# Inhoudsopgave

# Hoofdstuk 1

# COMPLEXITEITSTHEORIE

## 1.1 Inleiding

Hoe lastig is een probleem en hoe goed is een algoritme? Dit zijn de kernvragen in de complexiteitstheorie. Onder een *probleem* verstaan we in de complexiteitstheorie een algemene vraagstelling. Bijvoorbeeld: 'Het kortste pad probleem' of 'Het lineaire programmeringsprobleem'.[1] Als de parameters gespecificeerd zijn, bijvoorbeeld in het kortste pad probleem als het netwerk $N$ is gegeven (een netwerk is een gerichte graaf met één of meer functies op de pijlen; voor het kortste pad probleem is er één functie, namelijk de lengte), dan spreken we van een *instantie* van het probleem. Een *algoritme* voor een bepaald probleem is een recept dat kan worden gebruikt om het probleem op te lossen, d.w.z. het geeft voor iedere instantie van het probleem het juiste antwoord.

**Voorbeeld 1.1** *Grootste gemene deler*
Als voorbeeld beschouwen we het probleem om de *grootste gemene deler* van twee natuurlijke getallen $n$ en $m$ met $n \geq m$ te bepalen. Een bekend algoritme hiervoor is het *algoritme van Euclides*. Dit algoritme is gebaseerd op het herhaald toepassen van de eigenschap dat $ggd(n, m) = ggd(n - m, m)$, zodat $ggd(n, m) = ggd(m, l)$ met $l \equiv n \, (mod \; m)$. Het algoritme luidt als volgt.

**Algoritme 1.1** *Algoritme van Euclides om g.g.d(n,m) te bepalen*[2]

1. $l := n \, (mod \; m)$; $n := m$; $m := l$.
2. Als $l = 0$: $ggd(n, m) = n$ en STOP.

   Anders: ga naar stap 1.

Als we dit toepassen op $n = 461952$ en $m = 116298$, dan vinden we na 8 iteraties de grootste gemene deler via de tussenstappen (voor $n$): 461952, 116298, 113058, 3240, 2898, 342, 162, 18. Dus $ggd(461952, 116298) = 18$.

[1] Deze twee problemen zijn in het college Caleidoscoop aan de orde geweest.
[2] Dit algoritme is behandeld in het college Algebra 1.

Het is gewenst om algoritmen met elkaar te kunnen vergelijken. Als maat voor de *performance* van een algoritme is het gebruikelijk om te kijken naar de tijdsduur voordat het eindantwoord wordt verkregen. We spreken dan van *tijdcomplexiteit*. Soms wordt ook gekeken naar de hoeveelheid (geheugen)ruimte die de uitvoering van het algoritme in beslag neemt. Dit correspondeert met *ruimtecomplexiteit*. We zullen ons in dit college beperken tot het begrip tijdcomplexiteit, wat we kortweg *complexiteit* zullen noemen.

De tijdsduur is sterk afhankelijk van de gebruikte computer. Vandaar dat we niet de tijd zelf nemen, maar dat we het aantal *elementaire stappen* tellen. Een elementaire stap is een optelling, vermenigvuldiging, toekenning, vergelijking e.d. op een hypothetische computer, de *Turing machine*.[3] We maken geen onderscheid tussen de verschillende operaties: alle hebben tijdsduur één. De Turing machine zelf zullen we hier niet bespreken en wat geteld wordt zal later uit de context duidelijk worden. We zullen de complexiteitstheorie op een wat informele manier bespreken.[4]

Zij $P$ een bepaald probleem, $I$ een instantie van het probleem en $A$ een algoritme om het probleem op te lossen. De complexiteit om de instantie $I$ van probleem $P$ met algoritme $A$ op te lossen noteren we dan met $t_A(P, I)$. De complexiteit van algoritme $A$ om probleem $P$ op te lossen noteren we met $t_A(P)$ en definiëren we op de *worst case* wijze, d.w.z. we nemen hiervoor de complexiteit van de meest ongunstige instantie:

$$t_A(P) = sup_I \; t_A(P, I).$$

We meten de complexiteit van een algoritme als een functie van de *invoergrootte* van het algoritme. Maar, wat is de grootte van de invoer? De invoer is vaak een graaf, een verz. getallen, een matrix, etc. Om de invoer in de computer te krijgen moeten we deze op de een of andere manier coderen als een rij symbolen. De invoergrootte definiëren we als de lengte van deze rij.

Een gebruikelijke manier om getallen te coderen is de *binaire codering*. Voor getal $N$ zijn dan $\lceil log_2\,(N+1)\rceil$ 0'en en 1'en nodig. Zo wordt het getal $N = 39$ binair geschreven als 100111 (merk op dat $\lceil log_2\,(40)\rceil = 6$).

Een graaf $G = (V, E)$, waarbij $V$ de knooppuntenverz. en $E$ de takkenverz. is, kan men op diverse manieren coderen. Een mogelijkheid is de *structuurmatrix* te gebruiken. Dit is een $n \times n$ matrix $Q = (q_{ij})$ met

$$q_{ij} = \text{het aantal takken (pijlen bij een gerichte graaf) tussen de knooppunten } i \text{ en } j.$$

[3] Genoemd naar de Engelsman Alan Turing (1912-1954), die in 1936 reeds een computer ontwierp, dus voordat dit technologisch mogelijk was (A.M. Turing: *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2 42 (1936) 230–265, and a corregidum in 43 (1937) 544–546). Zie ook http://turnbull.mcs.st-and.ac.uk/~history/Mathematicians/Turing.html.

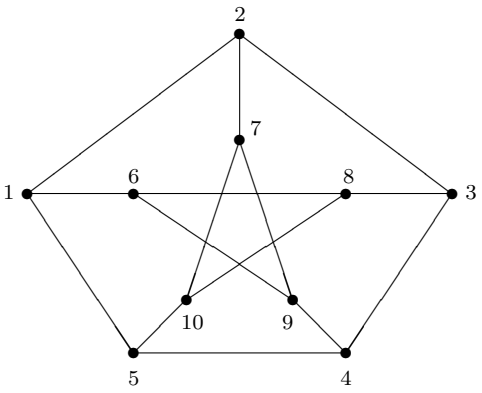
[4] Een uitvoerige beschrijving van de complexiteitstheorie met vele bewijzen kan gevonden worden in C.H. Papadimitriou: *Computational Complexity*, Addison-Wesley, 1994.

Dit geeft een invoer ter grootte $n^2$. Indien de graaf weinig takken (pijlen) heeft, dan is het vaak handiger om de *burenlijst* te gebruiken. Voor elk knooppunt $v$ geven we de lijst $L[v]$ van de buren:

$$L[v] = \{w \mid (v, w) \in E\}.$$

Omdat (in een niet-gerichte graaf) elke tak tweemaal in de lijst van buren voorkomt, hebben deze lijsten tezamen $2m$ elementen.

**Voorbeeld 1.2** *Representaties van grafen*




Representatie met de structuurmatrix:

$$Q = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Representatie met burenlijsten:

$L[1] = \{2, 5, 6\}$; $L[2] = \{1, 3, 7\}$; $L[3] = \{2, 4, 8\}$; $L[4] = \{3, 5, 9\}$; $L[5] = \{1, 4, 10\}$;
$L[6] = \{1, 8, 9\}$; $L[7] = \{2, 9, 10\}$; $L[8] = \{3, 6, 10\}$; $L[9] = \{4, 6, 7\}$; $L[10] = \{5, 7, 8\}$.

Meestal drukt men de complexiteit vaak niet direct uit in de grootte van de invoer, maar in meer globale termen. Bij een graaf gebruikt men vaak het aantal knooppunten ($n$) en het aantal takken ($m$); soms zelfs met alleen het aantal knooppunten. Verder vergelijken we globaal: zo maken we bijvoorbeeld geen onderscheid tussen de functies $n^2$ en $2n^2$, maar wel tussen $n$, $n^2$ en $2^n$, omdat deze voor grote waarden van $n$ een verschillende *groeisnelheid* hebben: de groeisnelheid van $n$, $n^2$ en $2^n$ noemen we resp. *lineair, kwadratisch* en *exponentieel*.

We kunnen dit wiskundig als volgt preciezer beschrijven.
Voor twee functies $f(n)$ en $g(n)$ (van de natuurlijke getallen naar de positieve reële getallen) schrijven we:

$f(n) = \mathcal{O}(g(n))$ als er positieve constanten $c$ en $n_0$ bestaan zdd. $f(n) \leq c \cdot g(n)$ voor alle $n \geq n_0$.

In dit geval zeggen we dat $f$ niet sneller groeit dan $g$. We schrijven

$f(n) = \Omega(g(n))$ als er positieve constanten $c$ en $n_0$ bestaan zdd. $f(n) \geq c \cdot g(n)$ voor alle $n \geq n_0$.

We zeggen dan dat $f$ niet langzamer groeit dan $g$. Tenslotte schrijven we

$$f(n) = \Theta(g(n)) \text{ als } f(n) = \mathcal{O}(g(n)) \text{ en } f(n) = \Omega(g(n)),$$

d.w.z. er positieve constanten $c_1, c_2$ en $n_0$ bestaan zdd. $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ voor alle $n \geq n_0$. Nu zeggen we dat $f$ en $g$ even snel groeien.

**Voorbeeld 1.3**
Laat $f(n) = 10n^3 + 100n^2 + 200$ en $g(n) = n^3$.
Dan is $f(n) = \Theta(g(n))$, want $10n^3 \leq f(n) \leq 11n^3$ voor alle $n \geq 101$.

**Voorbeeld 1.4** *Grootste gemene deler (vervolg)*
Om de complexiteit van het algoritme van Euclides te bepalen merken we allereerst op dat als $l \equiv n \,(mod\ m)$, met $n \geq m$, dan geldt dat $l < \frac{n}{2}$, immers:
als $m \leq \frac{n}{2}$, dan $l < m \leq \frac{n}{2}$ en als $m > \frac{n}{2}$, dan $l = n - m < \frac{n}{2}$.
Zij $n_i$ en $m_i$ de waarden van $n$ en $m$ in iteratie $i$, dan geldt:

$$n_1 = n,\ m_1 = m,\ n_i = m_{i-1} \text{ en } m_i = n_{i-1} \,(mod\ m_{i-1}).$$

Hieruit volgt:

$$m_i = n_{i-1} \,(mod\ m_{i-1}) < \frac{n_{i-1}}{2} = \frac{m_{i-2}}{2} \text{ voor alle } i \geq 3.$$

In iteratie $k$ (veronderstel $k$ oneven, zeg $k = 2d + 1$) is $m_k \leq \frac{m}{2^d}$. Het algoritme stopt zeker als $m_k \leq 1$, d.w.z. als $d \geq log_2\, m$. Het geval dat $k$ even is gaat analoog. Er zijn dus hoogstens $\mathcal{O}(log_2\, m)$ iteraties. Iedere iteratie heeft complexiteit $\mathcal{O}(1)$, zodat het algoritme van Euclides complexiteit $\mathcal{O}(log_2\, m)$ heeft.

Wanneer vinden we dat een algoritme voldoende snel is? In de praktijk blijkt dat algoritmen met een *polynomiale groeisnelheid*, d.w.z. dat de complexiteit $\mathcal{O}(n^k)$ is voor zekere $k > 0$, bevredigend werken. We spreken dan van een algoritme van de *orde* $k$. Dit staat in tegenstelling tot algoritmen met een *exponentiële groeisnelheid*, waarvoor de complexiteit $\Omega(k^n)$ is voor een zekere $k > 1$. De praktische toepasbaarheid van deze begrippen wordt in de volgende tabel geïllustreerd.

| functie | $n = 10$ | $n = 100$ | $n = 1000$ |
|---|---|---|---|
| $n\, log_{10}\, n$ | 10 | $2 \times 10^2$ | $3 \times 10^3$ |
| $n^2$ | $10^2$ | $10^4$ | $10^6$ |
| $n^3$ | $10^3$ | $10^6$ | $10^9$ |
| $2^n$ | $\approx 10^3$ | $\approx 10^{30}$ | $\approx 10^{300}$ |
| $n!$ | $\approx 10^6$ | $\approx 10^{158}$ | $\approx 10^{2567}$ |

We zien dat de toename van de machten bij $n = 10, 100$ en 1000 van de polynomiale functies *additief* is (bij $n^3$ van 3 via 6 naar 9, dus steeds +3), terwijl dat bij de exponentiële functies *multiplicatief* is (van 3 via 30 naar 300, dus steeds $\times 10$). Dit laatste loopt dus veel sneller op. Toch zal een groeisnelheid van $n^{100}$, die polynomiaal is, in de praktijk toch niet erg bevredigend

werken. Dit lijkt in tegenspraak met de bewering dat polynomiale algoritmen snel zijn. In de praktijk blijkt polynomiaal echter vrijwel altijd van de orde 4 of lager te zijn. Op grond van deze theoretische en praktische overwegingen noemen we polynomiale algoritmen *goed* of *efficiënt* en exponentiële algoritmen *slecht* of *inefficiënt*.[5]

We zullen nu de problemen zelf gaan indelen. Zij $f$ de functie die gemaximaliseerd moet worden en laat $X$ de verz. van toegelaten oplossingen zijn.
Een *optimaliseringsprobleem* is het probleem om de *beste oplossing* te vinden, d.w.z.

$$\text{Bepaal } x^* \in X \text{ zdd. } f(x^*) \geq f(x) \text{ voor alle } x \in X.$$

Onder het *evaluatieprobleem* verstaan we het probleem om de *beste waarde van de doelfunctie* te bepalen, d.w.z.

$$\text{Bepaal } max\{f(x) \mid x \in X\}.$$

Onder het *herkenningsprobleem* verstaan we het probleem of er een *toegelaten oplossing met waarde minstens een gegeven getal* $L$ bestaat, d.w.z.

$$\text{Is er een } x^* \in X \text{ zdd. } f(x^*) \geq L?$$

In tegenstelling tot de eerste twee versies (optimaliseringsprobleem en evaluatieprobleem) is het herkenningsprobleem een probleem dat met 'ja' of met 'nee' beantwoord wordt. Daarom worden deze problemen ook wel *ja-nee-problemen* genoemd. Het is duidelijk dat het optimaliseringsprobleem zelf, het evaluatieprobleem en het herkenningsprobleem in deze volgorde steeds makkelijker worden. De vraag is nu: hebben deze problemen dezelfde complexiteit of niet? Met andere woorden: kunnen we het evaluatieprobleem oplossen met een (hypothetisch) algoritme voor het herkenningsprobleem en kunnen we het optimaliseringsprobleem oplossen met een algoritme voor het evaluatieprobleem?

Onder de (praktische en realistische) aanname dat het optimum geheeltallig is en ligt in het interval $[0, U]$, kan worden aangetoond dat het evaluatieprobleem in polynomiale tijd oplosbaar is d.e.s.d. als het herkenningsprobleem in polynomiale tijd oplosbaar is. Dit kan door een *binaire zoekmethode* toe te passen, waarmee door maximaal $\lceil log_2 U \rceil + 1$ keer het herkenningsprobleem op te lossen het evaluatieprobleem kan worden opgelost. Immers, beschouw eerst het herkenningsprobleem of er een toegelaten oplossing is met waarde minstens $\frac{1}{2}U$. Als het antwoord ja is, dan ligt het optimum in het interval $[\lceil \frac{1}{2}U \rceil, U]$, en als het antwoord nee is, dan in het interval $[0, \lfloor \frac{1}{2}U \rfloor]$; vervolgens gaan we verder op een interval dat half zo groot is en als het interval slechts één getal bevat, dan is het herkenningsprobleem hetzelfde als het evaluatieprobleem.

Er is in het algemeen geen methode bekend om het optimaliseringsprobleem op te lossen met een algoritme voor het evaluatieprobleem. Wel kan dat in vele concrete gevallen. Daarom veronderstellen we:

[5] Deze karakterisering van algoritmen is in 1965 door Edmonds geïntroduceerd in zijn artikel: J. Edmonds, *Path, trees and flowers*, Canadian Journal of Mathematics 17 (1965) 449–467.

**Aanname 1.1**
*Het optimaliseringsprobleem is equivalent met het herkenningsprobleem.*

In de complexiteitstheorie gaan we er daarom vanuit dat de problemen als herkenningsproblemen zijn geformuleerd: problemen die met 'ja' of met 'nee' beantwoord worden. We geven hiervan enkele voorbeelden (definities van begrippen als 'keten', 'kring', 'pad' en 'ronde' worden later in het hoofdstuk Grafentheorie besproken; zie ook de index).

*Samenhangprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$.
Probleem: Is $G$ samenhangend (d.w.z. is er voor ieder tweetal knooppunten een keten die deze knooppunten verbindt)?

*Eulerkringprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$.
Probleem: Heeft $G$ een Euler kring (d.w.z. is er een kring zdd. iedere tak precies één keer in deze kring voorkomt)?

*Hamiltonkringprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$.
Probleem: Heeft $G$ een Hamilton kring (d.w.z. is er een kring zdd. ieder knooppunt precies één keer in deze kring voorkomt)?

*Hamiltonrondeprobleem*
Gegeven: Gerichte graaf $G = (V, A)$.
Probleem: Heeft $G$ een Hamilton ronde (d.w.z. is er een ronde zdd. ieder knooppunt precies één keer in deze ronde voorkomt)?

*Hamiltonketenprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$.
Probleem: Heeft $G$ een Hamilton keten (d.w.z. is er een keten zdd. ieder knooppunt precies één keer in deze keten voorkomt)?

*Hamiltonpadprobleem*
Gegeven: Gerichte graaf $G = (V, A)$.
Probleem: Heeft $G$ een Hamilton-pad (d.w.z. is er een pad zdd. ieder knooppunt precies één keer in deze keten voorkomt)?

*Graafisomorfieprobleem*
Gegeven: Twee grafen $G_1 = (V_1, E_1)$ en $G_2 = (V_2, E_2)$.
Probleem: Zijn $G_1$ en $G_2$ isomorf (d.w.z. dat beide grafen evenveel knooppunten hebben en dat de knooppunten van $G_2$ hernummerd kunnen worden zdd. in de nieuwe nummering $(i, j)$ een tak is van $G_2$ d.e.s.d. als $(i, j)$ een tak is in $G_1$)?

*Kliekenprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$ en een natuurlijk getal $k$.
Probleem: Heeft $G$ een kliek met $k$ elementen (d.w.z. is er een $k$-tal knooppunten zdd. ieder tweetal van deze $k$ knooppunten door een tak verbonden is)?

*Knooppuntenbedekkingsprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$ en een natuurlijk getal $k$.
Probleem: Heeft $G$ een knooppuntenbedekking met $k$ elementen (d.w.z. is er een $k$-tal knooppunten zdd. iedere tak van de graaf incident is met minstens één van deze $k$ knooppunten)?

*Onafhankelijkheidsprobleem*
Gegeven: Niet-gerichte graaf $G = (V, E)$ en een natuurlijk getal $k$.
Probleem: Heeft $G$ een onafhankelijke verzameling met $k$ elementen (d.w.z. is er een $k$-tal knooppunten zdd. geen enkel tweetal van deze $k$ knooppunten door een tak van de graaf verbonden is)?

*Partitieprobleem*
Gegeven: Een verz. natuurlijke getallen $a_1, a_2, \ldots, a_n$.
Probleem: Is er een partitie in twee delen zdd. de sommen over de partities aan elkaar gelijk zijn (d.w.z. is er een $S \subseteq \{1, 2, \ldots, n\}$ zdd. $\sum_{j \in S} a_j = \sum_{j \notin S} a_j$)?

*Knapzakprobleem*
Gegeven: Een verz. natuurlijke getallen: $c_1, c_2, \ldots, c_n, b$.
Probleem: Kunnen aan de variabelen $x_1, x_2, \ldots, x_n$ waarden 0 of 1 worden toegekend zdd. $\sum_{j=1}^{n} c_j x_j = b$?

*Kortste-padprobleem*
Gegeven: Netwerk $N = (V, A)$ met lengtefunctie $l : A \to \mathbb{R}_+$, twee knooppunten $v_1, v_n \in V$ en een $k \in \mathbb{N}$.
Probleem: Is er een pad van knooppunt $v_1$ naar knooppunt $v_n$ met lengte hoogstens $k$?

*Handelsreizigersprobleem*
Gegeven: Een volledig netwerk $N = (V, A)$ met lengtefunctie $l : A \to \mathbb{R}_+$, en een $k \in \mathbb{N}$.
Probleem: Heeft $N$ een Hamilton-ronde met lengte hoogstens $k$?

*LP-probleem (lineaire programmering)*
Gegeven: $A$ een $m \times n$-matrix, $p \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ en $k \in \mathbb{R}$.
Probleem: Is er een $x \in \mathbb{R}^n$ met $Ax \leq b,\ x \geq 0$ met $p^T x \geq k$?

**Vraag 1.1**
Bepaal voor ieder van de volgende functies $f(n)$ een functie $g(n)$ zdd. $f(n) = \Theta(g(n))$ met $g(n)$ een zo eenvoudig mogelijke functie:
a. $f(n) = 4n^3 + n^3\ log_2\, n$;
b. $f(n) = 2^n + 10n^5$.

**Vraag 1.2**

Laten de functies $f$ en $g$ op $\mathbb{N}$ gedefinieerd zijn door:

$$f(n) = \begin{cases} n^2 & \text{als } n \text{ oneven is} \\ n^3 & \text{anders} \end{cases} \qquad g(n) = \begin{cases} n^2 & \text{als } n \text{ een priemgetal is} \\ n^3 & \text{anders} \end{cases}$$

Ga na welke van de volgende uitspraken waar zijn:

a. $f(n) = \Omega(n^2)$.

b. $f(n) = \mathcal{O}(g(n))$.

c. $f(n) = \Theta(g(n))$.

**Vraag 1.3**

a. Veronderstel dat een algoritme met complexiteit $n^2$ op computer I in één uur een probleem met een input $n = 100$ aan kan. Computer II is 100 maal zo snel als computer I.
Hoe groot kan de input zijn als het algoritme één uur op computer II draait?

b. Dezelfde vraag, maar nu voor een algoritme met complexiteit $2^n$.

c. Dezelfde vraag, maar nu voor een algoritme met complexiteit $n!$.

## 1.2 De klassen $\mathcal{P}$ en $\mathcal{NP}$

Laat

$$\mathcal{P} = \{\text{herkenningsproblemen die oplosbaar zijn met een polynomiaal algoritme}\}.$$

Als een polynomiaal algoritme gevonden is, dan behoort het probleem zeker tot klasse $\mathcal{P}$. Echter, indien er nog geen polynomiaal algoritme gevonden is, zou dit wel kunnen bestaan. Alle problemen oplosbaar met een polynomiaal algoritme (gevonden of niet) rekenen we tot de klasse $\mathcal{P}$.

Vervolgens introduceren we de klasse $\mathcal{NP}$:

$\mathcal{NP}$ = {herkenningsproblemen waarvoor geldt dat voor iedere *ja-instantie* er een certificaat bestaat, waarmee in polynomiale tijd nagegaan kan worden dat dit antwoord juist is}.

Een *certificaat* is een recept, waarmee kan worden nagegaan dat het antwoord van een ja-instantie juist is. We lichten het begrip certificaat toe voor het Handelsreizigersprobleem, waarvan we zullen laten zien dat het in $\mathcal{NP}$ zit.

Als een volledig netwerk $N = (V, A)$ met lengtefunctie $l : A \to \mathbb{R}_+$ en een $k \in \mathbb{N}$ gegeven is zdd. er een Hamiltonronde met lengte hoogstens $k$ bestaat, dan kan - door deze Hamiltonronde te geven (zo'n ronde is voor dit probleem het certificaat) - in $\mathcal{O}(n)$ worden nagegaan dat de gegeven ronde inderdaad bestaat (voor iedere gegeven verbinding gaan we na of deze in het netwerk aanwezig is) en een lengte van hoogstens $k$ heeft (tel de lengtes van de $n$ gegeven verbindingen bij elkaar op).

Uit de definitie van $\mathcal{NP}$ volgt ook dat $\mathcal{P} \subseteq \mathcal{NP}$, immers voor iedere $P \in \mathcal{P}$ kan als certificaat het polynomiale algoritme om $P$ op te lossen worden genomen.

Waarom is de klasse $\mathcal{NP}$ interessant? Dat is niet alleen omdat $\mathcal{P}$ ertoe behoort, maar vooral omdat vrijwel alle combinatorische optimaliseringsproblemen hiertoe behoren. We zullen hier later een aantal voorbeelden van zien. Een centrale vraag in de complexiteitstheorie is of $\mathcal{P}$ en $\mathcal{NP}$ werkelijk verschillend zijn.[6] Zou $\mathcal{P} = \mathcal{NP}$, dan zou er voor elk probleem uit $\mathcal{NP}$ een polynomiaal algoritme zijn. Dit lijkt uiterst onwaarschijnlijk, maar tot op heden is men er nog niet in geslaagd om te bewijzen dat $\mathcal{P} \neq \mathcal{NP}$. Als men dit wil bewijzen, dan moet worden aangetoond dat het moeilijkste probleem uit $\mathcal{NP}$ nog polynomiaal oplosbaar is. Om te weten te komen wat het moeilijkste probleem uit $\mathcal{NP}$ is, is het nodig om de moeilijkheid van problemen met elkaar te kunnen vergelijken. Daarvoor gebruiken we het begrip *polynomiale reduceerbaarheid*.

