Handelsresandeproblemet och dess approximation med simulerade myrkolonier

Anders Ahlström

Utkast – 2.4.17

Sammanfattning

Handelsresandeproblemet, ett klassiskt problem inom datavetenskap, går ut på att en handelsman vill sälja sina varor i en given mängd städer, och denne söker nu den kortaste vägen som besöker alla dessa och låter handelsresenären återvända till sin hemstad. Vi ser på dess komplexitet och studerar algoritmer som ger en exakt lösning samt sådana som kan användas för att få en ungefärlig lösning. Några av dess tillämpningar och flera relaterade problem tas upp. Vi betraktar myrkolonioptimering, en form av metaheuristik inspirerad av den svärmintelligens myrstackar uppvisar. Myrkolonioptimering har tillämpats på bland annat handelsresandeproblemet.

Sökord: *handelsresandeproblemet, kombinatorisk optimering, heuristik, myrkolonioptimering, approximering*

Innehåll

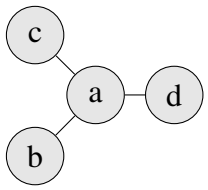
1 Inledning

Handelsresandeproblemet (stycke 2) är ett klassiskt problem inom datavetenskap. Problemet går ut på att en handelsman vill sälja sina varor i en given mängd städer, och denne söker nu den kortaste vägen som besöker alla dessa och låter handelsresenären återvända till sin hemstad. Problemet har stor betydelse för datavetenskap både teoretiskt (se särskilt stycke 1.2) och för tillämpningar (stycke 2.4). Handelsresandeproblemet är också av intresse inom tillämpad matematik, särskilt kombinatorisk optimering.

Eftersom handelsresandeproblemet är svårt att lösa exakt har olika former av heuristik tillämpats för att hitta ungefärliga lösningar. Naturliga fenomen, särskilt olika biologiska processer, har varit en stor inspirationskälla för nya heuristiska metoder. Genetiska algoritmer (stycke 2.3.2) imiterar naturlig selektion. Partikelsvärmalgoritmer [22] och myrkolonioptimering (stycke 3) är inspirerade av svärmintelligens som förekommer i naturen. Simulerad glödgning (stycke 2.3.3) är inspirerad av fysikaliska fenomen. Förutom handelsresandeproblemet har dessa tillämpats på många andra problem.

Människor [11], chimpanser och bin har uppvisat god förmåga att uppskatta lösningar på problemet.

1.1 Grafteori



Figur 1. Grafen $K_{1,3}$

En *graf* [1] är ett ordnat par $G = (V, E)$, där V är en mängd *noder* och E är en mängd *kanter*, med en associerad funktion $\psi : E \to \{\{v, w\} : v, w \in V\}$. Ibland betecknar vi kant- och nodmängderna för en graf $H = (V_H, E_H)$ för att skilja mellan olika grafers kant- och nodmängder. För att förenkla notation betecknar vi en kant $\psi(e) = \{v, w\}$ som e_{vw}. Vi definierar dessutom en *viktfunktion* $w : E \to [0,\infty[$. Exemplet i figur 1 är $K_{1,3} = (\{a, b, c, d\}, \{e_{ab}, e_{ac}, e_{ad}\})$.

(a) K_5 **(b)** C_5

Figur 2. Cykliska och kompletta grafer med 5 noder

En *rutt* $r \subseteq E$ från en nod v till w är en ordnad multimängd så att $r = \{e_{vn_1}, e_{n_1n_2}, ..., e_{n_kw}\} = vn_1n_2...n_kw$. En rutt r besöker en nod v om och endast om $\exists_{e \in r} v \in \psi(e)$. En *cykel* är en sådan rutt där $v = w$. En *Hamiltonstig* är en rutt som besöker varje nod i en graf exakt en gång. En *Hamiltoncykel* är en cykel som besöker varje nod exakt en gång, förutom startnoden som besöks två gånger. Låt *distansen* för en rutt (eller kantmängd) vara $d(r) = \sum_{e \in r} w(e)$.

En *komplett graf* K_n är en graf med n noder så att $\forall_{v,w \in V, w \neq v} e_{vw} \in E$ och en *cyklisk graf* C_n är en graf med n noder så att om $V = \{v_1, v_2, ..., v_n\}$ är $E = \{e_{v_1v_2}, e_{v_2v_3}, ..., e_{v_nv_1}\}$. En cyklisk graf har alltid en trivial Hamiltoncykel $r_{C_n} = v_1v_2...v_nv_1$. Eftersom $V_{C_n} = V_{K_n}$ samtidigt som $E_{C_n} \subseteq E_{K_n}$ har också alla kompletta grafer Hamiltoncykeln r_{C_n}. Alternativt gäller för en godtycklig permutation $P = \{p_1, p_2, ..., p_n\}$ av V_{K_n} att $r = p_1p_2...p_np_1$ är en Hamiltoncykel på K_n. Figur 2 visar exempel på cykliska och kompletta grafer. Båda dessa har som förklarats Hamiltoncyklar r (och Hamiltonstigar som fås genom att lämna bort sista kanten i r), men $K_{1,3}$ i figur 1 har varken Hamiltoncykel eller Hamiltonstig.

En *sammanhängande graf* är en graf så att för alla $v, w \in V$ finns en rutt r från v till w. Ett *träd* är en sammanhängande graf så att r är entydig (bortsett från upprepade kanter eller kanter från en nod till sig själv). Den sammanhängande grafen G har ett *spännträd* $T = (V_G, E_T \subseteq E_G)$ så att T utgör ett träd. Det *minsta spännträdet* för G är ett spännträd T så att $d(E_T)$ minimeras (detta illustreras i figur 8 på sida 13). En nod v har *gradtalet* $\deg(v) = |\{e : e \in E, v \in \psi(e)\}|$. Om $\forall_{v \in V_G} \deg(v) = k$ kallas G *k-reguljär*. En *jämn graf* är en k-reguljär graf där k är ett jämnt tal. En 3-reguljär graf kallas för en *kubisk graf*.

1.2 Algoritmers komplexitet

Om $T(n)$ uttrycker tiden en algoritm tar för att lösa ett problem (t.ex. mängden instruktioner som måste köras) bestående av n element och algoritmen har den *asymptotiska komplexiteten* $\mathcal{O}(f(n))$ gäller att $\exists_{x_0 \in \mathbb{N}, k \in \mathbb{R}_+} \forall_{\mathbb{N} \ni n > x_0} T(n) \leq k f(n)$ [5]. Ekvivalent kan detta definieras:

$$\lim_{n \to \infty} \frac{T(n)}{f(n)} < \infty \iff T(n) \in \mathcal{O}(f(n)).$$

Ofta kommer $T(n)$ förutom mängden element n bero på egenskaper hos själva elementen. Exempelvis kör många sorteringsalgoritmer inom linjär tid ($\mathcal{O}(n)$) på redan sorterad data. Vanligtvis är vi intresserade av vad $T(n)$ är i det värsta fallet [5] d.v.s. vi vill uttrycka en övre gräns som gäller oberoende egenskaper hos elementen. Förutom körtid används notationen också ibland för att beteckna algoritmers minnesanvändning.

