Een klein lek doet een groot schip zinken

Wiskundig model voor het optimaliseren van de onderhoudsstrategie

Afstudeerscriptie Toegepaste Wiskunde

29 mei 2020, Gorinchem
De Haagse Hogeschool
Dorinda van den Dool, 16041577

Begeleiders: Nico van den Heuvel
Martijn de Munnik
Janine van Krimpen

DE HAAGSE HOGESCHOOL

Een klein lek doet een groot schip zinken

Wiskundig model voor het optimaliseren van de onderhoudsstrategie

Auteur: Dorinda van den Dool
Studentnummer: 16041577
E-mailadres: 16041577@student.hhs.nl

Opleiding: Toegepaste Wiskunde
Organisatie: De Haagse Hogeschool
Faculteit: Technologie, Innovatie en Samenleving
Begeleider: Janine van Krimpen
Stagebedrijf: Damen Shipyards Group
Eerste stagebegeleider: Nico van den Heuvel
Tweede stagebegeleider: Martijn de Munnik

Bachelorscriptie
Inleverdatum: 29 mei 2020
Stageperiode: 3 februari - 29 mei 2020
Stageplaats: Gorinchem

Voorwoord

Voor u ligt de afstudeerscriptie van mijn afstudeeropdracht ter afronding van de studie Toegepaste Wiskunde aan De Haagse Hogeschool in Delft. Het afstuderen was een leerzaam proces, waarbij ik mijn opgedane kennis en ervaring zelfstandig heb kunnen toepassen bij Damen Shipyards Group in Gorinchem. Daarnaast heb ik veel nieuwe dingen geleerd binnen de scheepsvaartwereld waar ik voorheen onbekend mee was.
Mijn afstudeeropdracht was het maken van een geautomatiseerd model die de optimale onderhoudsstrategie bepaalt voor de componenten van een schip die bij Damen geproduceerd wordt. Lezers die meer willen weten over onderhoudsstrategieën en hoe Damen Shipyards Group tot op heden de onderhoudsstrategie bepaald kunnen informatie hierover vinden in hoofdstuk 2. Hoe de betrouwbaarheid van een systeem berekend wordt staat beschreven in hoofdstuk 3. In hoofdstuk 4 staat beschreven wat het voorbereidend werk is voor de optimalisatie, namelijk het schip schematisch weergeven, de formule voor de betrouwbaarheid en de formule voor de onderhoudskosten van het schip bepalen en het opstellen van een zoekboom om de optimale onderhoudsstrategie per component te bepalen. In hoofdstuk 5 staan verschillende optimalisatiemethoden uitgelegd. De geschikte optimalisatiemethoden worden geïmplementeerd in hoofdstuk 6. Lezers die geïnteresseerd zijn in de resultaten van de optimalisatie worden doorverwezen naar hoofdstuk 7. De conclusie en de aanbevelingen staan in de hoofdstukken 8 en 9.
Allereerst wil ik mijn begeleiders Nico van den Heuvel en Martijn de Munnik bedanken voor hun fijne begeleiding en de goede feedback de zij mij gaven over mijn werk, maar ook over mijn eindverslag. Daarnaast wil ik mevrouw Van Krimpen bedanken voor haar waardevolle begeleiding en feedback bij het schrijven van mijn afstudeerscriptie. Ook wil ik mijn zus Caroline Verweij bedanken voor het met enthousiasme lezen van en het feedback geven op mijn eindverslag. Als laatste wil ik mijn broer Frederik van den Dool bedanken voor nakijken van mijn Engelse samenvatting.

Gorinchem, mei 2020
Dorinda van den Dool

Samenvatting (Nederlands)

Damen Shipyards Group (hierna genoemd Damen) is een internationale scheepswerfgroep. De schepen die gebouwd worden bestaan uit veel componenten. Een deel van die componenten heeft onderhoud nodig. Dat onderhoud is beschreven in een set onderhoudstaken en wordt veelal gedefinieerd door de leverancier van de betreffende component. Voor elke onderhoudstaak geldt een onderhoudsstrategie waarmee het onderhoud gepland wordt. De beste onderhoudsstrategie voor de onderhoudstaken van een component worden overgenomen van de leverancier van de component of handmatig bepaald met methoden die arbeidsintensief zijn en steeds opnieuw uitgevoerd moeten worden per schip. Het doel van de opdracht was om een geautomatiseerd model op te stellen die aan de hand van de wensen van de klant een goede onderhoudsstrategie kan bepalen voor de onderhoudstaken van de componenten. De klant geeft aan of hij een zo laag mogelijke faalkans wil bij een bepaald budget of zo min mogelijk onderhoudskosten bij een maximale toegestane faalkans.

Dit leidde tot de volgende hoofdvraag: *Wat is een passend model dat een goede onderhoudsstrategie bepaalt voor de componenten van een schip dat geproduceerd wordt bij Damen Shipyards Group?*

Bij het opstellen van het model is ervan uitgegaan dat elke component één onderhoudstaak heeft en daardoor ook één onderhoudsstrategie. Componenten geven variabele door aan andere componenten. Bijvoorbeeld de brandstofpomp die brandstof doorgeeft aan de hoofdmotor. Bij deze opdracht wordt ervan uitgegaan dat elke component geeft één variabele door aan een volgend component. Een component kan wel meerdere variabele ontvangen. Daarnaast is deze opdracht gericht op een kleiner deel van een schip.
Het model optimaliseert per KPI (Key Performance Indicator) de onderhoudsstrategie voor elke component van een systeem van een schip. Dit doet het model aan de hand van de formule voor de betrouwbaarheid en de formule voor de onderhoudskosten van een systeem. De belangrijkste parameter voor deze formules is de MTBF (Mean Time Between Failure) van elke component, wat afhangt van de onderhoudsstrategie. Deze formules worden bepaald door het systeem te vereenvoudigen tot één samengevoegd component aan de hand van drie basissystemen; seriesysteem, parallelsysteem en half serie-/parallelsysteem.
Vervolgens stelt het model een zoekboom op waarin alle mogelijke oplossingen staan. Een oplossing is voor iedere component een onderhoudsstrategie. Bijvoorbeeld voor alle componenten de onderhoudsstrategie run-to-failure. Met de opgestelde formules kan een optimalisatiemethode de faalkans en de onderhoudskosten van een oplossing berekenen en aan de hand daarvan de optimale of een goede oplossing bepalen. Er zijn twee optimalisatiemogelijkheden. De klant kan ervoor kiezen de faalkans te optimaliseren bij een bepaald budget of de onderhoudskosten te optimaliseren bij een maximale toegestane faalkans. Voor de eerste optimalisatiemogelijkheid zijn er twee methoden geïmplementeerd, namelijk Brute-force en het Greedy algoritme. Voor de tweede optimalisatiemogelijkheid zijn er drie methoden geïmplementeerd, namelijk Brute-force, het Greedy algoritme en de Breadth-first heuristiek. De optimalisatiemethode Brute-force geeft altijd de optimale oplossing. Echter deze methode heeft een te lange rekentijd en is daarom niet geschikt voor beide optimalisatiemogelijkheden. Het Greedy algoritme geeft niet het globale optimum, maar geeft wel een goede oplossing. Daarnaast heeft het Greedy algoritme een kortere rekentijd. De Breadth-first-heuristiek voor de tweede optimalisatiemogelijkheid geeft ook geen optimale oplossing, maar geeft wel een goede oplossing. Echter heeft de Breadth-first heuristiek een langere rekentijd dan het Greedy-algoritme. Daarom is voor beide optimalisatiemogelijkheden het Greedy algoritme het meest geschikt. Het Greedy algoritme geeft een goede oplossing en heeft geen lange rekentijd.

Samenvatting (Engels)

Damen Shipyards Group (hereinafter referred to as Damen) is an international shipyard group. The vessels Damen builds consist of many components. Some of those components require maintenance. This is described in a set of maintenance tasks and is often defined by the supplier of the component. Each maintenance task has a maintenance strategy by which maintenance is planned. The best maintenance strategy for the maintenance tasks of a component is manually determined by methods that are labor intensive and need to be repeated over and over foreach vessel. The purpose of the assignment was to develop an automated model that is able to determine the optimal maintenance strategy for the maintenance tasks of the components based on the requirements of the customer. The customer indicates whether he wants the lowest possible failure rate for a certain budget or as little costs as possible with a maximum permitted failure rate.

This leads to the following main question: *What is an appropriate model to determine a good maintenance strategy for the components of a ship that is produced at the Damen Shipyards Group*?

When developing the model, it is assumed that each component has one maintenance task and therefore one maintenance strategy. Components pass variables to other components. For example, the fuel pump that transfers fuel to the main engine. This assignment assumes that each component passes one variable to a subsequent component. A component can receive multiple variables. In addition, this assignment focuses on a smaller part of a ship.
The model optimizes per KPI (Key Performance Indicator) the maintenance strategy for each component of a system of a vessel. The model does this by determining the formula for the reliability and the formula for the maintenance costs of a system. The most important parameter in these formulas is the MTBF (Mean Time Between Failure) of each component which depends on the maintenance strategy of a component. These formulas are determined by simplifying the system into one virtual component based on three basic systems; series system, parallel system and half series / parallel system.
The model generates a search tree containing all possible solutions. A solution is for each component a maintenance strategy. For example, all components have the maintenance strategy run-to-failure. Based on the determined formulas, an optimization method is able to calculate the failure rate and maintenance costs of a solution and determine the optimal or a good solution based on this. There are two optimization options. The customer can choose to optimize the failure rate for a specific budget or to optimize the maintenance costs for a maximum permitted failure rate. For the first optimization option, two methods have been implemented, namely Brute-force and the Greedy algorithm. For the second optimization option, three methods have been implemented, namely Brute-force, the Greedy algorithm and the Breadth-first heuristic. The Brute-force optimization method always provides the optimal solution. However, the calculation time for running this method takes too much time and therefore is this method not suitable for both optimizations. The Greedy algorithm does not give the global optimum, but it does provide a good solution and has a shorter calculation time. The Breadth-first heuristic for the second optimization option does not provide an optimal solution either, but it does provide a good solution. Finally, the Breadth-first heuristic has a longer calculation time than the Greedy algorithm. The conclusion is that the Greedy algorithm is best suited for both optimization options. The Greedy algorithm provides a good solution and does not have a long calculation time.

Inhoud

1. Inleiding

Damen Shipyards Group (hierna genoemd Damen) is een internationale scheepswerfgroep en tegelijk een familiebedrijf. Damen werd in 1927 opgericht door de twee broers Jan en Rien Damen. Meer dan 40 jaar bleef het een klein maar prominent bedrijf. In 1969 kocht Kommer Damen het bedrijf van zijn vader Jan en introduceerde het concept van standaardisatie. Hierbij werden modulaire constructies toegepast voor het bouwen van kleine boten en sloepen. Het concept was meteen een succes en in 1973 breidde het bedrijf uit naar grotere vestingen in Gorinchem. Sinds de introductie van het standaardisatieconcept heeft Damen meer dan 6.000 schepen opgeleverd. Tegenwoordig is Damen actief in vele scheepsbouwsectoren en heeft het wereldwijd een prominente en vertrouwde reputatie verworven.
De schepen die Damen bouwt, zijn onder andere offshore boten, sleepboten, baggerboten, jachten en maritieme schepen. Damen doet meer dan alleen schepen bouwen, het heeft ook een internationaal netwerk van levenscyclus ondersteunende diensten, inclusief onderhoud en reparatie- en conversiefaciliteiten. Vandaag de dag is Damen actief in vijf continenten en heeft Damen wereldwijd ongeveer 12.000 werknemers in dienst (Damen Shipyards, z.d.-a).

De schepen die gebouwd worden bestaan uit veel componenten. Elke component bevat een set van onderhoudstaken. Bij de motor van een schip kan dit zijn het verversen van de olie of het filter vervangen. Voor elke taak geldt een onderhoudsstrategie waarmee het onderhoud gepland wordt. Een onderhoudsstrategie kan zijn dat er onderhoud uitgevoerd wordt na een bepaald gebruiksinterval, bijvoorbeeld na iedere 500 vaaruren of 100.000 liter brandstof, of na een tijdsinterval, bijvoorbeeld na iedere 3 maanden. Aan de verschillende onderhoudsstrategieën zijn kosten en risico's verbonden.
De huidige onderhoudsstrategieën voor de componenten van schepen worden overgenomen van de leveranciers van de componenten of handmatig bepaald met methoden die arbeidsintensief zijn en steeds opnieuw uitgevoerd moeten worden per schip. Het moet steeds opnieuw omdat de schepen verschillen van elkaar en de omstandigheden waarin de schepen varen en de eisen van de klant verschillen per schip. Daarom wil Damen dat er een geautomatiseerd model wordt opgesteld, zodat aan de hand van dit model bepaald kan worden wat de optimale onderhoudsstrategie is per component van een schip. De klant moet hierbij aan kunnen geven of de optimalisatie uitgevoerd moet worden aan de hand van een bepaald budget of aan de hand van een gewenste betrouwbaarheid. Als de klant aangeeft dat de optimalisatie uitgevoerd moet worden aan een bepaald budget, dan wordt de faalkans van het schip geoptimaliseerd waarbij de onderhoudskosten van alle componenten niet boven het budget uit mag komen. Als de klant aangeeft dat de optimalisatie uitgevoerd moet worden aan de hand van een gewenste betrouwbaarheid, dan worden de onderhoudskosten van alle componenten geoptimaliseerd waarbij de betrouwbaarheid minimaal zo hoog is als dat de klant wenst.
De opdracht is uitgevoerd voor de afdeling Simulation Development. Deze afdeling is een subafdeling van de afdeling Research en Development. Op de afdeling Simulation Development werken vier mensen, waarvan één in Oekraïne. Op de afdeling Research en Development werken zestig mensen.

De hoofdvraag die hierbij opgesteld is luidt: *Wat is een passend model dat een goede onderhoudsstrategie bepaalt voor de componenten van een schip dat geproduceerd wordt bij Damen Shipyards Group?*

Deze hoofdvraag wordt beantwoord aan de hand van de volgende deelvragen:

1. Hoe bepaalt Damen tot op heden de onderhoudsstrategie?
2. Hoe wordt de betrouwbaarheid van een systeem berekent?
3. Hoe worden de onderhoudskosten van een systeem berekent?
4. Welke optimalisatiemethoden zijn er toepasbaar op deze optimalisatie?
5. Hoe kan het optimalisatiemodel passend gemaakt worden voor dit probleem?

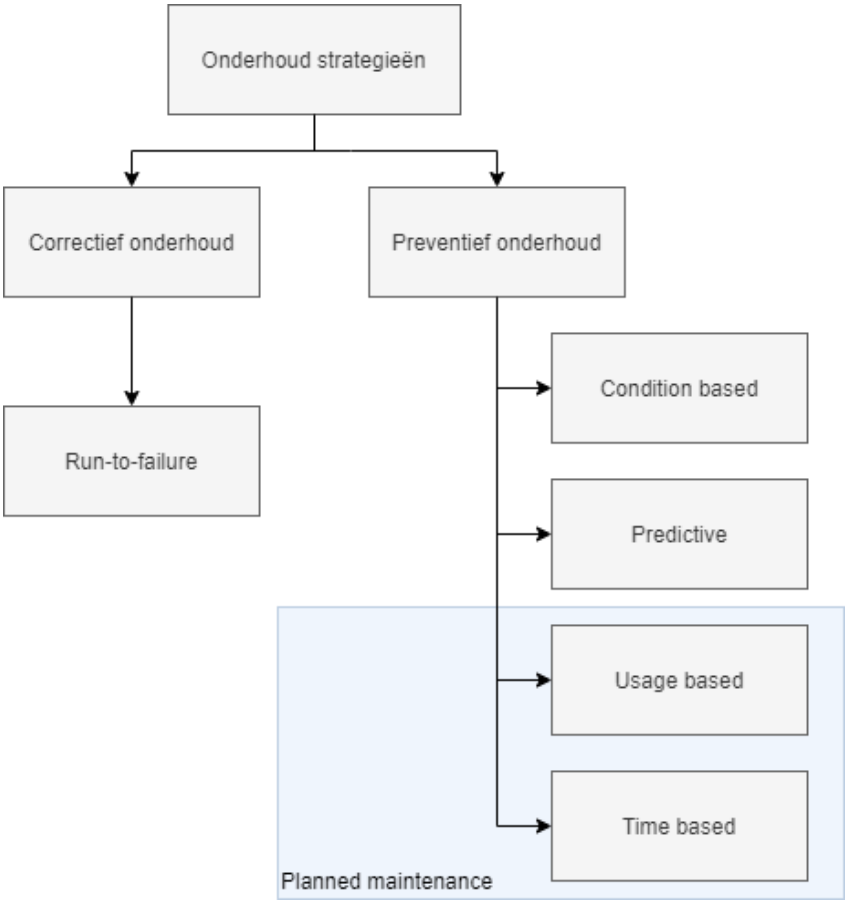
Als eerste wordt er literatuuronderzoek en deskresearch gedaan naar de verschillende onderhoudsstrategieën, de berekening van de betrouwbaarheid en de berekening van de onderhoudskosten van een systeem. Dit is het voorbereidende werk voor de optimalisatie en daarbij het eerste stukje van het model. Daarna wordt er literatuuronderzoek gedaan naar verschillende optimalisatiemethoden. Uiteindelijk zal de best toepasbare optimalisatiemethode geïmplementeerd worden, zodat er een wiskundig model ontstaat die de optimale onderhoudsstrategie van alle componenten kan bepalen. Het model wordt geprogrammeerd in Python met behulp van Neo4j.
Een component van een schip bevat meerdere onderhoudstaken. Om de opdracht duidelijker te maken wordt ervan uitgegaan dat elke component één onderhoudstaak heeft. Hierdoor krijgt een component één onderhoudsstrategie. Daarnaast geven componenten variabelen aan elkaar door. Bijvoorbeeld de brandstofpomp die brandstof doorgeeft aan een motor. Als één component stuk gaat, kan hierdoor een hele keten aan componenten niet functioneren. Als de brandstofpomp geen brandstof doorgeeft aan de motor. Kan de motor geen mechanische kracht leveren aan as. De as krijgt geen mechanische kracht van de motor, dus kan de as ook geen mechanische kracht geven aan de koppeling. Op zijn beurt kan de koppeling hierdoor geen mechanische kracht leveren aan de schroef. Doordat dus de brandstofpomp stuk gaat, functioneren de motor, de as, de koppeling en de schroef niet meer. In dit onderzoek wordt ervan uitgegaan dat elke component één variabele doorgeeft. Wel kan een component meerdere variabele ontvangen. Tevens bestaat een schip uit veel componenten. Daarom wordt er in deze opdracht gericht op een kleiner deel van een schip. Maar als het model werkt voor een kleiner deel van een schip, werkt het model ook voor het hele schip.

2. Huidige werkwijze van Damen

Om een model op te kunnen stellen dat de optimale onderhoudsstrategie bepaalt voor elk component is het eerst nodig om te weten op welke manier Damen tot op heden de optimale onderhoudsstrategie bepaalt. Daarvoor wordt in paragraaf 2.1 eerst uitgelegd wat de onderhoudsstrategieën zijn. Vervolgens wordt in paragraaf 2.2 besproken welke methoden Damen toepast om de onderhoudsstrategie van een component van een schip te bepalen.

2.1. De onderhoudsstrategieën

Een schip bestaat uit veel componenten die allemaal een set van onderhoudstaken hebben. Elke onderhoudstaak heeft een eigen onderhoudsstrategie. In het onderstaande schema (**Fout! V erwijzingsbron niet gevonden.**) staan de onderhoudsstrategieën die door Damen toegepast worden of die Damen in de toekomst toe wil gaan passen.
Onderhoudsstrategieën kunnen verdeeld worden over twee groepen: correctief en preventief onderhoud. Bij correctief onderhoud horen de onderhoudstaken die worden uitgevoerd om defecte componenten te repareren of te vervangen (UpKeep Technologies Inc., z.d.). Als er onderhoud uitgevoerd wordt voordat de storing is opgetreden wordt dit preventief onderhoud genoemd. Met preventief onderhoud wordt de kans op storingen geminimaliseerd (Road to reliability, 2020). De onderhoudsstrategie run-to-failure is bij correctief onderhoud. De onderhoudsstrategieën usage-based maintenance, time-based maintenance, condition-based maintenance en predictive maintenance zijn preventief onderhoud.



Figuur 1 Schema onderhoudsstrategieën

De correctieve onderhoudsstrategie run-to-failure is de eenvoudigste onderhoudsstrategie. In deze strategie mogen componenten werken totdat ze defect zijn, waarna reactief onderhoud wordt uitgevoerd. De enige trigger voor deze onderhoudsstrategie is dus het defect raken of een storing van de component. Er wordt geen onderhoud uitgevoerd voordat de componenten defect zijn. Er is echter soms een plan voor als een component defect raakt, zodat de component snel kan gerepareerd of vervangen worden. Het is bij deze strategie belangrijk om reserveonderdelen en personen beschikbaar te hebben om de defecte component te vervangen en het schip varend te houden. Een veelgebruikt voorbeeld van run-to-failure is de onderhoudsstrategie van de gloeilamp. De lamp blijft branden tot hij defect raakt. Op dat moment wordt er een nieuwe gloeilamp uit de voorraad gehaald en op een geschikt tijdstip vervangen. Voordelen van deze onderhoudsstrategie is dat het weinig planningsvereisten heeft, daarnaast is het eenvoudig te begrijpen. De nadelen zijn dat deze onderhoudsstrategie onvoorspelbaar, inconsistent is en kostbaar kan zijn. Daarnaast zijn er voorraadkosten, omdat de reserveonderdelen op voorraad gehouden worden (Fiix, z.d.-b).

Zoals genoemd zijn vier onderhoudsstrategieën preventief onderhoud, namelijk usage-based maintenance, time-based maintenance, condition-based maintenance en predictive maintenance. Bij usage-based maintenance is de trigger voor het uitvoeren van onderhoud een bepaald aantal kilometers of vaaruren (Bruce, 2018). Time-based maintenance wordt op kalender basis uitgevoerd. Een voorbeeld hiervan is dat voor elke zomer volgens een onderhoudsplan onderhoud wordt uitgevoerd aan de airco (Fiix, z.d.-c). Bij Damen worden de onderhoudsstrategieën usage-based maintenance en time-based maintenance als één onderhoudsstrategie beschouwd die planned maintenance heet. Het voordeel van planned maintenance is dat onderhoud van tevoren gepland zijn in een agenda of iets dergelijks. Hierdoor kan deze strategie gebruikt worden naast predictive maintenance (Fiix, z.d.-c).

In het geval van condition-based maintenance wordt de conditie van de componenten bewaakt om te beslissen welk onderhoud er moet worden uitgevoerd. Conditiegegevens worden continu of volgens bepaalde intervallen verzameld. Onderhoud wordt alleen uitgevoerd nadat een verslechtering van de toestand van de component is geconstateerd. Hierin verschilt deze onderhoudsstrategie van de usage-based maintenance, waar onderhoud wordt uitgevoerd op basis van vooraf gedefinieerde geplande intervallen. Voordelen van condition-based maintenance zijn onder andere dat het onderhoud wordt uitgevoerd terwijl het schip actief werkt, omdat tijdens het varen van het schip de conditie van de componenten van het schip bewaakt wordt. Daarnaast minimaliseert deze onderhoudsstrategie ongeplande downtime. Downtime is de tijd dat het schip niet kan varen doordat het schip defect is of doordat er onderhoud uitgevoerd moet worden. De nadelen zijn onder andere dat het installeren van testapparatuur voor conditiemonitoring duur is, het analyseren van databases geld kost en vermoeidheid en slijtage moeilijk zijn te detecteren met deze strategie (Fiix, z.d.-a).

Predictive maintenance is de onderhoudsstrategie waarbij op basis van huidige en historische data voorspelt wordt wanneer een component defect raakt. Hierbij wordt data verzameld door sensoren. Op de data wordt machine learning en deep learning toegepast om modellen op te stellen die het onderhoud kunnen voorspellen (Worksuite, z.d.).

2.2. FMECA en RCM

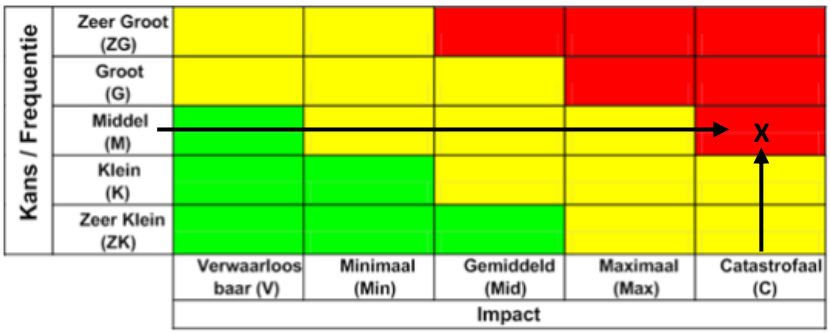
Iedere component van een schip heeft een set van onderhoudstaken. Voor elke onderhoudstaak geldt een onderhoudsstrategie waarmee het onderhoud gepland wordt. De huidige werkwijze van Damen is dat de onderhoudsstrategie die de leverancier vaststelt gebruikt wordt. Soms voert Damen RCM (met FMECA) uit om een goede onderhoudsstrategie te bepalen. Daarom zal in paragraaf 0 uitgelegd worden wat FMECA is. In paragraaf **Fout! Verwijzingsbron niet gevonden.** wordt uitgelegd wat RCM i s en in paragraaf 0 wordt beschreven hoe RCM (en FMECA) gebruikt wordt binnen Damen.

2.2.1. Wat FMECA is

FMECA staat voor Failure Modes Effect Criticality Analyses. Dit betekent dat er onderzocht (Analysis) wordt op welke wijze een component van een schip kan stuk gaan (Failure Mode), welk effect (Effect) dit heeft (Spie, z.d.) en hoe kritiek dat effect is voor het schip (Criticality). Een voorbeeld van een failure van een fietsband is dat de band lek is. De failure mode hierbij is dat er geen lucht meer in de band zit. Het effect is dat er niet meer gereden kan worden op de fiets. Vervolgens wordt bepaald hoe nadelig het effect is dat optreedt bij het falen (Criticality). Hierbij worden de volgende stappen ondernomen (CMS Asset Management, z.d.-b):

1. Wat zijn de functies en de bijbehorende prestatienormen van de componenten van een schip in zijn huidige gebruiksprofiel?
2. Op welke manieren kunnen de componenten van een schip falen?
3. Wat is de oorzaak van het falen?
4. Wat gebeurt er wanneer een storing plaatsvindt?
5. In welk opzicht is een storing van belang?

Bij FMECA worden risicomatrices opgesteld die afgestemd zijn op bedrijfsdoelstellingen. In Figuur 2 is een voorbeeld van een risicomatrix te zien (Ingenieursbureau Westenberg B.V., 2016).



