Inhoudstabel

Hoofdstuk VI: GenGA: Een genetisch algoritme voor code word design

Hoofdstuk VII: Het experiment (2002-2003)

Hoofdstuk VIII: Algemene conclusie

Lijst van figuren

Voorwoord

Graag had ik hierbij enkele mensen willen bedanken voor hun inzicht en steun gedurende de verwezenlijking van deze thesis. Eerst en vooral wil ik mijn promotor prof. Jan Van Den Bussche bedanken voor zijn inzet en tijd. Zijn inzicht en kennis op het vlak van informatica alsook DNA computing hebben mij enorm geholpen. Elke samenkomst en presentatie was leerrijk en interessant. Ik wil ook mijn begeleider Dr. Karl Tuyls bedanken voor zijn kennis en hulp bij het ontwikkelen van het genetische algoritme.
Vervolgens wil ik mijn vriendin Lore bedanken die doorheen de jaren een echte rots in de branding is geweest. Zonder haar inzet zou deze thesis lang niet zijn wat ze nu is. Ook mijn ouders en mijn zus Inge ben ik heel veel dank verschuldigd voor hun steun en geduld. Ondanks alles zijn ze steeds in mij blijven geloven.
Verder wil ik ook mijn collega's, medestudenten en kotgenoten bedanken voor hun interessante en plezante momenten die we samen beleefd hebben.

Samenvatting

Toen in 1994 de paper ‘Molecular computation of solutions to combinatorial problems’ van professor Adleman verscheen, opende dit een heel nieuw onderzoeksveld binnen de informatica: DNA computing. Hierbij zou met behulp van DNA en verscheidene laboratorium technieken combinatorische problemen opgelost kunnen worden. Er werd ook al snel duidelijk dat DNA verschillende specifieke voordelen biedt ten op zichte van de hedendaagse computers.
In het afgelopen decennium heeft er veel onderzoek plaatsgevonden in dit domein en is men reeds tot enkele belangrijke vaststellingen gekomen. Ook de nadelen en beperkingen van werken op moleculair niveau zijn grondig besproken waarbij men kan concluderen dat DNA computing nog een lange weg te gaan heeft alvorens ze effectief kan toegepast worden. Het domein waarin DNA computing kan toegepast worden hoort bij het meest recente onderzoek. Dankzij een abstractie van DNA computing zou het dan mogelijk zijn een vergelijkende studie te maken met andere toepassingen.
Een apart onderdeel van DNA computing is het opstellen van de structuur van het DNA zelf. De keuze van DNA als representatie voor allerlei data blijkt niet altijd evident en vooral probleem afhankelijk te zijn. Een algemene beschrijving voor het ontwerpen van DNA strengen wordt beschreven in het domein van de ‘Code Word Design’. Voor elk probleem zullen er een reeks beperkingen gelegd kunnen worden op de structuur van het DNA. Er bestaan verschillende beperkingen waaraan een verzameling DNA strengen moeten voldoen. Ook elke streng op zich moet ‘structure free’ zijn.
De zoektocht naar geschikte representaties is moeilijk. De verzameling mogelijke strengen DNA waaruit men kan kiezen is immens groot zodat een goed zoekalgoritme geen overbodige luxe is.
Het genetisch algoritme biedt hiervoor een geschikte oplossing. Dit heuristisch algoritme, gebaseerd op de natuurlijke evolutie, kent haar sterkte vooral in het zoeken naar oplossingen in een reusachtige, al dan niet oneindige verzameling elementen. Het werd in de jaren ‘60 ontworpen door John Holland en wordt steeds meer en meer gebruikt in verschillende domeinen waarbij vooral artificiële intelligentie in het oog springt. De bedoeling van het algoritme bestaat erin vanuit een generatie mogelijke oplossingen te evolueren naar een superieure generatie dat een globaal optimum benadert voor het gegeven probleem. De elementen die instaan voor de volgende generatie worden gekozen op basis van een natuurlijke selectie methode. Vervolgens zullen gekende natuurlijke operaties gebruikt worden om de evolutie uit te voeren. Uiteindelijk zal elk element een score of ‘fitness’ krijgen aan de hand van een welgekozen fitness functie. Via parameters wordt probabiliteit aan het algoritme toegevoegd om de evolutie theorie concreter te benaderen.
De basis van dergelijk algoritme is eenvoudig maar de implementatie ervan is eerder complex en probleem afhankelijk. Een correcte implementatie zal veel afhangen van de gekozen methoden en parameters hoewel het vaak onmogelijk is op voorhand te bepalen welke methoden of parameters tot het optimale resultaat zullen leiden.

Voor dit proefschrift werd een dergelijk genetisch algoritme ontwikkeld dat het ‘code word design’ voor het 3-bit majority probleem oplost. Gedurende de implementatie werd al gauw duidelijk dat het ontwerpen van een stabiel en correct genetisch algoritme niet vanzelfsprekend is.
Het programma zal uiteindelijk een degelijke oplossing geven voor het ‘code word design’ van het 3-bit majority probleem. Het resultaat, bestaande uit strengen DNA, dient als input voor een algoritme dat het 3-bit majority probleem oplost met DNA computing. Dit algoritme stelt een concrete oplossing voor vertrekkende vanuit een meer algemene oplossing dat het n-bit majority probleem oplost via DNA computing.
Met het resultaat is het dan mogelijk het algoritme met DNA computing in de praktijk uit te voeren. Enkele jaren geleden heeft men aan het Limburgs Universitair Centrum getracht dit uit te voeren maar kende toen geen succes. Een mogelijke reden hiervoor zou te wijten zijn aan het toen ontwikkelde ‘code word design’.

Hoofdstuk I: Inleiding

1.1 Doelstelling

Bij de aanvang van dit proefwerk werden er twee hoofddoelen vooropgesteld. Na een inleiding in hoofdstuk 1 over het begrip ‘DNA’ en een kort overzicht van enkele belangrijke biotechnologische technieken zal ten eerste een overzicht gegeven worden van de evolutie van DNA computing en worden de verwezenlijkingen van DNA computing kritisch benaderd (hoofdstuk 2).

Ten tweede zal, in hoofdstuk 4, het n-bit majority probleem met DNA computing moeten worden opgelost. Hierbij zal er veel aandacht geschonken worden aan het ontwerp van de input voor dit probleem. Deze procedure kan omkaderd worden in het domein van het code word design. Code word design zal besproken worden in hoofdstuk 3, waar een algemeen overzicht gegeven wordt.

Voor het ontwikkelen van dergelijke code words voor dit specifieke probleem wordt er beroep gedaan op evolutionair programmeren. Het genetisch algoritme treedt dan als evolutionair algoritme op de voorgrond en zal eerst grondig besproken worden in hoofdstuk 5 om zo meer inzicht te krijgen in de mogelijkheden en implementatie ervan.

Voor het code word design van het 3-bit majority probleem zal effectief een genetisch algoritme ontwikkeld en geïmplementeerd worden. De constructie, implementatie, resultaten, voor- en nadelen zullen vervolgens besproken worden in hoofdstuk 6.

Uiteindelijk is het de bedoeling om in de toekomst met de resultaten van het ontwikkelde programma, het 3-bit majority probleem in de praktijk uit te voeren. Dit werd reeds uitgevoerd aan het Limburgs Universitair Centrum maar kende toen niet het gehoopte succes. In hoofdstuk 7 zal er kort maar bondig gekeken worden naar de mogelijke oorzaken van het falen, om zo de kans op toekomstig succes te kunnen vergroten.

1.2 DNA: Basisbegrippen en biotechnologische technieken

Aangezien een goede kennis van het begrip ‘DNA’ absoluut noodzakelijk is voor de hele DNA computing studie, zal eerst een korte inleiding gegeven worden over ‘DNA’.

1.2.1 DNA: de bits en de bytes van het leven

Levende organismen zoals planten, dieren of mensen zijn opgebouwd uit een massa cellen [1]. De cel is de kleinste levende eenheid die zelfstandig kan groeien en zich kan vermenigvuldigen. Micro-organismen zoals bacteriën of gisten bestaan uit één cel. Vele meercellige organismen, zoals de mens, zijn ontstaan uit slechts één cel bij de bevruchting. Een volwassen mens bestaat uit 10 000 000 000 000 of met andere woorden 10 biljoen cellen. Bij bijna alle meercellige organismen krijgen groepen van cellen specifieke functies en zijn ze georganiseerd in weefsels (zoals spieren, huid,…) en organen (zoals het hart, een nier, …). Binnenin de organen, weefsels en cellen zorgen eiwitten voor het goede verloop van de verschillende biologische processen.
Elke cel bevat een kern met daarin een aantal chromosomen. Deze korte staafjes bevatten het zeer compact opgerolde DNA (in elke cel zit bijna twee meter DNA!). Het DNA (desoxyribo nucleïnezuur (= acid)) is de informatiedrager van de cel en controleert welke eiwitten wanneer worden aangemaakt. Het DNA bepaalt dus in grote mate hoe een plant of een mens er uit ziet en heeft een belangrijke invloed op onze andere kenmerken zoals gedrag en intelligentie. Deze informatie wordt doorgeven van ouders op hun kinderen: het DNA is de drager van de erfelijke informatie. De erfelijke informatie ligt opgeslagen in het DNA onder de vorm van een code (cfr. infra). Elk levend organisme bevat DNA, dat gemakkelijk kan worden geïsoleerd.

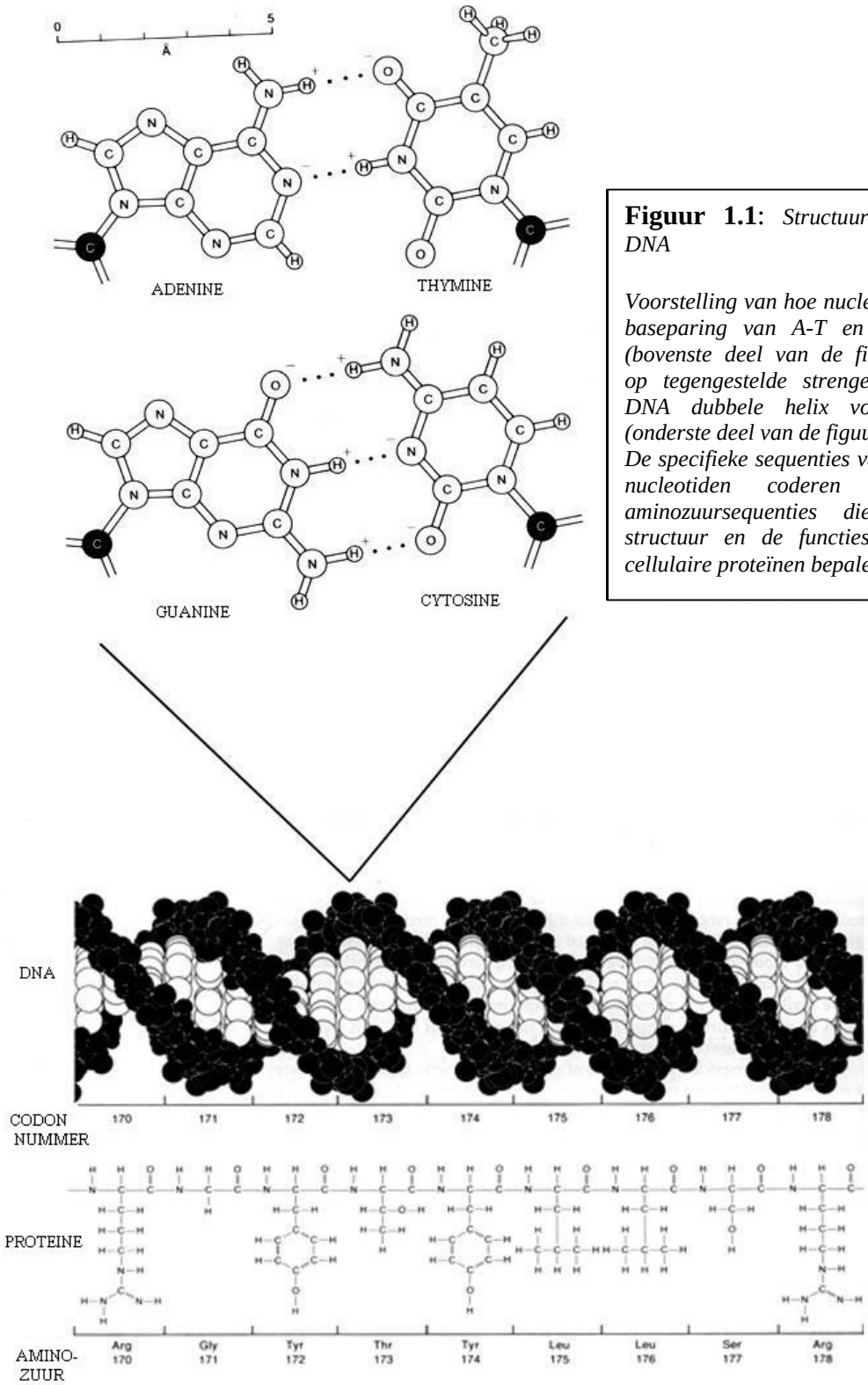
1.2.2 Structuur van DNA

Nucleïnezuren zijn lange moleculen, opgebouwd uit betrekkelijk eenvoudige bouwstenen, nucleotiden genoemd [1]. Elk nucleotide bestaat uit drie stukjes: een fosfaatmolecule, een suikermolecule en een base. In de nucleïnezuren van de celkern komt als suiker deoxyribose voor. Vandaar de naam deoxyribonucleïnezuur, of deoxyribo-nucleic acid, of DNA. Het belangrijkste onderdeel van een nucleotide is de base. Er komen vier basen voor in DNA: adenine, cytosine, guanine en thymine (meestal afgekort tot A, C, G en T) (zie figuur 1.1 [2]).
De ruimtelijke structuur van het DNA molecule werd als eerste voorgesteld in 1953 door James Watson en Francis Crick [1, 3]. Het belangrijkste kenmerk van die structuur is het feit dat de nucleotide-basen twee aan twee aan elkaar koppelen als stukjes van een puzzel. Die eigenschap, die ze ‘complementaire basenparing’ noemden, suggereerde dadelijk een mechanisme voor de verdubbeling van DNA.

DNA, aldus Watson en Crick, komt voor als een dubbele keten. De ruggengraat van elke keten is een streng opgebouwd door een aaneenschakeling van {deoxyribose-fosfaat} eenheden. Van deze …{suiker-fosfaat} – {suiker-fosfaat}… keten draagt de suiker telkens één van de mogelijk basen A, C, G en T. De twee strengen zijn helixvormig rond elkaar gedraaid, een beetje zoals een wenteltrap. Ze raken elkaar met hun basen, en daarbij liggen steeds dezelfde basen tegen elkaar, omdat ze zo perfect in elkaar passen: tegenover een A op de ene streng ligt altijd een T op de andere, tegenover een C ligt altijd een G. Bij verdubbeling van het DNA gaan de twee strengen uit elkaar en wordt tegenover elke streng een nieuwe gebouwd, waarbij de basenvolgorde vastligt door het puzzelprincipe van de complementaire basenparing: ook nu zal tegenover elke A enkel een T passen en zo verder. Daardoor ontstaan twee dubbelstrengen die allebei identiek zijn aan de oorspronkelijke.

Het model van Watson en Crick is een mijlpaal in de geschiedenis van de wetenschap. Hoewel het maar een hypothese was, leek het idee zo aantrekkelijk dat de werkelijkheid gewoon niet anders kon zijn. Het model werd later bevestigd mede door het geduldige analytische werk van Maurice Wilkins, en tot grote voldoening van alle betrokkenen werd de Nobelprijs in 1962 uitgereikt aan Watson, Crick en Wilkins samen. Niet alleen het briljante idee, maar ook het volgehouden experimentele werk kreeg op die manier de gepaste erkenning [1, 3].

De opheldering van de structuur van DNA was het startpunt voor geheel nieuwe denkpistes. Al gauw werd ontdekt dat de genetische informatie die opgeslagen ligt in het DNA vanuit de kern naar de plaats van eiwitsynthese wordt gebracht [3, 4]. Dit gebeurt als volgt: het DNA in de kern wordt gekopieerd (transcriptie). De informatie op deze kopie, mRNA (boodschapper of messenger RNA, ribonucleïnezuur (= acid)), wordt gelezen door een kleine molecule, het ribosoom (een klein eiwitproducerend fabriekje). De informatie is gecodeerd. Om de boodschap goed te begrijpen, leest het ribosoom drie basen tegelijk (dit noemt men een codon). Elk codon stemt overeen met één bouwsteen van een eiwit: een aminozuur. Er bestaan 20 verschillende aminozuren en de volgorde van de aminozuren bepaalt het eiwit. Voor elke drie basen (of codon) die het ribosoom leest, wordt één aminozuur toegevoegd. Wanneer alle aminozuren aan de groeiende eiwitketen zijn gehangen, is het eiwit gevormd en kan het zijn functie uitvoeren [5].



Figuur 1.1: *Structuur van DNA*

Voorstelling van hoe nucleotide baseparing van A-T en G-C (bovenste deel van de figuur) op tegengestelde strengen de DNA dubbele helix vormen (onderste deel van de figuur). De specifieke sequenties van de nucleotiden coderen voor aminozuursequenties die de structuur en de functies van cellulaire proteïnen bepalen.

1.2.3 Moleculair biologische technieken

1.2.3.1 Smelttemperatuur

Eén van de eigenschappen van DNA waar moleculair biologische technieken veelvuldig gebruik van maken is dat de vorm van een dubbele helix bij hoge temperaturen "uitsmelt" in twee enkele strengen (denaturatie). Afkoeling daarna zorgt vervolgens dat de twee enkele strengen weer in elkaar "ritsen" (annealing) om weer een dubbele helix te vormen. De temperatuur waarbij de dubbele helix uitsmelt hangt af van de samenstelling van het DNA en van eventueel aanwezige zouten of andere stoffen. G/C rijk DNA heeft een hogere smelttemperatuur ('melting temperature' of smelttemperatuur) dan A/T rijk DNA, en de aanwezigheid van formamide bijvoorbeeld verlaagt de DNA smelttemperatuur (de smelttemperatuur is die temperatuur waarop 50% van de DNA sequenties gedenatureerd zijn (enkelstrengig zijn)).
Er bestaat een lineair verband tussen de smelttemperatuur en het %G+C van een dubbele helix: $y = bx + a$, met als x het percentage G+C en als y de T_m, de helling b hangt af van het soort duplex en de constante a hangt ondermeer af van de zoutcondities.

Er zijn in de literatuur een aantal formules beschreven voor de smelttemperatuur T_m [6]:

- T_m zoals initiëel beschreven door Marmur: (voor G+C tussen 30 en 75%)

  T_m = 69.3 + 0.41 [%GC] in 0.15 M NaCl + 0.015 M Na-citraat pH 7.0
  = 53.9 + 0.41 [%GC] in 0.015 M NaCl + 0.015 M Na-citraat pH 7.0

  (helling blijft onveranderd maar de waarde van de constante a daalt bij lagere zoutconcentratie)

- De contributie van de ionische concentratie tot de waarde van de constante a werd naderhand aangepast door Wetmur:

  <u>eerst tot:</u>

  $T_m = 81.5 + 16.6 \log[M^+] + 0.41$ [%GC] als $[M^+] < 0.4$

  <u>en dan tot:</u>

  $T_m = 81.5 + 16.6 \log\{[M^+]/(1+0.7[M^+])\} + 0.41$ [%GC] als $[M^+] < 0.4$

  voor $[M^+]$ tot 1M NaCl

- Voor korte strengen begint de lengte de T_m te beïnvloeden: een factor -500/n moet toegevoegd worden:

  Aldus: voor DNA-DNA:

$$T_m\,(C°) = 81.5 + 16.6\,\log\{[M^+]/(1+0.7[M^+])\} + 0.41\,[\%GC] - F - 500/n - P$$

met F = 0.63 x % formamide (formamide veroorzaakt een vermindering van 0.63 °C per procent) en P = % mismatch (niet alle posities in de DNA strengen zijn perfect complementair tegenover elkaar (mismatch-posities genoemd)); n is de grootte van het dubbelstrengige segment.
(De hellingen zijn verschillend, evenals de invloed van formamide en andere factoren dan de ionische condities.)

- Met oligonucleotiden zijn deze formules niet meer bruikbaar: T_m wordt berekend uit thermodynamische waarden (hier niet getoond).

  Lange tijd werd de Regel van Wallace vaak gebruikt:

  $$T_m = (2 \times n_{AT}) + (4 \times n_{GC}) \qquad n_{AT} = \#A's + \#T's \quad n_{GC} = \#G's + \#C's$$

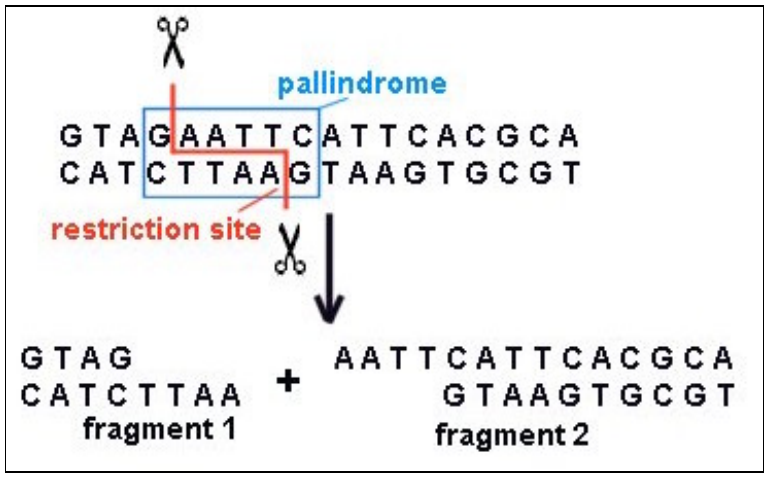
  maar deze regel is zeer sterk gelimiteerd tot lengtes rond 17, en wordt al te vaak verkeerdelijk gebruikt (en is bovendien weinig betrouwbaar).

1.2.3.2 Enzymen

Naast bovenstaande fysieke eigenschappen van DNA maken moleculair biologische technieken gebruik van een breed scala van enzymen [1]. Het meest gebruikte enzym, *DNA polymerase*, maakt een nieuwe streng DNA waarbij het een bestaande streng als 'mal' of 'template' gebruikt. Het bouwt een C in tegenover een G en een T tegenover een A. Dit enzym wordt veelvuldig gebruikt in PCR's (Polymerase Chain Reactions, cfr. infra). Omdat de temperatuur tijdens een PCR vaak hoog oploopt wordt een hittebestendige DNA polymerase gebruikt dat afkomstig is van Thermus aquaticus, een bacterie die in heetwater bronnen leeft en dus bij hoge temperaturen DNA kan maken, Taq polymerase genoemd (cfr. infra). Naast de opbouwende werking van DNA polymerases hebben sommigen de eigenschap om DNA af te breken.
Een ander veelgebruikt enzym is *reverse transcriptase*, een enzym dat DNA genereert van een RNA 'mal' (template). Omdat dit proces in de natuur slechts gebruikt wordt door bepaalde virussen (retrovirussen) wordt dit enzym geïsoleerd uit cellen die met dergelijke virussen geïnfecteerd zijn.
Restrictie-enzymen (endonucleases) zijn een andere groep van enzymen van belang voor de moleculaire biologie. Deze enzymen worden geïsoleerd uit bepaalde bacteriën waar ze vreemd DNA op specifieke plaatsen knippen (bv. Hind III knipt de sequentie AAGCTT, EcoRI knipt de sequentie GAATTC (zie figuur 1.2),…). Bacteriën gebruiken restrictie-enzymen als afweer tegen bacterievirussen, die, als ze de bacterie binnenkomen, in stukjes worden geknipt. Ongeveer 900 verschillend restrictie-enzymen (moleculaire scharen) zijn momenteel beschikbaar.

Verder zijn er nog verschillende andere enzymen, waaronder ligasen, die twee DNA fragmenten aan elkaar kunnen lassen.

Biochemici hebben geleerd om al die enzymen af te zonderen. De enzymen kunnen dan in een proefbuis dezelfde taken uitvoeren, en zo vormen zij de gereedschappen die de moleculaire bioloog gebruikt om DNA moleculen te manipuleren.
Voor veel moleculair biologische toepassingen is een minimale hoeveelheid DNA nodig (het DNA moet bijvoorbeeld zichtbaar zijn op gel, of de basenpaar volgorde ervan moet worden bepaald). Meestal is de aanwezige hoeveelheid DNA beperkt, en moet het DNA door middel van PCR vermeerderd worden (DNA amplificatie).



Figuur 1.2: *Werking van het EcoRI enzym.*

1.2.3.3 De technieken

Het is onmogelijk om een volledig overzicht te geven van alle moleculair biologische technieken die momenteel worden aangewend. Enkele zeer belangrijke technologieën, die eveneens belangrijk zijn voor DNA computing, worden hieronder kort besproken. Voor een uitvoerige bespreking van deze technieken wordt verwezen naar de thesis: 'DNA Computing in Praktijk', door Ward Conings en Kurt Vanbrabant [27].

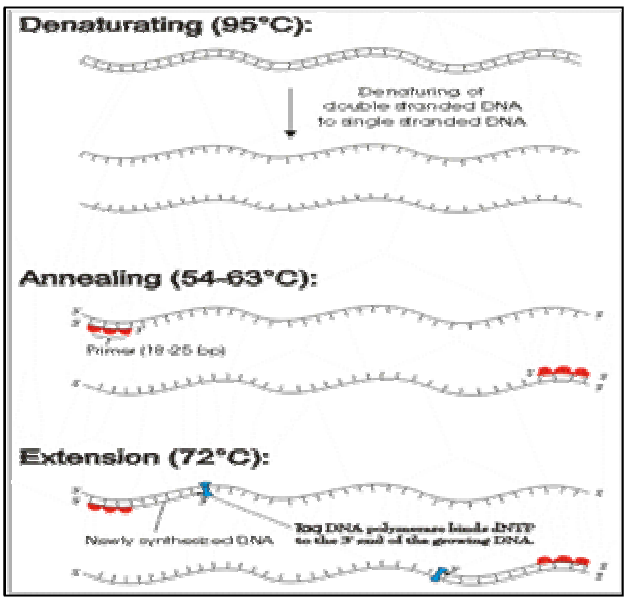
1) Polymerase Chain Reaction of PCR

Een recente ontwikkelde techniek die ondertussen al onmisbaar is voor de biotechnologie is PCR of 'Polymerase Chain Reaction' [1, 7]. In 1993 werd deze vinding met de Nobelprijs bekroond. De PCR-techniek laat toe om *in vitro* in enkele uren tijd stukjes DNA (DNA-fragmenten) exponentieel te vermeerderen. Hiervoor is weinig materiaal (template) nodig (1 cel, 1-5 moleculen) maar zijn specifieke reactiecondities vereist evenals een thermostabiel DNA polymerase en specifieke primers (oligonucleotiden) die, gebaseerd op kennis van de DNA sequentie, de regio afbakenen voor het klonen.

Het uitvoeringsprincipe is eenvoudig en imiteert de natuurlijke DNA-duplicatie of –replicatie en bestaat uit drie stappen (figuur 1.3).
Stap 1 (denaturatie): De dubbelstrengige DNA-helix wordt eerst verwarmd tot 94°C, waardoor de streng openbreekt tot twee complementaire enkelstrengige DNA moleculen.
Stap 2 (annealing of hybridisatie): Na het scheiden van de twee DNA strengen wordt het DNA gekoeld (tot 40 à 60°C) in aanwezigheid van een grote overmaat van twee specifieke oligonucleotide primers. Deze temperatuur bevordert de hybridisatie (= het samenvoegen van twee complementaire DNA strengen tot een dubbele helix) van deze primers aan de complementaire sequenties in de twee opengebroken DNA strengen. Primers bevatten namelijk sequenties die complementair zijn aan de begin- en eindsequenties van het te repliceren DNA. Voor de primers gebruikt bij PCR gelden enkele vuistregels:

- 18-25 nucleotiden
- 40-60% G+C
- T_m van primers moet gebalanceerd zijn
- Complementariteit aan 3'uiteinden dient vermeden (primer dimeer synthese)
- Geen interne palindromische sequenties (hairpin vorming)
- Liefst geen opeenvolging van identieke nucleotiden
- Liefst een G of C aan het 3'uiteinde (maar slechts één G of C in de vijf laatste nucleotiden)
- Niet-hybridiserende nucleotiden worden aan het 5'uiteinde geplaatst en bevatten drie extra nucleotiden voor een efficiënte knipping met enzymen.

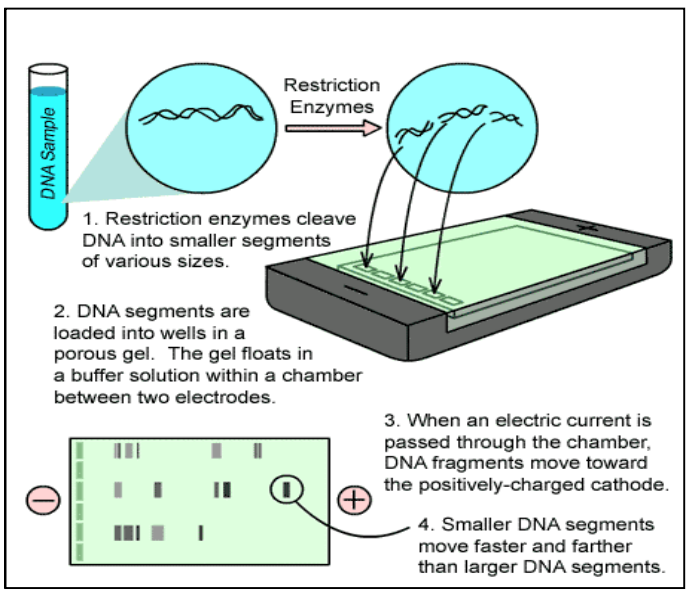
Stap 3 (extentie of elongatie): Vertrekkende van deze primers zal het DNA-polymerase enzym het complementaire deel van de twee strengen opbouwen met behulp van de toegevoegde bouwstenen (deoxyribonucleoside trifosfaten of dNTP's). Zoals hoger vermeld, wordt meestal een hittebestendig DNA-polymerase gebruikt dat ook bij hoge temperaturen DNA kan maken (Taq polymerase).
Er zijn dan twee dubbelstrengige DNA strengen ontstaan uit één dubbelstrengige streng. Deze twee strengen geven na een volgende cyclus vier strengen, na nog een cyclus acht strengen,… Zo wordt het DNA exponentieel verder vermenigvuldigd.