Ett problem inom komplexitetsklassen $\mathcal{P}$ (polynomiell) är ett problem för vilket finns en algoritm som kan köras inom asymptotiskt polynomiell tid ($\mathcal{O}(n^p), p \in \mathbb{N}$). En sådan algoritm kallas effektiv. Ett problem för vilket en lösning kan verifieras inom polynomiell tid tillhör klassen $\mathcal{NP}$ (från engelskans "non-deterministic polynomial", icke-deterministiskt polynomiell, i och med att ett sådant problem kan lösas inom polynomiell tid på en icke-deterministisk Turingmaskin). [2, kap. 15, s. 377-412] Frågan om $\mathcal{P} = \mathcal{NP}$ är kanske det största öppna problemet inom datavetenskap [10]. En klar majoritet av ledande experter tror att $\mathcal{P} \neq \mathcal{NP}$ [28], men inget har bevisats.

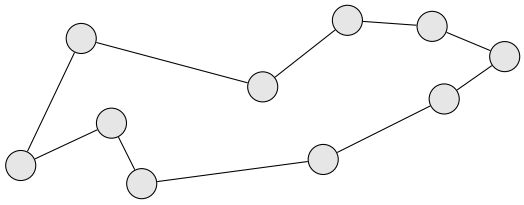
Delmängdsummaproblemet går ut på att givet en mängd S hitta en (icke-tom) delmängd $U \subseteq S$ så att $\sum_{x \in U} x = 0$. Det är trivialt att inom $\mathcal{O}(n)$ kontrollera att summan för en given delmängd är 0, men att faktiskt hitta en sådan delmängd verkar vara betydligt svårare. Problemet finns alltså i $\mathcal{NP}$, men det är inte känt om det finns i $\mathcal{P}$. Att enumerera alla delmängder skulle ge oss en $\mathcal{O}(2^n)$ algoritm.

Ett $\mathcal{NP}$-komplett problem är lika svårt som de svåraste problemen inom $\mathcal{NP}$. Delmängdsummaproblemet är $\mathcal{NP}$-komplett [5, sats 34.15, s. 1097], liksom problemet att hitta en Hamiltoncykel på en godtycklig graf [29]. För att konstatera att ett problem är $\mathcal{NP}$-komplett räcker det att visa att problemet kan reduceras inom polynomiell tid till något redan känt $\mathcal{NP}$-komplett problem, och vice-versa, som Karp gjorde med 21 $\mathcal{NP}$-kompletta problem i sin kända artikel från 1972 [29].

Ett $\mathcal{NP}$-svårt problem är åtminstone lika svårt som de svåraste problemen i $\mathcal{NP}$. Handelsresandeproblemet är ett $\mathcal{NP}$-svårt problem [2, sats 15.43, s. 405], varför en effektiv algoritm för handelsresandeproblemet skulle betyda att $\mathcal{P} = \mathcal{NP}$, dock kan $\mathcal{P} = \mathcal{NP}$ utan att det finns någon effektiv algoritm för TSP. För $\mathcal{NP}$-kompletta delmängdsummaproblemet skulle en effektiv algoritm implicera samma, men implikationen gäller nu båda vägarna.

2 Handelsresandeproblemet

Handelsresandeproblemet (TSP) [4] är det kombinatoriska optimeringsproblemet som går ut på att hitta en Hamiltoncykel r på en komplett graf så att $d(r)$ minimeras. Kravet att grafen är komplett påverkar egentligen inte problemet betydelsefullt; vi kan ge alla "förbjudna" kanter en så pass hög vikt att ingen minimal Hamiltoncykel någonsin kommer att inkludera dessa, förutom då Hamiltoncykeln nödvändigtvis kräver en sådan kant.

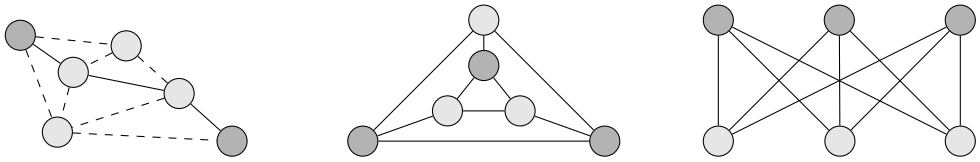


Figur 3. Lösningsexempel på handelsresandeproblemet

TSP betraktades (inte matematiskt) 1832 i en handbok för handelsresande [21]. Den irländska matematikern William Hamilton (som Hamiltoncyklar fått namnet från) formulerade problemet matematiskt samma sekel, och problemet betraktades matematiskt på 30-talet. År 1954 publicerades artikeln [4] som anses vara den första datavetenskapliga behandlingen av TSP, i vilket man löste ett s.k. storskaligt problem bestående av 49 städer. Idag har problem med nästan 100 000 noder lösts exakt, och problem med över en miljon har approximerats väldigt nära.

2.1 Relaterade problem

Optimeringsproblem på grafer som har effektiva lösningsalgoritmer inkluderar kortaste ruttproblemet (t.ex. Dijkstras algoritm [26]) och det redan nämda minsta spännträdsproblemet (t.ex. Kruskals algoritm [27]) [5]. TSP är ett av många optimeringsproblem på grafer som inte har någon känd effektiv algoritm. Andra exempel (båda $\mathcal{NP}$-svåra) är att hitta en maximal stabil mängd [1, s. 189] eller minimal färgning (däremot om grafen är bipartit, d.v.s. färgerna är två till



(a) Kortaste ruttproblemet går ut på att hitta en rutt $r = v...w$ (v, w är här mörkgråa) så att $d(r)$ minimeras

(b) En maximal stabil mängd på G är en delmängd $S \subseteq V_G$ så att $\forall_{v,w \in S} e_{vw} \notin E_G$ och $|S|$ (här 3) maximeras

(c) En minimal färgning ger varje nod en färg $\chi_v \in C$ så att $e_{vw} \in E \Rightarrow \chi_v \neq \chi_w$ och $|C|$ (här 2) minimeras

Figur 4. Optimeringsproblem på grafer

antalet som i figur 4c, kan detta göras inom polynomiell tid) [1, s. 359]. Dessa optimeringsproblem illustreras i figur 4.

Ett *linjäroptimeringsproblem* är ett optimeringsproblem där vi vill maximera värdet på $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ genom att välja ut en reell vektor $\mathbf{x} \geq 0$ så att bivillkoren $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ uppfylls. Om $\mathbf{x} \in \mathbb{Z}^n$ är detta ett *heltalsoptimeringsproblem*.

Liksom många andra kombinatoriska optimeringsproblem (t.ex. de $\mathcal{NP}$-svåra problemen i figur 4) kan TSP formuleras som ett heltalsoptimeringsproblem [3, exempel 13.1, s. 308]. Heltalsoptimering är $\mathcal{NP}$-svårt; något annat skulle leda till en kontradiktion då TSP är $\mathcal{NP}$-svårt.

De tidigare nämnda grafoptimeringsproblemen i $\mathcal{P}$ kan formuleras som linjäroptimeringsproblem [3]. För att lösa linjäroptimeringsproblem används ofta simplexalgoritmen [2, 3]. I värsta fall har simplexalgoritmen exponentiell körtid ($\mathcal{O}(b^n), b \in \mathbb{R}$) ([49] bevisar detta för den ursprungliga varianten Dantzig föreslog [50]). Algoritmen kör dock inom polynomiell tid för de flesta problem [51]. Då detta inte är fallet kan ett liknande (för de flesta tillämpningar "ekvivalent") problem lösas av algoritmen inom polynomiell tid genom att införa små slumpmässiga förändringar i inputparametrarna [52]. Spielman och Kelner föreslog en polynomiell algoritm år 2006 [53], alltså är linjäroptimering i $\mathcal{P}$.

