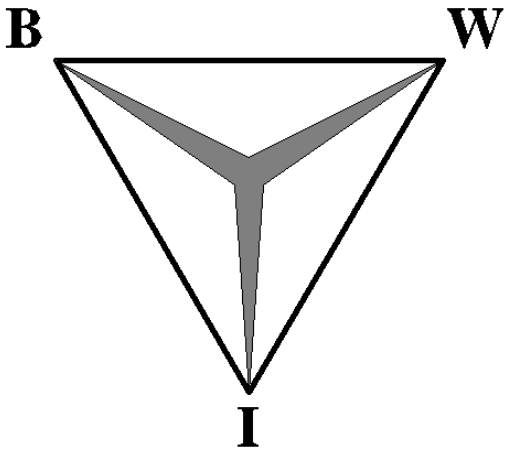
Evolutionaire Methoden voor het Job Shop Scheduling Probleem

- De overwinning op Operations Research -

Marco Bijvank

BWI werkstuk
25 juni 2004



Vrije Universiteit
Faculteit der Exacte Wetenschappen
Studierichting Bedrijfswiskunde en Informatica
De Boelelaan 1081a
1081 HV Amsterdam

Voorwoord

In de laatste fase van mijn opleiding Bedrijfswiskunde en Informatica (BWI) aan de *vrije* Universiteit te Amsterdam moet een werkstuk gemaakt worden over een zelf gekozen onderwerp. Het is de bedoeling dat dit onderwerp past binnen het kader van de studie.

Mijn interesse gaat uit naar logistieke vraagstukken die via een modelmatige en/of wiskundige techniek opgelost worden. Aangezien in de studie relatief weinig aandacht besteed is aan productieplanning en scheduling, was het voor mij vrij eenvoudig om te doen besluiten hier meer over te willen weten. Een veelvuldig toegepast probleem is het job shop scheduling probleem (JSSP). Er zijn echter veel mogelijke oplostechnieken ontwikkeld om dit probleem op te lossen. In dit verslag heb ik mij echter toegespitst op slechts één mogelijke aanpak.

Het merendeel van mijn studie pakt zulke en andere (combinatorische) probleemstellingen vaak via een exacte methodiek aan. Vandaar dat het mij leuk en interessant leek om mijn horizon te verbreden en een minder conventionele aanpak te onderzoeken. Dit heeft ertoe geleid dat ik mij ga bevinden op het gebied van de Artificial Intelligence (AI) en in het specifiek evolutionaire algoritmen, omdat deze techniek sinds ruim 20 jaar toegepast wordt voor job shop problemen en op dit moment met vooruitstrevende resultaten komt.

Gaandeweg kwam ik erachter dat er erg veel over het gekozen onderwerp gepubliceerd is en moest ik een aantal keuzes maken. De belangrijkste overweging hierin is iedere keer de mate van relevantie voor mijn werkstuk. Die relevantie bepaal ik aan de hand van een vraagstelling die in de inleiding geformuleerd wordt. Hierdoor heeft het redelijk wat tijd gekost om alles duidelijk op papier te zetten. Daarbij komt dat ik tegelijkertijd stage liep en dat er allerlei onverwachte opstoppingen waren. Het was al met al een zeer leerzaam traject waarin ik veel plezier heb gehad en het leereffect hoog ligt. Hopelijk denkt de lezer met net zoveel plezier en enthousiasme terug aan dit verslag na het gelezen te hebben.

Verder wil ik van de gelegenheid gebruik maken om mijn begeleider Guszti Eiben te bedanken voor zijn nuttige tips en zijn inzet voor mijn werkstuk. Tevens bedank ik de bedrijven ORTEC en Quintiq voor hun medewerking aan mijn werkstuk.

Samenvatting

In dit werkstuk is geprobeerd de volgende vraagstelling zo volledig mogelijk te beantwoorden:

Kunnen evolutionaire methoden een toegevoegde waarde leveren aan het oplossen van job shop scheduling problemen?

De hypothese is dat evolutionaire methoden zeker een potentieel hebben om extra meerwaarde te leveren, maar in welke mate is echter onbekend. Bij het beantwoorden van deze vraag zal tevens gekeken worden naar de verschillende type job shop scheduling problemen (JSSPs).

Bij het job shop scheduling probleem moeten *n* taken op *m* machines uitgevoerd worden, waarbij elke taak onderverdeeld is in operaties op de afzonderlijke machines. De vraag is om de operaties zo optimaal mogelijk in te plannen, zodat elke machine hooguit één operatie per tijdseenheid uitvoert en dat de operaties één voor één afgehandeld worden volgens een bepaalde volgorde relatie tussen de operaties behorend bij dezelfde taak.

Sinds de jaren 1940 is er veel onderzoek gedaan naar het JSSP binnen de Operations Research (OR). Dit heeft geresulteerd in een aantal exacte oplostechnieken, waarvan de branch-and-bound aanpak de beste resultaten oplevert. Deze methoden kunnen echter alleen een oplossing bieden op het moment dat het een deterministische productieomgeving betreft en het een kleinschalig probleem is. Vandaar dat praktische toepassingen uitblijven. Aangezien er toch wel behoefte is naar een ondersteunend schedulingssysteem, maar het gat met de theorie te groot was, heeft Artificial Intelligence (AI) het onderzoeksgebied opgenomen en met name evolutionaire algoritmen (EAs) onderzocht op toepasbaarheid voor dit type probleem. De afgelopen twee decennia heeft dit geresulteerd in de ontwikkeling van een aantal EAs die net zo goed presteren als de exacte methoden.
Eind jaren 1990 is er aandacht besteed aan het flexibele karakter van een EA door enerzijds de EA uit te breiden met andere technieken (zoals een lokale zoekmethode) en anderzijds de EA toe te passen in een meer dynamische productieomgeving. Door het robuuste karakter van een EA is het mogelijk om bestaande ontwikkelingen toe te passen op nieuwe situaties. Daarnaast zorgt de stochastische eigenschap van een EA ervoor dat de methode niet vroegtijdig convergeert. Er zijn zelfs een aantal technieken om dit verschijnsel verder te onderdrukken, dit leidt tot geavanceerde evolutionaire algoritmen die echter wel aanzienlijk meer rekentijd kosten.

Er kan dus geconcludeerd worden dat evolutionaire methoden wel zeker een toegevoegde waarde kunnen leveren aan scheduling vraagstukken en zelfs de bestaande oplosmethodieken kunnen verbeteren. In de praktijk blijkt dat ze dit ook daadwerkelijk doen. Met name op het gebied van een ingewikkelde (niet-deterministische) en/of relatief grote productieomgeving zal de bijdrage aanzienlijk zijn. De rekentijd is tevens niet onoverkomelijk. Dit heeft er met name mee te maken dat computers steeds sneller worden, maar belangrijker is nog wel het feit dat het vinden van een toegelaten oplossing in zo'n (ingewikkelde) situatie net zo belangrijk is als het vinden van een optimale oplossing.

Inhoudsopgave

1. Inleiding

Iedereen krijgt in zijn leven op één of andere manier wel te maken met schema's, roosters, planningen etc. Neem bijvoorbeeld de dienstregeling van de bussen of spoorwegen of het productieschema in een fabriek. Het proces om zo'n schema op te stellen wordt scheduling[1] genoemd. De termen schema en schedule zullen door elkaar gebruikt worden.
Praktische schedulingproblemen zijn volop aanwezig in allerlei verschillende varianten. Toch hebben ze één eigenschap gemeen, ze zijn lastig op te lossen waardoor er geen efficiënte oplosalgoritmes bestaan voor de meeste van dit soort problemen. Over het algemeen kan gezegd worden dat scheduling problemen tot de categorie NP-moeilijk behoren. Dit houdt in dat er geen enkel algoritme bestaat dat gegarandeerd de optimale oplossing geeft binnen polynomiale tijd. Het is zelfs één van de moeilijkste problemen om op te lossen binnen de klasse van NP-moeilijke problemen. Dit heeft ertoe geleid dat er veel technieken bedacht zijn op het gebied van AI en OR, waarbij zowel algemene benaderingsalgoritmes als probleemspecifieke exacte oplossingstechnieken ontwikkeld zijn. Benaderingsalgoritmes kunnen er wel toe bijdragen dat er een oplossing gevonden wordt, maar dit gaat meestal ten koste van de kwaliteit van de oplossing. De meeste van deze algoritmes berusten op lokale beslissingsregels. Dit verslag zal echter de rol van evolutionaire methoden uitlichten.

Er zal met name op het klassieke job shop scheduling probleem toegespitst worden. Dit is namelijk een alom bekend moeilijk op te lossen combinatorisch optimalisatieprobleem, waarbinnen de OR en AI verscheidene uitgangsposities uitgewerkt zijn. Conventionele oplosmethoden kunnen zelden een (bijna) optimale oplossing vinden binnen redelijke tijd. Vandaar dat een niet-conventionele aanpak, zoals evolutionaire methoden, wellicht zijn vruchten zal afwerpen. Mijn vraagstelling luidt daarom als volgt:

> *Kunnen evolutionaire methoden een toegevoegde waarde leveren aan het oplossen van job shop scheduling problemen?*

Mijn hypothese is dat evolutionaire methoden zeker een potentieel hebben om extra meerwaarde te leveren, zodat ze in de praktijk daadwerkelijk gebruikt kunnen worden voor scheduling vraagstukken. Maar in welke mate is echter onbekend. Bij het beantwoorden van deze vraag zal tevens gekeken worden naar de verschillende type job shop scheduling problemen (JSSPs).

Het verslag kan in drie verschillende onderdelen gesplitst worden. In hoofdstuk 2 en 3 zal stilgestaan worden bij het algemene kader van scheduling en in het specifiek bij job shop scheduling. Vervolgens worden oplosmethoden besproken. Allereerst zal er een kort overzicht gegeven worden van de belangrijkste conventionele methoden die ontwikkeld zijn in de OR (hoofdstuk 4). Vervolgens wordt het principe van lokaal zoeken uitgelegd. In dat kader worden evolutionaire methoden uitvoerig behandeld en besproken, zowel een algemene introductie (hoofdstuk 5) als de toepassing hiervan op het job shop scheduling probleem zullen aan bod komen (hoofdstuk 6 t/m hoofdstuk 9). Als derde en laatste onderdeel zal het gebruik van evolutionaire methoden bij scheduling vraagstukken getoetst worden in de praktijk (hoofdstuk 10) om uiteindelijk tot de slotconclusie te komen (hoofdstuk 11).

[1] In dit verslag zullen enkele engels talige termen aangehouden worden, aangezien de meeste literatuur in het engels gepubliceerd is en sommige engels talige termen zich niet lenen om vertaald te worden (en volledig geaccepteerd zijn).

2. Scheduling

Scheduling kan gezien worden als een probleem waarin gezocht wordt naar de optimale volgorde waarin en tijdstippen waarop een reeks operaties uitgevoerd moet worden onder een bepaalde reeks voorwaarden waaraan voldaan moet worden zodat het gewenste resultaat op tijd geleverd wordt. Een planner probeert de productiemiddelen altijd zo efficiënt en effectief mogelijk te benutten en ook de tijd waarin het gehele proces afgehandeld wordt te minimaliseren. Het is echter erg lastig om een optimaal schema vast te stellen. Een exacte methode heeft er alle moeite mee om met de optimale oplossing te komen binnen een redelijke tijd. Maar alvorens enkele oplossingsmethodieken aangedragen worden, zal in dit hoofdstuk kort besproken worden wat verstaan wordt onder scheduling en wat voor soorten scheduling vraagstukken er zijn.

2.1 Rol van scheduling in de supply chain

Samen met planning wordt scheduling gezien als de belangrijkste drijfsfeer voor de logistieke keten van een organisatie. Het grote verschil tussen planning en scheduling is de mate van detail. Daar waar planning nog op een meer geaggregeerd niveau werkzaam is, zal er op het scheduling niveau gedetailleerder te werk gegaan worden. Dit brengt met zich mee dat planning in data voorziet, die in de scheduling fase omgezet wordt in een daadwerkelijk productieschema. Bij het maken van een (globale) planning, spelen zaken als vraagvoorspelling en het voorzien in de productiemiddelen zoals materiaal, arbeid, machines etc. een belangrijke rol. Deze worden voor een langere tijdsperiode vastgesteld. Vervolgens kan er voor de korte termijn (dagbasis bijvoorbeeld) een schema opgesteld worden. Er is dus een goede afstemming en coördinatie tussen beide fasen noodzakelijk. Op het moment dat één van beide processen niet goed uitgevoerd wordt, zal dit tot gevolg hebben dat de bedrijfsvoering van de organisatie (gedeeltelijk) verstoord wordt.

2.2 Formulering scheduling problematiek

Scheduling theorie is een zich voortdurend uitbreidende tak van de OR. Traditioneel worden scheduling problemen geformuleerd in termen van het uitvoeren van taken op machines. Veel praktische problemen kunnen geformuleerd worden als scheduling problemen en als zodanig opgelost worden.
Voor het scheduling vraagstuk wordt aangenomen dat een verzameling machines M = {1, 2, ..., m} en een verzameling taken J = {1, 2, ..., n} tot de beschikking staan. De taken moeten worden uitgevoerd op de machines. Een taak kan weer opgesplitst worden in één of meerdere operaties, die ieder op een andere machine uitgevoerd dienen te worden. De tijd die nodig is om een willekeurige taak $j \in J$ uit te voeren op een willekeurige machine $k \in M$ is bekend. De tijd om de operatie o_{jk} uit te voeren zal bewerkingstijd genoemd worden en wordt aangegeven met p_{jk}.

2.3 Aannames scheduling problematiek

Gebruikelijke aannames zijn dat elke machine ten hoogste één taak tegelijk kan uitvoeren en dat elke taak op ten hoogste één machine tegelijk uitgevoerd kan worden. In veel praktische situaties kan een machine pas beginnen met het uitvoeren van een operatie nadat enige noodzakelijke voorbereidingen (een zogenaamde *setup*) verricht zijn. Zo kan het ook noodzakelijk zijn om na het uitvoeren van een operatie enig werk aan de betrokken machine te verrichten (een zogenaamde *removal*). In dit verslag wordt aangenomen dat deze activiteiten onafhankelijk zijn van de operaties die vooraf respectievelijk achteraf plaatsvinden. Dit heeft tot gevolg dat de tijd voor een *setup* en *removal* in de bewerkingstijd van een operatie geplaatst kan worden. Verder zijn er nog meer specifieke aannames die probleemafhankelijk zijn, deze zullen daar genoemd worden wanneer ze van toepassing zijn.

2.4 Soorten scheduling problemen

Er zijn een heleboel verschillende soorten scheduling problemen, die ieder hun eigen aannames hebben. Voor ieder van die scheduling vraagstukken is een andere oplostechniek

vereist. De notatie voor deze verschillende scheduling problemen bestaat uit een classificatieschema met de vorm α|β|γ. Graham e.a. (1979) hebben deze notatie geïntroduceerd en de letters beschrijven respectievelijk de productieomgeving, de karakteristieken van de operaties en de doelfunctie. De drie velden zullen hieronder toegelicht worden.

Veld α: de omgeving van de productiemiddelen

In de literatuur over scheduling worden de machines gespecificeerd met een reeks van twee parameters (α_1 α_2). De waarde van α_1 beschrijft de machineomgeving en α_2 staat voor het aantal machines in deze omgeving (als een parameter de waarde ° heeft of ontbreekt, dan is de parameter niet van toepassing). De verschillende waarden voor α_1 staan in tabel 2.1.

$\alpha_1 \in$ {F, J, O, X, S, I, U, R, PMPM, QMPM}	
°	enkele machine
F	flow shop
J	job shop
O	open shop
X	mixed shop
S	super shop
I	identieke machines
U	uniforme machines
R	ongerelateerde machines
PMPM	multi-purpose machine met identieke snelheid
QMPM	multi-purpose machine met uniforme snelheid

Tabel 2.1: De verschillende waarden voor de parameter α_1.

Het *flow shop* model is het bekendste van de shop modellen. In dit model wordt aangenomen dat elke taak uit een verzameling J de machine-doorlooproute *1, 2, ..., m* heeft. Dit betekent dat elke taak eerst dient te worden uitgevoerd op machine 1, dan op machine 2 enz., totdat de taak uitgevoerd is op de laatste machine (machine *m*).
Een algemener model staat bekend als *job shop*. In dit model wordt verondersteld dat elke taak een machine-doorlooproute heeft die van tevoren bekend is, maar die niet voor elke taak dezelfde hoeft te zijn. Elke taak kan zelfs een verschillende route volgen. Aangezien het merendeel van het verslag gaat over job shop scheduling zal de benaming taak vervangen worden door job. Voor de consistentie zal deze benaming pas in het volgende hoofdstuk toegepast worden.
Veel praktische situaties geven aanleiding tot de bestudering van het *open shop* model. In dit model dient elke taak weer op elke machine uitgevoerd te worden (tenzij de corresponderende uitvoeringstijd nihil is), maar de machine-doorlooproute is niet van tevoren bekend en moet zelfs nog worden vastgesteld.
Deze drie basismodellen kunnen worden gegeneraliseerd door enige of alle modellen te combineren met elkaar. Indien men bijvoorbeeld de *flow* en *open shop* modellen samenvoegt, dan resulteert dit in het *mixed shop* model. Het model dat alle drie de basismodellen in zich verenigt, wordt *super shop* genoemd. In dit model hebben sommige taken een vaste, en mogelijkerwijs verschillende, machine-doorlooproute terwijl voor andere taken een route gekozen dient te worden.
Een andere mogelijkheid is dat er, naast het in serie staan van de machines, meerdere identieke machines parallel aanwezig zijn. Dit wordt dan een *flexibele shop* genoemd en kan in combinatie met een *flow shop* (*FF*) of een *job shop* (*FJ*) plaatsvinden.

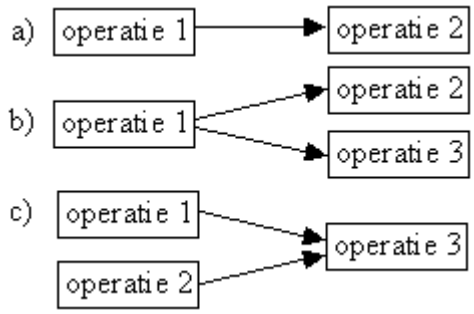
Naast de volgorde waarin de machines gekoppeld zijn aan de operaties, kan ook gespecificeerd worden wat voor soort machines er aanwezig zijn. Ten eerste kunnen het identieke machines zijn in de zin dat de bewerkingstijd voor iedere operatie identiek is (*I*). Ten tweede kunnen de machines een eigen productiesnelheid hebben, die onafhankelijk is van de operatie (*U*). Ten derde kan deze snelheid wel afhankelijk zijn van de operatie (*R*).

Veld β: de karakteristieken van de operaties

Naast de verschillende eigenschappen van de productieomgeving kunnen er ook enige extra restricties aan een schema toegevoegd worden, die betrekking hebben op de operaties.

$\beta \subset \{\beta_1, \beta_2, ...\}$

In de scheduling theorie wordt onderscheid gemaakt tussen preemptieve en niet-preemptieve schedules. In het eerste geval zijn geen onderbrekingen tijdens het uitvoeren van een operatie op een machine toegestaan. In het tweede geval zijn onderbrekingen wel toegestaan en kan de operatie later weer hervat worden op dezelfde machine.
Een schema heeft geen wachttijd tijdens uitvoering als elke taak, zodra deze gestart is op een machine, continu uitgevoerd wordt zonder enige wachttijd tussen de machines totdat de laatste machine de uitvoering voltooid heeft.
Als er precedentie relaties (volgorde relaties) gespecificeerd zijn, dan zijn de taken niet onafhankelijk van elkaar: deze relaties leggen vast dat sommige operaties vooraf gaan aan andere operaties. Op het moment dat er hoogstens één operatie vooraf en één operatie achteraf vereist is, spreekt men van een *chain*. Als er hoogstens één operatie vooraf vereist is, wordt dit een *outtree* genoemd en als er hoogstens één operatie achteraf vereist is heet dit een *intree*.



Figuur 2.1: De verschillende relaties tussen operaties: chain (a), outtree (b) en intree (c).

Verder kunnen een aantal eigenschappen gedefinieerd worden die de tijd specificeren waarin een operatie mag of moet plaatsvinden. Ten eerst kan de uitgiftedatum gespecificeerd worden voor een taak *j* (*release date* r_j). Dit geeft het tijdstip aan waarop de eerste operatie van een taak uitgevoerd mag worden. Ten tweede kan de vervaldatum of opleverdatum gespecificeerd worden van een taak (*due date* d_j). Dit geeft het tijdstip aan waarop de laatste operatie van een taak *j* uitgevoerd moet zijn. Ten derde kunnen de bewerkingstijden p_{ij} gedefinieerd worden, deze geven aan hoe lang de machine bezig is om operatie *i* van taak *j* uit te voeren.

$\beta \subset \{\beta_1, \beta_2, ...\}$	
pmtn	preemptief
no wait	operaties worden continu uitgevoerd als ze eenmaal begonnen zijn
prec	volgorde relaties
r_i	uitgiftedatum gedefinieerd (*release date*)
d_j	vervaldatum gedefinieerd (*due date*)
p_{ij}	bewerkingstijd voor een operatie

Tabel 2.2: De verschillende waarden voor de parameter β.

Veld γ: de doelfunctie

Voor scheduling problemen dient niet enkel een toegelaten oplossing gevonden te worden (wat op zichzelf al een veeleisende opgave kan zijn), maar vaak de beste toegelaten oplossing (een optimaal schema). Alvorens de verschillende doelfuncties geïntroduceerd kunnen worden, moeten er enige notaties gedefinieerd worden:

Notatie	
d_j = *due date* van taak j	
r_j = *release date* van taak j	
C_j = *completion time* van taak j	tijdstip waarop de laatste operatie van taak j klaar is
F_j = *flow time* van taak j	$C_j - r_j$
L_j = *lateness* van taak j	$C_j - d_j$
T_j = *tardiness* van taak j	$\max\{L_j, 0\}$
E_j = *earliness* van taak j	$\max\{0, -L_j\}$
U_j = indicatorfunctie of taak j te laat is	$\begin{cases} 1 & \text{als } C_j > d_j \\ 0 & \text{anders} \end{cases}$
w_j = gewicht van taak j	

Tabel 2.3: De notatie van de gegevens die betrekking hebben op een operatie.

In tabel 2.4 zijn een aantal veel gebruikte doelfuncties terug te vinden. De doelwaarde behorend bij een oplossing *x* wordt vaak aangegeven met *f(x)*. Met de doelfunctie kan bijvoorbeeld het gebruik van de machines gemaximaliseerd worden (min C_{max}, min C_{wt}) of de taken zo snel mogelijk uitgevoerd worden (min F_{wt}). Een andere mogelijkheid is dat de taken zo min mogelijk te laat klaar zijn (min L_{max}, min L_{wt}, min N_{wt}) en dat de taken juist zo goed mogelijk overeenkomen met het tijdstip waarop ze gewild zijn (min E_{max}, min E_{wt}). Dit laatste is bijvoorbeeld van belang in een just-in-time (JIT) situatie.

doelfunctie	definitie
C_{max} = makespan	$\max\{C_1, ..., C_n\}$
L_{max} = maximum lateness	$\max\{L_1, ..., L_n\}$
T_{max} = maximum tardiness	$\max\{T_1, ..., T_n\}$
E_{max} = maximum earliness	$\max\{E_1, ..., E_n\}$
C_{wt} = weighted completion time	w_jC_j
F_{wt} = weighted flow time	$w_j(F_j)$
L_{wt} = weighted lateness	w_jL_j
T_{wt} = weighted tardiness	w_jT_j
E_{wt} = weighted earliness	w_jE_j
N_{wt} = weighted number of tardy jobs	w_jU_j
ET_{wt} = weighted earliness + weighted tardiness	$w_j(E_j+T_j)$

Tabel 2.4: De verschillende waarde voor de parameter γ.

Over het algemeen worden problemen beschouwd waarbij de makespan C_{max} geminimaliseerd wordt, of anders gezegd, waarbij de totale verstreken tijd tussen het begin van de eerste operatie en het afronden van de laatste operatie (het tijdsbeslag van het schema) zo klein mogelijk moet zijn. Er worden wel andere maten voor de kwaliteit van een schema aangehouden, maar de makespan is het eenvoudigste en ook het meest gebruikte criterium.

2.5 Scheduling versus Timetabling

Zoals aan het begin van dit hoofdstuk vermeldt staat, kan scheduling gedefinieerd worden als het probleem waarin gezocht wordt naar de optimale volgorde waarin een reeks operaties uitgevoerd moet worden onder een reeks voorwaarden. Bij timetabling moet ook een planning gemaakt worden van de uit te voeren operaties op middelen die beperkt beschikbaar zijn, maar toch zijn er een aantal essentiële verschillen.
Bij scheduling is het bepalen van de volgorde waarin de operaties uitgevoerd moeten worden de belangrijkste doelstelling, terwijl bij timetabling het meer om het opvullen van timeslots gaat. Naast het type voorwaarden is de domeinkennis ook wezenlijk anders. Bij scheduling komt het planningsprobleem overeen met het inplannen van taken op meerdere machines, bestaande uit operaties. Bij timetabling daarentegen moet een rooster opgesteld worden dat aan harde voorwaarden moet voldoen, maar ook enkele zwakkere voorwaarden kent.

Er is in dit verslag voor gekozen om scheduling te onderzoeken, aangezien de voorwaarden eenduidig en iets ingewikkelder zijn. Daarnaast speelt het behoudt van volgorde relaties een grote rol en is dit een uitdaging voor evolutionaire methoden om hier op een efficiënte wijze mee om te gaan. Verder kunnen sommige timetabling situaties hergeformuleerd worden als scheduling vraagstukken.
In het specifiek zal job shop scheduling besproken worden. Dit soort problemen zullen in het volgende hoofdstuk uitvoerig geïntroduceerd worden.

3. Het job shop scheduling probleem (JSSP)

Beschouw een productieomgeving waarin *n* jobs vervaardigd moeten worden door *m* machines. Iedere job wordt daarom onderverdeeld in een aantal operaties die per machine gedefinieerd worden en deze operaties kennen een aantal voorwaarden over de volgorde waarin ze uitgevoerd moeten worden. Iedere operatie maakt gebruik van één van de *m* machines. Een machine kan slechts één operatie per keer uitvoeren en op het moment dat een operatie uitgevoerd wordt dan moet deze ook afgemaakt worden zonder onderbreking (niet-preemptief). Verder zijn alle jobs vanaf het begin beschikbaar op de werkvloer en zijn er geen opleverdata gedefinieerd. Het job shop scheduling probleem is het vinden van die volgorde van operaties op iedere machine, waarbij een bepaalde doelfunctie geminimaliseerd wordt (meestal de makespan C_{max}).
Om een volledig beeld te geven, staan hieronder nogmaals de belangrijkste aannames voor het JSSP:

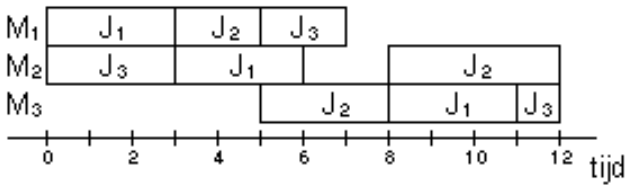
- een job bezoekt iedere machine maximaal één maal
- iedere machine kan één operatie per keer uitvoeren
- operaties kunnen niet halverwege gestopt worden
- er bestaat geen volgorde relatie tussen de operaties van de verschillende jobs
- *setup* en *removal* tijden zijn onafhankelijk van de vorige respectievelijk volgende operatie
- per type machine is er één aanwezig
- machines zijn ten alle tijden beschikbaar
- jobs kennen geen uitgifte- of vervaldatum

Aangezien het aantal jobs (*n*) en het aantal machines (mi) karakteristiek is voor de grootte van JSSP, zal vaak over een n×m JSSP gesproken worden. Een voorbeeld van een 3×3 JSSP wordt gegeven in tabel 3.1. De tabel bevat de routering van iedere job langs iedere machine en de bewerkingstijden voor ieder van deze operaties (tussen haakjes).

job	routering operaties (bewerkingstijd)		
J1	M1 (3)	M2 (3)	M3 (3)
J2	M1 (2)	M3 (3)	M2 (4)
J3	M2 (3)	M1 (2)	M3 (1)

Tabel 3.1: Een 3×3 job shop scheduling probleem, waarin zowel de volgorde relaties tussen de operaties als de bewerkingstijden vermeld staan.

Een mogelijke oplossing voor dit specifieke probleem wordt weergegeven in onderstaande Gantt chart. De Gantt chart is ontwikkeld door Henry L. Gantt in begin jaren 1990 als een eenvoudig visuele techniek om de job volgordes op de machines weer te geven.



