BACHELOR

Over carousel pods

Bogerd, F.S.

Award date:
2005

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
26 augustus 2005

Bachelor Project
Over Carousel Pods

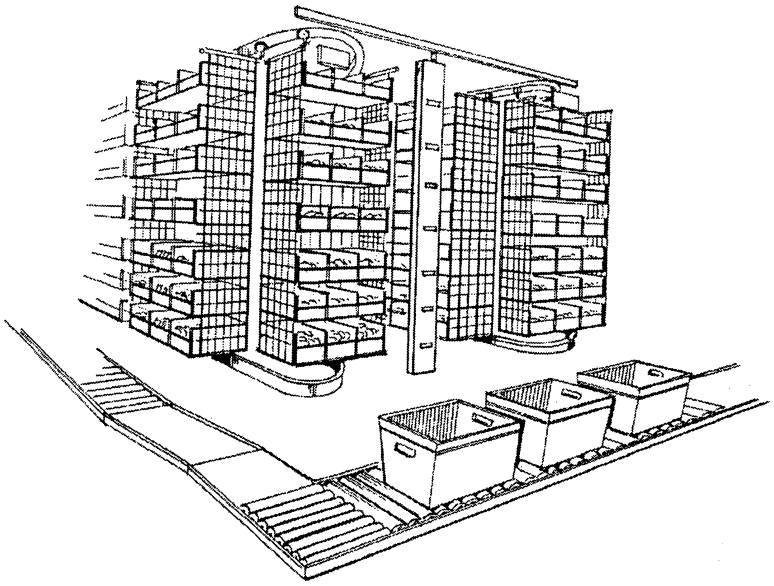
Finbar Bogerd 0474580

Inhoudsopgave

1 Inleiding

In steeds meer bedrijven is er de noodzaak om efficiënt met opslag om te gaan, vooral de tijd die het kost om orders te verzamelen kan erg omlaag. Veel bedrijven zijn dan ook overgestapt op automatische opslag systemen, zoals o.a. carousels.

Een carousel opslag systeem is een systeem van schappen die kunnen roteren. Ieder schap heeft een aantal opslagplaatsen onder elkaar. In figuur 1 is een voorbeeld te zien van een carousel opslag systeem. Het idee van een carousel is dat de producten naar de orderpicker toe komen in plaats van andersom. Dit heeft als groot voordeel dat de orderpicker tussen het pakken van items andere taken uit kan voeren (bijv. labelen). Ook kan één orderpicker centraal geplaatst worden tussen meerdere carousels. Als nu de orderpicker aan de ene carousel bezig is, kan de volgende carousel al voordraaien zodat de orderpicker veel minder hoeft te wachten en dus veel efficiënter kan werken. Dat is weer een reductie in 'picktijd' en heeft dus een hogere doorvoer van het systeem als gevolg.



Figuur 1: Voorbeeld van een carousel

Enkele voorbeelden van plaatsen waar carousel systemen worden toegepast:

- kleine onderdelen in een machinefabriek,
- medicijnen in een apotheek,
- andere plaatsen waar orders uit vele kleine delen bestaan.

2 Literatuur

In dit hoofdstuk zal gekeken worden naar verschillende artikelen die over carousel opslag systemen geschreven zijn. Er zal bij ieder artikel bekeken worden welk model en welke aannamen behandeld worden. Ook zal in het kort een samenvatting van de resultaten gegeven worden.

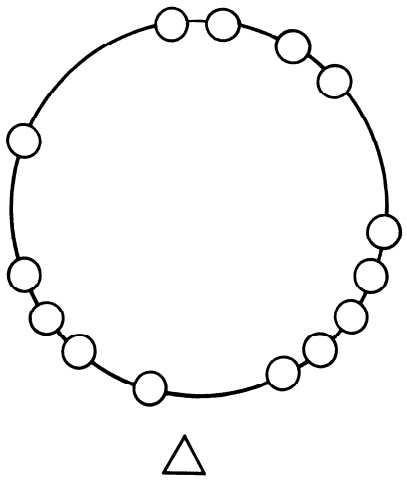
2.1 Bartholdi en Platzman [1]

Een elementair werk op het gebied van order pick strategieën voor een carousel systeem is van Bartholdi en Platzman [1]. Bartholdi en Platzman gaan uit van een enkele carousel met één picker[1]. De picker moet een bekende en vastgestelde hoeveelheid items verzamelen. Het is de bedoeling om dit zo snel mogelijk te doen zodat de service verbeterd en de kosten gereduceerd worden.

Een order is een verzameling items die tezamen verwerkt moeten worden (bijvoorbeeld voor een enkele klant). Er wordt van uitgegaan dat er slechts één order tegelijk gepickt kan worden. Er moeten meerdere orders verwerkt worden. Het probleem is om de volgorde van de te picken orders en de pick volgorde van items binnen een order zodanig te kiezen dat de orders zo snel mogelijk verzameld kunnen worden. Er worden een aantal algoritmes voorgesteld waarmee dit probleem opgelost kan worden.

De totale tijd die nodig is om een item te picken is opgesplitst in twee delen. Ten eerste de tijd die de carousel nodig heeft om naar de item locatie te bewegen. Ten tweede de tijd die de carousel stilstaat om er het item vanaf te halen. Verder wordt er aangenomen dat de tijd die nodig is om een item van de carousel af te halen onafhankelijk is van volgorde waarin de items gepickt worden. Om de totale tijd die nodig is voor een order omlaag te brengen moet de tijd die de carousel bezig is met roteren tussen de verschillende itemlocaties verkort worden. De carousel roteert met constante snelheid, zodat de de tijd nodig om te roteren proportioneel is met de afstand die de carousel moet verdraaien. Ieder item is slechts op één locatie op de carousel opgeslagen en mag niet verplaatst worden op de carousel tijdens het picken van de orders.

De carousel wordt gemodelleerd als een cirkel met daarop gelabelde punten die de item locaties voorstellen, zoals is weergegeven in figuur 2. Verder wordt er gerekend met afstanden in plaats van tijden. Dit is echter gerechtvaardigd vanwege bovenstaande aanname over de constante rotatiesnelheid.



Figuur 2: Het Bartholdi en Platzman model van een carousel

Er zijn twee niveaus van optimalisering mogelijk. De tijd die nodig is voor één order kan geminimaliseerd worden en verder kan de tijd tussen twee orders geminimaliseerd worden. Deze twee zijn niet onafhankelijk van elkaar, want de tijd nodig om een order te picken is afhankelijk van de itemlocatie waar de voorgaande order eindigde met picken. Voor ieder van deze niveaus wordt

[1]Picker: Robot of persoon die de items van de carousel afhaalt en verzameld voor verdere verwerking.

een aparte parameter ingevoerd. De *order rate* is het aantal orders dat gemiddeld per tijdseenheid binnenkomt om af te werken. De *item-density* is het gemiddeld aantal items per order.

Voor ieder van deze parameters zijn er extreme situaties met bijhorende (eenvoudige) order pick heuritieken:

1. *De order rate is relatief laag t.o.v. de snelheid waarmee orders gepickt worden.* De carousel moet bijna altijd wachten tussen twee opeenvolgende orders. De orders kunnen gewoon FIFO[2] afgehandeld worden en de tijdswinst wordt gehaald uit het optimaliseren van de picktijd per order.
2. *De order rate is relatief hoog t.o.v. de snelheid waarmee orders gepickt worden.* Nieuwe orders komen binnen terwijl de carousel nog bezig is. Orders moeten wachten terwijl de carousel van de eindlocatie van de ene order moet roteren naar de beginlocatie van de volgende, veel tijdswinst is te halen door deze wachttijd te verkorten.
3. *De item-density is groot t.o.v. het aantal itemlocaties op de carousel.* Als de item density maar groot genoeg is (Bartholdi en Platzman noemen hier een grens van 40% van de itemlocaties) dan zal om de order te picken de carousel langs de meeste locaties moeten. Hier is de bijna optimale oplossing het simpelweg maken van één gehele rotatie en onderweg ieder item picken zodra je het tegenkomt.
4. *De item-density is klein t.o.v. het aantal itemlocaties op de carousel.* Er hoeft maar een klein gedeelte van de locaties bezocht te worden. Het vinden van een goede pickvolgorde binnen een order kan een enorme reductie van de tijd opleveren, maar deze is echter meestal niet simpel te vinden.

Het is dus erg afhankelijk van de order rate en item-density welke optimalisering je het beste kunt toepassen.

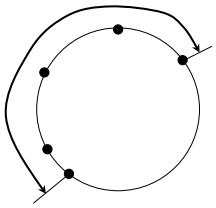
Bij lage order rate valt het minimaliseren van de doorlooptijd uiteen in een serie onafhankelijke problemen. De grootste tijdswinst is te behalen uit het zo efficiënt mogelijk picken van iedere order afzonderlijk, zonder te kijken naar vorige/volgende orders. Voor het picken van afzonderlijke orders zijn een algoritme en drie heuristieken gegeven.

- **'Optimal Retrieval' Algoritme** Bekijk alle mogelijke routes met maximaal 1 omkering in de route die alle itemlocaties van de order bezoekt. Neem van deze $2n+1$ opties (waar n het aantal te bezoeken locaties is) diegene die de kortste afstand heeft.
- **'Nearest Item' Heuristiek** Roteer altijd naar het dichtstbijzijnde item. Deze heuristiek is niet optimaal, maar er kan wel bewezen worden dat deze nooit meer dan een volle rotatie nodig heeft. Ook heeft deze heuristiek nooit meer dan twee maal de optimale afstand nodig.
- **'Shorter Direction' Heuristiek** Bekijk de route die rechtsom draait tot alle items gepickt zijn en bekijk de route die linksom draait tot alle items gepickt zijn. Neem van deze twee routes de kortste. Hoewel deze heuristiek nooit een route zal vinden die langer is dan een volledige rotatie kan hij toch vele malen langer zijn dan de optimale route. Deze heuristiek kan vooral toegepast worden als de item-density erg hoog is en de optimale route niet veel korter zal zijn dan een volledige rotatie. De rekentijd blijft klein. (er zijn nog steeds maar 2 mogelijkheden tegenover de $2n+1$ van de optimale route).
- **'Monomanical' heuristiek** Draai altijd rechtsom, en pick items als ze tegengekomen worden. Logge oplossing die dicht bij de optimale oplossing komt te liggen als de item-density maar hoog genoeg wordt.

[2]FIFO: First In First Out.

Bij hoge order rate is de volgorde van orders van grote invloed en moet zowel de volgorde van items binnen een order als de volgorde van orders bekeken worden. Hier is geen optimale oplossing meer te vinden omdat er een probleem ontstaat met rollende horizon. Wat betekend dat nieuwe orders binnenkomen terwijl je orders aan het picken bent. Al gemaakte keuzes die optimaal waren, kunnen niet meer optimaal blijken zodra nieuwe informatie bekend is.

Bartholdi en Platzman kiezen ervoor om de orders te picken volgens hun minimaal opspannend interval, dat is het complement van het grootste gap[3]. Een voorbeeld van het minimaal opspannend interval is te zien in figuur 3. Aan iedere order worden dan twee punten toegekend, de twee itemlocaties aan de rand van het minimaal opspannend interval. Deze twee locaties worden het begin en eindpunt van deze order. Omdat de carousel twee richtingen op kan roteren is het vrij te kiezen of de order linksom of rechtsom gepickt wordt. Het staat dus nog niet vast welke van de twee punten het beginpunt is. Deze twee punten noemen we de randpunten behorende bij de order.



