T.C. van Barneveld

Besliskunde in Tennis

Optimaal serveren in tennis

Het plannen van een tennistoernooi

Bachelorscriptie, 14 juni 2009

Scriptiebegeleider: prof.dr. L.C.M. Kallenberg

Mathematisch Instituut, Universiteit Leiden

Voorwoord

Voor u ligt de eindscriptie van mijn bachelorproject. Dit project heb ik onder begeleiding van prof. dr. L.C.M. Kallenberg gedurende de maanden februari-juni 2009 gedaan. Het vakgebied is Besliskunde. Het hebben van ver gevorderde voorkennis is voor het lezen niet vereist: alles wordt redelijk elementair uitgelegd. Bij de hoofdstukken waar enige voorkennis wel vereist is, heb ik dat aangegeven. Deze scriptie bestaat uit twee delen:

1. In de hoofdstukken 1 t/m 8 wordt er ingegaan op het probleem van optimaal serveren: het is bekend dat een tennisser twee services heeft: een harde en een langzame, en dat een tennisser in een punt van de partij twee keer mag opslaan. In dit gedeelte wordt ingegaan hoe de tennisser het beste kan serveren om de partij te kunnen winnen. Voor meer informatie over het probleem verwijs ik naar hoofdstukken 1, 2 en 3. In hoofdstuk 4 wordt een manier om dit probleem op te lossen behandeld: Dynamische programmering. Dit blijkt voor dit probleem een verrassend eenvoudige methode te zijn, waarbij onder enkele logische aannames de optimale tactiek eenvoudig te bepalen is. In hoofdstuk 5 zullen we, met behulp van de computer, deze aannames rechtvaardigen. In de hoofdstukken 6 en 7 wordt een andere manier behandeld: Markov beslissingstheorie. Hiervoor is enige kennis van grafen- en Markovtheorie vereist. Hoofdstuk 6 concentreert zich op de achterliggende theorie, terwijl in hoofdstuk 7 deze theorie op ons probleem wordt toegepast. Hoofdstuk 8 ten slotte, behandelt een simulatiemethode om een tennispartij te simuleren.

2. In de hoofdstukken 9 t/m 13 concentreren we ons op het optimaal plannen van een tennistoernooi onder een aantal voorwaarden. Zie hiervoor de hoofdstukken 9 en 10. In hoofdstuk 11 zullen we ingaan op het Maximum Kliek probleem. Ons planningsprobleem blijkt namelijk zo geformuleerd te kunnen worden. Er zullen in dit hoofdstuk drie oplossingsmethoden behandeld worden. Enige kennis van grafentheorie is hiervoor dus vereist. In hoofdstuk 12 zullen we deze methoden toepassen op het planningsprobleem. Tot slot zullen we in hoofdstuk 14 de complexiteit van dit probleem onderzoeken, waarbij we ook de wereldberoemde stelling van Cook zullen bewijzen. Voor dit hoofdstuk is het hebben van elementaire kennis van de complexiteitstheorie een pré.

Bij beide delen hoort ook een voordracht. Voor beide delen heb ik veel geprogrammeerd in MATLAB. Wegens de omvang van deze programma's, heb ik besloten deze niet op te nemen in de scriptie. Wel heb ik enkele voorbeelden van de uitvoer van deze programma's vermeld. Mocht men toch deze programma's willen ontvangen om te testen, dan kan men mij mailen op tbarneve@math.leidenuniv.nl. Tot slot bedank ik prof. dr. L.C.M. Kallenberg voor zijn goede begeleiding tijdens dit project. Rest mij u veel leesplezier te wensen.

Thije Christiaan van Barneveld
14 juni 2009

Inhoudsopgave

1 Inleiding optimaal serveren: Tennis, ontstaan en regels

Tennis is een balsport voor twee (enkelspel) of vier (dubbelspel) spelers waarbij een kleine bal met een racket over een net moet worden gespeeld. Tennis is ontstaan in Engeland en wordt sinds 1873 in zijn huidige vorm gespeeld. De naam is afgeleid van het Frans "Tenez!": "Houd (de bal)!" Bij een wedstrijd moet de bal binnen de speelhelft van de tegenstander worden geslagen met als doel het de tegenstander onmogelijk te maken terug te slaan over het net en binnen het eigen speelveld. Zie [1]

Bij een tenniswedstrijd worden er maximaal drie (of vijf) *sets* gespeeld. Wie als eerste twee (of drie) sets gewonnen heeft, wint de partij. Een set bestaat uit een aantal *games*. De speler die als eerste zes games heeft gewonnen, wint de set. Hij moet dit echter wel met minimaal twee games verschil doen, anders wordt er nog een game gespeeld. Is de stand 6-6, dan volgt er een *tie-break*. In een game geldt de volgende puntentelling: 0, 15, 30, 40, game. Hierbij moeten de spelers met minimaal twee punten verschil winnen. Als het dus 40-40 staat, worden er nog minimaal twee punten gespeeld. Deze stand wordt dan ook wel *deuce* genoemd. In een set serveren de spelers om en om een game lang. Per punt heeft de speler die aan service is twee services. Als de *eerste service* fout is, mag hij het nog een keer proberen. Is ook deze *tweede service* fout, dan gaat het punt naar de tegenstander.

Bij een tie-break serveren beide spelers: eerst serveert een speler één keer, dan de ander twee keer, dan de één twee keer etc. Ook hier hebben de spelers twee services. Wie als eerste zeven punten heeft behaald, wint de tie-break en dus ook de set. Dit moet echter wel met een verschil van minimaal twee punten zijn, anders speelt men door net zolang tot een speler twee punten meer heeft dan zijn tegenstander.

2 Probleemstelling: Hoe optimaal te serveren?

We beschouwen een simpel model, waarin beide spelers kunnen kiezen uit twee typen services: type 1 (*harde service*) en type 2 (*langzame service*). De harde service gaat vaker mis, maar is moeilijker te retourneren als deze goed is; de langzame service is betrouwbaarder, maar makkelijker te retourneren.
De vraagstelling luidt: gegeven de score in de game en of je een eerste of een tweede service moet slaan, wat is de beste strategie om de game te winnen?

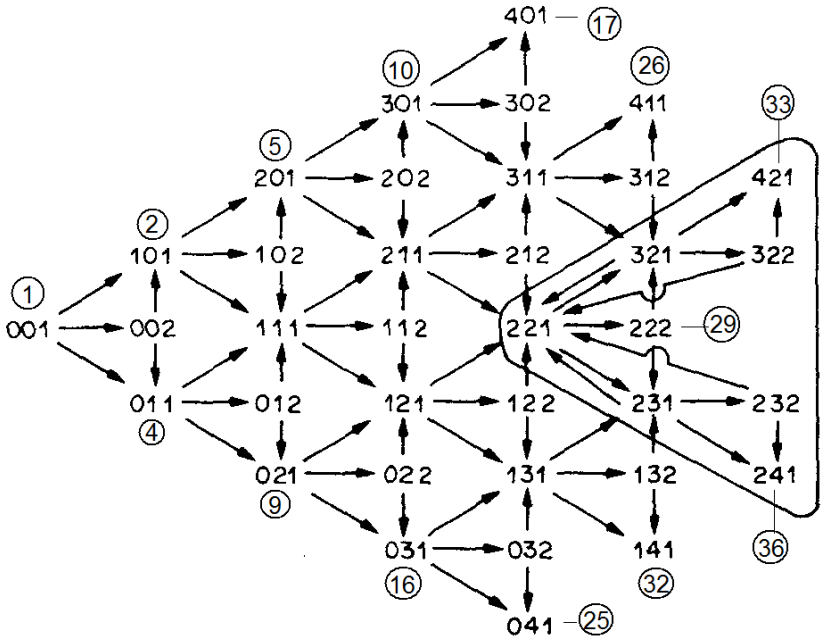
3 Het model

We gaan bovenstaand probleem modelleren.

Hiervoor definiëren we het volgende:

- Laat p_1 resp. p_2 de kans zijn dat de harde resp. langzame service goed is.
- Laat q_1 resp. q_2 de kans dat het punt gewonnen wordt, gegeven dat de harde resp. langzame service goed is.

Het ligt voor de hand aan te nemen dat $p_1 < p_2$. Een langzame service heeft immers meer kans om goed te zijn dan een harde service. Een tweede logische aanname is de volgende: $q_1 > q_2$. Immers, een harde service is moeilijker te retourneren en de kans op puntwinst is groter. Het scoresysteem (0, 15, 30, 40, game) zullen we ook versimpelen. Beschouw dit in termen van de integers van 0 tot 4. Als de stand 40-40 is, dan wordt de game gewonnen door de speler die als eerste twee punten meer heeft dan zijn tegenstander, dus 40-40 is equivalent met 30-30. In ons 0-4-scoresysteem is 40-40, evenals 30-30, equivalent met 2-2.



Figuur 1: De toestanden van een tennisgame en hun overgangen [2]

In ons model zijn er 36 toestanden. Deze zijn weergegeven in de figuur 1. Bij de knooppunten staan twee getallen: een omcirkeld getal en een getal bestaande uit drie cijfers. Bij deze driecijferige getallen, staat het eerste getal voor de score van de serveerder (in ons 0-4-scoresysteem), het tweede voor de score van de tegenspeler en het laatste voor de hoeveelste service geslagen wordt (eerste of tweede). De pijlen geven de overgangen weer. Om het één en ander overzichtelijk te houden, hernummeren we de toestanden met de omcirkelde cijfers die er boven dan wel onder staan. Om tot een oplossing van dit probleem te komen, zullen we dit model op twee manieren analyseren: eerst gebruiken we de techniek van *dynamische programmering*, zie hoofdstuk 4, en daarna zullen we methoden uit de *Markovbeslissingstheorie* toepassen, zie hoofdstukken 6 en 7.

3.1 Parameterwaarden

Om het één en ander te kunnen testen, zullen we realistische waarden nodig hebben voor p_1, p_2, q_1 en q_2. Er zijn echter erg weinig statistieken te verkrijgen over de snelheid van de service en dan met name hoe vaak een harde dan wel langzame service wordt toegepast. Om toch tot een goede schatting voor parameterwaarden te komen, nemen we aan dat proftennissers over het algemeen hard op hun eerste en langzaam op hun tweede service serveren. We gaan parameterwaarden schatten voor Roger Federer (SUI) en Rafael Nadal (SPA), in het begin van 2009 de resp. nummer 2 en 1 van de wereld. We baseren de waarden op de finale van de Australian Open 2009, die Nadal in vijf sets wist te winnen (7-5, 3-6, 7-6, 3-6, 6-2). Relevante statistieken zijn in de volgende tabel gezet:

	R. Nadal	R. Federer
1^e service goed	112 van 175 = 64%	90 van 172 = 52%
Dubbele fouten	4	6
Winst op 1^e service	74 van 112 = 66%	66 van 90 = 73%
Winst op 2^e service	30 van 63 = 48%	37 van 82 = 45%

Wedstrijdstatistieken van de Australian Open finale tussen Nadal en Federer. [3]

We kunnen uit deze tabel gelijk waarden voor p_1 en q_1 schatten, als we gebruik maken van onze aanname. We zien dan dat:

$$\text{Voor Nadal: } p_1 = 0.64 \text{ en } q_1 = 0.66$$
$$\text{Voor Federer: } p_1 = 0.52 \text{ en } q_1 = 0.73$$

Voor de andere waarden moeten we meer werk doen. Uit de tabel volgt dat Nadal $175 - 112 = 63$ tweede services heeft geslagen. Daarvan waren er 4 fout. Hij heeft dus 59 tweede services goed

geslagen. Dit komt overeen met 94%. Op dezelfde manier berekenen we dat Federer 93% van zijn tweede services goed heeft geslagen. Nu hebben we al onze waarden, gebruikmakend van onze aanname:

$$\text{Voor Nadal: } p_2 = 0.94 \text{ en } q_2 = 0.48$$
$$\text{Voor Federer: } p_2 = 0.93 \text{ en } q_2 = 0.45$$

NB: Waarden voor q_1 en q_2 hangen altijd af van twee personen. Immers, als Nadal tegen een amateurtennisser zou tennissen, zou hij veel minder moeite hebben een punt te winnen dan tegen Federer, ongeacht of de service goed is. Zijn q-waarden liggen tegen een amateurtennisser dan een stuk hoger. Hierdoor kan men geen echt 'profiel' van de tennisser maken.

4 Dynamische programmering

Dynamische programmering is een techniek die vaak met succes toegepast kan worden bij problemen die een dynamisch karakter hebben. Kenmerkend bij dynamische programmering is dat voor het oplossen van het probleem een *recursieve formulering* wordt opgesteld. Deze techniek zullen we op een slimme manier toepassen op het probleem. Zie ook [2] en [4]. Laat $m - h$ $(m, h = 0, 1, 2, 3, 4)$ de huidige score is het versimpelde scoresysteem zijn en y $(y = 1, 2)$ het nummer van de service die geslagen gaat worden. We definiëren $f(m, h, y)$ als de kans om de game te winnen op score $m - h$ met de y-de service te slaan, gebruikmakend van een optimale strategie. Onze recursieformule ziet er dan als volgt uit:

$$f(m, h, 1) = \max_{k=1,2}\{p_k q_k f(m+1, h, 1) + p_k(1-q_k)f(m, h+1, 1) + (1-p_k)f(m, h, 2)\}$$

$$f(m, h, 2) = \max_{k=1,2}\{p_k q_k f(m+1, h, 1) + (1-p_k q_k)f(m, h+1, 1)\}$$

Immers, op de eerste service kan de serveerder

- In serveren en het punt winnen (met kans $p_k q_k$).
- In serveren en het punt verliezen (met kans $p_k(1-q_k)$).
- Uit serveren en een tweede service nodig hebben (met kans $1-p_k$).

Op de tweede service wint of verliest de serveerder het punt met respectievelijk kans $p_k q_k$ en $(1-p_k q_k)$. Het is duidelijk dat $f(4, \cdot, \cdot) = 1$ en $f(\cdot, 4, \cdot) = 0$.
Merk op dat er per punt vier strategieën zijn. Deze geven we weer in de volgende tabel.

	Service	Service	
Strategie	1	2	Kans op puntwinst
1	Hard	Langzaam	$p_1 q_1 + p_2 q_2\{1-p_1\}$
2	Hard	Hard	$p_1 q_1\{2-p_1\}$
3	Langzaam	Langzaam	$p_2 q_2\{2-p_2\}$
4	Langzaam	Hard	$p_2 q_2 + p_1 q_1\{1-p_2\}$

Mogelijke strategieën. [4]

Herinner dat $p_1 < p_2$ en $q_1 > q_2$. Uit deze aanname volgt dat strategie 4 (Langzaam, Hard) nooit optimaal is, want de kans om met die strategie het punt te winnen is altijd minder dan de kans om met strategie 1 (Hard, Langzaam) het punt te winnen. Dit is als volgt in te zien:

$$\mathbb{P}\{\text{Kans op puntwinst} \mid \text{strategie 1}\} - \mathbb{P}\{\text{Kans op puntwinst} \mid \text{strategie 4}\} =$$
$$p_1 q_1 + p_2 q_2\{1-p_1\} - p_2 q_2 - p_1 q_1\{1-p_2\} =$$
$$-p_1 p_2 q_2 + p_2 p_1 q_1 > 0$$

Delen door p_1p_2 geeft $q_1 < q_2$, dus deze bewering volgt enkel al uit deze aanname. Strategie 4 zullen we dus in het vervolg buiten beschouwing laten. Om de optimale tactiek te bepalen, zou men van alle 36 verschillende toestanden de $f(m, h, y)$ kunnen bepalen voor beide typen services, en daarna kijken bij welke service de kans op puntwinst op stand $f(m, h, y)$ het grootst is. Het kan echter veel makkelijker: hierbij maken we gebruik van de volgende twee aannames:

$$f(m+1, h, 1) > f(m, h+1, 1)$$
$$f(m, h, 2) > f(m, h+1, 1).$$

Waarom we dit aannemen is eenvoudig is te zien: De eerste aanname zegt dat de kans op gamewinst voor de serveerder groter is als hij het punt wint dan wanneer hij het punt verliest. Dit idee is intuïtief duidelijk. De tweede aanname zegt dat de kans op gamewinst voor de serveerder groter is als hij een tweede service mag slaan, dan wanneer de tegenstander het punt al behaald heeft. Ook dit is een logische aanname. Beschouw nu de zeven verticale rijtjes in figuur 1. Volgens onze aannames is de kans op gamewinst strict dalend in de toestanden van deze rijtjes. Dat dit idee correct is, bewijzen we elders (numeriek).

Eerst beschouwen we, geheel tegen de verwachtingen in, de tweede service. Het is beter om hard te serveren dan en slechts dan als:

$$f_1(m, h, 2) - f_2(m, h, 2) > 0,$$

waarbij $f_1(m, h, 2)$ en $f_2(m, h, 2)$ resp. corresponderen met de kans op puntwinst op score $m - h$ met de tweede service te slaan wanneer een harde service wordt gebruikt dan wel wanneer een langzame service wordt gebruikt. Bovenstaande ongelijkheid is te reduceren tot:

$$p_1q_1 f(m+1, h, 1) + (1 - p_1q_1) f(m, h+1, 1) - p_2q_2 f(m+1, h, 1) - (1 - p_2q_2) f(m, h+1, 1) \text{ i.e.}$$
$$(p_1q_1 - p_2q_2) f(m+1, h, 1) + [(1 - p_1q_1) - (1 - p_2q_2)] f(m, h+1, 1) > 0,$$
$$\text{i.e. } (p_1q_1 - p_2q_2)(f(m+1, h, 1) - f(m, h+1, 1)) > 0,$$
$$\text{i.e. } p_1q_1 - p_2q_2 > 0.$$

Merk op dat bij de laatste reductie aanname 1 is gebruikt.
Beschouw nu de eerste service. Hier is het beter om hard te serveren dan en slechts dan als:

$$f_1(m, h, 1) - f_2(m, h, 1) > 0$$

Dit is te reduceren tot:

$$p_1q_1 f(m+1, h, 1) + p_1(1 - q_1) f(m, h+1, 1) + (1 - p_1) f(m, h, 2) - p_2q_2 f(m+1, h, 1)$$
$$-p_2(1 - q_2) f(m, h+1, 1) - (1 - p_2) f(m, h, 2) \text{ i.e.}$$
$$(p_1q_1 - p_2q_2)[f(m+1, h, 1) + [p_1(1 - q_1) - p_2(1 - q_2)] f(m, h+1, 1) + [(1 - p_1) - (1 - p_2)] f(m, h, 2) > 0,$$
$$\text{i.e. } (p_1q_1 - p_2q_2)[f(m+1, h, 1) - f(m, h+1, 1)] + (p_1 - p_2)[f(m, h+1, 1) - f(m, h, 2)] > 0.$$

Merk aan deze vergelijking het volgende op:

- $f(m+1, h, 1) - f(m, h+1, 1) > 0$ wegens aanname 1.
- $(p_1 - p_2) < 0$ wegens onze eerdere aanname.
- $f(m, h+1, 1) - f(m, h, 2) < 0$ wegens aanname 2.