Att hitta en Hamiltoncykel är $\mathcal{NP}$-komplett. Ett relaterat problem är att hitta en Eulercykel, om sådan finns. En Eulercykel är en cykel som inkluderar alla kanter i

grafen exakt en gång. Detta problem som verkar rätt så liknande är dock lätt att lösa (d.v.s. finns i $\mathcal{P}$). [1, kap. 3.3, s. 86-88] Leonhard Euler studerade detta problem 1736 [6] och lade på så sätt grunden för grafteori.

Det *kinesiska postiljonsproblemet* går ut på att hitta en cykel r som innehåller alla kanter (en eller flera gånger) så att $d(r)$ minimeras (postiljonen vill gå längs alla gator för att dela ut post och återvända till postkontoret). Problemet finns i $\mathcal{P}$.

Grafhandelsresandeproblemet (GrTSP) är en variant av TSP där $w(e)$ är konstant, grafen inte nödvändigtvis är komplett, och vi tillåts besöka noder mer än en gång. Detta $\mathcal{NP}$-svåra problem har på senaste tid genererat mycket intresse i och med bra approximationsalgoritmer [34] som diskuteras i stycke 2.3.

Handelsresandeproblemet har ett relaterat beslutsproblem: givet en komplett graf K_n och en viktfunktion $w(e)$, finns det en Hamiltoncykel på K_n som har distansen d eller mindre? Detta problem är $\mathcal{NP}$-komplett.

2.1.1 Specialfall

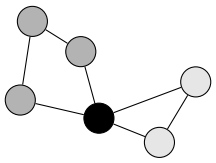
Givet en algoritm för att lösa TSP kan vi hitta en Hamiltoncykel på G genom att låta $w(e) = 1$ för alla kanter i E_G och $w(e) = 2$ för de andra. Lösningen har då totaldistansen $|V|$ om och endast om G har en Hamiltoncykel. Eftersom problemet att hitta en Hamiltoncykel är $\mathcal{NP}$-komplett och detta problem är ett specialfall av TSP vet vi att TSP (samt specialfallet $(1, 2)$-TSP som begränsar $w : E \to \{1, 2\}$ [18]) är $\mathcal{NP}$-svårt. Dessutom, då Hamiltoncykelproblemet är $\mathcal{NP}$-komplett, kan också alla andra $\mathcal{NP}$-kompletta problem lösas av en algoritm för TSP, efter några reduktioner (som kan göras inom polynomiell tid).

För det *metriska handelsresandeproblemet* [2] gäller triangelolikheten: $w(e_{ab}) + w(e_{bc}) \geq w(e_{ac})$. Ett specialfall av metriska TSP är det *Euklidiska handelsresandeproblemet* [2], där $V \subset \mathbb{R}^2$ (delmängden V är nödvändigtvis ändlig) och målet är att hitta en Hamiltoncykel r som minimerar $\sum_{n=2}^{|r|} ||r_{n-1} - r_n||_2$. Båda dessa är fortfarande $\mathcal{NP}$-svåra problem, men kan approximeras med vissa garan-

tier på hur bra lösningen är, vilket inte går för TSP allmänt. Detta diskuteras vidare i stycke 2.3.

2.1.2 Generaliseringar

I det *asymmetriska handelsresandeproblemet* kan $w(e_{ab}) \neq w(e_{ba})$ (detta kräver dessutom att vi definierar om $\psi(e)$ så att $\psi : E \rightarrow V^2$), eller så är det kanske omöjligt att gå andra vägen. Ett praktiskt exempel är ett vägnätverk där vissa vägar är enkelriktade. Ett asymmetriskt TSP kan omvandlas till ett symmetriskt TSP genom att algoritmiskt definiera en ny graf och viktfunktion, samtidigt som problemstorleken fördubblas [38]. *Handelsmanstigproblemet* (TSPP) är en variant där start- och slutnoderna (v och w, respektive) är givna. Då $v = w$ har vi det vanliga TSP, men annars skall vi hitta en minimal Hamiltonstig från v till w.



Figur 5. Fordonsruttningsproblemet

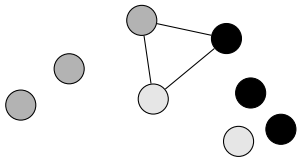
Fordonsruttningsproblemet [17] är en generalisering där n fordon tillsammans skall besöka alla noder exakt en gång (startnoden besöks $2n$ gånger), så att alla fordon börjar och slutar vid samma nod och $\sum_{k=1}^{n} d(r_k)$ minimeras, där r_k är rutten som fordon k tar. TSP är fordonsruttningsproblemet då $n = 1$. Problemet har klara tillämpningar inom logistik, och har använts för att bestämma sökrutter för räddningsflottor vid t.ex. flygkrasher då man har en lista över sannolika platser där vraket skulle kunna finnas. Figur 5 visar ett exempel där $n = 2$. Startnoden är svart, och de ljusa noderna besöks av ena fordonet och de mörka av det andra.

Fordonsruttningsproblemet har många ytterligare generaliseringar. Exempelvis kan man införa tidpunkter inom vilka ett fordon måste senast nå en viss punkt, eller om man tänker sig varor kan man införa kravet att en viss vara måste upphämtas från en viss nod för att sedan föras till en annan. Liknande modifieringar har också studerats för vanliga TSP.

Vandrande köpmansproblemet går ut på att en köpman ämnar införskaffa en

mängd varor, men alla varor finns inte tillgängliga i alla städer och priserna kan variera från stad till stad. Man minimerar alltså summan av färdkostnaderna och priserna där man köper varorna. Då varje stad har en unik vara måste alla städer besökas och problemet är ekvivalent med TSP.

I det *generaliserade handelsresandeproblemet* (GTSP) [2] är noder grupperade (grupper är disjunkta delmängder av V) och målet är att hitta en minimal cykel som besöker åtminstone en nod i varje grupp, men inte nödvändigtvis alla noder i V (se figur 6). Problemet är också känt som vandrande politikerproblemet, där en politker inför ett val vill besöka åtminstone en stad per delstat, som kan ha en eller flera städer.



Figur 6. Generaliserade handelsresandeproblemet

Givet en mängd $\mathcal{M} = \{m_1,m_2,...,m_n\}$ maskiner, en mängd $\mathcal{J}$ jobb och en tidsfunktion $c : (\mathcal{M},\mathcal{J}) \rightarrow \mathbb{R}$ som anger hur länge ett visst jobb tar för en viss maskin går *jobbschemaläggningsproblemet* ut på att associera varje jobb med en maskin i $n \times |\mathcal{J}|$ matrisen $\mathcal{X}$, så att $\mathcal{X}_{ij}$ är jobbet maskin i utför som j:te jobbet, så att $\max\{\sum_{j=1}^{|\mathcal{J}|} c(m_i, \mathcal{X}_{ij}) : i \in \{1,2,...,n\}\}$ minimeras (en maskin som utför $k < |\mathcal{J}|$ jobb har som alla "jobb" efter det k:te ett oegentligt jobb med tid 0).

En variant av jobbschemaläggningsproblemet låter oss variera $c(m_i, \mathcal{X}_{ij})$ efter $\mathcal{X}_{i(j-1)}$. TSP är specialfallet av den här varianten då $n = 1$; jobben är städerna, maskinen är handelsresenären och tiden är avståndet mellan städerna. Detta problem (för $n \in \mathbb{Z}_+$) generaliserar också en variant av fordonsruttningsproblemet där vi istället för att minimera $\sum_{k=1}^{n} d(r_k)$ minimerar $\max\{d(r_k) : k \in \{1,2,...,n\}\}$ (exempelvis vill vi minimera tiden för det sista fordonet att återvända). Jobbschemaläggning är $\mathcal{NP}$-komplett, och varianten TSP är ett specialfall av är (förstås) ett $\mathcal{NP}$-svårt problem.