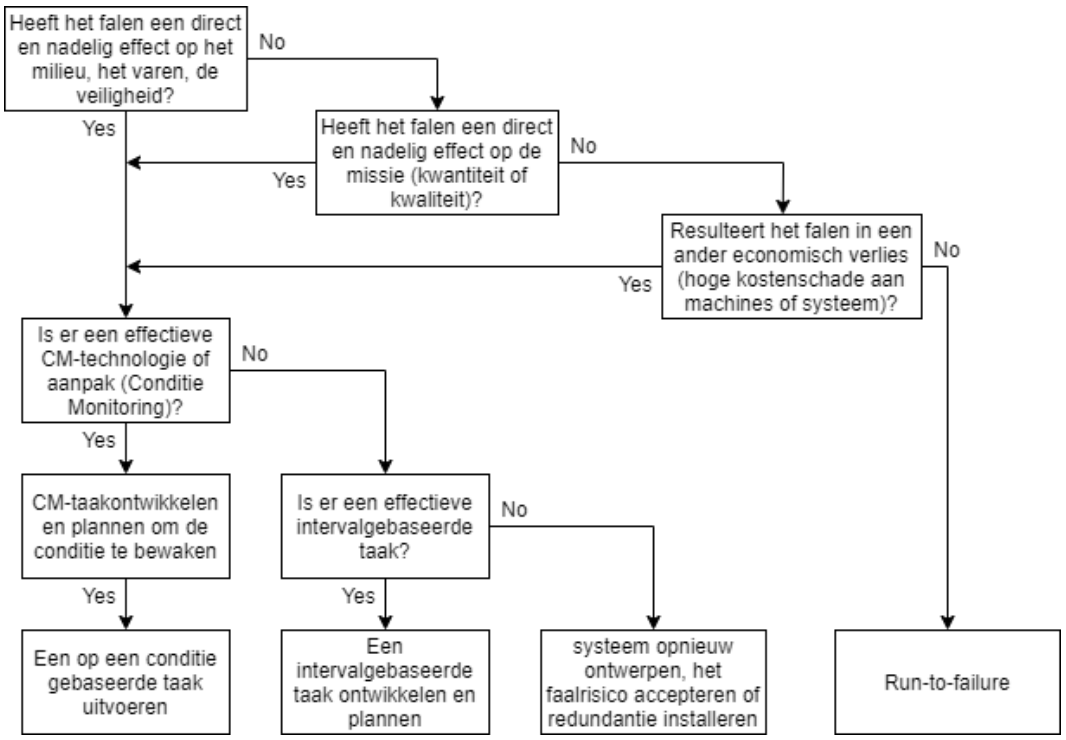
Figuur 2 Voorbeeld risicomatrix

Met de risicomatrix wordt de risico van de component bepaald. De risico hierbij is de kans van het falen maal de impact. Hierbij wordt de kans van falen verticaal weergegeven en de impact horizontaal. Een ingevulde risicomatrix is het vertrekpunt om de onderhoudsstrategie te bepalen (CMS Asset Management, z.d.-c).

Stel, een pomp heeft als failure mode lager schade. Het effect is dan dat de pomp niet op toeren kan komen en het is zo kritiek dat de pomp dan geen brand kan blussen. De kans dat er lager schade optreedt is 'middel' (verticaal) en de impact is 'catastrofaal' (horizontaal). Er kan namelijk geen brand geblust worden wat grote gevolgen kan hebben. Op de plaats waar de pijlen elkaar kruisen wordt dan een kruisje ingevuld in de cel. In dit geval bevindt het kruisje zich in een rode cel, de risico van de pomp is dus groot. Dit is het vertrekpunt voor het bepalen van de onderhoudsstrategie van de pomp. Had het kruisje zich in een groene cel bevonden, dan had de pomp weinig risico gehad en was dat het vertrekpunt geweest voor het bepalen van de onderhoudsstrategie.

2.2.2. Wat RCM is

RCM staat voor Reliability Centered Maintenance en maakt gebruik van FMECA. RCM is een vervolg op FMECA waarbij een beslisdiagram gebruikt wordt die afgestemd is op bedrijfsdoelstellingen. In is een voorbeeld te zien van een beslisdiagram (Pride, 2016). In de bovenste drie blokken worden de effecten onderzocht van het falen van een component, aan de hand van de antwoorden op de vragen in de drie blokken kan uitgezocht worden welk onderhoudsstrategie het best past bij een component. De bovenste drie blokken wordt omschreven als FMECA en de overige blokken als RCM.



Figuur 3 Voorbeeld beslisdiagram

Het kan zijn dat wanneer een component in het ene schip heel belangrijk is, bijvoorbeeld de pomp van het brandblussysteem. Deze pomp in een ander schip op een andere plaats gebruikt wordt waardoor het nadelig effect veel kleiner kan zijn als de pomp stuk gaat. Daarom moet voor ieder schip telkens opnieuw RCM (vooral het gedeelte van FMECA) uitgevoerd worden. Hierbij is vooral FMECA is arbeidsintensief. Daarom wordt er een model opgesteld die de vragen in de eerste drie blokken automatisch beantwoord, waardoor er snel een onderhoudsstrategie per component gekozen kan worden.
In het algemeen wordt RCM ingezet in de aanloopfase van een nieuwe installatie of om hardnekkig structureel falen te analyseren na een langdurige productieperiode. Bij het uitvoeren van RCM-analyse worden de vijf stappen van een FMECA-analyse doorlopen met daarna de volgende twee stappen (CMS Asset Management, z.d.-a):

6. Wat kan worden gedaan om een storing te voorspellen of te voorkomen?
7. Wat dient te worden gedaan wanneer geen geschikte proactieve taak kan worden gevonden?

2.2.3. Huidig gebruik RCM (en FMECA) door Damen

De componenten van de schepen die bij Damen geproduceerd worden, komen bij leveranciers vandaan. De leveranciers hebben voor de component al een onderhoudsstrategie vastgesteld, wat Damen overneemt als onderhoudsstrategie. In de meeste gevallen is dit usage-based maintenance. Soms is de vastgestelde onderhoudsstrategie inspection (condition) based maintenance, maar voor de

leverancier is deze onderhoudsstrategie lastig voor het aanbieden van garantie. Wat de leveranciers dan soms doen, is een contract afsluiten dat de leverancier zelf de inspectie doet.
RCM wordt uitgevoerd op verzoek van de klant. Tot nu toe is dat vaak de Marine of organisaties waar uptime heel belangrijk is. Uptime is het tegenovergestelde van downtime, dus de tijd dat het schip kan varen. Het wordt vaak alleen uitgevoerd voor de Marine of organisaties waarbij de uptime belangrijk is, omdat RCM te duur is om uit te voeren. Er is nog nooit een volledige RCM (en daarbij FMECA) uitgevoerd voor een heel schip, omdat RCM vaak te duur is. Wel wordt er onderzocht welk subsysteem van een schip zorgt voor een lagere betrouwbaarheid en daarom belangrijk is voor de betrouwbaarheid van het hele schip. Op dit subsysteem wordt dan wel RCM uitgevoerd, dit kan bijvoorbeeld het voortstuwingsysteem zijn.
Het eerste stuk van RCM is FMECA. Bij FMECA worden eerst de functies omschreven van de componenten van een schip. Daarna de failure modes en wat de oorzaak kan zijn van de failure mode. Hierbij ook het lokale effect van de failure mode in het systeem. Bijvoorbeeld als de ventilator defect is, dan is de failure mode dat er geen druk meer is. De oorzaak kan zijn dat er geen kracht is waarmee de ventilator aangedreven wordt. Het lokale effect is dan dat de ventilator niet werkt en het algemene effect is dat de machinekamer oververhit raakt, omdat er dan geen ventilatie is die ruimte is. Uiteindelijk wordt het RPN (Risk Priority Number) uitgerekend aan de hand van de volgende drie criteria:

1. Hoe nadelig is het dat deze failure mode optreedt?
2. Hoe vaak treedt deze failure mode op?
3. Hoe detecteerbaar is deze failure mode?

Aan al deze drie criteria wordt een waarde gehangen op een schaal van 1 tot 4. Hoe nadeliger het is dat de failure mode optreedt, hoe hogere waarde hieraan gehangen wordt. Datzelfde geld voor hoe vaker de failure mode optreedt en hoe minder detecteerbaar de failure mode is. Deze drie waarden worden met elkaar vermenigvuldigd en vormen het RPN. RPN staat voor Risk Priority Number. Hoe hoger het RPN, hoe hoger de prioriteit deze failure mode krijgt om het systeem te verbeteren (Patagonia, z.d.).
Met het tweede en laatste stuk van RCM wordt een onderhoudsconcept gemaakt. Hierbij wordt gebruikt gemaakt van beslissingsstrategieën en FMECA, zoals beschreven in de vorige alinea, om te bepalen welke onderhoudsstrategie nodig is voor een component. Uiteindelijk wordt er een rapport opgesteld met daarin onder andere FMECA, kosten en het uiteindelijke onderhoudsconcept met daarbij de concepttaken. Een voorbeeld van zo'n onderhoudsconcept staat in Figuur 4 (Damen Shipyards, 2019)

Object	FM	Storingsvorm	Taak	Taaktype	Inter-val	Een-heid	Rol	Kennisgebied	Kolom
	1a1	No Electrical supply	Repair powersupply	Corrective Action	n/a				
	1a2	Valves closed, due to human error	Create operational procedures for this system, incl pre-start system checks	Desirable Redesign	n/a				
	1a3	Pump Failed due to normal wear	Make notes of the actual flow and record it in a log. Compare notes for trending. when flow is decreased by 10%, schedule a task for repairing/replacing the pump	Inspection Task	6	wk	Crew		
	1a4	Operating error (mode selector switch in OFF position)	Create operational procedures for this system, incl pre-start system checks	Desirable Redesign	n/a				
	1a5	Pump not running due to failed electric motor	Swap pumpset with spare one Determine if the failed pumpset is worth restoring, order spare parts or a new pumpset accordingly.	Corrective Action	n/a				
	1b1	AT least one of the two valves is (partially) closed	Create operational procedures for this system, incl pre-start system checks	Desirable Redesign	n/a				
	1b2	Expansion tank breather failed	Replace expansion tank breather cap	Discard Task	5	yr	Crew		
	1b3	Coolingwater level low, due to leakage	Examine all couplings and flanges for signs of leaks	Inspection Task	1	yr	Crew		
	1b4	Pump impeller failed, due to normal wear	with the system at operating temperature make notes of the actual flow. Compare notes for trending analysis, when consistent flow decrease is found, an impeller replacement job has to be scheduled	Inspection Task	6	wk	Crew		
	1b5	Clogged strainer	Clean the external side of the	Inspection Task	12	wk	Crew		

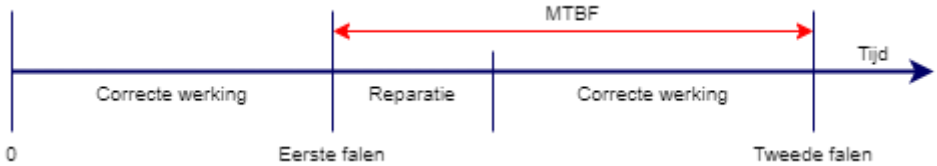
Figuur 4 Voorbeeld onderhoudsconcept

3. Betrouwbaarheid van een systeem

Het eerste stuk van RCM, namelijk FMECA, is arbeidsintensief en moeten steeds opnieuw uitgevoerd worden voor elk schip, omdat elk schip anders is. Een component kan een belangrijke functie hebben in het ene schip, maar ditzelfde component kan in een ander schip weer geen belangrijke functie hebben. Daarom wordt er een geautomatiseerd model opgesteld die de optimale onderhoudsstrategie bepaald voor de componenten van een schip. Een klant bij Damen geeft aan of het onderhoud moet voldoen aan een bepaald budget of aan een maximale toegestane faalkans. Deze faalkans is de faalkans dat het schip stuk gaat op taak niveau. Bijvoorbeeld de kans dat het schip niet meer kan varen, of de kans dat een schip niet meer brand kan blussen. Voor iedere taak geeft de klant een toegestane faalkans aan. Als de klant een bepaald budget aangeeft is het onderhoud optimaal als de faalkans met dat budget het meest gereduceerd wordt. Als het onderhoud niet optimaal is wordt de faalkans minder gereduceerd voor het door de klant aangegeven budget. Hierdoor is de faalkans hoger dan wat mogelijk is voor het budget. Dit betekend dat de kans groter is dat een component van het schip stuk gaat. Wanneer de klant aangeeft dat het onderhoud moet voldoen aan een maximale toegestane faalkans is het onderhoud optimaal als de onderhoudskosten het laagst zijn, waarbij de faalkans niet hoger is dan dat is toegestaan. Als het onderhoud niet optimaal is betekend dit dat het onderhoud niet de laagste onderhoudskosten heeft en dus het onderhoud voor de klant duurder is dan wanneer het onderhoud optimaal is.
Een belangrijke variabele die bij het optimaliseren een rol speelt is de faalkans. In dit hoofdstuk wordt in paragraaf 3.1 uitgelegd hoe de faalkans van een component berekend wordt. In paragraaf 3.2 wordt de berekening van de betrouwbaarheid van een systeem aan de hand van de faalkansen van de componenten uitgewerkt voor verschillende basissystemen. De betrouwbaarheid van een systeem is de kans dat het systeem werkend is.

3.1. MTBF

Bij Damen wordt er gerekend met de MTBF van componenten. MTBF staat voor Mean Time Between Failure, dat is gemiddelde tijd tussen falen. Bijvoorbeeld dat een component ongeveer om de vier weken stuk gaat. Dan is de MTBF van dit component vier weken. In figuur 5 staat schematisch weergegeven wat de MTBF is (Wikipedia, 2020). De reparatie wordt meegerekend bij de tijd tussen falen. Dit is niet erg, omdat de reparatietijd vaak in uren is en verwaarloosbaar is ten opzichte van de tijd van correcte werking. De tijd van correcte werking is in dagen en meestal boven de honderd dagen.



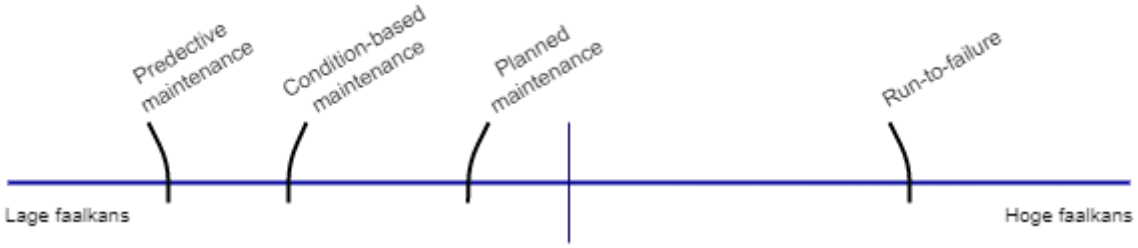
Figuur 5 Schematische weergave MTBF

De MTBF wordt geschat door Damen of is opgegeven door de fabrikant. De MTBF is eigenlijk een gemiddelde van een verdeling. Een component gaat niet altijd precies om de vier weken stuk. Het kan zijn dat de ene keer de component na drie weken stuk gaat en een andere keer na vijf weken. De MTBF is dus het gemiddelde van een verdeling en kan dus afwijken van de tijd tussen falen in werkelijkheid. In dit onderzoek wordt ervan uitgegaan dat de MTBF één waarde is. Er wordt geen rekening mee gehouden dat de MTBF een gemiddelde van een verdeling is.

Om de betrouwbaarheid van een systeem te berekenen is de faalkans nodig. De MTBF is de inverse van de faalkans (AutomationDirect, z.d.), daarom wordt de faalkans berekend aan de hand van de volgende formule:

$$Faalkans = \frac{1}{MTBF} \quad (1)$$

Als de faalkans berekend wordt met de MTBF houdt dit in dat de faalkans over de tijd gelijk is. Er zit namelijk geen afhankelijkheid van de tijd in deze formule. De MTBF is de gemiddelde tijd tussen falen. Deze gemiddelde blijft over de tijd hetzelfde. Daarom is de faalkans, als hij berekend wordt met deze formule, ook over de tijd gelijk.
De MTBF is opgegeven wanneer er geen onderhoud uitgevoerd wordt, wanneer de onderhoudsstrategie run-to-failure is toegepast. Om de MTBF voor te blijven wordt er onderhoud uitgevoerd. Omdat de MTBF een verdeling is wordt de MTBF niet gereduceerd tot 0, maar wel tot een lagere waarde en dus een lagere faalkans. De verschillende onderhoudsstrategieën hebben verschillende reducties.
Onderstaande grafiek (figuur 6) geeft aan hoe de faalkansen zich per onderhoudsstrategie verhouden (Fiix, 2020). Hierin is te zien dat de strategie run-to-failure de hoogste faalkans heeft en de strategie predictive maintenance de laagste faalkans.



Figuur 6 Verhouding faalkans per onderhoudsstrategie

3.2. Betrouwbaarheid van de basissystemen

Voordat de berekening van de betrouwbaarheid van het systeem uitgelegd wordt, is het belangrijk om de volgende variabelen te definiëren:

- R_i = betrouwbaarheid (reliability) van component i;
- F_i = kans van falen (failure rate) van component i;
- R_s = reliability van het systeem;
- F_s = failure rate van het systeem.

Als R_i bekend is en F_i onbekend, dan kan aan de hand van de complementregel F_i berekend worden, omdat F_i het tegenovergestelde van R_i is. De complement regel is (Hofstede, z.d.):

$$P(A) = 1 - P(niet\ A) \quad (2)$$

A is bijvoorbeeld de situatie dat een component werkt. De kans dat een component niet werkt is dan één min de kans dat de component wel werkt.
De berekening van F_i aan de hand van de gedefinieerde variabelen is dan:

$$F_i = 1 - R_i$$

Hetzelfde geldt als F_i bekend is en R_i onbekend en voor R_s en F_s, als één van beide onbekend is.

Door middel van de faalkansen van alle componenten kan de betrouwbaarheid van het hele systeem berekend worden. Er zijn drie basissystemen waarmee complexe systemen vereenvoudigd kunnen worden tot één component. Deze drie basissystemen zijn (University of Georgia, z.d.):

- Seriesysteem
- Parallelsysteem
- Half serie-/parallelsysteem

In paragraaf 3.2.1 wordt uitgelegd hoe de betrouwbaarheid van een seriesysteem berekend wordt, in paragraaf 3.2.2 wordt uitgelegd hoe de faalkans van een parallelsysteem berekend wordt en in paragraaf 3.3.3 wordt uitgelegd hoe de betrouwbaarheid van een half serie-/parallelsysteem berekend wordt.

3.2.1. Seriesysteem

In figuur 7 staat weergegeven hoe componenten gekoppeld zijn in een seriesysteem. De blokken met daarin de nummers zijn de componenten. De blokken met de letters zijn de variabelen die de componenten leveren aan het volgend component of nodig hebben van de vorige component. Bijvoorbeeld de motor levert mechanische kracht een as. De as heeft dus mechanische kracht nodig van de motor. Doordat de as mechanische kracht krijgt van de motor, kan de as weer mechanische kracht geven aan de volgende component.

Figuur 7 Seriesysteem

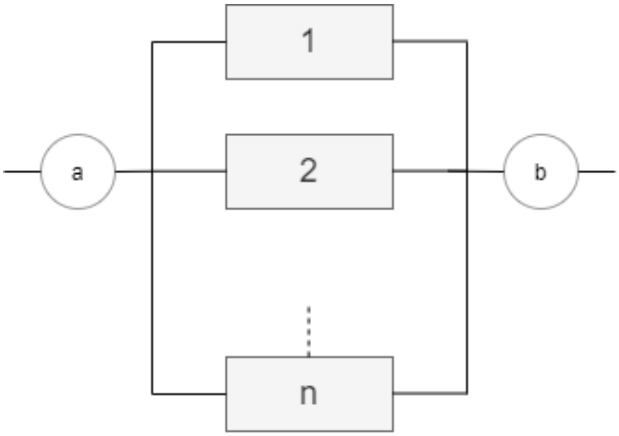
Componenten in een seriesysteem worden onafhankelijk van elkaar beschouwd, omdat hiervoor geldt dat het systeem werkt als alle componenten werken. Faalt één component, dan is het systeem al niet werkend meer. Om de betrouwbaarheid van een seriesysteem te berekenen wordt de volgende formule gebruikt (University of Georgia, z.d.):

$$R_s = \prod_{i=1}^{n} R_i \tag{3}$$

Waarbij R_i de betrouwbaarheid van component *i* en R_s is de betrouwbaarheid van het systeem. Het maakt bij deze formule niet uit of R_i voor elke component verschillend is, omdat alle R_i's meegenomen worden in de formule. Hetzelfde geldt voor de hoeveelheid van de variabele die de componenten leveren, deze mogen ook verschillend zijn voor iedere component.
Levert één component niet genoeg variabele aan de volgende component, dan zou het seriesysteem en daarbij het volledige systeem niet meer functioneren. Wanneer een motor niet genoeg vermogen heeft en dus niet genoeg mechanische kracht levert aan de as, kan de as niet genoeg mechanische kracht leveren aan de koppeling en kan de koppeling niet genoeg mechanische kracht leveren aan de schroef. Op deze wijze faalt het hele systeem doordat de motor niet genoeg vermogen heeft.

3.2.2. Parallelsysteem

In figuur 8 staat weergegeven hoe componenten gekoppeld zijn in een parallelsysteem. De blokken met daarin de nummers zijn de componenten. De blokken met de letters zijn de variabelen die de componenten leveren aan het volgend component of nodig hebben van de vorige component. Bij een parallelsysteem leveren alle componenten die parallel zijn dezelfde variabele(n) en hebben dezelfde variabele(n) nodig.



Figuur 8 Parallelsysteem

Waar het bij een seriesysteem niet uitmaakt of alle R_i's en de hoeveelheid van een variabele die geleverd wordt hetzelfde is, maakt dit voor een parallelsysteem wel uit. Dit wordt uitgelegd aan de hand van de volgende vier situaties:

1. Alle componenten hebben dezelfde R_i en alle componenten leveren dezelfde hoeveelheid variabele;
2. Niet alle componenten hebben dezelfde R_i, maar alle componenten leveren wel dezelfde hoeveelheid variabele;
3. Alle componenten hebben dezelfde R_i, maar niet alle componenten leveren dezelfde hoeveelheid variabele;
4. Niet alle componenten hebben dezelfde R_i en niet alle componenten leveren dezelfde hoeveelheid variabele.

Daarnaast heeft een parallelsysteem niet altijd alle componenten nodig om het volledig systeem te laten functioneren. Als twee service pompen elk 34 liter water per minuut leveren en drie brandslangen 30 liter water per minuut nodig hebben om een brand te kunnen blussen. Dan is minstens één servicepomp nodig om een brand te kunnen blussen, omdat één service pomp meer dan 30 liter water per minuut kan leveren. Het kan dus zo zijn dat een parallelsysteem niet alle componenten nodig heeft om een volledig systeem te laten functioneren. Er zijn drie situaties die zich voor kunnen doen:

a. Het parallelsysteem heeft minstens één component nodig om het volledige systeem te laten functioneren;
b. Het parallelsysteem heeft minstens *k* componenten nodig om het volledige systeem te laten functioneren;
c. Het parallelsysteem heet alle componenten nodig om het volledige systeem te laten functioneren.

Als alle componenten dezelfde hoeveelheid variabele leveren die nodig is om het systeem te laten werken kan met de volgende formule berekend worden hoeveel componenten nodig zijn om een volledig systeem te laten functioneren:

$$k = ceil\left(\frac{a}{b}\right) \quad (4)$$

Waarvoor geldt:

- *k* = aantal componenten die nodig zijn om het systeem te laten werken;
- *a* = de totale hoeveelheid van de variabele die nodig is voor de componenten die de variabele ontvangen in het parallelsysteem;
- *b* = de hoeveelheid wat één van de componenten van het parallelsysteem levert van een variabele.

Hierbij wordt *k* altijd naar boven afgerond, omdat componenten niet half functioneren. Componenten functioneren of functioneren niet. Bij het voorbeeld van twee service pompen die elk 34 liter water per minuut leveren en drie brandslangen die samen 30 liter water per minuut nodig hebben van de service pompen zou de formule als volgt ingevuld worden: *k* = 30/34 ≈ 1. Als componenten niet dezelfde hoeveelheid variabele leveren kan deze formule niet gebruikt worden, omdat *b* de hoeveelheid is wat één van de componenten van het parallelsysteem levert. Is dus deze hoeveelheid niet voor elke component hetzelfde komt er een ander antwoord uit. Stel, drie brandslangen leveren water, de eerste 10 liter per minuut, de tweede 20 liter per minuut en de derde 30 liter per minuut en er is 20 liter water per minuut nodig om een brand te blussen. Dan kan niet met alleen de eerste brandslag een brand geblust worden. Echter de tweede brandslang zou dit wel alleen kunnen. Het aantal van de componenten die nodig zijn om het systeem te laten functioneren verschillen in een situatie waarbij niet alle componenten dezelfde hoeveelheid variabele leveren.

Elke genummerde situatie die bij een parallelsysteem zich voor kan doen wordt apart uitgelegd:

1. *Alle componenten hebben dezelfde R_i en alle componenten leveren dezelfde hoeveelheid variabele:*
In dit geval wordt eerst met behulp van formule 4 berekend hoeveel componenten er nodig zijn om een volledig systeem te laten functioneren. Daarna doet zich één van de volgende situaties voor:

   a. H*et parallelsysteem heeft één component nodig om het volledige systeem te laten functioneren:*
   In deze situatie wordt de volgende formule gebruikt om de betrouwbaarheid van het systeem te berekenen (University of Georgia, z.d.):

$$R_s = 1 - \prod_{i=1}^{n} F_i \tag{5}$$

   b. *Het parallelsysteem heeft minstens k componenten nodig om het volledige systeem te laten functioneren:*
   In deze situatie wordt de volgende formule gebruikt om de betrouwbaarheid van het systeem te berekenen (ReliaSoft Corporation, 2015, p. 36):

$$R_s(k,n,R) = \sum_{r=k}^{n} \left(\binom{n}{r} * R^r * (1-R)^{n-r} \right) \tag{6}$$

   Waarvoor geldt:

   - k = minimumaantal componenten die nodig zijn;
   - n = totaalaantal componenten;
   - R = de betrouwbaarheid van de component.

   Het gedeelte $\binom{n}{r}$ wordt berekend aan de hand van de volgende formule (Buijs, 2012, p. 103):

$$\binom{n}{r} = \frac{n!}{(n-r)!\,r!} \tag{7}$$

c. *Het parallelsysteem heeft alle componenten nodig om het volledige systeem te laten functioneren:*
In deze situatie wordt formule 3 gebruikt van het seriesysteem, omdat bij een seriesysteem ook alle componenten nodig zijn om een volledig systeem te laten werken.

2. *Niet alle componenten hebben dezelfde R_i, maar alle componenten leveren wel dezelfde hoeveelheid variabele*:
Als niet alle componenten dezelfde R_i hebben kunnen alle formules die tot nu toe gebruikt worden niet gehanteerd worden, omdat er in deze formule eenmalig een R_i ingevoerd wordt die alle R_i's vertegenwoordigd van de componenten. Daarom wordt in dit geval alle mogelijke combinaties bepaald waarvoor het systeem functioneert. Hierbij wordt eerst vooraf bepaald hoeveel componenten nodig zijn om het volledig systeem te laten functioneren met behulp van formule 4. Stel, er zijn minstens vier van de zes componenten nodig, dan worden alle combinaties bepaald waarbij vier componenten functioneren, alle combinaties waarbij vijf componenten functioneren en de combinatie waarbij alle componenten, dus zes componenten, functioneren. Voor alle combinaties wordt R_s berekend. Als laatst worden alle R_s's bij elkaar opgeteld. Stel, er zijn twee service pompen die een verschillende R_i hebben, waarvan er minstens 1 moet functioneren. Dan worden eerste alle combinaties bepaald, dit zijn er drie:

- Alleen de eerste service pomp functioneert;
- Alleen de tweede service pomp functioneert;
- Beide service pompen functioneren.

Dan wordt R_s voor deze drie combinaties berekend (ReliaSoft Corporation, 2015, p. 36):

- $R_s = R_{eerste\ service\ pomp} * (1 - R_{tweede\ service\ pomp})$
- $R_s = (1 - R_{eerste\ service\ pomp}) * R_{tweede\ service\ pomp}$
- $R_s = R_{eerste\ service\ pomp} * R_{tweede\ service\ pomp}$

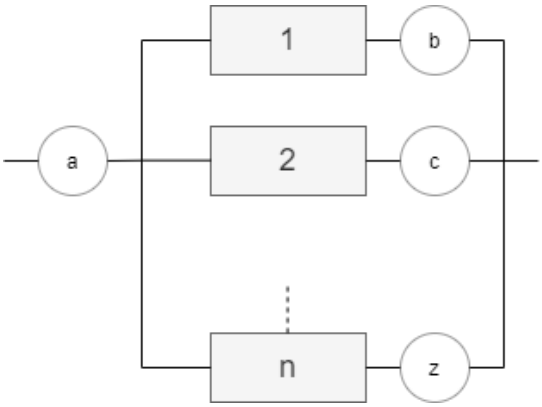
Vervolgens worden deze drie R_s's bij elkaar opgeteld, waardoor de betrouwbaarheid van het parallelsysteem berekend is.