We zien nu dat als onze eerder verkregen ongelijkheid $(p_1q_1 - p_2q_2) > 0$ geldt, dan geldt ook de bovenstaande (lange) ongelijkheid. Hieruit volgt dat als het beter is om hard te serveren op de tweede service, het dan ook beter is om hard te serveren op de eerste service.

Bekijk nu het geval dat $(p_1q_1 - p_2q_2) < 0$. Het is duidelijk dat het dan beter is om langzaam te serveren op de tweede service. Ook geldt dat:

$$f(m, h, 2) = p_2q_2 f(m+1, h, 1) + (1 - p_2q_2) f(m, h+1, 1).$$

Op de eerste service is het dan beter om hard te serveren dan en slechts dan als (met de vergelijking voor $f(m,h,2)$ alvast ingevuld):

$$(p_1q_1 - p_2q_2)[f(m+1,h,1) - f(m,h+1,1)] + (p_1 - p_2)[f(m,h+1,1) - p_2q_2f(m+1,h,1) \\ -(1-p_2q_2)f(m,h+1,1)] > 0$$

Men kan eenvoudig verifiëren dat deze ongelijkheid te reduceren is tot de volgende:

$$p_1q_1[f(m+1,h,1) - f(m,h+1,1)] - p_2q_2[f(m+1,h,1) + f(m,h+1,1)] \\ +(p_1 - p_2)[p_2q_2f(m,h+1,1) - p_2q_2f(m+1,h,1)] > 0 \\ \text{i.e. } p_1q_1 - p_2q_2(1 + p_1 - p_2) > 0$$

Omgekeerd, het is beter om langzaam te serveren op de eerste service als

$$p_1q_1 - p_2q_2(1 + p_1 - p_2) < 0$$

Nu hebben we een optimale strategie gevonden:

Pas strategie 1 (Hard, Langzaam) toe als
$(p_1q_1 - p_2q_2) < 0$ en $p_1q_1 - p_2q_2(1 + p_1 - p_2) > 0$, i.e. $1 > \frac{p_1q_1}{p_2q_2} > 1 + (p_1 - p_2)$.
Pas strategie 2 (Hard, Hard) toe als
$(p_1q_1 - p_2q_2) > 0$, i.e. $\frac{p_1q_1}{p_2q_2} > 1$.
Pas strategie 3 (Langzaam, Langzaam) toe als
$(p_1q_1 - p_2q_2) < 0$ en $p_1q_1 - p_2q_2(1 + p_1 - p_2) < 0$, i.e. $1 + (p_1 - p_2) > \frac{p_1q_1}{p_2q_2}$.

Voorbeeld 4.1 *R. Nadal vs. R. Federer*
Beschouw de partij tussen Nadal en Federer. We berekenen:

$$\textit{Voor Nadal: } \frac{p_1q_1}{p_2q_2} = 0.94 \textit{ en } 1 + (p_1 - p_2) = 0.70. \\ \textit{Voor Federer: } \frac{p_1q_1}{p_2q_2} = 0.91 \textit{ en } 1 + (p_1 - p_2) = 0.59.$$

We zien dat voor beide spelers strategie 1 (Hard, Langzaam) het best is, maar eigenlijk hadden we bij het schatten van de parameterwaarden dit ook aangenomen. Verder zien we dat voor zowel Nadal als voor Federer strategie 2 (Hard, Hard) beter uit zou pakken dan strategie 3 (Langzaam, Langzaam). Ook valt op dat het interval van strategie 1 voor Nadal kleiner is dan dat voor Federer.

Het is opmerkelijk dat de beslissingen in de optimale strategie onafhankelijk zijn van de score in de game. Men zou verwachten dat men, als men bijvoorbeeld met 40-0 voorstaat, meer risico kan nemen en vaker de harde service toe kan passen. Dit blijkt dus niet het geval te zijn.

5 Winstkansen

In het vorige hoofdstuk hebben we de volgende aannames gedaan:

$$f(m+1,h,1) > f(m,h+1,1) \\ f(m,h,2) > f(m,h+1,1).$$

Intuïtief is duidelijk dat dit juist is, maar dit zullen we hier gaan aantonen. Om dit te bewijzen, zullen we het model nog een beetje versimpelen: we beschouwen slechts één service, zodat onze Markovbeslissingsketen een (gewone) Markovketen wordt.

Om per toestand deze f-waarden, m.a.w. kansen op gamewinst, te bepalen, maken we gebruik van een stelling uit de Markovketentheorie. Hiervoor moeten we echter eerst een begrip definiëren:

Definitie 5.1 *Absorptiekans*
Zij $(X_0, X_1, \ldots)$ een discrete Markovketen met toestandsruimte S en overgangsmatrix P. De absorptiekans *is de kans dat de keten, startend in toestand i, ooit in toestand j komt. Notatie: f_{ij}.*

Stelling 5.1 *Absorptiekans*
$f_{ij} - \sum_{k \neq j} p_{ij} f_{kj} = p_{ij}$ *voor alle* $i, j \in S$.

Voor het bewijs, zie [5]. Merk op dat we zes *absorberende toestanden* hebben. Deze zijn, in ons hernummering, de toestanden 17, 25, 26, 32, 33 en 36. Dit zijn de toestanden waarbij de game afgelopen is. Komt men in toestand 17, 26 of 33, dan heeft men de game gewonnen. De kans op gamewinst is dus de som van de kansen om in toestand 17, 26 en 33 te absorberen. We willen dus $f_{i,17} + f_{i,26} + f_{i,33}$ berekenen voor $i \in S$. Dit doen we met behulp van bovenstaande stelling. We schrijven de oplossing echter niet met de hand uit, maar gebruiken hiervoor het computerpakket MATLAB. Hier volgt nu een beschrijving van het geschreven programma:

We maken een (36×36)-matrix P behorende bij de overgangskansen. Daarna maken we drie (36×36)-matrices aan: f17, f26 en f33, behorende bij resp. $f_{i,17}, f_{i,26}$ en $f_{i,33}$. Beschouw de matrix f17. Deze heeft enen op de diagonaal: dit zijn de $f_{i,17}$-waarden die we willen berekenen. Verder wordt deze matrix de tegengestelde van de overgangsmatrix. Deze matrix stelt in de stelling onze linkerterm voor. Op dezelfde manier maken we f26 en f33.

Ook maken we vectoren vector17, vector26 en vector33, elk met lengte 36. Beschouw vector17. Voor de elementen van deze vector geldt dat ze allemaal 0 zijn, behalve voor de toestanden waaruit men toestand 17 rechtstreeks kan bereiken, in dit geval dus de toestanden 10 en 18. Op deze plekken worden $P_{10,17}$ resp. $P_{18,17}$ geplaatst. Evenzo definiëren we vector26 en vector33. In onze stelling is dit de rechterterm.

We hebben nu drie stelsels van elk 36 vergelijkingen met 30 onbekenden. Immers, voor de absorberende toestanden weten we de f-waarden. Deze zijn 0 of 1. Deze stelsels kunnen we dus oplossen. De drie verkregen oplossingsvectoren tellen we bij elkaar op, en dan krijgen we een vector met per toestand de kans op gamewinst. Zie bijlage A.
Merk op dat de elementen van de kansenvector stuksgewijs dalende rijtjes getallen zijn, dus ons in het vorige hoofdstuk gedane aanname lijkt correct.

6 Markov Beslissingstheorie: Theorie

Dit hoofdstuk geeft een beknopte inleiding in de Markov Beslissingstheorie. Er worden enkel enige definities en resultaten behandeld. Voor bewijzen van deze resultaten, verwijs ik u door naar [6].

6.1 Het model

Eerst geven we, als korte herhaling, de definitie van een eindige, discrete en tijdshomogene Markovketen. In dit artikel zullen we enkel dit soort Markov (beslissingsketens) behandelen.

Definitie 6.1 *(Eindige, discrete, tijdshomogene Markovketen)*
Zij P *een* $N \times N$ *-matrix met elementen* $\{p_{ij} : i, j = 1, ..., N\}$. *Een random proces* $(X_0, X_1, ...)$ *met eindige toestandsruimte* $S = \{1, ..., N\}$ *heet een* (eindige, discrete, tijdshomogene) Markovketen *als, voor alle* $t \in \mathbb{N}_0$ *en alle* $i, j \in \{1, ..., N\}$ *en alle* $i_0, ..., i_{n-1} \in \{1, ..., N\}$ *geldt dat:*

$$\begin{aligned} &\mathbb{P}\{X_{t+1} = j | X_0 = i_0, X_1 = i_1, ..., X_{t-1} = i_{t-1}, X_t = i\} \\ &= \mathbb{P}\{X_{t+1} = j | X_t = i\} \\ &= p_{ij}. \end{aligned}$$

P *heet de* overgangsmatrix.

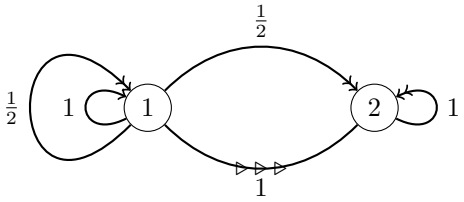
Een *Markovbeslissingsketen* is, losjes gezegd, in feite niets anders dan een stapel Markovketens. We beschouwen de tijdsstippen $t = 1, 2, ...$ De Markov beslissingsketen heeft dezelfde *toestandsruimte* S als de 'stapel' Markovketens waaruit hij bestaat. Het grote verschil tussen de Markovketen en

de Markov beslissingsketen, zit in het feit dat voor iedere toestand $i \in S$ er een *actieverzameling* $A(i)$ is. Elementen uit deze actieverzameling corresponderen met de Markovketen die *gekozen* kan worden (vandaar de naam *beslissings*keten). Iedere Markovketen uit de 'stapel' heeft zijn eigen *overgangsmatrix*. De overgangskans van de verschillende toestanden in de Markovbeslissingsketen, noteren we als $p_{ij}(a)$: hiermee wordt de overgangskans van toestand i naar toestand j $(i, j \in S)$, met als gekozen Markovketen de keten a $(\in A(i))$ bedoeld. Tot slot definiëren we een *directe opbrengst* $r_i(a)$, waarbij $i \in S$, $a \in A(i)$. Deze opbrengst wordt uitgekeerd zodra actie $a \in A(i)$ wordt gekozen.

Definitie 6.2 *(Markovbeslissingsketen)*
Laat X_t en Y_t de toestand en actie op tijdstip t aanduiden. Een Markovbeslissingsketen *is een viertal objecten $(S, A, p_{ij}(a), r_i(a))$, met*

- *S de toestandsruimte*
- *A de actieruimte (de verzameling Markovketens)*
- *$p_{ij}(a) = \mathbb{P}\{X_{t+1} = j | X_t = i, Y_t = a\}$*
- *$r_i(a)$ de directe opbrengst als in toestand i, actie $a \in A(i)$ gekozen wordt.*

Voorbeeld 6.1 *(Markovbeslissingsketen)*



We zien dat:
$S = \{1, 2\}$
$A(1) = \{1, 2, 3\}$, $A(2) = \{1\}$
$p_{11}(1) = 1$, $p_{11}(2) = \frac{1}{2}$, $p_{12}(2) = \frac{1}{2}$, $p_{12}(3) = 1$, $p_{22}(1) = 1$
$r_1(1) = 2$, $r_1(2) = -1$, $r_1(3) = -1$, $r_2(1) = 0$ (niet getekend).

6.2 Strategieën

Bij een Markovbeslissingsketen wil men de opbrengst zien te maximaliseren. Dit is een niet-triviaal probleem. Het middel hiervoor, is het verkrijgen van een *voorschrift* voor de te kiezen actie in een willekeurige toestand. Zo'n voorschrift voor alle toestanden tezamen heet een *beslisregel*.

Definitie 6.3 *(Beslisregel)*
Een beslisregel $\pi : S \to A$ *geeft de kans, als functie van de toestandsruimte naar de actieruimte om een bepaalde actie te kiezen. Voor iedere π_{ia}, met $i \in S$ en $a \in A(i)$, geldt bovendien dat:*

- *$\pi_{ia} \geq 0$ voor iedere $i \in S$ en $a \in A(i)$*
- *$\sum_a \pi_{ia} = 1$ voor alle $i \in S$*

In het algemeen is men niet geïnteresseerd in één enkele beslisregel, maar in een aantal van deze beslisregels, voor elk tijdstip een afzonderlijke. Zo'n rij beslisregels wordt een *strategie* genoemd. Strategieën noteren we als $R = (\pi^1, \pi^2, ...)$, met π^t de beslisregel op tijdstip t. Als in zo'n strategie geen enkele beslisregel van het verleden afhangt, alleen van het heden (de Markoveigenschap), heet zo'n strategie een *Markovstrategie*.

Definitie 6.4 *(Stationaire deterministische Markovstrategie)*
Een strategie is een stationaire deterministische Markovstrategie *als het volgende geldt:*

- *Iedere beslisregel is stationair, m.a.w. hangt niet van de tijd af.*

- *Iedere beslisregel is deterministisch, m.a.w.* $\pi_{ia} \in \{0,1\}$ *voor iedere* $i \in S$ *en* $a \in A(i)$.
- *Iedere beslisregel hangt enkel van de toestand af waarin het systeem op dat moment verkeert.*

Iedere strategie die we vanaf nu beschouwen is een stationaire deterministische Markovstrategie. Gemakshalve zullen we deze gewoon strategie noemen. Een stationaire deterministische Markovstrategie wordt volledig bepaald door een functie $f : S \to A$: aan iedere toestand wordt immers een voorgeschreven actie gekoppeld. Op deze manier gedraagt de Markovbeslissingsketen zich als een gewone Markovketen. Omdat we enkel stationaire deterministische Markovstrategieën behandelen, zullen we in het vervolg een beslisregel met f aanduiden.

Definitie 6.5 *(Overgangsmatrix en opbrengstvector)*
Zij f *een beslisregel. We definiëren:*

1. *De matrix* $P(f)$*, met* $\{P(f)\}_{ij} = p_{ij}\{f(i)\}$ *voor iedere* $(i,j) \in S \times S$*, heet de* overgangsmatrix.
2. *De vector* $r(f)$*, met* $r(f)_i = r_i\{f(i)\}$ *voor iedere* $i \in S$*, heet de* opbrengstvector.

Voorbeeld 6.2 *(Vervolg)*
Zij f *in bovenstaand voorbeeld de volgende beslisregel:* $f(1) = 2$, $f(2) = 1$.
Dan zien we dat $P(f) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{pmatrix}$ *en* $r(f) = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$.

Laat toestand i de begintoestand zijn en R de gevolgde strategie. Dit brengt ons bij de volgende stelling:

Stelling 6.1 *Voor een strategie* $R = (f, f, ...)$ *en begintoestand* i *geldt:*

1. $\mathbb{P}_{i,R}\{X_t = j, Y_t = a\} = (P^{t-1}(f))_{ij}$ *voor* $(j,a) \in S \times A$.
2. $\mathbb{E}_{i,R}\{r_{X_t}(Y_t)\} = (P^{t-1}(f)r(f))_i$.

Voorbeeld 6.3 *(Vervolg)*
Laat toestand 1 *de begintoestand zijn en* $R = (f, f, \ldots)$ *de gevolgde strategie, met* f *als hierboven gedefinieerd.*

Dan zien we dat $\mathbb{P}_{1,R}\{X_3 = 1, Y_3 = 2\} = \frac{1}{4}$.

Ook is duidelijk dat $\mathbb{E}_{1,R}\{r_{X_3}(Y_3)\} = -\frac{1}{4}$.

In woorden: op tijdstip $t = 3$ *is de kans om in toestand 1 te zijn en actie 2 te kiezen gelijk aan* $\frac{1}{4}$ *en de verwachte opbrengst is* $-\frac{1}{4}$.

6.3 Optimalisatiecriterium

Er zijn verschillende *optimalisatiecriteria* voor de Markovbeslissingsketen. Het optimalisatiecriterium dat we hier zullen behandelen, is dat van *totale verwachte opbrengsten over een oneindige horizon*. Met oneindige horizon wordt bedoeld dat de Markovbeslissingsketen oneindig lang door kan gaan. De *totale verwachte opbrengst*, gegeven begintoestand i en strategie R, noteren we met $v_i(R)$ en wordt gedefinieerd door

$$v_i(R) = \sum_{t=1}^{\infty} \mathbb{E}_{i,R}\{r_{X_t}(Y_t)\} = \sum_{t=1}^{\infty} \sum_{j,a} \mathbb{P}_{i,R}\{X_t = j, Y_t = a\} \cdot r_j(a).$$

Deze totale verwachte opbrengst hoeft niet goed gedefinieerd te zijn. Beschouw de volgende Markovbeslissingsketen:

Voorbeeld 6.4 *(Totale verwachte opbrengsten niet goed gedefinieerd)*
$S = \{1, 2\}; A(1) = 1, A(2) = 2; p_{12}(1) = 1, p_{21}(2) = 1;$
$r_1(1) = 1, r_2(2) = -1; f(1) = 2, f(2) = 1.$
Als deze keten, beginnend in toestand 1 oneindig lang loopt, krijgt men een opbrengst van $1 - 1 + 1 - 1 + \ldots$. *De verwachte opbrengsten over een oneindige horizon zijn dan niet goed gedefinieerd, want deze kunnen* 0 *of* 1 *zijn.*

Om bovenstaand euvel te verhelpen, doen we twee aannames:

1. Het model is *substochastisch*, m.a.w. $\sum_j p_{ij}(a) \leq 1$ voor alle $(i, a) \in S \times A$.

2. Iedere strategie is *transiënt*, d.w.z. $\sum_{t=1}^{\infty} \mathbb{P}_{i,R}\{X_t = j, Y_t = a\} < \infty$ voor alle i, j en a. M.a.w., de kansen 'doven uit'.

Aanname 2 zorgt ervoor dat bovenstaand geval niet kan voorkomen. Aanname 1 is echter een voorwaarde voor de geldigheid van aanname 2, dus ook deze is van belang.

Nu we ervoor hebben gezorgd dat de totale verwachte opbrengsten over een oneindige horizon goed gedefinieerd zijn, definiëren we de *waardevector* v en *optimale strategie* R_* door:

$$v_i = \sup_R v_i(R), i \in S \text{ en } v_i(R_*) = v_i, i \in S.$$

Er kan worden aangetoond dat er een optimale stationaire deterministische strategie f^∞ bestaat. [6]

6.4 Optimalisatiemethoden

Er zijn verschillende rekenmethoden ontwikkeld om een optimale strategie van een Markovbeslissingsketen, i.e. per toestand een optimale beslisregel, te bepalen. Ik zal drie van deze methoden behandelen: de methodes van *strategie verbetering*, *lineaire programmering* en *waarde iteratie*.