2.2 Exakta algoritmer

TSP kan naivt lösas genom att räkna distansen för alla möjliga permutationer av V_{K_n} och välja ut den kortaste Hamiltoncykeln, men denna $O(n!)$ algoritm är opraktisk för allt utom de allra minsta problemen (t.ex. $59! \approx 1{,}39 \cdot 10^{80}$ är större än antalet atomer i universum). Eftersom TSP kan formuleras som ett heltals-optimeringsproblem kan vi genom att lösa heltalsoptimeringsproblemet hitta en exakt lösning på en TSP instans.

Held-Karp algoritmen, en tidig tillämpning av dynamisk programmering, ger en exakt lösning inom $O(n^2 2^n)$ [7]. Som för andra algoritmer som bygger på dynamisk programmering minskar vi körtiden genom att använda mera minne. Minnesanvändingen för Held-Karp är asymptotiskt begränsad av $O(n2^n)$ [7].

2.2.1 Branch-and-bound

Branch-and-bound bygger på att vi avgränsar lösningsmängden (branch) och för varje avgränsning räknar vi ut en nedre gräns (bound). Om vi redan har hittat en bättre lösning än den nedre gränsen för en avgränsing kan vi förkasta flera lösningar på en gång, annars avgränsar vi ytterligare och räknar nya undre gränser. På så sätt kan vi gå igenom hela lösningsmängden utan att behöva betrakta alla enskilda lösningar. Tekniken är på så sätt lik α-β-beskärning som används för att begränsa de sökträd mini-max algoritmen konstruerar [16].

Tekniken kan tillämpas på alla problem där vi kan avgränsa lösningsmängden och räkna ut en nedre gräns för avgränsningen. Tekniken har alltså tillämpats på flera problem än endast TSP (t.ex. på heltalsoptimering [3]) och på flera olika sätt på just TSP genom att avgränsa eller räkna undre gränser med olika metoder [37].

Ett sätt att implementera branch-and-bound för TSP är att lägga märke till att en undre gräns för problemet fås genom att summera över alla noder v de två kanter $e_1, e_2 \in \{e : e \in E, v \in \psi(e)\}$ med minst vikt. En avgränsning kan göras genom att lämna bort eller inkludera (oberoende om kanten faktiskt är en av de två med

minst vikt för en given nod) vissa kanter. [48]

Branch-and-bound algoritmer har i praktiken visat bättre prestanda på TSP än dynamisk programmering [3]. Den relaterade branch-and-cut tekniken ligger som grund för de största exakt lösta instanserna idag, såsom rekordet för största grafen man har löst TSP exakt på (85 900 noder) från 2006 [23].

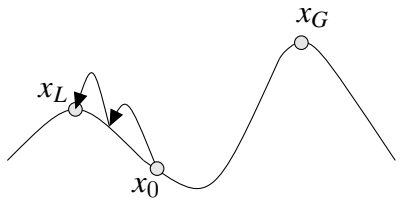
2.3 Approximering och heuristik

Förutsatt att $\mathcal{P} \neq \mathcal{NP}$ kan TSP inte lösas effektivt. Vi betraktar nu approximationsalgoritmer och heuristik för att komma fram till bra men inte nödvändigtvis optimala lösningar.

Beteckna P_{approx} för värdet på en approximation av problemet P, och P_{opt} för värdet på en optimal lösning. En k-approximationsalgoritm ger alltid en approximation så att $\frac{1}{k}P_{opt} \leq P_{approx} \leq kP_{opt}$ [13]. För TSP allmänt kan ingen sådan algoritm finnas, förutsatt att $\mathcal{P} \neq \mathcal{NP}$ [2, sats 21.1, s. 557]. För metriska TSP finns däremot sådana algoritmer. Vi betraktar en sådan i stycke 2.3.1.

För $(1, 2)$-TSP finns en $\frac{8}{7}$-approximationsalgoritm [35]. Den bästa approximationsalgoritmen för GrTSP har $k = \frac{7}{5}$ [34]. På kubiska grafer kan GrTSP approximeras med $k = \frac{4}{3}$ [36]. Om vi inte kräver en cykel utan ger två godtyckliga $v, w \in V$ som start- och slutnoder kan GrTSP approximeras med $k = \frac{3}{2}$, vilket är aningen bättre än den $\frac{5}{3}$-approximationsalgoritm som är den bästa kända approximationsalgoritmen för metriska TSPP [34].

Lösningsmängden $p_1p_2...p_n$ (slutar implicit med p_1 så att Hamiltoncykeln fullbordas) för en instans av TSP är mängden permutationer $P = \{p_1,p_2,...,p_n\}$ av V. En lösningskandidat till TSP är *k-optimal* då inga byten av plats på k stycken noder sinsemellan ger en bättre lösning. En sådan k-optimal lösning x utgör ett lokalt optimum i en *k-omgivning* av x, i likhet med de δ-omgivningar som finns kring lokala optimum för reella funktioner.



Figur 7. Bergsklättringsalgoritmen

Figur 7 visar bergsklättringsalgoritmen [16]. Vi startar från x_0 och förbättrar vår lösning genom att "klättra upp för berget" så att vi till sist kommer fram till den lokalt optimala lösningen x_L, som dock är sämre än den globalt optimala lösningen x_G. De flesta heuristiska algoritmer börjar från någon lösningskandidat som förbättras iterativt, men mer sofistikerade algoritmer har olika sätt att "lämna" lokala optimum som skiljer sig från globala optimum.

En liknande algoritm för TSP skulle givet en lösningskandidat x iterativt välja den bästa lösningskandidaten i en 2-omgivning av x, tills att x är 2-optimal. Denna algoritm kallas *2-opt* [46] och mera allmänt *k-opt* [37]. För byten mellan k noder har vi $\binom{|V|}{k}k!$ kombinationer, varför det inte är praktiskt att använda k-opt redan för större enkelsiffriga k, som vanligtvis väljs till 2 eller 3 [37]. Lin-Kernighan heuristiken [45] är en variant av k-opt där k väljs dynamiskt.

Närmaste grannalgoritmen är en enkel girig algoritm för TSP. Algoritmen startar från en godtycklig nod och fortsätter sedan till nästa obesökta nod tills att alla besökts. Om vi betraktar det Euklidiska TSP så att V är ett sampel likformigt fördelade punkter på enhetskvadraten överskrider denna lösning P_{opt} med 24% i medeltal [20], men för vissa instanser överskrider felet flera 100% [25].

Världs-TSP är en TSP instans bestående av 1 904 711 städer runt om i världen. Genom att använda en variant av Lin-Kernighan heuristiken [15] har en lösning med distans på runt 7,5 miljoner km hittats, vilket är mindre än 0,05% mer än den bästa uträknade nedre gränsen för vad som är möjligt. [14]

2.3.1 Approximering av metriska TSP

Här presenterar vi en enkel $O(n^2)$ 2-approximationsalgoritm för metriska TSP [2, sats 21.4, s. 559]. Christofides algoritm [31] med komplexiteten $O(n^3)$ är den bästa hittils kända approximationsalgoritmen ($k = \frac{3}{2}$) för metriska TSP, men är

aningen mer invecklad, trots att den i huvudsak bygger på samma idé. Förutsatt att $\mathcal{P} \neq \mathcal{NP}$ kan ingen polynomiell algoritm uppnå $k < \frac{123}{122}$ för metriska TSP (eller $k < \frac{75}{74}$ ifall asymmetriskt) [30]. Detta betyder inte att någon sådan algoritm finns, och Christofides algoritm är ännu den bästa 40 år efter att den publicerades.