Figuur 3: Voorbeeld van een order met zijn minimaal opspannend interval.

Omdat al vast staat hoe iedere order afzonderlijk gepickt moet worden is er alleen nog tijdswinst te halen door de ordervolgorde slim te kiezen. Voor het vinden van een ordervolgorde zijn de volgende heuristieken opgesteld.

- **'Hierarchical' heursitiek** Bekijk van iedere order alleen de twee randpunten. Maak nu een keten van orders. Dat wil zeggen verbind ieder randpunt van een order met een randpunt van een andere order zodanig dat één lange keten ontstaat (neem het startpunt van de carousel ook mee als randpunt). Er is een methode genaamd minimale matching, die dit zodanig doet dat de afstanden van de verbindingen tussen de orders minimaal is. Pick vervolgens de orders volgens deze keten. Het is bewezen dat ongeacht het aantal orders dat gepickt moet worden, de carousel nooit meer dan één volle rotatie meer zal afleggen dan de kortst mogelijke pickroute.
- **'Nearest Order' heuristiek** Roteer naar het dichtstbijzijnde eindpunt van een ongepickte order en pick deze. Als de carousel S itemlocaties heeft, dan zal de carousel onder deze heuristiek nooit meer dan $Log_S S$ rotaties meer nodig hebben dan de optimale pickroute.
- **'Monomanical' heuristiek** Roteer rechtsom tot je een eindpunt van een ongepickte order tegenkomt en pick deze. In de worst case kan deze heuristiek erbarmelijk slechte resultaten opleveren, toch zijn er zwakke garanties te geven. Zodra de item-density echter voldoende groot wordt zullen de meeste locaties bezocht moeten worden en zal deze heuristiek het steeds beter gaan doen.

De keuze van heuristiek zal dus sterk afhangen van de order rate en item-density. Als deze groter worden dan zullen de simpele heuristieken met veel minder rekentijd oplossingen vinden die dicht bij de optimale oplossing liggen. In de worst cases kunnen de simpele heuristieken oplossingen geven die ver van de optimale oplossing verwijderd liggen.

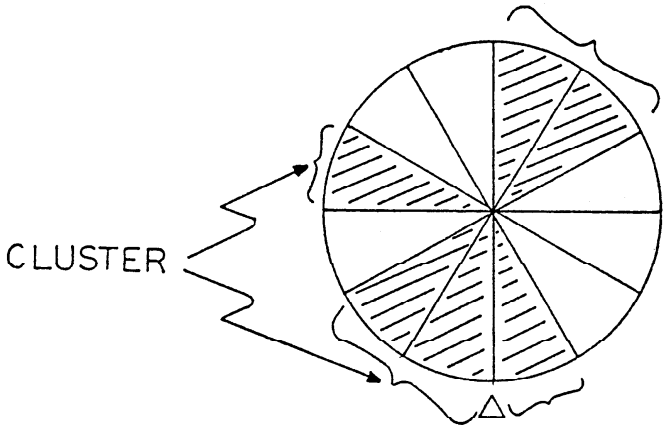
[3]Gap: afstand tussen twee itemlocaties van de order op de carousel.

2.2 Ghosh en Wells [3]

Ghosh en Wells [3] bekijken in hun paper een zelfde situatie als Bartholdi en Platzman [1]. Ook zij gaan uit van een enkele carousel bemand door een enkele picker (zij het menselijk danwel een robotarm). Alle aannamen van het model van [1] gelden nog steeds. Er wordt echter één significant andere aanname gedaan. In [1] mochten de orders nog in willekeurige volgorde afgewerkt worden, in [3] wordt er van uitgegaan dat de orders FIFO afgewerkt worden.

Er wordt ook hier onderscheidt gemaakt tussen het optimaliseren binnen een order en het optimaliseren van de tussen de orders. De optimale volgorde om een enkele order te picken hoeft niet de optimale volgorde te zijn als er meerdere orders gepickt moeten worden.

De carousel wordt in dit geval gemodelleerd als een continue cirkel met daarop alternerend *clusters* en *gaps*, waarbij een cluster een deel van de cirkel is dat correspondeert met een serie aan elkaar grenzende itemlocaties (binnen een order). Dit is een afwijking van [1] die uitgingen van een discrete verdeling van itemlocaties. Verder is een gap de ruimte tussen twee opeenvolgende clusters. Het is duidelijk dat een gap geen itemlocaties bevat die voor deze order bezocht hoeven worden. Het is dus niet nodig om over een gap te reizen als de order gepickt wordt. Verder wordt het startpunt van de carousel ook als de rand van een cluster gezien, ook al valt het startpunt midden in een cluster. Zoals ook duidelijk wordt uit figuur 4 waar een cluster als een gearceerd stuk cirkel wordt gezien en een gap als een blank stuk.



Figuur 4: Het Ghosh en Wells model van een carousel

Potentieel optimale paden zullen in het geval van een enkele order nooit meer dan één volledige rotatie maken. Ook zullen potentieel optimale paden niet meer dat 1 omkering (waar de carousel dus van draairichting veranderd) bevatten. De paden die midden in een cluster omkeren zijn ook zeker niet optimaal. De verzameling van paden die nog over blijven als we de bovengenoemde zeker niet optimale paden verwijderen uit de set van alle mogelijke paden noemen we de dominante set. Deze set beval naast de volledige rotatie alle paden die één omkering hebben aan de rand van een gap en na omkering stoppen aan de andere andere rand van hetzelfde gap. Ieder gap heeft twee mogelijk paden in de dominante set, degene die linksom gaat tot het gap en omkeert en het pad dat rechtsomgaat tot het gap en dan omkeert. Ook de volledige rotatie zit in de dominante set. De dominante set bevat dus $2G + 1$ paden. Voor het geval waarbij geen rekening gehouden wordt met de ordervolgorde en alleen per order geoptimaliseerd wordt, zijn twee algoritmes gegeven.

Het basis algoritme

Dit algoritme is een realisatie van het 'optimal retrieval' algoritme zoals in de vorige paragraaf beschreven. Het gaat alle mogelijke paden af die met maximaal 1 omkering alle items van de order picken. Het resultaat van dit algoritme is een order pick volgorde die de kleinste afstand op de carousel heeft en dus in de kortst mogelijke tijd gepickt kan worden.

Het algoritme gaat als volgt:

Gegeven een enkele order als een alternerende serie clusters en gaps (mogelijk met uitzondering bij de startpositie) dan wordt gedefinieerd:

LP = lengte van het pad rechtsom
LP' = lente van het pad linksom
LP^* = lente van het korste pad
LC = totale lengte van de cirkel
LG = lengte van het gap
D = afstand rechtsom om het gap te bereiken vanaf de startpositie
D' = afstand linksom om het gap te bereiken vanaf de startpositie

Het algoritme dat het optimum vind bevat de volgende stappen:

1. Begin op de startpositie.
2. Bepaal zowel linksom als rechtsom voor ieder gap de bijbehorende padlengtes $LP = D + LC - LG$ en $LP' = D' + LC - LG$.
3. Vergelijk alle kandidaatpaden en noem de kortste padlengte van alle kandidaatpaden LP^*. Als er geen gap is, of als $LP^* > LC$, dan is het kortste pad simpelweg de volledige rotatie en stel $LP^* = LC$.

Bovenstaand algoritme is vrij algemeen en ongevoelig voor de startpositie t.o.v. de clusters en gaps. Er is dan ook rekening houdend met de startpositie t.o.v. de clusters en gaps een algoritme dat een aantal van de niet optimale paden uit de dominante set herkend en niet doorrekent.

Het verbeterde algoritme

Dit algoritme vindt dezelfde oplossing als het basis algoritme zoals hierboven beschreven. Het verschil in de beide algoritmes is dat dit algoritme minder mogelijkheden afgaat op zoek naar het optimale pad. De overgeslagen paden zijn precies die paden die altijd langer zullen zijn dan de volledige rotatie, dus de paden die nooit optimaal kunnen zijn.

Het verbeterde algoritme gaat als volgt:

1. Begin op de startpositie.
2. Ga rechtsom, zolang $D < \frac{LC}{3}$ bepaal voor ieder gap met $LG > D$ de bijbehorende padlengte $LP = D + LC - LG$.
3. Ga linksom, zolang $D' < \frac{LC}{3}$ bepaal voor ieder gap met $LG > D'$ de bijbehorende padlengte $LP' = D' + LC - LG$.
4. Vergelijk alle op deze manier verkregen padlengtes en noem de kortste LP^*. Als geen geldige pad gevonden is stel dan de volledige rotatie als optimaal pad en $LP^* = LC$.

Het laatstgenoemde algoritme verkleint de omvang van de dominante set en dus het aantal evaluaties dat nodig is. Aangenomen dat iedere opslag locatie na schaling lengte 1 heeft. In het ergste geval is het totale aantal gaps dat bekeken moet worden $\frac{LC}{3}$. Er hoeven echter maar $\mathcal{O}(Log_2 \frac{LC}{3})$ padlengtes berekend te worden.

Voor het geval van meerdere orders wordt een dynamische oplossing gevonden. Hier moeten de orders FIFO afgewerkt worden. Ook moet een order helemaal gepickt worden voordat aan de volgende order begonnen kan worden. In [1] werd dit probleem opgelost door de aanname dat een order volgens zijn minimaal opspannend interval gepickt wordt en de aanname dat de orders in willekeurige volgorde afgewerkt konden worden. In dit geval worden beide aannames niet gemaakt. Dit levert een totaal andere situatie en dus ook andere oplossing op. De oplossing die voor dit geval gepresenteerd wordt is een dynamisch programeren uitbreiding op het eerder gebruikte model van clusters en gaps.

Het geval is hier dat de ordervolgorde vast ligt, echter de volgorde waarin de items binnen een order gepickt worden is vrij te kiezen. Het probleem is dus om de pickvolgorde van de items binnen iedere order zo te kiezen dat de totale afstand die door de carousel wordt afgelegd minimaal is.

Om dit probleem op te lossen stellen Ghosh en Wells een algoritme op dat gebruik maakt van het dynamisch programmeren. Ook maken ze gebruik van methodes die ook gebruikt zijn in het basis en verbeterde algoritme. De uitkomst van dit algoritme is voor iedere order de volgorde waarin de items gepickt moeten worden. Onder de aannames dat de orders FIFO afgewerkt dienen te worden is dit de pickvolorde met de kleinste afstand afgelegd door de carousel en dus met de kortste totale picktijd.

Voor de volledigheid is in appendix A het algoritme weergegeven. Voor het geval waarin halverwege het afhandelen van de orders nieuwe binnenkomen is een uitbreiding op dit algoritme gegeven. Ook deze uitbreiding is te vinden in appendix A.