3. *Alle componenten hebben dezelfde R_i, maar niet alle componenten leveren dezelfde hoeveelheid variabele*:
Als niet alle componenten dezelfde hoeveelheid variabele leveren, maar wel allemaal dezelfde R_i hebben, dan worden alle combinaties bepaald waarbij het volledig systeem functioneert. Elke combinatie die er mogelijk is met de componenten wordt bepaald. Er wordt niet meer van tevoren berekend hoeveel variabelen er nodig zijn, omdat elke component een verschillende variabele levert en formule 4 dus niet gebruikt kan worden. Voor elke combinatie apart wordt bepaald of deze combinatie genoeg van het variabele levert wat nodig is. Stel, er zijn drie service pompen, waarbij de eerste 10 liter water, de tweede 20 liter en de derde 30 liter per minuut levert. Een brandslang heeft 20 liter water per minuut nodig van deze drie servicepompen. Dan is de combinatie waarbij alleen de eerste servicepomp functioneert niet goed, omdat er dan te weinig water per minuut geleverd wordt aan de brandslang. Het volledig systeem functioneert dan niet meer. De combinatie waarbij de eerste en de twee service pomp functioneren is wel goed, omdat ze dan samen 30 liter water per minuut leveren, wat voldoende is voor de brandslang. Op deze wijze worden alle combinaties bepaald. Daarna wordt voor elke combinatie die genoeg van levert van de variabele R_s berekend op dezelfde manier als bij de tweede punt. Als laatste worden alle R_s's bij elkaar opgeteld.

4. *Niet alle componenten hebben dezelfde R_i en niet alle componenten leveren dezelfde hoeveelheid variabele*: In dit geval wordt manier van de betrouwbaarheid berekenen van de tweede punt en het derde punt bij elkaar gevoegd. Alle mogelijke combinaties die er zijn worden bepaald. Vervolgens wordt voor elke combinatie bepaald of er genoeg geleverd wordt van de variabele die nodig is. Voor elke combinatie die genoeg levert wordt R_s berekend. Als laatste worden alle R_s's bij elkaar opgeteld.

3.2.3. Half serie-/parallelsysteem

Een systeem is niet meer volledig parallel als iedere component van het systeem van de vorige component dezelfde variabele nodig heeft, maar aan de volgende component een eigen variabele levert. Een voorbeeld hiervan is te zien in figuur 9. De blokken met daarin de nummers zijn de componenten. De blokken met de letters zijn de variabelen. Alle componenten in dit voorbeeld hebben de variabele a nodig, maar ze leveren allen een eigen variabele.



Figuur 9 Voorbeeld half serie-/parallelsysteem

Bij een half serie-/parallelsysteem geldt dat alle componenten nodig zijn om het systeem te laten functioneren. Dit omdat iedere component verantwoordelijk is voor een variabele en niet gemist kan worden in het systeem. Daarom wordt voor de berekening van de betrouwbaarheid van zo'n systeem dezelfde formule gebruikt als voor de berekening van de betrouwbaarheid van een seriesysteem (formule 3) en wordt dit systeem een half serie-/parallelsysteem genoemd.

4. Voorbereiding voor optimalisatie

In dit hoofdstuk wordt uitgelegd wat het model voor het optimaliseren van de onderhoudsstrategie voor elke component doet als voorbereiding op de optimalisatie. Om de betrouwbaarheid te berekenen van een systeem is het van belang om de basissystemen (seriesysteem, parallelsysteem en half serie-/parallelsysteem) op te sporen in het systeem en vervolgens de formule voor de betrouwbaarheid en daarmee de faalkans te bepalen voor het systeem. Door het schip schematisch weer te geven in een graaf kunnen de basissystemen opgespoord worden. In paragraaf 4.1 wordt uitgelegd hoe een schip in een graaf gezet wordt. In paragraaf 4.2 wordt beschreven hoe het systeem opgedeeld wordt per KPI (Key Performance Indicator). Vervolgens wordt in paragraaf 4.3 uitgelegd hoe de formule van de betrouwbaarheid (faalkans) bepaald wordt van het hele systeem. Om de onderhoudsstrategieën te optimaliseren is echter niet alleen de formule voor de faalkans nodig, maar ook de formule van de onderhoudskosten van een schip. Hoe deze formule opgesteld wordt, wordt uitgelegd in paragraaf 4.4. Paragraaf 4.5 beschrijft hoe de belangrijkheid van de componenten van een schip bepaald wordt. Vervolgens wordt in paragraaf 4.6 uitgelegd wat de startoplossing is voor het optimaliseren en hoe mogelijke buuroplossingen gezocht kunnen worden. Als laatste wordt in paragraaf 4.7 uitgelegd hoe nauwkeurig de optimalisatie uitgevoerd moet worden.

4.1. Het systeem in een graaf

Om de onderhoudsstrategieën voor alle componenten van een schip te kunnen optimaliseren is een graaf met de relaties tussen al deze componenten nodig. In een graaf is het voor het oog in een keer duidelijk wat de relaties zijn tussen de componenten, daarnaast kan een algoritme vanuit een graaf detecteren wat de relaties zijn tussen de componenten. Om een graaf tot stand te brengen wordt een JSON-bestand aangereikt vanuit Damen. Een JSON-bestand is een bestand dat eenvoudige datastructuren en objecten opslaat. Het bevat gegevens in een standaard data-uitwisselingsformaat en is menselijk leesbaar (FileTypes, z.d.). Figuur 10 laat een stukje van een JSON-bestand van een schip (ASD 3212) zien. Zo'n JSON-bestand met alle gegevens wordt aangereikt vanuit Damen.


```
{
  "ship": {
    "name": "Multratug 31",
    "imo": "9695614",
    "length": 30.4,
    "beam": 10.8,
    "components": [
      {
        "ctype": "main engine",
        "id": 1,
        "make": "Cat",
        "type": "3516",
        "unexpected_repair_costs": 3000,
        "current_strategy": "run to failure",
        "downtime": 1,
        "maintenance_strategy": [
          {
            "name": "run to failure",
            "MTBF": 60,
            "expected_repair_costs": 0,
            "number_of_tasks": 0
          },
          {
            "name": "planned maintenance",
            "MTBF": 400,
```


Figuur 10 Deel van JSON-bestand van een ASD 3212

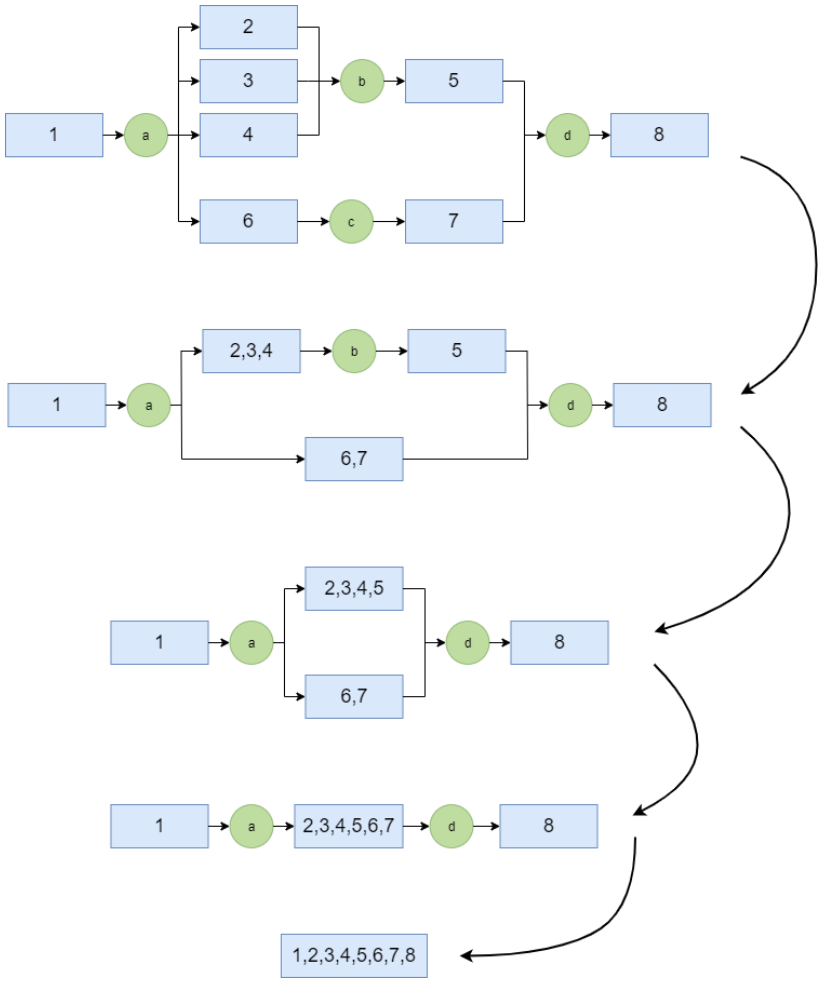
Ieder schip heeft een eigen JSON-bestand waarin de benodigde gegevens voor de optimalisatie van de onderhoudsstrategieën van de componenten van het schip staan. In bijlage 1 staat een tabel met alle benodigde variabelen. Globaal genoemd zijn dit enkele eigenschappen van het schip, zoals de naam, lengte, breedte. Verder staan er vijf lijsten in met gegevens. De eerste lijst bevat alle componenten waaruit een schip is opgebouwd en die één of meerdere onderhoudsstrategieën bevatten. De tweede lijst bevat alle variabelen die tussen de componenten doorgegeven worden, bijvoorbeeld elektriciteit of brandstof. In de derde lijst staan de KPI's van een schip, bijvoorbeeld varen, veiligheid. De vierde lijst bevat de relaties tussen een component en een variabele, waarbij de component een variabele levert. In de laatste lijst staan de relaties tussen een component en een variabele, waarbij de component een variabele nodig heeft.
Dit bestand wordt ingelezen in Neo4j. Neo4j is een grafendatabase, speciaal gebouwd om gebruik te maken van gegevensrelaties (Neo4j, 2020). Door middel van een query kunnen opdrachten gegeven worden aan de database en data/informatie opgehaald of aangepast worden (Dingemans, 2019). Om een JSON-bestand van een schip in een graaf in Neo4j te zetten zijn de volgende vijf query's opgesteld. De eerste query is voor het maken van de componenten in een graaf. De componenten vormen knopen (vertices) van de graaf. Alle eigenschappen van de componenten uit het JSON-bestand worden per component opgeslagen als eigenschap van de knoop, behalve de onderhoudsstrategieën. Deze vormen aparte knopen die met een zijde (edge) aan de bijbehorende component verbonden zijn. De tweede query is voor het maken van variabelen in een graaf. De variabelen vormen ook knopen van de graaf. Alle eigenschappen van de variabelen uit het JSON-bestand worden per variabele opgeslagen als eigenschap van de knoop. De derde query is voor het maken van de KPI's in een graaf. De KPI's vormen als derde knopen van de graaf. Alle eigenschappen van de KPI's uit het JSON-bestand worden per KPI opgeslagen als eigenschap van de knoop. De vierde query is voor het maken van de relaties tussen de componenten en de variabelen waarbij de componenten een variabele leveren. Deze relatie wordt aangeduid met een zijde tussen de knoop van de component en de knoop van de variabele en heeft als eigenschap de hoeveelheid van de variabele die geleverd wordt door de component. De vijfde query is voor het maken van de relaties tussen de componenten en de variabelen waarbij de componenten variabelen ontvangen. Ook deze relatie wordt aangeduid met een zijde tussen de knoop van de component en de knoop van de variabele en heeft als eigenschap de hoeveelheid van de variabele die de component. Aan de hand van deze vijf query's ontstaat er een graaf met alle benodigde gegevens van een schip zie bijlage 2. De vijf query's staan in bijlage 3.

4.2. Systeem opdelen per KPI

Een KPI is een Key Performance Indicator, de kritieke prestatie-indicator die uitgedrukt wordt in een getal. Een klant heeft bepaalde wensen aan taken wat een schip moet kunnen. Deze taken worden omgezet in KPI's waardoor prestaties beoordeeld kunnen worden (LeanSixSigmaTools, 2020). Een klant heeft bijvoorbeeld als wens dat het schip moet kunnen varen, maar daarnaast wil de klant ook dat het schip brand kan blussen. Dit schip heeft dan twee KPI's, namelijk 'varen' en 'brand blussen'. Wat het model doet is per KPI de componenten opzoeken die verantwoordelijk zijn voor deze KPI. De componenten die verantwoordelijk zijn voor een KPI leveren direct of indirect variabelen aan een KPI. Bij de KPI 'varen' is het brandblussysteem niet nodig. Het brandblussysteem levert ook geen variabelen aan de KPI 'varen'. De componenten die bij het brandblussysteem horen kunnen dus voor de KPI 'varen' weggelaten worden. Echter een schroef levert de variabele 'mechanische kracht' aan de KPI 'varen' (bijlage 1). Deze component is dus verantwoordelijk voor de KPI 'varen'. Maar alle componenten van de KPI 'varen' kunnen niet weggelaten worden voor de KPI 'brand blussen'. Zonder de hoofdmotor kunnen de servicepompen van het brandblussysteem niet functioneren. De hoofdmotor is dus nodig voor de KPI 'varen' en de KPI 'brand blussen'. Een component kan dus nodig zijn bij meerdere KPI's.
Uiteindelijk ontstaan er op deze wijze voor elke KPI van een schip een systeem. Het model optimaliseert vervolgens deze systemen apart.

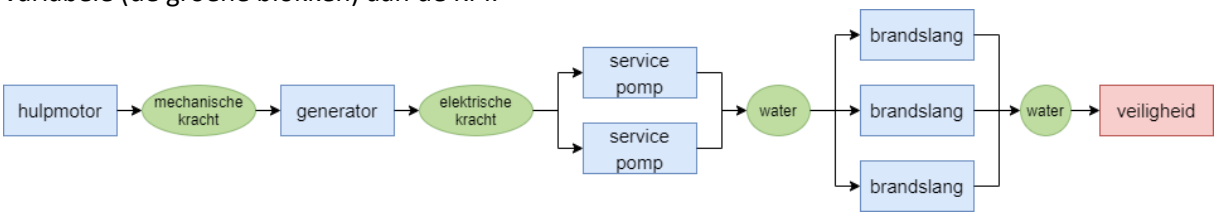
4.3. Formules voor de faalkans van het systeem opstellen

Voor elke KPI wordt de formule voor de faalkans opgesteld om de betrouwbaarheid van het systeem voor de KPI te kunnen berekenen. Bij het optimaliseren moet de beste onderhoudsstrategie gekozen worden voor elke component. De formule van de faalkans wordt opgesteld met de MTBF van de componenten. De MTBF van een component hangt af van de gekozen onderhoudsstrategie. Daarom wordt er een formule opgesteld om de faalkans te berekenen waarbij de MTBF van iedere component de parameters zijn. Op deze wijze hoeft bij het optimaliseren alleen voor iedere component de MTBF ingevuld te worden in de formule voor de faalkans en hoeft niet telkens het hele systeem vereenvoudigd te worden.
Het model stelt de formule voor de faalkans op aan de hand van een recursieve functie. Iedere keer wordt onderzocht of het systeem een serie subsysteem, parallel subsysteem of een half serie-/parallel subsysteem bevat. Dit subsysteem wordt dan vereenvoudigd tot één samengevoegd component. De functie stopt als het model niet meer vereenvoudigd kan worden en er één samengevoegd component overgebleven is. Tijdens het vereenvoudigen worden de formules voor de faalkans en voor de onderhoudskosten van een subsysteem opgesteld, zodat bij de laatste vereenvoudiging de formules voor de faalkans en de onderhoudskosten van het hele systeem bekend zijn. Als de functie stopt met vereenvoudigen heeft het overgebleven virtuele component een formule voor de faalkans van het hele systeem. Een compleet voorbeeld van een vereenvoudiging van een systeem tot één component staat in figuur 11. De blauwe blokken zijn de componenten en de groene blokken zijn de variabelen. Dat geldt voor alle verder genoemde voorbeelden. De componenten 2, 3 en 4 zijn samen een parallelsysteem. Deze componenten worden in de eerste stap vereenvoudigd tot één component. De componenten 6 en 7 zijn samen een seriesysteem en worden eveneens vereenvoudigd tot één component. Op deze wijze gaat het hele proces door tot het systeem nog maar één component bevat.



Figuur 11 Voorbeeld vereenvoudigen van een systeem

Voor het opsporen van de drie subsystemen zijn drie algoritmes opgesteld. Deze algoritmes sporen niet alleen een subsysteem op, maar stellen ook de formule van de faalkans en de formule voor de onderhoudskosten van het opgespoorde subsysteem op. Deze algoritmes worden uitgelegd met behulp van het voorbeeld van het schip de ASD 3212 uit bijlage 2. Van dit systeem wordt een deel genomen om de algoritmes uit te leggen. Dit deel van het systeem staat in figuur 12 en hoort bij de KPI 'veiligheid' (rode blok), want de componenten (de blauwe blokken) leveren direct of indirect een variabele (de groene blokken) aan de KPI.

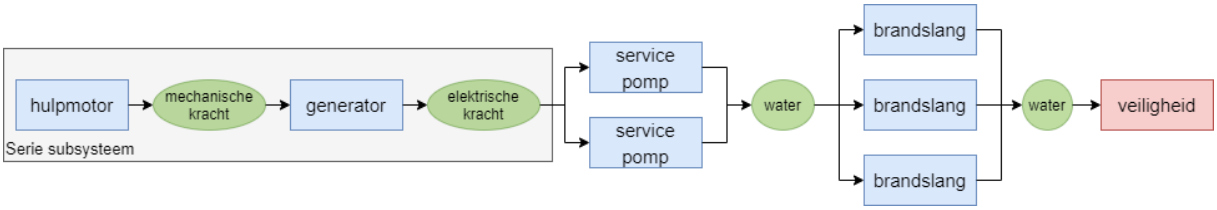


Figuur 12 Stukje systeem van een ASD 3212

Het eerste algoritme spoort een serie subsysteem op in een systeem. Dit algoritme wordt uitgelegd in paragraaf 4.3.1. Het tweede algoritme spoort een parallel subsysteem op in een systeem en wordt uitgelegd in paragraaf 4.3.2. Vervolgens wordt het derde algoritme uitgelegd in paragraaf 4.3.3. Dit algoritme spoort een half serie-/parallel subsysteem op in een systeem. Als laatst wordt in paragraaf 4.3.4 uitgelegd hoe de formules van de subsystemen bij elkaar gevoegd worden tot één formule voor het hele systeem.

4.3.1. Vereenvoudigen van een serie subsysteem

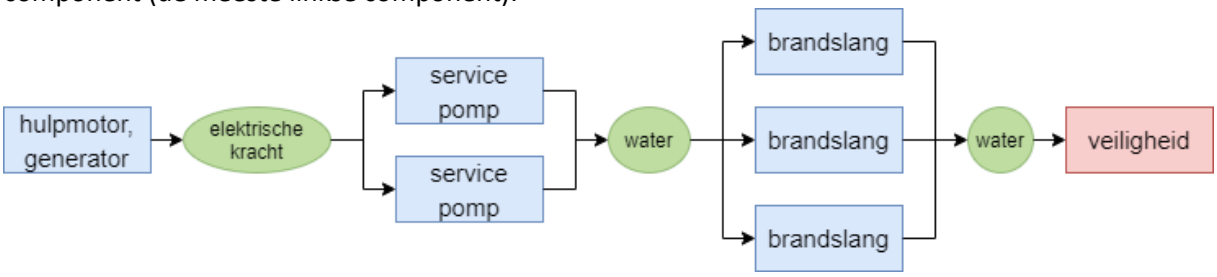
In figuur 13 staat aangegeven met een kader welk deel van het systeem een serie subsysteem is.



Figuur 13 Serie subsysteem

Om serie subsystemen op te zoeken in het systeem worden eerst de componenten geselecteerd die een variabele leveren aan één of geen component en een variabele ontvangen van één of geen component. In dit voorbeeld is dat de hulpmotor. De hulpmotor ontvangt geen variabele van een component en levert een variabele aan één component. De generator ontvangt wel een variabele van maar één component, maar levert een variabele aan twee componenten. Daarom wordt de generator niet geselecteerd.
Vervolgens wordt onderzocht of de geselecteerde componenten zich in een serie subsysteem bevinden. In dit geval is er één geselecteerde component, namelijk de hulpmotor. De eerste stap die hierbij gedaan wordt is het opzoeken van de componenten die een variabele ontvangen van de hulpmotor. Dit is de generator. Als de generator niet meer dan één component heeft waar hij een variabele van ontvangt, hoort deze component bij het serie subsysteem. De generator ontvangt alleen een variabele van de hulpmotor. De generator hoort dus bij het serie subsysteem. Omdat de generator een variabele ontvangt van de hulpmotor hoort de generator aan de rechterkant van het serie subsysteem. Het serie subsysteem is nu dus: hulpmotor – generator.

Nu worden de componenten opgezocht die een variabele krijgen van de generator. Dit mogen er niet twee of meer zijn. In dit geval levert de generator een variabele aan twee componenten, namelijk twee service pompen. Deze componenten worden dus niet verder onderzocht en niet toegevoegd aan het serie subsysteem. De generator is in dit geval de laatste component aan de rechterkant van het serie subsysteem. Het algoritme onderzoekt net zolang de rechterkant van het systeem tot een component aan meerdere componenten een variabele levert of aan geen één component meer een variabele levert. Dit is dan het laatste component aan de rechterkant van het serie subsysteem.
De linkerkant van het serie subsysteem moet ook onderzocht worden. Daarom worden de componenten opgezocht die een variabele leveren aan de hulpmotor. Er is geen component die een variabele levert aan de hulpmotor. De hulpmotor is dus het laatste component aan de linkerkant van het serie subsysteem. Stel, er is een component die een variabele levert aan de hulpmotor. Dan wordt onderzocht of dit component het enige component is die een variabele levert aan de hulpmotor. Wanneer dit het geval is, wordt dit component aan de linkerkant van het serie subsysteem toegevoegd. Het serie subsysteem wordt dan: component – hulpmotor – generator.
Het algoritme onderzoekt net zolang de linkerkant van het systeem tot een component van meerdere componenten een variabele ontvangt of van geen één component een variabele ontvangt. Dit is dan het laatste component aan de linkerkant van het serie subsysteem.
Aan beide kanten van het serie subsysteem is het laatste component gevonden. Het serie subsysteem is gevonden, namelijk hulpmotor – generator. Als er meerdere componenten worden geselecteerd die een variabele leveren aan één of geen component en een variabele ontvangen van één of geen component, dan wordt de volgende component opgezocht of deze zich in een serie subsysteem bevindt. Hiervoor wordt eerst gekeken of de component al toegevoegd is aan een serie subsysteem. Stel, de generator is ook geselecteerd, dan zit de generator al in de serie subsysteem hulpmotor – generator. De generator hoeft dan niet meer onderzocht te worden.
Als laatste wordt onderzocht of het serie subsysteem langer is dan één component. Als een serie subsysteem één component bevat hoeft het serie subsysteem niet samengevoegd worden tot één component. In dit geval bevat het serie subsysteem twee componenten en moet het serie subsysteem samengevoegd worden tot één component. Dit nieuwe component krijgt de namen van alle componenten die zich in het serie subsysteem bevinden. In dit geval de namen hulpmotor en generator. In figuur 14 zijn de componenten van het serie subsysteem samengevoegd tot één component (de meeste linkse component).



Figuur 14 Serie subsysteem hulpmotor - generator samengevoegd

De nieuwe component krijgt als variabelen een formule voor de faalkans en een formule voor de onderhoudskosten van dit serie subsysteem. Hoe de formule van de onderhoudskosten wordt opgesteld, wordt uitgelegd in hoofdstuk 4.4. Om de formule van de faalkans te bepalen wordt de formule van de betrouwbaarheid van het serie subsysteem opgesteld. De formule voor de betrouwbaarheid van een seriesysteem is:

$$R_s = \prod_{i=1}^{n} R_i$$

Voor dit serie subsysteem is de formule voor de betrouwbaarheid dan als volgt:

$$R_s = R_{hulpmotor} * R_{generator}$$

De betrouwbaarheid van de componenten is niet bekend, echter de MTBF van de component wel. Daarom wordt de formule omgeschreven naar de faalkans van de component. Met behulp van de MTBF van de component kan de faalkans berekend worden. Aan de hand van de complementregel (formule 2) is bekend dat één min de betrouwbaarheid hetzelfde is als de faalkans, de formule wordt dan als volgt:

$$R_s = (1 - F_{hulpmotor}) * (1 - F_{generator})$$

Omdat de formule voor de faalkans van het systeem nodig is en niet de formule voor de betrouwbaarheid wordt ook hier de complementregel toegepast:

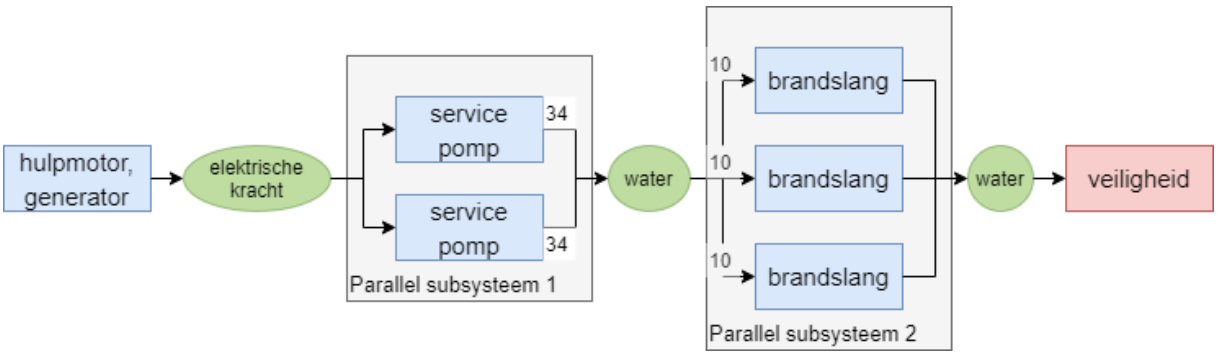
$$F_s = 1 - \left((1 - F_{hulpmotor}) * (1 - F_{generator})\right)$$

Daarnaast rekent Damen met de MTBF van componenten en niet met de faalkans, waardoor de formule er als volgt uitziet (formule 1):

$$F_s = 1 - \left(\left(1 - \frac{1}{MTBF_{hulpmotor}}\right) * \left(1 - \frac{1}{MTBF_{generator}}\right)\right)$$

4.3.2. Vereenvoudigen van een parallel subsysteem

Nu er geen serie subsystemen meer zijn in het systeem wordt het systeem onderzocht of het systeem parallel subsystemen bevat. In dit systeem bevinden zich twee parallel subsystemen. Deze parallel subsystemen staan aangegeven in figuur 15. De componenten met dezelfde namen worden van boven naar beneden genummerd. De namen van de componenten van de service pompen worden dan als volgt: service pomp 1 en service pomp 2. De namen van brandslangen worden brandslang 1, brandslang 2 en brandslang 3. De variabelen met dezelfde namen worden van links naar rechts genummerd. De variabele tussen de service pompen en de brandslangen wordt water 1. De variabele tussen de brandslangen en de KPI (veiligheid) wordt water 2.



Figuur 15 Twee parallel subsystemen

Om een parallel subsysteem op te zoeken in een systeem wordt er geïtereerd over alle componenten van het systeem. Bij de uitleg van dit voorbeeld wordt er begonnen bij de component service pomp 1. Voor elk van de componenten worden de volgende stappen ondernomen, waarbij in *cursief* de uitleg staat van het eerste parallel subsysteem van dit voorbeeld:

1. De variabele die de component ontvangt en de variabele die de component levert opzoeken;
   Service pomp 1 ontvangt elektrische kracht en levert water 1.