We zeggen dat een probleem $P$ *polynomiaal gereduceerd* kan worden tot probleem $Q$ (informeel gezegd: $P$ is niet moeilijker dan $Q$ en genoteerd met $P \preceq Q$) als iedere invoer van een instantie van $P$ in polynomiale tijd kan worden omgezet in invoer van een instantie van $Q$ zdd. een ja-instantie van $P$ overgaat in een ja-instantie van $Q$ en hetzelfde geldt voor nee-instanties.

Als er een polynomiaal algoritme voor $Q$ is, dan is er ook een polynomiaal algoritme voor $P$, immers:
Stel er is een polynomiaal algoritme voor $Q$ dat $m^k$ elementaire stappen nodig heeft om een instantie van $Q$ met inputgrootte $m$ op te lossen. Neem verder aan dat een instantie $I$ van $P$ met inputgrootte $n$ met $n^l$ elementaire stappen kan worden omgezet in een instantie $J$ van $Q$ zdd. ja- en nee-instanties precies in elkaar overgaan. Dan is de inputgrootte van $J$ hoogstens $n^l$, zodat het algoritme hoogstens $(n^l)^k = n^{lk}$ elementaire stappen nodig heeft, wat weer polynomiaal is.
Op analoge wijze als hierboven kan worden aangetoond dat de eigenschap van polynomiale reduceerbaarheid *transitief* is (ga dit zelf na).

**Voorbeeld 1.5** *Partitieprobleem $\preceq$ Knapzakprobleem*
Laat $a_1, a_2, \ldots, a_n$ een instantie zijn van het Partitieprobleem.
Merk op dat er een partitie van $\{1, 2, \ldots, n\}$ in $S$ en het complement van $S$ bestaat zodanig dat $\sum_{i \in S} a_i = \sum_{i \notin S} a_i$ is d.e.s.d. als

$$\sum_{i=1}^{n} a_i = 2 \sum_{i \in S} a_i = 2 \sum_{i=1}^{n} a_i x_i \text{ met } x_i \in \{0, 1\} \text{ en } x_i = 1 \text{ d.e.s.d. als } i \in S.$$

Hieruit volgt hoe een instantie van het Knapzakprobleem wordt geconstrueerd:
Neem $m = n;\ c_j = a_j,\ 1 \leq j \leq m$ en $b = \frac{1}{2} \sum_{j=1}^{m} c_j$.

[6] Het probleem of $\mathcal{P} = \mathcal{NP}$ is een van de zeven grote open problemen uit de wiskunde. Ze zijn door een internationaal comité op 24 mei 2000 de *millenniumproblemen* genoemd. Het Clay Mathematical Institute uit Cambridge, Massachusetts, USA, heeft voor het oplossen van deze problemen een beloning van één miljoen dollar in het vooruitzicht gesteld: zie www.claymath.org/millennium/P_vs_NP/.
Een andere interessante site voor de $\mathcal{P}$ versus $\mathcal{NP}$ kwestie is www.win.tue.nl/~gwoegi/P-versus-NP.htm.

Dan is $\sum_{j=1}^{m} c_j x_j = b$ d.e.s.d. als $2\sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n} a_i$, waaruit de reductie volgt.
Ook is direct duidelijk dat de reductie polynomiaal is (lineair in $n$).

Op deze wijze kunnen we op zoek gaan naar de moeilijkste problemen van $\mathcal{NP}$. Dit zijn de problemen $P$ waarvoor geldt dat $Q \preceq P$ voor alle $Q \in \mathcal{NP}$. We noemen deze problemen de *$\mathcal{NP}$-volledige* problemen en noteren deze verzameling met $\mathcal{NPC}$:

$$\mathcal{NPC} = \{\text{herkenningsproblemen } P \text{ uit } \mathcal{NP} \text{ waarvoor geldt dat } Q \preceq P \text{ voor alle } Q \in \mathcal{NP}\}.$$

De klasse $\mathcal{NPC}$ heeft de volgende eigenschappen:
1. Voor geen enkel probleem uit deze klasse is een polynomiaal algoritme bekend.
2. Als er een polynomiaal algoritme bestaat voor één $\mathcal{NP}$-volledig probleem, dan bestaat er een polynomiaal algoritme voor elk $\mathcal{NP}$-volledig probleem en is $\mathcal{P} = \mathcal{NP}$.

Algemeen wordt vermoed dat er voor geen enkel $\mathcal{NP}$-volledig probleem een polynomiaal algoritme bestaat. Men zal vaak pas tevreden zijn als men voor een bepaald probleem òfwel een polynomiaal algoritme heeft gevonden òfwel heeft aangetoond dat het probleem $\mathcal{NP}$-volledig is.

De bewijzen dat een probleem $P$ tot $\mathcal{NPC}$ behoort gaan meestal als volgt.
1. Bewijs dat $P$ tot $\mathcal{NP}$ behoort.
2. Reduceer $Q$ polynomiaal tot $P$ voor een probleem $Q$, waarvan al is aangetoond dat het tot $\mathcal{NPC}$ behoort.

Om dit te kunnen doen moet er een eerste probleem zijn dat tot $\mathcal{NPC}$ behoort en daarvoor is dus ook een ander bewijs nodig. Dit is in 1971 door Cook [7] gedaan voor het *Satisfiabilityprobleem*, dat hieronder wordt geïntroduceerd.

Een *Boolse variabele* $x$ is een variabele die de waarde 0 of 1 kan aannemen; de ontkenning $\overline{x}$ is weer een Boolse variabele met waarde $1 - x$. Een *letter* is een $x$ of een $\overline{x}$. Een *alfabet* is een verz. $X = \{x_1, x_2, \ldots, x_n\}$ , waarbij $x_i$ een Boolse variabele is, $1 \leq i \leq n$. Een *woord* $C$ is een disjunctie van letters, bijv. $C = x_1 \cup \overline{x}_2 \cup \overline{x}_5$. Een woord is *waar* als tenminste één van de variabelen de waarde 1 heeft; de disjunctie heeft dan de waarde 1. Het woord $x_1 \cup \overline{x}_2 \cup \overline{x}_5$ is niet waar d.e.s.d. als $x_1 = 0,\ x_2 = x_5 = 1$. We mogen wel aannemen dat in een woord iedere Boolse variabele hoogstens één keer voorkomt (als zowel $x$ als $\overline{x}$ in een disjunctie voorkomt is het woord altijd waar en kan het worden weggelaten): een woord heeft dus maximaal $n$ letters. Een *zin* $\mathcal{C}$ is een conjunctie van een aantal woorden:

[7] S.A. Cook: *The complexity of theorem-proving procedures*, Proceedings 3rd Annual ACM Symposium on Theory of Computing (1971) 151-158. Dit artikel is in feite de start van een uitgebreide literatuur over complexiteit. Cook sprak ook het vermoeden uit dat $\mathcal{NP} \neq \mathcal{P}$. Karp (R.M. Karp: *Reducibility among combinatorial problems*, in R.E. Miller and J.W. Thatcher (eds.): *Complexity of computer computations*, Plenum Press, New York (1972) 85–103.) heeft voor een groot aantal combinatorische optimaliseringsproblemen aangetoond dat ze $\mathcal{NPC}$ behoren. De formele beschijvingen van $\mathcal{P}$, $\mathcal{N}$ en $\mathcal{NPC}$ zijn eveneens van Karp afkomstig (R.M. Karp: *On the computational complexity of combinatorial problems*, Networks 5 (1975) 45–68.).

$$\mathcal{C} = C_1 \cap C_2 \cap \cdots \cap C_m, \text{ met } C_j \text{ een woord, } 1 \leq j \leq m.$$

Een zin heet *waar* als ieder woord van de zin waar is, d.w.z. dat de Boolse formule waarin de woorden verbonden zijn door conjuncties de waarde 1 heeft.

Het Satisfiabilityprobleem, afgekort $SAT$, is nu het volgende probleem:
Gegeven een zin, is er een toekenning van 1'en en 0'en aan de Boolse variabelen zodanig dat de zin waar is.
De input parameters zijn $n$ (het aantal letters) en $m$ (het aantal woorden).
Als voorbeeld beschouwen we de zin

$$\mathcal{C} = (x_1 \cup \overline{x}_2) \cap (x_1 \cup \overline{x}_3) \cap (x_2 \cup \overline{x}_3) \cap (\overline{x}_1 \cup x_3).$$

Deze zin is waar, want de toekenning $x_1 = x_2 = x_3 = 1$ voldoet.

Cook heeft de volgende baanbrekende stelling bewezen, die we zonder bewijs geven.[8]

**Stelling 1.1** *(Stelling van Cook, 1971)*
$SAT \in \mathcal{NPC}$.

We zullen wel bewijzen dat ook 3-$SAT$, de speciale versie van $SAT$ waarin ieder woord 3 letters heeft, tot $\mathcal{NPC}$ behoort.

**Stelling 1.2**
*3-$SAT \in \mathcal{NPC}$.*

**Bewijs**
Het is voldoende om te laten zien dat iedere instantie van $SAT$ in polynomiale tijd is om te zetten in een instantie van 3-$SAT$ zdd. ja-instanties in elkaar overgaan.
Beschouw de zin

$$\mathcal{C} = C_1 \cap C_2 \cap \cdots \cap C_m,$$

waarbij ieder woord $C_i$ een disjunctie is met letters uit het alfabet

$$X = \{x_1, x_2, \ldots, x_n\}$$

met $x_i$ een Boolse variabele $1 \leq i \leq n$.
We zullen een zin $\mathcal{C}'$ met woorden van 3 letters construeren, waarbij de letters afkomstig zijn van

$$X' = X \cup \{\cup_{j=1}^{m} X'_j\}$$

zdd. $\mathcal{C}'$ waar is d.e.s.d. als $\mathcal{C}$ waar is.

[8] Voor het bewijs van deze stelling verwijzen we naar C.H. Papadimitriou and K. Steiglitz: *Combinatorial optimization: Algorithms and complexity*, Prentice Hall (1982) 353–358.

Neem een $C_j$, zeg

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

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 $C'_j = (y_1 \cup y_2 \cup z_1) \cap (\overline{z}_1 \cup y_3 \cup z_2) \cap (\overline{z}_2 \cup y_4 \cup z_3) \cap$ $\cdots \cap (\overline{z}_{k-4} \cup y_{k-2} \cup z_{k-3}) \cap (\overline{z}_{k-3} \cup y_{k-1} \cup y_k)$.

We zullen nu laten zien dat $C_j$ waar is d.e.s.d. als $C'_j$ waar is.

Het is direct in te zien dat dit juist is als $k \leq 3$; laat dus $k \geq 4$.

Stel $C_j$ is waar, d.w.z. ten minste één $y$-variabele is gelijk 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 $z_i = \begin{cases} 1 & \text{voor} \quad 1 \leq i \leq l-2; \\ 0 & \text{voor} \quad l-1 \leq i \leq k-3. \end{cases}$

We zien dat hiermee ook $C'_j$ waar is.

Stel $C_j$ is niet waar, d.w.z. $y_i = 0,\ 1 \leq i \leq k$.

Veronderstel 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 3 letters. Het opstellen van $\mathcal{C}'$ heeft dus complexiteit $\mathcal{O}(nm)$, wat polynomiaal is. □

We zullen nu aantonen dat het kliekenprobleem, dat in de literatuur wordt aangeduid met $CLIQUE$, ook tot $\mathcal{NPC}$ behoort.

**Stelling 1.3**
$CLIQUE \in \mathcal{NPC}$.

**Bewijs**
Het is duidelijk dat $CLIQUE$ tot $\mathcal{NP}$ behoort. We zullen bewijzen dat $3\text{-}SAT \preceq CLIQUE$. Neem een zin met $k$ woorden, zeg de woorden

$$C_i = x_{i1} \cup x_{i2} \cup x_{i3},\ i = 1, 2. \ldots, k,$$

waarbij iedere $x_{ij}$ de Boolese variabele of de ontkenning ervan is, behorend bij de $j$-de letter van het $i$-de woord.
Neem de volgende graaf

$$G = (V, E),\ \text{met } V = \{(i, j)\},\ \text{waarbij } 1 \leq i \leq k,\ j = 1, 2, 3.$$

De knooppunten $(i,1),(i,2),(i,3)$ corresponderen met het $i$-de woord. We verbinden een knooppunt $(i,j)$ met alle knooppunten die horen bij een letter van een ander woord, tenzij die letter de ontkenning is van de letter die hoort bij $(i,j)$. De constructie van $G$ is polynomiaal.
Uit de definitie van de takken van de graaf volgt: als $W$ met $|W| = k$ een kliek is, dan bevat $W$ precies één letter uit ieder woord en nooit zowel een letter als zijn ontkenning. De variabelen van $W$ kunnen dus alle de waarde 1 krijgen en daarmee worden alle woorden van de zin waar. Ook geldt dat als een zin waar is er een kliek met $k$ elementen is te vinden (ga dit zelf na). De zin is dus waar d.e.s.d. als er een kliek met $k$ elementen is. □

Als men voor een probleem geen polynomiaal algoritme kan vinden, dan kan men zich afvragen of het probleem misschien $\mathcal{NP}$-volledig is. Dit is voor vele problemen uitgezocht. Zonder bewijs vermelden we dat het Hamiltonkringprobleem, het Hamiltonrondeprobleem en het Handelsreizigersprobleem in $\mathcal{NPC}$ zitten. Onderstaand voorbeeld laat zien dat als het Hamiltonrondeprobleem tot $\mathcal{NPC}$ behoort ook het Hamiltonpadprobleem tot $\mathcal{NPC}$ behoort.

**Voorbeeld 1.6**
Noem het Hamiltonrondeprobleem probleem $Q$ en het Hamiltonpadprobleem probleem $P$. Het is duidelijk dat $P$ tot $\mathcal{NP}$ behoort. Het is dus voldoende om te bewijzen dat $Q$ polynomiaal tot $P$ kan worden gereduceerd. Zij $G = (V, A)$ een instantie van $Q$ met $V = \{1, 2, \ldots, n\}$. Construeer nu als volgt een andere graaf $G' = (V', A')$: neem voor $V'$ de knooppunten van $V$ en voeg daaraan knooppunt $n+1$ toe, en neem voor $A'$ de pijlen van $A$ en vervang daarin iedere pijl $(i,1)$ (de pijlen naar knooppunt 1) door een pijl $(i, n+1)$ (pijlen naar knooppunt $n+1$). Dit is een polynomiale constructie, namelijk $\mathcal{O}(n)$. Het is voldoende om te bewijzen dat $G$ een Hamiltonronde heeft d.e.s.d. als $G'$ een Hamiltonpad heeft. Stel $G$ heeft een Hamiltonronde, zeg $\{1 = i_1, i_2, \ldots, i_n, i_1 = 1\}$. Dan heeft $G'$ het Hamiltonpad $\{1 = i_1, i_2, \ldots, i_n, n+1\}$.
Omgekeerd, stel $G'$ heeft een Hamiltonpad. Omdat knooppunt 1 geen binnenkomende pijlen heeft moet het pad met knooppunt 1 beginnen; omdat knooppunt $n+1$ geen uitgaande pijlen heeft moet het pad met knooppunt $n+1$ eindigen. Het pad heeft dus de vorm $\{1 = j_1, j_2, \ldots, j_n, n+1\}$. Maar dan is er in $G$ een Hamiltonronde, namelijk $\{1 = j_1, j_2, \ldots, j_n, 1\}$.

Een probleem heet $\mathcal{NP}$-*moeilijk* als er een $Q \in \mathcal{NPC}$ bestaat met $Q \preceq P$, d.w.z. $P$ is minstens zo moeilijk als een moeilijkste probleem in $\mathcal{NP}$.

De meest bekende problemen uit de grafentheorie en de combinatorische optimalisering behoren tot $\mathcal{P}$ of $\mathcal{NPC}$. De volgende stelling[9] laat zien dat het erg waarschijnlijk is dat er nog andere problemen zijn.

**Stelling 1.4**
*Als $\mathcal{P} \neq \mathcal{NP}$, dan geldt $\mathcal{P} \cup \mathcal{NPC} \neq \mathcal{NP}$.*

[9] Voor het bewijs zie R.E. Ladner: *On the structure of polynomial time reducibility*, Journal of ACM 22 (1975) 155-171.

Zodra er een probleem gevonden wordt uit $\mathcal{NP} - (\mathcal{P} \cup \mathcal{NPC})$ is dus bewezen dat $\mathcal{P} \neq \mathcal{NP}$. Er wordt sinds het begin van de zeventiger jaren van de vorige eeuw intensief gezocht naar zo'n probleem. Het LP-probleem is een tijd kandidaat geweest totdat in 1979 werd bewezen [10] dat dit probleem tot $\mathcal{P}$ behoort. Op dit moment is van het Grafenisomorfieprobleem,[11] dat uiteraard tot $\mathcal{NP}$ behoort, nog onbekend in welke deelverz. van $\mathcal{NP}$ het zit.

Bij herkenningsproblemen (ja-nee problemen) kunnen we ook spreken over het *complement* van het probleem, en dat is de ontkenning van het oorspronkelijke probleem, d.w.z. een ja-antwoord van het oorspronkelijke probleem geeft een nee-antwoord van het complement en omgekeerd. Bij een probleem $P$ noteren we het complement met $P^c$. Merk op dat $(P^c)^c = P$.

Voor het Hamiltonkringprobleem luidt het complement:
Gegeven: Niet-gerichte graaf $G = (V, E)$.
Probleem: Heeft $G$ geen Hamiltonkring?
Het is niet duidelijk of voor het ja-antwoord van dit probleem, d.w.z. dat er geen Hamiltonkring is, een polynomiaal certificaat bestaat. Het antwoord is in feite alleen te controleren door alle permutaties van de knooppunten te bekijken (dit zijn er exponentieel veel) en na te gaan dat geen enkele permutatie een Hamiltonkring oplevert.

We definiëren de volgende verzamelingen:

$$\begin{aligned} co\text{-}\mathcal{P} &= \{P^c \mid P \in \mathcal{P}\}; \\ co\text{-}\mathcal{NP} &= \{P^c \mid P \in \mathcal{NP}\}; \\ co\text{-}\mathcal{NPC} &= \{P^c \mid P \in \mathcal{NPC}\}. \end{aligned}$$

**Stelling 1.5**
*(1)* $co\text{-}\mathcal{P} = \mathcal{P}$.

*(2) Als* $\mathcal{NPC} \cap co\text{-}\mathcal{NP} \neq \emptyset$, *dan geldt* $\mathcal{NP} = co\text{-}\mathcal{NP}$.

*(3) Als* $co\text{-}\mathcal{NP} \neq \mathcal{NP}$, *dan geldt* $\mathcal{P} \neq \mathcal{NP}$.

*(4) Als* $\mathcal{P} \neq \mathcal{NP}$, *dan geldt* $\mathcal{P} \neq co\text{-}\mathcal{NP}$.

**Bewijs**
(1) Voor een probleem $P \in \mathcal{P}$ bestaat een polynomiaal algoritme om na te gaan of het een ja-instantie of een nee-instantie is. $P^c$ behoort dus ook tot $\mathcal{P}$. Omdat $(P^c)^c = P$, geldt dat $P \in co\text{-}\mathcal{P}$.
Omgekeerd, als $P \in co\text{-}\mathcal{P}$, dan $P^c \in \mathcal{P}$. Omdat er nu voor het complement een polynomiaal algoritme bestaat om na te gaan of het een ja-instantie of een nee-instantie is, behoort $P = (P^c)^c$ tot $\mathcal{P}$.

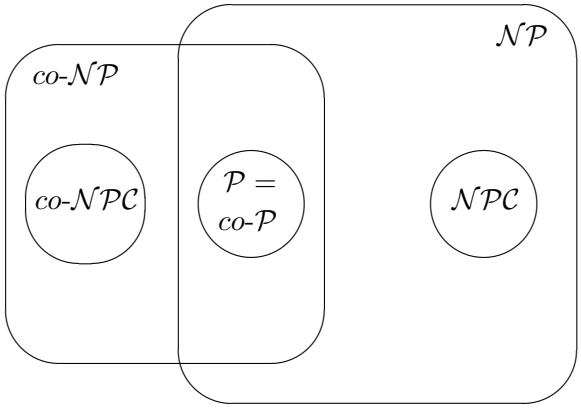
[10] L.G. Khachiyan (*A polynomial algorithm in linear programming*, Soviet Mathematics Doclady 20 (1979) 191-194) bewees dat het LP-probleem tot $\mathcal{P}$ behoort, maar zijn methode werkte in de praktijk toch onbevredigend. N. Karkarkar (*A new polynomial-time algorithm for linear programming*, Combinatorica 4 (1984) 373-395) publiceerde een ander polynomiaal algoritme dat ook in de praktijk wel snel is.

[11] Zie ook: http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK4/NODE180.HTM

(2) We bewijzen eerst dat $\mathcal{NP} \subseteq co\text{-}\mathcal{NP}$. Stel $P \in \mathcal{NP}$, dan is te bewijzen dat $P^c \in \mathcal{NP}$, d.w.z. dat een nee-instantie van $P$ polynomiaal controleerbaar is. Neem een nee-instantie $I$ van $P$. Laat $Q \in \mathcal{NPC} \cap co\text{-}\mathcal{NP} \neq \emptyset$. Dan is $I$ in polyniomiale tijd om te zetten in een nee-instantie $J$ van $Q$. Omdat $Q \in co\text{-}\mathcal{NP}$, is $J$ polynomiaal controleerbaar. Hieruit volgt dat ook $I$ polynomiaal controleerbaar is. Als $P \in co\text{-}\mathcal{NP}$, dan $P^c \in \mathcal{NP} \subseteq co\text{-}\mathcal{NP}$, zodat $P \in \mathcal{NP}$. Hiermee is bewezen dat $\mathcal{NP} = co\text{-}\mathcal{NP}$.
(3) Stel $\mathcal{P} = \mathcal{NP}$. Dan: $\mathcal{NP} = \mathcal{P} = co\text{-}\mathcal{P} = co\text{-}\mathcal{NP}$: tegenspraak.
(4) Stel $\mathcal{P} = co\text{-}\mathcal{NP}$. Dan: $\mathcal{P} = co\text{-}\mathcal{P} = co\text{-}(co\text{-}\mathcal{NP}) = \mathcal{NP}$: tegenspraak. □

**Gevolg 1.6**
Uit (2) volgt: als $co\text{-}\mathcal{NP} \neq \mathcal{NP}$, dan geldt $\mathcal{NPC} \cap co\text{-}\mathcal{NP} = \emptyset$, d.w.z. $\mathcal{NPC} \subseteq (\mathcal{NP} \backslash co\text{-}\mathcal{NP})$.
Door het complement van bovenstaande uitspraak te nemen krijgen we:
als $\mathcal{NP} \neq co\text{-}\mathcal{NP}$, dan geldt $co\text{-}\mathcal{NPC} \cap \mathcal{NP} = \emptyset$, d.w.z. $co\text{-}\mathcal{NPC} \subseteq (co\text{-}\mathcal{NP} \backslash \mathcal{NP})$.
Uit het voorbeeld van het complement van het Hamiltonkringprobleem lijkt het een redelijke aanname te veronderstellen dat $\mathcal{NP} \neq co\text{-}\mathcal{NP}$. Dit is echter (nog) niet bewezen. Ook is nog geen antwoord bekend op de vraag of $\mathcal{P} = \mathcal{NP} \cap co\text{-}\mathcal{NP}$.

Onder de aanname dat $\mathcal{NP} \neq co\text{-}\mathcal{NP}$ hebben de verschillende verzamelingen de hieronder schematisch weergegeven relaties.




**Vraag 1.4**
Toon aan dat het Grafenisomorfieprobleem tot $\mathcal{NP}$ behoort.

**Vraag 1.5**
Beschouw de volgende instantie van $SAT$:
$X = \{x_1, x_2, x_3, x_4\}$ en $\mathcal{C} = (x_1 \cup \overline{x}_3) \cap (\overline{x}_1 \cup x_4) \cap (x_2 \cup x_3 \cup \overline{x}_4) \cap (\overline{x}_1 \cup x_2 \cup \overline{x}_3 \cup x_4)$.
a. Is de zin $\mathcal{C}$ waar?
b. Construeer een zin $\mathcal{C}'$ van 3-$SAT$ die waar is d.e.s.d. als $\mathcal{C}$ waar is.

## 1.3 Opgaven

**Opgave 1.1**

Vergelijk in groeisnelheid de volgende functies:

a. $f(n) = n^2\sqrt{n}$ en $g(n) = 2n^2 \cdot ln\, n$;

b. $f(n) = 2^n$ en $g(n) = n!$

**Opgave 1.2**

Toon aan dat $n^{log_2\, n} = \mathcal{O}(2^n)$ en $n^{log_2\, n} \neq \Theta(2^n)$.

**Opgave 1.3**

Toon aan dat het Hamiltonrondeprobleem polynomiaal reduceerbaar is tot het Handelsreizigersprobleem.

**Opgave 1.4**

a. Toon aan dat voor de normale graaf $G = (V, E)$ (een graaf heet normaal als tussen ieder tweetal knooppunten 0 of 1 tak aanwezig is) de volgende beweringen equivalent zijn:

(1) $W \subseteq V$ is een knooppuntenbedekking;

(2) $V - W$ is een onafhankelijke verzameling;

(3) $V - W$ is een kliek in de complementaire graaf $\overline{G}$ ($\overline{G} = (\overline{V}, \overline{E})$ met $\overline{V} = V$ en $(v, w) \in \overline{E}$ d.e.s.d. als $(v, w) \notin E$).

b. Toon aan dat het knooppuntenbedekkingsprobleem, het onafhankelijkheidsprobleem en het kliekenprobleem via polynomiale reducties in elkaar zijn over te voeren.

