

Generelle Optimeringsheuristikker

- en introduktion

af Per S. Laursen

Datalogisk Institut (DIKU)
Københavns Universitet
Universitetsparken 1
2100 København Ø

Marts 1994

Indholdsfortegnelse

1 Indledning	1
1.1 Kombinatorisk Optimering	1
1.2 Et eksempel: Grafdeling	2
1.3 En eksakt optimeringsmetode	2
1.4 Approksimative metoder	5
2 Simuleret Udgødning	7
2.1 Den grundlæggende algoritme og fysiske analogier	7
2.2 Kølingsrater	9
2.2.1 Starttemperatur	9
2.2.2 Sænkning af temperaturen - kølingsraten	10
2.2.3 Stopkriterium	10
2.3 Simuleret Udgødning for Grafdelingsproblemet	11
2.4 Tips og tricks	12
3 Tabu Søgning	15
3.1 Den grundlæggende algoritme	15
3.2 Centrale elementer i Tabu Søgning	17
3.2.1 Tabulisten	17
3.2.2 Aspirationskriterier	18
3.2.3 Langtidshukommelse	19
3.3 Tabu Søgning anvendt på Vægtet Knudeoverdækning	19
3.4 Tips og tricks	21
4 Genetiske Algoritmer	24
4.1 De grundlæggende principper	24
4.2 Mere om udvælgelse, krydsning og mutation	25
4.2.1 Udvalgelse	25
4.2.2 Krydsning	26
4.2.3 Mutation	27
4.3 En Genetisk Algoritme for TSP	28
4.4 Tips og tricks	29

5 Neurale Netværk	31
5.1 Boltzmann maskiner	31
5.1.1 Basale elementer i Boltzmann maskiner	31
5.1.2 Overgang til nye tilstande	32
5.1.3 Kombinatorisk Optimering med Boltzmann maskiner	33
5.2 En Boltzmann maskine for Grafdelingsproblemet	34
5.3 Tips og tricks	36
6 Parallelisering af optimeringsheuristikker	38
6.1 Paralleldatamater	38
6.2 Generelle observationer og en simpel parallelisering	39
6.3 Parallel Simuleret Udglødning	40
6.4 Parallel Tabu Søgning	41
6.5 Parallele Genetiske Algoritmer	42
6.6 Parallele Boltzmann maskiner	43
Referencer	44
Projektkatalog	47

Forord

Disse noter er udarbejdet i forbindelse med kurset "Generelle Optimeringsheuristikker", som blev afholdt på Datalogisk Institut (DIKU) i efteråret 1993. Noterne er herefter - bl.a. på basis af kommentarer fra de studerende på kurset - blevet let reviderede, før de har opnået denne endelige form. Meningen med noterne er at give en samlet, relativt kortfattet, præsentation af fire optimeringsheuristikker: Simuleret Udglødning, Tabu Søgning, Genetiske Algoritmer samt Neurale Netværk. Disse heuristikker er naturligvis blevet behandlet en lang række andre steder før dette (se blot referencerne); mange af disse fremstillinger er dog fokuseret på algoritmernes teoretiske egenskaber. Jeg mener derfor der er behov for en samlet, dansk fremstilling, som mere fokuserer på de egentlige algoritmiske principper, frem for mere teoretiske egenskaber ved heuristikkerne. Det er mit ønske at opfylde dette mål med disse noter.

Noterne falder i seks hovedafsnit. Kapitel 1 giver en kort introduktion til optimering (specielt kombinatorisk optimering), og diskuterer generelle principper for hhv. eksakte og approksimative søgemetoder. Kapitel 2 til 5 præsenterer hhv. Simuleret Udglødning, Tabu Søgning, Genetiske Algoritmer samt Neurale Netværk. Det beskrives, hvorledes heuristikkerne kan benyttes til kombinatorisk optimering, hvilket i alle fire tilfælde eksemplificeres ved at skitsere, hvorledes algoritmen kan anvendes til at løse et specifikt optimeringsproblem. Desuden udstikkes der nogle generelle retningslinier for brug af algoritmerne. I kapitel 6 rider forfatteren sin egen kæphest, idet det diskuteres hvorledes man kan parallelisere disse heuristikker, med henblik på implementation på paralleldatamater. Efter de egentlige kapitler følger et lille "katalog" over mulige projekter indenfor emneområdet Generelle Optimeringsheuristikker. Projekterne er ikke grydeklare, men kan forhåbentligt give inspiration til mere specifikke projekter.

Det understreges, at noterne ikke søger at give nogen teoretisk motivation for anvendeligheden af disse heuristikker. Andre har gjort dette før (i de tilfælde, hvor det overhovedet er muligt), og den nysgerrige læser henvises derfor til den supplerende litteratur samt referencerne.

1 Indledning

Dette kapitel giver en kort introduktion/opfriskning af emnet Optimering som sådan, og præsenterer de generelle principper for hhv. eksakte og approksimative søgemetoder. Som et første eksempel på et kombinatorisk optimeringsproblem præsenteres Grafdelingsproblemet. Sluttelig gives referencer til mere udtømmende fremstillinger.

1.1 Kombinatorisk Optimering

Optimering kan lidt uformelt beskrives som *en proces, hvorved vi søger at opnå mest muligt ved hjælp af en begrænset mængde resourcer*. Mere kvantitativt kan optimering beskrives som maksimering eller minimering af en funktion af et antal variable, hvor værdien af disse variable er underkastet givne begrænsninger. Vi holder os til minimering i det følgende. Minimering af en funktion af kontinuerte variable er i princippet let; det er blot et spørgsmål om at undersøge tilpas mange differentialkvotienter. Helt så let er det nu ikke i praksis, men gode numeriske metoder findes til at løse sådanne problemer. Værre bliver det, hvis der er tale om *diskrete* variable, f.eks. variable, som kun må antage heltallige værdier. Selv om løsningsrummet (mængden af mulige løsninger) nu bliver endeligt (egentlig kun tælleligt, men vi koncentrerer os her om endelige løsningsrum), kan en sådan begrænsning af de variables værdier gøre problemet adskilligt sværere. Et optimeringsproblem, hvor de variable kun kan antage et endeligt antal værdier, betegnes et *kombinatorisk optimeringsproblem*. Det er problemer af denne type, vi vil koncentrere os om her.

Et kombinatorisk optimeringsproblem er således defineret ved et (endeligt) løsningsrum S , samt en objektfunktion $f: S \rightarrow \mathbf{R}$. Målet med at løse dette problem bliver således at finde en *optimal* løsning $s_{opt} \in S$, hvor optimal defineres som:

$$\forall s \in S : f(s_{opt}) \leq f(s) \quad (1.1)$$

Der er efterhånden blevet defineret en enorm mængde kombinatoriske optimeringsproblemer. En del af dem er ret lette af løse. Tag for eksempel *korteste vej - problemet*. Givet en orienteret graf $G(V,E)$, med vægte W på kanterne, find da den korteste vej fra en given knude v_0 til alle de øvrige knuder i grafen, hvor længden af en vej fra v_0 til en knude v er defineret som summen af vægtene på de kanter, som udgør vejen fra v_0 til v . Antallet af mulige løsninger til dette problem er astronomisk; vi skal finde $|V|-1$ forskellige veje, som hver især kan bestå af op til $|V|-1$ forskellige kanter. Tydeligvis vokser antallet af mulige løsninger eksponentielt i grafens størrelse. Alligevel kan vi finde en optimal løsning i polynomiel tid i grafens størrelse, $O(|V|^2)$. Dette skyldes, at dette problem kan løses ved at konstruere sig frem til en løsning, v.h.a. en algoritme som beviseligt fører til en optimal løsning i løbet af $O(|V|^2)$ (se f.eks. [Hor87]). Altså behøver vi ikke undersøge hver enkelt mulig løsning.

Optimeringsproblemer, som kan løses i polynomiel tid i problemets størrelse, kaldes for *P-problemer*. Disse problemer løses næsten altid med en form for konstruktions-teknik. Andre *P-problemer* er udspændende træ, lineær tildeling, parring i grafer, maksimalstrømning, o.m.a..

Desværre findes der også en stor mængde problemer, som så vidt vides ikke kan løses v.h.a. konstruktionsteknikker. I stedet må man - såfremt man ønsker at finde en optimal løsning - mere eller mindre eksplicit undersøge alle de mulige løsninger. Disse problemer kaldes *NP-hårde*. Klasserne *P* og *NP*, samt kompleksitetsteori i det hele taget, er et stort emne i sig selv, og er behandlet i langt større detalje i f.eks. den klassiske bog *Computers and Intractability* af Garey & Johnson [Gar79], samt af Wilf i en lidt mere letlæst form [Wil86]. Vi nøjes blot med at konstatere, at vi for mange problemers vedkommende ikke har noget bedre alternativ end at søge gennem løsningsrummet, i håbet om at finde en god - måske optimal - løsning. Et eksempel på et sådant problem er Grafdelingsproblemet.

1.2 Et eksempel: Grafdeling

Grafdelingsproblemet lyder besnærende enkelt: Givet en graf $G(V,E)$, opdel da knuderne i to lige store grupper, således at der går så få kanter som muligt mellem de to grupper af knuder. Alligevel er problemet *NP-hårdt*. Lidt mere formelt får problemet følgende formulering: Givet en graf $G(V,E)$, samt en matrix C , hvor $c_{ij}=1$ hvis kanten $e_{ij} \in E$, og $c_{ij}=0$ hvis dette ikke er tilfældet. En mulig løsning s til problemet er da en binær vektor $B = \{b_1, \dots, b_{|V|}\}$, hvor $b_i=0$ hvis knuden i befinder sig i den første gruppe, og $b_i=1$ hvis knuden i befinder sig i den anden gruppe. Vi ønsker da at minimere $f(s)$, hvor

$$f(s) = \frac{1}{2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} c_{ij} |b_i - b_j| \quad , \quad s \in B \quad (1.2)$$

over alle mulige binære vektorer $\{0,1\}^{|V|}$, pålagt den betingelse, at summen af elementerne i vektoren skal være $|V|/2$, så vi får lige mange knuder i hver gruppe.

Grafdelingsproblemet har flere praktiske anvendelser. Det klassiske eksempel er fra design af elektroniske systemer: Antag, at vi ønsker at placere et antal elektroniske moduler (f.eks. chips) på to "kort". De kan ikke alle sidde på det samme kort, så vi må fordele dem på to. Hvis to moduler på forskellige kort skal forbindes, må vi benytte et antal ledninger. Ønsker vi at bruge færrest mulig ledninger, har vi netop et Grafdelingsproblem, hvor modulerne er knuder, og de nødvendige forbindelser er kanter. Vi skal se eksempler på andre *NP-hårde* kombinatoriske optimeringsproblemer i de følgende kapitler.

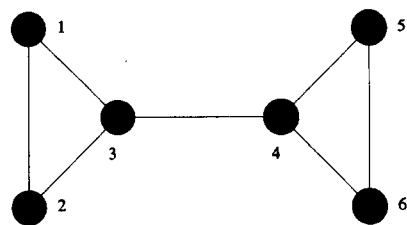
1.3 En eksakt optimeringsmetode

Ønsker vi at finde en optimal løsning til et *NP-hårdt* optimeringsproblem, må vi søge gennem hele løsningsrummet S , og mere eller mindre eksplicit teste hver eneste løsning. Heldigvis er

der udviklet algoritmer, som tillader os at smide store dele af løsningsrummet væk undervejs i denne søgning, således at vi kun behøver at undersøge en forholdsvis lille mængde løsninger eksplicit, men stadig bevarer garantien for at finde en optimal løsning. Metoder, hvorved vi er sikre på at finde en optimal løsning, hører under kategorien *eksakte optimeringsmetoder*, d.v.s. den fundne løsning er med garanti optimal m.h.t. hele løsningsrummet. Den altdominerende metode til eksakt løsning af kombinatoriske optimeringsproblemer kaldes for *Branch and Bound*¹.

Hovedidéen i Branch and Bound algoritmer er at foretage en trinvis opdeling af løsningsrummet i mindre bidder (underrum), ved at foretage et antal forgreninger (eng.: *branching steps*), ved successivt at fastlægge værdien af et antal variable. Ved hjælp af en grænsefunktion (eng.: *bound function*) kan der findes en nedre grænse for værdien af objektfunktion for alle løsninger i det givne underrum. Denne nedre grænse kan så sammenlignes med objektfunktionsværdien for en kendt løsning, f.eks. fundet på et tidligere tidspunkt i søgningen. Hvis den nedre grænse er lavere end værdien af den kendte løsning, må vi opdele det givne underrum yderligere, idet det muligvis kan rumme en bedre løsning end den kendte. Er den nedre grænse derimod lig med eller højere end værdien af den kendte løsning, kan vi smide hele underrummet væk, idet det ikke kan rumme en bedre løsning end den, vi kender i forvejen. Grænsefunktionen er oftest den helt afgørende faktor i en Branch and Bound algoritme. Først og fremmest skal den naturligvis være "korrekt", d.v.s. vitterligt give en nedre grænse, så vi ikke kan risikere fejlagtigt at smide et underrum væk, og samtidigt skal den helst være så "skarp" som muligt, så vi ikke skal undersøge flere underrum end strengt nødvendigt. Udvikling af gode grænsefunktioner er et emne for sig...

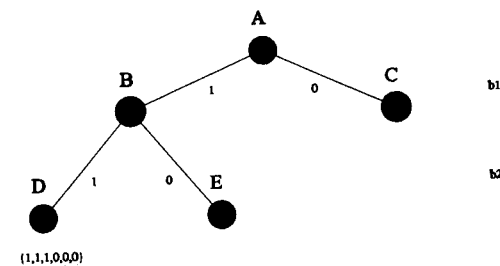
Lad os se et eksempel på en Branch and Bound algoritme for Grafdelingsproblemet. Vi ønsker at finde den optimale opdeling af grafen på Figur 1 herunder.



Figur 1: En (meget) lille graf

¹ Jeg har valgt ikke at oversætte denne betegnelse til dansk, i mangel af en god oversættelse...

Det kræver ikke det store falkeblæk at se, at en optimal løsning til dette problem har værdien 1, og at $s = \{1,1,1,0,0,0\}$ er en optimal løsning. På den anden side giver eksemplets størrelse os mulighed for at "håndkøre" algoritmen uden at overanstrenge os. For at forgrene fra et givet underrum fastlægger vi værdien af netop én variabel, d.v.s. vi lægger en enkelt af knuderne i enten den ene eller anden gruppe. Som grænsefunktion bruger vi ganske enkelt antallet af kanter mellem de knuder, som allerede er placeret i en gruppe. Slagets gang er illustreret på Figur 2, som viser det såkaldte *søgetræ*, som søgningen giver anledning til. Vores udgangspunkt er den "tomme" vektor $v = \{_, _, _, _, _, _ \}$ (_ betyder undefineret), som således repræsenterer hele løsningsrummet S . For at undgå symmetriske løsninger kan vi dog tillade os at placere en enkelt knude fast i en gruppe, så vi vælger i stedet $v = \{1, _, _, _, _, _ \}$ som udgangspunkt (knude A i søgetræet på Figur 2). Idet vi ikke har nogen kendt løsning at sammenligne med endnu, sætter vi værdien af den kendte løsning til 7 (det totale antal kanter). Grænsefunktionsværdien for A er 0, så vi må opdele A. Vi vælger nu at fastlægge værdien af b_2 . Da b_2 kan være enten 1 eller 0, skaber vi herved to nye (disjunkte) underrum, knude B og C og søgetræet. Vi lægger foreløbig C på lager, og går videre med B. I dette underrum ligger knude 1 og 2 i samme gruppe, så grænsefunktionsværdien bliver 0. B må da opdeles, hvilket gøres ved at fastlægge b_3 , hvilket giver os knude D og E i søgetræet. E lægges på lager, og vi undersøger først D. I dette underrum er $v = \{1,1,1, _, _, _ \}$, men da der skal være lige mange knuder i hver løsning, kan vi uden videre "forlænge" v til $\{1,1,1,0,0,0\}$. Altså har vi fundet en løsning, med objektfunktionsværdi 1. D skal således ikke deles yderligere op, så vi henter knude E ind fra lageret. Her er $v = \{1,1,0, _, _, _ \}$, hvilket giver en grænsefunktionsværdi på 2. E kan således smides væk, og vi henter C ind. Her er $v = \{1,0, _, _, _, _ \}$, og grænsefunktionen dermed 1. C kan derfor også smides væk. Hermed er problemet løst, idet der ikke er flere delproblemer tilbage.



Figur 2: Branch and Bound søgetræ

Med et eksempel som dette ser Branch and Bound algoritmen meget tilforladelig ud. Imidlertid skal man som regel ikke særligt højt op i problemstørrelse, før det tager *meget* lang tid at færdiggøre søgningen. Branch and Bound fjerner med andre ord ikke den eksponentielle natur fra disse problemer. Det har den kedelige konsekvens, at man for en lang række problemer ikke kan finde en (garanteret) optimal løsning indenfor rimelige tidsrammer, men må stille sig tilfreds med en suboptimal løsning. Det er her, at de approksimative metoder kommer ind i billedet.

Branch and Bound er et stort emne i sig selv, og rummer væsentligt flere detaljer end skitseret her. Flere "klassiske" bøger om optimering beskriver Branch and Bound, f.eks. Papadimitriou & Steiglitz [Pap82] samt Garfinkel & Nemhauser [Gar72].

1.4 Approksimative metoder

De approksimative metoder kan groft taget inddrages i to grupper; *deciderede approksimationsalgoritmer*, der finder løsninger, som med garanti ikke er mere end f.eks. en konstant faktor dårligere end en optimal løsning, samt *heuristikker*, som på mere eller mindre velbegrunder vis er i stand til at finde "gode" løsninger i den forstand, at man mener at have empirisk dokumentation for løsningernes kvalitet. Approksimationsalgoritmerne er næsten udelukkende problemspecifikke, og principperne for en algoritme kan sjældent overføres til et andet problem. Der findes også en hel del problemspecifikke heuristikker, f.eks. også en for Grafdelingsproblemet, udviklet af Lin & Kernighan [Lin70]. Indenfor det seneste årti har der dog samlet sig stor interesse omkring de såkaldte *generelle optimeringsheuristikker*, der som navnet antyder ikke binder sig til en speciel type problemer, men tværtimod blot udstikker en algoritmisk ramme, som så kan fyldes ud, alt efter hvilket problem man ønsker at anvende heuristikken på. Interessen har næsten udelukkende samlet sig om fire heuristikker; Simuleret Udglødning, Tabu Søgning, Genetiske Algoritmer og Neurale Net. Det er således disse fire heuristikker, som er hovedemnet for disse noter. Interessen er (naturligvis) primært forårsaget af det faktum, at en eller flere af disse heuristikker har været i stand til at udkonkurrere eksisterende metoder for en lang række problemer, og for en del andre problemer har de været det første fornuftige bud overhovedet på en god løsningsstrategi. Heuristikkerne beskrives i detaljer senere; her vil vi blot beskrive et princip, som til en vis grad deles af flere af heuristikkerne: princippet om *lokal søgning*.

I Branch and Bound prøver man så at sige at holde alle bolde i luften på én gang; under hele søgningen har man alle dele af det udforskede løsningsrum repræsenteret ved de lagrede, ubehandlede knuder i søgetræet. Søgningen foregår på et så "makroskopisk" niveau som muligt. Lokal søgning er stik modsat; her har man til enhver tid kun en enkelt løsning til problemet repræsenteret, og man søger så - v.h.a. et sæt prædefinerede "trafikregler" - at bevæge sig rundt i løsningsrummet på en sådan måde, at man forhåbentligt møder en eller flere "gode" løsninger undervejs. Den vigtigste del af disse "trafikregler" er definitionen af

en *omegn* (eng.: *neighbourhood*) til en løsning. Omegnen $N(s)$ til en løsning $s \in S$ er netop de løsninger, som vi kan "gå til" fra s . Oftest defineres omegnen til en løsning således, at det er enkelt at gå fra løsning til løsning. For Grafdelingsproblemet kunne en oplagt definition på en omegn være at udtage en enkelt knude fra hver gruppe, og bytte deres placering om (d.v.s. lægge knuden fra gruppe 1 i gruppe 0, og vice versa). Lokal søgning kan således beskrives ved nedenstående programskitse:

```

procedure lokal_søgning;
begin
  find_en_initiel_løsning(s);
  repeat
    generér_løsning_fra_omegn(s,s');
    if acceptér_ny_løsning(s,s') then s:=s';
  until stop_kriterium;
end;

```

Algoritme 1: Lokal søgning

Det helt afgørende trin i algoritmen er accepten eller forkastelsen af den foreslåede løsning fra omegnen. Ét er at få tilbudt en ny løsning fra omegnen, et andet er, om vi vil benytte denne løsning som vores nye udgangspunkt. Vi må således definere et *acceptkriterium*. Idet vi jo ønsker at finde den bedst mulige løsning m.h.t. objektfunktionsværdi, vil acceptkriteriet typisk være baseret på denne funktion. Én mulighed er kun at acceptere en løsning, hvis den har lavere objektfunktionsværdi end den nuværende løsning. Denne strategi kaldes for *trinvis forbedring* (eng.: *iterative improvement*), og har længe nydt en vis popularitet. Den har imidlertid den oplagte ulempe, at vi kun kan nå et lokalt optimum, d.v.s. en løsning s , hvorom det gælder, at

$$\forall s' \in N(s) : f(s') \geq f(s) \quad (1.3)$$

Trinvis forbedring tillader os med andre ord ikke at gå "op ad bakke". Man benytter derfor ofte trinvis forbedring som en "genstarts"-algoritme, d.v.s. man finder en tilfældig startløsning, lader algoritmen finde ned til det lokale optimum, genstarter med en ny tilfældig løsning, o.s.v.. På denne måde kan man nå et stort antal lokale optima, og følgelig have begrundet håb om at finde bedre og bedre løsninger. Alternativt kan man give algoritmen lov til at gå op ad bakke på egen hånd. Man accepterer således fra tid til anden løsninger med dårligere (højere) objektfunktionsværdi end den øjeblikkelige løsning. I den ekstreme situation kunne man vælge at acceptere *alle* foreslåede løsninger, uanset deres kvalitet, men på denne måde reducerer søgningen blot til tilfældig søgning, hvor man "blindt" traver gennem løsningsrummet, og håber på at støde ind i noget godt undervejs... Mere sofistikeret ville det være kun at gå op ad bakke engang imellem, d.v.s. med en vis sandsynlighed. De fire heuristikker beskrevet her er i princippet alle variationer over dette tema. Det ses nok mest tydeligt for heuristikken Simuleret Udglødning.

2 Simuleret Udguldning

Simuleret Udguldning er nok den af de fire heuristikker, som har påkaldt sig mest opmærksomhed i de sidste 5-10 år. Heuristikken stammer i princippet helt tilbage fra 1953, hvor den blev anvendt til simulering af fysiske systemer [Met53]. Første gang den blev anvendt på et kombinatorisk optimeringsproblem var imidlertid i 1983, af Kirkpatrick et al. [Kir83], og metoden har siden nydt stor popularitet. Specielt fysikerkredse har taget metoden til sig, idet det er muligt at drage endog meget vidtgående analogier mellem metoden, og den måde visse fysiske systemer opfører sig på. Dette er dog mest et fortolkningsspørgsmål, samt et spørgsmål om hvilke etiketter man hæfter på de forskellige ingredienser af algoritmen. I hvert fald kan man sagtens forstå og benytte den basale algoritme uden kendskab til fysik.

2.1 Den grundlæggende algoritme og fysiske analogier

Simuleret Udguldning er en lokal søgemetode, af typen beskrevet før. Altså skal vi definere et løsningsrum S , en omegn $N(s)$ til en løsning s , samt en objektfunktion $f(s)$, før vi kan gå i gang. Algoritmen fungerer i princippet helt som den lokale søgemetode beskrevet før, d.v.s. vi accepterer altid en løsning, hvis den har lavere objektionsværdi² end den aktuelle løsning, og accepterer kun dårligere løsninger med en vis sandsynlighed strengt mindre end 1. Det, som adskiller Simuleret Udguldning fra andre varianter af denne strategi, er definitionen af denne *overgangssandsynlighed*. Lad s være den øjeblikkelige løsning. Sandsynligheden for at acceptere en dårligere løsning s' er da defineret som

$$P_{acc}(s'/s) = e^{-\frac{f(s')-f(s)}{T}} \quad (2.1)$$

hvor T er en kontrolparameter (ofte betegnet *temperaturen*). T er ikke konstant, men sænkes løbende under søgningen. Hvordan dette gøres, er et af de store emner i Simuleret Udguldning, og vi vender tilbage til det lidt senere.

De fleste vil nok spørge, om der er særligt gode argumenter for at anvende denne definition på overgangssandsynlighed frem for andre definitioner. Ét argument er, at det er denne sandsynlighed, naturen selv anvender i visse "optimeringsprocesser". Som navnet antyder, kan Simuleret Udguldning betragtes som en simulering af en fysisk proces kaldet udguldning. I denne proces tager man et fast stof - typisk et metal - og varmer det kraftigt op. Stoffets atomer får derved stor energi, og kan relativt frit bevæge sig rundt mellem hinanden (stoffet bliver flydende). Atomerne i et fast stof vil helst sætte sig på en måde, som gør den totale sum af deres energi mindst mulig. For f.eks. et metals vedkommende opnås

² En populær betegnelse - især blandt fysikere - for objektionsværdien er *energien* af en løsning.

dette, hvis alle atomerne sætter sig i hjørnepunkterne i et tre-dimensionalt gitterværk. Sidder atomerne sådan, vil metallet typisk få en del "gode egenskaber" (god strøm- og varmeleder, etc.). Det er denne struktur, man prøver at opnå ved udguldning. Sænker man metallens temperatur ganske langsomt, får de fleste af atomerne tid til at finde på plads, og vi kan få et næsten perfekt gitterværk, med meget lav total energi. Køler vi systemet meget brat og hurtigt - som en hestesko i en spand vand - fryses atomerne fast i en uregelmæssig struktur med høj energi. Lidt mere formelt kan vi betragte enhver placering af atomerne i metallet som en løsning til et optimeringsproblem. Rykkes et enkelt atom en smule, har vi en naboløsning. Pointen er nu, at (2.1) præcis er den sandsynlighed, naturen giver systemet for at flytte sig til en given nabotilstand, d.v.s. for at atomet kan flytte sig til den nye position. Da atomer jo er "dumme", er det en meget langsommelig proces at få dem alle hen på det rette sted, og systemet må følgelig afkøles meget forsigtigt, specielt omkring "faseovergange", d.v.s. hvor stoffet går fra f.eks. flydende til fast form. På den anden side kan man også opnå gode resultater, hvis man udviser tilstrækkelig omhyggelighed, hvilket sikkert har inspireret ideen om at overføre disse principper til rent abstrakte systemer. I en vis forstand er det således troen på at Moder Natur gør det rigtige, som er det bedste argument for at anvende (2.1) som overgangssandsynlighed. Dog kan man også vise, at hvis man anvender (2.1), samt sænker temperaturen på en nærmere foreskrevet måde, finder man et globalt optimum med sandsynlighed 1. Dog kræver dette, at temperaturen sænkes uendeligt (i matematisk forstand) langsomt, så dette resultat er måske mere af akademisk interesse. I hvert fald går denne konvergens egenskab fløjten, hvis vi "kun" bruger en endelig mængde tid.

For at rekapitulere, så er Simuleret Udguldning blot en lidt sofistikeret version af algoritmen for lokal søgning fra afsnit 1.4. En algoritmeskitse er givet herunder:

```
procedure simuleret_udguldning;
begin
  find_en_initiel_loesning(s);
  find_initiel_temperatur(T);
  repeat
    generer_loesning_fra_omegn(s,s');
    Δf:=f(s')-f(s);
    if Δf≤0 then s:=s';
    if Δf>0 then
      begin
        p:=random[0;1];
        if p<exp(-Δf/T) then s:=s';
      end;
    beregn_ny_temperatur(T);
  until stop_kriterium;
end;
```

Algoritme 2: Simuleret Udguldning

