## Komplexitet

Teoretisk bas för frågorna:

- Är en viss metod bra eller dålig?
- Är ett visst problem lätt eller svårt?

Teori och praktik inte alltid överens, men i stort sett...

**Algoritmkomplexitet:**
Hur många operationer krävs, som funktion av indatas storlek, i värsta fall?
(Problemstorlek $\to \infty$, alla möjliga koefficienter.)

**Definition**

En **polynomisk algoritm** har polynomisk tidkomplexitet som funktion av indatastorleken. (Bra, dvs. snabb)

En **exponentiell algoritm** har exponentiell tidkomplexitet som funktion av indatastorleken. (Dålig, dvs. långsam)



## Algoritmkomplexitet

**Exempel 1:** Dijkstras metod i en graf med $n$ noder:

Varje nod märks permanent *en* gång, alltså behövs $n$ permanenta märkningar.
För varje permanent märkning, behöver man löpa igenom alla $n$ noderna (2 gånger).
Komplexiteten blir $O(n^2)$, och algoritmen är polynomisk.

**Exempel 2:** Genomlöpa alla hörn i en hyperkub i $n$ dimensioner:
$O(2^n)$. Exponentiell algoritm.



## Problemkomplexitet

Vilken tidskomplexitet har den bästa kända metoden för problemet?

Optimeringsversionen: Finn optimala (tillåtna) lösningen.

Igenkänningversionen: Finns det en tillåten lösning som har $c^T x \leq L$?
(Svar: Ja eller nej.)

**Sats**

Om igenkänningsversionen inte är polynomiskt lösbar, så är inte optimeringsversionen det.



## Komplexitet: $P$ och $NP$

**Definition**

$P$ är den klass av igenkänningsproblem som kan lösas (besvaras) av en polynomisk algoritm.

**Definition**

$NP$ är den klass av igenkänningproblem där ett ja-svar kan bekräftas på polynomisk tid.

Om svaret till igenkänningsproblemet är ja, så måste det finnas en lösning, som kan kontrolleras på polynomisk tid.

**Sats**

$P \subseteq NP$.

Är $P = NP$? Njae.

Många har försökt bevisa det, men ingen har lyckats.

## Komplexitet: Polynomiska relationer

**Polynomisk transformation** från A till B: Konstruera ett exempel av B ur ett exempel av A på polynomisk tid, så att exemplet av B ger ja-svar om och endast om exemplet av A ger ja-svar.

### Definition

Ett igenkänningsproblem A $\in$ *NP* är *NP*-**fullständigt** om alla andra problem i *NP* har en polynomisk transformation till A.

Obs: Om man skulle hitta en polynomisk algoritm för ett enda *NP*-fullständigt problem, så har man bevisat att $P = NP$.

### Slutsats

Det finns ingen polynomisk algoritm för något *NP*-fullständigt problem (om inte $P = NP$).



## Komplexitet: *NP*-svårt

### Definition

Ett problem A kallas *NP*-**svårt**, om något *NP*-fullständigt problem kan transformeras polynomiskt till A, men vi ej kan visa att A $\in$ *NP*.

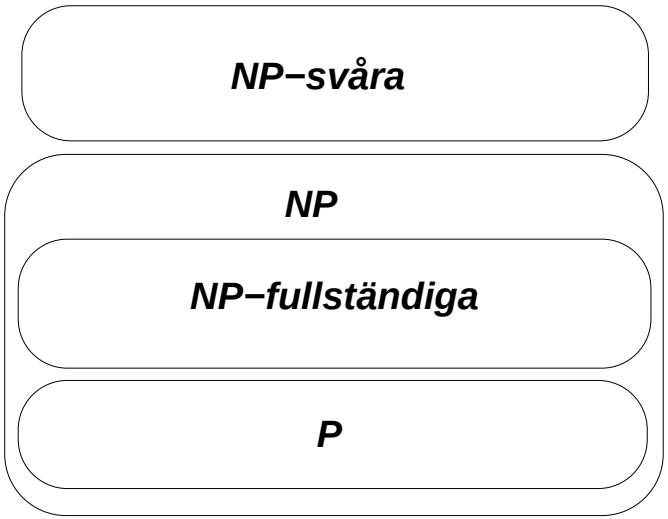
*NP*-svår minst lika svår som *NP*-fullständig.

