Hans L. Bodlaender
Departement Informatica, Universiteit Utrecht, en,
Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven
h.l.bodlaender@uu.nl

Oratie

Hoe verbonden is je

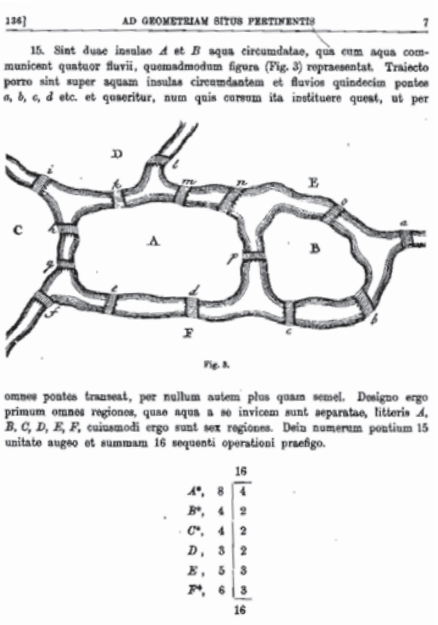
In 2014 werd Hans Bodlaender benoemd tot hoogleraar Algorithms and Complexity aan de Universiteit Utrecht. Zijn onderzoek richt zich op complexe netwerken, waarbij de eigenschappen 'samenhang' en 'verbondenheid' een belangrijke rol spelen. Dit artikel is een verkorte en bewerkte versie van zijn oratie, uitgesproken op 12 oktober 2015.

In dit verhaal wil ik het hebben over algoritmen, netwerken en complexiteit, en de vraag "Hoe verbonden is je netwerk?" loopt daar op verschillende manieren doorheen. Eerst kijken we naar het onderwerp van de netwerken met de bijbehorende wiskundige structuren die *grafen* worden genoemd.

De bruggen van Königsberg

Het begin van het wetenschappelijk onderzoek aan grafen en netwerken wordt geplaatst rond het jaar 1736 in het Russische stadje Königsberg, het tegenwoordige Kaliningrad. Midden door de stad loopt de rivier de Pregel, en midden in de rivier lagen een aantal eilanden. De eilanden en oevers waren verbonden met een aantal bruggen.

Het verhaal gaat dat de inwoners van Königsberg de volgende vraag aan de wiskundige Euler stelden: is het mogelijk een rondwandeling te maken zodat elke brug precies één keer gepasseerd wordt? Tegenwoordig noemen we zo'n route een Eulertour. Samen met Euler kunnen we snel zien dat voor het achttiende-eeuwse Königsberg zo'n Eulertour niet bestaat: in een Eulertour gaan we even vaak naar een eiland toe als dat we het verlaten, dus een noodzakelijke voorwaarde voor het bestaan van een Eulertour is dat elk eiland (of oever) een even aantal bruggen heeft, en Königsberg heeft een eiland met drie bruggen. Als we het netwerk als een graaf modelleren, dan betekent dit dat elke knoop even graad moet hebben (een even aantal grenzende kanten).


136] AD GEOMETRIAM SITUS PERTINENTIS 7

15. Sint duae insulae A et B aqua circumdatae, qua cum aqua communicent quatuor fluvii, quemadmodum figura (Fig. 3) repraesentat. Traiecto porro sint super aquam insulas circumdantem et fluvios quindecim pontes a, b, c, d etc. et quaeritur, num quis earum ita instituere queat, ut per

Fig. 3.

omnes pontes transeat, per nullum autem plus quam semel. Designo ergo primum omnes regiones, quae aqua a se invicem sunt separatae, litteris A, B, C, D, E, F, cuiusmodi ergo sunt sex regiones. Dein numerum pontium 15 unitate augeo et summam 16 sequenti operationi praefigo.

16
A^a, 8 | 4
B^a, 4 | 2
C^a, 4 | 2
D, 3 | 2
E, 5 | 3
F^a, 6 | 3
16

Figuur 1 Een bladzijde uit Eulers artikel.