Figuur 1.3: *De verschillende stappen bij PCR, namelijk denaturatie, annealing (of hybridisatie) en extentie.*

2) Gelelektroforese

Nadat een lange DNA molecule in kleinere stukjes is geknipt met behulp van restrictie-enzymen (zoals gezegd knippen deze enzymen het DNA volgens een bepaald patroon), kunnen de verschillende DNA fragmenten van elkaar gescheiden worden [7, 8]. Via de gelelektroforese-techniek kunnen de verschillende DNA stukjes volgens lengte van elkaar gescheiden worden in een agarose (of polyacrylamide) gel, die een microscopisch netwerk van poriën bevatten (zie figuur 1.4). Agarose gel is een gel gemaakt uit zeewier en wordt eerst gekookt en vervolgens nog warm in een vorm gegoten. Aan één kant van de gel wordt een kam in de warme gel geplaatst. Na het opstijven van de gel wordt de kam verwijderd en blijven er gaatjes over waarin het DNA staal wordt geplaatst (gepipetteerd). Aan beide kanten van de gel bevinden zich elektroden die een constante stroom door de gel sturen. De buffervloeistof zorgt voor een goede geleiding van de stroom door de gel.
De DNA-elektroforese-techniek maakt het mogelijk om de verschillende stukjes DNA zichtbaar te maken. De techniek trekt de stukjes DNA van verschillende lengte uit elkaar en toont dus de verschillende stukjes aanwezig in één staal. Dit komt omdat DNA negatief geladen is door de aanwezigheid van fosfaatgroepen op de DNA keten. Zodra de stroom wordt ingeschakeld, verplaatsen de DNA stukjes zich dan ook naar de positieve elektrode. Kleine stukjes verplaatsen zich snel doorheen de gel (doordat ze klein zijn, worden ze bijna niet tegengehouden door de gel). Grote stukken DNA blijven gemakkelijker hangen en zijn dus veel trager. Op deze manier trekt men dus de stukjes DNA uit elkaar en kan men de verschillende stukjes DNA van elkaar scheiden. DNA stukjes met dezelfde grootte blijven bij elkaar.
De DNA banden op een agarose of polyacrylamide gel zijn onzichtbaar tenzij men het DNA labelled of kleurt. Een gevoelige methode om DNA zichtbaar te maken bestaat erin het DNA bloot te stellen aan een stof die fluoresceert onder ultraviolet licht wanneer het aan DNA gebonden is. Figuur 1.4 geeft een overzicht van de verschillende stappen, uitgevoerd bij de gelelektroforese techniek.

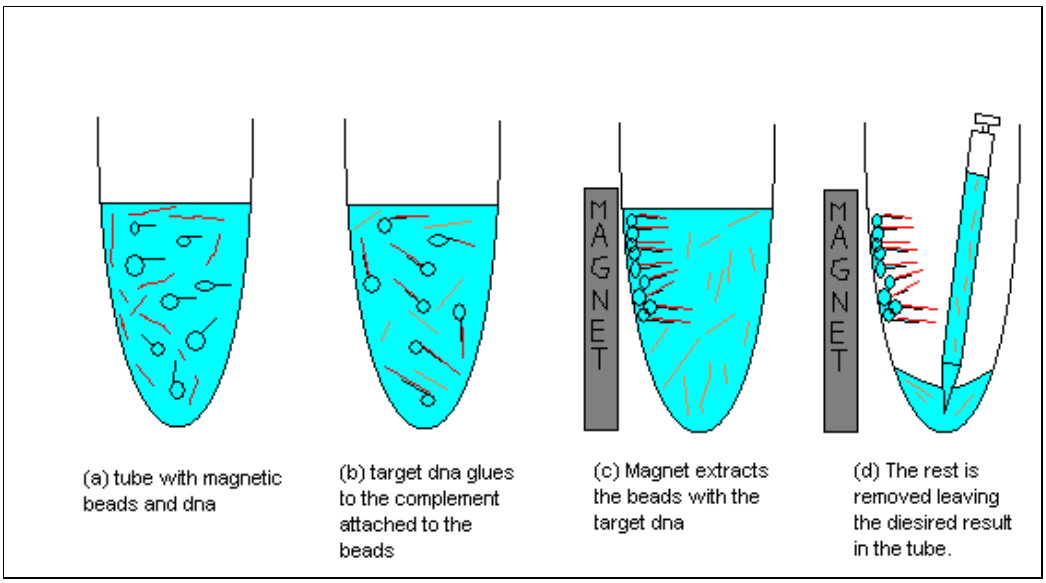


Figuur 1.4: *Gelelektroforese*

1) Restrictie-enzymen knippen het DNA in kleinere fragmenten.
2) De DNA fragmenten worden in de poreuze gel gepipetteerd.
3) Wanneer de stroom wordt ingeschakeld bewegen de DNA fragmenten zich naar de positief geladen kathode.
4) Kleiner moleculen bewegen zich sneller en verder doorheen de gel.

3) Magnetic beads separation

De isolatie van specifieke moleculen, gebaseerd op zijn interactie met zijn complementaire binding partner, is een belangrijke technologie in verschillende onderzoeksvelden, waaronder DNA computing.
Er zijn verschillende kits op de markt beschikbaar die werken op het principe van magnetische labelling en directe isolatie van gebiotinyleerde moleculen zoals DNA, RNA of proteïnen via streptavidine gecoate magnetische 'beads' (figuur 1.5) [9]. Deze gebiotinyleerde moleculen (probes genoemd) kunnen op hun beurt gebruikt worden voor indirecte isolatie van niet-gebiotinyleerde target moleculen (de moleculen die men wenst te isoleren), die interageren met elkaar. Deze procedure houdt een complexe vorming in tussen de gebiotinyleerde probe (DNA, RNA of proteïne) en de target molecule (dit is het interagerende biomolecule DNA, RNA of proteïne). Gebaseerd op de interactie biotine-streptavidine kan het probe-target complex vervolgens gescheiden worden van de rest van de componenten door toevoeging van streptavidine gecoate magnetische beads. Omwille van de magnetische eigenschappen van deze beads kan het intacte DNA-bead complex vervolgens gescheiden worden met behulp van een magneet, waarna het complex twee keer gewassen wordt met wasbuffer om niet specifieke binding moleculen weg te wassen. De niet-gebiotinyleerde target moleculen kunnen vervolgens van het complex gescheiden worden door elutie.



Figuur 1.5: *Het principe van magnetic beads separation*

Hoofdstuk II: DNA computing

2.1 Inleiding

Met het publiceren van de paper ‘Molecular computation of solutions to combinatorial problems’ opende professor Leonard M. Adleman de weg naar DNA computing. Hij beschreef in 1994 de mogelijkheid om een gekend probleem in de informatica op te lossen met het gebruik van DNA moleculen en moleculaire technieken. Niet veel later ontwikkelde Richard J. Lipton, professor aan de Princeton Universiteit, een DNA algoritme dat het ‘satisfiability’ probleem oplost. Een abstractie van het werken op moleculair niveau waarbij de nadruk werd gelegd op de berekenbaarheid van een dergelijke DNA computer werd onlangs samengevat door Martyn Amos. Dit zal eveneens in dit hoofdstuk beschreven worden.
Indien DNA computing inderdaad de toekomst is, is een grondige kennis van de gevolgen en de invloeden ervan in de informatica noodzakelijk.

2.2 Experimenten

In dit deel worden enkele problemen besproken die reeds zijn opgelost via DNA computing. Deze oplossingen zijn later in een abstract model gebracht zoals verder zal toegelicht worden. Er zijn reeds verschillende modellen beschreven alsook de berekenbaarheid ervan in theorie. Hier zullen eerst enkele oplossingen gegeven worden van NP-complete (niet polynomiale complete) problemen die in de DNA computing opgelost kunnen worden.

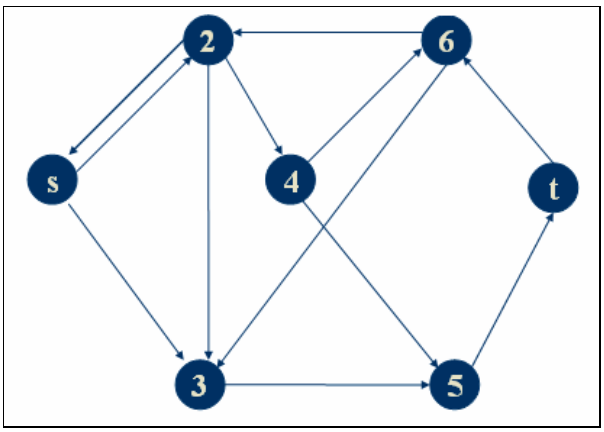
2.2.1 Adleman

Het probleem dat Adleman [10] met DNA heeft willen oplossen is het volgende: bepaal in een gerichte graaf of er een hamiltoniaans pad aanwezig is. Dit probleem is alom bekend in de informatica wereld. Tot heden zijn er verschillende algoritmen ontwikkeld die dit probleem oplossen maar allen hebben een worst case complexiteit die exponentieel is in het aantal knopen. Dit probleem kent voorlopig geen efficiëntere oplossing die beter is dan de gekende oplossingen en valt onder de categorie van NP-complete problemen.
Het volgende algoritme lost het hamiltoniaans probleem op in exponentiële tijd:

- Genereer alle mogelijke paden in de gerichte graaf.
- Filter alle paden eruit die met de gewenste beginknoop beginnen en in de gewenste eindknoop eindigen.
- Filter alle paden eruit die exact het aantal knopen bevatten als er knopen zijn in de graaf.
- Filter alle paden eruit die elke knoop exact één maal passeren.

Indien er een pad overblijft, bestaat er een hamiltoniaans pad in de graaf en is het resultaat positief, anders negatief.

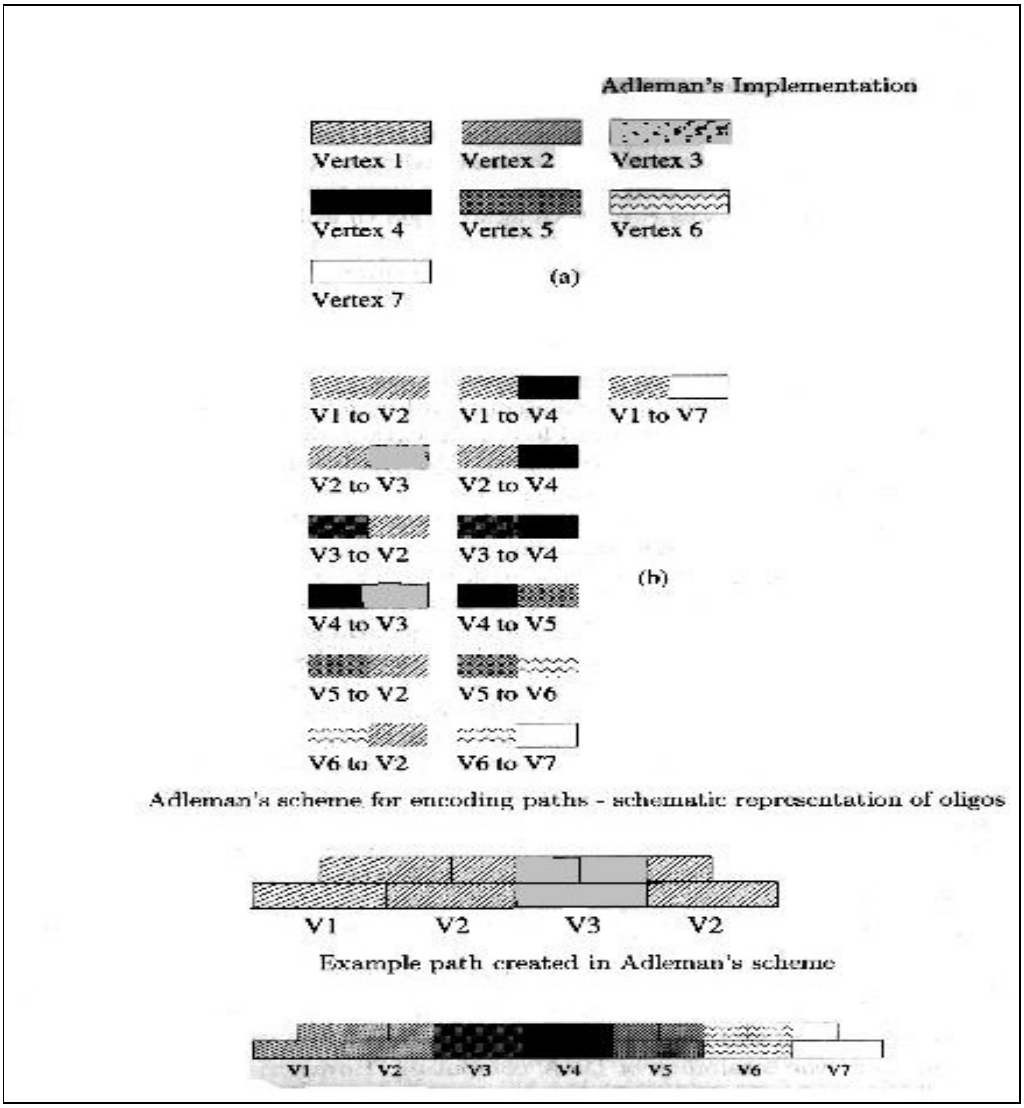
Door alle mogelijke paden te moeten genereren/doorlopen zal er pas na exponentiële tijd een resultaat gekend zijn. Dit is enorm belastend voor een processor, vooral bij toename van het aantal knopen.



Figuur 2.1: *Voorstelling van de gerichte graaf gebruikt door Adleman.*

In de paper beschrijft professor Adleman een oplossing voor het beslissen van een hamiltoniaans pad gegeven een gerichte graaf van 7 knopen (zie figuur 2.1). Er werd gebruik gemaakt van hetzelfde algoritme als hierboven beschreven maar de oplossing van Adleman gebruikte hiervoor DNA en gekende labotechnieken die daarop toegepast kunnen worden.
Door het gebruik van de gekende DNA structuur werd eerst en vooral de opbouw van de graaf beschreven. Een graaf bestaat uit knopen en bogen die bepaalde knopen met elkaar verbinden. Voor elke knoop O_i werd een DNA streng genomen bestaande uit 20 basen. Om nu een boog te beschrijven tussen twee dergelijke knopen $O_{i->j}$ werd de 3' 10-meer van O_i genomen en de 5' 10-meer van O_j. In een gerichte graaf is eveneens de richting van de boog belangrijk. Ook dit is met het gebruik van DNA geen probleem aangezien een DNA streng ook oriëntatie bezit ('5 begin en '3 einde). Zodoende beschrijft een DNA streng $O_{i->j}$ van 20 basen dus een pad (boog) van twee knopen i en j. Al deze knopen O_i kunnen dan gesynthetiseerd en gebruikt worden als input van het algoritme. Om nu paden te kunnen ontwikkelen werden er reverse-complements van knopen toegevoegd aan de input: $\hat{O}_{i->j}$. De input is dan een testbuisje gevuld met de benodigde buffers en het DNA (dat in grote hoeveelheden aangebracht werd).

- Stap 1: Het genereren van alle mogelijke paden.
  Door toevoeging van de reverse-complements $\hat{O}_{i->j}$ en het gebruikelijke ligase proces werden de verschillende bogen met elkaar verbonden om zo alle paden te ontwikkelen in het testbuisje (zie figuur 2.2) Een voorbeeld van een mogelijk ontwikkeld pad zou kunnen zijn: $Os_{->2->3->5->t}$



Figuur 2.2: *Grafische voorstelling van (a) de knopen, (b) de verbinding tussen twee knopen, (c) een pad doorheen de graaf.*

- Stap 2: De paden die beginnen met O_s en eindigen met O_t worden eruit gefilterd.
  Het resultaat uit stap 1 werd vermenigvuldigd met PCR (cfr. supra) gebruik makend van de primers $\hat{O}_s$ en $\hat{O}_t$. Zo werden enkel de paden die een mogelijke oplossing kunnen geven vermeerderd.

- Stap 3: Enkel de paden met de juiste lengte moeten overgehouden worden.
  Enkel de paden die 7 knopen en dus 140 basenparen (bp) lang zijn, kunnen een mogelijke oplossing bieden. Via gelelektroforese (cfr. supra) worden de strengen van 140 bp duidelijk en worden vervolgens geëxtraheerd uit de oplossing.

- Stap 4: Enkel de paden die elke knoop één maal passeren moeten overgehouden worden.
  Op het resultaat van stap 3 wordt de techniek van magnetic beads separation (cfr. supra) gebruikt. Dit werd als volgt toegepast: eerst en vooral werden de mogelijke paden die dubbelstrengig zijn uit elkaar gehaald tot enkelstrengig DNA. Vervolgens werden de strengen gefilterd via magnetic beads separation met $\hat{O}_2$ als koppelaar. Het gevolg is dat alle paden die knoop 2 minstens één maal passeren eruit gefilterd worden. Dit proces wordt herhaald voor alle volgende knopen ($O_3O_4O_5O_6$), behalve de laatste O_t, met als koppelaar respectievelijk $\hat{O}_3$, $\hat{O}_4$, $\hat{O}_5$, $\hat{O}_6$.

- Stap 5: Het resultaat analyseren.
  Het resultaat uit stap 4 wordt via PCR vermenigvuldigd en vervolgens op agarose gel gevisualiseerd. Indien er DNA aanwezig is zal dit te zien zijn op de agarose gel waarbij er dus een resulterend hamiltoniaans pad aanwezig is. Het resultaat van het probleem is dan positief. Indien er geen DNA zichtbaar is, is er dus geen hamiltoniaans pad aanwezig in de graaf en is het antwoord negatief.

Adleman heeft dit probleem dus opgelost door gebruik te maken van andere technieken dan de gebruikelijke computertechnieken. Via het gebruik van de structurele eigenschappen van DNA en de gekende laboratorium technieken op DNA kan dit algoritme tot een oplossing leiden. Hierbij is er een grondige kennis vereist van de eigenschappen van DNA als ook de expertise in het uitvoeren van de nodige laboratorium experimenten.

Is dit beter dan de gekende oplossingen? Er zijn enkele opmerkingen over deze manier van werken. De oplossing met het gebruik van DNA kan voordelen opleveren in de toekomst maar op dit moment werden de experimenten allemaal handmatig uitgevoerd. Dit wil zeggen dat het experiment enkele dagen geduurd zal hebben alvorens een resultaat verkregen werd. Zo is de tijd nodig voor het ligase proces te laten plaatsvinden in stap 1 al aanzienlijk meer dan bij een hedendaagse computer. Dit staat uiteraard in groot contrast met de seconden die de hedendaagse computer nodig heeft voor een oplossing te berekenen.
De voordelen van het gebruik van DNA zijn voorlopig grotendeels theoretisch, mede omdat alles handmatig moet gebeuren, maar zijn toch van die aard dat DNA computing een apart onderzoeksveld is geworden. We zullen twee van de belangrijkste voordelen in de oplossing van Adleman kort aanhalen. Eerst en vooral kunnen alle operaties op de strengen in parallel met elkaar verlopen. Dit staat in groot contrast met de sequentiële aanpak van de hedendaagse computers. Zo zullen bij het filteren van bepaalde paden niet alle paden sequentieel moeten doorlopen worden. De operaties op DNA lopen allemaal in parallel zodat dit toch in polynomiale tijd afgehandeld kan worden in de grootte van het aantal paden.
Een tweede opmerkelijke bemerking is de grootte van de gebruikte energie tijdens het uitvoeren van operaties. Zo kunnen er bij het gebruik van DNA en haar operaties ongeveer 2×10^{19} operaties uitgevoerd worden per Joule energie. Dit is in

vergelijking met de 2×10^9 operaties per Joule bij hedendaagse supercomputers een enorme verbetering.

Zoals reeds vermeld zit er enorm potentieel in het werken met DNA en dit werd via de paper van professor Adleman alleen maar versterkt.

2.2.2 Satisfiability probleem

Na Adleman ontwikkelde Lipton [11] een op DNA gebaseerd algoritme dat het 'satisfiability' probleem oplost.

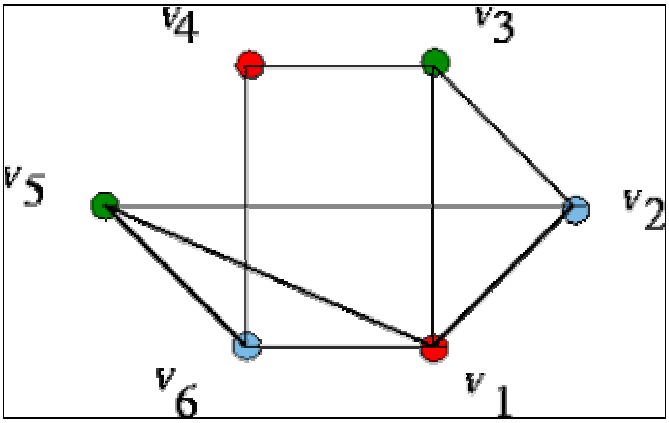
Het satisfiability probleem:
Gegeven n booleaanse formules C_1, …, C_n, die samen in een disjunctieve booleaanse formule F staan. Elke formule C_i heeft een aantal variabelen die in conjunctie zitten. Het satisfiabilility probleem probeert een verzameling van booleaanse waarden te vinden zodat elke variabele gekoppeld kan worden aan zijn overeenkomstige waarde waardoor uiteindelijk elke formule C_i waar is en dus ook F tot waar geëvalueerd kan worden.

Voor k variabelen worden er 2^k strengen DNA aangemaakt die net zoals bij Adleman gelinkt zullen worden om zo alle mogelijke oplossingen voor te stellen. Via een gepaste procedure wordt het resultaat geëxtraheerd. Voor een uitgebreide uitleg wordt verwezen naar [21] maar het principe komt overeen met dat van Adleman.

2.2.3 Het 3-vertex-kleurbaarheidsprobleem

Het 3-vertex-colorability- of kleurbaarheidsprobleem behoort ook tot de klasse van NP-complete problemen en kan opnieuw via dezelfde strategie opgelost worden met DNA computing.

Het 3-vertex-kleurbaarheidsprobleem:
Gegeven een graaf van n knopen en m bogen. Deze graaf is kleurbaar met drie kleuren als elke twee aangrenzende knopen een andere kleur kan toegewezen worden. Met aangrenzend wordt bedoeld dat er een boog bestaat tussen de twee knopen. Figuur 2.3 geeft een voorbeeld van een graaf die inkleurbaar is met drie verschillende kleuren.



Figuur 2.3: *Voorstelling van een ingekleurde graaf die duidelijk 3 kleurbaar is.*

Voor de voorstelling van de graaf heeft men gebruik gemaakt van de driedimensionale structuur van DNA. Voor meer informatie hierover wordt verwezen naar [12] [13].
Het principe gaat als volgt. Voor elke soort knoop (afhankelijk van het aantal geconnecteerde bogen) werd een 'building block' ontwikkeld waarbij er telkens voor elke mogelijke kleur één ontwikkeld werd. Ook voor elke boog werd een streng ontwikkeld die aan de juiste twee knopen kan gaan plakken. Door deze elementen allemaal te mengen en te laten hybridiseren ontstaan alle mogelijke inkleuringen van grafen alsook een heel aantal andere grafen. Verwijder vervolgens alle grafen die verschillen van de originele graaf (via gelelektroforese) alsook de niet volledig ontwikkelde grafen na ligase (via een enzym). Indien het resultaat een streng (graaf) bevat is het antwoord positief en is de graaf dus 3 kleurbaar. Zo niet is het antwoord negatief.

2.2.4 Turing compleetheid

Er zijn reeds vele modellen beschreven die de berekenbaarheid van DNA aantonen en daarenboven 'turing' compleet zijn. Dit wil zeggen dat elk algoritme dat met het model beschreven kan worden eveneens op een turing machine kan geïmplementeerd worden. Zoals verder vermeld zal worden zal het verbindingsmodel (cfr. supra) de turing compleetheid (completeness) verzorgen [14][16][25].
Deze eigenschap kan erop wijzen dat algoritmen op DNA gebaseerd, een competitieve rol kunnen spelen met hedendaagse computerprogramma's in het kiezen van een oplossingsstrategie. Zodoende moet er verder onderzoek in het veld van DNA computing plaatsvinden opdat uiteindelijk het nut van DNA computing specifieker wordt.
Een voorbeeld van een dergelijk model is het 'Boolean Circuit Model' van Ogihara en Ray [15].

2.3 Modellen voor moleculaire berekeningen

In dit deel zullen enkele abstracte modellen beschouwd worden die de berekenbaarheid en nuttigheid van moleculaire technieken behandelen [17]. We spreken hier niet meer concreet over DNA computing maar verruimen het gebied tot op algemeen moleculair niveau. Voor de modellen worden er abstracte operaties beschouwd die gebruikt worden in de gedefinieerde modellen. Het zijn deze operaties en hun implementatie die het succes van een model zullen bepalen. Door deze abstractie is het mogelijk een (tijds)complexiteit voor een operatie te bepalen. Deze complexiteit kan dan gebruikt worden om een globale (tijds)complexiteit te geven van een ontwikkeld algoritme (zoals bijvoorbeeld het algoritme van Adleman). Hiervoor moet men steeds de volgende drie elementen in het achterhoofd houden:

- Ruimte complexiteit: De hoeveelheid DNA die nodig is om de berekening uit te voeren.
- Tijdscomplexiteit: Elke operatie heeft een bepaalde tijd nodig om uitgevoerd te kunnen worden in een laboratorium. De gekozen technieken hebben dus een enorme invloed op de tijdscomplexiteit van een op DNA gebaseerd algoritme. Ook het opstellen van de structuur van het DNA, als de extractie van het resultaat zullen mede deze complexiteit beïnvloeden.
- Een foutenmarge: Elke operatie die uitgevoerd wordt in een labo is afhankelijk van een bepaalde foutenmarge. De keuze van de technieken zullen dus eveneens hun invloed laten gelden op de efficiëntie van een algoritme.

2.3.1 Filter Model

In het filter model [11][17][18] zullen de bewerkingen bestaan uit een reeks operaties die zullen worden uitgevoerd op een eindige verzameling strengen waarbij de elementen behoren tot een bepaald alfabet. Deze verzameling mag en zal daarbij bestaan uit strengen die allen meerdere keren voorkomen. Dergelijke verzameling noemen we ook wel een ‘multi-set’. De basis van dit model is een dergelijke start verzameling die alle mogelijke oplossingen van een bepaald probleem bevat. Gedurende de operaties zullen de strengen die niet voldoen als een oplossing eruit gefilterd worden. Het resultaat is dan een niet lege verzameling met de oplossing in, indien er één bestaat, anders is de verzameling leeg.

2.3.1.1 Unrestricted model en satisfiability model

Adleman [10] en Lipton [11] beschrijven twee dergelijke filter modellen alsook de operaties die daarbij toegelaten zijn. Dit model staat centraal in hun oplossingen. Hoe dit model gebruikt wordt, werd uitgebreid besproken in punt 2.2 waar de algoritmen van Adleman en Lipton uitgelegd staan. Voor de operaties beschouw T de verzameling strengen en S een streng:

- *Separate(T, S)*: Creëert twee nieuwe verzamelingen, één die alle strengen bevat waarbij S een deelstreng is en één die alle andere strengen bevat.
- *Merge(T_1, …, T_n)*: Creëert T_1 U ... U T_n
- *Detect(T)* : Geeft 'true' indien T niet leeg is, en 'false' indien T leeg is.

De operaties die in dit model beschreven worden kunnen uiteraard ook in de praktijk uitgevoerd worden. De labotechnieken die in hoofdstuk 1 beschreven staan kunnen hiervoor gebruikt worden. De operatie *Separate(T, S)* kan uitgevoerd worden door een magnetic beads separatie. De beads bevatten het Watson-Crick complement van S en zullen zo alle strengen uit T die S bevatten scheiden van de rest. De operatie *Merge(T_1, …, T_n)* is eenvoudig. De verschillende substanties DNA T_1,…, T_n moeten gewoon bij elkaar gebracht worden tot één substantie. De substantie T visualiseren via gelelectroforese simuleert de *Detect(T)* operatie. Als er op de gel DNA aanwezig is zal het antwoord 'true' zijn, anders 'false'.

2.3.1.2 Parallel filter model

Dit model [18] beschrijft een veralgemening dat als basis kan dienen om elk NP probleem met DNA computing op te lossen. Hierbij zijn de operaties aangepast, verlopen ze parallel en zien ze er als volgt uit:

- *Remove(T,{S_i}):* Elke streng die één of meerdere keren als deelstreng een streng S_i bevat, wordt verwijderd uit T.
- *Union({T_i},T):* Creëer in parallel verzameling T die de unie is van alle verzamelingen T_i.
- *Copy(T,{T_i}):* Er worden in parallel i kopieën genomen van de verzameling T.
- *Select(T):* Er wordt uit de verzameling T willekeurig één streng gekozen. Indien T leeg is wordt 'empty' geretourneerd.

Deze operaties worden beschouwd een parallelle tijdscomplexiteit te hebben die constant is. Voor de oplossingen in punt 2.2 kunnen er algoritmen beschreven worden in zo een abstract model. Hieronder wordt het algoritme voor het 3-vertex-kleurbaarheidsprobleem (1) en het hamiltoniaans pad probleem (2) gegeven. Voor de eenvoud zal er eerst een algoritme gegeven worden dat de permutatie van een verzameling gegevens produceert.
Opnieuw kunnen deze operaties ook in de praktijk gesimuleerd worden. Zo is de *Remove(T,{S_i})* operatie analoog aan de separate operatie in het unrestricted filter model net zoals de *Union({T_i},T)* analoog kan uitgevoerd worden als de merge operatie. Let wel dat voor dit model de operaties in parallel moeten uitgevoerd worden. Dit wil zeggen dat de magnetic beads separatie voor elke S_i tegelijk afgehandeld moeten worden. Voor het bij elkaar mengen van de verschillende substanties T_i wil dit zeggen dat ze tegelijk (door bijvoorbeeld een robotarm) samen gebracht worden.
Copy(T,{T_i}) zal in de praktijk iets meer tijd in beslag nemen. Eerst en vooral is het nodig een PCR te doen op T. Vervolgens worden daaruit dan i stalen genomen en in een apart testbuisje geplaatst. *Select(T)* tenslotte is ook niet simpel. Eerst en vooral zal er via gelelectroforese moeten bevestigd worden dat er wel degelijk

DNA in de substantie aanwezig is. Zo niet wordt reeds 'empty' geretourneerd. Het willekeurig kiezen van een DNA streng zal enkel mogelijk zijn indien er geweten is welke strengen er juist aanwezig zijn. Daarenboven moeten alle strengen uniek zijn ten op zichte van elkaar zodat een complement van één streng niet zou binden met een ander. Wanneer dit het geval is, is het aan de laborant om willekeurig een streng te kiezen en met het complement een magnetic beads separatie uit te voeren.
Dit model gaat ervan uit dat de acties allemaal parallel kunnen verlopen. In de praktijk is dit zeker niet eenvoudig, soms zelfs onmogelijk.

