COMPUTERS EN (ON-)DOENLIJKE PROBLEMEN

J. van Leeuwen

RUU-CS-80-8
September 1980

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

COMPUTERS EN (ON-)DOENLIJKE PROBLEMEN

J. van Leeuwen

Technical Report RUU-CS-80-8

September 1980

Department of Computer Science
University of Utrecht
P.O. Box 80.002
3508 TA Utrecht, the Netherlands

Dit rapport bevat de uitgewerkte tekst van de gelijknamige bijdrage tot het MC Colloquium "Complexiteit en Algorithmen", gehouden in het najaar van 1980 aan het Mathematisch Centrum te Amsterdam.

COMPUTERS EN (ON-)DOENLIJKE PROBLEMEN

J. van Leeuwen
Rijksuniversiteit te Utrecht

1. ALGORITHMEN

Voor tal van praktische vraagstukken is het niet voldoende te weten dat een oplossing existeert, maar moet men een oplossing ook daadwerkelijk kunnen uitrekenen. Een algorithme is een duidelijke specificatie van de stappen (instructies) die men in een berekening dient uit te voeren om het gewenste "antwoord" te verkrijgen. Men kan algorithmen met veel woorden omschrijven of in een geschikte programmeertaal uitdrukken, afhankelijk van de beoogde toepassing. In ieder geval moeten de individuele instructies tot het elementaire repertoire behoren van de machine waarop men de algorithmen uitvoeren wil.

Het is om allerlei redenen gewenst algorithmen te meten. Zo is de uitvoering van een algorithme bijvoorbeeld zelden kostenloos en wil men van te voren een indruk hebben van de benodigde hoeveelheid tijd en/of geheugenruimte (of zelfs van het benodigde aantal processoren). Het is echt niet altijd zo dat het goedkoopste algorithme voor een vraagstuk ook het "beste" is, omdat andere eigenschappen (zoals de mate van numerieke stabiliteit) wel eens minder gunstig kunnen uitpakken. Niettemin zijn preciese uitspraken en schattingen over de duurte van algorithmen een belangrijk en vaak doorslaggevend gegeven, indien men uit verschillende algorithmen voor een vraagstuk kiezen wil. Alleen door analyse kan men er achter komen in welke mate een zeker algorithme beslag legt op de middelen (resources) van een computer.

Het is gebruikelijk om de duurte van een algorithme uit te drukken in termen van parameters die bepalend zijn voor de grootte van een input. Zo kan men bijvoorbeeld van een algorithme zeggen dat het n getallen in $O(n \log n)$ stappen sorteert. (Men kan dergelijke algorithmen vinden in e.g. KNUTH [27] of LORIN [32].) Het komt ook voor dat men alleen bewerkingen van een bepaalde soort telt. Zo is er een algorithme dat een n^e graads polynoom $p(x)$ en al zijn genormaliseerde afgeleiden $p^{(i)}(x)/i!$ (voor $1 \leq i \leq n$) kan evalueren in 2,5 n vermenigvuldigingen en delingen (DE JONG & VAN LEEUWEN [15]). Maar dat algorithme gebruikt nog wel $O(n^2)$ optellingen en aftrekkingen.

Men moet uitspraken over algorithmen vaak zorgvuldig lezen. Zo zegt men bijvoorbeeld snel dat het gewone algorithme voor de vermenigvuldiging van twee $n \times n$ matrices $O(n^3)$ stappen vergt. Een stap is dan een optelling, een

vermenigvuldiging of het opvragen/opslaan van een waarde uit/in een geheugenplaats. In de analyse van andere algorithmen gebruikt men misschien een ander stapbegrip. Het is dus van belang te weten welk berekenings model gehanteerd wordt, dus wat als "enkele stap" mag doorgaan. We houden het in het vervolg op het veelal gebruikte model van de random access machine (zie e.g. AHO, HOPCROFT & ULLMAN [1] of VITANYI [63]), dat in wezen een eenvoudige processor en een oneindig random access geheugen biedt. Er ligt nog een heel terrein van onderzoek open om uitspraken over algorithmen te krijgen waarin op realistische wijze rekening wordt gehouden met de doorgaans optredende overhead door I/O transporten op echte hardware.

De analyse van algorithmen heeft zich in de laatste jaren in versterkte mate gemanifesteerd, wellicht omdat men op veel ruimere schaal computers gebruikt en efficientere algorithmen verlangt om economische redenen. Zo worden regelmatig snellere algorithmen voor bestaande vraagstukken geponeerd, al wordt de "winst" soms pas behaald voorbij elke praktische waarde van n. De betekenis van dergelijke resultaten ligt dan in een iets andere richting, maar daarover later.

We bezien het testen van $A \subseteq B$ als voorbeeld van de afweging van mogelijke algorithmen door analyse (ontleend aan BENTLEY [2]). Stel dat twee ongeordende verzamelingen A (m elementen) en B (n elementen) als arrays gegeven zijn. We willen nagaan of elk element van A in B voorkomt. Neem aan dat $m \leq n$.

Het nieuwe algorithme zal eruit bestaan dat voor elk element x van A de verzameling B sequentieel doorlopen wordt tot x (al of niet) gevonden is. Men mag aannemen dat voor het eenmaal doorzoeken van B gemiddeld n/2 stappen nodig zijn, zodat de totale duurte van het naieve algorithme op ongeveer m.n/2 stappen uitkomt. Men kan het steeds opnieuw doorzoeken van B voorkomen door A en B eerst te sorteren. Dat kost weliswaar zo'n $m \log m + n \log n$ stappen, maar daarna kan men volstaan met het één maal gelijktijdig doorlopen van de beide arrays. Dit tweede algorithme vergt dus ongeveer $m \log m + n \log n + m + n$ stappen. Vervolgens kan men bedenken dat het sorteren van B al voldoende was om het zoeken erin door gebruik van binary search tot zo'n $\frac{1}{2} \log n$ per keer te versnellen. Het hierop gebaseerde algorithme vergt zo'n $n \log n + \frac{1}{2}m.\log n$ stappen. Als $m << n$ dan kan het slimmer zijn niet B maar A te sorteren! Test elk element van B op A met binary search en vlag elk element (in A) dat raak is. Aan het eind dient men louter nog te verifieren dat elk element van A gevlagd is. Het aldus besloten derde algorithme vergt zo'n $m \log m + \frac{1}{2}n \log m + m$ stappen.

Het hangt kennelijk van de grootte van $\frac{1}{2}n \log m$ in vergelijking tot $n \log n + n$ af of het derde algorithme wezenlijk sneller is dan het tweede. Er is echter nog een vierde algorithme denkbaar, gebaseerd op de techniek van

hashing (KNUTH [27]). Men dient dan wel over de nodige geheugenruimte te beschikken voor een hash-table, waarin de elementen van (zeg) B zonder groot gevaar van clustering ingevoegd kunnen worden. Theorie leert dat zo'n table minstens ongeveer 1,5 n posities lang moet zijn om met een geschikte hash-functie gemiddeld hoogstens 2 collisions per invoeging te mogen verwachten. Het hashen van B en vervolgens, ook met hashing, opzoeken van elke x uit A zal naar alle waarschijnlijkheid dus niet meer dan ongeveer 2n + 2m stappen vergen, de snelste manier tot nu toe. Maar we hebben wel heel wat extra geheugenruimte nodig. (Als m << n, dan moet men natuurlijk niet een hash-table voor B maar voor A opzetten.)