### Slutsats

Det finns ingen polynomisk algoritm för något *NP*-svårt problem (om inte $P = NP$).



## Komplexitet






## Vilka problem är *NP*-fullständiga/*NP*-svåra?

- Binära heltalsproblemet
- Allmänna heltalsproblemet
- Handelsresandeproblemet
- Kappsäcksproblemet
- Steinerträdsproblemet
- Minkostnadsflödesproblemet med olika varusorter
- Valensbegränsade billigaste uppspännande träd-problemet (VMST)
- Billigaste-enkla-vägproblemet
- Maximal oberoende nodmängd, minimal nodövertäckning, minimal nodfärgning, maxsnitt, maximal klick, etc.
- ... samt ca 3000 till.

## Heuristiker

"Smarta" metoder som ger "skapliga" lösningar.

Ger ej garanterat optimum.

Men kan göra det om man har tur. (Fast det vet man inte.)

Snabbare än optimerande metoder.

Enda möjligheten för riktigt stora svåra problem.

Ofta: *NP*-svårt problem, polynomisk heuristik.



## Heuristiker

Modifieringar av trädsökningsmetoder:

- Kapa grenar då $(\bar{z} - \underline{z})/\bar{z} \leq \varepsilon$.
- Kapa grenar då $\bar{z} - \underline{z} \leq \varepsilon$.
- Avbryt då den första tillåtna lösningen erhålls.

Men metoden fortfarande exponentiell.



## Heuristiker: Giriga metoder

Giriga metoder är snabba, men löser bara vissa enkla problem (matroider) optimalt.
(Kruskal löser MST.)

De kan dock användas som heuristik för många problem.

Kappsäcksproblemet:
Sortera kvoterna $c_j/a_j$. Fyll på med bästa först.
Löser LP-relaxationen exakt.

Samma metod på heltalsproblemet (dvs avrunda neråt), kan ge upp till 100% relativt fel.



## Heuristiker

$\max z = 3x_1 + 4x_2 + 5x_3$ då $2x_1 + 3x_2 + 4x_3 \leq 6$, $x_j \in \{0,1\}$ för alla $j$

Kvoter: $x_1 : 3/2$, $x_2 : 4/3$, $x_3 : 5/4$
ger lösningen $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, och $z = 7$.
Optimum: $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $z = 8$.

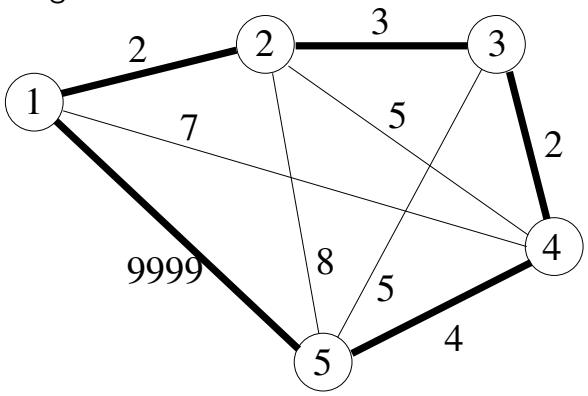
$\max z = 3x_1 + 100x_2$ då $2x_1 + 99x_2 \leq 100$, $x_j \in \{0,1\}$ för alla $j$

Kvoter: $x_1 : 3/2$, $x_2 : 100/99$ ger lösningen $x_1 = 1$, $x_2 = 0$ och $z = 3$.
Optimum: $x_1 = 0$, $x_2 = 1$, $z = 100$.

## Heuristiker för handelsresandeproblemet

**Närmaste granne-algoritmen:**

Partiell handelsresandetur, utöka med en nod i taget.
Lägg till närmaste grannen till sista noden i turen.




Kan fallera helt för icke fullständig graf.



## Specialutvecklade (ad hoc) algoritmer

Utnyttja problemets struktur.

**Trädalgoritmen för handelsresandeproblemet ΔTSP:**

1. Finn billigaste uppspännande träd.
2. Konstruera en graf med parallella bågar genom att dubblera varje båge i trädet.
3. Finn en Eulercykel.
4. Transformera Eulercykeln till en Hamiltoncykel (genom att ta genvägar vid återbesök).