Permutaties:
Gegeven een verzameling U met gegevens {1, …, n}, bepaal de permutaties. Om de data voor te stellen als een streng wordt volgende codering gebruikt: $p_1i_1….p_ni_n$ stelt de streng voor waarbij i_j een element uit de verzameling voorstelt (het is mogelijk dat het element i meerdere keren in de streng voorkomt) en elke p_j uniek de positie in de streng representeert. De start verzameling bevat dan een reeks strengen van deze vorm. Het algoritme dat in dit model geldt, wordt hieronder beschreven.

```
for(j = 1; j < n; j++)
{
copy(U, {U1, …, Un});
for(i = 1,2,.., n en voor alle k > j)
        in parallel do remove(Ui,{pki met pj ≠ i});  //verwijderen van de slechte input
union({U1, ..., Un},U);
}
```

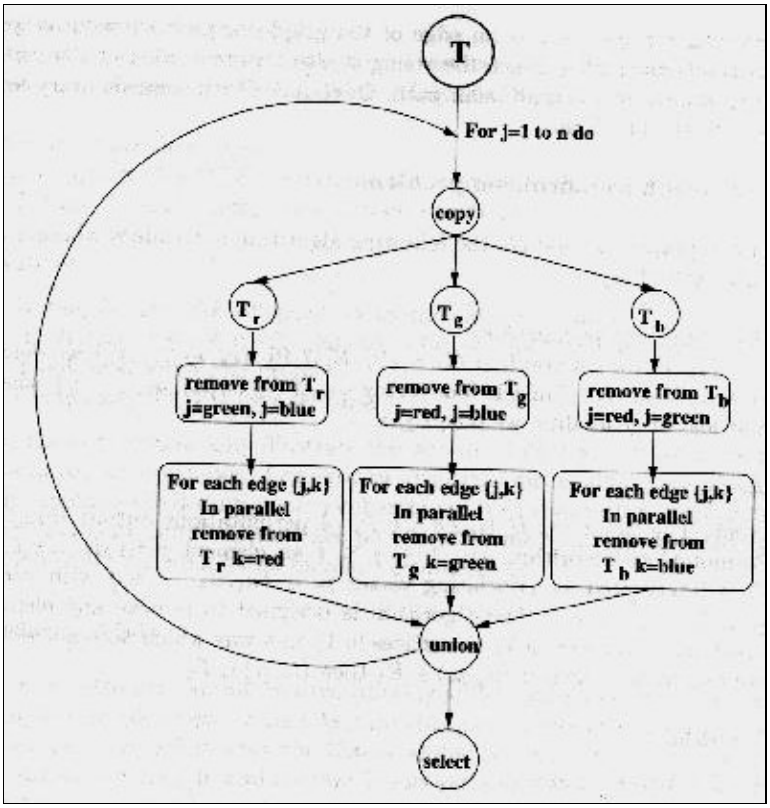
'U' wordt dan uiteindelijk de gevraagde verzameling en wordt P_n genoemd.
Dit algoritme heeft als parallelle tijdscomplexiteit *O*(n).

1) 3-vertex-kleurbaarheidsprobleem voor een graaf G

De start verzameling U bevat alle strengen van de vorm: $p_1c_1…..p_nc_n$ waarbij elke p_ic_i een knoop uit de graaf G voorstelt. p_i stelt opnieuw een unieke positie voor (en dus over welke knoop het gaat) terwijl elke c_i één van de drie kleuren representeert voor de positie i. De hele streng stelt dus een mogelijke inkleuring van de graaf voor. Het volgende algoritme zal de juiste oplossing geven en figuur 2.4 geeft meer inzicht in de werking van het algoritme.

```
for(j = 1; j < n+1; j++)
{
copy(U,{U1, U2,U3});
for(i = 1,2,3 en voor alle k zodat er een boog is tss j en k)
        in parallel do remove(Ui,{ pki met pj ≠ i};
union({U1,U2,U3},U);
}
select(U);
```

De parallelle tijdscomplexiteit is opnieuw *O*(n).



Figuur 2.4: *Flow-graph van het algoritme voor het 3-vertex color probleem*

2) Het hamiltoniaans pad in graaf G

De multi-set waarvoor het volgende algoritme werkt is de verzameling P_n (permutaties) van de knopen die er in de graaf G voorkomen. De representatie van de elementen is net hetzelfde als in de vorige twee gevallen.

```
for( 2≤ i ≤ n-1 en voor alle j,k waarvoor er geen boog bestaat in graaf G)
        in parallel do remove(U,{jpik});
select(U);
```

Deze operaties verlopen in constante tijd zodat er hier van een constante parallelle tijdscomplexiteit kan gesproken worden, namelijk P_n.

Voor een verdere bespreking over de werking van de algoritmen wordt verwezen naar [17].

2.3.2 Verbindingsmodel (*splice model*)

Zoals reeds vermeld, ontbreekt er een operatie zodat dat model dezelfde rekenbare kracht krijgt als een turing machine. De splice-operator zou dit mogelijk maken. De operator werkt als volgt:

> Zij S en T twee strengen. Splits de twee strengen op een bepaalde positie en fusioneer de verkregen delen. Plak het tweede deel van T achter het eerste van S en het tweede van S achter het eerste deel van T. Dit wordt ook wel crossover (cfr. infra) genoemd.

Deze operatie zou met enzymen gevolgd door ligase uitgevoerd kunnen worden in een laboratorium. Voor een algemeen overzicht van dit model wordt er verwezen naar [19][20].

2.3.3 Constructief Model

De modellen die onder deze categorie vallen, zijn gebaseerd op het principe van ‘self-assembly’[28][29]: DNA structuren zullen zelf een eigen structuur opbouwen. Het mooiste voorbeeld is de dubbele helix zoals we DNA algemeen kennen. De wetenschap staat zo ver dat de groei van dergelijke structuren tot op een bepaald niveau kan gecontroleerd worden. De constructieve modellen gebruiken deze kennis om zo bepaalde problemen met DNA en haar structuren op te lossen. Een voorbeeld van dergelijk model is het ‘Tile Assembly Model’ [21]. Dit model kan gebruikt worden om bijvoorbeeld binair te tellen. Voor de werking hiervan wordt verwezen naar [17].

2.3.4 Membraan Model

Dit relatief nieuw abstract model [22] beschrijft de moleculaire berekenbaarheid van levende cellen. Het zijn de structuur en functionaliteit van een levende cel die gebruikt worden om berekeningen uit te voeren. We spreken in dit onderzoeksveld niet van een model maar eerder van systemen. Een voorbeeld van zo een systeem werd door George Paun geïntroduceerd en wordt een P-systeem genoemd.
Een cel bestaat uit regio’s die omsloten worden door een membraan. Dit membraan bepaalt op chemisch vlak wat er in en uit die regio mag. Zo heeft elke regio in de cel een vast aantal ‘regels’ alsook een aantal ‘objecten’.
Onderzoek in dit veld is absoluut nog noodzakelijk. Toch kan er gesteld worden dat werken in een systeem niet-deterministisch is (elke regio heeft een vast aantal objecten en ‘regels’). Het gebruik van levende cellen kan ook opnieuw zoals bij DNA computing, parallelliteit verzorgen [23].

2.4 De toekomst van DNA computing

DNA computing is na één decennium heel wat veranderd. Het optimisme en succes na Adleman is ondertussen omgeslagen naar een realistische, wetenschappelijke, nuchtere toon. Het nut van DNA computing en de bruikbaarheid voor applicaties is bijgeschroefd. Om DNA computing in de toekomst verder te ontwikkelen is het noodzakelijk de bruikbaarheid ervan meer en meer te gaan specifiëren. De voordelen ervan werden in het begin uitgebreid bekend gemaakt aan de wetenschappelijke wereld. Toch is het ondertussen duidelijk geworden dat DNA computing ook heel wat beperkingen kent. In dit hoofdstuk is duidelijk geworden dat NP problemen opgelost kunnen worden via algoritmen in het filter model. Deze problemen die exponentieel groeien naar mate de input groter wordt kan nog steeds veel sneller opgelost worden door het parallellisme van DNA computing. Helaas neemt ook de multi-set met oplossingen exponentieel toe naar mate de grootte toeneemt. Juris Hartmanis [24] heeft reeds aangetoond dat DNA computing onrealistisch wordt naar mate de grootte van het probleem toeneemt. Zo zal er een hoeveelheid DNA nodig zijn ter grootte van de aardbol om het Hamiltoniaans probleem op te lossen met 200 knopen. Het wordt duidelijk dat de toekomst van DNA computing in het domein van de NP problemen beperkt blijft. Een domein waarin DNA computing steevast boven andere implementaties uitsteekt moet gevonden worden, wil deze wetenschap een toekomst hebben.
De complexiteit van een algoritme in de DNA computing werd reeds aangetoond. Dit is nodig om een vergelijking te kunnen maken met de hedendaagse (sequentiële) implementaties. Abstracte modellen waarin operaties gedefinieerd worden, gelinkt aan DNA computing, bieden hier een oplossing. Er zijn reeds verschillende modellen gedefinieerd waarin complexiteit (tijd en ruimte) behandeld worden. Toch is de analyse in dergelijke modellen helaas nutteloos omdat er geen objectieve, uniforme definitie bestaat over het concept tijd en ruimte in de DNA computing. Onderzoek naar een equivalentie op dit gebied tussen het hedendaagse standaard machine model en een op DNA gebaseerd model zal onvermijdelijk zijn.
Daarbij zal het ook noodzakelijk zijn de gebruikte technieken om DNA te manipuleren, te vereenvoudigen en/of te versnellen. Het domein van self-assembly (constructief model) houdt zich hiermee bezig. Het DNA de stappen in een proces automatisch laten uitvoeren is het hoofddoel. Dit idee wordt ook onderzocht in de studie van de “one-pot”-computing [25][26][27]. Hier zullen enzymen het DNA gecontroleerd verwerken. De DNA moleculen kunnen dan als software en de enzymen als hardware beschouwd worden. Door een goede controle op het gebruik van enzymen kan zo de input DNA omgevormd worden tot een oplossing van een bepaald probleem [30].
Zoals gezegd zou er naar verschillende domeinen gekeken moeten worden. DNA zou ook volgens de schrijver nuttig kunnen zijn in andere domeinen dan enkel het ontwikkelen van algoritmen. Er zijn nog enkele velden waarin DNA zeker een toekomst kan hebben. Zo zou DNA een ideaal middel zijn om een stap verder te gaan in het miniaturisatie proces van processors en andere hardware. Vooral opslag van informatie is nog een potentieel aspect. DNA kan gebruikt worden voor het representeren van data en dankzij haar grootte kan er op éénzelfde ruimte tot

100.000 keer meer aan data opgeslagen worden [31] dan de hedendaagse opslagmogelijkheden.
Een ander en relatief nieuw domein is dat van “injectable computers” [32][33]. Shapiro en zijn collega’s hebben reeds een nanocomputer geconstrueerd met het gebruik van DNA. Toch zijn daarbij nog enkele experimenten nodig in een labo en de ‘nanocomputer’ werd alleen uitgevoerd in een testbuisje. De toekomstvisie zou zijn om deze “computer” in een levende cel te injecteren en daar zijn proces te laten uitvoeren. Deze methode mag dan nog grotendeels fantasie zijn, een DNA computer die een berekening uitvoert binnen in het menselijk lichaam zou tot één van de mogelijkheden behoren. Dit onderzoek staat nog maar in zijn kinderschoenen en zal dan ook niet voor morgen zijn.

2.5 Conclusie

De droom om DNA computers te ontwikkelen die alle dromen van een informaticus waar zouden maken, ligt ver verwijderd van de realiteit. Er zal nog heel wat onderzoek moeten gebeuren, vooral op de bruikbaarheid en haalbaarheid van DNA computing, alvorens het werkelijk toegepast zal kunnen worden in echte applicaties. Ondertussen wordt DNA computing in verschillende wetenschappelijke domeinen onderzocht waarbij de mogelijkheden in de toekomst toch weer euforisch klinken.

Hoofdstuk III: Code word design

3.1 Inleiding

Nadat men had ingezien dat het mogelijk was combinatorische problemen op te lossen met behulp van DNA is men verder gaan kijken dan enkel de oplossing van Adleman. Maar net zoals bij de oplossing van het hamiltoniaanse pad worden strenge eisen gesteld aan de keuze van de te gebruiken DNA strengen. De problemen ontstaan vooral bij de hybridisatie van strengen met hun Watson-Crick complement. Dit probleem is een studie op zich beginnen vormen en wordt beschreven als 'code word design'. Het is de bedoeling de input van een probleem zo op te stellen dat de te gebruiken strengen met elkaar gaan interageren zoals het vooraf opgesteld wordt.
Deze studie kende ondertussen heel wat vooruitgang waarbij het werk van A. Condon en M. Garzon toch wel de grootste invloed hebben gehad.

Het probleem kan als volgt beschreven worden: ontwerp een verzameling DNA codes (strengen) over een alfabet {A, C, G, T} die aan bepaalde combinatorische beperkingen voldoen. Het doel van de beperkingen is de kans op vals positieven of 'false positives' en vals negatieven of 'false negatives' te minimaliseren en zo het aantal mogelijk te gebruiken strengen te beperken. Er zijn immers 4^n (met n de lengte van de code words) mogelijke DNA code words dat in een algoritme kan gebruikt worden. De codes op zich zijn in de meeste gevallen van éénzelfde lengte en enkelstrengig. In de verdere beschrijving wordt steeds van deze veronderstelling uitgegaan.

False positives: false positives ontstaan tijdens de hybridisatie van strengen met hun Watson-Crick complement. Indien er meerdere strengen zijn die een (sterke) binding aan kunnen gaan met dergelijk complement zullen de hybridisaties leiden tot verkeerde resultaten. Daarom is het belangrijk dat de keuze van de input strands (= strengen) elk een unieke hybridisatie aangaan met hun eigen Watson-Crick complement.

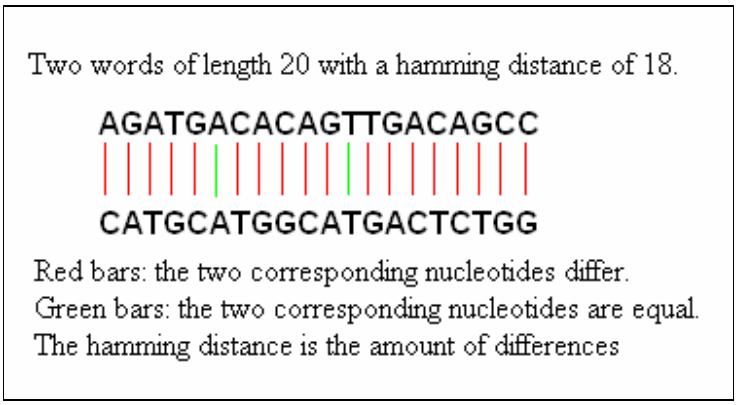
False negatives: men spreekt van false negatives indien een resultaat van een probleem bestaat maar de gegeven DNA strengen één of meerdere oplossingen niet kunnen produceren.

Zij $x = x_1 \ldots\ldots x_n$ een code word van n basen dan is de/het
Reverse van x (x^R): $x_n x_{n-1} \ldots x_1$
Complement van x (x^C): de streng waarbij elke A vervangen wordt door een T en vice versa, en elke C door een G en vice versa.

3.2 Beperkingen ('constraints')

3.2.1 The Hamming constraint

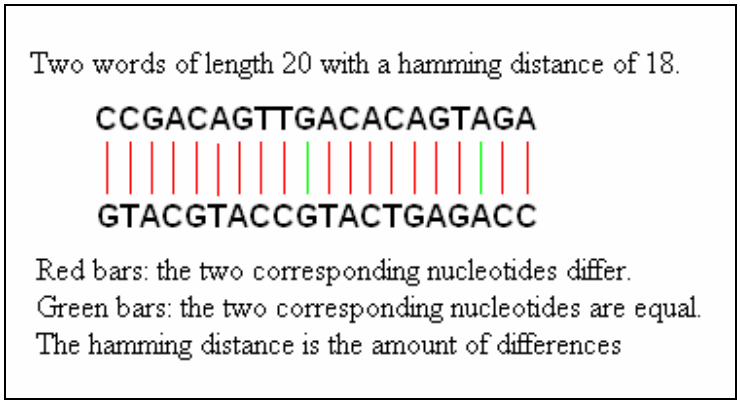
De 'Hamming' afstand H(w,x) van twee code words w en x is het aantal verschillende elementen op elke positie [34]. Dus voor elke positie i wordt gecontroleerd of $x_i \neq w_i$. Deze beperking wordt als volgt gebruikt: zij d de minimale Hamming afstand tussen twee code words dan is de verzameling van code words die aan deze beperking voldoen alle strengen die onderling een Hamming afstand hebben van minimaal d ($H(w,x) \geq d$). Hoe de Hamming afstand bepaald wordt is duidelijk te zien in een voorbeeld met code word w = AGATGACACAGTTGACAGCC en code word x = CATGCATGGCATGACTCT GG en Hamming afstand 18.



Het maximum aantal code words dat een verzameling kan bevatten gegeven een Hamming afstand d, wordt als volgt genoteerd: $A_q(n,d)$. Hier is q de grootte van het alfabet (in dit geval 4), n de lengte van de code words en d de minimale Hamming afstand.

3.2.2 The Reverse-Complement constraint

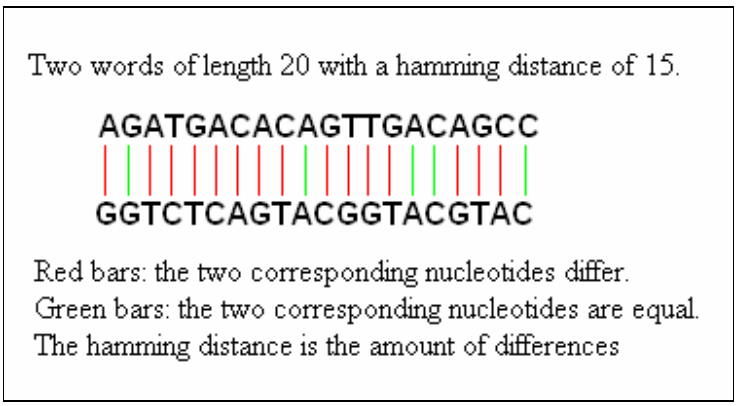
Deze beperking beschrijft eveneens een Hamming afstand tussen twee code words [34]. Hierbij wordt de reverse van het ene en het complement van het andere in vergelijking gesteld. Zij opnieuw d de minimale Hamming afstand dan is de resulterende verzameling alle code words waarbij de Hamming afstand van de reverse t.o.v. een complement minimaal afstand d hebben: ($H(w^R,x^C) \geq d$). Verdergaand op vorig voorbeeld zal w^R = CCGACAGTTGACACAGTAGA en x^C = GTACGTACCGTACTGAGACC. De Hamming afstand is dan 18 zoals te zien is in volgend voorbeeld:



Ook hier wordt een maximum beschreven op het aantal mogelijke code words die aan de beperking voldoen: $A_q^{RC}(n,d)$.

3.2.3 The Reverse Constraint

Opnieuw wordt de Hamming afstand genomen tussen alle code words onderling [34]. Deze keer wordt de afstand genomen tussen elk woord met elk ander zijn reverse. Dus de code words w en x behoren tot een verzameling met reverse constraint minimaal d als $H(w,x^R) \geq d$ geldt. Verdergaand op het gegeven voorbeeld resulteert dit in de oorspronkelijke w en x^R = GGTCTCAGTACGGTACGTAC.



Het maximum aantal code words die aan de reverse constraint voldoen wordt beschreven als $A_q^R(n,d)$.

3.2.4 The free energy constraint

Deze beperking gaat rekening houden met de smelttemperatuur van de code words die in de verzameling zitten [35]. Dit zal uitgerbeid behandeld worden in punt 3.3.

The GC content constraint:
Men gaat hier een beperking opleggen op het voorkomen van G's en C's. Deze twee basen hebben immers een grote invloed op de smelttemperatuur, zoals vermeld in punt 1.2.3.1. Door alle code words van een verzameling een vast of maximum aantal G's en C's te laten bevatten kan er gemakkelijker rekening gehouden worden met de smelttemperatuur.

3.2.5 The forbidden subword constraint

Dit is een beperking die alle code words van een verzameling verbiedt een bepaalde DNA streng als subword te hebben [35].

3.2.6 The frame shift constraint

Dit is een beperking op een reeks strengen waarbij één streng niet mag voorkomen als deelstreng, nadat de andere strengen geconcateneerd zijn [36]. Volgend voorbeeld geeft deze beperking weer in het geval van drie elementen.

x = ACTGATCGTA
w = CTCGATCGAT
z = ***GTACTCGA***

xw = ACTGATC***GTACTCGA***TCGAT

De streng z is een verboden element in deze verzameling omdat ze na de concatenatie een deelstreng is van xw.

3.2.7 Comma-freeness

Zij S een verzameling code words dan is deze verzameling comma-free als de volgende stelling geldt:

De overlapping van elke twee (niet noodzakelijk verschillende) codes in de verzameling S resulteert niet in een nieuw code word voor die verzameling S.

Met andere woorden: zij $x = x_1 \ldots x_n$ en $y = y_1 \ldots y_n$ twee code words in een verzameling S, dan is $z = x_{r+1} \ldots x_n y_1 \ldots y_r$ (met $0 < r < n$) reeds aanwezig in S.
Indien de overlapping van elk koppel woorden uit S resulteert in woorden die minstens op d lokaties verschillen met één uit S dan wordt S ook wel "comma-free met index d" genoemd [33][37].

3.3 Gevolgen

Al deze beperkingen zijn uitgebreid onderzocht en hebben geleid tot een aantal interessante eigenschappen die verzamelingen DNA strengen kunnen hebben.

Zoals reeds vermeld bestaat een verzameling S maximaal uit q^n code words, met q het aantal elementen in het alfabet en n de lengte van de code words. Door een beperking op te leggen aan de code words gaan we eigenlijk een deelverzameling van S bepalen. De grootte van een deelverzameling kan bijvoorbeeld voor de Hamming afstand gemakkelijk berekend worden. Laat ons de verzameling bepalen van code words die van een gegeven streng s een Hamming afstand d hebben, genoteerd als V(s,d). Omdat de verzameling van code words onafhankelijk is van de streng s, schrijven we V(d) in plaats van V(s,d) en deze is gelijk aan:

$$\sum_{i=0}^{d} \binom{n}{i} (q-1)^i$$

Nog enkele interessante relaties gerelateerd aan de Hamming afstand zijn de volgende:

$$A_q(n,n) = q$$

Deze vergelijking kunnen we gemakkelijk aantonen. Voor het alfabet bestaande uit de vier nucleotiden {A,C,G,T} en een lengte n = 10 kunnen er inderdaad maar vier code words gevormd worden die een Hamming afstand hebben van 10. Dit zijn de woorden bestaande telkens uit één element:

AAAAAAAAAA
CCCCCCCCCC
GGGGGGGGGG
TTTTTTTTTT

Van zodra er ergens een base verandert, zal niet meer aan de voorwaarde $H(w,x) \geq 10$ kunnen voldaan worden.

$$A_q(n,d) \geq A_q(n+1,d+1)$$
$$A_q(n,d) \geq A_q(n+1,d) / q$$

Deze twee relaties worden hier niet verder uitgelegd omdat ze niet van toepassing zijn op dit proefschrift.

Ook voor de reverse constraint bestaan er belangrijke relaties:

The halving bound:

$$A_q^R(n,d) \leq A_q(n,d) / 2$$

Construction voor d = 2:

$$A_q^R(n,2) = q^{n-1} / 2 \qquad \text{n even}$$
$$(q^{n-1} - q^{[n/2]}) / 2 \leq A_q^R(n,2) \leq q^{n-1} / 2 \qquad \text{n oneven}$$

Product bound:

$$A_4^R(n,d) \geq A_2^R(n,d)\ A_2^R(n,d)$$

Er bestaat tevens een sterk verband tussen de verzamelingen van code words die voldoen aan de reverse constraint en aan de reverse-complement constraint:

$A_4^R(n,d) = A_4^{RC}(n,d)$ n even
$A_4^R(n,d) \leq A_4^{RC}(n,d) \leq A_4^R(n,d)$ n oneven

Deze relaties worden hier niet verder uitgelegd omdat ze eveneens niet van toepassing zijn op dit proefschrift. Tal van andere relaties en vergelijkingen zijn reeds ontdekt en bewezen. Voor meer informatie hierover wordt verwezen naar [34].

Wat als we een verzameling woorden hebben gevonden? Dan moet er aangetoond worden dat ze, na het vormen van de lange strengen, geen secundaire structuren of 'secondary structures' vormen [32][35]. De free energy constraint handelt hierover.

'The free energy of a DNA strand':

Elk code word op zich kan ook een probleem opleveren. We zeggen dat ze secundaire structuren of 'secondary structures' kan hebben. D.w.z. dat de streng op zich bij een bepaalde temperatuur met zijn eigen kan gaan binden indien daar de mogelijkheid toe bestaat. Het vormen van een hairpin of een duplex zijn hier twee voorbeelden van.
Een hairpin wordt gevormd door enkelstrengig DNA wanneer ze twee clusters basen bevat waarbij de één het complement is van de andere. Hier worden hairpins besproken omdat hiermee in dit proefschrift ook rekening gehouden werd. Een voorbeeld geeft een idee hoe een hairpin gevormd wordt bij een DNA streng:

3'ACGTACGT CATACGTACGTAGCTACGT ACGTACGT 5'

Deze streng vormt in een bepaalde toestand een hairpin als volgt:

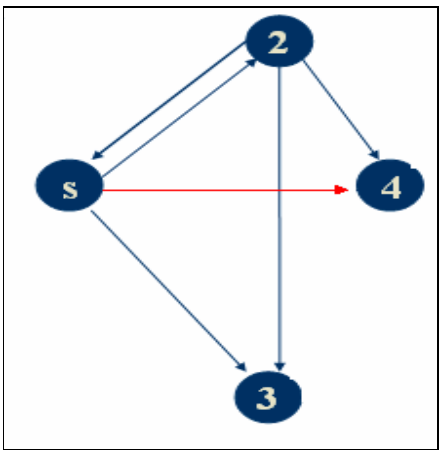
```
3' ACGTACGT CATACGTAC
   | | | | | | | |      G
5' TGCATGCA TGCATCGAC
```

De theorie rond free energy van een DNA streng houdt dus rekening met dergelijke mogelijke secondary structures. Er bestaan nog een aantal andere mogelijke secondary structures zoals een 'Internal Loop' of een 'Bulge'. Door voor elk deel van de streng met een bepaalde formule de energie te berekenen van een bepaalde secondary structure kan uiteindelijk een globale energie berekend worden. Een DNA streng wordt dan 'structure free' verklaard wanneer de totale energie positief is. Voor verdere informatie hieromtrent wordt verwezen naar [32][35].

3.4 Code word design bij Adleman

In theorie is het werken met DNA strengen een mooi gegeven, in de praktijk daarentegen kan er veel mislopen. Eerst en vooral is er de expertise vereist van de laboranten om met DNA te kunnen werken. Het is immers werken met iets wat je niet met het blote oog kan waarnemen. De kennis over DNA en de operaties die men op DNA kan toepassen is doorheen de jaren enorm verbeterd, toch blijft het nog altijd werken met iets wat niet in een exacte wetenschap omkaderd is. Contaminatie van het product alsook foute handelingen leiden onvermijdelijk tot mislukking van het experiment.
Het grote probleem, niet alleen bij de oplossing van Adleman, is het kiezen van de DNA strengen waarmee gewerkt zal moeten worden. Het is immers niet mogelijk een oplossing via DNA te verwezenlijken met een willekeurige keuze van strengen. Code word design houdt zich bezig met deze keuze. Er zal dan ook grondig gekeken worden naar het code word design bij Adleman

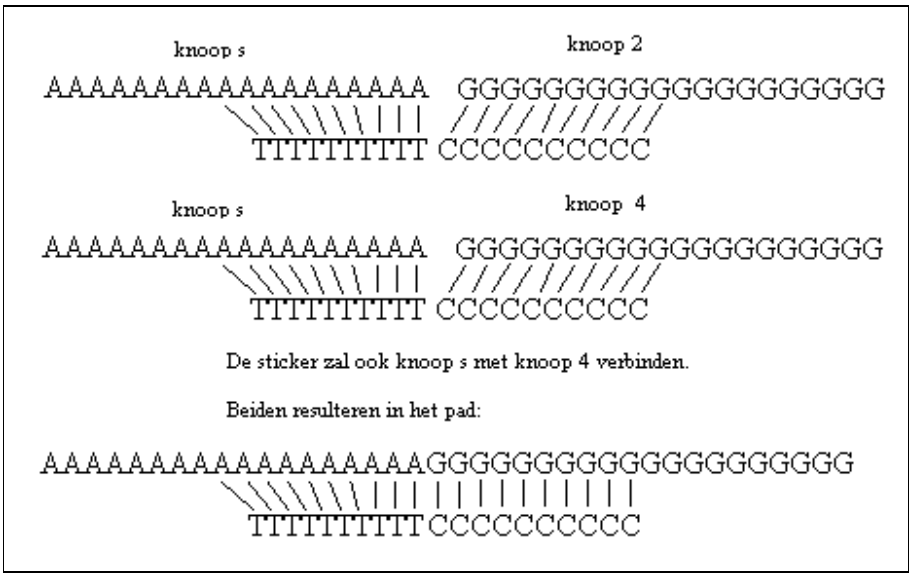
Om het probleem van Adleman in de praktijk mogelijk te maken moeten er zeven DNA strengen gekozen worden, om de knopen te representeren van telkens 20 basenparen. De zeven 20-meren (knopen) kunnen gekozen worden uit een verzameling bestaande uit 4^{20} DNA strengen van 20 bp lang. Een verkeerde keuze zal onvermijdelijk leiden tot 'false positives' en dus tot het mislukken van het algoritme. Zowel het uitblijven van resultaten als het genereren van verkeerde resultaten behoren tot de mogelijkheden.
In de eerste stap van het algoritme moeten er paden gegenereerd worden met de gekozen DNA strengen. Een groot probleem bij deze stap is de creatie van niet bestaande paden door hybridisatie van verkeerde knopen. Bij een verkeerde keuze van het DNA resulteert deze stap in foute, niet bestaande paden. Deze verkeerde 'bindingen' leveren false positives op waardoor verdere fasen van het algoritme overbodig worden. Dit wordt verduidelijkt in figuur 3.1 waar een deel van de graaf wordt voorgesteld.