För en graf G med ett minsta spännträd T gäller att $d(E_T) \leq d(\gamma)$, där γ är en (optimal) lösning på TSP på G. För att bevisa påståendet, anta motsatsen. Ta bort en kant från γ så att vi får trädet T' ($d(E_{T'}) \leq d(\gamma)$). Om $d(\gamma) < d(E_T)$ gäller att $d(E_{T'}) < d(E_T)$, men då skulle T inte vara ett minsta spännträd. □

Alla sammanhängande jämna grafer har en Eulercykel [1, sats 3.4, s. 87]. Från T, som vi kan hitta inom polynomiell tid [27], konstruerar vi nu multigrafen M genom att lägga till en extra uppsättning av alla kanter i T så att M garanterat är en jämn graf. Figur 8b visar hur M skulle se ut om vi tillämpade algoritmen på K_5 med viktfunktion enligt figur 8a, medan kanterna för minsta spännträdet T finns i figur 8b som streckade linjer.

(a) $w(e)$ (dolda kanter har vikt 3) **(b)** Ett minsta spännträd med dubbla kanter

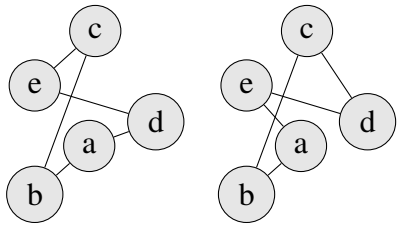
Figur 8. Minsta spännträd för K_5 (med viktfunktion), där kanterna är dubbla

Vi konstruerar nu Eulercykeln r_e på M, vilket igen kan göras inom polynomiell tid. Det är klart att $2d(E_T) = d(E_M) = d(r_e) \leq 2d(\gamma)$. För att nu få en Hamiltoncykel från r_e tar vi bort sådana kanter som leder till att vi besöker samma nod flera gånger och går istället direkt till nästa nod (t.ex. då vi redan besökt x men inte y ändras rutten vxy till vy) och får r_h. Då vi får anta triangelolikheten måste det gälla att $d(r_h) \leq d(r_e) \leq 2d(\gamma)$, alltså är detta en 2-approximationsalgoritm. □

Approximationsalgoritmer som Christofides har uppvisat betydligt bättre em-

pirisk prestanda än den övre gränsen faktorn k föreslår [25]. Många [8, 24] rekommenderar tillämpning av en approximationsalgoritm (eller enkel girig algoritm såsom närmaste grannalgoritmen) för att hitta en första lösningskandidat för någon heuristik.

2.3.2 Genetiska algoritmer



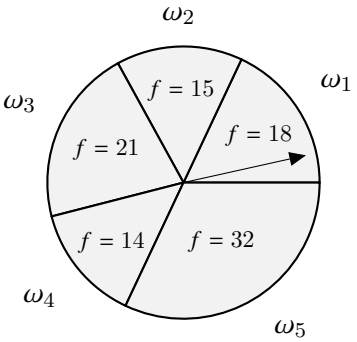
(a) BADEC **(b)** BAEDC

Figur 9. Genströngar som representerar olika Hamiltoncykler

Genetiska algoritmer (GA) [24] är en klass heuristik inspirerad av naturlig selektion. För att tillämpa GA måste vi representera lösningskandidater som genströngar. För TSP väljer vi en permutation av V som genströng likt figur 9.

Givet en mängd Ω (*population*) lösningar $\omega \in \Omega$ (*individer*) representerade som genströngar definierar vi en lämplighetsfunktion $f : \Omega \to \mathbb{R}$. En genetisk algoritm bygger på olika *genetiska operatorer*, som vanligtvis inkluderar någon form av selektion av individer för förökning och några operatorer som förändrar dessa utvalda individer för att producera en ny population (nästa *generation*).

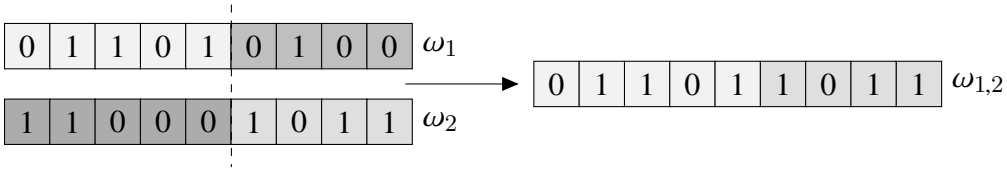
Rouletthjulselektion, en vanlig selektionsoperator, fungerar som ett rouletthjul (se figur 10) där individer med högre lämplighet har större area på hjulet och väljs därmed oftare. Istället för att kasta kula genererar vi ett slumptal, representerat av pilen i figur 10, där ω_1 valdes denna gång.



Figur 10. Rouletthjulselektion

De vanligaste operatorerna som förändrar individer är mutation och sexuell förökning. *Mutationer* är slumpmässiga förändringar av genströngen, exempelvis permutationer (jämför figur 9a och figur 9b) eller ändring av ett tecken till ett annat. *Sexuell*

förökning kombinerar två eller flera föräldrar till en ny individ. I figur 11 kombineras ω_1 och ω_2 till $\omega_{1,2}$ genom att korsa genssträngarna med en slumpmässigt vald övergångspunkt. Genssträngarna delas inte nödvändigtvis i tu; t.ex. endast en liten del av ena föräldern kunde väljas.



Figur 11. Sexuell förökning

Den klassiska genetiska algoritmen [16] börjar med selektion, följt av sexuell förökning och sedan mutationer. Sannolikheten att sexuell förökning och mutation sker (för varje enskild gen i genssträngen) är justerbara parametrar. Låt α respektive β representera dessa sannolikheter. Typiska värden är $\alpha = 0{,}9$ och $\beta = 0{,}02$. Väljer vi $\beta = 1$ har vi något likt en slumpvandring i planet. Väljs β för lågt riskerar vi att snabbt konvergera mot ett lokalt optimum som är betydligt sämre än vad som är möjligt. Efter att en ny generation producerats kommer algoritmen producera ytterligare en ny generation o.s.v. tills att något av användaren definierat terminationsvillkor uppfylls.

2.3.3 Simulerad glödgning

Simulerad glödgning (SA) [41] börjar med en slumpmässigt vald kandidat x till lösning, och avgör stokastiskt om en ny kandidat x' i en omgivning av x väljs. Algoritmen använder sig av en med tiden t avtagande temperatur $T(t)$. Vanligtvis väljer man ett sådant $T(t)$ så att nedkylningen blir långsammare vid de lägre temperaturerna, t.ex. $T(t) = \alpha^t T_0$ och $\alpha = 0{,}95$ [41].

Ett vanligt val av sannolikhetsfunktion för att välja x', givet att vi vill minimera

$f(x)$, är

$$\mathbb{P}(x, x', t) = \begin{cases} \exp\left(\dfrac{f(x) - f(x')}{T(t)}\right), & \text{om } f(x') \geq f(x) \\ 1, & \text{annars} \end{cases}.$$

Vi väljer alltså ibland en sämre lösning nu för att lämna ett lokalt optimum i hopp om att i längden komma närmare det globala. Ju högre temperatur, destu större sannolikhet att också mindre optimala kandidater blir valda. Då temperaturen avtar kommer algoritmen eventuellt stabilisera sig vid en omgivning och till sist lösning som med god sannolikhet ligger nära den bästa globalt.

