Y_r + e_i \in S)$, $c(\alpha)$ de kritieke waarde voor een $1 - \alpha$ betrouwbaarheidsinterval is en we definiëren:

$$s_R^2 := \sigma_R^2(\Phi^{(1)}) - 2\bar{\Gamma}_R\sigma_R^2(\Phi, \Phi^{(1)}) + \bar{\Gamma}_R\sigma_R^2(\Phi).$$

Hierbij geldt dat

$$\sigma_R^2(\Phi, \Phi^{(1)}) = \frac{1}{R-1}\sum_{r=1}^{R}(\Phi_r - \bar{\Phi}_R)(\Phi_r^{(1)} - \bar{\Phi}_R^{(1)}).$$

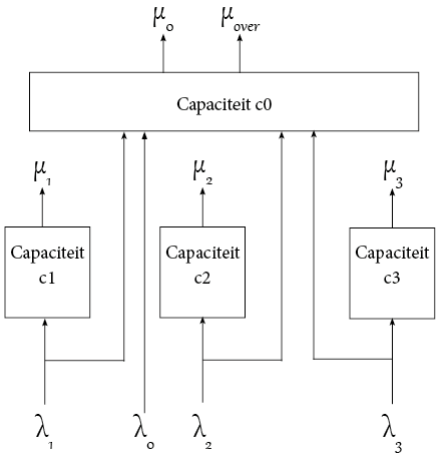
De functie $\mathbb{1}(Y_r + e_i \in S)$ geeft aan of er een klant van type i geholpen kan worden in de toestand Y_r.

7 Resultaten

In deze sectie zullen een aantal resultaten voor de verschillende modellen worden besproken. Als eerste zullen de modellen voor een eenvoudig model getest worden. Dit model is door ons zelf opgesteld en bedacht. Als tweede zal er een model dat gebaseerd is op gegevens van Capio, worden getest op alle modellen. Als laatste zal er besproken worden wat er met de blokkeringspercentages gebeurt als de aankomstintensiteiten en capaciteiten worden gevarieëerd.

7.1 Eenvoudig Model

Allereerst hebben we een eenvoudig model opgesteld, om de verschillende modellen hierop te testen. Deze situatie ziet u in figuur 8. Dit model is opgesteld voor één talencombinatie, waarin drie specialisaties zijn verwerkt. In dit model zijn nog geen gegevens van het bedrijf Capio verwerkt. We zullen hiervoor eerst het Erlang Loss model bekijken, vervolgens het overflow model en als laatste zal er ook nog worden gekeken naar het model met trunk reservation.



Figuur 8: Eenvoudig model

7.1.1 Erlang Loss model

Allereerst hebben we het model bekeken, waarbij alle klanten als hetzelfde worden gezien en alle tolken alle soorten klanten kunnen helpen. Hierbij is de capaciteit van het systeem is gelijk aan het totaal aantal tolken in alle groepen. De λ zijn het gemiddelde aantal klanten dat aankomen in een uur. De λ wordt verkregen door alle individuele $\lambda's$ bij elkaar op te tellen. Om de bedieningsduur van dit systeem te weten, wordt het gemiddelde van de bedieningsduren van de verschillende groepen genomen. De input voor het Erlang Loss model is te vinden in tabel 1. De bijbehorende output is te vinden in tabel 2.

Parameter	Waarde
C	15
λ	27
μ	5.5

Tabel 1: Input waarbij elke tolk iedere klant kan helpen.

Gemiddeld aantal klanten in systeem	4.91
Procent klanten dat wordt geblokkeerd in systeem	0.01

Tabel 2: Output waarbij elke tolk iedere klant kan helpen.

Vervolgens is het model bekeken door het Erlang loss model te gebruiken op alle verschillende groepen, waarbij wordt aangenomen dat alle groepen onafhankelijk zijn van elkaar. De input is te zien in tabel 3 . De output die hierbij hoort is te vinden in tabel 4. Het totale procent klanten dat wordt geblokkeerd in het hele systeem wordt dan 45.81%.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
C_0	6	λ_0	10	μ_0	10
C_1	2	λ_1	5	μ_1	4
C_2	3	λ_2	5	μ_2	3
C_3	4	λ_3	7	μ_3	5

Tabel 3: Input met verschillende groepen

Groepen	Gemiddeld aantal klanten	Blokkeringspercentage in procenten
0	1	0.05
1	0.93	28.5
2	1.4	15.98
3	1.43	4

Tabel 4: Output met verschillende groepen

In de groepen 1 en 2 is het procent van de klanten dat wordt geblokkeerd een stuk hoger dan in de groepen 0 en 3, terwijl er welk in elke groep gemiddeld rond de één klant in het systeem zit.

7.1.2 Overflow model

Voor dit eenvoudige model zijn alle rekenstappen uitgelegd in de Appendix. Hier zullen we kort laten zien wat de resultaten zijn. Voor het model geldt dat de aankomst bij de 'eerste primary groep' 5 klanten per uur is, dit betekent dat $\lambda_1 = 5$. De bedieningsduur is 4 klanten per uur, dit betekent $\mu_1 = 4$. De maximale capaciteit van deze groep wordt gegeven door $R_1 = 2$. De parameters van de andere goepen staan in tabel 5. In de tabel 6 staan de eerste resultaten, die behoren bij de input. Deze resultaten hebben we nodig omdat we hiermee uiteindelijk het aantal geblokkeerde klanten kunnen uitrekenen. Hiervoor maken we gebruik van formule (36), daarnaast passen we ook de 'Katz approximation' toe. De resultaten die

we dan krijgen staan in tabel 7.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
R_0	6	λ_0	10	μ_0	10
R_1	2	λ_1	5	μ_1	4
R_2	3	λ_2	5	μ_2	3
R_3	4	λ_3	7	μ_3	5
				μ_{over}	5

Tabel 5: Input overflow model

Parameter	Waarde
E	1.4716
V	1.5779
Equivalente capaciteit	4
Equivalente 'load' (in Erl)	3.93

Tabel 6: Eerste output overflow model

Parameter	Waarde
W	0.0187
k_0	0.0047
k_1	0.0450
k_2	0.0238
k_3	0.0542
$E(n_{00})$	0.9953
$E(n_{01})$	0.2461
$E(n_{02})$	0.1560
$E(n_{03})$	0.0530

Tabel 7: Resultaten overflow model

Het blijkt dus dat er gemiddeld 0.0187 klanten per uur geblokkeerd worden, dit is gelijk aan 0.069 procent. De grootste blokkeringskans hebben de klanten van groep 3. Daarnaast is te zien dat gemiddeld de meeste klanten van groep 1 zich in de overflow groep bevinden, naast de klanten van de overflow groep zelf.

7.1.3 Trunk reservation

We gaan kijken naar een aantal verschillende trunk reservations, aangegeven met t voor het eenvoudige model, gegeven in tabel 9. De input gegevens staan in tabel 8. De resultaten voor verschillende trunk reservations staan gegeven in tabel 9. We zien dat de blokkeringskansen erg klein zijn, wat ook te verwachten was met de gekozen waarden. Voor het uitrekenen van deze waarden hebben we een matlab script geschreven die $\pi(\mathbf{n})$ bepaalt voor alle mogelijke

n. Als het systeem veel groter zou worden, kunnen we alles niet meer uitrekenen met behulp van deze methode.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
C_1	2	λ_1	5	μ_1	4
C_2	3	λ_2	5	μ_2	3
C_3	4	λ_3	7	μ_3	5
C_0	6	λ_0	10	μ_0	10

Tabel 8: Input voor eenvoudig model

t_i	B in %	t_i	B in %
[0,0,0,0]	0.1948	[2,0,0,0]	0.2001
[1,0,0,0]	0.1968	[0,2,0,0]	0.1995
[0,1,0,0]	0.1966	[0,0,2,0]	0.2084
[0,0,1,0]	0.1980	[0,0,0,2]	0.2030
[0,0,0,1]	0.1983	[2,1,0,0]	0.2017
[1,1,0,0]	0.1986	[2,0,1,0]	0.2025
[1,0,1,0]	0.1995	[2,0,0,1]	0.2034
[1,0,0,1]	0.2008	[1,2,0,0]	0.2011
[0,1,1,0]	0.1993	[0,2,1,0]	0.2018
[0,1,0,1]	0.2006	[0,2,0,1]	0.2027
[0,0,1,1]	0.2012	[1,0,2,0]	0.2094
[1,1,1,0]	0.1994	[0,1,2,0]	0.2093
[1,1,0,1]	0.2038	[0,0,2,1]	0.2104
[1,0,1,1]	0.2030	[1,0,0,2]	0.2051
[0,1,1,1]	0.2027	[0,1,0,2]	0.2049
		[0,0,1,2]	0.2056

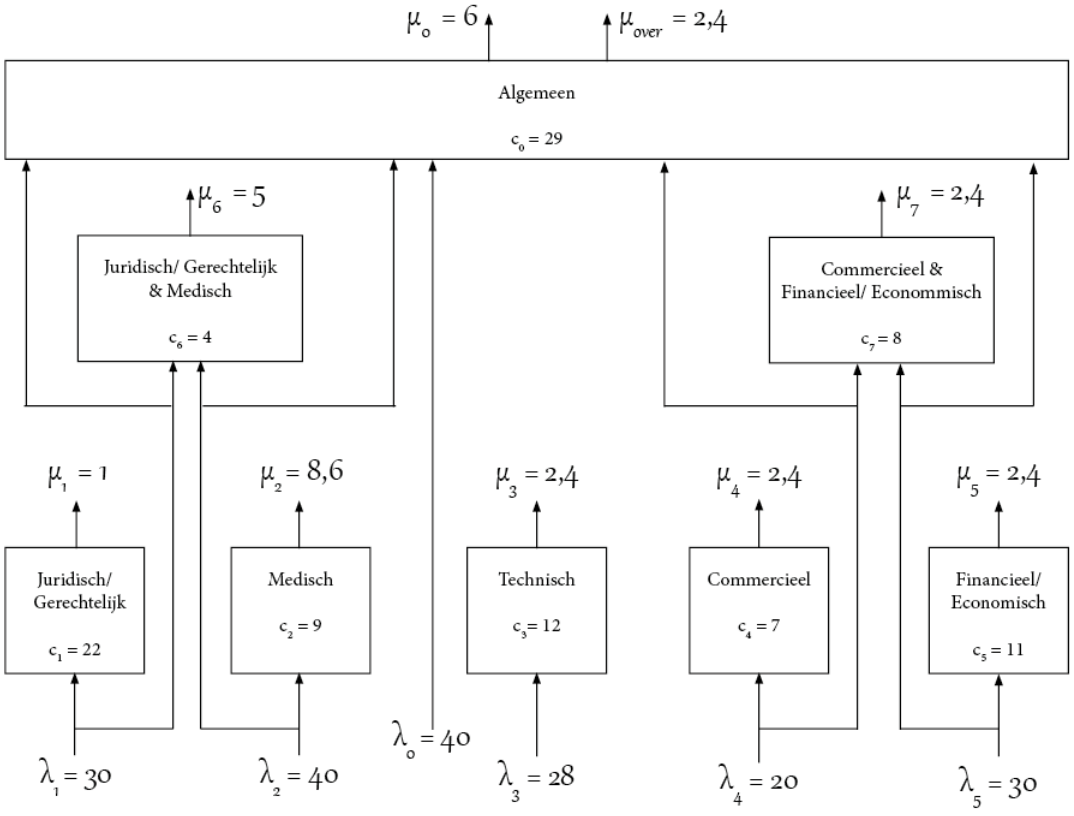
Tabel 9: Resultaten voor trunk reservation

De laagste blokkeringskans is wanneer $t_i = [0, 0, 0, 0]$, hierbij mogen alle tolken alle type klanten gebruiken. De hoogste blokkeringskansen zijn te zien bij $t_i = [0, 0, 2, 1]$ en $t_i = [0, 0, 2, 0]$. We kunnen hieruit zien dat de capaciteit van groep 3 achterhouden om bij andere call types in te zetten een erg slecht idee is.

7.2 Model Capio

Nu gaan we kijken naar de situatie waar Capio naar verwachting mee begint voor de talen combinatie Nederlands-Frans. Deze situatie staat in figuur 9. Er zijn 5 specialisaties aanwezig en maar 2 skill groepen die 2 specialisaties kunnen. Wederom is er een algemene overflow groep waar alles naar toe gaat als het niet geholpen kan worden door een tolk met de juiste skill. Zoals gewoonlijk geeft λ_i de aankomst intensiteit voor call type i, μ_j de bedieningsduur van een skill groep en c_j de capaciteit van een bepaalde skill groep. Als eerste bekijken we het Erlang Loss model, vervolgens het overflow model en daarna threshold. We zullen deze keer niet kijken naar trunk reservation, aangezien deze situatie te groot is om goed met trunk

reservation te berekenen. Threshold is een soort van benadering voor trunk reservation en zullen we gebruiken om dit model te benaderen.
In deze sectie wordt ingegaan op het model van Capio, we hebben in overleg met Capio een model opgesteld waarvan zij verwachten dat dit in het begin gebruikt kan worden. Daarnaast is de data die ons aangeleverd zijn verwerkt in het model. De data die we hebben ontvangen, is een verdeling van de capaciteit van elke groep en de bijbehorende bedieningsduur. De capaciteit is gebasseerd op het totaal aantal tolken en het gemiddeld aantal uren dat een tolk beschikbaar is. We hebben ook een tabel waarin het aantal verwachte gebruikers staat, op deze tabel is onze aankomstintensiteit gebaseerd. Het model bevat dus realistische data, maar sommige gegevens zijn bedacht. Dit kan omdat de data niet beschikbaar is of omdat deze data nog niet bestaat. Dit bedenken is wel dusdanig gedaan dat er waarden uit zijn gekomen die in de realiteit kunnen voorkomen. De aankomstintensiteit is Poisson verdeeld en geeft het aantal klanten weer dat per uur aankomt. De bedieningsduur is exponentieel en geeft het aantal klanten dat per uur door een tolk geholpen kan worden.