2.3 Van Den Berg [8]

Van Den Berg geeft in zijn paper [8] een herleiding van het vinden van de optimale ordervolgorde als een variant van het *rural postman problem*. Van Den Berg gaat uit van de volgende aannamen:

- De operator maakt eerst een order af voordat hij aan de volgende begint.
- Als een product op meerdere locaties opgeslagen ligt dan zal het gepickt worden van die locatie waar het item al het langst ligt (er is dus een FIFO regel van kracht op de items op de carousel).
- De carousel roteert met constante snelheid.
- De tijd die de operator nodig heeft om tussen twee itemlocaties boven of onder elkaar te bewegen is verwaarloosbaar.
- De tijd die nodig is om de items na het picken weg te zetten is verwaarloosbaar.
- De tijd die nodig is om een item van de carousel af te picken is onafhankelijk van de volgorde waarin de items afgewerkt worden.
- De startpositie van de carousel is gegeven.
- De carousel mag op ieder willekeurig punt eindigen.

In het model wordt de carousel voorgesteld als een continue cirkel. De itemlocaties die gepickt moeten worden zijn *nodes* of knopen op deze cirkel. De combinatie van bovengenoemde aannames houdt in dat het minimaliseren van de doorlooptijd gelijk is aan het minimaliseren van de afstand die de carousel aflegt. Dit is gelijk aan het vinden van een Hamilton Pad van de gegeven startpositie tot een willekeurige eindpositie dat alle itemlocaties van alle orders af gaat. Net als in de voorgaande modellen moet een order helemaal verwerkt zijn voor aan de volgende begonnen mag worden. Deze eis moet ook verwerkt worden in het Hamilton Pad. Alleen die paden die aan deze eis voldoen zijn mogelijke oplossingen.

Voor het meerdere order probleem met vaste ordervolgorde vindt Van Den Berg een algoritme dat equivalent is met het in paragraaf 2.2 beschreven algoritme van Ghosh en Wells [3] voor een vaste ordervolgorde optimalisatie.

Voor het meerdere order probleem met vrije ordervolgorde gaat Van Den Berg uit van dezelfde aanname als Bartholdi en Platzman [1] en Ghosh en Wells [3]. Namelijk dat elke losse order gepickt wordt volgens zijn minimaal opspannend interval. Net als bij de twee voorgaande papers versimpelt dit de situatie enorm. Het vinden dan een optimale pickroute, waarbij de pickvolgorde binnen de orders en de volgorde van de orders bepaald mag worden, versimpelt dan tot alleen het vinden van een optimale volgorde van de orders. Iedere order wordt weer geïdentificeerd met twee punten, de twee randpunten van het opspannend interval waarover de order gepickt wordt. Het enige wat nog te doen staat is het vinden van een goede volgorde van de orders.

In dit probleem wordt een speciale variant van het *rural postman problem* bekeken. Het rural postman probleem is een bekend probleem, het omvat het vinden van een kortste route in een graaf waarbij een aantal van tevoren vastgelegde kanten bezocht moet worden, waarbij het niet uitmaakt in welke richting de kant bezocht wordt. Het rural postman probleem staat bekent als *NP-Hard* of NP-moeilijk. Omdat we hier met een speciaal geval te maken hebben is het mogelijk om het probleem optimaal op te lossen in polynomiale tijd.

Om het probleem op te lossen is het *matchtree algoritme* gegeven. Dit algoritme vind een matching, van de randpunten van iedere order, zodanig dat de de afstand die tussen de orders afgelegd wordt minimaal is.

Het matchtree algoritme vindt een optimaal pad waarin iedere order volgens zijn kleinste opspannend interval afgewerkt wordt. Als deze restrictie losgelaten wordt dan zijn er kortere paden vindbaar. Echter, de lengte van ieder pad dat alle orders afwerkt is minimaal zo lang als de som van de kleinste opspannende intervallen (LB) van de orders. Het matchtree algoritme vindt een oplossing die nooit meer dan anderhalve rotatie langer is dan LB onafhankelijk van het aantal orders en het aantal items per order. Van den Berg concludeert aan de hand van simulaties dat het gevonden pad meestal slechts 0.3 - 0.4 rotaties boven de LB ondergrens zit.

Dit matchtree algoritme is een realisatie van het opengelaten probleem in Bartholdi en Platzman [1]. Namelijk bij de 'Hierarchical' heuristiek geven Bartholdi en Platzman deze oplossing al aan. Zij geven echter geen expliciete manier om de keten van orders te bepalen, Van Den Berg heeft hier dit matchtree algoritme voor geschreven.

2.4 Meller en Klote [6]

Meller en Klote [6] leiden in hun paper formules af voor de gemiddelde tijd nodig om n items te picken. Na een overzicht te geven van de tot dan toe gepubliceerde literatuur beginnen ze met het afleiden van een model voor een enkele carousel met één menselijke orderpicker.

Aangenomen wordt dat de itemlocaties continu zijn, in plaats van een serie discrete locaties. Er wordt een strikte no-reverse policy gebruikt, dat wil zeggen dat er binnen een order niet van rotatierichting veranderd zal worden. De orders zullen verder in een FIFO manier afgehandeld worden. De items zijn uniform verdeeld over de carousel. Er is dus geen centrering van de meest gevraagde items. Dit alles wil zeggen dat de schattingen die gemaakt worden aan de hand van dit model nogal conservatief zullen zijn.

Vervolgens wordt een model gemaakt voor een serie van carousels met één picker en een zogenaamde *pod* van carousels. Er wordt van uitgegaan dat de picker utilisatie (het deel van de tijd dat de picker daadwerkelijk aan het picken is) bekend is. In dit geval is er een functie gegeven die een benadering geeft van de tijd die gemiddeld nodig is om een order te picken. Het is een benadering omdat er geen model is voor één picker en c carousels, daarom wordt het benaderd door een model met een picker met capaciteit $1/c$ en slechts één carousel. Ook is er een model gegeven waarmee de picker utilisatie berekend kan worden.

Uit vergelijkingen met simulaties is gebleken dat het model altijd met waarden komt die binnen 3% van de gemiddelde absolute fout en binnen de 6% van de maximaal absolute fout liggen. Dus hoewel het model nog op details aangepast kan worden, levert het met de gegeven parameter en model keuze al zeer goede benaderingen.

2.5 Park, Park en Foley [7]

In hun paper behandelen Park, Park en Foley [7] het berekenen van gemiddelde doorvoer voor een enkele picker die alternerend van twee verschillende carousels items pickt. Als er een uitdrukking is voor de gemiddelde doorvoer, dan is hieruit heel makkelijk de tijd te halen die gemiddeld nodig is om n items te picken. Dus hoewel anders opgeschreven bekijken zij hetzelfde probleem als Meller en Klote [6]. De uitwerking van Park, Park en Foley is echter anders dan die van Meller en Klote omdat ze gebruik maken van een ander model.

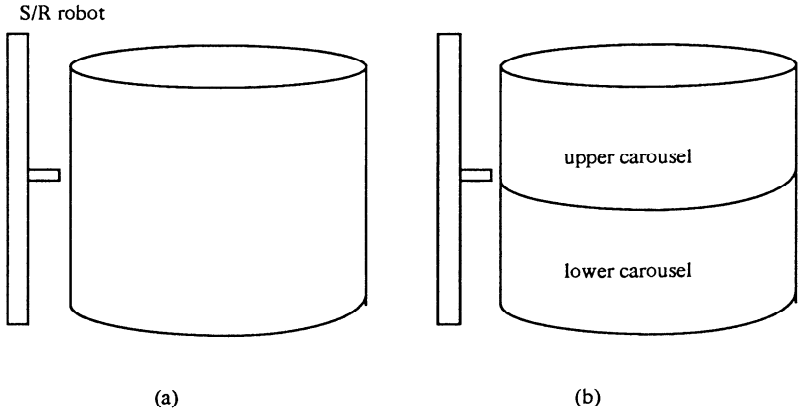
Uitgegaan wordt van de volgende situatie. Er zijn twee identieke carousels met één picker. De picker pakt alternerend iets van de ene en iets van de andere carousel. De picktijd wordt gedefinieerd als de tijd vanaf het moment dat de picker een item van de carousel af begint te halen tot het moment dat de picker gereed is om een item van de andere carousel af te pakken. Één picktijd kan dus meerdere activiteiten bevatten, zoals bijvoorbeeld het picken van meerdere items van dezelfde itemlocatie. De n-de picktijd wordt aangeduid met P_n. Verder wordt aangenomen dat de picker slechts één item van iedere picklocatie nodig heeft. De even items van de order komen dus van de ene carousel af terwijl de oneven items van de andere carousel afkomen. De carousel roteert met constante snelheid en accelereert instantaan tot maximale snelheid. Bij iedere carousel is er een oneindige lijst met te picken items. De carousel wordt gezien als een continuë loop in plaats van een groot aantal discrete locaties. De tijd wordt genormaliseerd, één tijdseenheid is de maximale tijd die nodig is om van de ene naar de andere picklocatie te komen. In het geval van een bi-directionale carousel is dat de tijdsduur van een halve rotatie, bij een uni-directionale carousel is het de tijdsduur van een volledige rotatie.

Voor exponentiële, deterministische en willekeurige picktijd verdelingen worden de gemiddelde doorvoer (aantal gepickte items per tijdseenheid) en het gedeelte van de tijd dat de picker bezig is afgeleid. De oplossing van de differentiaal vergelijking in dit paper gebeurt nogal ad-hoc, dit omdat er geen eenvoudige oplossing voor de gebruikte differentiaalvergelijking is. Ook wordt er een benadering gegeven voor de doorvoer. Er is geen bekendheid over de nauwkeurigheid van de gegeven benadering bij willekeurige picktijd verdelingen. Bij exponentiële picktijden is de op deze manier gevonden doorvoer een bovengrens op de echte doorvoer met een relatieve fout van minder dan 7%

2.6 Hwang en Ha [4]

Het paper van Hwang en Ha [4] behandeld de doorlooptijden van een variatie op de carousel. Zij gaan uit van een dubbele carousel. Zoals in figuur 5 schematisch is weergegeven. Een dubbele carousel is hetzelfde als een gewone carousel met dit verschil dat de carousel twee lagen bevat die onafhankelijk van elkaar kunnen roteren. Het probleem is het vinden van een uitdrukking voor de gemiddelde cyclustijden. In het geval van een enkele en een dubbele carousel, met een één of twee commando cyclus.

Het grote verschil met de modellen van Meller en Klote [6] en Park, Park en Foley [7] is dat bij Hwang en Ha de picker ook een verticale afstand af moet leggen. Omdat het item na picken door de robotarm helemaal beneden afgeleverd moet worden kost het dus meer tijd om een item van de bovenste carousel af te picken dan van de onderste carousel.



Figuur 5: Een standaard carousel (a) en de equivalente dubbele carousel (b)

Voor hun model zijn Hwang en Ha uitgegaan van de volgende aannamen:

- Een carousel wordt bediend door een enkele S/R[4] robot, deze robot kan slechts één item tegelijk vasthebben.
- De lengte en breedte van de carousel, zijn rotatiesnelheid en de verticale snelheid van de S/R robot zijn van te voren bekend.
- Het invoer/uitvoer punt van de machine zit helemaal beneden middenvoor.
- De S/R robot kan verticaal bewegen terwijl de carousel aan het roteren is.
- De opslaglocaties van de items op de carousel zijn willekeurig. Dat wil zeggen ieder schap van de carousel heeft gelijke kans om een item te bevatten dat gepickt moet worden, of om een item opgeslagen te krijgen.
- Het ophalen en terugzetten van een item op de carousel duurt altijd even lang.