# Hoofdstuk 2

# GEHEELTALLIGE LINEAIRE OPTIMALISERING

## 2.1 Model, formuleringen en voorbeelden

### 2.1.1 Model en formuleringen

Veel optimaliseringsproblemen hebben geheeltallige variabelen, bijvoorbeeld als het gaat om ondeelbare grootheden zoals een aantal personen. We spreken dan van *geheeltallige optimalisering*. Een geheeltallig optimaliseringsprobleem zullen we ook aanduiden met IP-probleem (*integer programming*). Een speciaal geval van een geheeltallige variabele is een variabele, waarvoor geldt dat deze slechts de waarde 0 of 1 kan aannemen: we spreken dan van een *(0,1)-variabele* of *binaire variabele* en het optimaliseringsprobleem waarin de geheeltallige variabelen (0,1)-variabelen zijn, heet een *combinatorisch optimaliseringsprobleem* ook wel aangeduid als CO-probleem. We spreken van *gemengd geheeltallige optimalisering* als sommige variabelen wel en andere niet geheeltallig zijn.

In tegenstelling tot de methoden voor lineaire optimalisering, die meestal efficiënt werken en snel tot een optimale oplossing leiden, bestaat er voor de geheeltallige optimalisering geen efficiënte methode. Het afronden van de bijbehorende LP-oplossing kan leiden tot een ontoelaatbare of een niet-optimaal punt. Het is zelfs uiterst onwaarschijnlijk dat er ooit een efficiënte methode voor gevonden kan worden. De reden hiervoor is dat het geheeltallige programmeringsprobleem tot de complexiteitsklasse $\mathcal{NPC}$ behoort.[1]

Het geheeltallige lineaire programmeringsprobleem is geformuleerd als:

$$max\{p^T x \mid Ax \leq b\,;\ x \geq 0 \text{ en geheel}\}, \tag{2.1.1}$$

waarbij $p, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ en $A$ een $m \times n$-matrix is.

[1] Voor een definitie van de complexiteitsklasse $\mathcal{NPC}$ zie Besliskunde 1.

Een combinatorisch optimaliseringsprobleem schrijven we als

$$max\{p^T x \mid Ax \leq b\,;\ x \in \{0,1\}^n\}, \tag{2.1.2}$$

waarbij $x \in \{0,1\}^n$ betekent dat iedere component $x_j \in \{0,1\},\ 1 \leq j \leq n$.

De volgende stelling laat zien dat het combinatorisch optimaliseringsprobleem, en daarmee ook het IP-probleem tot de complexiteitsklasse $\mathcal{NPC}$ behoort.

**Stelling 2.1**
*Het combinatorisch optimaliseringsprobleem behoort tot de complexiteitsklasse $\mathcal{NPC}$.*

**Bewijs**
Het is direct duidelijk dat het combinatorisch optimaliseringsprobleem in $\mathcal{NP}$ zit (als certificaat nemen we de gegeven oplossing en het nagaan van de toelaatbaarheid en of de doelfunctie minstens een gegeven waarde heeft, is polynomiaal).
Beschouw het CO-probleem in de vorm: is er een $x \in \{0,1\}^n$ met $Ax \leq b$?
Het is nu voldoende om te laten zien dat $SAT \preceq$ het combinatorisch optimaliseringsprobleem.[2]
Neem een 'zin', d.w.z. een conjunctie van een aantal 'woorden', waarbij een woord de conjunctie is van een aantal Boolse variabelen of hun ontkenning (de ontkenning van $x$ noteren we met $\overline{x}$). Laten $x_1, x_2, \ldots, x_n$ de Boolse variabelen zijn ($x_j = 1$ als de $j$-de Boolse variabele waar is en anders 0) en beschouw de zin

$$\mathcal{C} = C_1 \cap C_2 \cap \cdots \cap C_m \text{ met } C_i = \cup_{j=1}^{p_i} x_{i_j} \cup_{j=p_i+1}^{n_i} \overline{x}_{i_j}.$$

Ieder woord moet waar zijn, dus minstens een van de (0,1)-getallen $x_{i_j},\ 1 \leq j \leq p_i$, en $1 - x_{i_j},\ p_i + 1 \leq j \leq n_i$, moet 1 zijn, d.w.z. $\sum_{j=1}^{p_i} x_{i_j} + \sum_{j=p_i+1}^{n_i} (1 - x_{i_j}) \geq 1,\ 1 \leq i \leq m$, ofwel

$$-\sum_{j=1}^{p_i} x_{i_j} + \sum_{j=p_i+1}^{n_i} x_{i_j} \ \leq\ n_i - p_i - 1,\ 1 \leq i \leq m. \tag{2.1.3}$$

Neem $a_{ii_j} = \begin{cases} -1, & 1 \leq j \leq p_i \\ +1, & p_i + 1 \leq j \leq n_i \\ 0, & \text{anders} \end{cases}$ en $b_i = n_i - p_i - 1,\ 1 \leq i \leq m.$

Dan is (2.1.3) equivalent met

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i,\ 1 \leq i \leq m \ \text{ ofwel in vectornotatie } \ Ax \leq b. \tag{2.1.4}$$

Hiermee is aangetoond dat $SAT \preceq$ het combinatorisch optimaliseringsprobleem. □

[2] Voor een definitie van de complexiteitsklasse $SAT$ zie Besliskunde 1.

**Vraag 2.1**
a. Geef een voorbeeld van een IP-probleem, waarvan de afronding van optimale oplossing van het bijbehorende LP-probleem een ontoelaatbare oplossing geeft.
b. Geef een voorbeeld van een IP-probleem, waarvan de afronding van optimale oplossing van het bijbehorende LP-probleem een toelaatbare, maar niet optimale oplossing oplevert.

### 2.1.2 Voorbeelden

In deze paragraaf bespreken we een aantal problemen die als geheeltallig programmeringsprobleem gedefinieerd kunnen worden.

**Voorbeeld 2.1** *Vaste kosten*
Beschouw een productieprobleem met $n$ producten en waarbij de productiekosten deels vaste kosten zijn en deels kosten lineair in de hoeveelheid:

$$C_j(x_j) = \begin{cases} K_j + c_j x_j & \text{als } x_j > 0; \\ 0 & \text{als } x_j = 0. \end{cases}$$

De doelfunctie wordt dan $\sum_{j=1}^{n} C_j(x_j)$ en is niet meer lineair in $x_j$ vanwege de discontinuïteit in $x_j = 0$. Deze functie is toch lineair te maken door voor iedere $x_j$ een binaire variabele $y_j$ in te voeren met de interpretatie dat $y_j = 1$ als $x_j > 0$. Door nu te eisen dat $0 \leq x_j \leq M y_j$, met $M$ een groot getal dat in alle realitische situaties een bovengrens voor de productie is, wordt $C_j(x_j) = K_j y_j + c_j x_j$, zodat de doelfunctie lineaire is. Het oorspronkelijke probleem heeft niets geheeltalligs, maar door deze modellering wordt het een gemengd geheeltallig optimaliseringsprobleem.

**Voorbeeld 2.2** *Stuksgewijs lineaire doelfunctie*
Veronderstel dat $a_1 < a_2 < \cdots < a_k$ en dat we een continue stuksgewijze lineaire doelfunctie $f(x)$ hebben, gedefinieerd op het interval $[a_1, a_k]$ en die door de punten $(a_i, f(a_i)),\ 1 \leq i \leq k$ gaat. Iedere $x \in [a_1, a_k]$ is op een unieke manier te schrijven als convexe combinatie van $a_1, a_2, \ldots, a_k$, d.w.z.

$$x = \textstyle\sum_{i=1}^{k} \lambda_i a_i \text{ met } \lambda_i \geq 0 \text{ voor alle } i \text{ en } \sum_{i=1}^{k} \lambda_i = 1,$$

en waarbij (hoogstens) twee $\lambda$'s positief zijn die dan opeenvolgend moeten zijn, zeg

$$x = \lambda_i a_i + \lambda_{i+1} a_{i+1}.$$

Omdat de functie op het interval $[a_i, a_{i+1}]$ lineair is, is dan ook $f(x) = \lambda_i f(a_i) + \lambda_{i+1} f(a_{i+1})$.
Om te modelleren dat hoogstens twee $\lambda$'s positief zijn die dan opeenvolgend moeten zijn, introduceren we binaire variabelen $y_i,\ 1 \leq i \leq k-1$ met de interpretatie dat als $y_i = 1$ dan mogen alleen $\lambda_i$ en $\lambda_{i+1}$ positief zijn. Door te eisen dat $\sum_{i=1}^{k-1} y_i = 1$ wordt bereikt dat slechts eenmaal twee opeenvolgende $\lambda$'s positief kunnen zijn. Door verder te eisen dat

$$\lambda_1 \leq y_1;\ \lambda_i \leq y_{i-1} + y_i,\ 2 \leq i \leq k-1 \text{ en } \lambda_k \leq y_{k-1},$$

wordt bereikt dat als $y_j$ de enige $y$-variabele is die positief is, dan is $\lambda_i = 0$ voor $i \neq j, j+1$.
Hiermee is de doelfunctie te beschrijven als functie van $\lambda_1, \lambda_2, \ldots, \lambda_k$ en $y_1, y_2, \ldots, y_{k-1}$:

$\sum_{i=1}^{k} \lambda_i f(a_i);\ \sum_{i=1}^{k} \lambda_i = 1;\ \lambda_i \geq 0,\ 1 \leq i \leq k;$

$\lambda_1 \leq y_1;\ \lambda_i \leq y_{i-1} + y_i,\ 2 \leq i \leq k-1;\ \lambda_k \leq y_{k-1};\ \sum_{i=1}^{k-1} y_i = 1;;\ y_i \in \{0,1\},\ 1 \leq i \leq k-1.$

**Voorbeeld 2.3** *Scheduling probleem*
Beschouw een scheduling probleem waarbij $n$ verschillende operaties op één machine moeten worden uitgevoerd. Veronderstel dat er drie soorten beperkingen zijn:
(1) volgorde-restricties voor een aantal taken;
(2) niet-tegelijkertijd kunnen uitvoeren van twee taken op deze machine;
(3) voor iedere taak $j$ een tijdstip $d_j$: het tijdstip waarop men deze taak af wil hebben.
De vraagstelling luidt om een planning te maken die de maximale overschrijding van de tijden $d_j$ minimaliseert, rekening houdend met de overige beperkingen.
Laat $x_j$ de beslissingsvariabele zijn voor het tijdstip waarop de $j$-de taak op de machine begint en laat $t_j$ de (gegeven) tijdsduur zijn voor de bewerking van taak $j$ op de machine, $1 \leq j \leq n$.
De volgorde-restrictie taak $i$ komt voor taak $j$ houdt in dat $x_i + t_i \leq x_j$.
De taken $i$ en $j$ niet tegelijkertijd op de machine betekent:

$$\text{òfwel } x_i + t_i \leq x_j, \text{ òfwel } x_j + t_j \leq x_i.$$

Deze eis is met een binaire variabele $y_{ij}$, waarbij $y_{ij} = 1$ correspondeert met de eerste mogelijkheid en $y_{ij} = 0$ met de tweede, om te vormen tot twee beperkingen die beide moeten gelden:

$$x_i + t_i \leq x_j + M(1 - y_{ij}) \text{ en } x_j + t_j \leq x_i + M y_{ij},$$

waarbij $M$ voldoende groot is, bijv. het laatste tijdstip waarop een taak af moet zijn.
Het minimaliseren van de maximale overschrijdingen is te modelleren door

$$T \text{ te minimaliseren en te eisen dat } T \geq x_j + t_j - d_j \text{ voor alle } j.$$

**Voorbeeld 2.4** *Locatie probleem*
Veronderstel dat er $m$ mogelijke plaatsen zijn om filialen van een bedrijf te vestigen. Het vestigen van het $i$-de filiaal kost $f_i$ en filiaal $i$ heeft capaciteit $b_i$, $1 \leq i \leq m$. Het bedrijf heeft $n$ klanten en de $j$-de klant heeft een hoeveelheid $d_j$ nodig, $1 \leq j \leq n$. Als er vanuit filiaal $i$ aan klant $j$ wordt geleverd, dan kost dit $c_{ij}$ per eenheid.
Welke filialen moeten worden gevestigd en hoe moet de bevoorrading van de klanten worden uitgevoerd om de totale kosten te minimaliseren?

$$\text{Zij } \begin{cases} x_{ij} & = \text{ de hoeveelheid die vanuit filiaal } i \text{ aan klant } j \text{ wordt geleverd} \\ y_i & = \begin{cases} 1 & \text{als filiaal } i \text{ wordt gevestigd} \\ 0 & \text{anders} \end{cases} \end{cases}$$

Dit probleem is als volgt te formuleren als een gemengd geheeltallig programmeringsprobleem:

$$min \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i \;\middle|\; \begin{array}{l} \sum_{i=1}^{m} x_{ij} = d_j,\ j = 1, 2, \ldots, n \\ \sum_{j=1}^{n} x_{ij} \leq b_i y_i,\ i = 1, 2, \ldots, m \\ x_{ij} \geq 0,\ 1 \leq i \leq m,\ 1 \leq j \leq n \\ y_i \in \{0, 1\},\ 1 \leq i \leq m \end{array} \right\}.$$

**Vraag 2.2**

Beschouw het gebied in de $\mathbb{R}^2$ met $0 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 10$ met als extra voorwaarden:
als $x_1 > 5$, dan is $x_2 \leq 5$ en als $x_2 > 5$, dan is $x_1 \leq 5$. Geef m.b.v. continue en geheeltallige variabelen een lineaire beschrijving van dit niet-convexe gebied.

**Vraag 2.3**

Veronderstel dat we $n$ voorwerpen hebben met gegeven volumes $a_i \in (0, 1],\ 1 \leq i \leq n$. Deze voorwerpen moeten in dozen worden verpakt. De dozen zijn alle identiek met volume 1 en er zijn voldoende dozen aanwezig, zeg $n$ stuks. Gevraagd wordt hoe de voorwerpen in de dozen te verpakken zdd. zo min mogelijk dozen worden gebruikt.
Formuleer dit probleem als een combinatorisch programmeringsprobleem.

**Vraag 2.4**

Zij $G = (V, E)$ een niet-gerichte graaf met $n$ knooppunten.
Beschouw het probleem om de knooppunten van de graaf met zo min mogelijk kleuren te kleuren zdd. knooppunten die aan elkaar grenzen een verschillende kleur krijgen.
Formuleer dit probleem als een combinatorisch programmeringsprobleem.

### 2.1.3 Opgaven

**Opgave 2.1**

Veronderstel dat er $m$ (identieke) machines zijn waarop taken uitgevoerd kunnen worden. Er zijn $n$ taken en taak $i$ heeft een (geheeltallige) bewerkingstijd $p_i,\ 1 \leq i \leq n$. Een taak hoeft maar op één machine te worden uitgevoerd en een machine kan slechts één taak tegelijk aan.
Gevraagd is een planning te vinden zdd. al het werk zo vroeg mogelijk af is.
Formuleer dit probleem als een combinatorisch optimaliseringsprobleem.

**Opgave 2.2**

Beschouw de volgende situatie: $n$ verschillende gebouwen moeten worden neergezet op $n$ gegeven locaties. De vraag luidt: welk gebouw komt op welke locatie. Het is bekend hoeveel mensen per

dag van gebouw $j$ naar gebouw $l$ gaan, zeg $b_{jl}$. Het is ook bekend wat de afstand tussen lokatie $i$ en lokatie $k$ is, zeg $a_{ik}$.
Waar moeten de gebouwen worden geplaatst opdat de totale afstand die de mensen tezamen per dag moeten afleggen minimaal is?
Formuleer dit probleem als een combinatorisch (niet-lineair) optimaliseringsprobleem.

**Opgave 2.3**
Veronderstel dat er $n$ plaatsen zijn. Iedere plaats heeft een aanbod of een vraag: plaats $i$ heeft een 'netto productie' (aanbod/vraag) $p_i$, waarbij $\sum_{i=1}^{n} p_i = 0$ ($p_i \geq 0$ is een aanbod $p_i$ en $p_i < 0$ een vraag $-p_i$).
Als over de verbinding $(i, j)$ iets wordt gestuurd, dan zijn er vaste kosten $f_{ij}$ en variabele kosten van $c_{ij}$ per eenheid van vervoer; bovendien heeft zo'n verbinding een capaciteit van $b_{ij}$ (d.w.z. dat er hoogstens $b_{ij}$ over vervoerd kan worden).
Geef een formulering voor het probleem om een zo goedkoop mogelijk vervoersschema op te stellen.

**Opgave 2.4**
Beschouw het volgende vestigingsprobleem. Een aantal winkels, zeg $n$, moet vanuit een aantal magazijnen bevoorraad worden. Winkel $j$ heeft $d_j$ eenheden nodig. Er zijn $m$ mogelijke vestigingsplaatsen voor de magazijnen. In vestigingsplaats $i$ zijn vaste investeringskosten $f_i$ en is de productiecapaciteit $b_i$. Bij de bevoorrading van winkel $j$ vanuit een magazijn in plaats $i$ zijn er vaste kosten $f_{ij}$ en variabele kosten $c_{ij}$ per eenheid; bovendien is de capaciteit op deze route $b_{ij}$.
De vraag luidt: waar kunnen deze magazijnen het beste gevestigd worden opdat de totale kosten minimaal zijn?
Formuleer dit probleeem als een optimalseringsprobleem met continue en geheeltallige variabelen.

## 2.2 Branch-and-Bound

### 2.2.1 Het generieke algoritme

De branch-and-bound methode is een algemene techniek,[3] waarmee onder andere een geheeltallig optimaliseringsprobleem opgelost kan worden. Het oorspronkelijke probleem wordt hierbij opgesplitst in *deelproblemen*, zeg $P_k, 1 \leq k \leq K$, die tezamen equivalent zijn met het oorspronkelijke probleem $P_0$. Beschouw het zuivere geheeltallige lineaire optimaliseringsprobleem (2.1.1) en laat

$$S = \{x \mid Ax \leq b\,;\ x \geq 0 \text{ en geheel}\}. \tag{2.2.1}$$

[3]Deze techniek is voor het eerst voorgesteld in A.H. Land and A.G. Doig, *An automatic method of solving discrete programming problems*, Econometrica 28 (1960) 497–520. De methode is voor het eerst toegepast op het handelsreizigersprobleem door J.D.C. Little, K.G. Murty, D.W. Sweeny and C. Karel, *An algorithm for the traveling salesmanproblem*, Operations Research 11 (1963) 972–989.

Het basisidee van de branch-and-bound methode is om $S$ op te splitsen in deelverz. $S_1, S_2, \ldots, S_K$ en laat

$$P_k: \ max\{p^T x \mid x \in S_k\}, \ 1 \leq k \leq K. \tag{2.2.2}$$

De deelproblemen worden òfwel exact opgelost, òfwel benaderd om een bovengrens van het optimum te vinden. De argumentatie voor de branch-and-bound methode is dat als een bovengrens van een deelprobleem niet groter is dan de waarde van een reeds gevonden toelaatbare oplossing, dan kan dit deelprobleem verder buiten beschouwing worden gelaten.
Bovengrenzen kunnen worden verkregen door een *relaxatie* van het probleem te nemen. Meestal wordt hiervoor de *LP-relaxatie* genomen, d.w.z. dat de eis '$x$ geheel' wordt weggelaten. Omdat het optimum van het IP-probleem geheel moet zijn (we nemen aan dat de coëfficiënten in de doelfunctie geheel zijn), kunnen we het optimum van de LP-relaxatie naar beneden afronden.
Laat $L$ de lijst zijn van de deelproblemen $P_k$, en laat $\overline{z}_k$ de bovengrens zijn van deelprobleem $P_k, \ 1 \leq k \leq K$. Noteer met $\underline{z}$ de waarde van de best bekende toelaatbare oplossing.
Het generieke branch-and-bound algoritme ziet er dan als volgt uit.

**Algoritme 2.1** *Branch-and-bound methode*

1. *Initialisatie*
   $L = \{P_0\}, \ \overline{z}_0 = +\infty, \ \underline{z} = -\infty, \ t = 0.$

2. *Stopcriterium*
   Als $L = \emptyset$, dan:
   (a) Als $t = 0$, dan is het probleem ontoelaatbaar.
   (b) Als $t = 1$, dan is $x^*$ de optimale oplossing met optimum $\underline{z}$.

3. *Keuze deelprobleem*
   Kies een element $P_k$ van $L$, verwijder dit uit $L$ en kies een relaxatie voor $P_k$.

4. *Oplossen relaxatie*
   Los de relaxatie van $P_k$ op:
   (a) Als $P_k$ ontoelaatbaar is, dan is $\overline{z}_k = -\infty$.
   (b) Als $P_k$ een oneindige oplossing heeft, dan is $\overline{z}_k = +\infty$.
   (c) Als $P_k$ een eindige oplossing $x^k$ heeft, dan is $\overline{z}_k = \lfloor p^T x^k \rfloor$.

5. *Afhandelen deelprobleem en 'snoeien'*
   Als $\overline{z}_k \leq \underline{z}$: ga naar stap 2.
   Als $\overline{z}_k > \underline{z}$ en $x^k$ geheeltallig en toelaatbaar is:
   (a) $x^* := x^k, \ \underline{z} := \overline{z}_k$ en $t := 1$.
   (b) Verwijder alle problemen $P_i$ met $\overline{z}_i \leq \underline{z}$ uit $L$.
   (c) Ga naar stap 2.

6. *Vertakken*
   (a) Splits $P_k$ op in $P_{k_1}, P_{k_2}, \ldots, P_{k_p}$.

(b) Voeg de nieuwe deelproblemen aan $L$ toe met bovengrens $\overline{z}_k$.

(c) Ga naar stap 2.

### 2.2.2 Behandeling van en opsplitsing in deelproblemen

*Afhandelen deelprobleem*
In stap 5 van het algoritme kan het in stap 2 gekozen deelprobleem $P_k$, waarvan de relaxatie in stap 3 is opgelost, in de volgende gevallen worden afgehandeld:
(1) als de relaxatie een ontoelaatbaar probleem is: $P_k$ zelf is dan ook ontoelaatbaar;
(2) als de bovengrens $\overline{z}_k$ niet groter is dan de waarde $\underline{z}$ van een toegelaten oplossing van $P_0$: het deelprobleem heeft dan geen betere oplossing;
(3) als de relaxatie een betere toegelaten geheeltallige oplossing heeft: dit is dan een kandidaat voor de optimale oplossing.
In de andere gevallen kan $P_k$ niet als afgehandeld worden beschouwd en splitsen we $P_k$ in deelproblemen.

*Keuze deelprobleem*
In stap 3 van het algoritme wordt een van de deelproblemen van lijst $L$ gekozen. Deze keuze is van invloed op het kunnen vinden van een betere toelaatbare oplossing, de kans om een deelprobleem te kunnen afhandelen en het totaal aantal deelproblemen dat moet worden onderzocht voordat we kunnen stoppen met de optimale oplossing.
Er zijn verschilende strategieën mogelijk voor de keuze van het deelprobleem:

(1) Het *laatste deelprobleem* dat is gecreëerd: dit heeft het voordeel dat het meestal direct beschikbaar is, want zo'n LP-relaxatie komt uit het vorige LP-probleem met daaraan toegevoegd één nieuwe beperking; dit houdt in dat de duale oplossing toelaatbaar blijft. Hiervoor is de *duale simplex methode* zeer geschikt en vaak is slechts één extra iteratiestap voldoende om de nieuwe optimale oplossing te vinden.

(2) Het deelprobleem met de *hoogste bovengrens*: we hopen hiermee een betere toelaatbare oplossing te vinden, waarna hopelijk een aantal andere deelproblemen niet meer onderzocht hoeft te worden omdat hun bovengrens niet beter is dan de waarde van een toegelaten oplossing die met het gekozen deelprobleem is gevonden.

(3) Het deelprobleem met de *grootste niet-geheeltalligheid*, waarbij de niet-geheeltalligheid $s$ is gedefinieerd door $s = \sum_j min(f_j, 1 - f_j)$ met $f_j = x_j - \lfloor x_j \rfloor$, d.w.z. de *fractie*, waarbij $x$ de oplossing is van de relaxatie van het deelprobleem: we hopen hiermee dat het deelprobleem kan worden afgehandeld, omdat de geheeltalligheidseis de bovengrens zover verlaagt dat het deelprobleem niet meer beschouwd hoeft te worden.

(4) Het deelprobleem met de *kleinste niet-geheeltalligheid*: we hopen hiermee een betere toelaatbare oplossing te vinden, waarna hopelijk een aantal andere deelproblemen niet meer onderzocht hoeft te worden.

*Vertakken (branching)*
Het opsplitsen in nieuwe deelproblemen gebeurt meestal via een *splitsingsvariabele*. Vaak is dit een variabele die in de oplossing van de relaxatie niet-geheeltallig is: zeg $x_i$ met waarde $b_i^*$. We beschouwen dan twee deelproblemen door enerzijds te eisen $x_i \leq \lfloor b_i^* \rfloor$ en anderzijds $x_i \geq \lceil b_i^* \rceil$.