Figuur 3.1: Een mogelijke oplossing voor het hierboven beschreven 3×3 JSSP in een Gantt Chart.

Alvorens het probleem grafisch en formeel geformuleerd wordt (in respectievelijk paragraaf 3.1 en 3.2) zal kort ingegaan worden op de verschillende type JSSPs: statisch, dynamisch, deterministisch en stochastisch.

Statisch versus dynamisch

Een statische productieomgeving houdt in dat alle informatie vooraf bekend is; Alle jobs kunnen vanaf het begin ingepland worden (tijdstip van uitgifte is 0, $r_j = 0 \ \forall \ j$) en de

vervaldatum speelt ook geen rol. In de praktijk zijn de meeste scheduling problemen echter meer dynamisch van aard, aangezien de productieomgeving onderhevig is aan allerlei niet te voorspellen gebeurtenissen, zoals een job die op een willekeurig moment vrijgegeven wordt, een machine die kapot gaat, jobs die geannuleerd worden, vervaldata en bewerkingstijden kunnen wijzigen etc. Door deze dynamische aard krijgen de scheduling problemen uit de praktijk een extra complexiteit ten opzichte van statische JSSPs.
De doelfunctie in een dynamisch JSSP is meestal anders dan het minimaliseren van de makespan, dit komt omdat de makespan alleen let op het afronden van de laatste job en niet op het tijdstip wanneer de overige jobs afgehandeld worden. Er zal daarom in de doelfunctie ook rekening gehouden worden met de uitgiftedatum en vervaldatum van jobs.

Deterministisch versus stochastisch

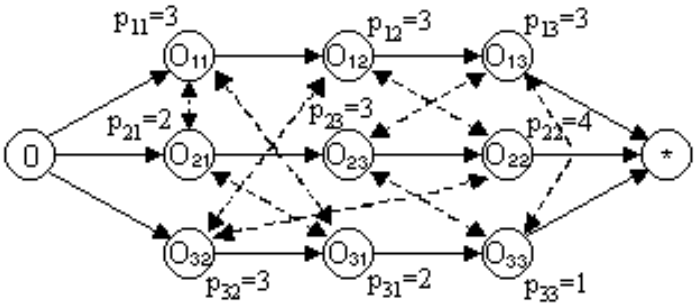
Op het moment dat het JSSP dynamisch van aard is, kan er binnen deze klasse onderscheid gemaakt worden tussen deterministische en stochastische JSSPs. Bij een deterministische JSSP is van tevoren bekend wanneer welke jobs vrijgegeven worden en wanneer hun vervaldatum is. Op het moment dat al deze informatie nog onbekend is op tijdstip 0, zal gesproken worden van een stochastisch JSSP.

3.1 Grafische notatie

Job shop scheduling problemen worden vaak beschreven in de vorm van een disjuncte graaf, G = (N, A, E), waarin:

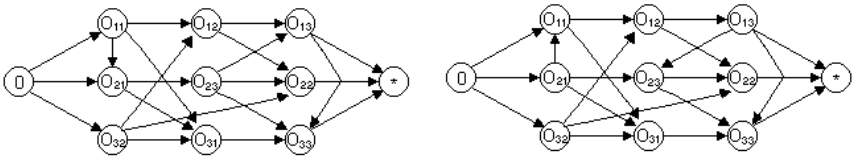
- N de reeks knopen is: De knopen N corresponderen met alle operaties en twee dummy knopen (een zogenaamde source (0) en een sink(*)). Iedere knoop krijgt een label, dat overeenkomt met de bewerkingstijd van de operatie.
- A de reeks conjuncte pijlen is: De gerichte (conjuncte) pijlen A representeren met de volgorde relaties tussen de operaties (i, j) van een bepaalde job. Er bestaat een pijl van punt *i* naar punt *j* dan en slechts dan als operatie *i* onmiddellijk voorafgaat aan operatie *j*.
- E de reeks disjuncte pijlen is: De ongerichte (disjuncte) pijlen E komen overeen met alle mogelijke paren van operaties die op dezelfde machine uitgevoerd kunnen worden.

De source heeft conjuncte pijlen naar alle eerste operaties van de jobs en de sink heeft conjuncte pijlen vanuit alle laatste operaties. De labels voor beide knopen bedragen nul. Een toelaatbare oplossing komt nu overeen met een selectie van precies één pijl uit ieder disjuncte pijlenpaar zodat de resulterende gerichte graaf acyclisch is. Het probleem waarin de makespan geminimaliseerd moet worden, komt overeen met het zoeken naar die reeks van disjuncte pijlen die de lengte van het langste pad minimaliseert. Dit pad wordt ook wel het kritieke pad genoemd. Iedere operatie in het kritieke pad wordt een kritieke operatie genoemd. Een kritieke pad kan worden gesplitst in een aantal blokken, waarin een blok de maximale aanéénvolging van kritieke operaties op dezelfde machine is. In figuur 3.2 is de 3×3 JSSP uit tabel 3.1 grafisch weergegeven.



Figuur 3.2: De grafische weergave van het probleem, met conjuncte pijlen (doorgetrokken) en disjuncte pijlen (gestippeld).

De oplossing zoals deze in figuur 3.1 weergegeven is, wordt in de eerste graaf van figuur 3.3 weergegeven. De afstand tussen twee schema's S en T kan gedefinieerd worden als het aantal verschillen in de volgorde waarin de operaties uitgevoerd worden. Dit komt overeen met het sommeren van alle disjuncte pijlen die een andere richting hebben. Deze afstand wordt ook wel de disjuncte graaf (DG) afstand genoemd.



Figuur 3.3: Twee mogelijke oplossingen zijn grafisch weergegeven. De DG-afstand tussen de twee oplossingen bedraagt twee, aangezien de pijl van O_{11} naar O_{21} en van O_{23} naar O_{13} omgekeerd zijn.

3.2 Formele notatie

Laat de reeks jobs aangeduid worden met J = {1, 2, ..., n} en de machines door M = {1,2, ..., m}. Dit levert n×m operaties, die weergegeven worden door de verzameling O = {1, 2, ..., n×m}. Vervolgens kunnen de verzamelingen N, E en A gedefinieerd worden zoals in de vorige sectie:

N = de reeks van alle operaties

E_k = alle paren van operaties die op dezelfde machine k uitgevoerd moeten worden

A = alle paren van operaties (i, j) waarin i voor j moet plaatsvinden (de volgorde relaties)

Vervolgens moeten de bewerkingstijden van operatie $i \in N$ gedefinieerd worden als p_i en de starttijden voor operatie $i \in N$ als t_i. Dan kan het JSSP als volgt gedefinieerd worden:

$$\begin{aligned} \min \quad & t_{n+1} \\ \text{onder} \quad & t_{n+1} \geq t_i + p_i && \forall i \in N \\ & t_j \geq t_i + p_i && \forall (i,j) \in A \\ & t_i \geq t_j + p_j \vee t_j \geq t_i + p_i && \forall (i,j) \in E_k, \forall k \in M \\ & t_i \geq 0 && \forall i \in N \end{aligned}$$

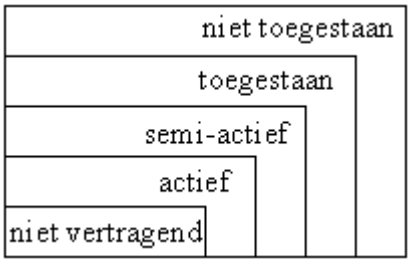
De grootste doorlooptijd om alle operaties uit te voeren moet dus geminimaliseerd worden, onder de voorwaarden dat de operaties behorend bij dezelfde job achter elkaar plaatsvinden volgens de volgorde relaties en dat alle machines maximaal één operatie per keer uitvoeren.

3.3 Oplossingsruimte

Er kunnen verschillende schema's in elkaar gezet worden, waarbij de ene beter is dan de andere. Naast de niet toelaatbare schema's kunnen de toelaatbare schema's geclassificeerd worden als de volgende types:

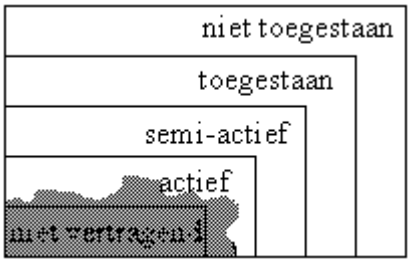
- semi-actieve schema's: deze schema's worden verkregen door de operaties zo vroeg mogelijk achter elkaar uit te voeren. Dus geen enkele operatie kan eerder uitgevoerd worden, zonder de volgorde te veranderen.
- actieve schema's: in deze schema's kan geen enkele operatie eerder uitgevoerd worden, zonder een andere operatie te vertragen of de volgorde relaties te schenden. De volgorde mag dus wel veranderd worden.
- niet-vertragende schema's: in deze schema's wordt geen enkele machine onnodig vrij gehouden als er een operatie uitgevoerd kan worden.

De verschillende classificaties zijn subtypes van elkaar en weergegeven in onderstaand figuur:



Figuur 3.4: De zoekruimte voor het job shop scheduling probleem kan onderverdeeld worden in een aantal (sub)klassen.

Een optimaal schema is altijd actief, hierbij wordt optimaal gedefinieerd in termen van minimale makespan. Dit heeft als voordeel dat de zoekruimte beperkt kan worden tot alleen de actieve schema's. Aangezien de zoekruimte (van actieve schema's) alsnog vrij groot is, kan ervoor gekozen worden om een maximum vertraging op de toelaatbare oplossingen te zetten. Als de maximum vertraging gelijk is aan 0, dan wordt de zoekruimte beperkt tot de niet-vertragende schema's. Met behulp van deze maximum vertraging wordt de zoekruimte beperkt, zoals weergegeven in onderstaand figuur.



Figuur 3.5: De zoekruimte voor het job shop scheduling probleem wordt beperkt tot oplossingen met een maximale vertraging.

Actieve schema's kunnen gegenereerd worden met het Giffler en Thompson (G&T) algoritme. Hiervoor worden eerst nieuwe notaties geïntroduceerd.

Notatie voor het Giffler & Thompson algoritme	
PS_t	een gedeelte van het schema bestaande uit t operaties
S_t	een reeks van operaties die ingepland kunnen worden tijdens iteratie t
r_{jk}	het vroegste tijdstip waarop operatie $o_{jk} \in S_t$ zou kunnen starten
p_{jk}	de verwerkingstijden van operatie o_{jk}
φ_{jk}	het vroegste tijdstip waarop operatie $o_{jk} \in S_t$ afgehandeld zou kunnen zijn
C_t	een reeks conflicterende operaties tijdens iteratie t

Tabel 3.2: Notaties die gehanteerd worden voor het Giffler en Thompson (G&T) algoritme.

De G&T procedure om een schema te construeren luidt als volgt:

G&T algoritme

stap 1. Zet t = 0, PS_t wordt een leeg schema, S_t wordt de reeks van alle operaties die als eerst uitgevoerd moeten worden voor de jobs en $r_{jk} = 0$ voor alle operaties $o_{jk} \in S_t$

stap 2. bepaal $\varphi_t^* = \min_{\{o_{jk} \in S_t\}} \{\varphi_{jk}\} = \min_{\{o_{jk} \in S_t\}} \{r_{jk} + p_{jk}\}$ en de machine k* die overeenkomt met de operatie waarop φ* gerealiseerd moet worden. Als er meerdere machines zijn, dan wordt er random één uitgekozen.

stap 3. Creëer een conflicterende reeks C_t waarin alle operaties $o_{jk} \in S_t$ zitten die op machine k* uitgevoerd moeten worden en waarvoor $r_{jk} < \varphi_t^*$ geldt.

stap 4. Kies een operatie uit de conflicterende reeks C_t volgens een prioriteitsregel en plaats deze operatie zo vroeg mogelijk in PS_t. Dit wordt het nieuwe schema PS_{t+1}. Als er meerdere operaties geselecteerd worden volgens de prioriteitregel, dan wordt er random één gekozen.

stap 5. Vernieuw de reeks S_t door de geselecteerde operatie hieruit te verwijderen en de volgende operatie van de job eraan toe te voegen. Bepaal de nieuwe waarden voor r_{jk} in S_t en hoog iteratie t met 1 op.

stap 6. Herhaal stap 2 tot en met stap 5 totdat alle operaties ingepland zijn en er dus een volledig schema gegenereerd is.

Het genereren van een niet-vertragend schema lijkt veel op het G&T algoritme. Er is echter een kleine aanpassing noodzakelijk, aangezien gekeken wordt naar de operaties die als eerst ingepland kunnen worden in plaats van de operaties die als eerst klaar zijn (in stap 2). Het gehele algoritme is terug te vinden in Appendix A.

3.4 Complexiteit

Een probleem is oplosbaar in polynomiale tijd indien er een algoritme bestaat dat een optimale oplossing vindt waarvan de looptijd begrensd is door een polynoom in de grootte van het probleem. De grootte van het probleem wordt bij job shop scheduling bepaald door het aantal operaties. Voor de meeste scheduling problemen, waaronder het JSSP, is geen polynomiaal algoritme bekend. Bovendien is voor het merendeel van deze problemen aangetoond dat ze NP-moeilijk zijn (Lenstra en Rinnooy Kan 1979), hetgeen inhoudt dat het onwaarschijnlijk is dat een polynomiaal algoritme bestaat.

Nu het probleem uitvoerig beschreven is, zal in het volgende hoofdstuk de voornaamste oplostechnieken behandeld worden.

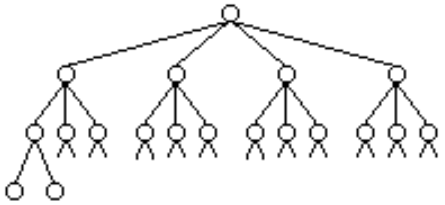
4. Oplossingstechnieken

De afgelopen decennia is er veel geschreven over scheduling problemen. Op het moment dat scheduling problemen ontstonden werd er door academici tegen aangekeken alsof het een wiskundig probleem was. Vandaar dat er veel onderzoek naar verricht is binnen de Operations Research. Om het probleem enigszins te modelleren en wiskundig op te lossen is het probleem erg vereenvoudigd (deterministisch, statisch, weinig machines en operaties). Door het handhaven van deze aannames werd het toepassingsgebied van de ontwikkelde technieken sterk verkleind. De kloof tussen theorie en praktijk werd te groot, waardoor veel praktische toepassingen uitbleven. Daarbij komt dat de rekentijd van de meeste scheduling technieken erg groot is, waardoor het toepassen van de technieken niet aantrekkelijk werd.
Pas in de jaren 1960 werden beide problemen erkend en werd er gezocht naar eventuele oplossingen. Wetenschappelijk onderzoek begon zich meer te richten op de problematiek uit de praktijk. Met behulp van mathematisch programmeren konden realistische scheduling problemen opgelost worden. Daarnaast werden heuristische zoekalgoritmes geïntroduceerd, die (bijna) optimale oplossingen vonden. Deze ontwikkelingen mochten echter niet baten, want hun invloed bleek in de praktijk tegen te vallen door de te complexe en instabiele productieomgeving. Daarnaast speelt de afhankelijkheid met andere functies binnen de organisatie een rol, evenals de invloed van de mens. Deze laatste factor is zelfs onvermijdelijk en onoverbrugbaar, aangezien bedrijven van mening zijn dat een scheduling techniek de mens moet ondersteunen in plaats van vervangen. Dit komt voornamelijk doordat de mens een hoop kennis en ervaring heeft om snel om te kunnen gaan met verstoringen in de productieomgeving.

Deterministische JSSPs kunnen opgelost worden met een exacte of heuristische aanpak. Exacte methoden zijn echter alleen toepasbaar bij het oplossen van kleine probleeminstanties. Deze zullen in paragraaf 4.1 besproken worden. De zoekruimte van mogelijke oplossingen neemt namelijk exponentieel toe met de grootte van het probleem (NP-moeilijk). Een 10x10 probleem is hierbij nog wel oplosbaar, maar een 15x15 probleem valt alweer buiten het bereik van de (tot nu toe) bekende methoden. Voor zulke problemen bestaat de behoeft aan een goede heuristiek. Deze oplosmethoden zullen in paragraaf 4.2 t/m paragraaf 4.4 uitgelicht worden. Enkele voorbeelden hiervan zijn snelle prioriteitsregels, de shifting bottleneck aanpak en lokale zoekmethoden.

4.1 Exacte methoden

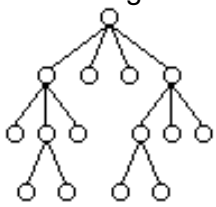
Een bruikbare methode om veel combinatorische problemen op te lossen is de branch-and-bound (B&B) methode (Carlier, 1982 en Florian, Trepant en McMahon, 1971, en Lageweg, Lenstra en Rinnooy Kan, 1977). Zoals de naam al doet vermoeden, bestaat deze methode uit twee fundamentele procedures. Branching is het proces waarin een groot probleem opgesplitst wordt in twee of meerdere subproblemen en bounding is het proces waarin een grens berekend wordt op de optimale oplossing van een gegeven subprobleem. Op het moment dat een bepaalde oplossing boven deze grens komt, dan is het niet zinvol om de desbetreffende vertakking verder te onderzoeken. Dit vertalingproces is weergegeven in figuur 4.1.



Figuur 4.1: De boomstructuur van het branch-and-bound algoritme bestaat uit vertakkingen van oplossingen, waarin bij iedere vertakking een extra restrictie opgelegd wordt aan het probleem.

Het nadeel van de B&B methode is dat de rekentijd exponentieel groeit met de grootte van het probleem. Vandaar dat er ook allerlei manieren ontwikkeld zijn om het aantal

vertakkingen te beperken. Zo bestaat de branch-and-price methode (branch-and-bound in combinatie met kolomgeneratie) en de branch-and-cut methode (branch-and-bound in combinatie met cutting planes). Een vrij recente ontwikkeling is beam search, waarbij er een maximum zit aan het aantal mogelijke vertakkingen.



Figuur 4.2: De boomstructuur bij beam search, waarbij maximaal 2 knopen mogen vertakken per niveau.

Een andere exacte methode is dynamisch programmeren (King en Spachis, 1980). Ook deze methode heeft het nadeel dat veel probleeminstanties onderzocht moeten worden. Toch is deze methode efficiënter dan de complete enumeratie van alle mogelijke volgordes, omdat dynamisch programmeren bepaalde volgordes indirect aanschouwt zonder deze echt te evalueren.

Er kan dus geconcludeerd worden dat een exacte methode volgens een mathematisch programmering aanpak tot weinig goede resultaten leidt, aangezien het probleem al gauw te groot wordt om opgelost te worden binnen redelijke tijd.

4.2 Heuristische methoden

Heuristische methoden zijn van speciaal belang voor praktische toepassingen. De meest gebruikte heuristieken berusten op prioriteitsregels, waarin schema's geconstrueerd worden aan de hand van de jobs met de hoogste prioriteit. Deze prioriteit kan afhangen van een parameter, die gerelateerd is aan de jobs, de operaties of de machines zoals de bewerkingstijden, vervaldata, uitgiftedata en de load op de machine. Het grote voordeel van prioriteitregels is dat ze eenvoudig te implementeren zijn en geen complexiteit met zich meebrengen. Het algoritme van Giffler en Thompson (1960) kan gezien worden als de basis voor alle prioriteitregels gebaseerde heuristieken.

regel	prioriteit
RANDOM	een operatie wordt random gekozen
FCFS (first come first serve)	$1/r_j$
WSPT (weighted shortest processing time)	w_j/p_{jm}
WLWKR (weighted least work remaining)	w_j/R_j
WTWORK (weighted total work)	w_j/P_j
EGD (earliest global due date)	$1/d_j$
EOD (ealiest operation due date)	$1/[r_j+(d_j-r_j)R_j/P_j]$
MST (minimum slack time)	$-(d_j-R_j-t)$
WS/OP (weighted slack per operation)	$w_j[1-(d_j-R_j-t)/n_j]/p_{jm}$

Tabel 4.1: Veel gebruikte prioriteitsregels.

De shifting bottleneck heuristiek (Adams, Balas en Zawack(1988)) is waarschijnlijk de meest krachtige procedure om het JSSP op te lossen. Het idee is om voor iedere machine een 1-machine scheduling probleem op te lossen waarbij rekening gehouden moet worden met een bestaand schema. Dit 1-machine scheduling probleem is ook NP-moeilijk, maar met een B&B methode kan deze bepaald worden. De shifting bottleneck heuristiek is dus een iteratief proces, waarbij iedere iteratie uit twee subroutines bestaat. In de eerste subroutine (SB1) wordt bepaald welke machine, van de reeds nog niet ingeplande machines, de bottleneck is en vervolgens moet de volgorde van de operaties op die machine bepaald worden. Terwijl bij de tweede subroutine (SB2) er opnieuw geoptimaliseerd wordt, maar dan onder de reeds ingeplande machines.

Shifting Bottleneck

stap 0. $M_0 = \varnothing$

stap 1. Identificeer de bottleneck machine *m* tussen de machines $M \setminus M_0$ en bepaal de optimale volgorde van operaties op deze machine, gegeven het gedeeltelijke schema van de machines M_0. $M_0 = M_0 + \{m\}$

stap 2. Ga achtereenvolgens de volgorde per machine $m \in M_0$ opnieuw optimaliseren, gegeven het gedeeltelijke schema van de machines $M_0 \setminus \{m\}$.

stap 3. herhaal stap 1 en 2 totdat M_0 gelijk is aan M, dan zijn alle machines ingepland.

4.3 Simulatie

Het gebruik van simulatie is niet echt gebruikelijk voor scheduling vraagstukken, ondanks dat ze vrije realistische systemen kunnen representeren in redelijke berekeningstijd. Daarnaast heeft simulatie het voordeel dat een natuurlijke aanpak gebruikt kan worden met menselijke expertise. Maar de resultaten zijn echter alles behalve optimaal, waardoor deze aanpak niet binnen de mogelijkheden valt.

4.4 Zoekmethoden (neighborhood search)

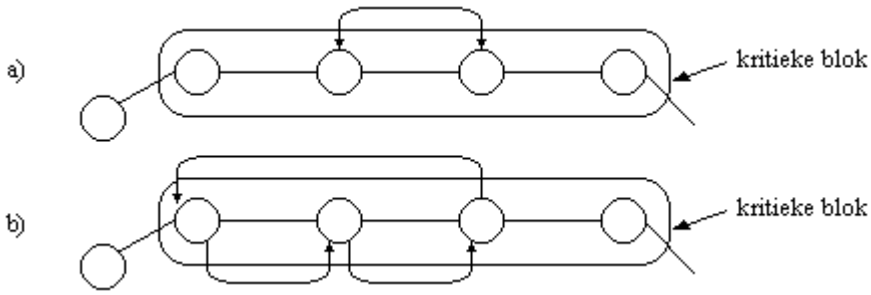
Er bestaan een heleboel zoekmethoden, waarbij de aanpak nogal verschillend is. Allereerst zal er stil gestaan worden bij de meer conventionele zoekmethoden en daarna zal neighborhood search uitgelegd worden.

Er zijn een aantal traditionele zoekmethoden, zoals

- calculus gebaseerde methoden: deze methoden maken gebruik van de gradiëntfunctie om de zoekrichting te bepalen,
- enumeratie methoden: deze methoden bekijken de doelwaarde van alle punten in de zoekruimte (de branch-and-bound methode en dynamisch programmeren zijn hier voorbeelden van) en
- willekeurige zoekmethoden: deze methoden doorlopen de zoekruimte willekeurig op zoek naar de beste oplossing.

De laatste twee methoden lijken niet echt geschikt voor scheduling problemen, aangezien de zoekruimte te groot is. De op calculus beruste methode maakt gebruik van de aanname dat er een afgeleidenfunctie gedefinieerd is voor het zoekgebied. Dit is echter niet bekend voor scheduling problemen. Daarbij komt dat het zoekgebied veel verschillende lokale optima kan bevatten, waardoor deze methode óf vast gaat lopen in een lokaal optimum óf nooit zal eindigen.

Om het job shop scheduling probleem op te lossen, zal er toch een ander soort zoekmethode gedefinieerd moeten worden. *Neighborhood search* is hiervoor een goede techniek. Bij *neighborhood search* wordt een oplossing *x* gerepresenteerd als een punt in de zoekruimte en een reeks van oplossingen verbonden met *x* wordt gedefinieerd als *neighborhood N(x)*. Dit komt overeen met een reeks van toelaatbare oplossingen die bereikt kunnen worden vanuit *x* door precies één wijziging uit te voeren op *x*. Twee veel gebruikte manieren om deze overgang te verwezenlijken bij job shop scheduling, is de *adjacent swapping* (AS) neighborhood en de *critical block* (CB) neighborhood. Bij de eerste methode komen de buren van een oplossing *x* overeen met alle oplossingen waarvan twee opeenvolgende operaties omgewisseld zijn, binnen hetzelfde kritieke blok. Terwijl bij de laatst genoemde methode de buren overeenkomen met alle oplossingen waarvan een operatie in een kritiek blok naar het begin of eind verplaatst is.



Figuur 4.3: Het bepalen van buren via adjacent swapping neighborhood (a) of via critical block neighborhood (b).

Als de buren *N(x)* gedefinieerd zijn, kan vervolgens gezocht worden naar een betere oplossing (zie onderstaand algoritme).

Neighboorhood search

stap 1. selecteer een startpunt $x = x_0 = x_{best}$
stap 2. als $f(x) < f(x_{best})$ dan $x_{best} = x$
stap 3. genereer en evalueer alle volgordes in de neighborhood N(x) van de oplossing x
stap 4. selecteer een punt $y \in N(x)$ volgens een bepaald criterium gebaseerd op f(y) en zet x = y
stap 5. herhaal stap 2 tot en met stap 4 totdat een bepaald stopcriterium voldaan is

De verschillende zoekmethoden onderscheiden zich door de manier waarop de buren bepaald worden en waarop ze een nieuw punt *y* uit de buren N(*x*) kiezen. Hieronder zullen een aantal veel gebruikte zoekmethode beschreven worden, die berusten op het *neighborhood search* principe: De *steepest ascent* (*descent)* methode selecteert het beste punt *y* uit alle buren N(*x*), de *greedy ascent* methode selecteer het eerste punt dat verbetering oplevert, terwijl bij *simulated annealing* een punt probabilistisch geselecteerd wordt en bij *tabu search* juist het beste punt dat niet op de tabu lijst staat. Deze laatste twee methoden zullen hieronder iets nader uitgelicht worden, aangezien ze veelvuldig toegepast worden bij het oplossen van het JSSP.

Simulated Annealing (SA)

Deze techniek is ontwikkeld door Kirkpatrick (1982) en kent zijn oorsprong in de fysica. Er wordt begonnen met een willekeurig punt in de zoekruimte en dan wordt er willekeurig bewogen in deze ruimte. Als het nieuwe punt een betere oplossing is, dan wordt deze geaccepteerd. Anders wordt het punt met een bepaalde kans p(t) geaccepteerd. Deze kans hangt af van het tijdstip en zal in het begin groot zijn (bijna 1), maar gaandeweg zal deze dalen tot ongeveer 0.

```
selecteer een individu i ∈ S (de zoekruimte)
selecteer een begin temperatuur T > 0
zet teller t = 0
herhaal zolang het stopcriterium niet voldoet
        zet teller n = 0
        herhaal totdat n = N(t)
                genereer individu j ∈ N(t)
                δ = f(j) - f(i)
                als δ < 0 dan i = j
                anders als een willekeurig getal ∈ [0,1) < exp(-δ/T) dan i = j
                n = n + 1
        eind
        t = t + 1
        T = T(t)
Eind
```

Algoritme 4.1: Het raamwerk voor simulated annealing.

Van Laarhoven, Aarts en Lenstra (1992) hebben gekeken in hoeverre simulated annealing toegepast kon worden op het job shop probleem. Zij kwamen tot de conclusie dat het probleem wel degelijk tot één van de toepassingsgebieden van SA kan behoren. In vergelijking met de shifting bottleneck aanpak presteerde SA beter.

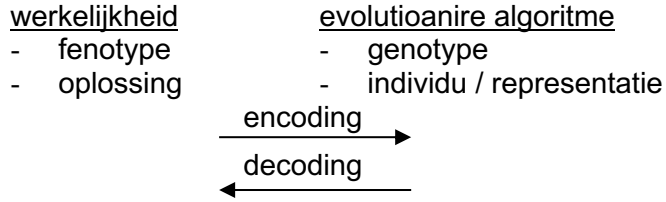
Tabu Search (TS)

Bij *tabu search* wordt in principe gebruik gemaakt van de *descent* methode, maar dan met het verschil dat een aantal buren tabu zijn en dus niet gekozen mogen worden als startoplossing in een volgende iteratie bij de *neighborhood search* methode. De oplossingen die op de lijst voorkomen zijn voornamelijk de laatste stappen die gezet zijn in de zoekruimte. Dit heeft als voordeel dat de methode altijd op zoek moet gaan naar andere buren en dus niet vast blijft zitten in een lokaal optimum. Deze is immers tabu geworden.