6.4.1 Strategie verbetering

In de methode van *strategie verbetering* wordt een rij van strategieën $f_1^\infty, f_2^\infty, \cdots$ geconstrueerd zodanig dat:

$$v(f_{k+1}^\infty) > v(f_k^\infty) \text{ voor } k = 1, 2, ...$$

De $>$-relatie houdt hier in: $\geq$ voor iedere i, en $>$ voor minstens één i, $i \in S$. Omdat er slechts een eindig aantal stationaire deterministische Markovstrategieën is, is deze methode ook eindig. Bij dit idee maken we gebruik van de volgende verzameling $A(i, f)$, die de *ruimte van verbeterende acties* wordt genoemd. Het idee is om een actie $f(i)$ te vervangen door een actie $a \in A(i, f)$, net zolang tot $A(i, f) = \emptyset$ voor alle $i \in S$. $A(i, f)$ wordt als volgt gedefinieerd:

$$A(i, f) := \{a \in A(i) | r_i(a) + \sum_j p_{ij}(a) v_j(f^\infty) > v_i(f^\infty)\}$$

Dit idee is correct [6]. Het algoritme ziet er als volgt uit:

Algoritme 6.1 *Strategie verbetering*

1. Start met willekeurige f^∞.

2. Bepaal $r(f)$ en $P(f)$.

3. Bereken $v(f^\infty)$ als de unieke oplossing van het stelsel $r(f) + P(f)x = x$.

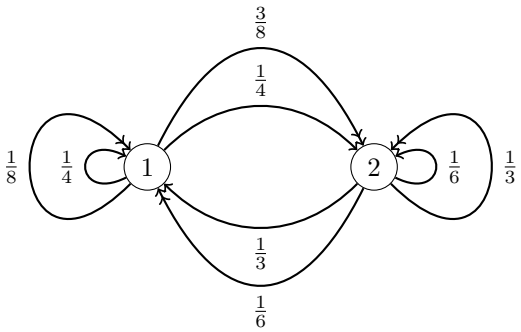
4. **a.** Bereken $s_{ia}(f) := r_i(a) + \sum_j p_{ij}(a)v_j(f^\infty) - v_i(f^\infty)$ voor iedere $(i, a) \in S \times A$.

   b. Bepaal $A(i, f) = \{a \in A(i) | s_{ia}(f) > 0\}$ voor iedere $i \in S$.

5. Als $A(i, f) = \emptyset$ voor iedere $i \in S$: f^∞ is een optimale strategie (STOP).
   Anders: Bepaal $s_{ig(i)}(f) = \max_a s_{ia}(f)$, $i \in S$ als $g(i) \neq f(i)$.

6. $f := g$ en ga naar stap 2.

In het volgende voorbeeld is te zien hoe dit algoritme wordt toegepast op het volgende voorbeeld:



Voorbeeld 6.5 *Strategie verbetering*
Beschouw $S = \{1, 2\}$; $A(1) = A(2) = \{1, 2\}$.
$r_1(1) = 1, r_1(2) = 0$; $r_2(1) = 2, r_2(2) = 2$.
$p_{11}(1) = \frac{1}{4}, p_{12}(1) = \frac{1}{4}$; $p_{11}(2) = \frac{1}{8}, p_{12}(2) = \frac{3}{8}$.
$p_{21}(1) = \frac{1}{3}, p_{22}(1) = \frac{1}{6}$; $p_{21}(2) = \frac{1}{6}, p_{22}(2) = \frac{1}{3}$.
Merk op dat dit model substochastisch is.

Iteratie 1

1. We starten met $f(1) = f(2) = 1$.

2. $P(f) = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{6} \end{pmatrix}$ en $r(f) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

3. Oplossen van het stelsel geeft $v(f) = (\frac{32}{13}, \frac{44}{13})$.

4. **a.** We berekenen: $s_{11}(f) = 0$, $s_{12}(f) = -\frac{23}{26}$; $s_{21}(f) = 0$, $s_{22}(f) = \frac{2}{13}$.
   b. We zien dat $A(1, f) = \emptyset$, $A(2, f) = \{2\}$.

5. We nemen $g(1) = 1, g(2) = 2$, want $s_{22}(f) > s_{21}(f)$.

6. Bovenstaande g wordt onze nieuwe f.

Iteratie 2

1. We starten met $f(1) = 1, f(2) = 2$.

2. $P(f) = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{6} & \frac{1}{3} \end{pmatrix}$ en $r(f) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

3. Oplossen van het stelsel geeft $v(f) = (\frac{28}{11}, \frac{40}{11})$.

4. **a.** We berekenen: $s_{11}(f) = 0$, $s_{12}(f) = -\frac{19}{22}$; $s_{21}(f) = -\frac{2}{11}$, $s_{22}(f) = 0$.
   b. We zien dat $A(1, f) = \emptyset$, $A(2, f) = \emptyset$.

5. f^∞ met $f(1) = 1, f(2) = 2$ is een optimale strategie, met waardevector $v = (\frac{28}{11}, \frac{40}{11})$.

6.4.2 Lineaire programmering

Definitie 6.6 *Superharmonisch*
Een vector $v \in \mathbb{R}^n$ *heet* superharmonisch *als* $v_i \geq r_i(a) + \sum_j p_{ij}(a)v_j$ *voor alle paren* (i,a) *met* $i \in S$ *en* $a \in A(i)$.

Stelling 6.2 *De waardevector* v *is de (componentsgewijs) kleinste superharmonische vector.*

[6] Een gevolg hiervan is dat v de unieke oplossing is van het volgende lineaire programmeringsprobleem:

$$\text{LP} = min\left\{\sum_j \beta_j v_j \;\middle|\; \sum_j \{\delta_{ij} - p_{ij}(a)\}v_j \quad \geq r_i(a) \text{ voor alle } (i,a) \in S \times A \right\}$$

met $\beta_j > 0$ voor alle $j \in S$, $S \times A = \{(i,a) | i \in S, a \in A(i)\}$ en δ_{ij} de Kronecker delta.

Het bijbehorende duale probleem is:

$$\text{DLP} = max\left\{\sum_{(i,a)} r_i(a)x_i(a) \;\middle|\; \begin{array}{l} \sum_{(i,a)}\{\delta_{ij} - p_{ij}(a)\}x_i(a) = \beta_j, j \in S \\ x_i(a) \geq 0 \text{ voor alle } (i,a) \in S \times A \end{array} \right\}$$

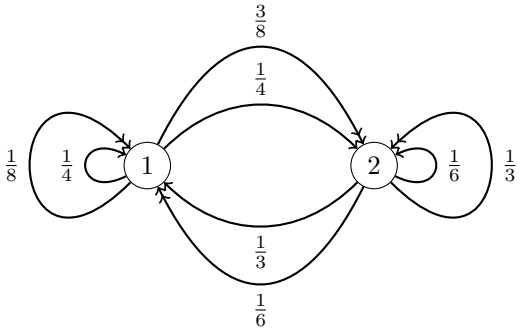
Stelling 6.3 *Laat* x^* *een optimale oplossing zijn van DLP.*
Dan is de deterministische strategie f_* *met* $x_i^*(f_*(i)) > 0, i \in S$, *goed gedefinieerd en een optimale strategie.*

Voor het bewijs, zie [6].

Algoritme 6.2 *Lineaire programmering*

1. Neem een vector β met $\beta_j > 0$, $j \in S$.
2. Bepaal optimale oplossingen v^* en x^* van respectievelijk LP en DLP.
3. Neem f_*^∞ zdd. $x_i^*(f_*(i)) > 0$ voor iedere $i \in S$.
   v^* is de waardevector v en f_*^∞ is een optimale strategie (STOP).

We zullen het algoritme toepassen op hetzelfde voorbeeld als hierboven.



Voorbeeld 6.6 *Lineaire Programmering*
Beschouw $S = \{1,2\}$; $A(1) = A(2) = \{1,2\}$.
$r_1(1) = 1, r_1(2) = 0$; $r_2(1) = 2, r_2(2) = 2$.
$p_{11}(1) = \frac{1}{4}, p_{12}(1) = \frac{1}{4}$; $p_{11}(2) = \frac{1}{8}, p_{12}(2) = \frac{3}{8}$.
$p_{21}(1) = \frac{1}{3}, p_{22}(1) = \frac{1}{6}$; $p_{21}(2) = \frac{1}{6}, p_{22}(2) = \frac{1}{3}$.
Merk op dat dit model substochastisch is.

1. We nemen $\beta_1 = \beta_2 = \frac{1}{2}$.
2. $$\text{LP} = min\left\{\frac{1}{2}v_1 + \frac{1}{2}v_2 \;\middle|\; \begin{array}{llll} \frac{3}{4}v_1 & -\frac{1}{4}v_2 & \geq & 1 \\ \frac{1}{8}v_1 & -\frac{3}{8}v_2 & \geq & 0 \\ -\frac{1}{3}v_1 & +\frac{5}{6}v_2 & \geq & 2 \\ -\frac{1}{6}v_1 & +\frac{2}{3}v_2 & \geq & 2 \end{array} \right\}$$

$$\text{DLP} = max\left\{x_1(1) + 2x_2(1) + 2x_2(2) \;\middle|\; \begin{array}{lllll} \frac{3}{4}x_1(1) & +\frac{7}{8}x_1(2) & -\frac{1}{3}x_2(1) & -\frac{1}{6}x_2(2) & = \frac{1}{2} \\ -\frac{1}{4}x_1(1) & -\frac{3}{8}x_1(2) & +\frac{5}{6}x_2(1) & +\frac{2}{3}x_2(2) & = \frac{1}{2} \\ x_1(1), & x_2(1), & x_1(2), & x_2(2) & \geq 0 \end{array} \right\}$$

Het LP-probleem heeft als oplossing $v^* = (\frac{28}{11}, \frac{40}{11})$.
De oplossing van het DLP-probleem is: $x^* = (x_1(1), x_2(1), x_1(2), x_2(2)) = (\frac{9}{10}, 0, 0, \frac{11}{10})$.

3. Merk op dat $x_1(1) > 0$ en $x_2(2) > 0$, dus de optimale strategie f_*^∞ is: $f_*(1) = 1, f_*(2) = 2$.
   De waardevector is $(\frac{28}{11}, \frac{40}{11})$.

Merk op dat de methodes van strategie verbetering en van lineaire programmering dezelfde oplossing geven.

6.4.3 Waarde iteratie

Bij de methode van *waarde iteratie* wordt de waardevector v in iedere iteratie steeds beter benaderd. We stoppen als deze voldoende dicht benaderd is. Het algoritme ziet er nu als volgt uit:

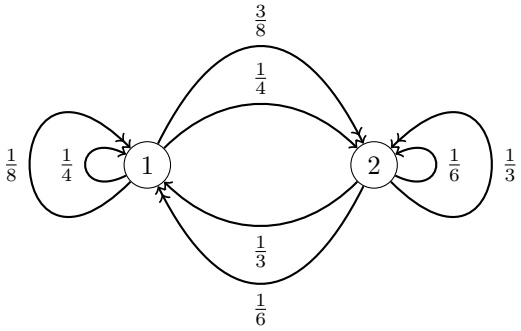
Algoritme 6.3 *Waarde iteratie*

1. Kies $\varepsilon > 0$ en $x \in \mathbb{R}^N$ willekeurig.
2. **a.** Bereken $y_i = \max_a \{r_i(a) + \sum_j p_{ij}(a)x_j\}, i \in S$.

   b. Zij $f(i) = argmax_a\{r_i(a) + \sum_j p_{ij}(a)x_j\}, i \in S$.
3. Als $\sup_k |y_k - x_k| \leq \varepsilon$, $1 \leq k \leq N$: f^∞ is een optimale strategie en y een approximatie van de waardevector v (STOP).

   Anders: $x := y$ en ga naar stap 2.

We passen dit algoritme toe op hetzelfde voorbeeld.



Voorbeeld 6.7 *Waarde iteratie*
Beschouw $S = \{1, 2\}$; $A(1) = A(2) = \{1, 2\}$.
$r_1(1) = 1, r_1(2) = 0$; $r_2(1) = 2, r_2(2) = 2$.
$p_{11}(1) = \frac{1}{4}, p_{12}(1) = \frac{1}{4}$; $p_{11}(2) = \frac{1}{8}, p_{12}(2) = \frac{3}{8}$.
$p_{21}(1) = \frac{1}{3}, p_{22}(1) = \frac{1}{6}$; $p_{21}(2) = \frac{1}{6}, p_{22}(2) = \frac{1}{3}$.
Merk op dat dit model substochastisch is.

Iteratie 1

1. We kiezen $\varepsilon = 0.2$ en $x = (2, 3)$.
2. **a.** We berekenen $y = (\frac{9}{4}, \frac{10}{3})$.

   b. We zien dat $f(1) = 1$ en $f(2) = 2$.
3. Er geldt dat $\sup_k |y_k - x_k| = \dfrac{3}{10} \not< 0.2 = \varepsilon$, dus we doen nog een iteratie.

Iteratie 2

1. We hebben $\varepsilon = 0.2$ en $x = (\frac{9}{4}, \frac{10}{3})$.
2. **a.** We berekenen $y = (\frac{115}{48}, \frac{251}{72})$.

   b. We zien dat $f(1) = 1$ en $f(2) = 2$.

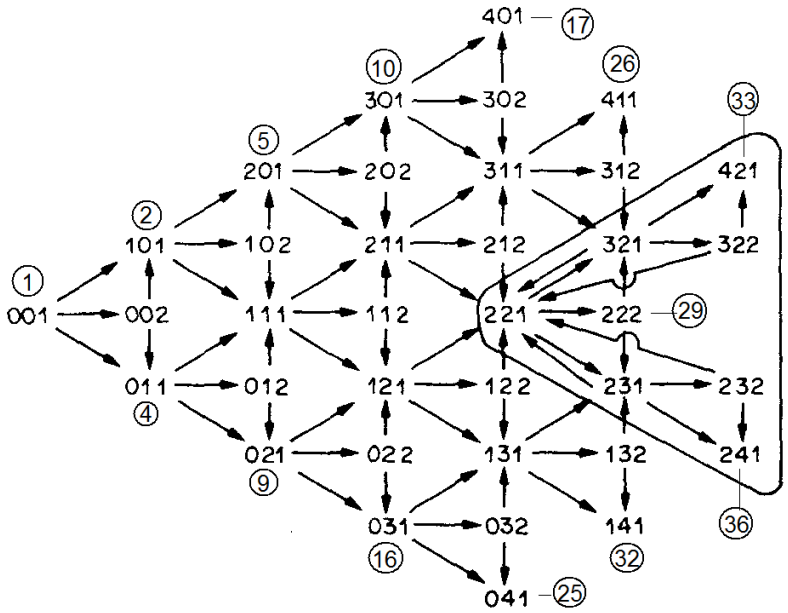
3. Er geldt dat $\sup_k |y_k - x_k| = \frac{11}{72} \approx 0.15 < 0.2 = \varepsilon$, dus we zijn klaar. De optimale strategie is $f(1) = 1, f(2) = 2$. Een approximatie van de waardevector v is $(\frac{115}{48}, \frac{251}{72})$.

Merk op dat dit antwoord overeenkomt met de eerder verkregen resultaten van de andere twee algoritmen.

7 Markov Beslissingstheorie: Toepassing

7.1 Het model

Het model ziet er als volgt uit, waarbij de nummering van de toestanden die van de figuur is.



Figuur 2: De toestanden van een tennisgame en hun overgangen [2]

$S = \{1, ..., 36\}$ en $A(i) = \{1, 2\}, i \in S$ waarbij 1 en 2 corresponderen met de resp. harde en langzame service. We hebben de volgende overgangskansen:

$$p_{ij}(k) = \begin{cases} p_k q_k & \text{voor pijlen schuin omhoog,} \\ p_k(1-q_k) & \text{voor pijlen schuin omlaag,} \\ 1-p_k & \text{voor pijlen horizontaal naar rechts,} \\ p_k q_k & \text{voor pijlen recht naar boven,} \\ 1-p_k q_k & \text{voor pijlen recht naar beneden,} \\ 0 & \text{anders.} \end{cases}$$

voor $k = 1, 2$; $i, j \in S$.

Verder zijn de opbrengsten overal 0, behalve bij de gewonnen toestanden, i.e. de toestanden 17, 26 en 33. Daar is de opbrengst gelijk aan 1. We willen de *totale verwachte opbrengsten* maximaliseren. Merk op dat deze Markovbeslissingsketen *stationair* is. Ook is het model *transiënt*. Bovendien geldt voor de gezochte strategie:

- De gezochte strategie is *deterministisch*, i.e. voor een optimale strategie staat per toestand de beslissing die genomen moet worden, vast.
- De gezochte strategie is *Markov*, i.e. alle beslisregels hangen enkel af van de toestand waarin het systeem zich op dit moment bevindt.

7.2 De rekenmethode

We gebruiken de methode van *waarde iteratie* voor het bepalen van de optimale strategie in het tennisprobleem. We gebruiken deze methode, omdat deze methode het makkelijkst te programmeren is. We werken de iteraties van dit algoritme dus niet met de hand uit: hiervoor gebruiken we het computerpakket MATLAB. Hier volgt nu een beschrijving van het computerprogramma.

We beschouwen 37 in plaats van 36 toestanden. Deze 37^e toestand kan men opvatten als het einde van de game. Vanuit een gewonnen of verloren stand gaat men altijd met kans 1 naar deze 37^e toestand door middel van de zogenaamde *derde actie*. We definiëren de volgende vier (37×37)-matrices:

P: De matrix met overgangskansen behorende bij de *harde service*.

Q: De matrix met overgangskansen behorende bij de *langzame service*.

R: De matrix van opbrengsten. Hier geldt $R_{17,37} = R_{26,37} = R_{33,37} = 1$ en alle andere opbrengsten zijn 0.

T: De matrix van de derde actie. Hier geldt $T_{17,37} = T_{25,37} = T_{26,37} = T_{32,37} = T_{33,37} = T_{36,37} = 1$ en alle andere overgangskansen zijn 0.

We nemen voor x de nulvector en voor ε nemen we de waarde 0.0001. We definiëren drie hulpvectoren, voor elke actie één, ter lengte 37: $y1, y2$ en $y3$. Hierin stoppen we de waarden van resp. P, Q en T vermenigvuldigd met $x +$ (de bijbehorende rij uit de opbrengstmatrix). Van deze drie vectoren nemen we per component het maximum van $y1, y2$ en $y3$. Dit stoppen we in een nieuwe vector y (stap 2a). Ook onthouden we voor welk van de drie hulpvectoren ($y1, y2$ en $y3$) het maximum wordt aangenomen. Dit stoppen we in een vector f (stap 2b).