Figuur 3.1: *Een deel van de graaf van Adleman. De rode verbinding is er één die niet aanwezig is in de oorspronkelijke graaf.*

De knoop s heeft een pad naar knoop 2 en 3. Indien de structuur van knoop 4 gelijk is aan die van knoop 2 zal er ook een pad van knoop s naar knoop 4 ontwikkeld worden. De sticker is samengesteld uit het complement van de laatste 10 basen van de ene knoop gevolgd door het complement van de eerste tien basen van de volgende knoop

knoop s = “AAAAAAAAAAAAAAAAAAAA”
knoop 2 = “GGGGGGGGGGGGGGGGGGGG”
knoop 4 = “GGGGGGGGGGGGGGGGGGGG”
sticker = “TTTTTTTTTTCCCCCCCCCC”



Het pad tussen knoop s en knoop 4 zal in de praktijk ontwikkeld worden hoewel dit niet voorzien is in de graaf. Om dergelijke problemen te vermijden moeten de strengen die de knopen voorstellen, onderworpen worden aan een aantal beperkingen.
De beperking op de keuze van de strengen bij het probleem van Adleman zijn:

- De strengen moeten uniek zijn ten opzichte van elkaar, d.w.z. dat de complementen gebruikt als stickers geen verbindingen maken met andere knopen.
- De knopen zelf mogen geen secundaire structuren vormen (zoals hairpins), anders worden ze onbruikbaar.
- De hybridisatie temperatuur moet goed zijn zodat de volledige strengen op hetzelfde moment gaan ‘plakken‘.

De verzameling van mogelijke strengen DNA wordt op die manier verkleind. De keuze zal dan uit deze verzameling genomen moeten worden opdat de kans op succes gegarandeerd wordt.

Algemeen geldt dat alvorens elk probleem met DNA computing wordt opgelost, de keuze van het DNA het succes of de mislukking bepaalt.

3.5 Conclusie

Het domein dat zich bezig houdt met code word design gaat proberen een algemene omschrijving te geven van een verzameling strengen DNA. Deze verzameling is een deelverzameling van alle mogelijke strengen die over een bepaald alfabet en de lengte van de strengen gedefinieerd kunnen worden. De deelverzameling zal dan strengen bevatten die voldoen aan vooropgestelde beperkingen zodat ze gebruikt kunnen worden in een algoritme van DNA computing. De beperkingen zijn meestal probleem afhankelijk maar voor het algemene geval zijn er reeds interessante eigenschappen bewezen die al dan niet kunnen gebruikt worden bij een algoritme. Ze geven vaak een boven- en ondergrens op de grootte van deze deelverzamelingen.
Het is reeds aangetoond dat de ‘free energy’ beperking een belangrijke factor speelt tijdens het ontwerp van een verzameling code words.

Hoofdstuk IV: Het berekenen van booleaanse functies met DNA computing

4.1 Inleiding

In dit hoofdstuk zal er gekeken worden naar booleaanse functies en de evaluatie ervan met het gebruik van DNA computing. Daarbij zal er concreet gekeken worden naar het n-bit majority probleem en de mogelijke implementatie ervan in de DNA computing. Het 3-bit majority probleem is hier een voorbeeld van en er is reeds gepoogd dit probleem op te lossen met DNA en de gekende labotechnieken.

Dit experiment heeft plaatsgevonden aan het Limburg Universitair Centrum (LUC) in het jaar 2003 maar kende niet het gehoopte succes. De mogelijke oorzaken hiervoor kunnen niet met zekerheid bepaald worden. Een mogelijke verklaring van het falen van het experiment zal kort besproken worden in hoofdstuk 7. Een meer algemene beschrijving van het probleem en de oplossing ervan zal in dit hoofdstuk uitgebreid behandeld worden.

4.2 Probleembeschrijving

Een booleaanse functie F is een functie waarbij er n variabelen elk de waarde 0 ('false') of 1 ('true') gegeven kan worden. Wanneer van elke variabele zijn waarde gekend is, kan de functie geëvalueerd worden. Het resultaat is dan eveneens 0 ('false') of 1 ('true').

$$F: \{0,1\}^n \rightarrow \{0,1\}$$

Het n-bit majority probleem is een welgekend probleem in de informatica en wordt dus voorgesteld door de functie F. Het probleem staat bekend als eenvoudig en kent een oplossing die eveneens eenvoudig is. Ze wordt als volgt beschreven:

Gegeven zijn n booleaanse variabelen. Beslis of het merendeel van de variabelen op 'true' staan. Het resultaat is positief indien minstens n/2 + 1 variabelen als 'true'

gemarkeerd worden, anders negatief. Hieronder wordt een voorbeeld gegeven in het geval van n = 3:

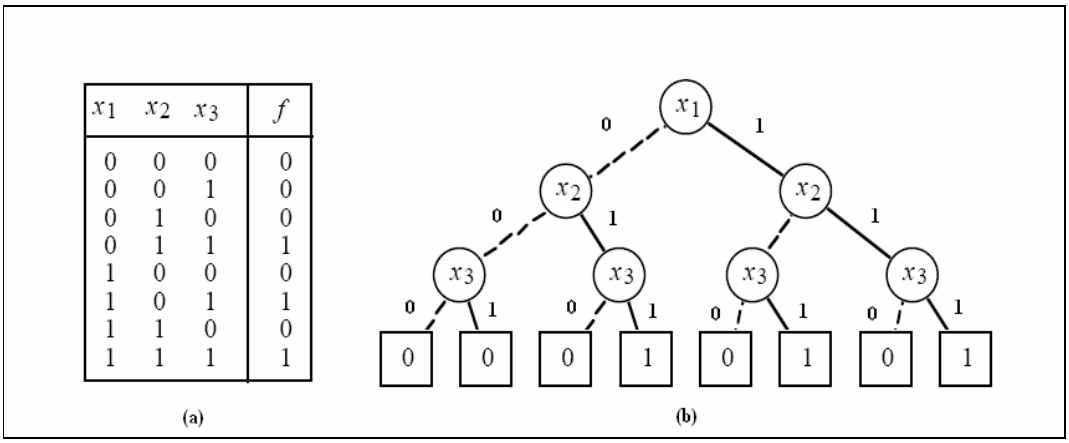
> 3 variabelen x_1, x_2, x_3 met als waarden waarbij we 1 als 'true' en 0 als 'false' aannemen:
>
> $x_1 = 1$
> $x_2 = 0$
> $x_3 = 1$
>
> Bij dit voorbeeld mag besloten worden dat er minstens 2 van de 3 variabelen op true staan en is het resultaat dus positief.

Het probleem wordt opgelost door elke variabele sequentiëel te overlopen en uit hun waarden tot een correcte beslissing te komen.

Om het probleem nu met n variabelen op te lossen bestaan er ondertussen middelen die zowel rekenkundige als grafische hulp bieden om tot een juiste resultaat te komen. Eén daarvan is het 'Ordered Binary Decision Diagram (OBDD)'.

4.2.1 Ordered Binary Decision Diagram (OBDD)

Een OBDD is een gerichte acyclische graaf die booleaanse functies representeert waarbij ze het testen en beslissen van booleaanse eigenschappen eenvoudig maakt. Voor het n-bit majority probleem zal een OBDD een binaire boom worden. Elke waarde van een variabele ligt vast en zal een uniek pad hebben naar een blad van de boom. Dat blad is dan de oplossing voor het probleem en zal uiteraard ofwel 0 (negatief) ofwel 1 (positief) zijn.
Een voorbeeld van een OBDD voor drie booleaanse variabelen is gegeven in figuur 4.1 maar kan uiteraard simpelweg uitgebreid worden voor elke grootte van n. Voor een gespecialiseerde uitleg zie [27][38].



Figuur 4.1: *Een OBDD voor 3-bit majority probleem met drie variabelen.*

4.2.2 Een programma

Voor het n-bit majority probleem is het ook eenvoudig een computerprogramma te schrijven dat een oplossing geeft voor een bepaalde fuctie F. Opnieuw zal elke variabele overlopen moeten worden en uiteindelijk ondubbelzinnig tot het juiste antwoord leiden. Figuur 4.2 geeft een programma voor n = 3.

```
If ( x1 == 1)
{
        if ( x2 == 1)
        {
                return “true”;
        } else
        {
                if (x3 == 1)
                {
                        return “true”;
                } else
                {
                        return “false”;
                }
            }
} else
{
        if ( x2 == 1 && x3 == 1)
        {
                return “true”;
            } else
            {
                return “false”;
            }
}
```

Figuur 4.2: *Een eenvoudig computerprogramma dat het 3-bit majority probleem oplost.*

4.3 n-bit majority probleem en DNA computing

Een algoritme ontwikkelen in DNA computing dat het n-bit majority probleem oplost is afhankelijk van de representatie van de variabelen. Daarom zal er eerst aandacht geschonken worden aan de voorstelling van de input.

De belangrijkste stap wordt het bepalen van de structuur van een DNA streng die de bepaalde ‘bitstring’ gaat voorstellen. Er zijn n variabelen die elk twee waarden kunnen aannemen: namelijk true (1) of false (0). Laat ons deze twee waarden afkorten tot ‘t’ voor true en ‘f’ voor false. Ze worden geïndexeerd zodat ze aan de juiste variabele gekoppeld kunnen worden. Zo kan dan de variabele x_1 de waarde t_1 of f_1 krijgen. De algemene expressie die alle mogelijke ‘bitstrings’ weergeeft wordt dan:

$[t_1 + f_1][t_2 + f_2]\ldots[t_n + f_n]$

vb. $101 = t_1\, f_2\, t_3$ voor n = 3

Om de 'bitstring' volledig voor te stellen zodat de n waarden (van de variabelen) zichtbaar en uniek kunnen onderscheiden worden, is het noodzakelijk voor elke waarde {0,1} van elke variabele $\{x_1,\ldots , x_n\}$ een aparte streng te ontwerpen. Daardoor zullen er voor n variabelen dus 2n strengen moeten ontworpen worden. Van deze strengen zullen er dan n gebruikt worden voor elke biststring zodat de globale lengte van de DNA streng (20 x n) wordt. We nemen dezelfde lengte voor een element als bij de oplossing van Adleman.
Omdat er een aantal keer een PCR procedure gebruikt zal worden, zullen er een upper en lower primer toegevoegd worden aan de DNA voorstelling. Beide primers (p_1, p_2) zullen eveneens bestaan uit 20 basenparen. De algemene representatie wordt dan:

$p_1\,[t_1 + f_1][t_2 + f_2]\ldots[t_n + f_n]\,p_2$

Een DNA streng die dan bijvoorbeeld de 'bitstring' 101 voorstelt is de volgende:

$101 = p_1\, t_1\, f_2\, t_3\, p_2$ voor n = 3

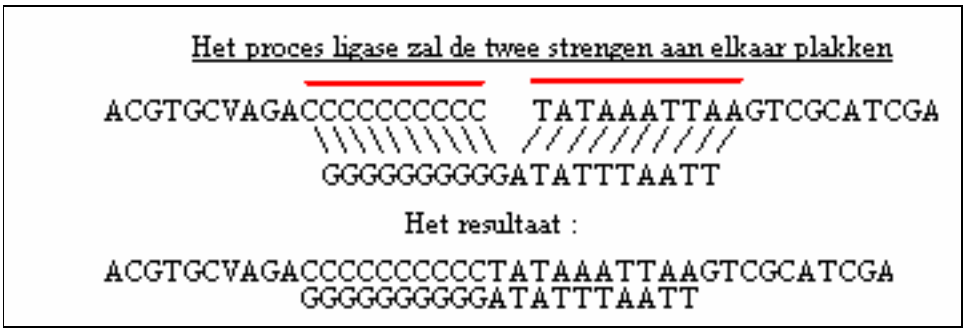
Het is heden mogelijk DNA strengen te synthetiseren, maar slechts van beperkte lengte (tot 70 à 80 basenparen). Indien er dus meer dan drie variabelen gebruikt zullen worden ontstaat het probleem dat het DNA niet meer aangemaakt kan worden. Een oplossing moet dus gevonden worden om de volledige streng te verkrijgen die de bitstring gaat representeren. We kijken hiervoor terug naar de oplossing van Adleman. Voor elke knoop in de graaf werd een streng bepaald. Om een pad te definiëren, gebruikte Adleman de structuur van die strengen, alsook de procedure 'ligase' en plakte zo de verschillende knopen aan elkaar. Op die manier werd dan een langere streng DNA verkregen die een pad in de graaf voorstelde. Dit principe zal ook hier toegepast worden.
Tussen de strengen die de waarden representeren plaatsen we extra strengen die we stickers zullen noemen. De lengte ervan wordt opnieuw op 20 basenparen gezet. De stickers hebben een tweede nut in de opbouw van onze strengen. Ze geven een duidelijke afscheiding tussen de verschillende variabelen. Het is dan uiteraard noodzakelijk dat ze totaal verschillen van de booleaanse waarden.
De werking van stickers wordt verduidelijkt in figuur 4.3. Na een booleaanse waarde wordt een eerste sticker geplaatst. Een andere wordt dan voor de volgende waarde gezet. Hun complement zal ervoor zorgen dat tijdens ligase de twee delen aan elkaar geplakt zullen worden.

Voor de eenvoud nemen we strengen van lengte 10. Het principe blijft behouden voor andere lengtes.

t_1 = ACGTGCVAGA met sticker 1 = CCCCCCCCCC (e)
f_2 = GTCGCATCGA met sticker 2 = TATAAATTAA (b)

We beschouwen als input:

t_1e = ACGTGCVAGACCCCCCCCCC
en
bf_2 = TATAAATTAAGTCGCATCGA
en
het complement van eb = GGGGGGGGGGATATTTAATT



Figuur 4.3: *Het ligase proces waarbij de twee delen aan elkaar 'geplakt' worden.*

Indien we dit principe nu toepassen op de gehele structuur van het DNA, hebben we een sticker na elke [t_i + f_i] (e_i) en één voor [t_{i+1} + f_{i+1}] nl. (b_{i+1}) met i in [1,n-1]. De gehele expressie die een DNA streng definieert om de waarden van de variabelen te representeren wordt dan:

$$p_1\,[t_1 + f_1]\,e_1\,b_2\,[t_2 + f_2]\,e_2\ \ldots\ b_{n-1}\,[t_{n-1} + f_{n-1}]\,e_{n-1}\,b_n\,[t_n + f_n]\,p_2$$

Een DNA streng die de bitstring van booleaanse waarden representeert zal een lengte hebben van (n x 20) + (20 x 2 x (n-1)) + (2 x 20) bp (# variabelen + #stickers + #primers). Voor n = 3 spreken we dan van (3 x 20) + (20 x 2 x (3-1)) + (2 x 20) = 180 bp (9 keer 20 basenparen) en ziet er als volgt uit:

$$p_1\,[t_1 + f_1]\,e_1\,b_2\,[t_2 + f_2]\,e_2\,b_3\,[t_3 + f_3]\,p_2$$

Met deze structuur zal er vervolgens een algoritme ontworpen worden in de DNA computing om het probleem op te lossen. Indien we dus het 3-bit majority probleem willen berekenen met DNA computing zullen er in totaal 12 verschillende strengen DNA nodig zijn (de 6 waarden, de 4 stickers en de 2 primers). Algemeen zullen er 4n strengen bepaald moeten worden om het n-bit majority probleem te kunnen aanvangen.

$$\underset{\text{(primers)}}{2} + \underset{\text{(de waarden)}}{2n} + \underset{\text{(de stickers)}}{2(n-1)} = 4n$$

4.4 Oplossing met DNA computing

Nu we weten hoe het DNA gestructureerd is kunnen we het algoritme bepalen dat gevolgd moet worden om het n-bit majority probleem op te lossen in de DNA computing.

Het algoritme voor n variabelen

- **Stap 1:** Bepaal de verschillende DNA input strengen (zie punt 4.3).
- **Stap 2:** Voer PCR uit zodat de input voldoende aanwezig is in een stabiele omgeving.
- **Stap 3:** Voer een ligase proces uit zodat de volledige strengen gevormd worden die dan de input representeren voor de volledige bitstring genaamd S. Deze substantie wordt in een verzameling M gestoken. Hou twee verzamelingen ACCEPT en DECLINE bij. Deze zijn voorlopig leeg.
- **Stap 4**: Voer een magnetic beads separatie uit op S met als probe het complement van de positieve waarde van de eerste variabele x_1. Dit resulteert in twee nieuwe substanties.
    1. Het mengsel dat de strengen bevat met de positieve waarde voor de eerste variabele, genoemd T1.
    2. Het mengsel dat de strengen bevat die de positieve waarde voor de eerste variabele niet bevat. Ze bevat dus alle strengen waarbij de waarde voor de eerste variabele negatief is. We noemen deze F1.

  Verwijder S uit de verzameling M en voeg de twee substanties T1 en F1 hieraan toe.
- **Stap 5:** Zet een teller i op 2.
  Voer een magnetic beads separatie uit op elk element Y in M met als probe het complement van de positieve waarde van de i-de variabele.
  Voor elk element Y uit M resulteert dit in twee nieuwe substanties:
    1. Het mengsel dat de strengen bevat met de positieve waarde voor de i-de variabele en dit wordt YT_i genoemd.
    2. Het mengsel dat de strengen bevat die de positieve waarde voor de i-de variabele niet bevat. Ze bevat dus alle strengen waarbij de waarde voor de i-de variabele negatief is. We noemen deze YF_i.

  - Indien het aantal T's van het eerste mengsel > n/2 (d.w.z. dat er zeker meer dan de helft van de variabelen reeds de waard 1 hebben) voeg YT_i toe aan de verzameling ACCEPT. Zo niet voeg YT_i toe aan M
  - Indien het aantal F's van het tweede mengsel $\geq$ n/2 (d.w.z. dat het aantal positieve variabelen in de bitstring zeker niet meer dan de helft kan zijn) voeg YF_i toe aan DECLINE. Zo niet voeg YF_i toe aan M.
  - verwijder Y uit M.

  Indien de verzameling M k elementen bevat zullen er 2^k nieuwe substanties ontwikkeld worden.

- **Stap 6:** Verhoog de teller i met 1 en herhaal stap 5 zolang $i \leq n$. Indien $i > n$ zal de verzameling M leeg zijn en zullen alle mogelijke combinaties van de input streng overlopen zijn.
- **Stap 7:** Giet alle substanties van de verzameling ACCEPT samen in één testbuisje en noem dat 'Accept'. Giet alle substanties van de verzameling DECLINE in één tesbuisje en noem dit 'Decline'.
- **Stap 8:** Voer een gelelektroforese uit op beide substanties. In één van de twee mengsels zal DNA zitten. Indien dit het 'Accept-mengsel' is, is de uitkomst van het probleem positief, anders negatief.

Het algoritme voor n = 3 (zie figuur 4.4) [27]

- **Stap 1:** Bepaal de verschillende DNA input strengen (zie punt 4.3).
- **Stap 2:** Voer PCR uit zodat de input voldoende aanwezig is in een stabiele omgeving.
- **Stap 3:** Voer een ligase proces uit zodat de volledige strengen gevormd worden die dan de input representeren voor de volledige bitstring.
- **Stap 4:** Voer een magnetic beads separatie uit met als probe het complement van de positieve waarde van de eerste variabele x_1. Dit resulteert in twee nieuwe substanties.
    1. Het mengsel dat de strengen bevat met de positieve waarde voor de eerste variabele, genoemd T1.
    2. Het mengsel dat de strengen bevat die de positieve waarde voor de eerste variabele niet bevat. Ze bevat dus alle strengen waarbij de waarde voor de eerste variabele negatief is. We noemen deze F1.

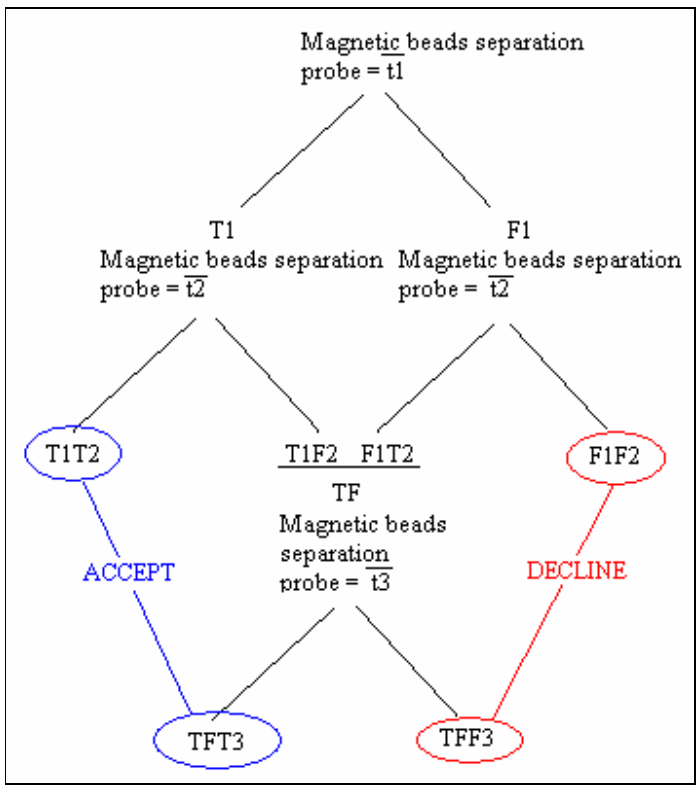
    Voeg beide resultaten toe aan de verzameling M.
- **Stap 5:**
    1. Neem het mengsel T1 en voer magnetic beads separatie uit met als probe het complement van de positieve waarde voor de tweede variabele x_2.
    2. Neem het mengsel F1 en voer magnetic beads separatie uit met als probe het complement van de positieve waarde voor de tweede variabele x_2.

    Elk van de twee onderdelen zullen opnieuw in twee nieuwe substanties resulteren die analoog T2 en F2 genoemd zullen worden. Zo zijn er dan in totaal vier resultaten: het zijn de substanties T1T2, T1F2, F1T2 en F1F2. Het mengsel T1T2 heeft meer dan 1 T en kan dus reeds bij de verzameling ACCEPT toegevoegd worden. Ook F1F2 kan zo bij de verzameling DECLINE gezet worden. T1F2 en F1T2 voegen we terug toe aan M nadat T1 en F1 eruit verwijderd zijn.
- **Stap 6:** Substantie twee en drie kunnen beide samengenomen worden in één substantie omdat ze beide één van de twee variabelen op 'true' hebben staan. We hernoemen deze substantie tot TF. We voeren enkel op deze substantie nog een laatste magnetic beads separatie uit met als probe het complement van de

positieve waarde van de derde variabele. Opnieuw krijgen we twee resultaten analoog als in stap 4:

1. Het mengsel dat de strengen bevat met de positieve waarde voor de derde variabele. Deze noemen we nu TFT3 (voeg dit toe aan ACCEPT).
2. Het mengsel dat de strengen met de positieve waarde voor de derde variabele niet bevat. Deze noemen we nu TFF3 (Voeg dit toe aan DECLINE).

- **Stap 7:** We gieten de substanties van ACCEPT samen en die van DECLINE samen. Het eerste mengsel noemen we het 'Accept-mengsel', het tweede het 'Decline-mengsel'.
- **Stap 8:** Voer een gelelektroforese uit op beide resultaten. In één van de twee mengsels zal DNA zitten. Indien dit het 'Accept-mengsel' is, is de uitkomst van het probleem positief, anders negatief.



Figuur 4.4: *Grafische voorstelling van het algoritme voor drie variabelen. De rode en blauwe verbindingen zijn de twee uiteindelijke substanties. De substantie (accept of decline) waar iets in zit is de output.*

Zoals reeds eerder vermeld is dit experiment reeds uitgevoerd aan het Limburgs Universitair Centrum. Verdere details kunnen uitvoerig gevonden worden in [27]. Helaas kende dit experiment niet het gehoopte succes. Het falen gebeurde tijdens de uitvoer van stap 3 in het algoritme hierboven beschreven. De oorzaak daarvan zou de keuze van het DNA in stap 1 kunnen zijn. We zullen dit daarom grondiger

aanpakken en stap voor stap analyseren zodat we met grote waarschijnlijkheid kunnen stellen dat eventueel een nieuwe faling van het experiment niet aan de keuze van het DNA zal liggen. We zullen in dit proefwerk veel tijd besteden aan het design van de te gebruiken DNA code words of codewoorden.

Het algoritme dat hierboven beschreven wordt valt onder het (unrestricted) filter model (cfr. supra). De input wordt vermenigvuldigd via PCR en zal verder gefilterd worden. Het filteren zal de strengen gaan verdelen in verschillende categorieën (substanties) wat overeenkomt met de abstracte "*separate(T,S)*"-operatie. Uiteindelijk zal de input gefilterd worden tot twee delen (substanties) waarbij de verschillende substanties die bij elkaar horen samengebracht worden ("*Merge(T_1, …, T_n)*"). Via gelelectroforese wordt het uiteindelijke resultaat vervolgens bekend gemaakt ("*detect(T)*").

4.5 Code word design

Voor een DNA streng die een bitstring gaat representeren in het n-bit majority probleem, is het noodzakelijk deze zo te kiezen dat ze geen problemen veroorzaakt. Zoals reeds vermeld moeten er 4n strengen bepaald worden om alle input data te kunnen produceren. Deze zijn de waarden van de variabelen t_i, f_i, de stickers e_i, b_{i+1}, en de twee primers p_1 en p_2. Hun structuur zal net zoals bij Adleman van uiterst belang zijn. We zullen nu elke categorie bekijken en de vereisten beschrijven waaraan hun structuur moet voldoen.

4.5.1 De variabelen

Eerst en vooral worden de strengen bepaald die de waarden van de n variabelen gaan representeren. Alle t_i en f_i zullen aan bepaalde vereisten moeten voldoen opdat ze gebruikt kunnen worden gedurende het experiment en geen 'false positives' vormen.

1. Elke waarde voor elke variabele moet uniek bepalen om welke waarde het gaat. Het gevolg is dat de structuur van de waarden t_i en f_i (met $i \in [1,n]$) zoveel mogelijk van elkaar moeten verschillen. Dit is noodzakelijk om de magnetic beads separation uit te voeren. Elke probe zal enkel en alleen met zijn overeenkomstige waarde mogen binden. De Hamming constraint kan hier gebruikt worden als hulpmiddel om de strengen te bepalen die voldoen aan deze eigenschap. De afstand tussen twee strengen onderling zal dan voor alle mogelijke koppels zo groot mogelijk moeten zijn: $H([t_i + f_i],[t_j + f_j]) \geq x$; met $i,j \in [1,n]$, $i > j$ en x zo groot mogelijk.
2. Er moet ook rekening gehouden worden met het feit dat de streng van een waarde niet het exacte complement is van een andere. Ze zouden in een bepaalde toestand gaan binden waardoor ze onbruikbaar worden in verdere procedures. Ook hier moet dus op gelet worden. Dit kan opnieuw met de Hamming constraint berekend worden. Maar deze keer geldt dit voor elke

streng met het complement van de andere: $H([t_i + f_i],[t_j^C + f_j^C]) \geq x$ met met i,j $\in$ [1,n] en i > j en x zo groot mogelijk.

3. De 2n strengen moeten verder ook zo verschillend mogelijk zijn van de stickers. Anders zou het complement van de stickers ook gaan plakken aan een variabele wat uiteraard verboden is. We gebruiken opnieuw de Hamming constraint zodat $H([t_i + f_i],[e_j + b_j]) \geq x$ met met i $\in$ [1,n], j $\in$ [1, n-1] voor e en j $\in$ [2, n]voor b. Het resultaat x moet opnieuw zo groot mogelijk zijn.
4. Ook de stickers mogen geen complement zijn van één van de waarden, anders zouden ze een ongewenste binding tot gevolg kunnen hebben. De Hamming constraint voor dit deel ziet er als volgt uit: $H([t_i + f_i],[e_j^C + b_j^C]) \geq x$ met met i $\in$ [1,n], j $\in$ [1, n-1] voor e en j $\in$ [2, n] voor b, met x zo groot mogelijk.
5. Hetzelfde principe als in punt 3 moet ook gelden voor de twee gekozen primers. Eerst en vooral het verschil met de primers: $H([t_i + f_i],[p_j]) \geq x$ met i $\in$ [1,n], j $\in$ [1, 2], met x zo groot mogelijk.
6. Vervolgens eveneens hetzelfde naar analogie als in punt 4: $H([t_i + f_i],[p_j^C]) \geq x$ met i $\in$ [1,n], j $\in$ [1, 2], met x zo groot mogelijk.
7. Een laatste maar niet onbelangrijke vereiste is de structuur van elke waarde opdat ze geen secundaire structuur, zoals een hairpin (cfr. supra), zouden vormen. Dit komt overeen met de structure freeness van de streng. Deze eis kunnen we vertalen naar een reverse complement constraint waarbij de twee strengen dezelfde zijn: $H(y^R,y^C) \geq x$; met y een streng uit de verzameling $\{ t_1, f_1, t_2, f_2, \ldots, t_n, f_n\}$ en x zo groot mogelijk.

4.5.2 De stickers

We hebben hierboven gezien dat de stickers e_i en b_{i+1} met i $\in$ [1,n-1], reeds aan een aantal eigenschappen moeten voldoen t.o.v. de waarden van de variabelen. We zullen hier nog enkele beperkingen voor de stickers opstellen die daar onafhankelijk van zijn.