Figuur 9: Start model van Capio voor Nederlands-Frans

7.2.1 Erlang Loss

Net zoals bij het eenvoudige model dat besproken is, hebben we dit model bekeken voor het geval dat alle tolken alle typen klanten kunnen helpen en het geval dat er wel verschillende soorten tolken zijn per aankomstgroep. In tabel 10 is de input gegeven voor het Erlang loss

model, waarbij alle tolken alle soorten klanten kunnen helpen. In tabel 11 staan hiervan de resultaten.

Parameter	Waarde
C	102
λ	188
μ	3.8

Tabel 10: Input waarbij elke tolk iedere klant kan helpen.

Gemiddeld aantal klanten in systeem	49.47
Procent klanten dat wordt geblokkeerd in systeem	0

Tabel 11: Output waarbij elke tolk iedere klant kan helpen.

Vervolgens is het Erlang loss model gebruikt voor de verschillende groepen met als aanname dat de groepen onafhankelijk van elkaar zijn. Er is echter een probleem met de Juridisch/-Gerechtelijk & Medische groep, omdat dit een aparte groep is waar klanten van de groepen Juridisch/Gerechtelijk en Medisch apart aankomen en geen eigen aankomstintensiteit heeft. Dit is opgelost door die groep te splitsen en toe te voegen aan de kleinere groepen. Dat betekent dat die groepen er dus twee tolken bij krijgen. Ditzelfde is gedaan voor de groep Commercieel & Financieel/Economisch. De input voor het Erlang Loss model staat in tabel 12. De resultaten die hierbij horen zijn te vinden in tabel 13.

C_0	29	λ_0	40	μ_0	6
C_1	24	λ_1	30	μ_1	1
C_2	11	λ_2	40	μ_2	8.6
C_3	12	λ_3	28	μ_3	2.4
C_4	11	λ_4	20	μ_4	2.4
C_5	15	λ_5	30	μ_5	2.4

Tabel 12: Input met verschillende groepen

Groepen	Gemiddeld aantal klanten	Blokkeringspercentage in procenten
0	6.67	0
1	21.87	27.09
2	4.63	0.53
3	9.5	18.54
4	7.55	9.4
5	11.24	10.05

Tabel 13: Output met verschillende groepen

Het totaal procent geblokkeerde klanten van het gehele systeem komt dan uit op 65.60%. Het aantal klanten in de verschillende groepen variëren nogal. Ook het procent aantal klanten dat wordt geblokkeerd per groep verschilt erg. Zo is het procent klanten dat in groep 1 wordt geblokkeerd erg hoog, maar in groep 0 worden er geen klanten geblokkeerd.

7.2.2 Overflow

Het model waarmee we nu gaan werken is wat complexer dan wat uitgelegd is in de sectie model. Het verschil zit hem in het vaker uitvoeren van ERM en dat deze aparte systemen dan moeten worden samenvoegd. Dit wordt gedaan zodat er uiteindelijk één ERM-model overblijft waaruit dan de conclusie gehaald kan worden. We zoeken dus een equivalent systeem, van één server, voor de gedeelten:

- Medisch, Juridisch, Medisch-Juridisch
- Commercieel, Financieel, Commercieel-Financieel

Allereerst wordt er gekeken naar de eerste combinatie, dus die van (Jur,Med,Jur-Med). In tabel 14 staan de input waarden voor dit systeem.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
R_1	22	λ_1	30	μ_1	1
R_2	9	λ_2	40	μ_2	8.6
R_6	4			μ_6	5

Tabel 14: Input overflow model medisch en juridisch

In dit geval is μ_6 gelijk aan μ_{over}.

Met deze data wordt de output geregenereerd die van belang is voor het equivalente systeem. Allereerst wordt het gemiddelde en de variantie van het nieuwe systeem bepaald, deze worden respectievelijk gegeven door $E = 2.165$ en $V = 4.081$. Door middel van het gemiddelde en de variantie worden de gegevens voor het equivalente systeem bepaald, die te vinden zijn in tabel 15.

Equivalente capaciteit	17
Equivalente 'load' (in Erl)	15.186

Tabel 15: Gegevens equivalent systeem medisch en juridisch

Deze data wordt gebruikt om de uiteindelijke uitkomst voor onze vraag te bepalen, namelijk hoeveel er mensen worden geblokkeerd door het systeem. In tabel 16 worden hiervan de resultaten gegeven. Het totaal aantal klanten dat geblokkeerd wordt W wordt gegeven, net als de individuele blokkeringskans van een groep k_i. Daarnaast wordt de verwachting van het aantal klanten van groep i in de overflow groep gegeven ($E(n_{0i})$).

Het blijkt dus dat er per uur gemiddeld 0.518 klanten geweigerd worden door dit systeem. Dat is gelijk aan 0.74 procent. Vooral klanten van groep 1 bevinden zich in de overflow groep en deze groep heeft ook de hoogste blokkeringskans.

Parameter	Waarde
W	0.518
k_1	0.291
k_2	0.266
$E(n_{01})$	1.379
$E(n_{02})$	0.161

Tabel 16: Resultaten overflow model medisch en juridisch

We doen nu hetzelfde voor de tweede combinatie, die van (Comm,Fina,Comm-Fina). Omdat dit op precies dezelfde manier werkt als hierboven, geven we alleen een opsomming van de belangrijke waarden. In tabel 17 staat de input van het systeem en in tabel 18 staat de output van het systeem.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
R_4	7	λ_4	20	μ_4	2.4
R_5	11	λ_5	30	μ_5	2.4
R_7	8			μ_7	2.4

Tabel 17: Input overflow model commercieel en financieel

Waarbij $\mu_7 = \mu_{over}$.

Parameter	Waarde
E	6.070
V	11.641
Equivalente capaciteit	17
Equivalente 'load' (in Erl)	21.057
W	1.417
k_4	0.208
k_5	0.261
$E(n_{04})$	2.156
$E(n_{05})$	2.474

Tabel 18: Output overflow model commencieel en financieel

Voor dit systeem geldt dus dat er gemiddeld 1.417 klanten per uur geweigerd worden, dat is gelijk aan 2.834 procent. Daarnaast zijn er net wat meer klanten van groep 5 aanwezig in de overflow groep en is hiervan de blokkeringskans ook hoger.

Omdat we nu twee equivalente systemen hebben gemaakt kunnen we nu ERM toepassen op het hele systeem, hiervoor moeten we wel van de equivalente systemen één server maken. Dit kan door de capaciteit van de equivalente server en die van de overflow groep bij elkaar op te tellen. Dit is mogelijk doordat we alleen geïnteresseerd zijn in de stroom die de overflow groep overflowt. De formule hiervoor (36) maakt ook gebruik van de optelling van beide capaciteiten.

Van deze server weten we de load, de μ (dit is gelijk aan μ_{over}), de capaciteit en het gemiddelde aantal klanten dat het systeem verlaat. Doordat we deze waarden allemaal weten kunnen we de aankomstintensiteit (λ), van deze nieuwe server, bepalen door één van de volgende twee formules te gebruiken:

$$load = \rho = \frac{\lambda}{\mu} \tag{47}$$

$$W = \frac{\lambda}{\mu_{over}} B(R, \rho) \tag{48}$$

We kiezen ervoor om (48) te gebruiken. De reden hiervoor is puur praktisch. Het systeem komt er vervolgens als figuur 8 uit te zien. Merk op dat hierbij de groepsnamen niet zijn ingevuld. De inputgegevens staan in tabel 19. De resultaten die hierbij horen, zijn te vinden in tabel 20.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
R_0	29	λ_0	40	μ_0	6
R_1	21	λ_1	75.889	μ_1	1
R_2	12	λ_2	20	μ_2	2.4
R_3	25	λ_3	50.546	μ_{over}	2.4

Tabel 19: Input

Parameter	Waarde
E	9.670
V	14.839
Equivalente capaciteit	11
Equivalente 'load' (in Erl)	19.736
W	$4.195 \cdot 10^{-4}$
k_0	$2.817 \cdot 10^{-4}$
k_1	$8.048 \cdot 10^{-4}$
k_2	$3.887 \cdot 10^{-4}$
k_3	$8.711 \cdot 10^{-4}$
$E(n_0)$	6.665
$E(n_{01})$	1.075
$E(n_{02})$	0.510
$E(n_{03})$	1.417

Tabel 20: Resultaten

Merk op: $E(n_{02}) \leq W(systeem1)$ en $E(n_{03}) \leq W(systeem2)$. Dit hoort ook zo te zijn.

Je kunt zien dat het aantal klanten dat gemiddeld per uur geblokkeerd wordt bijna 0 is, het lijkt er dus op dat er genoeg tolken in de overflow groep zijn. Wat ook opvalt is dat er gemiddeld ongeveer 10 klanten in de overflow groep zijn, terwijl er 29 tolken zijn om deze klanten te helpen. Hierdoor zal het aantal geblokkeerde klanten redelijk laag liggen.

Een ander resultaat waar wel iets aan gedaan kan worden is dat er 'veel' klanten van groep 3 in de overflow groep aanwezig zijn. Dit betekent dat er door groep 3 veel klanten geweigerd wordt. Deze groep was ons oudere systeem 2. Hierin werden er ook al meer klanten geblokkeerd dan in systeem 1. Dus mocht er besloten worden om meer tolken in te zetten, dan kan dit het beste worden gedaan in systeem 2.

7.2.3 Threshold

Voor het grote model van Capio, afgebeeld in figuur 9 gebruiken we wederom het knapsack model, maar deze keer met threshold politieken. Dit doen we in plaats van trunk reservation, omdat de brute-force methode die we gebruikt hebben bij trunk reservation niet meer werkt voor een systeem van dit formaat. We gaan als eerste kijken naar het gewone knapsack systeem, dus zonder C_i. In tabel 21 zijn de verschillende λ en μ te zien. We nemen $R = 100000$ en $C = 102$. C_i bestaan niet voor $i = 1...I$. Verder nemen we $\alpha = 0.05$, oftewel we maken een 95% betrouwbaarheidsinterval voor de blokkeringskans. De resultaten voor de eerste en tweede situatie staan in tabel 22.

λ		μ	
λ_1	30	μ_1	1
λ_2	40	μ_2	8.6
λ_3	28	μ_3	2.4
λ_4	20	μ_4	2.4
λ_5	30	μ_5	2.4
λ_0	40	μ_0	6

Tabel 21: Input voor simpele knapsack model.

Call type	95% betrouwbaarheidsinterval voor Blokkeringskans
1	0.032 ± 0.009
2	0.032 ± 0.009
3	0.032 ± 0.009
4	0.032 ± 0.009
5	0.032 ± 0.009
6	0.032 ± 0.0009

Tabel 22: Blokkeringskansen voor situaties 1 en 2 in %

Het is duidelijk te zien dat de blokkeringskans gelijk is voor elke type klant. Dit is logisch aangezien alle tolken elke type klant helpt, dus als het systeem vol is, wordt elke type klant geblokkeerd. We gaan hier verder niet variëren met C_i aangezien de blokkeringskans heel erg klein is. We zullen in het hoofdstuk over extreme gevallen wel kijken naar verschillende C_i.

7.3 Verschillende combinaties van λ en de capaciteit

In deze sectie gaan we het model van Capio bekijken. Hierbij zorgen we ervoor dat de λ hoger ligt, dit betekent dus dat we een model bekijken waarbij het klanten register is gegroeid. Er komen dus gemiddeld meer klanten aan dan in het begin. We kijken naar dit model omdat

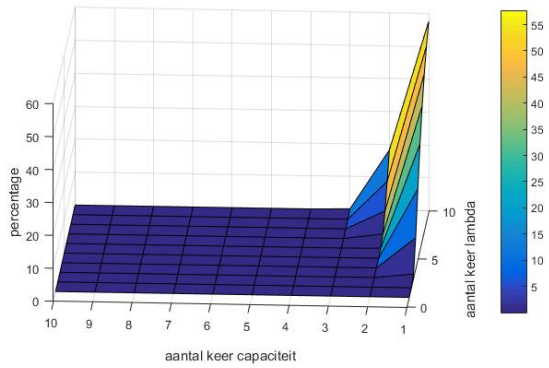
we verwachten dat het klanten register gaat groeien, dat Capio bekender wordt en dus meer aanvragen verwerft. Ook zijn er in deze sectie verschillende 3D grafieken toegevoegd, deze 3D grafieken bevatten het blokkeringspercentage bij verschillende combinaties van $\lambda^{'}s$ en capaciteiten. Uit de grafieken kan gehaald worden hoeveel capaciteit er nodig is bij een willekeurige λ om ervoor te zorgen dat je blokkeringspercentage onder je zelfgekozen grens ligt. Voor het Erlang model is er gekeken naar het model van Capio en zijn hierbij de $\lambda's$ en capaciteiten gevarieërd. Bij de sectie 'Overflow model' zijn deze grafieken gemaakt voor het model van Capio en het simpele model. Daarnaast is er voor het simpele model nog een extreme situatie bekeken waarin de λ zeer hoog ligt. Voor deze situatie zijn verschillende mogelijkheden bekeken hoe en waar extra capaciteit in te zetten en wat dit voor invloed heeft. Voor Treshold is er ook weer gekeken naar het model van Capio.

7.3.1 Erlang Loss model

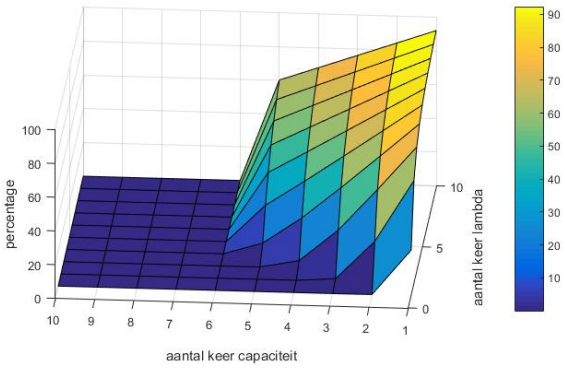
Hier zullen we kijken wat er met het percentage geblokkeerde klanten gebeurt als we de capaciteit en aankomstintensiteiten verlagen en verhogen. Dit hebben we alleen gedaan voor het geval dat de verschillende groepen worden bekeken door verschilllende Erlang Loss modellen. Hierbij nemen we aan dat de bedieningsduur hetzelfde blijft, zoals deze ook is besproken bij het model hierboven van Capio. We hebben steeds c en λ verhoogt met een bepaalde factor. Deze factoren zijn respectievelijk a en b en hebben we beiden laten lopen van 1 tot en met 10. In tabel 23 zijn de input waarden van het model nog eens weergegeven.