Hierna vergelijken we x met y. Als $\sup_i |x_i - y_i| > \varepsilon$, dan wordt x onze nieuwe y en herhalen we het recept. Als $\sup_i |x_i - y_i| \leq \varepsilon$, dan stoppen we. Onze f-vector is dan een optimale strategie en y is een benadering van de waardevector, i.e. de vector met de kans op gamewinst per toestand van het systeem, onder een optimale strategie.

Voorbeeld 7.1 *R. Nadal vs R. Federer*
Als we het MATLAB-programma draaien met de parameterwaarden voor Federer en Nadal, dan wordt berekend dat voor beide spelers de strategie (Hard, Langzaam) optimaal is. Voor Nadal $(p_1 = 0.64, q_1 = 0.64, p_2 = 0.94$ *en* $q_2 = 0.48)$ *geeft het MATLAB-programma de uitwerking in bijlage B.*
Dit komt overeen met het eerder verkregen antwoord via dynamische programmering.

Na veelvuldig uitvoeren van dit programma, zien we dat de via dynamische programmering gevonden optimale strategie exact hetzelfde is als de hier gevonden optimale strategie. We hebben nu echter ook uitsluitsel over de *randpunten* 1 en $1 + (p_1 - p_2)$.

- Pas strategie 1 (Hard, Langzaam) toe als $1 > \frac{p_1 q_1}{p_2 q_2} \geq 1 + (p_1 - p_2)$.
- Pas strategie 2 (Hard, Hard) toe als $\frac{p_1 q_1}{p_2 q_2} \geq 1$.
- Pas strategie 3 (Langzaam, Langzaam) toe als $1 + (p_1 - p_2) > \frac{p_1 q_1}{p_2 q_2}$.

8 Simulatie

Nu we een optimale strategie hebben bepaald, kunnen we het één en ander gaan simuleren, bijvoorbeeld de partij tussen Rafael Nadal en Roger Federer. Met deze optimale strategie is de Markovbeslissingsketen immers een gewone Markovketen met overgangsmatrix P geworden. Deze P is dezelfde P als de overgangsmatrix van de Markovbeslissingsketen. Deze Markovketen kunnen we simuleren. Dit doen we met behulp van een zogenaamde *update functie* $\phi : S \times [0,1] \to S$. Deze update functie heeft de volgende eigenschappen:

- Voor vaste $i \in S$ is de functie $\phi(i,x)$ stuksgewijs constant.
- Voor elke $i,j \in S$ is de totale lengte van de intervallen waarop $\phi(i,x) = j$ gelijk aan P_{ij}.

Voor $x \in [0,1]$ gebruiken we een random getalgenerator: deze trekt telkens uniform random een getal tussen 0 en 1. Iedere toestand heeft zijn eigen update functie. We zullen de update functie van de toestanden 1, 3 en 17 geven. Alle update functies voor toestanden waarbij de eerste service geslagen wordt, zijn analoog aan die van toestand 1; alle update functies voor toestanden waarbij de tweede service wordt geslagen, zijn analoog aan die van toestand 3 en alle update functies voor absorberende toestanden zijn analoog aan die van toestand 17:

$$\phi(1,x) = \begin{cases} 2 & \text{voor } \ x \in [0, pq) \\ 3 & \text{voor } \ x \in [pq, pq+1-p) \\ 4 & \text{voor } \ x \in [pq+1-p, 1] \end{cases}$$

$$\phi(3,x) = \begin{cases} 2 & \text{voor } \ x \in [0, pq) \\ 4 & \text{voor } \ x \in [pq, 1] \end{cases}$$

$$\phi(17,x) = \begin{cases} 17 & \text{voor } \ x \in [0,1] \end{cases}$$

Simulatie van deze Markovketen doen we met behulp van het computerpakket MATLAB. Gevraagd wordt om de invoer van de parameters voor beide spelers. Hiermee wordt de optimale strategie bepaald door middel van de gevonden voorschriften. Dit geeft de overgangsmatrix P. De begintoestand van de keten is toestand 1. Er wordt een eerste random getal getrokken. Dit bepaalt, door middel van inpluggen van de update functie in een hele lange if-else-loop, waar de keten zich op het volgende tijdstip bevindt. Hierna wordt er weer een nieuw random getal getrokken, waarna weer gekeken wordt naar welke toestand de keten gaat. Dit herhaalt zich totdat men in een absorberende toestand komt. Dan stopt de simulatie.

8.1 Tie-break

De tie-break simuleren we op een andere manier dan een gewone game: we beschouwen slechts vier toestanden, behorende bij het spelen van slechts één punt. Dit zijn in feite de toestanden 1 tot en met 4 van ons oorspronkelijke model. We nemen aan dat de optimale strategie om een tie-break te winnen dezelfde is als de optimale strategie om een game (punt) te winnen. Immers, de optimale strategie hangt niet af van de score in de game, dus zal dit bij de tie-break ook wel het geval zijn. Na het spelen van het eerste, derde, vijfde,... punt wisselt de servicebeurt - en dus de gekozen parameters - om. Dit wordt herhaald tot een speler de tie-break, en dus de set, heeft gewonnen.

Op deze manier kan men punten, games, sets en zelfs wedstrijden simuleren. Zie bijlage C voor een stukje uitwerking van een wedstrijd tussen Nadal en Federer.

9 Inleiding optimaal plannen: het RRTT-toernooi

In het tweede deel van deze scriptie houden we ons bezig met tennis op amateurniveau: het plannen van een *Round Robin Tennis Toernooi*, verder *RRTT* genoemd. Veel (amateur)tennisclubs organiseren dergelijke toernooien. RRT-toernooien kenmerken zich door het feit dat er enkele *series* zijn, met elk een aantal spelers. Iedere speler speelt één keer tegen alle andere spelers uit zijn/haar serie. Het voordeel van een RRTT boven een Knock Out Toernooi, is het feit dat ook de mindere spelers de zekerheid hebben dat ze een vooraf vastgesteld aantal wedstrijden spelen. Niet alleen in het tennis komen RRT-toernooien voor, maar ook in sporten als, onder andere, voetbal, hockey en basketbal. Hier speelt men echter meestal met een thuis-uit-schema.

Iedere speler heeft echter zijn/haar eigen *beschikbaarheid*, die van week tot week kan verschillen. Hierdoor kunnen mogelijk niet alle wedstrijden gespeeld worden. Het doel van de clubleiding is om zoveel mogelijk wedstrijden in te kunnen plannen, terwijl toch ieders beschikbaarheid gerespecteerd wordt, zodat zo min mogelijk competitievervalsing optreedt. Er kan bewezen worden dat het RRTT-probleem tot de klasse van moeilijkste problemen behoort. Zie hiervoor hoofdstuk 13. Dit deel behandelt een *oplossingsheuristiek* voor dit probleem.

10 Probleemstelling: Hoe optimaal te plannen?

Beschouw een aantal tennissers bij een club. Deze delen we in m series in, zodanig dat elke serie n spelers heeft. We nemen aan dat n even is. We kunnen immers dummyspelers toevoegen als dit niet het geval is. Zie ook [7] Het RRTT probleem ziet er dan als volgt uit:

- Elke speler speelt in een speelronde tegen een andere speler uit zijn serie.
- Elke wedstrijd duurt een uur.
- Er is één speelronde per week. Elke speler speelt dus slechts één uur per week.
- Elke speler speelt tegen alle andere spelers uit zijn serie. Er zijn dus $n-1$ speelrondes.
- Iedere week zijn er $p = \frac{1}{2}mn$ uren beschikbaar.
- Iedere speler heeft per week zijn eigen beschikbaarheid.
- Doel is het maximaliseren van het aantal mogelijke wedstrijden.

We definiëren het volgende:

- We noteren met C de verzameling mogelijke koppels van spelers. Hierin is ieder koppel maar één keer geteld, dus het is duidelijk dat
  $|C| = \frac{1}{2}mn(n-1) = p(n-1)$.
- Laat γ een koppel zijn, dus $\gamma \in C$.
- Laat $\gamma(i)$ een koppel waarin speler i gerepresenteerd is, $i = 1, ..., mn$.
- Laat $c_{\gamma,k,l}$ de als volgt gedefinieerde constante zijn:

$$c_{\gamma,k,l} = \begin{cases} 1 & \text{als koppel } \gamma \text{ kan spelen in uur } l \text{ van week } k \\ 0 & \text{anders} \end{cases}$$

  waarbij $\gamma \in C,\ l = 1, \ldots, p,\ k = 1, \ldots, n-1$

- Laat $x_{\gamma,k,l}$ onze beslissingsvariabele zijn. Hiervoor geldt:

$$x_{\gamma,k,l} = \begin{cases} 1 & \text{als koppel } \gamma \text{ gepland is op uur } l \text{ van week } k \\ 0 & \text{anders} \end{cases}$$

  waarbij $\gamma \in C,\ l = 1, \ldots, p,\ k = 1, \ldots, n-1$

10.1 IP-formulering

We kunnen het RRTT-probleem nu in IP-vorm gieten. We willen het aantal mogelijke wedstrijden maximaliseren. De doelfunctie ziet er dus als volgt uit:

$$max \sum_{\gamma \in C} \sum_{k=1}^{n-1} \sum_{l=1}^{p} c_{\gamma,k,l} \; x_{\gamma,k,l}$$

Nu gaan we naar de beperkingen kijken.

1. Elke speler speelt precies één keer tegen dezelfde speler. Voor elk koppel geldt dus dat het in een planning slechts één keer voorkomt. We sommeren dus over het aantal speeluren en het aantal speelrondes. In IP-vorm ziet deze beperking er als volgt uit:

$$\sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l} = 1 \qquad \forall \gamma \in C$$

2. Ieder beschikbaar uur kan aan slechts één koppel toegewezen worden. We sommeren dus over alle mogelijke koppels. In IP-vorm ziet deze beperking er als volgt uit:

$$\sum_{\gamma \in C} x_{\gamma,k,l} = 1 \qquad k = 1, \ldots, n-1; \; l = 1, \ldots, p$$

3. Iedere speler speelt slechts één wedstrijd per speelronde. Deze beperking is iets lastiger in IP-vorm te gieten. Hiervoor hebben we $\gamma(i)$ nodig. We sommeren over het aantal speeluren en het aantal koppels waarin speler i is gerepresenteerd. De beperking ziet er dan als volgt uit:

$$\sum_{\gamma(i) \in C} \sum_{l=1}^{p} x_{\gamma(i),k,l} = 1 \qquad k = 1, \ldots, n-1; \; i = 1, \ldots, 2p$$

4. De laatste beperking is de makkelijkste en ziet er als volgt uit:

$$x_{\gamma,k,l} \in \{0,1\} \qquad \gamma \in C; \; k = 1, \ldots, n-1; \; l = 1, \ldots, p$$

Het gehele RRTT-probleem ziet er dus als volgt uit:

$$max \left\{ \sum_{\gamma \in C} \sum_{k=1}^{n-1} \sum_{l=1}^{p} c_{\gamma,k,l} \; x_{\gamma,k,l} \;\middle|\; \begin{array}{lll} \sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l} & = 1 & \forall \gamma \in C \\ \sum_{\gamma \in C} x_{\gamma,k,l} & = 1 & k = 1, \ldots, n-1; \; l = 1, \ldots, p \\ \sum_{\gamma(i) \in C} \sum_{l=1}^{p} x_{\gamma(i),k,l} & = 1 & k = 1, \ldots, n-1; \; i = 1, \ldots, 2p \\ x_{\gamma,k,l} & \in \{0,1\} & \gamma \in C; \; k = 1, \ldots, n-1; \; l = 1, \ldots, p \end{array} \right\}$$

Deze formulering correspondeert met het plannen van alle partijen.

10.2 Alternatieve IP-formulering

Merk op dat het aantal beslissingsvariabelen in de vorige IP-formulering erg groot is. Deze is namelijk gelijk aan $(n-1)^2 p^2$. Dit getal wordt al snel erg groot. Stel dat we 40 spelers verdeeld over vier series hebben, dan is het aantal beslissingsvariabelen gelijk aan 32400. Dit aantal kunnen we als volgt reduceren. We noteren met $\gamma \cap \{k,l\} = 0$ elke onmogelijke koppeling. Hierdoor wordt dus het aantal partijen gemaximaliseerd en de $c_{\gamma,k,l}$ worden overbodig. Onze doelfunctie ziet er nu als volgt uit:

$$max \sum_{\gamma \in C} \sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l}$$

De beperkingen 1, 2, 3 en 4 worden omgezet in:

1. $\sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l} \leq 1 \qquad \forall \gamma \in C$
2. $\sum_{\gamma \in C} x_{\gamma,k,l} \leq 1 \qquad k = 1, \ldots, n-1;\ l = 1, \ldots, p$
3. $\sum_{\gamma(i) \in C} \sum_{l=1}^{p} x_{\gamma(i),k,l} \leq 1 \quad k = 1, \ldots, n-1;\ i = 1, \ldots, 2p$
4. $x_{\gamma(i),k,l} \in \{0,1\} \qquad \gamma \in C;\ k = 1, \ldots, n-1;\ l = 1, \ldots, p$

Om $\gamma \cap \{k,l\} = 0$ te realiseren, voeren we nog een extra beperking in. Deze beperking zet de beslissingsvariabelen behorende bij de onmogelijke wedstrijden op 0 en ziet er als volgt uit:

5. $x_{\gamma,k,l} = 0 \quad \gamma \cap \{k,l\} = 0$

Merk op dat beperkingen 1, 2 en 3 nu ongelijkheden zijn in plaats van de gelijkheden in de eerste IP-formulering. De round robin eigenschap (iedere speler speelt exact één wedstrijd per week) lijkt hiermee verloren te gaan. Dit is echter niet het geval. Beschouw een gegeven instantie. Er zijn nu twee mogelijkheden:

1. Alle $(n-1)p$ wedstrijden kunnen gepland worden. Als dit het geval is, dan is de waarde van de optimale oplossing van het probleem dus gelijk aan $(n-1)p$ en de bijbehorende oplossing is een geheel mogelijke planning. In de tweede IP-formulering betekent dit dat voor beperkingen 1-3 de gelijkheid geldt.

2. Niet alle $(n-1)p$ wedstrijden kunnen gepland worden. De optimale oplossing van de tweede IP-formulering respecteert dan niet de round robin eigenschap, maar de gevonden oplossing geeft wel een bovengrens voor het probleem. Deze bovengrens correspondeert met de waarde van de optimale oplossing van de eerste IP-formulering.

De gehele alternatieve formulering ziet er dus als volgt uit:

$$max \left\{ \sum_{\gamma \in C} \sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l} \;\middle|\; \begin{array}{lll} \sum_{k=1}^{n-1} \sum_{l=1}^{p} x_{\gamma,k,l} & \leq 1 & \forall \gamma \in C \\ \sum_{\gamma \in C} x_{\gamma,k,l} & \leq 1 & k = 1, \ldots, n-1;\ l = 1, \ldots, p \\ \sum_{\gamma(i) \in C} \sum_{l=1}^{p} x_{\gamma(i),k,l} & \leq 1 & k = 1, \ldots, n-1;\ i = 1, \ldots, 2p \\ x_{\gamma,k,l} & \in \{0,1\} & \gamma \in C;\ k = 1, \ldots, n-1;\ l = 1, \ldots, p \\ x_{\gamma,k,l} & = 0 & \gamma \cap \{k,l\} = 0 \end{array} \right\}$$

Uit de tweede IP-formulering volgt dat het RRTT-probleem ook te modelleren is als een *maximum kliek-probleem (MC)* [7]. Op de volgende pagina's wordt uitgelegd hoe dit in zijn werk gaat.

11 Maximum kliek-probleem

Definitie 11.1 *Kliek*
Zij $G = (V, E)$ een graaf. Een kliek *C is een deelverzameling van V zodanig dat elk knooppunt*

uit C verbonden is met alle andere knooppunten uit C. Een kliek vormt samen met de incidente takken dus een volledige graaf.

Het MC-probleem luidt als volgt:
Gegeven: $G = (V, E)$.
Gevraagd: Wat is de maximale grootte van een kliek?

Als IP-probleem ziet het MC-probleem er als volgt uit:

$$max \left\{ \sum_{i \in V} x_i \;\middle|\; \begin{array}{lll} x_i + x_j & \leq 1 & \forall (i,j) \in \overline{E} \\ x_i & \in \{0,1\} & i = 1, \ldots, |V| \end{array} \right\}$$

Het is duidelijk dat deze formulering juist is: Als een knooppunt i in de kliek zit, dan geldt $x_i = 1$. De eerste beperking zegt dat voor elk tweetal knooppunten waartussen *geen* tak bestaat, er hooguit één in de kliek kan zitten. Het doel is maximaliseren van het aantal knopen, en dus het aantal enen.

Als men naar de tweede IP-formulering van het RRTT-probleem kijkt, kan men opmerken dat het RRTT-probleem als een MC-probleem te formuleren is. Dit gaat als volgt in zijn werk: voor de knopen in de graaf nemen we alle *mogelijke* wedstrijden, i.e., de wedstrijden die zonder schending van de beschikbaarheid van de verschillende spelers, gespeeld kunnen worden, i.e., knooppunt $v = \gamma(i, j)kl$ bestaat als beide spelers i, j van koppel $\gamma(i, j)$ gepland kunnen worden op uur l van week k. De grootte van de graaf wordt dus volledig bepaald door het beschikbaarheidsschema.

Takken worden als volgt gegenereerd: we verbinden twee knooppunten door een tak als beide wedstrijden in een planning gepland kunnen worden. We krijgen meer inzicht in de samenhang met het MC-probleem als we kijken wanneer er *geen* tak tussen knooppunten bestaat, i.e., wanneer beide wedstrijden niet allebei in één planning gepland kunnen worden. Er bestaat *geen* tak tussen knooppunt $v_1 = \gamma(i_1, j_1)k_1l_1$ en $v_2 = \gamma(i_2, j_2)k_2l_2$ als (minstens) één van de volgende uitspraken geldt:

- Beide knopen representeren hetzelfde koppel, i.e., $i_1 = i_2 \wedge j_1 = j_2$.
- Beide knopen representeren hetzelfde tijdstip, i.e., $k_1 = k_2 \wedge l_1 = l_2$.
- (Minstens) één van de spelers zit in dezelfde week op een ander tijdstip (spelers spelen maar één keer per week), i.e., $k_1 = k_2 \wedge (i_1 = i_2 \vee j_1 = j_2)$.