Under vissa förutsättningar är konvergens garanterad, men för många problem skulle en tillräckligt långsam nedkylning leda till att SA tar längre tid än många exakta algoritmer. Specifikt för TSP förväntas SA hitta en Hamiltoncykel r så att $d(r) = P_{opt}$ långsammare än enumeration av alla lösningar. [40]

2.4 Tillämpningar

Förutom teoretisk betydelse och den självklara tillämpningen TSP har fått sitt namn ifrån (och klara moderna motsvarigheter som paketleveranser och bussturer) har TSP tillämpats på flera problem. Här beskriver vi ett antal sådana.

Optimal borrningsrutt Hitta den kortaste vägen att flytta en automatisk borr för att borra n hål. Detta är metriska TSP, med vikter baserade på en lämplig vektornorm såsom L_∞ då borrhuvudet klarar av simultana rörelser längs båda axlarna eller L_1 då rörelserna inte kan göras simultant.

Bästa spellista Vi har en mängd låtar S. Vi vill spela alla dessa en gång, men vissa låtar passar bättre efter varandra än andra (givet av $f : S^2 \to \mathbb{R}$). Vi låter då $V = S$ och $w(e_{vw}) = \max\{f(v, w) : v, w \in V\} - f(v, w)$ och söker lösningen på (eventuellt asymmetriska) TSP på denna graf.

Minimal universell DNA-sträng En universell DNA-sträng är en DNA-sträng som innehåller alla $s \in \Sigma$ DNA-strängar som delsträngar. För att hitta en sådan

sträng så att strängen är så kort som möjligt kan vi tillämpa TSP genom att låta $V = \Sigma$ och $w(e_{vw}) = |w| - k(v, w)$, där $|w|$ är längden på strängen $w \in \Sigma$ och $k(v, w)$ är antalet tecken v och w överlappar med som mest. [42]

Jobbschemaläggning på en maskin Som redan nämndes i stycke 2.1.2 är TSP ekvivalent med ett specialfall av jobbschemaläggning med tidsfunktion beroende av vilket jobb gjordes före, exempelvis då förberedelserna för nästa jobb beror på det föregående. [37]

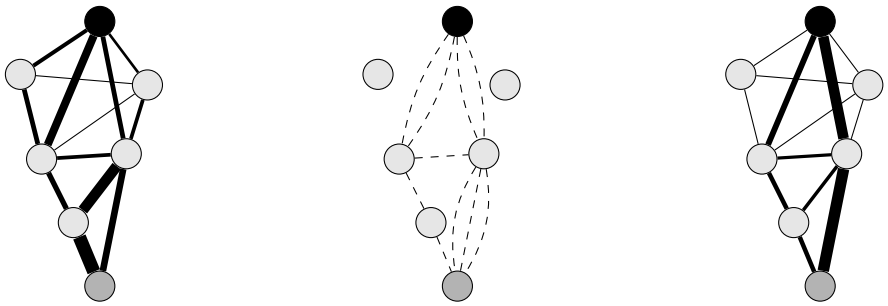
Kabelring Om vi vill lägga kabel i en sluten ring så att kabel går via alla punkter, samtidigt som vi minimerar mängden kabel har vi ett Euklidiskt TSP. Detta har tillämpats på telefonnät. Då kabeln går i ring kan nätverket tåla upp till ett länkfel då trafiken kan ruttas andra vägen. [43]

Ruttplanering för 3D skrivare En 3D skrivare smälter plast lagervis för att forma figurer. Handelsresandeproblemet utgör en del av ruttplaneringen för skrivhuvudet. Idag använder de flesta skrivare närmaste grannalgoritmen, men bättre heuristik skulle snabba upp skrivningsprocessen.

Lagerbeställningar En mängd $\mathcal{V}$ varor beställs från ett lager. Olika varor finns på olika hyllor i planet $\mathbb{R}^2$. Positionen är given av $f : \mathcal{V} \to \mathbb{R}^2$. Problemet att hitta den kortaste distansen för att plocka upp alla beställda varor i $\mathcal{V}$ är ett Euklidiskt TSP med $V = \{f(v) : v \in \mathcal{V}\}$.

3 Myrkolonioptimering

Myrkolonioptimering (ACO) [12] är en klass heuristik som bygger på myrors svärmintelligens. Exempelvis om det finns en myrstack och föda någon annanstans (representerade av de svarta och mörkgråa noderna i figur 12) och hinder mellan dessa kommer myrorna slumpmässigt välja någon väg runt hindren för att sedan följa samma väg tillbaka till stacken med födan. Samtidigt lämnar myrorna efter sig fermoner på vägen. Fermonerna ökar sannolikheten att andra myror också väljer just denna väg i framtiden. Korta vägar kommer att föredras då de snabbaste myrorna först lämnar efter sig fermoner på vägen tillbaka.



(a) Fermonspår efter ett fåtal iterationer

(b) Myrorna föredrar kanter med mera fermoner

(c) Korta och ofta valda rutter får mest fermoner

Figur 12. Simulerade myror konvergerar mot den kortaste rutten

För kortaste ruttproblemet finns effektiva deterministiska algoritmer såsom Dijkstras algoritm [26] (eller varianten A* [39]); vad vi är intresserade av är att använda ACO på sådana problem som vi inte kan enkelt kan lösa exakt. Myrkolonioptimering har tillämpats på $\mathcal{NP}$-svåra problem som graffärgning och multipla kappsäcksproblemet, TSP och generaliseringar som fordonsruttningsproblemet, samt ruttning över datanätverk (där rutten måste uppdateras dynamiskt) [12, tabell 2.1, s. 39-40]. Heuristiken har också tillämpats på kontinuerliga optimeringsproblem [44].

3.1 Myrsystem för TSP

Här betraktar vi den ursprungliga myrkolonialgoritmen *myrsystemet* tillämpat på TSP [8, 12]. Presentationen är mera rigorös än det inledande exemplet. Vi simulerar a stycken myror iterativt så att varje myra k under varje iteration fullbordar en Hamiltoncykel r_k på en given komplett graf. Myran k som befinner sig vid nod v väljer slumpmässigt att gå via en kant e_{vw} till noden w med sannolikheten

$$\mathbb{P}_k(v,w) = \begin{cases} \dfrac{\tau(v,w)^\alpha \eta(v,w)^\beta}{\sum_{u \in U} \tau(v,u)^\alpha \eta(v,u)^\beta}, & \text{om } w \in U_k \\ 0, & \text{annars} \end{cases}, \tag{1}$$

där $U_k \subseteq V$ är den mängd noder myran k inte ännu har besökt, $\tau(v,w)$ är mängden fermoner på kanten e_{vw} och $\eta(v,w)$ är en funktion som baserar sig på distansen mellan noderna, exempelvis $\eta(v,w) = (||v-w||_2)^{-1}$ för Euklidiska TSP. Då noden v besökts av k ändras $U_k \leftarrow U_k \backslash \{v\}$. Nämnaren normaliserar sannolikheten så att $\sum_{u \in U} \mathbb{P}_k(v,u) = 1$. I praktiken implementeras det här steget ofta med rouletthjulselektion [12, s. 107] (som beskrevs i stycke 2.3.2). Genom att variera parametrarna α och β kan algoritmens beteende justeras ([12] rekommenderar $\alpha = 1$ och $2 \leq \beta \leq 5$). Till sist (då $U_k = \emptyset$) återvänder myran till startnoden.