C_0	29a	λ_0	40b	μ_0	6
C_1	24a	λ_1	30b	μ_1	1
C_2	11a	λ_2	40b	μ_2	8.6
C_3	12a	λ_3	28b	μ_3	2.4
C_4	11a	λ_4	20b	μ_4	2.4
C_5	15a	λ_5	30b	μ_5	2.4

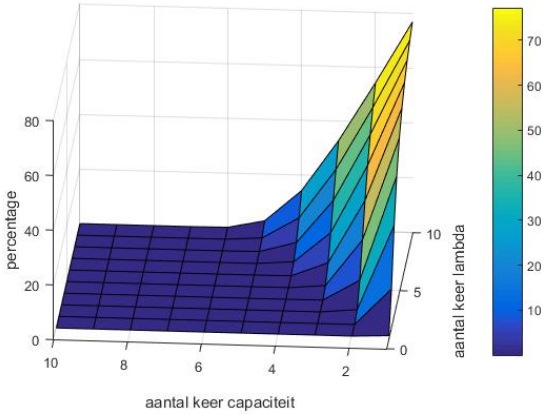
Tabel 23: Input verhogen van λ en capaciteiten



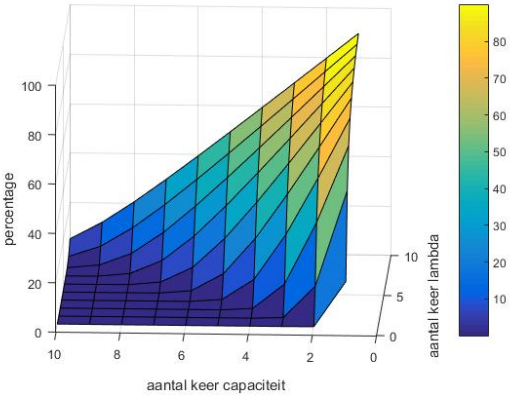
(a) Groep 0



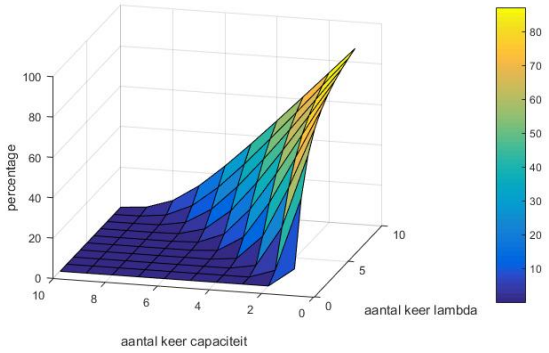
(b) Groep 1



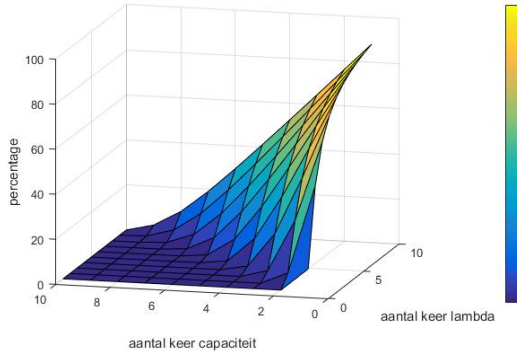
(c) Groep 2



(d) Groep 3



(e) Groep 4



(f) Groep 5

Figuur 10: Grafieken voor het verhogen van λ en capaciteiten

In figuur 10 zijn de verschillende grafieken weergegeven voor het verhogen van de capaciteit en $\lambda's$ voor de verschillende groepen. Je ziet dat bij groep 0 bij de huidige capaciteit niet veel meer aankomsten wenselijk zijn, omdat het percentage geblokkeerde klanten al snel toeneemt. Als de capaciteit echter drie keer zoveel wordt, kan het alle aankomsten die tien keer zoveel zijn als de huidige aankomsten al aan. Als je naar groep 1 kijkt, is in de huidige situatie de

blokkeringskans al 20 tot 30 procent. Als op dat moment de aankomsten toenemen, ontstaan er dus hele hoge blokkeringspercentages. Dit kan voorkomen worden, door de capaciteit met zes keer toe te laten nemen. Echter is wel de vraag of dit mogelijk is. In groep 2 is eigenlijk hetzelfde te zien als bij groep 1, behalve dat het blokkeringspercentage sneller afneemt als de capaciteit toeneemt. In groep 3 is het huidige blokkeringspercentage al aan de hoge kant en neemt uiteraard alleen maar toe als alleen de aankomsten worden verhoogd. De capaciteit moet daarvoor ongeveer evenveel verhoogt worden om voor deze groep het blokkeringspercentage laag te houden. De grafieken van groep 4 en 5 lijken erg veel op elkaar. Op het moment dat de aankomsten erg worden verhoogd, zal ook de capaciteit mee moeten groeien. Als de aankomsten met een bepaalde waarden worden verhoogd, moet de capaciteit met diezelfde waarde min 1 worden verhoogd.

7.3.2 Overflow model

Hieronder staat een situatie geschetst die redelijk extreem is, er is gekozen voor deze situatie om te laten zien hoe het extra inzetten van capaciteit ervoor kan zorgen dat er minder klanten geweigerd wordt. Er is gekozen om het systeem te bekijken die hetzelfde is opgebouwd als figuur 5. Dit is gedaan omdat de berekeningen dan sneller en makkelijker zijn, maar waardoor we wel kunnen zien wat het inzetten van extra capaciteit voor effect heeft. De inputgegevens voor dit systeem staan in tabel 24.

Capaciteit	Waarde	Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
R_0	20	λ_0	80	μ_0	2
R_1	15	λ_1	100	μ_1	2
R_2	25	λ_2	150	μ_2	2.5
R_3	20	λ_3	120	μ_{over}	2

Tabel 24: Input overflow model

Waarbij λ (μ) het aantal klanten is dat per uur aankomt (geholpen wordt). Met deze input genereren we de output voor het gemiddelde en de variantie. Dit staat in tabel 25.

Parameter	Waarde
E_0	40
E_1	35.398
E_2	44.568
E_3	31.278
V_0	40
V_1	48.530
V_2	71.707
V_3	54.000

Tabel 25: Gemiddelden en varianties

Er is gekozen om voor elke groep zijn gemiddelde en variantie te geven, dit is gedaan omdat we dan kunnen kijken bij welke groep er veel sprake is van geblokkeerde klanten. Het blijkt dat het gemiddelde en de variantie van groep 2 de grootste zijn, dit betekent dat er veel

klanten van groep 2 geblokkeerd worden, dit is ook redelijk logisch want: $\rho = \frac{\lambda}{\mu} = \frac{150}{2.5} = 75$ dit is veel hoger dan de capaciteit in de groep. Er is dus sprake van veel overflow. Daarnaast is het gemiddelde van groep 0, de overflow groep, hoog.
Merk op; de ρ van iedere groep is hoger dan zijn capaciteit, dit om ervoor te zorgen dat er overflow is.
Het gemiddeld aantal klanten dat geblokkeerd wordt door dit systeem is gegeven door $W = 131.457$ per uur. Om het aantal klanten dat geblokkeerd wordt te verminderen kan ervoor gekozen worden om meer capaciteit te generenen, we kiezen ervoor om het verschil tussen een lage extra capaciteit en een hoge extra capaciteit te onderzoeken. Allereerst wordt de capaciteit met 30 verhoogd en later met 75. We bekijken dan verschillende mogelijkheden om de capaciteit te verhogen, welke groepen zijn handig en waarom.

Allereerst gaan we kijken naar het verhogen van de capaciteit met 30.
Situatie 1:
We verhogen de capaciteit van groep 0 met 30, de nieuwe capaciteit wordt 50. We kiezen voor deze groep omdat dit de overflow groep is en omdat er veel klanten geblokkeerd werden in deze groep (een hoge E). Het aantal klanten dat in deze situatie geblokkeerd wordt is dan $W = 101.925$ per uur. Dit is een verandering van $\frac{101.925-131.457}{131.457} = -0.225$ dit betekent een verlaging van 22.5 procent ten opzichte van de oorspronkelijke situatie.
Situatie 2:
We verhogen bij de groepen 1, 2 en 3 de capaciteit met 10. Dit betekent dat groep 1 een nieuwe capaciteit van 25 heeft, groep 2 van 35 en groep 3 van 30. Hiermee willen we ervoor zorgen dat er al minder klanten geblokkeerd worden bij de 'belangrijke' groepen. Het aantal klanten dat nu geblokkeerd wordt is $W = 96.935$ per uur. Dit betekent dat deze situatie een verlaging van 26.3 procent ten opzichte van de oorspronkelijke situatie.
Situatie 3:
We verhogen de capaciteit van groep 2 met 30, dus de nieuwe capaciteit wordt 55. Dit wordt gedaan omdat het grootste aantal klanten dat van te voren geblokkeerd wordt, geblokkeerd wordt bij groep 2. We hopen doordat we groep 2 dusdanig verhogen er hier veel minder klanten geblokkeerd worden en er dus uiteindelijk minder klanten in de overflow groep komen. Voor deze nieuwe situatie geldt dat het aantal geblokkeerde klanten per uur gelijk is aan $W = 98.738$. Dit betekent een verlaging van 24.9 procent ten opzichte van de oorspronkelijke situatie.

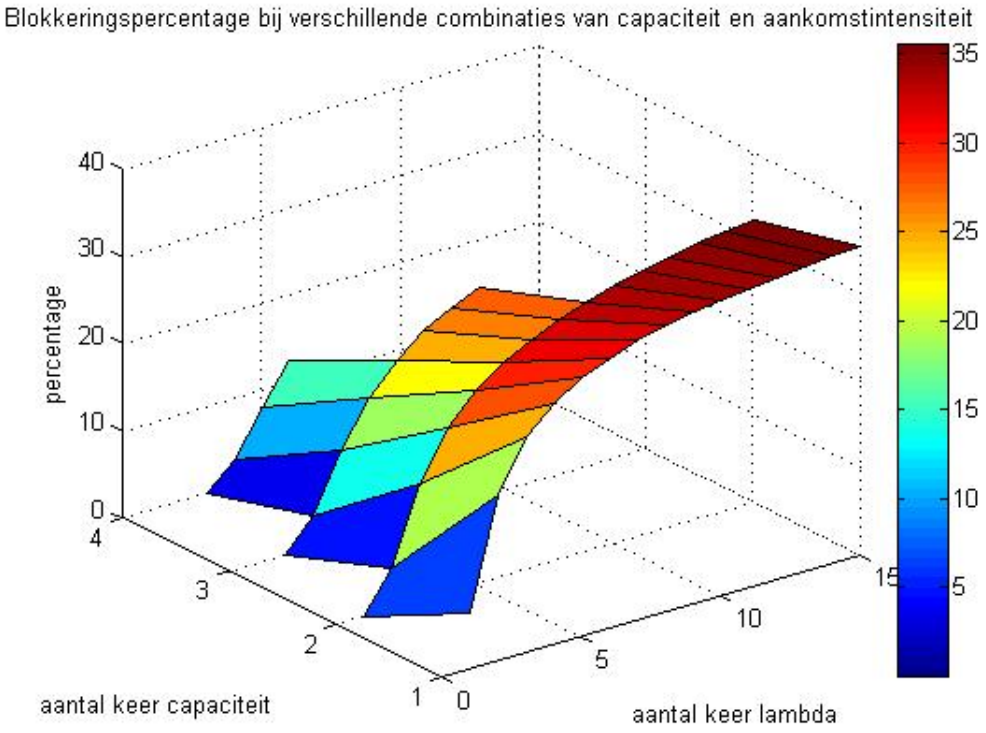
We gaan nu de capaciteit met een totaal van 75 verhogen.
Situatie 1:
We verhogen in deze situatie groep 0 met 30, de nieuwe capaciteit wordt dus 50. Daarnaast verhogen we de capaciteit van de groepen 1, 2 en 3 met 15. De nieuwe capaciteit wordt dus respectievelijk 30, 40 en 35. We kiezen voor deze opzet omdat dan de overflow groep extra klanten aan kan en er minder klanten naar deze groep worden toegestuurd, omdat de individuele groepen ook een hogere capaciteit hebben. Het aantal klanten dat per uur geblokkeerd wordt is dan $W = 45.245$. Dit is een verlaging van 65.6 procent ten opzichte van de oorspronkelijke situatie.
Situatie 2:
In deze situatie verdelen we de extra capaciteit over de 3 individuele groepen, we willen kijken of hierdoor er zo weinig klanten worden doorgestuurd dat de overflow groep geen extra capaciteit nodig heeft. De nieuwe capaciteit wordt dus respectievelijk 40, 50 en 45. Hierdoor

wordt het aantal klanten dat per uur geblokkeerd wordt $W = 52.659$, dit is een verlaging van 59.9 procent ten opzichte van de oorspronkelijke situatie.
Situatie 3:
We verhogen de capaciteit van de overflow groep met 75 naar 95. We hopen hierdoor dat de overflow groep alle klanten aankan. Dan wordt het aantal geblokkeerde klanten per uur $W = 55.981$, dit is een verlaging van 57.4 procent ten opzichte van het oorspronkelijke.