In de praktijk lukt het vaak niet om een sneller algorithme voor een vraagstuk te vinden en lijkt er geen noemenswaardige vooruitgang meer te behalen. We schijnen dan op een intrinsieke complexiteit van het vraagstuk te stuiten, die ieder algorithme dwingt tot het uitvoeren van een zeker minimum aantal bewerkingen. Alleen door analyse van de diepere structuur van het vraagstuk kan men er achter komen of het ons alleen aan slimheid heeft ontbroken óf binnen het gehanteerde berekeningsmodel er inderdaad geen efficiente algorithmen bestaan. Zo is het bekend dat in een model waarin elementen "vergeleken" mogen worden elk algorithme voor het testen van $A \subseteq B$ in voorkomende gevallen zeker $\Omega(n \log n)$ vergelijkingen maken moet (REINGOLD [45]). In die zin kan men "n log n" de "complexiteit" van het set-inclusie vraagstuk noemen. Merk op dat het algorithme gebaseerd op hashing buiten het kader van dit berekeningsmodel valt.

Er zijn maar weinig vraagstukken waarvan de algorithmische complexiteit precies bekend is. Zelfs goede ondergrenzen ontbreken vaak, wegens onvoldoende inzicht in de wiskundige structuur die in vele vraagstukken verscholen ligt. Zo ligt er een grote kloof tussen de bewezen ondergrens van ongeveer $2n^2$ vermenigvuldigingen aan n × n matrix multiplicatie (e.g. VAN LEEUWEN & VAN EMDE BOAS [61]) en het recent gevonden $O(n^{2,51..})$ algorithme voor deze taak (SCHÖNHAGE et al [48]), dat het wellicht bekendere $O(n^{2,80..})$ algorithme (STRASSEN [54]) inmiddels heeft verslagen. De voortdurende pogingen om steeds maar weer algorithmen te vinden die een antwoord met minder stappen dan tot nu toe bekend bereiken hebben zo het tweeledig doel gekregen om zowel economischer rekenmethoden te leveren voor de praktijk als "van boven af" de intrinsieke complexiteit van vraagstukken te benaderen. De dubbele rol die hieruit spreekt voor algorithmen pleegt nogal eens verwarrend te zijn voor de louter praktisch ingestelden.

De moderne paradigmas in het ontwerpen en analyseren van algorithmen vindt men in AHO, HOPCROFT & ULLMAN [1] of in de leesbare surveys van bijvoorbeeld HOPCROFT [18], MEYER [37], RABIN [44] en TARJAN [55]. De navolgende

notities pogen een indruk te geven van wat fundamentele themas in de wereld van complexiteit en algorithmen.

2. ONTWERPEN

Een belangrijk motief in de systematische studie van efficiente algorithmen is de voortdurende ontdekking van in wezen probleem-onafhankelijke methoden en benaderingen. Wat misschien eens voor zomaar een programmeertruuk doorging, is niet zelden een algemene techniek gebleken die ook in veel andere problemen toepasbaar is en tot snellere algorithmen voert. Een ontwerper (programmeur) zal met een redelijk aantal van die algorithmische technieken vertrouwd moeten zijn.

Het is bekend dat de efficiency van een algorithme in aanzienlijke mate afhankelijk is van de gebruikte data structuren voor gegevens. Vertrouwdheid met de vele mogelijkheden van linked lists is vereist (zie bijvoorbeeld HOROWITZ & SAHNI [19] of STANDISH [52]). Men heeft geleerd om bij ontwerpen niet meteen de data structuren te fixeren, maar eerst de fundamentele operaties te ontdekken die men op de data wil uitvoeren. Deze benadering middels "abstracte data structuren" stelt ons in staat de efficiente implementatie redelijk los van andere zaken te bestuderen. Zo zijn er nogal wat problemen met eindige verzamelingen V waarop men in wezen de volgende bewerkingen moet kunnen uitvoeren

INSERT(x) : voeg x toe aan V
DELETE(x) : laat x weg uit V
SEARCH(x) : bezoek x in V (zo x element van V is)

Door gebruik te maken van een geschikt type van gebalanceerde zoekbomen (e.g. [1]) toont men

Stelling 2.1. Er is een data structuur voor V zodat INSERT, DELETE en SEARCH instructies nooit meer dan O(log n) stappen vergen (n =# V).

Als men sommige elementen van V vaker bezoekt dan andere, dan zal men een zoekboom wensen waarin de gemiddelde zoektijd minimaal is. MEHLHORN [36] heeft getoond hoe men in die zin optimale zoekbomen ook dynamisch bijna optimaal kan houden.

Bij ontwerpen stelt men verzamelingen meestal maar meteen door gebalanceerde bomen voor "omdat het daar toch altijd op uitdraait". Maar stel dat we een soort lijst willen hebben waarop de volgende instructies zijn uit te voeren

INSERT(k,x) : voeg x in juist vòòr het k^e element
DELETE(k) : laat het k^e element weg
SEARCH(k) : bezoek het k^e element

(k willekeurig). Met een gewone gebalanceerde zoekboom komt men aan O(log n) per instructie, met n =# V. Men kan echter bewijzen (zie e.g. CLANCY & KNUTH [13])

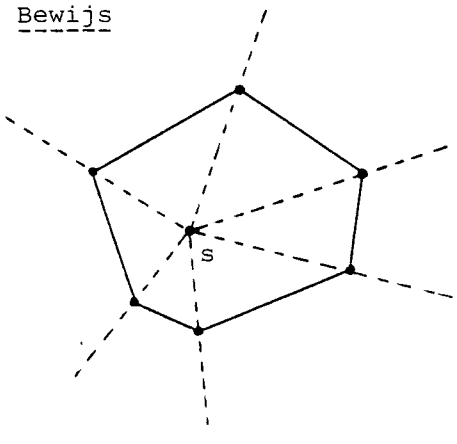
Stelling 2.2. Er is een data structuur voor V zodat een INSERT(k,x), DELETE(k) of SEARCH(k) instructie nooit meer dan O(log k) stappen vergt, onafhankelijk van de grootte van V.

Vervolgens is het bekend dat veel efficiente algorithmen uiteindelijk zijn verkregen na een verder onderzoek van de structuur van een probleem. Men heeft een aantal technieken leren onderscheiden die, mits een probleem zich leent voor toepassing, vaak tot extra tijdwinst voeren. We noemen en illustreren enkele van deze algorithmische technieken.

Een eerste techniek is binary search, vooral bekend geworden om in O(log n) stappen elke specifieke waarde in een gesorteerd n-plaatsig array op te zoeken. (Om binary search op lijsten toe te passen, moet men weer van gebalanceerde zoekbomen gebruik maken.) Maar binary search is in feite altijd toepasbaar, als men maar een geschikte zoekruimte in een probleem kan onderkennen. Het volgende voorbeeld is ontleend aan SHAMOS [51]

Stelling 2.3. Men kan een convexe n-hoek zodanig representeren dat voor elk willekeurig punt p in het vlak in slechts O(log n) stappen beslist kan worden of p in het binnen- of in het buitengebied van de n-hoek ligt.