Voor de keuze welke variabele als splitsingsvariabele wordt genomen, zijn weer diverse strategieën:
(1) De *grootste fractie regel:* kies als splitsingsvariabele de variabele $x_i$ met $min(f_i, 1-f_i)$ maximaal; door de variabele met de grootste fractie te kiezen hopen we dat de grenzen van de bijbehorende deelproblemen zo laag worden dat ze daardoor kunnen worden afgehandeld.
(2) De *kleinste fractie regel:* kies als splitsingsvariabele de variabele $x_i$ met $min(f_i, 1-f_i)$ minimaal; door de variabele met de kleinste fractie te kiezen hopen we in de bijbehorende deelproblemen een goede oplossing te vinden.
(3) De *grootste coëfficiënt regel:* kies als splitsingsvariabele de variabele $x_i$ die in de doelfunctie de grootste coëfficiënt heeft; hiermee beogen we hetzelfde als met de grootste fractie regel.
(4) De *kleinste coëfficiënt regel:* kies als splitsingsvariabele de variabele $x_i$ met in de doelfunctie de kleinste coëfficiënt. We hopen nu hetzelfde te bereiken als met de kleinste fractie regel.
Het lijkt verstandig de regels (2) en (4) in het begin van het algoritme toe te passen en (1) en (3) later. We zullen de methode nader toelichten aan de hand van het volgende voorbeeld. Als keuze van het deelprobleem nemen we het deelprobleem met de grootste bovengrens en als splitsingsvariabele de variabele met de grootste fractie.

**Voorbeeld 2.5**

$$P_0 \ : \ max \left\{ 5x_1 + 8x_2 \ \middle| \ \begin{array}{rcrcl} x_1 & + & x_2 & \leq & 6 \\ 5x_1 & + & 9x_2 & \leq & 45 \\ \multicolumn{5}{c}{x_1, x_2 \geq 0 \text{ en geheel}} \end{array} \right\}.$$

*Initialisatie:*
$L = \{P_0\};\ \overline{z}_0 = +\infty;\ \underline{z} = -\infty;\ t = 0.$

*Iteratie 1:*
Kies $P_0$; $L = \emptyset$; los de LP-relaxatie op: $x_1^0 = \frac{9}{4},\ x_2^0 = \frac{15}{4},\ \overline{z}_0 = \lfloor \frac{165}{4} \rfloor = 41.$
Kies als splitsingsvariabele $x_2$ en beschouw de deelproblemen:
$P_1 : P_0$ met toegevoegd $x_2 \leq 3;\ P_2 : P_0$ met toegevoegd $x_2 \geq 4;\ \overline{z}_1 = \overline{z}_2 = 41;\ L = \{P_1, P_2\}.$

*Iteratie 2:*
Kies deelprobleem $P_1$; $L = \{P_2\}$; los de LP-relaxatie op: $x_1^1 = 3,\ x_2^1 = 3,\ \overline{z}_1 = \lfloor 39 \rfloor = 39.$
$x_1^* = 3,\ x_2^* = 3;\ \underline{z} = 39$ en $t = 1.$

*Iteratie 3:*
Kies deelprobleem $P_2$; $L = \emptyset$; los de LP-relaxatie op: $x_1^2 = \frac{9}{5},\ x_2^2 = 4, \overline{z}_2 = \lfloor 41 \rfloor = 41.$
Kies als splitsingsvariabele $x_1$ en beschouw de deelproblemen:
$P_3 : P_2$ met $x_1 \leq 1;\ P_4 : P_2$ met $x_1 \geq 2;\ \overline{z}_3 = \overline{z}_4 = 41;\ L = \{P_3, P_4\}.$

*Iteratie 4:*

Kies deelprobleem $P_3$; $L = \{P_4\}$; los de LP-relaxatie op: $x_1^3 = 1,\ x_2^3 = \frac{40}{9},\ \overline{z}_3 = \lfloor \frac{365}{9} \rfloor = 40$.
Kies als splitsingsvariabele $x_2$ en beschouw de deelproblemen:
$P_5 : P_3$ met $x_2 \leq 4$; $P_6 : P_3$ met $x_1 \geq 5$; $\overline{z}_5 = \overline{z}_6 = 40$; $L = \{P_4, P_5, P_6\}$.

*Iteratie 5:*

Kies deelprobleem $P_4$; $L = \{P_5, P_6\}$; los de LP-relaxatie op: $P_4$ is ontoelaatbaar, dus $\overline{z}_4 = -\infty$.

*Iteratie 6:*

Kies deelprobleem $P_5$; $L = \{P_6\}$; los de LP-relaxatie op: $x_1^5 = 1,\ x_2^5 = 4$; $\overline{z}_5 = \lfloor 37 \rfloor = 37$ (geen verbetering van de best bekende oplossing).

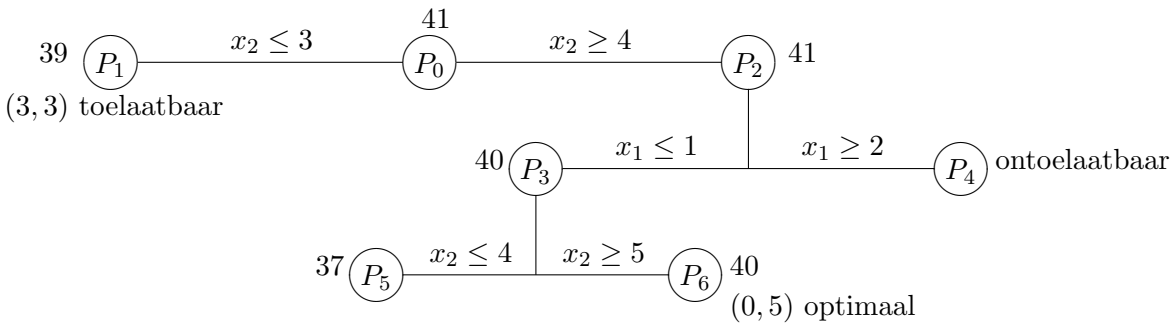
*Iteratie 7:*

Kies deelprobleem $P_6$; $L = \emptyset$; los de LP-relaxatie op: $x_1^6 = 0,\ x_2^6 = 5,\ \overline{z}_6 = \lfloor 40 \rfloor = 40$.
$x_1^* = 0,\ x_2^* = 5$; $\underline{z} = 40$ en $t = 1$.

*Iteratie 8:*

Omdat $L = \emptyset$ zijn we klaar en is de optimale oplossing: $x_1^* = 0,\ x_2^* = 5$ met waarde 40.

Schematisch kan het bovenstaande als volgt worden weergegegeven.




Opmerkingen:

1. Als de variabelen begrensd zijn, zeg $0 \leq x_j \leq u_j,\ 1 \leq j \leq n$, dan is de branch-and-bound methode eindig. De branch-and-bound methode is te representeren via een binaire boom, waarvan de knooppunten corresponderen met de deelproblemen (zie de tekening van bovenstaand voorbeeld). Als een deelprobleem wordt geplitst via splitsingsvariabele $x_j$, dan krijgt het knooppunt twee zonen met disjuncte deelverz. voor de mogelijke waarden $\{0, 1, \ldots, u_j\}$ van $x_j$. Dit betekent dat zo'n boom ten hoogste de diepte van $\sum_{j=1}^{n} u_j$ heeft, d.w.z. dat de methode eindig is.
2. De branch-and-bound techniek kan ook worden toegepast op een gemengd geheeltallig programmeringsprobleem. Uiteraard komen dan alleen de geheeltallige variabelen in aanmerking om splitsingsvariabele te worden en kan het LP-optimum niet naar beneden worden afgerond.

**Vraag 2.5**

Los het volgende IP-probleem op met de branch-and-bound methode. Kies het deelprobleem met de hoogste bovengrens en kies als splitsingsvariabele de variabele met de grootste fractie.

$$max \left\{ 13x_1 + 8x_2 \;\middle|\; \begin{array}{rcl} x_1 + 2x_2 & \leq & 10 \\ 5x_1 + 2x_2 & \leq & 20 \\ x_1, x_2 \geq 0 \text{ en geheel} & & \end{array} \right\}.$$

### 2.2.3 Opgaven

**Opgave 2.5**

Los het volgende IP-probleem met de branch-and-bound op. Kies het deelprobleem met de grootste niet-geheeltalligheid en kies de splitsingsvariabele volgens de grootste coëfficiënt regel.

$$max \left\{ 4x_1 - x_2 \;\middle|\; \begin{array}{rcl} 7x_1 - 2x_2 & \leq & 14 \\ 2x_1 - x_2 & \leq & 3 \\ x_2 & \leq & 3 \\ x_1, x_2 \geq 0 \text{ en geheel} & & \end{array} \right\}.$$

**Opgave 2.6**

Los het volgende IP-probleem op met de branch-and-bound methode. Kies het deelprobleem met de hoogste bovengrens en kies als splitsingsvariabele de variabele met de grootste fractie.

$$max \left\{ x_1 + x_2 \;\middle|\; \begin{array}{rcl} 4x_1 + 3x_2 & \leq & 12 \\ -4x_1 + 3x_2 & \leq & 0 \\ -x_2 & \leq & -1 \\ x_1, x_2 \geq 0 \text{ en geheel} & & \end{array} \right\}.$$

**Opgave 2.7**

Beschouw het volgende probleem:

$$max \left\{ x_1 \;\middle|\; \begin{array}{c} 2x_1 + 2x_2 + \cdots + 2x_n = n \\ x_j \in \{0,1\},\ 1 \leq j \leq n \end{array} \right\}$$

met $n$ oneven, zodat dit probleem ontoelaatbaar is.

a. Zij $J$ een deelverz. van $\{1, 2, \ldots, n\}$ met hoogstens $\frac{n-1}{2}$ elementen.
Geef iedere $x_j,\ j \in J$ een waarde 0 of 1 en bewijs dat de LP-relaxatie (nadat deze variabelen uit $J$ gefixeerd zijn op 0 of 1), d.w.z. het probleem

$$max \left\{ x_1 \;\middle|\; \begin{array}{l} \sum_{j \notin J} 2x_j = n - \sum_{j \in J} 2x_j \\ 0 \leq x_j \leq 1,\ j \notin J \end{array} \right\}$$

altijd toelaatbaar is.

b. Bewijs dat iedere branch-and-bound procedure om het oorspronkelijke probleem op te lossen minstens $2^{(n+3)/2} - 1$ deelproblemen moet onderzoeken.

## 2.3 Sneden

Als de LP-relaxatie een geheeltallige oplossing heeft, dan is deze ook optimaal voor het geheeltallige probleem. Als dit niet het geval is dan kunnen we aan de LP-formulering een ongelijkheid toevoegen waaraan iedere geheeltallige toegelaten oplossing voldoet, maar waaraan het huidige LP-optimum niet voldoet. Zo'n ongelijkheid heet een *snede*.

### 2.3.1 Gomory's fractie-snede algoritme

Beschouw probleem (2.1.1) en los de LP-relaxatie ervan op met de simplex methode. Veronderstel dat het optimale LP-tableau de volgende gedaante heeft:

$$(x_B)_i = a^*_{i0} - \sum_j a^*_{ij}(x_N)_j, \ i = 0, 1, \ldots, m, \tag{2.3.1}$$

waarbij de 0-de rij correspondeert met de doelfunctie, en $x_B \cup x_N$ de oorspronkelijke plus de verschil- en schijnvariabelen bevat. In verband met de optimaliteit geldt $a^*_{0j} \geq 0$ voor alle $j$. Als de LP-relaxatie geen geheeltallige oplossing geeft, is er een rij, zeg rij $r$, met $a^*_{r0}$ niet geheel. Uit (2.3.1) volgt dat iedere geheeltalige oplossing voldoet aan

$$(x_B)_r + \sum_j \lfloor a^*_{rj} \rfloor (x_N)_j \leq a^*_{r0}, \tag{2.3.2}$$

waarbij $\lfloor x \rfloor$ het grootste gehele getal $\leq x$ is. Omdat voor iedere geheeltallige oplossing het linkerlid van (2.3.2) geheeltallig is, moet ook voor iedere geheeltallige oplossing gelden dat

$$(x_B)_r + \sum_j \lfloor a^*_{rj} \rfloor (x_N)_j \leq \lfloor a^*_{r0} \rfloor. \tag{2.3.3}$$

Vullen we in (2.3.3) voor $(x_B)_r$ de uitdrukking uit (2.3.1) in, dan geeft dat de ongelijkheid

$$\sum_j \{a^*_{rj} - \lfloor a^*_{rj} \rfloor\}(x_N)_j \geq a^*_{r0} - \lfloor a^*_{r0} \rfloor. \tag{2.3.4}$$

Noteer de fracties $a^*_{rj} - \lfloor a^*_{rj} \rfloor$ met $f_{rj}$ voor alle $j$, dan geeft dit de zogenaamde *fractie-snede*

$$\sum_j f_{rj}(x_N)_j \geq f_{r0}. \tag{2.3.5}$$

Omdat $f_{r0} > 0$ voldoet de optimale oplossing van de LP-relaxatie, d.w.z. $x_B = a^*_{\bullet 0}$, $x_N = 0$, inderdaad niet aan (2.3.5). Deze ongelijkheid kapt dus de LP-oplossing van het toegelaten gebied af en is dus een snede. De rij $r$ heet de *bronrij* van de snede (de mag ook rij 0 zijn). De snede (2.3.5) is in 1958 door Gomory voorgesteld.[4]

Omdat (2.3.3) geheeltallige coëfficiënten heeft, is voor iedere geheeltallige $x$ het verschil tusssen het rechter- en het linkerlid geheel. Omdat de fractie-snede (2.3.5) hieruit is ontstaan, na combinatie

[4] R.E. Gomory, *An algorithm for integer solutions to linear programs*, Princeton IBM Mathematical Research Report (1958).

met de gelijkheid (2.3.1) voor $i = r$, geldt dat voor iedere geheeltallige $x$ ook het verschil tussen het rechter- en linkerlid van (2.3.5) geheel is. Schrijven we (2.3.5) met een verschilvariabele $s$, d.w.z.

$$s = -f_{r0} + \sum_j f_{rj}(x_N)_j, \tag{2.3.6}$$

dan is $s$ weer een niet-negatieve geheeltallige variabele. Voegen we de vergelijking (2.3.6) aan het tableau toe, dan is de bijbehorende oplossing wel duaal, maar niet primaal toelaatbaar (omdat $-f_{r0} < 0$). De duale simplex methode is dus de aangewezen methode om het nieuwe probleem te optimaliseren.

Door het toevoegen van een snede gaat de primaire toelaatbaarheid verloren (immers $-f_{r0} < 0$). De onderaan het tableau toegevoegde snede wordt als pivotrij gekozen en de bijbehorende $s$-variabele verdwijnt bij het pivoten uit de basis. Als $s$ op een gegeven moment weer terugkeert in de basis, d.w.z. dat de snede in de op dat moment verkregen oplossing niet bindend is, dan kunnen we deze beperking verder weglaten.

Er is nog vrijheid in de keuze van de bronrij $r$. Als de snede (2.3.5) aan het tableau wordt toegevoegd en kolom $k(r)$ is de pivotkolom als rij $r$ de snede genereert, dan daalt de waarde van de doelfunctie met $\frac{f_{r0} \cdot a^*_{0\,k(r)}}{f_{rk(r)}}$. Hoe groter deze daling is, hoe *dieper* de snede.
Criteria voor de keuze van $r$ zijn:

(1) $f_{r0} = max_i\, f_{i0}$; (2) $\frac{f_{r0}}{f_{rk(r)}} = max_i\, \frac{f_{i0}}{f_{ik(i)}}$; (3) $\frac{f_{r0} \cdot a^*_{0k(r)}}{f_{rk(r)}} = max_i\, \frac{f_{i0} \cdot a^*_{0k(i)}}{f_{ik(i)}}$.

**Voorbeeld 2.6**

$$max \left\{ 3x_1 - x_2 \;\middle|\; \begin{array}{rcr} 3x_1 - 2x_2 & \leq & 3 \\ -5x_1 - 4x_2 & \leq & -10 \\ 2x_1 + x_2 & \leq & 5 \\ x_1, x_2 \geq 0 \text{ en geheel} & & \end{array} \right\}.$$

We voeren verschilvariabelen $y_1$, $y_2$ en $y_3$ in en lossen het LP-probleem op. Daarna kiezen we bronrij $r$ volgens het criterium $f_{r0} = max_i\, f_{i0}$. Dit geeft de volgende tableaus:

| | | $y_1$ | $y_3$ |
|---|---|---|---|
| $x_0$ | $\frac{30}{7}$ | $\frac{5}{7}$ | $\frac{3}{7}$ |
| $x_1$ | $\frac{13}{7}$ | $\frac{1}{7}$ | $\frac{2}{7}$ |
| $x_2$ | $\frac{9}{7}$ | $-\frac{2}{7}$ | $\frac{3}{7}$ |
| $y_2$ | $\frac{31}{7}$ | $-\frac{3}{7}$ | $\frac{22}{7}$ |

Neem de rij van $x_1$ om de snede te genereren:
$s_1 = -\frac{6}{7} + \frac{1}{7}y_1 + \frac{2}{7}y_3$.
Voeg deze snede toe, dit geeft het volgende tableau.

| | | $y_1$ | $y_3$ |
|---|---|---|---|
| $x_0$ | $\frac{30}{7}$ | $\frac{5}{7}$ | $\frac{3}{7}$ |
| $x_1$ | $\frac{13}{7}$ | $\frac{1}{7}$ | $\frac{2}{7}$ |
| $x_2$ | $\frac{9}{7}$ | $-\frac{2}{7}$ | $\frac{3}{7}$ |
| $y_2$ | $\frac{31}{7}$ | $-\frac{3}{7}$ | $\frac{22}{7}$ |
| $s_1$ | $-\frac{6}{7}$ | $-\frac{1}{7}$ | $-\frac{2}{7}$ |

Volgens de duale simplex methode wordt $-\frac{2}{7}$ als pivot genomen, wat het volgende tableau geeft.

| | | $y_1$ | $s_1$ |
|---|---|---|---|
| $x_0$ | 3 | $\frac{1}{2}$ | $\frac{3}{2}$ |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | 0 | $-\frac{1}{2}$ | $\frac{3}{2}$ |
| $y_2$ | $-5$ | $-2$ | 11 |
| $y_3$ | 3 | $\frac{1}{2}$ | $-\frac{7}{2}$ |

Dit is geen optimaal tableau. De volgende pivotrij is de rij van $y_2$ en de pivot is -2.

| | | $y_2$ | $s_1$ |
|---|---|---|---|
| $x_0$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $\frac{17}{4}$ |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | $\frac{5}{4}$ | $-\frac{1}{4}$ | $-\frac{5}{4}$ |
| $y_1$ | $\frac{5}{2}$ | $-\frac{1}{2}$ | $-\frac{11}{2}$ |
| $y_3$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $-\frac{3}{4}$ |

Neem de rij van $y_3$ als bronrij:
$s_2 = -\frac{3}{4} + \frac{1}{4}y_2 + \frac{1}{4}s_1$.
Deze snede toevoegen geeft het volgende tableau.

| | | $y_2$ | $s_1$ |
|---|---|---|---|
| $x_0$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $\frac{17}{4}$ |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | $\frac{5}{4}$ | $-\frac{1}{4}$ | $-\frac{5}{4}$ |
| $y_1$ | $\frac{5}{2}$ | $-\frac{1}{2}$ | $-\frac{11}{2}$ |
| $y_3$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $-\frac{3}{4}$ |
| $s_2$ | $-\frac{3}{4}$ | $-\frac{1}{4}$ | $-\frac{1}{4}$ |

$y_2$ wordt de pivotkolom, wat het volgende tableau oplevert.

| | | $s_2$ | $s_1$ |
|---|---|---|---|
| $x_0$ | 1 | 1 | 4 |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | 2 | $-1$ | $-1$ |
| $y_1$ | 4 | $-2$ | $-5$ |
| $y_3$ | 1 | 1 | $-1$ |
| $y_2$ | 3 | $-4$ | 1 |

Dit tableau is optimaal met een geheeltallige oplossing, die dus optimaal is. De optimale oplossing luidt:
$x_1 = 1,\ x_2 = 2$ met waarde 1.

**Eindigheid**

Deze methode is niet zonder meer eindig. Door de keuze van de bronrij en de keuzes in de duale simplex methode nader te specificeren wordt een eindige methode verkregen onder de veronderstelling dat het toegelaten gebied $\{x \mid Ax \leq b;\ x \geq 0 \text{ en geheel}\}$ begrensd is.

Keuze bronrij

Als keuze voor de bronrij nemen we $r$ zdd. $r = min\{i \mid f_{i0} > 0\}$.

Keuzes in de duale simplex methode

Allereerst veronderstellen we dat er geen twee kolommen van $A$ zijn die een multiplicatieve constante verschillen (als de coëfficiënten in de doelfunctie verschillen, dan kan de kolom van de kleinste coëfficiënt worden weggelaten; als de coëfficiënten in de doelfunctie hetzelfde zijn, dan kunnen de kolommen worden samengevoegd).
We zullen vervolgens laten zien dat alle kolommen van de tableaus lexicografisch positief kunnen zijn. Met $\succ^L$ en $\succeq^L$ noteren we lexicografisch groter resp. lexicografisch groter of gelijk.
Los eerst het bijbehorende LP-probleem op.
Als $a^*_{0j} > 0$ voor alle $j$, dan zijn de kolommen van dit tableau lexicografisch positief.
Als $a^*_{0j} = 0$ voor sommige $j$'s, zeg voor $\{j \mid j \in J_0\}$, dan voegen we direct na de doelfunctie de beperking $\sum_{j \in J_0} (x_N)_j \leq M$ toe, waarbij $M$ voldoende groot is zodat de optimale oplossing hieraan voldoet (dit kan i.v.m. de veronderstelling dat het toegelaten gebied begrensd is). Hiermee wordt dit tableau lexicografisch positief en moeten we laten zien dat bij toepassing van de duale simplex methode de kolommen lexicografisch positief kunnen blijven.

Beschouw een iteratie van de duale simplex methode en veronderstel dat het tableau lexicografisch positieve kolommen heeft. Laat rij $l$ als pivotrij worden gekozen. Voor de pivotkolom $k$ moet gelden dat

$$\frac{a^*_{0k}}{-a^*_{lk}} = min\left\{ \frac{a^*_{0j}}{-a^*_{lj}} \;\middle|\; a^*_{lj} < 0 \right\}.$$

Als hiervoor meer kolommen in aanmerking komen, dan kiezen we kolom $k$ zdd. $\frac{a^*_{\bullet k}}{-a^*_{lk}}$ de lexicografisch *kleinste* is. Deze keuze is uniek bepaald, omdat er anders twee kolommen zijn die een multiplicatieve factor verschillen, wat in strijd is met de gemaakte veronderstelling.
De nieuwe pivotkolom ontstaat uit de oude door te delen door het positieve element $-a^*_{lk}$ en blijft dus lexicografisch positief (voor het pivotelement zelf geldt dit niet, maar omdat $a^*_{\bullet k} \succ^L 0$, is een $a^*_{ik} > 0$ met $i < l$, zodat de lexicografische eigenschap niet in gevaar komt).
Noteer de getransformeerde elementen met $(a^*_{ij})'$.
Voor de kolommen $j \neq k$ geldt (afgezien van de elementen in rij $l$): $(a^*_{\bullet j})' = a^*_{\bullet j} + (a^*_{\bullet k})' \cdot a^*_{lj}$.

Als $a^*_{lj} \geq 0$: $(a^*_{\bullet j})' \succeq^L a^*_{\bullet j} \succ^L 0$.

Als $a^*_{lj} < 0$: $\frac{a^*_{\bullet j}}{-a^*_{lj}} \succ^L (a^*_{\bullet k})'$, d.w.z. $(a^*_{\bullet j})' = a^*_{\bullet j} + (a^*_{\bullet k})' \cdot a^*_{lj} \succ^L 0$.

We moeten tenslotte laten zien dat de pivotrij geen probleem oplevert.

Als $a^*_{lj} \leq 0$, dan is $(a^*_{lj})' \geq 0$, zodat dit geen probleem geeft.

Als $a^*_{lj} > 0$, dan is er (omdat buiten rij $l$ $(a^*_{\bullet j})' \succ^L 0$ is) een probleem als $(a^*_{ij})' = 0$ voor $i = 0, 1, \ldots, l-1$. Maar dat kan alleen als $a^*_{ij} = a^*_{ik} = 0$ voor $i = 0, 1, \ldots, l-1$ (aan te tonen met inductie naar $i$). Omdat $a^*_{lk} < 0$ impliceert dit dat $a^*_{\bullet k}$ lexicografisch negatief is: tegenspraak.

Het toevoegen van de sneden tast de lexicografische eigenschap niet aan, omdat de sneden onderaan het tableau worden geplaatst. De lexicografische eigenschap wordt ook niet aangetast als we een $s$-variabele die weer terugkeert in de basis weglaten, want dan laten we de getransformeerde pivotrij $l$ weg en we hebben gezien dat het tableau zonder de getransformeerde pivotrij $l$ ook al lexicografisch positief is. Hiermee hebben we het volgende lemma aangetoond.