Merk op dat deze drie uitspraken corresponderen met de eerste drie beperkingen in de tweede IP-formulering. Aangezien de IP-formulering voor het MC-probleem de takken die *niet* in de graaf zitten, beschouwd, is het duidelijk dat het RRTT-probleem te formuleren is als een MC-probleem.

Voorbeeld 11.1
Beschouw een instantie met slechts één serie met vier spelers, dus $m = 1, n = 4$. Per week zijn $p = 2$ uren gereserveerd, en er wordt drie weken gespeeld. We nemen aan dat iedere speler altijd beschikbaar is. Het aantal mogelijke wedstrijden is $\binom{n}{2}p(n-1)$, dus in ons geval $\binom{4}{2} \times 2 \times 3 = 36$. De bijbehorende graaf heeft dus 36 knopen.

Laat $v = \gamma(i, j)kl$ een knoop uit de graaf zijn. Het aantal knopen waarmee v niet *verbonden is, is 14: er zijn 5 andere knopen die hetzelfde koppel representeren; er zijn 5 andere knopen die hetzelfde tijdstip representeren en het aantal knooppunten op het andere tijdstip van week k waarin één persoon van $\gamma(i, j)$ speelt, is gelijk aan 4. Ieder knooppunt is dus met 21 andere knopen verbonden, wat een totaal van $\frac{1}{2} \times 36 \times 21 = 378$ takken oplevert. Het is duidelijk dat voor deze instantie de maximum kliek grootte 6 heeft. Immers, alle wedstrijden kunnen gepland worden, want iedereen is altijd beschikbaar.*

Er kan worden bewezen dat $MC \in \mathcal{NPC}$, dus het probleem is niet polynomiaal oplosbaar [5]. Wel zijn er enkele heuristieken om het MC-probleem op te lossen. We zullen drie lokale zoekmethoden behandelen: 1-opt, H-1-opt en k-opt. Hiervoor definiëren we eerst de volgende notatie:

- Voor een deelverzameling $S \subseteq V$, definiëren we de *deelgraaf* $G(S) = \{S, E \cap S \times S\}$.
- Met CC noteren we de huidige kliek.
- PA is de verzameling mogelijke toevoegingen aan CC, i.e., de knooppunten die aan alle knooppunten van CC grenzen.
- De graad van een knooppunt $v \in S$ in de deelgraaf $G(S)$ noteren we met $deg_{G(S)}(v)$.

11.1 1-opt

De 1-opt lokale zoekmethode, voorgesteld in [8] is een gretige methode die een kliek alleen maar kan uitbreiden. Aan de huidige kliek (een knooppunt op zich is ook een kliek) voegt het algoritme telkens een knooppunt toe dat met alle knooppunten uit de kliek verbonden is. Dit gaat door totdat zulke knooppunten niet meer bestaan. Deze methode wordt 1-opt genoemd, omdat deze per iteratie telkens één knooppunt toevoegt. Deze methode kan ook gebruikt worden om (snel) een kliek van behoorlijke grootte te bepalen. In pseudo-code ziet het algoritme er als volgt uit:

Algoritme 11.1 *1-opt*

1. Kies een willekeurige kliek CC waarvoor geldt dat $PA \neq \emptyset$ en bepaal $deg_{G(PA)}(v)$ voor $v \in PA$.

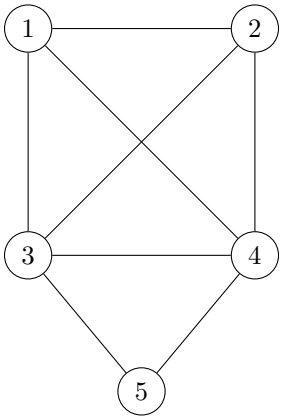
2a. Zoek een knoop met v met $\max\limits_{G(PA)} \{deg_{G(PA)}(i)\}$.
Als er meerdere knopen met dezelfde maximale graad gevonden worden, selecteer de eerste.
$CC := CC \cup \{v\}$

b. Update PA door te kijken welke knooppunten aan alle knooppunten van CC grenzen. Bepaal ook $deg_{G(PA)}(i), \forall i \in PA$.

c. Als $PA \neq \emptyset$, ga naar stap 2a.
Als $PA = \emptyset$, STOP

Een variant op dit algoritme is, door in stap 2a een willekeurig knooppunt met maximale graad te nemen, in plaats van de eerste. Het is eenvoudig in te zien dat de complexiteit van dit algoritme $\mathcal{O}(n^2)$ is. Immers, stap 2a en 2b zijn beide van de $\mathcal{O}(n)$ en er zijn maximaal n iteraties. Er kleeft echter wel een groot nadeel aan dit algoritme: een 'verkeerd' gekozen beginkliek wordt er niet meer uitgehaald door het algoritme. Beschouw het volgende voorbeeld, waarbij de beginkliek 'verkeerd' gekozen wordt.

Voorbeeld 11.2 *1-opt*



Het is duidelijk dat de $\{1, 2, 3, 4\}$ de maximum kliek is. We gaan nu het algoritme toepassen, waarbij we knooppunt 5 als beginkliek kiezen:

$CC = \{5\}, PA = \{3, 4\}, deg_{G(\{3,4\})}(3) = 1, deg_{G(\{3,4\})}(4) = 1, v = \{3\},$

$CC = \{3, 5\}, PA = \{4\}, deg_{G(\{4\})}(4) = 0, v = \{4\},$

$CC = \{3, 4, 5\}, PA = \emptyset$, dus STOP.

De gevonden kliek is $\{3, 4, 5\}$, terwijl we al hadden vastgesteld dat $\{1,2,3,4\}$ de maximum kliek is.

11.2 H-1-opt

De H-1-opt zoekmethode is een modificatie van het 1-opt-algoritme. Deze methode (H = herhaald) voert precies $|V|$ iteraties uit, waarbij in iedere iteratie een andere knoop als beginkliek gekozen wordt. De complexiteit van dit algoritme is dus $\mathcal{O}(n^3)$. Voor grote grafen kan deze methode echter erg lang duren. Deze methode is wel zeer nauwkeurig: hij is getest op ongeveer 10.000 random (enkelvoudige) grafen (zonder lussen), met een random aantal knooppunten, t/m grootte 35, en een random aantal takken. Ook is van al deze grafen de exacte grootte van de kliek bepaald m.b.v. *Branch & Bound* [9]. Bij slechts één van deze 10.000 grafen gaf het H-1-opt-algoritme een heuristische grootte die kleiner was dan de exacte grootte. Dit gebeurde bij een graaf met 19 knopen en 129 takken. De heuristische resp. exacte grootte van de kliek bedroegen 7 resp. 8. Zij bijlage D voor deze graaf.

11.3 k-opt

De k-opt zoekmethode, tevens voorgesteld in [8], is een uitbreiding van de 1-opt zoekmethode. Naast het toevoegen van knopen, haalt k-opt ook knooppunten uit de huidige kliek weg. De k-opt-heuristiek dankt zijn naam aan het feit dat 1-opt herhaaldelijk een aantal keer wordt toegepast. De idee is als volgt:

We starten, net zoals bij 1-opt, met een beginkliek $CC^{(0)}$ met bijbehorende $PA^{(0)}$, waarbij $PA^{(0)}$ niet leeg mag zijn. Eerst worden er een aantal 1-opt toevoegingen gedaan, waarbij we de grootte van iedere $CC^{(i)}$ onthouden, $i = 0, \ldots, s$. Dit doen we totdat $PA^{(s)}$ leeg is. Er kunnen dan geen knooppunten meer worden toegevoegd, dus moeten we gaan verwijderen. We bepalen hiervoor voor elke $v \in CC^{(s)}$ de bijbehorende $PA^{(s)}$. We verwijderen het knooppunt waarbij, na verwijdering, de $PA^{(s)}$ gemaximaliseerd wordt. Als er meerdere knooppunten zijn waarbij de resulterende $PA^{(s)}$ maximaal is, kiezen we de eerste. Hierna kan er weer toegevoegd worden. Dit gaat zo door tot er aan een stopcriterium is voldaan, waarbij we weer de grootte van elke $CC^{(i)}, s < i \leq t$, onthouden. We stoppen als alle knooppunten uit de huidige kliek verwijderd zijn.

Als deze procedure stopt, hebben we t oplossingen gevonden: $CC^{(1)}, CC^{(2)}, \ldots, CC^{(s)}, \ldots, CC^{(t)}$. Uit deze t oplossingen kiezen we de beste, i.e., de kliek $CC^{(k)}, k = 1, \ldots, t$, van maximale grootte. Als er meerdere zijn, kiezen we de eerste. Daarna herhalen we de gehele procedure met beginkliek $CC^{(0)} := CC^{(k)}$. De k-opt zoekmethode wordt herhaald totdat er geen betere oplossing wordt gevonden.

In dit algoritme is g resp. g_{max} gedefinieerd als het verschil in grootte resp. maximaal gevonden verschil in grootte tussen de beginkliek (van de iteratie) en de huidige kliek. Ook is een verzameling P geïntroduceerd die cycling tussen de oplossingen voorkomt. Immers, elk knooppunt dat bekeken is wordt uit de verzameling gehaald en kan niet meer toegevoegd of weggehaald worden in de huidige iteratie. Stap 2 stopt als $D = \emptyset$. Dit is ons stopcriterium. Met CC_{prev} wordt de beginkliek in de huidige iteratie aangegeven, en met CC_{best} noteren we de kliek van maximale grootte, gevonden in de huidige iteratie. In pseudo-code ziet het algoritme er als volgt uit:

Algoritme 11.2 *k-opt*

1a. Kies een willekeurige kliek CC waarvoor geldt dat $PA \neq \emptyset$ en bepaal $deg_{G(PA)}(v)$ voor $v \in PA$.

b. $CC_{prev} := CC, D := CC_{prev}, P = \{1, \ldots, n\}, g = 0, g_{max} = 0$.

2a. Als $\{PA \cap P\} \neq \emptyset$ (Toevoegen)

Zoek knoop v met $\max_{v \in \{PA \cap P\}} \{deg_{G(PA)}(v)\}$.
Als er meerdere knopen met dezelfde maximale graad gevonden worden, selecteer

de eerste.
$CC := CC \cup \{v\},\ g = g + 1,\ P := P \backslash \{v\}$
Als $g > g_{max}$, dan $g_{max} = g, CC_{best} = CC$.

Als $\{PA \cap P\} = \emptyset$ (Weghalen)

Zoek knoop $v \in CC \cap P$ zodanig dat, na weglaten van v, de resulterende $|PA|$ gemaximaliseerd wordt.
Als er meerdere van zulke knopen zijn, selecteer de eerste.
$CC := CC \backslash \{v\},\ g = g - 1,\ P := P \backslash \{v\}$.
Als $v \in CC_{prev}$, dan $D := D \backslash \{v\}$.

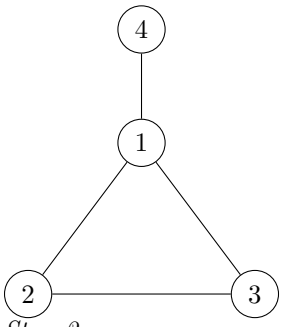
b. Update PA en $deg_{G(PA)}(i), \forall i \in PA$.

c. Als $D = \emptyset$, ga naar stap 3.
Als $D \neq \emptyset$, ga naar stap 2a.

3. Als $g_{max} > 0$, dan $CC := CC_{best}$ en ga naar stap 1b.
Als $g_{max} \leq 0$, dan $CC := CC_{prev}$ en STOP.

Voor de complexiteit geldt het volgende: het is duidelijk dat de complexiteit van stap 1 $\mathcal{O}(n)$ is. De complexiteit van de toevoeghandeling in stap 2a is $\mathcal{O}(n)$, want dit is immers dezelfde iteratiestap als bij 1-opt gedaan wordt. De complexiteit van de verwijderhandeling in stap 2a is $\mathcal{O}(n^2)$. Voor ieder knooppunt in PA moet immers, na verwijdering van v uit de kliek, de resulterende PA bepaald worden. De totale complexiteit van stap 2a is dus $\mathcal{O}(n^2)$. Ook stap 2b heeft complexiteit $\mathcal{O}(n^2)$. Stap 2 wordt maximaal n keer gedaan: daarna is D immers leeg, want er zaten nooit meer dan n knopen in de kliek. Deze totale procedure wordt ook maximaal n keer gedaan. De totale complexiteit is dus $\mathcal{O}(n) \times \mathcal{O}(n) \times \mathcal{O}(n^2) = \mathcal{O}(n^4)$.

Voorbeeld 11.3 *k-opt*



Beschouw nevenstaande graaf. Het is duidelijk dat $\{1, 2, 3\}$ de maximum kliek is. We kiezen als beginkliek $\{4\}$ om te laten zien dat de k-opt-heuristiek, in tegenstelling tot 1-opt, om kan gaan met verkeerd gekozen beginklieken. Merk op dat $PA \neq \emptyset$.

Iteratie 1
Stap 1
a. $CC = \{4\}$, $PA = \{1\}$.
b. $D = CC_{prev} = \{4\}$, $P = \{1, 2, 3, 4\}$, $g = 0$, $g_{max} = 0$.
Stap 2
$(PA \cap P) = \{1\}$, $v = 1$, $CC = \{1, 4\}$, $g = 1$, $P = \{2, 3, 4\}$, $g_{max} = 1$, $CC_{best} = \{1, 4\}$, $PA = \emptyset$.
$(PA \cap P) = \emptyset$, $v = 4$, $CC = \{1\}$, $g = 0$, $P = \{2, 3\}$, $D = \emptyset$, $PA = \{2, 3\}$.

Stap 3
$CC = CC_{best} = \{1, 4\}$

Iteratie 2
Stap 1
a. $CC = \{1, 4\}$, $PA = \emptyset$.
b. $D = CC_{prev} = \{1, 4\}$, $P = \{1, 2, 3, 4\}$, $g = 0$, $g_{max} = 0$.

Stap 2
$(PA \cap P) = \emptyset$, $v = 4$, $CC = \{1\}$, $g = -1$, $P = \{1, 2, 3\}$, $D = \{1\}$, $PA = \{2, 3\}$.
$(PA \cap P) = \{2, 3\}$, $v = 2$, $CC = \{1, 2\}$, $g = 0$, $P = \{1, 3\}$, $PA = \{3\}$.

$(PA \cap P) = \{3\}$, $v = 3$, $CC = \{1, 2, 3\}$, $g = 1$, $g_{max} = 1$, $CC_{best} = \{1, 2, 3\}$, $P = \{1\}$, $PA = \emptyset$.
$(PA \cap P) = \emptyset$, $v = 1$, $CC = \{2, 3\}$, $g = 0$, $P = \emptyset$, $D = \emptyset$.

Stap 3
$CC = CC_{best} = \{1, 2, 3\}$

Iteratie 3
Stap 1
a. $CC = \{1, 2, 3\}$, $PA = \emptyset$.
b. $CC_{prev} = D = \{1, 2, 3\}$, $P = \{1, 2, 3, 4\}$, $g = 0$, $g_{max} = 0$.

Stap 2
$(PA \cap P) = \emptyset$, $v = 1$, $CC = \{2, 3\}$, $g = -1$, $P = \{2, 3, 4\}$, $D = \{2, 3\}$, $PA = \{1\}$.
$(PA \cap P) = \emptyset$, $v = 2$, $CC = \{3\}$, $g = -2$, $P = \{3, 4\}$, $D = \{3\}$, $PA = \{2\}$.
$(PA \cap P) = \emptyset$, $v = 3$, $CC = \emptyset$, $g = -3$, $P = \{4\}$, $D = \emptyset$.

Stap 3
$CC = CC_{prev} = \{1, 2, 3\}$.

12 Toepassing op RRTT-probleem

Nu kunnen we 1-opt, H-1-opt of k-opt toepassen om tot een optimale oplossing van het RRTT-probleem te komen. Omdat, zoals we gezien hebben, voor een klein probleem de bijbehorende graaf al erg groot wordt, schrijven we een *computerprogramma* in MATLAB. Dit programma doet het volgende:

- Invoer: De gebruiker voert hier het aantal series (m), het aantal spelers per serie (n), de beschikbaarheid en de gewenste zoekmethode in.
- Het programma berekent het aantal speeluren per week (p) en het totaal aantal speeluren (q).
- Er wordt een matrix aangemaakt, waarbij iedere rij een speler voorstelt en iedere kolom een speeluur. Er zijn dus $m \times n$ rijen en q kolommen.
- Voor iedere rij wordt random een permutatie van de getallen $1, \ldots, q$ gekozen.
- Hierna wordt de matrix ingevuld. Ieder getal dat kleiner is dan $(beschikbaarheid) \times q$ wordt omgezet in een 1. De andere getallen blijven 0. Dit is het *beschikbaarheidsschema*: als er op plaats (i, j) een 1 staat, is speler i beschikbaar op uur j.
- Hierna wordt het aantal knopen in de graaf bepaald door per uur te kijken naar hoeveel koppels per serie op dat uur *kunnen* spelen.
- Deze mogelijke wedstrijden worden onder elkaar gezet. Ook worden deze wedstrijden genummerd.
- Er wordt een vierkante matrix aangemaakt met evenveel rijen als dat er knopen zijn.
- Er wordt bepaald tussen welke knooppunten er een tak bestaat.
- Het algoritme (1-opt, k-opt of H-1-opt) wordt toegepast.
- Het programma onthoudt zowel de grootte van de kliek als welke knooppunten in de kliek zitten.
- Bij deze knooppunten worden de corresponderende wedstrijden teruggezocht.

12.1 Toepassing op voorbeeld

Beschouw een instantie met 3 series, 4 spelers per serie en beschikbaarheid $\frac{1}{2}$. We passen de zoekmethodes k-opt en H-1-opt toe. Zie bijlage E voor de uitvoer.
In dit voorbeeld worden er dus 15 van de 18 wedstrijden gespeeld. Er worden dus 3 wedstrijden *niet* gespeeld en dus 3 uren *niet* gebruikt. Dit zijn het eerste uur van de eerste week, het vijfde uur van de tweede week en het zesde uur van de derde week. De wedstrijden die *niet* gespeeld worden zijn: 2 tegen 3, 5 tegen 6 en 6 tegen 7. We zetten enkele resultaten over deze instantie in de volgende tabel:

Beschik-baarheid	Opl. 1-opt	Opl. H-1-opt	Tijd H-1-opt	Opl. k-opt	Tijd k-opt
1	18/18	18/18	50.5	18/18	2.1
0.9	18/18	18/18	27.9	18/18	1.4
0.8	18/18	18/18	9.1	18/18	1.3
0.7	16/18	18/18	3.8	17/18	1.0
0.6	16/18	18/18	1.6	17/18	0.7
0.5	14/18	16/18	0.9	15/18	0.4
0.4	10/18	12/18	0.2	12/18	0.2
0.3	8/18	9/18	0.1	9/18	0.2
0.2	6/18	6/18	<0.1	6/18	<0.1
0.1	1/18	1/18	<0.01	1/18	<0.1

Er vallen een aantal zaken op:

- Voor een beschikbaarheid van 80% en hoger kunnen alle wedstrijden met elke zoekmethode gepland worden.
- Over het algemeen is H-1-opt de beste methode, daarna k-opt en als laatste 1-opt.
- H-1-opt is verschrikkelijk traag voor een hoge beschikbaarheid. Immers, bij een hoge beschikbaarheid zijn er zeer veel knopen. Voor al deze knopen moet H-1-opt het 1-opt-algoritme uitvoeren. Dit duurt dus erg lang. Deze tijdsduur daalt echter wel drastisch naarmate de beschikbaarheid lager wordt.

