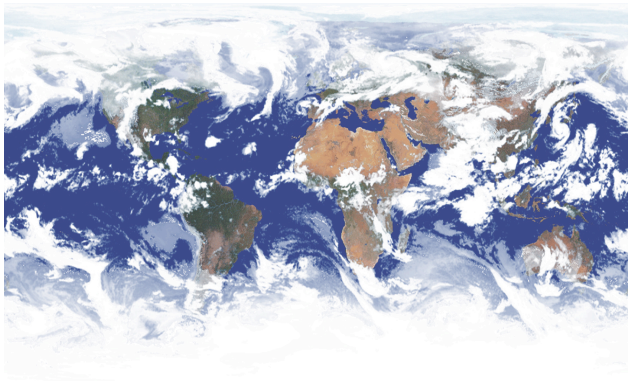


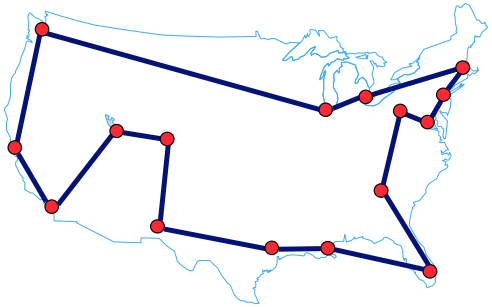
Den rejsende sælgers problem

ved
Keld Helsgaun

Den rejsende sælgers problem

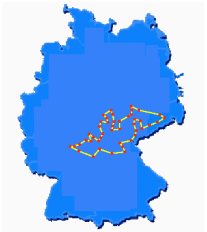
En sælger skal besøge ***n*** byer og vende tilbage til sit udgangspunkt

I hvilken rækkefølge skal han besøge byerne, hvis han ønsker at minimere sine rejseomkostninger?



Tur for 16 amerikanske byer

Første omtale af problemet 1832



45 byer i Tyskland

Der Handlungsreisende - wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein
- Von einem alten Commis-Voyageur

Voigt

Konkurrence fra 1962

Matematisk formulering

Rejseomkostninger kan f.eks. måles i afstand, tid eller penge

Givet en matrix $\boldsymbol{C} = \{\boldsymbol{c_{ij}}\}$, hvor $\boldsymbol{c_{ij}}$ repræsenterer omkostningen ved at gå fra by $\boldsymbol{i}$ til by $\boldsymbol{j}$ ($\boldsymbol{i}, \boldsymbol{j} = 1, ..., \boldsymbol{n}$)

Find en permutation ($\boldsymbol{i}_1, \boldsymbol{i}_2, ..., \boldsymbol{i}_n$) af tallene fra 1 til $\boldsymbol{n}$, der minimerer summen

$$c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_{n-1} i_n} + c_{i_n i_1}$$

Problemtyper

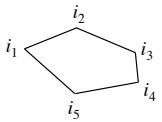
Egenskaber ved $\boldsymbol{C}$ benyttes til at klassificere problemer:

Hvis $\boldsymbol{c_{ij}} = \boldsymbol{c_{ji}}$ for alle $\boldsymbol{i}$ og $\boldsymbol{j}$, siges problemet at være **symmetrisk**. Ellers er det **asymmetrisk**

Hvis trekantuligheden gælder ($\boldsymbol{c_{ik}} \leq \boldsymbol{c_{ij}} + \boldsymbol{c_{jk}}$ for alle $\boldsymbol{i}, \boldsymbol{j}$ og $\boldsymbol{k}$), siges problemet at være **metrisk**

Hvis $\boldsymbol{c_{ij}}$ er Euklidiske afstande imellem punkter i planet, siges problemet at være **Euklidisk**

Grafformulering



Lad $G = (V, E)$ være en vægtet graf, hvor $V = \{1, 2, ..., n)$ er mængden af *knuder*, og $E = \{(i, j) \mid i \in V, j \in V\}$ er mængden af *kanter*.

En *cykel* er en mængde af kanter $\{(i_1, i_2), (i_2, i_3), ..., (i_k, i_1)\}$, hvor $i_p \neq i_q$ for alle $p \neq q$. *Længden* af en cykel er summen af dens kantomkostninger.

En *tur* (*Hamiltoncykel*) er en cykel, hvor $k = n$.

TSP kan nu formuleres således:
"Givet en graf G. Find en tur i G af minimal længde".

Motivation (1)

Hvorfor beskæftige sig med den rejsende sælgers problem (TSP)?

(1) **Prototype-problem**. Algoritmer og teknikker udviklet til løsning af TSP kan overføres til andre kombinatoriske optimeringsproblemer

Alle større landvindinger inden for kombinatorisk optimering kan direkte eller indirekte henføres til studiet af TSP:
Branch and bound, dynamisk programmering, beregningskompleksitet, snitplaner i LP, metaheuristikker

Motivation (2)

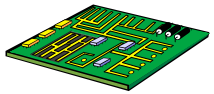
(2) Mange anvendelser

TSP opstår naturligt som delproblem ved løsning af problemer inden for transport og logistik:

- afhentning af børn med en skolebus
- udbringning af mad til pensionister

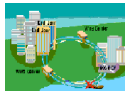
Men TSP opstår også inden for mange andre områder:

- boring af huller på en printplade (VLSI)
- planlægning af opgaveafviklingen på en maskine

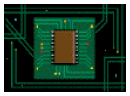


Flere praktiske anvendelser

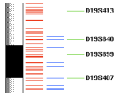
Design af et optisk fibernetværk



Kredsløb til afprøvning af integrerede kredse



Genetisk sekvensanalyse



Beregningskompleksitet

For et symmetrisk problem med ***n*** byer er antallet af mulige ture lig med

$$\frac{(n-1)\cdot(n-2)\cdot\ldots\cdot 2\cdot 1}{2} = \frac{(n-1)!}{2}$$

Hvis vi antager, at det tager 10^{-9} sekunder at bestemme hver mulig tur, bliver tidsforbruget stort, selv for små værdier af ***n***:

n	*Antal ture*	*Tidsforbrug*
20	$\approx 10^{17}$	$\approx$ 2 år
25	$\approx 10^{24}$	$\approx$ 10 millioner år
30	$\approx 10^{31}$	$\approx 10^{15}$ år