**Lemma 2.2**
*Als het LP-probleem begrensd is, dan heeft de duale simplex methode een implementatie waarin alle kolommen van de tableaus lexicografisch positief zijn.*

**Stelling 2.3**
*Veronderstel dat het IP-probleem begrensd is. Met de aangegeven keuzes van de bronrij en de implementatie van de duale simplex methode is Gomory's fractie-snede algoritme eindig.*

**Bewijs**
Veronderstel dat er geen eindigheid is en laat $t$ de iteratie-index zijn. De rij $\{a^t_{00}\}$ van de waarden van de doelfunctie is niet-stijgend en naar beneden begrensd door het IP-optimum. Deze rij heeft een limiet, zeg $L$. Veronderstel tevens dat $L$ niet geheel is, zeg $L = \lfloor L \rfloor + f$ met $0 < f < 1$.
Na een eindig aantal iteraties, zeg $q$, is er een primaal en duaal optimaal tableau met

$$a^q_{00} = \lfloor L \rfloor + f_q \text{ met } f \leq f_q < 1.$$

Volgens onze keuze van de bronrij fungeert de 0-de rij nu als bronrij en wordt als snede genomen

$$s = -f_q + \textstyle\sum_j f_{0j}(x_N)_j;\ s \geq 0 \text{ en geheel.}$$

Omdat $a^q_{0k} \geq f_{0k} > 0$ ($-f_{0k}$ is het pivotelement), geldt:

$$a_{00}^{q+1} = a_{00}^q - f_q \frac{a_{0k}^q}{f_{0k}} \leq a_{00}^q - f_q \frac{a_{0k}^q}{a_{0k}^q} = \lfloor L \rfloor < L,$$

tegenspraak, dus $f = 0$ en $a_{00}^t = L$ met $L$ geheel voor alle $t \geq T_0$ voor zekere $T_0$. Dan is $a_{0k_t}^t = 0$ voor alle $t \geq T_0$ met $k_t$ het nummer van de pivotkolom in de $t$-de iteratie. Maar dan is, vanwege de lexicografisch positieve kolommen, $a_{1k_t}^t \geq 0$ voor alle $t \geq T_0$, waaruit volgt dat $\{a_{10}^t\}$ niet-stijgend is voor alle $t \geq T_0$. Vervolgens beschouwen we $a_{10}^t$ voor $t \geq T_0$. Veronderstel dat $a_{10}^t$ niet geheel is. Dan wordt rij 1 de bronrij, is $a_{1k_t}^t > 0$ (vanwege de lexicografische positiviteit) en is

$$a_{10}^{t+1} = a_{10}^t - f_{10}^t \frac{a_{1k_t}^t}{f_{1k_t}^t} \leq a_{10}^t - f_{10}^t \frac{a_{1k_t}^t}{a_{1k_t}^t} = \lfloor a_{10}^t \rfloor.$$

Daarom is òfwel $a_{10}^t$ een geheel getal voor alle $t \geq T_1$, òfwel $a_{10}^t$ wordt op een zeker moment negatief. In dit laatste geval is rij 1 de pivotrij en dus $a_{1k_t}^t < 0$ voor zekere $t$. Maar dan is de kolom van $k_t$ niet meer lexicografisch positief: tegenspraak. Er geldt dus dat $a_{10}^t$ een constant geheel getal is voor alle $t \geq T_1$ voor zekere $T_1 \geq T_0$.
Door dit herhaald toe te passen krijgen we een $T$ zdd. $a_{i0}^t$ constant en geheel is voor alle $t \geq T$, waarbij $T$ een zeker eindig getal is, en voor alle $i$ (het tableau heeft eindig veel rijen omdat we de rijen van snedes die weer in de basis terugkeren verwijderen). Het algoritme heeft dus hoogstens $T$ iteraties: tegenspraak. Hieruit volgt dat het algoritme eindig is. □

**Algoritme 2.2** *Gomory's fractie-snede algoritme*

1. *Initialisatie*
   Los het bijbehorende LP-probleem op.

2. *Stopcriterium*
   Als het tableau een geheeltallige oplossing $x^*$ heeft: $x^*$ is de optimale oplossing (STOP).

3. *Iteratie*

   (a) Kies een bronrij $r$ zdd. $a_{r0}^*$ niet geheel is.

   (b) Voeg de snede (2.3.5) aan het tableau toe.

   (c) Bepaal een optimaal tableau met de duale simplex methode (als een $s$-variabele terugkeert in de basis, dan laten we de desbetreffende rij weg) en ga naar stap 2.

Opmerkingen:

1. Jeroslow en Kortanek [5] hebben laten zien dat voor iedere $N$ er een $t$ is zdd. het probleem

$$max \left\{ -x_1 \;\middle|\; \begin{array}{l} x_1 - \frac{1}{t} x_2 = \frac{1}{2} \\ x_1, x_2 \geq 0 \text{ en geheel} \end{array} \right\}$$

minstens $N$ iteraties nodig heeft: er is dus geen uniforme bovengrens aan het aantal iteraties.

2. Dantzig [6] heeft een bijzonder eenvoudige snede voorgesteld: $\sum_j (x_N)_j \geq 1$.

[5] R.G. Jeroslow and K.O. Kortanek, *On an algorithm of Gomory*, SIAM Journal 21 (1971) 55–60.
[6] G.B. Dantzig, *Note on solving linear programs in integers*, Naval Research Logistics Quarterly 6 (1959) 75–76.

Gomory en Hoffman [7] hebben laten zien dat deze methode in het algemeen niet eindig is. Bowman en Nemhauser hebben vervolgens een modificatie voorgesteld die wel eindig is.[8]

**Vraag 2.6**

Beschouw het IP-probleem $max \left\{ 3x_1 - x_2 \;\middle|\; \begin{array}{rcr} 3x_1 - 2x_2 & \leq & 3 \\ -5x_1 - 4x_2 & \leq & -10 \\ 2x_1 + x_2 & \leq & 5 \\ x_1, x_2 \geq 0 \text{ en geheel} & & \end{array} \right\}$.

Pas op dit probleem Gomory's fractie-snede algoritme toe met de bronrij en de implementatie van de duale simplex methode volgens de keuzes van Stelling 2.3.

### 2.3.2 Gomory's snede voor gemengd geheeltallige optimalisering

Beschouw het volgende gemengd geheeltallig lineair optimaliseringsprobleem met $x$ de vector van de geheeltallige variabelen en $y$ de vector van de continue variabelen:

$$max \left\{ \sum_{j=1}^{n} p_j x_j + \sum_{k=1}^{r} q_k y_k \mid Ax + Cy \leq b; x, y \geq 0 \text{ en } x \text{ geheel} \right\}, \tag{2.3.7}$$

met $A$ een $(m \times n)$-matrix, $C$ een $(m \times r)$-matrix en $b \in \mathbb{R}^m$. Los de LP-relaxatie van (2.3.7) op met de simplex methode en veronderstel dat het optimale LP-tableau de volgende gedaante heeft:

$$\begin{cases} (y_B)_i = a^*_{i0} - \sum_{j \in J_1} a^*_{ij}(x_N)_j - \sum_{j \in J_2} a^*_{ij}(y_N)_j, \; i \in M_1 \\ (x_B)_i = a^*_{i0} - \sum_{j \in J_1} a^*_{ij}(x_N)_j - \sum_{j \in J_2} a^*_{ij}(y_N)_j, \; i \in M_2 \end{cases} \tag{2.3.8}$$

waarbij de 0-de rij correspondeert met de doelfunctie en $M_1 \cup M_2 = \{0, 1, 2, \ldots, m\}$. We merken op dat aan (2.3.7) verschilvariabelen moeten worden toegevoegd die behandeld worden als continue $y$-variabelen. $J_1$ is de verz. van de $x$-variabelen die in het optimale tableau niet in de basis zitten en $J_2$ is de verz. van de $y$-variabelen (zowel de oorspronkelijke als de verschilvariabelen) die in het optimale tableau niet in de basis zitten.

Als $a^*_{i0}$ geheel is voor alle $i \in M_2$, dan is $x$ geheeltallig en het tableau optimaal. Veronderstel dus verder dat $a^*_{r0}$ niet geheel is voor een $r \in M_2$. De volgende stelling geeft een toegelaten snede.

**Stelling 2.4**

*Kies een $r \in M_2$ met $f_0 = a^*_{r0} - \lfloor a^*_{r0} \rfloor > 0$ en laat $f_j = a^*_{rj} - \lfloor a^*_{rj} \rfloor$, $j \in J_1 \cup J_2$.*

*Dan is*

$$\sum_{j \in J_1 | f_j \leq f_0} f_j (x_N)_j + \sum_{j \in J_1 | f_j > f_0} \frac{f_0(1 - f_j)}{1 - f_0} (x_N)_j + \sum_{j \in J_2 | a^*_{rj} > 0} a^*_{rj} (y_N)_j + \sum_{j \in J_2 | a^*_{rj} < 0} \frac{f_0}{f_0 - 1} a^*_{rj} (y_N)_j \geq f_0 \tag{2.3.9}$$

*een toegelaten snede.*

[7] R.E. Gomory and A.J. Hoffman, *On the convergence of an integer programming process*, Naval Research Logistics Quarterly 10 (1963) 121–123.

[8] V.J. Bowman and G.L. Nemhauser, *A finiteness proof for modified Dantzig cuts for integer programming*, Naval Research Logistics Quarterly 17 (1970) 309–313.

**Bewijs**

Laat $\alpha_j \equiv a^*_{rj}\ (mod\ 1)$ voor $j \in J_1$. Dan volgt uit (2.3.8) dat voor iedere geheeltallige $x$ geldt:

$$f_0 \equiv \sum_{j\in J_1} \alpha_j (x_N)_j + \sum_{j\in J_2} a^*_{rj}(y_N)_j\ (mod\ 1) \tag{2.3.10}$$

Neem $\alpha_j = \begin{cases} f_j & \text{als } f_j \leq f_0; \\ f_j - 1 & \text{als } f_j > f_0. \end{cases}$ Voor iedere toelaatbare $(x, y)$ geldt:

Als $\sum_{j\in J_1} \alpha_j (x_N)_j + \sum_{j\in J_2} a^*_{rj}(y_N)_j \geq 0$, dan:

$$\sum_{j\in J_1 | f_j \leq f_0} f_j (x_N)_j + \sum_{j\in J_2 | a^*_{rj} > 0} a^*_{rj}(y_N)_j \geq \sum_{j\in J_1} \alpha_j (x_N)_j + \sum_{j\in J_2} a^*_{rj}(y_N)_j \geq f_0 \tag{2.3.11}$$

Als $\sum_{j\in J_1} \alpha_j (x_N)_j + \sum_{j\in J_2} a^*_{rj}(y_N)_j < 0$, dan:

$\sum_{j\in J_1 | f_j > f_0} (f_j - 1)(x_N)_j + \sum_{j\in J_2 | a^*_{rj} < 0} a^*_{rj}(y_N)_j \leq \sum_{j\in J_1} \alpha_j (x_N)_j + \sum_{j\in J_2} a^*_{rj}(y_N)_j \leq f_0 - 1$,

zodat geldt:

$$\frac{f_0}{f_0 - 1} \left\{ \sum_{j\in J_1 | f_j > f_0} (f_j - 1)(x_N)_j + \sum_{j\in J_2 | a^*_{rj} < 0} a^*_{rj}(y_N)_j \right\} \geq \frac{f_0}{f_0 - 1} \cdot (f_0 - 1) = f_0 \tag{2.3.12}$$

Omdat

$$\sum_{j\in J_1 | f_j \leq f_0} f_j (x_N)_j + \sum_{j\in J_2 | a^*_{rj} > 0} a^*_{rj}(y_N)_j \geq 0 \text{ en } \frac{f_0}{f_0 - 1} \left\{ \sum_{j\in J_1 | f_j > f_0} (f_j - 1)(x_N)_j + \sum_{j\in J_2 | a^*_{rj} < 0} a^*_{rj}(y_N)_j \right\} \geq 0 \tag{2.3.13}$$

volgt uit (2.3.11), (2.3.12) en (2.3.13) dat (2.3.9) geldt. Omdat in de huidige optimale LP-oplossing $x_N = y_N = 0$, voldoet deze oplossing niet aan (2.3.9), zodat (2.3.9) inderdaad een snede is. □

Opmerking:

Deze snede is door Gomory [9] in 1960 voorgesteld. Ook voor deze methode kan voor een speciale implementatie de eindigheid worden aangetoond op grond van lexicografische argumenten.[10]

**Voorbeeld 2.7**

Beschouw Voorbeeld 2.6, maar zonder de eis dat $x_2$ geheel is. Het eerste LP-tableau dat we beschouwen is hetzelfde als het eerste tableau uit Voorbeeld 2.6.

| | | $y_1$ | $y_3$ |
|---|---|---|---|
| $x_0$ | $\frac{30}{7}$ | $\frac{5}{7}$ | $\frac{3}{7}$ |
| $x_1$ | $\frac{13}{7}$ | $\frac{1}{7}$ | $\frac{2}{7}$ |
| $x_2$ | $\frac{9}{7}$ | $-\frac{2}{7}$ | $\frac{3}{7}$ |
| $y_2$ | $\frac{31}{7}$ | $-\frac{3}{7}$ | $\frac{22}{7}$ |

Nemen we de rij van $x_1$ om de snede te genereren. Deze snede is:
$s_1 = -\frac{6}{7} + \frac{1}{7}y_1 + \frac{2}{7}y_3$.
Voegen we deze snede toe, dan ontstaat het volgende tableau.

| | | $y_1$ | $y_3$ |
|---|---|---|---|
| $x_0$ | $\frac{30}{7}$ | $\frac{5}{7}$ | $\frac{3}{7}$ |
| $x_1$ | $\frac{13}{7}$ | $\frac{1}{7}$ | $\frac{2}{7}$ |
| $x_2$ | $\frac{9}{7}$ | $-\frac{2}{7}$ | $\frac{3}{7}$ |
| $y_2$ | $\frac{31}{7}$ | $-\frac{3}{7}$ | $\frac{22}{7}$ |
| $s_1$ | $-\frac{6}{7}$ | $-\frac{1}{7}$ | $-\frac{2}{7}$ |

Volgens de duale simplex methode wordt nu $-\frac{2}{7}$ (in de onderste rij) als pivot genomen, waarmee het volgende tableau ontstaat.

| | | $y_1$ | $s_1$ |
|---|---|---|---|
| $x_0$ | 3 | $\frac{1}{2}$ | $\frac{3}{2}$ |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | 0 | $-\frac{1}{2}$ | $\frac{3}{2}$ |
| $y_2$ | $-5$ | $-2$ | 11 |
| $y_3$ | 3 | $\frac{1}{2}$ | $-\frac{7}{2}$ |

Dit tableau is nog niet optimaal; we voeren vervolgens een pivotstap uit in de rij van $y_2$ en met pivot -2.

[9] R.E. Gomory, *An algorithm for the mixed integer problem*, RAND Report P-1885 (1960).
[10] Zie T.C. Hu, *Integer programming and network flows*, Addison-Wesley, 1970, pp. 255-259.

| | | $y_2$ | $s_1$ |
|---|---|---|---|
| $x_0$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $\frac{17}{4}$ |
| $x_1$ | 1 | 0 | 1 |
| $x_2$ | $\frac{5}{4}$ | $-\frac{1}{4}$ | $-\frac{5}{4}$ |
| $y_1$ | $\frac{5}{2}$ | $-\frac{1}{2}$ | $-\frac{11}{2}$ |
| $y_3$ | $\frac{7}{4}$ | $\frac{1}{4}$ | $-\frac{3}{4}$ |

Omdat $x_1$ geheeltallig is,
is nevenstaand tableau optimaal.
De optimale oplossing luidt dus:
$x_1 = 1,\ x_2 = \frac{5}{4}$ met waarde $\frac{7}{4}$.

**Vraag 2.7**

Los met Gomory's snede algoritme het volgende gemengd geheeltallige probleem op.

$$max \left\{ 7x_1 + 9x_2 \;\middle|\; \begin{array}{rcrcl} -x_1 & + & 3x_2 & \leq & 6 \\ 7x_1 & + & x_2 & \leq & 35 \\ \multicolumn{5}{c}{x_1, x_2 \geq 0;\ x_1 \text{ geheel}} \end{array} \right\}.$$

### 2.3.3 Opgaven

**Opgave 2.8**

Los met Gomory's fractie-snede het volgende probleem op:

$$max \left\{ -4x_1 - 5x_2 \;\middle|\; \begin{array}{c} -x_1 - 4x_2 \leq -5 \\ -3x_1 - 2x_2 \leq -7 \\ x_1, x_2 \geq 0 \text{ en geheel} \end{array} \right\}.$$

Kies voor de bronrij $r$ zdd. $f_{r0} = max_i\, f_{i0}$.

**Opgave 2.9**

Los met Gomory's snede het volgende gemengd geheeltallige probleem op:

$$max \left\{ 4x_1 - x_2 \;\middle|\; \begin{array}{rcrcl} 7x_1 & - & 2x_2 & \leq & 14 \\ & & x_2 & \leq & 3 \\ 2x_1 & - & 2x_2 & \leq & 3 \\ \multicolumn{5}{c}{x_1 \geq 0 \text{ en geheel};\ x_2 \geq 0} \end{array} \right\}.$$

**Opgave 2.10**

Beschouw een zuiver geheeltallig lineair programmeringsprobleem. Hierop kan zowel Gomory's fractie-snede als Gomory's snede voor een gemengd geheeltallige probleem worden toegepast.
Toon aan dat Gomory's snede voor een gemengd geheeltallige probleem sterker is dan Gomory's fractie-snede, d.w.z. dat als $x$ voldoet aan het LP-probleem met de beperking van Gomory's snede voor een gemengd geheeltallige probleem, dan voldoet $x$ ook aan het LP-probleem met de beperking van Gomory's fractie-snede.

## 2.4 Handelsreizigersprobleem

### 2.4.1 Inleiding en formuleringen

Er zijn $n$ steden en de afstand $l_{ij}$ om van stad $i$ naar stad $j$ te gaan in bekend voor iedere $i$ en $j$. Een handelsreiziger staat voor de opdracht om een rondreis langs alle steden te maken zdd. de totale reisafstand minimaal is. Veronderstel dat hij begint in stad 1. Er zijn dan $n-1$ mogelijkheden voor de tweede stad; voor de derde stad zijn er dan $n-2$ mogelijkheden, etc. Er zijn dus $(n-1)!$ mogelijke routes en daaruit moet de kortste worden gevonden. Expliciet onderzoek werkt in de praktijk niet: bijv. bij 24 steden zijn er 23! $\approx 2.6 \times 10^{22}$ routes. Er kan bewezen worden dat dit probleem $\mathcal{NP}$-volledig is.[11] In het Engels spreken we over het Traveling Salesman Problem, afgekort TSP. We zullen deze afkorting hier ook gebruiken.

De reden om het handelsreizigersprobleem uitvoerig te bestuderen is dat de technieken die we voor het TSP zullen bespreken illustratief zijn voor de manier waarop $\mathcal{NP}$-moeilijke problemen in het algemeen worden aangepakt. We behandelen zowel de branch-and-bound methode om een exacte optimale oplossing te vinden als benaderingsalgoritmen.
Pas vanaf 1954 is er sprake van een serieuze wiskundige bestudering van het TSP. Tot die tijd was men in staat om problemen op te lossen tot 20 steden.[12] Dertig jaar later, in 1984, stond het record op 318 steden. Het huidige record, uit 2006, staat op 85.900 steden. Deze vooruitgang is te danken aan twee factoren: sterk verbeterde wiskundige technieken en sterk verbeterde computerapparatuur.[13]

We kunnen het TSP als combinatorisch optimaliseringsprobleem formuleren. Introduceer daartoe (0,1)-variabelen $x_{ij}$ met de volgende interpretatie: $x_{ij} = 0$ (1) betekent dat de directe verbinding van stad $i$ naar stad $j$ niet (wel) wordt gekozen. Op deze wijze ligt het voor de hand het volgende probleem te bekijken:

$$min \left\{ \sum_{i,j=1}^{n} l_{ij} x_{ij} \;\middle|\; \begin{array}{l} \sum_j x_{ij} = 1,\ 1 \le i \le n \\ \sum_j x_{ji} = 1,\ 1 \le i \le n \\ x_{ij} \in \{0,1\} \text{ voor alle } (i,j) \end{array} \right\}. \tag{2.4.1}$$

Hierbij hebben de voorwaarden $\sum_j x_{ij} = 1,\ 1 \le i \le n$, de betekenis dat er voor iedere stad $i$ precies één andere stad is waar we vanuit stad $i$ naar toe gaan; zo geven de voorwaarden $\sum_j x_{ji} = 1,\ 1 \le i \le n$, aan dat er voor iedere stad $i$ ook precies één andere stad is van waaruit we naar stad $i$ toegaan. De doelfunctie telt alle afstanden die in de route zitten bij elkaar op.

[11] Zie: R.M. Karp, *Reducibility among combinatorial problems*, in: R.E. Miller and J.W. Thatcher, *Complexity of computer computations*, Plenum Press, New York (1972) 85–103.

[12] Dantzig, Fulkerson en Johnson slaagden er in 1954 in om een probleem met 49 steden op te lossen, wat voor die tijd als een grote doorbraak werd beschouwd. Zie G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, *Solution of a large scale traveling salesman problem*, Operations Research 2 (1954) 393–410.

[13] Een mooie website waar veel informatie over het TSP te vinden is, is www.tsp.gatech.edu. In 2006 is een boek over het TSP verschenen met een vrij compleet overzicht over de stand van zaken: David L. Applegate, Robert E. Bixby, Vasek Chvátal and William J. Cook: *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, 2006.

De formulering (2.4.1) is echter niet correct, want er worden ook subrondes toegelaten, terwijl er maar één ronde mag zijn die alle steden bevat. Een goede formulering krijgen we bijvoorbeeld door aan het probleem de volgende beperkingen toe te voegen:

$$u_i - u_j + nx_{ij} \leq n-1 \text{ voor alle } i,j \geq 2. \tag{2.4.2}$$

In Opgave 2.11 wordt gevraagd aan te tonen dat dit een correcte formulering is voor het TSP.[14]

We zullen nog twee andere formuleringen geven.
Laat $S \subseteq \{2,3,\ldots,n\}$ met $|S| \geq 2$. Als we kijken naar de verbindingen die beide eindpunten in $S$ hebben, dan kunnen in een ronde die alle knooppunten bevat hoogstens $|S|-1$ van dergelijke verbindingen zitten (anders is er subronde in $S$). Hieruit volgt de volgende formulering van dit probleem:

$$min \left\{ \sum_{i,j=1}^{n} l_{ij}x_{ij} \;\middle|\; \begin{array}{l} \sum_j x_{ij} = 1,\ 1 \leq i \leq n \\ \sum_j x_{ji} = 1,\ 1 \leq i \leq n \\ \sum_{i\in S, j\in S} x_{ij} \leq |S|-1 \text{ voor alle } S \subseteq \{2,3,\ldots,n\} \text{ met } |S| \geq 2 \\ x_{ij} \in \{0,1\} \text{ voor alle } (i,j) \end{array} \right\}. \tag{2.4.3}$$

Een derde formulering krijgen we door weer deelverz. $S \subseteq \{2,3,\ldots,n\}$ te nemen met $|S| \geq 2$. Als we kijken naar de pijlen met beginpunt in $S$ en eindpunt niet in $S$, dan heeft iedere ronde die alle knooppunten bevat minstens één van dergelijke pijlen, terwijl als er subtours zijn er een $S$ te vinden is waarvoor dit niet geldt. Door de beperkingen

$$\sum_{i\in S, j\notin S} x_{ij} \geq 1 \text{ voor alle } S \subseteq \{2,3,\ldots,n\} \text{ met } |S| \geq 2 \tag{2.4.4}$$

worden subtours wel en een tour langs alle knooppunten niet uitgesloten. Dit geeft de formulering:

$$min \left\{ \sum_{i,j=1}^{n} l_{ij}x_{ij} \;\middle|\; \begin{array}{l} \sum_j x_{ij} = 1,\ 1 \leq i \leq n \\ \sum_j x_{ji} = 1,\ 1 \leq i \leq n \\ \sum_{i\in S, j\notin S} x_{ij} \geq 1 \text{ voor alle } S \subseteq \{2,3,\ldots,n\} \text{ met } |S| \geq 2 \\ x_{ij} \in \{0,1\} \text{ voor alle } (i,j) \end{array} \right\}. \tag{2.4.5}$$

Opmerking
Formulering (2.4.2) heeft $\mathcal{O}(n^2)$ beperkingen, terwijl de formuleringen (2.4.3) en (2.4.5) $\mathcal{O}(2^n)$ beperkingen hebben. De eerste formulering is dus veel compacter, wat niet per se betekent dat deze ook beter is.
De tweede en derde formulering hebben het voordeel dat niet direct alle beperkingen hoeven te worden toegevoegd. Je zou pas nadat een huidige formulering een subronde oplevert deze subronde kunnen uitsluiten door dan de beperking toe te voegen met $S$ de verz. van de knooppunten van de subronde en dan het probleem opnieuw op te lossen. Dit geeft een rij van verwante optimaliseringsproblemen. Vaak is dit een handige oplossingsstrategie.

[14] De formulering met (2.4.2) is afkomstig van Tucker, zie: C.E. Miller, A.W. Tucker and R.A. Zemlin, *Integer programming formulation and traveling salesman problem*, Journal of the ACM 7 (1960) 326–329.