Evolutionaire algoritmes zijn ook lokale zoekmethoden. Het principe hiervan zal in het volgende hoofdstuk besproken worden.

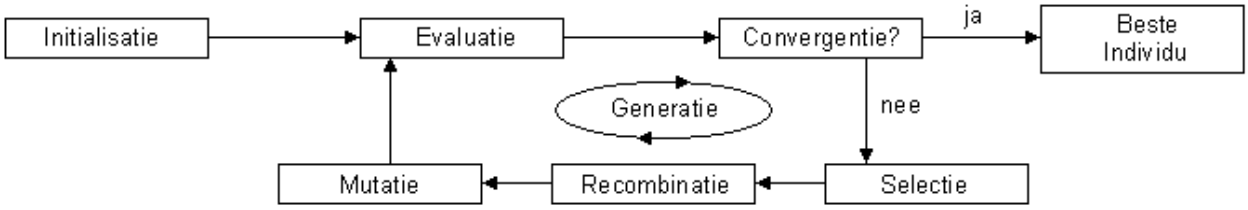
5. Evolutionaire algoritmes

Evolutionaire algoritmes (EAs) zijn stochastische optimalisatietechnieken, die gebaseerd zijn op de evolutieleer en het principe van survival of the fittest (Charles Darwin, The Origin of Species). Deze klasse van algoritmes legt namelijk de nadruk op het veranderen van oplossingen dat analoog is aan de biologische evolutie.
Een EA berust op een verzameling representaties (genotype) van kandidaat oplossingen voor het probleem (fenotype). Iedere kandidaat oplossing in een EA wordt een individu genoemd en alle individuen samen vormen de populatie.



Figuur 5.1: De werkelijkheid versus de representatie van een probleem in een evolutionair algoritme.

De populatie evolueert vervolgens via een reeks stochastische operatoren. Uit de populatie worden namelijk individuen geselecteerd om als ouder te fungeren en nadat de genetische operatoren de ouders veranderd hebben, ontstaat er het nageslacht (de kinderen). Vervolgens wordt bepaald welke individuen overleven en de populatie van de nieuwe generatie te vormen.



Figuur 5.2: Het raamwerk van een evolutionaire algoritme.

De evolutionaire aanpak is een sterke zoek- en optimalisatietechniek die in het bijzonder geschikt is voor problemen waarvan de zoekruimte van mogelijke oplossingen te groot is om met een traditionele techniek op te lossen. Daarnaast is deze methode uitermate geschikt wanneer de structuur van het probleem onderhevig is aan veranderingen of wanneer de parameters regelmatig wijzigen.

Bij het gebruik van evolutionaire algoritmes voor optimalisatie problemen wordt uitgegaan van de basisaanname dat optimale oplossingen gevonden kunnen worden door gebruiken te maken van geschikte eigenschappen uit de suboptimale oplossingen.

Alvorens alle aparte onderdelen besproken worden (in paragraaf 5.3 t/m paragraaf 5.6), zal in paragraaf 5.1 de historie van evolutionaire methoden behandeld worden. Vervolgens zullen de verschillende classificaties binnen evolutionaire algoritmes besproken worden (in paragraaf 5.7) en uiteindelijk vergelijk worden met lokale zoekmethoden (in paragraaf 5.8).

5.1 Historie

De eerste evolutionaire algoritmes (of in ieder geval voorstellen die die richting op gingen) dateren uit midden jaren 1960, toen John Holland genetische algoritmes (GAs) introduceerde in 1960 en Ingo Rechenberg begon met het ontwikkelen van ideeën over Evolutionaire Strategieën (ES). Het werk van deze pioniers heeft ertoe geleid dat veel optimalisatie methoden ontstonden die geschikt waren voor ingewikkelde problemen waarvan weinig bekend is over de onderliggende zoekruimte. In 1992 maakte John Koza gebruik van GAs bij het ontwikkelen van programma's die gericht zijn om een specifieke taak uit te voeren. Hij

noemde deze methode genetisch programmeren (GP). De verschillen tussen deze methoden zullen in paragraaf 5.7 besproken worden, maar eerst wordt het mechanisme achter evolutionaire algoritmes verder uitgewerkt.

5.2 Mechanisme

Ruwweg kan gezegd worden dat een evolutionaire algoritme ten doel heeft om een (bijna) optimale oplossing te vinden door een reeks toegelaten oplossingen te transformeren waarbij een selectieschema aangehouden wordt, die de voorkeur geeft aan hoge kwaliteit oplossingen. De mate waarin deze voorkeur bestaat, is variërend.
De transformaties bestaan uit een recombinatie stap en een mutatie stap. Het doel van de recombinatie stap is het identificeren van goede eigenschappen en die combineren, terwijl mutatie juist voor een willekeurige verandering zorgt.

```
initialiseer populatie P_t
evalueer populatie P_t
herhaal zolang het stopcriterium niet voldoet
    selecteer ouders uit populatie P_t
    recombinatie van ouders en stop in populatie
    P_t+1
    mutatie van populatie P_t+1
    evalueer populatie P_t+1
    selecteer overlevenden
    P_t = P_t+1
eind
```

Algoritme 5.1: Het raamwerk van een evolutionaire methode

De pseudo-code in algoritme 5.1 is erg algemeen. Er bestaan daarom veel verschillende varianten voor de verschillende problemen die opgelost kunnen worden met een EA. De verschillende stappen zullen daarom hieronder in meer detail behandeld worden.

5.3 Representatie

Alvorens een genetisch algoritme uitgevoerd kan worden, moet het probleem gecodeerd (of gerepresenteerd) worden. Beter gezegd, iedere oplossing van het probleem moet op een bepaalde manier gecodeerd worden. De oplossing wordt gedefinieerd als fenotype en de gecodeerde representatie als genotype (zie figuur 5.1). Het fenotype stelt dus de werkelijke oplossing voor, terwijl het genotype evolueert. Een enkel individu uit de populatie (een instantie van het genotype) wordt ook wel chromosoom genoemd en bestaat uit allerlei genen.

chromosoom = (gen1, gen2, ...)

Er zijn echter meerdere mogelijkheden als representatie voor een probleem. De keuze van deze representatie is sterk probleemafhankelijk, aangezien het alle informatie moet bevatten om een oplossing van het probleem weer te geven.

De meest gebruikte representatie is de binaire representatie, waarin een chromosoom overeenkomt met een reeks 0'en en 1'en:

chromosoom 1: 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0
chromosoom 2: 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0

Er zijn uiteraard nog vele andere manieren om een oplossing te representeren, zoals permutaties, reële waarden en boomstructuren. Permutaties worden vaak gebruik wanneer er sprake is van een volgorde die gedefinieerd moet worden, zoals bij het JSSP.
Vervolgens kan er aan ieder individu een kwaliteitswaarde toegekend worden met behulp van een evaluatiefunctie (ook wel doel- of fitness functie genoemd).

5.4 Selectie ouders (reproductie)

De bedoeling van een evolutionaire algoritme is om de populatie beter te maken over meerdere generaties, waarin beter gedefinieerd wordt als een lagere gemiddelde doelwaarde van de populatie. Vandaar dat de ouderselectie meestal plaatsvindt op basis van de fitness waarde. Dit selectieproces wordt ook wel reproductie genoemd. Er zijn echter velr methoden om de ouders te selecteren, die berusten op het principe van *survival of the fittest*. Hieronder worden de belangrijkste besproken.

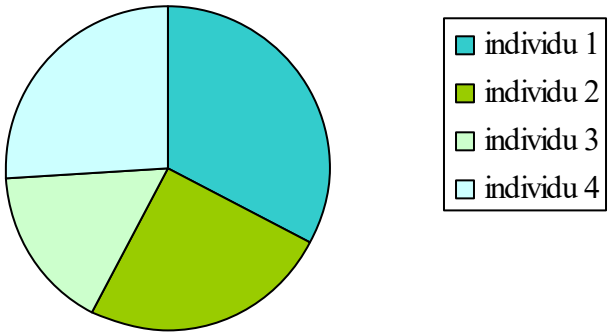
roulette wheel selectie

De ouders worden gekozen op basis van een kans om geselecteerd te worden. Die kans is proportioneel aan de doelwaarde die de individuen uit de populatie bezitten. Stel een roulette wiel (taartdiagram) voor waarin alle individuen van de populatie geplaatst worden, de ruimte die ze innemen is afhankelijk van hun doelwaarde. Een draai aan dit wiel bepaalt welk individu geselecteerd wordt. Een chromosoom met een betere doelwaarde wordt dan eerder gekozen, omdat deze meer ruimte inneemt op het rad. Dit is goed te zien in het voorbeeld hieronder.
Neem de situatie waarin de bevolking bestaat uit vier individuen en dat hun makespan C_{max} gedefinieerd is zoals in tabel 5.1 weergegeven staat.

individu	makespan (C_{max})	$1/C_{max}$	% van totaal
1	12	0,0833	32,8
2	16	0,0625	24,6
3	24	0,0417	16,4
4	15	0,1067	26,2
		0,2542	100

Tabel 5.1: De individuen inclusief de procentuele ruimte die zij innemen op het roulette wheel.

Om de proportionele selectiekansen te bepalen, wordt eerst de inverse van de doelfunctie (makespan) genomen en vervolgens wordt deze genormeerd. Dat levert uiteindelijk de kansen op zoals deze in kolom 4 van tabel 5.1 weergegeven staan. Individu 1 heeft de laagste makespan en dus de grootste kans om gekozen te worden als ouder. Dit is ook grafisch goed te zien in figuur 5.3.



Figuur 5.3: De ruimte die de vier individuen innemen op het rad bij roulette wheel selectie.

Het nadeel van deze methode is dat als een klein aantal individuen een relatief hoge doelwaarden bezit, dan zullen die individuen erg vaak gekozen worden als ouder.

rank selectie

Bij deze selectiemethode wordt eerst de populatie gerangschikt naar hun doelwaarde en volgens deze rangschikking krijgen de individuen een nieuwe doelwaarde toegewezen. Het slechtste individu krijgt waarde 1, het op een na slechtste individu de waarde 2 etc. Vervolgens kan deze rangschikking vertaald worden naar bepaalde kansen. Dit kan op veel

manieren plaatsvinden, zoals bijvoorbeeld een lineaire of exponentiële functie. Ook zou de roulette wheel selectie weer toegepast kunnen worden, maar nu op de nieuwe doelwaarden. Het schalen van de doelwaarde zorgt ervoor dat het nadeel van de 'pure' roulette wheel selectie weggenomen wordt.

tournament selectie

De q-tournament selectie methode selecteert een individu door eerst willekeurig q individuen uit de gehele populatie te kiezen en vervolgens wordt de beste van deze q individuen geselecteerd.

5.5 Genetische operatoren

De geselecteerde ouders moeten vervolgens voor het nageslacht zorgen door recombinatie en mutatie. Bij recombinatie vindt er een uitwisseling plaats van een willekeurig gekozen gedeelte(s) van het chromosoom en bij mutatie een verandering van een willekeurig gekozen gedeelte van het chromosoom. De bedoeling van de recombinatie operator is het verhogen van de kwaliteit van de bevolking, zodat er uiteindelijk convergentie[2] optreedt. De mutatie operator moet daarentegen zorgen voor diversiteit. Beide operaties hoeven niet per definitie uitgevoerd te worden, maar kunnen ook met een bepaalde kans of helemaal niet plaatsvinden.

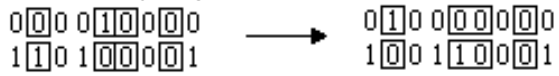
Recombinatie

De representatie van een individu is sterk bepalend wat voor soort recombinatie operator gebruikt kan worden. Er zijn verschillende mogelijkheden. Hieronder zullen de meest gebruikte operatoren kort behandeld worden. Een veel gebruikte benaming voor recombinatie is crossover.

Voor de binaire representatie wordt vaak gebruik gemaakt van de N-point crossover. Hierbij worden willekeurig N punten gekozen en de tussenliggende delen van de individuen worden vervolgens onderling uitgewisseld.

Figuur 5.4: Bij 2-point crossover worden de genwaarden tussen het geselecteerde gedeelte omgewisseld.

Een andere veel gebruikte methode voor de binaire representatie is de uniform crossover (Spears and DeJong (1991)), waarin per gen een kans (al dan niet bevooroordeeld) bepaald welke ouder de genwaarde zal bijdragen.



Figuur 5.5: Bij uniform crossover worden de genwaarden van de geselecteerde gedeelten omgewisseld op basis van kansen.

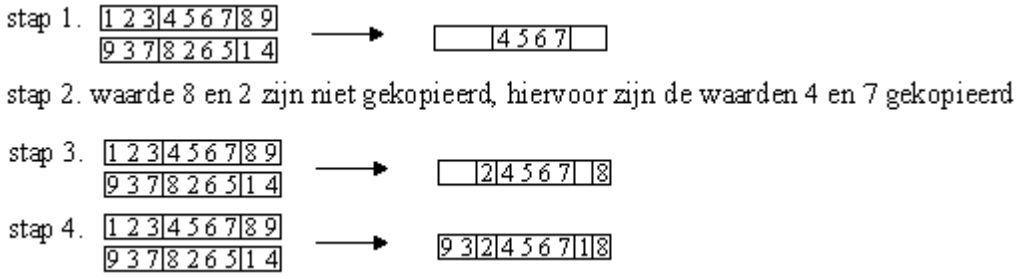
Voor permutatie representaties is dit echter een stuk minder eenvoudig, aangezien iedere genwaarde slechts eenmaal mag voorkomen. Hieronder zullen de meest gebruikte methoden besproken worden.

De Partially Mapped Crossover (PMX) is geïntroduceerd door Goldberg en Lingle en werkt als volgt:

Partially Mapped Crossover (PMX)

[2] Onder convergentie wordt verstaan dat alle genen geconvergeerd zijn en er treedt convergentie bij een gen op als minimaal 95% van de bevolking dezelfde waarde voor die gen heeft.

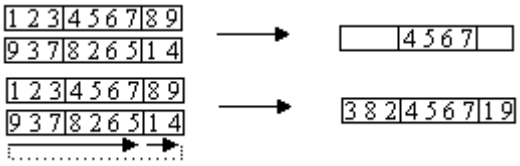
stap 1. kies twee willekeurige posities en kopieer het tussenliggende gedeelte van ouder 1 naar het kind
stap 2. bekijk de niet gekopieerde elementen in het tussenliggende gedeelte van ouder 2 en bepaal welke elementen hiervoor in de plaats zijn gekomen en zoek die op in ouder 1.
stap 3. plaats de niet gekopieerde elementen van ouder 2 naar die posities die reeds gekopieerd zijn van ouder 1
stap 4. de rest van het elementen kan vervolgens gekopieerd worden van ouder 2



Figuur 5.6: Een stapsgewijze uitleg van Partially Mapped Crossover aan de hand van een voorbeeld.

Een andere recombinatie operator voor de permutatie representatie is de order crossover, deze is ontwikkeld door Davis. Deze operator begint op dezelfde wijze als PMX, maar gaat vervolgens verder op een manier zodat de informatie over de relatieve volgorde van de tweede ouder overgedragen wordt.

Order crossover
stap 1. kies twee willekeurige posities en kopieer het tussenliggende gedeelte van ouder 1 naar het kind
stap 2. bekijk de niet gekopieerde elementen van ouder 2, beginnend na het geselecteerde gedeelte en plaats de niet gekopieerde elementen achter het reeds gekopieerde gedeelte in het kind.

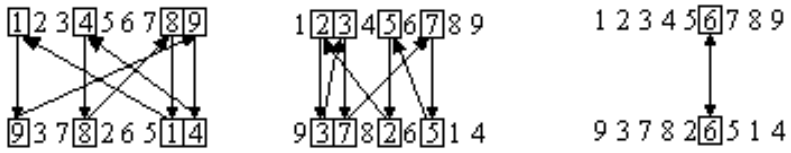


Figuur 5.7: Bij order crossover wordt eerst een gedeelte gekopieerd van de ene ouder en vervolgens wordt de andere ouder doorlopen om het individu aan te vullen, waarbij dubbele waarden worden overgeslagen.

Een andere mogelijkheid is, dat vooraan wordt begonnen in stap 2 (dus bij '9' in figuur 5.7).

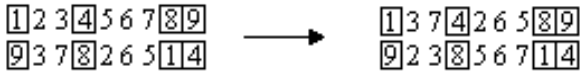
De laatste recombinatie operator die hier besproken wordt, is de cycle crossover, waarin informatie over de absolute positie van ieder element overgedragen wordt. Eerst worden de cycles geïdentificeerd. Een cycle is een deelverzameling van elementen met de eigenschap dat ieder element gepaard voorkomt met een ander element uit dezelfde cycle. Deze worden als volgt gevonden:

Cycle crossover: zoek de cycles
Stap 1. Begin met de eerste niet gebruikte positie uit ouder 1 en voeg deze toe aan de cycle
Stap 2. bekijk de waarde van het gen op dezelfde positie in ouder 2 en voeg deze toe aan de cycle
Stap 3. zoek deze waarde ook in ouder 1
Stap 4. herhaal stap 2 en stap 3 totdat de cycle bij de eerst toegevoegde waarde uitkomt



Figuur 5.8: De gevonden cycles tussen de twee individuen.

Vervolgens worden de kinderen gecreëerd door posities van de cycle te verwisselen.



Figuur 5.9: Bij cycle crossover worden de waarden van de cycles omgewisseld tussen de ouders.

Mutatie

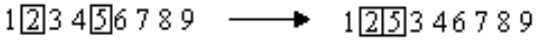
Voor een binaire representatie kan willekeurig een gen uitgekozen worden om die van waarde te veranderen, dit wordt ook wel *bitwise* mutatie genoemd.

Figuur 5.10: Bij bitwise mutatie wordt random een (binaire) waarde gewijzigd.

Voor permutaties is dit echter niet toegestaan. Voor die representatie bestaan andere mutatie operatoren. Zoals de swap mutatie, waarin willekeurig twee posities gekozen worden om verwisseld te worden.

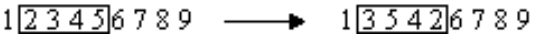
Figuur 5.11: Bij swap mutatie worden waarden omgedraaid.

Een andere mutatie is de insertie mutatie, waarin een willekeurig gen naast een andere gen geplaatst wordt.



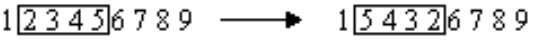
Figuur 5.12: Bij insertie mutatie worden waarden naar elkaar toe gebracht.

Bij de scramble mutatie worden alle waarden binnen een bepaald geselecteerd gedeelte van een individu verwisseld.



Figuur 5.13: Bij scramble mutatie worden de waarden binnen een geselecteerd gedeelte willekeurig verplaatst.

Bij inversie mutatie worden alle waarden binnen een bepaald gedeelte van een individu omgekeerd.



Figuur 5.14: Bij inversie mutatie worden de waarden binnen een geselecteerd gedeelte omgekeerd.

5.6 Selectie overlevenden (replacement)

Voor het selecteren van de individuen die meegenomen worden naar de volgende populatie, kunnen wederom een aantal strategieën vastgesteld worden. Naast de reeds genoemde selectiemethoden, kunnen ook een aantal regels vastgesteld worden als selectiemethode. Ten eerste zouden de oude individuen vervangen kunnen worden door de nieuwe (FIFO: first-in-first-out). Ten tweede zouden de beste individuen kunnen overleven. Als laatste wordt het principe van elitisme genoemd, waarin het beste individu ten aller tijden onthouden wordt (Goldberg (1989)). Het voordeel is dat de beste oplossing alsmaar verbeterd wordt over de verschillende generaties. Het nadeel is echter wel dat de kans op vroegtijdige convergentie van de populatie in een lokaal minimum sterk toeneemt.

5.7 Classificatie evolutionaire algoritmen

Er zijn een viertal evolutionaire algoritmen: Evolutionaire Strategie (ES), Evolutionaire Programming (EP), Genetische Algoritmes (GA) en Genetisch Programmeren (GP). De belangrijkste kenmerken van alle vier zijn hieronder kort samengevat en zullen vervolgens bondig besproken worden.

	ES	EP	GA	GP
Representatie	reële waarden	reële waarden	binaire waarden	boomstructuur
Kwaliteit gemeten aan	waarde doelfunctie	geschaalde waarde doelfunctie	geschaalde waarde doelfunctie	classificatie-nauwkeurigheid
Zelfadaptatie	standaarddeviatie en rotatiehoek	zelden (variantie, correlatie coëfficiënt)	niet	niet
Mutatie	voornamelijk Gaussian	alleen Gaussian	verwisseling, als achtergrond operator	willekeurige veranderingen in boom
Recombinatie	belangrijk voor zelfadaptatie	niet	n-point crossover, uniform crossover	uitwisseling van subbomen
Selectie	deterministisch; uitstervend of gebaseerd op behoud	probabilistisch; uitstervend	probabilistisch; gebaseerd op behoud	probabilistisch; gebaseerd op behoud
Voorwaarden	willekeurige ongelijkheidseisen	niet	simpele grenzen	classificatie schema zoeken
Probleemstelling	continue optimalisatie	optimalisatie	combinatorische optimalisatie	modelleren

Tabel 5.2: Een overzicht van de verschillende evolutionaire aanpakken.

Alle vier delen ze het identieke meta-model, maar iedere aanpak benadrukt andere eigenschappen als belangrijkste voor het succesvol modelleren van het evolutionaire proces. Afgezien van de representatie is het belangrijkste verschil de rol die de genetische operatoren innemen. In EP wordt geen recombinatie gebruikt, terwijl een GA of GP aan de recombinatie operator de belangrijkste rol toekent en de rol van mutaties bijna negeert. ES maakt gebruik van beide operatoren. Verder berusten ES en EP op het principe van zelfadaptatie. EP en GA maken gebruik van probabilistische selectie (alhoewel anders geïmplementeerd), terwijl ES meer gebruik maakt van deterministische selectie. Bij GA krijgt ieder individu een kans groter dan 0 om te overleven (gebaseerd op behoud) terwijl bij EP enkele individuen uitgesloten worden (uitstervend).

Over het algemeen is het belangrijk om te onthouden dat een goede EA de volgende eigenschappen draagt (Ono, Yamamura en Kobayashi 1996):

- *completeness*: iedere oplossing moet een representatie kennen
- *soundness*: iedere representatie moet overeenkomstig zijn met een oplossing
- *non-redundancy*: representaties en oplossingen moeten één-op-één overeenkomstig zijn
- *characteristics-preservingness*: kinderen moeten gebruik maken van de meest bruikbare informatie van de ouders

De eerste drie aandachtspunten hebben betrekking op de representatie, deze zijn voldoende onderzocht. Maar het laatste punt is een eigenschap waar nog weinig rekening mee gehouden wordt.

5.8 Vergelijking andere zoekmethoden

Een evolutionaire methode kan ook gezien worden als een lokale zoekmethode. Evolutionaire methoden onderscheiden zich echter door een gehele populatie van toegelaten oplossingen te manipuleren. In vergelijking met lokale zoekalgoritmes zoals simulated

annealing en tabu search, die gebaseerd zijn op het manipuleren van één toegestane oplossing, kan een evolutionaire methode snel een groot gedeelte van de zoekruimte aftasten. Hierdoor kunnen eigenschappen geïdentificeerd en onderzocht worden die goede oplossingen gemeenschappelijk hebben, waardoor de zoekruimte sterk gereduceerd wordt. Simulated annealing en tabu search kunnen echter wel gezien worden als speciale instanties van evolutionaire algoritmen met een populatie ter grootte van één.
Een ander voordeel van EAs is dat ze gebruik maken van een probabilistische aanpak in plaats van een deterministische. Waardoor de kans dat het globale optimum gevonden wordt een stuk groter is en de methode niet vastloopt in een lokaal optimum.

Nu het principe van een EA bekend is, kan deze toegepast worden op het JSSP in hoofdstuk 6 t/m hoofdstuk 8. Maar eerst zal in de volgende paragraaf ingegaan worden op de beginselen hieraan. Mocht de lezer echter nog meer willen weten over evolutionaire methoden, dan staan in Appendix B en C nog een aantal geavanceerde technieken, waardoor het algoritme betere resultaten kan leveren.

5.9 Toepassing van evolutionaire algoritmes in scheduling

Door het beperkte succes dat de Operations Research bewerkstelligde in het verbeteren van scheduling technieken, ontstond er een potentiaal voor Artificial Intellegence om uitkomst te bieden voor scheduling vraagstukken. Voor het job shop probleem zal voornamelijk over genetische algoritmen gesproken worden, aangezien deze de meeste aandacht hebben gekregen en dus het meeste onderzoek naar verricht is. De term GA komt daarom het meest voor in de literatuur en zal hoofdzakelijk aangehouden worden in de rest van het verslag.

Eén van de eerst gepubliceerde werken over het toepassen van genetische algoritmen voor het scheduling probleem is geschreven door Davis (1985). Alhoewel hij een erg versimpeld probleem aanpakte, heeft hij toch een aantal interessante ontdekkingen gedaan. Zo maakt hij duidelijk dat veel real-life scheduling problemen veelal niet goed gedefinieerde voorwaarden kennen, waardoor het erg gecompliceerd (zoniet onmogelijk) wordt om deze in een formeel framewerk te plaatsen. Het toepassen van een OR techniek lijkt hierdoor zo goed als uitgesloten. Er zijn echter wel een aantal alternatieven aangedragen door anderen (Fox e.a. 1983, Werner, Aydin en Fogarty 2000, Vilela, Brito, Rocha en Neves), maar die leverden vaak suboptimale oplossingen. Davis vermoedde dat een genetisch algoritme echter wel uitkomst zou kunnen bieden om met de vele voorwaarden om te kunnen gaan, zonder vast te lopen in een lokaal minimum door de stochastische aard van evolutionaire algoritmes.
Sinds dit eerste werk over het gebruik van EAs voor scheduling problemen is het gebruik van EAs in velerlei productie scheduling applicaties toegepast. Het meeste van dit werk houdt zich bezig met het bepalen van een optimaal schema in een statische omgeving. In dit verslag zal deze statische omgeving besproken worden en in Appendix D zal de stochastische en dynamische omgeving ook besproken worden.

De grootste moeilijkheid bij het toepassen van evolutionaire algoritmen, is het behoud van toelaatbare oplossingen. Deze moeilijkheid ontstaat door de vele voorwaarden die gesteld worden aan het JSSP. Dit probleem wordt meestal opgelost door de recombinatie en mutatie operatoren probleemspecifiek te definiëren of door de niet-toelaatbare oplossingen een bepaalde penalty te geven in de doelfunctie.

De basis is gelegd voor het daadwerkelijk toepassen van EAs voor job scheduling. De grote vraag is alleen welke representatie en welke genetische operatoren gekozen moeten worden. Dit zijn namelijk de belangrijkste factoren bij toepassingen. Deze zullen in de volgende hoofdstukken uitvoerig besproken worden.

Benchmark

Om te bepalen of een bepaalde representatie of genetische operator het gewenste resultaat oplevert, zijn er een aantal benchmark problemen opgesteld. Dit zijn een aantal standaard

JSSPs die veelal in de literatuur gebruikt worden als vergelijkingsmateriaal en dus aangeven hoe goed een bepaalde EA is. Benchmark problemen zijn echter niet representatief voor werkelijke JSSP uit de praktijk. Om een duidelijk beeld te krijgen hoe de verschillende evolutionaire algoritmes ten opzichte van elkaar presteren staan een aantal uitkomsten in Appendix E.

6. Representatie

De keuze voor de representatie is één van de meest kritieke factoren die bepaalt of het toepassen van een evolutionaire algoritme succesvol is. Hieronder staan een aantal veel toegepaste representaties voor het JSSP, waarbij onderscheid gemaakt wordt tussen een binaire representatie (in paragraaf 7.1) en een (in)directe representatie (in paragraaf 7.2).