Bewijs



Kies een punt s in het binnengebied van de convexe n-hoek en trek rechten vanuit s door alle hoekpunten (zie figuur). Op deze wijze wordt het vlak in sectoren verdeeld, die op ruimtehoek bij s gesorteerd kunnen worden opgeslagen. Gegeven een punt p, dan kan men met binary search in O(log n) stappen de sector bepalen waartoe p behoort. Men behoeft dan alleen nog maar de positie van p ten opzichte van de in de sector aanwezige randlijn te bepalen om het gewenste antwoord te krijgen. □

Met wat moeite kan men de stelling tot willekeurige, enkelvoudige n-hoeken uitbreiden (zie [51]). KIRKPATRICK [25] verkreeg onlangs een eenvoudig algorithme om voor elke planaire vlakverdeling met n knooppunten in maar O(log n) stappen te bepalen tot welk gebied van de verdeling een punt behoort.

Een tweede techniek is divide-and-conquer. Hieronder verstaat men de verdeling (splitsing) van een probleem in twee of meer deelproblemen van kleiner formaat, waarvan de oplossingen eenvoudig samengesteld kunnen worden tot een oplossing van het oorspronkelijke probleem. Men komt typisch op recursieve algorithmen uit. Diverse sorteer algorithmen zoals merge-sort en quicksort (zie [1]) berusten op divide-and-conquer. Ook het volgende resultaat van BLUM et al [6] berust op een slimme toepassing van deze strategie.

Stelling 2.4. Voor elke t kan het op t na grootste element van een ongeordende verzameling V in O(n) stappen gevonden worden (n =# V).

Het volgende voorbeeld is ontleend aan BENTLEY & SHAMOS [4].

Stelling 2.5. Het convex omhulsel van een verzameling van n punten in het vlak is te berekenen in O(n log n) stappen.

Bewijs

Verdeel de punten over twee deelverzamelingen van n/2 punten elk en bepaal het convex omhulsel van de beide deelverzamelingen door recursieve toepassing van het algorithme. Men kan bewijzen dat de resulterende convexe i- en j-hoeken (i, j ≤ n/2) in O(n) stappen tot het convex omhulsel van de gehele verzameling zijn samen te stellen. Is T(n) het benodigde aantal stappen van het algorithme op een verzameling van n punten, dan geldt voor zekere c

$$T(n) \leq 2T(n/2) + c.n$$

Er volgt dan $T(n) = O(n \log n)$. □

YAO [65] bewees dat de stelling in redelijke zin optimaal is, PREPARATA & HONG [42] bewezen dat zij ook in 3 dimensies doorgaat.

Een derde techniek is dynamic programming. Dit betreft strategieën die, om tot de constructie van een oplossing te kunnen overgaan, in feite eerst een "rap" kwalitatief onderzoek moeten plegen van alle deelproblemen van kleiner

formaat. Het $O(n^3)$ algorithme van MURAOKA & KUCK [38] voor bepaling van de optimale evaluatie volgorde van een product van n matrices van willekeurige formaten is een typisch voorbeeld van dynamic programming. Om de n^3-grens te verklaren die in deze problemen nogal eens gevonden wordt, bedenke men dat dynamic programming nogal eens neerkomt op het "invullen" van een n × n tabel c waarvoor

$$c[i,i] = 0$$
$$c[i,j] = w(i,j) + \min_{i<k\leq j}(c[i,k-1] + c[k,j]) \text{ voor } i < j$$

met een of andere goedkope increment functie w. Stel eens dat w voldoet aan de volgende vierhoeksongelijkheid.

$$w(i,j) + w(i',j') \leq w(i',j) + w(i,j') \text{ voor } i \leq i' \leq j \leq j'$$

YAO [66] heeft getoond dat in dát geval dynamic programming versneld kan worden, want zij bewees

Stelling 2.6. Indien w aan de voornoemde vierhoeksongelijkheid voldoet en bovendien interval-monotoon is (d.w.z. dat $w(i,j) \leq w(i',j')$ als $i' \leq i \leq j \leq j'$), dan kan men c in $O(n^2)$ stappen invullen.

Het bekende $O(n^2)$ algorithme voor constructie van optimale zoekbomen (KNUTH [26]) is uiteindelijk een vrij direct gevolg van deze stelling gebleken.

Een vierde techniek is dynamisatie. Daarbij beperkt men zich aanvankelijk tot het vinden van een efficiente statische data structuur voor een probleem en poogt men vervolgens daaruit langs directe weg een efficiente dynamische oplossing te maken. Soms lukt dat met lokaal bijwerken (zoals in AVL en 2-3 bomen [1]) of door het geregeld totaal herbouwen van uit balans geraakte delen van de data structuur (zoals in e.g. OVERMARS & VAN LEEUWEN [39]). Een recente benadering is een dynamisch systeem van statische componenten bij te houden. Dat vereist dat een zoekprobleem in zekere zin "ontbindbaar" is (zie BENTLEY [3]).

Definitie. Een zoekprobleem Q(V) op verzamelingen V heet ontbindbaar indien voor elke partitie $V_1 \cup V_2$ van V geldt dat

$$Q(V) = \square(Q(V_1), Q(V_2))$$

met een eenvoudige bewerking. □

Zo is het naaste-buur-probleem, dat voor een willekeurig punt vraagt het dichtstbijgelegen element van V te vinden, een ontbindbaar zoekprobleem. Stel dat de statische versie van een ontbindbaar zoekprobleem op n punten

in Q(n) stappen op te lossen is, door gebruik van een data structuur die B(n) stappen kostte om te bouwen. Onder milde aannamen over Q en B volgt uit VAN LEEUWEN & WOOD [62] bijvoorbeeld

Stelling 2.7. Er is een volledig dynamisch oplossing van het bedoelde ontbindbare zoekprobleem, waarin updates maar O(B(n/log n)) stappen vergen en zoekacties altijd binnen O(log n.Q(n/log n)) stappen het gewenste antwoord leveren.

Zo is bijvoorbeeld een statische oplossing van het naaste-buur-probleem bekend met Q(n) = log n en B(n) = n log n (LIPTON & TARJAN [31]). Uit de stelling volgt dat er dan een dynamische oplossing is die "maar" O(n) per update en $O(\log^2 n)$ per zoekactie vergt.

Het is bij het ontwerpen van algorithmen vaak helemaal niet duidelijk of een bepaalde algorithmische techniek wel toepasbaar is. Een dieper onderzoek van de onderliggende structuur van een probleem kan dan gewenst zijn, hetgeen niet zelden tot onontdekte stukjes wiskunde voert. Zo hebben de Fast Fourier Transform en Newton Iteratie uiteindelijk toepassing gevonden in de computerarithmetiek van integers (zie e.g. BORODIN & MUNRO [7]). Een ander voorbeeld is de "planar separator" stelling van LIPTON & TARJAN [30]

Stelling 2.8. Men kan elke n-knopige planaire graph in O(n) stappen in drie stukken A, B en S partitioneren zodanig dat geen knoop van A met een knoop van B verbonden is, A en B elk ten hoogste $\frac{2}{3}$n knopen bezitten en S uit ten hoogste $2\sqrt{2}.\sqrt{n}$ knopen bestaat.