Als men de capaciteit met 30 verhoogd kan dit het beste gedaan kan worden doormiddel van situatie 2, dit verlaagt het aantal geblokkeerde klanten namelijk met 26.3 procent. Dit betekent dat de capaciteit het beste over de individuele groepen verdeeld kan worden. Als men de capaciteit met 75 verhoogd kan dit het beste worden gedaan doormiddel van situatie 1, dit is namelijk een verlaging van 65.6 procent. Je zet dan de extra capaciteit voor een groot deel in in de overflow groep, maar je zet ook een deel van de capaciteit in, verdeeld over de individuele groepen.
Het verschil in 30 of 75 extra capaciteit generenen zit hem in het volgende, je zet een factor $\frac{75}{30} = 2.5$ extra capaciteit in en dit levert een $\frac{65.6}{26.3} = 2.49$ extra verlaging van het percentage geblokkeerde klanten op. Het lijkt er dus op dat er een lineair verband bestaat. Op het moment dat het aantal extra capaciteit inzetten minder 'kost' dan dat de opbrengsten zijn van de extra geholpen klanten dan is het verstandig om 75 extra capaciteit in te zetten. Er kan nog onderzocht worden welk aantal extra capaciteit het beste rendement oplevert, dit kan gedaan worden door op allerlei nieuwe situaties te testen.

Hierboven staat voor een heel systeem uitgelegd hoe het werkt als men extra capaciteit gaat inzetten bij een redelijk grote λ. Maar we kunnen natuurlijk ook gaan kijken naar verschillende combinaties van λ en de capaciteit. Deze hebben we weergegeven in een 3D figuur. Uit de figuren kan men halen wat er gedaan moet worden met de capaciteit wanneer de λ groeit, dusdanig dat het blokkeringspercentage toch binnen de grenzen blijft.
Op de x-as is het aantal keer λ weergegeven, op de y-as het aantal keer capaciteit en op de z-as staat het blokkeringspercentage. Daarnaast geven de vlakken ook weer wat het blokkeringspercentage in dat vlak is.
Er is eerst gekeken naar het model van Capio, het model ziet er dan uit als figuur 11. Alleen de grootte van de aankomstintensiteit, bedieningsduur en capaciteit zijn aangepast. Dit is gedaan omdat er dan meer berekeningen gedaan kunnen worden, omdat de 'grootte' waarden van Capio veel rekentijd en rekenkracht kosten.

Hieronder staan de figuren weergegeven:



Figuur 11

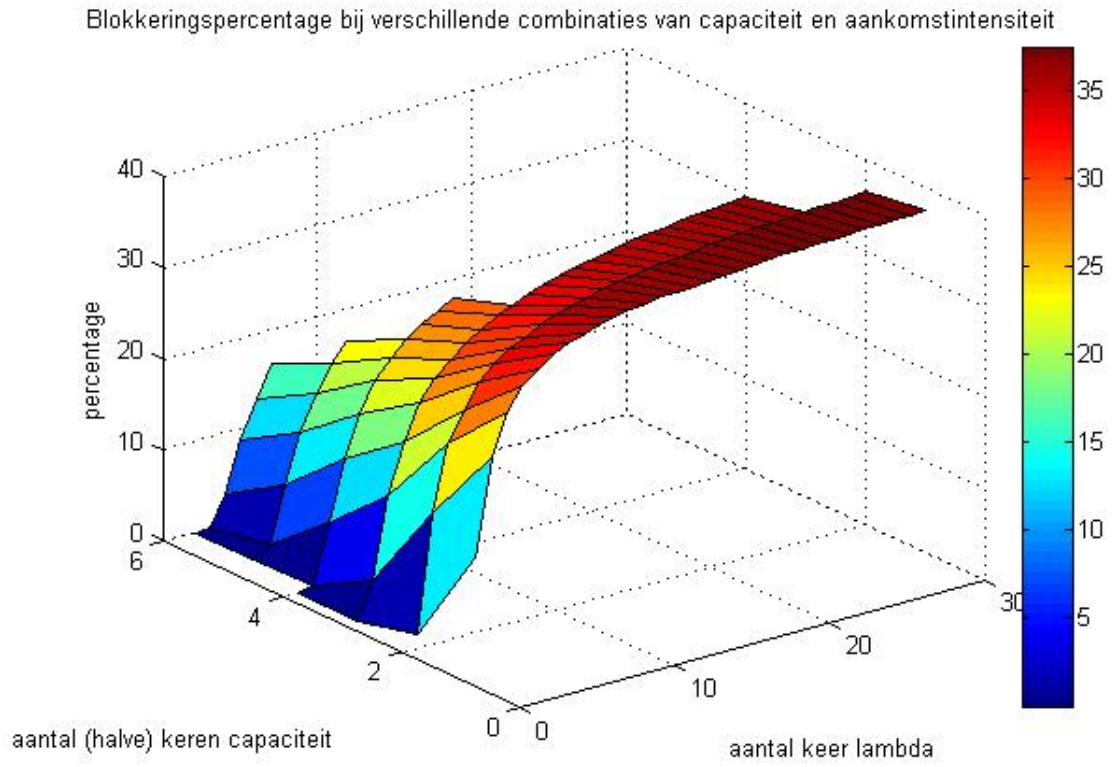
In dit figuur is gekozen voor de volgende input waarden:

Parameter	Waarde	Parameter	Waarde
λ_0	12	R_1	2
λ_1	23	R_2	4
λ_2	14	R_3	3
λ_3	18	R_4	7
λ_4	21	R_5	5
λ_5	13	R_6	4
R_0	9	R_7	2

De μ_i is dezelfde als bij het model van Capio.
Er is gekozen om λ en de capaciteit iedere keer met een gehele factor toe te laten nemen, dus een factor x met $x \in Z$. De λ laten we meer toenemen dan de capaciteit, dit omdat meer capaciteit geen duidelijke waarden meer geeft.
Uit de figuur kun je de lijnen voor de verschillende aankomstintensiteiten en de verschillende capaciteiten halen. Je kunt zien dat het totale blokkeringspercentage toeneemt tot ongeveer 35 procent, daarna stijgt die minimaal. De eerste toename van de λ geeft een grotere toename van het blokkeringspercentage dan verdere toenames.
Wat ook te zien valt is dat hoe meer capaciteit je inzet hoe lager het blokkeringspercentage

wordt. Als je al 4 keer de capaciteit neemt dan zit het blokkeringspercentage bij de eerste paar $\lambda^{'}s$ altijd onder de 15 procent. Wanneer men twee van de drie onbekende weet kan de derde bepaald worden, aan de hand hiervan kun je eisen aan je systeem stellen.
Bij een hogere factor voor de aankomstintensiteit neemt het blokkeringspercentage minder snel af per extra capaciteit dan bij een lagere factor voor de aankomsintensiteit. In het 3D figuur kun je dit zien aan het bollende vlak van het figuur. Het blokkeringspercentage neemt als een soort wortelfunctie toe voor de aankomstintensiteit. Daarnaast neemt het met een soort dalparabool af voor de capaciteit. Waarbij de grootte hiervan afneemt voor beiden kanten.

Voor dit figuur zijn we van een nieuwe situatie uitgegaan.

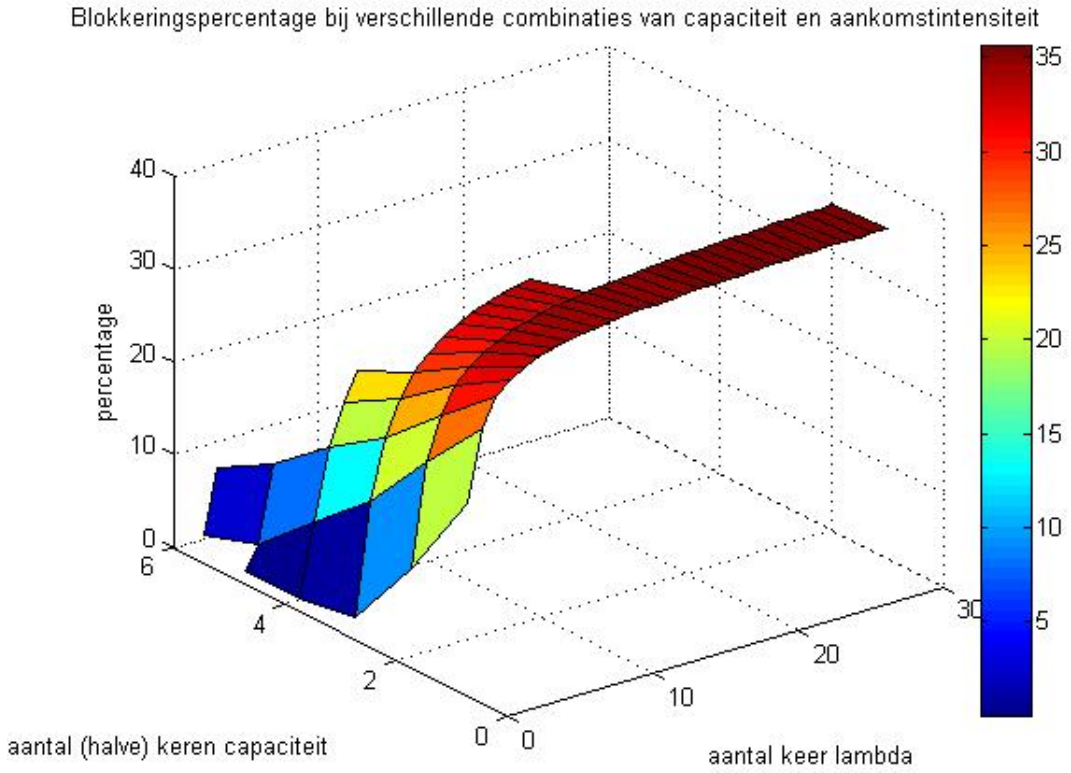


Figuur 12

We hebben de λ_i van de vorige situatie gebruikt maar we hebben in dit geval de capaciteit van iedere groep verhoogd. De capaciteit van iedere individuele groep is met 2 opgehoogd.
λ neemt iedere keer toe met een gehele factor, dit is de factor x waarvoor geldt $x \in Z$. Daarnaast neemt de capaciteit iedere keer met een factor half toe, dus een factor $\frac{y}{2}$, met $y \in Z$. Dus de capaciteit kan toenemen met een bijvoorbeeld 0.5 & 1 & 1.5 enz. De λ laten we weer meer toenemen dan de capaciteit, dit omdat meer capaciteit geen duidelijke waarden meer geeft. Dit komt omdat ons rekenmodel deze berekeningen niet kan maken.
Uit de figuur kun je de lijnen voor de verschillende aankomstintensiteiten en de verschillende capaciteiten halen. Je kunt zien dat het totale blokkeringspercentage toeneemt tot ongeveer

37 procent. Net als bij het vorige figuur kun je zien dat de eerste toename van de λ een grotere toename geeft van het blokkeringspercentage dan verdere toenames.
Wat ook te zien valt is dat hoe meer capaciteit je inzet hoe lager het blokkeringspercentage wordt. Als je al 1.5 keer de capaciteit neemt dan zit het blokkeringspercentage bij de eerste paar $\lambda^{'}s$ altijd onder de 15 procent.
Het verschil met het vorige figuur is dat we hier meer verschillen kunnen zien, er zijn meer vlakken te zien. Daarnaast zien we dat de afname in percentage per extra capaciteit lager is dan het vorige figuur.
Het vlak heeft wel weer dezelfde vorm als in figuur 11.

We hebben hierna weer een ander situatie bekeken.



Figuur 13

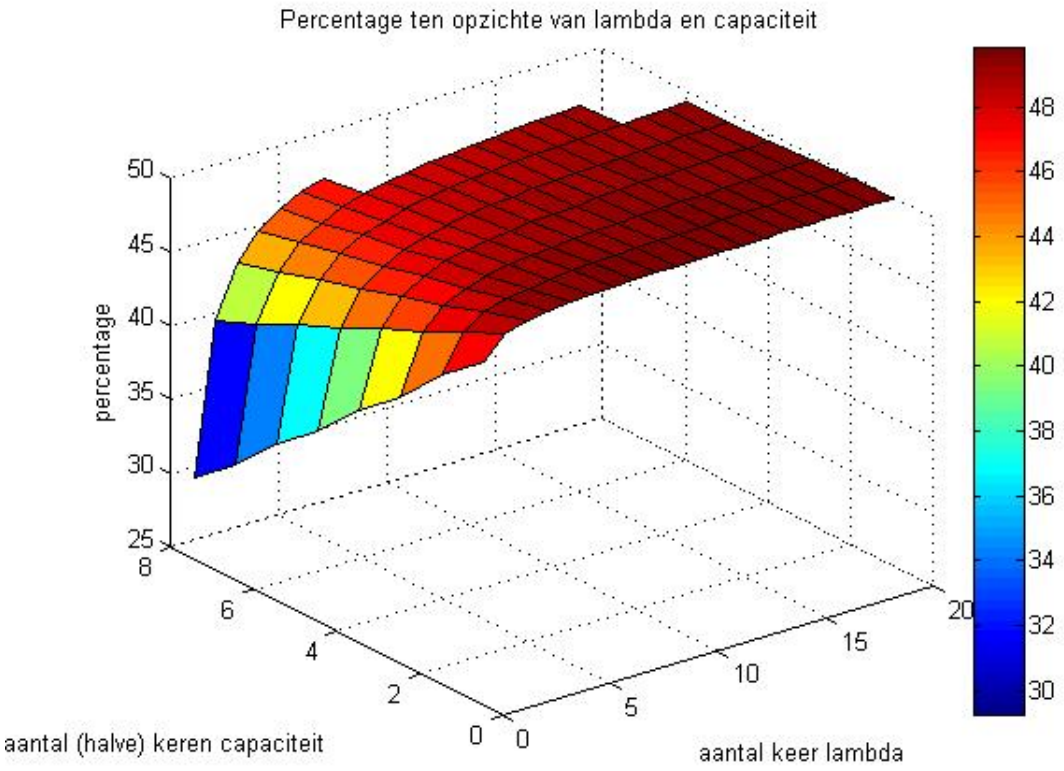
In deze situatie is gekozen voor de capaciteit die we ook bij figuur 12 gebruikt hebben, daarnaast hebben we de λ_i genomen van het model van Capio. Dus de λ die gegeven is in figuur 9. Er is gekozen om net als in figuur 12 de capciteit met een factor een half toe te laten nemen en de aankomstintensiteit weer met een gehele factor. Het verschil zit hem dit keer in de inputwaarden die we gekozen hebben. De λ is hoger gekozen waardoor het blokkeringspercentage direct al hoger ligt. Hierdoor is er net iets meer capaciteit nodig om dit percentage onder de 15 procent te laten dalen.
Net als in figuur 11 en 12 heeft de grafiek dezeldfe vorm. We gaan er dus vanuit dat dit de vorm is die deze soorten grafieken volgen voor dit specifieke model.