Er wordt onderscheid gemaakt tussen twee mogelijkheden. Een zogenaamde *single command cycle* of één commando cyclus en een *dual command cycle* of dubbel commando cyclus. Bij een één commando cyclus wordt er per cyclus één item opgegehaald (gepickt) of opgeborgen. Bij een

[4]S/R: Storage / Retrieval of Opbergen / Ophalen.

dubbel commando cyclus wordt er per cyclus één item opgeborgen en tevens één item opgehaald.

De informatie over andere orders (voorgaande orders en opvolgende orders) is bekend zodat de carousel al kan voordraaien terwijl de S/R robot nog bezig is het item te verplaatsen naar het I/O punt. Voor de standaard carousel worden in deze paper de verwachte cyclustijd (tijd nodig om één volledige cyclus te doorlopen) afgeleid voor zowel de één als de twee commando cylcus. Voor de dubbele carousel en de één commando cyclus wordt de verwachte cyclustijd berekend als de carousels altererend gebruikt worden (dus afwisselend de bovenste en de onderste). Voor de dubbele carousel en de twee commando cyclus wordt de verwachte cyclustijd berekend voor vier verschillende gevallen:

1. Zowel het opbergen als het ophalen gebeurt op de onderste carousel.
2. Zowel het opbergen als het ophalen gebeurt op de bovenste carousel.
3. Het opbergen gebeurt op de onderste carousel en het ophalen gebeurt op de bovenste carousel.
4. Het opbergen gebeurt op de bovenste carousel en het ophalen gebeurt op de onderste carousel.

Er wordt geconcludeerd dat het gebruik van de informatie over voorgaande en opvolgende orders altijd een verbetering oplevert al hangt het heel erg van de modelparameters af hoe groot de verbetering precies is. Ook het gebruiken van een dubbele carousel levert altijd een verbetering op, al is het ook hier erg afhankelijk van de modelparameters hoe groot de verbetering precies is.

2.7 Jacobs, Peck en Davis [5]

Jacobs, Peck en Davis [5] bekijken in hun paper een heel ander probleem. Uitgaande van een carousel met eindige opslagruimte en een hoeveelheid orders beschrijven zij een oplossing voor het probleem hoeveel van ieder item te laden zodanig dat zoveel mogelijk orders verwerkt kunnen worden zonder de carousel opnieuw te hoeven laden. In hun paper leiden Jacobs, Peck en Davis een aantal expliciete formules af die een bijna optimale oplossing geven voor het probleem hoe de carousel te laden.

Voor hun model van de carousel maken Jackobs, Peck en Davis gebruik van de volgende aannamen. Een carousel heeft N verschillende itemlocaties. De locaties hebben allemaal dezelfde vastgestelde grootte, ongeacht hun inhoud. Er zijn j verschillende items, maar per itemlocatie zijn alle items identiek. De carousel moet geladen worden met volle itemlocaties, met c_i items van soort i, $1 \leq i \leq j$. Items worden van de carousel afgehaald in orders. Iedere order bestaat uit n_i items van soort i. Het probleem is het maximaliseren van de aantal orders g dat gepickt kan worden zonder dat er een tekort optreed in de voorraad. Dit wordt gedaan door het bepalen van de juiste waarden m_i, het aantal itemlocaties op de carousel dat met itemsoort i geladen moet worden.

Formeel is het probleem dan als volgen linear programmeer probleem te schrijven:

> Gegeven de positieve gehele getallen $c_1, \ldots, c_j, n_1, \ldots, n_j$ en N bepaal het grootste gehele getal g zodanig dat voor niet negatieve gehele getallen m_i geldt:
> $g \leq \frac{c_1 m_1}{n_1}, \ldots, g \leq \frac{c_j m_j}{n_j}, \sum_{i=1}^{j} m_i = N$.

Als $\alpha = \text{Min}\{\frac{c_i}{n_i}\}$, dan kan g nooit groter zijn dan αN. De oplossing van het probleem is in polynomiale tijd te vinden d.m.v. een binaire zoekmethode op de gehele getallen in het interval $[0, \alpha N]$. Jacobs , Peck en Davis stellen als doel het vinden van een benadering in lineaire tijd. Als de exacte oplossing gewenst is kan deze benadering als startwaarde gebruikt worden in de binaire

zoekmethode. Als benadering wordt een versoepeling van het lineair programmeer probleem opgesteld waarbij de niet negatieve gehele getallen vervangen worden door een niet negatief getal uit $\mathbb{R}$. Verder worden de $\leq$'s vervangen door ='s

Deze benadering is efficiënt en kan eenvoudig met de hand gedaan worden. Vooral in gevallen met grote aantallen parameters (> 1000) is de versoepeling sneller dan de exacte oplossingsmethode. De oplossing is nog steeds in de orde van enkele seconden gevonden. Bij het bekijken van dit probleem is er opzettelijk geen rekening gehouden met het feit dat verschillende items verschillende afmetingen kunnen hebben.

2.8 Bengü [2]

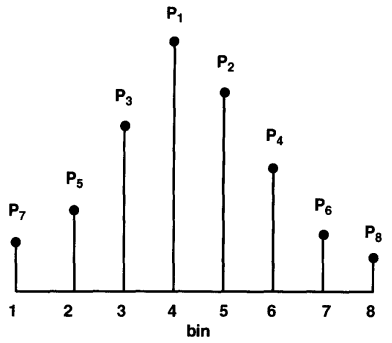
In zijn paper [2] bekijkt Bengü het probleem van het plaatsen van items op een carousel zodat de tijd om een order te picken verder verkleind wordt.

De volgende aannamen worden gemaakt over de carousel:

- De locatie van de picker is vast ten opzichte van de draaiende carousel. De itemlocatie op de carousel moet recht voor de picker staat wil hij iets van deze locatie kunnen picken.
- Een carousel kan zowel rechtsom als linksom roteren. En een item kan zowel linksom als rechtsom benaderd worden afhankelijk van welke van de twee korter is.
- Iedere order bestaat uit slechts één order en de orders worden FIFO afgehandeld.
- De carousel blijft wachten op de locatie waar een item gepickt is totdat de volgende order verwerkt wordt.
- De voorraad die de carousel kan bevatten is eindig en de kosten van de carousel zijn de transactiekosten. Doel is de doorvoer zo hoog mogelijk te krijgen, dus in zo kort mogelijke tijd zo veel mogelijk items te picken.

Het doel van een opslagbeleid is om de afstand die de carousel moet roteren te verkleinen. Als de carousel minder hoeft te roteren kunnen in dezelfde tijd meer items gepickt worden en gaat dus de doorvoer omhoog. Door de items zo te groeperen dat de items met de grootste vraag dicht bij elkaar staan wordt de kans dat de carousel kleine afstanden af hoeft te leggen groter. Het optimale opslagbeleid is ook bekend als *the Organ Pipe Arrangement* of de orgelpijp opstelling.

De orgelpijp opstelling gaat als volgt. Ken aan ieder item i de relatieve kans p_i toe. Dit is voor item i gelijk aan het aantal gepickt van item i gedeeld door het totaal aantal gepickte items. Zet het item met de hoogste p_i neer op een willekeurige locatie. Plaats nu aflopend in p_i de items i alternerend links en rechts van de al geplaatste items. In figuur 6 is dit schematisch weergegeven.



Figuur 6: De orgelpijp opstelling zoals beschreven in de paper van Bengü

Het is bewezen dat de orgelpijp opstelling de verwachte rotatietijd minimaliseert bij de gegeven aannames. De orgelpijp opstelling is ook optimaal wanneer er andere aannames gemaakt worden. Als de carousel zodra hij niets te doen heeft terugdraait naar een vaste locatie of als de carousel altijd tussen items eerst terugdraait naar een vaste locatie is deze opstelling nog steeds optimaal. In deze speciale gevallen moet echter wel het item met de hoogste p_i op de locatie staan waar de carousel naar terugdraait.

Vickson en Lu [9] komen in hun paper tot dezelfde conclusie. Zij bekijken de orgelpijp opstelling in een breder gebied, maar de gevonden resultaten voor carousel opslag systemen zijn identiek aan de door Bengü genoemde orgelpijp opstelling.

3 Opdracht

De orgelpijpopstelling maximaliseert de verwachte doorvoer. Er is echter weinig bekend over de invloed van de itemopstelling op doorvoer van de carousel. Het doel van deze opdracht is dan ook om uit te zoeken hoe groot de invloed is van de plaatsing van de items op de doorvoer.

Er zal gekeken worden naar meerdere vraagverdelingen[5] van de items. Ook zal er gekeken worden naar meerdere indelingen van items op de carousel.

3.1 Aannamen

- Er wordt uitgegaan van n verschillende items die op de carousel geplaatst zijn.
- Ieder item wordt op één locatie op de carousel opgeborgen.
- De verdeling van relatieve vraag naar ieder item is willekeurig te kiezen.
- Iedere order bestaat uit één item.
- De orders dienen FIFO afgewerkt te worden.
- Een carousel heeft precies n itemlocaties die op gelijke afstand van elkaar liggen.
- De rotatietijd van een carousel is 1.

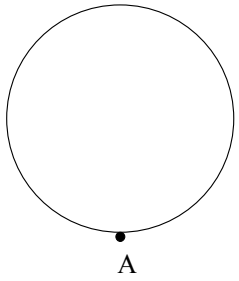
[5] vraagverdeling van item i: de kans dat een te picken item, item i is.

- De carousel roteert met vaste snelheid tussen twee locaties, waarbij het versnellen en afremmen van de carousel verwaarloosd wordt.
- Er is altijd een order beschikbaar om te pikken.
- De tijd nodig om een item te picken is onafhankelijk van de itemlocatie waarop het item gepickt wordt.

3.2 Aantal mogelijkheden om items op een carousel te plaatsen

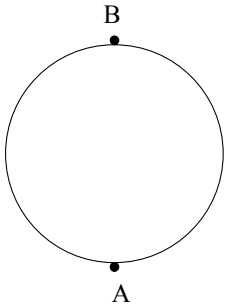
Uitgaande van een bi-directionale carousel met gelijke afstanden tussen de itemlocaties, hoeveel unieke mogelijkheden zijn er dan om n items op deze carousel te plaatsen. Waarbij spiegelingen of rotaties niet meegenomen worden. Spiegelingen of rotaties zullen namelijk dezelfde doorlooptijd geven. Deze hoeven dus niet ieder apart doorgerekend te worden.

Bij 1 item is er slechts 1 mogelijkheid zoals is weergegeven in figuur 7.



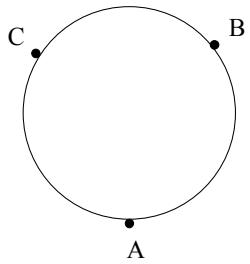
Figuur 7: Het aantal mogelijke plaatsingen van één item op de carousel.

Bij 2 items is er ook slechts 1 mogelijkheid, zoals is weergegeven in figuur 8, want als we de twee items omwisselen krijgen we dezelfde indeling terug maar dan halve rotatie gedraaid.