Antal ruter

N = 3: 1 rute.
N = 4: 3 ruter.
N = 5: 12 ruter.
N = 6: 60 ruter.
N = 7: 360 ruter.
N = 8: 2.520 ruter.
N = 9: 20.160 ruter.
N = 10: 181.440 ruter.
N = 11: 1.814.400 ruter.
N = 12: 19.958.400 ruter.
N = 13: 239.500.000 ruter.
N = 14: 3.113.500.000 ruter.
N = 15: 43.589.000.000 ruter.
N = 17: 10.461.000.000.000 ruter.
N = 19: 3.201.200.000.000.000 ruter.
N = 21: 1.216.500.000.000.000.000 ruter.
N = 23: 562.000.000.000.000.000.000 ruter.
N = 25: 310.220.000.000.000.000.000.000 ruter.
N = 30: 4.420.900.000.000.000.000.000.000.000.000 ruter.
N = 35: 147.620.000.000.000.000.000.000.000.000.000.000.000 ruter.
N = 40: 10.019.900.000.000.000.000.000.000.000.000.000.000.000.000.000 ruter.

Problemkompleksitet

TSP er $\mathcal{NP}$-fuldstændigt

Enhver algoritme, der med garanti finder optimum, vil have et tidsforbrug, der vokser eksponentielt med problemstørrelsen

Løsningsalgoritmer

- **Eksakte** algoritmer
  Dynamisk programmering
  Lineær programmering
  (snitplaner, facetter)

$$
\begin{aligned}
&\text{minimize } \sum_e c_e x_e \\
&\text{subject to} \\
&x(\delta(v)) = 2 \text{ for all } v \in V \\
&x(\delta(S)) \geq 2 \text{ for all } S \subset V \text{ with } \emptyset \neq S \neq V \\
&x_e \leq 1 \\
&x_e \geq 0 \\
&x_e \text{ integral}
\end{aligned}
$$

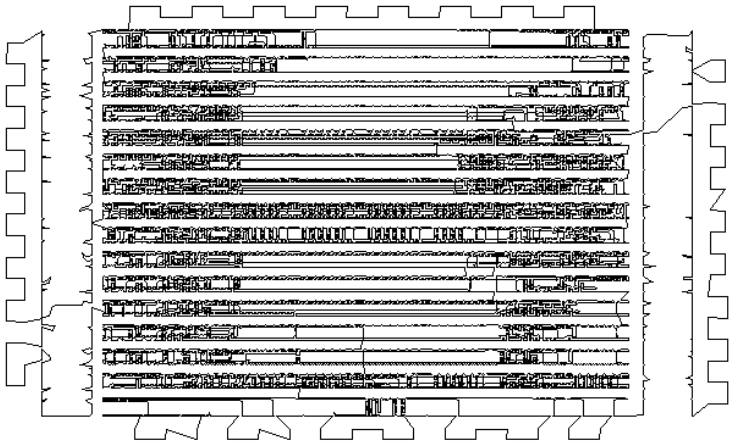
- **Approksimative** algoritmer
  Heuristikker
  Metaheuristikker
  (simuleret udglødning, myrekolonier, genetiske algoritmer, tabusøgning)

Eksakte løsninger (milepæle)

År	Forskere	Byer
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R. M. Karp	64
1975	P. M. Camerini, L. Fratta, F. Maffioli	100
1977	M. Grötschel	120
1980	H. Crowder, M.W. Padberg	318
1987	M. Padberg, G. Rinaldi	532
1987	M. Grötschel, O. Holland	666
1987	M. Padberg, G. Rinaldi	2.392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7.397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13.509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15.112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24.978
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun, S. Dash, D. Espinoza, R. Fukasawa, M. Goycoolea	85.900

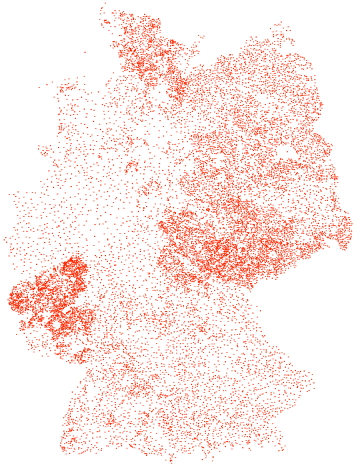
15,112-by problemet blev løst ved hjælp af 110 cpu'er med et samlet CPU-forbrug på cirka **22 år**.
24,978-by problemet blev løst med et samlet CPU-forbrug på cirka **84 år**.

pla85900: optimal tur



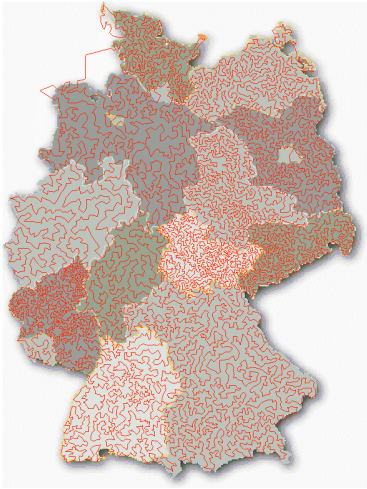
CPU-tidsforbrug: **136 år**

15.112-by problemet



15.112 byer i Tyskland

Optimal løsning



1 ud af $10^{57.000}$ mulige ture

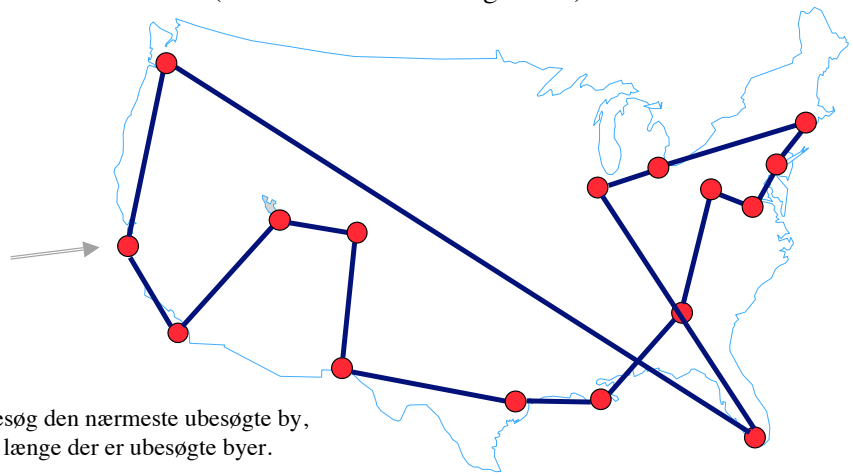
Antallet af atomer i hele det kendte univers antages at være 10^{83}