6.1 Binaire representatie

Klassieke genetische algoritmes maken gebruik van een binaire representatie voor de mogelijke oplossing om een probleem weer te geven. Het voordeel van deze aanpak is dat de standaard genetische operatoren (zoals k-point crossover en uniform crossover) toegepast kunnen worden zonder enige aanpassingen. Deze vorm van representeren is echter niet logisch voor vraagstukken waarin volgorde een belangrijke rol speelt, omdat er geen directe en efficiente manier is om alle mogelijke oplossingen één-op-één weer te geven in een binaire representatie. Nakano en Yamada maakten echter toch gebruik van deze binaire representatie voor het JSSP. Er moet dan echter wel veel werk worden verricht om de schema's te (de)coderen. Daarnaast moeten er reparatie operatoren gedefinieerd worden om ervoor te zorgen dat de schema's ook daadwerkelijk voldoen aan de voorwaarden die gesteld worden aan een toelaatbare oplossing.
Het genotype bestaat uit een binaire matrix M met op de rijen de paren jobs en op de kolommen de machines, zodat

$$M_{ij} = \begin{cases} 1 & \text{als job 1 eerder dan job 2 (van het paar i) uitgevoerd wordt op machine j} \\ 0 & \text{anders} \end{cases}$$

Het voorbeeld uit Hoofdstuk 3 zal hieronder weergegeven worden als binaire representatie

job	routering		
J1	M1	M2	M3
J2	M1	M3	M2
J3	M2	M1	M3

probleeminstantie

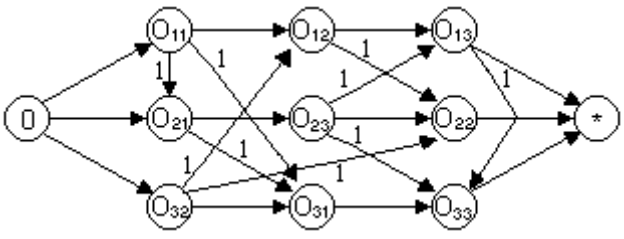
job paar	M_{ij} waarde		
J1 & J2	1	1	0
J1 & J3	1	0	1
J2 & J3	1	0	1

binaire representatie

machine	job volgorde		
M1	J1	J2	J3
M2	J3	J1	J2
M3	J2	J1	J3

schema

Deze representatie wordt ook wel de job-pair relation-based representatie genoemd, in verband met de binaire relatiestructuur tussen de jobs. Een andere benaming is de disjunctive graph-based representatie, omdat deze representatie in principe overeen komt met het vinden van de richting van de disjuncte pijlen in de grafische representatie. In figuur 7.1 is namelijk een representatie gegeven van dezelfde oplossing, zoals deze hierboven beschreven staat.



Figuur 6.1: De binaire representatie in grafische weergave, de pijlen zonder waarde zijn '0' in de binaire representatie.

Deze representatie zorgt echter wel voor redundantie en wellicht ook soundness. Daarnaast is het zo dat niet alle representaties overeenkomen met een toelaatbare oplossing.

6.2 (In)directe representatie

Er zijn echter ook twee andere aanpakken ontwikkeld. De eerste is een indirecte representatie, waarin de instructies voor een *schema bouwer* staan. Zo'n *schema bouwer* is

een speciaal algoritme om een daadwerkelijk schema te construeren. Deze *schema bouwers* zorgen altijd voor toelaatbare oplossingen. Het nadeel van sommige indirecte representaties is dat er sprake kan zijn van redundantie, wat inhoudt dat twee verschillende representaties hetzelfde schema voorstellen. Dit heeft als nadeel dat beide representaties in een populatie kunnen voorkomen, terwijl het maar één unieke oplossing oplevert en dus niets toevoegt aan een populatie. Dit wordt daarom ook wel *false competition* genoemd.
Daarentegen zijn er ook directe representaties, waar het schema direct uit vast te stellen is. De grootste moeilijkheid bij het toepassen van deze representatie, is het behoud van toelaatbare oplossingen. Deze moeilijkheid ontstaat door de vele voorwaarden die gesteld worden aan het JSSP. Dit probleem wordt meestal opgelost door genetische operatoren te definiëren met domein specifieke kennis van het JSSP (waaronder reparatiemethoden) of door de niet-toelaatbare oplossingen een bepaalde penalty te geven in de doelfunctie. Voor job shop scheduling is deze laatste aanpak echter niet efficiënt, aangezien de zoekruimte van de toegelaten oplossingen vele malen kleiner is dan de zoekruimte van alle mogelijke oplossingen. De EA zou dan te veel tijd kwijt zijn aan berekeningen voor niet-toelaatbare oplossingen.

Hieronder volgen een aantal directe en indirecte representaties.

Operation-based representatie

Deze representatie codeert een schema als een opeenvolging van operaties, waarin ieder gen staat voor een operatie. Je zou iedere operatie kunnen laten overeenkomen met een natuurlijk getal, maar omdat de volgorde relaties een belangrijke rol spelen zou de keuze voor genetische operatoren zeer beperkt worden. Het alternatief is om alle operaties van een job overeen te laten komen met hetzelfde symbool en ze vervolgens overeen te laten komen met de volgorde waarin ze afgehandeld moeten worden. Nu komen permutaties overeen met mogelijke oplossingen voor het probleem, hiervoor zijn al een aantal genetische operatoren bekend in de literatuur (zie Hoofdstuk 5). In het voorbeeld van tabel 7.1 staat een operation-based representatie met daaronder het uiteindelijke schema

representatie		1	3	2	1	3	2	2	1	3
komt overeen met schema	M1	J1		J2		J3				
	M2		J3		J1			J2		
	M3						J2		J1	J3

Tabel 6.1: Een operation-based representatie van het 3×3 JSSP van hoofdstuk 3.

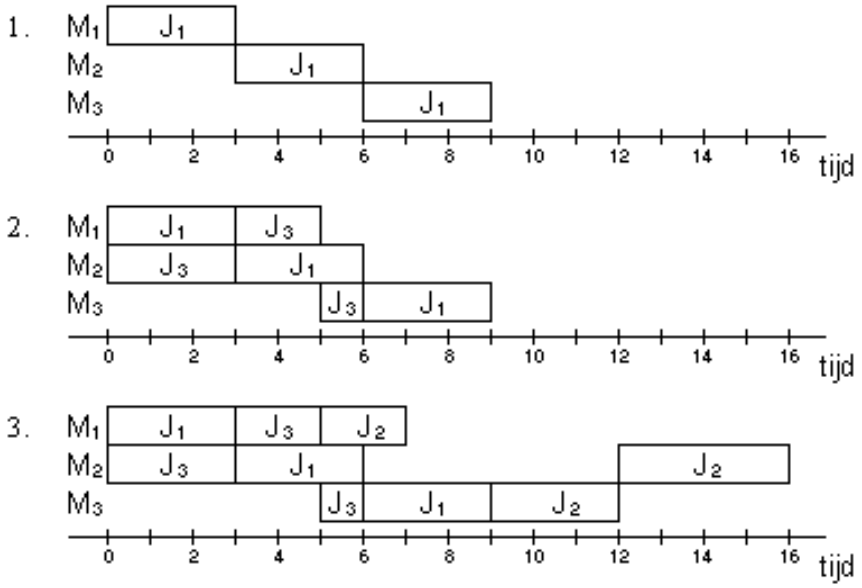
Het nadeel van deze representatie is dat er redundantie kan optreden.

Job-based representatie

Deze representatie bestaat uit een lijst van *n* jobs en een schema wordt gemaakt aan de hand van deze volgorde. Voor een gegeven opeenvolging van jobs, zullen alle operaties van de eerste job als eerst ingepland worden en vervolgens worden de operaties van de tweede job ingepland etc. totdat alle jobs (operaties) ingepland zijn en het schema klaar is. Hier geldt weer dat iedere permutatie van de jobs een toegelaten oplossing levert.
Neem bijvoorbeeld het volgende individu

[1 3 2]

Eerst moeten dan de operaties van job 1 op het vroegste tijdstip ingepland worden, vervolgens worden de operaties van job 3 eraan toegevoegd om uiteindelijk de laatste operaties van job 2 ook in te plannen, zoals weergegeven in figuur 7.2.



Figuur 6.2: De job based representatie van het chromosoom [1 3 2] stelt vast dat eerst alle operaties van job 1 ingescheduled worden, bij de tweede iteratie worden de operaties voor job 3 ingescheduled en de overige operaties in de derde iteratie.

Deze representatie mag dan wel makkelijk in het gebruik zijn en altijd een toelaatbare oplossing leveren. Het grote nadeel van deze aanpak is dat het aantal onderzochte schema's maar beperkt blijft, omdat er niet voldaan wordt aan de completeness eigenschap. Daarnaast is redundantie ook niet uitgesloten.

Preference list-based representatie

Voor het job shop scheduling probleem met *n* jobs en *m* machines, bestaat in deze representatie een chromosoom uit *m* subchromosomen (voor iedere machine één). Iedere subchromosoom is een opeenvolging van *n* symbolen en ieder symbool identificeert een operatie dat uitgevoerd moet worden op de desbetreffende machine. Iedere subchromosoom stelt niet de operatievolgorde voor, maar een preferentielijst. Het schema wordt namelijk bepaald aan de hand van een simulatie, die de toestand van de wachtrij voor de machine analyseert en met de preferentielijst de volgende operatie bepaalt. Aan de hand van een voorbeeld zal het een en ander duidelijk worden. Neem het volgende chromosoom:

2 3 1	1 3 2	2 1 3
M1	M2	M3

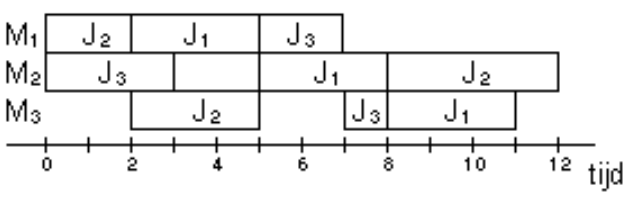
Figuur 6.3: Een preference list-based representatie.

De eerst drie waarden komen overeen met de preferentielijst voor machine 1. Dit individu prefereert de volgende operaties: job 2 op machine 1, job 1 op machine 2 en job 2 op machine 3. Volgens de volgorde relatie kan alleen job 2 op machine 1 ingepland worden.

preferentielijst	2 3 1	1 3 2	2 1 3
mogelijk uit te voeren	O_{11}	O_{21}	O_{32}

Figuur 6.4: Tijdens de eerste iteratie wordt de operatie van job 2 op machine 1 als eerst ingepland.

Job 2 op machine 1 komt dan te vervallen en job 3 op machine 1 komt op de preferentielijst te staan. Vervolgens vinden de overige iteraties plaats, zodat uiteindelijk het schema ontstaat zoals weergegeven in figuur 7.5.



Figuur 6.5: Het schema behorend bij de preference list-based representatie.

In appendix F is een volledige uitleg gegeven hoe dit schema tot stand is gekomen.

Priority rule-based representatie

In deze representatie stelt het chromosoom een opeenvolging van prioriteitregels voor. Prioriteitregels zijn waarschijnlijk een van de meest gebruikte heuristieken voor het oplossen van scheduling problemen, omdat ze eenvoudig te implementeren zijn en geen complexiteit met zich meebrengen. Het algoritme van Giffler en Thompson kan gezien worden als de algemene basis van alle heuristieken, die op prioriteitregels gebaseerd zijn.
Voor het n×m job shop probleem bestaat een chromosoom uit de genen (p_1, p_2, ..., p_{nxm}), waarin gen p_i overeenkomt met één van de prioriteitsregels. Tijdens de i^{de} iteratie van het Giffler en Thompson algoritme wordt dan prioriteitsregel p_i toegepast in stap 4.

Neem nogmaals het voorbeeld uit Hoofdstuk 3 en de prioriteitregels zoals deze in tabel 7.2 staan.

p_i	regel	omschrijving
0	SOT (shortest operation time)	een operatie met de kortste bewerkingstijd op de machine
1	LOT (longest operation time)	een operatie met de langste bewerkingstijd op de machine
2	SPT (shortest processing time)	een job met de kortste totale bewerkingstijd
3	LPT (longest processing time)	een job met de langste totale bewerkingstijd
4	SNRO (smallest no of remaining opns)	een operatie met het kleinste aantal volgende job operaties
5	LNRO (longest no of remaining opns)	een operatie met het grootste aantal volgende job operaties
6	LRPT (longest remaining proc. time)	een operatie met de langste resterende job bewerkingstijd
7	SRPT (shortest remaining proc. time)	een operatie met de kortste resterende job bewerkingstijd

Tabel 6.2: Enkele prioriteitsregels.

Als het chromosoom weergegeven wordt door

0 2 1 4 6 5 7 3 0

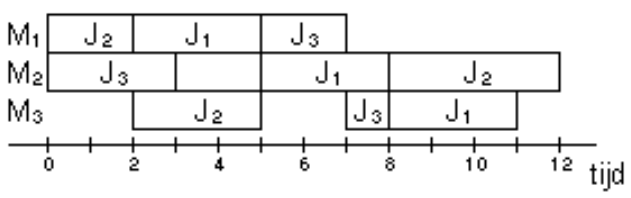
dan geldt in de eerste stap voor het G&T algoritme

$S_1 = \{O_{11}, O_{21}, O_{32}\}$

$\varphi^*_1 = \min \{3, 1, 3\} = 1 \rightarrow k^* = 1$

$C_1 = \{O_{11}, O_{21}\}$

Aangezien de genwaarde op de eerste positie overeenkomt met de SOT prioriteitregel (shortest operation time), wordt job 2 op machine 1 gekozen om als eerste in te plannen operatie. De reeks S_2 komt vervolgens overeen met $\{O_{11}, O_{23}, O_{32}\}$ etc. Stap voor stap wordt het schema opgesteld, zodat uiteindelijk het schema ontstaat zoals deze in figuur 7.6 weergegeven staat.



Figuur 6.6: Het schema behorend bij de priority-based representatie.

Voor een volledige uitwerking wordt verwezen naar Appendix G.

job sequence matrix

Een ander mogelijkheid voor een representatie is om per machine de volgorde van de uit te voeren jobs te bepalen. Een schema wordt dan weergegeven door een reeks van permutaties van de jobs per machine (dit wordt ook wel de machine-based representatie) genoemd.

M1	M2	M3
J1 J3 J2	J3 J2 J1	J3 J1 J2

Figuur 6.7: Een voorbeeld van een job sequence matrix.

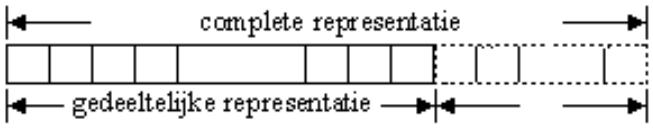
Het nadeel van deze methode is dat de representatie niet altijd een toegelaten oplossing oplevert (soundness wordt geschonden).

completion time-based representatie

Een directie representatie wordt gegeven door het tijdstip waarop een bepaalde operatie uitgevoerd moet worden als genwaarde te nemen.

6.3 Gedeeltelijke representatie

Het blijkt dat het eerste gedeelte van een genotype eerst convergeert en later pas de rest. Dit komt doordat het begin van een genotype preciezer een schema vastlegt en het eind de overige operaties invult. Daarom zijn er representaties die het laatste gedeelte weglaten (Y. Song, J.G. Hughes, N. Azarmi en C. Voudouris), zoals weergeven in figuur 7.9. Dit zorgt voor een snellere uitvoering van de EA.



Figuur 6.8: Een complete representatie kan ingekort worden tot een gedeeltelijke representatie door het laatste gedeelte niet in de representatie op te nemen.

Als de representatie eenmaal vastgelegd is, kan een initiële populatie geconstrueerd worden. Dit wordt in het volgende hoofdstuk besproken.

7. Initiële populatie

De keuze van de eerste generatie speelt vaak een ondergeschikte rol in het zoekproces naar een goede (zoniet optimale) oplossing. Meestal wordt zo'n initiële populatie willekeurig gegenereerd. Het voordeel van een beginpopulatie die random geïnitialiseerd is, is dat er geen voorkeur naar bepaalde oplossingen uitgaat. Maar in de realiteit worden vaak hetzelfde soort problemen opgelost. Vandaar dat het handig kan zijn om reeds opgedane kennis over de zoekruimte te hergebruiken (Louis 2002). Er wordt dan gebruik gemaakt van ervaring uit het verleden om nieuwe problemen te begeleiden naar een kwalitatief goede oplossing. Hierdoor ontstaat echter wel een voorkeur naar de relevante zoekgebieden uit het verleden. Om toch diversiteit te bewaren zal een gedeelte van de startpopulatie wel random gegenereerd moeten worden. Uit experimenteel resultaat blijkt dat deze manier van werken sneller een oplossing oplevert, die kwalitatief beter is. Op het moment dat namelijk de startpopulatie totaal niet geschikt blijkt te zijn voor het nieuwe probleem, dan is dit geen probleem aangezien de ongeschikte individuen (met een hoge doelwaarde) snel verwijderd worden uit de populatie.
Wordt er echter voor gekozen om de initiële populatie willekeurig te bepalen dan kan dit problemen opleveren voor het job shop probleem aangezien de oplossing aan een aantal voorwaarden moet voldoen. Er zou daarom gekozen kunnen worden om met een bepaalde heuristiek, zoals prioriteitregels, een schema samen te stellen. Vervolgens kan het schema veranderd worden met genetische operatoren, zodat er een verzameling oplossingen ontstaan die de initiële populatie vormen. Een andere optie is het gebruik van de G&T methode. Een derde mogelijkheid zal hieronder in meer detail besproken worden.

Eerst worden de operaties willekeurig achter elkaar gezet, waarbij rekening gehouden wordt met de volgorde relaties. Dit wordt weergegeven in de machine volgordematrix S, waarin op rij *i* staat welke machines achtereenvolgens doorlopen moeten worden om job i uit te voeren.

$S_i = [m_{i1} \quad m_{i2} \quad ... \quad m_{iM}]$

Er wordt dan willekeurig een job (rij S_i) gekozen en de eerst volgende operatie wordt uitgekozen om uit te voeren, totdat alle operaties uitgevoerd zijn.

Genereer een toelaatbare oplossing (semi-actief)
Stap 1. selecteer een willekeurige niet lege rij S_i ($i \in N$)
stap 2. gen_j = operatie (i, m_{i1}), $m_{ik} = m_{i(k+1)}$ (k = 1, ..., M-1), $m_{iM} = Ø$, j = j+1
stap 3. herhaal stap 1 en stap 2 totdat $m_{ij} = Ø$, $\forall$ i $\in$ N, j $\in$ M

Dit levert slechts een toegestane oplossing. Om dit om te vormen tot een actief schema, moet eerst de oplossing vertaald worden naar een job sequence matrix. In deze matrix staat op rij *i* beschreven welke jobs achtereenvolgens uitgevoerd moeten worden op machine *i*.

$P_i = [j_{i1} \quad j_{i2} \quad ... \quad j_{iN}]$

Vervolgens wordt een willekeurige machine (rij P_i)gekozen en de eerst volgende job wordt uitgekozen. Op het moment dat de eerst volgende operatie van die job overeenkomt met de geselecteerde machine, dan wordt deze operatie zo snel mogelijk uitgevoerd.

Transformeer naar een actief schema
stap 1. selecteer een willekeurige niet lege rij P_i ($i \in M$), als $m_{j_{i1}1} = i$ zal job j_{i1} als eerst uitgevoerd worden
stap 2. $j_{ik} = j_{i(k+1)}$ (k = 1, ..., N-1), $j_{iN} = Ø$, $m_{j_{i1}k} = m_{j_{i1}(k+1)}$ (k = 1, ..., M-1), $m_{j_{i1}M} = Ø$
stap 3. herhaal stap 1 en stap 2 totdat $j_{lk} = Ø$, $\forall$ l $\in$ M, k $\in$ N, $m_{ij} = Ø$, $\forall$ i $\in$ N, j $\in$ M

De populatie moet vervolgens evolueren volgens genetische operatoren, die in hoofdstuk 8 behandeld zullen worden.

8. Genetische Operatoren

De binaire of indirecte representatie kan gebruik maken van een aantal bestaande genetische operatoren die besproken zijn in Hoofdstuk 5. Naast deze reguliere genetische operatoren, zullen speciale operatoren ontwikkeld moeten worden voor het job shop probleem. Sommige representaties zullen anders zorgen voor niet-toelaatbare oplossingen. Er moeten dus domein specifieke operatoren ontwikkeld worden of er moet een reparatiemechanisme uitgevoerd worden om de oplossingen om te zetten in toegestane oplossingen. Met name bij toepassingen van de directe representatie zullen domeinspecifieke genetische operatoren gebruikt moeten worden. Deze operatoren zijn vaak geïnspireerd door het G&T algoritme.

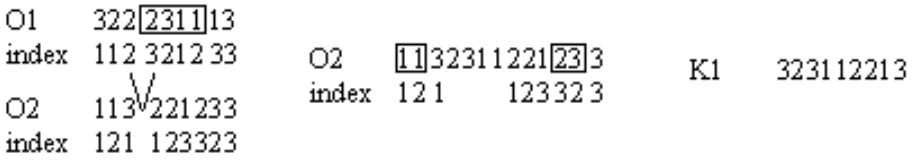
In dit hoofdstuk zullen een aantal recombinatie- en mutatie operatoren besproken worden voor het job shop scheduling probleem in paragraaf 8.1 en paragraaf 8.2 respectievelijk.

8.1 Recombinatie

De bedoeling van een recombinatie operator is dat twee individuen (de ouders O1 en O2) zorgen voor nieuwe individuen (de kinderen). Soms komen er twee kinderen (K1 en K2) voort uit een recombinatie operator. Dit is echter niet altijd het geval. Op het moment dat het resultaat slechts één kind is, dan kan dezelfde operator nogmaals uitgevoerd worden, maar dan met een omgekeerde rol van de ouders (dus O1 wordt O2 en andersom).

Modified Order Crossover (MOC)

In hoofdstuk 5 is uniform order crossover (UOX) besproken. Een mogelijke variant hierop wordt besproken in Choi e.a. 2003. In deze variant wordt een substring willekeurig gekozen uit O1 en in O2 toegevoegd op dezelfde positie waar deze vandaan komt uit O1. Vervolgens worden alle genen verwijderd waarvan de index van de job overeenkomt met die van de substring.



Figuur 8.1: De variant op order crossover gaat eerst twee schema's samenvoegen en vervolgens worden de dubbele waarden verwijderd.

GA/GT recombinatie

Yamada en Nakano maakten gebruik van de completion times van iedere operatie in hun representatie. Zij stelden een GA/GT recombinatie voor, die een toelaatbaar en actief schema verzekert door op ieder beslispunt in het G&T algoritme (stap 4) random een ouder te kiezen in plaats van de operatie die als eerst klaar is. Dit komt overeen met uniforme recombinatie en dan de niet-toelaatbare oplossingen omzetten naar toelaatbaar, met het G&T algoritme. Deze methode kreeg zijn navolging van Dorndorf en Pesch (1993), door niet de completion time te nemen maar de starttijden van iedere operatie in de representatie. Zij pasten echter standaard recombinatie toe. Tabel 8.1 geeft een lijst weer van de G&T-algoritme-gebaseerde GA benaderingen:

referentie	representatie	recombinatie
Yamada en Nakano (1992)	completion time	uniform
Storer e.a. (1992)	perturbed processing time	standaard
Dorndorf en Pesch (1993)	starting time	standaard
Dorndorf en Psche (1995)	priority rule	standaard
Kobayashi e.a. (1995)	job order	subsequence exchange

Tabel 8.1: Een overzicht van een aantal GA benaderingen die gebruik maken van de GT methode.

Time horizon exchange (THX)

In tegenstelling tot veel andere aanpakken, die werkzaam zijn op het chromosoomniveau, werkt de THX crossover op het schemaniveau (Manuel Vázquez en L. Darrell Whitley). Door slechts naar het chromosoomniveau te kijken, wordt er vaak alleen gekeken naar een klein gedeelte van het schema. Dit heeft tot gevolg dat er bijna niet gekeken wordt naar de invloed van de relaties tussen de operaties van het gehele schema.
Bij THX recombinatie wordt er willekeurig een crossover punt gekozen, net zoals bij standaard 1-point recombinatie. Nu wordt dit punt echter niet gebruikt voor de uitwisseling van informatie tussen twee chromosomen, maar als beslissingspunt voor het G&T algoritme om informatie tussen twee schema's om te wisselen.
Dus een gedeelte van het kind krijgt dezelfde relaties als de ene ouder en van de andere ouder erft het kind de overige relaties tussen de operaties, zover dit mogelijk is (zodat de toelaatbaarheid van de oplossing behouden blijft).
Bij deze recombinatie operator wordt vaak voor een directe representatie gekozen, omdat deze genetische operator werkt op het schema niveau. Een aangepast schema moet tenslotte weer terugvertaald worden in een chromosoom. Bij een indirecte representatie is dit erg lastig.

Subsequence Exchange Crossover (SXX)

Deze recombinatietechniek, ontwikkeld door Kobayashi, Ono en Yamamura (1995), is een uitbreiding van de subtour exchange crossover (een recombinatie operator voor een permutatie representatie). Deze recombinatie operator zoekt naar achtereenvolgende jobs bij de ouders, die uitwisselbaar zijn. Ze zijn uitwisselbaar als de jobs van beide segmenten betrekking hebben op dezelfde jobs. Figuur 8.4 laat dit zien aan de hand van een voorbeeld

	M1	M2	M3			M1	M2	M3
O1	1 2 3 6 4 5	3 2 1 5 6 4	2 3 5 6 1 4	⇒	K1	2 1 3 4 6 5	3 2 5 1 6 4	2 6 3 5 1 4
O2	6 2 1 3 4 5	3 2 6 4 5 1	6 3 5 4 2 1		K2	6 1 2 3 4 5	3 2 6 4 1 5	3 5 6 4 2 1

Figuur 8.2: Bij subsequence exchange crossover wordt gekeken naar gelijkwaardige subreeksen van aaneengesloten operaties en die worden uitgewisseld.

Deze methode let niet op volgorde relaties en het zou dus kunnen dat een bepaald schema niet uitvoerbaar is. Dit kan verholpen worden door het G&T algoritme te gebruiken om een actief schema te vormen. Daarnaast garandeert deze methode niet dat er altijd een crossover bestaat. Wel is bewezen dat deze methode beter is dan order crossover (OX).

Job-based Order Crossover (JOX)

De Job-based Order Crossover (Ono, Yamamura en Kobayashi 1996) let op het behoud van de eigenschappen die de ouders bezitten door een gedeelte van de ene ouder te kopieren naar het kind (op dezelfde positie) en de resterende waarden van de andere ouder te halen (in dezelfde volgorde).

stap 1. kies random de jobs waarvan hun positie beschermd wordt
stap 2. kopieer de jobs uit stap 1 van ouder O1 naar kind K1 en van ouder O2 naar kind K2
stap 3. kopieer de overige jobs van ouder O1 naar kind K2 en van ouder O2 naar kind K1, waarbij de jobs op de lege posities worden ingevuld in dezelfde volgorde als dat ze voorkomen bij de ouder

Neem het volgende voorbeeld:

	O1	M1: J1 J2 J3 J4 J5 J6	O2	M1: J3 J4 J2 J5 J6 J1
stap 1		M2: J3 J1 J2 J5 J6 J4		M2: J3 J2 J4 J1 J5 J6
		M3: J2 J3 J1 J4 J6 J5		M3: J6 J1 J2 J5 J4 J3
		⇓		
	K1	M1: J2 J5 J3 J4 J1 J6	K2	M1: J3 J4 J1 J2 J6 J5
stap 2 + stap 3		M2: J3 J2 J1 J5 J6 J4		M2: J3 J1 J4 J2 J5 J6
		M3: J1 J3 J2 J4 J6 J5		M3: J6 J2 J1 J5 J4 J3

Figuur 8.3: Bij job based order crossover (JOX) wordt eerst bepaald welke jobs hun positie behouden en de rust wordt aangevuld met de andere ouder.

Het nadeel is echter wel dat niet alle kinderen toegelaten oplossingen zijn (soundness wordt geschonden). Dit moet echter verholpen worden met behulp van het G&T algoritme (verander stap 4 in het volgende):

Laat J_k* de eerst niet ingeplande job zijn op machine k*. Als deze job voorkomt in de reeks S_t, dan moet random een operatie op die machine k* gekozen worden die geen element is van S_t. Vervolgens kunnen beide operaties omgewisseld worden.

Uit experimenteel resultaat blijkt dat deze operator beter werkt dan SXX. Deze operator zou nog eventueel in een iets andere vorm uitgevoerd kunnen worden door eerst alle jobs evenveel posities naar links of rechts te laten verschuiven alvorens de jobs van O2 ingevuld worden.

Precedence Preservation Crossover (PPX)

Bij deze methode (Yamada en Nakano 1997) wordt de absolute volgorde van de genen van de ouders behouden door per gen te kijken van welke ouder de waarde genomen wordt. Dit wordt gedaan door een willekeurige bitstring *h* ter grootte van n×m te hanteren. Op het moment dat een bepaalde genwaarde van de ene ouder gekozen wordt, dan wordt deze bij de andere ouder ook weggestreept.

Op de eerste positie staat een 0 in bitstring h, dus de eerste waarde van het gen is van ouder O1 afkomstig. Deze waarde wordt bij beide ouders weggestreept.

```
O1  3 2 2 3 1 1 3 1 2
 h  0 1 1 1 0 0 1 0 1
O2  2 3 3 2 1 3 1 1 2
```

De tweede positie bevat de waarde 1, dit komt overeen met de waarde 2 voor het kind en ze worden weggestreept bij beide ouders etc.