1. De stickers zullen hun complement gebruiken tijdens een ligase proces. Het is dan ook noodzakelijk dat ze allemaal een uniek complement hebben dat enkel met de overeenkomstige sticker zal hybridiseren. Dit kan bepaald worden door de uniciteit van de stickers onder elkaar te garanderen naar analogie van punt 1 in de vorige sectie. Net zoals daar zullen we de Hamming constraint als volgt gebruiken: voor elk koppel (z,y) z $\neq$ y uit de verzameling $\{ e_1, b_2, e_2, b_3, ..., e_i, b_{i+1, \ldots,} e_{n-1}, b_n \}$: $H(z,y) \geq x$ met x zo groot mogelijk.
2. Het is ook verboden dat één van de stickers een complement is van een andere opdat ze onderling niet zouden gaan binden. De volgende regel is hier dan van toepassing: voor elk koppel (z,y) z $\neq$ y uit de verzameling $\{ e_1, b_2, e_2, b_3, ..., e_i, b_{i+1, \ldots,} e_{n-1}, b_n \}$: $H(z,y^C) \geq x$ met x zo groot mogelijk.
3. Ook t.o.v. de primers hebben ze ‘een verantwoording' af te leggen. Ze moeten verschillen van de primers opdat het complement van een primer tijdens PCR niet zou hybridiseren met een sticker. Voor elke y uit de verzameling $\{ e_1, b_2, e_2, b_3, ..., e_i, b_{i+1, \ldots,} e_{n-1}, b_n \}$: $H(y,p_i) \geq x$; i $\in$ [1, 2] en x zo groot mogelijk.

4. Ook mag een sticker niet het complement zijn van één van de twee primers zodat ze niet ongewenst met elkaar zouden binden. Voor elke y uit de verzameling { e_1, b_2, e_2, b_3, ..., e_i, $b_{i+1, \ldots,}$ e_{n-1}, b_n }: $H(y,p_i^C) \geq x$; i Є [1, 2] en x zo groot mogelijk.
5. Net zoals in punt 7 in de vorige sectie zijn secundaire structuren in een sticker verboden. Naar analogie zal opnieuw de reverse complement constraint opgelegd worden aan elke sticker: $H(y^R,y^C) \geq x$; met y een streng uit de verzameling { e_1, b_2, e_2, b_3, ..., e_i, $b_{i+1, \ldots,}$ e_{n-1}, b_n } en x zo groot mogelijk.

4.5.3 De primers

Ook de primers zijn reeds meerdere keren voorgekomen in de vorige twee onderdelen maar ook zij hebben nog een aantal beperkingen. Opnieuw is de analogie naar de vorige delen duidelijk zichtbaar.

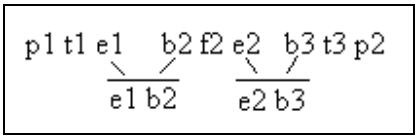
1. Van de twee primers p_1 en p_2, wordt verwacht dat ze uniek zijn ten opzichte van elkaar. D.w.z. dat ze zoveel mogelijk van elkaar moeten verschillen waarbij opnieuw de Hamming constraint gebruikt kan worden: $H(p_1,p_2) \geq x$ met x zo groot mogelijk.
2. Ook mogen de primers niet met elkaar binden: $H(p_1,p_2^C) \geq x$ met x zo groot mogelijk.
3. Uiteraard zijn opnieuw secundaire structuren niet toegelaten: $H(p_i^R,p_i^C) \geq x$; met i Є [1, 2] en x zo groot mogelijk.

4.5.4 De uiteindelijke code words

Om het experiment te kunnen starten moeten de ontworpen strengen gesynthetiseerd worden. We hebben gesproken van een ligase proces om verschillende delen achter elkaar te plakken zodat de gehele bitstring voorgesteld kan worden door één volledige DNA streng. Het is daarbij niet de bedoeling dat elk element apart zal moeten aangevraagd worden. Uiteindelijk volstaat het om n strengen DNA te synthetiseren per bitstring. Een voorbeeld maakt dit duidelijk:

$$101 = p_1\ t_1\ e_1\ b_2\ f_2\ e_2\ b_3\ t_3\ p_2 \qquad \text{voor } n = 3$$

De stickers werden ingevoerd omdat de volledige streng niet zou aangemaakt kunnen worden. Daarom wordt de streng opgedeeld in drie onderdelen $p_1t_1e_1$, $b_2f_2e_2$ en $b_3t_3p_2$. De stickers zullen dan in stap 3 van het algoritme de verschillende delen aan elkaar koppelen zoals voorgesteld in volgend voorbeeld:



Algemeen zullen er n onderdelen gesynthetiseerd moeten worden om de ‘bitstrings’ te kunnen representeren. Deze zijn dan eerst en vooral $p_1[t_1 + f_1]e_1$ en $b_n[t_n + f_n]p_2$ gevolgd door de delen ertussen: $b_i[t_i + f_i]e_i$, i Є [2,n-1]. Vervolgens zullen ook het complement van de stickers e_ib_i en het complement van de primers, p_1 en p_2 nodig zijn voor het algoritme.

4.5.5 De smelttemperatuur

Wanneer we de derde stap willen toepassen in het hierboven beschreven algoritme zal de temperatuur waarop dit uitgevoerd wordt een belangrijke rol spelen. Het binden van de stickers zoals te zien is in figuur 4.3 zou het beste tegelijk gebeuren op alle lokaties. We zullen daarom nog een volgende beperking opleggen aan het design van de codewoorden. De smelttemperatuur van e_ib_{i+1} (i Є [1, n-1]) moet zo dicht mogelijk bij elkaar liggen, in het beste geval zelfs hetzelfde zijn.

4.6 Conclusie

We weten nu aan welke voorwaarden de DNA strengen moeten voldoen opdat ze gebruikt kunnen worden voor het experiment van het n-bit majority probleem. We zullen dus 4n strengen DNA moeten bepalen uit een verzameling van mogelijkheden. Deze keuze is zo goed als onmogelijk zomaar te bepalen. Een gepast zoekalgoritme zou hier dus op zijn plaats zijn. Er bestaan hiervoor verschillende methoden alsook heuristieken om deze strengen te specifiëren. Eén daarvan is een genetisch algoritme. In de volgende hoofdstukken zullen het ontwerp van dergelijk algoritme alsook de voor- en nadelen ervan uitgebreid besproken worden.

Hoofdstuk V: Genetische algoritme

5.1 Korte geschiedenis

In de natuur ontwikkelen zich organismen die in een bepaalde omgeving leren overleven. De sterksten overleven en planten zich voort, de zwakkere organismen sterven af. Zelfs indien de omgeving gestadig verandert zullen de afstammelingen mee evolueren om ook in de nieuwe omgeving te kunnen overleven.
In de jaren ‘50 hebben verschillende wetenschappers zich op dit systeem gebaseerd om zelf een evolutionair systeem te ontwikkelen in de wereld van de informatica. Met de kennis van de evolutie in de natuur is men gaan proberen strategieën en algoritmen te ontwikkelen. In sommige gevallen werd de computer gebruikt om evoluties te berekenen, maar de evolutie theorie werd des te meer gebruikt als tool om bepaalde algoritmen te ontwikkelen.

Een ‘genetic algortihm’ of genetisch algoritme (GA) is zo een algoritme dat door John Holland in de jaren ‘60 werd ontwikkeld [39]. Het gebruikt dus het idee dat mechanismen of organismen evolueren, generatie na generatie. Het basis idee achter dit algoritme is dan ook een populatie van gegevens te laten evolueren via natuurlijke selectie tot een nieuwe populatie die zich zal hebben aangepast aan een eventuele nieuwe omgeving. Hierbij wordt er gebruik gemaakt van organische processen zoals crossover en mutatie.
Jarenlang zijn onderzoekers reeds bezig genetische algoritmen te beschrijven en het nut ervan aan te tonen. Dit soort algoritme is gestadig maar zeker zijn opmars aan het maken in de wereld van de informatica. Genetische algoritmen zijn ondertussen immens opgegroeid in de ontwikkelingen van de ‘artificial intelligence systems’. Een genetisch algoritme heeft net zoals andere heuristieken problemen. Het kan niet garanderen dat er altijd naar een globaal optimum geconvergeerd wordt en de mogelijkheid bestaat dat het vastgeraakt in een lokaal optimum. In dit hoofdstuk zullen zowel voor- als nadelen en de mogelijke problemen grondig besproken worden.

In dit proefschrift zullen we dieper ingaan op het gebruik en het nut van de genetische algoritmen, alsook het ontwikkelen van een genetisch algoritme in functie van het n-bit majority probleem.

5.2 Evolutionair programmeren

'Evolutionary computation' of evolutionair programmeren [40][41] is het simuleren van evolutie met een computer om te dienen als algoritme voor het zoeken van globale optima voor een probleem. Evolutie gebruiken om problemen op te lossen klinkt in eerste instantie vreemd. Toch kent deze theorie een aantal eigenschappen die bij het ontwikkelen van algoritmen zeer nuttig, soms zelfs noodzakelijk zijn.
Eerst en vooral blijkt dat met evolutionair programmeren een goed heuristisch algoritme gevormd kan worden. Het zoeken en berekenen van een oplossing doorheen een grote hoeveelheid oplossingen kan met evolutionair programmeren leiden tot een goede benadering van de oplossing.
De evolutie staat gekend zich aan te passen aan de omgeving, zelfs als die verandert doorheen de tijd. Vele problemen in de informatica vereisen algoritmen die zich bij een veranderende omgeving mede aanpassen en toch hetzelfde blijven functioneren. Denk maar aan het programmeren van een robot die in verschillende omgevingen toch zijn taak moet kunnen volbrengen.
De veranderingen van de organismen tijdens de evolutie en de mogelijkheid van het muteren van organismen, leiden zeer vaak tot nieuwe en verrassende 'geboortes in de natuur'. Aangezien ontdekkingen een belangrijk onderdeel zijn in onze zoektocht doorheen de wetenschap tracht men vaak computerprogramma's te schrijven die snel iets nieuw en origineel proberen te ontwikkelen. Evolutie theorie kan dus duidelijk een handje helpen in het ontwikkelen van dergelijke programma's.
Tenslotte wordt evolutionair programmeren reeds veel gebruikt in het ontwikkelen van 'Artificial Intelligence' of artificiële intelligentie. Het vinden van een oplossing voor een bepaald probleem kan door gebruik van evolutie benaderd worden zonder alle mogelijke oplossingen daarvoor te moeten onderzoeken. Het veranderen van omgeving en de mogelijkheid zich hieraan aan te passen kan leiden tot het leren van oplossingen. Op dit gebied wordt nog steeds zeer veel onderzoek verricht.
Genetische algoritmen worden ook omschreven als zijnde impliciet parallel. Elke stap in het algoritme omvat het produceren van een nieuwe populatie uit de vorige. Hierbij worden afstammelingen bepaald als 'kinderen' van twee leden van de huidige populatie. Het bepalen van de 'kinderen' zou in parallel kunnen verlopen.
Deze eigenschappen kunnen dus een handigheid zijn bij het ontwikkelen van algoritmen. Het principe van genetische algoritmen wordt reeds veel gebruikt in verschillende domeinen [42]:

- Zoekalgoritme: zoals reeds meerdere keren aangegeven, worden genetische algoritmen vaak gebruikt als heuristisch zoekalgoritme.
- Optimalisatie
- Machine learning: genetische algoritmen kunnen evoluties voorspellen of evoluties gebruiken om systemen te leren.
- Economie: genetische algoritmen werden reeds gebruikt om bepaalde aspecten in de economie te modelleren of te voorspellen. De resultaten werden dan gebruikt om belangrijke economische beslissingen te nemen.
- …

Er zijn dus tal van domeinen waar het genetisch algoritme als hulpmiddel kan dienen. Het is niet moeilijk om zelf een domein te bedenken waarbij deze techniek wel eens handig zou zijn.

5.3 Genetische algoritmen en zoekalgoritmen

Hieronder zal dieper ingegaan worden op het gebruik van genetische algoritmen als traditioneel zoekalgoritme. We kunnen het zoekalgoritme opdelen in drie categorieën [40]:

- het zoeken naar een element in een grote hoeveelheid data (zie punt 5.3.1)
- het zoeken van een pad naar een bepaalde oplossing voor een probleem (zie punt 5.3.2)
- het zoeken van een oplossing (zie punt 5.3.3)

5.3.1 Zoeken naar gestockeerde data

Wanneer een grote hoeveelheid gegevens opgeslagen zijn in een geheugen, is het zoeken naar een bepaald gegeven niet zo moeilijk. Er zijn dan ook al vele algoritmen ontwikkeld en beschreven voor deze taak. De snelheid waarop het gegeven gevonden wordt speelt hier een belangrijke rol want alle gegevens afgaan in een grote database is zeer inefficiënt (zeker als de data gesorteerd is). Een genetisch algoritme zou hier kunnen worden gebruikt om de ‘search space’ of zoekruimte te verkleinen en zo sneller het resultaat te lokaliseren. In deze vorm van zoeken zal een genetisch algoritme niet altijd beter zijn dan een ander zoekalgoritme en zal daardoor niet vaak in dit domein gebruikt worden.

5.3.2 Het zoeken naar een pad dat leidt tot een bepaald doel

Algoritmen in dit domein zullen nagaan hoe men best tot een bepaalde toestand/gegeven kan komen. Indien een genetisch algoritme gebruikt zou worden zou aan de hand van de veranderingen in de gegenereerde populaties de weg tot een oplossing kunnen bepaald worden. Ook hier zijn er reeds tal van algoritmen ontwikkeld die vaak een betere aanpak beschrijven dan het genetisch algoritme.

5.3.3 Het zoeken naar oplossingen

Het doel hier is een oplossing te zoeken voor een probleem waarvoor er een enorme keuze is aan kandidaat oplossingen. De verzameling kandidaat oplossingen (search space) kan zelfs oneindig groot zijn. Bij dergelijke grote search spaces zal een genetisch algoritme tot een zelfde of tot een beter resultaat leiden dan andere algoritmen. De genetische algoritmen zullen efficiënt blijken bij het oplossen van dergelijke problemen.

5.3.4 Het gebruik

Het gebruik van een genetisch algoritme is niet altijd de juiste keuze. Wanneer zou nu voor een genetisch algoritme moeten gekozen worden? Op deze vraag is helaas zeer moeilijk een antwoord te geven. Enkel uitgaan van de zoekruimte zou immers een zeer beperkt beeld geven van de efficiëntie van een algoritme. Toch zijn er wel enkele richtlijnen indien we weten hoe de zoekruimte eruit ziet, die een goede keuze bevorderen maar niet kunnen garanderen:

- Indien de zoekruimte niet zo groot blijkt te zijn zal een standaard zoekalgoritme gekozen worden boven een genetisch algoritme. Deze zal dan altijd de juiste oplossing geven terwijl een genetisch algoritme altijd tracht te convergeren naar een globaal optimum.
- Indien de zoekruimte gekend is en niet verandert of de gegevens erin zijn 'smooth', zal een reeds bestaand algoritme vaak betere resultaten geven. Een voorbeeld van 'smooth data' zijn data die bepaald worden door een (lokaal) continue functie.
- Indien de zoekruimte bekend is en er zijn reeds efficiënte algoritme voor ontwikkeld dan zal een genetisch algoritme dit hoogstwaarschijnlijk niet overtreffen. Het is bijgevolg aan te raden het gekende algoritme te gebruiken.

Deze richtlijnen suggereren niet dat alle andere problemen beter met een genetisch algoritme opgelost kunnen worden. De efficiëntie van een genetisch algoritme wordt immers door meer parameters dan alleen de zoekruimte bepaald. Dit zal duidelijk worden wanneer overlopen zal worden hoe de basis van een genetisch algoritme er uiteindelijk zal uitzien.

5.4 De opbouw van een genetisch algoritme

Nu het nut van een genetisch algoritme aangetoond is, zal gekeken worden naar de structuur van een standaard genetisch algoritme. John Holland's genetisch algoritme wordt nog altijd als basis genomen om voor eender welk probleem een genetisch algoritme te ontwikkelen. Het opbouwen van dergelijk algoritme kan opgedeeld worden in vier initiële stappen.

5.4.1 Het coderen van de kandidaat oplossingen

Wanneer men wil beginnen met het implementeren van een genetisch algoritme is het belangrijk op voorhand te beslissen hoe de oplossingen gecodeerd zullen worden. D.w.z. dat men een code gaat gebruiken om de kandidaat oplossingen (de elementen van de search space) voor te stellen zodat dit door de computer/programmeertaal leesbaar is. Een ander belangrijk punt is de lengte dat zo een voorstelling inneemt.

5.4.1.1 Soorten codering ('encoding')

1) Binary encoding

De meest eenvoudige en meest gebruikte codering is een bitstring van vaste lengte. Ook de orde van de bits wordt vastgelegd. Elke bitstring zal dan een element uit de verzameling voorstellen. Er bestaan varianten op de binary encodings zoals gray encoding [43] en diploid binary encoding. Hier wordt niet verder op ingegaan.

2) Many-character and Real-valued encodings

De binaire codering is het meest eenvoudige maar voor vele applicaties is het vaak even gemakkelijk strengen te gebruiken van karakters of 'floating point' getallen. Op die manier wordt de representatie in dergelijke gevallen overzichtelijker en meer 'naturel' zonder echt nieuwe problemen te creëren.

3) Tree encodings

Deze codering gaat de elementen van de populatie voorstellen in een boomstructuur waardoor de search space open-ended wordt. Dit wil zeggen dat elke grootte van boom een mogelijke nieuwe populatie is. De creatie van dergelijke bomen (nieuwe populatie) vormt geen probleem maar er zijn dan weer wel enkele nadelen aan verbonden. Een nieuwe boom kan niet alleen enorm groeien, ook de structuur van de boom kan soms ongecontrolleerd opgebouwd worden, d.w.z. dat de resulterende structuur van de boom enkele nuttige eigenschappen kan verliezen. Zo kan de hiërarchie te complex worden of zelfs gedeeltelijk verdwijnen, alsook de mogelijkheid tot vereenvoudiging van de boom. Op het gebruik van tree encodings was en is nog onderzoek nodig ten opzichte van andere coderingen [44][45][46].

Deze drie vormen van codering zal men in de meeste genetische algoritmes terugvinden. De juiste keuze van codering is niet vanzelfsprekend en in de meeste gevallen zelfs onmogelijk op voorhand te bepalen. De kennis van problemen waar genetische algoritmes voor gebruikt worden is op voorhand in vele gevallen zelfs te beperkt om een juiste keuze van codering te kunnen bepalen. Ook hier werd reeds grondig onderzoek naar gedaan maar de volgende basisregel blijft vaak nog van toepassing: kies de codering die het meest natuurlijk overkomt voor de representatie van de populatie [47].

5.4.1.2 Aanpassingsmogelijkheden van de codering

Zoals hierboven werd aangehaald is het vaak onmogelijk een juiste keuze te maken van codering van kandidaat oplossingen. Men is dan op voorhand niet volledig op de hoogte over hoe het probleem voorgesteld zou moeten worden. Men zou zelfs zo ver kunnen gaan dat men een ander genetisch algoritme gaat ontwikkelen die berekent welke codering het meest geschikt zou zijn. Aangezien dit het probleem enkel verplaatst, wordt er bij het ontwikkelen van een genetisch algoritme rekening

gehouden met eventuele aanpassing van de representatie. Het is dan de bedoeling dat de gebruikte codering zich gedurende de 'evolutie' in het programma gaat aanpassen aan eventuele veranderingen in de omgeving van de kandidaat oplossingen. Een tweede reden waarom de codering in staat zou moeten zijn zich aan te passen, is de lengte van de codering. Wanneer bijvoorbeeld met bitstrings gewerkt wordt, is de lengte van strengen op voorhand vastgelegd. Aangezien men vaak niet weet hoe lang een representatie zal zijn, zou het dus interessant zijn dat de lengte van de bitstrings kan variëren. Dergelijke aanpassing zou ook tot de mogelijkheden van een codering moeten horen. De mogelijkheid om de geheugen capaciteit voor een representatie te laten variëren, leidt tot een meer open-ended evolutie volgens Lindgren (1992). Het gebruik van tree encodings voorziet deze mogelijkheden reeds maar zoals hierboven vermeld, moet er dan ook met andere zaken rekening gehouden worden.
Een ander probleem bij strengen dat toch moet vermeld worden, is het "linkage problem". Een streng kan tijdens de evolutie reeds één of meer 'substrings' of deelstrengen hebben die steeds tot een goed resultaat leiden. Het is dan ook wenselijk dat deze 'substrings' niet verstoord worden doorheen de verdere evolutie. Helaas is het vaak onmogelijk op voorhand te bepalen welke deelstrengen behouden zouden moeten blijven. Door dit gegeven tijdens de evolutie te laten aanpassen kan een genetisch algoritme weer verbeterd worden.
Hieronder worden kort enkele mogelijke aanpassingsmethoden besproken.

1) Inversion

J. Holland beschreef 'inversion' of inversie reeds in het originele concept van zijn genetisch algoritme. Het gaat het linkage probleem zo goed mogelijk proberen op te lossen. Het idee achter inversion is welgekend in de genetische wetenschap [39][48]. Het doel van inversion is de 'bitstring' of deelstreng te laten aanpassen zonder dat de semantische waarde van de streng verandert. Hiervoor werd een nieuwe notatie ingevoerd om een streng voor te stellen. Aan elk element wordt een index gekoppeld dat zijn lokatie in de streng weergeeft. Op die manier kunnen de elementen verwisseld worden terwijl hun oorspronkelijke plaats en dus semantische waarde behouden blijven alsook de waarde van de geschiktheid (cfr. supra). Een voorbeeld zal dit verhelderen. Beschouw de 10-bit streng 1000010100 dan zal deze als volgt genoteerd worden:

(1,1) (2,0) (3,0) (4,0) (5,0) (6,1) (7,0) (8,1) (9,0) (10,0)

Wanneer inversion op een streng wordt toegepast worden twee lokaties in de streng genomen en de orde van de elementen ertussen wordt geïnverteerd. Stel: we nemen plaats 3 en 7 dan leidt dit tot het volgende resultaat:

(1,1) (2,0) (7,0) (6,1) (5,0) (4,0) (3,0) (8,1) (9,0) (10,0)

Het resultaat van inversion is een nieuwe streng van bits die volgens een gegeven functie beter in staat zou zijn te overleven in de omgeving. Toch rijst er een nieuw probleem op. Dit wordt onmiddellijk duidelijk met een voorbeeld.

Neem twee 10-bitstrings 0100001010 en 1010101010. In de notatie van J. Holland worden de eerste dan na inversie tussen plaatsen 3 en 8:

(1,0) (2,1) (8,1) (7,1)|(6,0) (5,0) (4,0) (3,0) (9,1) (10,0)
en
(1,1) (2,0) (3,1) (4,0)|(5,1) (6,0) (7,1) (8,0) (9,1) (10,0)

wanneer we nu een crossover doen met deze twee strengen als ‘ouders’ na bit 4 levert dit volgende ‘kinderen’ op:

(1,0) (2,1) (**8**,1) (**7**,1) (5,1) (6,0) (**7**,1) (**8**,0) (9,1) (10,0)
en
(1,1) (2,0) (**3**,1) (**4**,0) (6,0) (5,0) (**4**,0) (**3**,0) (9,1) (10,0)

Het is duidelijk dat de twee ‘kinderen’ geen volwaardige bitstring vormen. Er ontbreken bij beide strengen bits die vervangen zijn door bits die reeds in de streng aanwezig zijn. Dit is zeker niet toegestaan en Holland had hier twee mogelijke oplossingen voor: ofwel crossover toelaten enkel indien beide ouders dezelfde ordening hebben, ofwel één van de twee ouders herordenen naar het voorbeeld van de andere gedurende de crossover.
Inversion werd later niet vaak meer teruggevonden in genetische algoritmes omdat het weinig invloed had op de performantie.

2) Evolving crossover hot spots [49]

Deze benadering gaat niet de bits van plaats veranderen maar bepaalt de plaats(en) waar een crossover uitgevoerd mag worden. Deze extra informatie kan bijvoorbeeld bijgehouden worden in een andere streng die aan een bitstring gekoppeld wordt. Een mogelijke streng zou zijn: een bitstring waarbij een 1 voor een mogelijke crossover plaats staat (“hot spot”) en een 0 die crossover verbiedt op die plaats. Bij crossover zullen de “hot spots” mee geërfd worden en bij mutatie moet de gekoppelde streng mee aangepast worden. Een voorbeeld ter verduidelijking:

Neem twee 10-bit strengen en hun bijhorende informatie:

0100001111 - 0000100010 genoteerd als 01000!0111!1
en
1111000100 - 0000001000 genoteerd als 1111000!100
(Een ! staat voor een hot spot)

Na crossover resulteert dit in de volgende ‘kinderen’:

01000!00!11!0
en
1111001101

3) Messy genetische algoritmes

De bedoeling bij dit principe is het opbouwen van strengen uit kleinere blokjes. Het resultaat moet een streng opleveren die een zo goed mogelijke oplossing voorstelt. Voor elk blokje op zich dat zo een streng opbouwt, is reeds aangetoond dat het helpt bij de bouw. Dergelijke blokjes worden ook wel 'building blocks' genoemd. Dit principe is niet onbekend. Gedurende de evolutie zijn steeds grotere organismen ontstaan uit kleinere organismen die sterk genoeg bleken om te overleven. Dit wordt hier ook toegepast. Voor een voorbeeld en uitgebreidere informatie wordt verwezen naar [50][51].

Verdere nuttige infomatie over adaptatie van een codering kan men vinden in [61][62].

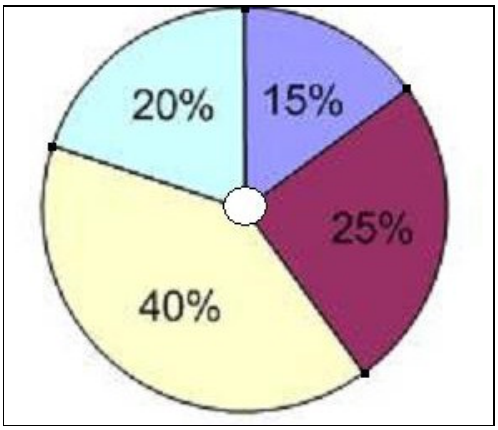
5.4.2 Selectie Methoden

Een tweede stap in het ontwikkelen van een genetisch algoritme is het selecteren van de 'ouders'. Welke twee chromosomen (elementen van de populatie) zullen worden gebruikt voor het ontwikkelen van 'kinderen' met crossover en de eventuele mutatie? Met andere woorden: welke elementen zullen voor de volgende generatie zorgen. Het is daarbij de bedoeling dat de nieuwe generatie elementen gaat bevatten die beter zijn dan de huidige populatie en dus een betere benadering vormen van de optimale kandidaat oplossing.
De keuze is niet vanzelfsprekend. De twee beste chromosomen in de populatie nemen zou aanvankelijk logisch klinken opdat ze samen nog betere afstammelingen zouden geven. Dit is zeker niet altijd het geval. Het is wel de bedoeling dat de betere elementen van de populatie genomen worden om afstammelingen te creëren maar toch moet de diversiteit van de elementen gegarandeerd blijven. Wanneer selectie gebeurt op basis van enkel de beste chromosomen bestaat de kans immers dat deze de populatie gaan overheersen. Hierdoor zal de zoekruimte van kandidaat oplossingen grotendeels beperkt worden waardoor mogelijke kandidaten die er buiten vallen niet meer bereikbaar zullen zijn. Dit probleem wordt ook wel 'premature convergence' genoemd [52]. De evolutie gebeurt te snel waardoor de zoekruimte alsmaar meer en meer beperkt zal worden. Indien bij selectie alle elementen in rekening worden gebracht en de kans kleiner wordt een betere generatie te ontwikkelen, zal de evolutie te traag verlopen en wordt de efficiëntie van het algoritme grondig geschaad.
Het is dus de bedoeling een balans te vinden in de nieuwe generatie tussen groeiende evolutie en de blijvende diversiteit van de elementen. Er bestaan methoden om dit te verwezenlijken maar opnieuw zijn die probleem afhankelijk. Er bestaat geen vaste regel die op voorhand zegt welke methode optimaal is voor welk probleem. Hieronder zullen weer enkele methoden besproken worden zodat er een

beeld kan gevormd worden van wat het nut is van een goede selectie in de ontwikkeling van een genetisch algoritme.

5.4.2.1 Roulette wiel / rad van fortuin

Bij deze methode wordt aan elke chromosoom een kans gegeven die bepaalt of het als ‘ouder’ gebruikt zal worden of niet. De elementen die beter zijn dan anderen zullen een hogere kans toegewezen krijgen. Neem een revolver en beschouw de elementen als kogels van verschillende grootte. Speel dan een aantal keer Russische roulette zodat het chromosoom dat overeenkomt met de afgevuurde kogel gekozen wordt. Je kan dit ook zien als het ‘rad van fortuin’ waarbij de grootte van elk vakje bepaald wordt door de kans dat een chromosoom heeft. Spin het rad zoveel keer als er chromosomen in de populatie zitten. Bij elke stop: neem de chromosoom die met dat vakje overeenkomt. Een voorbeeld wordt gegeven in figuur 5.1: een populatie van vier elementen elk met hun kans berekend op de geschiktheid (‘fitness’) van elk element t.o.v. elkaar. Wanneer het rad dan gedraaid wordt zal het goede chromosoom (met kans 40%) dus de grootste kans hebben om gekozen te worden.



Figuur 5.1: *Een roulette wiel van vier elementen.*

Het resultaat van roulette wiel is dat goede elementen vaak gekozen zullen worden terwijl de diversiteit bewaard blijft omdat ook mindere goede toch nog gekozen kunnen worden en een rol spelen in de volgende generatie.
Geheel probleemloos is deze methode natuurlijk niet, mede omdat statistiek zijn intrede doet. In een ‘worst case scenario’ kan het zijn dat enkel de mindere elementen instaan voor de nieuwe populatie die daardoor niet beter kan worden. Bij kleinere populaties neemt de kans op een ‘worst case scenario’ toe. Bij dit principe moet tevens rekening gehouden worden met de keuze van elke spin. Indien twee keer na elkaar hetzelfde element gekozen wordt en deze worden dan als ‘ouders’ genomen zal dit geen nieuwe afstammelingen opleveren met enkel crossover. Dit probleem is uiteraard gemakkelijk op te lossen: spin opnieuw telkens éénzelfde ‘ouder’ gekozen wordt.

Het tegenovergestelde probleem kan zich ook voordoen. Wanneer alleen de betere elementen telkens gekozen worden krijgen we weer het probleem van ‘premature convergence’ (cfr. supra).
Deze selectie methode is dus afhankelijk van de variantie in de geschiktheid van de elementen van een populatie en bepaalt dus de evolutie. Aangezien op voorhand niet kan voorspeld worden hoe deze variantie zich zal gedragen tijdens de evolutie, is het niet te bewijzen dat roulette wiel al dan niet geschikt is voor een bepaald probleem. Toch wordt deze methode vaak gebruikt omdat ze toch een goede balans geeft voor de selectie en ook omdat ze eenvoudig te begrijpen is.