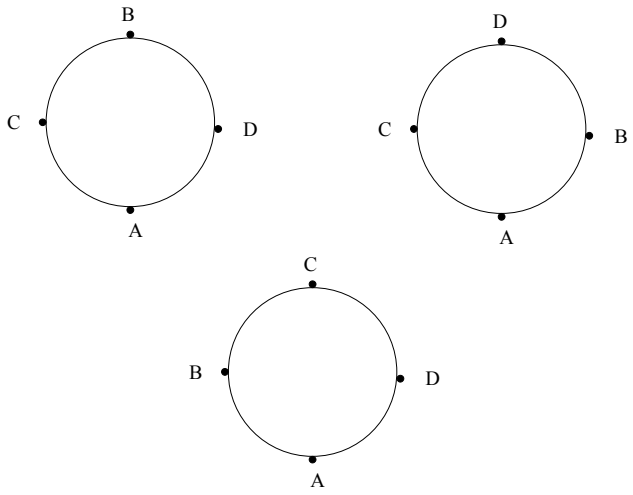
Figuur 8: Het aantal mogelijke plaatsingen van twee items op de carousel.

Bij 3 items is er ook slechts 1 mogelijkheid, want als we de drie items omwisselen krijgen we dezelfde indeling terug maar over één of twee derde geroteerd en mogelijk gespiegeld zoals is weergegeven in figuur 9.



Figuur 9: Het aantal mogelijke plaatsingen van drie items op de carousel.

Bij 4 items zijn er 3 mogelijkheden. Als we item A altijd vast onderin zetten dan blijven er voor de overige drie items drie locaties over. De in figuur 10 weergegeven plaatsingen zijn geen rotaties en / of spiegelingen van elkaar.



Figuur 10: Het aantal mogelijke plaatsingen van vier items op de carousel.

In het algemeen bij n items (voor $n > 4$) zijn er

$$\frac{(n-1)!}{(n-3)!*2} * (n-3)! = \frac{(n-1)!}{2}$$

mogelijke indelingen. De items zijn genummerd van meest naar minst gevraagd. Het meest gevraagde item zetten we midden onder. de items links en rechts van midden onder kiezen we tezamen. Dit is gelijk aan het trekken van 2 ballen uit een verzameling van $n-1$. Voor de overige $n-3$ items zijn ook $n-3$ plaatsen over, waarbij iedere andere plaatsing ook een andere indeling tot gevolg heeft.

Het is eenvoudig in te zien dat dit $\frac{(n-1)!}{2}$ oplevert. Om rotatie varianten eruit te halen staat het eerste item vast. Nu zijn er nog $n-1$ items over, deze kunnen op $(n-1)!$ mogelijkheden

geplaatst worden. Iedere indeling is nu echter twee keer geteld, namelijk de originele en zijn spiegelbeeld. Er zijn dus $\frac{(n-1)!}{2}$ **unieke** indelingen.

Het geval van de uni-directionale carousel is een stuk eenvoudiger. Hier hoeft alleen het eerste item vastgelegd te worden. Iedere andere plaatsing levert ook meteen een andere indeling op. Er zijn dus in totaal bij n items $(n-1)!$ unieke indelingen.

3.3 Analyse van het probleem

Er wordt uitgegaan van één bi-directionale carousel met n items. De relatieve vraag voor item op locatie j is $P(j)$. Om een geldige vraagverdeling te hebben moet gelden

$$\sum_{j=1}^{n} P(j) = 1$$

De afstand (en tijd) die de carousel aflegt om van locatie i naar locatie j te roteren is:

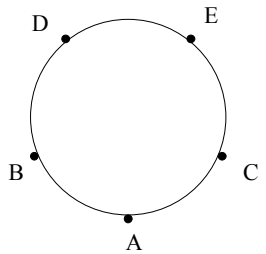
$$D_{ij} = \frac{Min[|i-j|, n-|i-j|]}{n}$$

De tijd die gemiddeld nodig is om tussen twee orders te roteren is:

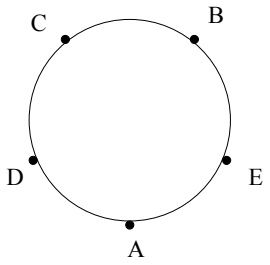
$$\sum_{j=1}^{n} \sum_{i=1}^{n} P(j)P(i)D_{ij}$$

Voorbeeld:
Een carousel met 5 itemlocaties. Er zijn dan $\frac{(n-1)!}{2} = \frac{4!}{2} = 12$ mogelijke indelingen. Als als vraagverdeling nemen we $V = \{0.5, 0.3, 0.1, 0.08, 0.02\}$ dus $P_1 = 0.5$ is de kans dat item A gevraagd wordt, $P_2 = 0.3$ de kans op item B enz.

De indeling met de laagste gemiddelde tijd tussen twee orders (de hoogste doorvoer) is zoals weergegven in figuur 11. Dit komt precies overeen met de orgelpijp opstelling. De gemiddelde rotatie tijd tussen twee orders is dan 0.16624 rotaties. De indeling met de hoogste gemiddelde tijd tussen twee orders (de indeling met de laagste doorvoer) is weergegeven in figuur 12. De gemiddelde rotatie tijd tussen twee orders is hier 0.21986.



Figuur 11: De indeling met de laagste gemiddelde tijd tussen twee orders.



Figuur 12: De indeling met de hoogste gemiddelde tijd tussen twee orders.

Wanneer er wordt uitgegaan van een gemiddelde picktijd van 1 is er in het beste geval een doorvoer van $\frac{1}{1+0.16624} = 0.857456$ items per tijdseenheid. Het slechtste geval heeft een doorvoer van $\frac{1}{1+0.21968} = 0.819887$ items per tijdseenheid. De hier genoemde waarden hangen sterk af van de gemiddelde picktijd. In het algemeen, als de gemiddelde picktijd x is, dan zal het procentuele verschil in doorvoer gelijk zijn aan:

$$\frac{\frac{1}{x+0.16624} - \frac{1}{x+0.21968}}{\frac{1}{x+0.21968}} * 100\% = \frac{0.05344}{0.16624+x} * 100\%$$

Met een gemiddelde picktijd van 1 levert dit een procentueel verschil in doorvoer op van ongeveer 4.6%.

3.4 Optimaliteit van de Orgelpijp indeling

Bengü [2] heeft in zijn paper al bewezen dat de orgelpijp indeling voor 1 carousel optimaal is. Voor de twee carousel situatie is dit echter niet gedaan. De indeling die de rotatietijd minimaliseert, zal in de twee carousel situatie (met 1 item per order) nog steeds optimaal zijn.

Voor de verzameling van item paren $A(t)$ nemen we de verzameling van item paren die een afstand $\leq t$ tot elkaar hebben. Voor vaste picktijd T geldt dan: Als er een indeling zou zijn met betere resultaten dan de orgelpijp indeling, dan zou de bij deze indeling horende verzameling $A(T)$ itemparen bevatten die een hogere gezamenlijke kans hebben dan de item paren in de $A(T)$ van de orgelpijp indeling. De orgelpijp indeling is echter zo gekozen dat de item paren in de $A(T)$ verzameling altijd de grootste gezamenlijke kans hebben. Dus er is geen betere indeling dan de orgelpijpindeling in de twee carousel situatie.

3.5 simulatie

Om de grotere systemen te kunnen bekijken zijn verschillende simulaties geschreven. De meest eenvoudige simulatie bepaalt de doorvoer van één enkele bi-directionale carousel met picktijd gelijk aan 0.

Als deze simulatie gedaan wordt met 5 items op de carousel, een vraagverdeling zoals de V uit het voorbeeld van paragraaf 3.3 en de orgelpijp opstelling van items. Dan levert de simulatie bij 100.000 orders een gemiddelde rotatietijd per order van 0.1662. Dit komt overeen met de analytische resultaten van paragraaf 3.3.

Als dezelfde simulatie gedaan wordt met een picktijd van 1. Dan krijgen geeft de simulatie een

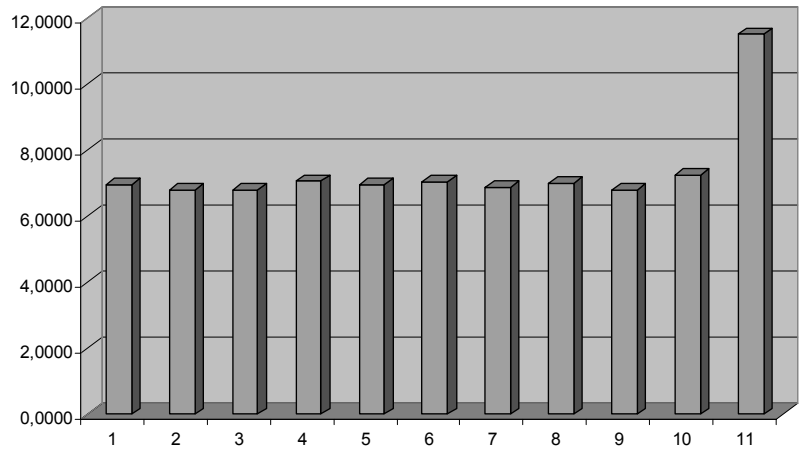
gemiddelde tijd per order (bij 100.00 verwerkte orders) van 1.1662. Dit komt overeen met een doorvoer van 0.8575. Dit komt ook overeen met de analytische resultaten van paragraaf 3.3. De broncode van deze simulatie is terug te vinden in appendix C.

Ook is er een simulatie geschreven voor het twee carousel probleem. Hierbij pickt de picker alternerend van de ene en de andere carousel. Een carousel begint met roteren naar het volgende (van die carousel) gevraagde item op het moment dat de picker klaar is met picken van deze carousel. De picker wacht vervolgens tot de andere carousel geroteerd is naar de itemlocatie (als deze nog aan het roteren is) en pikt het item van de andere carousel. De broncode van deze simulatie is terug te vinden in appendix D.

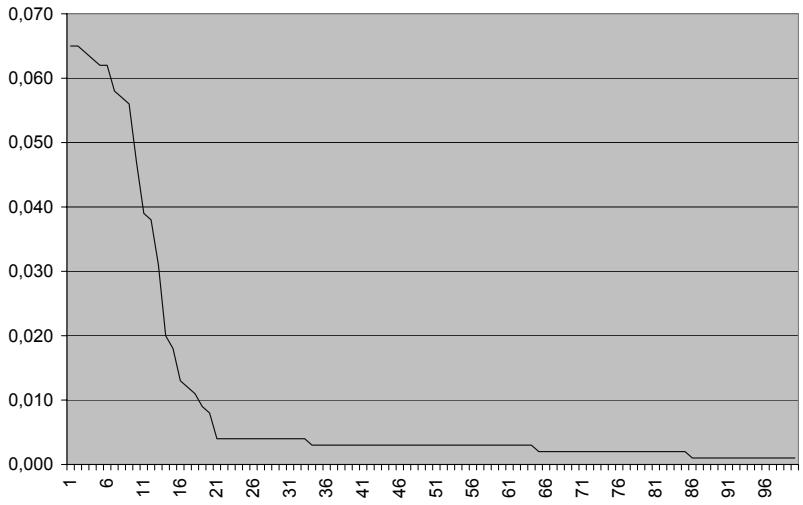
3.6 Simulatieresultaten

De invoer van de twee carousel simulatie is een vraagkans verdeling van de items op de carousel. Vervolgens wordt de doorvoer bepaald voor een aantal willekeurig gekozen itemindelingen op de carousel. Ook wordt de doorvoer bepaald van de orgelpijp opstelling. Voor iedere indeling worden 10 simulaties gedaan met ieder 100.000 orders. Van de 10 doorvoer waarden wordt het gemiddelde genomen als doorvoer voor die indeling.