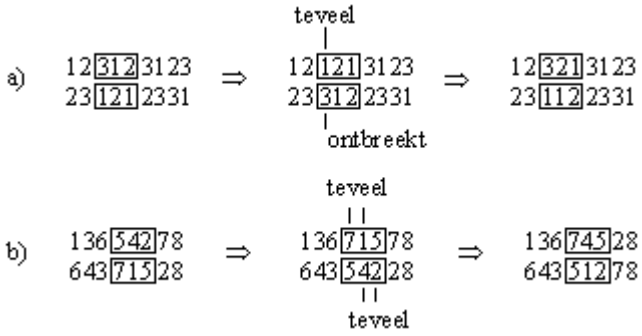
```
O1  ~~2~~ 2 2 3 1 1 3 1 2
 h  0 1 1 1 0 0 1 0 1
O2  2 ~~3~~ 2 1 3 1 1 2
K1  3 2 3 2 1 1 3 1 2
```

Uiteindelijk ontstaat het volgende individu.

Figuur 8.4: Het uiteindelijke individu volgens Precedence Preservation Crossover (PPX) ontstaat door per positie te kijken van welke ouder het kind een waarde krijgt en die waarde wordt bij de andere ouder tegelijkertijd weggestreept.

Partial Schedule Crossover (PSX)

Deze recombinatie operator (Tsujimura, Mafune en Gen 2001) is een aangepast versie van Partial Mapped Crossover (PMX). Hierin vindt in eerste instantie 2-point crossover plaats, waarna een reparatie procedure de dubbele waarden tussen de kinderen onderling verwisseld.



Figuur 8.5: Er zijn verschillende manieren om dubbele waarden bij partial mapped crossover te vervangen door ontbrekende waarden.

Heuristische crossover

Deze recombinatie operator maakt gebruik van het G&T algoritme. Beschouw hierbij een binaire matrix H ter grootte van n × m, waarin

$$H_{ir} = \begin{cases} 0 & \text{de } i^{de} \text{ operatie van machine r moet bepaald worden op basis van O1} \\ 1 & \text{de } i^{de} \text{ operatie van machine r moet bepaald worden op basis van O2} \end{cases}$$

De vierde stap in G&T algoritme moet vervolgens vervangen worden door:

kies een van de twee ouders {O1, O2} volgens H_{ir}. Selecteer de operatie uit de conflicterende reeks C met de vroegst ingeschedulde operatie uit die reeks in de gekozen ouder

8.2 Mutatie

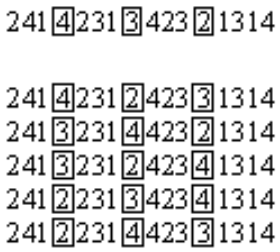
Een andere belangrijke operatie in een EA is de mutatie van individuen. Over het algemeen kunnen twee type mutatie operatoren onderscheiden worden: de swap mutatie en de shift mutatie. Bij de eerste mutatie operator worden genen omgewisseld, terwijl bij de andere mutatie operator een gen verplaatst wordt en de rest plaats maakt.

THX mutatie

De THX mutatie operator is gebaseerd op de disjuncte graaf van het schema (zie Hoofdstuk 3). Het is eenvoudig in te zien dat als er een verbetering mogelijk is, dit alleen kan plaatsvinden door het kritieke pad te veranderen. In de aanpak van Nowicki en Smutnicki (1996) wordt gekeken naar de volgorde van de blokken op dit kritieke pad. Alhoewel het verwisselen van opeenvolgende taken op dezelfde machine van een kritiek pad de acyclische eigenschap van de gerichte graaf behoudt, zal het aantal kinderen dat beter is dan de ouders echter zeer beperkt zijn (Grabowski e.a. (1986)). Het omdraaien van een kritieke pijl kan alleen tot een verbetering leiden als één van de operaties betrekking heeft op de eerste of laatste operatie van het blok. Tenzij de verwisseling betrekking heeft op het eerste (of laatste) blok, dan kunnen alleen de laatste (of eerste) twee operaties verwisseld worden. Nadat een kind gegenereerd is door twee operaties in een blok om te draaien, wordt deze met behulp van het G&T algoritme omgezet in een actief schema.

Neighbor Search Mutation (NSM)

Bij deze mutatie operator (Tsujimura, Mafune en Gen 2001) wordt eerst een bepaald aantal genen geselecteerd en worden alle mogelijke combinaties onderzocht door deze genen met elkaar om te wisselen.



Figuur 8.6: Bij neighbor search mutation worden alle volgordes van de geselecteerde waarden doorlopen.

Job-based Mutation

In deze operator worden alle operaties van een bepaalde job een willekeurig aantal posities naar voren of naar achteren verschoven. Aan de hand van een voorbeeld is dit duidelijk:

de te muteren chromosoom: 2 2 3 4 3 1 4 3 1 1 2 2 1 4 3 4

Selecteer alle operaties behorend bij job 1

tussenresultaat: - - - - - 1 - - 1 1 - - 1 - - -

Vervolgens wordt besloten dat alle operaties van job 1 drie posities naar voren verschuiven

tussenresultaat: - - 1 - - 1 1 - - 1 - - - - - -

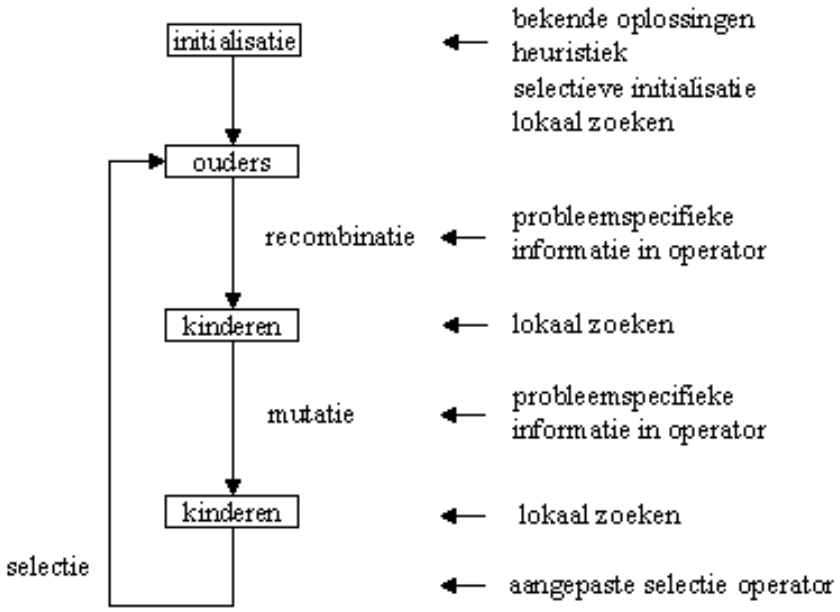
en de overige jobs worden er weer aan toegevoegd:

resultaat: 2 2 1 3 4 1 1 3 4 1 3 2 2 4 3 4

Deze operatoren vormen gezamenlijk het standaardmechanisme van een EA. Deze kan uitgebreid worden met andere lokale zoekmethoden. Dit wordt hybridisatie genoemd en zal in het volgende hoofdstuk behandeld worden.

9. Hybridisatie

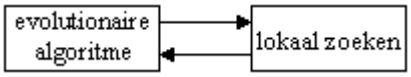
Naast het gebruik van een enkelvoudige evolutionaire algoritme zou deze ook gecombineerd kunnen worden met andere methoden of zelfs opgenomen worden in een groter geheel. Hierdoor is het mogelijk om probleemspecifieke heuristieken te combineren met een EA in een hybride algoritme. Dit soort algoritmen worden ook wel memetic algoritmes genoemd (Ombuki en Ventresca 2002, Goncalves, Mendex en Resende 2002).
Er zijn echter veel mogelijkheden om kennis en/of andere methoden toe te voegen aan een EA. Zoals figuur 9.1 weergeeft is het zelfs toe te passen in bijna alle fasen van een EA.



Figuur 9.1: De plaatsen waar hybridisatie op kan treden in een evolutionaire methode is zeer divers en kan van verschillende aard zijn.

In de vorige hoofdstukken zijn al een aantal van deze hybridisatieprincipes (impliciet) aangehaald. Zo is in hoofdstuk 7 beschreven hoe probleemspecifieke informatie gebruikt kan worden voor de initialisatie van de populatie. Daarnaast is continu het G&T algoritme gebruikt, dit is uiteraard ook een vorm van hybridisatie aangezien hierin specifieke kennis over de zoekruimte zit verwerkt.

De meest gebruikte vorm van hybridisatie bij een EA is het verbeteren van individuen uit de populatie door een heuristische zoekmethode of een heuristische rule-based methode toe te passen. Hierin wordt de EA gebruikt als zoektechniek in een ruimte van afgeleide oplossingen en dat dan een heuristiek of andere methode deze punten omzet in betere kandidaat-oplossingen die vervolgens gebruikt kunnen worden in de volgende generatie voor een EA.



Figuur 9.2: De interactie tussen een evolutionaire algoritme en lokaal zoeken bij een local search GA.

Dit principe wordt ook wel local search GA (LSGA) genoemd. Het idee hierachter is dat het nageslacht gebruikt wordt als beginpunt voor een lokaal zoekprincipe, waardoor het kind van dit beginpunt naar het dichtstbijzijnde lokale optimum gaat. Dit optimale punt wordt dan gebruikt voor de volgende generatie. Een lokale zoekstrategie wordt probabilistisch toegepast door een verbeterde mutatie operator aan de EA toe te voegen, die lokaal zoeken als uitgangspositie neemt. Hieronder staat een uitgebreid voorbeeld van dit principe. Hierin zal tweemaal gezocht worden naar een verbetering door opeenvolgende operaties om te

wisselen. Mocht er geen enkele verwisseling gevonden worden die een verbetering oplevert, dan verandert er niets.

```
Kies willekeurig een machine k ∈ {1, 2, ..., m} en kies willekeurig een operatie (j,k) met j ∈
{1,...,n-1} en verwissel operatie j met de daaropvolgende operatie op dezelfde machine, als
dit toegestaan is met de volgorde relaties.
herhaal voor machine k = 0 tot m
        operatie j = 0
        besteOmwisseling = j
        voordeel = voordeel bij omwisseling operatie 0 en 1 op machine k
        meesteVoordeel = voordeel
        herhaal voor operatie j = 1 tot n - 1
                voordeel = voordeel bij omwisseling operatie j en j+1 op machine k
                als voordeel > meesteVoordeel dan besteOmwisseling = j, meesteVoordeel =
                voordeel
        eind
        als meesteVoordeel > 0 dan
                voer de omwisseling uit
                als machine k voor eerst verbeterd
                        doorloop deze stappen voor dezelfde machine nog één maal
                eind
        eind
eind
```

Algoritme 9.1: Een voorbeeld van een lokale zoekprocedure.

Het nadeel van een local search GA is dat de rekentijd een aantal maal zo lang wordt ten opzicht van een enkelvoudige GA.

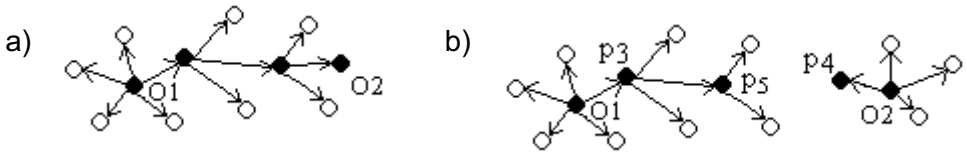
Hieronder zullen een aantal genetische operatoren besproken worden, die gebruik maken van het lokaal zoekprincipe.

9.1 Multi-Step Crossover (MSX) en Mutatie (MSM)

Eerst worden de buren van de eerste ouder gedefinieerd als N(O1). Het idee achter MSX (Yamada en Nakano 1995) is om een punt $x \in$ N(O1) te evalueren op basis van de afstand d(x,O2) tussen het punt *x* en de tweede ouder O2 in plaats van de doelfunctie *f(x)*. Wanneer er begonnen wordt in O1, wordt *x* stap voor stap aangepast via een uni-directionally manier naar O2:

uni-directional MSX
stap 1. x = O1, initialiseer de lijst met kinderen K = $\varnothing$
stap 2. selecteer een punt y $\in$ N(x) zodat y= $\arg\min_{z \in N(x)} \{d(z,O2)\}$ en voeg y toe aan lijst K, x=y
stap 3. herhaal stap 2 tot er voldaan wordt aan een stopcriterium

Er is voor gekozen om de buur van *x* te selecteren, die het dichtst bij O2 ligt. Maar dit proces zou net zo goed kunnen plaatsvinden volgens een ander criterium, zolang deze maar gebaseerd is op de afstand met ouder O2. Vervolgens moet een kind uit de lijst met kinderen (K) gekozen worden. Hierbij wordt allereerst gekeken naar het beste kind. Als deze niet beter is dan één van de ouders, dan wordt het kind gekozen dat ongeveer de gelijke afstand heeft naar beide ouders.



Figuur 9.3: Bij de uni-directional MSX (a) wordt naar die buur gezocht die het dichts bij O2 ligt, terwijl bij bi-directional MSX (b) naar die buren gezocht wordt zodat de ouders naar elkaar toe bewegen.

Het punt in het uni-directional model beweegt zich dus voort van de ene ouder naar de andere. Wat echter ook mogelijk is, is dat de ouders naar elkaar toe bewegen. Dit wordt bi-directional MSX genoemd.

bi-directional MSX

stap 1. i = 1, initialiseer de lijst met kinderen K = $\varnothing$, p_1 = O1 en p_2 = O2

stap 2. selecteer een punt $p_{i+2} \in N(p_i)$ zodat $p_{i+2} = \arg\min_{z \in N(p_i)} \{d(z,p_{i+1})\}$.

als $d(p_{i+2},p_{i+1}) > d(p_i,p_{i+1})$ dan stoppen anders voeg p_{i+2} toe aan K

stap 3. i = i + 1 en ga naar stap 2

Dit principe kan echter ook gebruikt worden voor een mutatie operator. In deze operator wordt steeds de verste buur van een punt gekozen als mutant. Na dit een aantal maal vastgesteld te hebben wordt die mutant gekozen die het meest superieur is. Als dit er meerdere zijn dan wordt ook gekeken naar de afstand en degene die het verst weg gelegen is van het origineel wordt geselecteerd.

Multi-step mutation (MSM)

stap 1. x = p, initialiseer de mutatie lijst M = $\varnothing$

stap 2. selecteer een punt $y \in N(x)$ zodat $y = \arg\max_{z \in N(x)} \{d(z,p)\}$. Voeg y toe aan lijst M, x = y

stap 3. herhaal stap 2 voor een vast aantal iteraties

Er wordt dus zo hard mogelijk weggegaan uit de te muteren oplossing p. De bedoeling is echter dat het JSSP opgelost wordt. Dus deze operatoren moeten uiteindelijk gecombineerd worden in een evolutionaire algoritme. Hieronder zal een voorbeeld gegeven worden.

GA met MSX en MSM voor JSSP

stap 1. initialiseer de populatie door random een reeks schema's te genereren en pas vervolgens lokaal zoeken toe

stap 2. selecteer twee schema's O1 en O2 met voorkeur voor individuen met een lagere makespan

stap 3. als de DG afstand < x, pas MSM toe op O1: dit levert K1
anders pas MSX toe op O1: dit levert K1

stap 4. pas lokaal zoeken toe op het kind K1: dit levert K1'

stap 5. als de makespan van het kind K1' kleiner is dan het slechtste individu van de generatie, dan wordt deze vervangen

stap 6. herhaal stap 2 t/m stap 5 totdat er geen verbeteringen meer kunnen plaatsvinden na een bepaald aantal evaluaties

9.2 Neighborhood Search Crossover (NSX)

Reeves (1994) heeft een neighborhood search crossover ontwikkeld. Neem hiervoor twee individuen *x* en *z*. Dan wordt een individu *y* tussen *x* en *z intermediate* genoemd als d(x,z) = d(x,y) + d(y,z) geldt, waarin d(x,y) overeenkomt met de afstand tussen x en y. Dit wordt ook

wel aangegeven met x ◊ y ◊ z. Voor het JSSP wordt als afstandsmaat d(x, y) vaak de DG afstand tussen de twee schema's genomen.
De k^{de} orde 2-neighborhood van x en z wordt gedefinieerd als de reeks van alle *intermediate* individuen met een afstand ter grootte k van x of z.

$N_k(x, z) = \{y \mid x \lozenge y \lozenge z$ en $(d(x, y) = k$ of $d(y, z) = k)\}$

Bij neighborhood search crossover van order k (NSX_k) worden alle individuen uit $N_k(O1, O2)$ onderzocht en zullen de beste gekozen worden voor het nageslacht.

9.3 Multi-Step Crossover Fusion (MSXF)

Yamada en Nakano (1997) breidden het idee van NSX uit.

```
selecteer de ouders O1 en O2
x = O1 = q
herhaal zolang het stopcriterium niet voldaan is
        voor ieder individu y_i ∈ N(x) bereken d(y_i, O2)
        sorteer y_i ∈ N(x) in oplopende volgorde van d(y_i, O2)
        herhaal totdat y_i geaccepteerd wordt
                selecteer y_i willekeurig uit N(x), met de voorkeur naar y_i's met een kleine index
                bereken f(y_i) als y_i nog niet bezocht is
                accepteer y_i met kans 1 als f(y_i) ≤ f(x) en accepteer anders met kans P_c(y_i)
                verander de index van y_i van i naar n
                schuif de overige y_k's met k > i een positie omhoog
        eind
        x = y_i
        als f(x) < f(y) dan q = x
eind
q wordt gebruikt voor de volgende generatie
```

Algoritme 9.2: Een voorbeeld van multi-step crossover fusion (MSXF).

Het grootte voordeel van deze methode ten opzichte van NSX is dat het gebruikt maakt van een stochastisch lokaal zoekalgoritme in plaats van een deterministisch zoekalgoritme.

9.4 Multi-Step Mutation Fusion (MSMF)

Deze mutatie operator kan op dezelfde manier gedefinieerd worden als MSXF, maar dan voor één punt. Het grote verschil is echter wel dat de voorkeur naar bepaalde punten precies omgekeerd wordt (sorteer de buren in aflopende volgorde van $d(y_i,O2)$.

De belangrijkste theorie voor het statische JSSP is behandeld in afgelopen hoofdstukken. In het volgende hoofdstuk zal daarom meer de nadruk gelegd worden op praktische toepassingen van deze theorie.

10. Praktijk

Scheduling problemen worden in de praktijk nog al te vaak met de hand opgelost. Dit heeft enigszins voordelen, zoals het direct kunnen toepassen van de ervaring en er is een grote mate van begrip. Dit wordt echter te ingewikkeld op het moment dat er uitbreidingen optreden en modernisatie gewenst wordt. Deze handmatige aanpak zal zelden tot nooit tot een optimale oplossing leiden, aangezien er erg veel keuzemogelijkheden zijn. Vandaar dat er tegenwoordig naar een systeem gezocht wordt, dat de gebruiker/planner kan ondersteunen in dit beslissingsproces.

Voordelen handmatig schedulen	Nadelen handmatig schedulen
- snel en nauwkeurig menselijke kennis en formele prioriteiten combineren - brede algemene kennis om adequaat in te grijpen ten tijde van een crisissituatie	- het is niet mogelijk om een groot aantal verschillende formele prioriteiten te testen - het is niet mogelijk om de gevolgen van bepaalde handelingen nauwkeurig in te schatten - moeilijkheden om expertise te combineren - moeilijkheden om snel op veranderingen te reageren - de systemen worden al snel complex

Tabel 10.1: Een overzicht van de voor- en nadelen van handmatig plannen.

Aangezien de mens een belangrijke rol blijft spelen bij scheduling, zal een computersysteem een goede interactie met de gebruiker moeten ondersteunen in de vorm van een goede user interface (Hart en Ross). De handmatige functies en de grafische representatie moeten de gebruiker het gevoel geven dat ze invloed kunnen uitoefenen in het scheduling proces. Daarnaast spelen ook zaken als prestatiefeedback en transparantie een essentiële rol. Er bestaan echter al wel een aantal scheduling systemen (zie tabel 10.2), maar die maken geen gebruik van evolutionaire methoden.

systeem	**grondlegger**	**Opmerkingen**
ISIS	Fox e.a., 1983	Één van de eerst gedocumenteerde scheduling system, OR heuristieken
Knowledgecragt		Opvolger ISIS
CALLISTO	Sathi e.a., 1985	Project scheduling, OR heuristieken, Opvolger ISIS
PATRIARCH	Morton e.a., 1988	Two-level planning and scheduling system: bottleneck dynamics en OPIS
OPIS	Ow and Smith, 1987, 1990	Expertise moet samenwerken
MERLE	Prietula and Thompson	Multi-expertise scheduling system
MARS	Marsh, 1985	Schedule resources for the space transportation system
OPAL	Bensana e.a., 1988	
XSITE		Expert site planner’s assistant
IMACS		Aid manufacturing
ISA		Aid scheduling
IPMA		Project management
ILRPS		Lange termijn scheduling

Tabel 10.2: Een aantal schedulingssystemen (Bongaerts 1998).

De toepasbaarheid van zo'n systeem in de praktijk lijkt veel belovend, maar er is helaas niets te vinden in de literatuur over daadwerkelijke implementaties. Vandaar dat een drietal bedrijven benaderd zijn om te vragen in hoeverre zij gebruik maken van scheduling systemen en dan met name door gebruik te maken van evolutionaire methoden. Hierbij zal

ook van de gelegenheid gebruik gemaakt worden om de theorie aan de praktijk te toetsen. De drie bedrijven zijn:

- Perot Systems,
- Quintiq en
- ORTEC bv.

Er is bewust voor gekozen om het toetsingsgebied iets te verruimen naar scheduling in het algemeen in plaats van job shop scheduling, aangezien het anders te specifiek zal worden.

10.1 Perot Systems

Perot Systems is een internationaal bedrijf dat wereldwijd technologisch gebaseerde bedrijfsoplossingen aanbiedt. Na telefonisch contact met Marc-Paul van der Hulst bleek echter dat ze niet mee wouden werken. Het enige wat ze zeiden, was dat ze actief zijn op een paar gebieden die hiermee te maken hebben. Verder wouden ze geen nadere toelichting geven, aangezien *"het zich niet leent voor dit verslag"* volgens hen.

10.2 Quintiq

De softwareoplossingen van Quintiq stellen organisaties in staat om bedrijfsprocessen te optimaliseren en om samen te werken met hun klanten. Quintiq kent ongeveer 60 medewerkers en is gevestigd in ’s Hertogenbosch.
Allereerst is gesproken met Mark Willems (Vice President Quality & Products). Hij kon echter alleen aangeven dat binnen Quintiq een algoritme ontwikkeld is dat echter meer weg heeft van simulated annealing dan van een evolutionaire methode, aangezien één oplossing iteratief deels afgebroken en weer opgebouwd wordt in plaats van een verzameling oplossingen. Coen Verberg heeft echter het algoritme ontwikkeld en kon meer vertellen over de achtergrond van het algoritme.

Voor scheduling en transport problemen maakt Quintiq onder andere gebruik van een combinatie van evolutionaire methoden en lokale zoekprincipes (zoals simulated annealing en tabu search). Verder kunnen de meer conventionele methoden ook gecombineerd worden met eenvoudige tot ingewikkelde zoekstrategieën. Er is voor deze aanpak gekozen, aangezien de standaard conventionele methoden niet voldoende kracht bleken te zijn om de problemen bevredigend op te lossen.
Deze methoden zijn zelf door Quintiq bedacht en onderzocht, aangezien er een aantal medewerkers zijn die redelijk veel ervaring hebben met evolutionaire algoritmes. Vandaar dat zij eigen software ontwikkeld hebben. Hier zijn ze sinds een paar jaar heel direct mee bezig en op dit moment worden ook al applicaties in de praktijk gebruikt.

De evolutionaire methoden maken gebruik van een indirecte representatie. Verder valt er niet zo heel erg veel te zeggen over de specifieke elementen van een evolutionaire algoritme, aangezien deze nogal verschillen afhankelijk van het probleem. De gemiddelde omvang van de populatie kan bijvoorbeeld variëren van heel klein tot relatief groot. Hetzelfde geldt voor het aantal generaties dat gemiddeld doorlopen wordt; Bij een simpel probleem (en kleine instanties) zullen echter veel minder generaties doorlopen hoeven worden ten opzicht van een moeilijk probleem (en grote instanties). Maar over het algemeen worden er relatief weinig generaties gecreëerd. Verder past de evolutionaire methode vaak een mutatie operator toe, waarbij voornamelijk de meeste elementaire en simpele mutatie operatoren uitgevoerd worden. Recombinatie vindt daarentegen niet plaats. Voor de selectieprocedures wordt gebruik gemaakt van hill climbing.
Tijdens het uitvoeren van het evolutionaire algoritme vindt er indien nodig een verbeterstap plaats. De slimme combinatie tussen verbeterstappen en mutaties levert extreem goede resultaten op. Dit is zelfs de kracht van het gehele algoritme. Verder kunnen deze verbeterstappen ook achteraf plaatsvinden.

Er wordt echter niet altijd voor een evolutionaire aanpak gekozen, aangezien een deel van de problemen beter opgelost worden met traditionele methoden. Op het moment dat deze

slecht presteren, zal overgegaan worden op een evolutionaire methode. Dit is met name het geval bij logistieke problemen.
Beide aanpakken zijn volledig door de (eind)gebruiker geaccepteerd, aangezien zij alleen geïnteresseerd zijn in een goede en praktische uitkomsten. De gebruikers die ermee moeten werken ervaren het modelleren en implementeren als gemakkelijk.

In de toekomst zal Quintiq het gebruik van evolutionaire methoden verder ontwikkelen, omdat ze goede ervaringen met deze aanpak hebben en de praktische toepassingen enorm groot zijn.

10.3 ORTEC bv

ORTEC is toonaangevend op het gebied van geavanceerde planningsvraagstukken. Sinds april 1981 zijn ze al actief op het gebied van de Operations Research. Inmiddels zijn er 350 medewerkers werkzaam op 12 locaties in 4 continenten. Het hoofdgebouw bevindt zich in Gouda.
Er zijn twee producten binnen ORTEC die gebruik maken van evolutionaire methoden en/of lokale zoekprincipes. Eerst zal ingegaan worden op HARMONY, dit is een systeem dat medewerkers inroostert. Hiervoor is gesproken met Gerhard Post. Het tweede systeem betreft SHORTREC, dit is een routeringssysteem dat routes voor voertuigen bepaalt.

HARMONY

HARMONY is opgezet om ziekenhuispersoneel in te roosteren, waarbij gelet moet worden op allerlei persoonlijke wensen en voorkeuren, maar ook op de wetgeving. Het programma bevat een algoritme dat uit twee fasen is opgebouwd:

fase 1: construeer een aantal schema's door recombinatie en mutatie toe te passen, totdat er geen verbeteringen meer mogelijk zijn
fase 2: ga lokaal optimaliseren totdat er geen verbeteringen meer mogelijk zijn

Dit zijn, in tegenstelling tot local search genetische algoritmes, twee gescheiden fasen die echt na elkaar plaatsvinden. De beide fasen zullen apart behandeld worden

Fase 1: evolutionaire methode

Erg opvallend aan deze fase is dat de evolutionaire methode standaard een populatie bevat van slechts twee individuen. Dit komt voornamelijk doordat het combineren van meerdere roosters niet echt mogelijk is. Vandaar dat een rooster voornamelijk van zichzelf moet leren, aangezien het uitwisselen van informatie de consequentie heeft dat het gehele rooster opnieuw geconstrueerd moet worden. Het is echter wel zo dat dit aantal van twee door de gebruiker aangepast kan worden. Dit zal leiden tot betere resultaten. Daarentegen zullen meer generaties berekend moeten worden om tot convergentie te komen. Beide individuen zijn directe representaties, zodat direct op het schema gewerkt wordt.

Bij iedere populatie wordt via tournament selectie bepaald welke individuen gekozen worden om zich voort te planten. Vervolgens vindt recombinatie en mutatie plaats en daarna wordt gekeken of er enige verbeteringen optreden. Als dit het geval is, dan wordt de '*slechtste*' ouder (met de hoogste doelwaarde) vervangen door het kind.

Er wordt gebruik gemaakt van twee type recombinatie operatoren:

- CrossOverOnePoint: het chromosoom wordt opgedeeld in drie opeenvolgende intervallen, vervolgens wordt het middelste interval van het rooster leeggemaakt en worden de niet toegekende taken opnieuw ingeroosterd
- CrossOverTwoPoint: het chromosoom wordt opgedeeld in drie opeenvolgende intervallen, vervolgens worden het eerste en laatste interval van het rooster leeggemaakt en worden de niet toegekende taken opnieuw ingeroosterd

Wat direct opvalt is dat de definitie van een recombinatie operator niet overeenkomt met hetgeen wat in de literatuur bekend staat als een recombinatie operator. Het grote verschil is

dat bij deze crossover operatoren maar één individu betrokken is. Dit heeft ermee te maken dat het combineren van meerdere oplossingen uiteindelijk resulteert in het geheel opnieuw construeren van een rooster.