5.4.2.2 Elitism

Deze simpele methode staat toe chromosomen van ene populatie door te geven aan de volgende generatie. Door de beste(n) niet te laten deelnemen aan crossover maar te laten overleven zullen ze niet verdwijnen of ‘verminkt’ worden via crossover en/of mutatie. Deze toepassing garandeert dat de maximale geschiktheid niet zal dalen. Dit principe wordt vaak gebruikt samen met een andere selectie methode. Onderzoekers hebben kunnen vaststellen dat het gebruik van ‘elitism’ de performantie van een genetisch algoritme ten goede komt [53] [54].

5.4.2.3 Rank selection

‘Rank selection’ is een methode die vooral snelle convergentie tegenwerkt om zo de diversiteit van de chromosomen te behouden. Het voordeel van deze methode is dat ze fitness-onafhankelijk werkt in tegenstelling tot roulette wiel. Elk element van de populatie krijgt een rangnummer op basis van de fitness. Zo worden de verschillen tussen de fitness van de verscheidene elementen verborgen. Indien een bepaald element een zeer hoge fitness heeft zal deze informatie dan weer niet gebruikt worden wat in sommige gevallen een nadeel kan zijn. De fitness of geschiktheid van een element beschrijft hoe goed het element (zie punt 5.4.5) het optimum benadert.

5.4.2.4 Tournament selection

‘Tournament selection’ is te vergelijken met rank selection wat betreft de voor- en nadelen. Dit principe is echter efficiënter en zou in parallelle implementaties gebruikt kunnen worden. Bij rank selection moet de populatie gesorteerd worden op basis van de rang van elk element, wat een tijdrovende bezigheid kan worden voor een programma. Tournament selection werkt anders. Het kiest telkens twee elementen uit de populatie en een random of willekeurig getal tussen 0 en 1. Indien het getal kleiner is dan een parameter wordt de meest geschikte van de twee chromosomen gekozen, anders de andere. Deze parameter kan ingesteld worden als een getal tussen 0.5 en 1 en zal de verhoogde kans aangeven waarbij het meest geschikte chromosoom gekozen zal worden. De random elementen worden achteraf terug in de populatie gezet zodat ze opnieuw gekozen kunnen worden voor een crossover.

5.4.2.5 Steady-state selection

De vorige selectie methoden behoren allen tot eenzelfde categorie waarbij ze allen trachten een (bijna) volledig nieuwe populatie te creëren die beter is dan de huidige. Deze principes worden 'generational methods' genoemd. Steady-state selection is een andere categorie. Enkel de zwakkeren uit de populatie zullen zo vervangen worden door crossover en mutatie, gebruik makend van de beteren in de populatie. Ook deze methode wordt vaak gebruikt, vooral bij problemen waarbij incrementeel leren van belang is [55].
Een vergelijking met de andere 'generational' selectie methodes kan men vinden in o.a. [56].

5.4.3 Genetische operaties

De derde stap, na het kiezen van de elementen die voor de 'nakomelingen' gaan zorgen, zijn de operaties die dit gaan berekenen. Deze operaties zijn gebaseerd op hun werking in de natuur. Ze gaan namelijk simuleren hoe twee chromosomen in de natuur "paren" en zo nieuwe chromosomen produceren. De twee operaties die daarbij het belangrijkste zijn en ook altijd in het genetisch algoritme verwerkt worden zijn crossover en mutatie.

5.4.3.1 Crossover

Er zijn twee vormen van crossover: 'single-point' en 'multi-point' crossover. Hier zal single-point crossover besproken worden met een voorbeeld zoals bij de bitstring codering. De twee chromosomen zullen op één plaats gesplitst worden en vervolgens 'fusioneren'. Multi-point crossover werkt op dezelfde manier maar dan worden de chromosomen op meerdere plaatsen gesplitst in plaats van op één enkele lokatie. De twee gekozen chromosomen zullen met elkaar paren op de manier zoals uitgelegd in volgend voorbeeld:

> Neem twee 20-bit strengen:
>
> 01011111000101011110
> en
> **10101010111110000000**

De volgende stap is een lokatie in de streng bepalen groter dan 0 en kleiner dan N, de lengte van de strengen (die hier dus 20 bedraagt).
We kiezen plaats 8. Dan zullen de strengen na de 8^{ste} bit gesplitst worden en vervolgens 'mergen'.

> 01011111 - 000101011110
> en
> **10101010 - 111110000000**

Na het ‘fusioneren’ levert dit de volgende twee afstammelingen:

0101111**111110000000**
en
10101010000101011110

De plaats waar de chromosomen gesplitst worden, bepaalt of het resultaat al dan niet betere chromosomen oplevert. De mogelijkheid bestaat dat dit slechte resultaten oplevert in de nieuwe populatie en daarbij een goed resultaat doet verdwijnen. Dit is een nadeel van crossover. Zoals hoger vermeld in punt 5.4.2.1, zijn er manieren om dit te verbeteren.
Single-point crossover is beperkt in het leveren van nieuwe elementen. Er zijn immers zeer veel bitstrings die niet gevormd kunnen worden door single-point crossover. Zo zal de streng 0101-101000010-0000000 niet gevormd kunnen worden door single-point crossover in het voorbeeld hierboven. Two-point crossover biedt een oplossing maar opnieuw kan hetzelfde probleem zich voordoen. Hier bestaat uiteindelijk geen algemene oplossing voor. Een geparametriseerde crossover wordt voorgesteld in [57]. Dit houdt in dat elke positie in een chromosoom een potentiële kans heeft verwisseld te worden met die uit het andere chromosoom. Dit zou een goede oplossing zijn om zoveel mogelijk van de kandidaat oplossingen te bereiken. Opnieuw bestaat dan de kans dat goede deelstrengen afgebroken zullen worden. Een balans tussen enerzijds het bijhouden van een zo ruim mogelijke zoekruimte tegenover het bewaren van goede deelstrengen moet gezocht worden. Er zijn reeds vele papers gepubliceerd die het nut van elke crossover algemeen en specifiek proberen te vergelijken.
Finaal kan men zeggen dat het nog niet duidelijk is welke crossover gebruikt moet worden voor welk probleem.

5.4.3.2 Mutatie

Crossover zal ervoor zorgen dat een grote variatie van elementen uit de zoekruimte onderzocht worden. Het kan zijn dat na enige tijd door de keuze van selectie er geen nieuwe elementen geconstrueerd worden of dat er op een bepaalde deelverzameling van de search space gefixeerd wordt. Hierdoor zouden potentiële kandidaat oplossingen daarbuiten niet meer onderzocht worden. Door mutatie toe te voegen wordt er plots een nieuwe variatie gebracht in de populatie en wordt dit probleem aangepakt.
Mutatie werkt net zoals in de natuur. Hier zal het bij bijvoorbeeld de bitstring een ‘flop’ uitvoeren op een bepaalde plaats in de streng. Net zoals bij crossover zal mutatie een verstoring kunnen veroorzaken op plaatsen waar dit beter niet gebeurt. Toch is mutatie een niet weg te denken onderdeel in een genetisch algoritme. Het nut van mutatie werd reeds meermaals aangetoond [58][59].
Andere operaties zijn mogelijk en zullen meestal gevonden worden in de natuur zelf. De problemen die de operaties met zich meebrengen vraagt opnieuw naar het aanpassen van de operaties gedurende de uitvoering [60].

5.4.4 Parameter settings

Een laatste belangrijke beslissing moet genomen worden over de parameters in het genetisch algoritme. Er zijn verschillende mogelijke parameters afhankelijk van het soort implementatie. Er zijn vier basis parameters die bijna altijd in het genetisch algoritme voorkomen:

1 De grootte van de populatie: uit hoeveel elementen zal een generatie bestaan? Dit moet weer een balans zijn tussen niet te veel en niet te weinig elementen. Te veel elementen zal het algoritme tijdrovend en intensief maken, te weinig elementen zal dan weer het aantal mogelijk te onderzoeken kandidaat oplossingen verkleinen die door crossover en mutatie geproduceerd worden.
2 De crossover rate: een statistische parameter die de kans aangeeft waarop crossover moet plaatsvinden.
3 De mutation rate: een statistische parameter die de kans aangeeft waarop mutatie moet gebeuren.
4 Een stop parameter: om het algoritme niet oneindig te laten doorlopen moet er op één of andere manier bepaald worden wanneer het algoritme moet stoppen en het globaal optimum zou bereikt hebben. Een mogelijkheid is het aantal iteraties vastleggen, dus een bovengrens op het aantal te creëren generaties. Een andere mogelijkheid is het algoritme te stoppen wanneer een minimaal optimum bereikt zou zijn. Uiteraard moet in het tweede geval geweten zijn wat men juist wil vinden en is dit in de meeste gevallen niet op voorhand geweten.

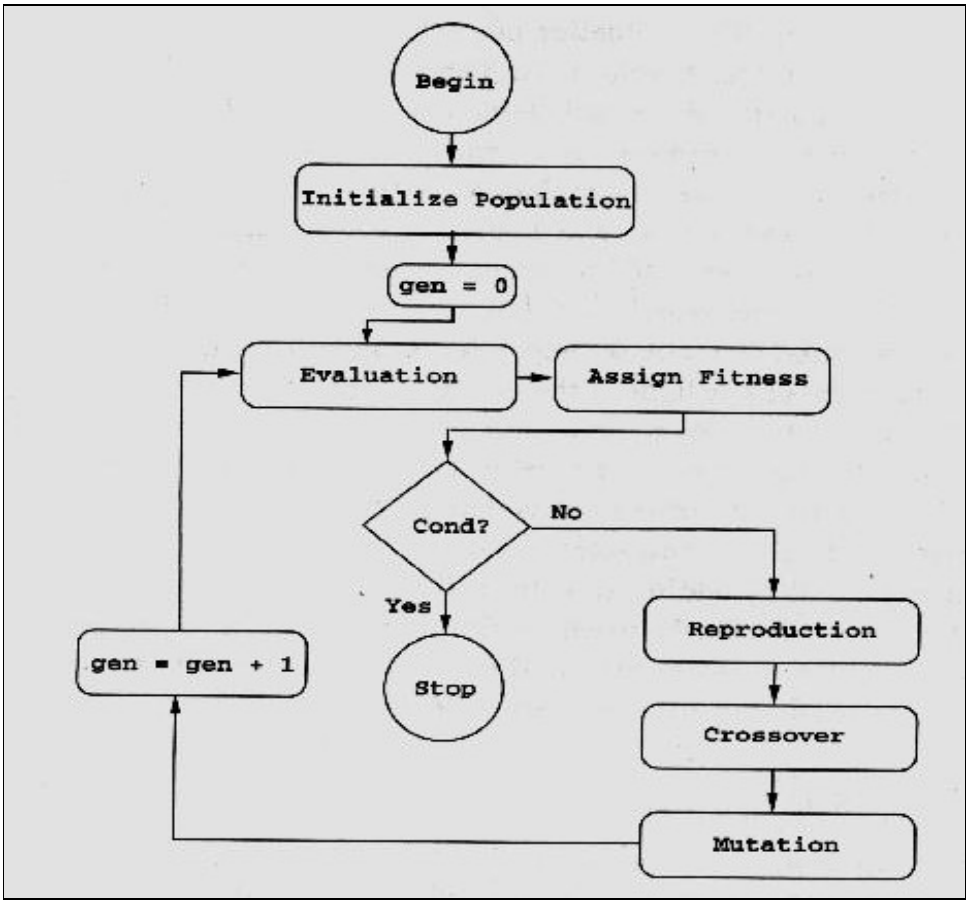
Andere parameters zijn implementatie of probleem afhankelijk. Zo kan men een kans parameter op ‘elitism’ toevoegen indien dit voorkomt of een parameter op een andere genetische operatie die gebruikt wordt. De parameters hangen aan elkaar vast zodat ze moeilijk elk apart geoptimaliseerd kunnen worden.
De waarde van de parameters is zeer moeilijk te bepalen. Het bepalen van optimale parameters wordt al lange tijd onderzocht. Een algemene regel hierover zal er waarschijnlijk niet snel komen. Meer en meer ontstaat het idee dat na de genetische operaties ook de parameters zichzelf gaan aanpassen doorheen de uitvoering van het algoritme [63][64][65].

5.4.5 De fitness functie

We hebben reeds veel gesproken over de geschiktheid of 'fitness' in een genetisch algoritme. Daarbij werd er veel gesproken over de geschiktheid/fitness van elementen in de populatie. Dit wil zeggen dat een waarde gekoppeld wordt aan een chromosoom. Deze waarde geeft weer hoe kort een element gelegen is bij een optimum.

f: X → Y met X een streng uit de populatie en Y de waarde voor X.

Hoe geschikter een element, hoe beter het dus een oplossing voor een probleem benadert. Om een waarde aan een element te geven wordt een bepaalde functie gebruikt: de fitness functie. De keuze van deze functie zal een zeer belangrijke rol spelen in het verkrijgen van een optimale oplossing omdat haar resultaat doorheen het algoritme geregeld gebruikt zal worden (zie punt 5.4.2.1).
De bedoeling van deze functie is dus een waarde toe te kennen aan een element om zo zijn geschiktheid in de zoekruimte te representeren. Het doel van het genetisch algoritme is immers ervoor te zorgen dat de waarde steeds gaat stijgen voor goede kandidaat oplossingen. Dit wil ook zeggen dat slechte elementen in de populatie 'gestraft' moeten worden zodat ze plaats kunnen maken voor betere elementen.
De functie moet zo opgebouwd worden dat de meest geschikte elementen 'beloond' en de minder geschikte elementen 'gestraft' worden. En zoals het van een functie verwacht wordt moet de beloning of straf volgens hun onderlinge ligging in de oplossingsruimte toegekend worden. Het opbouwen van dergelijke functie is opnieuw probleem afhankelijk en zal veel creativiteit vragen van de ontwikkelaar. Ook een goede kennis van het probleem is vereist. Weten waaraan een goede oplossing zou moeten voldoen zal mede de functie bepalen.
De moeilijkheidsgraad van dit probleem zal nog groter worden wanneer verschillende eigenschappen van een kandidaat oplossing vereist zijn. Zeker wanneer meerdere eigenschappen onafhankelijk van elkaar een bepaalde invloed hebben op de geschiktheid van een element. Een goed compromis over de impact van de vereisten opstellen zal de waarde van een element bepalen. Onderzoek hierover is reeds gestart maar is nog altijd bezig en is gekend als het onderzoek in 'multiple objectives in evolutionary algorithms' (cfr. infra).



Figuur 5.2: *Flow-chart van een standaard genetisch algoritme.*

5.4.6 Multiple objective evolutionary algorithm

5.4.6.1 Inleiding

Een genetisch algoritme heeft als doel een optimale oplossing te vinden in een zoekruimte. Wanneer de te zoeken oplossing moet voldoen aan slechts één eigenschap is dit niet zo moeilijk. Vooral het ontwerp van de fitness functie zal in complexiteit toenemen naarmate er meer objectieven vastgesteld worden waaraan een oplossing moet voldoen [66]. Dit wordt nog eens bemoeilijkt wanneer de objectieven in conflict zijn met elkaar. Een heel simpel voorbeeld hiervan is te zien in volgende probleembeschrijving:

Zoek de goedkoopste auto met zoveel mogelijk luxe.

Het bepalen van de optimale wagen is afhankelijk van twee voorwaarden. Ze moet goedkoop zijn maar zoveel mogelijk luxe bevatten. Deze twee eigenschappen zijn in conflict want hoe meer (minder) luxe hoe duurder (goedkoper) de wagen immers gaat worden. Voor dit probleem zullen er dan ook verschillende optimale oplossingen mogelijk zijn. De keuze van de meest optimale zal dan afhankelijk

worden van extra informatie of extra (minderwaardige) objectieven. De methoden die dit gaan bepalen kunnen onderverdeeld worden in vier categorieën:

No-preference methoden: maken geen gebruik van extra informatie. Hiervoor wordt vaak een heuristiek gebruikt die slechts tot één enkele optimale oplossing komt en de anderen niet beschouwt.
Posteriori methoden: maken gebruik van de eigenschappen van elk objectief en zal zo tot verschillende optimale oplossingen komen. De gekende extra informatie zal de uiteindelijke keuze van de optimale oplossing bepalen.
A priori methoden: gebruiken de kennis van de objectieven (eventueel theoretische achtergrond) om tot één enkele optimale oplossing te komen.
Interactieve methoden: gebruiken de extra informatie die doorheen het zoekproces verandert, om tot een optimale oplossing te komen.

In de volgende sectie zal één van de meest gebruikte methoden beschreven worden.

5.4.6.2 The weighted Sum Methode

Zoals hierboven vermeld, zijn er meerdere objectieven gegeven voor het zoeken naar een optimale oplossing. De 'weighted sum' methode valt onder de categorie van de à priori methoden. De vraag is nu hoe die verschillende objectieven afgehandeld moeten worden. Deze methode gaat het multi-objective probleem herleiden tot een single-objective probleem door de verscheidene objectieven aan elkaar te koppelen. Hiervoor wordt een verzameling parameters opgesteld die elk gerelateerd zijn aan een objectief.
De parameters gaan de objectieven scaleren en normaliseren zodat ze achteraf opgeteld kunnen worden. Hierdoor ontstaat één enkel objectief zodat het probleem eenvoudiger tot een optimale oplossing kan komen.
Het bepalen van de parameters is zeker niet eenvoudig en probleemafhankelijk. Ze moeten de objectieven aan elkaar relateren zodat elk objectief nog steeds zijn invloed uitoefent op het resultaat. Ook de grootte van de invloed is hierbij van belang. Niet elk objectief moet even belangrijk zijn. Ook hiermee moet rekening gehouden worden bij het bepalen van de parameters.
Wanneer de parameters uiteindelijk bepaald zijn worden ze vermenigvuldigd met hun overeenkomstig objectief. De resultaten worden gesommeerd zodat er uiteindelijk een single-objective optimizatie probleem ontstaat.

Deze methode is eenvoudig, vrij intuïtief (afhankelijk van het probleem) en garandeert een optimale oplossing. Toch kent ze enkele gebreken. Het bepalen van de parameters is duidelijk niet zo simpel hoewel het gebruik van verschillende parameters vaak tot éénzelfde optimale oplossing leiden. Het bewijzen dat verschillende parameters en welke parameters tot éénzelfde oplossing komen is dan opnieuw niet eenvoudig. De oplossing die uiteindelijk gevonden wordt zal via een single-objective optimizatie gebeuren.
Deze methode is in sommige gevallen niet bruikbaar. Indien de ruimte van de oplossingen niet convex is zal deze methode niet kunnen gebruikt worden.

Uiteraard zijn er nog andere methoden maar die vallen buiten het bereik van dit proefschrift. Voor meer informatie wordt verwezen naar [66].

5.5 Conclusie

Een genetisch algoritme vindt zijn basis in de evolutieleer. Door gebruik te maken van de evolutie van elementen tracht het zo efficiënt en snel mogelijk een globaal optimum te vinden (te benaderen) voor een gegeven probleem. Als zoekalgoritme verschilt het compleet met de meer traditionele zoekalgoritmen en heuristieken.
Een genetisch algoritme (GA) vertrekt vanuit een populatie elementen en zal deze populatie via genetisch gekende operaties laten evolueren naar een betere populatie. Er wordt dus op een verzameling elementen gewerkt i.p.v. één enkel element. In dit proces wordt eveneens probabiliteit toegevoegd i.p.v. van de deterministische functies van de traditionele algoritmen.
Het principe van een GA wordt reeds in vele domeinen gebruikt en zal zeker nog in gebruik toenemen. Toch moet er nog steeds veel onderzoek gebeuren. Het ontwikkelen van een GA is geen eenvoudig proces. Er kunnen verschillende moeilijkheden optreden waarbij de keuze van operaties en parameters de belangrijkste zijn. De werking van een GA wordt daarenboven ook nog eens grotendeels bepaald door een eigen ontwikkelde fitness functie. Ook dit proces vereist heel veel creativiteit en achtergrond kennis om het GA correct te kunnen laten werken. Hoewel de verschillende onderdelen onafhankelijk van elkaar geïmplementeerd zouden kunnen worden zal een GA in zijn geheel volledig afhangen van de onderlinge samenwerking van deze verschillende onderdelen.

Hoofdstuk VI: GenGA: een genetisch algoritme voor code word design

6.1 Inleiding

In de vorige twee hoofdstukken werd uitgebreid gesproken over het n-bit majority probleem en de moeilijkheden die hierbij optreden. Er werd geëindigd met de vraag hoe men het beste het DNA zou samenstellen. Daarna werd dieper ingegaan op het genetisch algoritme, haar gebruik en nut.
Omdat we eerst en vooral willen bepalen welk DNA we zouden moeten gebruiken om aan de praktijk te kunnen beginnen, zullen we op zoek moeten gaan naar een optimale oplossing. Het gebruik van een genetisch algoritme zou hierbij een oplossing kunnen bieden.

Het doel is dus een genetisch algoritme te schrijven dat een goede verzameling DNA strengen oplevert voor het n-bit majority probleem in de praktijk te kunnen aanvangen. Nu geweten is hoe zo een genetisch algoritme opgebouwd moet worden kan dit in de praktijk uitgevoerd worden. Voor het 3-bit majority probleem werd het programma GenGA ontwikkeld in een java omgeving om het code word design probleem op te lossen.

In dit hoofdstuk zal nagegaan worden hoe dit algoritme werd geconstrueerd en welke problemen er allemaal aan verbonden zijn. In figuur 5.2 wordt een flow-chart getoond van een standaard genetisch algoritme waarop het concept gebaseerd is.

6.2 De kennis van het probleem

In hoofdstuk 4 werd uitgebreid het n-bit majority probleem besproken. Om dit probleem met DNA in de praktijk te gaan oplossen hebben we DNA strengen nodig die de data zouden kunnen voorstellen. Er werden daarbij verschillende eisen gesteld waaraan het DNA moet voldoen. Deze staan uitgebreid beschreven in punt 4.4.
Nu we weten aan welke eisen het DNA zal moeten voldoen kan op zoek gegaan worden naar de verzameling van strengen die operationeel kunnen gebruikt worden. Dit wordt concreet toegepast voor n = 3.
De opdracht wordt dan: Geef een verzameling van 12 (4n) strengen DNA van lengte 20 bp die voldoen aan de gegeven vereisten.

Waarom een genetisch algoritme?

In eerste instantie lijkt het eenvoudig een algoritme te schrijven dat dit probleem oplost. We beschrijven intuïtief een eerste algoritme:

- Beschouw elke mogelijke verzameling van 12 DNA strengen (lengte 20 bp) en plaats die in een grote verzameling S.
- Neem een deelverzameling s van S en controleer of deze voldoet aan de eisen.
- Verwijder deze verzameling s uit S.
- Voldoet s stop met als oplossing s. Zo niet keer terug naar stap 2.

Helaas is dit geen efficiënte aanpak. Er zijn twee problemen bij deze oplossing:
Ten eerste is de verzameling S zeer groot. We willen DNA strengen van lengte 20 zodat er 4^{20} mogelijke strengen bestaan (zie hoofdstuk III). Voor de oplossing van dit probleem zoeken we een verzameling van 12 DNA strengen. Een oplossingsverzameling s zou moeten bestaan uit 12 verschillende DNA strengen. Daarenboven wordt hier geen rekening gehouden met de volgorde in elke combinatie.
Het aantal oplossingsverzamelingen zou dan zijn:

$(4^{20})! / (12!) \times (4^{20} - 12)!$
$= (1.1 \times 10^{12})! / (12! \times (1.1 \times 10^{12} - 12)!)$

$(1.1 \times 10^{12})! / (1.1 \times 10^{12} - 12)!$
= Het product van de 12 getallen die groter zijn dan de noemer.
$> (1.1 \times 10^{12} - 12)^{12}$
$> (1.0 \times 10^{12})^{12}$

Dit levert ons:
$(1.0 \times 10^{12})^{12} / 4.8 \times 10^{8}$ $\qquad$ $12! = 4.8 \times 10^{8}$

De verzameling S zou dan nog steeds te groot zijn om door het beschreven algoritme uitgevoerd te kunnen worden. De zoekruimte blijkt dus enorm groot te zijn. Daarvoor bestaan er heuristieken die zo snel mogelijk een oplossing proberen te benaderen.

Ten tweede weten we niet hoe een resultaat er precies uitziet. Het is meestal het geval dat er meerdere verzamelingen aan de vereisten voldoen waardoor er dus meerdere oplossingen mogelijk zijn. We willen hieruit natuurlijk één die het optimale is of althans benadert. De zoekruimte op zich zal niet veranderen maar hoe de data verdeeld zijn in deze zoekruimte is niet gekend ('smoothness' is onbekend).
Deze opmerkingen in acht genomen lijkt de keuze van een genetisch algoritme wel gerechtvaardigd (zie punt 5.3.3).

6.3 Constructie van GenGA

6.3.1 Constructie stap 1(a): Gebruikte codering

We willen werken op DNA strengen van lengte 20 bp. De meest natuurlijke keuze lijkt in eerste instantie het gebruik van het type String van een vaste grootte. Hierbij gaan we dus strengen ontwikkelen bestaande uit het alfabet {A, C, G, T} van lengte 20 bp. Het gebruik van een bitstring zou de representatie niet vergemakkelijken, net zoals een andere structuur.
Een element uit de populatie zal niet enkel deze streng bevatten. We zoeken immers een verzameling van 12 strengen waarop we beperkingen leggen. De populatie zou dan best opgebouwd worden uit dergelijke verzamelingen. Elke verzameling op zich stelt dan eigenlijk een chromosoom voor van de populatie. Een voorbeeld van een mogelijke kandidaat oplossing en element van de populatie wordt gegeven in volgend voorbeeld:

```
{   "ACACACTATCTGCACGCTAC",
    "GTCTAGCTGTACGGGGGGCT",
     "GTGCTGCTGCTGGCCGTTCC",
    "AACGACGCGCAAAAACGCGA",
    "TTTTTTGGGGGGAAAAAACC",
    "CGTAGCTAGCTGCCCGCGCG",
    "CGTACTAGCTAGCTGATCGA",
    "TGCTAGCTACGTAGCTAGCT",
    "AGCTAGCTAGCGACGACTGT",
    "CATGCTAGCTGCGGGTGTGT",
    "GTGCTAGTCACCGGCCGCGC",
    "GGCTGCGGCTAATATCAGTG",
    "CTGCTAGCTAGACTACGTCG",
      }
```

Voor deze structuur werd een aparte klasse “ElementStrings” ontwikkeld. Deze klasse bevat ook de genetische operaties die op de strengen uitgevoerd zullen worden en wordt getoond in figuur 6.1.

```
 7
 8 import java.util.*;
 9
10 class ElementStrings {
11
12     private String m_true1;
13     private String m_true2;
14     private String m_true3;
15     private String m_false1;
16     private String m_false2;
17     private String m_false3;
18
19     private String m_end1;
20     private String m_end2;
21     private String m_begin2;
22     private String m_begin3;
23
24     private String m_primer1;
25     private String m_primer2;
26     private double m_temp;
27     private double m_fitness;
28
29     private void CrossOver(){}
30     private void Mutate(){}
31     private void setFitness(){}
32
```

Figuur 6.1: *Structuur van een element van de populatie*.

6.3.2 Constructie stap 1(b): Aanpassen van de codering

In het algoritme worden geen stappen ondernomen die de codering gaan aanpassen of veranderen. Inversion of andere methoden zijn hier niet van toepassing omwille van volgende redenen:

1. De lengte van de data mag niet aangepast worden tijdens de uitvoering. Er wordt begonnen met strengen van vaste lengte, namelijk 20 bp, en die moet behouden blijven.
2. Ook de orde of inhoud van de data mag niet veranderen. We weten immers niet op voorhand of er deelstrengen bestaan die eventueel niet onderbroken mogen worden. Het volgende idee klinkt misschien wel logisch: indien in een oplossingsverzameling enkele strengen reeds voldoen aan de eigenschappen, behoudt deze dan via bijvoorbeeld elitism. Hierdoor zal een gedeelte van een oplossing vastliggen waardoor de rest van de verzameling afhankelijk wordt van deze gegevens. Dit heeft tot gevolg dat hun representatie ofwel te beperkt wordt zodat de zoekruimte te klein wordt, ofwel dat de hele oplossingsverzameling uiteindelijk minder geschikt zal worden. Een voorbeeld kan helpen.

Beschouw de volgende oplossingsverzameling:

{ (true_1)	AAAAAAAAAAAAAAAAAAA,
(true_2)	TTTTTTTTTTTTTTTTTTT,
(true_3)	CCCCCCCCCCCCCCCCCCC,
(false_1)	GGGGGGGGGGGGGGGGGGG,
(false_2)	ACGTAGCTGCATGCTAGCTA,
(false_3)	GTCGATGCTAGCTGATGCTA,
(end_1)	GCTAGCTAGCTAGCTAGCGA,
(end_2)	TCGCGAGCCGCCGCGCCCGC,
(begin_2)	CGCGCGCGATATATCGCGCA,
(begin_3)	TGCATATAAACACATGTCAG,
(primer_1)	GTCGATGCTAGCTGATGCTA,
(primer_2)	AAACGTGCTAGCTACGTCTA
}	

De eerste vier elementen zullen uiteindelijk onderling zeer goede resultaten geven. Indien we deze vier elementen zouden vastpinnen zouden zij met de andere elementen geen goede vergelijkingen geven waardoor de geschiktheid (fitness) van het geheel zeker niet meer kan verbeteren. Voor het bepalen van de geschiktheid wordt verwezen naar punt 5.4.5: 'De fitness functie'. Laat ons voor dit voorbeeld kijken naar eigenschap 1: alle andere strengen (buiten de eerste vier) moeten zo verschillend mogelijk zijn van de eerste vier elementen: d.w.z. dat ze onderling zo weinig mogelijk gemeenschappelijk mogen hebben op elke lokatie in de streng. Dit wordt zeer moeilijk aangezien elke streng dan op elke plaats best een andere nucleotide heeft staan dan A, C, G of T. Helaas is dit onmogelijk omdat het alfabet slechts uit deze vier elementen bestaat.