In figuur 13 zijn de doorvoer waarden weergegeven voor een gegenereerde 80-20 vraagverdeling. Deze vraagverdeling komt vaak voor in de praktijk. Het gaat hier om een vraagverdeling waarbij 20% van de items 80% van de kansmassa heeft. In figuur 14 is de vraagkans uitgezet voor ieder van de 100 items (gesorteerd van hoge naar lage vraagkans).



Figuur 13: Doorvoer van 10 willekeurige indelingen en de OPA bij een 80-20 vraagverdeling.

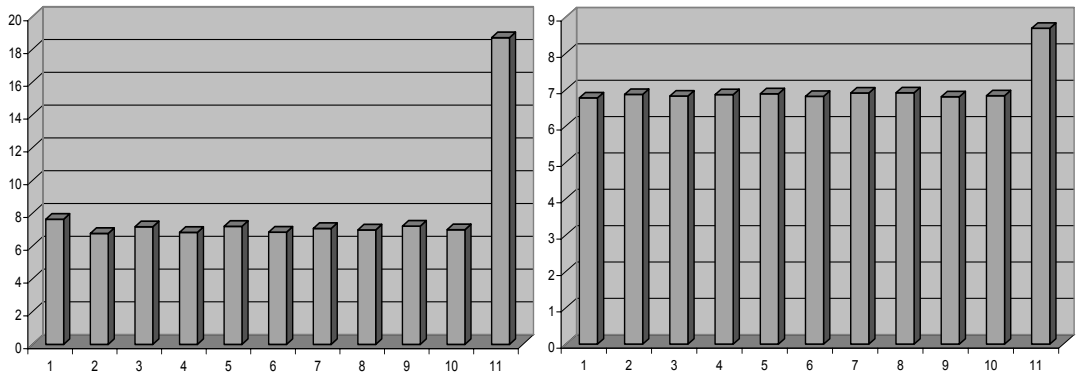


Figuur 14: De 80-20 vraagverdeling waarbij 20% van de items 80% van de kansmassa heeft.

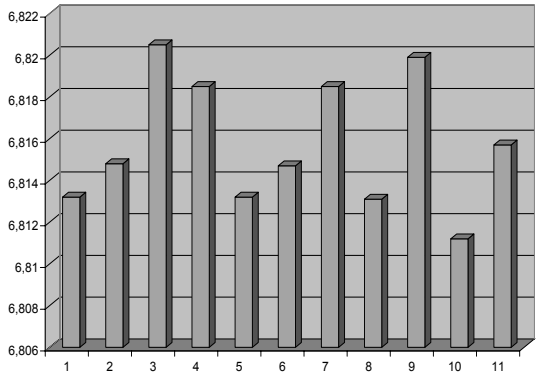
In de volgende tabel staan de waarden uit de grafiek, inclusief hun betrouwbaarheidsinterval. Het gaat hier om een 95% procent betrouwbaarheidsinterval. Dit interval is bepaald door de standaarddeviatie tweemaal bij het gemiddelde op te tellen voor de bovengrens en tweemaal er van af te halen voor de ondergrens. De standaarddeviatie is bepaald aan de hand van de 10 verschillende doorvoerwaarden die bij iedere indeling verkregen zijn.

Indeling	ondergrens	waarde	bovengrens
Random 1	6,8900	6,9612	7,0324
Random 2	6,6796	6,7803	6,8810
Random 3	6,7551	6,8074	6,8597
Random 4	7,0082	7,0629	7,1176
Random 5	6,8933	6,9484	7,0034
Random 6	6,9431	7,0355	7,1279
Random 7	6,8413	6,8881	6,9348
Random 8	6,9519	7,0008	7,0498
Random 9	6,7439	6,8040	6,8642
Random 10	7,1941	7,2474	7,3006
Orgelpijp	11,2464	11,5062	11,7661

De orgelpijp indeling geeft een doorvoer die ongeveer anderhalf keer zo veel is als een willekeurige indeling. Dit is duidelijk een verbetering. Om te kijken hoe groot de invloed van de item vraagverdeling is. Is dezelfde simulatie ook gedaan met een andere vraagverdeling als invoer.



Figuur 15: Doorvoer van 10 willekeurige indelingen en de OPA bij een 90-10 vraagverdeling (links) en een 70-30 vraagverdeling (rechts).

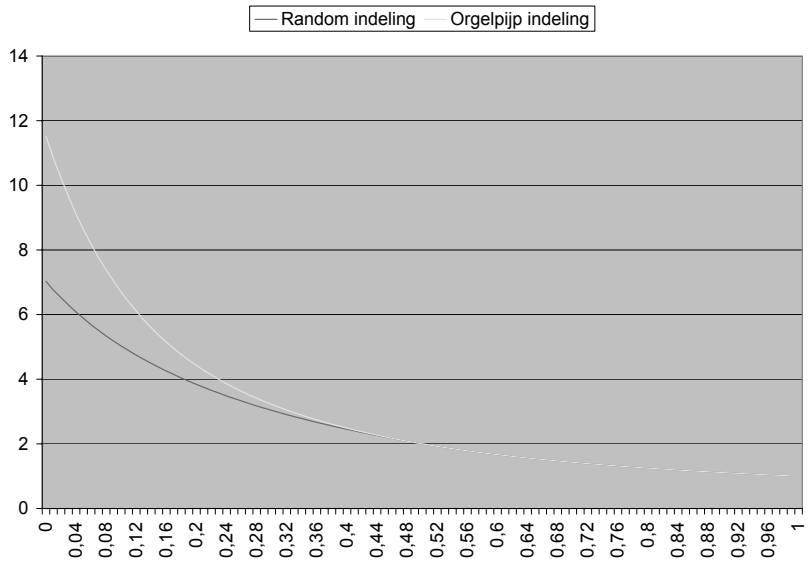


Figuur 16: Doorvoer van 10 willekeurige indelingen en de OPA bij een uniforme vraagverdeling.

In de figuren 15 en 16 is de doorvoer weergegeven van drie andere invoer verdeling nl. de 90-10 (waarbij 10% van de items een vraagkansmassa heeft van 90%), de 70-30 verdeling en de homogene verdeling. De doorvoer van van alle random indelingen in ieder van de drie grafieken zit ongeveer op 6,8. Het lijkt dus dat een willekeurige itemindeling op de carousel de kennis van de vraagverdeling teniet doet.

In figuur 16 is er eigenlijk geen sprake meer van een orgelpijp indeling, omdat ieder item exact dezelfde vraagkans heeft is de vraag naar items uniform over de carousel verdeeld of de orgelpijp indeling nu gevolgd wordt of niet.

De doorvoer van de orgelpijp indeling is heel hoog bij de 90-10 verdeling (bijna 3x zo veel) en bij de 70-30 verdeling is hij nog steeds groter maar slechts 1,3 maal. De orgelpijp indeling lijkt dus vooral veel invloed te hebben als er sprake is van een scheve vraagkans verdeling. Ofwel een vraagkans verdeling waarbij er grote verschillen in vraagkans zit tussen de items onderling. Een willekeurige verdeling zal onafhankelijk van de vraagkans verdeling altijd ongeveer dezelfde doorvoer geven.



Figuur 17: Invloed van de picktijd op de doorvoer, voor zowel een willekeurige indeling als de orgelpijp indeling.

In figuur 17 is de doorvoer uitgezet tegen de picktijd bij een 80-20 invoer verdeling. De doorlooptijd is bepaald voor zowel een willekeurige indeling and voor de orgelpijpindeling. Het gaat hier nog steeds om de twee carousel situatie. Bij een bi-directionale carousel is de carousel altijd in 0,5 tijdseenheid van de ene naar de andere itemlocatie geroteerd. Bij een picktijd van 0,5 of meer is de ene carousel altijd klaar met roteren op het moment dat het item van de andere carousel gepickt is. De doorvoer is in dan geval alleen nog afhankelijk van de picktijd en er zit geen verschil in doorvoer meer tussen de willekeurige indeling en de orgelpijp indeling. Bij een picktijd van 0,5 is de doorvoer dan ook $\frac{1}{0,5} = 2$ en bij een picktijd van 1 is de doorvoer $\frac{1}{1} = 1$. Hoe kleiner de picktijd wordt, hoe groter de invloed van de indeling de carousel is op de doorvoer. Zoals ook duidelijk uit de grafiek blijkt.

In bijlage B is de data weergegeven die in de verschillende grafieken in deze paragraaf is uitgezet.

4 Conclusie

In de gevallen waarbij de picktijd klein is levert de orgelpijpindeling een grote verbetering op. In het geval waar de vraagkans verdeling schever is, wat wil zeggen meer kansmassa bij een kleine groep items, zal de orgelpijp indeling ook grote verbeteringen opleveren.

In een veel voorkomende situatie waarbij de vraagkans verdeling een 80-20 regel volgt (waarbij 20% van de items 80% van de kansmassa heeft) heeft de orgelpijp een verbetering van 63% (dus 63% meer items in dezelfde tijd verwerkt) bij een picktijd van 0. Bij een picktijd van 0,1 rotatie is de verbetering nog 33%.

Bij een picktijd die groter is dan 0,5 maakt de indeling van items op de carousel niet uit. De maximum afstand tussen twee items op een bi-directionale carousel is gelijk aan 0,5. Na het picken is de andere carousel dus altijd geroteerd naar het volgende item. De doorvoer is hier alleen nog afhankelijk van de picktijd.

4.1 Vervolgonderzoek

Er zijn nog een aantal eenvoudige uitbreidingen mogelijk op de simulatie om een duidelijker beeld te krijgen van de invloed van de indeling op de doorvoer. Zo kan er een picktijdverdeling gekozen worden, in plaats van deterministische waarden zoals hier gebruikt is. Ook kan gekeken worden naar meerdere items per order in een één en twee carousel situatie.

Ook interessant zou zijn om te onderzoeken hoe een systeem zich in een worst case scenario gedraagt. Er zal dan echter wel bepaald moeten worden wat de worst case indeling zou zijn. Gevoelsmatig lijkt het een indeling die op soortgelijke manier als de orgelpijp indeling gemaakt wordt. Alleen dan worden de items zo geplaatst dat de afstand het grootst is tot de al geplaatste items. Of dit ook echt de worst case is, is niet bekend.

Referenties

[1] Bartholdi, J.J. and Platzman, L.K. (1986) Retrieval strategies for a carousel conveyor. *IIE Transactions*, **18**(2), 166-173.

[2] Bengü, G. (1995) An optimal storage assignment for automated rotating carousels. *IIE Transactions*, **27**, 105-107.

[3] Ghosh, J.B. and Wells, C.E. (1992) Optimal retrieval strategies for carousel conveyors. *Mathematical Computer Modelling*, **16**(10), 59-70.

[4] Hwang, H. and Ha, J.-W. (1991) Cycle time models for single/double carousel system. *International Journal of Production Economics*, **25**, 129-140.

[5] Jacobs, D.P., Peck, J.C. and Davis, J.S. (2000) A simple heuristic for maximizing service of carousel storage. *Computers & operations research*, **27**, 1351-1356

[6] Meller, R.D. and Klote J.F. (2004) A throughput model for carousel/VLM pods. *IIE Transactions*, **36**, 725-741.