Verder zijn er drie mutatie operatoren, waarbij eerst bepaalde diensten verwijderd worden om ze vervolgens weer in te roosten:

- MutateGrossWorkingTime: verwijder alle diensten van de twee middelen met de hoogste en laagste "rest-beschikbaarheid"
- MutateRandomBlock: verwijder alle diensten op een willekeurig geselecteerd interval
- MutateShortSeries: verwijder alle diensten die op zichzelf staan

Andere genetische operatoren zijn wel geprobeerd, maar leverden geen betere resultaten waardoor ze niet zijn opgenomen in het uiteindelijke product. Na ongeveer 40 tot 150 generaties convergeert de bevolking. Zoals eerder al aangegeven is, worden de resultaten verbeterd als de bevolking uit meerdere individuen zal bestaan. Het is daarom ook raadselachtig waarom er standaard twee individuen gekozen worden in een bevolking. Een echte verklaring is hier ook niet voor, behalve dan dat de rekentijd gereduceerd wordt door slechts twee individuen te handhaven.

Fase 2: Lokaal verbeteren

Na de eerst fase waren er met het blote oog echter nog verbeteringen zichtbaar, waardoor er na de evolutionaire methode nog een tweede fase uitgevoerd wordt. Via 1-OPT en 2-OPT wordt de uiteindelijke oplossing vele malen beter.

Na deze tweede fase zal een redelijk goed schema geconstrueerd zijn. Het schema zou nog eventueel met de hand aangepast kunnen worden, zodat de gebruiker vertrouwen krijgt in het geproduceerde schema.

SHORTREC

Het tweede computer softwaresysteem dat gebruik maakt van een lokale zoekmethode is SHORTREC. Dit systeem wordt gebruikt voor het inplannen van ritten voor onder andere vrachtwagens. Dit systeem kan onder meer gebruik maken van tabu search, maar niet van een evolutionaire algoritme zoals deze beschreven staat in hoofdstuk 5. Vandaar dat het product alleen genoemd wordt, maar er zal niet verder op ingegaan worden.

11. Conclusie

Evolutionaire algoritmes werden vaak genegeerd door Operations Research. Toch is afgelopen jaren bewezen dat evolutionaire methoden goede resultaten kunnen genereren.Vandaar dat het gebruik van EAs in veel scheduling problemen steeds meer groeit. De afgelopen jaren is er veel werk verricht in het verkleinen van het gat in de resultaten tussen de beste oplossingen van de branch-and-bound methode en de oplossingen volgens de EA benadering. Inmiddels is het punt bereikt dat deze oplossingen van equivalente kwaliteit zijn en het gebruik van een evolutionaire methode de overige methoden zelfs kan overtreffen.
Deze resultaten zijn slechts aangetoond met behulp van een klein aantal (eenvoudige) benchmark problemen. Meer complexere problemen geven echter een realistischer beeld. Deze zijn niet eenvoudig op te lossen met een exacte methodiek. Het gebruik van een evolutionaire methode is daarentegen wel mogelijk. Er is echter nog zeer weinig bekend hoe goed (of slecht) de aanpak werkt voor zulke probleeminstanties. Vandaar dat Emma Hart en Peter Ross een probleemgenerator ontwikkeld hebben, die problemen kan genereren op verschillende moeilijkheidsniveaus. Aan de hand hiervan hebben zij een aantal conclusies getrokken over het extrapoleren van de resultaten die verkregen zijn van de benchmark problemen. Zo concludeerden zij dat EAs relatief robuust zijn voor een groot aantal problemen en dat de EAs veelal redelijk goede oplossingen geven. Dit is veelbelovend voor veel scheduling applicaties in de praktijk. Met name omdat slechts een goede oplossing geëist wordt, die in relatief korte tijd verkregen moet kunnen worden. Een optimale oplossing is vaak niet het belangrijkste.

Wanneer gekeken wordt naar het verschil in de aanpak tussen de eerste resultaten van een EA en de verbeterde resultaten van tegenwoordig, dan is waarneembaar dat de nieuwere aanpak veelal gebruik maakt van een bepaalde vorm van lokaal zoeken. Of dit nou gebruikt wordt tijdens het uitvoeren van de EA of na de EA maakt niet uit, ze zullen een duidelijk verbetering opleveren. Dus in de toekomst zal veelvuldig gebruik gemaakt gaan worden van hybride EAs, aangezien een hybride methode superieur is over de pure EA. Er moet alleen wel altijd gekeken worden naar de afweging tijd versus kwaliteit. Voor een hogere kwaliteit is het namelijk een vereiste om meer tijd in de uitvoering van het algoritme te steken. Het is dan maar altijd de vraag in hoeverre de extra berekeningen gerechtvaardigd worden door de verbeterde kwaliteit. Daarnaast moet de generaliteit van een EA onderzocht worden. Door extra hybridisatie in een EA op te nemen, wordt er soms met te veel probleemspecifieke informatie omgegaan en verliest de EA zijn algemene toepasbaarheid (minder flexibel).

Daarnaast is de rekentijden van een EA vaak een struikelblok. Dit hoeft echter niet het geval te zijn. Ten eerste omdat er onderscheid gemaakt moet worden tussen off-line en on-line systemen. Op het moment dat het systeem on-line werkt, dan moet het snel het gewenste resultaat kunnen opleveren. Voor off-line systemen is dit niet noodzakelijk. Ten tweede worden computers steeds sneller, waardoor de rekentijden in de (nabije) toekomst acceptabel zijn. En ten derde kan er afgevraagd worden of het vinden van een optimale oplossing wel zo belangrijk is, toegelate oplossingen zijn immers ook uitvoerbaar.

		Model	
		Exact	Benadering
Optimale Oplossing	Exact	?	Operations Research
	Benadering	Evolutionaire Methoden	X

Tabel 11.1: Operations Research versus Evolutionaire Methoden.

Bovenstaande tabel laat hierin heel duidelijk zien dat een OR techniek het gemodelleerde probleem exact oplost, maar de vraag is in hoeverre het model in overeenstemming is met

de werkelijkheid. Vandaar dat EAs het probleem exact omschrijven, waarna een benaderingsoplossing gevonden wordt.

Concluderend kan gezegd worden dat als er een speciale deterministische (OR) techniek bestaat voor het probleem dat opgelost moet worden, dan moet deze gebruikt worden aangezien ze een betere oplossing genereren in relatief korte rekentijd. Maar als er ruis in de gegevens zit en bepaalde aspecten van het probleem moeilijk te vertalen zijn in een analytisch model, zal een traditionele methode niet toerijkend genoeg zijn. Vanwege de grote mate van flexibiliteit is het gebruik van een evolutionaire methode daarentegen zeer geschikt. Hierdoor is een evolutionaire methode namelijk eenvoudig toe (en aan) te passen. Daarnaast bestaan er geen technieken voor een meer ingewikkelde productieomgeving. Wanneer het JSSP bijvoorbeeld te groot of dynamisch van aard wordt, zal er geen exacte oplosmethodiek bestaan. Een EA daarentegen kan de omgeving goed modelleren en oplossen. Wellicht dat de EA in combinatie met andere (lokale zoek)technieken gecombineerd wordt, om de uiteindelijke resultaten te verbeteren.

12. Literatuurlijst

Adams, J., Balas, E., Zawack, D. (1988), The shifting bottleneck procedure for job-shop scheduling, Management Science, 34(3), pp. 391-401

Bierwirth, C., Kropfer, H., Mattfeld, D.C. en Rixen, I. (1995), Genetic Algorithm based Scheduling in an Dynamic Manufacturing Environment, IEEE Conf. on Evolutionary Computation, pp. 439-443

Bongaerts, L. (1998), Integration of scheduling and control in holonic manufacturing systems, Katholieke Universiteit Leuven, België

Cai, L.W., Wu, Q.H., and Yong, Z.Z (2000), A Genetic Algorithm with Local Search for Solving Job Shop Problems, EvoWorkshops 2000, Real-World Applications of Evolutionary Computing, pp. 107-116

Choi, H.R., Kim, H.S., Park, B.J., Park, Y.J., and Whinston, A.B (2004), An agent for selecting optimal order set in EC marketplace, Decision Support System (36:4), pp. 371-383

Davis, L. (1985), Job-shop scheduling with genetic algorithms, Proceedings of the 1st International Conference on Genetic Algorithms, pp. 136-140

Eiben, A.E. (2002), Multiparent Recombination in evolutionary computing, advances in Evolutionairy Computing, pp. 175-192

Eiben, A.E., and Smith, J.E. (2003), Introduction to Evolutionary Computing, Springer-Verlag, New York

Eshelman, L.J. , Schaffer, J.D. (1991), Preventing premature convergence in genetic algorithms by preventing incest, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kauffman, pp. 115-122

Esquivel, S., Ferrero, S., Gallard, R., Salto, C., Alfonso, H., and Schütz, M., (2002), Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem. Knowledge-Based Systems 15 (1-2), pp. 13-25

Fang, H., Ross, P. and Corne, D. (1993), A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems, Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 375-382

Fang, H., Ross, P. and Corne, D. (1994), A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems, 11th European Conference on Artificial Intelligence, pp. 590-594

Garen, J. Multiobjective Job-Shop Scheduling With Genetic Algorithms Using a New Representation and Standard Uniform Crossover, Department of Economics, University of Osnabrück

B. Giffler and G. L. Thompson (1960), Algorithms for solving production scheduling problems, Operations Research (8), pp. 487-503

Goncalves, J.F., Mendes, J.J. de Magalhaes, Resende, M.G.C. (2002), A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem, AT&T Labs Research Technical Report TD-5EALJ6J, September 2002

Grabowski, J., Nowicki, E., and Zdrzalka, S. (1986), A Block Approach for Single Machine Scheduling with Release Dates and Due Dates, Euopean Journal of Operational Research, Vol. 26, pp. 278-285

Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. (1979), Optimization and Approximation in Deterministic Sequencing and Scheduling: A survey, Annals of Discrete mathematics (5), pp 287-326

Hart, E., Ross, P. (2000), A Systematic Investigation of GA Performance on Jobshop Scheduling Problems, EvoNET Workshop, Edinburgh, Real-World Applications of Evolutionary Computing, EvoWorkshops 2000, Edinburgh

Hart, E., Ross, P. (2000), Enhancing the Performance of a GA through Visualisation, Genetic and Evolutionary Computation Conference, Las Vegas

Hart, E., Ross, P. (1998), A Heuristic Combination Method for Solving Job-Shop Scheduling Problems, Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, pp. 845-854

Kobayashi, S., Ono, I., and Yamamura, M. (1995), An Efficient Genetic Algorithm for Job Shop Scheduling Problems, Proceedings of the 6th International Conference on Genetic Algorithms, pp. 506-511

Lenstra, J.K. and Rinnoy Kan, A.H.G. (1979), Computational complexity of discrete optimisation problems, Annals of Discrete Mathematics, Vol. 4, pp. 121-140

Lin, S.C., Goodman, E.D., Punch, W.F. III (1997), A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems, Proceedings of the 7th International Conference on Genetic Algorithms, pp. 481-488

Lin, S.C., Goodman, E.D., Punch, W.F. III (1997), Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems, Evolutionary Programming VI, Proc. Sixth Internat. Conf., pp.383-394

Louis, S (2002), www.cs.unr.edu/~sushil/papers/conference/newpapers/2002/gecco2002/ cigar Scheduling/

Madureira, A., Ramos, C., en Silva, S.C. (2001), A Genetic Approach for Dynamic Job-Shop Scheduling Problems, 4th Metaheuristics International Conference, pp. 41-47

Mesghouni, K. Hammadi, S. and Borne, P. On Modeling Genetic Algorithms for Flexible Job-shop Scheduling Problems, Ecole Centrale de Lille

Nowicki, E. and Smutnicki, C. (1996), A Fast Taboo Search Algorithm for the Job-Shop Problem, Management Science, Vol. 42, No. 6, pp. 797-813

Oh, J., Wu, C. (2004), Genetic-algorithm-based real-time task scheduling with multiple goals, The Journal of Systems and Software, pp. 245-258

Ombuki, M, Ventresca, M. (2002), Local Search Genetic Algorithms for the Job Shop Scheduling Problem, Technical Report # CS-02-22, November 2002

Ono, I., Yamamura, M., Kobayashi, S. (1996), A Genetic Algorithm for Job-shop Scheduling Problems Using Job-based Order Crossover, Proceedings of 1996 IEEE International Conference on Evolutionary Computation, pp. 547-552

Pareto, V. (1896), Cours d'Economie Politique, Rouge, Lausanne, Switzerland

Raman, N., Rachamaldugu, R.V., Talbot, F.B. (1989), Real-time Scheduling of an Automated Manufacturing center, European Journal of Operations Research, vol. 40, pp. 222-242

Reeves, C.R. (1994), Genetic algorithms and neighbourhood search, In Evolutionary Computing, AISB Workshop, pp. 115-130

Smith, S., Fox, M., Allen, B, Strohm, G. (1983), Isis: A constraint-directed reasoning approach to job-shop scheduling, Trends & Applications Automating Intelligent Behaviour, pp. 76-81

Song, Y., Hughes, J.G., Azarmi, N., and Voudouris, C. (1999), A Genetic Algorithm with an incomplete representation for the Job Shop Scheduling Problems, UK Planning and Scheduling SIG 18

Tsujimura, Y., Mafune, Y., Gen, M. (2001), Effects of Symbiotic Evolution in Genetic Algorithms for Job-Shop Scheduling, Proceedings of the 34th Hawaii International Conference on System Sciences

Vázquez, M., and Whitley, L.D. (2000), A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem, GECCO 2000, pp. 1011-1018

Vázquez, M., and Whitley, L.D. (2000), A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem, Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, pp. 303-312

Vilela, C., Brito, F., Rocha, M., Neves, J. (1999), An Evolutionary and Genetic View of the Job-Shop Scheduling Problem, Proceedings of the 11th European Simulation Symposium and Exhibition (ESS99)

Werner, J.C., Aydin, M.E., Fogarty, T.C. (2000), Evolving genetic algorithm for Job Shop Scheduling problems, Proceedings of ACDM 2000, PEDC, University of Plymouth, UK

Yamada, T., and Nakano, R. (1995), A Genetic Algorithm with Multi-Step Crossover for Job-Shop Scheduling Problems, Proceedings of the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95), pp. 146-151

Yamada, T., and Nakano, R. (1997), Genetic Algorithms for Job-Shop Scheduling Problems, Proceedings of Modern Heuristic for Decision Support, UNICOM seminar, London, pp. 67-81

Appendix A. Niet-vertragende algoritme

Niet-vertragend algoritme

stap 1. Zet t = 0, PS_t wordt een leeg schema, S_t wordt de reeks van alle operaties die als eerst uitgevoerd moeten worden voor de jobs en $r_{jk} = 0$ voor alle operaties $o_{jk} \in S_t$

stap 2. bepaal $r_{ji}^* = \min_{(j,k) \in S_t} \{r_{jk}\}$ en de machine k* die overeenkomt met de operatie waarop φ* gerealiseerd moet worden. Als er meerdere machines zijn, dan wordt er random één uitgekozen.

stap 3. Creëer een conflicterende reeks C_t waarin alle operaties $o_{jk} \in S_t$ zitten die op machine k* uitgevoerd moeten worden en waarvan $r_{jk} = r_{jk}^*$ geldt.

Stap 4. Kies een operatie uit de conflicterende reeks C_t volgens de prioriteitsregel p_t en plaats deze operatie zo vroeg mogelijk in PS_t. Dit wordt het nieuwe schema PS_{t+1}. Als er meerdere operaties geselecteerd worden volgens de prioriteitregels, dan wordt er random één gekozen.

stap 5. Vernieuw de reeks S_t door de geselecteerde operatie hieruit te verwijderen en de volgende operatie van de job eraan toe te voegen. Bepaal de nieuwe waarden voor r_{jk} in S_t en hoog iteratie t met 1 op.

stap 6. Herhaal stap 2 tot en met stap 5 totdat alle operaties ingepland zijn en er dus een volledig schema gegenereerd is

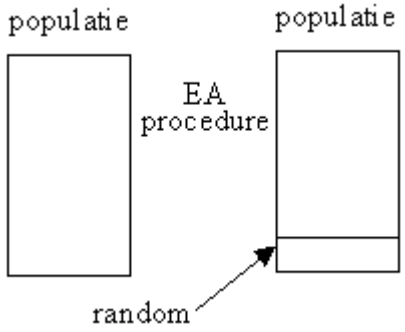
Appendix B. Geavanceerde EAs

Naast de gebruikelijke evolutionaire algoritmes, zoals deze besproken zijn in hoofdstuk 5, zijn er een aantal geavanceerde technieken ontwikkeld om de evolutionaire methoden net iets beter te maken. Dit kan door te zorgen voor nageslacht dat gewoon net iets beter is, door sneller de zoekruimte af te zoeken of door vroegtijdige convergentie te voorkomen. In deze appendix zullen een aantal onderwerpen aan bod komen die een beeld geven in hoeverre de standaard EAs uitgebreid kunnen worden naar iets geavanceerdere methoden die betere resultaten opleveren. Het nadeel is echter wel dat de extra behaalde kwaliteit ten koste kan gaan van de snelheid van de methode.

B.1 Vroegtijdige convergentie

In het evolutionaire proces bestaan twee belangrijke factoren die conflicterend met elkaar zijn: diversiteit van de populatie en de voorkeur die het mechanisme heeft om bepaalde individuen te selecteren. Aan de ene kant moet de zoekruimte goed doorzocht worden (*exploration*), maar aan de andere kant moet de verzamelde informatie uitgebuit worden (*exploitation*). Selectie speelt hierin een belangrijke rol, aangezien selectie aan de ene kant kan zorgen voor vroegtijdige convergentie maar aan de andere kant ook voor een inefficiënte zoekprocedure.
Zoals eerder al aangegeven is, wordt vroegtijdige convergentie in een lokaal minimum tegengegaan door te werken met een populatie van oplossingen.

Een belangrijke manier om dit verschijnsel te voorkomen is al veelvuldig bespreken, namelijk door het muteren van individuen. Een andere manier om vroegtijdige convergentie te voorkomen, is door alleen nieuwe waarden voor de doelfunctie toe te staan in een populatie. De diversiteit van een populatie kan ook gehandhaafd worden door een populatie van iedere generatie aan te vullen met willekeurig gegenereerde instanties (Goncalves, Mendes en Resende 2002). Bijvoorbeeld op dezelfde manier zoals de initialisatie heeft plaatsgevonden.



Figuur B.1: Een gedeelte van de bevolking wordt willekeurig vastgesteld.

Het is echter niet zo dat vroegtijdige convergentie nooit kan plaatsvinden, maar de kans hierop neemt wel af. Verder zijn er nog een aantal andere technieken die vroegtijdige convergentie voorkomen. Een aantal hiervan zal besproken worden (Esquivel e.a. 2002).

B.2 Multirecombinatie

In EAs is het gebruikelijk om twee ouders eenmaal kinderen voort te laten brengen. Deze procedure staat ook wel bekend als Single Crossover Per Couple (SCPC). Maar het zou best kunnen dat er meerdere kinderen voortgebracht worden uit een enkel paar ouders. Dit principe heet Multiple Crossover Per Couple (MCPC). Het aantal kinderen is dan variabel. Het voordeel hiervan is niet alleen dat er betere resultaten uit voortvloeien, maar ook de rekentijd gereduceerd wordt door een betere aftasting van de zoekruimte.

B.3 Multiparent

Naast het krijgen van meerdere kinderen, kunnen ook meerdere ouders zorgen voor deze kinderen. Door het gebruik van meer dan twee ouders wordt er voorzien in diversiteit.

Hierdoor wordt vroegtijdige convergentie vermeden. Eiben (1995) maakt gebruik van drie Scanning Crossover (SX) methodes: Uniform Scanning Crossover (USX), Occurence Based Scanning (OBSX) en Fitness Based Scanning (FBSX). Bij USX wordt iedere gen in het kind voorzien van één van de corresponderende genen van de ouders, met gelijke kans. OBSX selecteert die genwaarde die het meest voorkomt op een bepaalde positie bij de chromosomen van de ouders. FBSX kiest de te erven waarde proportioneel aan de doelwaarde van de ouders.
Wanneer multiparenting gecombineerd wordt met multirecombinatie wordt dit Multiple Crossovers on Multiple Parents (MCMP) genoemd.

B.4 Incest preventie (IP)

Het voorkomen van incest houdt in dat individuen die gemeenschappelijke voorouders hebben uit vorige generaties, niet tegelijkertijd geselecteerd mogen worden voor recombinatie. Dit heeft het voordeel dat het uitwisselen van dezelfde genen gereduceerd wordt, wat ertoe leidt dat vroegtijdige convergentie tegengewerkt wordt. Dit principe heeft echter ook tot gevolg dat het zoekproces vertraagd wordt, omdat de diversiteit in de populatie hoog blijft.
Het idee om incest te voorkomen is als eerst aangedragen door Eshelman en Shaffer (1991). De methode legde een paringsverbod op individuen op basis van de afstand tussen de ouders. Daarnaast kan dit uitgebreid worden door informatie van grootouders bij te houden voor chromosomen en de selectie voor reproductie op zo'n manier te wijzigen zodat voorkomen wordt dat beide ouders uit dezelfde 'familie' afkomstig zijn.

De volgende procedure zorgt ervoor dat meer dan twee ouders geselecteerd worden voor het produceren van nageslacht, maar voorkomen wordt dat individuen van dezelfde of opeenvolgende generaties (broer-zus of ouder-kind) hierbij betrokken zijn.

```
sla de geselecteerde ouders op in een array: ouders[# ouders]
sla de gecreëerde kinderen op in een array: kinderen[# kinderen]
produceer nageslacht
herhaal voor alle individuen
        selecteer een individu indiv1 uit de huidige populatie
        ouders[1] = indiv1
        i = 2
        herhaal voor i ≤ # ouders
                herhaal totdat een geselecteerd individu niet overeenkomt met een individu
                uit dezelfde of opeenvolgende generatie
                        selecteer een individu indivi uit de huidige populatie
                eind
                ouder[i] = indivi
                i = i + 1
        eind
        creëer kinderen door recombinatie (MCM) en mutatie uit te voeren
        selecteer het beste kind voor de volgende generatie
eind
```

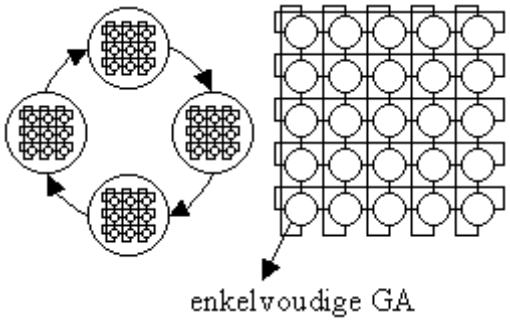
Algoritme B.1: Een voorbeeld hoe vroegtijdige convergentie verholpen kan worden.

B.5 Parallelle GA (PGA)

Een andere aanpak om vroegtijdige convergentie te vermijden, is door parallellisatie van het evolutionaire algoritme (Lin, Goodman en Punch). Beter gezegd kunnen bepaalde individuen wel en anderen niet met elkaar zorgen voor nageslacht. In de literatuur worden twee type parallelle GAs (PGAs) onderscheidden die op velerlei gebieden toegepast worden; coarse-grain GAs (cgGAs) en fine-grain GAs (fgGAs).
Bij fine-grained GAs worden individuen ruimtelijk zo opgeslagen zodat een individu alleen interactie kan hebben met individuen die 'dichtbij' zijn. Er worden dus een aantal buurten

(*neighborhoods*) onderscheiden in de populatie en binnen zo'n buurt wordt er gezorgd voor nageslacht. Het is echter mogelijk dat een individu zich voortbeweegt van de ene buurt naar de andere. De tweede PGA is de coarse-grained GA, ook wel island-parallel GA genoemd. Bij deze methode wordt de totale populatie opgesplitst in subpopulaties, die ieder afzonderlijk van elkaar behandeld worden als een reguliere GA. In deze aanpak is het echter ook wel mogelijk dat individuen van de ene subpopulatie overgaan naar de andere subpopulatie (migreren), maar wel minder frequent dan bij fgGAs.

Uit onderzoek blijkt dat cgGAs beter zijn, omdat fgGAs eerder de diversiteit verliezen. Verder blijkt uit onderzoek dat hoe groter het aantal subpopulaties in cgGAs, des te beter de diversiteit bewaard blijft. Dus meer eilanden in plaats van grotere eilanden is beter. Een combinatie van de twee PGAs is echter ook mogelijk. Het is dan beter om de cgGA structuur toe te passen in de fgGA structuur dan andersom. Beide mogelijkheden zijn terug te vinden in figuur B.2



Figuur B.2: Meerdere eilanden met buurten (links) is slechter dan per buurt een eiland (rechts). Dit heeft er weer mee te maken dat het beter is om meerdere kleine eilanden te hebben dan minder, maar grotere eilanden.

Parallelle evolutionaire algoritmes zijn in het bijzonder erg effectief voor optimalisatie problemen aangezien ze de rekentijd sterk kunnen reduceren terwijl dit niet ten koste gaat van de kwaliteit van de oplossingen.

B.6 Symbiotic evolutie

Een andere manier om de diversiviteit te bewaren is door tussentijds gedeeltelijk een andere doelstelling aan te houden. Dit wordt ook wel *symbiotic* evolutie genoemd (Tsujimura, Mafune en Gen 2001). Dit houdt in dat er een co-evolutieproces plaatsvindt, waarbinnen één hoofddoelstelling gekozen wordt die gewoon evalueert en eens in de x aantal generaties gaat een gedeelte van de bevolking naar een sub-evolutieproces waarin die individuen onderhevig worden aan een andere doelstelling.

	10x10 FT[3] (930 optimaal)			20x5 FT[3] (1165 optimaal)		
	GA	SyGA1	SyGA2	GA	SyGA1	SyGA2
beste	996	937	930	1210	1189	1178
gemiddelde	993	965.85	965.50	1251.10	1214.90	1233.75
std. afwijk.	19.45	15.85	17.09	24.21	15.51	23.24

Tabel B.1: Wanneer gekeken wordt naar GAs met alleen makespan als doelfunctie (GA), naar co-evolutie (SyGA1) en co- en sub-evolutie (SyGA2) dan blijkt co-evolutie de oplossingen te verbeteren.

B.7 Gene-Variance based Operator Targeting (GVOT)

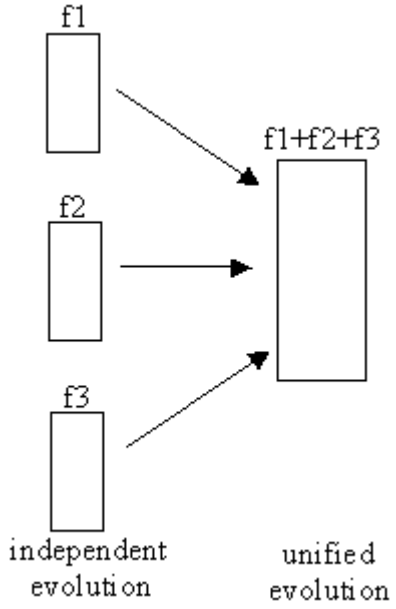
Bij *gene-variance based operator targeting* wordt de diversiteit van iedere gen uitgedrukt in de variantie (Fang, Ross en Corne 1993). Deze variantie geeft dan aan hoe groot de kans op recombinatie en mutatie is, volgens het roulette-wheel principe. Voor de mutatie moet echter

[3] Deze probleeminstanties zullen in Appendix E besproken worden.

de inverse van de variantie genomen worden, zodat genen met hoge variantie eerder gekozen zullen worden voor crossover en met lage variantie voor mutatie.

Appendix C. Meerdere doelstellingen

Er zijn meerdere manieren om met meerdere doelstellingen om te gaan (Oh en Wu 2003, Garen). Ten eerste kan ervoor gekozen worden om de verschillende doelstellingen om te zetten in een doelfunctie en die vervolgens optimaliseren. Dit proces zou verbeterd kunnen worden via multistage evolution (MSE). Hierin worden eerst sub-populaties gecreëerd voor iedere doelstelling en die worden vervolgens onafhankelijk van elkaar geëvolueerd totdat er convergentie optreedt (independent evolution). Vervolgens worden deze oplossingen van de gedeeltelijke doelfunctie samengenomen en kan de evolutie beginnen voor de geaggregeerde doelfunctie (unified evolution). Na convergentie kan de populatie weer opgedeeld worden in sub-populaties (door de individuen van unified evolutie te sorteren op hun doelstellingen) en kan er weer van voren af aan gestart worden totdat er aan een bepaald stopcriterium wordt voldaan.