Uit de 3 figuren die we nu hebben laten zien blijkt dat voor een model in de vorm van die Capio, het model ziet er dus uit als figuur 9, het blokkeringspercentage toeneemt bij een toename van het aantal keer λ en dat deze toename steeds kleiner wordt. De grafiek lijkt te convergeren, omdat de stijging steeds minder wordt, maar blijft wel degelijk stijgen. Dit gebeurt dan wel minimaal. Daarnaast valt er te zien dat voor een toenemende capaciteit het blokkeringspercentage daalt, voor hogere $\lambda's$ gaat dit minder snel.

We hebben ook een 3D grafiek gemaakt voor het simpele model, dus het model dat is weergegeven in figuur 8. Daarbij is gebruikt gemaakt van de volgende input waarden:

Parameter	Waarde	Parameter	Waarde
λ_0	80	μ_2	2.5
λ_1	100	μ_3	3
λ_2	150	R_0	8
λ_3	120	R_1	7
μ_0	2	R_2	9
μ_1	2	R_3	8
μ_{over}	2	-	-

Bij deze input hoort de volgende grafiek:

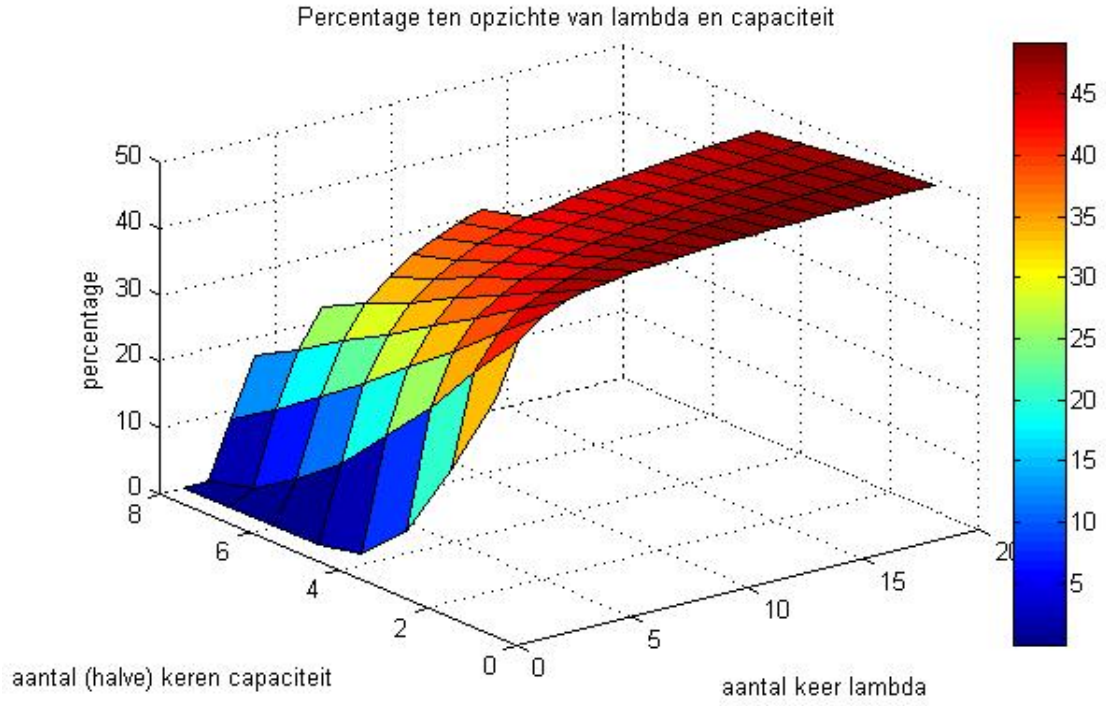


Figuur 14

De capaciteit neemt weer iedere keer met een factor 0.5 toe en de λ met een hele factor. Ook voor het simpele systeem zien we dat de grafiek dezelfde vorm heeft als bij het model van

Capio. De grafiek gaat zeer snel naar een bepaalde waarde toe, daarna is de stijging bijna verwaarloosbaar. Dit komt omdat λ zo groot gekozen is.
Daarnaast neemt die met iedere λ toe in blokkeringspercentage en neemt het blokkeringspercentage af voor een grotere capaciteit. Alleen is die niet het geval bij grotere $\lambda's$ daar blijft het blokkeringspercentage rond de 50 procent omdat er dan teveel klanten aankomen voor de capaciteit die er is.
Door deze hoge λ gaat het percentage ook niet gauw onder de 30 procent komen.

Hieronder hebben we een grafiek voor dezelfde situatie alleen hebben we de oorspronkelijke $\lambda's$ met een factor 4 verkleind. Daardoor stijgt de grafiek minder snel en neemt die voor extra capaciteit eerder af.



Figuur 15

Je ziet ook nu weer dezelfde vorm terug in de grafiek als bij de eerdere grafieken. Doordat we de begin λ dusdanig verkleind hebben neemt het blokkeringspercentage wel af tot onder de 10 procent voor een grotere capaciteit. Voor de eerste paar verhogingen van λ kan doormiddel van extra capaciteit inzetten er nog voor gezorgd worden dat het blokkeringspercentage onder de 10 procent komt. Daarnaast begint de grafiek al met een lager blokkeringspercentage, omdat er minder klanten aankomen in het begin.

Als we alle figuren goed bekijken dan stijgt het blokkeringspercentage voor grotere $\lambda's$ maar wordt deze stijging steeds minder en daalt het blokkeringspercentage voor extra capaciteit, maar deze daling wordt ook steeds minder. Dit is terug te zien in zowel het model van Capio als in het simpele model. We gaan er dan dus ook vanuit dat dit het figuur is wat een overflow model volgt, hoe groot deze ook is.

7.3.3 Threshold

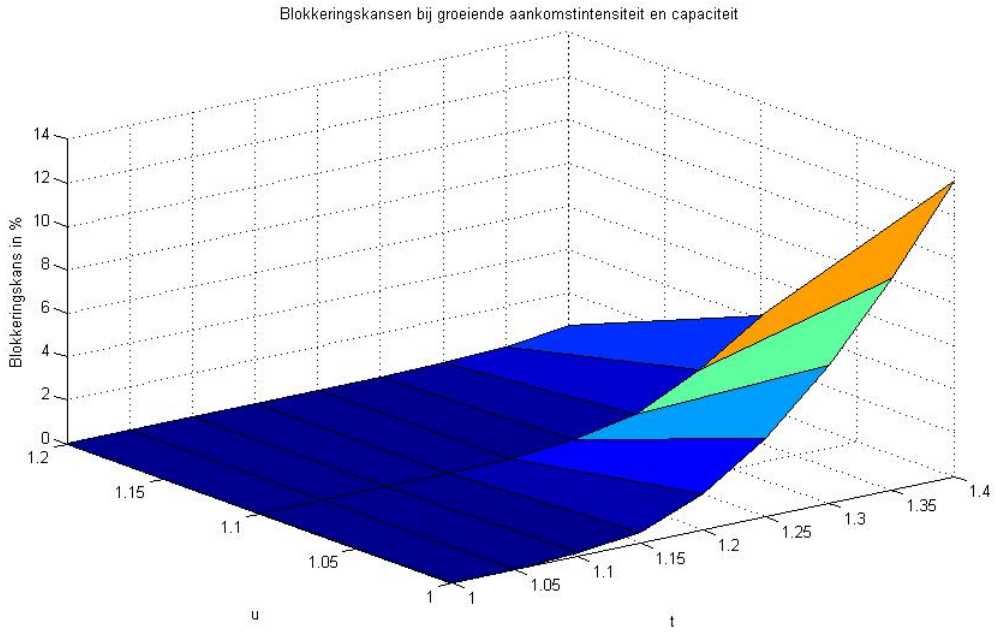
Als eerste gaan we weer kijken naar het gewone knapsack model, dus er is geen C_i gedefinieerd. We laten λ met een constante toenemen, dit betekent dat we een veelvoud nemen van de orginele λ_i van het Capio model. De input van het syteem is te zien in tabel 16. Hierbij geeft t de factor aan waarmee we de intensiteit mee gaan vermedigvuldigen. We nemen aan dat $C = 102u$, waarbij u de factor aangeeft waarmee we de capaciteit mee gaan vermedigvuldigen. De resultaten zijn te zien in tabel 17. Deze resultaten hebben we vervolgens geplot. In figuur 18 is deze te zien. Bij deze plot hebben we geen rekening gehouden met het betrouwbaarheidsinterval en alleen naar de schatting van B_k.

Aankomstintensiteiten	Waarde	Bedieningsduur	Waarde
λ_1	30t	μ_1	1
λ_2	40t	μ_2	8.6
λ_3	28t	μ_3	2.4
λ_4	20t	μ_4	2.4
λ_5	30t	μ_5	2.4
λ_0	40t	μ_0	6

Figuur 16: Input voor simpele knapsack model.

	$u = 1$	$u = 1.1$	u=1.2
$t = 1$	0.0042 ± 0.0016	0.000010 ± 0.000017	0 ± 0
$t = 1.1$	0.16 ± 0.03	0.0014 ± 0.0006	0 ± 0
$t = 1.2$	1.69 ± 0.15	0.060 ± 0.012	0.00045 ± 0.00027
$t = 1.3$	6.34 ± 0.42	0.87 ± 0.10	0.030 ± 0.007
$t = 1.4$	13.57 ± 0.81	4.18 ± 0.31	0.51 ± 0.06

Figuur 17: Blokkeringskansen in % voor iedere call typen bij verschillende t en u.

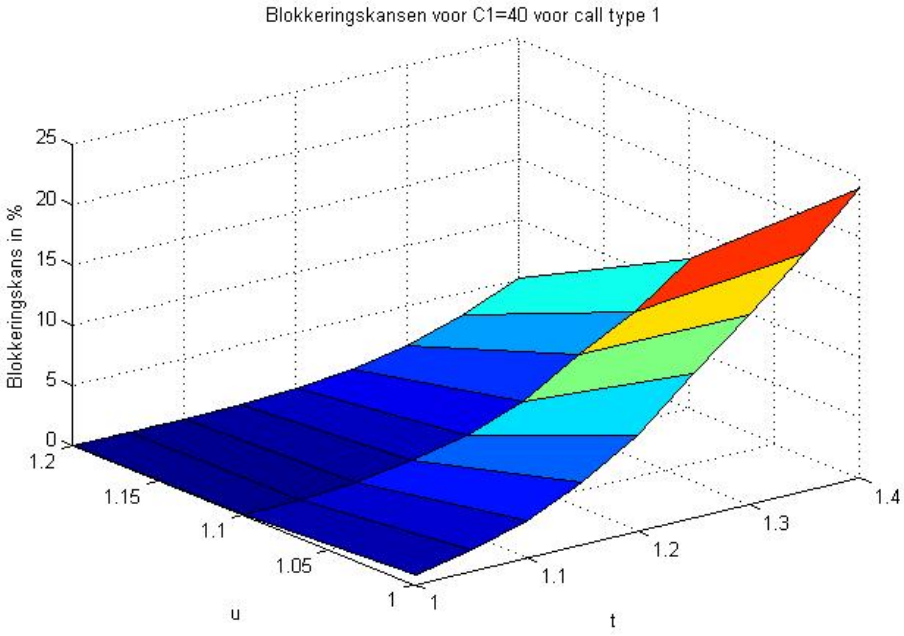


Figuur 18: Blokkeringskansen in % voor verschillende t en u.

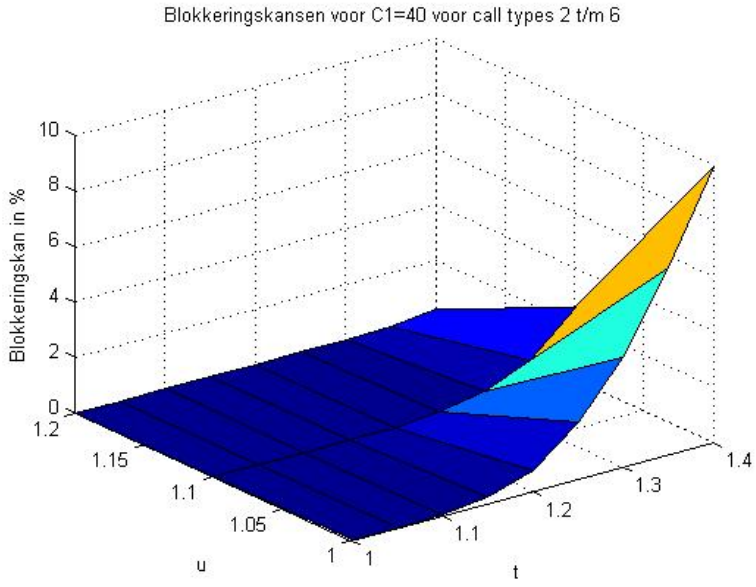
In deze resultaten is er een duidelijk verband te zien tussen de groeiende capaciteit en aankomstintensiteit. Indien de blokkeringskans groot wordt, bijvoorbeeld bij $t = 1.4$ is er een lagere u nodig om dit naar een acceptabele waarde te brengen, bijvoorbeeld $u = 1.1$. Het is misschien mogelijk om de blokkeringskansen af te laten nemen, door een goed gekozen C_i te kiezen. Wij gaan hier kijken naar een waarde voor C_1, aangezien de load ($\rho = \frac{\lambda}{\mu}$) daar het hoogste is. We kijken naar $C_1 = 40$. De resultaten staan in tabel 19. De plot van deze data is te zien in 20 en 21. Er is goed te zien dat de blokkeringskansen voor de call typen 2-6 afneemt, terwijl die van call type 1 stijgt. Het kan dus voordelig zijn om een inderdaad een C_1 in te stellen, zodat de blokkeringskansen van de andere groepen afneemt. In ruil daarvoor wordt de blokkeringskans van klanten van type 1 significant hoger.