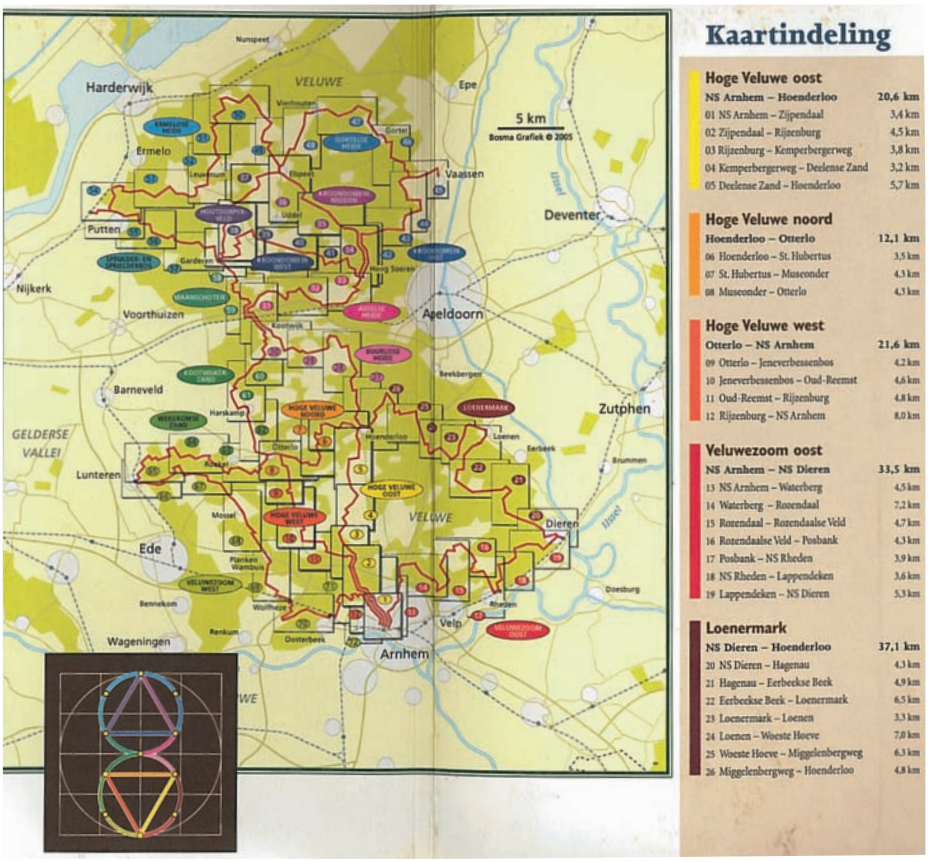
Er is nog een tweede makkelijk te controleren noodzakelijke voorwaarde voor het bestaan van een Eulertour. In IJsland wil wel eens een vulkaanuitbarsting een vloedgolf in een rivier teweegbrengen, die bruggen wegspoelt. Als nu de rivier het bruggenstelsel in twee delen splitst die onderling niet verbonden zijn en beide gedeeltes hebben minstens één brug, dan zal een wandeling altijd aan één kant van de rivier moeten blijven en er daarom geen Eulertour zijn. Dat levert het tweede criterium op: het netwerk van bruggen moet samenhangend zijn.

Eulers vondst [10] was dat deze twee voorwaarden ook voldoende zijn: een samenhangend netwerk van bruggen waarbij elke locatie een even aantal bruggen heeft, heeft altijd een Eulertour. Eulers oorspronkelijke artikel bevat overigens niet het bewijs van deze stelling; het eerst bekende bewijs is van de Duitse wiskundige Carl Hierholzer uit 1871 [13]. Hierholzers bewijs is constructief: het levert ook een snelle methode op om de route te construeren als deze bestaat.

Een aardig voorbeeld van een graaf die een Eulertour laat zien is een schematische

Hans Bodlaender

netwerk?

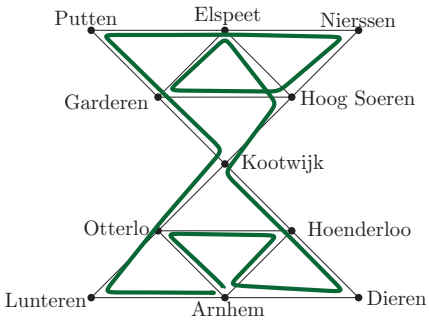


Figuur 2 Kaart van het Veluwe Zwerfpad.

weergave van het Veluwe Zwerfpad [17], zie Figuur 2 en 3. Het Veluwe Zwerfpad is een van de streekpaden in Nederland, een prachtige wandelroute van totaal 357 kilometer. Als we een paar delen waar trajecten overlappen buiten beschouwing laten, dan legt de wandelaar tijdens deze 357 kilometer een Eulertour door de Veluwe af. (De onlangs verschenen derde druk heeft een iets gewijzigde route, maar volgt nog steeds een Eulertour.)

De methode van Hierholzers bewijs is als volgt: begin met een rondwandeling die een deel van de kanten eenmaal gebruikt; zolang niet alle kanten gebruikt zijn, kan je ergens beginnen, net zo lang doorgaan tot je weer op dat punt begonnen bent, en die tweede rondwandeling invoegen in de eerste, dit herhalend tot alle kanten gebruikt zijn.

Uit dit bewijs krijgen we een algoritme: een stappenplan dat op een precieze manier voorschrijft hoe met een gegeven invoer de gewenste uitvoer kan worden berekend; in dit geval bepaalt het voor



Figuur 3 Schematische weergave van het Veluwe Zwerfpad met een Eulertour.

een gegeven graaf of deze een Eulertour heeft en zo ja, levert het zo'n tour op. Het woord *algoritme* is een verwijzing naar de Perzische wetenschapper Mohammed ibn Moesa al-Chwarizmi die rond het jaar 800 leefde. Een algoritme kan door een mens worden uitgevoerd, maar liever maken we tegenwoordig een programma zodat een computer voor ons het algoritme uitvoert.

De formalisering van het begrip algoritme vond met name plaats in het midden van de twintigste eeuw, waarbij het werk van Alan Turing voor de algoritmiek van essentieel belang is. De film *The Imitation Game* brengt op een prachtige manier zowel veel aspecten van zijn leven als van zijn wetenschappelijk werk in beeld. Het onderwerp van de *complexiteit* wordt ook op een heel mooie manier in die film verbeeld; hierover later meer.