Figuur C.1: Bij multistage evolutie wordt voor iedere doelstelling een onafhankelijk evolutionaire algoritme uitgevoerd, om ze vervolgens samen te voegen voor de evolutie van de samengestelde doelfunctie.

	enkelvoudige GA				Multiple stage evolutie			
	makespan	Max earliness	Weighted complet.	Som	makespan	Max earliness	Weighted complet.	Som
La26	1705	0	20021	1047.5	1691	14	19874	1047.6
La27	1720	9	21119	1065.9	1727	2	20313	1062.3
La28	1698	4	20150	1046.5	1692	10	20119	1046.7
La29	1591	8	18823	982.9	1595	4	18568	981.7
La30	1879	18	20516	1158.6	1870	27	20412	1158.6

Tabel C.1: Benchmark instanties voor multistage evolutie (Esquivel e.a. 2002).

Vilfredo Pareto (1896) kwam echter met een geheel andere aanpak. Hij gaat er vanuit dat oplossingen met elkaar vergeleken kunnen worden en dat geconstateerd kan worden of bepaalde oplossingen elkaar domineren. Bij het gebruik van meerdere doelstellingen zullen er namelijk altijd bepaalde doelen conflicteren (anders is het tenslotte een enkelvoudige doelstelling). Iedere doelstelling correspondeert dan met een andere optimale oplossing, maar geen enkele van deze trade-off oplossingen is optimaal wanneer alle doelstellingen in ogenschouw worden genomen. Dit houdt in dat bij het gebruik van meervoudige doelstellingen er niet gezocht wordt naar één optimale oplossing, maar naar alle trade-off oplossingen. Een oplossing domineert dan een andere oplossing als deze op alle doelstellingen even goed of beter presteert en voor minimaal één doelstelling echt beter is.

uitleg dominantie	de waarden voor de doelstellingen van oplossing x zijn (x_1, x_2, ..., x_n) de waarden voor de doelstellingen van oplossing y zijn (y_1, y_2, ..., y_n) x domineert y als geldt: $x_i \leq y_i, \forall$ i en $x_i < y_i, \exists$ i

Het genetische algoritme moet dan als volgt aangepast worden: Er moet een archief geïntroduceerd worden die niet-dominerende oplossingen bevat. Als een kind één van deze oplossingen domineert, dan wordt het kind opgenomen in het archief ten koste van de andere oplossing. De fitness-value van een chromosoom wordt berekend door de inverse te nemen van het aantal oplossingen die het chromosoom domineren.
De selectie van de ouders voor recombinatie kan echter wel op een slimme manier plaatsvinden, door de populatie te sorteren op de verschillende doelstellingen en vervolgens de beste individuen van de verschillende doelstellingen te selecteren.

Appendix D. Dynamische JSSP

De meerderheid van het onderzoek naar het gebruik van evolutionaire methoden voor het JSSP richt zich op het statische, deterministische JSSP. Maar een productieomgeving kent in de praktijk veel veranderingen, zoals nieuwe jobs die binnenkomen, machines die kapot gaan etc. Door deze dynamische aard worden scheduling problemen in de praktijk erg complex. Het is daarom erg lastig om hier rekening mee te houden in een toepassing. Dit wordt vanwege de grote complexiteit ook bijna niet onderzocht. Vaak worden deze dynamische problemen aangepakt met een groot aantal prioriteitregels. Zulke regels zijn gemakkelijk vast te stellen en in gebruik te nemen. Het enige nadeel is dat ze zelden tot nooit een optimaal schema produceren. Het is daarom zeker een onderzoeksgebied waarin evolutionaire algoritmes uitkomst kunnen bieden.
Binnen de klasse van dynamische JSSPs kan onderscheid gemaakt worden tussen stochastische en deterministische JSSPs. Zoals in Hoofdstuk 3 is uitgelicht verschilt het stochastische JSSP in het opzicht dat de gebeurtenissen niet van tevoren vaststaan (zoals dat bij een deterministisch JSSP wel het geval is) maar willekeurig kunnen plaats vinden. Hoe er met beide situaties omgegaan kan worden in een EA zal in deze appendix beschreven en uitgewerkt worden (Madureira, Ramos en Silva 2001).

D.1 Deterministische JSSP

De oplosmethode die Lin e.a. voordragen om met een deterministisch job shop scheduling probleem om te gaan, is vrij eenvoudig en is in principe niet veel anders dan de eerste stap van G&T algoritme te vervangen door:

stap 1. Zet t = 1, PS_t wordt een leeg schema, S_t wordt de reeks van alle operaties die als eerst uitgevoerd moeten worden voor de jobs en $r_{jk} = r_j$ voor alle operaties $o_{jk} \in S_t$

Hierdoor kan vanaf het begin af aan al rekening gehouden worden met de jobs die in de toekomst pas vrijgegeven worden.

D.2 Van stochastische JSSP naar deterministische JSSP

Om een oplossing voor het stochastische JSSP te kunnen bepalen, moet het stochastische gedeelte vervangen worden door iets deterministisch. Een stochastische JSSP zou hiervoor opgedeeld kunnen worden in een aantal deterministische problemen met de methode die Raman e.a. (1989) voorstelt. Een deterministisch probleem wordt dan gegenereerd op het moment dat een niet-deterministische gebeurtenis plaatsvindt, deze kan vervolgens met behulp van een EA opgelost worden. Hoe dit in zijn werking gaat, staat in de volgende sectie beschreven.

D.3 Rescheduling

Het reschedulingsprobleem heeft betrekking op het aanpassen van een reeds bestaand schema, dat wellicht al uitgevoerd wordt. Eventuele aanpassingen kunnen veroorzaakt worden door gewijzigde, geannuleerde of nieuwe jobs. Bij gewijzigde jobs moet gedacht worden aan andere bewerkingstijden of gewijzigde uitgiftetijdstippen van operaties. Dit soort problemen komt in de praktijk vaak voor bij organisaties die ook te maken hebben met JSSPs (Gang, Ross en Corne 1993).

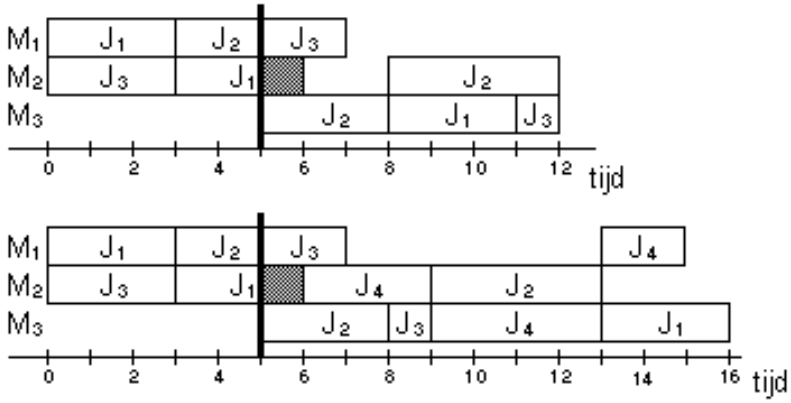
Over het algemeen wordt er wel onderscheid gemaakt tussen gedeeltelijke gebeurtenissen, waardoor maar een gedeelte van het huidige schema gewijzigd wordt. Denk hierbij aan een verandering in de job attributen, zoals het uitgiftetijdstip of de vervaldatum. Dit leidt tot een modificatie procedure en een nieuwe evaluatie van de fitness functie. Een totale gebeurtenis daarentegen is een verandering in structuur van de gehele populatie. Denk hierbij aan nieuwe of afannuleerde jobs. Hierdoor moet de samenstelling van iedere chromosoom aangepast worden, aangezien genen toegevoegd of verwijderd moeten worden. Een nieuwe evaluatie van de fitness functie is ook noodzakelijk.

Een mogelijke oplossing is om de oude populatie te negeren en een compleet nieuw schema te maken vanuit het niets. Dit is duidelijk niet effectief, omdat het niet alleen veel tijd kost maar het zal ook vaker voorkomen. Er zijn verschillende manieren om dit probleem aan te pakken. Ten eerste kan het al reeds uitgevoerde werk (voor het bestaande schema) hergebruikt worden en deze dan iteratief wijzigen. Een andere methode maakt gebruik van een speciale eigenschap van EAs, die Bierwirth e.a. (1995) opmerkte. Wanneer de populatie in de laatste periode bekeken wordt, zal maar een kleine fractie van de informatie in de populatie gewijzigd worden omdat maar een klein aantal operaties gewijzigd zullen worden uit het oude (deterministische) job shop probleem. Dus bij de initialisatie stap van de EA kan gebruik gemaakt worden van deze individuen. Er moeten dan echter nog wel enige aanpassingen plaatsvinden.

Lin e.a. (1997) verwezenlijken deze veranderingen aan de hand van het G&T algoritme. Als job j op dat tijdstip afgerond is, verdwijnt job j uit het systeem. Wanneer dit echter niet het geval is, worden die operaties verwijderd die uitgevoerd zijn (of op dat moment uitgevoerd worden). Verder verandert de uitgiftedatum ook. Aangezien een machine niet gelijk beschikbaar hoeft te zijn, wordt aangenomen dat op tijdstip a_k machine k weer beschikbaar is. De eerste stap van het G&T algoritme moet veranderd worden in:

stap 1. Zet t = 1, PS_t wordt een leeg schema, S_t wordt de reeks van alle operaties die als eerst uitgevoerd moeten worden voor de jobs en $r_{jk} = \max\{r_j, a_k\}$ voor alle operaties $o_{jk} \in S_t$

Stel dat er bijvoorbeeld op tijdstip 5 een nieuwe job uitgegeven wordt in het bestaande schema van Hoofdstuk 3. In figuur D.1 is te zien hoe het aangepast algoritme in zijn werking treedt. In paragraaf D.5 wordt een uitgebreid overzicht gegeven hoe de omzetting van het schema plaatsgevonden heeft.



Figuur D.1: Op tijdstip 5 komt (onverwachts) een extra job binnen (J4). Het oude schema wordt daardoor aangevuld.

Dit principe kan ook toegepast worden op andere dynamische JSSPs, zoals machines die kapot gaan, geannuleerde jobs, wijzigingen in vervaldata etc.

Hart & Ross stellen een andere aanpak voor om het dynamische JSSP op te lossen. Hierin bestaat ieder gen in het chromosoom uit het paar (Methode, Heuristiek). Waarbij 'Methode' staat voor het G&T algoritme of het niet-vertragende algoritme en 'Heuristiek' staat voor de te hanteren heuristiek bij het construeren van het schema. De methode stelt dus vast hoe de conflicterende reeks gedefinieerd is en de heuristiek bepaalt welke operatie uit deze conflicterende reeks gehaald wordt. Verder kunnen simpele operatoren gedefinieerd worden, zoals uniform crossover op (M,H)-paren en mutatie op de heuristiek.
Deze manier van representeren is getest tegen de verschillende componenten waaruit de representatie is opgebouwd. De resultaten onder 48 experimenten staan in onderstaande tabel:

	beter	gelijk	slechter
vergelijking met GA	40	5	3
vergelijking met priority rules	44	2	2
vergelijking met alleen G&T	25	20	3
vergelijking met alleen ND	40	4	4

Een derde methode is het gebruik van order-based operatoren in combinatie met technieken om actieve en non-delay schema's te maken. Het algoritme wordt ook wel Order-Based Giffler en Thompson (OBGT) genoemd en is ontwikkeld door Manuel Vazquez en L. Darrell Whitley. Order-based operatoren zullen niet altijd een toegelaten schema produceren, dus moet het G&T algoritme toegepast worden om het niet-toegelaten nageslacht te repareren.
Bij deze aanpak wordt gebruik gemaakt van een directe representatie; een permutatie van alle jobs per machine. Verder worden order-based operatoren toegepast. Davis ontwikkelde de eerste order-based operatoren die geschikt waren voor scheduling problemen. In deze OBGT methode wordt gebruik gemaakt van Uniform Order-based Crossoveren Order-based Scramble Mutation.

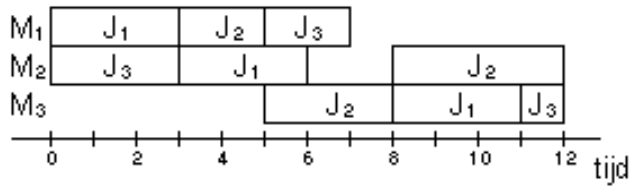
```
initialiseer de beginpopulatie (50% non-delay schema's (ND) en 50% active schema's (GT))
herhaal tot convergentie
        herhaal voor ieder individu uit de bevolking i = 1, 2, ..., n
                pas order crossover toe met kans pc, waarin O1 = individu i en O2 = random
                pas order mutatie toe met kans pm
                repareer beide kinderen met het G&T algoritme
                vervang het slechtste individu met het beste kind, indien verbetert
                sorteer de bevolking op afnemende fitness
        eind
eind
```

Naast deze drie methodes zijn er nog meerdere methodes bekend:

- Bierwirth en Mattfeld (1999): de GA strategie is gebaseerd op permutatie operatoren, waarin de prioriteiten van iedere operatie vastgelegd wordt in de permutaties. Deze prioriteiten worden vervolgens gebruikt voor het selecteren van de operatie uit de conflicterende reeks.
- Fang (1994): de gekozen representatie geeft impliciet instructies aan de schema bouwer.

D.4 Uitwerking Dynamische JSSP

Als uitgangspositie wordt hetzelfde voorbeeld aangehouden zoals deze beschreven staat in hoofdstuk 3 (zie figuur E.2)



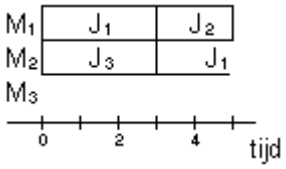
Figuur D.2: Het schema zoals deze in de statische situatie bepaald is.

Op het moment dat de omgeving dynamisch van aard is, dan kan het voorkomen dat op een ander tijdstip dan tijdstip 0 nieuwe jobs arriveren. Als dit van tevoren bekend is, dan spreekt men van een deterministisch dynamisch JSSP. Stel dat bekend is dat op tijdstip 5 een nieuwe job arriveert. Dan moet het aangepaste G&T algoritme gebruikt worden met de volgende kennis:

job	routering operaties (bewerkingstijd)			uitgiftedatum
J1	M1 (3)	M2 (3)	M3 (3)	0
J2	M1 (2)	M3 (3)	M2 (4)	0
J3	M2 (3)	M1 (2)	M3 (1)	0
J4	M2 (3)	M3 (4)	M1 (2)	5

Tabel D.1: Op tijdstip 5 komt een extra job (J4) aan.

Wanneer deze nieuwe job echter niet bekend is, maar op tijdstip 5 toch arriveert, dan wordt dit een stochastisch dynamisch JSSP genoemd. Om hier goed mee om te gaan, moeten tussentijds een aantal wijzigen doorgevoerd worden. Op tijdstip 5 is de situatie weergegeven in figuur D.3.



Figuur D.3: Het schema zoals deze afgewerkt is (en wordt) tot tijdstip 5.

Er zal opnieuw het G&T algoritme toegepast moeten worden vanaf tijdstip 5, maar ditmaal met een beperking op machine 2 aangezien deze nog bezig is op tijdstip 5. De volgende informatie wordt daarom gebruikt:

job	routering operaties (bewerkingstijd)			uitgiftedatum
J1	-	-	M3 (3)	6
J2	-	M3 (3)	M2 (4)	5
J3	-	M1 (2)	M3 (1)	5
J4	M2 (3)	M3 (4)	M1 (2)	5

machine	M1	M2	M3
tijdstip beschikbaarheid	5	6	5

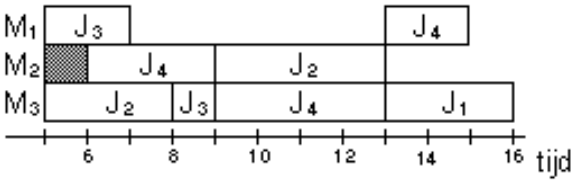
Tabel D.2: Het probleem en de overige gegevens zoals deze zich op tijdstip 5 voordoet.

Vervolgens wordt het G&T algoritme uitgevoerd, waarbij als prioriteitregel de random regel gebruikt wordt. In iedere iteratiestap zal de gekozen operatie onderstreept worden:

stap 1	$S_5 = \{o_{13}, o_{23}, o_{31}, o_{42}\}$ $\varphi^*_5 = \min\{6+3, 5+3, 5+2, 6+3\} = 7 \quad k^* = 1$ $C_5 = \{\underline{o_{31}}\}$

stap 2	$S_6 = \{o_{13}, o_{23}, o_{33}, o_{42}\}$ $\varphi^*_6 = \min\{9, 8, 8, 9\} = 8$ $C_6 = \{o_{13}, \underline{o_{23}}, o_{33}\}$	$k^* = 3$
stap 3	$S_7 = \{o_{13}, o_{22}, o_{33}, o_{42}\}$ $\varphi^*_7 = \min\{11, 12, 9, 9\} = 9$ $C_7 = \{o_{33}, \underline{o_{42}}\}$	$k^* = 2$
stap 4	$S_8 = \{o_{13}, o_{22}, o_{33}, o_{43}\}$ $\varphi^*_8 = \min\{11,13, 9, 13\} = 9$ $C_8 = \{\underline{o_{33}}\}$	$k^* = 3$
stap 5	$S_9 = \{o_{13}, o_{22}, o_{43}\}$ $\varphi^*_9 = \min\{12, 13, 13\} = 12$ $C_9 = \{o_{13}, \underline{o_{43}}\}$	$k^* = 3$
stap 6	$S_{10} = \{o_{13}, o_{22}, o_{41}\}$ $\varphi^*_{10} = \min\{16, 13, 15\} = 13$ $C_{10} = \{\underline{o_{22}}\}$	$k^* = 2$
stap 7	$S_{11} = \{o_{13}, o_{41}\}$ $\varphi^*_{11} = \min\{16, 15\} = 15$ $C_{11} = \{\underline{o_{41}}\}$	$k^* = 1$
stap 8	$\underline{o_{13}}$	

Dat levert uiteindelijk het volgende schema op



Figuur D.4: Vanaf tijdstip 5 verandert het schema door het arriveren van de nieuwe job 4.

D.5 Benchmark

Ook voor het dynamische job shop scheduling probleem zijn een aantal benchmark probleeminstanties bekend. Deze zijn toegepast op de verschillende aanpakken en daarbij is ook onderscheid gemaakt tussen de verschillende doestellingen, zoals blijkt uit tabel D.3 tot en met tabel D.8 (Lin e.a. 1997, Hart en Ross 1998, Vázquez en Whitley).

Probleem	Size	PDR[4]	Fang	THX-S[5]	THX-P[6]	HGA	HGA (ND)	HGA (G&T)	HGA (NR[7])	OBGT

[4] PDR = priority dispatch rule, een eenvoudige deterministische regel wordt gebruikt bij stap 4 van het G&T algoritme.
[5] THX-S = single GA met een THX operator
[6] THX-P = parallelle GA met een THX operator

JB1	10x3	0.178	0.164	0.162	0.162	0.163	0.180	0.163	0.163	0.163
JB2	10x3	0.086	0.087	0.086	0.086	0.086	0.086	0.086	0.086	0.086
JB4	10x5	0.560	0.556	0.559	0.559	0.056	0.557	0.556	0.556	0.556
JB9	15x3	0.185	0.177	0.169	0.169	0.170	0.170	0.181	0.170	0.169
JB11	15x5	0.000	0.000	0.000	0.000	0.000	0	0	0	0.000
JB12	15x5	0.218	0.139	0.139	0.139	0.139	0.219	0.139	0.139	0.139
LJB1	30x3	0.276	0.215	0.224	0.190	0.192	0.279	0.192	0.194	0.201
LJB2	30x3	0.460	0.459	0.410	0.395	0.407	0.419	0.437	0.407	0.407
LJB7	50x5	0.109	0.110	0.090	0.060	0.055	0.069	0.091	0.055	0.055
LJB9	50x5	0.796	0.982	0.742	0.615	0.630	0.674	0.917	0.630	0.630
LJB10	50x8	0.479	0.455	0.478	0.438	0.418	0.442	0.438	0.418	0.407
LJB12	50x8	0.489	0.478	0.433	0.399	0.392	0.397	0.424	0.399	0.342

Tabel D.3: De verschillende waarden met weighted tardiness T_{wt} als doelfunctie.

Probleem	size	PDR	Fang	THX-S	THX-P	HGA	HGA (ND)	HGA(G&T)	HGA(N R)	OBGT
JB1	10x3	-0.713	-0.168	-0.173	-0.173	-0.167	-0.167	-0.167	-0.167	-0.173
JB2	10x3	-0.812	-0.824	-0.816	-0.839	-0.819	-0.814	-0.816	-0.819	-0.819
JB4	10x5	0.497	0.490	0.493	0.493	0.489	0.494	0.489	0.489	0.489
JB9	15x3	-0.047	-0.073	-0.078	-0.078	-0.079	-0.073	-0.072	-0.079	-0.079
JB11	15x5	-0.607	-0.655	-0.730	-0.751	-0.702	-0.702	-0.580	-0.636	-0.743
JB12	15x5	-0.082	-0.102	-0.103	-0.103	-0.102	-0.101	-0.102	-0.102	-0.102
LJB1	30x3	-0.112	-0.195	-0.206	-0.214	-0.224	-0.126	-0.224	-0.224	-0.224
LJB2	30x3	0.013	-0.043	-0.077	-0.078	-0.078	-0.051	-0.022	-0.051	-0.078
LJB7	50x5	-0.411	-0.414	-0.453	-0.507	-0.523	-0.501	-0.399	-0.523	-0.523
LJB9	50x5	0.622	0.702	0.498	0.354	0.388	0.423	0.595	0.388	0.388
LJB10	50x8	-0.038	-0.096	-0.082	-0.108	-0.139	-0.136	-0.041	-0.134	-0.413
LJB12	50x8	0.256	0.221	0.192	0.113	0.121	0.121	0.310	0.136	0.086

Tabel D.4: De verschillende waarden met weighted lateness L_{wt} als doelfunctie.

Probleem	size	PDR	Fang	THX-S	THX-P	HGA	HGA (ND)	HGA(G&T)	HGA(NR)	OBGT
JB1	10x3	1.231	1.237	1.231	1.231	1.236	1.236	1.236	1.236	1.231
JB2	10x3	1.772	1.778	1.768	1.768	1.766	1.771	1.766	1.766	1.766
JB4	10x5	1.111	1.109	1.108	1.108	1.107	1.112	1.107	1.107	1.107
JB9	15x3	1.947	1.768	1.754	1.754	1.754	1.759	1.760	1.754	1.753
JB11	15x5	1.795	1.794	1.723	1.706	1.706	1.716	1.834	1.706	1.706
JB12	15x5	1.257	1.259	1.256	1.256	1.256	1.258	1.256	1.256	1.256
LJB1	30x3	1.494	1.431	1.424	1.391	1.389	1.487	1.389	1.389	1.389
LJB2	30x3	1.924	1.826	1.783	1.777	1.777	1.804	1.837	1.782	1.777
LJB7	50x5	1.692	1.669	1.623	1.557	1.542	1.563	1.709	1.542	1.542
LJB9	50x5	2.490	2.659	2.475	2.324	2.337	2.394	2.626	2.337	2.341
LJB10	50x8	1.776	1.728	1.731	1.697	1.673	1.680	1.761	1.675	1.663
LJB12	50x8	2.207	2.138	2.115	2.080	2.058	2.058	2.237	2.068	2.007

Tabel D.5: De verschillende waarden met weighted flowtime F_{wt} als doelfunctie.

probleem	size	PDR	Fang	THX-S	THX-P	HGA	OBGT
JB1	10x3	0.529	0.475	0.474	0.474	0.474	0.474
JB2	10x3	0.901	0.758	0.690	0.499	0.753	0.753
JB4	10x5	0.622	0.620	0.621	0.621	0.619	0.619
JB9	15x3	0.395	0.384	0.369	0.369	0.370	0.369

[7] NR = non random heuristeuk (de random prioriteitsregel doet niet mee)

JB11	15x5	0.415	0.263	0.262	0.262	0.271	0.262
JB12	15x5	0.494	0.247	0.246	0.246	0.247	0.247
LJB1	30x3	0.654	0.322	0.322	0.279	0.280	0.279
LJB2	30x3	0.868	0.632	0.627	0.601	0.598	0.598
LJB7	50x5	0.515	0.374	0.345	0.254	0.269	0.246
LJB9	50x5	0.935	1.178	0.833	0.739	0.797	0.819
LJB10	50x8	0.882	0.621	0.688	0.598	0.548	0.512
LJB12	50x8	0.667	0.607	0.584	0.461	0.466	0.399

Tabel D.6: De verschillende waarden met weighted earliness E_{wt} als doelfunctie.

probleem	size	PDR	Fang	THX-S	THX-P
JB1	10x3	0.082	0.082	0.082	0.082
JB2	10x3	0.055	0.055	0.055	0.055
JB4	10x5	0.203	0.152	0.152	0.152
JB9	15x3	0.067	0.061	0.044	0.044
JB11	15x5	0.002	0	0	0
JB12	15x5	0.071	0.060	0.060	0.060
LJB1	30x3	0.056	0.041	0.039	0.032
LJB2	30x3	0.078	0.071	0.060	0.060
LJB7	50x5	0.022	0.019	0.016	0.016
LJB9	50x5	0.050	0.075	0.048	0.039
LJB10	50x8	0.043	0.040	0.040	0.034
LJB12	50x8	0.035	0.043	0.035	0.031

Tabel D.7: De verschillende waarden met maximum tardiness T_{max} als doelfunctie.

probleem	size	PDR	Fang	THX-S	THX-P	HGA	HGA (ND)	HGA(G&T)	HGA(NR)
JB1	10x3	0.529	0.475	0.474	0.474	0.474	0.527	0.474	0.474
JB2	10x3	0.901	0.758	0.690	0.499	0.753	0.905	0.753	0.753
JB4	10x5	0.622	0.620	0.621	0.621	0.619	0.620	0.619	0.619
JB9	15x3	0.395	0.384	0.369	0.369	0.370	0.379	0.381	0.374
JB11	15x5	0.415	0.23	0.262	0.262	0.271	0.403	0.271	0.271
JB12	15x5	0.494	0.247	0.246	0.246	0.247	0.488	0.247	0.247
LJB1	30x3	0.654	0.322	0.322	0.279	0.280	0.653	0.280	0.280
LJB2	30x3	0.868	0.632	0.627	0.60	0.602	0.751	0.612	0.598
LJB7	50x5	0.515	0.374	0.345	0.254	0.284	0.449	0.287	0.269
LJB9	50x5	0.935	1.178	0.833	0.739	0.832	0.854	0.967	0.797
LJB10	50x8	0.882	0.621	0.688	0.598	0.566	0.0691	0.548	0.567
LJB12	50x8	0.667	0.607	0.584	0.461	0.522	0.469	0.473	0.466

Tabel D.8: De verschillende waarden met weighted earliness en tardiness ET_{wt} als doelfunctie.

Uit de benchmark instanties blijkt dat een parallelle EA betere resultaten levert, maar deze aanpak kost wel 10 keer zoveel rekentijd in vergelijking tot OBGT. OBGT doet het wel goed voor de wat grotere probleeminstanties.

Appendix E. Benchmark

In deze appendix staat een overzicht van de resultaten behorend bij de verschillende aanpakken van een evolutionaire algoritme die besproken zijn in dit verslag.

Allereerst zullen de twee instanties van Muth en Thompson (1963) gebenchmarkt worden, aangezien deze veelvuldig toegepast worden.

		MT 10x10 (930)		MT 20x5 (1165)	
methode		Beste	gemiddelde	Beste	gemiddelde
Branch-and-bound	McMahon ‘75	972		1165	
	Baker ‘85	960		1303	
	Carlier ‘89	930		1165	
Zoek methoden	Adams e.a. ’88 (SB)	930		1178	
	CBSA	930	930.8	1178	1178
	GA/GT+ECO	930	963	1181	1213
	PGA+SBP	930	947	1165	1188
	GA/MSX	930	934.5	1165	1177.3
	Nakano ‘91	965		1215	
	Yamada ‘92	930		1184	
	Storer ‘93	954		1180	
	Dorndorf ‘93	930		1165	
	Fang ‘93	960	985	1213	1225
	+GVOT	949	977	1189	1215
	Juels ‘94	937		1174	
	Mattfeld ‘94	930		1165	
	Dorndorf ‘95	938		1178	
	Bierwirth ‘95	936		1181	
	Kobayashi ‘95	930		1178	
	Lin (THX)	930		1165	
	Yamada ‘96	930		1165	

Tabel E.1: Benchmark resultaten uit Fang, Ross en Corne (1993), Lin et al., Yamada en Nakano (1995, 1997).