		$u = 1$	$u = 1.1$	u=1.2
$t = 1$	Call type 1	0.79 ± 0.07	0.070 ± 0.011	0.0034 ± 0.0012
	Call type 2-6	0.0014 ± 0.0008	0 ± 0	0 ± 0
$t = 1.1$	Call type 1	3.11 ± 0.19	0.55 ± 0.05	0.054 ± 0.010
	Call type 2-6	0.066 ± 0.016	0.00052 ± 0.00033	0 ± 0
$t = 1.2$	Call type 1	7.92 ± 0.37	2.27 ± 0.15	0.42 ± 0.91
	Call type 2-6	0.75 ± 0.10	0.026 ± 0.008	0.00027 ± 0.00021
$t = 1.3$	Call type 1	15.70 ± 0.57	6.66 ± 0.32	1.69 ± 0.13
	Call type 2-6	3.96 ± 0.31	0.42 ± 0.06	0.015 ± 0.005
$t = 1.4$	Call type 1	24.00 ± 0.79	6.66 ± 0.32	5.07 ± 0.27
	Call type 2-6	9.94 ± 0.59	2.56 ± 0.23	0.26 ± 0.04

Figuur 19: Blokkeringskansen in % voor iedere call typen bij verschillende t en u met $C_1 = 60$



Figuur 20: Blokkeringskansen in % voor verschillende t en u met $C_1 = 40$ voor call type 1



Figuur 21: Blokkeringskansen in % voor verschillende t en u met $C_1 = 40$ voor call typen 2 tot 6

8 Conclusie

In dit verslag hebben we gekeken naar verschillende modellen die een verband leggen tussen de capaciteit, aankomst van klanten en de blokkeringskans. We hebben gekeken naar het Erlang Loss model, overflow model, trunk reservation en threshold. Het Erlang loss systeem geeft een snel antwoord op de vraag. Echter moeten er veel aannames gemaakt worden, bijvoorbeeld het gelijk beschouwen van alle call types. Bij het overflow model wordt er rekening gehouden met de verschillende specialisaties. Het nadeel is dat er aannames gemaakt moeten worden over de bedieningsduur van klanten bij de overflow groepen. Trunk reservation zorgt voor een goede controle van het aantal klanten van welke soort je in het syteem toelaat. Het nadeel is dat het lastig te berekenen is voor grote systemen, hiervoor hebben wij gebruik gemaakt van threshold. Threshold is een manier om trunk reservation te benaderen. Threshold politieken zijn onderdeel van het knapsack model. Bij dit model hou je geen rekening met welke tolken er bezet zijn en welke niet. Hierdoor weet je niet welke capaciteit er nog aanwezig is voor een bepaald soort klant.
Het uitrekenen van elk van deze modellen is afhankelijk van de volgende variabelen: de aankomst van klanten, de bedieningsduur van elke soort klant, de capaciteit van tolken en de blokkeringskans. De bedieningsduur is bekend. Dit betekent dat als je twee van de overige drie weet, je de derde kan uitrekenen. Dus uit de capaciteit en aankomst van klanten kun je de blokkeringskans bepalen, maar ook kun je uit de blokkeringskans en de aankomst van klanten de nodige capaciteit afleiden. Capio heeft ons een model gegeven, waar de aankomst van de klanten en de capaciteit bekend zijn. Van dit model hebben we de blokkeringskans bepaald. Deze blokkeringskans was erg laag, wat betekent dat Capio in het begin zich geen zorgen hoeft te maken over de capaciteit tolken. Als Capio wil weten hoeveel capaciteit zij nodig hebben bij een bepaalde vraag, bedieningsduur en maximale blokkeringskans, kunnen ze één van deze modellen gebruiken om deze benodigde capaciteit te bepalen.

9 Aanbevelingen

Capio wil graag weten wanneer ze extra capaciteit nodig hebben om de stijgende vraag te kunnen behandelen. Indien Capio de maximale blokkeringkans, de bedieningsduur en verwachte vraag weet, kunnen ze de benodige capaciteit berekenen. Als Capio vermoedt dat de vraag aan het stijgen is ten opzichte van wat ze verwachtten, kunnen ze op die manier de nodige capaciteit bepalen. Het komt er op neer dat Capio zijn benodigde capaciteit op ieder moment kan bepalen. Ze hebben dan alleen de aankomstintensiteit en het maximale blokkeringspercentage nodig. Aan de hand van de drie modellen die gegeven zijn kan Capio bepalen welke capaciteit bij hun input hoort. Welk model Capio moet gebruiken is afhankelijk van het beleid van Capio. Wij kunnen geen uitspraak doen over welk model het beste is voor de situatie van Capio.

10 Discussie

In dit verslag zijn we niet ingegaan op het feit dat Capio graag 3 tolken wil aanbieden, zodat de klant de keuze heeft welke tolk hij wil hebben. De manier die Capio voor zich ziet is dat het aan de hand van een soort 'ranking' gaat. Na een gesprek krijgt de klant de kans om de tolk te beoordelen, bij een goede beoordeling stijgt de tolk op de ranking en bij een slechte beoordeling daalt hij. Tolken die hoger op de ranking staan worden eerder en vaker aangeboden, maar er moet ook een kleine random methode inzitten. Deze random methode zit erin om ervoor te zorgen dat iedere tolk wel kans krijgt om aangeboden te worden, ook al staat deze laag op de ranking, want dan krijgt deze tolk de kans zich omhoog te werken op de ranking. Daarnaast moet deze random methode ervoor zorgen dat niet steeds dezelfde tolk de klus krijgt. Een random methode die gebruikt kan worden is om een tolk die net een klant geholpen heeft op een soort wachtlijst te zetten voor een bepaalde tijdsperiode, in deze periode is de tolk dan eigenlijk niet inzetbaar door het systeem, behalve als er te weinig tolken zijn om aan de vraag te voldoen. Dan wordt de tolk natuurlijk wel ingezet.
Hoe komt de klant uiteindelijk tot een selectie van 3 tolken, dit kan door verschillende mogelijkheden. Mocht er in totaal nog maar 3 tolken beschikbaar zijn, dan worden deze aangeboden. Maar als er nog x-aantal tolken beschikbaar zijn, met $x \geq 3$, dan krijgt de klant vragen voorgeschoteld. Dit zijn bijvoorbeeld vragen in de richting welke specialiteit ze willen, wat voor een soort prijsklasse, wat voor een soort ranking en dergelijke. Met de verschillende modellen die in het verslag zijn uitgelegd, kan gekeken worden hoeveel tolken er in iedere speciefieke groep nog beschikbaar zijn. Mocht het bijvoorbeeld zo zijn dat de klant een medische tolk wil en er zijn nog 3 tolken beschikbaar in de medische groep, dan kan ervoor gekozen worden om deze 3 tolken aan te bieden. Maar mocht het zo zijn dat er nog heel veel tolken beschikbaar zijn in de groep medisch-juridisch, dan kan er ook voor worden gekozen om hier tolken uit aan te bieden en de tolken in de medische groep nog even te reserveren. Dit reserveren kan het beste gedaan worden door middel van trunk reservation, danwel threshold. De overflow groep kan dan weer het beste gedaan worden door het overflow model.
De keuze tussen 3 tolken implementeren in het model is heel erg afhankelijk van welk model je kiest. Bij Trunk kan dit bijvoorbeeld door een deel van de tolken te reserveren voor een specifiek type call, bij overflow kan dit doormiddel van alleen specifieke tolken aan te bieden als hier echt om gevraagd wordt en anders voor de overflow groep te kiezen.

Een punt waar nog gekeken kan worden is de rekenkracht van trunk reservation, het model van Capio was te groot om te berekenen met Matlab. Dit komt omdat er meer dan 400 miljoen toestanden bekeken moesten worden. Voor een kleinere model is er wel te werken met Matlab. Een manier om dit op te lossen is om te kijken of het te progammeren is in $C++$, of om een betere computer (een computer met meer geheugen) te hebben waarop je het matlabscript een hele lange tijd kan laten runnen.
Daarnaast is een punt wat beter had gekund het betrouwbaarheidsinterval van threshold, om een klein betrouwbaarhiedsinterval te hebben moet je meer random toestanden genereren. Hierdoor wordt je antwoord steeds betrouwbaarder. Maar extra random toestanden genereren betekent ook dat het extra tijd kost om het uit te rekenen, hierdoor neemt de rekentijd toe. Dit kan ervoor zorgen dat het uitrekenen teveel tijd kost waardoor het niet door een normale computer kan gebeuren. Om het in ons verslag wel te kunnen hebben we voor minder random toestanden gekozen, waardoor het betrouwbaarheidsinterval breder is geworden.

Een ander punt waar nog naar gekeken kan worden is de data, de data die wij gebruiken in ons verslag is vooral gebaseerd op realistische data of data die ons is aangereikt door Capio. Maar van sommige data die we gebruikt hebben weten we niet zeker of dit klopt. We hebben met enkele gegevens zo goed mogelijk de werkelijkheid proberen te bereiken, maar dit is waarschijnlijk niet in alle gevallen gelukt. Daarnaast is het model dat gebruikt wordt gebasseerd op de verwachting hoe het in het begin gaat, maar als Capio snel groeit is ons model naar een bepaalde periode niet goed meer bruikbaar en moeten de formules toegepast worden op een complexere model. Voor Trunk/Treshold betekent dit dat er meer toestanden bekeken moeten worden en voor overflow betekent dit dat er vaker ERM toegepast moet worden.

Voor verder onderzoek zijn de volgende punten van belang, allereerst is het voor een groter model handig dat er wat gedaan wordt aan de rekenkracht. Onze laptops kunnen het niet aan om hele grote modellen, met grote waarden te berekenen. Mocht men willen dat dit wel wordt gedaan dan is het handig om gebruik te maken van computers met meer rekenkracht, hierdoor zijn de antwoorden sneller gegeven en kan er sneller worden gekeken welk model op welk moment het beste werkt. Daarnaast is een punt voor volgend onderzoek om te kijken of er een combinatie van overflow en trunk reservation te maken. Beiden modellen werken op hun eigen manier goed, maar een combinatie van deze twee zou ervoor kunnen zorgen dat er 'en gebruik gemaakt kan worden van de overflow groepen 'en dat er capaciteit gereserveerd kan worden voor de belangrijke aanvragen. We denken namelijk zelf als je een combinatie van deze twee modellen weet te maken dat dit het beste model is voor bijna elke situatie. Ook is er niet gekeken naar de optimale situatie van overflow, er zijn geen berekeningen gemaakt met de terugkoppeling. De reden waarom dit niet gedaan is omdat een klant en capio het niet gaan waarderen als er tijdens een gesprek gewisseld wordt van tolk. Als iemand een tolk heeft voor zijn gesprek dan wil hij hiermee ook zijn gesprek afmaken. Wel is het de moeite waard om hier naar te kijken, misschien blijkt het wel dat wanneer men de overflow terugkoppeld het blokkeringsgehalte afneemt en het dus een betere situatie oplevert.
Het kan ook handig zijn om de berekeningen voor ieder model nog een keer te doen als er meer data bekend is. Dit omdat de data waarop ons model nu gebasseerd is nog zeker aangevuld kan worden. De data is namelijk een indicatie, maar als Capio een periode loopt dan heeft het zelf genoeg data verzameld om een beter model op te stellen. Hierdoor kan het gebeuren dat er over gestapt moet worden op een ander wiskundig model.

In de appendix is een stuk bijgevoegd waarbij de optimalisatiefunctie wordt uitgelegd. Dit is toegevoegd omdat we eerst van plan waren om met een simulatie de modellen met elkaar te vergelijken. In deze simulatie waren we van plan om de capaciteit en de aankomstintensiteit van de tijd af te laten hangen.
De optimalisatiefunctie bevat een deel over de bezettingsgraad van het systeem, hoeveel klanten worden er geblokkeerd. Daarnaast is er een deel toegevoegd over de tolk tevredenheid, dit betekent hoeveel tolken werken op ieder moment in het systeem. Deze twee samen zorgen voor de optimalisatiefunctie. Mocht iemand van plan zijn om te simuleren dan kan hij gebruik maken van de optimalisatiefunctie die is opgesteld. Voor beiden delen is precies uitgelegd hoe de formules die hierbij horen, gebruikt moeten worden.

Referenties

[1] Ger Koole, Jérôme Talim, *Exponential Approximation of Multi-Skill Call Centers Architecture*, Vrije Universiteit - Division of Mathematics and Computer Science, 2000.

[2] Wyean Chan, Ger Koole and Pierre L'Ecuyer, *Dynamic Call Center Routing Policies Using Call Waiting and Agent Idle Times*, 2014.

[3] O. Zeynep Aksin and Patrick T. Harker, *Computing performance measures in a multi-class multi-resource processor-shared loss system*, september 1998.

[4] Tevfik Aktekin and Refik Soyer, *Bayesian Analysis of Queues with Impatient Customers: Applications to Call Centers*, juni 2012, DOI: 10.1002/nav.21499.

[5] Dr. János Sztrik, *Basic Queueing Theory*, University of Debrecen, Faculty of Informatics.

[6] Keith W.Ross, PhD, *Multiservice Loss Models for Broadband Telecommunication Networks*, University of Pennsylvania, Department of Systems Engineering, Philadelphia, USA, 1995.

[7] Roberto Cordone, Andrea Piselli, Paolo Ravizza, Giovanni Righini, *Optimization of Multi-skill Call Centers Contracts and Work-shifts*, 2011, DOI: 10.1287/serv.3.1.67.