Efter varje iteration avdunstar fermonerna på alla kanter $e_{vw} \in E$ enligt

$$\tau(v,w) \leftarrow (1-\rho)\tau(v,w). \tag{2}$$

Parametern $\rho \in [0,1]$ väljs enligt önskad avdunstningsmängd ($\rho = 0.5$ föreslås av [12]). Betydelsen av att fermonerna avdunstar är inte klar för verkliga myrkolonier, men detta har stor betydelse för vår simulation [12, s. 12]. Efter avdunstning lägger vi till fermoner på varje kant e_{vw} enligt

$$\tau(v,w) \leftarrow \tau(v,w) + \sum_{k=1}^{a} \Delta_k^\tau(v,w), \tag{3}$$

där fermonmängden myran k lägger på kanten e_{vw} är $\Delta_k^\tau(v,w) = d(r_k)^{-1}$ då $e_{vw} \in r_k$, och 0 då $e_{vw} \notin r_k$. Mängden fermoner ökar alltså ju kortare r_k är, för att

simulera det kontinuerliga flödet som skulle ske i naturen.

Vi kan lätt tillämpa föregående på GTSP genom att uppdatera $U_k \leftarrow U_k \backslash A$ då vi besöker en nod v i gruppen $A \subseteq V$.

Konvergens mot den optimala lösningen är garanterad av myrsystemet (och flera varianter), men inga resultat finns gällande tiden konvergensen kommer att ske inom [12, kap. 4, s. 121-152].

3.1.1 Varianter

I det *elitistiska myrsystemet* avlägger myran med den bästa rutten hittils över *alla* iterationer en extra mängd fermoner på alla kanter som inkluderades i denna bästa rutt. Denna variant har uppvisat betydligt bättre prestanda än det ursprungliga myrsystemet.

Det *rangbaserade myrsystemet* [33] är likt myrsystemet, men $\Delta_k^\tau(v, w)$ blir nu $R(r_k)^{-1}$ istället för $d(r_k)^{-1}$ då $e_{vw} \in r_k$, och andra fallet hålls likadant, där $R(r_k)$ är rangordningen för rutten r_k. Exempelvis har vi tre myror *a*,*b*,*c* och $d(r_a) < d(r_b) < d(r_c)$. Då gäller det att $R(r_a) = 1$, $R(r_b) = 2$ och $R(r_c) = 3$. Den bästa myran hittils avlägger extra fermoner som i den elitistiska varianten. Det rangbaserade myrsystemet förbättrar till viss mån prestandan jämfört med det elitistiska myrsystemet, och markant jämfört med myrsystemet [33].

Myrkolonisystemet [47, 12] bygger fortfarande på myrsystemet men skiljer sig mera än de tidigare varianterna. Fermonerna uppdateras fortfarande efter varje iteration men nu endast längs den bästa rutten r_α hittills enligt $\tau(v, w) \leftarrow (1 - \rho)\tau(v, w) + \rho\Delta_\alpha^\tau(v, w)$ för alla $e_{vw} \in r_\alpha$ (denna uppdatering ersätter alltså (2) och (3)). En fördel härav är att komplexiteten för uppdateringen sänks från $O(|V|^2)$ till $O(|V|)$.

Myrorna uppdaterar fermonspåren lokalt genast då de förflyttar sig från en nod v till en annan w enligt $\tau(v, w) \leftarrow (1-\xi)\tau(v, w)+\xi\tau_0$. Som värde för parameterarna $\xi \in [0, 1]$ och τ_0 rekommenderar [12] $\xi = 0{,}1$ och $\tau_0 = (|V|d(r_{NN}))^{-1}$, där r_{NN} är

Hamiltoncykeln given av närmaste grannalgoritmen. Den lokala fermonuppdateringen är viktig för att undvika stagnation [12].

Myran k som befinner sig vid nod v väljer nästa nod w slumpmässigt enligt

$$w = \begin{cases} \arg\max_{u \in U_k} \{\tau(v,u)\eta(v,u)^\beta\}, & \text{om } q \leq q_0 \\ W, & \text{annars} \end{cases},$$

där W är en nod slumpmässigt vald enligt (1) och $q \in [0,1]$ är ett slumptal. Parametern q_0 låter oss justera till vilken mån algoritmen ska utforska nya kanter.

Myrsystemet och dess varianter kan kombineras med k-opt (för något k), vilket rekommenderas och diskuteras i detalj i [12, stycke 3.7, s. 92-99]. Myrkolonioptimering har med viss framgång kombinerats med slumpmässiga mutationer (se stycke 2.3.2) av rutter.

3.1.2 Jämförelse av myrkolonialgoritmer

Här jämförs olika myrsystem på TSP instanser av olika storlek. Resultaten finns sammanfattade i tabell 1.

Algoritm	$n = 100$	$n = 1000$
Myrsystem	•	•
Rangbaserat myrsystem	•	•
Elitistiskt myrsystem	•	•

Tabell 1. Bästa lösningen olika algoritmer hittade inom x iterationer

3.2 Parallel myrkolonioptimering

Eftersom ACO går ut på att myror i huvudsak som oberoende aktörer (förutom påverkan av fermonerna lämnade av varandra) försöker heuristiskt lösa ett problem verkar ACO vara en god kandidat för att implementeras parallelt [8]. Detta är ändå ingen unik fördel för ACO; många andra heuristiska metoder såsom GA är triviala att parallelisera.

Parallela myrkolonialgoritmer kan klassas som *finkorniga* och *grovkorniga*. I en finkornig implementation kör varje processorkärna en eller endast ett fåtal myror. En grovkornig implementation kör hela myrkolonier parallelt. Myrkolonierna kommunicerar sinsemellan, exempelvis för att sammanslå fermonspår. Experimentella resultat visar dock att det är effektivare att kommunicera den bästa rutten hittils och öka fermonmängden längs med de andra koloniernas bästa rutter än att dela med hela fermonmatriser.

Myrkolonialgoritmer har implementerats på grafikkort för att lösa TSP, så att prestandan förbättrats tiofaldigt [32]. Grafikkortsimplementationer medför unika utmaningar bl.a. i.o.m. att grafikkort är olämpade för icke-linjära körstigar. Lösningar till detta m.m. diskuteras i [32].

4 Avslutning

Skulle vi hitta en effektiv exakt algoritm för TSP skulle datavetenskapens kanske största öppna problem lösas då en sådan algoritm skulle implicera att $\mathcal{P} = \mathcal{NP}$. Även om fallet vore att $\mathcal{P} \neq \mathcal{NP}$ har vi idag välpresterande heuristik för problemet. Att TSP är $\mathcal{NP}$-svårt betyder inte att problemet är olösbart, åtminstone om vi nöjer oss med bra lösningar som inte är optimala, men också stora TSP instanser har lösts exakt på superdator [23].

Samtidigt som vi redan idag har bra heuristik är det värt att forska vidare kring detta. Logistiksektorn utgör en stor del av världsekonomin och betydelsen kommer troligtvis öka då världen blir allt mer global. Förbättringen av ruttplanering bättre heuristik för TSP skulle bidra med skulle möjliggöra lägre transportkostnader såväl ekonomiskt som ekologiskt.

Det är mycket vanligt att nya heuristiska metoder först tillämpats på TSP, även då deras tillämpningsområden ofta är bredare. Myrkolonioptimering tillämpades ursprungligen på TSP [8] men har senare visat sig vara bland de bästa metoderna för att hitta kortaste rutter då grafen förändras dynamiskt (t.ex. datanätverk) [12]. På så sätt har TSP inspirerat framsteg inom datavetenskap utöver ny kunskap gällande själva problemet.