2. Als er één of meerdere variabelen ontvangen en geleverd worden door de component:
    a. De componenten opzoeken die de ontvangen variabele bij stap 1 ook ontvangen;
    Voor elektrische kracht zijn dit service pomp 1 en service pomp 2.
    b. De variabelen die deze componenten ontvangen en de variabelen die deze componenten leveren moet opgezocht worden. Hierbij wordt de component die gebruikt wordt bij stap 1 niet meer onderzocht;
    Service pomp 2 ontvangt elektrische kracht en levert water 1.
    c. De variabelen die ontvangen worden moeten gelijk zijn aan de variabele die de component van stap 1 ontvangt en de variabele die de geleverd worden moet gelijk zijn aan de variabele die de component van stap 1 levert;
    Service pomp 1, de component van stap 1, ontvangt elektrische kracht. Service pomp 2 ontvangt ook elektrische kracht. Dus deze zijn gelijk. Service pomp 1 levert water 1 en service pomp 2 levert ook water 1. Ze leveren dus beide dezelfde variabele. Tot nu toe zijn service pomp 1 en service pomp 2 een parallel subsysteem.
3. Als een component geen variabelen levert, dan worden de stappen a tot c van stap 2 gevolgd zonder dat er naar de variabelen die geleverd worden gekeken wordt bij de stappen b en c;
4. Als een component geen variabelen ontvangt, dan worden de stappen a tot c van stap 2 gevolgd zonder dat er naar de variabelen die ontvangen worden gekeken wordt bij de stappen b en c. Ook wordt bij a dan niet de componenten die ook de eerste variabele ontvangen opgevraagd, maar de componenten die de eerste variabele leveren.

Na deze stappen volgt een extra check of de componenten die tot nu toe parallel aan elkaar zijn, daadwerkelijk parallel aan elkaar zijn. Hiervoor wordt het volgende stappenplan gevolgd:

1. Voor alle componenten die tot nu toe als parallel beschouwd worden, worden de variabelen die de componenten ontvangen en leveren in lijsten gezet. Voor de variabelen die ontvangen worden en voor de variabelen die geleverd worden zijn aparte lijsten. Als een variabele die ontvangen wordt al in de lijst zit wordt deze er niet nog een keer ingezet. Datzelfde geldt voor als een variabele die geleverd wordt. Als één van deze twee lijsten meer dan één variabele bevat, dan is het tot nu toe beschouwde parallel subsysteem geen parallel subsysteem;
    De tot nu toe beschouwde parallel subsysteem bevat de componenten service pomp 1 en service pomp 2. Service pomp 1 ontvangt elektrische kracht, dus de variabele elektrische kracht wordt in de lijst gezet voor de ontvangen variabelen. Service pomp 2 ontvangt ook elektrische kracht. Deze variabele zit al in de lijst voor de ontvangen variabelen dus wordt deze variabele niet nog een keer in de lijst gezet. Service pomp 1 levert water 1. Deze variabele wordt in de lijst gezet voor de variabelen die geleverd worden. Service pomp 2 levert ook water 1. Deze variabele hoeft dus niet nog een keer in de lijst gezet te worden, omdat deze variabele er al in staat. De lijst voor ontvangen variabelen bevat dus alleen elektrische kracht. De lijst met variabelen die geleverd worden bevat alleen het variabele water 1. Beide lijsten bevatten één variabele. Daarom worden de componenten service pomp 1 en service pomp 2 als een parallel subsysteem beschouwd.
2. Van één component wordt de variabele die de component ontvangt opgezocht en de variabele die de component levert opgezocht;
    Service pomp 1 ontvangt elektrische kracht en levert water 1.
3. Als de component een variabele ontvangt worden de componenten opgezocht die deze variabele ontvangen en als de component een variabele levert worden de componenten opgezocht die deze variabele leveren. Het aantal van deze componenten moet gelijk zijn aan het aantal componenten die tot nu toe als parallel subsysteem beschouwd worden.
    De componenten die elektrische kracht ontvangen zijn service pomp 1 en service pomp 2, dus twee componenten. De componenten die water 1 leveren zijn service pomp 1 en service pomp 2, dus twee componenten. De componenten die tot nu toe als parallel subsysteem beschouwd worden zijn service pomp 1 en service pomp 2, dus ook twee componenten. De aantallen komen dus overeen. Service pomp 1 en service pomp 2 zijn dus samen een parallel subsysteem.

Uiteindelijk wordt door het algoritme dus het parallel subsysteem gevonden met de componenten service pomp 1 en service pomp 2. Naast dit uitgewerkte voorbeeld vindt het algoritme ook het parallel subsysteem met de componenten brandslang 1, brandslang 2 en brandslang 3. Deze twee gevonden parallel subsystemen worden beide samengevoegd tot één component. De nieuwe component van de

eerste parallel subsysteem krijgt de namen service pomp 1 en service pomp 2. De nieuwe component van de tweede parallelle subsysteem krijgt de namen van de drie brandslangen. In figuur 16 zijn de twee parallel subsystemen beide samengevoegd tot één component.

Figuur 16 Twee parallel subsystemen samengevoegd

De componenten krijgen als variabelen ook de formule voor de faalkans en de formule voor de onderhoudskosten. Hoe de formule voor de onderhoudskosten van een subsysteem opgesteld, wordt staat uitgelegd in hoofdstuk 4.4.
Om de formule voor de faalkans van een parallel subsysteem op te stellen moet eerst onderzocht worden of de componenten van het parallel subsysteem dezelfde hoeveelheid van de variabele die geleverd wordt leveren. Als dit verschillend is bij één of meerdere componenten, wordt de formule voor de faalkans opgesteld zoals uitgelegd in hoofdstuk 3.2.2. Als de hoeveelheid bij alle componenten hetzelfde is, moet voordat de formule van de faalkans opgesteld wordt, berekend worden hoeveel componenten er nodig zijn om het systeem te laten werken. Dit wordt op de volgende manier berekend:

1. Van één component van het parallel subsysteem wordt de hoeveelheid dat de component levert genomen;
   Service pomp 1 levert 34 liter water per minuut.
2. Van alle componenten die de variabele ontvangen die het parallel subsysteem levert worden de hoeveelheden van deze componenten nodig hebben opgeteld;
   De drie brandslangen hebben alle drie 10 liter per minuut van het variabele water 1 nodig. Bij elkaar opgeteld is dit dus 30 liter per minuut.
3. De variabele *k* wordt berekend met formule 4:

$$k = ceil\left(\frac{a}{b}\right)$$

   Waarvoor geldt:
   - *k* = aantal componenten die nodig zijn om het systeem te laten werken;
   - *a* = de totale hoeveelheid van het variabele wat nodig is voor de componenten die de variabele ontvangen van het parallel subsysteem;
   - *b* = de hoeveel wat één component levert van een variabele van de componenten van het parallel subsysteem;

   Voor dit parallel subsysteem geldt: k = 30/34 ≈ 1.

   a. Als *k* = 1, dan wordt de faalkans berekend aan de hand van formule 5;
   b. Als *k* ≠ 1 en k ≠ aantal componenten in parallel subsysteem, dan wordt formule 6 gebruikt voor het berekenen van de failure rate;
   c. Als *k* = aantal componenten in parallel subsysteem, dan wordt de formule voor de faalkans van dit parallel subsysteem opgesteld aan de hand van formule 3;

   In dit geval geldt k = 1, en het aantal componenten in het parallel subsysteem is 2. Dit betekent dat volgens stap a formule 5 gebruikt moet worden om de formule voor de faalkans op te stellen:

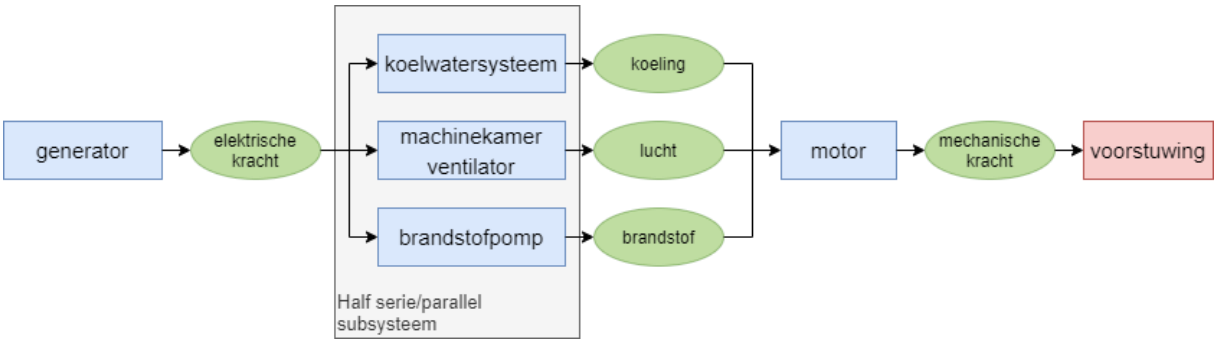
$$R_s = 1 - \prod_{i=1}^{n} F_i$$

Deze formule wordt op dezelfde manier omgezet tot de formule voor de faalkans als in hoofdstuk 4.3.1. Dit leidt tot de volgende formule voor de faalkans van het systeem:

$$R_s = 1 - \left(F_{service\ pomp\ 1} * F_{service\ pomp\ 2}\right)$$
$$F_s = 1 - \left(1 - \left(F_{service\ pomp\ 1} * F_{service\ pomp\ 2}\right)\right)$$
$$F_s = F_{service\ pomp\ 1} * F_{service\ pomp\ 2}$$
$$F_s = \frac{1}{MTBF_{service\ pomp\ 1}} * \frac{1}{MTBF_{service\ pomp\ 2}}$$

4.3.3. Vereenvoudigen van een half serie-/parallel subsysteem

Voor het opsporen van een half serie-/parallel subsysteem is een derde algoritme opgesteld. Het voorbeeld uit de voorgaande twee paragrafen bevat geen half serie-/parallel subsysteem. Daarom is voor dit subsysteem een ander voorbeeld opgesteld, zie figuur 17. Met een kader staat aangegeven welke componenten een half serie-/parallel subsysteem vormen. De blauwe blokken zijn de componenten, de groene blokken de variabelen en het rode blok is de KPI.



Figuur 17 Half serie-/parallel subsysteem

Om een half serie-/parallel subsysteem op te zoeken in een systeem wordt er geïtereerd over alle componenten van het systeem. Bij de uitleg van dit voorbeeld wordt er begonnen bij de component koelwatersysteem. Voor elk van de componenten worden de volgende stappen ondernomen, waarbij in *cursief* de uitleg staat van dit voorbeeld:

1. De componenten die een variabele leveren aan de component of een variabele ontvangen van de component opzoeken;
   De generator levert een variabele aan het koelwatersysteem. De motor ontvangt een variabele van het koelwatersysteem.
2. Als er één of meerdere componenten zijn die een variabele leveren aan de component en er één of meerdere componenten zijn die een variabele ontvangen van de component:
   a. De componenten opzoeken die een variabele ontvangen van de eerste component die gevonden is bij stap 1 (die een variabele levert aan de component);
      Voor de generator is dat het koelwatersysteem, de machinekamer ventilator en de brandstofpomp.
   b. Van de gevonden componenten van stap a de componenten opzoeken die een variabele leveren aan de component en een variabele ontvangen van de component;
      De generator levert een variabele aan het koelwatersysteem en de motor ontvangt een variabele van het koelwatersysteem. De generator levert eveneens een variabele aan de machinekamer ventilator en de motor ontvangt een variabele van de machinekamer ventilator. De generator levert ook een variabele aan de brandstofpomp en de motor ontvangt een variabele van de brandstofpomp.

c. De componenten die gevonden zijn bij stap b die een variabele leveren aan de componenten moeten gelijk zijn aan de componenten die gevonden zijn bij stap 1 die een variabele leveren aan de component. De gevonden componenten bij stap b die een variabele ontvangen van de componenten moet hetzelfde zijn als de gevonden componenten die een variabele ontvangen van de component bij stap 1.
Bij stap b is het de generator die aan alle componenten een variabele levert. Bij stap 1 is het ook de generator die een variabele levert aan de component. De component die een variabele ontvangt van alle componenten is de motor. Bij stap 1 is het eveneens de motor die een variabele ontvangt van de component. De componenten zijn dus gelijk aan elkaar, daarom worden de componenten koelwatersysteem, machinekamer ventilator en de brandstofpomp als een half serie-/parallel systeem beschouwd.

3. Als er geen componenten zijn die een variabele ontvangen van de component, dan worden de stappen a tot c van stap 2 gevolgd zonder dat er naar de componenten gekeken wordt die een variabele ontvangen van de component;
4. Als er geen componenten zijn die een variabele leveren aan de component, dan worden de stappen a tot c van stap 2 gevolgd zonder dat er naar de componenten gekeken wordt die een variabele leveren aan de component. Bij a wordt dan niet de componenten opgezocht gerelateerd aan de eerste component die een variabele levert aan de component, maar van de eerste component die een variabele ontvangt van de component.

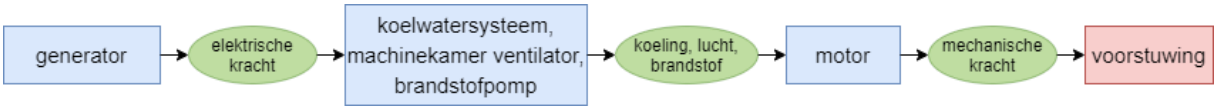
Na deze stappen volgt een extra check of de componenten die nu als half serie-/parallel beschouwd worden ook echt half serie-/parallel zijn. Hiervoor wordt het volgende stappenplan gevolgd:

1. Van de componenten die tot nu toe half serie-/parallel aan elkaar beschouwd worden, worden de variabelen opgezocht die deze componenten ontvangen en leveren;
Het koelwatersysteem, de machinekamer ventilator en de brandstofpomp ontvangen alle drie de variabele elektrische kracht. Het koelwatersysteem levert de variabele koeling, de machinekamer ventilator levert de variabele lucht en de brandstofpomp levert de variabele brandstof. De variabelen die geleverd worden door het tot nu toe beschouwde half serie-/parallel subsysteem geleverd worden zijn dus achtereenvolgens: koeling, lucht en brandstof.
2. Eén van de volgende drie situaties moet zich voordoen:
    a. Het aantal variabelen die ontvangen worden en het aantal variabelen die geleverd worden komen beide overeen met het aantal half serie-/parallel beschouwde componenten;
    b. 1 of 0 variabelen die ontvangen worden en het aantal variabelen die geleverd worden komt overeen met het aantal half serie-/parallel beschouwde componenten;
    c. 1 of 0 variabelen die ontvangen worden en het aantal variabelen die geleverd worden komt overeen met het aantal half serie-/parallel beschouwde componenten.

    In dit geval doet situatie b zich voor. Er is één variabele die ontvangen wordt, namelijk elektrische kracht. Er zijn drie variabelen die geleverd worden. Dit aantal komt overeen met het aantal componenten die tot nu toe half serie-/parallel beschouwd worden.
3. Van één component uit de componenten die half serie-/parallel beschouwd worden, worden de componenten die variabelen leveren aan dit component of variabelen ontvangen van dit component opgezocht;
De component waarvan uitgegaan wordt is het koelwatersysteem. De generator levert een variabele aan het koelwatersysteem. De motor ontvangt een variabele van het koelwatersysteem.
4. Van de variabelen die geleverd worden door de component bij stap 1 worden één voor één de componenten opgezocht die deze variabele ontvangen. Deze moeten overeenkomen met de componenten die een variabele leveren aan de component bij stap 3;
De variabelen die het tot nu toe half serie-/parallel beschouwde subsysteem levert uit stap 1 zijn koeling, lucht en brandstof. De motor ontvangt deze drie variabelen. Dit komt overeen met de gevonden component uit stap 3 die een variabele ontvangt van het koelwatersysteem.

5. Van de variabelen die ontvangen worden door de component bij stap 1 worden één voor één de componenten opgezocht die deze variabele leveren. Deze moeten overeenkomen met de componenten die een variabele leveren aan de component bij stap 3.
   Er is een variabele gevonden die geleverd wordt aan het tot nu toe beschouwde half serie-/parallel subsysteem. Dit is de variabele elektrische kracht. De component die deze variabele levert is de generator. Dit komt overeen met de component die gevonden is bij stap 3 die een variabele levert aan het koelwatersysteem.

Als één van de stappen niet klopt, dan zijn de componenten niet samen een half serie-/parallel subsysteem. Als de stappen wel kloppen, dan zijn de componenten samen een half serie-/parallel subsysteem. Bij de componenten koelwatersysteem, machinekamer ventilator en brandstofpomp zijn alle stappen doorlopen en kloppen alle stappen. Deze componenten worden daarom samengevoegd tot één component. Dit component krijgt dan de namen van alle componenten uit het half serie-/parallel subsysteem. In dit geval de namen koelwatersysteem, machinekamer ventilator en brandstofpomp. De variabelen die het subsysteem ontvangen wordt, worden ook samengevoegd tot één component. Hetzelfde geldt voor de variabelen die geleverd worden door het subsysteem. In dit geval ontvangt het subsysteem alleen de variabele elektrische kracht en hoeft deze niet samengevoegd te worden tot één variabele. Er zijn echter wel drie variabelen die het subsysteem levert. Deze variabelen worden samengevoegd tot één nieuwe variabele. Deze variabele krijgt dan de namen van alle drie de variabelen, namelijk koeling, lucht en brandstof. In figuur 18 zijn de componenten en de variabelen van het half serie-/parallel subsysteem samengevoegd.



Figuur 18 Half serie-/parallel subsysteem samengevoegd

De nieuwe samengevoegde component krijgt als variabele de formule van de faalkans en de formule voor de onderhoudskosten van dit subsysteem. Hoe de formule voor de onderhoudskosten van een subsysteem opgesteld wordt staat uitgelegd in hoofdstuk 4.4. Bij half serie-/parallel subsystemen wordt de faalkans op dezelfde manier berekent als bij een seriesysteem, daarom wordt de formule voor een half/serie subsysteem opgesteld aan de hand van formule 3:

$$R_s = \prod_{i=1}^{n} R_i$$

Van het half serie-/parallel subsysteem met het koelwatersysteem, de machinekamer ventilator en de brandstofpomp wordt formule voor de faalkans als volgt:

$$R_s = R_{koelwatersysteem} * R_{machinekamer\ ventilator} * R_{brandstofpomp}$$
$$R_s = (1 - F_{koelwatersysteem}) * (1 - F_{machinekamer\ ventilator}) * (1 - F_{brandstofpomp})$$
$$F_s = 1 - \left((1 - F_{koelwatersysteem}) * (1 - F_{machinekamer\ ventilator}) * (1 - F_{brandstofpomp})\right)$$
$$F_s = 1 - \left(\left(1 - \frac{1}{MTBF_{koelwatersysteem}}\right) * \left(1 - \frac{1}{MTBF_{machinekamer\ ventilator}}\right) * \left(1 - \frac{1}{MTBF_{brandstofpomp}}\right)\right)$$

Deze formule is op dezelfde manier omgezet naar een formule voor de faalkans als in hoofdstuk 4.3.1.

4.3.4. Samenvoegen van formules tot één formule voor het hele systeem

Het voorbeeld van de ASD 3212 ziet er na het opsporen van de serie subsysteem uit paragraaf 4.3.1 en de twee parallel subsystemen uit paragraaf 4.3.2 als volgt uit (figuur 19):

Figuur 19 Voorbeeld ASD3212 na eerste keer opsporen van subsystemen

Het eerste samengevoegde component van de hulpmotor en de generator bevat de volgende formule voor de faalkans:

$$F_{hulpmoter,generator} = 1 - \left(\left(1 - \frac{1}{MTBF_{hulpmotor}}\right) * \left(1 - \frac{1}{MTBF_{generator}}\right)\right)$$

De tweede samengevoegde component met de twee service pompen bevat de volgende formule voor de faalkans:

$$F_{service\ pomp\ (2x)} = \frac{1}{MTBF_{service\ pomp\ 1}} * \frac{1}{MTBF_{service\ pomp\ 2}}$$

Voor de derde samengevoegde component met de drie brandslangen wordt ervan uitgegaan dat er ook minstens één component nodig is om het hele systeem te laten functioneren. De formule voor dit samengevoegde component wordt dan op dezelfde manier opgesteld als de samengevoegde component met de service pompen. De formule voor de faalkans van dit derde samengevoegde component wordt dan:

$$F_{brandslang\ (3x)} = \frac{1}{MTBF_{brandslang\ 1}} * \frac{1}{MTBF_{brandslang\ 2}} * \frac{1}{MTBF_{brandslang\ 3}}$$

Zoals te zien in figuur 19 bestaat het systeem alleen nog maar uit een serie subsysteem, omdat alle componenten een variabele leveren aan één ander component en niet meer dan één variabele ontvangen van een ander component. De formule voor de betrouwbaarheid van een serie subsysteem is:

$$R_s = \prod_{i=1}^{n} R_i$$

Voor dit serie subsysteem is de formule voor de betrouwbaarheid dan als volgt:

$$R_s = R_{hulpmotor,generator} * R_{service\ pomp\ (2x)} * R_{brandslang\ (3x)}$$
$$R_s = \left(1 - F_{hulpmotor,generator}\right) * \left(1 - F_{service\ pomp\ (2x)}\right) * \left(1 - F_{brandslang\ (3x)}\right)$$

De formule voor de faalkans wordt dan:

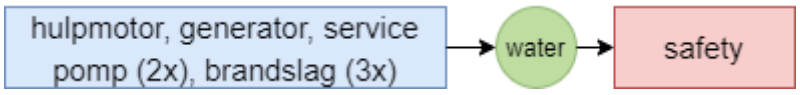
$$F_s = 1 - \left(\left(1 - F_{hulpmotor,generator}\right) * \left(1 - F_{service\ pomp\ (2x)}\right) * \left(1 - F_{brandslang\ (3x)}\right)\right)$$

Al deze drie componenten zijn samengevoegde componenten, daarom wordt op de plaats van $F_{component}$ de formule voor de faalkans van dit samengevoegde component ingevuld. Als één component hiervan geen samengevoegde component zou zijn, werd hiervoor de formule voor de faalkans van één component ingevuld, welke 1 delen door de MTBF van de component is. Om te

onderzoeken of een component een samengevoegd component is, onderzoekt het algoritme of een component een komma in de naam heeft. Alle samengevoegde componenten bevatten namelijk de namen van de componenten van het subsysteem dat samengevoegd is tot één component, met een komma tussen de namen. Het serie subsysteem van de hulpmotor met de generator zijn samengevoegd tot één component met de nieuwe naam 'hulpmotor, generator'. Het algoritme ziet dat dit een samengevoegd component is door de komma in de naam.
In dit geval zijn alle componenten van het overgebleven serie subsysteem samengevoegde componenten. Alle componenten hebben een komma in de naam. Het algoritme vult dan op de plaats van de faalkans van de component de formule voor de faalkans van dit samengevoegde component in. De formule ziet er dan als volgt uit:

$$F_s = 1 - \left(\left(1 - \left(1 - \left(\left(1 - \frac{1}{MTBF_{hulpmotor}} \right) * \left(1 - \frac{1}{MTBF_{generator}} \right) \right) \right) \right) \right. \\ * \left(1 - \left(\frac{1}{MTBF_{service\ pomp\ 1}} * \frac{1}{MTBF_{service\ pomp\ 2}} \right) \right) \\ \left. * \left(1 - \left(\frac{1}{MTBF_{brandslang\ 1}} * \frac{1}{MTBF_{brandslang\ 2}} * \frac{1}{MTBF_{brandslang\ 3}} \right) \right) \right)$$

De drie samengevoegde componenten worden samengevoegd tot één nieuw component. Dit nieuwe component krijgt als nieuwe naam alle namen van de componenten van het serie subsysteem. De nieuwe naam van dit component wordt 'hulpmotor, generator, servicepomp 1, servicepomp 2, brandslang 1, brandslang 2, brandslang 3'. In figuur 20 staan de service pompen aangegeven als service pomp (2x) en de brandslangen als brandslang (3x). Daarnaast krijgt dit nieuwe component de formule voor de faalkans en de formule voor de onderhoudskosten van het serie subsysteem. Hoe de formule voor de onderhoudskosten opgesteld wordt staat uitgelegd in paragraaf 4.4.



Figuur 20 ASD 3212 vereenvoudigd tot één component

Het systeem van de ASD 3212 bestaat nu nog uit één component, wat betekend dat heel het systeem vereenvoudigd is. Hierdoor is de formule voor de faalkans en de formule voor de onderhoudskosten van dit overgebleven component de formule voor de faalkans en de formule voor de onderhoudskosten van het hele systeem.