[7] Park, B.C., Park, J.Y. and Foley, R.D. (2003) Carousel system performance. *Journal of applied probability*, **40**(3), 602-612.

[8] Van Den Berg, J.P. (1996) Multiple order pick sequencing in a carousel system: a solvable case of the rural postman problem. *Journal of the operational research society*, **47**, 1504-1515.

[9] Vickson, R.G. and Lu, X. (1998) Optimal product and server locations in one-dimensional storage racks. *European journal of operational research*, **105**, 18-28.

A Het Gosh en Wells dynamisch programeren algoritme

Het picken van de k^{de} order in de FIFO volgorde geeft een toestand van het dynamische programma. Een mogelijke toestand is een kant[6] waar de order helemaal gepickt is. Aangenomen wordt de notatie E_k voor de verzameling van kanten e_k die ofwel een kant van de huidige (k-de) order zijn, of een kant van een van de vorige orders waar de $(k-1)^{de}$ order geëindigt is met picken.

Gegeven de initiële voorwaarden:

e_0 = de startpositie van de eerste order
$CLP_0^* = 0$

en een staat e_k in E_k bij order k, de doelfunctie die we willen minimaliseren voor alle alternatieven e_{k-1} in E_{k-1} kan recursief gedefiniëerd worden als:

$CLP_k(e_{k-1}, e_k) = CLP^*_{k-1}(e_{k-1}) + LP(e_{k-1}, e_k)$ voor $k = 1, 2, \ldots, n$ met:

$CLP_k(e_{k-1}, e_k)$ = de cumulatieve padlengte van de startpositie tot kant e_k aan het eind van order k via e_{k-1} als eindpositie van order $k-1$,
$CLP^*_{k-1}(e_{k-1})$ = de optimale padlengte van de startpositie tot kant e_{k-1} als eind van order $k-1$, en
$LP(e_{k-1}, e_k)$ = de lengte van het pad nodig om order k te picken dat begint bij kant e_{k-1} en eindigt bij kant e_k.

[6] Een kant is een scheiding tussen een cluster en een gap.

Het volgende algoritme geeft voor een set van n orders het optimale pad.

1. Stel $k = 1$.
2. Evalueer $CLP_k(e_{k-1}, e_k) = CLP^*_{k-1}(e_{k-1}) + LP(e_{k-1}, e_k)$ bij iedere e_k voor ieder van de e_{k-1}'s.[7]
3. Bereken $CLP^*_k(e_k) = \min\{CLP_k(e_{k-1}, e_k)\}$, waar het minimum berekend wordt over alle mogelijke combinaties van $e_k \in E_k$ en $e_{k-1} \in E_{k-1}$.
4. als $k < n$ stel $k := k + 1$ en ga naar stap 2, anders stop.

Het algoritme vind het kortste cumulatieve pad vanaf de startpositie tot elke kant $e_k \in E_k$ voor $k = 1, 2, \ldots, n$. De waarden van $CLP^*_n(e_n)$ voor alle mogelijke $e_n \in E_n$ kunnen vergeleken worden om het optimale pad te vinden dat de n orders pickt.

De dynamische oplossing updaten voor nieuwe orders

Er wordt ervan uitgegaan dat er een optimale strategie bedacht is voor de n beschikbare orders. Verder zijn de eerste $j - 1$ orders gepickt volgens deze strategie en de j-de order is bezig gepickt te worden ($j < n$) wanneer er r nieuwe orders binnenkomen om gepickt te worden. Als we geen aanpassingen maken dan zal de huidige order eindigen op kant $e^*_j \in E_j$ op het optimale pad. Het pad om de volgende $n - j$ orders te picken kan echter niet meer optimaal zijn zodra de r nieuwe orders toegevoegd worden.

Een algoritme dat gebruik maakt van de bestaande gegevens van de oplossing voor de eerste n orders, de $CLP^*_k(e_k)'s$ update en het optimale pad vind voor de resterende $n+r-j$ orders gaat als volgt:

1. Stel $k = j + 1$, Laat $A(e_k) = CLP_k(e^*_{k-1}, e_k)$ voor iedere $e_k \in E_k$.
2. Bereken $\Delta(e_k) = A(e_k) - CLP^*_k(e_k)$ en $CLP^*_k(e_k) = A(e_k)$ voor iedere $e_k \in E_k$
3. Creëer de set $E'_k = \{e'_k | \Delta(e'_k) > 0\}$. Als $E'_k = \phi$ ga naar stap 7.
4. Als $k = n$ ga naar stap 7 anders $k := k + 1$.
5. Her-bereken voor elke $e'_{k-1} \in E'_{k-1}$, $CLP_k(e'_{k-1}, e_k) = CLP_k(e'_{k-1}, e_k) + \Delta(e'_{k-1})$ bij iedere $e_k \in E_k$.
6. Laat $A(e_k) = \min\{CLP_k(e_{k-1}l, e_k)\}$ zijn, waar het minimum genomen wordt over alle $e_{k-1} \in E_{k-1}$ bij iedere $e_k \in E_k$. Ga naar stap 2.
7. Gebruik de aangepaste waarden van $CLP^*_n(e_n)$ bij elke $e_n \in E_n$ en het voorgaande algoritme (het algoritme voor een set van n orders) om $CLP^*_k(e_k)$ te berekenen bij elke $e_k \in E_k$ voor $k = n + 1, n + 2, \ldots, n + r$. Bepaal vervolgens het optimale pad voor de orders $j + 1$ t/m $n + r$ (beginnende bij kant e^*_j van order j).

[7] Voor een gegeven e_{k-1}, kan $LP(e_{k-1}, e_k)$ gevonden worden met een simpele aanpassing aan stappen 1 t/m 3 van het eerst gegeven algortime voor één enkele order met e_{k-1} als startpositie.

B Simulatieresultaten

Van iedere vraagkansverdeling is bij iedere indeling de gevonden doorvoer waarde weergeven. Tevens is de boven- en ondergrens aangegeven van het 95% betrouwbaarheidsinterval.

De 90-10 vraagkansverdeling:

Indeling	ondergrens	waarde	bovengrens
Ramdom 1	7,5481	7,6366	7,7251
Ramdom 2	6,7274	6,7699	6,8123
Ramdom 3	7,0826	7,1749	7,2671
Ramdom 4	6,7611	6,8338	6,9064
Ramdom 5	7,0917	7,2005	7,3093
Ramdom 6	6,7857	6,8470	6,9084
Ramdom 7	6,9753	7,0750	7,1747
Ramdom 8	6,9306	6,9865	7,0424
Ramdom 9	7,1320	7,2179	7,3039
Ramdom 10	6,9263	6,9970	7,0676
Orgelpijp	17,7386	18,7095	19,6805

De 80-20 vraagkansverdeling:

Indeling	ondergrens	waarde	bovengrens
Random 1	6,8900	6,9612	7,0324
Random 2	6,6796	6,7803	6,8810
Random 3	6,7551	6,8074	6,8597
Random 4	7,0082	7,0629	7,1176
Random 5	6,8933	6,9484	7,0034
Random 6	6,9431	7,0355	7,1279
Random 7	6,8413	6,8881	6,9348
Random 8	6,9519	7,0008	7,0498
Random 9	6,7439	6,8040	6,8642
Random 10	7,1941	7,2474	7,3006
Orgelpijp	11,2464	11,5062	11,7661

De 70-30 vraagkansverdeling:

Indeling	ondergrens	waarde	bovengrens
Random 1	6,6832	6,7734	6,8637
Random 2	6,8044	6,8688	6,9331
Random 3	6,7503	6,8235	6,8966
Random 4	6,7886	6,8647	6,9409
Random 5	6,8286	6,8876	6,9465
Random 6	6,7673	6,8171	6,8669
Random 7	6,8266	6,9081	6,9896
Random 8	6,8366	6,9145	6,9923
Random 9	6,7463	6,8042	6,8620
Random 10	6,7469	6,8325	6,9181
Orgelpijp	8,5538	8,6902	8,8266

De uniforme vraagkansverdeling[8]:

Indeling	ondergrens	waarde	bovengrens
Random 1	6,7113	6,8132	6,9150
Random 2	6,7527	6,8148	6,8770
Random 3	6,7581	6,8205	6,8829
Random 4	6,7532	6,8185	6,8838
Random 5	6,7263	6,8132	6,9000
Random 6	6,7492	6,8147	6,8802
Random 7	6,7548	6,8185	6,8822
Random 8	6,7311	6,8131	6,8951
Random 9	6,7196	6,8199	6,9202
Random 10	6,7594	6,8112	6,8629
Orgelpijp	6,7639	6,8157	6,8676

[8]Omdat er hier een uniform verdeelde itemvraag is, is er eingenlijk geen sprake van een orgelpijp indeling.

De doorvoer van een random indeling en de orgelpijpindeling bij een gegeven picktijd. De vraagkansverdeling is de eerder gebruikte 80-20 verdeling.

Picktijd	Random	Orgelpijp
0,00	7,0256	11,5120
0,01	6,7530	10,8257
0,02	6,5234	10,2227
0,03	6,2920	9,6471
0,04	6,0788	9,1027
0,05	5,8796	8,6187
0,06	5,6844	8,1754
0,07	5,5076	7,7494
0,08	5,3364	7,3753
0,09	5,1776	7,0334
0,10	5,0268	6,6902
0,11	4,8849	6,3866
0,12	4,7424	6,1057
0,13	4,6115	5,8290
0,14	4,4885	5,5875
0,15	4,3691	5,3537
0,16	4,2509	5,1463
0,17	4,1434	4,9444
0,18	4,0381	4,7599
0,19	3,9339	4,5873
0,20	3,8388	4,4245
0,21	3,7440	4,2696
0,22	3,6542	4,1257
0,23	3,5685	3,9915
0,24	3,4826	3,8600
0,25	3,4025	3,7410
0,26	3,3248	3,6244
0,27	3,2482	3,5151
0,28	3,1740	3,4123
0,29	3,1026	3,3143
0,30	3,0344	3,2203
0,31	2,9675	3,1298
0,32	2,9034	3,0457
0,33	2,8417	2,9644
0,34	2,7803	2,8867
0,35	2,7224	2,8120
0,36	2,6655	2,7408
0,37	2,6106	2,6730
0,38	2,5558	2,6080
0,39	2,5030	2,5453
0,40	2,4515	2,4855
0,41	2,4011	2,4279
0,42	2,3522	2,3726
0,43	2,3043	2,3196
0,44	2,2577	2,2686
0,45	2,2124	2,2194
0,46	2,1682	2,1723
0,47	2,1245	2,1268
0,48	2,0820	2,0830
0,49	2,0405	2,0407
0,50	2,0000	2,0000