Approksimative algoritmer

- **Turkonstruktuerende algoritmer**
  En tur konstrueres fra "bunden"
- **Turforbedrende algoritmer**
  En given tur forbedres så meget som muligt
- **Sammensatte algoritmer**
  En tur konstrueres og forbedres

Nærmeste nabo

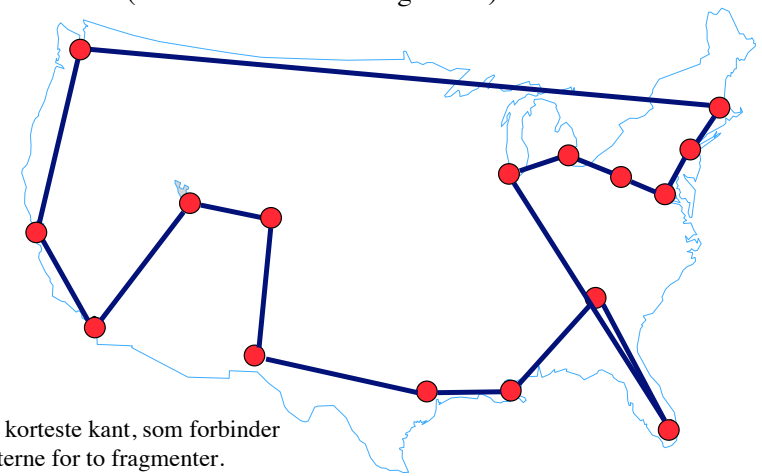
(en turkonstruerende algoritme)



Besøg den nærmeste ubesøgte by, så længe der er ubesøgte byer.

Multi-fragment (Greedy)

(en turkonstruerende algoritme)

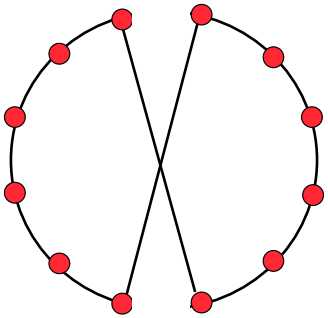


Tilføj den korteste kant, som forbinder endepunkterne for to fragmenter.

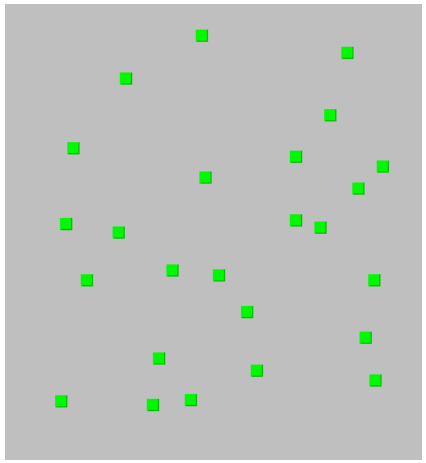
2-opt

(en turforbedrende algoritme)

Erstat 2 forbindelser med 2 andre forbindelser, så der fremkommer en kortere tur

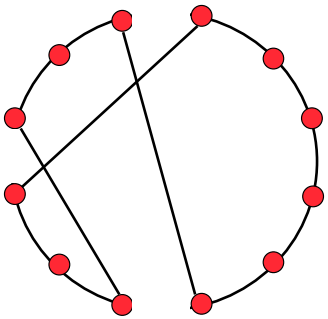


Animering af 2-opt



3-opt

Erstat 3 forbindelser med 3 andre forbindelser, så der fremkommer en kortere tur



Løsningskvalitet

15.112-by problemet

Algoritme	Afvigelse fra optimum
Nærmeste nabo	25%
Multi-fragment	15%
2-opt	14%
3-opt	8%

λ-opt

En tur siges, at være **λ-optimal**, hvis det er umuligt at opnå en kortere tur ved at erstatte λ af dens forbindelser med en mængde af λ forbindelser

Hvis en tur er λ-optimal, er den også λ'-optimal for $1 \leq \lambda' \leq \lambda$

En tur med ***n*** byer er optimal, hvis og kun hvis den er ***n***-optimal

Kompleksitet af λ-opt

Kvaliteten af en tur øges, når λ øges

Men

(1) tiden for at undersøge, om en tur er λ-optimal, vokser meget hurtigt. Med en naiv implementering er tidskompleksiteten $O(n^{\lambda})$

(2) der er ingen ikke-triviel øvre grænse for antallet af λ-ombytninger

Lin-Kernighan

Ofte vælges λ=2 eller λ=3
Men det er en ulempe, at λ skal vælges på forhånd

Lin og Kernighan fjernede denne ulempe ved at introducere en **variabel** λ-opt algoritme

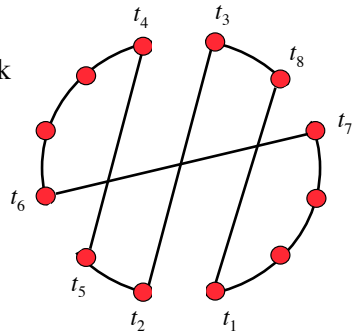
Algoritmen undersøger for stigende værdier af λ, om en ombytning af λ forbindelser vil resultere i en kortere tur. Dette fortsætter, indtil et fastsat stopkriterium er opfyldt

Begrænsning af søgningen (1)

(1) Sekventielle ombytninger

Forbindelser fjernes og tilføjes skiftevis, og således at to på hinanden følgende forbindelser har et endepunkt tilfælles