**Toepassing 2.1** *Machine scheduling*
Een aantal taken, zeg $n$, moet achter elkaar, in een nader te bepalen volgorde, op één machine worden uitgevoerd. Als taak $j$ de vorige taak is geweest, dan is de insteltijd van de machine $t_{jk}$ als taak $k$ direct na taak $j$ wordt gekozen. Voor de eerste taak geldt een insteltijd $t_{0k}$ als taak $k$ als eerste wordt gekozen. Verder is $t_k$ de tijdsduur voor het verwerken van taak $k$ op de machine. De vraagstelling luidt: in welke volgorde moeten de taken op de machine worden gescheduled om al het werk zo vroeg mogelijk af te hebben?
Beschouw een bepaalde volgorde, zeg $i_1, i_2, \ldots, i_n$. De totale tijdsduur is dan:
$t_{0i_1} + t_{i_1} + t_{i_1 i_2} + t_{i_2} + \cdots + t_{i_{n-1} i_n} + t_{i_n} = t_{0i_1} + t_{i_1 i_2} + \cdots + t_{i_{n-1} i_n} + \sum_{k=1}^{n} t_{i_k}$.
De term $\sum_{k=1}^{n} t_k$ is onafhankelijk van de gekozen volgorde. Het machine scheduling probleem is dus equivalent met het volgende TSP:
$V = \{0, 1, \ldots, n\};\ l_{0k} = t_{0k},\ 1 \leq k \leq n;\ l_{jk} = t_{jk},\ 1 \leq j, k \leq n;\ l_{k0} = 0,\ 1 \leq k \leq n.$

### 2.4.2 Branch-and-Bound methode

Het TSP behoort tot de klasse $\mathcal{NPC}$. Het is dus zeer onwaarschijnlijk dat hiervoor een efficiënt algoritme bestaat. De in deze paragraaf te behandelen branch-and-bound [15] methode is dan ook niet polynomiaal. De branch-and-bound methode voor het TSP gaat als volgt.

Allereerst reduceren we de afstandenmatrix. Iedere route door alle steden bevat precies één element van iedere rij en van iedere kolom (vanuit iedere plaats gaan we immers naar één andere plaats en in iedere plaats komen we vanuit één andere plaats). Als we dus van een rij of een kolom hetzelfde getal aftrekken, dan krijgen we een equivalent probleem. Trekken we van iedere rij het kleinste element uit die rij af, dan krijgen we een nieuwe afstandenmatrix met in iedere rij minstens één 0. Daarna trekken we in deze gereduceerde matrix van iedere kolom het kleinste element af. Aldus ontstaat een volgende gereduceerde afstandenmatrix met in iedere rij en in iedere kolom minstens één 0. Het totaal dat we aldus hebben gereduceerd is een ondergrens voor de lengte van iedere rondreis.

Vervolgens gaan we bij iedere 0 noteren wat er extra gereduceerd kan worden als we deze 0 niet kiezen (d.w.z. we vervangen 0 door $\infty$ en kijken wat verder gereduceerd kan worden); dit heet het *reductiegetal*. Deze reductiegetallen leveren een criterium op voor het vertakken en wel als volgt.

1. Zoek de 0 met het grootste reductiegetal, zeg op plaats $(i, j)$.
2. Splits het huidige deelprobleem in de volgende twee deelprobelen:
a. voeg aan het oude deelprobleem toe dat $(i, j)$ wel in de route wordt opgenomen;
b. voeg aan het oude deelprobleem toe dat $(i, j)$ niet in de route wordt opgenomen.

[15] De term 'branch-and-bound' is geïntroduceerd door Little, Murty, Sweeney en Karel. Zie J.C.D. Little, K.G. Murty, D.W. Sweeney and C. Karel, *An algorithm for the travelling salesman problem*, Operations Research 11 (1963) 972–989.

De grenzen van de twee nieuwe deelproblemen worden als volgt bepaald:
In geval a: Maak de gereduceerde afstandenmatrix kleiner door rij $i$ en kolom $j$ weg te laten en kijk wat daardoor extra kan worden gereduceerd. Let er daarbij ook op dat subtours worden uitgesloten. De nieuwe grens wordt verkregen door bij de oude grens de extra reductie op te tellen.
In geval b: De nieuwe grens is de oude grens plus het reductiegetal van de 0 op plaats $(i, j)$.

**Voorbeeld 2.8**

$$L = \begin{pmatrix} - & 35 & 43 & 23 & 33 & 19 \\ 35 & - & 47 & 42 & 21 & 26 \\ 43 & 47 & - & 36 & 31 & 30 \\ 23 & 42 & 36 & - & 50 & 17 \\ 33 & 21 & 31 & 50 & - & 45 \\ 19 & 26 & 30 & 17 & 45 & - \end{pmatrix}$$

Reducties
eerste rij: 19
tweede rij: 21
derde rij: 30
vierde rij: 17
vijfde rij: 21
zesde rij: 17
eerste kolom: 2
derde kolom: 10

$$L' = \begin{pmatrix} - & 16 & 14 & 4 & 14 & 0 \\ 12 & - & 16 & 21 & 0 & 5 \\ 11 & 17 & - & 6 & 1 & 0 \\ 4 & 25 & 9 & - & 33 & 0 \\ 10 & 0 & 0 & 29 & - & 24 \\ 0 & 9 & 3 & 0 & 28 & - \end{pmatrix}$$

Totale reductie: 137

*Iteratie 1*

Vervolgens gaan we bij de 0'en de reductiegetallen bepalen. Kiezen we de 0 op plaats $(2, 5)$ niet, dan moeten we vanuit plaats 2 naar een andere plaats, wat minstens 5 kost; eveneens wordt plaats 5 uit een andere plaats dan plaats 2 bereikt, wat minstens 1 kost. Het reductiegetal bij deze 0 is dus $5 + 1 = 6$. Dit doen we voor iedere 0, wat het volgende tableau geeft (de reductiegetallen staan rechts boven de desbetreffende 0'en en linksboven staat de totale reductie).

| 137 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | 16 | 14 | 4 | 14 | $0^4$ |
| 2 | 12 | - | 16 | 21 | $0^6$ | 5 |
| 3 | 11 | 17 | - | 6 | 1 | $0^1$ |
| 4 | 4 | 25 | 9 | - | 33 | $0^4$ |
| 5 | 10 | $0^9$ | $0^3$ | 29 | - | 24 |
| 6 | $0^4$ | 9 | 3 | $0^4$ | 28 | - |

Het grootste reductiegetal is 9 bij het element $(5, 2)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_1$ : $(5, 2)$ behoort wel tot de tour; ondergrens is $137 + (3+5+1) = 146$ (immers: als 5-de rij en 2-de kolom worden weggelaten kan in 3-de kolom 3 worden gereduceerd; omdat $(2, 5)$ niet mag kan in 2-de rij 5 en in 5-de kolom 1 worden gereduceerd).
$P_2$ : $(5, 2)$ behoort niet tot de tour; ondergrens $137 + 9 = 146$.

*Iteratie 2*

Kies deelprobleem $P_1$. Na het weglaten van de 5-de rij en 2-de kolom en na het reduceren wordt de tabel:

| 146 | 1 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | - | 11 | 4 | 13 | $0^4$ |
| 2 | 7 | 8 | 16 | - | $0^7$ |
| 3 | 11 | - | 6 | $0^{13}$ | $0^0$ |
| 4 | 4 | 6 | - | 32 | $0^4$ |
| 6 | $0^4$ | $0^6$ | $0^4$ | 27 | - |

Het grootste reductiegetal is 13 bij het element $(3, 5)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_3$: voeg aan $P_1$ toe: $(3, 5)$ behoort wel tot de tour; ondergrens is 146 (ga zelf na dat geen verdere reductie mogelijk is en dat $(2, 3)$ niet is toegestaan).
$P_4$: voeg aan $P_1$ toe: $(3, 5)$ behoort niet tot de tour; ondergrens $146 + 13 = 159$.

*Iteratie 3*

Kies deelprobleem $P_3$. Na het weglaten van de 3-de rij en 5-de kolom en het verbieden van $(2,3)$ wordt de tabel:

| 146 | 1 | 3 | 4 | 6 |
|---|---|---|---|---|
| 1 | - | 11 | 4 | $0^4$ |
| 2 | 7 | - | 16 | $0^7$ |
| 4 | 4 | 6 | - | $0^4$ |
| 6 | $0^4$ | $0^6$ | $0^4$ | - |

Het grootste reductiegetal is 7 bij het element $(2,6)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_5$: voeg aan $P_3$ toe: $(2,6)$ behoort wel tot de tour; $(6,3)$ is verboden en de ondergrens is $146 + (6+4) = 156$ (ga dit zelf na).
$P_6$: voeg aan $P_3$ toe: $(2,6)$ behoort niet tot de tour; ondergrens $146 + 7 = 153$.

*Iteratie 4*

Kies deelprobleem $P_2$. Neem de eerste tabel en verbied $(5,2)$ en voer de reductie uit. Dit geeft de volgende tabel:

| 146 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | 7 | 14 | 4 | 14 | $0^4$ |
| 2 | 12 | - | 16 | 21 | $0^6$ | 5 |
| 3 | 11 | 8 | - | 6 | 1 | $0^1$ |
| 4 | 4 | 16 | 9 | - | 33 | $0^4$ |
| 5 | 10 | - | $0^{13}$ | 29 | - | 24 |
| 6 | $0^4$ | $0^7$ | 3 | $0^4$ | 28 | - |

Het grootste reductiegetal is 13 bij het element $(5,3)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_7$: voeg aan $P_2$ toe: $(5,3)$ behoort wel tot de tour; ondergrens is 146.
$P_8$: voeg aan $P_2$ toe: $(5,3)$ behoort niet tot de tour; ondergrens $146 + 13 = 159$.

*Iteratie 5*

Kies deelprobleem $P_7$. Na reductie wordt de tabel:

| 146 | 1 | 2 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 1 | - | 7 | 4 | 14 | $0^4$ |
| 2 | 12 | - | 21 | $0^{19}$ | 5 |
| 3 | 11 | 8 | 6 | - | $0^6$ |
| 4 | 4 | 16 | - | 33 | $0^4$ |
| 6 | $0^4$ | $0^7$ | $0^4$ | 28 | - |

Het grootste reductiegetal is 19 bij het element $(2,5)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_9$: voeg aan $P_7$ toe: $(2,5)$ behoort wel tot de tour; ondergrens is 146.
$P_{10}$: voeg aan $P_7$ toe: $(2,5)$ behoort niet tot de tour; ondergrens $146 + 19 = 165$.

*Iteratie 6*

Kies deelprobleem $P_9$. Na reductie wordt de tabel:

| 146 | 1 | 2 | 4 | 6 |
|---|---|---|---|---|
| 1 | - | 7 | 4 | $0^4$ |
| 3 | 11 | - | 6 | $0^6$ |
| 4 | 4 | 16 | - | $0^4$ |
| 6 | $0^4$ | $0^7$ | $0^4$ | - |

Het grootste reductiegetal is 7 bij het element $(6,2)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_{11}$: voeg aan $P_9$ toe: $(6,2)$ behoort wel tot de tour; ondergrens is $146 + (6+4) = 156$.
$P_{12}$: voeg aan $P_9$ toe: $(6,2)$ behoort niet tot de tour; ondergrens $146 + 7 = 153$.

*Iteratie 7*

Kies deelprobleem $P_{12}$. Na reductie wordt de tabel:

| 153 | 1 | 2 | 4 | 6 |
|---|---|---|---|---|
| 1 | - | $0^9$ | 4 | $0^0$ |
| 3 | 11 | - | 6 | $0^6$ |
| 4 | 4 | 9 | - | $0^4$ |
| 6 | $0^4$ | - | $0^4$ | - |

Het grootste reductiegetal is 9 bij het element $(1, 2)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_{13}$: voeg aan $P_{12}$ toe: $(1, 2)$ behoort wel tot de tour; ondergrens is 153.
$P_{14}$: voeg aan $P_{12}$ toe: $(1, 2)$ behoort niet tot de tour; ondergrens $153 + 9 = 162$.

*Iteratie 8*

Kies deelprobleem $P_{13}$. Na reductie wordt de tabel:

| 153 | 1 | 4 | 6 |
|---|---|---|---|
| 3 | - | 6 | 0 |
| 4 | 4 | - | 0 |
| 6 | 0 | 0 | - |

Tot nu toe hadden we al de verbindingen $(1, 2, 5, 3)$. Dit laat zich als volgt optimaal afmaken: $(1, 2, 5, 3, 6, 4, 1)$ met lengte 157. Hierdoor bevatten de volgende deelproblemen geen betere rondreis: $P_4, P_8, P_{10}$ en $P_{14}$. Over blijven dus nog de deelproblemen $P_5, P_6$ en $P_{11}$.

*Iteratie 9*

Kies deelprobleem $P_6$. De bijbehorende tabel is:

| 153 | 1 | 3 | 4 | 6 |
|---|---|---|---|---|
| 1 | - | 11 | 4 | $0^4$ |
| 2 | $0^9$ | - | 9 | - |
| 4 | 4 | 6 | - | $0^4$ |
| 6 | $0^0$ | $0^6$ | $0^4$ | - |

Het grootste reductiegetal is 9 bij het element $(2, 1)$. Dit geeft de splitsing in de volgende twee deelproblemen:
$P_{15}$: voeg aan $P_6$ toe: $(2, 1)$ behoort wel tot de tour; ondergrens is 153.
$P_{16}$: voeg aan $P_6$ toe: $(2, 1)$ behoort niet tot de tour; ondergrens $153 + 9 = 162$. Dit deelprobleem hoeft dus ook niet verder onderzocht te worden.

*Iteratie 10*

Kies deelprobleem $P_{15}$. Na reductie wordt de tabel:

| 153 | 3 | 4 | 6 |
|---|---|---|---|
| 1 | - | 4 | 0 |
| 4 | 6 | - | 0 |
| 6 | 0 | 0 | - |

Tot nu toe hadden we al de verbindingen $(3, 5, 2, 1)$. Dit laat zich als volgt optimaal afmaken: $(3, 5, 2, 1, 4, 6, 3)$ met lengte 157.

*Iteratie 11*

Kies deelprobleem $P_5$. Na reductie wordt de tabel:

| 156 | 1 | 3 | 4 |
|---|---|---|---|
| 1 | - | 1 | 0 |
| 4 | 4 | 0 | - |
| 6 | 0 | - | 0 |

Tot nu toe hadden we al de verbindingen $(3, 5, 2, 6)$. Dit laat zich als volgt optimaal afmaken: $(3, 5, 2, 6, 1, 4, 3)$ met lengte 156. Hierdoor heeft ook het laatste deelprobleem dat nog over is, namelijk $P_{11}$ met grens 156, geen betere oplossing en zijn we klaar.

Een optimale oplossing van het TSP is dus de rondreis $(3, 5, 2, 6, 1, 4, 3)$ met lengte 156.

**Vraag 2.8**

Los het TSP met branch-and-bound op voor de volgende afstandentabel:

$$L = \begin{pmatrix} - & 54 & 48 & 92 & 24 \\ 54 & - & 32 & 61 & 35 \\ 48 & 32 & - & 45 & 23 \\ 92 & 61 & 45 & - & 67 \\ 24 & 35 & 23 & 67 & - \end{pmatrix}.$$

### 2.4.3 Heuristieken

Een heuristiek is een algoritme dat een *benadering* geeft voor de exacte oplossing van een probleem. Heuristieken zijn zinvol als er is geen polynomiale methode beschikbaar om het probleem exact op te lossen. Een heuristiek moet bij voorkeur zelf is wel een polynomiaal algoritme zijn. Hoe dichter de benaderende oplossing bij de exacte oplossing ligt, des te beter is de heuristiek. De *kwaliteit* van heuristiek $H$, genoteerd met $k(H)$, meten we af aan de slechtste instantie voor de heuristiek: de zogenaamde *worst case analyse*. Laat $H(P)$ de waarde van de doelfunctie zijn als heuristiek $H$ op instantie $P$ wordt toegepast, en laat $O(P)$ het optimum zijn van instantie $P$. Bij minimaliseringsproblemen is dus altijd $H(P) \geq O(P)$. De kwaliteit $k(H)$ van heuristiek $H$ wordt als volgt gedefinieerd:

$$k(H) = sup_P \frac{H(P)}{O(P)}. \tag{2.4.6}$$

De kwaliteit van een heuristiek is dus minstens 1. Vaak kennen we $O(P)$ niet exact, maar hebben we wel een ondergrens voor $O(P)$, zeg $L(P)$. We kunnen dan $k(H)$ als volgt afschatten:

$$k(H) = sup_P \frac{H(P)}{O(P)} \leq sup_P \frac{H(P)}{L(P)}. \tag{2.4.7}$$

De volgende stelling, afkomstig van Sahni en Gonzalez,[16] laat zien dat de kwaliteit van iedere polynomiale heuristiek voor het TSP willekeurig slecht is. Dit geeft een indicatie dat het snel oplossen van het TSP zeer moeilijk, zo niet onmogelijk is.

**Stelling 2.5**

*Als $\mathcal{P} \neq \mathcal{NP}$, dan geldt dat voor iedere $c \geq 1$ er geen polynomiale heuristiek $H$ voor het TSP is met $k(H) \leq c$.*

**Bewijs**

Stel dat een dergelijke heuristiek $H$ wel bestaat, dan zullen we aantonen dat het Hamilton-kring probleem tot $\mathcal{P}$ behoort. Omdat het Hamilton-kring probleem tot $\mathcal{NPC}$ behoort, volgt daaruit dat $\mathcal{P} = \mathcal{NP}$, wat de gewenste tegenspraak oplevert.

Veronderstel dat we willen nagaan of de graaf $G = (V, E)$ met $n$ knooppunten een Hamilton-kring bevat. We construeren daartoe de volgende instantie $P$ van het TSP:

[16] S. Sahni and T. Gonzales, *P-complete approximation problems*, Journal of the ACM 23 (1976) 555–565.

$$l_{ij} = \begin{cases} 1 & \text{als } (v_i, v_j) \in E \\ c \cdot n + 1 & \text{anders} \end{cases}$$

Nu geldt: $G$ heeft een Hamilton-kring d.e.s.d. als het TSP een optimale oplossing heeft met lengte $n$. In dat geval is dus $O(P) = n$. Omdat $k(H) \leq c$, geldt dat $H(P) \leq c \cdot O(P) = c \cdot n$. Dus als $G$ een Hamilton-kring heeft, dan geeft $H$ een approximatie met lengte maximaal $c \cdot n$. Stel $G$ heeft geen Hamilton-kring. Dan heeft iedere oplossing van het TSP minimaal de lengte $(n-1) + (c \cdot n + 1) = (c+1) \cdot n > c \cdot n$.
Omdat $H$ polynomiaal is, geeft dit het volgende polynomiale algoritme voor het Hamilton-kring probleem: Pas $H$ toe op de bijbehorende instantie $P$ van het TSP en er is een Hamilton-kring d.e.s.d. als $H(P) \leq c \cdot n$. □

**Invoeg-heuristieken**

Bij invoeg-heuristieken zijn tijdens het algoritme de steden $\{1, 2, \ldots, n\}$ verdeeld in twee groepen, zeg $S$ en $T$. De steden van $S$ vormen een *subtour* en de steden van $T$ zijn nog niet opgenomen in de subtour. Als $S = \{i_1, i_2, \ldots, i_k\}$, dan hoort hierbij de subtour $\{i_1, i_2, \ldots, i_k, i_1\}$. In iedere iteratie wordt één element van $T$ naar $S$ overgeheveld. Bij de start nemen we $S = \{1\}$ en we stoppen als $S$ $n$ elementen bevat. Een invoeg-heuristiek heeft dus $n-1$ iteraties.

**Algoritme 2.3** *Generieke vorm van een invoeg-heuristiek*

1. $S = \{1\};\ T = \{2, 3, \ldots, n\};\ k = 1.$
2. Als $k < n$ en $S = \{i_1 = 1, i_2, \ldots, i_k\}$:
   (a) Kies een $j \in T;\ T := T - \{j\};$
   (b) Bepaal een $1 \leq l \leq k$, laat $S = \{i_1 = 1, i_2, \ldots, i_l, j, i_{l+1} \ldots, i_k\}$ en $k := k + 1$.
3. Als $k = n : S$ is de met de heuristiek verkregen approximatie (STOP).
   Anders: ga naar stap 2.

Het volgende algoritme kiest als nieuwe stad de stad het dichtst bij de laatst gekozen stad.

**Algoritme 2.4** *Naaste-buur heuristiek*

1. $S = \{1\};\ T = \{2, 3, \ldots, n\};\ k = 1.$
2. Als $k < n$ en $S = \{i_1 = 1, i_2, \ldots, i_k\}$:
   (a) Kies $j \in T$ zdd. $l_{i_k j} = min_{m \in T}\ l_{i_k m};\ T := T - \{j\};$
   (b) $S := \{i_1 = 1, i_2, \ldots, i_k, j\}$ en $k := k + 1$.
3. Als $k = n : S$ is de met de heuristiek verkregen approximatie (STOP).
   Anders: ga naar stap 2.

**Voorbeeld 2.9**

Neem de afstandsmatrix uit Voorbeeld 2.8.

$$L = \begin{pmatrix} - & 35 & 43 & 23 & 33 & 19 \\ 35 & - & 47 & 42 & 21 & 26 \\ 43 & 47 & - & 36 & 31 & 30 \\ 23 & 42 & 36 & - & 50 & 17 \\ 33 & 21 & 31 & 50 & - & 45 \\ 19 & 26 & 30 & 17 & 45 & - \end{pmatrix}$$

$S = \{1\}$
$j = 6;\ S = \{1, 6\}$
$j = 4;\ S = \{1, 6, 4\}$
$j = 3;\ S = \{1, 6, 4, 3\}$
$j = 5;\ S = \{1, 6, 4, 3, 5\}$
$j = 2;\ S = \{1, 6, 4, 3, 5, 2\}$
Approximatie $\{1, 6, 4, 3, 5, 2, 1\}$ met lengte 159.

Opmerkingen:

1. De complexiteit van dit algoritme is $\mathcal{O}(n^2)$ (ga dit zelf na).
2. Er kan worden aangetoond [17] dat, zelfs als aan de driehoeksongelijkheid is voldaan, $k(H) = \Theta(log\ n)$.

In het volgende algoritme wordt als nieuwe stad gekozen de stad die het dichtst bij een van de steden van $S$ ligt en deze wordt zo goed mogelijk ingevoegd tussen de huidige steden van $S$.

**Algoritme 2.5** *Dichtstbijzijnde-stad heuristiek*

1. $S = \{1\};\ T = \{2, 3, \ldots, n\};\ k = 1.$
2. Als $k < n$ en $S = \{i_1 = 1, i_2, \ldots, i_k\}$:
   (a) Bepaal voor alle $j \in T : l_{min}(j) = min_{i \in S}\ l_{ij}$;
   (b) Kies $j \in T$ zdd. $l_{min}(j) = min_{m \in T}\ l_{min}(m);\ T := T - \{j\}$;
   (c) Voeg $j$ aan $S$ toe op die plaats zdd. de subtour bij $S \cup \{j\}$ een minimale lengte heeft;
   (d) $k := k + 1$.
3. Als $k = n : S$ is de met de heuristiek verkregen approximatie (STOP).
   Anders: ga naar stap 2.

**Voorbeeld 2.10**

Neem weer de afstandsmatrix uit Voorbeeld 2.8.

$$L = \begin{pmatrix} - & 35 & 43 & 23 & 33 & 19 \\ 35 & - & 47 & 42 & 21 & 26 \\ 43 & 47 & - & 36 & 31 & 30 \\ 23 & 42 & 36 & - & 50 & 17 \\ 33 & 21 & 31 & 50 & - & 45 \\ 19 & 26 & 30 & 17 & 45 & - \end{pmatrix}$$

$S = \{1\}$
$j = 6;\ S = \{1, 6\}$
$j = 4;\ S = \{1, 6, 4\}$
$j = 2;\ S = \{1, 2, 6, 4\}$
$j = 5;\ S = \{1, 5, 2, 6, 4\}$
$j = 3;\ S = \{1, 3, 5, 2, 6, 4\}$
Approximatie $\{1, 3, 5, 2, 6, 4, 1\}$ met lengte 161.

[17] D.J. Rozenkrantz, R.E. Steans and P.M. Lewis, "An analysis of several heuristics for the traveling salesman problem", SIAM Journal on Computing 6 (1977) 563-581.

Opmerkingen:

1. Het algoritme is zo te implementeren dat de complexiteit van dit algoritme $\mathcal{O}(n^2)$ is (zie Opgave 2.14).
2. Er kan worden aangetoond [18] dat, indien aan de driehoeksongelijkheid is voldaan, $k(H) = 2$.

In het volgende algoritme wordt als nieuwe stad gekozen de stad die het verst bij een van de steden van $S$ vandaan ligt en deze wordt zo goed mogelijk ingevoegd. Op het eerste gezicht lijkt dit misschien wat vreemd. Echter, omdat alle steden in de tour moeten komen, is het niet onlogisch om eerst de steden die ver weg liggen in de route op te nemen.

**Algoritme 2.6** *Verste-stad heuristiek*

1. $S = \{1\};\ T = \{2, 3, \ldots, n\};\ k = 1.$
2. Als $k < n$ en $S = \{i_1 = 1, i_2, \ldots, i_k\}$:
   (a) Bepaal voor alle $j \in T : l_{min}(j) = min_{i \in S}\ l_{ij}$;
   (b) Kies $j \in T$ zdd. $l_{min}(j) = max_{m \in T}\ l_{min}(m);\ T := T - \{j\}$;
   (c) Voeg $j$ aan $S$ toe op die plaats zdd. de subtour bij $S \cup \{j\}$ een minimale lengte heeft;
   (d) $k := k + 1$.
3. Als $k = n : S$ is de met de heuristiek verkregen approximatie (STOP).
   Anders: ga naar stap 2.