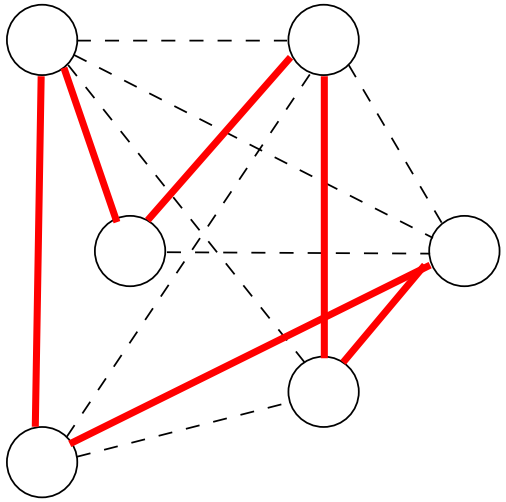
Komplexitet: $O(n^2)$ (Prim, steg 1).

Målfunktionsvärdet: $z_{MST} \leq z^*$, $z_E = 2z_{MST}$, $z_H \leq z_E \Rightarrow z_H \leq 2z^*$.

Sämre för TSP utan Δ.



## Trädalgoritmen för handelsresandeproblemet





## Heuristiker för handelsresandeproblemet

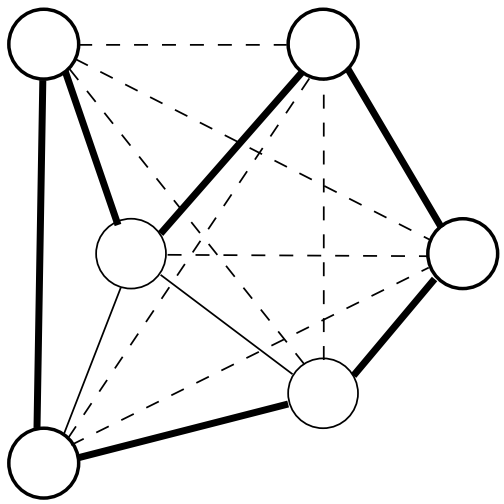
Christofides algoritm: Addera bara bågar till noder med udda valens.

1. Finn billigaste uppspännande träd $T$.
2. Finn alla noder som har udda valens i $T$ och finn den billigaste perfekta matchningen $M$ i den fullständiga grafen med dessa noder.
3. Finn en Eulercykel i $T + M$ och den handelsresandetur den innehåller.

Steg 2: Matchningsproblem, $O(n^4)$.

$z_H \leq 3/2z^*$ för ΔTSP.

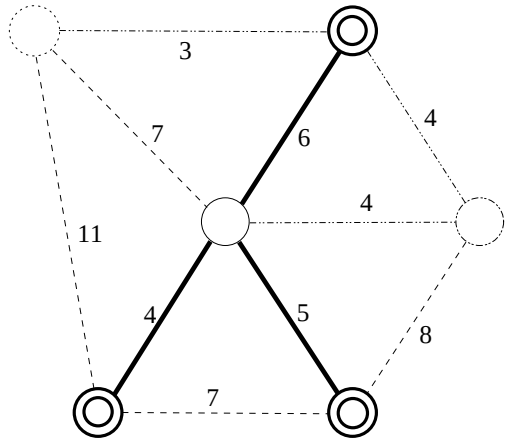
## Heuristiker för handelsresandeproblemet





## Heuristiker för Steinerträdsproblemet

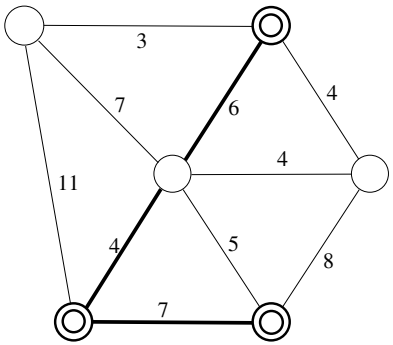
1. Finn billigaste uppspännande träd.
2. Ta bort alla noder $i \notin N'$ (dvs. som inte måste besökas) som har valens *ett* (samt anslutande båge).
3. För varje nod $i \notin N'$ som har valens *två*, kontrollera om det är billigare att använda direktbågen istället för att besöka nod $i$.