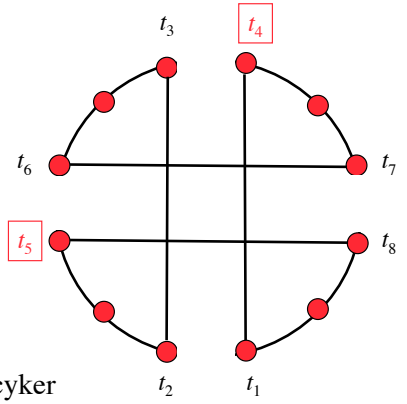
4-opt træk



Kæden af kanter udgør en **alternaternde cykel**

Eksempel på et ikke-sekventielt træk

Dobbelt-bro-træk

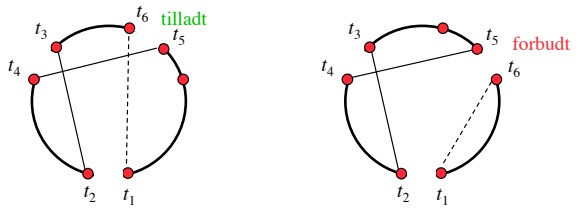


To alternerende cyker

Begrænsning af søgningen (2)

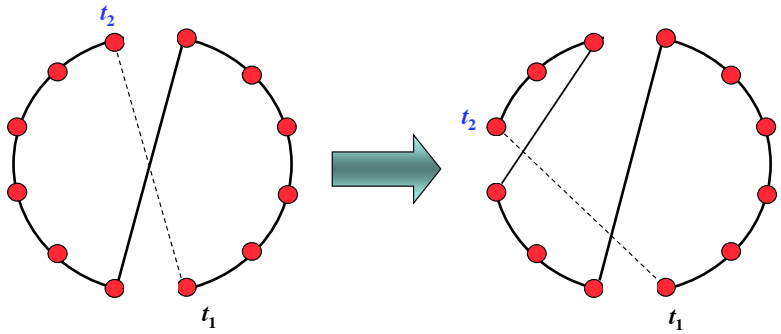
(2) Gennemførlighed

For $i \geq 3$ må en forbindelse (t_{2i-1}, t_{2i}) kun fjernes, hvis tilføjelsen af forbindelsen (t_{2i}, t_1) resulterer i en tur



I den originale algoritme, LK, kan et λ-opt træk repræsenteres som et 2- eller 3-opt træk, efterfulgt af en serie af 2-opt træk

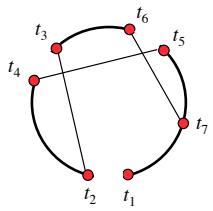
Træk som en følge af 2-opt træk



Begrænsning af søgningen (3)

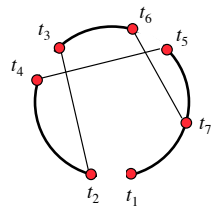
(3) Positiv gevinst
Ved konstruktion af et λ-opt træk, skal den foreløbige gevinst, G_i, altid være positiv, hvor

$$G_i = \boldsymbol{c}(t_1, t_2) - \boldsymbol{c}(t_2, t_3) + \boldsymbol{c}(t_3, t_4) - \ldots - \boldsymbol{c}(t_{2i}, t_{2i+1})$$



Begrænsning af søgningen (4)

(4) Disjunkte forbindelser
En tidligere fjernet forbindelse kan ikke blive tilføjet, og en tidligere tilføjet forbindelse kan ikke blive fjernet



Begrænsning af søgningen (5, 6 og 7)

(5) Beskæring af søgebredden
Søgning efter en forbindelse, der skal tilføjes, begrænses til de 5 nærmeste naboer til t_{2i}

(6) Gode faste forbindelser bibeholdes
For $i \geq 4$ må en forbindelse ikke fjernes, hvis den er fælles for et givet antal hidtil korteste ture

(7) Overflødigt overarbejde undgås
Søgning efter en forbedring stoppes, hvis den aktuelle tur er den samme som en tidligere tur

Dirigering af søgningen

Udover regler til begrænsning af søgningen indeholder den originale algoritme også en række **heuristiske** regler, hvis formål er at forbedre chancen for at finde gevinstgivende træk

- Hvis der er flere muligheder for at tilføje en forbindelse, vælges den, der giver den største tilvækst af gevinsten
- Hvis der er flere muligheder for at fjerne en forbindelse, vælges den længste

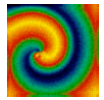
Effektiviteten af LK

LK betragtes i dag som den mest effektive approksimative algoritme til løsning af TSP

LK kan på rimelig tid bestemme løsninger, som afviger fra optimum med omkring 2%

Med **itereret** LK opnås typisk en kvalitet på 1%

Itereret LK



```
bestTour = LK(randomTour());
for (run = 2; run <= numberOfRuns; run++) {
    tour = LK(kick(bestTour));
    if (cost(tour) < cost(bestTour))
        bestTour = tour;
}
```

`kick`-operationen kan, for eksempel, være et tilfældigt dobbelt-bro-træk.

LKH

Lin-Kernighan-Helsgaun algoritmen

En turforbedrende algoritme, der finder næroptimale løsninger til TSP

Faktisk finder algoritmen ofte optimum

Centrale ændringer i forhold til LK

(1) Der benyttes 5-opt træk, som det basale træk (i stedet for 2-opt træk)

(2) Mængden af kandidater for de forbindelser, der kan indgå i en tur bestemmes ved sensitivitetsanalyse (i stedet for ved bestemmelse af nærmeste nabo)

Sensitivitetsanalyse i LKH

Forbindelsers chance for at tilhøre en optimal tur estimeres ved hjælp af det såkaldte **α-mål**

For hver kant i den til problemet svarende graf bestemmes kantens **α-værdi** som den tilvækst omkostningen for et mindste udspændende træ ville få, hvis kanten skulle være med i et sådan træ

Bestemmelse af α-værdierne kan gøres i $O(n^2)$ tid

Effektiviteten af α-målet

α-målet kan forbedres ved **Lagrange relaksation** og **subgradient optimering**

Målet er så godt, at man i mange tilfælde kan nøjes med kandidatmængder, der for hver by blot omfatter de 5 α-nærmeste naboer

Andre effektiviseringer

(1) Startture genereres ikke helt tilfældigt
Forbindelserne vælges blandt kandidatforbindelserne

(2) Den første kant, der fjernes i et træk, (t_1, t_2), må ikke tilhøre den hidtil bedste tur

(3) Ture repræsenteres med en datastruktur, som tillader udførelse af et λ-opt træk i $O(\lambda\sqrt{n})$ tid

(4) Tiden for afstandsberegninger reduceres (caching)

(5) Tiden til at undersøge, at der ikke er mulighed for flere forbedringer, reduceres (hashing, don't look bits)