Opmerkingen:

1. Het algoritme is zo te implementeren dat de complexiteit van dit algoritme is $\mathcal{O}(n^2)$ is (zie Opgave 2.14).
2. Er kan worden aangetoond [19] dat, zelfs als de driehoeksongelijkheid geldt, $k(H) \geq \frac{13}{2}$.

In de laatste invoeg-heuristiek wordt als nieuwe stad die stad gekozen die, na zo goed mogelijk ingevoegen tussen de huidige steden van $S$, de kortste subtour oplevert.

**Algoritme 2.7** *Beste-invoeg heuristiek*

1. $S = \{1\};\ T = \{2, 3, \ldots, n\};\ k = 1.$
2. Als $k < n$ en $S = \{i_1 = 1, i_2, \ldots, i_k\}$:
   (a) Bepaal voor alle $j \in T : b_{min}(j) =$ toename van de lengte van de subtour als $j$ zo goed mogelijk wordt ingepast.

[18] D.J. Rozenkrantz, R.E. Steans and P.M. Lewis, "An analysis of several heuristics for the traveling salesman problem", SIAM Journal on Computing 6 (1977) 563-581.

[19] C.A.J. Hurkens, "Nasty TSP instances for classical insertion heuristics", Report University of Technology, Eindhoven (1991).

(b) Kies $j \in T$ zdd. $b_{min}(j) = min_{m \in T}\ b_{min}(m)$; $T := T - \{j\}$;

(c) Voeg $j$ aan $S$ toe op die plaats zdd. de subtour bij $S \cup \{j\}$ een minimale lengte heeft;

(d) $k := k + 1$.

3. Als $k = n : S$ is de met de heuristiek verkregen approximatie (STOP).

   Anders: ga naar stap 2.

Opmerkingen:

1. De bepaling van $b_{min}(j)$ heeft complexiteit $\mathcal{O}(n^2)$ (ga dit zelf na). Daardoor heeft het algoritme complexiteit $\mathcal{O}(n^3)$.

2. De kwaliteit van dit algoritme is dezelfde als van algoritme 2.5, d.w.z. $k(H) = 2$ als aan de driehoeksongelijkheid is voldaan.[20]

### Heuristieken met een kwaliteitsgarantie

Er bestaan ook heuristieken die een garantie voor de kwaliteit geven. We zullen hiervan twee voorbeelden laten zien. We nemen, behalve de *symmetrie*, ook aan dat de *driehoeksongelijkheid* geldt: $l_{ij} \leq l_{ik} + l_{kj}$ voor alle $i, j$ en $k$. In deze heuristieken wordt eerst een *minimale opspannende boom*, zeg $T$, geconstrueerd. Dit kan met een $\mathcal{O}(n^2)$-algoritme.[21]

De eerste heuristiek doet daarna het volgende: dupliceer alle takken van $T$ tot $S$, waardoor een *Euler graaf* ontstaat, zodat alle takken van $S$ tezamen een kring vormen, zeg kring $C$; vervolgens doorlopen we $C$ vanuit stad 1, waarbij als een stad $j$ voor de tweede keer wordt bezocht, zeg via $\{i, j, k\}$, de takken $(i, j)$ en $(j, k)$ worden vervangen door één tak $(i, k)$. Op deze manier snijden we delen van $C$ af. Door de driehoeksongelijkheid wordt de nieuwe ronde daardoor niet groter.

**Algoritme 2.8** *Boom verdubbeling heuristiek*

1. Construeer een minimale opspannende boom $T$.

2. Laat $S$ bestaan uit de verdubbeling van de takken van $T$.

3. Construeer uit $S$ één grote kring $C$.

4. Doorloop $C$ vanuit stad 1 en als een stad $j$ voor de tweede keer wordt bezocht, zeg via $\{i, j, k\}$, dan worden de takken $(i, j)$ en $(j, k)$ vervangen door de tak $(i, k)$. Dit geeft een rondreis $C^*$.

[20] Ook dit resultaat kan worden gevonden in het artikel van Rozenkrantz, Steans en Lewis: D.J. Rozenkrantz, R.E. Steans and P.M. Lewis, "An analysis of several heuristics for the traveling salesman problem", SIAM Journal on Computing 6 (1977) 563-581.

[21] Dit kan met de methode van Prim of de methode van Kruskal. Zie hiervoor Besliskunde 1.

**Voorbeeld 2.11**

Neem weer de afstandsmatrix uit Voorbeeld 2.8.

1. $T = \{(4,6),(1,6),(2,5),(2,6),(3,6)\}$ met lengte $l(T) = 113$.
2. $S = \{(4,6),(1,6),(2,5),(2,6),(3,6),(4,6),(1,6),(2,5),(2,6),(3,6)\}$.
3. $C = \{1,6,2,5,2,6,3,6,4,6,1\}$ met $l(C) = 226$.
4. $C^* = \{1,6,2,5,3,4,1\}$ met $l(C^*) = 156$.

**Stelling 2.6**

*De boom verdubbeling heuristiek heeft complexiteit $\mathcal{O}(n^2)$ en de kwaliteit is 2.*

**Bewijs**

De complexiteit van stap 1 is $\mathcal{O}(n^2)$. Stap 2, stap 3 en stap 4 hebben ieder als complexiteit de orde van het aantal takken van $T$, d.w.z. complexiteit $\mathcal{O}(n)$. De totale complexiteit van de heuristiek is dus $\mathcal{O}(n^2)$.

Zij $C_{opt}$ de optimale rondreis. Door uit $C_{opt}$ één tak weg te laten ontstaat een boom, dus is $l(T) \leq l(C_{opt})$. Omdat door het afsnijden in stap 4 de lengte ook niet toeneemt, kunnen we schrijven:

$$l(C^*) \leq l(C) = l(S) = 2 \cdot l(T) \leq 2 \cdot l(C_{opt}),$$

waaruit volgt dat de kwaliteit van de heuristiek hoogstens 2 is.

Beschouw vervolgens een graaf met $2n$ knooppunten die liggen op twee concentrische cirkels met straal $R$ en straal $R+r$, respectievelijk. Op de grote cirkel liggen de $n$ knooppunten $v_1, v_2, \ldots, v_n$, die op gelijke afstand van elkaar liggen, dus op afstand $\frac{2\pi(R+r)}{n}$.

De knooppunten op de binnenste cirkel, $w_1, w_2, \ldots, w_n$, liggen op de lijn van het centrum naar de knooppunten op de buitenste cirkel, dus op afstand $\frac{2\pi R}{n}$ van elkaar.

Deze graaf heeft als takken de cirkelranden en de verbindingen $(v_i, w_i), 1 \leq i \leq n$. Deze laatste hebben de lengte $r$, en zijn de kleinste takken als $r < \frac{2\pi R}{n}$.

We nemen nu $R = 1$ en $r = \frac{1}{n^2}$. Dan wordt de minimale opspannende boom:

$$T = \{(v_i, w_i),\ 1 \leq i \leq n;\ (w_i, w_{i+1}),\ 1 \leq i \leq n-1\} \text{ met } l(T) = n \cdot r + (n-1) \cdot \tfrac{2\pi R}{n}.$$

Dupliceren we de takken van $T$ tot $S$, dan ontstaat de kring

$$C = \{v_1, w_1, w_2, \ldots, w_n, v_n, w_n, w_{n-1}, v_{n-1}, w_{n-1}, \ldots, w_2, v_2, w_2, w_1, v_1\}.$$

Door af te snijden ontstaat hieruit de rondreis $C^* = \{v_1, w_1, w_2, \ldots, w_n, v_n, v_{n-1}, \ldots, v_2, v_1\}$ met

$$l(C^*) = 2r + (n-1)\left\{\tfrac{2\pi(R+r)}{n} + \tfrac{2\pi R}{n}\right\} = 2r + (n-1)\left\{\tfrac{4\pi R}{n} + \tfrac{2\pi r}{n}\right\} \approx 4\pi$$

voor grote waarden van $n$. De optimale rondreis is

$$C_{opt} = \{v_1, w_1, w_2, v_2, v_3, w_3, \ldots, w_n, v_n, v_1\} \text{ met } l(C_{opt}) = n \cdot r + \tfrac{n}{2} \cdot \left\{\tfrac{2\pi(R+r)}{n} + \tfrac{2\pi R}{n}\right\} \approx 2\pi$$

voor grote waarden van $n$. Hieruit volgt dat de kwaliteit inderdaad 2 is. □

De tweede heuristiek doet het volgende, na eerst een minimale opspannende boom $T$ bepaald te hebben. Bekijk welke punten van $T$ een oneven graad hebben (dit aantal punten is even). Voor deze punten wordt een *minimale volmaakte koppeling* $M$ geconstrueerd, d.w.z. dat deze knooppuntenverz. in disjuncte tweetallen wordt verdeeld, zeg $(i_1, i_2), (i_3, i_4), \ldots, (i_{2k-1}, i_{2k})$, zdd. de som van de lengtes van verbindingen van deze tweetallen, d.w.z. $\sum_{j=1,3,\ldots,2k-1} l_{i_j,i_{j+1}}$, minimaal is.[22] Door deze koppeling $M$ aan $T$ toe te voegen verkrijgen we weer dezelfde eigenschap als in het vorige algoritme, namelijk dat iedere stad incident is met een even aantal verbindingen: de bijbehorende graaf is Eulers en de bijbehorende takken vormen tezamen één kring. Dit leidt tot het volgende algoritme.[23]

**Algoritme 2.9** *Boom en koppeling heuristiek*

1. Construeer een minimale opspannende boom $T$ en laat $W$ de verz. zijn van de steden die in $T$ incident zijn met een oneven aantal verbindingen.

2. Construeer op $W$ een minimale volmaakte koppeling $M$.

3. Construeer uit $T \cup M$ één grote kring $C$.

4. Doorloop $C$ vanuit stad 1 en als een stad $j$ voor de tweede keer wordt bezocht, zeg via $\{i, j, k\}$, dan worden de takken $(i, j)$ en $(j, k)$ vervangen door de tak $(i, k)$. Dit geeft een rondreis $C^*$.

**Stelling 2.7**
*De boom en koppeling heuristiek heeft complexiteit $\mathcal{O}(n^3)$ en de kwaliteit is gelijk aan $\frac{3}{2}$.*

**Bewijs**
De complexiteit van stap 1 is $\mathcal{O}(n^2)$, van stap 2 $\mathcal{O}(n^3)$ en van de stappen 3 en 4 $\mathcal{O}(n)$. De totale complexiteit van de heuristiek is dus $\mathcal{O}(n^3)$.
De lengte van $T$ is weer hoogstens de lengte van de optimale tour, zeg deze is $C_{opt}$, en het afsnijden maakt de route ook nooit groter: $l(C^*) \leq l(C) = l(T) + l(M) \leq l(C_{opt}) + l(M)$.
Beschouw de optimale tour en beperk deze door 'af te snijden' tot de steden van $W$, zeg dit geeft de route $C' = [i_1, i_2, \ldots, i_{2k}, i_1]$. Door het afsnijden wordt de lengte niet groter. Beschouw op $C'$ de volmaakte koppelingen

$$M_1 = \{(i_1, i_2), (i_3, i_4), \ldots, (i_{2k-1}, i_{2k})\} \text{ en } M_2 = \{(i_2, i_3), (i_4, i_5), \ldots, (i_{2k}, i_1)\}.$$

[22] Er bestaat een $\mathcal{O}(n^3)$ algoritme om een minimale volmaakte koppeling te bepalen; dit wordt in een ander college behandeld

[23] Dit algoritme is afkomstig van Christofides: N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976. Dit algroritme is het - in de *worst-case* - beste polynomiale algoritme dat bekend is. Als alle lengtes 1 of 2 zijn, dan is er een polynomiaal algoritme met kwaliteit $\frac{7}{6}$: C.H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research 18 (1993) 1–12.

Omdat $M$ een minimale volmaakte koppeling is, geldt

$$l(C_{opt}) \geq l(C') = l(M_1) + l(M_2) \geq 2 \cdot l(M).$$

Dit geeft $l(C^*) \leq l(C_{opt}) + l(M) \leq \frac{3}{2} \cdot l(C_{opt})$, waaruit volgt dat de kwaliteit van de heuristiek hoogstens $\frac{3}{2}$ is.
Beschouw het volgende voorbeeld waarin de knooppunten $v_1, v_2, \ldots, v_{n+1}$ op een rechte lijn liggen, op afstand 1 van elkaar, met daarboven de punten $w_1, w_2, \ldots, w_n$ zdd. de driehoeken $\{v_i, w_i, v_{i+1}\}, i = 1, 2, \ldots n$, gelijkzijdige driehoeken zijn met de lengte van de zijden gelijk aan 1.
De graaf is volledig en de lengte van de takken is de Euclidische afstand tussen de eindpunten.
De minimale opspannende boom is $T = \{(v_i, w_i),\ i = 1, 2, \ldots n;\ (w_i, v_{i+1}),\ i = 1, 2, \ldots n\}$ met $l(T) = 2n$ en met alleen de knooppunten $v_1$ en $v_{n+1}$ als knooppunten van oneven graad.
Dus $M = \{(v_1, v_{n+1})\}$ met $l(M) = n$. Dan is $C = \{v_1, w_1, v_2, w_2, \ldots, v_n, w_n, v_{n+1}, v_1\}$ met $l(C) = 2n + n = 3n$. We zien direct in dat $C^* = C$, dus $l(C^*) = l(C) = 3n$.
Het is eenvoudig in te zien dat de optimale tour $C_{opt} = \{v_1, w_1, w_2, \ldots, w_n, v_{n+1}, v_n, \ldots, v_2, v_1\}$ is met $l(C_{opt}) = 2n + 1$. Omdat

$$lim_{n\to\infty} \frac{C^*}{C_{opt}} = \frac{3}{2},$$

is de kwaliteit van de heuristiek dus minstens $\frac{3}{2}$. Gecombineerd met de eerdere constatering dat de kwaliteit hoogstens $\frac{3}{2}$ is, is nu bewezen dat de kwaliteit gelijk is aan $\frac{3}{2}$. □

**Methoden van Lokaal Zoeken**

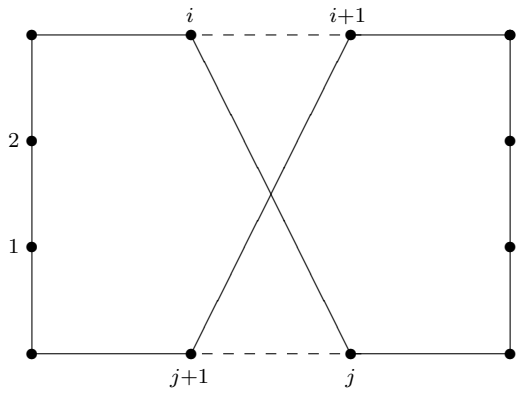
Lokale zoekmethoden werken in het algemeen als volgt. Voor iedere toegelaten oplossing $x$ is er een *omgeving* $N(x)$. Dit is een deelverz. van de toegelaten oplossingen met $x \in N(x)$. Tijdens een iteratie wordt $x$ vervangen door een $y \in N(x)$, waarna $y$ de rol van $x$ overneemt.
Een lokale zoekmethode bestaat uit de volgende elementen:
1. Een startoplossing.
2. Een definitie van het begrip 'omgeving'.
3. Een regel die aangeeft hoe uit $x$ een $y \in N(x)$ wordt verkregen.
4. Een stopcriterium.

**De heuristiek $r$-opt voor het TSP**
Deze heuristiek veronderstelt dat het TSP symmetrisch is. De heuristiek is nu als volgt:
1. Startoplossing: kies een willekeurige tour $C$ langs de steden.
2. Laat $\mathcal{S}$ de collectie zijn bestaande uit de verzamelingen van $r$ takken van $C$.
3. Voor iedere S $\in \mathcal{S}$: voor alle rondes $C'$ die mogelijk zijn door de $n - r$ takken van $C \backslash$S uit te breiden tot een goede tour: als $l(C') < l(C)$: $C := C'$ en ga naar stap 2.
4. Als in stap 3 geen verbetering wordt gevonden, dan stoppen we.

Opmerking:
De omgeving van een tour $C$ bestaat dus uit de rondreizen die mogelijk zijn door $r$ takken weg te laten en de ketens die zo ontstaan op de een of andere wijze tot een tour te verbinden.

Het meest gebruikelijk zijn de versies met $r = 2$ of $r = 3$. We zullen eerst 2-opt bespreken.

Veronderstel dat we uitgaan van de tour $\{1, 2, \ldots, n, 1\}$. In 2-opt heeft het alleen zin om twee niet-aangrenzende verbindingen, zeg $(i, i+1)$ en $(j, j+1)$ te verwijderen. De enige andere verbinding van deze twee ketens tot een tour die langs alle steden loopt is de route $\{1, 2, \ldots, i, j, j-1, \ldots, i+1, j+1, j+2, \ldots, n, 1\}$, waarbij in plaats van de lengtes $l_{i,i+1}$ en $l_{j,j+1}$ nu de lengtes $l_{i,j}$ en $l_{i+1,j+1}$ worden gebruikt. De nieuwe tour is dus beter indien geldt dat $l_{i,j} + l_{i+1,j+1} < l_{i,i+1} + l_{j,j+1}$.




**Algoritme 2.10** *2-opt heuristiek*

1. Kies een willekeurige rondreis $C$, zeg $C = \{1, 2, \ldots, n, 1\}$.

2. Voer het volgende uit totdat het niet meer tot een verbetering leidt:

   Voor $i = 1, 2, \ldots, n-2$:

   Voor $j = i+2, i+3, \ldots, n$ (als $i = 1$ gaan we niet verder dan $j = n-1$):

   Als $l_{i,j} + l_{i+1,j+1} < l_{i,i+1} + l_{j,j+1}$: ga naar stap 3 (als $j = n$, dan $j + 1 \equiv 1$).

3. $C := \{1, 2, \ldots, i, j, j-1, \ldots, i+1, j+1, j+2, \ldots, n, 1\}$ en ga naar stap 2

   (let daarbij op 'hernummering' tot $\{1, 2, \ldots, n\}$ zoals in stap 1).

Opmerking

In stap 2 van het algoritme worden maximaal $n - 3 + \sum_{i=2}^{n-2} (n - i - 1) = \frac{1}{2}n(n-3)$ alternatieve routes onderzocht (de eerste term $n - 3$ hoort bij $i = 1$). Per stap van het algoritme is de complexiteit dus $\mathcal{O}(n^2)$. Het aantal stappen hoeft echter niet polynomiaal te zijn. In deze zin is het theoretisch gezien geen efficiënt algoritme. In de praktijk blijkt het aantal iteraties meestal vrij gering. De $r$-opt heuristieken zijn al in 1965 door Lin voorgesteld.[24] Het heeft lang geduurd voordat kwaliteitsresultaten bekend werden. In 1999 hebben Chandra, Karloff en Tovey [25] voor de kwaliteit van 2-opt in het geval dat de driehoeksongelijkheid geldt het volgende aangetoond:

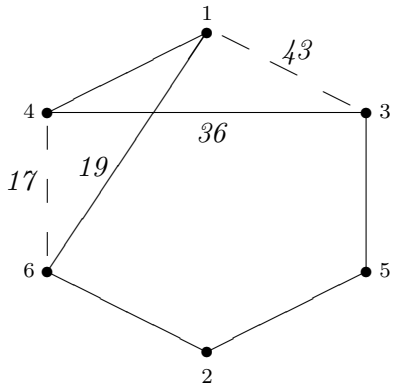
$$k(2\text{-}opt) \leq 4\sqrt{n} \text{ voor alle } n \text{ en } k(2\text{-}opt) \geq \tfrac{1}{4}\sqrt{n} \text{ voor oneindig veel waarden van } n.$$

[24] S. Lin, *Computer solutions of the traveling salesman problem*, Bell Systems Technology Journal 44 (1965) 2245–2269.

[25] B. Chandra, H. Karloff and C. Tovey, *New results on the old k-opt algorithm for the traveling salesman problem*, SIAM Journal on Computing 28 (1999) 1998–2029.

**Voorbeeld 2.12**

Neem weer de afstandsmatrix uit Voorbeeld 2.8.
Start met $\{1,3,5,2,6,4,1\}$. Deze rondreis heeft lengte 161.
Voor $i=1$ en $j=6$ krijgen we:
$l_{16}+l_{34}=19+36=55<l_{13}+l_{64}=43+17=60$.
Dit geeft de nieuwe rondreis $\{1,6,2,5,3,4,1\}$ met lengte 156.
Het blijkt dat deze rondreis niet meer verbeterd wordt met 2-opt.




Bij 3-*opt* worden drie takken verwijderd. Het aantal manieren om drie takken uit $n$ takken te kiezen is $\binom{n}{3}=\frac{1}{6}n(n-1)(n-2)$ en er zijn 8 manieren om de ketens met elkaar te verbinden (zie Vraag 2.11). Het algoritme voor 3-opt is wat gecompliceerder dan van 2-opt, maar een iteratiestp is nog altijd polynomiaal, namelijk $\mathcal{O}(n^3)$.

**Algoritme 2.11** *3-opt heuristiek*

1. Kies een willekeurige rondreis $C$.
2. Voer het volgende uit totdat het niet meer tot een verbetering leidt:
   (a) Kies drie takken van $C$, laat deze weg, waardoor er ketens ontstaan.
   (b) Ga voor alle mogelijke manieren om de rondreis te verbinden na wat de manier is die de kortste rondreis oplevert, zeg $C'$.
3. $C:=C'$ en ga naar stap 2.

Het is niet direct duidelijk welke waarde van $r$ ($r=2$, $r=3$ of een andere $r$) het beste gekozen kan worden; $r$ noemen we de *diepte* van de methode. Experimenten hebben uitgewezen dat 3-opt veel betere resultaten geeft dan 2-opt, maar dat 4-opt nauwelijks verbeteringen geeft i.v.m. 3-opt. Lin en Kernighan[26] hebben een algoritme met een dynamische diepte opgesteld. Dit algoritme blijkt in de praktijk goed te voldoen, maar er zijn geen theoretische resultaten van bekend.
We gaan weer uit van een rondreis $C$. Het idee er achter is dat als we een aantal takken weglaten en de ketens die daardoor ontstaan op een andere manier met elkaar verbinden, de keten die bestaat uit de weggelaten en de nieuwe takken een keten is waarvan de takken om en om wel en niet tot $C$ behoren.
Een keten $P=\{v_0,v_1,\ldots,v_{2p}\}$ heet een *wisselketen* (m.b.t. $C$) als de takken van $P$ afwisselend wel en niet tot $C$ behoren. Dus $(v_i,v_{i+1})$ behoort tot $C$ d.e.s.d. als $i$ even is. Als $v_0=v_{2p}$, dan heet de wisselketen *gesloten*. De keten hoeft niet enkelvoudig te zijn, dus knooppunten mogen meer dan één keer voorkomen in $P$. Takken mogen in een keten hoogstens één keer voorkomen,

[26] S. Lin and B.W. Kernighan, *An effective heuristic algorithm for the traveling salesman problem*, Operations Research 21 (1973) 498–516.

dus omdat $C$ een rondreis is en vanwege de eis dat de takken afwisselend wel en niet tot $C$ behoren komt een knooppunt in $P$ hoogstens twee keer voor.
Laat $P$ een gesloten wisselketen zijn. Beschouw het symmetrische verschil

$$C' = C \oplus P = C \cup P - C \cap P$$

(de vereniging minus de doorsnede). De lengte voldoet aan:

$$l(C') = l(C) - g(P), \text{ waarbij } g(P) = \sum_{i=0}^{p-1} \{l_{2i,2i+1} - l_{2i+1,2i+2}\}$$

(ga dit zelf na). $P$ heet *bruikbaar* als $C'$ een rondreis is en $g(P) > 0$, want dan is $C'$ een betere rondreis dan $C$. De wisselketen heet *positief* als $g(P_k) > 0$ voor $k = 1, 2, \ldots, p$, waarbij

$$g(P_k) = \sum_{i=0}^{k-1} \{l_{2i,2i+1} - l_{2i+1,2i+2}\}.$$

Een bruikbare wisselketen is altijd op te vatten als een positieve wisselketen door het beginpunt geschikt te kiezen, begin namelijk in $v_{2k}$, met $k$ de grootste index waarvoor $g(P_k)$ minimaal is, immers:
Voor $i = k+1, k+1, \ldots, p$:
$g(\{v_{2k}, v_{2k+1}, \ldots, v_{2i}\}) = g(P_i) - g(P_k) > 0.$
Voor $i = 1, 2, \ldots, k$:

$$\begin{aligned} g(\{v_{2k}, v_{2k+1}, \ldots, v_{2p} = v_0, v_1, \ldots, v_{2i}\}) &= g(\{v_{2k}, v_{2k+1}, \ldots, v_{2p}\}) + g(\{v_0, v_1, \ldots, v_{2i}\}) \\ &\geq g(\{v_{2k}, v_{2k+1}, \ldots, v_{2p}\}) + g(\{v_0, v_1, \ldots, v_{2k}\}) \\ &= g(P) > 0. \end{aligned}$$

In het algoritme van Lin en Kernighan wordt gezocht naar een positieve gesloten wisselketen (m.b.t. de huidige rondreis) met lengte 4 of 6. Het algoritme stopt als zo'n keten niet bestaat. Het eindigt dus met een tour die 3-opt is. Het geeft echter minder werk dan eerst 2-opt en daarna 3-opt toe te passen, want het combineert op een handige manier 2-opt en 3-opt. We geven hieronder het algoritme, gevolgd door een voorbeeld. Het algoritme begin in een mogelijk knooppunt en zoekt vandaar naar de beste positieve gesloten rondreis ($g^*$ houdt de beste verbetering bij en $P^*$ is de beste positieve gesloten rondreis). Als er geen verbetering (meer) is, dan wordt een volgend beginpunt gekozen. Het karakter van het algoritme is backtracking met index $i$ de diepte van het zoeken vanuit $v_0$.