Over het algemeen is het dus het verstandigst om H-1-opt te gebruiken voor een beschikbaarheid van 70% of minder. Hierboven kan men het beste 1-opt of k-opt gebruiken. Voor instanties met meer series en spelers is het aan te raden om 1-opt te gebruiken, daar k-opt voor 'grotere' instanties zeer traag is.

13 Complexiteitstheorie

In dit hoofdstuk zullen we de complexiteit van het RRTT-probleem beschouwen. Hiervoor herhalen we eerst de definitie van de verschillende *complexiteitsklassen*:

$$\begin{array}{lcl} \mathcal{P} & = & \{\text{beslissingsproblemen die oplosbaar zijn met een polynomiaal algoritme}\}. \\ \mathcal{NP} & = & \{\text{beslissingsproblemen waarvan iedere } \textit{ja-instantie} \text{ polynomiaal gecontroleerd kan worden}\}. \\ \mathcal{NPC} & = & \{\text{beslissingsproblemen } P \text{ uit } \mathcal{NP} \text{ waarvoor geldt dat } Q \preceq P \text{ voor alle } Q \in \mathcal{NPC}\}. \end{array}$$

Dit is slechts een *informele* definitie van deze complexiteitsklassen. Verderop zullen deze begrippen geformaliseerd worden. We zullen stap voor stap de volgende reductieketen volgen om te bewijzen dat RRTT $\in \mathcal{NPC}$:

$$SAT \preceq^3 3SAT \preceq^2 3BSAT \preceq^1 RRTT$$

De indices corresponderen met de desbetreffende paragraaf waarin de reductie bewezen wordt. Tot slot zullen we in de vierde paragraaf nog bewijzen dat $SAT \in \mathcal{NPC}$.

13.1 $3BSAT \preceq RRTT$

We gaan aantonen dat iedere instantie van het $3BSAT$-probleem op polynomiale wijze is om te vormen tot een instantie van het $RRTT$-probleem. We zullen gebruik maken van het $RRTT$-probleem als beslissingsprobleem, zoals ook in [7] gedaan wordt:

Gegeven: Aantal series m, aantal spelers per serie n en ieders beschikbaarheid.
Gevraagd: Is er een toelaatbare planning met S wedstrijden?

Het is duidelijk dat $RRTT \in \mathcal{NP}$. Beschouw immers een *ja-instantie* van het probleem en de eerste IP-formulering. Om na te gaan dat dit een ja-instantie is, vereist de verificatie van $4p(n-1)$ beperkingen. Dit kan op polynomiale wijze. Verder zullen we bewijzen dat iedere instantie van $3BSAT$ polynomiaal is om te zetten naar een instantie van $RRTT$ zdd. ja-instanties in elkaar overgaan. Het $3BSAT$-probleem luidt als volgt:

Gegeven: Een verzameling letters $X = \{x_1, x_2, \ldots, x_\upsilon\}$, een zin C met woorden over X, waarin iedere letter (of zijn ontkenning) hoogstens drie keer voorkomt in de zin.
Gevraagd: Is er een waarheidstoekenning zodat C waar is?

Zonder verlies van algemeenheid kunnen we aannemen dat iedere letter $x_i \in X$ twee of drie keer voorkomt. Immers, komt de letter maar één keer voor, dan draagt de toekenning niets bij aan de waarheid van de zin. Als x_i twee keer voorkomt, dan komt hij één keer voor als x_i en één keer als $\overline{x_i}$. Mocht x_i drie keer voorkomen, dan kunnen we zonder verlies van algemeenheid aannemen dat we twee keer x_i en één keer $\overline{x_i}$ tegenkomen. Het aantal letters is υ. Het aantal woorden noteren we met χ. We weten dat $\chi \leq \frac{3\upsilon}{2}$, want iedere letter komt hoogstens drie keer voor en woorden bestaan minstens uit twee letters.

Construeer nu een $RRTT$-probleem met υ series, dus voor iedere letter hebben we een variabele. Het aantal spelers per serie stellen we gelijk aan 4 en het aantal beschikbare uren per week is dus $\frac{1}{2} \times \upsilon \times 4 = 2\upsilon$. Alléén in de eerste χ uren van de eerste week is er minstens één koppel beschikbaar, terwijl in alle andere uren en weken er geen enkel koppel kan spelen. Hieruit volgt dat er ten hoogste χ wedstrijden mogelijk zijn. Immers, een speler kan vaker dan één keer in zo'n mogelijk koppel zitten, maar mag slechts één van die wedstrijden spelen, want iedere speler speelt hoogstens één keer per week. We gaan nu zoeken naar een $RRTT$ met minstens χ mogelijke wedstrijden. Dit doen we als volgt:

Elk woord correspondeert met een uur van de eerste week. Iedere letter representeert de koppels van een serie op de volgende manier:

- Als letter x_i in een woord j voor het *eerst* voorkomt, dan geeft dit aan dat beide spelers P_1 en P_2 uit serie i beschikbaar zijn in uur j van de eerste week.
- Als letter $\overline{x_i}$ in een woord k voorkomt, dan betekent dit dat beide spelers P_1 en P_3 uit serie i beschikbaar zijn in uur k van de eerste week. Herinner dat $\overline{x_i}$ slechts één keer voorkomt.
- Als letter x_i in een woord l voor de *tweede keer* voorkomt (als deze letter of zijn complement in totaal drie keer voorkomt), dan zegt dit dat beide spelers P_3 en P_4 uit serie i beschikbaar zijn in uur l van de eerste week.

Merk op dat wanneer P_1 en P_3 spelen in uur k, beide koppels P_1, P_2 en P_3, P_4 niet kunnen spelen in uur j en l respectievelijk. Andersom is dit ook het geval. Er zijn nu twee mogelijkheden:

1. De optimale oplossing van het $RRTT$-probleem vult al de eerste χ uren van de eerste week. Dan is er waarheidstoekenning aan het $3BSAT$-probleem: men kan immers de letters die corresponderen met de koppels die toegewezen zijn aan een beschikbaar uur de waarde 1 geven.

2. De optimale oplossing van het $RRTT$-probleem vult *niet* al de eerste χ uren van de eerste week. Er is dan *minstens* één uur niet gevuld en dus *minstens* één woord in het $3BSAT$-probleem niet waar. Er is dus geen waarheidstoekenning aan het $3BSAT$-probleem.

□

Hiermee is het bewijs geleverd. Om nog het één en ander te verduidelijken, zullen we een voorbeeld beschouwen met een gegeven zin:

Voorbeeld 13.1
Beschouw de zin

$$C = (x_1 \cup x_2 \cup x_3) \cap (x_1 \cup \overline{x_3}) \cap (\overline{x_1} \cup \overline{x_2}) \cap (x_2 \cup x_3),$$

met $\chi = 4$ en $\upsilon = 3$. Snel is in te zien dat deze zin waar is. We beschouwen twee waarheidstoekenningen:

1. $x_1 = x_3 = 1, x_2 = 0$.
2. $x_2 = 1, x_1 = x_3 = 0$.

We maken het volgende RRTT met 3 series, 4 spelers per serie en waarin alleen in de eerste 4 uren van de eerste week er minstens één koppel beschikbaar is. Hier noteren we met P, Q en R de spelers uit de eerste resp. tweede en derde serie. In deze tabel corresponderen de kolommen met de koppels en de rijen met de χ uren:

$(P_1, P_2)^*$	$(Q_1, Q_2)^\bullet$	(R_1, R_2)
$(P_3, P_4)^*$	$-$	$(R_1, R_3)^\bullet$
$(P_1, P_3)^\bullet$	$(Q_1, Q_3)^*$	$-$
$-$	$(Q_3, Q_4)^\bullet$	$(R_3, R_4)^*$

1. *Men ziet dat door de wedstrijden met een $*$ te spelen op de corresponderende uren, er een planning is die de eerste χ uren van de eerste week vult. Deze planning komt overeen met de eerste waarheidstoekenning aan de zin. Deze planning is echter niet uniek: bij dezelfde waarheidstoekenning, zou men ook de wedstrijden anders kunnen plannen door de rollen van de eerste en derde serie om te draaien.*
2. *Ook de wedstrijden met een $\bullet$ corresponderen met een planning die de eerste χ uren van de eerste week vult. Deze planning correspondeert met de tweede waarheidstoekenning. Ook deze planning is niet uniek, want men zou weer de rol van de eerste en derde serie kunnen omdraaien.*

13.2 $3SAT \preceq 3BSAT$

Het $3SAT$-probleem luidt als volgt:

Gegeven: Een verzameling letters $X = \{x_1, x_2, \ldots, x_\upsilon\}$, een zin C met woorden over X, waarin ieder woord drie letters heeft.
Gevraagd: Is er een waarheidstoekenning zodat C waar is?

Het is voldoende te laten zien dat iedere instantie van $3SAT$ in polynomiale tijd is om te zetten in een instantie van $3BSAT$ zdd. ja-instanties in elkaar overgaan.
Gegeven een instantie van $3SAT$. Als iedere variabele *hoogstens* drie keer voorkomt, zijn we klaar. Stel dus dat een variabele x_i, $i = 1, \ldots, \upsilon$ *minstens* vier keer voorkomt in de zin, zeg $n \geq 4$ keer. We maken nu n nieuwe variabelen $y_1, y_2, \ldots, y_n$ aan en vervangen het j-de optreden van x_i door y_j, $j = 1, \ldots, n$. Verder voegen we het volgende zinstuk toe:

$$(y_1 \cup \overline{y_2}) \cap (y_2 \cup \overline{y_3}) \cap \ldots \cap (y_{n-1} \cup \overline{y_n}) \cap (y_n \cup \overline{y_1})$$

De zin kan op twee manieren waar worden gemaakt:

- $\forall j$ geldt dat $y_j = 1$. Dit correspondeert met $x_i = 1$.
- $\forall j$ geldt dat $y_j = 0$. Dit correspondeert met $x_i = 0$.

Dit doen we voor alle variabelen die vier keer of vaker voorkomen, waarbij we telkens wel weer frisse variabelen aanmaken. Dit kan in polynomiale tijd. Nu komt iedere variabele in de nieuwe uitdrukking hoogstens drie keer voor, dus het is een geldige instantie van $3BSAT$. Ook geldt dat de nieuwe uitdrukking precies dan waargemaakt kan worden als de originele dat ook kon: $x_i = 1$ correspondeert immers met $y_1 = y_2 = \ldots = y_{n-1} = y_n = 1$. Ja-instanties gaan dus in elkaar over.

□

13.3 $SAT \preceq 3SAT$

Het SAT-probleem luidt als volgt:

Gegeven: Een verzameling letters $X = \{x_1, x_2, \ldots, x_v\}$, een zin $\mathcal{C} = C_1 \cap C_2 \cap \ldots \cap C_m$ met C_j woorden over X.
Gevraagd: Is er een waarheidstoekenning zodat $\mathcal{C}$ waar is?

Het is voldoende te laten zien dat iedere instantie van SAT in polynomiale tijd is om te zetten in een instantie van $3SAT$ zdd. ja-instanties in elkaar overgaan. We zullen een zin $\mathcal{C}'$ met woorden van drie letters construeren, waarbij de letters afkomstig zijn van $X' = X \cup \{\cup_{j=1}^{m} X'_j\}$ zdd. $\mathcal{C}'$ waar is dan en slechts dan als $\mathcal{C}$ waar is. [5]

Neem een C_j, zeg

$$C_j = y_1 \cup y_2 \cup \ldots \cup y_k \text{ met } y_i \in X,\ 1 \leq i \leq k \leq n.$$

We definiëren met k dus het aantal letters in het woord. Voor k zijn er een aantal mogelijkheden:

- Als $k = 1$: Neem $X'_j = \{z_1, z_2\}$ en $C_j = (y_1 \cup z_1 \cup z_2) \cap (y_1 \cup z_1 \cup \overline{z_2}) \cap (y_1 \cup \overline{z_1} \cup z_2) \cap (y_1 \cup \overline{z_1} \cup \overline{z_2})$.
- Als $k = 2$: Neem $X'_j = \{z_1\}$ en $C'_j = (y_1 \cup y_2 \cup z_1) \cap (y_1 \cup y_2 \cup \overline{z_1})$.
- Als $k = 3$: Neem $X'_j = \emptyset$ en $C'_j = C_j$.
- Als $k \geq 4$: Neem $X'_j = \{z_1, z_2, \ldots, z_{k-3}\}$ en
  $C'_j = (y_1 \cup y_2 \cup z_1) \cap (\overline{z_1} \cup y_3 \cup z_2) \cap (\overline{z_2} \cup y_4 \cup z_3) \cap \ldots \cap (\overline{z}_{k-4} \cup y_{k-2} \cup z_{k-3}) \cap (\overline{z}_{k-3} \cup y_{k-1} \cup y_k)$.

We zullen nu laten zien dat C_j waar is dan en slechts dan als C'_j waar is. Als $k \leq 2$, dan worden er via bovenstaande constructie letters toegevoegd aan het woord. De waarheid van dit woord komt hiermee niet in het geding. Als $k = 3$ dan is het SAT-woord C al een $3SAT$-woord C'. We beschouwen dus het geval dat $k \geq 4$. We hebben nu twee mogelijkheden:

C_j **is waar:** Hieruit volgt dat er ten minste één y-variabele gelijk is aan 1, zeg $y_l = 1$.

- Als $l \leq 2$: Neem $z_i = 0$ voor $1 \leq i \leq k-3$. Dan is C'_j waar.
- Als $l \geq k-1$: Neem $z_i = 1$ voor $1 \leq i \leq k-3$. Dan is C'_j waar.
- Als $3 \leq l \leq k-2$: Neem $z_i = \begin{cases} 1 & \text{voor} \quad 1 \leq i \leq l-2 \\ 0 & \text{voor} \quad l-1 \leq i \leq k-3 \end{cases}$
  Dan is C'_j waar.

C_j **is niet waar:** Dan geldt dat $y_i = 0$ voor $1 \leq i \leq k$.
Stel dat C'_j wel waar is. Dan is $z_i = 1$, $1 \leq i \leq k-3$. Maar dan is het laatste woord van C'_j niet waar, zodat C'_j ook niet waar is.

Bovenstaande constructie is polynomiaal. Omdat $k \leq n$, bestaat iedere C'_j maximaal uit $n-2$ woorden van drie letters. Het opstellen van $\mathcal{C}'$ heeft dus complexiteit $\mathcal{O}(nm)$, wat polynomiaal is.

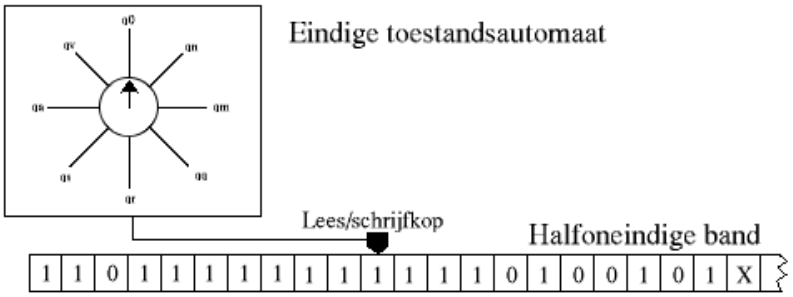
□

13.4 $SAT \in \mathcal{NPC}$

Om dit te kunnen bewijzen, is het nodig om het begrip 'algoritme' te formaliseren: Een probleem P is *algoritmisch berekenbaar* als er een algoritme voor bestaat. Algoritmisch berekenbaar wordt ook wel *mechanische berekenbaar* genoemd.

13.4.1 Turing Machines

Al sinds de introductie van de *Turing Machine* in 1936 bestaat het vermoeden dat zo'n Turing machine een perfect model is voor mechanische berekenbaarheid: alles wat mechanisch berekend kan worden, kan berekend worden door een Turing machine. En wat niet berekend kan worden door een Turing machine, is niet mechanisch berekenbaar. I.e., er bestaat geen algoritme voor. In het vervolg zullen we enkel een *Deterministische één-tape Turing Machine (DTM)* beschouwen. [10]



Figuur 3: Turing machine [11]

Definitie 13.1 *Deterministische één-tape Turing Machine (DTM)*
Deze machine kan beschreven worden door een 8*-tupel* $(S, \Gamma, \Sigma, f, b, s_0, s_y, s_n)$*, waarbij:*

- S *een* eindige toestandsverzameling *is. In deze toestandsverzameling zijn* s_0, s_y *en* s_n *de toestanden die corresponderen met de* begin- *resp.* ja- *resp.* nee-toestand.
- Γ *een* eindige verzameling *is van* tape-symbolen.
- Σ *een stricte deelverzameling is van* Γ*, bestaande uit de* input-symbolen *en* $b = \Gamma \backslash \Sigma$ *is het* blanco-symbool.
- $f : (S \backslash \{s_y, s_n\}) \times \Gamma \to S \times \Gamma \times \{-1, +1\}$ *is de* overgangsfunctie.

Verder heeft een DTM een *oneindige tape* bestaande uit de cellen $\ldots, c(-2), c(-1), c(0), c(1), c(2), \ldots$, die ieder één tape-symbool ($\in \Gamma$) kunnen bevatten. De machine wordt op discrete tijdstippen $t = 0, 1, \ldots$ waargenomen en bevindt zich op tijdstip t in toestand $s(t)$. Er is dan een wijzer

die op cel $c(h(t))$ is gericht. De inhoud van cel $c(i)$ op tijdstip t noteren we met $\gamma(i,t)$. In de begintoestand $t=0$ hebben we: $s(0)=s_0, h(0)=1, \gamma(i,0)=\begin{cases} x_i & \text{voor} \quad 1 \leq i \leq n \\ b & \text{voor} \quad i \leq 0 \ \text{ of } \ i \geq n+1 \end{cases}$
Hierbij is $\{x_1, x_2, \ldots, x_n\}$ de input.