4.4. Formule opstellen voor de onderhoudskosten van het systeem

In paragraaf 4.3 is uitgelegd hoe de formule voor de faalkans van het systeem opgesteld wordt. Om de optimale onderhoudsstrategie te bepalen van alle componenten zijn er twee optimalisatiemogelijkheden. De eerste mogelijkheid is dat de faalkans geoptimaliseerd wordt bij een bepaald budget. De tweede mogelijkheid is dat de onderhoudskosten geoptimaliseerd worden bij een maximale toegestane faalkans. Voor deze optimalisaties is niet alleen de formule voor de faalkans nodig, maar ook de formule voor de onderhoudskosten van het systeem. Er zijn vier soorten onderhoudskosten per component:

1. Jaarlijkse kosten bij ongeplande downtime. Dit zijn de kosten die veroorzaakt worden doordat een KPI niet meer gegarandeerd kan worden, doordat een component stuk is en gerepareerd wordt. Stel, alle drie de brandslangen van de ASD 3212 zijn nodig om de veiligheid te garanderen en één brandslang is stuk. Dan kan er geen veiligheid gegarandeerd worden. Om dit op vangen in de optimalisatie worden er kosten aan KPI's gehangen zodat het niet garanderen van een KPI geld kost. De formule voor deze kosten is:

$$Jaarlijkse\ kosten\ ongeplande\ downtime = \frac{1}{MTBF} * t * a \qquad (8)$$

2. Jaarlijkse kosten bij geplande downtime. Dit zijn de kosten die veroorzaakt worden doordat er gepland onderhoud wordt uitgevoerd aan de component waardoor de KPI niet meer gegarandeerd kan worden, omdat de component op dat moment niet in werking is. De formule voor deze kosten is:

$$Jaarlijkse\ kosten\ geplande\ downtime = n * t * b \qquad (9)$$

3. Jaarlijkse kosten voor onverwachte reparatie. Dit zijn de kosten op componentniveau voor de reparatie van het onverwachts stuk gaan van de component. De formule voor deze kosten is:

$$Jaarlijkse\ kosten\ onverwachte\ reparatie = \frac{1}{MTBF} * c \qquad (10)$$

4. Jaarlijkse kosten voor verwachte reparatie. Dit zijn de kosten op componentniveau voor de reparatie van een component tijdens gepland onderhoud. De formule voor deze kosten is:

$$Jaarlijkse\ kosten\ verwachte\ reparatie = n * d \qquad (11)$$

Hierbij is *t* de downtime in dagen van de component, *a* de kosten voor ongeplande downtime van de KPI in euro's per dag, *b* de kosten voor geplande downtime van de KPI in euro's per dag, *n* het aantal keer dat gepland onderhoud uitgevoerd wordt aan de component per jaar, *c* de onverwachte reparatiekosten van de component, *d* de verwachte reparatiekosten van de component en de MTBF, de gemiddelde tijd tussen falen van de component, in dagen.
De vier formules voor de vier soorten kosten worden samengevoegd tot één formule:

$$Totaal\ jaarlijkse\ kosten = \left(\frac{1}{MTBF} * (t * a + c)\right) + (n * (t * b + d)) \qquad (12)$$

Bij het vereenvoudigen van subsystemen, zoals uitgelegd in paragraaf 4.3 wordt ook telkens de formule voor de onderhoudskosten van het subsysteem opgeteld. Voor iedere component wordt bovenstaande formule ingevuld. Als voorbeeld wordt de hulpmotor van de ASD 3212 genomen voor de KPI 'varen/trekkracht'. Als er onderhoud uitgevoerd wordt aan de hulpmotor van het schip de ASD 3212 is de downtime (*t*) 1 dag. De onverwachte reparatiekosten (*c*) 1.200 euro. Als de

onderhoudsstrategie planned maintenance is, dan zijn de verwachte reparatiekosten (*d*) 400 euro. Er wordt dan 12 keer per jaar onderhoud uitgevoerd, dus (*n*) is 12. De MTBF bij deze onderhoudsstrategie voor de hulpmotor is dan 400 dagen. De kosten voor één dag ongeplande downtime (*a*) voor de KPI 'varen/trekkracht' is 10.0000 euro per dag. De kosten voor één dag geplande downtime (*b*) voor is 6.000 euro's. Als deze variabelen ingevuld worden in de formule voor de totaal jaarlijkse kosten, wordt de formule als volgt:

$$Totaal\ jaarlijkse\ kosten = \left(\frac{1}{400} * (1 * 10.000 + 1.200)\right) + (12 * (1 * 6.000 + 400)) = 76.828{,}00\ euro$$

De onderhoudsstrategie wordt per component geoptimaliseerd. Tijdens de optimalisatie kan er dus tussen de verschillende onderhoudsstrategieën van de component gekozen worden om de optimale onderhoudsstrategie van de component te vinden. De MTBF en het aantal keer dat het onderhoud uitgevoerd wordt (*n*) is per onderhoudsstrategie verschillend. De MTBF van de hulpmotor bij de onderhoudsstrategie run-to-failure is 60 dagen. Dit is dus een kortere MTBF dan bij de strategie condition-based maintenance, waar de MTBF 400 dagen is. Het aantal keer dat het onderhoud uitgevoerd wordt bij de strategie run-to-failure is 0. Dit kom doordat onderhoud alleen uitgevoerd wordt als een component stuk gaat. De component moet dan gerepareerd worden. Er wordt geen onderhoud vooraf gepland. Bij de strategie condition-based maintenance wordt er 12 keer per jaar gepland onderhoud uitgevoerd.
De MTBF en het aantal keer dat er onderhoud uitgevoerd wordt verschilt dus per onderhoudsstrategie. Daarom wordt de formule opgesteld met een variabele MTBF en een variabele *n*.

$$Totaal\ jaarlijkse\ kosten = \left(\frac{1}{MTBF_{hulpmotor}} * (1 * 10.000 + 1.200)\right) + \left(n_{hulpmotor} * (1 * 6.000 + 400)\right)$$

Op deze wijze wordt er voor iedere component van een subsysteem een formule voor de onderhoudskosten opgesteld. Deze formules worden achter elkaar gezet, zodat de kosten opgeteld worden. Blijkt een component een samengevoegd component te zijn, dan heeft dit component al een formule voor de onderhoudskosten en wordt deze formule achter de formule voor het gevonden subsysteem geplaatst.
Stel, er is nog een hulpmotor in een gevonden subsysteem. Dan wordt de formule voor deze hulpmotor achter de formule van de andere hulpmotor gezet. De totale formule wordt dan:

$$\begin{aligned} Totaal\ jaarlijkse\ kosten = & \left(\frac{1}{MTBF_{hulpmotor\ 1}} * (1 * 10.000 + 1.200)\right) \\ & + \left(n_{hulpmotor\ 1} * (1 * 6.000 + 400)\right) \\ & + \left(\frac{1}{MTBF_{hulpmotor\ 2}} * (1 * 10.0000 + 1.200)\right) \\ & + \left(n_{hulpmotor\ 2} * (1 * 6.000 + 400)\right) \end{aligned}$$

Op deze wijze gaat het algoritme door tot het hele systeem vereenvoudigd is tot één component en dit component uiteindelijk de formule van de onderhoudskosten voor alle componenten bevat.

4.5. Bepalen volgorde van belangrijkheid van componenten

Voordat de optimale onderhoudsstrategie bepaald wordt per component, is het goed om te weten welke componenten de meeste invloed hebben op de faalkans en/of de onderhoudskosten van het schip. De componenten die veel invloed hebben op de faalkans en/of de onderhoudskosten van een systeem van een schip zijn belangrijker dan componenten die (bijna) geen invloed hebben op de faalkans en/of onderhoudskosten van het schip. Het vermoeden is dat de wanneer de belangrijke componenten geoptimaliseerd worden en de minder belangrijke componenten als onderhoudsstrategie run-to-failure krijgen, dit een goede oplossing is. De minder belangrijke componenten hebben weinig invloed hebben op de faalkans van een systeem van een schip. Daarom wordt voor elke component van het systeem van een schip berekend hoeveel de faalkans daalt per geïnvesteerde euro. Iedere component heeft één of meerdere onderhoudsstrategieën. Als een component één onderhoudsstrategie heeft kan voor deze component de onderhoudsstrategie niet geoptimaliseerd worden en daalt de faalkans niet per geïnvesteerde euro. Voor de componenten met meerdere onderhoudsstrategieën wordt voor iedere mogelijke onderhoudsstrategie, behalve de huidige onderhoudsstrategie, uitgerekend wat de belangrijkheidsratio is aan de hand van de volgende formule:

$$belangrijkheidsratio = \frac{daling\ faalkans}{stijging\ kosten} = \frac{faalkans_{run-to-failure} - faalkans_{strategie}}{kosten_{strategie} - kosten_{run-to-failure}}$$

Deze belangrijkheidsratio betekent in woorden de daling van de faalkans per geïnvesteerde euro. De daling van de faalkans en de stijging van de onderhoudskosten wordt altijd ten opzichte van de faalkans en de onderhoudskosten van de onderhoudsstrategie run-to-failure bepaald. Aan het begin van de optimalisatie heeft iedere component de onderhoudsstrategie run-to-failure, omdat deze strategie voor elke component in werkelijkheid uitvoerbaar is. Voor iedere component kan gekozen worden om pas onderhoud aan de component uit te gaan voeren als de component defect is. Daarentegen, de onderhoudsstrategie predictive maintenance is (bijvoorbeeld) niet voor elke component mogelijk, omdat predictive maintenance gebruik maakt van de data van de component. Van sommige componenten is er simpelweg geen data beschikbaar. Bij deze componenten is predictive maintenance niet mogelijk. Om de belangrijkheid van de componenten te bepalen wordt de hoogste belangrijkheidsratio van de component genomen. De component met de hoogste belangrijkheidsratio is de meest belangrijke component. Deze component heeft het meest invloed op de faalkans en de onderhoudskosten van het hele systeem. Op deze wijze ontstaat er een volgorde in belangrijkheid van de componenten.
Aan het begin van de optimalisatie is bij alle componenten de toegepaste onderhoudsstrategie run-to-failure. Om de maximale ratio te bepalen van een component moet voor dit component de faalkans en de onderhoudskosten berekend worden bij elke onderhoudsstrategie. Als voorbeeld wordt de hulpmotor van het schip de ASD 3212 genomen. De hulpmotor heeft drie onderhoudsstrategieën die toegepast kunnen worden: run-to-failure, planned maintenance en condition-based maintenance. In onderstaande tabel (tabel 1) staat per onderhoudsstrategie wat de MTBF is van de hulpmotor, wat de faalkans is voor het hele systeem en wat de onderhoudskosten zijn voor het hele systeem. Deze waarden zijn niet realistisch, maar slechts weergegeven om deze formule en werkwijze te verduidelijken.

Tabel 1 MTBF, faalkans en kosten per onderhoudsstrategie

Onderhoudsstrategie	MTBF	Faalkans	Kosten
Run-to-failure	60	0,99	1.000.000
Planned maintenance	450	0,90	4.000.000
Condition-based maintenance	400	0,93	5.000.000

In de volgende tabel (tabel 2) wordt per onderhoudsstrategie uitgerekend wat de belangrijkheidsratio is van de component hulpmotor. De onderhoudsstrategie met de hoogste belangrijkheidsratio is planned maintenance. Voor iedere euro die geïnvesteerd wordt, daalt de faalkans met 0,00000003.

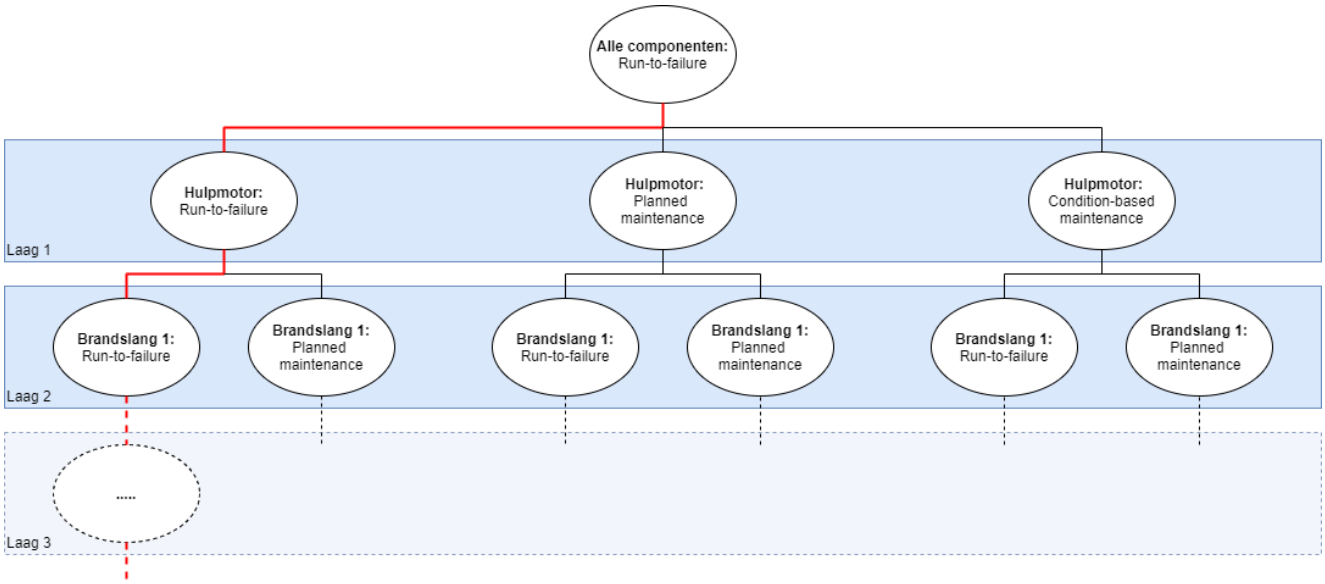
Tabel 2 Belangrijkheidsratio per onderhoudsstrategie

Onderhoudsstrategie	Belangrijkheidsratio
Run-to-failure	$\frac{0{,}99 - 0{,}99}{1.000.000 - 1.000.000} = 0$
Planned maintenance	$\frac{0{,}99 - 0{,}90}{4.000.000 - 1.000.000} = 0{,}00000003$
Condition-based maintenance	$\frac{0{,}99 - 0{,}93}{5.000.000 - 1.000.000} = 0{,}000000015$

Op deze wijze wordt voor iedere component van het hele systeem de hoogste belangrijkheidsratio bepaald. Vervolgens worden alle componenten gesorteerd in een lijst op basis van hoogste belangrijkheidsratio naar laagste belangrijkheidsratio. De componenten die het meest belangrijk zijn, staan op deze wijze aan het begin van de lijst. De componenten die het minst belangrijk zijn staan aan het eind van de lijst.

4.6. Het opstellen van een zoekboom aan de hand van buuroplossingen

Aan de hand van de volgorde van belangrijkheid van de componenten wordt een zoekboom opgesteld. Een zoekboom is een schematische weergave van alle mogelijke oplossingen waarin gezocht kan worden naar de optimale oplossing. In figuur 21 is een voorbeeld van een zoekboom te zien van een systeem van een schip. De startoplossing, waar bij iedere component de onderhoudsstrategie run-to-failure is, staat bovenaan. Vanuit deze startoplossing worden buuroplossingen gezocht. Dat wordt gedaan aan de hand van de volgorde van de belangrijkheid van de componenten. De buuroplossingen van de startoplossingen zijn dan de oplossingen waarbij de onderhoudsstrategie van de meest belangrijke component is veranderd naar een andere strategie en één oplossing waarbij de strategie hetzelfde is gebleven.
Stel, de volgorde van belangrijkheid van componenten van een deel van het schip de ASD 3212 is: hulpmotor, brandslang 1, brandslang 2, brandslang 3, service pomp 1, service pomp 2. De hulpmotor heeft, zoals in het vorige hoofdstuk te zien is, drie mogelijke onderhoudsstrategieën, namelijk run-to-failure, planned maintenance en condition-based maintenance. Dan heeft de startoplossing drie buuroplossingen, namelijk voor elke onderhoudsstrategie één. Dit is te zien in figuur 21. De tweede meest belangrijke component is brandslang 1. Deze heeft bijvoorbeeld twee mogelijke onderhoudsstrategieën, namelijk run-to-failure en planned maintenance. De drie buuroplossing van de startoplossing hebben dan elk weer twee eigen buuroplossingen. Namelijk voor één elke onderhoudsstrategie van brandslang 1. Op deze manier wordt een zoekboom opgesteld met alle mogelijke oplossingen die er zijn voor de optimalisatie. Elke component heeft zijn eigen laag in de zoekboom. De hulpmotor vormt laag 1, brandslang 1 vormt laag 2, enzovoorts. Het systeem van de het schip de ASD 3212 die in dit voorbeeld gebruikt wordt heeft bijvoorbeeld zeven componenten. De zoekboom krijgt dan zeven lagen.



Figuur 21 Voorbeeld zoekboom van de ASD 3212

Een oplossing van de optimalisatie bevat voor elke component een onderhoudsstrategie. Een tak in de zoekboom vormt een oplossing, de rode lijn in figuur 21 is een voorbeeld van een tak. Een oplossing moet altijd laag voor laag gelezen worden, te beginnen bij de eerste laag. In de eerste laag heeft, als de rode lijn gevolgd wordt, de hulpmotor run-to-failure als onderhoudsstrategie. Daarna wordt de tweede laag gelezen. Hierbij heeft, de rode lijn volgend, brandslang 1 ook run-to-failure als onderhoudsstrategie. Op deze wijze wordt een hele tak gevolgd, zodat bekend is welke onderhoudsstrategieën de componenten hebben bij een oplossing.

4.7. Nauwkeurigheid optimalisatie

Er zijn twee optimalisatiemogelijkheden. De klant kan aangeven of de faalkans geoptimaliseerd moet worden bij een bepaald budget of dat de onderhoudskosten geoptimaliseerd moeten worden bij een maximale toegestane faalkans. Bij beide optimalisatiemogelijkheden speelt de faalkans van een systeem een belangrijke rol. De faalkans van een systeem wordt berekend aan de hand van de MTBF van iedere component. Zoals eerder aangegeven is de MTBF een gemiddelde van een verdeling. De tijd tussen het falen van de componenten is niet altijd precies vier weken. Het kan ook drie weken of vijf weken zijn. Als er geoptimaliseerd wordt tot het globale optimum gevonden is, zou er geoptimaliseerd worden in de onzekerheid van de MTBF. Daarom hoeft bij deze optimalisatie niet het globale optimum gevonden te worden, omdat een goede oplossing ook volstaat. Zou de faalkans van een systeem bij het globale optimum 0,997 zijn en bij een goede gevonden oplossing 0,995. Dan is dit geen groot verschil, omdat beide faalkansen een gemiddelde van een verdeling zijn.

5. Optimalisatie methoden

Er zijn twee manieren waarop de optimalisatie uitgevoerd kan worden. De eerste is dat de faalkans geoptimaliseerd wordt, waarbij de onderhoudskosten niet hoger mogen zijn dan het door de klant gewenste budget. De tweede manier is dat de kosten geoptimaliseerd waarbij de faalkans van het hele systeem van een schip niet hoger mag zijn dan de door de klant aangegeven maximale toegestane faalkans. In dit hoofdstuk wordt voor verschillende optimalisatiemethoden onderzocht of ze geschikt zijn voor de optimalisatie van de onderhoudsstrategieën. Bij deze optimalisatie is er niet één formule waarbij het minimum gevonden wordt, maar deze optimalisatie heeft een zoekboom bestaande uit alle mogelijke oplossingen waarin het globale optimum bepaald moet worden. Daarom zijn er verschillende methoden opgezocht die in een zoekboom of aan de hand van buuroplossingen het globale optimum of een goede oplossing kunnen vinden. De volgende optimalisatiemethoden zijn gevonden:

1. Brute-force (paragraaf 5.1)
2. Pruning heuristieken (paragraaf 5.2)
    a. Breadth-first
    b. Depth-first
3. Hill Climbing (paragraaf 5.3)
4. Tabu Search (paragraaf 5.4)
5. Simulated Annealing (paragraaf 5.5)
6. Greedy algoritme (paragraaf 5.6)

Bij elke methode wordt uitgelegd hoe de methode werkt en of de methode toepasbaar is voor de optimalisatie van de onderhoudsstrategieën. Elke methode die toepasbaar blijkt zal geïmplementeerd worden. In paragraaf 5.7 zal uiteindelijk kort aangegeven worden welke optimalisatiemethoden uiteindelijk geïmplementeerd worden.

5.1. Brute-force

Brute-force gaat alle mogelijke oplossingen langs en kiest dan de optimale oplossing. Dit algoritme kan niet gebruikt worden voor complexe problemen, omdat er dan te veel oplossingen mogelijk zijn (Oxford Reference, 2019). Het nadeel van het gebruik van dit algoritme is dat het doel van de optimalisatie niet bekend is. De optimale oplossing is alleen bekend als alle mogelijke oplossingen berekend zijn (Schut, 2018).
Een voorbeeld van het gebruik van Brute-force zijn Brute-force aanvallen. Een Brute-force aanval is een illegale poging om je wachtwoord of pincode te achterhalen. Het doet niet anders dan net zo lang wachtwoorden raden tot men toegang heeft (van Rhee, z.d.).
Als dit model gebruikt voor een echt schip, wordt er gekeken naar ongeveer 50 tot 500 componenten. Wanneer een schip ongeveer 300 componenten heeft die in het model gestopt worden en elk van deze componenten heeft drie onderhoudsstrategieën die mogelijk zijn, dan betekent dit dat er $3^{300} = 1.36891479 \ldots * 10^{143}$ oplossingen zijn (Hellings, 2011). Dit hoge aantal oplossingen zal zorgen voor een zeer lange rekentijd.
Het voordeel van deze optimalisatiemethode is dat altijd het globale optimum bereikt wordt, daarom zal deze optimalisatiemethode geïmplementeerd worden. Het voordeel is dat aan de hand van deze methode de antwoorden van eventuele andere optimalisatiemethoden getest kunnen worden of deze een goede oplossing geven of niet.

5.2. Pruning heuristieken

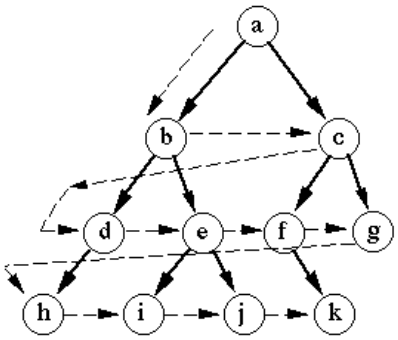
Pruning heuristieken zijn ontwikkeld om de zoekruimte voor een oplossing te beperken. De definitie van heuristiek is volgens Van Dale: 'de leer van het vinden, de wetenschap die langs methodische weg tot ontdekkingen of uitvindingen leert komen' (Ensie, 2018). Pruning betekend letterlijk snoeien. Een pruning heuristiek kapt bijvoorbeeld een deel van een zoekboom af, omdat volgens een pruning regel blijkt dat deze tak minder goede oplossingen bevat dan de andere takken (Veerman, 2008). Er worden dan voor deze tak geen nieuwe buuroplossingen bepaald.
Door het afkappen van een tak van een zoekboom de zoekruimte voor een oplossing beperkt wordt, is de rekentijd van pruning heuristieken korter dan voor brute force. Er zijn twee soorten pruning heuristieken (Veerman, 2008):

1. Breadth-first heuristiek
2. Depth-first heuristiek

De werking en de toepasbaarheid van de breadth-first heuristiek zal in paragraaf 0 uitgelegd worden. In paragraaf 0 wordt de werking en de toepasbaarheid van de depth-first heuristiek uitgelegd.

5.2.1. Breadth-first heuristiek

Breadth-first heuristieken zoeken op een bepaald niveau van de zoekboom alle verschillende knopen af. Als daar nog geen toegelaten oplossing tussen zit, dan wordt vanuit de reeds onderzochte knopen naar een volgend niveau met deelproblemen van de zoekboom gegaan. Ook op dit niveau worden eerst weer alle knopen onderzocht voordat weer naar een volgend niveau wordt gegaan (Veerman, 2008). In figuur 22 staat ter verduidelijking met pijlen aangegeven hoe een breadth-first heuristiek door een zoekboom gaat.



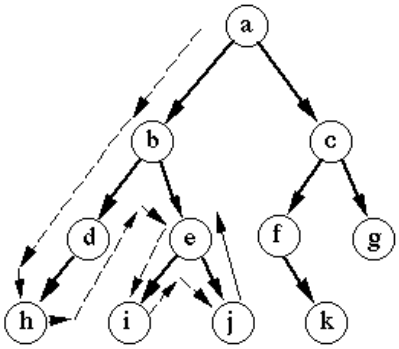
Figuur 22 Pad van een breadth-first heuristiek door een zoekboom

In figuur 21 van hoofdstuk 4.6 staat een voorbeeld van een zoekboom voor de optimalisatie van onderhoudsstrategieën. Deze zoekboom is opgesteld aan de hand van de volgorde van meest belangrijk component naar minst belangrijk component. Dit algoritme zoekt in de breedte door de zoekboom. Hierdoor worden eerst de oplossingen onderzocht voor de meest belangrijke component, daarna voor de iets minder belangrijke component, enzovoorts. Daarom lijkt dit algoritme goed implementeerbaar.

5.2.2. Depth-first heuristiek

Depth-first heuristieken zoeken eerst een bepaalde tak helemaal af. Wanneer er dan geen mogelijkheden meer over zijn, wordt met backtracking terug gegaan totdat weer een andere oplossing kan worden bereikt (Veerman, 2008). De laatste knoop waar nog een andere oplossing was, wordt opgezocht en van daaruit worden buuroplossingen onderzocht. In figuur 23 staat ter verduidelijking met pijlen aangegeven hoe een depth-first heuristiek door een zoekboom gaat.

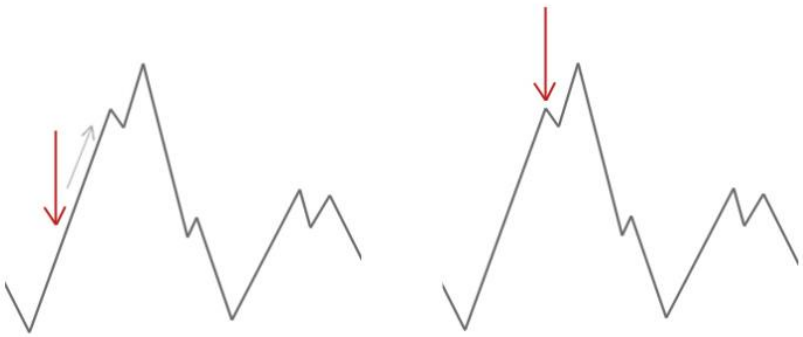
In figuur 21 van hoofdstuk 4.6 staat een voorbeeld van een zoekboom voor de optimalisatie van onderhoudsstrategieën. Deze zoekboom is opgesteld aan de hand van de volgorde van meest belangrijk component naar minst belangrijk component. De Depth-first heuristiek gaat in de diepte door de zoekboom en is daarom minder goed toepasbaar dan de Breadth-first heuristiek. Stel, de optimale oplossing in figuur 23 bevindt zich bij bolletje g onder de tak van bolletje c en het algoritme werkt eerst alle takken af onder bolletje b, dan is de rekentijd langer dan wanneer de heuristiek in de breedte door de zoekboom gaat. Daarom wordt deze heuristiek niet geïmplementeerd en Breadth-first heuristiek wel.



Figuur 23 Pad van een Depth-first heuristiek door een zoekboom

5.3. Hill Climbing

Hill Climbing zoekt vanuit een startoplossing naar buuroplossingen zonder gebruikt te maken van extra informatie (Stackoverflow, 2011). Als een buuroplossing beter is dan de huidige oplossing wordt deze geaccepteerd. Als Hill Climbing geen betere buuroplossing kan vinden stopt het algoritme (Jacobson, 2012), zoals te zien is in figuur 24. In de linkerhelft begint Hill Climbing bij de rode pijl en de grijze pijl wijst aan welke kant op het algoritme de buuroplossingen accepteert. In de rechterhelft is het algoritme bij een lokaal optimum gekomen, het algoritme vindt dan geen betere buuroplossingen meer en stopt.

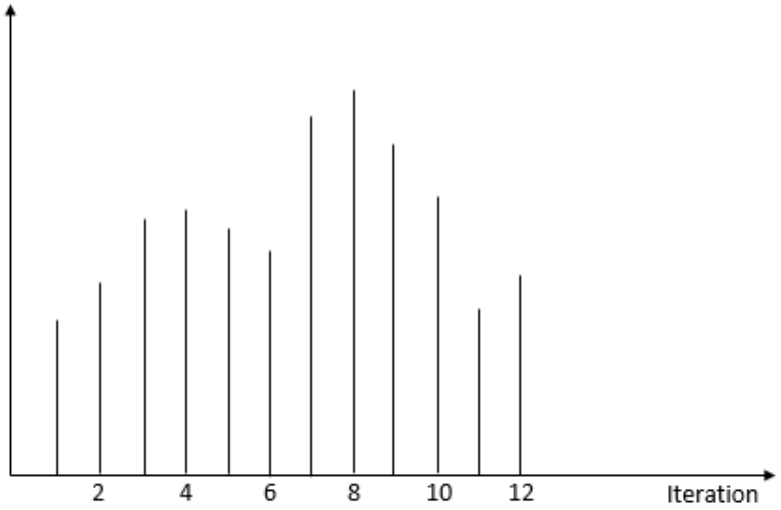


Figuur 24 Werking Hill Climbing

Bij Hill Climbing wordt geen gebruikt gemaakt van extra informatie om te optimaliseren. Dus kan de volgorde van belangrijkheid van de componenten niet gebruikt worden. Hierdoor is het mogelijk dat dit algoritme in een lokaal optimum blijft, zoals in rechterhelft van figuur 24 te zien is. Het algoritme kan een tak onderzocht hebben voor de oplossing. Als uiteindelijk geen betere oplossing gevonden wordt in die tak dan stopt het algoritme. Het algoritme onderzoekt geen andere tak meer. Bij dit algoritme is de kans dus groot dat een lokaal optimum gevonden. Hierbij is het niet gegarandeerd dat de gevonden lokale optimum een goede oplossing is. Daarom wordt dit algoritme niet geïmplementeerd.

5.4. Tabu Search

Net als Hill Climbing zoekt Tabu Search naar betere buuroplossingen. Wanneer er geen betere buuroplossingen zijn, kiest Tabu Search een buuroplossing die het minst slecht is. Hierdoor volgt Tabu Search een patroon zoals te zien is in figuur 25.



Figuur 25 Patroon Tabu Search

Het gevaar hierbij is dat als het algoritme uit een lokaal optimum stapt, het weer precies dezelfde pad terug gaat volgen. Om dit te voorkomen heeft Tabu Search een tabu (verboden) lijst met buuroplossingen die recent bezocht zijn. Bij een volgende iteratie mogen deze buuroplossingen niet bezocht worden. De grootte van de tabu lijst hangt af van de grootte van het probleem. Het algoritme stopt wanneer een iteratie geen andere buuroplossingen heeft dan die in de tabu lijst staan, na een gegeven aantal iteraties, na een bepaalde rekentijd of hij stopt na een gegeven aantal iteraties waarvoor er geen verbetering meer optreedt. Steeds onthoudt Tabu Search de beste oplossing. Eindigt het algoritme in een iteratie die niet de beste gevonden oplossing is, dan geeft Tabu Search de beste oplossing terug als oplossing. De oplossing die met dit algoritme gegeven wordt kan de optimale oplossing zijn, maar dit is niet gegarandeerd (Hillier & Lieberman, 2015, blz. 625-633).
Dit algoritme lijkt geschikt om toe te passen, omdat het uit een slecht lokaal optimum kan stappen door de tabu lijst. Er is echter niet gegarandeerd dat de gevonden lokale optimum een goede oplossing is in de buurt van het globale optimum. Daarom wordt dit algoritme niet geïmplementeerd.