Met deze stelling wordt een krachtige vorm van divide-and-conquer op planaire graphen mogelijk (zie [31]). PHILIPP & PRAUSS [41] bewezen onlangs dat er n-knopige planaire graphen bestaan waarin elke separator ten minste $\sqrt{\frac{7}{6}}.\sqrt{n}$ knopen telt. Hoe het gebruik van beschikbare algorithmische technieken tot een geheel nieuwe visie op klassieke problemen kan voeren is onlangs nog weer bewezen in SHAMOS [50].

Een belangrijk hulpmiddel bij ontwerpen is de beschikbaarheid van efficiente subroutines, dus de parate kennis dat zekere taken binnen een bepaalde efficiency uitvoerbaar zijn. Niet zelden ziet men dat een ontwerper naar het gebruik van dergelijke subroutines toewerkt "omdat die tenminste een scherpe tijdgrens halen". Zo kan men een LU-decompositie algorithme formuleren dat geregeld matrix multiplicaties aanroept en uiteindelijk binnen

dezelfde $O(n^{\alpha})$ tijdgrens blijft (mits $\alpha > 2$, zie BUNCH & HOPCROFT [9]). Een interessant gevolg is

Stelling 2.9. Er is een algorithme dat binnen $O(n^{2,51..})$ stappen de determinant van een n × n matrix levert.

Dit algorithme behoeft nog niet numeriek aantrekkelijk te zijn, maar leert in ieder geval iets over de complexiteit van het determinant probleem. Vindt men een betere tijdgrens voor matrix multiplicatie, dan verandert de stelling mee.

Om een efficienter algorithme te vinden is het niet altijd nodig een geheel nieuwe benadering tot het oorspronkelijke probleem te vinden. Vele algorithmen zijn uiteindelijk het resultaat van sleutelen aan eerder algorithmen en/of hun implementaties. Een mooi voorbeeld van wat door algorithmische engineering bereikt kan worden toont de volgende "geschiedenis" van het maximum-flow probleem voor netwerken met V knopen en E kanten.

ALGORITHME	TIJDGRENS
Ford en Fulkerson (1956)	-
Edmonds en Karp (1969)	E^2V
Dinic (1970)	EV^2
Karzanov (1973)	V^3
Malhotra et al (1978)	V^3
Cherkasky (1976)	$E^{1/2}\ V^2$
Galil (1978)	$E^{2/3}\ V^{5/3}$
Galil en Naamad (1979)	$EV \log^2 V$
Sleator en Tarjan (1980)	$EV \log V$

ITAI & SHILOACH [20] vonden een $O(V^2 \log V)$ algorithme voor het maximum-flow probleem in ongerichte planaire graphen.

Tenslotte kan het de ontwerper gelukken een vraagstuk in zijn geheel te transformeren tot een ander, waarvoor een efficient algorithme reeds bestaat. Zo worden vragen in de operations research nogal eens tot een lineair programmerings probleem herleid, dat vervolgens met de simplex methode wordt opgelost. (Of Khachian's algorithme, zie e.g. LOVáSZ [33] of SCHRIJVER [49], hier praktische verbetering brengt, valt nog te bezien.) Interessante voorbeelden van algorithmische tranformatie vindt men ook in BROWN [8]. Maar het grootste belang van transformaties is wellicht dat daarmee de complexiteiten van geheel

uiteenlopende problemen aan elkaar gerelateerd kunnen worden (zie e.g. KARP [22] of de volgende sectie). Het kan leren dat de hoop op een efficienter algorithme zinloos is, indien dit reeds zo is voor het getransformeerde vraagstuk.

3. WAT HEET DOENLIJK

Een voor een zekere taak bestemd algorithme is soms helemaal niet realistisch, gelet op de benodigde hoeveelheden rekentijd en geheugen om het ermee corresponderende computer programma daadwerkelijk te kunnen executeren. Voor kleine inputs gaat het misschien wel, maar bij oplopend formaat kan het algorithme snel "ondoenlijk" worden.

Voorbeeld. Een chemicus houdt een file van structuren bij en zoekt een algorithme voor het GRAPH-ISOMORPHISM probleem als onderdeel van een retrieval procedure. U suggereert hem alle knoop-identificaties af te testen, maar belooft het probleem verder te onderzoeken.

Of het zoeken naar een beter algorithme zinvol is, zal van de intrinsieke complexiteit van een probleem afhangen. Men kan proberen een probleem in een bekende hierarchie van complexiteitsklassen in te schalen, ook al geeft dit maar heel ruwe informatie. We zullen in deze notities daar niet op in gaan en ons zelfs niet bekommeren om een al te preciese definitie van verder benodigde begrippen (zie e.g. GAREY & JOHNSON [17]).

Men noemt een probleem wel "doenlijk" indien er een algorithme voor bestaat waarvan de geschatte rekentijd T voldoet aan

$$T(n+i) \leq c.T(n) \text{ voor } 0 \leq i \leq n$$

voor zekere constante c. Er volgt dat T begrensd moet zijn door een polynoom in n.

Definitie. P is de klasse van problemen die door algorithmen met polynomiaal begrensde rekentijd oplosbaar zijn (uitgaande van een natuurlijke computer representatie voor de inputs).

Men kan zich nu bijvoorbeeld afvragen of een praktisch vraagstuk voor graphen zoals

> TRAVELING-SALESMAN: bestaat er een gesloten pad binnen zekere lengte dat alle punten van de graph juist één keer treft

in P zit. Het opsommen en vervolgens testen van alle paden in een graph is

stellig niet polynomiaal begrensd en voert tot een algorithme dat al voor graphen van 20 knopen meer seconden rekentijd kan vergen dan sinds het begin der jaartelling zijn verlopen (cf. KARP [24]). De moeilijkheid komt beter uit als we het algorithme formuleren als: raad de opvolgende kanten en verifieer of het resulterende pad aan de eisen voldoet. Het verifiëren kan eenvoudig in polynomiale tijd geschieden, maar geen vorm van backtracking is bekend waarmee het "raden" (als substituut voor opsommen) tot een polynomiaal begrensd deterministisch zoek-proces is om te vormen.

Laten we vanaf nu alle vraagstukken als ja/nee beslissingen formuleren. Een algorithme heet non-deterministisch indien is toegelaten dat in (sommige) stappen uit een begrensd aantal mogelijke acties gekozen wordt. Een non-deterministisch algorithme moet "ja" kunnen opleveren (door een rij van keuzen) dan en precies dan als de gecodeerde input als probleem inderdaad oplosbaar is. De "rekentijd" van een non-deterministisch algorithme is gedefinieerd als het kleinste aantal stappen nodig om op "ja" uit te komen, aannemend dat zo'n antwoord voor de gegeven input bestaat.

Definitie. NP is de klasse van problemen die door non-deterministische algorithmen met polynomiaal begrensde rekentijd oplosbaar zijn (uitgaande van een natuurlijke computer representatie voor de inputs).

De vraag of NP-problemen zoals TRAVELING-SALESMAN doenlijk zijn kan kortweg worden samengevat in

Open probleem : is P = NP?

Ga na dat ook GRAPH-ISOMORPHISM in NP zit en dus automatisch "doenlijk" zou zijn als P gelijk aan NP mocht blijken.