[8] Robert B. Cooper, *Introduction to Queueing Theory (Second Edition)*, Computer Systems and Management Science, Florida Atlantic University,

[9] Nelly Litvak, Marleen van Rijsbergen, Richard J. Boucherie and Mark van Houdenhoven, *Managing the overflow of intensive care patients*, University of Twente, The Netherlands, 2006.

[10] Maartje E. Zonderland, Richard J. Boucherie, Michael W. Carter and David A. Stanford, *The Emergency Observation and Assessment Ward*, University of Twente, Department of Applied Mathematics, The Netherlands, 2011.

[11] J. Cole Smith and Sheldon H. Jacobson, *An Analysis of the Alias Method for Discrete Random-Variate Generation.*, INFORMS Journal on Computing, Vol. 17, No.3, Summer 2005 pp321-327. `http://dx.doi.org/10.1287/ijoc.1030.0063`

[12] R.I. Wilkinson. Theories for toll traffic engineering in the USA. The Bell System Technical Journal 35 (1956) 421514.

[13] Y. Rapp. Planning of junction networks in a multiexchange area. Ericsson Technics 20 (1) (1964) 77130.

11 Appendix

11.1 Rekenvoorbeeld overflow model

Voor het model geldt dat de aankomst bij de 'eerste primary ward' 5 klanten per uur is, dit betekent dat $\lambda_1 = 5$. De bedieningsduur is 4 klanten per uur, dit betekent $\mu_1 = 4$. De maximale capaciteit van deze 'ward' wordt gegeven door $R_1 = 2$. Voor de rest gelden de volgende parameters: $\lambda_0 = 10,\ \mu_0 = 10,\ R_0 = 6,\ \lambda_2 = 5,\ \mu_2 = 3,\ R_2 = 3,\ \lambda_3 = 7,\ \mu_3 = 5,\ R_3 = 4$. En de bedieningsduur van de 'overflow' klanten wordt gezet op $\mu_{over} = 5$.
De staat van het systeem wordt dan:

$$S := (\mathbf{n} = n_{00}, n_{01}, n_{02}, n_{03}, n_1, n_2, n_3), \quad n_i \leq R_i \ \forall i$$

$$\sum_{i=0}^{3} n_{0i} \leq 6 \ \forall i \tag{49}$$

We zouden nu de algemene evenwichtsvergelijking kunnen in vullen, maar makkelijker is het om de 'versimpelde' evenwichtsvergelijking in te vullen. Maar omdat we geïntresseerd zijn in de E_i en V_i van het systeem, omdat we dan ERM kunnen toepassen en daardoor de blokkeringskansen en de verwachting van het aantal klanten in de verschillende 'wards' kunnen uitrekenen, gaan we nu kijken naar formule (22) en (23). Deze formules gebruiken we omdat we met de uitkomsten hiervan formule (27) en (28) kunnen gebruiken.
Hieronder staan de berekeningen van E_i en V_i voor $i = 1$ uitgewerkt.

$$E_1 = \frac{\lambda_1}{\mu_{over}} \cdot B(\rho_1, R_1) = \frac{5}{5} \cdot B(\frac{5}{4}, 2) = 0.2557 \tag{50}$$

$$V_1 = \frac{\lambda_1}{\mu_{over}} \cdot g_1(R_1) + E_1 - (E_1)^2 = \frac{5}{5} \cdot 0.1333 + 0.2557 - (0.2557)^2 = 0.3236 \tag{51}$$

Waarbij $g_1(R_1)$ als volgt is berekend (zie formule (22) en (23):
Allereerst wordt er het stelsel van vergelijkingen opgesteld:

$$1.)\ n_1 = 0: \quad 10g_1(0) = 4g_1(1)\ note: g_1(-1) = 0 \tag{52}$$

$$2.)\ n_1 = 1: \quad 14g_1(1) = 5g_1(0) + 8g_1(2) \tag{53}$$

$$3.)\ n_1 = 2 = R_1: \quad 13g_1(2) - 5 \cdot B(\rho_1, R_1) = 5g_1(1) \tag{54}$$

De enigste onbekende in dit stelstel is de $g_1(i)$, deze zouden we kunnen berekenen doormiddel van het systeem: $A \cdot x = B$; waarbij x de onbekende vector is. De oplossing wordt dan gegeven door $x = A^{-1} \cdot B$. Hieronder staat hoe deze A, B en x eruit zien:
-A is een matrix, deze bevat de factoren voor de $g_1(i)$;

$$\begin{bmatrix} 10 & -4 & 0 \\ -5 & 14 & -8 \\ 0 & -5 & 13 \end{bmatrix}$$

-B is de volgende vector, deze geeft de uitkomst van ieder van de vergelijkingen;

$$\begin{bmatrix} 0 \\ 0 \\ 0.2577 \end{bmatrix}$$

-En x is de vector die alle waarde voor $g_1(i)$ bevat;

$$\begin{bmatrix} g_1(0) \\ g_1(1) \\ g_1(2) \end{bmatrix}$$

De oplossingsvector x wordt dan gegeven door;

$$\begin{bmatrix} 0.0355 \\ 0.0899 \\ 0.1333 \end{bmatrix}$$

Dit betekent dat $g_1(2) = 0.1333$
Omdat we nu de berekening voor een 'ward' hebben laten zien staan hieronder de E_i en de V_i van de andere 'wards'.

$$E_0 = \frac{\lambda_0}{\mu_0} = \frac{10}{10} = 1$$
$$V_0 = \frac{\lambda_0}{\mu_0} = \frac{10}{10} = 1$$
$$E_1 = \frac{\lambda_1}{\mu_{over}} \cdot B(\rho_1, R_1) = \frac{5}{5} \cdot B(\frac{5}{4}, 2) = 1 \cdot 0.2557 = 0.2557$$
$$V_1 = \frac{\lambda_1}{\mu_{over}} \cdot g_1(R_1) + E_1 - (E_1)^2 = \frac{5}{5} \cdot 0.1333 + 0.2557 - (0.2557)^2 = 0.3236$$
$$E_2 = \frac{\lambda_2}{\mu_{over}} \cdot B(\rho_2, R_2) = \frac{5}{5} \cdot B(\frac{5}{3}, 3) = 1 \cdot 0.1598 = 0.1598$$
$$V_2 = \frac{\lambda_2}{\mu_{over}} \cdot g_2(R_2) + E_2 - (E_2)^2 = \frac{5}{5} \cdot 0.0456 + 0.1598 - (0.1598)^2 = 0.2094$$
$$E_3 = \frac{\lambda_3}{\mu_{over}} \cdot B(\rho_3, R_3) = \frac{7}{5} \cdot B(\frac{7}{5}, 4) = \frac{7}{5} \cdot 0.04 = 0.0561$$
$$V_3 = \frac{\lambda_3}{\mu_{over}} \cdot g_3(R_3) + E_3 - (E_3)^2 = \frac{7}{5} \cdot 0.0153 + 0.0561 - (0.0561)^2 = 0.0744$$

We kunnen nu de sommaties nemen van de E_i en V_i, deze hebben we nodig om voor de 'equivalente' server de 'load' en capaciteit te bepalen. De sommatie over E_i levert $E = 1.4716$ op en de sommatie over V_i levert $V = 1.5779$ op. Om dan de a en R van het nieuwe systeem te bepalen kunnen we gebruik maken van formule (31) en (32), maar beter is het om te schatten met behulp van 'Rapp':

$$a = V + 3 \cdot \frac{V}{E}(\frac{V}{E} - 1) = 1.5779 + 3 \cdot \frac{1.5779}{1.4716}(\frac{1.5779}{1.4716} - 1) = 1.8103 \tag{55}$$

$$R = \frac{a(E + \frac{V}{E})}{E} + \frac{V}{E - 1} - E - 1 = \frac{1.8103(1.4716 + \frac{1.5779}{1.4716})}{1.4716} + \frac{1.5779}{1.4716 - 1} - 1.4716 - 1 = 4.004 \tag{56}$$

Nu zegt 'Cooper' dat het beter is om R naar beneden af te ronden, zodanig dat het een geheel getal is, en dan a opnieuw te schatten met:

$$a = \frac{(\lfloor R \rfloor + E + 1)(E + \frac{V}{E} - 1)}{E + \frac{V}{E}} = \frac{(4 + 1.4716 + 1)(1.4716 + \frac{1.5779}{1.4716} - 1)}{1.4716 + \frac{1.5779}{1.4716}} = 3.93 \tag{57}$$

Dit geeft namelijk een betere benadering.
We kunnen nu een benadering geven van het aantal klanten dat gemiddeld geweigerd wordt bij de 'overflow' groep, de formule die hiervoor gebruikt wordt is:

$$W = aB(R_0 + R, a) = 3.93B(6 + 4, 3.93) = 3.93 \cdot 0.048 = 0.0187 \tag{58}$$

Deze formule is analoog aan formule (10). Dit is de zogenaamde gemiddelde waarde van de 'load' $\alpha^{'}$ die de 'overflow' groep overflowt.
We kunnen nu ook berekenen wat de proportie van klanten is die de 'overflow' groep bezet vindt gegeven dat ze de 'overflow' zijn van de primaire groepen.

$$\Pi_c = \frac{aB(R_0 + R, a)}{aB(R, a)} = \frac{W}{3.93B(4, 3.93)} = \frac{0.0187}{1.1942} = 0.0157 \tag{59}$$

Van deze twee waarden zijn wij vooral geintresseerd in W, dit geeft namelijk het gemiddelde aantal klanten die het systeem verlaten aan. Omdat dit de laatste server is en deze bezet is.
Een andere manier om het systeem te bekijken is gegeven door de de 'Katz' benadering, deze geeft de blokkeringskans van elke individuele 'ward' die aankomt bij de 'overflow' groep.
Berekening Katz. Allereerst moeten we recursie gebruiken, aangezien we $v(R, R_0)$ moeten weten. We weten dat $v(R, 0) = 1$, daarna gebruiken we de formule:

$$V(R, j) = \frac{a \cdot j}{aB(R, a) \cdot (R + j - a - aB(R, a) \cdot V(R, j - 1))} \quad j = 1, 2, .. \tag{60}$$

De volgende parameters zijn nu bekend: $R = 4$, $a = 3.93$ en $R_0 = 6$. De berekening uitvoerend krijgen we:

$$\begin{aligned} v(4, 0) &= 1 \\ v(4, 1) &= -26.4856 \\ v(4, 2) &= 0.01953 \\ v(4, 3) &= 3.4801 \\ v(4, 4) &= -152.8135 \\ v(4, 5) &= 0.0877 \\ v(4, 6) &= 3.3100 \end{aligned}$$

Nu kan voor elke groep de 'Katz' benadering uitgerekend worden en met behulp hiervan kan dan voor elke groep individueel de verwachte blokkering worden gegeven. Hieronder staat voor elke groep de 'Katz' benadering:
Uitgerekend met behulp van formule (12), met de volgende parameters, a=3.93,$R = 4$,$v(R, R_0) =$ 3.3100 en $\zeta = \frac{1.5779}{1.4716} = 1.0722$, de ζ_i is afhankelijk van de groep i.

$$k_i = \frac{aB(R + R_0, a)}{aB(R, a)} \cdot (V(R, R_0)^{-1} + \frac{\zeta_i - 1}{\zeta - 1} \cdot (1 - V(R, R_0)^{-1}) = \frac{3.93B(10, 3.93)}{B(4, 3.39)} \cdot (V(4, 6)^{-1} + \frac{\zeta_i - 1}{1.0722 - 1} \cdot (1 - V(4, 6)^{-1})$$

Dit geeft:

$$k_0 = 0.0047$$
$$k_1 = 0.0450$$
$$k_2 = 0.0238$$
$$k_3 = 0.0542$$

Het verwachte aantal klanten van een 'ward' i die aanwezig zijn in de 'overflow' groep wordt gegeven door:

$$E(n_{0i}) = \frac{\lambda_i}{\mu_{over}} B(\rho_i, R_i)(1 - k_i) \tag{61}$$

Voor het verwachte 'normale' aantal klanten in de 'overflow' groep geldt de volgende formule:

$$E(n_{00}) = \frac{\lambda_0}{\mu_0}(1 - k_0) \tag{62}$$

Hier komt geen 'Erlang' formule in voor omdat er geen sprake is van 'overflow' in deze groep. De verwachtingen zijn dan:

$$E(n_{00}) = 1(1 - 0.0047) = 0.9953$$
$$E(n_{01}) = \frac{5}{5} B(\frac{5}{4}, 2)(1 - 0.0450) = 0.2461$$
$$E(n_{02}) = \frac{5}{5} B(\frac{5}{4}, 3)(1 - 0.0238) = 0.1560$$
$$E(n_{03}) = \frac{7}{5} B(\frac{7}{5}, 4)(1 - 0.0542) = 0.0530$$

Aan de hand van dit voorbeeld kunnen we dus concluderen dat:
Voor het nieuwe 'equivalente' systeem geldt dat er een dus een nieuwe primary server komt met een een capaciteit van 4. De aankomstintensiteit van dit systeem is gegeven door a is 3.93 aankomsten per uur. De gemiddelde waarde van overflow wordt gegeven door de 'load' E dit is gelijk aan 1.4716, de variantie hiervan is $V = 1.5779$.
Het nieuwe systeem heeft nu nog een deel van de klanten die geblokkeerd worden en het systeem dus verlaat, deze waarde is gegeven door $W = 0.0187$. Dit is het gemiddelde aantal klanten dat geblokkeerd wordt. Daarnaast kunnen we ook kijken naar de individuele blokkeringswaarden, k_i. Uit de k_i kun je halen dat er veel klanten van groep 3 worden geblokkeerd, meer dan van bijvoorbeeld groep 0. Om hier een goede conclusie over te kunnen geven kijken we eerst naar de verwachting van het aantal klanten in de overflow groep. Uit deze $E(n_{0i})$ halen we dat vooral klanten uit groep 1 in de overflow groep aanwezig zijn, ook zijn er veel klanten van groep 0 aanwezig maar dit is logisch want groep 0 is de overflow groep.
Om het blokkeringsgehalte van het hele systeem naar beneden te halen is het handig om extra werknemers voor groep 1 en groep 2 te nemen.