Picktijd	Random	Orgelpijp
0,51	1,9608	1,9608
0,52	1,9231	1,9231
0,53	1,8868	1,8868
0,54	1,8519	1,8519
0,55	1,8182	1,8182
0,56	1,7857	1,7857
0,57	1,7544	1,7544
0,58	1,7241	1,7241
0,59	1,6949	1,6949
0,60	1,6667	1,6667
0,61	1,6393	1,6393
0,62	1,6129	1,6129
0,63	1,5873	1,5873
0,64	1,5625	1,5625
0,65	1,5385	1,5385
0,66	1,5152	1,5152
0,67	1,4925	1,4925
0,68	1,4706	1,4706
0,69	1,4493	1,4493
0,70	1,4286	1,4286
0,71	1,4085	1,4085
0,72	1,3889	1,3889
0,73	1,3699	1,3699
0,74	1,3514	1,3514
0,75	1,3333	1,3333
0,76	1,3158	1,3158
0,77	1,2987	1,2987
0,78	1,2821	1,2821
0,79	1,2658	1,2658
0,80	1,2500	1,2500
0,81	1,2346	1,2346
0,82	1,2195	1,2195
0,83	1,2048	1,2048
0,84	1,1905	1,1905
0,85	1,1765	1,1765
0,86	1,1628	1,1628
0,87	1,1494	1,1494
0,88	1,1364	1,1364
0,89	1,1236	1,1236
0,90	1,1111	1,1111
0,91	1,0989	1,0989
0,92	1,0870	1,0870
0,93	1,0753	1,0753
0,94	1,0638	1,0638
0,95	1,0526	1,0526
0,96	1,0417	1,0417
0,97	1,0309	1,0309
0,98	1,0204	1,0204
0,99	1,0101	1,0101
1,00	1,0000	1,0000

C De enkele carousel simulatie broncode

```
program simulatie;
{$APPTYPE CONSOLE}
uses
  SysUtils;

var
   i, j, k: integer;
   n: integer;
   aantalovergangen: integer;
   sprongafstand: real;
   gemiddelde: real;
   kanstabel: array [1..1000] of integer;

function afstand (start, finish, lengte: integer):integer;
{bepaald de kortste afstand tussen start en finish in tussenstukjes}
var
        uno, dos : integer;
begin
uno := abs(start - finish);
dos := lengte - abs(start - finish);
if uno < dos then afstand := uno else afstand := dos;
end; {afstand}

function picktijd: real;
{geeft een realisatie van een random picktijd in rotaties}
begin
picktijd := 1;
end;

procedure maaktabel;
var
   f: Text;
   tempvar, ntel, ptel: integer;
begin
   assign(f,'input.txt');
   reset(f);
   readln(f, n);
   ptel:=0;
   ntel:=0;
   while not eof(f) do
   begin
   readln(f, tempvar);
   ntel:=ntel + 1;
   while tempvar > 0 do
         begin
         ptel:=ptel + 1;
         kanstabel[ptel]:=ntel;
         tempvar:= tempvar - 1;
         end;
   end;
   close(f);
```

```
   if ntel <> n then writeln('Het aantal ingevoerde items komt niet overeen met N');
   if ptel <> 1000 then writeln('De kansen tellen niet op tot 1');
end;

procedure orderovergang (start, finish, lengte: integer);
begin
aantalovergangen := aantalovergangen + 1;
sprongafstand := sprongafstand + (afstand(start, finish, lengte) / n) + picktijd;
end;


begin
   maaktabel;
   randomize;
   aantalovergangen := 0;
   sprongafstand := 0;
   k:= 1;
   for i := 1 to 10000000 do
   begin
       j := kanstabel[random(1000) + 1];
       orderovergang(k,j,n);
       k := j;
   end;
   writeln(' ');
   writeln('Aantal sprongen: ',aantalovergangen);
   writeln('Aantal afgelegde rotaties: ',sprongafstand);
   gemiddelde := sprongafstand / aantalovergangen;
   writeln('Gemiddelde afstand per order: ');
   writeln(gemiddelde,' of ongeveer ',(round(gemiddelde * 10000)),' / ',10000);
   writeln(' ');
   writeln('Dit is een doorvoer van: ',
            round(10000 / gemiddelde),' items per 10000 tijdseenheden');
   readln;
end.
```

D De dubbele carousel simulatie broncode

```
program simulatie;
{$APPTYPE CONSOLE}
uses
  SysUtils;


type pointer=^listnode;
     listnode=record
            itemnummer:integer;
            kans:integer;
            next:pointer;
          end;
var
   i,n : integer;
   kanstabel: array [1..1000] of integer;
```

```
procedure listinsert (itemnummer, kans: integer; var root:pointer);
{voor root wijst naar plek waar item ingevoegd moet worden
na item is op deze plek ingevoegd}
var tijdel:pointer;
begin
 new(tijdel);
 tijdel^.itemnummer := itemnummer;
 tijdel^.kans := kans;
 tijdel^.next := root;
 root := tijdel;
end;

procedure listdelete (var root:pointer);
{voor root wijst naar plek waar item gedelete moet worden
na item is op deze plek verwijderd}
var tijdel:pointer;
begin
  root := root^.next;
end;

function listbuild : pointer;
var loper, toren: pointer;
    ntel,ptel,tempvar,gelezenkans: integer;
    f: text;
    check: boolean;

begin
   assign(f,'input.txt');
   reset(f);
   readln(f,n);
   ptel:=0;
   ntel:=0;
   check:= true;
   while not eof(f) do
   begin
     readln(f, tempvar);
     if tempvar < 0 then check := false;
     ntel:=ntel + 1;
     ptel:=ptel + tempvar;
   end;
   close(f);
   if ntel <> n then check:= false;
   if ptel <> 1000 then check:= false;
if check = true then
begin
reset(f);
readln(f,n);
new(toren);
toren^.itemnummer := 0;
toren^.kans := 100000;
toren^.next := nil;
```

```
for i := 1 to n do
   begin
       readln(f,gelezenkans);
       loper:= toren;
       while (loper^.next <> nil) and (loper^.next^.kans > gelezenkans) do
       loper := loper^.next;
       listinsert(i,gelezenkans,loper^.next);
   end;
listbuild := toren^.next;
close(f);
end
else
begin
listbuild := nil;
writeln('Paniek: De invoer is incorrect');
end;
end; {listbuild}

function orgelpijp (root:pointer) :pointer;
var paard,koning,pion : pointer;
begin
koning := root;
new(paard);
paard^.itemnummer := koning^.itemnummer;
paard^.kans := koning^.kans;
paard^.next := nil;
pion:=paard;
koning := koning^.next;
while (koning <> nil) and (koning^.next <> nil) do
  begin
  listinsert(koning^.itemnummer,koning^.kans,pion^.next);
  listinsert(koning^.next^.itemnummer,koning^.next^.kans,pion^.next);
  pion := pion^.next;
  koning := koning^.next^.next;
  end;
if koning <> nil then listinsert(koning^.itemnummer,koning^.kans,pion^.next);
orgelpijp := paard;
end;{orgelpijp}

function shuffel (var root: pointer) :pointer;
var korter,langer,loperkort,loperlang :pointer;
    kiezer : integer;
begin
new(korter);
korter^.itemnummer:= -1;
korter^.kans:=2000;
korter^.next := root;
new(langer);
langer^.itemnummer := 0;
langer^.kans := 10000;
langer^.next := nil;
loperlang := langer;
for i := 1 to n do
begin
```

```
   kiezer := random(n-i)+1;
   loperkort:= korter;
   while kiezer > 1 do
     begin
        loperkort := loperkort^.next;
        kiezer := kiezer - 1;
     end;
   listinsert(loperkort^.next^.itemnummer,loperkort^.next^.kans,loperlang^.next);
   loperlang := loperlang^.next;
   listdelete(loperkort^.next);
end;
shuffel := langer^.next;
root:= nil;
root := listbuild;
end;{shuffel}


procedure print (root:pointer; var output:text);
var loper :pointer;
begin
loper := root;
writeln(output,'De itemindeling is als volgt:');
writeln(output,'itemnr    kans');
while loper <> nil do
  begin
  writeln(output,loper^.itemnummer:5,'     ',loper^.kans);
  loper := loper^.next;
  end;
end;

function afstand (start, finish, lengte: integer):integer;
{bepaald de kortste afstand tussen start en finish in tussenstukjes}
var
        uno, dos : integer;
begin
uno := abs(start - finish);
dos := lengte - abs(start - finish);
if uno < dos then afstand := uno else afstand := dos;
end; {afstand}

function picktijd: real;
{geeft een realisatie van een random picktijd in rotaties}
begin
picktijd := 0;
end;

procedure maaktabel (root:pointer);
var ptel,ntel,tempvar: integer;
    q: text;
begin
ntel:=0;
ptel:=0;
   while root <> nil do
   begin
```

```
   ntel := ntel+1;
   tempvar := root^.kans;
   while tempvar > 0 do
         begin
         ptel:=ptel + 1;
         kanstabel[ptel]:=ntel;
         tempvar:= tempvar - 1;
         end;
   root := root^.next;
   end;
end; {maaktabel}

procedure orderovergang (start, finish, lengte: integer; var aantalovergangen: integer;
                         var sprongafstand,resttijd:real);
var
       temp: real;
{begint te tellen op einde van picken van carousel rotatie, telt tot einde
picken carousel pick, voegt deze tijd toe aan sprongafstand}
begin
aantalovergangen := aantalovergangen + 1;
temp := resttijd + picktijd;
sprongafstand := sprongafstand + temp;
resttijd := (afstand(start,finish,lengte)/n) - temp;
if resttijd < 0 then resttijd := 0;
end;

procedure doesimulatie(indeling:pointer; aantalorders: integer; var output: text);
var   aantalovergangen: integer;
      sprongafstand, resttijd, gemiddelde: real;
      i,j,k,l,numsim: integer;
      waarden: array [1..10] of real;
      sd : real;
begin
   maaktabel(indeling);
   for numsim := 1 to 10 do
 begin
   aantalovergangen := 0;
   sprongafstand := 0;
   resttijd := 0; {overgebleven wachttijd van carousel pick}
   k:= 1; {positie pick-carousel}
   l:= 1; {begin positie rotatie-carousel}
   for i := 1 to aantalorders do
   begin
       j := kanstabel[random(1000) + 1];  {eindpunt van carousel rotatie}
       orderovergang(l,j,n,aantalovergangen,sprongafstand,resttijd);
       l := k;
       k := j;
   end;
   gemiddelde := sprongafstand / aantalovergangen;
   waarden[numsim]:= (1 / gemiddelde);
 end;

 gemiddelde := 0;
 sd := 0;
```

```
 for numsim := 1 to 10 do
  begin
    gemiddelde := gemiddelde + (waarden[numsim] / 10);
  end;
 for numsim := 1 to 10 do
  begin
    sd := sd + sqr(waarden[numsim] - gemiddelde);
  end;
  sd := sqrt(sd);
  writeln(output,(gemiddelde - (2 * sd)):5:4,':',
          gemiddelde:5:4,':',(gemiddelde + (2 * sd)):5:4);
end; {doesim}


var
   head,tail: pointer;
   output: text;
   kinky :integer;


begin
head := listbuild;
randomize;
assign(output,'output.txt');
rewrite(output);
if head <> nil then
  begin
    for kinky := 1 to 10 do
      begin
        tail := shuffel(head);
        writeln(output,'=======================');
        writeln(output,'Random indeling # ',kinky:4);
        writeln(output,'=======================');
        {print(tail,output);}
        doesimulatie(tail,100000,output);
      end;
    writeln(output,'====================');
    writeln(output,'De orgelpijp indeling');
    writeln(output,'====================');
    tail := orgelpijp(head);
    doesimulatie(tail,100000,output);
    writeln('Simulatie voltooid!');
  end
else writeln(output,'Paniek: De invoer was onjuist, dus geen simulaties gedaan!');
  close(output);
  readln;
end.
```