Op tijdstip t voert de machine het volgende uit:

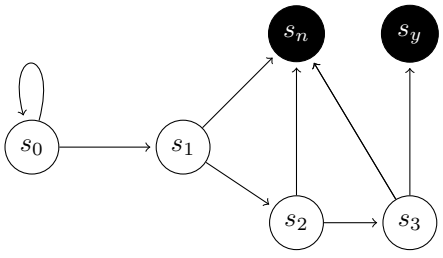
Als $s(t) \in \{s_y, s_n\}$**:** De berekening wordt stopgezet en het antwoord op het beslissingsprobleem is 'ja' als $s(t)=s_y$ en 'nee' als $s(t)=s_n$.

Anders: We definiëren $f(s(t), \gamma(h(t),t)) = (p,q,d)$ en we doen het volgende:

- $s(t+1)=p$. Dit is de toestand die door de overgangsfunctie gegenereerd wordt.
- $\gamma(h(t), t+1)=q$. De inhoud van de cel waarnaar de wijzer wijst, verandert dus.
- $\gamma(i,t+1)=\gamma(i,t)$ voor $i \neq h(t)$. De inhoud van de cellen waar de wijzer niet naar wijst, blijven dus hetzelfde.
- $h(t+1)=h(t)+d$. Omdat geldt dat $d=+1$ of $d=-1$, schuift de wijzer een plekje naar rechts resp. naar links.

Voorbeeld 13.2 *Turing Machine*
$S=\{s_0,s_1,s_2,s_3,s_y,s_n\}$; $\Gamma=\{0,1,b\}$; $\Sigma=\{0,1\}$.
$n=6$ *en* $\{x_1,x_2,x_3,x_4,x_5,x_6\}=\{1,0,1,0,0,0\}$
De overgangsfunctie ziet er als volgt uit:
$f(s_0,0)=(s_0,0,1)$; $f(s_1,0)=(s_2,b,-1)$; $f(s_2,0)=(s_3,b,-1)$; $f(s_3,0)=(s_y,b,-1)$;
$f(s_0,1)=(s_0,1,1)$; $f(s_1,1)=(s_n,b,-1)$; $f(s_2,1)=(s_n,b,-1)$; $f(s_3,1)=(s_n,b,-1)$;
$f(s_0,b)=(s_1,b,-1)$; $f(s_1,b)=(s_n,b,-1)$; $f(s_2,b)=(s_n,b,-1)$; $f(s_3,b)=(s_n,b,-1)$.
De hierboven beschreven DTM doet het volgende:

	$s(t)$	$h(t)$	$\gamma(h(t),t)$	$f(s(t),\gamma(h(t),t))$
$t=0$	s_0	*1*	*1*	$(s_0,1,1)$
$t=1$	s_0	*2*	*0*	$(s_0,0,1)$
$t=2$	s_0	*3*	*1*	$(s_0,1,1)$
$t=3$	s_0	*4*	*0*	$(s_0,0,1)$
$t=4$	s_0	*5*	*0*	$(s_0,0,1)$
$t=5$	s_0	*6*	*0*	$(s_0,0,1)$
$t=6$	s_0	*7*	*b*	$(s_1,b,-1)$
$t=7$	s_1	*6*	*0*	$(s_2,b,-1)$
$t=8$	s_2	*5*	*0*	$(s_3,b,-1)$
$t=9$	s_3	*4*	*0*	$(s_y,b,-1)$
$t=10$	s_y			



Merk op dat deze DTM altijd stopt. Hiernaast staat een schematische weergave van de toestanden en hun overgangen van deze DTM, met s_y en s_n als absorberende toestanden. Het is duidelijk dat zodra toestand s_1 bereikt is, de DTM binnen drie tijdstappen termineert. Toestand s_1 wordt uiteindelijk een keer bereikt, want zolang de DTM zich in toestand s_0 bevindt, wordt $h(t)$ opgehoogd. Op een gegeven moment geldt dat $h(t)=7>6$, dus dan geldt $\gamma(7,t)=b$. Dan gaat het systeem in toestand s_1 over.

Merk ook op dat de machine uit het voorbeeld niet met antwoord 'ja' kan stoppen als $n \leq 2$. Immers, men begint met $h(0)=1$ en blijft tot $t=2$ in s_0. Op $t=2$ geldt immers $h(2)=3$,

dus $\gamma(3,2) = b$ en $d = -1$. Dus op $t = 3$ is men in toestand s_1 met $h(3) = 2$. Merk op dat s_y alleen maar bereikt wordt vanuit $f(s_3, 0)$. Omdat echter $d = -1$ voor alle toestanden behalve s_0, kan $h(t)$ alleen maar dalen. Vanuit s_1 zijn er 2 tijdstappen nodig om in s_3 te komen, dus $h(5) = 2 - 2 = 0$ en $\gamma(0,5) = b$. M.a.w., $h(5) = 1$ of $h(5) = 2$ kan niet voorkomen.

Merk tevens op dat de $h(t)$ van bovenstaande DTM eerst oploopt tot $n+1$. Dan gaat het systeem in toestand s_1 over en kan de DTM alleen nog maar dalen. Om als antwoord 'ja' te krijgen, moet $h(t)$ drie keer dalen om in s_3 te komen. Voor $\gamma(h(t), t) = x_{h(t)}$ moet dan gelden dat deze 0 zijn, anders komt het systeem in toestand s_n. Hieruit volgt dat moet gelden dat $x_n = x_{n-1} = x_{n-2} = 0$. Als dit geldt, geeft de machine als antwoord 'ja'. Tezamen met bovenstaande opmerking, geeft deze machine dus enkel een 'ja'-antwoord als de input een binair getal is, waarin geen factor 4, 2 of 1 zit, dus wanneer het getal een veelvoud is van 8. In het voorbeeld wordt $8 + 32 = 40$ ingevoerd en dit geeft een 'ja'-antwoord.

13.4.2 De klasse $\mathcal{P}$

Nu kunnen we enkele al eerder gebruikte begrippen formaliseren: [10]

Definitie 13.2 *Algoritme*
Een algoritme *is een programma voor een Turing machine dat de machine na een eindig aantal stappen laat stoppen in toestand s_y (resp. s_n) voor elke instantie waarvan het antwoord 'Ja' (resp. 'Nee') is.*

Definitie 13.3 *Taal*
Een Turing machine en een algoritme definiëren een taal. *Een taal is een deelverzameling van de instantieverzameling zdd. deze instantie in toestand s_y eindigt, i.e., een ja-instantie is.*

Laat Π nu een beslissingsprobleem zijn en veronderstel dat de parameters van Π in de DTM worden gerepresenteerd volgens een *codering* e. Laat π een instantie zijn van Π en zij $I(\pi, e)$ en $n(\pi, e)$ de invoer resp. invoerlengte van instantie π, gegeven codering e. Veronderstel dat de DTM M een algoritme is voor Π. Nu kunnen we de *tijdscomplexiteit* definiëren:

Definitie 13.4 *Tijdscomplexiteit*
De tijd die een DTM M nodig heeft om instantie π te beantwoorden wordt genoteerd met $T_M(\pi, e)$ en is gedefinieerd door: $T_M(\pi, e)$ = aantal stappen dat M nodig heeft om tot een antwoord te komen. De tijdscomplexiteit $T_M(\Pi, e, n)$ *van een probleem Π wordt nu gedefinieerd door:*

$$T_M(\Pi, e, n) := \sup_{\pi} \{T_M(\pi, e) | \pi \textit{ is een instantie van } \Pi \textit{ en } n(\pi, e) = n\}$$

Binnen dit aantal tijdstappen stopt de DTM dus altijd.

M heet een *polynomiaal* algoritme als geldt dat $T_M(\Pi, e, n) \leq p(n)$, met p een polynoom in n. I.e., als het aantal tijdstappen dat de machine doet, begrensd wordt door een polynoom. Dit brengt ons bij de formele definitie van $\mathcal{P}$:

$$\mathcal{P} := \left\{ \Pi \;\middle|\; \begin{array}{l} \text{Er is een codering } e \text{ van de parameters van } \Pi \text{ en} \\ \text{er is een algoritme } M \text{ zdd. } M \text{ polynomiaal is.} \end{array} \right\}$$

13.4.3 De klasse $\mathcal{NP}$

We kunnen met het idee van Turing machines ook de klasse $\mathcal{NP}$ formaliseren. Dit is de complexiteitsklasse waarvan de beslissingen snel op grond van de invoer en een *certificaat* gemaakt kunnen worden.

Definitie 13.5 *Certificaat*
Een certificaat *is een recept, waarmee kan worden nagegaan dat het antwoord van een ja-instantie juist is. Dit wordt gerepresenteerd door een eindig stuk van de tape van een Turing machine, bestaande uit 0 of meer cellen. Elke cel bevat een $\sigma \in \Sigma$.*

Om een Turing machine als zo'n *controlerende* computer te gebruiken, plaatsen we de invoer $x = \{x_1, x_2, \ldots, x_n\}$ die de instantie beschrijft, in de cellen $1, 2, \ldots, n$. Veronderstel dat het certificaat $C(x)$ $m > 0$ cellen beslaat, waarbij geldt dat $m \leq p(n)$. Nu kan men een controlerend programma schrijven waarbij het programma verifieert of x inderdaad een woord uit de taal van de machine is. De machine kan dan ook het certificaat gebruiken. Zo'n Turing machine wordt een *Niet-Deterministische Turing Machine (NDTM)* genoemd. De hardware is hetzelfde als bij een DTM, alleen de manier van invoer en de vraag die gesteld wordt, verschilt: bij een NDTM betekent een nee-antwoord dat x niet die informatie bevat om in polynomiale tijd tot een antwoord te kunnen komen. Bij een DTM is dit wel het geval.

$$\mathcal{NP} := \left\{ \Pi \;\middle|\; \begin{array}{l} \text{Er is een codering } e \text{ van de parameters van } \Pi, \\ \text{er is een polynoom } p \text{ en een NDTM } M \text{ zdd.} \\ \text{voor iedere ja-instantie } \pi \text{ met invoer } x \text{ en invoerlengte } n \\ \text{er een certificaat } C(x) \text{ bestaat zdd. } T_M(\pi, e, n) \leq p(n). \end{array} \right\}$$

Het is duidelijk dat $\mathcal{P} \subset \mathcal{NP}$. Het is nog een open probleem of $\mathcal{P} = \mathcal{NP}$. Dit lijkt onwaarschijnlijk.

13.4.4 De klasse $\mathcal{NPC}$

Veronderstel dat Π_1 en Π_2 twee beslissingsproblemen zijn. We zeggen dat er een *polynomiale reductie* (notatie: $\Pi_1 \preceq \Pi_2$) van Π_1 naar Π_2 is als er een functie g is die voor iedere instantie π_1 van Π_1 de input $I(\pi_1, e_1)$ afbeeldt op de input $I(\pi_2, e_2)$ van een instantie π_2 van Π_2 zdd:

1. het antwoord van π_1 'ja' is dan en slechts dan als het antwoord van π_2 'ja' is.
2. $I(\pi_2, e_2) = g(I(\pi_1, e_1))$ met een polynomiaal begrensd algoritme wordt bepaald.

Twee beslissingsproblemen Π_1 en Π_2 heten *polynomiaal equivalent* als $\Pi_1 \preceq \Pi_2$ en $\Pi_2 \preceq \Pi_1$. Nu kunnen we de klasse van $\mathcal{NP}$*-volledige problemen* definiëren:

$$\mathcal{NPC} := \{\Pi \;\mid\; \Pi \in \mathcal{NP} \text{ en voor iedere } \Pi_1 \in \mathcal{NP} \text{ geldt } \Pi_1 \preceq \Pi \;\}$$

In woorden: $\Pi \in \mathcal{NPC}$ als iedere instantie van $\Pi_1 \in \mathcal{NP}$ polynomiaal reduceerbaar is tot een instantie van Π. Het is niet triviaal dat $\mathcal{NPC} \neq \emptyset$, want het is niet snel duidelijk waarom er een probleem bestaat dat de sleutel tot de polynomiale-tijd-oplosbaarheid van ieder probleem in de klasse $\mathcal{NP}$ heeft. Als we weten dat er zo'n probleem bestaat, dan is het makkelijk in te zien dat er meerdere bestaan. Dat zo'n probleem bestaat, is bewezen in 1971 door Stephen Cook. Hij bewees dat het eerder geïntroduceerde SAT zo'n probleem is. [12]

13.4.5 De stelling van Cook

Stelling 13.1 *Cook, 1971*
$SAT \in \mathcal{NPC}$

Bewijs
We zullen eerst aantonen dat $SAT \in \mathcal{NP}$: neem hiervoor een ja-instantie $x = \{x_1, x_2, \ldots, x_n\}$ en neem als inputlengte het aantal woorden. Neem verder als certificaat een toekenning van de variabelen die met 'ja' correspondeert. Het is duidelijk dat de controle of de zin waar is, polynomiaal begrensd is. Nu weten we dat er een NDTM is.

Laat $p(n)$ een polynoom in n zijn met de eigenschap dat de NDTM elk paar $(x, C(x))$ binnen tijd $p(n)$ herkent, waarbij x een woord van de taal van de NDTM is en $C(x)$ is zijn certificaat. Onze

intentie is nu om voor elk woord I in de taal van de NDTM, een instantie $f(I)$ van SAT waarvoor het antwoord 'ja' is, te construeren. Omgekeerd, als I niet in de taal zit, willen we hebben dat de SAT-zin niet waar is. Het idee van het bewijs kan als volgt worden samengevat:

De instantie van SAT die geconstrueerd zal worden, zal een disjunctie van woorden zijn die samen ervoor zorgen dat er een certificaat bestaat waarvoor de NDTM de zin waar maakt

Zodoende, om te testen of een woord al dan niet in de taal zit, is het nodig om na te gaan of de zin waar is.
We gaan een instantie van SAT construeren op zo'n manier dat de woorden waar zijn dan en slechts dan als x in de taal zit, i.e., wanneer in de NDTM $(x, C(x))$ in toestand s_y belandt. Hiervoor definiëren we de volgende letters:

- $G_{i,t,j} = 1$ dan en slechts dan als na stap t, symbool j in cel i zit, i.e., $\gamma(i,t) = \gamma_j$, waarbij $-p(n) + 1 \leq i \leq p(n) + 1$ en $0 \leq j \leq |\Gamma|$.
- $H_{i,t} = 1$ dan en slechts dan als na stap t, de wijzer naar cel i wijst, i.e., $h(t) = i$, waarbij i en t als boven gedefinieerd zijn.
- $S_{t,k} = 1$ dan en slechts dan als na stap t, de NDTM zich in toestand s_k bevindt, i.e., $s(t) = k$, met t als boven gedefinieerd en $0 \leq k \leq |S|$.

Nu gaan we de letters tellen: merk op dat de NDTM binnen $p(n)$ stappen termineert. De wijzer raakt dus nooit verder dan $|p(n)|$ cellen van zijn startplaats verwijderd, dus i is van $\mathcal{O}(p(n))$. Het is duidelijk dat $0 \leq t \leq p(n)$. Merk verder op dat zowel het aantal toestanden als het aantal tape-symbolen eindig is. Er is dus een totaal van $\mathcal{O}(p(n)^2)$ variabelen.

Het is duidelijk dat een willekeurige waarheidstoekenning aan deze letters niet altijd in de taal van de machine zit: als we bijvoorbeeld (erg onvoorzichtig) stellen dat $T_{9,4} = T_{10,33} = 1$, gaat dit niet goed, want binnen één tijdstap is de machine dan van cel $c(4)$ naar cel $c(33)$ gesprongen. Dit kan niet, want de wijzer kan maar één cel per tijdstap opschuiven. Er moet dus gelden dat minstens één van beide de waarheidswaarde 0 krijgt.

We gaan nu beschrijven onder welke voorwaarden de waarheidstoekenning aan de letters, in de taal zit voor $(x, C(x))$. Dan is het duidelijk dat iedere toekenning aan de letters die wordt gevonden door het SAT-probleem op te lossen dan in de taal zit. Dit kunnen we doen door te eisen dat een zin waar is, waarbij ieder woord correspondeert met één nodige voorwaarde. We zullen nu deze voorwaarden en de woorden waarmee ze corresponderen, beschrijven.

1. **Op ieder tijdstip bevindt de NDTM zich in minstens één toestand.**
   Hieruit volgt dat minstens één van de $|S|$ mogelijke toestandsletters $S(t,k)$ waar moet zijn. Dit leidt tot de volgende woordenverzameling, één voor elke t:

   $$(S_{t,1} \cup S_{t,2} \cup \ldots \cup S_{t,|S|})$$

   Omdat t van $\mathcal{O}(p(n))$ is, zijn er $\mathcal{O}(p(n))$ van deze woorden.

2. **Op ieder tijdstip bevindt de NDTM zich in hoogstens één toestand.**
   Hieruit volgt dat voor iedere stap t en ieder paar k', k'' disjuncte toestanden de het volgende woord waar moet zijn:

   $$(\overline{S}_{t,k'} \cup \overline{S}_{t,k''})$$

   Ook dit is een totaal van $\mathcal{O}(p(n))$ woorden.

3. **Op ieder tijdstip bevat een cel minstens één symbool uit de verzameling van tape-symbolen.**
   Het woord dat hiervoor waar moet zijn, is:

$$(G_{i,t,1} \cup G_{i,t,2} \cup \ldots \cup G_{i,t,|\Gamma|})$$

voor elk tijdstip t en cel i. Dit zijn dus $\mathcal{O}(p(n)^2)$ woorden.

4. **Op ieder tijdstip bevat een cel hoogstens één symbool uit de verzameling van tape-symbolen.**
   Het volgende woord moet hiervoor waar zijn:

   $$(\overline{G}_{i,t,j'} \cup \overline{G}_{i,t,j''})$$

   voor elk tijdstip t, cel i en paar j', j'' van disjuncte tape-symbolen. Dit zijn dus $\mathcal{O}(p(n)^2)$ woorden.

5. **Op ieder tijdstip is de wijzer op één cel gericht.**
   De woorden die hiervoor waar moeten zijn, zijn:

   $$(H_{-p(n)+1,t} \cup H_{-p(n)+2,t}, \cup \ldots \cup H_{p(n)+1,t}) \text{ (voor minstens)}$$
   $$(\overline{H}_{i',t} \cup \overline{H}_{i'',t}) \text{ (voor hoogstens)}$$

   met t van $\mathcal{O}(p(n))$, $-p(n)+1 \leq i \leq p(n)+1$ en paar i', i'' van disjuncte cellen. Dit zijn dus $\mathcal{O}(p(n)^2)$ woorden.