LKH version 2

Version 2 af LKH forbedrer den oprindelige implementation på følgende punkter:

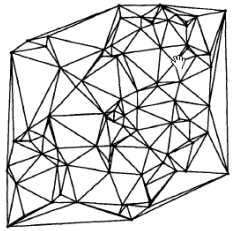
(1) Løsning af meget store problemer (mere end 1 million byer), er nu muligt

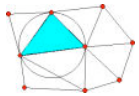
(2) Det er muligt at opnå endnu bedre løsningskvalitet

Løsning af meget store problemer

En god starttur kan bestemmes hurtigt med multi-fragment algoritmen

Delaunay-triangularisering kan benyttes til at effektivisere sensitivitetsanalysen





Øget løsningskvalitet

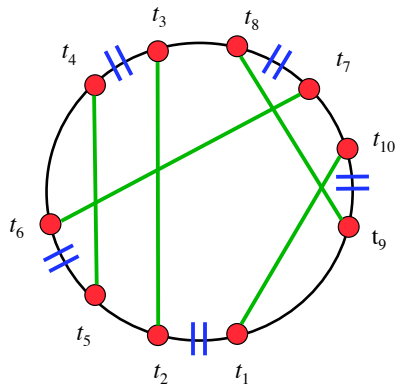
(1) Basistræk kan være *K*-opt for en vilkårlig værdi af *K* (ikke blot ≤ 5 som i version 1)

(2) Træk kan omfatte ikke-sekventielle træk

(3) Ture kan flettes

(4) Et problem kan opløses i delproblemer

Generelt *K*-opt træk



Delproblemer:

(1) Er trækket lovligt?

(2) Hvorledes udføres det i givet fald?

Er trækket lovligt?

Antallet af tilfælde, der skal undersøges, vokser hurtigt for stigende værdier af *K*:

K	Mulige sekvenser	Lovlige sekvenser	Andel
2	2	1	50.0%
3	8	4	50.0%
4	48	20	41.7%
5	384	148	38.5%
6	3.840	1.348	35.1%
7	46.080	15.104	32.8%
8	645.120	198.144	30.7%
9	10.321.920	2.998.656	29.1%
10	185.794.560	51.290.496	27.6%

Er trækket lovligt?

Implementeringsmuligheder

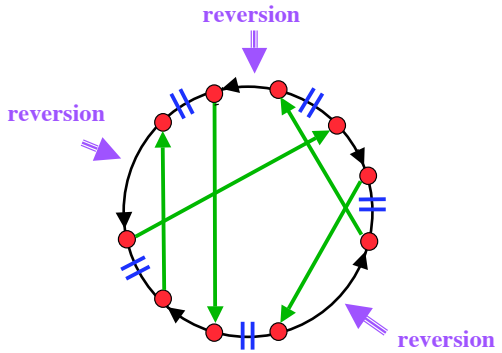
(1) Programgenerering (David Applegate)
Antal producerede programlinjer:

$K = 6$:	120.288
$K = 7$:	1.259.863
$K = 8$:	17.919.296

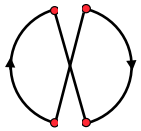
(2) Rekursion og sortering (Keld Helsgaun)
Lovlighed kan afgøres i $O(K \log K)$ tid

Hvorledes udføres et lovligt træk?

Et K-opt træk udføres ved at vende 0, 1 eller flere sekvenser af byer på turen om (foretage **reversioner**)



2-opt træk og reverseringer



Et 2-opt træk bevirker en reversering af et tursegment.

Sætning:

Ethvert K-opt træk ($K \geq 2$) er ækvivalent med en følge af højst K 2-opt træk.

Hvorledes udføres et lovligt træk?

(mest effektivt)

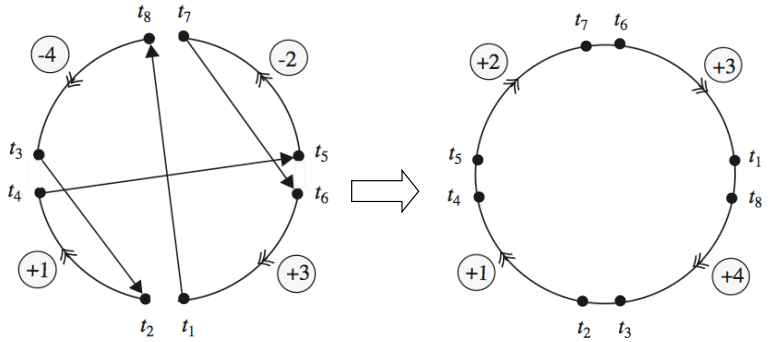
Bestemmelse af den korteste følge af reversioner, der skal bruges for at udføre et K-opt træk viser sig at være ækvivalent med et centralt problem ved sammenlignende studier af gener:

Sortering ved reversering af permutationer med fortegn

En **permutation med fortegn** er en sædvanlig permutation, hvor hvert element er forsynet med et fortegn, + eller -

Problemet kan løses i polynomiel tid: $O(K^2)$

Et 4-opt træk og dets tilsvarende permutation med fortegn



(+1, -4, -2, +3) ⇨ (+1, +2, +3, +4)

Udførelse 4-opt trækket ved sortering af dets permutation

(+1, -4, -2, +3)

⇩

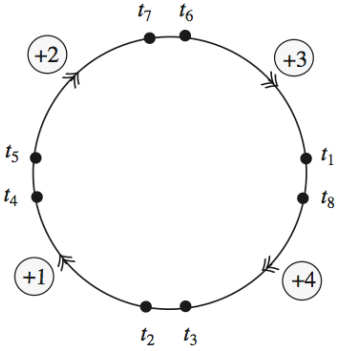
(+1, -4, -3, +2)

⇩

(+1, -2, +3, +4)

⇩

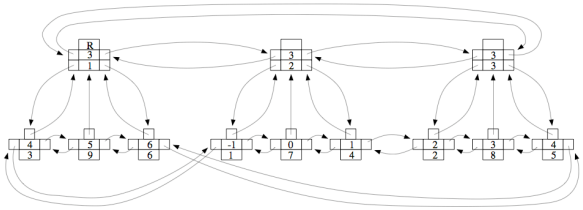
(+1, +2, +3, +4)



Trækket foretages med 3 reversioner (2-opt træk)