Bij het ontwerpen van algorithmen stuit men nogal eens op taken die wel in NP maar kennelijk niet in P zitten. Een bekend voorbeeld uit de propositielogica is

> 3SAT : is een gegeven formule in conjunctieve normaalvorm
> met precies 3 literals per clausule vervulbaar

en er zijn vele andere te noemen. Het klinkt misschien oneerlijk om de vraag naar een doenlijk algorithme voor TRAVELING-SALESMAN of 3SAT op één lijn te stellen met het hele P = NP probleem. Toch is dit voor sommige problemen juist.

Definitie. M heet p-reduceerbaar tot L indien een in polynomiale tijd berekenbare afbeelding f bestaat zodat voor elke x : x is oplosbaar als M-probleem $\Leftrightarrow$ f(x)

is oplosbaar als L-probleem.

Is M p-reduceerbaar tot L en L $\in$ P(NP), dan is ook M $\in$ P(NP).

Definitie. L heet NP-volledig indien L $\in$ NP en iedere M $\in$ NP p-reduceerbaar tot L is.

Is L NP-volledig, dan geldt natuurlijk: P = NP $\Leftrightarrow$ L $\in$ P. Een belangrijk resultaat van COOK [14] uit 1971 toont dat NP-volledige problemen inderdaad bestaan!

Stelling 3.1. 3SAT is NP-volledig.

KARP [22] gaf in 1972 al een lange lijst van bekende problemen, waaronder TRAVELING-SALESMAN, die NP-volledig bleken te zijn. En er zijn er sindsdien vele aan toegevoegd (zie [17]).

Om een probleem NP-volledig te bewijzen is het meestentijds onnodig de precies achtergronden van de theorie te leren, en kan men zich eenvoudigweg beroepen op

Stelling 3.2. Zij L NP-volledig. Is M $\in$ NP en L p-reduceerbaar tot M, dan is M NP-volledig.

Het wordt daarmee een kwestie van het ene probleem in het andere "vertalen" (door een p-reductie). Zo wordt in [17] uit 3SAT de NP-volledigheid bewezen van

> 3DM : gegeven eindige verzamelingen X, Y, Z met $|X| = |Y| = |Z| = q$ en een $M \subseteq X \times Y \times Z$, bestaat er een $M' \subseteq M$ zodat elk element van X, Y en Z in precies één triple van M' voorkomt

We illustreren hoe uit 3DM de NP-volledigheid van weer andere problemen is te bewijzen, gebruik makend van 3.2. en een geschikte p-reductie. Bezie de volgende variant van het "knapsack" probleem:

> KNAPSACK1 : gegeven positieve natuurlijke getallen $a_1, \ldots, a_n$ en b, is er een $J \subseteq 1, \ldots, n$ zodat $\sum_{i \in J} a_i = b$

Stelling 3.3. KNAPSACK1 is NP-volledig

Bewijs

Natuurlijk is KNAPSACK1 in NP: raad maar een J en verifieer dat het een oplossing geeft. Dat kan in "non-deterministische" polynomiale tijd. We tonen

voorts dat 3DM p-reduceerbaar is tot KNAPSACK1. Representeer de elementen van X, Y en Z door de getallen ("cijfers") 0 t/m q-1. Kies $\beta = q^3+1$. Vertaal naar een KNAPSACK1 probleem door elk triple $(i,j,k) \in M$ om te vormen tot een $a_{ijk} = \beta^i + \beta^{q+j} + \beta^{2q+k}$ en $b = \beta^{3q-1}-1/\beta-1 = 1 + \beta + \beta^2 + \ldots + \beta^{3q-1}$ te nemen. Getallen binair schrijvend, is dit inderdaad een p-reductie. Stellen we ons de getallen even voor in het "β-tallig" stelsel, dan is duidelijk dat de oplosbaarheid van 3DM gelijk geworden is aan de oplosbaarheid van het vertaalde KNAPSACK1 probleem. Immers, bestaat een $M' \subseteq M$ zodat elk element van X, Y en Z in juist één triple van M' voorkomt, dan geldt $\sum_{(i,j,k)\in M'} a_{ijk} = b$. Bestaat omgekeerd een M' (noodzakelijk met ten hoogste q^3 elementen) met $\sum_{(i,j,k)\in M'} a_{ijk} = b$, merk dan op dat in de linker som voor elke macht van β een "cijfer" hoogstens q^3 groot kan komen. Omdat $\beta > q^3$, geldt de wet van de unieke getal representatie en kan de linker som alleen gelijk aan b zijn als alle machten β^i, β^{q+j} en β^{2q+k} $(0 \leq i,j,k \leq q-1)$ maar precies één keer worden bijgedragen door de triples in M'. □

Men kan KNAPSACK1 ook formuleren als de vraag of een vergelijking $a_1x_1 + \ldots + a_nx_n = b$ oplosbaar is met $x_i \in \{0,1\}$. Het stellen van wat minder strakke eisen voor oplosbaarheid behoeft de moeilijkheden nog niet te vermijden. Bezie eens

> KNAPSACK2 : gegeven positieve gehele getallen $a_1, \ldots, a_n$, b en d, bestaat er een oplossing van $a_1x_1 + \ldots a_nx_n = b$ met natuurlijke getallen x_i zodat $\sum x_i = d$.

Voor vaste d is KNAPSACK2 met een "polynomiaal begrensd" algorithme op te lossen: som maar alle $O(n^d)$ rijtjes $\{x_i\}_i$ met $0 \leq x_i \leq d$ en $\Sigma x_i = d$ op en test of er een oplossing bij zit. Maar laten we d met de overige variabelen vrij, dan geldt

Stelling 3.4. KNAPSACK2 is NP-volledig.

Bewijs

Weer is KNAPSACK2 ∈ NP. Op nagenoeg dezelfde wijze als in 3.3. blijkt 3DM ook p-reduceerbaar tot KNAPSACK2. Kies $\beta = q^4+1$, vertaal als tevoren, en neem d = q. Omdat een oplossing M' van 3DM noodzakelijk q triples heeft, zal daaruit meteen een oplossing van $\sum_{(i,j,k)\in M} a_{ijk}x_{ijk} = b$ volgen met $\sum_{(i,j,k)\in M} x_{ijk} = q$. (Tot zover lijkt het nog erg op KNAPSACK1.) Bestaat omgekeerd een oplossing van $\sum_{(i,j,k)\in M} a_{ijk}x_{ijk} = b$ onder de gestelde eis aan $\sum_{(i,j,k)\in M} x_{ijk}$, merk dan op dat in de linker som nu elke macht van β hoogstens

q^3 keren (de grens op het aantal triples) met een factor q (de grens van elke x-waarde) kan worden aangedragen en dus een "cijfer" hoogstens q^4 groot voor zich kan krijgen. Omdat we nu $\beta > q^4$ kozen, geldt toch weer de wet van de unieke getal representatie en kan men concluderen dat noodzakelijk elke $x_{ijk} \in \{0,1\}$ en dat $M' = \{(i,j,k) \mid x_{ijk} = 1\}$ een oplossing van het oorspronkelijke 3DM probleem moet zijn. □

Het begint erop te lijken dat het oplossen van een eenvoudige lineaire vergelijking $a_1x_1 + \ldots + a_nx_n = b$ minder simpel is dan wellicht werd aangenomen als we een oplossing over $\mathbb{N}$ zoeken. Over $\mathbb{Z}$ is het in polynomiale tijd te doen door testen dat $ggd(a_1, \ldots, a_n) \mid b$, maar over $\mathbb{N}$ bestaat een soortgelijk criterium (nog) niet. We omschrijven het probleem als

> SLEQ : gegeven positieve natuurlijke getallen $a_1, \ldots, a_n$ en b, bestaat er een oplossing van $a_1x_1 + \ldots + a_nx_n = b$ met natuurlijke getallen x_i.