6. **Bij aanvang is de machine in toestand s_0 en de wijzer gericht op cel $c(1)$.**
   Het woord dat hiervoor waar moet zijn, is:

   $$(S_{0,0} \cup H_{1,0})$$

7. **Bij aanvang is de invoer x in cellen $c(1), \ldots, c(n)$.** Het woord dat hiervoor waar moet zijn, is:

   $$(G_{1,0,x_1} \cup G_{2,0,x_2} \cup \ldots \cup G_{n,0,x_n})$$

8. **Op tijdstip $p(n)$ is de toestand s_y als het probleem antwoord 'ja' heeft.**
   Het woord dat hiervoor waar moet zijn, is:

   $$(S_{p(n),y})$$

9. **Op ieder tijdstip bepaalt de overgangsfunctie in welke configuratie de NDTM overgaat.**
   Om woorden te vinden die hieraan voldoen, beschouwen we eerst de volgende voorwaarde: het symbool in cel $c(i)$ kan niet veranderen op tijdstip t van de berekening als de wijzer niet op $c(i)$ gericht is. Dit vertaalt zich in de volgende collectie woorden:

   $$(H_{i,t} \cup G_{i,t,j} \cup \overline{G}_{i,t+1,j})$$

   Deze $\mathcal{O}(p(n))$ woorden beschrijven de voorwaarde op de volgende manier: Óf de wijzer wijst naar cel $c(i)$ op tijdstip t en dus geldt $H_{i,t} = 1$, óf de wijzer wijst *niet* naar cel $c(i)$. In dit geval is symbool j óf niet in de cel $c(i)$ voor de tijdstap, óf symbool j is *wel (nog steeds)* in $c(i)$ nadat de tijdstap gezet is.

   Nu moeten we alleen nog uitdrukken dat de overgangen van de ene in de volgende configuratie van de NDTM het directe resultaat zijn van de overgangsfunctie. De volgende drie verzamelingen van woorden zorgen hiervoor:

   $$(\overline{S}_{t,k} \cup \overline{H}_{i,t} \cup \overline{G}_{i,t,j} \cup S_{t+1,k'})$$
   $$(\overline{S}_{t,k} \cup \overline{H}_{i,t} \cup \overline{G}_{i,t,j} \cup G_{i,t+1,j'})$$
   $$(\overline{S}_{t,k} \cup \overline{H}_{i,t} \cup \overline{G}_{i,t,j} \cup H_{i+d,t+1})$$

met (k', j', d) zdd. $(s_{k'}, \gamma_{j'}, d) = \begin{cases} f(s_k, \gamma_j) & k \neq 1, 2 \\ (s_k, \gamma_j, 0) & k = 1, 2 \end{cases}$

Deze collectie is waar dan en slechts dan als voor iedere (i, t, j, k) met $s(t) = k, h(t) = i$ en $\gamma(h(t), t) = \gamma_j$ $(t \neq p(n))$ geldt dat $s(t+1) = s_k, \gamma(h(t), t+1) = \gamma_{j'}$ en $h(t+1) = h(t) + d$, waarbij (k', j', d) zdd. $(s_{k'}, \gamma_{j'}, d) = \begin{cases} f(s_k, \gamma_j) & k \neq 1, 2 \\ (s_k, \gamma_j, 0) & k = 1, 2 \end{cases}$

Het aantal woorden is $\mathcal{O}(p(n)^2)$.

We hebben nu het volgende aangetoond:

- Alle woorden zijn polynomiaal begrensd en ook het aantal uitspraken is polynonomiaal begrensd, dus de zin voor SAT wordt met een polynomiaal algoritme opgesteld.
- Het antwoord van SAT is 'ja' dan en slechts dan als het NDTM algoritme voor π op tijdstip $p(n)$ in toestand s_y is, d.w.z. dat π antwoord 'ja' heeft, dus in de taal van de NDTM zit.

Hieruit volgt dat iedere taal in $\mathcal{NP}$ tot SAT gereduceerd kan worden, dus SAT is $\mathcal{NP}$-volledig.

□

14 Concluderende Samenvatting

In het eerste deel van deze scriptie is ingegaan op het probleem van optimaal serveren. Op twee manieren, m.b.v dynamische programmering en Markov beslissingstheorie, hebben we de volgende optimale strategie gevonden:

Pas strategie 1 (Hard, Langzaam) toe als
$(p_1q_1 - p_2q_2) < 0$ en $p_1q_1 - p_2q_2(1 + p_1 - p_2) > 0$, i.e. $1 > \frac{p_1q_1}{p_2q_2} > 1 + (p_1 - p_2)$.
Pas strategie 2 (Hard, Hard) toe als
$(p_1q_1 - p_2q_2) > 0$, i.e. $\frac{p_1q_1}{p_2q_2} > 1$.
Pas strategie 3 (Langzaam, Langzaam) toe als
$(p_1q_1 - p_2q_2) < 0$ en $p_1q_1 - p_2q_2(1 + p_1 - p_2) < 0$, i.e. $1 + (p_1 - p_2) > \frac{p_1q_1}{p_2q_2}$.

Ook hebben we voor de serveerder de winstkansen per stand in de game berekend. We besloten het eerste deel met de simulatie van een tennispartij.

In het tweede deel lag de focus op het optimaal plannen van een RRT-toernooi onder een aantal voorwaarden. Het bleek dat dit probleem te modelleren is als een Maximum Kliekprobleem, zoals is uitgelegd. Verder zijn er enkele heuristieken behandeld om dit probleem op te lossen: 1-opt, H-1-opt en k-opt. Ook is er ingegaan op de complexiteit van het $RRTT$-probleem, en, in het algemeen, op de complexiteitstheorie. Hiervoor is het begrip 'Turing Machine' geïntroduceerd. We besloten het tweede deel met het bewijs van de stelling van Cook.

APPENDICES

A Winstkansen

Zie de volgende uitvoer van het MATLAB-programma, met parameters $p_1 = 0.64$ en $q_1 = 0.66$:

```
Toestand 1: De kans op gamewinst op stand 00-00 (1) is 0.68014
Toestand 2: De kans op gamewinst op stand 15-00 (1) is 0.80177
Toestand 3: De kans op gamewinst op stand 00-00 (2) is 0.63667
Toestand 4: De kans op gamewinst op stand 00-15 (1) is 0.51594
Toestand 5: De kans op gamewinst op stand 30-00 (1) is 0.90448
Toestand 6: De kans op gamewinst op stand 15-00 (2) is 0.76506
Toestand 7: De kans op gamewinst op stand 15-15 (1) is 0.66311
Toestand 8: De kans op gamewinst op stand 00-15 (2) is 0.46335
Toestand 9: De kans op gamewinst op stand 00-30 (1) is 0.31727
Toestand 10: De kans op gamewinst op stand 40-00 (1) is 0.9727
Toestand 11: De kans op gamewinst op stand 30-00 (2) is 0.8801
Toestand 12: De kans op gamewinst op stand 30-15 (1) is 0.81237
Toestand 13: De kans op gamewinst op stand 15-15 (2) is 0.60977
Toestand 14: De kans op gamewinst op stand 15-30 (1) is 0.4616
Toestand 15: De kans op gamewinst op stand 00-30 (2) is 0.26569
Toestand 16: De kans op gamewinst op stand 00-40 (1) is 0.12241
Toestand 17: De kans op gamewinst op stand 50-00 (1) is 1
Toestand 18: De kans op gamewinst op stand 40-00 (2) is 0.96294
Toestand 19: De kans op gamewinst op stand 40-15 (1) is 0.93584
Toestand 20: De kans op gamewinst op stand 30-15 (2) is 0.76825
Toestand 21: De kans op gamewinst op stand 30-30 (1) is 0.6457
Toestand 22: De kans op gamewinst op stand 15-30 (2) is 0.39582
Toestand 23: De kans op gamewinst op stand 15-40 (1) is 0.21309
Toestand 24: De kans op gamewinst op stand 00-40 (2) is 0.090007
Toestand 25: De kans op gamewinst op stand 00-50 (1) is 0
Toestand 26: De kans op gamewinst op stand 50-15 (1) is 1
Toestand 27: De kans op gamewinst op stand 40-15 (2) is 0.91292
Toestand 28: De kans op gamewinst op stand 40-30 (1) is 0.84923
Toestand 29: De kans op gamewinst op stand 30-30 (2) is 0.57296
Toestand 30: De kans op gamewinst op stand 30-40 (1) is 0.37093
Toestand 31: De kans op gamewinst op stand 15-40 (2) is 0.15668
Toestand 32: De kans op gamewinst op stand 15-50 (1) is 0
Toestand 33: De kans op gamewinst op stand 50-30 (1) is 1
Toestand 34: De kans op gamewinst op stand 40-30 (2) is 0.79535
Toestand 35: De kans op gamewinst op stand 30-40 (2) is 0.27274
Toestand 36: De kans op gamewinst op stand 30-50 (1) is 0
```

B Optimale strategie voor R. Nadal

Voor Nadal ($p_1 = 0.64, q_1 = 0.64, p_2 = 0.94$ en $q_2 = 0.48$) geeft het MATLAB-programma de volgende uitwerking:

```
De optimale strategie op stand 00-00 (1) is: hard serveren
De optimale strategie op stand 15-00 (1) is: hard serveren
De optimale strategie op stand 00-00 (2) is: langzaam serveren
De optimale strategie op stand 00-15 (1) is: hard serveren
De optimale strategie op stand 30-00 (1) is: hard serveren
De optimale strategie op stand 15-00 (2) is: langzaam serveren
De optimale strategie op stand 15-15 (1) is: hard serveren
De optimale strategie op stand 00-15 (2) is: langzaam serveren
De optimale strategie op stand 00-30 (1) is: hard serveren
De optimale strategie op stand 40-00 (1) is: hard serveren
De optimale strategie op stand 30-00 (2) is: langzaam serveren
De optimale strategie op stand 30-15 (1) is: hard serveren
De optimale strategie op stand 15-15 (2) is: langzaam serveren
De optimale strategie op stand 15-30 (1) is: hard serveren
De optimale strategie op stand 00-30 (2) is: langzaam serveren
De optimale strategie op stand 00-40 (1) is: hard serveren
De optimale strategie op stand 50-00 (1) doet er niet toe
De optimale strategie op stand 40-00 (2) is: langzaam serveren
De optimale strategie op stand 40-15 (1) is: hard serveren
De optimale strategie op stand 30-15 (2) is: langzaam serveren
De optimale strategie op stand 30-30 (1) is: hard serveren
De optimale strategie op stand 15-30 (2) is: langzaam serveren
De optimale strategie op stand 15-40 (1) is: hard serveren
De optimale strategie op stand 00-40 (2) is: langzaam serveren
De optimale strategie op stand 00-50 (1) doet er niet toe
De optimale strategie op stand 50-15 (1) doet er niet toe
De optimale strategie op stand 40-15 (2) is: langzaam serveren
De optimale strategie op stand 40-30 (1) is: hard serveren
De optimale strategie op stand 30-30 (2) is: langzaam serveren
De optimale strategie op stand 30-40 (1) is: hard serveren
De optimale strategie op stand 15-40 (2) is: langzaam serveren
De optimale strategie op stand 15-50 (1) doet er niet toe
De optimale strategie op stand 50-30 (1) doet er niet toe
De optimale strategie op stand 40-30 (2) is: langzaam serveren
De optimale strategie op stand 30-40 (2) is: langzaam serveren
De optimale strategie op stand 30-50 (1) doet er niet toe
De optimale strategie op stand Deuce (1) is: hard serveren
De optimale strategie op stand Voordl(1) is: hard serveren
De optimale strategie op stand Deuce (2) is: langzaam serveren
De optimale strategie op stand Nadeel(1) is: hard serveren
De optimale strategie op stand Voordl(2) is: langzaam serveren
De optimale strategie op stand Nadeel(2) is: langzaam serveren
De optimale strategie op stand 50-40 (1) doet er niet toe
De optimale strategie op stand 40-50 (1) doet er niet toe
```

C Simulatie

Dit is een stukje uitwerking van een wedstrijd tussen Nadal en Federer.

```
Rafael Nadal serveert.
De gekozen (optimale) strategie is (Hard, Langzaam).
Rafael Nadal wordt gebroken met stand 15-50 (1).

score =

     2     5

-------------------------------------------------------------
Roger Federer serveert.
De gekozen (optimale) strategie is (Hard, Langzaam).
Roger Federer behoudt zijn servicegame met stand 50-00 (1).

score =

     2     6

Roger Federer wint de set met:

score =

     2     6

De setstand is:

setstand =

     2     3

-------------------------------------------------------------
-------------------------------------------------------------
Roger Federer wint de wedstrijd met de volgende setstanden:

ans =

     2     6
     7     5
     6     3
     6     7
     2     6
```

D Tegenvoorbeeld H-1-opt

De volgende graaf is een voorbeeld van het niet-exact zijn van H-1-opt.

```
aantalknopen =

    19

aantaltakken =

   129

groottemetheuristiek =

     7

grootteexact =

     8


G =

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1
0 0 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1
0 1 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1
1 0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1
1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
1 1 0 0 0 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1
1 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1
1 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 0 1 1
1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 0 1 1
1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1
1 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1
1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1
1 1 0 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 1
0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1
1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1
1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 1
0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0


kliekmetheuristiek =

1  0  0  0  0  1  1  0  1  1  1  0  0  0  0  0  0  0  1

exactekliek =

0  0  1  1  0  0  0  1  0  0  1  0  0  1  1  1  1  0  0
```

E Toepassing op RRTT-probleem

Het programma geeft de volgende uitvoer, voor een instantie met 3 series en 4 spelers per serie:

```
Het aantal koppels is:

q =

    18

Dit is ook het aantal wedstrijden dat gepland moet worden om
alle wedstrijden te spelen
-----------------------------------------------------------------
Het aantal beschikbare uren per week is:

p =

     6

-----------------------------------------------------------------
Het random permutatieschema is:
(Spelers v, uren ->)

permutatieschema =

18  14   6   8   1   7   3  11  16  15   9   4  13   2   5  12  10  17
 8  12   9  15  17   5  13   2   3   1   7  11  14  10  16   4   6  18
 5  18   3  15  17   1   4   9   8   6  13  10  11  12   7  16   2  14
18  11  16  17   6   8   2   3  14  13   1  12   4   7   5  10   9  15
14  12  13  11  17  15  16   3   9   7   5   4   2   6   8  18   1  10
 8   6  18   5  17  13  12  16   9  10   1  14   7  15   2  11   3   4
14   3  18   5   2  17  15   8  10  16  13   6   1  11   7  12   9   4
18  10  13   9   4   6   2  15  14   8  17   5  11   7   1  16   3  12
14  15  16   6   7  12   8  18   3  13   5  17   9  11  10   1   4   2
13   4   3   1  18   2  17   6   9  15   8  10   5  14  11   7  12  16
11   2   9   3   5  18   8  12  13  10  14   7  17   1   6  15   4  16
 3  11  14   4  17  16  18   8  13   6  15   9   2  12  10   7   5   1

Hierin representeert iedere rij een speler en iedere kolom een speeluur.
-----------------------------------------------------------------
Het beschikbaarheidsschema is:
(Spelers v, uren ->)

Beschikbaarheidsschema =

0  0  1  1  1  1  1  0  0  0  1  1  0  1  1  0  0  0
1  0  1  0  0  1  0  1  1  1  1  0  0  0  0  1  1  0
1  0  1  0  0  1  1  1  1  1  0  0  0  0  1  0  1  0
0  0  0  0  1  1  1  1  0  0  1  0  1  1  1  0  1  0
0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  0  1  0
1  1  0  1  0  0  0  0  1  0  1  0  1  0  1  0  1  1
0  1  0  1  1  0  0  1  0  0  0  1  1  0  1  0  1  1
0  0  0  1  1  1  1  0  0  1  0  1  0  1  1  0  1  0
0  0  0  1  1  0  1  0  1  0  1  0  1  0  0  1  1  1
```

```
0  1  1  1  0  1  0  1  1  0  1  0  1  0  0  1  0  0
0  1  1  1  1  0  1  0  0  0  0  1  0  1  1  0  1  0
1  0  0  1  0  0  0  1  0  1  0  1  1  0  0  1  1  1

Hierin representeert iedere rij een speler en iedere kolom een speeluur.
(i,j) = 1 betekent dat speler i beschikbaar is op uur j
-----------------------------------------------------------------
De gevonden grootte van de kliek na toepassen van 1-opt is:

grootte =

    15

-----------------------------------------------------------------
De gevonden grootte van de kliek na toepassen van k-opt is:

grootte2 =

    15

    -----------------------------------------------------------------
De gevonden grootte van de kliek na toepassen van H-1-opt is:

grootte =

    15

-----------------------------------------------------------------
Het optimale speelschema is:
(Wedstrijden v, spelers, week, uur ->)

Speelschema =

    10    11     0     1     2
     1     2     0     1     3
     9    12     0     1     4
     7     8     0     1     5
     3     4     0     1     6
     1     3     0     2     1
     2     4     0     2     2
     9    10     0     2     3
     5     8     0     2     4
    11    12     0     2     6
     5     7     0     3     1
     1     4     0     3     2
     6     8     0     3     3
    10    12     0     3     4
     9    11     0     3     5

Hierin representeren de eerste en tweede kolom het koppel.
De vierde kolom representeert het weeknummer.
De vijfde kolom representeert het uurnummer (van de week).
-----------------------------------------------------------------
Verstreken tijd is 0.376874 seconden.
```

Referenties

[1] http://nl.wikipedia.org/wiki/Tennis, *laatste wijziging:* 11 juni 2009

[2] J.M. Norman, *Dynamic Programming in Tennis - when to use a Fast Service*, Operational Research Society Ltd. 1985

[3] http://www.australianopen.com/en-AU/scores/stats/day19/1701ms.html, *laatste wijziging:* 1 februari 2009

[4] S.L. George, *Optimal Strategy in Tennis: A Simple Probabilistic Model*, University of Texas, 1971

[5] L.C.M. Kallenberg, *dictaat Besliskunde 1*, Universiteit Leiden

[6] L.C.M. Kallenberg, *dictaat Besliskunde 3*, Universiteit Leiden

[7] F. Della Croce, R. Tadei, P.S. Asioli, *Scheduling a round robin tennis tournament under courts and players availability constraints*, Politecnico di Torino, 1999

[8] K. Katayama, A. Hamamoto, H. Narihisa, *Solving the Maximum Clique Problem by k-opt Local Search*, Okayama University of Science, 2004

[9] L.C.M. Kallenberg, *dictaat Besliskunde 3*, Universiteit Leiden

[10] L.C.M. Kallenberg, *dictaat Complexiteitstheorie*, Universiteit Leiden, 1982

[11] http://nl.wikipedia.org/wiki/Turingmachine, *laatste wijziging:* 12 mei 2009

[12] Herbert S. Wilf, *Algorithms and Complexity*,hfdst.5, University of Pennsylvania, 1986