11.2 Alias Methode

De alias methode wordt behandelt in [11], we zullen hem hier ook behandelen. Als eerste definiëren we de discrete random variabele V met een zekere kansdichtheid functie $P\{V = v_i\} = p_i > 0$, $i = 1, 2, ..., n$. Hierbij is n het aantal random variabelen die we zullen genereren, aan het einde van de procedure wordt er gekozen tussen 1 van deze n waarden. Stel er is een waarde v_i waarvoor $p_i = 0$, zullen we deze waarde niet meenemen in de procedure. Het alias algoritme is gebaseerd op het genereren van een alias tabel, die de alias waarden a_j bevat met alias kans van s_j. Deze twee waarden samen vormen de alias tabel voor $j = 1, 2, ..., n$.

Procedure 1: Initialiseer $q_i = np_i$ voor $i = 1, 2, ..., n$, $s_j = 0$ voor $j = 1, 2, ..., n$, en creeër de groepen $G = \{i : q_i \geq 1\}$ en $H = \{i : q_i < 1\}$.

1. Als $(H == \emptyset)$, eindig de procedure.
   Anders, kies een $j \in H$ en $k \in G$. Laat $a_j = v_k$, $s_j = 1 - q_j$, en $q_k = q_k - q_j - 1$. Als $(q_k < 1)$, ga naar stap 2. Anders, ga naar stap 3.
2. Laat $G = G \setminus \{k\}$ en $H = H \cup \{k\}$. Ga naar stap 3.
3. Laat $H = H \setminus \{j\}$ en ga naar stap 1.

Als we de alias tabel hebben, kunnen we heel makkelijk een random V maken uit deze alias tabel. Dit gebeurd als volgt:

Procedure 2:

1. Genereer 1 uniform random getal u, met $0 < u < 1$, en selecteer $j = \lceil un \rceil$.
2. Als $(s_j == 0)$, geeft waarde v_j. Anders, ga naar stap 3.
3. Laat $S = (un - \lfloor un \rfloor)$. Als $(S \geq s_j)$, geef waarde v_j, en anders geef waarde a_j.

Dit is de simpele procedure. Meer informatie over deze methode, zoals snelheid van het algoritme, zie [11].

11.3 Optimalisatiefunctie voor simulatie

Er zijn twee elementen die echt belangrijk zijn voor onze optimalisatiefunctie, namelijk de tevredenheid van klanten én tolken. We maken gebruik van een penaltysysteem, deze willen we minimaliseren. Dit betekent dat we de 'slechtheid' van het systeem berekenen en ervoor willen zorgen dat deze minimaal is. We kiezen dus uiteindelijk de politiek die ervoor zorgt dat de penalty minimaal is.

11.3.1 Blokkeringsgraad/tevredenheid klanten

De blokkeringsgraad geeft aan hoeveel klanten het systeem verlaten zonder geholpen te worden door een tolk, deze klanten gaan dan verloren. We vinden dat een klant die het systeem verlaat zonder geholpen te worden, de tevredenheid van de klanten naar beneden haalt. Daarom zijn we geïntreseerd in een zo laag mogelijke blokkeringsgraad. De formule voor de blokkeringsgraad ziet er als volgt uit [2]:

$$A_i(\pi) = \frac{E(Z_i(\pi))}{E(M_i)} \tag{63}$$

Hierbij is $A_i(\pi)$ de blokkeringsgraad voor call type i bij politiek π. $Z_i(\pi)$ het aantal geblokkerde klanten in een tijdsperiode bij call type i. M_i is het aantal klanten van call type i in een bepaalde tijdsperiode t [2]. Als je de totale blokkeringsgraad van het systeem wilt achterhalen, haal je de indexering i weg, dan heb je de totale blokkeringsgraad in een tijdsperiode van het hele systeem. Deze formule is nog niet helemaal compleet. Om dit on onze optimalisatiefunctie te zetten, moeten we eerst wat factoren toevoegen. Hierdoor ontstaat de volgende functie:

$$F_B(\pi) = \sum_{i=1}^{L} c_{B,i} \max(A_i(\pi) - u_{B,i}, 0)^{e_{B,i}} \tag{64}$$

Hierbij is $F_B(\pi)$ is de penalty bij de blokkering van de klanten, L is aantal call types, $c_{Bi,}$ een factor voor de blokkeringsgraad bij call type i, $A_i(\pi)$ is de blokkeringsgraad voor call type i bij politiek π, $u_{B,i}$ is de threshold voor het weggaan van de klanten bij call type i en $e_{B,i}$ is een macht die je kunt gebruiken om een bepaalde call type i belangrijker te laten zijn [2]. Hierbij kunnen wij $c_{B,i}$ en $e_{B,i}$ kiezen, afhankelijk van welke call type we belangrijk vinden. We zullen waarschijnlijk $e_{B,i} = 1$ nemen, zodat het syteem lineair blijft. $c_{B,i}$ geeft dus een gewicht aan een calltype i, vinden we het bijvoorbeeld belangrijker om klanten van calltype i te behandelen in plaats van calltype j, dan is het bijbehorende gewicht voor calltype i hoger.

11.3.2 Bezettingsgraad/tevredenheid tolken

De bezettingsgraad geeft aan hoeveel het systeem gevuld is, als het systeem meer gevuld is dan geeft het dus aan dat er meer tolken aan het werk zijn. Als de bezettingsgraad laag is dan zijn er juist weinig tolken aan het werk. We noemen een tolk tevreden als hij aan het werk is.
De formule voor de bezettingsgraad ziet er als volgt uit[2]:

$$O_j(\pi) = \frac{1}{y_j T} E\left(\int_0^T G_j(\pi, t)dt\right) \tag{65}$$

Hierbij is $O_j(\pi)$ de bezettingsgraad voor skill groep j voor politiek π, y_j het aantal tolken in skill groep j, $G_j(\pi, t)$ is het aantal bezette tolken van skill groep j op tijdstip t voor politiek π. Een mogelijkheid om het komen en gaan van tolken in het model te brengen, is door de integraal in stukjes op te delen, afhankelijk van wanneer er tolken bijkomen of weggaan. De volgende formule geeft de penalty aan voor de bezetting van tolken [2]:

$$F_O(\pi) = \sum_{j=1}^{K} c_{0,j} |O_j(\pi) - \overline{O}|^{e_{O,j}} \tag{66}$$

Hierbij geeft $F_O(\pi)$ de penalty voor de bezetting van de tolken voor politiek π, $c_{O,j}$ geeft een factor aan die aangeeft hoe sterk een bepaalde tolkengroep j meetelt, $\overline{O}$ is gemiddelde bezettingsgraad per groep (ook wel $\overline{O} = \frac{1}{K} \sum_{j=1}^{K} O_j(\pi)$) en $e_{O,j}$ een macht die aangeeft hoe belangrijk een groep is [2]. Hierbij kunnen wij $c_{O,j}$ en $e_{O,j}$ kiezen, afhankelijk van welke call type we belangrijk vinden. Hier zullen we waarschijnlijk $e_{O,J} = 1$ nemen, zodat het syteem lineair blijft.

Als we beide penaltys combineren krijgen we een minimalisatiefunctie, die er als volgt uit ziet:

$$min\ F_B(\pi) + F_O(\pi) \tag{67}$$

Ons doel is om deze functie te minimaliseren, dit kan door de politieken slim te kiezen. Om tot de juiste politiek te komen kan er gebruik worden gemaakt van 'politiek-iteratie', als eerste wordt een gekozen tactiek getest. Daarna wordt er gekeken of er een andere tactiek beter, dan wel slechter werkt. Deze 'iteratie'-stap wordt herhaalt net zolang totdat de optimale politiek is gevonden. De keuze die we kunnen maken in de politieken is de keuze welke 'routing-policy' we kiezen.

11.4 Matlab Code Threshold

```
function [blok,var]=MonteCarlo()
K=6; lambda=[30,40,28,20,30,40];  mu=[1,8.6,2.4,2.4,2.4,6];
rho=zeros(1,K);
for k=1:K
    rho(k)=lambda(k)/mu(k); %Rho bepalen
end;
C=102; Ci=[102,102,102,102,102,102]; %Capaciteit en threshold politiek
crit=1.96;                  %cricital value voor alpha=0.05
N=zeros(1,K);
for k=1:K
    N(k)=min(Ci(k),C);      %vult de Nk in, waarbij Nk max is van nk
end;
R=50000;             %Het aantal random toestanden die we zullen genereren.
%Stap 1: Bepalen van gammak, deze hebben we nodig om de kansfunctie voor nk
%te definiren.
disp('Stap 1')
z=sum(rho)/C;
gammak=zeros(1,K);
for k=1:K
    gammak(k)=(1+0.15*(1-z))*rho(k);
end;
%Stap 2: Nu we de kansfunctie pgamman hebben kunnen een aantal random
%toestanden gaan genereren. Dit gebeurd met behulp van de alias methode.
Yi=zeros(R,K); %hierin worden alle random toestanden opgeslagen
disp('Stap 2')
for k=1:K
    [sj,aj]=aliasmethode2(gammak(k),N(k)); %Maken van de alias tabel
    for r=1:R                   %Genereren van nk
        u=rand(1);
        j=floor(N(k)*u+1);
        if (sj(j)==0)
            Yi(r,k)=j;
        else
            S=(u*N(k)-floor(u*N(k)));
            if (S>=sj(j))
                Yi(r,k)=j;
            else
                Yi(r,k)=aj(j);
            end
        end
    end;
end;
%Stap3: Met behulp van deze random kunnen we phi en phii bepalen voor 1 ...
    tot R
phi=zeros(1,R);
disp('Stap 3')
for r=1:R %Bepalen van Phi_r
    phi(r)=gyi2(K,rho, ...
        Yi(r,:))*stepfun11(Yi(r,:),C,Ci)/probdens2(K,N,gammak,Yi(r,:));
end
phii=0;
for r=1:R %Bepalen van de schatter van G
    phii=phii+phi(r)/R;
end
```

```
%Stap4: Blokkeringskans en variantie bepalen voor elke soort call type
disp('Stap 4')
blok=zeros(1,K);
var=zeros(1,K);
for k=1:K
    a1=0;a2=0;a3=0;
    Gamma=0;phii2=0;
    for r=1:R
        phii2=phii2+phi(r)*stepfun12(Yi(r,:),C,k,Ci(k))/R;
    end; %Bepalen van schatter voo Phi^(1)
    for r=1:R
        a1=a1+(phi(r)-phii)^2;
        a2=a2+(phi(r)-phii)*(phi(r)*stepfun12(Yi(r,:),C,k,Ci(k))-phii2);
        a3=a3+(phi(r)*stepfun12(Yi(r,:),C,k,Ci(k))-phii2)^2;
        Gamma=Gamma+phi(r)*stepfun12(Yi(r,:),C,k,Ci(k));
    end
    var1=a1/(R-1);      %Bepalen van de verschillende variantie
    var12=a2/(R-1);      % die we nodig hebben voor het bepalen van s_R
    var2=a3/(R-1);
    ph=0;
    for r=1:R
        ph=ph+phi(r);
    end
    Gamma=Gamma/ph;     %Bepalen van schatter
    blok(k)=1-Gamma;    %Blokkeringskans bepalen
    si=var2-2*Gamma*var12+Gamma^2*var1;
    var(k)=crit*sqrt(abs(si))/(phii*sqrt(R)); %Variantie voor blokkeringskans
end
```

```
function antwoord=probdens2(K,N,gammak,Yi) %geeft de waarde van de ...
    kansfunctie p_{gamma} (n) voor toestand Yi
c=0;cc=0;
for k=1:K
    for l=0:N(k)
        tel=0;
        for i=1:l
            if i==1
                tel=gammak(k);
            else
                tel=tel*gammak(k)/i;
            end
        end
        c=c+tel;
    end
    if k==1
        cc=c;
        tel=0;
        for i=1:Yi(k)
            if i==1
                tel=gammak(k);
            else
                tel=gammak(k)/i;
            end
        end
        antwoord=tel;
    else
```

```
27          cc=cc*c;
28          tel=0;
29          for i=1:Yi(k)
30              if i==1
31                  tel=gammak(k);
32              else
33                  tel=gammak(k)/i;
34              end
35          end
36          antwoord=antwoord*tel;
37      end
38  end
39  antwoord=antwoord/cc;
```

```
1  function fi=stepfun11(Yi,C,Ci) %Bepaald of toestand Yi een mogelijke ...
       toestand is
2  fi=0;
3  if sum(Yi)<=C
4      if (Yi<=Ci)
5          fi=1;
6      end
7  end;
```

```
1  function fi=stepfun12(Yi,C,k,Ci) %Bepaald of in toestand Yi een klant van ...
       type k wordt toegelaten
2  fi=0;
3  if sum(Yi)+1<=C
4      if Yi(k)+1<=Ci
5          fi=1;
6      end
7  end
```

```
1  function [sj,aj]=aliasmethode2(gammak,Nk)
2  %Aliasmethode om een getal te genereren tussen 0 en Nk, afhankelijk van een
3  %.
4  n=Nk;           %aantal rijen van de alias tabel
5  qi=zeros(1,n);
6  sj=zeros(1,n);
7  zz=0;
8  for l=0:Nk
9      zz=zz+(gammak^l)/factorial(l);
10 end;
11 G=[];H=[]; %Initialiseren
12 for i=1:n
13     qi(i)=n*(gammak^i)/(factorial(i)*zz);
14     if (qi(i)>=1) %invullen van H en G
15         G=[G i];
16     else
17         H=[H i];
18     end;
19     sj(i)=0;
20 end;
```

```
21 aj=zeros(1,n);
22 while (numel(H)~=0) %Invullen van de alias tabel
23     j=H(1);
24     if (numel(G)~=0)
25         k=G(1);
26         aj(j)=k;
27         sj(j)=1-qi(j);
28         qi(k)=qi(k)+qi(j)-1;
29         if (qi(k)<1)
30             G=G(G~=k);
31             H=[H k];
32         end
33     end
34     H=H(H~=j);
35 end
```