5.5. Simulated Annealing

Simulated Annealing is oorspronkelijk geïnspireerd door het proces van gloeien in de metaalbewerking. Gloeien omvat het verwarmen en koelen van een materiaal om de fysische eigenschappen te veranderen door de veranderingen in de interne structuur. Naarmate het metaal afkoelt is de nieuwe structuur bereikt, waardoor het metaal zijn nieuw verkregen eigenschappen behoudt. Bij Simulated Annealing wordt er een temperatuurvariabele bijhouden om dit verwarmingsproces te simuleren. Deze temperatuurvariabele wordt hoog ingesteld en vervolgens langzaam ‘afgekoeld’ terwijl het algoritme loopt. Als de temperatuurvariabele hoog is, mag het algoritme met een hogere frequentie oplossingen accepteren die slechter zijn dan de huidige oplossing. Dit geeft het algoritme de mogelijkheid om uit een lokaal optimum te springen. Naarmate de temperatuur wordt verlaagd, neemt ook de kans af om slechtere oplossingen te accepteren, waardoor het algoritme geleidelijk kan focussen op een gebied van de zoekruimte waarin hopelijk een bijna optimale oplossing kan worden gevonden. Dit geleidelijk ‘koel’-proces maakt het gesimuleerde gloei-algoritme opmerkelijk effectief bij het vinden van een nagenoeg optimale oplossing wanneer het gaat om grote problemen die talrijke lokale optima bevatten.

Het voordeel van Simulated Annealing ten opzichte van Hill Climbing (paragraaf 5.3) is dat Simulated Annealing beter is in het vermijden van het vastlopen in lokale optima en in het vinden van een globaal optimum bij benadering. Dit komt doordat Simulated Annealing met een acceptatiefunctie werkt waarin de temperatuurvariabele is verwerkt om uit een lokaal optimum te kunnen stappen (Jacobson, 2012). Deze acceptatiefunctie voor een maximalisatieprobleem is:

$$acceptatiekans = e^{\left(\frac{Z_n - Z_c}{T}\right)} \quad (13)$$

Hierbij is Z_c de uitkomst van de huidige oplossing, Z_n de uitkomst van de gevonden buuroplossing en *T* de temperatuurvariabele. De acceptatiefunctie voor een minimalisatieprobleem is hetzelfde, echter worden Z_n en Z_c omgedraaid.
Als de gevonden buuroplossing beter is dan de huidige oplossing wordt deze buuroplossing altijd geaccepteerd. Is de gevonden buuroplossing slechter dan de huidige oplossing, dan wordt met een random getal tussen 0 en 1 en de acceptatiefunctie bepaald of de buuroplossing geaccepteerd wordt of niet. Als $willekeurig\ getal < acceptatiekans$, dan wordt de slechtere buuroplossing geaccepteerd. Het voordeel van een willekeurig getal in het algoritme is dat er meer flexibiliteit is om in een ander deel van de zoekruimte te komen om een beter oplossing te vinden. Dit voordeel heeft Tabu Search niet.
Wanneer het gewenste aantal iteraties is uitgevoerd met de kleinste waarde van *T* in een temperatuurschema of als geen van de buuroplossingen wordt geaccepteerd, stopt Simulated Annealing (Hillier & Lieberman, 2015, blz. 637-639).
Dit algoritme is goed implementeerbaar, het geeft niet gegarandeerd het globale optimum, maar wel een nagenoeg optimale oplossing.

5.6. Greedy algoritme

Greedy is een algoritme dat stuk voor stuk een oplossing opbouwt (GeeksforGeeks, z.d.). Dit algoritme maakt altijd de keuze die op dat moment het best is, dat wil zeggen een lokaal optimale keus in de hoop dat hiermee het globale optimum bereikt wordt (Sharma, 2016). Dit algoritme is daarom het best toe te passen op problemen waarbij het kiezen van een lokaal optimum ook leidt tot een globaal optimum (GeeksforGeeks, z.d.). In tegenstelling tot Hill Climbing kan dit algoritme gebruik maken van extra informatie. Dit algoritme kan dus gebruik maken van de volgorde van belangrijkheid van de componenten. Het algoritme kan dan altijd de meest belangrijke component op dat ogenblik optimaliseren. Hierdoor wordt verwacht dat dit algoritme het globale optimum bereikt. Daarom zal dit algoritme geïmplementeerd worden.

5.7. Conclusie

De optimalisatiemethoden die uiteindelijk geïmplementeerd worden zijn:

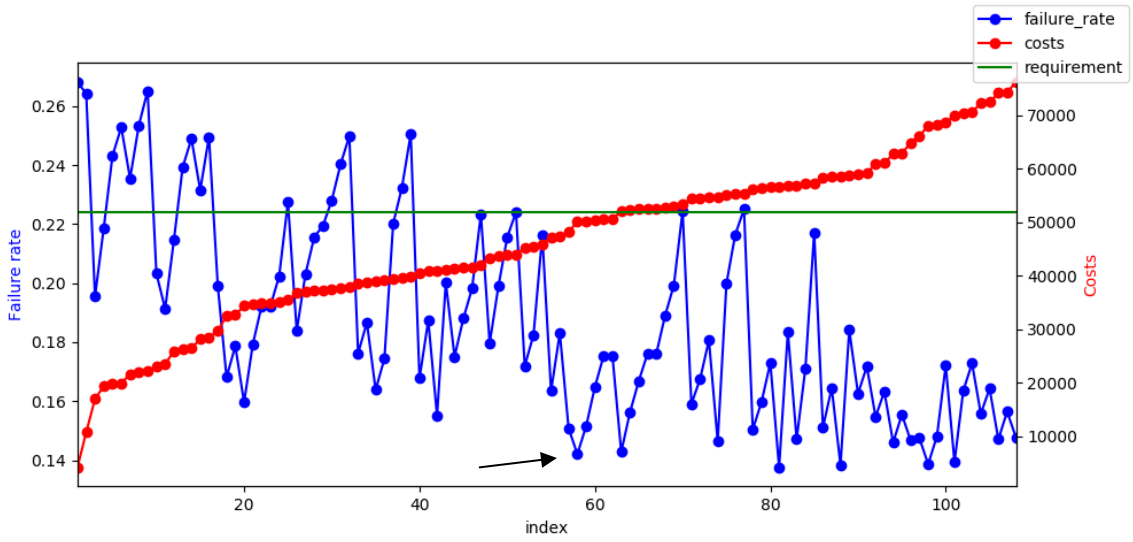
1. Brute-force
2. Breadth-first heuristiek
3. Greedy algoritme
4. Simulated Annealing

6. Implementatie optimalisatiemethoden

In dit hoofdstuk wordt uitgelegd hoe de eerste drie optimalisatiemethoden uit de vorige paragraaf 5.7 geïmplementeerd worden. Deze eerste drie optimalisatiemethoden zijn Brute-force (paragraaf 6.1), het Greedy algoritme (paragraaf 6.2) en de Breadth-first heuristiek (paragraaf 6.3). De vierde optimalisatiemethode uit paragraaf 5.7 is niet meer geïmplementeerd vanwege de beschikbare tijd.

6.1. Implementatie Brute-force

Bij Brute-force worden van alle mogelijke oplossingen die er zijn de faalkans en de onderhoudskosten berekend. De faalkans en de onderhoudskosten van een oplossing worden berekend aan de hand van de opgestelde formule voor de faalkans en de opgestelde formule voor de onderhoudskosten van een systeem. Als de klant de faalkans wil optimaliseren bij een bepaald budget worden alle oplossingen waarvan de onderhoudskosten hoger zijn dan het budget verwijderd. Van de overgebleven oplossingen wordt oplossing gekozen met de laagste faalkans. Als de klant de onderhoudskosten wil optimaliseren bij een maximale faalkans worden alle oplossingen waarvan de faalkans hoger is dan de maximale toegestane faalkans verwijderd. Van de overgebleven oplossingen wordt de oplossing gekozen met de laagste onderhoudskosten. In figuur 26 staat de grafiek met op de x-as de oplossingen, op de rechter y-as de onderhoudskosten en op de linker y-as de faalkans. De groene lijn is de wens van de klant. In dit geval is dat een bepaald budget, waarvoor de faalkans geoptimaliseerd moet worden. Alle oplossingen waarbij de onderhoudskosten hoger zijn dan de groene lijn, worden verwijderd. In dit geval de oplossing aan de rechterkant van de grafiek. Vervolgens wordt uit de linkerkant van de grafiek, waar de onderhoudskosten lager zijn dan de groene lijn de oplossing gekozen met de laagste faalkans. De oplossing staat aangegeven met de zwarte pijl.



Figuur 26 Voorbeeld Brute-force

6.2. Implementatie Greedy algoritme

Het Greedy algoritme is het algoritme dat stuk voor stuk een oplossing opbouwt. Hierbij kiest het algoritme een lokaal optimum in de hoop dat hiermee het globale optimum bereikt wordt. Als eerste wordt het Greedy algoritme geïmplementeerd voor het optimaliseren van de faalkans bij een bepaald budget in paragraaf 6.2.1. Vervolgens wordt dit algoritme geïmplementeerd voor het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans in paragraaf 6.2.2.

6.2.1. Greedy algoritme voor optimaliseren van faalkans bij bepaald budget

Het Greedy algoritme kan gebruik maken van extra informatie. Daarom gebruikt dit algoritme de volgorde van belangrijkheid van de componenten (paragraaf 4.5). Het algoritme maakt ook gebruik van een lijst waarin de componenten staan die geoptimaliseerd zijn. Het algoritme begint bij de component die het meest belangrijk is, waar de investering per euro de grootste daling van de faalkans oplevert. Nadat dit component geoptimaliseerd is gaat het algoritme naar de component die daarna het meest belangrijk is, enzovoorts. Voor elke component neemt het algoritme de volgende stappen door:

1. De component bevindt zich niet in de lijst met geoptimaliseerde componenten:
    a. Voor elke mogelijk onderhoudsstrategie van de component de bijbehorende faalkans en onderhoudskosten voor het hele systeem berekenen;
    b. Alleen de onderhoudsstrategieën onthouden waarbij de onderhoudskosten lager zijn dan de maximale toegestane kosten:
        i. Als er meer dan één toegestane onderhoudsstrategie is wordt de strategie gekozen waarbij de faalkans het laagst is;
        ii. Als er één toegestane onderhoudsstrategie is, is deze strategie de beste oplossing;
        iii. Als er geen toegestane onderhoudsstrategie is wordt dit component niet geoptimaliseerd en blijft de onderhoudsstrategie voor de component hetzelfde.
    c. De component toevoegen aan de lijst geoptimaliseerde componenten;
    d. Opnieuw kritieke componenten bepalen waarbij de onderhoudsstrategie van de component is aangepast naar de beste onderhoudsstrategie.
2. De component bevindt zich wel in de lijst met geoptimaliseerde component:
    a. Volgende belangrijke component optimaliseren, beginnen bij stap 1.
3. Als de beste strategie hetzelfde is als de strategie die de component al had voor het optimaliseren stopt het algoritme.

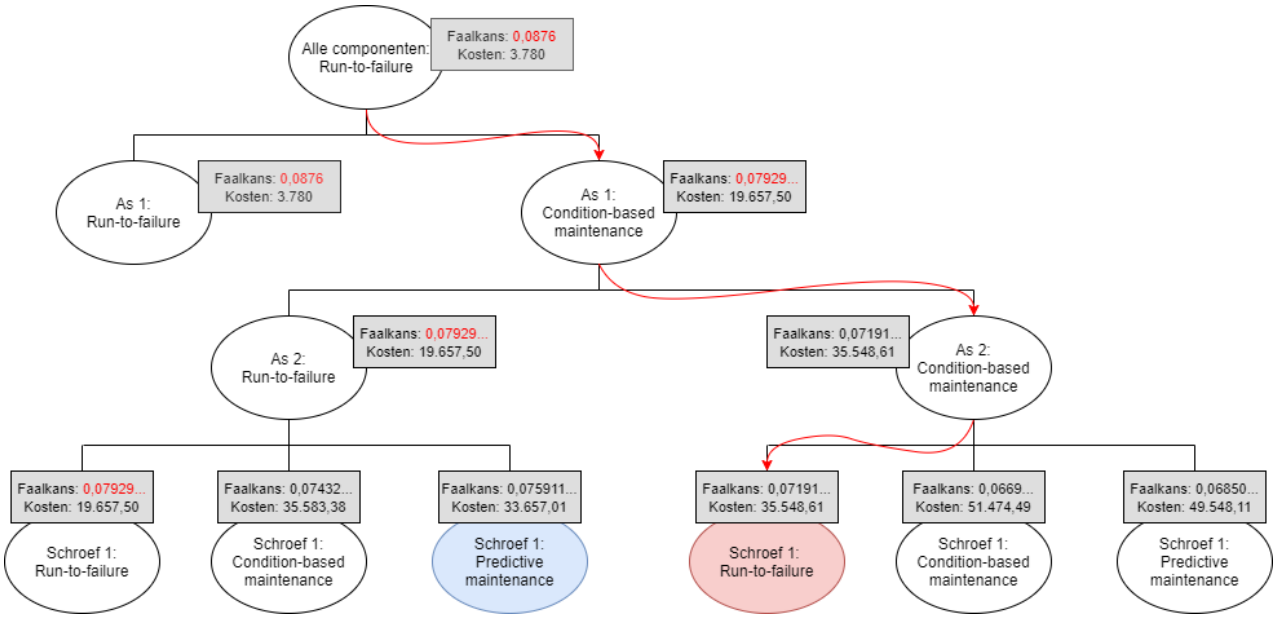
Met het toepassen van dit algoritme voor de optimalisatie van de faalkans bij een bepaald budget wordt verwacht dat de optimale oplossing wordt bereikt, omdat voor elke component de onderhoudsstrategie gekozen wordt waarbij de faalkans het meest daalt. De onderhoudskosten moeten hierbij altijd lager zijn dan het door de klant aangegeven budget. Tevens zijn de componenten gesorteerd van hoogste belangrijkheidsratio naar laagste belangrijkheidsratio. Dus de component waarbij de investering per euro de meeste daling in de faalkans oplevert wordt als eerst geoptimaliseerd. Door een aanpassing in onderhoudsstrategie veranderd de MTBF van de component en daarbij de faalkans, zoals uitgelegd in hoofdstuk 3.1. Hierdoor kan de belangrijkheidsratio (daling faalkans per geïnvesteerde euro) van de componenten die serie of parallel verbonden zijn met de geoptimaliseerde component ook veranderen. Daarom wordt na elke optimalisatie opnieuw de volgorde van de belangrijkheid van de componenten bepaald om te voorkomen dat het algoritme in een lokaal optimum terecht komt. Aangezien dit algoritme altijd voor elke component de op dat moment beste keuze maakt wordt verwacht dat het globale optimum bereikt wordt.

6.2.2. Greedy algoritme voor optimaliseren van kosten bij maximale toegestane faalkans

Opnieuw maakt het algoritme gebruik van de volgorde van belangrijkheid van de componenten en een lijst waarin de componenten staan die geoptimaliseerd zijn. Het algoritme begint bij de component die het meest belangrijk is, waar de investering per euro de grootste daling van de faalkans oplevert. Nadat dit component geoptimaliseerd is gaat het algoritme naar de component die daarna het meest belangrijk is, enzovoorts. Voor elke component neemt het algoritme de volgende stappen door:

1. De component bevindt zich niet in de lijst met geoptimaliseerde componenten:
    a. Voor elke mogelijk onderhoudsstrategie van de component de bijbehorende faalkans en onderhoudskosten voor het hele systeem berekenen;
    b. Als er onderhoudsstrategieën zijn waarbij de faalkans toegestaan is (lager of gelijk aan wat de klant wil), alleen deze oplossingen onthouden:
        i. Als er meer dan één toegestane onderhoudsstrategie is wordt de strategie gekozen waarbij de onderhoudskosten het laagst zijn;
        ii. Als er één toegestane onderhoudsstrategie is, is deze de beste oplossing.
    c. Als alle oplossingen een hogere faalkans hebben dan toegestaan is wordt de oplossing gekozen waarbij de faalkans het meest is gedaald;
    d. De component toevoegen aan de lijst geoptimaliseerde componenten;
    e. Opnieuw kritieke componenten bepalen waarbij de onderhoudsstrategie van de component is aangepast naar de beste onderhoudsstrategie.
2. De component bevindt zich wel in de lijst met geoptimaliseerde component:
    a. Volgende belangrijke component optimaliseren, beginnen bij stap 1.
3. Als de beste strategie dezelfde is als de strategie die de component al had voor het optimaliseren stopt het algoritme.

Als het algoritme voor deze optimalisatie toegepast wordt, wordt niet altijd de optimale oplossing bereikt. Dit komt doordat de eerste oplossing waarbij de faalkans toegestaan is, gekozen wordt als beste oplossing op dat moment, waardoor dit algoritme deze oplossing verder onderzoekt. Terwijl als er verder geoptimaliseerd wordt met een oplossing waarbij de faalkans nog niet toegestaan is er een betere oplossing bereikt kan worden. Dit wordt aangetoond in het voorbeeld in figuur 27. De maximale toegestane faalkans van dit voorbeeld is 0,079. In dit voorbeeld is te zien dat het algoritme kiest voor een lokaal optimum (de rode oplossing) terwijl het globale optimum niet bereikt wordt (de blauwe oplossing). Dit komt doordat bij de component as 2 de onderhoudsstrategie gekozen wordt die op dat moment het beste is. In dit geval is dat condition-based maintenance. Bij de volgende component, de schroef, worden opnieuw voor alle strategieën de faalkans en de onderhoudskosten berekend. Alle drie de strategieën hebben een faalkans die toegestaan is, daarom wordt van deze strategieën degene gekozen die de laagste onderhoudskosten heeft. In dit geval is dat de strategie run-to-failure. Het globale optimum wordt met het Greedy algoritme niet bereikt. Dit wil echter niet zeggen dat het Greedy algoritme niet geschikt is. Het kan zijn dat het Greedy algoritme, als het globale optimum niet gevonden wordt, wel een goede oplossing vindt.



Figuur 27 Greedy algoritme voor optimaliseren van kosten bij maximale toegestane faalkans

6.3. Implementatie Breadth-first heuristiek

De Breadth-first heuristiek is een pruning heuristiek. Deze heuristiek zoekt per niveau van de zoekboom de verschillende knopen af. Als een knoop minder goede oplossing bevat dan andere takken kapt de heuristiek deze tak af. Hierdoor wordt de zoekruimte voor een oplossing beperkt. Deze heuristiek wordt alleen geïmplementeerd voor het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans (paragraaf 6.3.1). Bij het implementeren van het Greedy algoritme voor het optimaliseren van de faalkans voor een bepaald budget is aangegeven dat de verwachting is dat het globale optimum gevonden zal worden. Daarom wordt voor deze optimalisatiemogelijkheid de Breadth-first heuristiek niet meer geïmplementeerd. Bij het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans wordt niet altijd het globale optimum gevonden, daarom wordt voor deze optimalisatiemogelijkheid de Breadth-first heuristiek wel geïmplementeerd.

6.3.1. Breadth-first voor optimaliseren van kosten bij maximale toegestane faalkans

Als eerste bepaalt de heuristiek de volgorde van meest kritieke component naar minst kritieke component. De meeste kritieke component is de component met de hoogste ratio. Daarna begint het algoritme met optimaliseren. Zoals figuur 28 aangeeft begint de heuristiek vanuit de startoplossing dat alle component als onderhoudsstrategie run-to-failure hebben. De heuristiek heeft drie variabelen die helpen bij de optimalisatie:

1. De beste oplossing tot nu toe;
2. De bijbehorende faalkans van het hele systeem voor deze oplossing;
3. De bijbehorende onderhoudskosten van het hele systeem voor deze oplossing.

Aan het begin van de optimalisatie is er nog geen beste oplossing. De onderhoudskosten van de ‘beste oplossing’ worden daarom heel hoog gezet en de faalkans van de beste oplossingen op 1.0 voordat de optimalisatie begint. Vervolgens gaat de heuristiek alle componenten af van meest de belangrijke component naar de minst belangrijke component, hierbij wordt per component de volgende stappen ondernomen:

1. Voor het optimaliseren moet de volgende formule kloppen:

$$\frac{(F_{tak} - F_{max})}{r} + C_{tak} \leq C_{best}$$

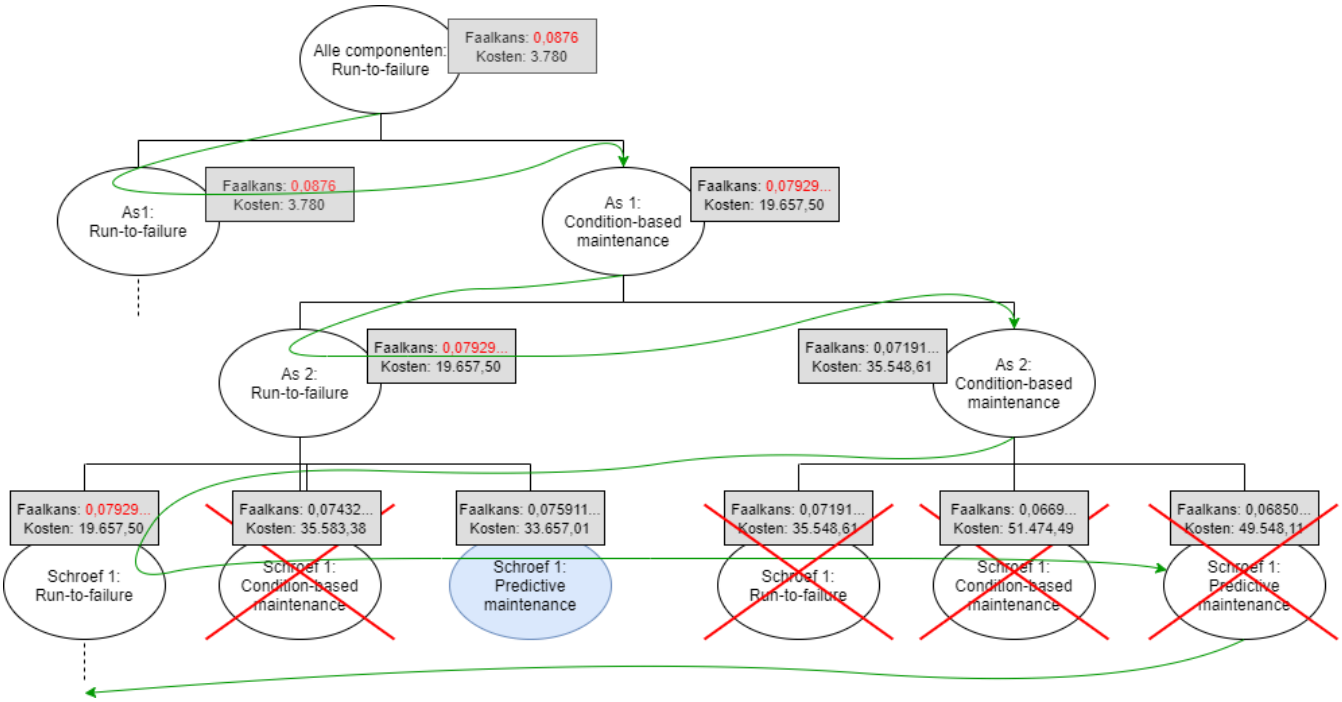
Waarbij F_{tak} de faalkans van de tak is waarbij de component nog niet geoptimaliseerd is. De door de klant toegestane maximale faalkans is F_{max}. De belangrijkheidsratio van de component is r. De onderhoudskosten van de tak waarbij de componenten nog niet geoptimaliseerd is, is C_{tak} en C_{best} zijn de onderhoudskosten van de beste oplossing tot nu toe.

Wanneer deze formule niet klopt, betekent dit dat het reduceren van de faalkans tot de maximale toegestane faalkans met de belangrijkheidsratio van de component tot hogere onderhoudskosten leidt dan de onderhoudskosten van de beste oplossing tot nu toe. Omdat de belangrijkheidsratio van andere componenten die na dit component geoptimaliseerd worden altijd lager zullen zijn, wordt verwacht dat deze tak geen betere oplossingen zal hebben dan een andere tak. De daling van de faalkans per euro wordt voor de volgende componenten alleen maar lager. De heuristiek kapt daarom takken waarvoor deze formule niet klopt af, zodat de zoekruimte beperkt wordt. De stappen 2 en 3 worden dan overgeslagen;

2. Voor elke mogelijk onderhoudsstrategie van de component worden de bijbehorende onderhoudskosten en faalkans voor het hele systeem berekend;
3. Als er oplossingen zijn waarbij de faalkans lager is dan de maximale toegestane faalkans wordt van deze oplossingen de oplossing met de laagste onderhoudskosten onthouden als de beste oplossing tot nu toe.

Als er sprake is van één van de volgende situaties bij een oplossing van een zoekboom wordt de tak van deze oplossing niet uitgebreid en verder onderzocht:

1. De oplossing heeft hogere onderhoudskosten dan de onderhoudskosten van de beste oplossing tot nu toe;
2. De oplossing heeft een lagere faalkans dan de maximale toegestane faalkans, maar de onderhoudskosten zijn hoger dan bij de beste oplossing tot nu toe;
3. De oplossing is de beste oplossing tot nu toe.



Figuur 27 Voorbeeld implementatie Breadth-first heuristiek

In figuur 28 staat aangegeven hoe deze heuristiek bij een goede oplossing komt en andere takken afkapt. Dit voorbeeld is hetzelfde voorbeeld als bij de uitleg van het Greedy algoritme (figuur 26). De faalkans mag hierbij maximaal 0,079 zijn. Als de heuristiek bij de oplossing komt waarbij de onderhoudsstrategie van as 2 condition-based wordt, wordt deze oplossing onthouden als beste oplossing. Deze oplossing heeft namelijk een faalkans die lager is dan 0,079. De onderhoudskosten van de beste oplossing zijn dus 35.548,61 euro. Vervolgens onderzoekt de heuristiek de takken onder de oplossing waarbij de onderhoudsstrategie van as 2 run-to-failure is. Deze oplossing heeft namelijk geen faalkans die lager is dan 0,079 en de onderhoudskosten zijn niet hoger zijn dan de onderhoudskosten van de beste oplossing tot nu toe. Bij schroef 1 zijn er drie onderhoudsstrategieën mogelijk. Bij run-to-failure is de faalkans nog niet onder 0,079 en de onderhoudskosten zijn nog steeds lager dan de onderhoudskosten van de beste oplossing. Bij de andere twee onderhoudsstrategieën is de faalkans lager dan 0,079. Echter bij de onderhoudsstrategie condition-based zijn de onderhoudskosten hoger dan de onderhoudskosten van de beste oplossing. De beste oplossing blijft dus hetzelfde en de tak onder de oplossing met condition-based als onderhoudsstrategie voor schroef 1 wordt niet meer onderzocht. De oplossing die onder deze tak volgen kunnen nooit lagere onderhoudskosten hebben, omdat telkens een niveau verder in de zoekboom een ander component geoptimaliseerd wordt waardoor de onderhoudskosten steeds hoger worden. Het kan ook zijn dat de onderhoudsstrategie run-to-failure blijft, maar dan wordt de faalkans niet gereduceerd. Door deze tak staat dus een kruis. Deze tak wordt niet meer verder onderzocht. De oplossing waarbij de onderhoudsstrategie voor schroef 1 predictive maintenance is, heeft wel lagere onderhoudskosten dan de onderhoudskosten van de beste oplossing tot nu toe. Dit wordt dus de nieuwe beste oplossing (de blauwe oplossing). Vervolgens gaat de heuristiek de takken onderzoeken onder de oplossing waarbij de onderhoudsstrategie van as 2 condition-based is. Hierbij wordt weer schroef 1 onderzocht waarbij er dus drie onderhoudsstrategieën zijn. Deze drie oplossingen hebben elk een faalkans die lager is dan 0,079. Echter, zijn voor al deze drie oplossingen de onderhoudskosten hoger dan de onderhoudskosten van de beste oplossingen. Deze takken wordt dus niet meer verder onderzocht. Op deze wijze onderzoekt de heuristiek elke tak van de zoekboom totdat uitgesloten is dat een tak niet een betere oplossing bevat. Daarom wordt verwacht dat deze heuristiek het globale optimum vindt.