In het volgend resultaat van VAN EMDE BOAS [59] wordt ons vermoeden bevestigd.

Stelling 3.5. SLEQ is NP-volledig.

Bewijs

SLEQ $\in$ NP door op te merken dat de x_i door b begrensd (moeten) zijn en het "raden" dus polynomiaal begrensd kan blijven. We tonen nu dat KNAPSACK2 p-reduceerbaar is tot SLEQ. Gegeven een KNAPSACK2 probleem, kies $G = 1 + \max\{b, d\sum_i a_i - b\}$ en stel $A_i = a_i + G$ en $B = b + dG$. Het SLEQ-probleem $\sum_i A_i x_i = B$ kan men na invullen omschrijven tot $G.(\sum_i x_i - d) = b - \sum_i a_i x_i$. Het is duidelijk dat een oplossing van KNAPSACK2 meteen een oplossing van het vertaalde SLEQ-probleem geeft. Stel omgekeerd dat een oplossing van $G.(\sum_i x_i - d) = b - \sum_i a_i x_i$ in natuurlijke getallen bestaat. Omdat $G > b$, kan $\sum_i x_i - d$ niet >0 zijn. Maar $\sum_i x_i - d$ kan ook niet <0 zijn, omdat dan $G \leq G.(d - \sum_i x_i) = \sum_i a_i x_i - b \leq d.\sum_i a_i - b$ zou zijn. Dus moet $\sum_i x_i = d$ en (dus) $\sum_i a_i x_i = b$ zijn, een oplossing van het oorspronkelijke KNAPSACK2 probleem. Ga na dat de afbeelding van KNAPSACK2 naar SLEQ inderdaad een p-reductie is. □

Lukt het dus om voor oplossing van lineaire vergelijkingen $a_1x_1 + \ldots + a_nx_n = b$ over $\mathbb{N}$ een deterministisch algorithme met polynomiaal begrensde rekentijd te vinden, dan zou men meteen het hele P = NP probleem hebben opgelost!

Men kan de theorie van NP-volledigheid verfijnen door andere (striktere) reducties te onderzoeken. BERMAN & HARTMANIS [5] bewezen dat in de huidige

theorie alle bekende NP-volledige problemen zelfs p-isomorf zijn. Verder, merk op dat als we weten dat een oplossing "bestaat" we nog geen oplossing "hebben" en een NP-volledig probleem zo nog wel eens extra moeilijkheden bij berekening kan geven (zie VALIANT [56]). Een probleem kan voorts extra moeilijk zijn omdat we "alle" oplossingen willen tellen. Is #P de klasse van "tel-varianten" van NP-problemen (voor een preciese definitie, zie VALIANT [58]), dan heet L #P-volledig indien $L \in \#P$ en elk probleem $M \in \#P$ in polynomiale tijd oplosbaar is indien een subroutine aanroep aan L als "een stap" mag gelden. Het analogon van 3.1. blijkt waar ([58]).

Stelling 3.6. Het tellen van de mogelijke "vervullingen" van een 3SAT probleem is #P-volledig.

Ook het tellen van alle oplossingen van een SLEQ probleem is #P-volledig. Een lijst van overige, bekende #P-problemen vindt men in [58]. Interessant is dat we hier het vanouds bekende probleem van de permanent berekening kunnen plaatsen (VALIANT [57])

PERMANENT : gegeven een integer matrix $A = (a_{ij})$, bereken $\sum_{\sigma} \prod_{i} a_{i\sigma(i)}$ gesommeerd over alle permutaties $\sigma \in S_n$.

Stelling 3.7. Berekening van PERMANENT is #P-volledig.

Dat men er nog altijd niet in geslaagd is permanenten even efficient te berekenen als determinanten (vgl 2.9.), blijkt dus nauw verweven met de algemene P-versus-NP problematiek.

Een bewijs dat een bekend NP-volledig probleem te reduceren is tot een zeker vraagstuk L wordt tegenwoordig vaak als argument (of excuus) voor de "ondoenlijkheid" van zo'n probleem L aangevoerd. Blijkt ooit nog eens dat $P = NP$, hetgeen onwaarschijnlijk lijkt, dan zal men de status van vele problemen moeten herbezien. Van maar betrekkelijk weinig concrete problemen uit de praktijk is bekend dat elk algorithme ervoor zeker, zeg, exponentieel of erger toenemende rekentijd moet hebben (zie e.g. STOCKMEYER & CHANDRA [53]).

En hoe zit het nu met GRAPH-ISOMORPHISM? Helaas is hiervan nóch bekend of het in P zit, nóch of het NP-volledig is! LADNER [28] bewees dat er dergelijke problemen moeten zijn als $P \neq NP$. Voor GRAPH-ISOMORPHISM ontbreekt nochtans het antwoord of het "doenlijk" is of niet. Onlangs bewees LUKS [34] dat isomorfie in polynomiale tijd beslisbaar is voor elke klasse van graphen waarin knopen begrensde graad hebben.

4. ONDOENLIJK, MAAR TOCH TE DOEN?

Al is een probleem ondoenlijk voor toenemende n (bijvoorbeeld door de bewezen NP-volledigheid), dan behoeft men de haalbaarheid van een goed algorithme voor de praktijk niet zomaar uit te sluiten. Blijven de verwachte inputs "klein", dan is van een explosie in de nodige rekentijd misschien wel niets te merken. Heuristische methoden kunnen bovendien tot verdere bezuinigingen in rekentijd leiden en de praktische reikwijdte van een algorithme vergroten. Met het beschikbaar komen van multiprocessor technologie (e.g. MEAD & CONWAY [35]), wordt het zelfs realistisch alle NP-problemen als doenlijk te gaan zien. Natuurlijk blijven andere problemen even ondoenlijk als tevoren (zie e.g. VITANYI [63]).

In de laatste jaren zijn allerlei manieren bedacht om onder de complexiteit van problemen uit te komen. Zo kan men voor sommige problemen (waaronder ook NP-volledige) algorithmen vinden die "meestal" snel zijn (zie e.g. KARP [23]). De vereiste probabilistische analyses zijn vaak moeilijk. WEIDE [64] biedt een gedegen inleiding tot het onderwerp. Een geheel andere mogelijkheid is een algorithme te vinden dat altijd snel maar slechts beperkt betrouwbaar is (zie e.g. RABIN [43]).

In nogal wat optimaliseringsvraagstukken die zelf NP-volledig (of erger) zijn, blijkt het mogelijk om snel (lees: met een polynomiaal begrensd algorithme) een antwoord te bepalen dat "dicht" bij het gezochte optimum moet liggen. Er is eigenlijk geen coherente theorie van "approximatie algorithmen", maar er zijn enkele interessante inzichten bereikt. We beperken ons eerst tot optimaliseringsvragen waarin een "minimum" gezocht wordt.

Definitie. Een approximate algorithme voor een probleem haalt een performance factor K indien voor alle toegelaten inputs geldt: $|\text{output}|/|\text{optimum}| \leq K$

Herdefinieer TRAVELING-SALESMAN nu zo dat om een kortste gesloten pad verzocht wordt dat alle punten in een graph precies een keer treft. SAHNI & GONZALEZ [47] merkten op

Stelling 4.1. Is $P \neq NP$, dan bestaat er voor geen enkele constante K een polynomiaal begrensd approximatie algorithme dat een performance factor K haalt op TRAVELING-SALESMAN.

Anders is het bij GEOMETRIC-TRAVELING-SALESMAN, waarin om een kortste gesloten pad door een stel punten in het vlak (met de euclidische metriek) gevraagd wordt. Het probleem zelf is NP-volledig (PAPADIMITRIOU [40]), maar CHRISTOFIDES [10] (zie ook [11]) bewees

Stelling 4.2. Er is een polynomiaal begrensd approximatie algorithme dat op GEOMETRIC-TRAVELING-SALESMAN een performance factor 3/2 haalt.

Bewijs

Beschouw in gedachten een kortste tour T, van lengte OPT. Laat men een kant weg, dan ontstaat een opspannende boom. Elke minimale opspannende boom heeft dus lengte $<$OPT. Neem zo'n minimale opspannende boom en beschouw de verzameling S van knopen van oneven graad daarin. S heeft noodzakelijk een even aantal elementen. Laat uit T alle punten niet in S weg, trek de verbindingslijnen recht en kleur kanten afwisselend rood en wit. (Omdat $|S|$ = even, krijgt elke kant precies één kleur.) Zowel de rode als de witte kanten vormen een perfecte matching (paring) op S. Elke minimale lengte perfecte matching kan dus niet meer dan $\frac{1}{2}$OPT lang zijn. Nu kan men zowel een minimale opspannende boom B als een minimale lengte perfecte matching M op zijn knopen van oneven graad in polynomiale tijd bepalen (e.g. LAWLER [29]). De graph die ontstaat door de kanten van M aan B toe te voegen is samenhangend, heeft louter knopen van even graad en bezit dus een Euler-pad. Zo'n pad is in polynomiale tijd te vinden (e.g. EVEN [16]) en zal lengte $<$OPT + $\frac{1}{2}$OPT = $\frac{3}{2}$OPT moeten hebben. Sla tijdens het doorlopen van het Euler-pad punten over zodra ze voor een tweede of latere keer bezocht worden en trek pas weer een directe "verbinding" als een nieuw punt wordt bereikt. De resulterende traveling salesman tour heeft lengte $\frac{3}{2}$OPT en het geschetste polynomiale approximatie algorithme haalt een performance factor 3/2. □

CHRISTOFIDES [11] geeft een goed overzicht van de thans bekende heuristieken voor (GEOMETRIC-)TRAVELING-SALESMAN. Optimale tours in het vlak doorsnijden zichzelf nooit, maar bijna-optimale tours die door approximatie-algorithmen worden afgeleverd voldoen vaak niet aan dat criterium. VAN LEEUWEN & SCHOONE [60] bewezen echter dat uit elk pad alle doorsnijdingen in polynomiale tijd verwijderd kunnen worden, indien zulks gewenst is.

Over KNAPSACK1 is ook wel wat te zeggen, indien we het omschrijven tot een optimaliseringsvraagstuk

> KNAPSACK1 : gegeven positieve natuurlijke getallen $a_1, \ldots, a_n$ en b, wat is de maximum waarde $\leq$b die een som $\sum_{i \in J} a_i$ met $J \subseteq \{1, \ldots, n\}$ kan hebben.

Definitie. Een approximatie algorithme voor een probleem heet een ε-approximatie algorithme indien voor alle toegelaten inputs

$$|\text{output} - \text{optimum}|/|\text{optimum}| \leq \varepsilon$$

JOHNSON [21] bewees

Stelling 4.3. Voor elke $\varepsilon > 0$ bestaat er een polynomiaal begrensd ε-approximatie algorithme voor KNAPSACK1.

Meer over approximatie algorithmen vindt men in [17]. SAHNI [46] geeft een overzicht van handige technieken die in approximatie algorithmen worden toegepast. In CHRISTOFIDES, MINGOZZI, TOTH & SANDI [12] worden heldere overzichten geboden van de vele exacte en heuristisch methoden voor oplossing van optimaliseringsvraagstukken.

5. REFERENTIES

[1] AHO, A.V., J.E. HOPCROFT & J.D. ULLMAN, The design and analysis of computer algorithms, Addison-Wesley Publ. Comp., Reading, Mass., 1974.

[2] BENTLEY, J.L., An introduction to algorithm design, CMU-CS-78-121, Carnegie Mellon University, Pittsburgh, 1978.

[3] BENTLEY, J.L., Decomposable searching problems, Inf. Proc. Lett. 8 (1979) 244-251.

[4] BENTLEY, J.L. & M.I. SHAMOS, Divide and conquer for linear expected time, Inf. Proc. Lett. 7 (1978) 87-91.

[5] BERMAN, L. & J. HARTMANIS, On isomorphisms and density of NP- and other complete sets, SIAM J. Computing 6 (1977) 305-323.

[6] BLUM, M., R.W. FLOYD, V.R. PRATT, R.L. RIVEST & R.E. TARJAN, Time bounds for selection, J. Comp. Syst. Sci. 7 (1972) 448-461.

[7] BORODIN, A.B., & I. MUNRO, The computational complexity of algebraic and numeric problems, Theory of Comput. Series, American Elsevier Publ. Comp., New York, 1975.

[8] BROWN, K.Q., Geometric transforms for fast geometric algorithms, Ph. D. Thesis, CMU-CS-80-01, Carnegie Mellon University, Pittsburgh, 1980.

[9] BUNCH, J. & J.E. HOPCROFT, Triangular factorization and inversion by fast matrix multiplication, Math. Comp. 28 (1974) 231-236.

[10] CHRISTOFIDES, N., Worst case analysis of a new heuristic for the traveling salesman problem, Techn. Rep., Grad. School of Industr. Admin., Carnegie Mellon University, Pittsburgh, 1976.

[11] CHRISTOFIDES, N., The travelling salesman problem, Chapter 6 (pp. 131-149) in [12].

[12] CHRISTOFIDES. N., A. MINGOZZI, P. TOTH & C. SANDI, Combinatorial optimization, Wiley & Sons, Chichester, 1979.

[13] CLANCY, M.J. & D.E. KNUTH, A programming and problem solving seminar, STAN-CS-77-606, Stanford University, Stanford, 1977.

[14] COOK, S.A., The complexity of theorem proving procedures, Proc. 3rd Annual ACM Symp. on Theory of Computing, 1971, pp. 151-158.

[15] DE JONG, L.S. & J. VAN LEEUWEN, An improved bound on the number of multiplications and divisions necessary to evaluate a polynomial and all its derivatives, SIGACT NEWS 7 (1975) 32-34.

[16] EVEN, S., Graph algorithms, Comp. Sci. Press, Potomac, 1979.

[17] GAREY, M.R. & D.S. JOHNSON, Computers and intractability: a guide to the theory of NP-completeness, Freeman & Comp., San Francisco, 1979.

[18] HOPCROFT, J.E., Complexity of computer computations, Proc. IFIP congres 74, North Holland Publ. Comp., Amsterdam, 1974, pp. 620-626.

[19] HOROWITZ, E. & S. SAHNI, Fundamentals of data structures, Computer Science Press Inc., Potomac, 1976.

[20] ITAI, A. & Y. SHILOACH, Maximum flow in planar graphs, Techn. Rep. 88, Technion, Haifa, 1979.

[21] JOHNSON, D.S., Approximation algorithms for combinatorial problems, J. Comp. Syst. Sci. 9 (1974) 256-278.

[22] KARP, R.M., Reducibility among combinatorial problems, in: R.E. MILLER & J.W. THATCHER (ed.), Complexity of computer computations, Plenum Press, New York, 1972, pp. 85-103.

[23] KARP, R.M., Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane, Math. Oper. Res. 2 (1977) 209-224.

[24] KARP, R.M., Combinatorial search problems: is exponential growth inevitable?, UC Electronics Newsletter, University of California, Berkeley, 1979, pp. 6-9.

[25] KIRKPATRICK, D.G., Optimal search in planar subdivisions, Techn. Rep., Dept. of Computer Science, University of British Columbia, Vancouver, 1979.

[26] KNUTH, D.E., Optimum binary search trees, Acta Inform. 1 (1971) 14-25.

[27] KNUTH, D.E., The art of computer programming, vol. 3: searching and sorting, Addison-Wesley Publ. Comp., Reading, Mass., 1973.

[28] LADNER, R.E., On the structure of polynomial time reducibility, J. ACM 22 (1975) 155-171.

[29] LAWLER, E.L., Combinatorial optimalization: networks and matroids, Holt, Rinehart & Winston, New York, 1976.

[30] LIPTON, R.J. & R.E. TARJAN, A separator theorem for planar graphs, STAN-CS-77-627, Stanford University, Stanford, 1977.

[31] LIPTON, R.J. & R.E. TARJAN, Applications of a planar separator theorem, STAN-CS-77-628, Stanford University, Stanford, 1977.

[32] LORIN, H., Sorting and sort systems, Syst. Progr. Series, Addison-Wesley Publ. Comp., Reading, Mass., 1975.

[33] LOVáSZ, L., A new linear programming algorithm - better or worse than the simplex method, Math. Intell. 2 (1980) 141-146.

[34] LUKS, E.M., Isomorphism of graphs of bounded valence can be tested in polynomial time, Proc. 21st Annual IEEE Symp. on Foundations of Computer Science, 1980, pp. 42-49.

[35] MEAD, C. & L. CONWAY, Introduction to VLSI systems, Addison-Wesley Publ. Comp., Reading, Mass., 1980.

[36] MEHLHORN, K., Dynamic binary search, SIAM J. Computing 8 (1979) 175-198.

[37] MEYER, A.R., Discrete computation: theory and open problems, MAC Techn. Memo 39, Lab. for Computer Science, MIT, Cambridge, 1974.

[38] MURAOKA, Y. & D.J. KUCK, On the time required for a sequence of matrix products, C. ACM 16 (1973) 22-26.

[39] OVERMARS, M.H. & J. VAN LEEUWEN, Dynamic multidimensional data-structures based on quad- and k-d trees, RUU-CS-80-2, University of Utrecht, Utrecht, 1980.

[40] PAPADIMITRIOU, C.H., The euclidean traveling salesman problem is NP-complete, Theor. Comp. Sci. 4 (1977) 237-244.

[41] PHILIPP, R. & E.J. PRAUSS, Über Separatoren in planaren Graphen, Acta Inform. 14 (1980) 87-106.

[42] PREPARATA, F.P. & S.J. HONG, Convex hulls of finite sets of points in two and three dimensions, C. ACM 20 (1977) 87-93.

[43] RABIN, M.O., Probabilistic algorithms, in: J.F. TRAUB (ed.), Algorithms and complexity: new directions and recent results, Acad. Press, New York, 1976, pp 21-39.

[44] RABIN, M.O., Complexity of computations, C. ACM 20 (1977) 625-633.

[45] REINGOLD, E.M., On the optimality of some set algorithms, J. ACM 19 (1972) 649-659.

[46] SAHNI, S., General techniques for combinatorial approximation, Oper. Res. 25 (1977) 920-936.

[47] SAHNI, S. & T. GONZALEZ, P-complete approximation problems, J. ACM 23 (1976) 555-565.

[48] SCHÖNHAGE, A. en anderen, te verschijnen.

[49] SCHRIJVER, A., Khachian's ellipsoide methode voor lineaire programmering, manuscript, Mathematisch Centrum, Amsterdam, 1980.

[50] SHAMOS, M.I., Geometry and statistics: problems at the interface, in: J.F. TRAUB (ed.), Recent results and new directions in algorithms and complexity, Acad. Press, New York, 1976, pp 251-280.

[51] SHAMOS, M.I., Computational geometry, Ph.D. Thesis, Dept. of Computer Science, Yale University, New Haven, 1978 (te verschijnen).

[52] STANDISH, T.A., Data structure techniques, Addison-Wesley Publ. Comp., Reading, Mass., 1980.

[53] STOCKMEYER, L.J. & A.K. CHANDRA, Intrinsically difficult problems, Scientific American (1979) 140-159.

[54] STRASSEN, V., Gaussian elimination is not optimal, Num. Math. 13 (1969) 354-356.

[55] TARJAN, R.E., Complexity of combinatorial problems, STAN-CS-77-609, Stanford University, Stanford, 1977.

[56] VALIANT, L.G., Relative complexity of checking and evaluating, Inf. Proc. Lett. 5 (1976) 20-23.

[57] VALIANT, L.G., The complexity of computing the permanent, Theor. Comp. Sci. 8 (1979) 189-202.

[58] VALIANT, L.G., The complexity of enumeration and reliability problems, SIAM J. Computing 8 (1979) 410-421.

[59] VAN EMDE BOAS, P., Complexity of linear problems, Rep. 79-06, Dept. of Mathematics, University of Amsterdam, Amsterdam, 1979.

[60] VAN LEEUWEN, J. & A.A. SCHOONE, Untangling a traveling salesman tour in the plane, RUU-CS-80-11, University of Utrecht, Utrecht, 1980.

[61] VAN LEEUWEN, J. & P. VAN EMDE BOAS, Some elementary proofs of lowerbounds in complexity theory, J. Lin. Alg. and its Appl. 19 (1978) 63-80.

[62] VAN LEEUWEN, J. & D. WOOD, Dynamization of decomposable searching problems, Inf. Proc. Lett. 10 (1980) 51-56.

[63] VITANYI, P.M.B., Berekeningsmodel en complexiteit, manuscript, Mathematical Centre, Amsterdam, 1980.

[64] WEIDE, B.W., Statistical methods in algorithm design and analysis, Ph.D. Thesis, CMU-CS-78-142, Carnegie Mellon University, Pittsburgh, 1978.

[65] YAO, A.C., A lower bound to finding convex hulls, STAN-CS-79-733, Stanford University, Stanford, 1979.

[66] YAO, F.F., Efficient dynamic programming using quadrangle inequalities, Proc. 12th Annual ACM Symp. on Theory of Computing, 1980, pp. 429-435.