3. Ook de plaats van crossover of mutatie zal niet beperkt mogen worden omwille van dezelfde reden.

6.3.3 Constructie stap 2: De selectie methode

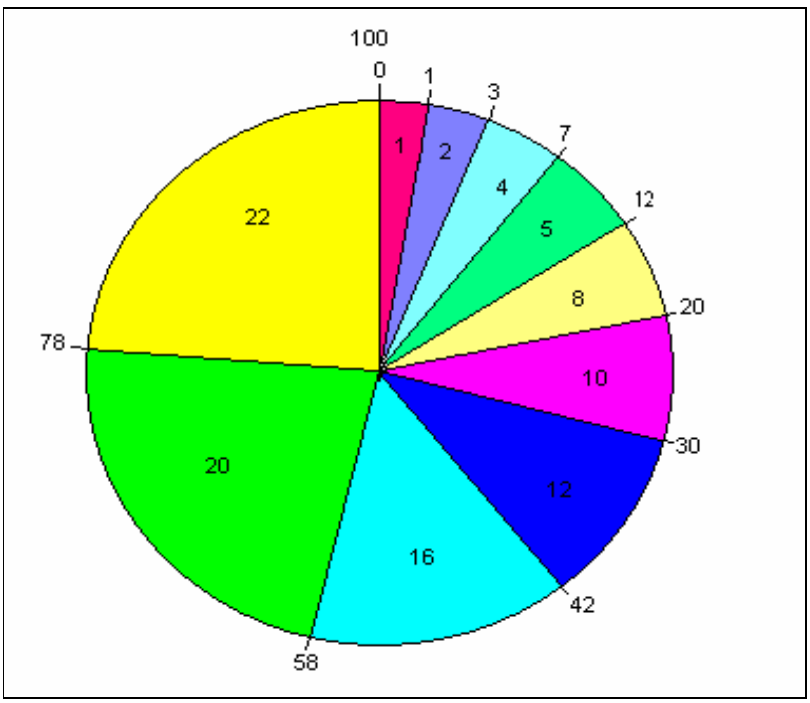
Aangezien er verschillende mogelijkheden zijn is het van belang de beste methode te kiezen om selectie uit te voeren. Helaas weten we niet of de populaties sterk verdeeld gaan zijn, of de geschiktheid of fitness van de elementen ver of dichtbij elkaar liggen. Het feit dat de geschiktheid van een element zou bepalen of haar 'kinderen' minstens beter zouden zijn, werd aangenomen. De keuze is dan gevallen op het roulette wiel principe. Ook elitism werd toegevoegd maar op willekeurige basis.
Dit werd als volgt aan het algoritme toegevoegd:

- Kies een willekeurig getal en controleer of dit getal de kans op elitism aanvraagt. Zo ja, voeg de twee beste elementen toe aan de nieuwe populatie.
- Bereken de som 'SUM' van alle geschiktheden in de populatie (N = de grootte van de populatie) en deel door 100. Dit element P is dan 1% van de oppervlakte van het wiel.

- Bereken voor elk element (van laagste tot hoogste geschiktheid) hoeveel procent van het wiel hiervoor bestemd is (hoeveel keer gaat P in de fitness) en tel.
- Bepaal twee willekeurige getallen in dit interval [0, 100].
- Bepaal de lokaties op het wiel en de twee overeenkomstige elementen die eraan gekoppeld zijn.
  Herhaal de vorige twee stappen indien de twee resulterende elementen hetzelfde zijn.
- Deze twee worden gebruikt voor crossover en/of mutatie.
- Herhaal dit N/2 keer of (N/2)-1 (indien elitism werd uitgevoerd) zodat de nieuwe generatie opnieuw uit N elementen bestaat.

Een voorbeeld:

een populatie van 10 elementen met als geschiktheden:
{1, 2, 4, 5, 8, 10, 12, 16, 20, 22}
SUM = 1 + 2 + 4 + … + 22 = 100 zodat P = 1
Het roulette wiel zal er uitzien zoals getoond in figuur 6.2.



Figuur 6.2: *Roulette wiel voor de populatie met geschiktheden {1, 2, 4, 5, 8, 10, 12, 16, 20, 22}*

Kies twee willekeurige getallen in het interval [0,100]: 10 en 85.
De twee overeenkomstige elementen op het wiel zijn dan 5 en 22.
Indien deze twee elementen hetzelfde zouden zijn, kies opnieuw twee willekeurige getallen tussen 0 en 100.

6.3.4 Constructie stap 3: Genetische operaties

Uiteraard worden de twee basis operaties aan het algoritme toegevoegd. Zowel crossover als mutatie van de strengen worden geïmplementeerd. Het algoritme valt duidelijk onder de categorie van generationele algoritmen. Laat ons eens dieper ingaan op de implementatie van de twee operatoren en dit statement hard maken.

6.3.4.1 Crossover

Wanneer er van de ene generatie naar de andere overgegaan wordt zullen de chromosomen van de huidige populatie gaan paren volgens de regel van single-point crossover. In plaats van crossover tussen twee strengen zal dit crossover voorstellen tussen twee verzamelingen van strengen. Dit is eenvoudig aan het algoritme toegevoegd. Elke streng in de verzameling zal een crossover aangaan met zijn overeenkomstige streng in de andere verzameling. De plaats waar dit in de streng gebeurt wordt willekeurig bepaald en geldt voor alle strengen.
Waarom moet er crossover plaatsvinden tussen de overeenkomstige strengen en waarom niet voor elke streng op een andere willekeurige plaats? Dit zou echter ook een mogelijkheid geweest zijn en zou zeker de diversiteit van de resultaten bevorderen. Toch zou waarschijnlijk deze aanpak met haar grotere diversiteit minder geschikt zijn om convergentie te verkrijgen naar een globaal optimum. Maar niets vertelt of dit slechter dan wel beter is voor het verkrijgen van resultaten. Er moet nu eenmaal een keuze gemaakt worden.
Volgend voorbeeld geeft duidelijk weer hoe in dit programma de operatie crossover wordt uitgevoerd tussen twee strengen:

Gegeven twee strengen van lengte 20 en willekeurige lokatie 8:

"GCTAGCTAGCTAGTCGTAGC"
en
"CGTTTGAATGAATCCCGGTA"

De eerste streng wordt gesplitst na nucleotide 8, de tweede na nucleotide 20-8 wat resulteert in:

"GCTAGCTA - ACTAGTCGTAGC"
en
"CGTTTGAATGAA - TCCCGGTA"

Crossover zal resulteren in de twee nieuwe strengen:

"GCTAGCTA - CGTTTGAATGAA"
en
"TCCCGGTA - ACTAGTCGTAGC"

6.4.3.2 Mutatie

Ook voor mutatie wordt dit principe toegepast. Indien mutatie plaats zal vinden zal dit op alle elementen van de verzameling gebeuren en allen op dezelfde willekeurige gekozen plaats. Mutatie van een streng bestaat erin om op de gekozen lokatie een nieuw element te plaatsen. Dus als er mutatie plaatsvindt in een streng zal op die plaats het nucleotide vervangen worden door een willekeurig gekozen nucleotide uit het alfabet {A,C,G,T}-{het nucleotide dat er staat}.
Stel dat een streng gemuteerd moet worden op de willekeurig gekozen plaats 5, dan zal voor de streng "GCAT**C**TCTGATCGACGATGCC" de **C** vervangen worden door een A, T of G.

Aangezien voor de meeste genetische algoritmes deze twee operaties voldoende zijn, zijn er ook hier geen andere operaties aan toegevoegd.

6.3.5 Constructie stap 4: parameter setting

Eerst en vooral moet er bepaald worden welke parameters een rol gaan spelen in het algoritme. Uiteraard is er de grootte van de populatie, een parameter die de kans op crossover bepaalt en één die mutatie bepaalt. Daarnaast wordt er op voorhand vastgelegd hoeveel generaties lang we het algoritme gaan laten lopen zodat het algoritme na eindige tijd zal stoppen. Aangezien we ook het principe van elitism een kans geven wordt hier ook een parameter voor aangemaakt. In totaal hebben we dus vijf parameters waarbij de drie eerste statistische parameters zijn en de volgende twee parameters die op voorhand vastgelegd zullen worden. Deze zijn:

```
private double m_mutationRate;      interval [0.0, 1.0]
private double m_crossover_rate;    interval [0.0, 1.0]
private double m_elitism;           interval [0.0, 1.0]
private int m_popsize;
private int m_iterations;
```

Zoals reeds vermeld, is de keuze van de parameters niet op voorhand te bepalen. Het zou dan interessant zijn deze parameters gedurende de loop van het algoritme zichzelf te laten aanpassen. Helaas weten we niet in hoeverre de toestand van een generatie deze parameters zou kunnen beïnvloeden opdat we betere resultaten kunnen genereren. Daardoor zullen we ervan uitgaan dat deze parameters hetzelfde blijven gedurende de hele run van het algoritme.
Van zodra het algoritme af is zullen we met deze parameters kunnen spelen en misschien daarbij bepaalde gegevens verzamelen die duidelijk maken welke waarden van parameters het beste zijn.
In volgend voorbeeld kan men zien hoe de statistische parameters in het programma gebruikt kunnen worden.

We beschouwen de parameter m_mutationRate:

```
        private boolean mutatie()
        {
            int MAX = 100000;
            Random random = new Random();
            int guess = random.nextInt(MAX);
            if(guess < MAX * m_mutationRate)
                return true;
            else
                return false;
    }
```

Kies een willekeurig getal in het interval [0,MAX]. Indien we de parameter bijvoorbeeld 0.5 kiezen voor m_mutationRate wil dit zeggen dat er 50% kans is dat mutatie zal moeten plaatsvinden. MAX * m_mutationRate bepaalt een nieuw interval [0, MAX/2]. Het willekeurig gekozen getal heeft dan inderdaad 50% kans dat het in dit interval gelegen is.

6.3.6 De fitness functie

Vervolgens is het de beurt aan de fitness functie. Om dit onderdeel te kunnen toelichten moeten eerst twee vragen gesteld worden:

1. Waaraan moeten de elementen van de populatie voldoen?
2. Op welke manier en in welke mate kunnen we ‘beloningen’ of ‘straffen’ toewijzen aan die elementen?

Op de eerste vraag werd reeds een duidelijk antwoord gegeven in punt 4.4. Dit brengt ons onmiddellijk tot de tweede vraag. Hoe gaan we de goede chromosomen belonen en hoe gaan we de minderen straffen?
Het is de bedoeling een geschiktheid te geven aan een oplossingsverzameling die aan verschillende vereisten moet voldoen. We moeten dus eerst een duidelijk onderscheid maken want de fitness functie die we willen creëren zal uit verschillende onderdelen bestaan.

We zullen daarom eerst de eisen onderverdelen in een aantal categorieën:

- De verschillen tussen de strengen onderling (zie punt 6.3.6.1)
- De verschillen tussen de complementen en de andere strengen (zie punt 6.3.6.2)
- De verschillen tussen het complement van de streng en de streng zelf (zie punt 6.3.6.3)
- De smelttemperatuur van de stickers (zie punt 6.3.6.4)

6.3.6.1 De verschillen tussen de strengen onderling

De 12 strengen in een oplossingsverzameling moeten zo verschillend mogelijk van elkaar zijn op elke positie in de streng. De Hamming afstand berekent deze verschillen en kan dus gebruikt worden als basis voor de fitness functie in dit onderdeel.
We berekenen voor elk koppel in de verzameling deze Hamming afstand en tellen ze achteraf allen bij elkaar op. Dit zou een mogelijke functie zijn voor dit onderdeel. De goede worden beloond met een hoog getal dat maximum 20 kan zijn. Zij die niet veel van elkaar verschillen moeten gestraft worden en zullen ook een laag getal bijdragen aan de geschiktheid van de verzameling. Dit zou misschien wel eens onvoldoende kunnen zijn. We weten dat de eis moet gelden voor alle koppels opdat het een goede verzameling zou zijn. Indien alle koppels, uitgezonderd één, allen een hoge waarde opleveren en één een lage zou de verzameling eigenlijk niet voldoen, hoewel de waarde toch hoog zal zijn. Dit wordt getoond in volgend voorbeeld:

Beschouw een verzameling van vier strengen (zes verschillende koppels)

(1) “CGTGACGTGACGCTAGCTCG”
(2) “ACGCTGTAATAAGGGCGCAT”
(3) “GTCTGAACCCGTAATAAAGA”
(4) “CGTGACGTGACGCTAGCAAA”

De Hamming afstanden worden:

H(1,2) = 20
H(1,3) = 20
H(1,4) = 3
H(2,3) = 20
H(2,4) = 19
H(3,4) = 18

100

De som is 100 wat heel hoog is hoewel de verzameling op zich niet voldoende zal zijn. H(1,4) zal voor een eventueel experiment hoogstwaarschijnlijk voor problemen zorgen. Terwijl de volgende verzameling een geschiktheid heeft van 96 zou deze zelfs veel beter voldoen aan de vereiste dan de eerste verzameling:

(1) “CGTGACGTGACGCTAGCTCG”
(2) “ACGCTGTAATAAGGGCCTCG”
(3) “GTCTGAACCCGTAATACTCG”
(4) “TAAACTCGTGTCTCCTCTCG”

De Hamming afstanden worden:

H(1,2) = 16
H(1,3) = 16
H(1,4) = 16
H(2,3) = 16
H(2,4) = 16
H(3,4) = 16

96

De koppels die een slechtere Hamming afstand hebben zouden dus meer gestraft moeten worden. Een formule die gebruikt wordt in de paper om het ECC (Error Correcting Code) probleem op te lossen biedt hier een uitkomst [67]:

$$1 / (\sum 1 / (H(x,y))^2)$$

Sommatie over alle elementen x en y met $x \neq y$

Dit zou in vorig voorbeeld tot volgende resultaten leiden:

In het eerste geval: 1 / (1/400 + 1/400 + 1/400 + 1/9 + 1/361 + 1/324)
= 1 / (0.1244676)
= 8.03

In het tweede geval: 1/ (1/256 + 1/256 + 1/256 + 1/256 + 1/256 + 1/256)
= 1 / 0.0234375
= 42.67

Deze formule levert dan ook een fitness op die de geschiktheid van de verzameling beter definieert.

6.3.6.2 De verschillen tussen de complementen en de andere strengen

De tweede vereiste kan op dezelfde wijze geïmplementeerd worden als hierboven beschreven. Toch is er een groot verschil met de vorige vereiste. We moeten in dit onderdeel niet alle strengen met elkaar gaan vergelijken maar het complement van elke streng met alle anderen uit de verzameling. Het aantal vergelijkingen dat hier verkregen wordt kan hier beperkt worden. Het vergelijken van een streng A met het complement van een andere streng B heeft hetzelfde resultaat als de vergelijking van streng B met het complement van A.
Het algoritme ziet er dan als volgt uit: vergelijk het complement van elke variabele met de stickers en de primers. Vervolgens worden nog het complement van de stickers met de primers vergeleken. Het voorbeeld onderaan geeft een verduidelijking over welke vergelijkingen het gaat bij de variabelen tegenover de stickers. Deze drie onderverdelingen verkrijgen dan elk een geschiktheid via de formule uit punt 6.3.6.1 en worden bij elkaar opgeteld.

Omdat ze alle drie evenveel van belang zijn moet er nog een evenwicht gevonden worden. Zo zal bij elke onderverdeling het eindresultaat nog gedeeld worden door het aantal vergelijkingen die tot de fitness leiden. Volgend voorbeeld geeft de Hamming afstanden weer van het complement van elk van de variabelen t.o.v. de verschillende stickers.

$H(true\_1^C, end\_1)$	$H(true\_1^C, begin\_2)$
$H(true\_1^C, end\_2)$	$H(true\_1^C, begin\_3)$
$H(true\_2^C, end\_1)$	$H(true\_2^C, begin\_2)$
$H(true\_2^C, end\_2)$	$H(true\_2^C, begin\_3)$
$H(true\_3^C, end\_1)$	$H(true\_3^C, begin\_2)$
$H(true\_3^C, end\_2)$	$H(true\_3^C, begin\_3)$
$H(false\_1^C, end\_1)$	$H(false\_1^C, begin\_2)$
$H(false\_1^C, end\_2)$	$H(false\_1^C, begin\_3)$
$H(false\_2^C, end\_1)$	$H(false\_2^C, begin\_2)$
$H(false\_2^C, end\_2)$	$H(false\_2^C, begin\_3)$
$H(false\_3^C, end\_1)$	$H(false\_3^C, begin\_2)$
$H(false\_3^C, end\_2)$	$H(false\_3^C, begin\_3)$

Het eindresultaat van dit onderdeel is dan de som van de drie bekomen geschiktheden.

6.3.6.3 De verschillen tussen het reverse-complement van een streng en de streng zelf

We willen vermijden dat de strengen secundaire structuren gaan ontwikkelen. In een derde stap zal nagegaan worden of er een mogelijkheid bestaat tot het ontwikkelen van een hairpin. Dit kan als volgt nagegaan worden: het reverse-complement van een streng moet zo verschillend mogelijk zijn van de streng zelf. Een voorbeeld van een streng met hairpin maakt dit duidelijk

Beschouw de volgende streng:

“ACGTAGTGCTGCGACGT” (1)

De eerste vier basen zouden met de laatste vier kunnen binden met als gevolg het ontstaan van een hairpin (zie punt 3.3). Indien we nu het reverse-complement nemen krijgen we:

“ACGTCGCAGCACTACGT” (2)

Wanneer we de Hamming afstand nemen van deze twee strengen krijgen we:

$H(1,2) = 11$

Dit resultaat zou veel beter zijn geweest indien er geen mogelijkheid zou zijn tot hairpin. Door opnieuw dezelfde formule toe te passen zal het resultaat een fitness weergeven die toont in hoeverre de streng in staat is deze secundaire structuur te vormen. Hoe kleiner (en dus slechter) het resultaat hoe kleiner de geschiktheid van de streng. Deze geschiktheden, 12 in totaal, zullen opgeteld worden en weer meespelen in de globale geschiktheid van de verzameling.

6.3.6.4 De smelttemperatuur van de stickers

Tijdens het ontwikkelen van de volledige DNA strengen zorgen de stickers voor de connecties zoals reeds in punt 4.5.4 werd aangegeven. De temperatuur waarbij dit gebeurt speelt hierbij een belangrijke rol. Om de twee koppelingspunten, 'end_1begin_2' en 'end_2begin_3', op hetzelfde moment te laten 'plakken' hebben ze beide best dezelfde smelttemperatuur.
Ook deze vereiste zal een belangrijke rol spelen in het algoritme en moet dus in de fitness functie opgenomen worden. Hoe groter het verschil in temperatuur, hoe slechter.
Het berekenen van dit onderdeel zal een procent opleveren in hoeverre de twee smelttemperaturen op elkaar gelijken. We gebruiken hier de techniek van variantie in de statistiek. Indien de twee gelijk zijn krijgen we als resultaat 100%. Dit procent zal dan ook weer deelnemen aan de globale fitness functie voor de verzameling.

6.3.6.5 De globale fitness functie

De verschillende onderdelen zullen elk een bepaald doel hebben in de globale fitness functie. Het resultaat van de eerste drie onderdelen worden bij elkaar opgeteld. Op die manier zal het resultaat hoog zijn als aan alle vereisten voldaan werd. Het resultaat van de smelttemperatuur werd voor dit proefschrift als belangrijkste aangenomen. Ze zal dan ook op een andere manier de globale fitness bepalen. Er werd daarbij een percentage verkregen dat op het voorlopige resultaat toegepast werd. Het percentage van dit resultaat wordt genomen en erbij opgeteld. De globale fitness functie F ziet er dan als volgt uit:

$$X = \sum_{i=1}^{3} \text{onderdeel}_i$$
$$T\% = \text{onderdeel}_4$$

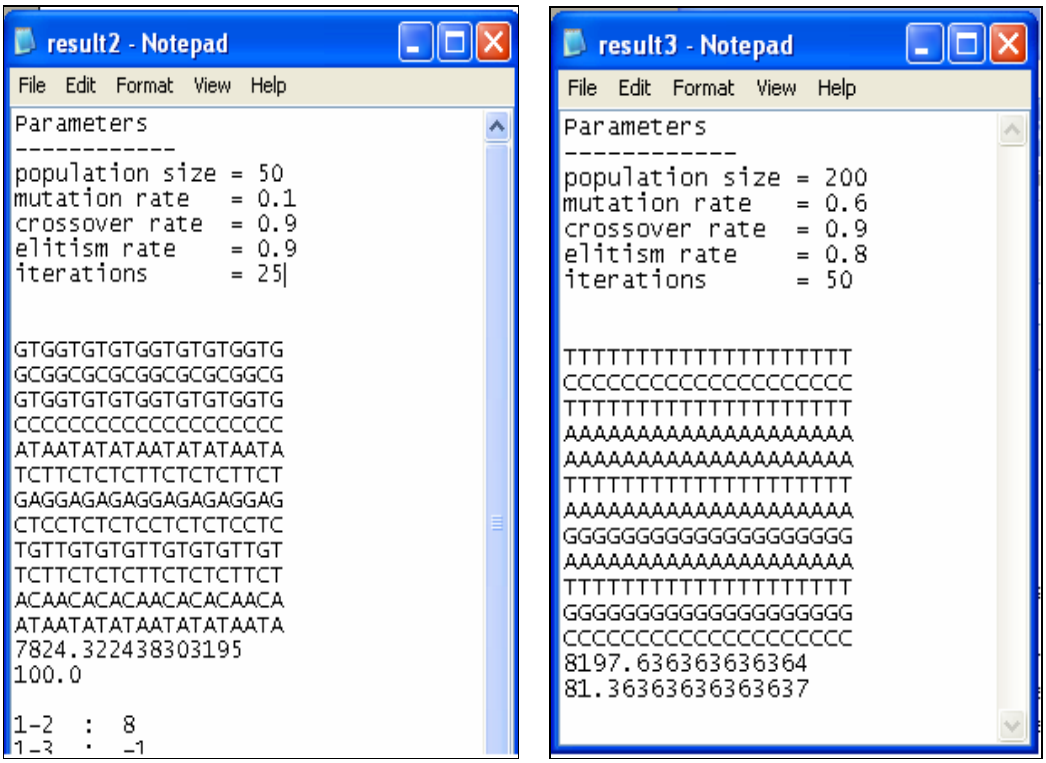
$$F = X + (T\% \text{ van } X)$$

6.4 Resultaten

6.4.1 Eerste poging

Het algoritme werd vervolgens geïmplementeerd in een Java omgeving JDK 1.5.0_2. Nadien werd het programma, GenGA genoemd, grondig getest. Er werden verschillende waarden genomen voor de parameters in het programma om zo tot

een globaal optimum te komen. Helaas werd al gauw duidelijk dat het verschil in parameters in bijna alle gevallen tot een slecht resultaat leidde. Ook de convergentie naar een optimum bleef uit.
Er ontstond ook een ander probleem dat daar aan gerelateerd was. Het programma geraakte gefixeerd op een zeer kleine deelverzameling van de search space. In figuur 6.3 zijn twee resultaten te zien die door het programma als optimaal werden beschouwd. Het is duidelijk te zien dat dit niet het geval is. De gecreëerde generaties geraakten gefixeerd op een beperkt deel van de zoekruimte, namelijk de elementen van de vorm zoals te zien is in figuur 6.3.
Wanneer het aantal iteraties klein werd gehouden leidde dit in sommige gevallen niet tot dergelijke resultaten. Dit wil niet zeggen dat het aan deze parameter ligt. Door het aantal iteraties klein te houden zou het programma gewoon nog niet de tijd gehad hebben oplossingen zoals in figuur 6.3 te produceren. Ook aan de andere parameters lag het niet. Voor de mutatie bijvoorbeeld, een operatie om diversiteit juist te bewaren, werd met kans 1 (dus altijd mutatie uitvoeren) gekozen. Toch convergeerde het programma telkens naar oplossingen van de aard in figuur 6.3.



Figuur 6.3: *Twee resultaten na het runnen van het programma.*

Wanneer meerdere resultaten in rekening gebracht werden, was het alsof het programma in de strengen de elementen ging clusteren. Daarom werd de selectie methode onderzocht en grondig getest op haar validiteit.
Het algoritme werd aangepast en in plaats van roulette wiel werden er gewoon willekeurig twee elementen uit de populatie genomen voor crossover. Opnieuw

werd het programma aan een hele reeks testen onderworpen voor de parameters. Het resultaat was ontmoedigend. Hoewel in de eerste fase van het algoritme de diversiteit in de strengen goed werd waargenomen, werd er toch, naar mate het aantal iteraties toenam, opnieuw het cluster probleem waargenomen.
De genetische operator crossover werd nog eens gecontroleerd maar ook deze factor kan niet bewijzen waarom de elementen in de streng gaan clusteren. Uiteraard zou crossover de elementen van dergelijke verzameling door elkaar gooien en uiteindelijk in een best case scenario alle strengen weer gediversifieerd krijgen. Hun geschiktheid daalt dan en het voorlopige slechte optimale resultaat blijft behouden.
Het probleem zou dus hoogstwaarschijnlijk aan de fitness functie liggen die we dan beter ook niet meer gebruiken. Maar eerst werd de interne werking van de fitness functie grondig onderzocht bij dergelijk slecht optimum. Laat ons eens kijken wat het resultaat zou zijn van de vergelijkingen van de strengen onderling met de functie die daarvoor beschreven werd. We gebruiken de tweede oplossingsverzameling van het voorbeeld hierboven, de strengen van boven naar onder genummerd van 1 tot 12. De Hamming afstanden van de koppels worden weergegeven in tabel 6.1:

Tabel 6.1: *De Hamming afstanden tussen de verschillende DNA strengen onderling.*

H(1,2) = 20	H(3,4) = 20	H(5,10) = 20	H(10,11) = 20
H(1,3) = 0	H(3,5) = 20	H(5,11) = 20	H(10,12) = 20
H(1,4) = 20	H(3,6) = 0	H(5,12) = 20	H(11,12) = 20
H(1,5) = 20	H(3,7) = 20	H(6,7) = 20	
H(1,6) = 0	H(3,8) = 20	H(6,8) = 20	
H(1,7) = 20	H(3,9) = 20	H(6,9) = 20	
H(1,8) = 20	H(3,10) = 0	H(6,10) = 0	
H(1,9) = 20	H(3,11) = 20	H(6,11) = 20	
H(1,10) = 0	H(3,12) = 20	H(6,12) = 20	
H(1,11) = 20	H(4,5) = 0	H(7,8) = 20	
H(1,12) = 20	H(4,6) = 20	H(7,9) = 0	
H(2,3) = 20	H(4,7) = 0	H(7,10) = 20	
H(2,4) = 20	H(4,8) = 20	H(7,11) = 20	
H(2,5) = 20	H(4,9) = 0	H(7,12) = 20	
H(2,6) = 20	H(4,10) = 20	H(8,9) = 20	
H(2,7) = 20	H(4,11) = 20	H(8,10) = 20	
H(2,8) = 20	H(4,12) = 20	H(8,11) = 0	
H(2,9) = 20	H(5,6) = 20	H(8,12) = 20	
H(2,10) = 20	H(5,7) = 0	H(9,10) = 20	
H(2,11) = 20	H(5,8) = 20	H(9,11) = 20	
H(2,12) = 0	H(5,9) = 0	H(9,12) = 20	

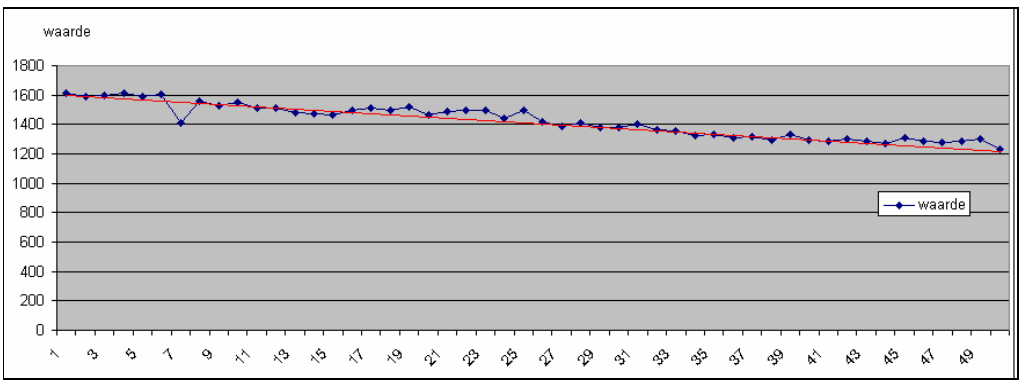
Indien we de formule voor de eerste streng (1) nemen in vergelijking met de anderen (2,…,12) levert dit:

1 / (1/400) + (1/0) + (1/400) + (1/400) + (1/0) + (1/400) + (1/400) + (1/400) + (1/0) + (1/400) + (1/400)

Omdat ‘delen door 0’ niet mag werd dergelijke deling gewoon op 0 gezet. Het resultaat na deze aanpassing is dan 50.
Dit getal is enorm groot t.o.v. resultaten die beter zijn (hun waarde ligt rond het getal 30). Het probleem is dat de strafmaat blijkbaar niet streng genoeg is om zich te kunnen onderscheiden van betere oplossingen.

6.4.2 Tweede poging

De oplossing voor het probleem hierboven leek in eerste instantie eenvoudig. Indien er een oplossingsverzameling bestaat zoals beschreven in het vorige deel wordt de geschiktheid simpelweg op 0 gezet voor dat onderdeel. Met deze oplossing zou dan een slecht onderdeel van de verzameling geen bijdrage leveren aan de globale geschiktheid die dan lager zal worden. Opnieuw werd het algoritme meerdere keren getest met verschillende parameters wat leidde tot een niet gehoopt resultaat. Het grootste deel van de oplossingen convergeerde weer naar dergelijke oplossingen. Daarenboven daalde de globale geschiktheid na elke nieuwe generatie. Figuur 6.4 geeft dit weer voor een bepaalde test. De meeste oplossingen vertonen een figuur van deze aard.