7. Resultaten

Om te testen of de optimalisatiemethoden die geïmplementeerd zijn in hoofdstuk 6 daadwerkelijk het globale optimum of een goede oplossing opleveren zijn er verschillende testsystemen bedacht. Deze testsystemen verschillen qua grootte en hoe ingewikkeld het systeem is. Zo bestaat één testsysteem alleen uit een seriesysteem. Maar een ander testsysteem bestaat uit een seriesysteem en een parallelsysteem. In tabel 3 staan de testsystemen weergegeven. Voor elk testsysteem staat aangegeven hoeveel componenten het testsysteem bevat en hoeveel mogelijke oplossingen er zijn. De eerste 5 testsystemen zijn geen realistische systemen uit een schip, maar zelf bedachte testsystemen. De laatste twee testsystemen zijn de systemen voor twee KPI's van het schip de ASD 3212 uit bijlage 2.

Tabel 3 De testsystemen

Voorbeeld	Systeem	Aantal componenten	Aantal mogelijke oplossingen
1	Serie	5	108
2	Parallel	6	324
3	Serie en parallel	7	576
4	Half serie-/parallel	7	648
5	Groot systeem	10	15.552
6	ASD 3212 (KPI: externe brandbestrijding)	12	1.048.576
7	ASD 3212 (KPI: varen/trekkracht)	14	4.194.304

Er zijn twee optimalisatie mogelijkheden, namelijk:

1. Optimaliseren van de faalkans bij een door de klant aangegeven budget;
2. Optimaliseren van de onderhoudskosten bij een door de klant aangegeven maximale toegestane faalkans.

Voor de eerste optimalisatie zijn de optimalisatiemethoden Brute-force en het Greedy algoritme geïmplementeerd. Voor de tweede optimalisatie zijn de optimalisatiemethoden Brute-force, het Greedy algoritme en de Breadth-first heuristiek geïmplementeerd. Voor elke van deze optimalisatie mogelijkheden wordt getest of de geïmplementeerde optimalisatiemethoden daadwerkelijk het globale optimum vinden. Dit wordt getest aan de hand van de oplossing van Brute-force. Brute-force vindt namelijk altijd het globale optimum, omdat Brute-force alle mogelijke oplossingen berekend en op basis daarvan de beste oplossing kiest. Daarnaast is de rekentijd voor de algoritmes bepaald om te onderzoeken welk algoritme minder rekentijd nodig heeft en of de rekentijd korter is dan de tijd die nodig is om RCM uit te voeren voor een schip. In paragraaf 7.1 staan de testresultaten voor de eerste optimalisatiemogelijkheid en in paragraaf 7.2 staan de testresultaten voor de tweede optimalisatiemogelijkheid.

7.1. Testresultaten voor optimaliseren van faalkans bij bepaald budget

Voor elk voorbeeld is de faalkans geoptimaliseerd bij een door de klant aangegeven budget. Deze optimalisatie is uitgevoerd door middel van Brute-force en het Greedy-algoritme. In tabel 4 staan de testresultaten per voorbeeld weergegeven. Voor de eerste vijf voorbeelden zijn drie optimalisaties uitgevoerd, voor de laatste twee voorbeelden één optimalisatie. De resultaten met de onderhoudsstrategie per component staan in bijlage 4. Bij alle twee de optimalisatiemethoden staan de faalkans en de onderhoudskosten van de oplossing. De meest rechtse kolom geeft aan wat het door de klant aangegeven budget was waarvoor de faalkans is geoptimaliseerd.

Tabel 4 Testresultaten optimalisatiemogelijkheid 1

Voorbeeld	Brute-force		Greedy algoritme		Requirement
	Faalkans	**Kosten**	**Faalkans**	**Kosten**	**Kosten**
1	0,155239	40.938,3	0,155239	40.938,30	42.000
	0,142082	50.104,3	0,142082	50.104,35	52.000
	0,137640	56.658,3	0,137640	56.658,30	62.000
2	0,068502	49.548,12	0,068502	49.548,12	50.000
	0,066904	51.474,5	0,066904	51.474,49	62.000
	0,066820	67.117,25	0,066820	67.117,25	72.000
3	0,261468	50.135,19	0,261852	50.142,02	52.000
	0,256881	52.051,85	0,256881	52.051,85	60.000
	0,245339	67.849,63	0,245339	67.849,63	68.000
4	0,266461	34.230,20	0,266461	34.230,20	47.000
	0,253592	48.996,9	0,253592	48.996,86	61.000
	0,241999	64.794,64	0,241999	64.794,64	71.000
5	0,101655	67.958,43	0,101655	67.958,43	75.000
	0,096776	80.360,04	0,096776	80.360,04	85.000
	0,092924	94.473,38	0,092924	94.473,38	95.000
6	0,005995	414.303,86	0,005995	414.303,86	450.000
7	0,006807	298.132	0,006890	285.241,45	300.000

Uit bovenstaande tabel kan geconcludeerd worden dat het Greedy algoritme niet altijd het globale optimum vindt. De oplossingen van Brute-force zijn altijd optimaal, omdat dit algoritme de beste oplossingen uit alle mogelijke oplossingen bepaald. Het Greedy algoritme geeft niet altijd dezelfde oplossing als Brute-force (de rode oplossingen in tabel 4). Dus vindt het Greedy algoritme niet altijd het globale optimum. Echter is al eerder aangegeven dat de faalkans berekend wordt met de MTBF. De MTBF is een gemiddelde van een verdeling. Daardoor is de faalkans ook een gemiddelde van een verdeling. Het globale optimum hoeft niet gevonden te worden, een goede oplossing volstaat ook. De faalkansen van de oplossingen waarbij het Greedy algoritme niet dezelfde oplossing heeft als Brute-force liggen wel dicht bij de faalkans van de oplossing van Brute-force. Hieruit kan geconcludeerd worden dat het Greedy algoritme wel een goede oplossing geeft, dus een goede optimalisatiemethode is voor het optimaliseren van de faalkans bij een bepaald budget.

In tabel 5 staat per optimalisatiemethode en per voorbeeld aangegeven wat de rekentijd in seconden is om een oplossing te bepalen van de optimalisaties uit tabel 4. De rekentijd van Brute-force voor voorbeeld 7 is getest op een andere computer dan de andere rekentijden.

Tabel 5 Rekentijden optimalisatiemethoden per voorbeeld

Voorbeeld	Rekentijd Brute-force (sec.)	Rekentijd Greedy algoritme (sec.)	Requirement
1	0,194320	0,277056	42.000
	0.165111	0.196141	52.000
	0,172246	0,205007	62.000
2	0,603029	0,297376	50.000
	0,685833	0,260694	62.000
	0,551979	0,264305	72.000
3	1,098585	0,339484	52.000
	1,305909	0,346514	69.000
	1,445358	0,364220	83.000
4	1,458808	0,530644	48.000
	1,236083	0,347889	61.000
	1,158089	0,354409	71.000
5	396,173659	1,183008	75.000
	391,009248	1,268597	85.000
	384,368905	1,257766	95.000
6	3338 ongeveer	2,142856	450.000
7	14.400 ongeveer	2,857933	300.000

Uit bovenstaande tabel kan geconcludeerd worden dat het Greedy algoritme een kortere rekentijd heeft dan Brute-force. Bij Brute-force is het effect op de rekentijd groot wanneer het voorbeeld meer componenten en oplossingen heeft. Het eerste voorbeeld heeft 108 oplossingen. De rekentijd is ongeveer een halve seconde. Als echter het optimalisatieprobleem ongeveer vier miljoen oplossing heeft, is de rekentijd ongeveer 14.400 seconden. Bij het Greedy algoritme is de rekentijd voor voorbeeld 7 duidelijk lager. De rekentijd is voor het Greedy algorimte dan ongeveer twee en een halve seconde. Dit maakt het Greedy algoritme geschikter voor dit optimalisatieprobleem.

7.2. Testresultaten voor optimaliseren van kosten bij maximale toegestane faalkans

Voor elk voorbeeld zijn de onderhoudskosten geoptimaliseerd bij een door de klant aangegeven maximale faalkans. De optimalisatie is uitgevoerd door middel van Brute-force, het Greedy algoritme en de Breadth-first heuristiek. In tabel 6 staan de testresultaten per voorbeeld weergeven. Voor de eerste vijf voorbeelden zijn twee optimalisaties uitgevoerd en voor de laatste twee voorbeelden één optimalisatie. De resultaten met de onderhoudsstrategie per component staan in bijlage 5. Bij alle drie de optimalisatiemethoden staan de faalkans en de onderhoudskosten van de oplossing. De meest rechtse kolom geeft aan wat de door de klant aangegeven maximale toegestane faalkans is.

Tabel 6 Testresultaten optimalisatiemogelijkheid 2

Voorbeeld	Brute-force		Greedy algoritme		Breadth-first heuristiek		Requirement
	Faalkans	**Kosten**	**Faalkans**	**Kosten**	**Faalkans**	**Kosten**	**Faalkans**
1	0,142082	50.104,35	0,142082	50.104,35	0,142082	50.104,35	0,145
	0,137640	56.658,30	0,137640	56.658,30	0,137640	56.658,30	0,140
2	0,075911	33.657,01	0,071917	35.548,61	0,075911	33.657,01	0,079
	0,068502	49.548,12	0,068502	49.548,12	0,068502	49.548,12	0,069
3	0,256881	52.051,9	0.256881	52.051,85	0,256881	52.051,85	0,261
	0,238461	83.727,13	0,238461	83.727,13	0,238461	83.727,13	0,241
4	0,253592	48.996,86	0.253592	48.996,86	0,253592	48.996,86	0,261
	0,232975	79.634,64	0,232975	79.634,64	0,232975	79.634,64	0,241
5	0,106629	65.478,28	0,101655	67.958,43	0,106629	65.478,28	0,110
	0,089586	108.473,89	0.089586	108.472,89	0.089586	108.472,89	0,090
6	0,005997	350834,63	0.005995	400.430,09	0,005995	400.430,09	0,006
7	0,006953	250.087	0.006985	258.587,66	0.006985	258.587,66	0,007

Uit bovenstaande tabel kan geconcludeerd worden dat het Greedy algoritme inderdaad niet altijd het globale optimum vindt en een lokaal minimum vindt (de rode oplossingen). Echter vindt de Breadth-first heuristiek ook niet altijd het globale optimum. De oplossingen die door het Greedy algoritme en de Breadth-first heuristiek gevonden worden liggen wel dicht bij het globale optimum. Er kan dus geconcludeerd worden dat beide optimalisatiemethoden een goede oplossing vinden. Beide methoden zouden daarom geschikt zijn voor het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans.

In tabel 7 staat voor ieder voorbeeld aangegeven wat de rekentijd is in seconden voor het bepalen van de oplossing van de optimalisaties uit tabel 6. De rekentijd van Brute-force voor voorbeeld 7 is getest op een andere computer dan de andere rekentijden.

Tabel 7 Rekentijden optimalisatiemethoden per voorbeeld

Voorbeeld	Rekentijd Brute-force (sec.)	Rekentijd Greedy algoritme (sec.)	Rekentijd Breadth-first heuristiek (sec.)	Requirement
1	0,175633	0,177921	0,103568	0,145
	0,162279	0,193711	0,111208	0,140
2	0,548686	0,704873	0,462180	0,079
	0,909169	0,339417	0,286539	0,069
3	2,155074	0,416010	0,614535	0,261
	1,061724	0,345325	0,422233	0,241
4	1,201762	0,346160	0,184513	0,261
	1,758856	0,392395	0,166532	0,241
5	400,004063	1,207651	0,413732	0,110
	404,029991	1,335514	0,831657	0,090
6	3.426 ongeveer	1,333509	38,110350	0,006
7	10.800 ongeveer	1,640033	81,420471	0,007

Uit bovenstaande tabel kan geconcludeerd worden dat Brute-force de langste rekentijd heeft. Voor de realistische voorbeelden loopt de rekentijd op tot één of zelfs vier uur, dat maakt Brute-force niet geschikt voor deze optimalisatie. Het Greedy algoritme heeft een kortere rekentijd dan de Breadth-first heuristiek. Daarom is het Greedy algoritme het meest geschikt voor deze optimalisatie.

8. Conclusie

De hoofdvraag van dit optimalisatieonderzoek luidt: *Wat is een passend model dat een goede onderhoudsstrategie bepaalt voor de componenten van een schip dat geproduceerd wordt bij Damen Shipyards Group?*

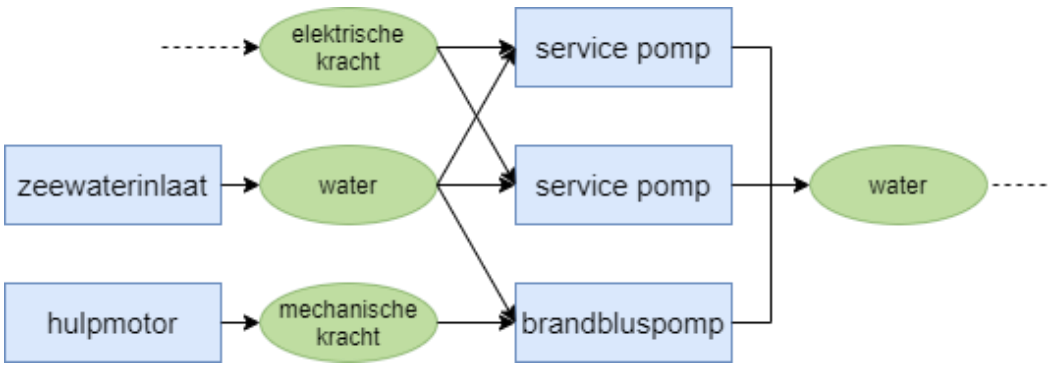
De componenten van de schepen die bij Damen geproduceerd worden, komen bij leveranciers vandaan. De leveranciers hebben voor de component al een onderhoudsstrategie vastgesteld, wat Damen overneemt als onderhoudsstrategie. Er zijn bij Damen vier onderhoudsstrategieën mogelijk per component: run-to-failure, planned maintenance, condition-based maintenance en predictive maintenance. Op verzoek van de klant wordt er RCM uitgevoerd. Dat is nu nog vaak voor de Marine of organisaties waarbij de uptime heel belangrijk is, omdat RCM nog te duur is om uit te voeren. Daarom is er een geautomatiseerd model opgesteld die de optimale onderhoudsstrategie bepaalt voor elke component.
Er zijn twee optimalisatiemogelijkheden:

1. Het optimaliseren van de faalkans voor een bepaald budget;
2. Het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans.

De klant geeft aan welke optimalisatiemogelijkheid hij wil en wat hierbij het budget of wat de maximale toegestane faalkans is per KPI (Key performance Indicator). Het model optimaliseert per KPI.
Voor het optimaliseren is het nodig dat het model de betrouwbaarheid en de onderhoudskosten van een systeem van een schip kan berekenen. Daarom wordt er een formule opgesteld voor de faalkans en een formule voor de onderhoudskosten van een systeem. Dit doet het model door het systeem te vereenvoudigen aan de hand van drie basissystemen, namelijk seriesysteem, parallelsysteem en een half serie-/parallelsysteem. De belangrijkste parameter van deze formules is de MTBF, de gemiddelde tijd tussen falen van een component. De MTBF verschilt per onderhoudsstrategie. In de meeste gevallen heeft de onderhoudsstrategie run-to-failure de laagste MTBF. Als er dus een andere onderhoudsstrategie toegepast wordt op de component stijgt de betrouwbaarheid van het systeem. De formule voor de onderhoudskosten bestaat uit vier soorten onderhoudskosten; kosten voor ongeplande downtime, kosten voor geplande downtime, kosten voor onverwachte reparatie, kosten voor verwachte reparatie.
Uiteindelijk moet de optimale of een goede oplossing gevonden worden. Daarom wordt er een zoekboom opgesteld waarin alle mogelijke oplossingen staan. Aan de hand van de opgestelde formules kunnen optimalisatiemethoden de faalkans en de onderhoudskosten van een oplossing uit de zoekboom berekenen. Er zijn drie optimalisatiemethoden geïmplementeerd. Brute-force is de eerste optimalisatiemethode die geïmplementeerd is voor beide optimalisatiemogelijkheden. Brute-force geeft altijd de optimale oplossing, maar heeft een zeer lange rekentijd naarmate er meer componenten geoptimaliseerd moeten worden en daarmee het aantal mogelijke oplossingen stijgt. De tweede optimalisatiemethode die geïmplementeerd is voor beide optimalisatiemogelijkheden is het Greedy algoritme. Bij de eerste optimalisatiemogelijkheid werd verwacht dat dit algoritme het globale optimum zou vinden. Dit is echter niet het geval. Het algoritme geeft wel een goede oplossing en heeft duidelijk een kortere rekentijd dan Brute-force. Daarom is dit algoritme het meest geschikt om de faalkans te optimaliseren bij een bepaald budget. De derde algoritme, Breadth-first heuristiek, is alleen geïmplementeerd voor de tweede optimalisatiemogelijkheid. Bij de tweede optimalisatiemogelijkheid gaven het Greedy algoritme en de Breadth-first heuristiek niet de optimale oplossing, maar wel een goede oplossing. De rekentijd van het Greedy algoritme is echter korter dan de rekentijd van de Breadth-first heuristiek. Daarom is voor het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans ook het Greedy algoritme het meest geschikt.

9. Aanbevelingen

Bij het maken van het model is ervan uitgegaan dat een component één variabele doorgeeft aan andere componenten. Als echter in de praktijk blijkt dat er een component is die meerdere variabelen doorgeeft aan andere componenten, zou aan het model toegevoegd kunnen worden dat van dit component meerdere componenten gemaakt worden. Voor iedere variabele één component, zodat er geen componenten zijn die meerdere variabelen doorgeven.
De berekeningen van de betrouwbaarheid worden tot nu toe met een vaste faalkans over de tijd gedaan. Er wordt geen rekening gehouden met het gegeven dat de faalkans van een component na een reparatie waarschijnlijk lager is dan de faalkans vlak voordat de component stuk gaat. In plaats van een vaste faalkans over de tijd zou het model een tijdsafhankelijke berekening kunnen gebruiken voor het berekenen van de faalkans van een component.
Om de betrouwbaarheid te berekenen worden de systemen met drie basissystemen vereenvoudigd. Deze drie basissystemen zijn een seriesysteem, een parallelsysteem en een half serie-/parallelsysteem. Voor de KPI 'veiligheid' van het schip de ASD 3212 uit bijlage 2 kan het systeem niet met deze drie basissystemen vereenvoudigd worden (zie figuur 29). De twee service pompen en de brandbluspomp ontvangen niet ieder een eigen variabele. De twee service pompen ontvangen wel samen dezelfde variabelen, maar de brandbluspomp ontvangt één van de variabele die de servicepompen ontvangen, namelijk water. Deze brandbluspomp ontvangt daarnaast mechanische kracht, wat de andere twee service pompen niet ontvangen. Hierdoor is het geen parallelsysteem en geen half serie-/parallelsysteem. Het is ook geen seriesysteem omdat alle drie de componenten meer dan één variabele ontvangen. Dit stukje systeem kan dus niet vereenvoudigd worden met de drie basissystemen. Voor dit soort systemen moet een nieuw basissysteem gevonden worden, wat een goede uitbreiding aan het model zou zijn.



Figuur 28 Stukje systeem wat niet vereenvoudigd kan worden

Er zijn ook systemen die cyclisch zijn. Dat is bijvoorbeeld bij de hoofdmotor het geval. De hoofdmotor krijgt elektrische kracht van de generator, maar de generator is ook nodig bij het opstarten van de hoofdmotor. In dit geval is het systeem cyclisch. Het model kan een cyclisch systeem niet detecteren. Een goede uitbreiding aan het model zou zijn om een basissysteem te vinden die een cyclisch systeem kan detecteren.
Daarnaast zijn het Greedy algoritme en de Breadth-first heuristiek twee optimalisatiemethoden die een goede oplossing vinden. De algoritme Simulated Annealing zou ook geïmplementeerd kunnen worden om te onderzoeken in hoeverre dit algoritme voor dit probleem een goede oplossing geeft of deze oplossing acceptabel is en of dit een betere oplossing is dan de oplossingen van het Greedy algoritme en de Breadth-first heuristiek.

Literatuurlijst

AutomationDirect. (z.d.). *MTBF and Product Reliability* (1). Geraadpleegd van https://ftp.automationdirect.com/pub/Product%20Reliability%20and%20MTBF.pdf

Bruce, H. (2018, 12 april). *Usage-based Preventive Maintenance: Important for Field Services*. Geraadpleegd op 6 februari 2020, van https://us.hitachi-solutions.com/blog/usage-basedpreventive-maintenance-important-field-services/

Buijs, A. (2012). *Statistiek om mee te werken* (9de editie). Groningen, Nederland: Noordhoff.

CMS Asset Management. (z.d.-a). *Opstellen onderhoudsconcepten FMECA-RCM*. Geraadpleegd op 17 maart 2020, van https://www.cmsam.nl/diensten/onderhoudsconcepten-fmeca-rcm/

CMS Asset Management. (z.d.-b). *Wat is een FMECA?* Geraadpleegd op 17 maart 2020, van https://www.cmsam.nl/kennisartikelen/wat-is-een-fmeca/

CMS Asset Management. (z.d.-c). *Wat is een risicomatrix?* Geraadpleegd op 17 maart 2020, van https://www.cmsam.nl/kennisartikelen/wat-is-een-risicomatrix/

Damen Shipyards. (z.d.-a). *About*. Geraadpleegd op 6 februari 2020, van https://www.damen.com/en/about

Damen Shipyards. (z.d.-b). *Stan Patrol 5009* [Foto]. Geraadpleegd van https://products.damen.com/en/ranges/stan-patrol/stan-patrol-5009

Damen Shipyards. (2019). *Process Analysis.* Geraadpleegd van file:///C:/Users/dorin/Downloads/RCM%20Analysis_Report.pdf

Dingemans, B. (2019, 18 november). *Wat is een query?* Geraadpleegd op 12 mei 2020, van https://programmeerplaats.nl/wat-is-een-query/

Ensie. (2018, 12 september). *Heuristiek | betekenis & definitie*. Geraadpleegd op 27 maart 2020, van https://www.ensie.nl/vandale1898/heuristiek

Fiix. (z.d.-a). *Condition-based Maintenance & Monitoring (CBM Maintenance)*. Geraadpleegd op 6 februari 2020, van https://www.fiixsoftware.com/condition-based-maintenance/

Fiix. (z.d.-b). *What is RTF? | Run to Failure Maintenance Examples*. Geraadpleegd op 6 februari 2020, van https://www.fiixsoftware.com/run-to-failure-maintenance/

Fiix. (z.d.-c). *What is Time-Based Maintenance? | TBM Maintenance Software*. Geraadpleegd op 6 februari 2020, van https://www.fiixsoftware.com/time-based-maintenance/

Fiix. (2020). *The essential guide to choosing a maintenance strategy for your assets* [Illustratie]. Geraadpleegd van https://www.fiixsoftware.com/blog/essential-guide-to-comparing-types-of-maintenance-strategies/

FileTypes. (z.d.). *Wat is een bestand JSON?* Geraadpleegd op 12 mei 2020, van https://www.filetypes.nl/extension/json

GeeksforGeeks. (z.d.). *Greedy Algorithms*. Geraadpleegd op 27 maart 2020, van https://www.geeksforgeeks.org/greedy-algorithms/

Hellings, P. (2011, 11 januari). *Hoeveel combinaties kan je maken met de cijfers 1,2,3 en 4?* Geraadpleegd op 15 mei 2020, van https://www.ikhebeenvraag.be/vraag/20637/Hoeveel-combinaties-kan-je-maken-met-de-cijfers-1-2-3-en-4

Hillier, F. S., & Lieberman, G. J. (2015). *Introduction to Operations Research* (10de editie). New York, Verenigde Staten: McGraw-Hill Education.