Het vakgebied van de algoritmiek legt zich toe op het ontwerpen en analyseren van algoritmen, voor allerlei soorten van vragen. Belangrijk is natuurlijk dat een algoritme correct is, en efficiënt. Efficiëntie wordt met name gemeten in de tijd die het algoritme gebruikt, maar kan ook gemeten worden op andere manieren, zoals bijvoorbeeld de hoeveelheid geheugen die het algoritme nodig heeft. We meten dan de tijd die een algoritme gebruikt als het aantal stappen dat het algoritme doet, in het algemeen als functie van het formaat of andere parameters van de invoer. Het Königsberger bruggenprobleem heeft een lineair algoritme: het aantal stappen dat nodig is om vast te stellen of een Eulertour bestaat, en zo ja deze te construeren, is hooguit een aantal keren het aantal knopen plus kanten in de graaf. Terwijl het Eulertourprobleem een efficiënt algoritme heeft, is dit niet het geval voor veel andere problemen. Vaak wordt de grens tussen doenbaar en ondoenbaar gelegd bij wanneer het algoritme tijd gebruikt die begrensd is door een polynoom van de grootte van de invoer, zogenaamde polynomiale algoritmen. Stel dat de inwoners van Königsberg vragen om een wandeling waarbij ze zo veel mogelijk bruggen willen gebruiken, elke brug nog steeds één keer. Dit probleem is NP-volledig — dat wil zeggen, dat het onwaarschijnlijk is dat er een polynomiaal algoritme voor is. (Wat preciezere terminologie: het probleem om een rondwandeling met een maximum aantal bruggen te vinden is *NP-moeilijk*; het is *NP-volledig* om, gegeven een netwerk en een getal K, te bepalen of er een rondwandeling met minstens K bruggen bestaat.) Het probleem is makkelijk op te lossen voor een klein netwerk — voor de bruggen van Königsberg is het makkelijk alle mogelijkheden na te gaan, maar het aantal verschillende mogelijkheden groeit als een exponentiële functie met het formaat van het netwerk. Dit effect heet wel de *combinatoriële explosie*.

Die combinatorische explosie wordt mooi geïllustreerd in de film *The Imitation Game*, die ik al eerder noemde. In de Tweede Wereldoorlog geeft Turing leiding aan een team dat de geheime code van de Enigma, een Duitse codeermachine, probeert te kraken. Elke ochtend is er een nieuwe code, waarmee die dag boodschappen van het Duitse leger worden versleuteld. In de film zien we een door Turing en zijn team ontworpen machine, die dit probleem moet oplossen. Het aantal door te rekenen oplossingen is echter zo groot, dat de machine door blijft rekenen zonder iets te antwoorden; en de berekening wordt de volgende ochtend nutteloos als de machine nog rekent op de code van de vorige dag. Een grotere machine helpt niet. Wat wel blijkt te helpen is gebruik te maken van een speciale eigenschap van de invoer: elke ochtend wordt het weerbericht gecodeerd doorgegeven, en door dit gegeven te gebruiken kan het team van Turing de machine de code laten breken.

De verschillende vragen die we over een combinatorische structuur, zoals een netwerk, kunnen stellen, kunnen dus heel verschillende complexiteit hebben: een kleine variatie op het Eulertourprobleem verandert van een snel oplosbare vraag in een vraag waar berekeningen veel tijd kosten. Het vaststellen van de complexiteit van een probleem kan op verschillende manieren: een algoritme voor het probleem kan op wiskundige manier worden geanalyseerd, waarbij vooral (maar zeker niet als enige) gekeken wordt naar de asymptotische looptijd van een algoritme als functie van de invoergrootte; de complexiteitstheorie geeft een heel arsenaal van hulpmiddelen om ondergrenzen te verkrijgen — deze laten ons zien wat voor soort algoritmen waarschijnlijk niet gemaakt kunnen worden.

Ik noemde al het begrip NP-volledigheid. Dit begrip werd in de zeventiger jaren van de twintigste eeuw ontwikkeld door Cook en Levin [6, 14], en is een van de belangrijkste voorbeelden, misschien wel hét belangrijkste voorbeeld van dit soort gereedschap. Daarnaast is de experimentele analyse belangrijk: uitprogrammeren en kijken voor geschikte testinvoeren hoe het algoritme zich in de praktijk gedraagt. Het vakgebied van de theoretische informatica heeft hier niet altijd voldoende oog voor; mijn ervaring is dat de theoretische/wiskundige analyse en het experiment elkaar — in beide richtingen — versterken.

Naast de combinatorische explosie hebben we tegenwoordig ook de *data-explosie*. Zoveel met elkaar verbonden apparaten en mensen produceren een gigantische hoeveelheid gegevens — dan wordt het vinden van goede antwoorden op een vraag over die gegevens ook een hele klus.

Algoritmen en netwerken

We zagen dat een netwerk van eilanden en bruggen kan worden gemodelleerd als een (eindige) graaf. We gebruiken de term *netwerk* als we extra informatie hebben voor de knopen en kanten in het netwerk, bijvoorbeeld lengtes of geografische locaties.

In veel toepassingen komen we structuren tegen die met grafen of netwerken kunnen worden gemodelleerd. Vanuit de toepassingen komt een veelvoud van vragen over de netwerken — soms willen we inzicht krijgen in de structuur van het netwerk; soms helpt het netwerk een heel andere vraag uit een toepassing te beantwoorden; soms vraagt een planningsprobleem om een oplossing.

Netwerkmodellen en vragen

Bij veel van die vragen spelen samenhang en verbondenheid van het netwerk een belangrijke rol.

Een van de manieren waarop samenhang in een netwerk wordt gemeten is de minimumsnede: voor twee knopen, hoeveel kanten moet je weghalen zodat de ene knoop niet te bereiken is vanaf de andere? In 1956 bereikten Ford en Fulkerson [7, 12] een klassiek resultaat: je kan dit efficiënt bepalen met behulp van stromingstechnieken. Het resultaat kan je ook zien als een wiskundige illustratie bij het feit dat verkeersopstoppingen vaak bij bruggen en tunnels zijn: de maximumhoeveelheid die je van A naar B kan vervoeren is gelijk aan de minimumsnede, en zo'n minimumsnede zal vaak door een rivier gevormd worden, want het bouwen van bruggen en

tunnels is veel duurder dan het bouwen van wegen.

Samenhang en verbondenheid in netwerken kom je in veel toepassingen tegen. In een aantal voorbeelden uit het werk van informatici en wiskundigen uit Utrecht en Eindhoven zien we die begrippen een belangrijke rol spelen in onderzoek naar kennisuitwisseling in bedrijven (Remko Helms), belissingsondersteuning met Bayesiaanse netwerken (Linda van der Gaag), frequentietoewijzing en andere vragen rond mobiele (telefoon)netwerken (Mark de Berg, Erik Jan van Leeuwen), optimale planning voor elektriciteitsnetwerken (Marjan van den Akker, Han Hoogeveen), software ecosystemen (Slinger Jansen), en natuurkundig onderzoek naar polymeren vertaalt zich naar het tellen van cycli in een gridnetwerk (Gerard Barkema, Rob Bisseling, Raoul Schram).

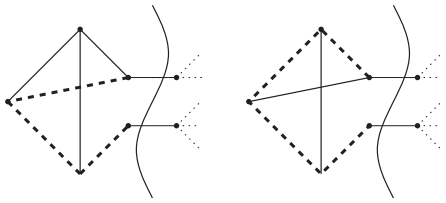
Bij deze en soortgelijke vragen is de uitdaging aan de algoritmicus om te komen met efficiënte algoritmen — "kunnen de vragen binnen redelijke tijd worden beantwoord?" Inzicht in de complexiteit van het probleem helpt daarbij.

Naast het ontwerpen van algoritmen voor *concrete* problemen willen we ook nieuwe *technieken* ontwerpen om op die manier de gereedschapskist van de algoritmeontwerper te vergroten. Ik bespreek nu een paar van zulke technieken.

Dynamisch programmeren, boomstructuren

Een belangrijke algoritmische techniek is *dynamisch programmeren*. De techniek en term komen uit het werk van Richard Bellman in de veertiger jaren van de vorige eeuw [1]. Het basisidee is om deelproblemen te identificeren, de antwoorden op deze deelproblemen op te slaan en zo te vermijden dat deze antwoorden vaker worden uitgerekend, en op die manier rekentijd te besparen.

Laten we eens opnieuw kijken naar de vraag van een zo lang mogelijke rondwandeling die elke brug hooguit één keer gebruikt. We merken eerst op dat Eulers resultaat ons toestaat om het anders te formuleren: vind een zo groot mogelijk samenhangend deel van de kanten van het netwerk waarbij elke knoop een even aantal kanten ziet. Als we die eis van samenhangendheid niet hadden, zou het probleem vrij makkelijk zijn — het is een variatie op het koppelingsprobleem en kan met Edmonds algoritme uit 1965 [9] worden opgelost. Met die samenhangendheid is het, zoals gezegd, NP-moeilijk.



Figuur 4 Twee deelroutes.

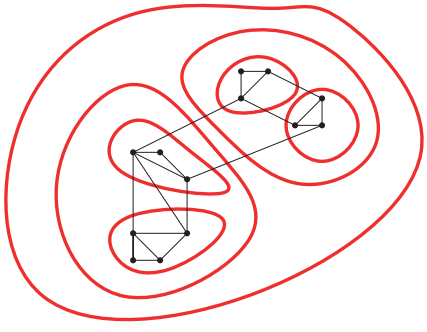
Maar stel nu dat midden door ons landschap een grote rivier stroomt met twee bruggen. We kunnen nu kijken wat de route doet aan de linkerkant van de rivier en aan de rechterkant: het is voldoende om te weten wat de langste rondwandeling is aan de linkerkant, de langste rondwandeling aan de rechterkant, en voor beide kanten van de rivier, het langste pad van de ene brug naar de andere dat aan één kant van de rivier blijft en steeds elke brug maar hooguit één keer gebruikt. In Figuur 4 zien we twee mogelijke routes aan de linkerkant van een rivier: de eerste zal nooit leiden tot een optimale oplossing, want hoe we die ook uitbreiden, de tweede kan op dezelfde manier worden uitgebreid tot een langere route. Die eerste route kan dus vergeten worden. Je zou een tabel kunnen maken van alle essentieel verschillende deelroutes aan de linkeroever, of, beter, voor elke karakterisering van een deelroute aan de linkeroever, de maximale lengte opslaan. Als er twee bruggen over de rivier zijn, zijn er twee waarden om op te slaan; bij een groter aantal bruggen groeit dit aantal, maar we kunnen — als het aantal bruggen niet te groot is — nog steeds ons beperken tot een tabel met veel minder gegevens dan het totaal aantal mogelijke routes aan de linkeroever.

We zien hier een netwerk met een lage samenhangendheid — door twee kanten weg te halen valt het netwerk in meerdere delen uiteen. Dat gebrek aan samenhang kan worden uitgebuit door het probleem te splitsen in kleinere delen. Als het aantal verbindingen tussen de twee delen groter dan twee is, dan stijgt het aantal gevallen dat we moeten bekijken, maar met een klein aantal verbindingen blijft het aantal gevallen beperkt genoeg. Sommige probleeminstanties laten toe dat we de delen weer verder opsplitsen in weer nog kleinere gedeeltes. Op deze manier kan je een geneste structuur van deelnetwerken krijgen, waarbij de essentiële informatie van een deelnetwerk te bepalen valt uit de essentiële informatie van de direct daar onderliggende delen. Zie Figuur 5.

Dit idee kan op een aantal verschillende manieren vorm krijgen. Een heel succesvolle is de *boomdecompositie* met het daarbij horende begrip *boombreedte* (of, in het Engels, 'treewidth'). Het begrip generaliseert de bomen, en de series-parallelle netwerken die sommige van ons op de middelbare school hebben gezien (bijvoorbeeld bij het bepalen van de weerstand van een elektrisch netwerk). Een boomdecompositie is een alternatieve representatie van een graaf of netwerk, die ons in staat stelt om het probleem met behulp van dynamisch programmeren op te lossen. Hoe kleiner de boombreedte, hoe sneller het algoritme, hoe minder tijd.

Het gaat te ver om hier de wat technische definitie van boombreedte te geven; de intuïtie is de volgende: in de voorbeelden hierboven splitsten we het netwerk op de kanten — door bruggen over rivieren weg te denken; bij boombreedte doen we iets soortgelijks, maar hakken we knopen door.

In de afgelopen decennia zijn boomdecomposities in veel gebieden met succes toegepast, zowel in de algoritmische grafentheorie, maar ook in veel toepassingsgebieden, waaronder operations research, en ook computationele biologie en logica. Een belangrijk gebied waar ze ook worden gebruikt zijn de probabilistische netwerken, zoals die door Linda van der Gaag en haar groep bestudeerd worden (bijvoorbeeld [15]). Een centraal probleem in deze netwerken is het uitrekenen van een kansverdeling, gegeven een aantal observaties (het *inferentie-probleem*): dit probleem is in zijn algemeenheid moeilijk (NP-moeilijk, en zelfs moeilijk voor de gro-



Figuur 5 Geneste dissectie.

tere complexiteitsklasse $\#P$), maar als de boombreedte van een geassocieerde graaf begrensd is door een constant getal dan is het inferentie-probleem in lineaire tijd op te lossen.

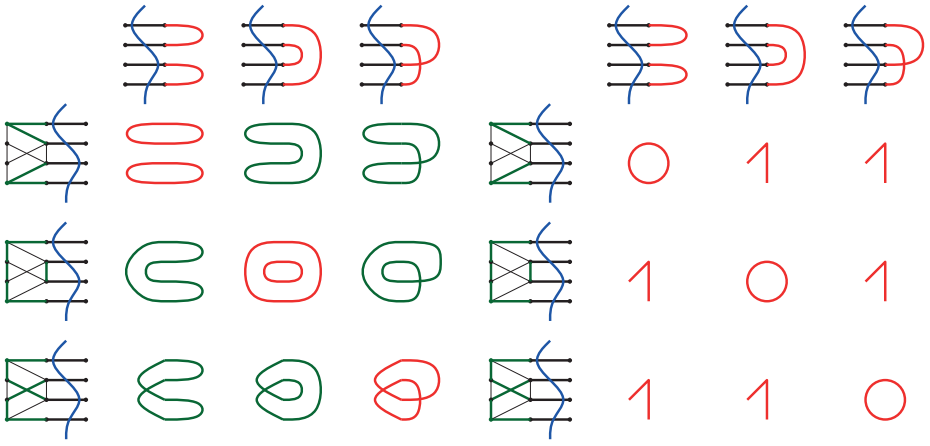
Voor ons probleem van de langste rondwandeling die elke brug hooguit één keer gebruikt, geldt ook dat als de boombreedte van de graaf begrensd is door een constant getal, we het probleem op kunnen lossen in tijd die slechts lineair is in de grootte van het netwerk. Daar staat wel tegenover dat de tijd exponentieel groeit met de boombreedte.

Belangrijke vragen om te beantwoorden zijn: hoe vind je boomdecomposities en hoe kan je algoritmen met behulp van deze structuur zo efficiënt mogelijk maken? Het algoritmisch onderzoek in Utrecht heeft de laatste jaren daar veel aan bijgedragen. Naast efficiënte algoritmen voor het vinden van boomdecomposities en het bepalen van de boombreedte van grafen, het oplossen van allerlei concrete problemen met behulp van deze structuren, hebben we ook een aantal nieuwe technieken ontworpen die dit soort algoritmen versnellen.

Het gaat er bij dynamisch programmeren om om te zorgen dat je geen zaken voor de tweede keer uitrekent. Door goed te kijken naar de structuur van het probleem dat je op wil lossen en naar de structuur van het netwerk, kan je verdere winst boeken. In het werk van Johan van Rooij zien we hoe snellere algoritmen bereikt kunnen worden door representaties van de deelproblemen te transformeren met behulp van *convoluties* [16]. Een prachtige techniek zien we in het werk met Marek Cygan, Jesper Nederlof en Stefan Kratsch [3]. We kijken eens naar een rivier met vier bruggen. Een dynamisch programmeeralgoritme rekent voor de linkeroever van de rivier een tabel uit. De tabel zou kunnen bestaan uit een collectie van deelroutes die zich geheel aan de linkeroever bevinden, met als eigenschap dat er een optimale route in het hele netwerk bestaat die een van de elementen van de tabel als deel bevat.

Als de tabel de volgende drie deeloplossingen bevat, dan kunnen we schematisch kijken naar alle essentieel verschillende manieren waarop die tot een volle tour uitgebreid kunnen worden.

In Figuur 6 ziet u de deeloplossingen, met de uitbreidingen. De uitbreidingen op de hoofddiagonaal geven geen correcte oplossingen, want ze bestaan uit meerdere cycli; de andere zes uitbreidingen geven wel een correcte oplossing. We zien dat de derde deeloplossing niet nodig is: elke uitbreiding van de derde deeloplossing die een tour geeft, werkt ook met de eerste of de tweede. Die derde kan uit de tabel weggelaten worden bij het dynamisch programmeren.



Figuur 6 Deeloplossingen en uitbreidingen.

Figuur 7 Matrix met binaire waarde voor deeloplossingen en uitbreidingen.

Het aardige is dat een *representatieve verzameling* van deeloplossingen gevonden kan worden door Gauss-eliminatie (modulo twee rekenend) los te laten op een matrix als getoond in het voorbeeld, waarbij we een goede combinatie met 1 en een slechte combinatie met 0 representeren, zie Figuur 7.

Het formaat van de tabel wordt dan begrensd door de *rang* van deze matrix en deze matrix blijkt een rang te hebben die veel kleiner is dan het aantal mogelijke combinaties. We zien een onverwacht verband tussen algoritmen op boomstructuren, en algebra, en voor allerlei problemen, inclusief het langste-cyclusprobleem, krijg je zo een significant sneller algoritme als de boombreedte klein genoeg is [3, 11].

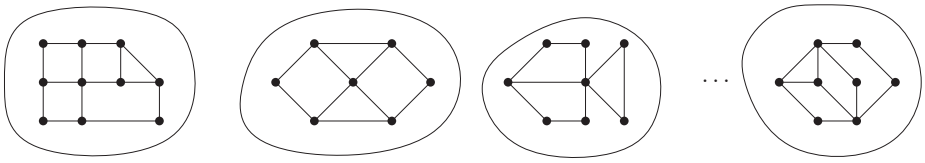
Naast boombreedte en boomdecomposities zijn een aanzienlijk aantal andere soortgelijke structuren en graafparameters voorgesteld en bestudeerd, met hun steeds wisselende eigenschappen, toepassingen, mogelijkheden, algoritmische vragen en oplossingen. Een zo'n graafparameter is *gonaliteit*, bestudeerd door Gunther Cornelissen [8], met belangrijke toepassingen in de getaltheorie.

Vaste-parametercomplexiteit

Het onderzoek aan boomstructuren is een voorbeeld van wat tegenwoordig *fixed parameter tractability* genoemd wordt. Uit de invoer lichten we een parameter, bijvoorbeeld de boombreedte, en kijken hoe snel we het probleem kunnen oplossen als die parameter klein is. Zo wordt de analyse van de complexiteit van een probleem veel diverser: voor verschillende parameters kan het probleem zich anders gedragen.

Soms kan een probleem dat in zijn algemeenheid niet snel genoeg is op te lossen dus wél efficiënt worden opgelost als er een speciale eigenschap van de invoer gebruikt kan worden. Bij het ontcijferen van de codes van de Enigma door Turing en zijn team was dat het vercijferde weerbericht; hier nemen we aan dat één van de parameters van de invoer klein is.

Als de wandelaars uit Königsberg vragen of er een rondwandeling is met minstens k bruggen, en we weten dat k een klein getal is, dan heeft het probleem een snel algoritme: we kunnen het oplossen in



Figuur 8 Compositionaliteit van het langste-cyclusprobleem.

tijd, die exponentieel is in k, maar lineair in het aantal knopen van de graaf. In dit geval is de doelwaarde, het aantal bruggen k, de uitgelichte parameter, en noemen we het probleem *fixed parameter tractable*.

Het vakgebied van de *vaste-parametercomplexiteit* kijkt naar dit soort algoritmen: algoritmen die efficiënt zijn als de bekeken parameter klein is.

Het gaat erom om van problemen met een uitgelichte parameter vast te stellen hoe snel ze oplosbaar zijn; we krijgen daarbij looptijden die een functie zijn van zowel de grootte van de invoer als van de betreffende parameter.

Het gebied van de vaste-parametercomplexiteit kent een ruim aantal technieken, zowel om vast te stellen dat onder complexiteitstheoretische aannames bepaalde algoritmen niet bestaan, als om vaste-parameteralgoritmen te ontwerpen. Eén van die technieken, *kernelisation* ('verkerning'), heeft daarbij een heel eigen plek, door de brede toepasbaarheid voor allerlei optimaliseringsproblemen.

Kernelisation

Als we een moeilijk probleem willen oplossen met een exact, maar wel langzaam algoritme, dan is het vaak zinvol om het probleem voor te bewerken; dat wil zeggen, te kijken of op een relatief eenvoudige manier de probleeminstantie omgevormd kan worden tot een equivalente maar wel kleinere instantie. In ons langstecyclusprobleem kunnen we bijvoorbeeld alle doodlopende wegen weglaten zonder het antwoord te beïnvloeden.

Zo'n voorbewerkingsalgoritme noemen we een *kernel* als het aan een aantal eisen voldoet: veiligheid, polynomiale tijd, en een grens op het formaat van resulterende instanties die alleen van de parameter afhangt. Een centraal resultaat is dat een beslisbaar geparameteriseerd probleem een kernel heeft, dan en slechts dan als het fixed parameter tractable is. Belangrijke vragen zijn hoe snel het kernelisationalgoritme is, en welke grens op de formaten er resulteert.

Opnieuw, naast veel resultaten voor concrete problemen zijn er ook een aantal algemene technieken ontwikkeld, waarbij ik hier vanwege de eigen bijdrage de metakernelisation [5], en ondergrenzen voor kernels [4] wil noemen.

Het centrale idee van die ondergrenzen wil ik hier uitlichten: het is een aardig voorbeeld van het exploiteren van het gebrek aan samenhangendheid van een netwerk.

Stel we hebben een instantie van het langste-cyclusprobleem, waarbij we een cyclus met k kanten willen vinden. Maar stel nu dat onze instantie bestaat uit een archipel met heel veel, zeg l, niet verbonden eilanden, met op elk eilandje een netwerkje, met l veel groter dan k.

Er is een cyclus met k kanten, dan en slechts dan als er tenminste één eiland is met zo'n cyclus. Die verschillende instanties op de eilanden hoeven niets met elkaar te maken te hebben. Het is daarom onwaarschijnlijk dat we snel deze invoer kunnen reduceren tot één met een aantal bits dat minder is dan het aantal eilanden dat we eerst hadden. Met zo'n archipelconstructie hebben we een voorbeeld van *compositionaliteit*: in dit geval laat het zien dat het langste-cyclusprobleem geen kernel met polynomiaal formaat heeft, onder een bepaalde complexiteitstheoretische aanname. Hier gebruiken we het gebrek aan verbondenheid tussen delen van de invoer om te beargumenteren dat bepaalde algoritmische oplossingen niet voorhanden zijn.

Een variatie op het begrip kernel is de polynomiale Turing-kernel; hier vertaal je niet een invoer naar één kleine, equivalente invoer, maar mag je dit een aantal keren doen, op een interactieve manier, Wat preciezer: je hebt een polynomiaal algoritme dat een orakel mag raadplegen dat het antwoord geeft voor instanties met formaat polynomiaal in de parameter. Aardig is dat het langste-cyclusprobleem wél een polynomiale Turing-kernel heeft: in een email-uitwisseling met Bart Jansen zagen we dat dit kan worden verkregen door gebruik te maken van een stelling van Bilinski en medeauteurs van 2011 [2], waaruit volgt dat als we een zogenaamde 3-kant samenhangende component in de graaf hebben met meer dan $k^{1,33}$ kanten dan heeft die component een cyclus met k kanten.

We zien hier dat algoritmische technieken hand in hand gaan met inzichten die komen uit de structuur van het netwerk — vaak moet je om een snel algoritme te ontwerpen eerst een graaftheoretische stelling bewijzen, of in de literatuur vinden — en opnieuw, omgekeerd is een algoritmische vraag vaak het startpunt van een studie naar de structuur van grafen.

Fundamenteel onderzoek in verbinding

Ik heb net twee nieuwe technieken aangestipt: een om snellere algoritmen te vinden, en een om beter de complexiteit van een probleem te kunnen vaststellen. Het vakgebied van de algoritmen en complexiteit heeft een flink arsenaal van zulke technieken, en het ontwikkelen van nieuwe algoritmische technieken of deze op nieuwe manieren toepassen of combineren, geeft belangrijke stappen in de ontwikkeling van het gebied.

Zulke fundamentele resultaten zijn erg het nastreven waard, maar laten die zich plannen...? Nu, het begint altijd met het *stellen van een goede vraag*, en daarna het goed kijken naar (de essentiële elementen in) het antwoord op die vraag.

Verschillende gebieden waaruit zulke goede (algoritmische) vragen kunnen komen zijn: de grafentheorie; de computationele geometrie; de algebraïsche geometrie; toepassingen zoals ik eerder besprak, maar ook softwaretechnologie; planningsproblemen uit de operations research, bijvoorbeeld uit de planning van openbaar vervoer; uit de analyse van complexe systemen; of vanuit de gametechnologie.

Fundamentele vragen liggen bijvoorbeeld op het gebied van de parameterisering van polynomiaal oplosbare problemen op heel grote netwerken: kunnen technieken uit de vaste-parameteralgoritmiek leiden tot nieuwe fundamentele inzichten in de 'big data'?

Een sterke koppeling met het wetenschappelijk onderwijs is iets waar groot belang aan moet worden gehecht. Het terrein van de algoritmiek is uitstekend geschikt om studenten te laten ervaren wat het is om te werken aan de frontlinie van de wetenschap, en daar belangrijke bijdragen te leveren.

In het fundamenteel onderzoek in de algoritmiek wordt het belang van experimentele verificatie van theoretisch verkregen resultaten vaak onderschat. Mijn ervaring is dat experimenteel en fundamenteel onderzoek elkaar kunnen versterken en inspireren.

Onderzoek doe je niet op een spreekwoordelijk eiland: onderzoeksonderwerpen hebben — vaak verrassende — verbanden met elkaar, en de verbindingen tussen onderzoekers zijn essentieel om die verbanden tot hun volle recht te laten komen; goede samenhang van het netwerk van onderzoekers doet het onderzoek aan kracht en belang winnen.

Referenties

1 R.E. Bellman en S.E. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, 1962.

2 M. Bilinski, B. Jackson, J. Ma en X. Yu. Circumference of 3-connected claw-free graphs and large Eulerian subgraphs of 3-edge-connected graphs, *Journal of Combinatorial Theory*, *Series B* 101(4) (2011), 214–236.

3 H.L. Bodlaender, M. Cygan, S. Kratsch en J. Nederlof, Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth, *Information and Computation* 243 (2015), 86–111.

4 H.L. Bodlaender, R.G. Downey, M.R. Fellows en D. Hermelin, On problems without polynomial kernels, *Journal of Computer and System Sciences* 75(8) (2009), 423–434.

5 H.L. Bodlaender, F.V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh en D.M. Thilikos, (Meta) kernelization, in *Proceedings of the 50th Annual Symposium on Foundations of Computer Science, FOCS 2009*, pp. 629–638. Te verschijnen in *Journal of the ACM*.

6 S.A. Cook, The complexity of theorem-proving procedures, in *Proceedings of the 3rd Annual Symposium on Theory of Computing, STOC'71*, ACM, 1971, pp. 151–158,.

7 T.H. Cormen, C.E. Leiserson, R.L. Rivest en C. Stein, *Introduction to Algorithms, Second Edition*, MIT Press, 2001.

8 G. Cornelissen, F. Kato en J. Kool, A combinatorial Li-Yau inequality and rational points on curves, *Mathematische Annalen* 361 (2012), 211–258.

9 J. Edmonds, Paths, trees, and flowers, *Canadian Journal of Mathematics* 17 (1965), 445–467.

10 L. Euler, Solutio problematis ad geometriam situs pertinentis, *Commentarii Academiae Scientiarum Petropolitanae* 8 (1736), 128–140.

11 S. Fafianie, H.L. Bodlaender en J. Nederlof, Speeding up dynamic programming with representative sets – an experimental evaluation of algorithms for steiner tree on tree decompositions, in *Proceedings IPEC 2013*, Lecture Notes in Computer Science, Vol. 8246, Springer, 2013, pp. 321–334.

12 L.R. Ford Jr. en D.R. Fulkerson, Maximal flow through a network, *Canadian Journal of Mathematics* 8 (1956), 399–404.

13 C. Hierholzer, Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren, *Mathematische Annalen* VI (1873), 30–32.

14 L. Levin, Universal search problems (in Russian), *Problems of Information Transmission* 3 (1973), 115–116.

15 A. Pastink en L.C. van der Gaag, Multi-classifiers of small treewidth, in *Proceedings ECSQARU 2015*, Lecture Notes in Computer Science, Vol. 9161, Springer, 2015, pp. 199–209.

16 *Streekpad 16 – Veluwe Zwerfpad*, Nivon, 2006, 1e druk.

17 J.M.M. van Rooij, H.L. Bodlaender en P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in A. Fiat en P. Sanders, eds., *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, Lecture Notes in Computer Science, Vol. 5757, Springer, 2009, pp. 566–577.