Figurer

Referenser

[1] A. Bondy and U.S.R. Murty, *Graph Theory* (Graduate Texts in Mathematics 244), Springer-Verlag, 2008.

[2] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms* (Algorithms and Combinatorics 21), 5th ed., Springer-Verlag, 2012.

[3] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Dover Books on Computer Science), Dover Publications, 1998.

[4] G. Dantzig, R. Fulkerson and S. Johnson, “Solution of a large-scale traveling-salesman problem,” *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393-410, 1954.

[5] T. Cormen et al., *Introduction to Algorithms*, 3rd ed., Cambridge, MA: MIT Press, 2009.

[6] Seven Bridges of Königsberg, Leonhard Euler, 1736.

[7] M. Held and R. M. Karp, Ä dynamic programming approach to sequencing problems", Journal for the Society for Industrial and Applied Mathematics, vol. 1, no. 10, 1962

[8] M. Dorigo and L. M. Gambardella, Änt colonies for the travelling salesman problem", in biosystems, vol. 43, no. 2, pp. 73-81, 1997.

[9] C. Papadimitriou, The Euclidean travelling salesman problem is NP-complete", in Theoretical computer science, vol. 4 no. 3, pp. 237-244, 1977.

[10] M. Sipser, The history and status of the P versus NP question,"in Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM, 1992.

[11] J. N. Macgregor and T. Ormerod, "Human performance on the traveling salesman problem", in Perception & Psychophysics, vol. 58, no. 4, pp. 527–539, June 1996 doi:10.3758/BF03213088.

[12] M. Dorigo and T. Stützle, *Ant Colony Optimization*, Cambridge, MA: MIT Press, 2006.

[13] V. Vazirani Äpproximation Algorithms"

[14] World Traveling Salesman Tour, `http://www.math.uwaterloo.ca/tsp/world/`

[15] `http://www.akira.ruc.dk/~keld/research/LKH/`

[16] P. Norvig, S. Russell, Artificial Intelligence: A Modern Approach

[17] G. Dantzig, J. Ramser, The Truck Dispatching Problem,"in Management Science, vol. 6 no. 1, pp. 80–91, October 1959 doi:10.1287/mnsc.6.1.80.

[18] C. Papadimitriou, M. Yannakakis, The traveling salesman problem with distances one and two,"in Mathematics of Operations Research, vol. 18 no. 1, pp. 1-11, 1993.

[19] J.K. Lenstra and A.R. Kan, Some simple applications of the travelling salesman problem,"in Journal of the Operational Research Society, vol. 26 no. 4, pp. 717-733, 1975.

[20] D.S. Johnson and L.A. McGeoch, The traveling salesman problem: A case study in local optimization,"in Local search in combinatorial optimization, 1, pp. 215-310, 1997.

[21] Der Handlungsreisende – wie er sein soll und was er zu tun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur"(Handelsresenären — hurdan han skall vara och vad han skall göra för att få kommission och vara säker på god framgång i sina affärer — av en gammal kommissionsresenär), 1812

[22] Shi, Xiaohu H., et al. Particle swarm optimization-based algorithms for TSP and generalized TSP,"in Information Processing Letters vol. 103 no. 5, pp. 169-176, 2007.

[23] Applegate et al., 2006

[24] Grefenstette, John, et al. Genetic algorithms for the traveling salesman problem.Proceedings of the first International Conference on Genetic Algorithms and their Applications. 1985.

[25] A. Haque et al. “An Empirical Evaluation of Approximation Algorithms for the Metric Traveling Salesman Problem”, 2013.

[26] E. W. Dijkstra “A note on two problems in connexion with graphs,” in Numer. Math. vol. 1, no. 1, pp. 269-271, 1959.

[27] J.B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” in Proceedings of the American Mathematical society, vol. 7, no. 1, pp.48-50, 1956.

[28] L.A. Hemaspaandra, SIGACT News Complexity Theory Column 36, https://www.cs.umd.edu/~gasarch/papers/poll.pdf, 2002

[29] R.M. Karp, “Reducibility among combinatorial problems,” in Complexity of computer computations, pp. 85-103, 1972.

[30] M. Karpinski et al., “New Inapproximability Bounds for TSP”, 2013.

[31] N. Christofides, “Worst-case analysis of a new heuristic for the travelling salesman problem,” No. RR-388. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[32] L. Dawson and I. Stewart, "Improving Ant Colony Optimization performance on the GPU using CUDA,"in Evolutionary Computation (CEC), 2013 IEEE Congress on. IEEE, 2013.

[33] B. Bullnheimer et al., Ä new rank based version of the Ant System. A computational study,"1997.

[34] A. Sebő, and J. Vygen, Shorter tours by nicer ears: 7/5-approximation for graphic TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs,ärXiv preprint arXiv:1201.1870, 2012.

[35] P. Berman and M. Karpinski, “8/7-approximation algorithm for (1,2)-TSP,” in Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA ’06). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 641-648, 2006.

[36] S. Boyd et al., “TSP on Cubic and Subcubic Graphs,” in Integer Programming and Combinatoral Optimization. IPCO 2011. Lecture Notes in Computer Science, vol 6655. Springer, Berlin, Heidelberg, 2011.

[37] G. Laporte, “The traveling salesman problem: An overview of exact and approximate algorithms,” in European Journal of Operational Research, vol. 59, no. 2, pp. 231-247, 1992.

[38] R. Jonker, and T. Volgenant, “Transforming asymmetric into symmetric traveling salesman problems,” in Operations Research Letter vol. 2, pp. 161–163, 1983. doi:10.1016/0167-6377(83)90048-2.

[39] P. E. Hart et al., “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” in IEEE Transactions on Systems Science and Cybernetics SSC4, vol. 4, no. 2, pp. 100–107, 1968. doi:10.1109/TSSC.1968.300136.

[40] E. Aarts et al., “Simulated Annealing,” in Local Search in Combinatorial Optimization 1st ed., E. Aarts and J.K. Lenstra, Eds. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[41] S. Kirkpatrick et al., Optimization by Simulated Annealing, IBM Research Report RC 9355, 1982.

[42] http://www.math.uwaterloo.ca/tsp/apps/dna.html

[43] http://www.math.uwaterloo.ca/tsp/apps/sonet.html

[44] S. Krzysztof and M. Dorigo, “Ant colony optimization for continuous and mixed-variable domains,” 2008.

[45] S. Lin and B.W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” in Operations research, vol. 21, no. 2, pp.498-516, 1973.

[46] G.A. Croes, “A method for solving traveling-salesman problems,” in Operations research, vol. 6, no. 6, pp.791-812, 1958.

[47] M. Dorigo and L.M. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” in IEEE Transactions on evolutionary computation, vol. 1, no. 1, pp. 53-66, 1997.

[48] http://lcm.csa.iisc.ernet.in/dsa/node187.html

[49] V. Klee and G.J. Minty, "How good is the simplex algorithm?" in Shisha, Oved. Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin). New York-London: Academic Press, pp. 159–175, 1972

[50] G. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in activity analysis of production and allocation, 1951.

[51] K. Borgwardt, "The average number of pivot steps required by the simplex-method is polynomial," in Mathematical Methods of Operations Research, vol. 26, no. 1, pp. 157-177, 1982.

[52] D.A. Spielman and S.H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," in Journal of the ACM, vol. 51, no. 3, pp.385-463, 2004.

[53] J.A. Kelner and D.A. Spielman, "A randomized polynomial-time simplex algorithm for linear programming," in Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, ACM, 2006.