Hofstede. (z.d.). *Complementregel*. Geraadpleegd op 25 februari 2020, van https://www.hhofstede.nl/modules/complementregel.htm

Ingenieursbureau Westenberg B.V. (2016). *Risicomatrix* [Tabel]. Geraadpleegd van https://www.westenberg.net/risicogestuurdonderhoud-met-iasset/

Jacobson, L. (2012, 11 april). *Simulated Annealing for beginners*. Geraadpleegd op 27 maart 2020, van http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6

LeanSixSigmaTools. (2020, 6 februari). *Uitleg over KPI's!* Geraadpleegd op 13 april 2020, van https://leansixsigmatools.nl/wat-zijn-key-performance-indactors?cli_action=1586791157.455

Maritime Journaal. (2012). *Damen unveils the asd tug 3212 at its 2012* [Foto]. Geraadpleegd van https://www.maritimejournal.com/news101/tugs,-towing-and-salvage/damen-unveils-the-asd-tug-3212-at-its-2012

Neo4j. (2020, 12 mei). *Neo4j Graph Platform – The Leader in Graph Databases*. Geraadpleegd op 12 mei 2020, van https://neo4j.com/

Oxford Reference. (2019, 31 oktober). *Brute force algorithm*. Geraadpleegd op 19 maart 2020, van https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095532553

Patagonia. (z.d.). *FMEA Risicomanagement*. Geraadpleegd op 17 maart 2020, van https://www.patagonia-bv.com/fmea/risicomanagement/

Pride, A. (2016). *Reliability-Centered Maintenance (RCM) Decision Logic Tree* [Illustratie]. Geraadpleegd van https://www.wbdg.org/resources/reliability-centered-maintenance-rcm

ReliaSoft Corporation (Red.). (2015). RBDs and Analytical System Reliability. In *System Analysis Reference* (1ste editie, pp. 24–61). Geraadpleegd van http://reliawiki.org/index.php/RBDs_and_Analytical_System_Reliability

Schut, R. (2018). *Introduction to Artificial Intelligence Machine Learning and Games* (1). Geraadpleegd van https://www.studeersnel.nl/nl/document/rijksuniversiteit-groningen/introduction-to-artificial-intelligence/werkstukessay/Brute-force-algorithms/3310769/view

Sharma, A. (2016, 27 april). *Basics of Greedy Algorithms Tutorials & Notes | Algorithms*. Geraadpleegd op 28 april 2020, van https://www.hackerearth.com:443/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/

Spie. (z.d.). *FMECA-module voor industrie*. Geraadpleegd op 17 maart 2020, van https://www.spie-nl.com/fmeca-module/

Stackoverflow. (2011, 9 november). *What is the difference between "hill climbing" and "greedy" algorithms?* Geraadpleegd op 1 mei 2020, van https://stackoverflow.com/questions/8060028/what-is-the-difference-between-hill-climbing-and-greedy-algorithms

University of Georgia. (z.d.). *Module PE.PAS.U14.5 Analysis of series/parallel systems comprised of non-repairable components*. Geraadpleegd op 13 februari 2020, van https://www.coursehero.com/file/34996566/

UpKeep Technologies Inc. (z.d.). *What is Corrective Maintenance? Definition & Examples*. Geraadpleegd op 14 februari 2020, van https://www.onupkeep.com/learning/maintenance-types/corrective-maintenance

van Rhee, J. (z.d.). *Wat is een Brute-force aanval?* Geraadpleegd op 19 maart 2020, van https://support.reviced.nl/nl/articles/2642886-wat-is-een-Brute-force-aanval

Veerman, P. (2008). *Automatisch geroosterd* (1). Geraadpleegd van https://scripties.uba.uva.nl/document/641800

Vivadifferences. (z.d.-a). *Breadth-first search* [Illustratie]. Geraadpleegd van https://vivadifferences.com/difference-between-dfs-and-bfs-in-artificial-intelligence/

Vivadifferences. (z.d.-b). *Depth-first search* [Illustratie]. Geraadpleegd van https://vivadifferences.com/difference-between-dfs-and-bfs-in-artificial-intelligence/

Wikipedia. (z.d.). *Een binaire zoekboom* [Illustratie]. Geraadpleegd van https://nl.wikipedia.org/wiki/Binaire_zoekboom

Wikipedia. (2020). *MTBF* [Illustratie]. Geraadpleegd van https://nl.wikipedia.org/wiki/Mean_time_between_failures

Worksuite. (z.d.). *Voorsp. onderhoud*. Geraadpleegd op 6 februari 2020, van https://worksuite.nl/applicaties/predictivemaintenance?gclid=Cj0KCQiAm4TyBRDgARIsAOU75soYO11t4XEq_yz-u8MytPrI8BU1xhAg6XQR1UEYNKvNdVjx_Ln3G2caAiUKEALw_wcB

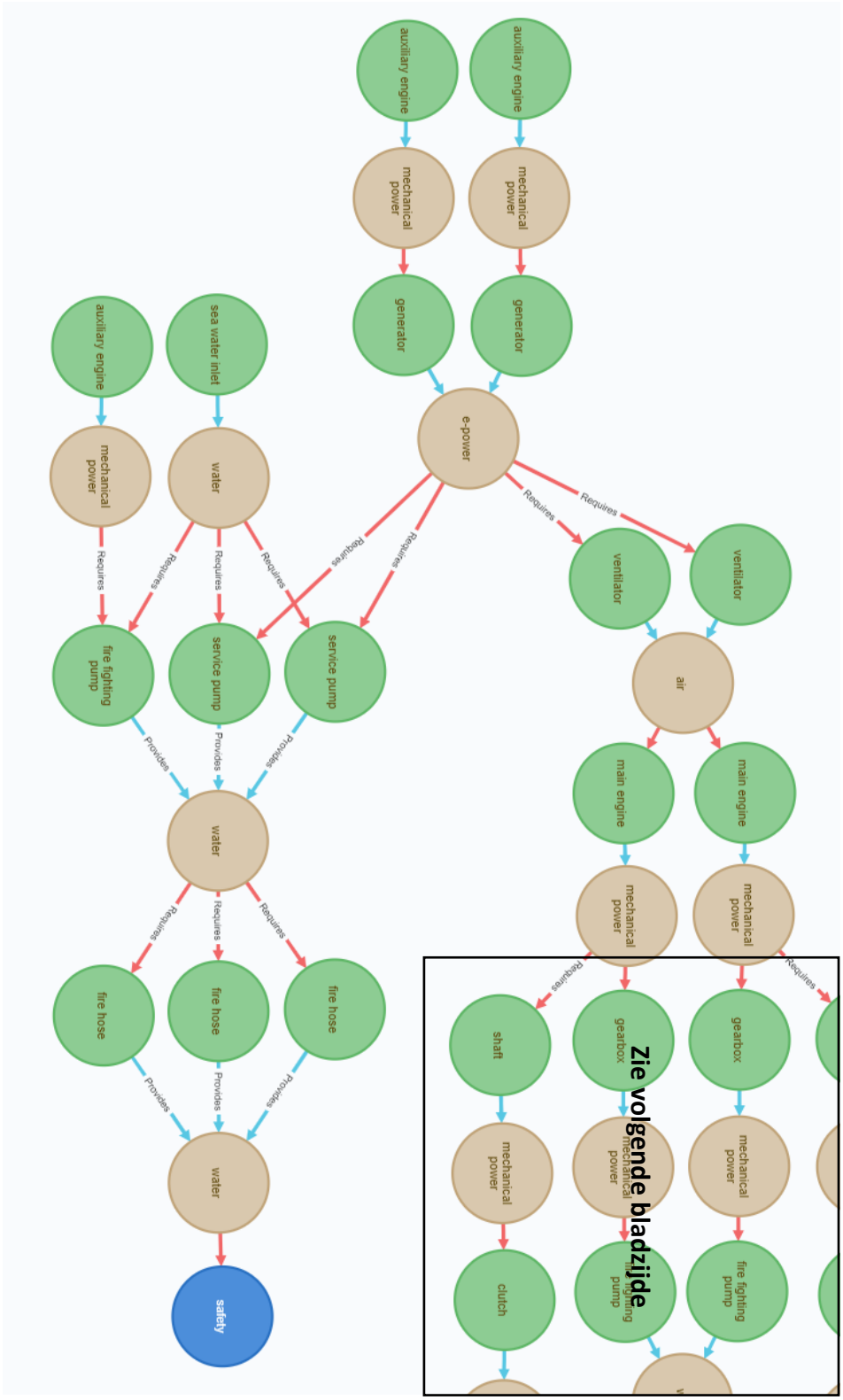
Bijlage 1: Variabelen uit JSON-bestand

Een JSON-bestand van een schip wordt door het model aan de hand van query's in Neo4j ingeladen en gebruikt bij de optimalisatie. In tabel 8 staan alle variabelen die het JSON-bestand bevat.

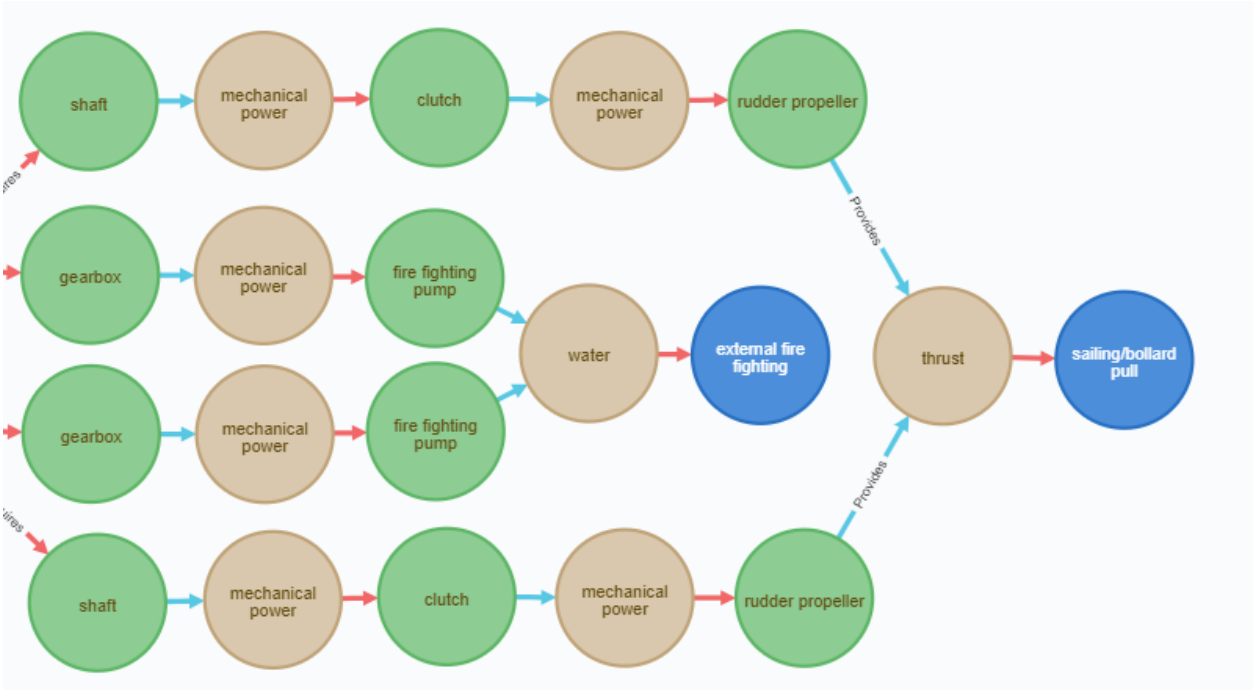
Tabel 8 Alle variabelen van een schip in een JSON-bestand

Schip		
Naam		
Imo		
Lengte		
Breedte		
Componenten	Naam	
	Id	
	Leverancier/fabrikant	
	Type	
	Onverwachte reperatiekosten	
	Huidige onderhoudsstrategie	
	Downtime voor onderhoud	
	Onderhoudsstrategieën	*Naam*
		MTBF
		Verwachte reparatiekosten
		Hoeveel keer per jaar onderhoud uitgevoerd wordt
Variabelen	Naam	
	Id	
KPI's	Naam	
	Id	
	Kosten ongeplande downtime	
	Kosten geplande downtime	
Relaties (leveren)	Naam van component die levert	
	Id van component die variabele levert	
	Variabele naam	
	Variabele id	
	Hoeveel er geleverd wordt	
Relaties (nodig)	Naam van component die variabele nodig heeft	
	Id van component die variabele nodig heeft	
	Variabele naam	
	Variabele id	
	Hoeveel er nodig is	

Een voorbeeld van hoe een graaf van de componenten van een schip eruitziet staat in figuur 30, met in figuur 31 de aanvulling van het zwarte kader.

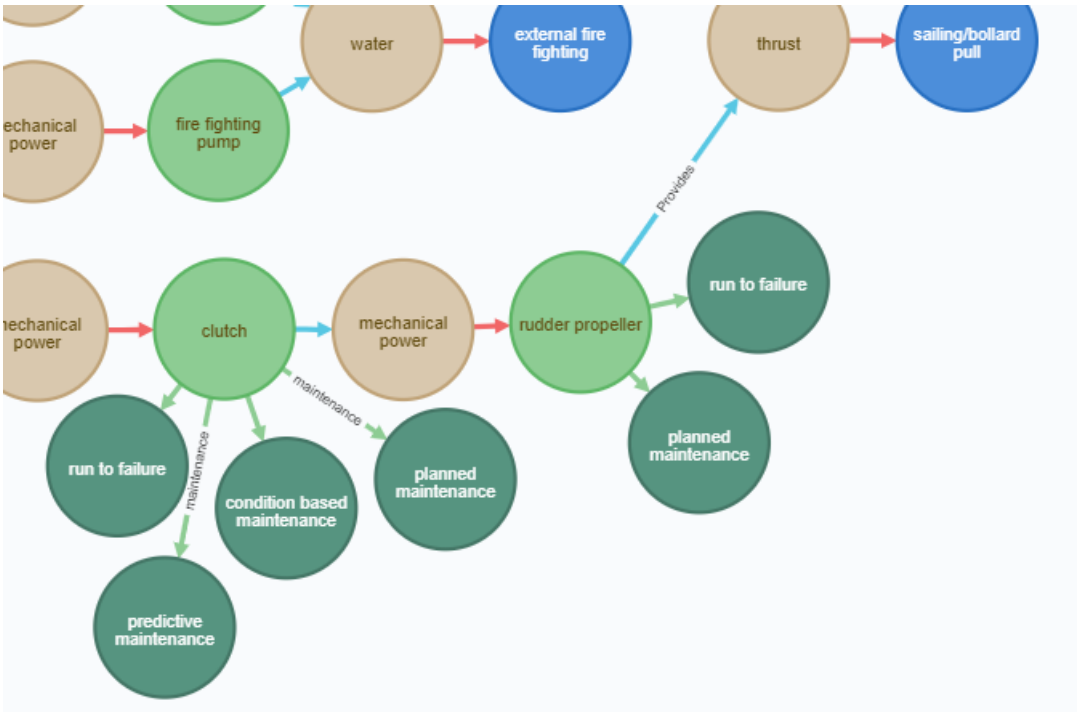


Figuur 29 Systeem ASD 3212 (1)



Figuur 30 Systeem ASD 3212 (2)

Daarnaast is elke component nog gerelateerd aan onderhoudsstrategieën, zoals voor enkele componenten te zien in de onderstaande figuur.



Figuur 31 Onderhoudsstrategieën in een graaf (ASD 3212)

Bijlage 3: De vijf query's

In deze bijlage staan de vijf query's die opgesteld zijn om vanuit een JSON-bestand een graaf te maken in Neo4j.
De eerste query is voor het inladen van de componenten:

```
CALL apoc.load.json($path) YIELD value
            UNWIND value.ship AS s
            FOREACH (c IN s.components |
            CREATE (component:Component)
            SET component.name=c.ctype, component.id=c.id,
            component.make=c.make,
            component.type=c.type,
            component.unexpected_repair_costs=c.unexpected_repair_costs,
            component.current_strategy=c.current_strategy,
            component.downtime=c.downtime
            FOREACH (m IN c.maintenance_strategy |
            CREATE (strategy:Strategy)
            SET strategy.name=m.name, strategy.MTBF=m.MTBF,
            strategy.expected_repair_costs=m.expected_repair_costs,
            strategy.number_of_tasks=m.number_of_tasks
            MERGE(component)-[:maintenance]-(strategy)))
```

De tweede query is voor het inladen van de variabelen:

```
CALL apoc.load.json($path) YIELD value
            UNWIND value.ship AS s
            UNWIND s.variables AS v
            CREATE (variable:Variable)
            SET variable.name=v.type, variable.id=v.id
```

De derde query is voor het inladen van de KPI's:

```
CALL apoc.load.json($path) YIELD value
            UNWIND value.ship AS s
            UNWIND s.KPIs AS k
            CREATE (kpi:KPI)
            SET kpi.name=k.type, kpi.id=k.id,
            kpi.costs_unplanned_downtime=k.costs_unplanned_downtime,
            kpi.costs_planned_downtime=k.costs_planned_downtime,
            kpi.requires=k.requires
```

De vierde query is voor het aangeven van de relaties tussen de componenten en de variabelen waarbij de componenten variabelen leveren:

```
CALL apoc.load.json($path) YIELD value
            UNWIND value.ship AS s
            UNWIND s.relation_provides AS r
            MATCH (a: Component) WHERE
            a.name=r.component_type and a.id=r.component_id
            MATCH (b: Variable) WHERE
            b.name=r.variable_type and b.id=r.variable_id
            MERGE (a)-[relation: Provides {amount:r.amount}]->(b)
```

De vijfde query is voor het aangeven van de relaties tussen de componenten en de variabelen waarbij de componenten variabelen ontvangen:

```
CALL apoc.load.json($path) YIELD value
            UNWIND value.ship AS s
            UNWIND s.relation_requires AS r
            MATCH (a) WHERE "
            a.name=r.component_type and a.id=r.component_id
            MATCH (b: Variable) WHERE
            b.name=r.variable_type and b.id=r.variable_id
            MERGE (a)<-[relation: Requires {amount:r.amount}]-(b)
```

Bijlage 4: Resultaten eerste optimalisatiemogelijkheid

In deze bijlage staan de resultaten van de optimalisatiemethoden voor het optimaliseren van de faalkans bij een bepaald budget. Per voorbeeld uit hoofdstuk 7 zijn de resultaten in een tabel weergegeven. De onderhoudsstrategieën zijn niet helemaal uitgeschreven maar als volgt afgekort:

- Run-to-failure: Run
- Planned maintenance Planned
- Condition-based maintenance Condition
- Predictive maintenance Predictive

Wanneer de oplossing van het Greedy algoritme niet gelijk is aan de oplossing van Brute-force wordt deze in het rood weergegeven.

Voorbeeld 1: seriesysteem

Tabel 9 Resultaten voorbeeld 1

Component	Brute-force (42.000)	Brute-force (52.000)	Brute-force (62.000)	Greedy (42.000)	Greedy (52.000)	Greedy (62.000)
Motor	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
As	Run	Condition	Condition	Run	Condition	Condition
Generator	Planned	Planned	Planned	Planned	Planned	Planned
Tandwielkast	Run	Run	Run	Run	Run	Run
Schroef	Condition	Run	Condition	Condition	Run	Condition
Kosten	40.938,30	50.104,35	56.658,30	40.938,30	50.104,35	56.658,30
Faalkans	0,155239	0,142082	0,137640	0.155239	0,142082	0,137640

Voorbeeld 2: parallelsysteem

Tabel 10 Resultaten voorbeeld 2

Component	Brute-force (50.000)	Brute-force (62.000)	Brute-force (72.000)	Greedy (50.000)	Greedy (62.000)	Greedy (72.000)
Schroef	Predictive	Condition	Condition	Predictive	Condition	Condition
As 1	Condition	Condition	Condition	Condition	Condition	Condition
As 2	Condition	Condition	Condition	Condition	Condition	Condition
Tandwielkast 1	Run	Run	Run	Run	Run	Run
Tandwielkast 2	Run	Run	Condition	Run	Run	Condition
Tandwielkast 3	Run	Run	Run	Run	Run	Run
Kosten	49.548,12	51.474,49	67.117,25	49.548,12	51.474,49	67.117,25
Faalkans	0,068502	0,066904	0,066820	0,068502	0,066904	0,066820

Voorbeeld 3: serie- en parallelsysteem

Tabel 11 Resultaten voorbeeld 3

Component	Brute-force (52.000)	Brute-force (60.000)	Brute-force (68.000)	Greedy (52.000)	Greedy (60.000)	Greedy (68.000)
Tandwielkast	Condition	Condition	Condition	Predictive	Condition	Condition
As 1	Run	Run	Run	Run	Run	Run
As 2	Run	Run	Run	Run	Run	Run
As 3	Run	Run	Condition	Run	Run	Condition
Schroef	Run	Run	Run	Run	Run	Run
Generator	Condition	Condition	Condition	Condition	Condition	Condition
Motor	Planned	Condition	Condition	Condition	Condition	Condition
Kosten	50.135,19	52.051,85	67.849,63	50.142,02	52.051,85	67.849,63
Faalkans	0,261468	0,256881	0,245339	0,261852	0,256881	0,245339

Voorbeeld 4: half serie-/parallelsysteem

Tabel 12 Resultaten voorbeeld 4

Component	Brute-force (47.000)	Brute-force (61.000)	Brute-force (71.000)	Greedy (47.000)	Greedy (61.000)	Greedy (71.000)
Brandstofpomp	Run	Run	Run	Run	Run	Run
Machinekamer ventilator	Run	Run	Condition	Run	Run	Condition
Generator	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Koelwater systeem	Run	Run	Run	Run	Run	Run
Motor	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Tandwielkast	Run	Run	Run	Run	Run	Run
Schroef	Run	Predictive	Predictive	Run	Predictive	Predictive
Kosten	34.230,20	48.996,86	64.794,64	34.230,20	48.996,86	64.794,64
Faalkans	0,266461	0.253592	0,241999	0,266461	0,253592	0,241999

Voorbeeld 5: groot systeem

Tabel 13 Resultaten voorbeeld 5

Component	Brute-force (75.000)	Brute-force (85.000)	Brute-force (95.000)	Greedy (75.000)	Greedy (85.000)	Greedy (95.000)
Motor 1	Run	Run	Run	Run	Run	Run
Motor 2	Run	Run	Run	Run	Run	Run
Motor 3	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Tandwielkast	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Brandstofpomp 1	Run	Run	Run	Run	Run	Run
Brandstofpomp 2	Run	Predictive	Predictive	Run	Predictive	Predictive
Brandstofpomp 3	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Brandstofpomp 4	Planned	Planned	Planned	Planned	Planned	Planned
Brandstofpomp 5	Run	Run	Predictive	Run	Run	Predictive
Generator	Run	Run	Run	Run	Run	Run
Kosten	67.958,43	80.360,04	94.473,38	67.958,43	80.360,04	94.473,38
Faalkans	0,101655	0,096776	0,092924	0,101655	0,096776	0,092924

Voorbeeld 6: ASD 3212 (KPI: externe brandbestrijding)

Tabel 14 Resultaten voorbeeld 6

Component	Brute-force (450.000)	Greedy (450.000)
Hoofdmotor 1	Planned	Planned
Hoofdmotor 2	Planned	Planned
Tandwielkast 1	Planned	Planned
Tandwielkast 2	Planned	Planned
Brandbestrijdingspomp 2	Planned	Planned
Brandbestrijdingspomp 3	Planned	Planned
Hulpmotor 2	Planned	Planned
Hulpmotor 3	Planned	Planned
Ventilator 1	Condition	Condition
Ventilator 2	Condition	Condition
Generator 1	Run	Run
Generator 2	Run	Run
Kosten	414.304,00	414.303,86
Faalkans	0,005995	0,005995

Voorbeeld 7: ASD3212 (KPI: varen/trekkracht)

Tabel 15 Resultaten voorbeeld 7

Component	Brute force (300.000)	Greedy (300.000)
Hulpmotor 2	Predictive	Run
Hulpmotor 3	Run	Run
Generator 1	Run	Run
Generator 2	Run	Run
Ventilator 1	Run	Predictive
Ventilator 2	Run	Condition
Hoofdmotor 1	Planned	Planned
Hoofdmotor 2	Planned	Planned
As 1	Predictive	Condition
As 2	Predictive	Predictive
Koppeling 1	Predictive	Planned
Koppeling 2	Predictive	Planned
Roerpropellor 1	Run	Planned
Roerpropellor 2	Planned	Planned
Kosten	298.132,00	286.464,45
Faalkans	0,006807	0,006890

Bijlage 5: Resultaten tweede optimalisatiemogelijkheid

In deze bijlage staan de resultaten van de optimalisatiemethoden voor het optimaliseren van de onderhoudskosten bij een maximale toegestane faalkans. Per voorbeeld uit hoofdstuk 7 zijn de resultaten in een tabel weergegeven. De onderhoudsstrategieën zijn niet helemaal uitgeschreven maar als volgt afgekort:

- Run-to-failure: Run
- Planned maintenance Planned
- Condition-based maintenance Condition
- Predictive maintenance Predictive

Wanneer de oplossing van het Greedy algoritme of van de Breadth-first heuristiek niet gelijk is aan de oplossing van Brute-force wordt deze in het rood weergegeven.

Voorbeeld 1: seriesysteem

Tabel 16 Resultaten voorbeeld 1

Component	Brute-force (0,145)	Brute-force (0,140)	Greedy (0,145)	Greedy (0,140)	Breadth-first (0,145)	Breadth-first (0,140)
Motor	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
As	Condition	Condition	Condition	Condition	Condition	Condition
Generator	Planned	Planned	Planned	Planned	Planned	Planned
Tandwielkast	Run	Run	Run	Run	Run	Run
Schroef	Run	Condition	Run	Condition	Run	Condition
Kosten	50.104,35	56.658,30	50.104,35	56.658,30	50.104,35	56.658,30
Faalkans	0,142082	0,137640	0,142082	0,137640	0,142082	0,137640

Voorbeeld 2: parallelsysteem

Tabel 17 Resultaten voorbeeld 2

Component	Brute-force (0,079)	Brute-force (0,069)	Greedy (0,079)	Greedy (0,069)	Breadth-first (0,079)	Breadth-first (0,069)
Schroef	Predictive	Condition	Run	Condition	Predictive	Condition
As 1	Condition	Condition	Condition	Condition	Condition	Condition
As 2	Run	Condition	Condition	Condition	Run	Condition
Tandwielkast 1	Run	Run	Run	Run	Run	Run
Tandwielkast 2	Run	Run	Run	Run	Run	Run
Tandwielkast 3	Run	Run	Run	Run	Run	Run
Kosten	33.657,01	49.548,12	35.548,61	49.548,12	33.657,01	49.548,12
Faalkans	0,075911	0,068502	0,071917	0,068502	0,075911	0,068502

Voorbeeld 3: serie- en parallelsysteem

Tabel 18 Resultaten voorbeeld 3

Component	Brute-force (0,261)	Brute-force (0,241)	Greedy (0,261)	Greedy (0.241)	Breadth-first (0,261)	Breadth-first (0,241)
Tandwielkast	Condition	Condition	Condition	Condition	Condition	Condition
As 1	Run	Condition	Run	Condition	Run	Condition
As 2	Run	Run	Run	Run	Run	Run
As 3	Run	Condition	Run	Condition	Run	Condition
Schroef	Run	Run	Run	Run	Run	Run
Generator	Condition	Condition	Condition	Condition	Condition	Condition
Motor	Condition	Condition	Condition	Condition	Condition	Condition
Kosten	52.051,85	83.727,13	52.051,85	83.727,13	52.051,85	83.727,13
Faalkans	0,256881	0,238461	0,256881	0,238461	0,256881	0,238461

Voorbeeld 4: half serie-/parallelsysteem

Tabel 19 Resultaten voorbeeld 4

Component	Brute-force (0,261)	Brute-force (0,241)	Greedy (0,261)	Greedy (0,241)	Breadth-first (0,261)	Breadth-first (0,241)
Brandstofpomp	Run	Predictive	Run	Predictive	Run	Predictive
Machinekamer ventilator	Run	Condition	Run	Condition	Run	Condition
Generator	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Koelwater systeem	Run	Run	Run	Run	Run	Run
Motor	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Tandwielkast	Run	Run	Run	Run	Run	Run
Schroef	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Kosten	48.996,86	79.634,64	48.996,86	79.634,64	48.996,86	79.634,64
Faalkans	0,253592	0,232975	0,253592	0,232975	0,253592	0,232975

Voorbeeld 5: groot systeem

Tabel 20 Resultaten voorbeeld 5

Component	Brute-force (0,090)	Brute-force (0,110)	Greedy (0,090)	Greedy (0,110)	Breadth-first (0,090)	Breadth-first (0,110)
Motor 1	Run	Run	Run	Run	Run	Run
Motor 2	Run	Run	Run	Run	Run	Run
Motor 3	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Tandwielkast	Predictive	Run	Predictive	Predictive	Predictive	Run
Brandstofpomp 1	Run	Run	Run	Run	Run	Run
Brandstofpomp 2	Predictive	Predictive	Predictive	Run	Predictive	Predictive
Brandstofpomp 3	Predictive	Predictive	Predictive	Predictive	Predictive	Predictive
Brandstofpomp 4	Planned	Planned	Planned	Planned	Planned	Planned
Brandstofpomp 5	Predictive	Run	Predictive	Run	Predictive	Run
Generator	Predictive	Run	Predictive	Run	Predictive	Run
Kosten	108.472,89	65.478,28	108.472,89	67.958,43	108.472,89	65.478,28
Faalkans	0,089586	0,106629	0,089586	0,101655	0,089586	0,106629

Voorbeeld 6: ASD 3212 (KPI: externe brandbestrijding)

Tabel 21 Resultaten voorbeeld 6

Component	Brute-force (0,006)	Greedy (0,006)	Breadth-first (0,006)
Hoofdmotor 1	Planned	Planned	Planned
Hoofdmotor 2	Planned	Planned	Planned
Tandwielkast 1	Planned	Planned	Planned
Tandwielkast 2	Planned	Planned	Planned
Brandbestrijdingspomp 2	Planned	Planned	Planned
Brandbestrijdingspomp 3	Planned	Planned	Planned
Hulpmotor 2	Predictive	Planned	Planned
Hulpmotor 3	Predictive	Planned	Planned
Ventilator 1	Run	Run	Run
Ventilator 2	Predictive	Predictive	Predictive
Generator 1	Run	Run	Run
Generator 2	Run	Run	Run
Kosten	350.834,63	400.430,09	400.430,09
Faalkans	0,005997	0,005995	0,005995

Voorbeeld 7: ASD3212 (KPI: varen/trekkracht)

Tabel 22 Resultaten voorbeeld 7

Component	Brute-force (0,007)	Greedy (0,007)	Breadth-first (0,007)
Hulpmotor 2	Run	Run	Run
Hulpmotor 3	Run	Run	Run
Generator 1	Run	Run	Run
Generator 2	Run	Run	Run
Ventilator 1	Run	Run	Run
Ventilator 2	Run	Run	Run
Hoofdmotor 1	Planned	Planned	Planned
Hoofdmotor 2	Planned	Planned	Planned
As 1	Predictive	Condition	Condition
As 2	Predictive	Predictive	Predictive
Koppeling 1	Predictive	Planned	Planned
Koppeling 2	Predictive	Planned	Planned
Roerpropellor 1	Planned	Run	Run
Roerpropellor 2	Planned	Planned	Planned
Kosten	250.087	258.587,66	258.587,66
Faalkans	0,006953	0.006985	0.006985

Handtekening begeleider: