ALGORITMER
OG DERES BEGRÆNSNINGER

$$\mathbb{E}\left[e^{\alpha XY}\right] = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{\alpha xy} e^{\frac{-x^2}{2}} e^{\frac{-y^2}{2}} dydx = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{\frac{-x^2}{2}} e^{\frac{-y^2+2\alpha xy-\alpha^2x^2}{2}} e^{\frac{\alpha^2x^2}{2}} dydx$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{\frac{-x^2+\alpha^2x^2}{2}} \int_{-\infty}^{\infty} e^{\frac{-(y-\alpha x)^2}{2}} dydx = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{\frac{-x^2+\alpha^2x^2}{2}} dx$$

$$= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{\frac{-x^2(1-\alpha^2)}{2}} dx = \frac{1}{\sqrt{1-\alpha^2}}$$

KASPER GREEN LARSEN

LEKTOR VED DATALOGISK INSTITUT, AARHUS UNIVERSITET
OG (DELTIDS) PROFESSOR VED DATALOGISK INSTITUT, KØBENHAVNS UNIVERSITET.

Indenfor algoritmik forsøger vi at udvikle *effektive* metoder (algoritmer) til at løse beregningsproblemer. For at forstå om en algoritme faktisk er mere effektiv end en anden, analyserer vi, hvor meget arbejde de skal lave for at blive færdige. Ønsket er, at en algoritme laver så lidt arbejde som muligt, i forhold til hvor stort et input den får.

Hvad vil du helst udregne med papir og blyant, 84'103'541 + 32'493'092 eller 84'103'541 • 32'493'092? De fleste vil nok foretrække at udregne summen af de to tal fremfor produktet. Men hvorfor? Intuitivt virker det sværere at gange de to tal. Lidt mere formelt kan vi se på de to "folkeskolealgoritmer", som vi alle kender, til at løse de to regnestykker. Når vi skal summere de to tal, så starter vi ved de to bagerste cifre og lægger dem sammen. Vi skriver resultatet ned og husker eventuelt en mente. Vi kan så arbejde os igennem cifrene fra bagerste til forreste. På den måde kigger vi på hvert ciffer en enkelt gang. Hvis vi derimod vil gange de to tal sammen, så starter vi igen med mindste ciffer i det ene tal, eksempelvis **2** i 32'493'09**2**. Vi ganger nu 2 på cifrene i 84'103'541, et af gangen, startende bagfra og husker eventuelle menter. Når vi er færdige med 2 tager vi det næste ciffer, 9, sætter et 0 bagerst og ganger det på hvert af cifrene i 84'103'541, med menter og det hele. På den måde bevæger vi os langsomt igennem alle cifrene, hvor hvert eneste ciffer i 32'493'092 bliver ganget sammen med hvert eneste ciffer i 84'103'541. Til sidst summer vi det hele sammen. Umiddelbart skal vi altså lave meget mere arbejde for at gange de to tal.

Indenfor mit forskningsfelt, algoritmik, forsøger vi at udvikle *effektive* metoder (algoritmer) til at løse beregningsproblemer. I eksemplet ovenfor har vi to beregningsproblemer; at udregne summen af to tal og at udregne produktet af to tal. Vi har også to algoritmer, som vi kender fra folkeskolen, hvor det ser ud til, at den ene algoritme er mere effektiv end den anden. For at forstå om en algoritme faktisk er mere effektiv end en anden, så analyserer vi, hvor meget arbejde de skal lave for at blive færdige. Ønsket er, at en algoritme laver så lidt arbejde som muligt, i forhold til hvor stort et input den får. Hvis vi skal summere to tal, der hver har x cifre, så skal vi udføre ca. 2x instruktioner, da vi skal kigge på hvert ciffer en gang. Hvis vi derimod skal gange to tal med hver x cifre, så skal vi gange hvert ciffer i det ene tal sammen med hvert ciffer i det andet, altså ca. $x \bullet x = x^2$ arbejde. Men hvor stor forskel gør det, om en algoritme laver 2x instruktioner eller x^2 instruktioner? Når vi udvikler algoritmer, får vi oftest computere til at udføre dem, og de er lynende hurtige i forhold til mennesker. En moderne computer kan sagtens udføre omkring 1 milliard instruktioner på et sekund. Så er det ikke ligegyldigt om en algoritme laver 2x eller x^2 instruktioner? Desværre langt fra. Mange beregninger bliver i dag udført på enorme datasæt. Som eksempel kan nævnes at finde variationer imellem genomet for en kræfttumor og et "raskt" referencegenom. Det menneskelige genom består af x = ca. 3 milliarder basepar (A, C, G og T). En moderne computer vil med en algoritme der laver 2x instruktioner kunne finde variationerne på bare 6 sekunder, hvorimod en algoritme der laver x^2 instruktioner først vil blive færdig efter 285 år! Effektive algoritmer er derfor nøglen til overhovedet at løse dette beregningsproblem. Det samme gør sig gældende for et hav af andre beregningsproblemer, såsom når din GPS finder den korteste vej fra A til B, når Google finder hjemmesider, der matcher dine søgekriterier, og når firmaer som Amazon optimerer distribueringen af varer til kunder.

En moderne computer vil med en algoritme der laver 2x instruktioner kunne finde variationerne på bare 6 sekunder, hvorimod en algoritme der laver x^2 instruktioner først vil blive færdig efter 285 år!

Men er det så sværere at beregne produktet af to tal end at beregne deres sum? Eller er det bare en dårlig algoritme, vi lærer i folkeskolen? Det er trods alt den samme algoritme, der har været brugt siden oldtidens Babylon. Tilbage i 1960 stillede den russiske matematiker Andrey Kolmogorov præcist dette spørgsmål. Mere præcist formodede han, at der ikke kan laves en algoritme, der beregner produktet af to tal med mindre end x^2 instruktioner. Kolmogorov arrangerede et seminar ved Moscow State University med

FOTO: PETER GARGILO / UNSPLASH

det formål at bevise formodningen. Men allerede en uge inde i seminaret fandt den studerende, Karatsuba, på en ny algoritme, som kan beregne produktet af to tal med $x^{1,585}$ instruktioner. Kolmogorov præsenterede algoritmen ved næste seminarmøde og afsluttede seminaret. Siden da er der blevet udviklet en lang række af hurtigere og hurtigere algoritmer til at gange tal sammen, kulminerende med Harvey og van der Hoevens algoritm i 2019, som kan beregne produktet i kun x log x instruktioner. Her er log x den meget langsomt voksende funktion logaritmen til x. Denne algoritme er så hurtig, at en moderne computer kan gange tal sammen med millioner, og endda op til milliarder af cifre, på meget kort tid. Men det er stadigt langsommere end at summere tallene. Kan vi finde på endnu hurtigere algoritmer?

I et resultat fra 2019 har vi i min forskningsgruppe vist, at der ikke kan eksistere en algoritme, der beregner produktet af to tal hurtigere end i x log x instruktioner! Dvs. at den nuværende algoritme af Harvey og van der Hoeven er optimal. Efter tusinder af år med at gange tal sammen kan vi altså stoppe jagten på hurtigere algoritmer – de findes ikke. Vores resultat synes også interessant fra et filosofisk synspunkt – gange *er* sværere end plus.

Ud over spørgsmålet om hvor svært det er at gange tal sammen, arbejder vi også på at forstå, hvor effektivt en lang række andre algoritmiske problemstillinger kan løses. Dette har resulteret både i hurtigere algoritmer, men også i mange resultater, som eksemplet ovenfor, hvor vi viser, at de nuværende metoder umuligt kan forbedres. Vi har desuden brugt disse teknikker i andre felter end klassisk algoritmik, som eksempelvis indenfor kryptering og machine learning. Vores resultater indenfor disse områder er igen med til at belyse, hvor mange ressourcer (mængden af data eller regnekraft etc.) man skal bruge for at løse en konkret problemstilling. Mange af teknikkerne kan derfor anvendes på tværs af forskningsområder, og arbejdet på tværs giver ofte overraskende nye indsigter.

$$\Pr\left[\|w\|_2 \cdot \sum_{i\in[k]} X_i Y_i \geq (1-\alpha)t\right] \leq 2e^{\frac{-0.48k(1-\alpha)^2t^2}{\|w\|_2^2}} = 2e^{\frac{-0.48k(1-\alpha)^2t^2}{1-\langle u,v\rangle^2}}$$

$$\alpha = \frac{\sqrt{0.48}\langle u,v\rangle}{\sqrt{0.48}\langle u,v\rangle + \sqrt{0.21(1-\langle u,v\rangle^2)}}$$