Tur repræsentation

To-niveau-træ



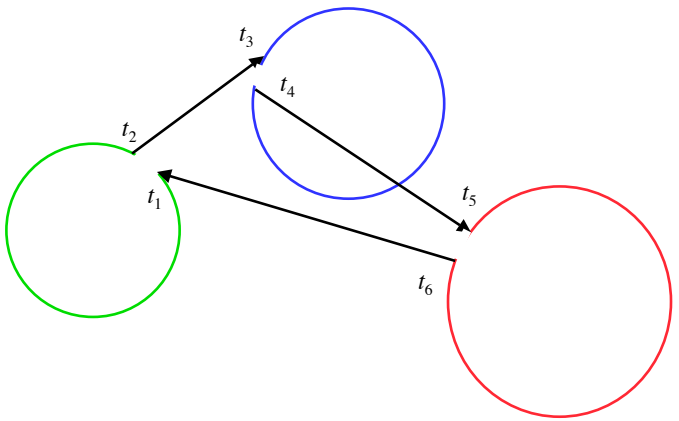
Operation	Kompleksitet
`PRED(t), SUC(t)`	$O(1)$
`BETWEEN(t1, t2, t3)`	$O(1)$
`FLIP(t1, t2, t3, t4)`	$O(\sqrt{n})$

Ikke-sekventielle træk

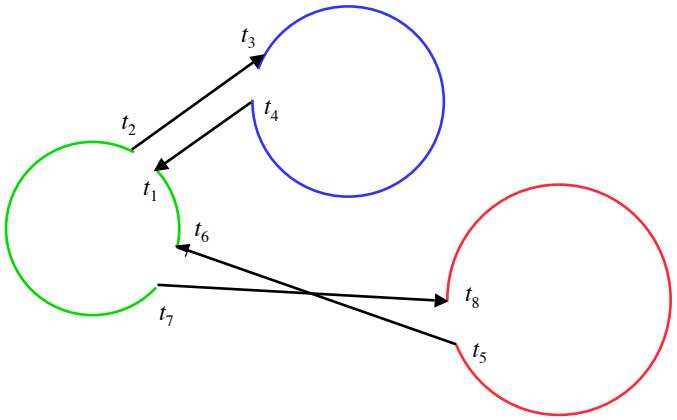
Hvis der under konstruktionen af et *K*-opt træk findes et turforbedrende, men **ulovligt** træk, forsøges de opståede cykler føjet sammen på en sådan måde, at der fremkommer et turforbedrende, lovligt træk

Cykler kan føjes sammen ved brug af en eller flere **alternerende cykler**

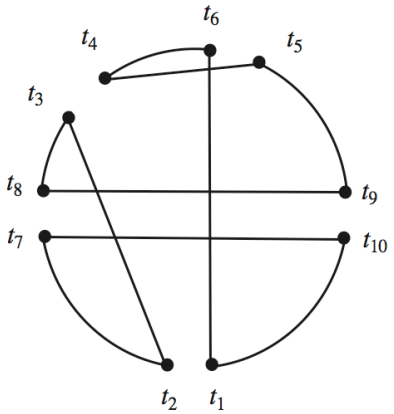
Sammenføjning med 1 alternerende cykel



Sammenføjning med 2 alternerende cykler

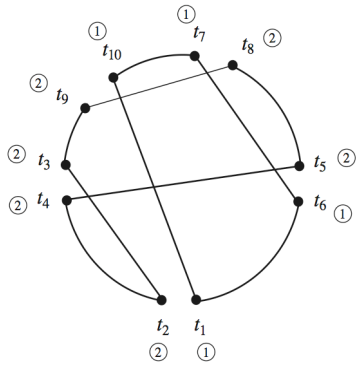


Repræsentation af potentielle træk



	t	*incl*
1	t_1	6
2	t_2	3
3	t_3	2
4	t_4	5
5	t_5	4
6	t_6	1
7	t_7	10
8	t_8	9
9	t_9	8
10	t_{10}	7

Afgørelse af cykelmedlemskab



1. Sorter t-følgen:
   $(t_1, t_2, t_4, t_3, t_9, t_{10}, t_7, t_8, t_5, t_6)$
   Tid: $O(K \log K)$

2. Bestem cykelnummeret for hver t-knude
   Tid: $O(K)$

Cykelnummeret for enhver knude kan bestemmes ved binær søgning i den sorterede t-følge
Tid: $O(\log K)$

Begrænsning af søgningen efter alternerende cykler

(1) To alternerende cykler må ikke haves fælles kanter.

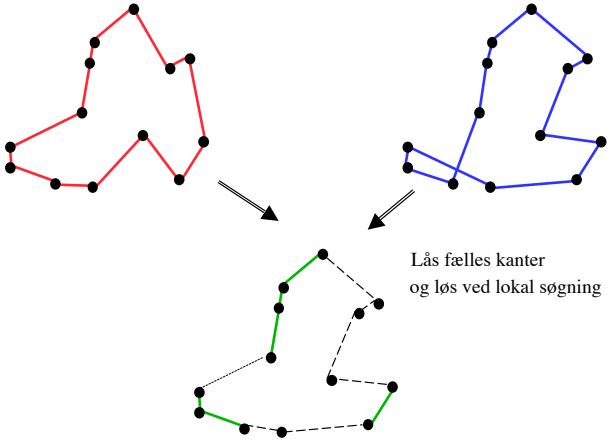
(2) Alle ud-kanter i en alternerende cykel skal tilhøre fællesmængden af kanter i den aktuelle tur og cyklernes kanter.

(3) Alle ind-kanter i en alternerende cykel skal forbinde to cykler.

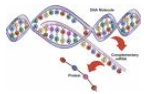
(4) Startknuden i en alternerende cykel skal tilhøre den korteste cykel (cyklen med færrest kanter).

(5) Konstuktion af en alternerende cykel påbegyndes kun, hvis den aktuelle gevinst er positiv.

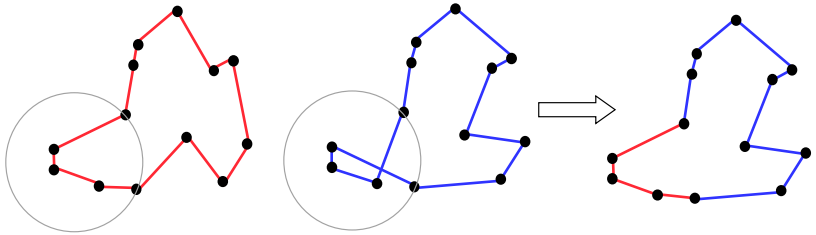
Fletning af ture



Iterativ partiel omskrivning

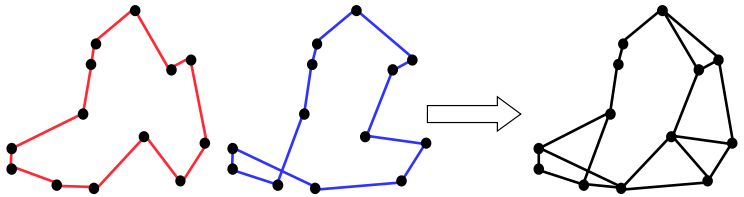


Givet to ture. Søg efter kantfølger, som indeholder de samme knuder, men i forskellig rækkefølge, og hvor start- og slutknuden er den samme. Erstat den dyreste kantfølge med den billigste.



Backbone-styret søgning

Kanterne i ture, der er frembragt efter et fast antal indledende iterationer, kan benyttes som kandidatkanter i de efterfølgende iterationer



Tur-baseret opdeling

En tur opdeles i segmenter af samme størrelse.

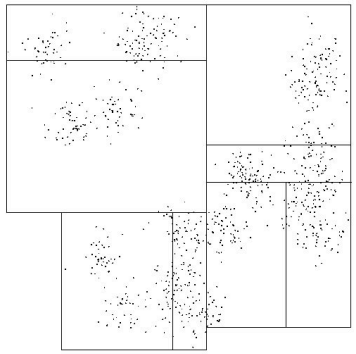
Hvert segment omdannes til en tur ved tilføjelse af en fast kant imellem dets to endepunkter, og turen forsøges forbedret.

Forbedrede segmenter sættes på plads igen.

Prøv igen med en opdeling, hvor hvert nyt segment omfatter halvdelen af byerne fra to gamle nabosegmenter.

Geometrisk opdeling

Opdel området, der indeholder alle byerne, i rektangler med lige mange byer i hvert.

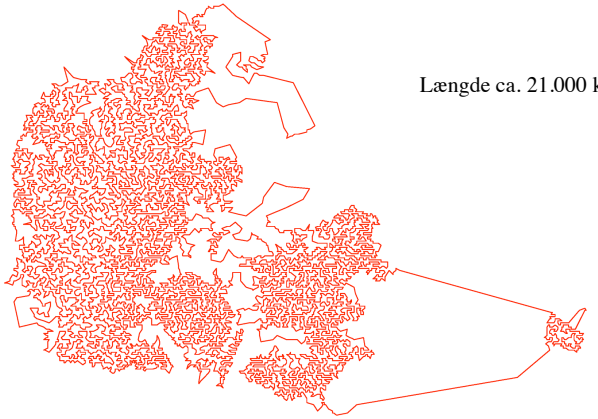


Resultater

I det følgende gives eksempler på løsninger fundet af LKH

For visse af eksemplerne kendes de optimale løsninger ikke. For disse vurderes løsningernes kvalitet i forhold til kendte nedre grænser (bestemt af Applegate m.fl. ved hjælp af snitplan-metoder)

dk11455

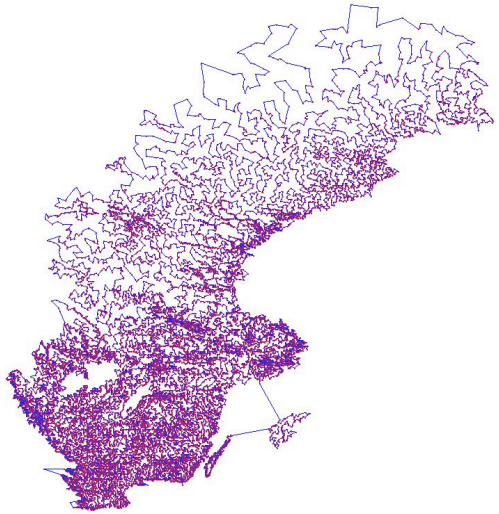


Længde ca. 21.000 km

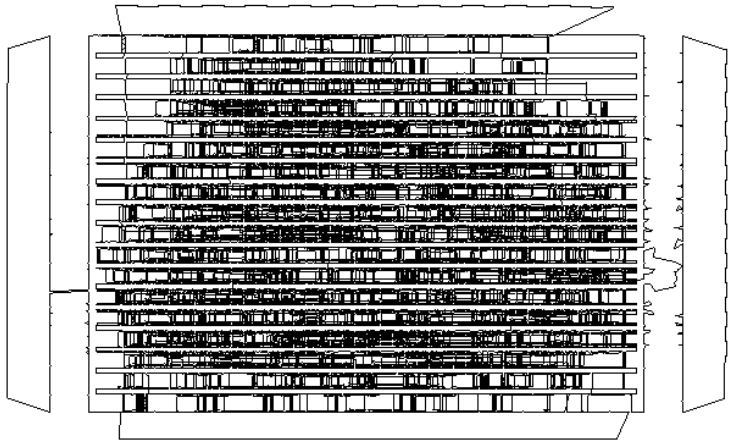
brd14051: optimum

d18512: optimum

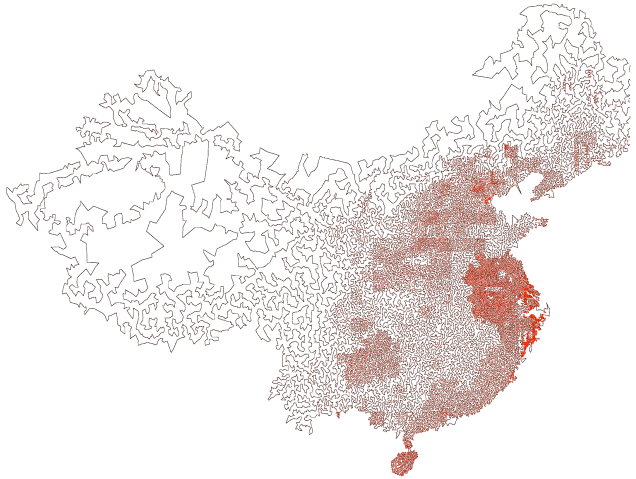
sw24978: optimum



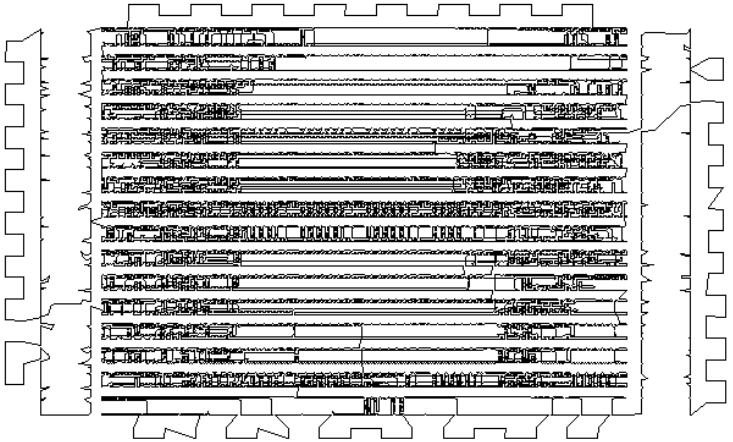
pla33810: optimum



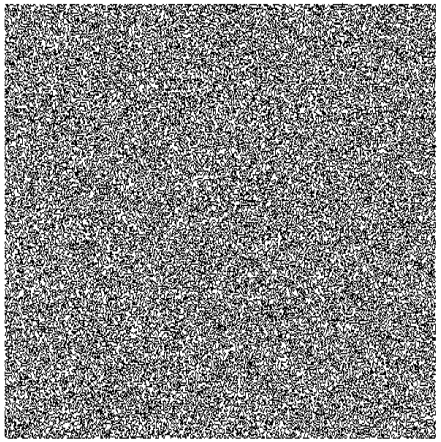
china71009: fejl ≤ 0,024%



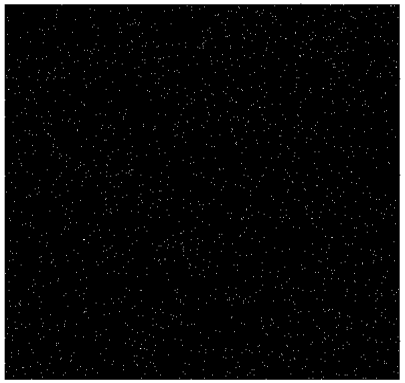
pla85900: optimum



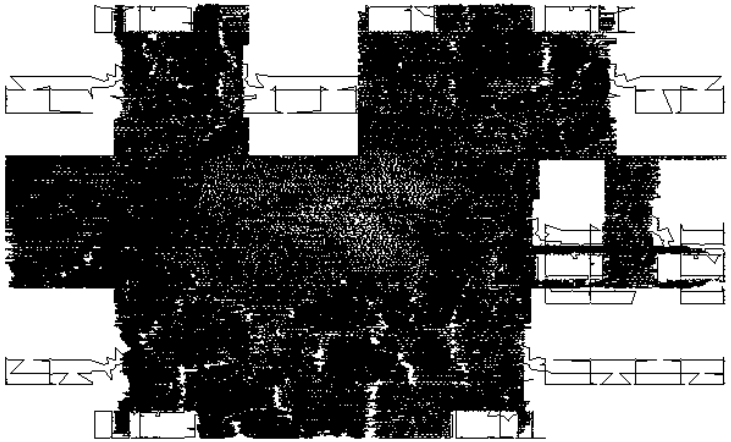
e100k.0: fejl ≤ 0,039%



E1M.0: fejl ≤ 0,019%



lrb744710



World (1.904.711 byer): fejl ≤ 0,049%

Antal ruter ≈ $10^{11.000.000}$

Igangværende arbejde

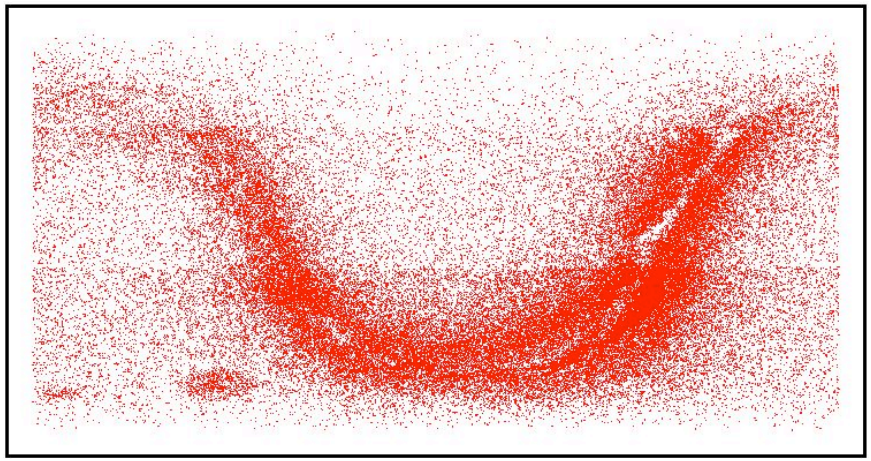
Distribueret løsning af meget store problemer:

- Et VLSI-problem med 9.907.768 "byer"
- Cirka 526 millioner stjerner (≈ $10^{5.000.000.000}$ mulige ruter)

I samarbejde med
David Applegate, William Cook, André Rohe og Sanjeeb Dash

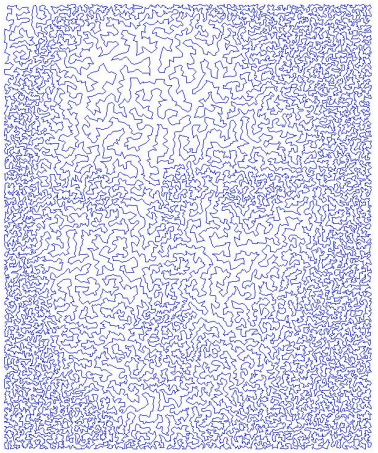
Parallel memetisk (hybrid genetisk) algoritme baseret på LKH

526 millioner stjerner



Den rejsende rummands problem

TSP-kunst



27.486 punkter

En ud af $10^{110.079}$ mulige ture