Daarnaast worden een aantal instanties van Steve Lawrence (LA) vaak aangehaald.

			Hybride GA		GRASP		TS	GA						
probl	grootte	beste	Goncalves e.a.	Wang e.a.	Aiex e.a.	Binato	Nowicki e.a.	Goncalves e.a.	Croce	Dorndorf & Pesch			Aarts e.a.	
										P-GA	SBGA	SBGA	GLS1	GLS2
MT 06	6x6	55	55	55	55	55	55	55	-	-	-	-		-
MT 10	10x10	930	930	930	930	938	930	936	946	960	-	-	935	945
MT 20	20x20	1165	1165	1165	1165	1169	1165	1177	1178	1249	-	-	1165	1167
LA 01	10x5	666	666	666	666	666	666	666	666	666	666	-	666	666
LA 02	10x5	655	655	-	655	655	655	666	666	681	666	-	668	659
LA 03	10x5	597	597	-	597	604	597	597	666	620	604	-	613	609
LA 04	10x5	590	590	-	590	590	590	590	-	620	590	-	599	594
LA 05	10x5	593	593	-	593	593	593	593	-	593	593	-	593	593
LA 06	15x5	926	926	926	926	926	926	926	926	926	926	-	926	926
LA 07	15x5	890	890	-	890	890	890	890	-	890	890	-	890	890
LA 08	15x5	863	863	-	863	863	863	863	-	863	863	-	863	863
LA 09	15x5	951	951	-	951	951	951	951	-	951	951	-	951	951
LA 10	15x5	958	958	-	958	958	958	958	-	958	958	-	958	958
LA 11	20x5	1222	1222	1222	1222	1222	1222	1222	1222	1222	1222	-	1222	1222
LA 12	20x5	1039	1039	-	1039	1039	1039	1039	-	1039	1039	-	1039	1039
LA 13	20x5	1150	1150	-	1150	1150	1150	1150	-	1150	1150	-	1150	1150
LA 14	20x5	1292	1292	-	1292	1292	1292	1292	-	1292	1292	-	1292	1292
LA 15	20x5	1207	1207	-	1207	1207	1207	1207	-	1237	1207	-	1207	1207
LA 16	10x10	945	945	945	945	946	945	977	979	1008	961	961	977	977
LA 17	10x10	784	784	-	784	784	784	787	-	809	787	784	791	791
LA 18	10x10	848	848	-	848	848	848	848	-	916	848	848	856	858
LA 19	10x10	842	842	-	842	842	842	857	-	880	863	848	863	859
LA 20	10x10	902	907	-	902	907	902	910	-	928	911	910	913	916
LA 21	15x10	1053	1046	1058	1057	1091	1047	1047	1097	1139	1074	1074	1084	1085
LA 22	15x10	927	935	-	927	960	927	936	-	998	935	936	954	944
LA 23	15x10	1032	1032	-	1032	1032	1032	1032	-	1072	1032	1032	1032	1032
LA 24	15x10	935	953	-	954	978	939	955	-	1014	960	957	970	981
LA 25	15x10	977	986	-	984	1028	977	1004	-	1014	1008	1007	1016	1010
LA 26	20x10	1218	1218	1218	1218	1271	1218	1218	1231	1278	1219	1218	1240	1236
LA 27	20x10	1269	1256	-	1269	1320	1236	1260	-	1378	1272	1269	1308	1300
LA 28	20x10	1216	1232	-	1225	1293	1216	1241	-	1327	1240	1241	1281	1265
LA 29	20x10	1195	1196	-	1203	1293	1160	1190	-	1336	1204	1210	1290	1260
LA 30	20x10	1355	1355	-	1355	1368	1355	1256	-	1411	1355	1355	1402	1386
LA 31	30x10	1784	1784	1784	1784	1784	1784	1784	1784	-	-	-	1784	1784
LA 32	30x10	1850	1850	-	1850	1850	1850	1850	-	-	-	-	1850	1850
LA 33	30x10	1719	1719	-	1719	1719	1719	1719	-	-	-	-	1719	1719
LA 34	30x10	1721	1721	-	1721	1753	1721	1730	-	-	-	-	1737	1730
LA 35	30x10	1888	1888	-	1888	1888	1888	1888	-	-	-	-	1894	1890
LA 36	15x15	1268	1279	1292	1287	1334	1268	1305	1305	1373	1317	1317	1324	1311
LA 37	15x15	1397	1408	-	1410	1457	1407	1441	-	1498	1484	1446	1449	1450
LA 38	15x15	1217	1219	-	1218	1267	1196	1248	-	1296	1251	1241	1285	1283

LA 39	15x15	1233	1246	-	1248	1290	1233	1264	-	1351	1282	1277	1279	1279
LA 40	15x15	1222	1241	-	1244	1259	1229	1252	-	1321	1274	1252	1273	1260

Tabel E.2: Goncalves, Mendes en Resende (2002)

Prob.	grootte	beste	MSMF	MSXF	Tabu Seach		SA			
					Nowi	Dell	CBSA	Aarts	GA (Matt)	Appl
LA21	15x10	1053	1046 (1049.9)	1046	1047	1048	1046	1053	1053	1053
LA24	15x10	935	935 (938.8)	935	939	941	935	935	938	935
LA25	15x10	977	977 (979.6)	977	977	979	977	983	977	977
LA27	20x10	-	1235 (1253.6)	1235	1236	1242	1235	1249	1236	1269
LA29	20x10	1130	1166 (1181.9)	1166	1160	1182	1154	1185	1184	1195
LA38	15x15	-	1196 (1198.4)	1196	1196	1203	1198	1208	1201	1209
LA40	15x15	1222	1224 (1227.9)	1224	1229	1233	1228	1225	1228	1222
Abz7	20x15	-	678 (692.5)	678	-	667	665	668	672	668
Abz8	20x15	-	686 (703.1)	686	-	678	675	670	683	687
Abz9	20x15	-	697 (719.6)	697	-	692	686	691	703	707

Tabel E.3: Yamada en Nakano (1997)

Naast de reeds genoemde instanties, worden ook instanties van Yamada en Nakano (YN) en Taillard (TA) en job gecorreleerde (JC) instanties voor het JSSP met elkaar vergeleken.

Prob.	beste	Tabu Search (Nowicki en Smutnicki)			THX (Lin)			HGA (Hart en Ross)			GA3 (Mattfeld)			OBGT (Vázquez en Whitley)		
		beste	gem.	st. afw.	beste	gem.	st. afw.	beste	gem.	st. afw.	beste	gem.	st. afw.	beste	gem.	st. afw.
MT06	55	55	55	0	55	55	0	55	55	0	55	55	0	55	55	0
MT10	930	930	943	11.5	930	1004	22.0	935	645	7.7	930	944	6.6	931	965	10.3
MT20	1165	1165	1171	6.0	11.65	1237	27.5	1179	1201	8.9	1165	1180	4.7	1165	1210	22.5
YN01	888	897	917	10.7	930	977	15.2	939	948	5.3	904	912	4.6	892	953	13.4
YN02	909	918	934	9.9	958	988	23.2	952	963	7.6	9.28	941	5.6	914	948	21.7
YN03	893	905	925	10.3	953	980	23.8	931	942	6.2	907	919	7.4	901	951.8	19.5
YN04	908	982	1000	12.8	1012	1049	16.9	991	1002	6.2	992	1012	10.2	972	1000	12.7
LA21	1046	1056	1060	3.3	1075	1097	23.4	1099	1107	6.7	1047	1059	6.4	1062	1078	20.6
LA27	1235	1267	1283	8.5	1349	1371	28.0	1308	1319	6.2	1236	1262	5.1	1276	1292	18.4
LA29	1152	1191	1211	13.7	1333	1355	41.7	1296	1306	9.7	11.80	1200	10.8	1205	1223	16.7
LA38	1196	1196	1211	15.8	1319	1337	29.9	1309	1318	7.6	1201	1223	12.2	1235	1251	13.4
TA51	2760	2765	2775	15.3	3004	3120	36.4	2858	2909	14.9	2870	2928	26.4	2947	3118	24.6
TA52	2756	2756	2767	22.2	3036	3127	35.0	2847	2889	16.3	2883	2933	26.4	2867	2993	30.3
TA53	2717	2718	2744	17.1	2860	2934	34.1	2770	2822	9.2	2742	2780	19.5	2836	2924	30.1
TA54	2839	2852	2862	16.8	2950	3054	41.2	2874	2909	15.7	2839	2852	14.3	2962	3066	26.5
TA55	2679	2682	2724	25.2	2929	3042	37.0	2798	2844	13.6	2798	2854	22.8	2940	3043	32.0
TA56	2781	2849	2860	23.1	3015	3112	31.2	2907	2938	12.3	2882	2921	23.4	3024	3103	26.5
TA57	2943	2943	2950	13.2	3116	3224	41.0	2971	3018	15.6	2989	3037	18.2	3129	3215	35.2
TA58	2885	2896	2918	18.1	3200	3252	33.6	2913	2994	12.9	2954	3001	21.1	3069	3150	29.7
TA59	2655	2655	2681	19.2	2922	3049	36.3	2745	2789	15.0	2742	2817	19.7	2968	3060	28.7
TA60	2723	2723	2742	17.1	2947	2995	34.1	2864	2887	8.5	2803	2827	11.3	2942	3020	26.7
JC01	760[8]	904	904	0	888	934	1438	922	926	2.6				885	909	8.3
JC02	1142[1]	1292	1292	0.2	1272	1352	12.5	1303	1326	6.6				1304	1345	9.4
JC03	1871[1]	2142	2142	0	2098	2143	28.6	2098	2143	8.1				2098	2165	12.4
JC04	2011[1]	2011	2011	0	2011	2011	0	2011	2011	0				2011	2011	0

Tabel E.4: Benchmark instanties (M. Vázquez en L.D. Whitley).

De OBGT methode en HGA zijn gebruikte methoden om het dynamische JSSP op te lossen (zie Appendix E), maar het zou ook gebruikt kunnen worden voor het statische JSSP.

[8] ondergrens

Appendix F. Preference-list based representatie

Het JSSP wordt gegeven door:

job	routering operaties (bewerkingstijd)		
J1	M1 (3)	M2 (3)	M3 (3)
J2	M1 (2)	M3 (3)	M2 (4)
J3	M2 (3)	M1 (2)	M3 (1)

Tabel F.1: Het 3×3 JSSP.

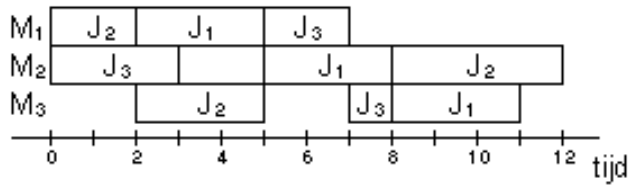
Stel dat een individu volgens de preference-list based representatie met het volgenhet chromosoom overeenkomt

[2 3 1 1 3 2 2 1 3]

De eerst drie waarden komen overeen met de preferentielijst voor machine 1. Dit individu prefereert de volgende operaties: job 2 op machine 1, job 1 op machine 2 en job 2 op machine 3. Volgens de volgorde relaties kan alleen job 2 op machine 1 ingepland worden. Dan komt deze operatie te vervallen en job 2 op machine 3 wordt op de preferentielijst gezet. De volgende operatie wordt vervolgens job 2 op machine 3. Nu kan er geen enkele operatie uit de preferentielijst gekozen worden, in verband met de vereiste volgordes. Er wordt dan gekeken naar de tweede geprefereerde operaties, dan kunnen zowel job 1 op machine 1 als job 3 op machine 2 uitgevoerd worden. Daarna kunnen job 3 op machine 1 en job 1 op machine 2 ingepland worden volgens de preferentielijst. Vervolgens kunnen de overige operaties ingescheduled worden.

1. preferentielijst 2 3 1, 1 3 2, 2 1 3,
   mogelijk uit te voeren O_{11} O_{21} O_{32}
2. preferentielijst ~~2~~ 3 1, 1 3 2, 2 1 3,
   mogelijk uit te voeren O_{11} O_{23} O_{32}
3. preferentielijst ~~2~~ 3 1, 1 3 2, ~~2~~ 1 3,
   mogelijk uit te voeren O_{11} O_{22} O_{32}
4. preferentielijst ~~2~~ 3 ~~1~~, 1 ~~3~~ 2, ~~2~~ 1 3,
   mogelijk uit te voeren O_{12} O_{22} O_{31}
5. preferentielijst ~~2~~ ~~3~~ ~~1~~, ~~1~~ ~~3~~ 2, ~~2~~ 1 3,
   mogelijk uit te voeren O_{13} O_{22} O_{33}

Uiteindelijk levert dat het volgende schema:



Figuur F.1: Het schema dat ontstaat bij de preference-list based representatie.

Appendix G. Priority rule-based representatie

Het JSSP wordt gegeven door:

job	routering operaties (bewerkingstijd)		
J1	M1 (3)	M2 (3)	M3 (3)
J2	M1 (2)	M3 (3)	M2 (4)
J3	M2 (3)	M1 (2)	M3 (1)

Tabel G.1: Het 3×3 JSSP.

Als de priority rule-based representatie gehandhaafd wordt en een individu overeenkomt met het volgenhet chromosoom

[0 2 1 4 6 5 7 3 0]

waarbij de corresponderende regels in tabel D.1 staan, dan kan het G&T algoritme gebruikt worden om het corresponderende schema op te stellen.

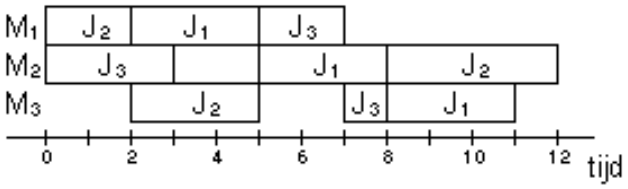
code	regel	omschrijving
0	SOT (shortest operation time)	een operatie met de kortste bewerkingstijd op de machine
1	LOT (longest operation time)	een operatie met de langste bewerkingstijd op de machine
2	SPT (shortest processing time)	een job met de kortste totale bewerkingstijd
3	LPT (longest processing time)	een job met de langste totale bewerkingstijd
4	SNRO (smallest number of remaining operations)	een operatie met het kleinste aantal volgende job operaties
5	LNRO (longest number of remaining operations)	een operatie met het grootste aantal volgende job operaties
6	LRPT (longest remaining proceeding time)	een operatie met de langste resterende job bewerkingstijd
7	SRPT (shortest remaining proceeding time)	een operatie met de kortste resterende job bewerkingstijd

Tabel G.2: Een aantal prioriteitsregels.

Hieronder zal stap voor stap besproken worden hoe zo'n schema opgesteld wordt. Als er meerdere machines gekozen kunnen worden, dan wordt deze random bepaald en de uiteindelijk geselecteerde operatie zal per iteratie onderstreept worden.

			prioriteitregel
stap 1	$S_1 = \{o_{11}, o_{21}, o_{32}\}$		
	$\varphi^*_1 = \min\{3, 2, 3\} = 2$	$k^* = 1$	
	$C_1 = \{o_{11}, \underline{o_{21}}\}$		0: SOT
stap 2	$S_2 = \{o_{11}, o_{23}, o_{32}\}$		
	$\varphi^*_2 = \min\{5, 5, 3\} = 3$	$k^* = 2$	
	$C_2 = \{\underline{o_{32}}\}$		2: SPT
stap 3	$S_3 = \{o_{11}, o_{23}, o_{31}\}$		
	$\varphi^*_3 = \min\{5, 5, 5\} = 5$	$k^* = 1$	
	$C_3 = \{\underline{o_{11}}, o_{31}\}$		1: LOT
stap 4	$S_4 = \{o_{21}, o_{23}, o_{31}\}$		
	$\varphi^*_4 = \min\{6, 5, 7\} = 5$	$k^* = 3$	
	$C_4 = \{\underline{o_{23}}\}$		4: SNRO
stap 5	$C_5 = \{o_{21}, o_{22}, o_{31}\}$		
	$\varphi^*_5 = \min\{8, 9, 7\} = 7$	$k^* = 1$	
	$C_5 = \{\underline{o_{31}}\}$		6: LRPT
stap 6	$S_6 = \{o_{12}, o_{22}, o_{33}\}$		
	$\varphi^*_6 = \min\{8, 9, 8\} = 8$	$k^* = 2$	
	$C_6 = \{\underline{o_{12}}, o_{22}\}$		5: LNRO
stap 7	$S_7 = \{o_{13}, o_{22}, o_{33}\}$		
	$\varphi^*_7 = \min\{11, 12, 8\} = 8$	$k^* = 3$	
	$C_7 = \{\underline{o_{33}}\}$		7: SRPT
stap 8	$S_8 = \{o_{13}, o_{22}\}$		
	$\varphi^*_8 = \min\{12, 12\} = 12$	$k^* = 3$	
	$C_8 = \{\underline{o_{13}}\}$		3: LPT
stap 9	$\underline{o_{22}}$		0: SOT

Uiteindelijk levert dat het volgende schema:



Figuur G.1: Het schema dat ontstaat bij de priority-rule based representatie.

Appendix H. Flexibele JSSP

Een flexibele JSSP houdt in dat de operaties op meerdere machines uitgevoerd kunnen worden (Mesghouni, Hammadi en Borni). Het probleem bestaat daarom ook uit twee vraagstukken: Ten eerste moet iedere operatie toegewezen worden aan machines en vervolgens moet een schema opgesteld worden waarbij een bepaalde doelfunctie geminimaliseerd wordt. Hieronder zal beschreven worden hoe een evolutionaire aanpak gebruikt kan worden om het flexibele JSSP aan te pakken.

H.1 Representatie

Voor het flexibele JSSP is gekozen voor een parallelle representatie. Dit houdt in dat een chromosoom gerepresenteerd wordt door een reeks van parallelle machines en iedere machine is een vector die bestaat uit de toegewezen operaties. Een operatie wordt weergegeven door drie termen (O_{ij}, J, t_{ijk})[9]. Dit zijn respectievelijk het operatienummer, de job en het tijdstip waarop de operatie uitgevoerd moet worden.
Stel dat er bijvoorbeeld 3 jobs uitgevoerd moeten worden op 3 machines en de volgorde van de operaties is als volgt:

J1: O_{11}, O_{21}, O_{31}
J2: O_{12}, O_{22}, O_{32}
J3: O_{13}, O_{23}

Hierin is O_{11} de eerste operatie van job 1 en O_{21} is de tweede operatie van job 1 etc. Verder geldt dat er volledige flexibiliteit is (dus alle operaties mogen op alle machines uitgevoerd worden) met de volgende bewerkingstijden p_{ijk}:

	M1	M2	M3
O_{11}	1	9	3
O_{21}	3	5	2
O_{31}	6	7	1
O_{12}	1	4	5
O_{22}	2	8	4
O_{32}	9	5	1
O_{13}	1	5	9
O_{23}	5	9	2

Een voorbeeld chromosoom kan dan de volgende zijn:

M1	(1,1,0)	(1,2,1)	(2,2,2)
M2	(2,1,1)	(1,3,6)	(3,1,1)
M3	(3,2,4)	(2,3,1)	

H.2 Recombinatie

Het crossover mechanisme wordt als volgt weergegeven:

```
Selecteer willekeurig O1, O2 en machine k
Kopieer de operaties op machine k van O1 naar K1, m = 1
Herhaal zolang m < M
        als m ≠ k kopieer de nog niet toegekende operaties van machine m van O2 naar K1,
        m = m + 1
eind
als een operatie ontbreekt in K1, controleer dan machine k van O2 en kopieer de
ontbrekende operaties
```

Algoritme H.1: Een crossover operatie voor het flexibele JSSP.

[9] i =operatie, j = job en k = machine

Deze recombinatie operator kan ervoor zorgen dat een kind niet uitvoerbaar is door een zogenaamde *deadlock*, dit is het geval als operaties op elkaar aan het wachten zijn. Neem bijvoorbeeld de volgende situatie:

J1: O_{11}, O_{21}
J2: O_{12}, O_{22}

Als het kind weergegeven wordt, zoals deze hieronder staat, dan is er sprake van een deadlock.

M1	(2,2,?)	(1,1,?)
M2	(2,1,?)	(1,2,?)

Machine 1 wil operatie 2 van job 2 uitvoeren, maar kan dat alleen doen als operatie 1 van job 2 uitgevoerd is. Deze operatie wordt pas gedaan als operatie 2 van job 1 klaar is. Maar deze operatie mag pas plaatsvinden als operatie 1 van dezelfde job uitgevoerd is. Dus beide machines wachten op elkaar.
Er zijn twee mogelijkheden om dit probleem aan te pakken:

- pas de genetische operator aan, zodat er altijd toegestane oplossingen geproduceerd worden of
- pas de representatie aan, zodat iedere representatie overeenkomt met een toegestane oplossing.

Bij de laatst genoemde mogelijkheid moet de EA in z'n geheel opnieuw bedacht worden en vervolgens is het dan nog weer de vraag of de nieuwe representatie gebruik kan maken van de genoemde recombinatie techniek. Vandaar dat de eerste mogelijkheid behandeld wordt. Het ontstaan van deze deadlock vindt zijn oorsprong in de laatste stap van algoritme F.1. Vandaar dat deze aangepast moet worden in:

als operatie O_{ij} ontbreekt in K1, bepaal dan de locatie op machine k via de volgende regels:

- als i = 1 stop O_{ij} aan het begin van machine k
- als i = x_j stop O_{ij} aan het eind van machine k (x_j is de laatste operatie van job j)
- als m $\in$ {2, 3, ..., x_{j-1}} zoek de positie van $O_{i-1,j}$ en de positie van $O_{i+1,j}$ van K1 en stop O_{ij} tussen deze posities op machine k

Aanschouw nogmaals hetzelfde voorbeeld van paragraaf F.1. Stel dat machine 3 gekozen is, samen met de volgende twee ouders:

O1

M1	(1,1,0)	(3,1,6)	(1,3,12)
M2	(2,1,1)	(1,2,6)	
M3	(2,2,10)	(3,2,14)	(2,3,15)

O2

M1	(1,1,0)	(1,2,1)	(2,2,2)
M2	(2,1,1)	(1,3,6)	(3,1,11)
M3	(3,2,4)	(2,3,11)	

Vervolgens worden de operaties die toegewezen zijn aan machine 3 van O1 (respectievelijk O2) gekopieerd naar machine 3 van K1 (respectievelijk K2)

K1

M1			
M2			
M3	(2,2,?)	(3,2,?)	(2,3,?)

K2

M1			
M2			
M3	(3,2,?)	(2,3,?)	

De nog niet toegewezen operaties van M1 en M2 worden gekopieerd van O2 (respectievelijk O1) naar K1 (respectievelijk K2)

K1

M1	(1,1,?)	(1,2,?)	
M2	(2,1,?)	(1,3,?)	(3,1,?)
M3	(2,2,?)	(3,2,?)	(2,3,?)

K2

M1	(1,1,?)	(3,1,?)	(1,3,?)
M2	(2,1,?)	(1,2,?)	
M3	(3,2,?)	(2,3,?)	

Bij K2 ontbreekt operatie 2 van job 2. Deze operatie is niet de eerste noch de laatste operatie van job 2, dus de positie van O_{12} in K2 en van O_{32} in K2 moet bepaald worden. Dit is op

respectievelijke positie 2 en positie 1. De operatie O_{22} moet voor O_{32} komen op machine 3. Nu kunnen de tijdstippen bepaald worden wanneer de operaties uitgevoerd kunnen worden.

K1				
	M1	(1,1,0)	(1,2,1)	
	M2	(2,1,1)	(1,3,6)	(3,1,11)
	M3	(2,2,2)	(3,2,6)	(2,3,11)

K2				
	M1	(1,1,0)	(3,1,6)	(1,3,12)
	M2	(2,1,4)	(1,2,1)	
	M3	(2,2,10)	(3,2,14)	(2,3,15)

H.3 Mutatie

Er worden twee mutatie operatoren genoemd: assigned mutatie en swap mutatie. Beide zullen behandeld worden, te beginnen met assigned mutatie.

assigned mutatie

stap 1. Selecteer willekeurig een chromosoom en een operatie.
stap 2. Ken de geselecteerde operatie toe aan een andere machine op dezelfde positie als dit mogelijk is (gelet op de volgorde relaties en machine voorwaarden)

Neem nogmaals het 3×3 JSSP voorbeeld van hierboven en stel dat het volgenhet chromosoom geselecteerd is samen met operatie 3 van job 1. Deze operatie wordt vervolgens toegekend aan machine 2.

M1	(1,1,0)	(1,2,1)	(2,2,2)		M1	(1,1,0)	(1,2,1)	(2,2,2)
M2	(2,1,1)	(1,3,6)		⇒	M2	(2,1,1)	(1,3,6)	(3,1,13)
M3	(3,2,4)	(2,3,11)	(3,1,13)		M3	(3,2,4)	(2,3,11)	

Tabel H.1: Bij de assigned mutatie wordt een operatie elders geplaatst.

swap mutatie

stap 1. Selecteer willekeurig een chromosoom, een positie, een richting en twee machines.
stap 2. als richting = 0; verwissel de operaties op de twee machines die links staan van de geselecteerde positie
als richting = 1; verwissel de operaties op de twee machines die rechts staan van de geselecteerde positie

Neem nogmaals het 3×3 JSSP voorbeeld van hierboven en stel dat het volgenhet chromosoom geselecteerd is samen met positie 2, richting 0 en de machines 2 en 3. Vervolgens worden de operaties (2,2,2) en (2,1,1) omgedraaid, aangezien deze op positie 1 staan (links van positie 2).

M1	(1,1,0)	(1,2,1)			M1	(1,1,0)	(1,2,1)	
M2	(2,2,2)	(1,3,10)	(3,1,15)	⇒	M2	(2,1,1)	(1,3,6)	(3,1,11)
M3	(2,1,1)	(3,2,10)	(2,3,15)		M3	(2,2,2)	(3,2,6)	(2,3,11)

Op het moment dat de richting 1 was geweest, zou het volgenhet chromosoom geproduceerd zijn:

M1	(1,1,0)	(1,2,1)			M1	(1,1,0)	(1,2,1)	
M2	(2,2,2)	(1,3,10)	(3,1,15)	⇒	M2	(2,2,2)	(1,3,10)	(2,3,15)
M3	(2,1,1)	(3,2,10)	(2,3,15)		M3	(2,1,1)	(3,2,10)	(3,1,11)

Appendix I. Open Shop Scheduling Probleem (OSSP)

Het OSSP is hetzelfde als het JSSP, met de uitzondering dat er vooraf geen volgorde relaties bekend zijn tussen de operaties van een job (Fang, Ross Corne 1993). Een veelgebruikte vereenvoudiging van het OSSP is dat eerst iedere operatie gekoppeld wordt aan één van de machines waarop deze uitgevoerd kan worden. Dit levert echter niet de optimale oplossing.

Een mogelijke representatie is dat het chromosoom bestaat uit $n \times m$ genen en iedere gen kan de waarde $i \in \{1, 2, ..., n\}$ aannemen. De waarde stelt dan de i^{de} niet volledig uitgevoerde job voor. Er moet dan echter nog wel een operatie gekozen worden. Dit kan op drie verschillende manieren (Fang, Ross en Corne 1994):

a) direct: verdubbel de lengte van chromosoom waarin de oneven positie vertelt welke niet volledig uitgevoerde job gekozen wordt en de even positie voor de (nog) niet uitgevoerde operatie van die job staat,
b) vaste heuristiek: een operatie wordt bepaald aan de hand van een heuristiek X en
c) evoluerende heuristiek: verdubbel de lengte van chromosoom waarin de oneven positie vertelt welke niet volledig uitgevoerde job gekozen wordt en de even positie voor de toe te passen heuristiek staat.

Een aantal heuristieken voren het open shop probleem komen overeen met het job shop probleem, zoals tabel G.1 laat zien:

heuristiek	beschrijving
LPT	de operatie met de grootste bewerkingstijd
SPT	de operatie met de kortste bewerkingstijd
EF-LPT	pas LPT toe op de operaties die het eerst ingepland kunnen worden
EF-SPT	pas SPT toe op de operaties die het eerst ingepland kunnen worden
EF-BTR	kies willekeurig een van de operaties die het eerst ingepland kunnen worden
SG-LPT	pas LPT toe op de operaties die ingepland kunnen worden tussen reeds ingeplande operaties (als leeg dan LPT)
LRG	de operatie die de meest ruimte tussen de reeds ingeplande operaties open laat
SRG	de operatie die de meest ruimte tussen de reeds ingeplande operaties opvult

Tabel I.1: Een aantal heuristieken voor het open shop scheduling probleem.