**Algoritme 2.12** *Lin-Kernighan heuristiek*

1. Kies een willekeurige rondreis $C$ (tijdens het algoritme is $P_i = \{v_0, v_1, \ldots, v_i\}$).
2. $V_0 = \{1, 2, \ldots, n\}$; $i = 0$; $g^* = 0$.
3. Als $V_i = \emptyset$ en $g^* > 0$: $C := C \oplus P^*$ en ga naar stap 2.
   Als $V_i = \emptyset$ en $g^* = 0$:
   als $i = 0$: stop;
   anders: $i := min(i - 1, 5)$ en ga naar stap 3.

4. Kies $v_i \in V_i;\ V_i := V_i \backslash v_i$.

   Als $i \geq 3$ en oneven is, $C \oplus P$ een rondreis is met $P = \{P_i, v_0\}$ en $g(P) > g^*$:

   $P^* := P$, $g^* := g(P)$ en ga naar stap 5.

5. Als $i$ oneven is: $V_{i+1} = \{v_j \neq v_0 \mid (v_i, v_j) \notin C \cup P_{i-1};\ g(\{P_i, v_j\}) > g^*\}$.

   Als $i$ even is en $i \leq 2$:

   $V_{i+1} = \{v_j \neq v_0 \mid (v_i, v_j) \in C \backslash P_i\}$.

   Als $i$ even is en $i > 2$:

   $V_{i+1} = \{v_j \neq v_0 \mid (v_i, v_j) \in C \backslash P_i;\ (v_0, v_j) \notin C \cup P_i;\ C \oplus \{P_i, v_j, v_0\}$ is een rondreis$\}$.

   $i := i + 1$ en ga naar stap 3.

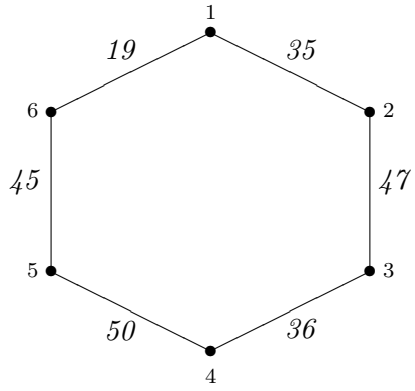
Toelichting

$V_i$ zijn de knooppunten die in aanmerking voor $v_i$ om $P_{i-1} = \{v_0, v_1, \ldots, v_{i-1}\}$ te verlengen tot $P_i$.
In stap 3 van het algoritme betekent $V_i = \emptyset$ dat de wisselketen niet verder kan worden uitgebreid. Als er een verbetering is gevonden ($g^* > 0$), dan wordt de beste verbetering doorgevoerd; als er geen verbetering is gevonden($g^* = 0$), dan wordt $i$ verlaagd ($i$ hoeft niet groter te zijn dan 5 voor 3-opt) en als geen verlaging mogelijk is ($i = 0$), dan stoppen we.
In stap 4 kijken we of voor oneven $i$ de gesloten wisselketen $P_i \cup (v_i, v_0)$ een rondreis oplevert en zo ja, of die een betere verbetering geeft dan de huidige; als dit ook zo is, dan voeren we deze verbetering door als kandidaat voor de beste verbetering.
In stap 5 maken we onderscheid tussen oneven en even waarden van $i$.
Als $i$ oneven is, dan kan de keten uitgebreid worden met een tak die nog niet in de wisselketen is opgenomen, niet tot $C$ behoort en die een positievere keten oplevert.
Als $i$ even is, dan maken we onderscheid tussen $i = 0$ of 2 enerzijds en $i \geq 4$ anderzijds. In het eerste geval is de keten nog niet lang genoeg en breiden we de keten uit met een nog niet gekozen tak van $C$. In het geval dat $i \geq 4$ is, dan breiden we de keten uit met een tak van $C$ die nog niet eerder is gekozen, die niet naar $v_0$ loopt, maar zdd. is dat $\{P_i, v_j, v_0\}$ wel een rondreis vormt (die dan lang genoeg is).

**Voorbeeld 2.13**

Start met $C = \{1, 2, 3, 4, 5, 6, 1\};\ l(C) = 232$. We voeren 3 iteraties uit.

*Iteratie 1*




$V_0 = \{1, 2, 3, 4, 5, 6\};\ i = 0;\ g^* = 0$.

$v_0 = 1;\ V_0 = \{2, 3, 4, 5, 6\};\ V_1 = \{2, 6\};\ i = 1$.

$v_1 = 2;\ V_1 = \{6\};\ V_2 = \{5, 6\};\ i = 2$.

$v_2 = 5;\ V_2 = \{6\};\ V_3 = \{4, 6\};\ i = 3$.

$v_3 = 4;\ V_3 = \{6\};\ P^* = \{1, 2, 5, 4, 1\};\ g^* = 41;\ V_4 = \{6\};\ i = 4$.

$v_4 = 6;\ V_4 = \emptyset;\ V_5 = \emptyset;\ i = 5$.

$C = \{1, 4, 3, 2, 5, 6, 1\}$ met $l(C) = 191$.

*Iteratie 2*

$V_0 = \{1,2,3,4,5,6\};\ i = 0;\ g^* = 0.$

$v_0 = 1;\ V_0 = \{2,3,4,5,6\};\ V_1 = \{4,6\};\ i = 1.$

$v_1 = 4;\ V_1 = \{6\};\ V_2 = \{6\};\ i = 2.$

$v_2 = 6;\ V_2 = \emptyset;\ V_3 = \{5\};\ i = 3.$

$v_3 = 5;\ V_3 = \emptyset:\ P^* = \{1,4,6,5,1\};\ g^* = 18;\ V_4 = \{3\};\ i = 4.$

$v_4 = 3;\ V_4 = \emptyset;\ V_5 = \{2\};\ i = 5.$

$v_5 = 2;\ V_5 = \emptyset;\ P^* = \{1,4,6,5,3,2,1\};\ g^* = 32;\ V_6 = \{6\};\ i = 6.$

$v_6 = 6;\ V_6 = \emptyset;\ V_7 = \emptyset;\ i = 7;\ C = \{1,2,5,3,4,6,1\}$ met $l(C) = 159.$

*Iteratie 3*

$V_0 = \{1,2,3,4,5,6\};\ i = 0;\ g^* = 0.$

$v_0 = 1;\ V_0 = \{2,3,4,5,6\};\ V_1 = \{2,6\};\ i = 1.$

$v_1 = 2;\ V_1 = \{6\};\ V_2 = \{6\};\ i = 2.$

$v_2 = 6;\ V_2 = \emptyset;\ V_3 = \{4\};\ i = 3.$

$v_3 = 4;\ V_3 = \emptyset:\ P^* = \{1,2,6,4,1\};\ g^* = 3;\ V_6 = \emptyset;\ i = 4.$
$C = \{1,4,3,5,2,6,1\}$ met $l(C) = 156.$

### Meer recente heuristieken

De meer recente heuristieken hebben gemeen dat ze trachten een lokaal minimum te vermijden. Een nieuw ingrediënt hiervoor is dat er een *stochastisch element* in de zoekprocedure wordt gebracht, op grond waarvan wordt toegestaan dat er ook rondreizen worden onderzocht met een grotere lengte dan de tot nu toe best bekende. We doen dit in de hoop via deze (tijdelijke) slechtere oplossingen later een betere te vinden en daardoor te ontsnappen aan een lokaal, maar geen globaal, optimum. We zullen hieronder de principes van een drietal methoden bespreken. Dit zijn meta-heuristieken: er zijn allerlei varianten van te bedenken door bepaalde parameters te specificeren.[27]

#### 1. Simulated annealing

Simulated annealing, ook wel *stochastisch koelen* genoemd, is een flexibele methode. Het algoritme kan worden uitgelegd aan de hand van de volgende analogie uit de *thermodynamica*. Als een smid een stuk metaal afkoelt zal hij dit langzaam doen, omdat hij weet dat dat een beter resultaat oplevert. Dit wordt veroorzaakt door het feit dat het systeem tijdens het afkoelen een toestand van lagere energie opzoekt. Snel afkoelen maakt de kans groot dat het systeem in een lokaal optimum blijft hangen. Langzaam afkoelen maakt deze kans kleiner, waardoor het systeem uiteindelijk een grotere kans heeft om op een lager, dus beter, energieniveau uit te komen.

[27] Voor een overzicht zie: C.R. Reeves (ed.), *Modern heuristic techniques for combinatorial problems*, McGraw-Hill, 1995.

De correspondentie met het TSP is bedacht door Kirkpatrick et al.[28] en Cerny.[29] Rondreizen corresponderen met toestanden van het systeem en de lengte van een rondreis is het energieniveau. Een optimale rondreis is dus een toestand met minimale energie.
Om dit in een algoritme in te bouwen gebruiken we voor de kans op toestand $i$ de *Boltzmann-verdeling* $e^{-E_i/t}$, waarbij $t$ de temperatuur is en $E_i$ de energie van toestand $i$. De verdeling wordt gebruikt om de kans op het nemen van stappen die ook verslechteringen mogelijk maken te modelleren (als er een betere rondreis wordt gevonden, dan wordt deze altijd gekozen). De kans op het nemen van een stap die een energie-verslechtering ter grootte $\Delta$ oplevert wordt $e^{-\Delta/t}$. De waarde van $t$ kunnen we zelf instellen. Daarbij is het de bedoeling dat er met een relatief grote waarde van $t$ wordt begonnen en dat die iedere keer na het nemen van een aantal, zeg $K$, stappen met een factor $\alpha$ wordt verlaagd, waarbij $\alpha < 1$, maar vrij dicht bij 1 ligt. Hiermee bereik je dat in het begin van het proces de kans op het nemen van een stap die geen verbetering oplevert nog vrij groot is, maar dat deze afneemt in de loop van de tijd. Hierdoor kan het proces in het begin nog alle kanten op en is de kans dat je in een lokaal minimum blijft hangen klein.

**Algoritme 2.13** *Simulated Annealing*

1. Start met een (willekeurige) rondreis $C^*$; kies $\alpha < 1$ en kies een begintemperatuur $t$.

2. (a) Kies $K$ (het aantal stappen van een iteratie); $k := 1$ en $C := C^*$;
   i. Wijzig $C$ (eventueel random) in een rondreis $C'$ en laat $\Delta = l(C') - l(C)$.
   ii. Als $\Delta \leq 0$: (1) $C := C'$;
   (2) als $l(C) < l(C^*) : C^* := C$;
   (3) ga naar stap 2a.
   Als $\Delta > 0$: (1) kies een random getal $\rho \in [0, 1]$;
   (2) als $\rho < e^{-\frac{\Delta}{t}} : C := C'$;
   (3) $k := k + 1$; als $k \leq K$: ga naar stap 2a, sub i;
   anders: ga naar stap 2b.

   (b) $t := \alpha \cdot t$ en ga naar stap 3.

3. Als aan het stopcriterium is voldaan: stop met $C^*$ als benadering voor de optimale tour.
   Anders: ga naar stap 2a.

Simulated annealing is een klasse van algoritmen met een enigszins experimenteel karakter. Dit geldt ook voor de constanten, zoals $t, K$ en $\alpha$, die nader moeten worden bepaald. Door de keuzes expliciet te maken wordt een algoritme verkregen. Onder bepaalde voorwaarden voor deze constanten kan worden aangetoond dat het algoritme convergeert naar een optimale oplossing. Een nadeel is echter dat deze (gegarandeerde) convergentie zeer traag is.[30]

[28] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, *Optimization by simulated annealing*, Science 220 (1983) 671–680.

[29] V. Cerny, *Thermodynamica approach to the traveling salesman problem: an efficent simulation algorithm*, Journal of Optimization Theory and Applications 45 (1985) 41–51.

[30] Voor een overzicht van allerlei varianten zie: P.J.M. Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory*

**2. Genetische algoritmen**

Genetische algoritmen zijn geïnspireerd door Darwin's theorie over de evolutie.[31] Het algoritme begint met verz. oplossingen; dit wordt een *populatie* genoemd. Hiermee wordt een nieuwe populatie gevormd (*nageslacht*). Een nieuwe oplossing wordt dus niet gegenereerd uit één vorige, maar uit een populatie. We streven er naar dat het nageslacht betere oplossingen bevat.
Dit gebeurt naar analogie van het mechanisme van de natuurlijke selectie (*survival of the fittest*), waarbij uit een verzameling van oplossingen de beste worden geselecteerd om nieuwe oplossingen te creëren. Deze nieuwe verzameling oplossingen wordt gegenereerd naar analogie van genetische gebeurtenissen zoals *kruising* en *mutatie*. Het volgende algoritme is in feite weer een klasse van algoritmen. Voor een specifiek algoritme moeten nadere keuzes worden gemaakt.[32]

**Algoritme 2.14** *Genetisch Algoritme*

1. Start met een bepaalde populatie $\mathcal{C}$ van rondreizen, zeg $N$ rondreizen.
2. Bepaal voor iedere $C \in \mathcal{C}$ de lengte $l(C)$ (*de fitheid*).
3. Creëer een nieuwe populatie door de volgende stappen uit te voeren:
   (a) *Selectie*: Kies elementen (*ouders*) uit de populatie, bijvoorbeeld op grond van hun fitheid (hoe fitter hoe groter de kans op selectie).
   (b) *Kruising*: Vorm *kinderen* uit de ouders.
   (c) *Mutatie*: Muteer de kinderen en plaats de mutaties in een nieuwe populatie $\mathcal{C}'$.
   (d) *Acceptatie*: Plaats de beste $N$ rondreizen van $\mathcal{C} \cup \mathcal{C}'$ in een nieuwe populatie $\mathcal{C}$.
4. Als aan het stopcriterium is voldaan: stop met als approximatie de beste rondreis uit $\mathcal{C}$.
   Anders: ga naar stap 2.

Dit is een erg algemeen algoritme. We zullen de diverse stappen afzonderlijk bespreken.

*Start populatie*

De eerste collectie $\mathcal{C}$ kan bijvoorbeeld worden bepaald door random een aantal permutaties van $1, 2, \ldots, n$ te kiezen en op elk van deze rondreizen 2-opt toe te passen.

*Selectie*

We kunnen beginnen met de twee fitste, daarna de volgende twee etc. We kunnen de keuze ook stochatisch maken: laat $l_i$ de lengte van de $i$-de rondreis uit de populatie zijn ($1 \leq i \leq N$), en laat $l_{max}$ een bovengrens van alle $l_i$'s zijn; kies rondreis $C_i$ met kans $p_i = \frac{l_{max} - l_i}{\sum_{j=1}^{N} (l_{max} - l_j)}$.

---

*and Applications*, Reidel, Dordrecht, 1987.

[31] John Holland was de eerste die dit in verband bracht met optimaliseringsproblemen, zie: J.H. Holland, *Adaptation in natural and artificial systems*, The University of Michigan Press, 1975.

[32] Voor een overzicht zie: D. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

*Kruising*

Het kruisen van twee rondreizen, zeg $C_1$ en $C_2$, met elkaar kan bijvoorbeeld als volgt gebeuren:
a. Kies een willekeurige stad, zeg stad $i$.
b. Neem van $C_1$ het deel $\{1, \ldots, i\}$ en van $C_2$ het deel $\{i, \ldots, 1\}$ en zet deze achter elkaar tot een nieuwe ronde.
c. Als in deze ronde knooppunten voor een tweede keer voorkomen, zeg knooppunt $j$ via $\{\ldots, i, j, k, \ldots\}$, dan snijden we af tot $\{\ldots, i, k, \ldots\}$.
d. Knooppunten die nog niet voorkomen worden één voor één ingevoegd (zie de eerder behandelde invoegheuristieken).

*Mutatie*

Voor het muteren kunnen we bijvoorbeeld een deel van de route kiezen en dit deel vervangen door een andere permutatie van de desbetreffende steden. Als dit geen betere rondreis oplevert, dan veranderen we de rondreis niet.

*Stopcriterium*

Er zijn verschillende mogelijkheden voor het stopcriterium, zoals:
- na het bepalen van de $K$-de populatie voor zekere $K$;
- als in de nieuwe populatie geen betere rondreis zit dan in de vorige;
- als een rondreis is gevonden met een lengte kleiner dan een gegeven waarde $l_0$.

**3. Tabu search**

Tabu search is een meta-heuristische methode bedacht door Glover.[33] Het is een lokale zoekmethode, die werkt met een 'omgeving'. Deze kunnen echter het nadeel hebben dat we op eerder onderzochte rondreizen stuiten of zelf dat cycling optreedt. Bij Tabu search worden eerder bekeken oplossingen in het vervolg uitgesloten (deze zijn taboe). De tot nu toe best gevonden rondreis noteren we met $C^*$

**Algoritme 2.15** *Tabu Search*

1. Start met een bepaalde rondreis $C$, laat $C^* = C$, en plaats $C$ op de Tabu lijst.
2. Bepaal de naburige oplossingen van $C$, die niet op de Tabu-lijst staan, zeg $C_1, C_2, \ldots, C_k$.
3. Laat $C'$ zdd. $l(C') = min_{1 \leq j \leq k}\, l(C_j)$ en als $l(C') < l(C^*)$, dan $C^* := C'$.
4. Pas de Tabu lijst aan.
5. Als aan het stopcriterium is voldaan: stop met als approximatie $C^*$.

   Anders: $C := C'$ en ga naar stap 2.

[33] F. Glover, *Tabu search - part I*, ORSA Journal on Computing 1 (1989) 190–206; *Tabu search - part II*, ORSA Journal on Computing 2 (1990) 4–32.

*Tabu-lijst*

Er zijn verschillende manieren om de Tabu lijst bij te houden, zoals:
- alle eerdere rondreizen komen op de Tabu lijst; een nadeel is dat dit veel opslagruime vergt;
- de lijst een vaste lengte, zeg $L$, en we plaatsen de laatste $L$ rondreizen op de Tabu lijst.

*Stopcriterium*

Er zijn verschillende mogelijkheden voor het stopcriterium, zoals:
- na een van te voren bepaald aantal iteraties;
- als in een van te voren bepaald aantal iteraties geen verbetering wordt gevonden;
- als een rondreis wordt gevonden met een lengte kleiner dan een gegeven waarde $l_0$.

Slotopmerking

Als we met een heuristiek een toelaatbare oplossing voor het TSP hebben gevonden, dan kunnen we een indicatie krijgen van hoe goed deze oplossing is door daarnaast een ondergrens van het probleem te bepalen. Hiervoor streven we naar een efficiënte methode die bovendien een vrij scherpe (d.w.z. dicht bij het optimum gelegen) grens geeft. Dit is enigszins met elkaar in strijd en hierop is geen eenduidig antwoord. Vaak probeert men enkele relaxaties en kiest dan de beste waarde. We hebben al enkele voorbeelden van efficiënte relaxaties voor het TSP gezien:
(1) Als toewijzingsprobleem: zie de formulering (2.4.1); dit is oplosbaar met een $\mathcal{O}(n^3)$-algoritme.[34]
(2) Als minimale opspannende boom: dit kan met een $\mathcal{O}(n^2)$-algoritme.[35]
(3) Omdat in iedere oplossing van het TSP ieder knooppunt incident is met twee takken, is een ondergrens ook te bepalen door in ieder knooppunt de lengte van de twee kortste takken te nemen en deze $2n$ lengtes bij elkaar op te tellen en door 2 te delen. Dit heeft eveneens complexiteit $\mathcal{O}(n^2)$.

**Vraag 2.9**

Pas Algoritme 2.6 op Voorbeeld 2.8 toe.

**Vraag 2.10**

Pas Algoritme 2.7 op Voorbeeld 2.8 toe.

**Vraag 2.11**

Toon aan dat als er $r$ takken uit een rondreis worden verwijderd zdd. geen losse knooppunten ontstaan, de overblijvende ketens op $2^{r-1}(r-1)!$ manieren met elkaar te verbinden zijn.

[34] zie Besliskunde 3.
[35] Het algoritme van Prim of het algoritme van Kruskal. Beide algoritmen zijn behandeld in Besliskunde 1.

### 2.4.4 Opgaven

**Opgave 2.11**

Toon aan dat de formulering van het handelsreizigersprobleem (2.4.1) met daaraan toegevoegd (2.4.2) correct is.

Aanwijzing:

Laat de route in plaats 1 beginnen en interpreteer $u_i = k$, wanneer plaats $i$ als $k$-de stad vanuit stad 1 wordt bezocht.

**Opgave 2.12**

Stel we willen $n$ stukken behang halen uit een voldoend lange rol met een patroon dat zich om de meter herhaalt. Op zo'n patroon van 1 meter begint stuk $i$ op positie $s_i$ en eindigt op $f_i$ en heeft dus lengte $f_i - s_i,\ i = 1, 2, \ldots, n$. We willen de stukken nu z'o afsnijden dat het totaal aantal meters dat we van de rol nemen om alle $n$ stukken eruit te snijden minimaal is.
Formuleer dit probleem als een handelsreizigersprobleem.

**Opgave 2.13**

Los het TSP met branch-and-bound op voor de volgende afstandentabel:

$$L = \begin{pmatrix} - & 3 & 93 & 13 & 33 & 9 \\ 4 & - & 77 & 42 & 21 & 16 \\ 45 & 17 & - & 36 & 16 & 28 \\ 39 & 90 & 80 & - & 56 & 7 \\ 28 & 46 & 88 & 33 & - & 25 \\ 3 & 88 & 18 & 46 & 92 & - \end{pmatrix}.$$

**Opgave 2.14**

a. Laat zien dat er een implementatie is van de invoeg-heuristiek 'dichtstbijzijnde stad' die complexiteit $\mathcal{O}(n^2)$ heeft.

b. Laat zien dat er een implementatie is van de invoeg-heuristiek 'verste stad' die complexiteit $\mathcal{O}(n^2)$ heeft.

**Opgave 2.15**

Beschouw grafen $G_k,\ k = 1, 2, \ldots$ van het volgende type. Neem $2^k$ knooppunten op een rechte lijn, zeg $v_1, v_2, \ldots, v_{2^k}$, en daarboven $2^k - 1$ knooppunten, eveneens op een rechte lijn, zeg $w_1, w_2, \ldots, w_{2^k-1}$, en teken gelijkzijdige driehoeken $(v_i, w_i, v_{i+1}),\ 1 \leq i \leq 2^k - 1$ met lengte van de zijde gelijk aan 1. We nemen aan dat de graaf volledig is en dat de lengte van de takken tussen knooppunten die niet op een zijde van een gelijkzijdige driehoek liggen gelijk is aan de lengte van de kortste route over deze driehoeken, d.w.z.

$$l(v_i, v_j) = |i-j|,\ l(w_i, w_j) = |i-j|+1,\ l(v_i, w_j) = \begin{cases} |i-j| & \text{als } i > j \\ |i-j|+1 & \text{als } i \leq j. \end{cases}$$

Bewijs (met inductie naar $k$) de volgende eigenschappen voor het naaste-buur algoritme:
a. Er is in $G_k$ een Hamilton pad $v_1, \ldots, w_{2^{k-1}}$, verkregen met de naaste-buur algoritme, dat een lengte van $(k+3)2^{k-1} - 2$ oplevert en met de laatste verbinding $(w_{2^{k-1}}, v_1)$ geeft dit een route met lengte $(k+4)2^{k-1} - 2$.
b. Beschouw de graaf $G$ die uit $G_k$ ontstaat door een knooppunt $v_0$ toe te voegen en dit te verbinden met $v_1$ en $v_{2^k}$ met takken van de lengte 1 (de lengte tussen $v_0$ en andere knooppunten is weer het minimum aantal takken over de driehoeken en de takken $(v_0, v_1)$ en $(v_0, v_{2^k})$). Toon aan dat een optimale tour in $G$ de lengte $2^{k+1}$ heeft en dat het mogelijk is dat het naaste-buur algoritme een tour met lengte $(k+4)2^{k-1}$ geeft.
c. Laat zien dat voor de naaste-buur heuristiek geldt $k(H) = \Omega(log\ n)$.

**Opgave 2.16**
Pas de heuristiek 'minimale opspannende boom met kortste volmaakte koppeling' toe op Voorbeeld 2.8.

**Opgave 2.17**
Beschouw het handelsreizigersprobleem met

$$L = \begin{pmatrix} - & 8 & 9 & 5 & 4 \\ 8 & - & 4 & 5 & 9 \\ 9 & 4 & - & 6 & 8 \\ 5 & 5 & 6 & - & 6 \\ 4 & 9 & 8 & 6 & - \end{pmatrix}.$$

Neem $C = \{1, 2, 3, 4, 5, 1\}$ als begintour (lengte $C = 28$).
a. Bepaal een 2-opt lokaal minimale tour, uitgaande van $C$.
b. Voer één stap van het Lin-Kernighan algoritme uit, uitgaande van $C$.

# Index