Figuur 6.4: *De hoogste geschiktheid van iedere generatie tijdens de run van het programma.*

6.4.3 Derde poging

Omdat het probleem bleef aanhouden werd een nieuwe aanpak onvermijdelijk. De fitness functie werd helemaal van vooraf aan opnieuw geanalyseerd. De slechte oplossingen zoals in figuur 6.3 werden harder gestraft zodat ze uiteindelijk niet zouden voorkomen in volgende generaties. Er werd besloten hiervoor een geheel nieuwe parameter in te voeren namelijk de ‘integer Hamming_bound’. Deze parameter bepaalt een minimal Hamming afstand van de elementen in een oplossingsverzameling gekozen in een interval [1,20]. De keuze van deze parameter speelt een belangrijke rol in zowel het opstellen van de geschiktheid als de creatie van nieuwe generaties zoals we verder zullen zien. Een voorbeeld van het gebruik van deze parameter wordt hieronder getoond. Deze nieuwe parameter krijgt de waarde 12:

In de vergelijking van de Hamming afstanden

Beschouw twee strengen DNA en de parameter Hamming_bound = 12:

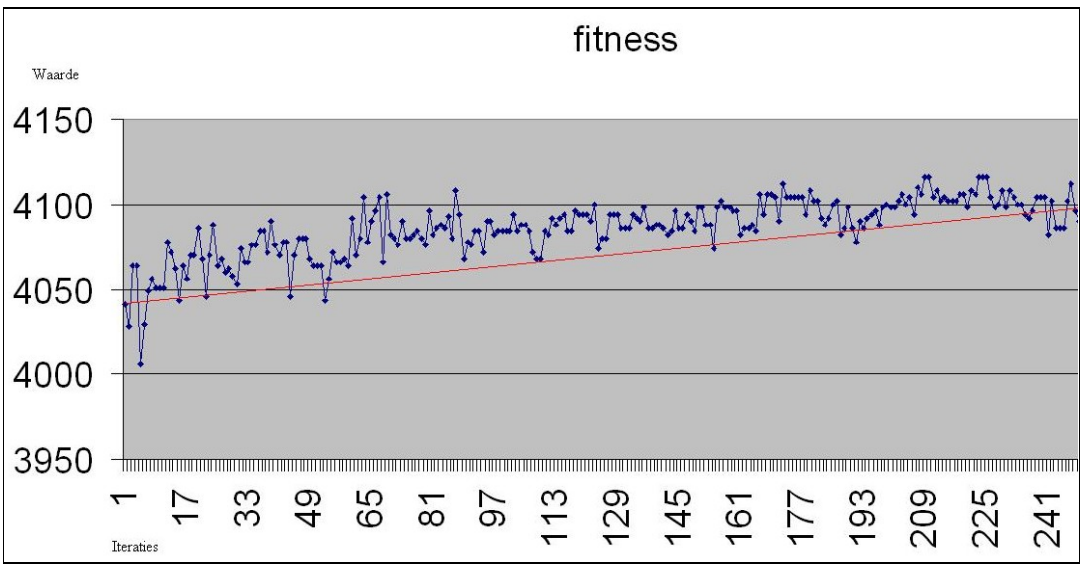
X = “TGTAGCTAGCTGATCGTCAT”
en
Y = “CACGTACGTACGAGCTAGTC”

Dan is H(X,Y) = 17

De Hamming afstand is ≥ 12 zodat deze twee strengen en hun Hamming afstand in de fitness functie opgenomen kunnen worden. Indien H(X,Y) kleiner zou zijn dan de Hamming_bound zal hun afstand niet deelnemen aan de fitness functie. Indien niet aan de eis voldaan werd, werd zelfs besloten de geschiktheid van de hele verzameling te straffen. De globale geschiktheid zou in dergelijke gevallen gewoonweg op 0 gezet worden wat de andere Hamming afstanden ook zouden zijn.

Dit heeft enkele zware gevolgen voor het algoritme. Het doel ervan is de slechte oplossingen te vermijden en dit blijkt te werken. Om dit te verduidelijken moet opnieuw de selectie methode erbij gehaald worden. De selectie methode van roulette wiel zal voor elk element in een populatie een procent bepalen dat de kans op selectie weergeeft. Dit procent is afhankelijk van de geschiktheid van het element. De slechte elementen hebben nu plots een geschiktheid die gelijk is aan 0. Hierdoor wordt hun kans op selectie ook 0 waardoor ze dus niet zullen gebruikt worden voor crossover en geheel verdwijnen in de volgende generaties.
We zullen eens concreet kijken naar het voorbeeld dat in de twee vorige pogingen gebruikt werd. We zullen de parameter opnieuw op 12 zetten. Zoals we in de vorige sectie hebben gezien zijn er Hamming afstanden die de waarde 0 hebben. Omdat 0 niet groter of gelijk is aan de Hamming_bound (12), wordt de globale geschiktheid op 0 gezet. Daarna wordt de selectie weer uitgevoerd en zal deze de oplossingsverzameling niet gebruikt worden voor het ontwikkelen van een nieuwe generatie en wordt dus weggegooid. ‘So far so good’.
De keuze van de parameter speelt dus een belangrijke rol. Een lage waarde zal veel mogelijke oplossingen toelaten. Deze mogelijke oplossingen kunnen dan niet veel van elkaar verschillen waardoor ze geen goede oplossing vormen. Een te hoge waarde zou heel weinig oplossingsverzamelingen toelaten en de geschiktheden allen op 0 zetten zodat ze geen nieuwe generatie kunnen tot stand brengen via het roulette wiel principe. Een goede keuze zal best via ‘trial and error’ gevonden worden.
Helaas treedt er opnieuw een probleem op. Wat als alle elementen in een populatie een geschiktheid hebben van grootte 0? Selectie zou niet kunnen plaatsvinden want alle elementen worden als slecht beschouwd en kunnen niet dienen als ‘ouders’ om nieuwe elementen te ontwikkelen. Deze situatie zal veel voorkomen, vooral als aan de nieuwe parameter een te hoge waarde wordt gegeven. Indien deze situatie zich

voordoet gebruiken we gewoon een andere selectie methode. We selecteren gewoon willekeurig twee elementen en geven deze door aan de crossover-operatie.
Wat als er maar één element is met een geschiktheid verschillend van 0. Dit ene element zal dan volledig instaan voor het ontwikkelen van een nieuwe generatie. Ook als het aantal degelijke elementen beperkt is zal een kleine groep verzamelingen instaan voor de verdere ontwikkeling van de populaties. Het is duidelijk dat de gevolgen voor het algoritme groot zijn. Er wordt een grote beperking opgelegd die de diversiteit zeker niet ten goede komt. Voor dergelijke situaties is het niet moeilijk een oplossing te bedenken maar er werd besloten te rekenen op de natuurlijkheid van de operaties om uiteindelijk genoeg diversiteit te verkrijgen die goede oplossingen zouden opleveren.
Door opnieuw te spelen met de parameters werd duidelijk dat het grote probleem van de twee vorige pogingen verdwenen was. De gedoemde oplossingen zoals te zien was in figuur 6.3 verdwenen compleet uit de resultaten alsook hun invloed op de evolutie van de ontwikkelde generaties.
Er ontstonden wel opnieuw problemen bij een verkeerde keuze van de nieuwe parameter. Vooral een te hoge waarde levert geen of slechte resultaten. Zoals reeds vermeld is dit niet verwonderlijk en ligt het probleem bij de slechte keuze van de parameter. Intuïtief werd de waarde van de parameter gekozen tussen 10 en 15. Het resultaat was dat er vaker mooie resultaten te voorschijn kwamen alsook een stijgende evolutie. Figuur 6.5 toont een grafiek zoals we zouden verwachten bij een goed genetisch algoritme.



Figuur 6.5: *De hoogste geschiktheid van elke generatie gedurende de uitvoering van het programma van 250 iteratie (generaties).*

In tabel 6.2 worden een aantal resultaten weergegeven van het programma GenGA (derde poging). Dit zijn de optima die het programma voor bepaalde waarden van de parameters heeft gevonden.

Tabel 6.2: *Parameters gebruikt bij het testen van GenGA (derde poging).*

	pop_size	mutation_rate	crossover_rate	elitism_rate	iterations	Hamming_bound	Best_Result	temperature %
1	50	0,2	1	0,2	150	12	3989	98
2	50	0,5	0,9	0,1	150	12	3967	98
3	150	0,4	1	0,1	150	12	4001	98
4	150	0,1	1	0,1	150	13	0	0
5	150	0,5	1	0,5	150	13	0	0
6	350	0,5	1	0,5	50	13	0	0
7	350	0,5	1	0,5	350	13	0	0
8	350	0,1	1	0,3	50	12	3949	98
9	350	0,1	1	0,3	350	12	3984	100
10	150	0,9	0,1	0,9	150	12	3751	88
11	150	0,9	0,1	0	150	12	3900	94
12	150	0,9	1	0	150	12	3997	98
13	1000	0,1	1	0	150	12	4072	100

Het beste bekomen resultaat wordt getoond in tabel 6.3:

Tabel 6.3: *Het meest optimale resultaat dat bekomn werd met GenGA (derde poging).*

true_1 (t_1)	5' CGCATGAATCCGCGCAGGGT 3'
true_2 (t_2)	5' GCTGAATCGGCTTTGGGATC 3'
true_3 (t_3)	5' AAACACGATTAGACACGCCA 3'
false_1 (f_1)	5' ACCTTACGTCCTACATCGAG 3'
false_2 (f_2)	5' AGTTAATACCGGGAATGTGT 3'
false_3 (f_3)	5' GAGTAACCAGAACAGTTTAG 3'
end_1 (e_1)	5' AACTCTCCTAACGTATGACG 3'
end_2 (e_2)	5' ATGCGTCCCATGCTACTTAT 3'
begin_2 (b_2)	5' GTGTTTGAAACCTGGTTACA 3'
begin_3 (b_3)	5' GATCAGTTGAATCTGCTTCA 3'
primer_1 (p_1)	5' CGCGGGGGTTGCCGCGCTAA 3'
primer_2 (p_2)	5' CAAAGATTCCATTACATCTA 3'

Dit resultaat zou een goede keuze zijn voor het uitvoeren van het 3-bit majority probleem in de praktijk. Wanneer we kijken naar de uniciteit van elke streng ten opzichte van de andere strengen levert dit volgende Hamming afstanden (tabel 6.4):

Tabel 6.4: *De Hamming afstanden tussen de verschillende DNA strengen onderling.*

H(1,2) = 18	H(3,4) = 15	H(5,10) = 16	H(10,11) = 16
H(1,3) = 16	H(3,5) = 14	H(5,11) = 17	H(10,12) = 13
H(1,4) = 14	H(3,6) = 17	H(5,12) = 15	H(11,12) = 16
H(1,5) = 13	H(3,7) = 13	H(6,7) = 13	
H(1,6) = 19	H(3,8) = 15	H(6,8) = 13	
H(1,7) = 17	H(3,9) = 16	H(6,9) = 13	

H(1,8) = 17	H(3,10) = 13	H(6,10) = 12
H(1,9) = 16	H(3,11) = 16	H(6,11) = 17
H(1,10) = 18	H(3,12) = 15	H(6,12) = 15
H(1,11) = 12	H(4,5) = 14	H(7,8) = 13
H(1,12) = 15	H(4,6) = 14	H(7,9) = 13
H(2,3) = 18	H(4,7) = 12	H(7,10) = 15
H(2,4) = 16	H(4,8) = 16	H(7,11) = 17
H(2,5) = 15	H(4,9) = 16	H(7,12) = 18
H(2,6) = 14	H(4,10) = 19	H(8,9) = 15
H(2,7) = 16	H(4,11) = 15	H(8,10) = 13
H(2,8) = 18	H(4,12) = 17	H(8,11) = 16
H(2,9) = 15	H(5,6) = 14	H(8,12) = 17
H(2,10) = 12	H(5,7) = 14	H(9,10) = 14
H(2,11) = 18	H(5,8) = 14	H(9,11) = 16
H(2,12) = 15	H(5,9) = 17	H(9,12) = 17

De uniciteit van de strengen onderling zou daarmee voldoende gedefinieerd moeten zijn. Ook de andere Hamming afstanden van de andere vereisten zullen dergelijke waarden vertonen.
Wat de smelttemperatuur van de stickers betreft, zijn deze 100% gelijk:

e_1b_2 = 5'AACTCTCCTAACGTATGACGGTGTTTGAAACCTGGTTACA 3'
e_2b_3 = 5' ATGCGTCCCATGCTACTTATGATCAGTTGAATCTGCTTCA 3'

T °C = 86.425°C

De uiteindelijke strengen die voor het experiment gesynthetiseerd moeten worden staan gegeven in tabel 6.5. Hierbij wordt er gekozen voor de bitstring 110 en 100 om het experiment mee uit te voeren.

Tabel 6.5: *De strengen die gesynthetiseerd moeten worden.*

$p_1t_1e_1$	5'CGCGGGGGTT GCCGCGCTAA CGCATGAATC CGCGCAGGGT AACTCTCCTA ACGTATGACG 3'
$b_2t_2e_2$	5' GTGTTTGAAA CCTGGTTACA GCTGAATCGG CTTTGGGATC ATGCGTCCCA TGCTACTTAT 3'
$b_2f_2e_2$	5' GTGTTTGAAA CCTGGTTACA AGTTAATACC GGGAATGTGT ATGCGTCCCA TGCTACTTAT 3'
$b_3f_3p_2$	5' GATCAGTTGA ATCTGCTTCA GAGTAACCAG AACAGTTTAG CAAAGATTCC ATTACATCTA 3'
$(e_1b_2)^C$	5' TTGAGAGGAT TGCATACTGC CACAAACTTT GGACCAATGT 3'
$(e_2b_3)^C$	5' TACGCAGGGT ACGATGAATA CTAGTCAACT TAGACGAAGT 3'
p_1^C	5' GCGCCCCCAA CGGCGCGATT 3'
p_2^C	5' GTTTCTAAGG TAATGTAGAT 3'

6.5 Utbreiding naar n-bit majority

Indien het programma wordt uitgebreid kan het niet alleen voor n = 3, maar voor alle mogelijke n gebruikt worden. Deze uitbreiding vereist de uitbreiding van de klasse “ElementStrings” (figuur 6.1). Deze klasse bevat de 4n strengen nodig voor het voorstellen van de input van het n-bit majority probleem. Zo zullen er dan 2n variabelen i.p.v. 6 zijn alsook 2(n-1) stickers i.p.v. van de 4. Het principe van het algoritme blijft hetzelfde, enkel de omvang van de berekeningen zal met de grootte van n toenemen.

6.6 Conclusie

Om de eerste stap in het algoritme (zie punt 4.4) op te lossen werd er gekozen een genetisch algoritme te ontwikkelen. Dit proces was zeker niet vanzelfsprekend. Een genetisch algoritme kan dan wel qua structuur gestandaardiseerd zijn, de implementatie van de verschillende onderdelen is afhankelijk van de keuze van de programmeur. Ook zal creativiteit een belangrijke rol spelen voor het al dan niet falen van het algoritme.
Het voorgestelde algoritme werd enkele keren verbeterd totdat het in een aanvaardbare oplossing (optimum) resulteerde. Het resultaat was een design van code words die als input kunnen dienen voor het 3-bit majority probleem.
Nu de strengen voor de input van het experiment bepaald zijn, mogen we de eerste stap van het algoritme afsluiten en verder gaan naar de volgende stap. Deze stap, net als alle volgende stappen, zijn reeds besproken in [27]. Ze beschrijven de protocollen die gebruikt werden tijdens het uitvoeren van de noodzakelijke experimenten in het laboratorium voor het 3-bit majority probleem.

Hoofdstuk VII: Het experiment (2002-2003)

7.1 Inleiding

Tijdens het academie jaar 2002-2003 hebben studenten aan het Limburgs Universitair Centrum reeds getracht het 3-bit majority probleem uit te voeren met DNA computing. Dit experiment kende helaas niet het gehoopte succes. Bij de keuze van de te gebruiken strengen werd er weinig aandacht geschonken aan het design. Het code word design werd weinig onderzocht en zou wel eens de belangrijkste reden kunnen zijn voor het falen. Dus reeds in de eerste stap zou het algoritme al verkeerd aangevat zijn. Dit resulteerde in het mislukken van de aanmaak van de input in stap 3 van het DNA algoritme (zie punt 4.5).

7.2 Resultaten

Tijdens het experiment dat uitgevoerd werd twee jaar geleden, werden de strengen voor de input zonder veel voorafgaand onderzoek gekozen uit de verzameling mogelijke strengen (4^{20}). Toch werd getracht aan de vooropgestelde eisen (zie hoofdstuk III) te voldoen. Hieronder worden enkele redenen geformuleerd die het falen van het experiment zouden kunnen verklaren bij de keuze van het DNA:

- De uniciteit van de strengen onderling zou een mogelijke oorzaak kunnen geweest zijn. In tabel 7.1 worden de Hamming afstanden van de strengen getoond.

Tabel 7.1: *De Hamming afstanden tussen de verschillende DNA strengen onderling.*

H(1,2) = 12	H(3,4) = 7	H(5,10) = 20	H(10,11) = 18
H(1,3) = 8	H(3,5) = 10	H(5,11) = 15	H(10,12) = 15
H(1,4) = 11	H(3,6) = 9	H(5,12) = 18	H(11,12) = 13
H(1,5) = 16	H(3,7) = 20	H(6,7) = 20	
H(1,6) = 11	H(3,8) = 20	H(6,8) = 20	
H(1,7) = 20	H(3,9) = 20	H(6,9) = 20	
H(1,8) = 20	H(3,10) = 20	H(6,10) = 20	
H(1,9) = 20	H(3,11) = 12	H(6,11) = 14	
H(1,10) = 20	H(3,12) = 16	H(6,12) = 16	
H(1,11) = 10	H(4,5) = 9	H(7,8) = 9	
H(1,12) = 13	H(4,6) = 14	H(7,9) = 7	
H(2,3) = 12	H(4,7) = 20	H(7,10) = 10	
H(2,4) = 11	H(4,8) = 20	H(7,11) = 17	
H(2,5) = 8	H(4,9) = 20	H(7,12) = 17	
H(2,6) = 7	H(4,10) = 20	H(8,9) = 12	

H(2,7) = 20	H(4,11) = 13	H(8,10) = 11
H(2,8) = 20	H(4,12) = 15	H(8,11) = 15
H(2,9) = 20	H(5,6) = 9	H(8,12) = 15
H(2,10) = 20	H(5,7) = 20	H(9,10) = 9
H(2,11) = 13	H(5,8) = 20	H(9,11) = 17
H(2,12) = 16	H(5,9) = 20	H(9,12) = 15

e_1 en b_2 (H(7,9)) hebben een Hamming afstand van 7. Hiervoor werden de volgende strengen gebruikt:

```
e1 = 5' GGTGTGGTGTTTGGTGTGGT 3'
          |  | |   |   ||      ||   | | ||   |
b2 = 5' GTTGGGTTGGGTGTTGTGTT 3'
```

Deze twee strengen hebben dus een Hamming afstand van slechts 7. Het is mogelijk dat deze twee strengen onderling zouden zijn gaan binden en zo de derde stap in het algoritme mede hebben doen falen. De lage Hamming afstanden (7, 8 of 9) zouden in staat zijn geweest een (zwakke) verboden binding te vormen in een bepaalde toestand.

- Ook de smelttemperatuur van de stickers kan in twijfel getrokken worden. Om de smelttemperatuur te berekenen werd de regel van Wallace (zie hoofdstuk I) gebruikt. Deze formule is slechts geschikt voor zeer kleine strengen en is bij dit experiment niet van toepassing.

- Hoewel er vaak geen rekening wordt gehouden met menselijke fouten en de kans hierop misschien zeer klein is, kan toch ook een foute handeling of contaminatie van de substanties een mogelijke oorzaak van het falen vormen.

7.3 Conclusie

De belangrijkste reden voor het falen van het experiment enkele jaren geleden, zou het code word design geweest zijn. Daarom kreeg dit in dit proefschrift alle aandacht. Helaas ontbreken de middelen om het resultaat van dat code word design na te gaan en zo misschien de redenen van het falen te kunnen achterhalen.

Hoofdstuk VIII: Algemene Conclusie

8.1 DNA computing

Berekeningen uitvoeren op moleculair niveau is na 10 jaar uitgegroeid tot een welvarend en opwindend onderzoeksdomein. Na de introductie van Adleman waaide er een nieuwe wind door de computerwereld die een nieuwe euforische toekomstvisie voor computers in het leven blies. Maar na een decennium van onderzoek is de euforie omgeslagen in realisme en objectiviteit. Veel en hard onderzoek hebben niet alleen de voordelen maar ook de beperkingen van DNA computing in kaart gebracht. Het domein heeft zich niet alleen tot DNA beperkt en is verder uitgegroeid tot een onderzoek over de mogelijkheden van alle aspecten van het biologische moleculaire niveau. Een onderdeel daarvan is het domein van het ontwerpen van degelijke input DNA voor een bepaald probleem. Code word design houdt zich hiermee bezig. Deze theoretische studie is reeds gevorderd en vormt al een eerste stap naar de praktijk.
Het is in ieder geval duidelijk geworden dat de constructie van een nieuwe generatie computers gebaseerd op DNA computing niet zomaar uit de grond zullen rijzen. Verder onderzoek naar de bruikbaarheid van DNA computing, zowel in de praktijk als in theorie, is vereist.

8.2 Evolutionaire Algoritmen

Om het code word design voor het 3-bit majority probleem uit te voeren werd gekozen om een programma te ontwikkelen gebaseerd op de theorie van de evolutionaire algoritmen. Dergelijke algoritmen gebruiken de kennis van de natuurlijke evolutie om een optimale oplossing te benaderen voor een bepaald probleem. Het wordt reeds gebruikt in tal van applicaties en kent een steeds verdere groei binnen het domein van de informatica.
Een genetisch algoritme is zo een evolutionair algoritme. Dit soort algoritme is eenvoudig te begrijpen en te implementeren. Toch is het ontwerpen ervan geen gemakkelijke opgave. Er moet met heel wat onzekerheden rekening gehouden worden alsook met een aantal parameters. Voor het ontwerp van een genetisch algoritme is een grondige kennis van het probleem waarvoor het algoritme ontwikkeld wordt, één vereiste. Een tweede vereiste is de creativiteit van de programmeur, vooral bij het ontwerpen van een geschikte fitness functie.

8.3 n-bit majority probleem en DNA computing

Oorspronkelijk was het ook de bedoeling DNA computing nog eens in de praktijk uit te voeren. Na de mislukking van twee jaar geleden was het de bedoeling dit experiment grondig voor te bereiden zodat de kans op slagen vergroot zou worden. Zoals hierboven staat vermeld is het design van de input een belangrijke, zo niet de belangrijkste stap om het experiment te kunnen aanvangen. Via een evolutionair algoritme werd gezocht naar een optimaal design. Dit algoritme werd zelf ontworpen om meer inzicht te krijgen in de moeilijkheden van het design. Het programma resulteert uiteindelijk in een goede verzameling DNA strengen die voor het 3-bit majority probleem gebruikt kunnen worden. Met dit programma kan de eerste stap in de oplossing van het 3-bit majority probleem overwonnen worden. Vanaf dan kan het experiment naar de praktijk omgezet worden en kunnen de voorgestelde stappen in het oplossingsalgoritme gevolgd worden, om tot een positief resultaat te komen. Ook een veralgemening van het programma naar het n-bit majority probleem is ter sprake gekomen.

Referenties

[1] Peter Marynen en Siska Waelkens. Het ABC van het DNA, mens en erfelijkheid. Davidsfonds/Leuven. 1996

[2] Afgeladen van: http://www.labbies.com/DNA.htm (15/07/2005).

[3] Pieter van Dooren. Klonen: mensen en dieren op bestelling Davidsfonds/Leuven. 1998.

[4] Neil A. Campbell, Jane B. Reece, Lawrence G. Mitchell. Biology (fifth edition). 1999

[5] Bruce Alberts, Dennis Bray, Alexander Johnson. Essential Cell Biology. 1997

[6] Afgeladen van http://www.agr.kuleuven.ac.be/dp/logt/gentech1-hybridis.DOC (16/07/'05)

[7] Agrinfo-Fevia-Oivo-VIB. Lespakket Biotechnologie. 1999

[8] Afgeladen van: www.stanford.edu/.../ diagnsis/gentest/s7.html (20/07/'05)

[9] Ivo Safarik, Mirka Safarikova. Magnetic techniques for the isolation and purification of proteins and peptides. BioMagnetic Research and technology 2004

[10] Leonard M. Adleman: Molecular Computations of solutions to combinatorial problems. 1994

[11] Richard J. Lipton: DNA solution of hard computational problems. 1995

[12] Jonoska N, Karl SA, Saito M.: Three dimensional DNA structures in computing. 1999

[13] Afgeladen van de University of Florida, department of mathematics: http://www.math.usf.edu/~jonoska/talknsf/ (16/07/'05)

[14] Erzsebet Csuhajvarju, Rudolf Freund, Lila Kari, Gheorghe Paun: DNA Computing based on SplicingUniversality Results. 1996

[15] Mitsunori Ogihara, Animesh Ray: Simulating Boolean circuits on a DNA computer. 1997

[16] John H. Reif: Parallel molecular computation: models and simulations. 1995

[17] Martyn Amos: Theoretical and experimental DNA computation. 2004

[18] Martyn Amos, Alan Gibbons, David Hodsgon: error-resistant implementation of DNA computations. 1998

[19] Thomas Head: Splicing schemes and DNA. 1992

[20] George Paun, Grzegorz, Arto Salomaa: DNA computing: new Computing Paradigms. 1998

[21] Paul W.K. Rothemund, Erik Winfree: The program complexity of self-assembled squares. 1999

[22] George Paun: Membrane computing: an introduction. 2002

[23] De web-pagina voor P systems: http://psystems.disco.unimib.it/ (13/07/'05)

[24] Juris Hartmanis: On the weight of computations. 1995

[25] Ken Komiya, Kensaku Sakamoto, Hidetaka Gouzu, Shigeyuki Yokoyama, Masa,ori Arita, Akio Nishikawa, Masami Hagiya: Succesive state transitions with IO interface by molecules. 2001

[26] Takashi Yokomori, Yasubumi Sakakibara, Satoshi Kobayashi: A magic pot: self-assembly computation revisited. 2002

[27] Ward Conings, Kurt Van Brabant: DNA in de praktijk ("One-pot" computing) 2003

[28] E. Winfree, X. Yang, N.C. Seeman: Universal Computation via Self-assembly of DNA: Some Theory and Experiments. 1999

[29] Erik Winfree , John J. Hopfield: Algorithmic self-assembly of dna. 1998

[30] Yaakov Benenson, Binyamin Gil2, Uri Ben-Dor1, Rivka Adar2 en Ehud Shapiro: An autonomous molecular computer for logical control of gene expression. 2001

[31] http://www.thinkquest.org afgeladen van: http://library.thinkquest.org/TQ0312650/areyou.htm (12/07/'05)

[32] Jennifer Sager Darko Stefanovic: Designing Nucleotide Sequences for Computation: A Survey of Constraints. 2005

[33] Masanori Arita: Writing information into DNA.

[34] Amit Marathe, Anne E. Condon, Robert M. Corn: On Combinatorial DNA word Design. 1999

[35] Mirela Andronescu, Danielle Dees, Laura Slaybauh, Yinglei Zhao, Anne Condon, Bary Cohen, Steven Skiena: Algorithms for testing that sets of DNA words concatenate without secondary structures.

[36] Sotirios A. Tsaftaris, Aggelos K. Katsaggelos, Thrasyvoulos N. Pappas, Eleftherios T. Papoutsakis: DNA Computing from a Signal Processing Viewpoint

[37] Masanori Arita: Comma-Free Design for DNAWords: Considering data-coding procedures and sequence design.

[38] Randal E. Bryant : Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams 1992

[39] John Holland: Adaptation in natural and artificial systems. 1975

[40] Melanie Mitchell: An introduction to genetic algorithms.

[41] A.E. Eiben, J.E. Smith: Introduction to evolutionary computing.

[42] Afgeladen van:http://www.talkorigins.org/faqs/genalg/genalg.html (05/05/'05)

[43] Caruana Rich, Schaffer J. David: Using Multiple Representations to Control Inductive Bias: Gray and Binary Codes for Genetic Algorithms. 1989

[44] Tackett W. : Recombination, Selection, and the Genetic Construction of Computer Programs. 1994

[45] Charles C. Palmer, Aaron Kershenbaum: Representing Trees in Genetic Algorithms. 1994

[46] Franz Rothlauf, David Goldberg, and Armin Heinzl: Network random keys: A tree network representation scheme for genetic and evolutionary algorithms. 1999

[47] Lawrence Davis: Handbook of Genetic Algorithms. 1991

[48] Definitie van Inversion, afgeladen van: http://ghr.nlm.nih.gov/ghr/glossary/inversion (15/02/'05)

[49] J. Schaffer, A. Morishima: An adaptive crossover distribution mechanism for genetic algorithms.1987

[50] Kalyanmoy Deb, David E. Goldberg: mGA in C: A Messy Genetic Algorithm in C. 1991

[51] David E. Goldberg, B. Korb, K. Deb: Messy genetic algorithms: Motivation 1989

[52] Yee Leung, Yong Gao, Zong-Ben Xu: Degree of population diversity – a perspective on premature convergence in genetic algorithms and its Markov chain analysis. 1997

[53] Biman Chakraborty1and Probal Chaudhuri: On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis. 2003

[54] Lino Costa, Pedro Oliveira: An elitist genetic algorithm for multiobjective optimization Source. 2004

[55] Shummet Baluja: Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning Source. 1994

[56] David Noever and Subbiah Baskaran: Steady-State vs. Generational Genetic Algorithms : A Comparison of Time Complexity and Convergence Properties 1992

[57] William M. Spears, Kenneth A. De Jong: An Analysis of Multi-Point Crossover. 1991

[58] David M. Tate, Alice E. Smith: Expected Allele Coverage and the Role of Mutation in Genetic Algorithms. 1993

[59] Thomas Bäck: Optimal Mutation Rates in Genetic Search. 1993

[60] William M. Spears: Adapting Crossover in Evolutionary Algorithms. 1995

[61] Jonatan Gómez, Dipankar Dasgupta, Fabio González: Using Adaptive Operators in Genetic Search.

[62] Robert Hinterding, Zbigniew Michalewicz, A.E. Eiben: Adaptation in Evolutionary Computation: A Survey. 1997

[63] J. Schaffer, R. A. Caruana, L. J. Eshelman, R. Das: A study of control parameters affecting online performance of genetic algorithms for function optimization.

[64] Kalyanmoy Deb, Samir Agrawal: Understanding Interactions Among Genetic Algorithm Parameters. 1998

[65] L. Davis: Adapting operator probabilities in genetic algorithms. 1989

[66] Kalyanmoy Deb: Multi-objective optimization using evolutionary algorithms

[67] Afgeladen van: http://neo.lcc.uma.es/opticomm/eccdescription.html: Error correcting codes description (N.E.O.) (12/02/'05)