## Heuristiker för Steinerträdsproblemet

1. Kalla en nod i $N'$ för $S$.
2. Finn den nod i $N' \setminus S$ som ligger närmast $S$.
3. Lägg till den funna billigaste vägen till lösningen, samt de noder som ingår i vägen till $S$, och upprepa detta tills alla noder i $N'$ är med i lösningen.






## Heuristiker för hinkpackningsproblemet

**Första plats** ("First fit"): Placera nästa objekt i den hink längst till vänster där den får plats.

**Sista plats** ("Last fit"): Placera nästa objekt i den hink längst till höger där den får plats.

**Nästa plats** ("Next fit"): Placera nästa objekt i hinken längst till höger (även om en ny krävs).

**Bästa plats** ("Best fit") Placera nästa objekt i den fullaste hink där den får plats.

**Värsta plats** ('Worst fit") Placera nästa objekt i den tommaste användna hinken.

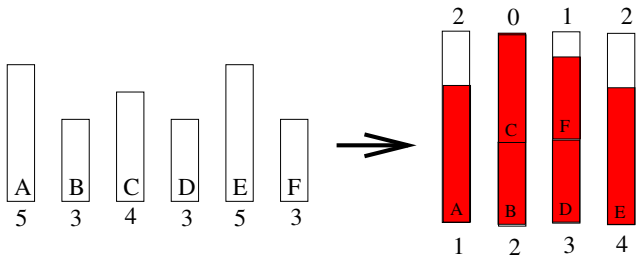
**Näst värsta plats** ('Almost worst fit") Placera nästa objekt i den näst tommaste hinken.

Förbättring: Sortera först objekten, så att de tyngsta kommer först.

Första/bästa plats: $z_H \leq 17/10\ z^* + 2$.
Sorterad första/bästa plats: $z_H \leq 11/9\ z^* + 4$.

# Heuristiker för hinkpackningsproblemet

### Första plats




Undre gräns. $\lceil \frac{23}{7} \rceil = 4$. Optimum.

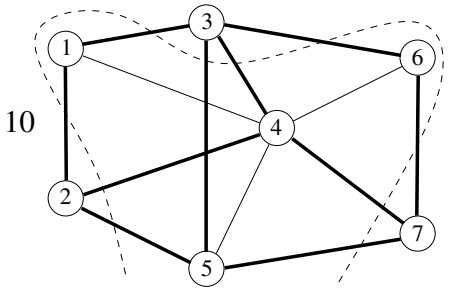


# Heuristiker för maxsnittsproblemet

Maxsnittsproblemet: Finn ett snitt genom en oriktad graf så att antalet bågar som passerar snittet är maximalt.

1. Starta med valfritt snitt.
2. Flytta över en nod till andra sidan snittet om detta förbättrar lösningen (dvs. ökar antalet bågar över snittet).
3. Upprepa tills ingen flyttning av en enstaka nod ger någon förbättring.

$z^* \leq 2z_H$.






# Heuristiker för andra grafproblem

Ofta ganska självklara:

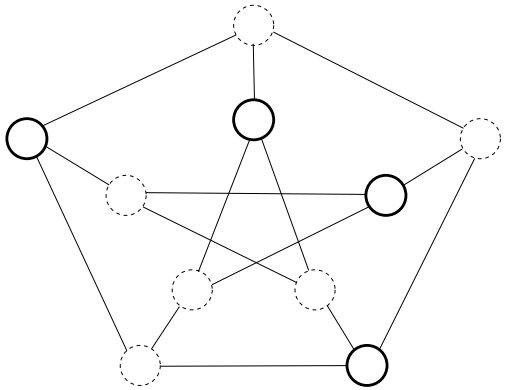
Ta noderna/bågarna i nummerordning. Bygg upp lösningen girigt.

- Maximal oberoende nodmängd: Ta med nästa nod om det går. Märk anslutande noder (får ej tas med).
- Nodfärgning med min antal färger: Färga nästa nod med lägsta möjliga färg.
- Bågfärgning med min antal färger: Färga nästa båge med lägsta möjliga färg.
- Minimal nodövertäckning: Ta med noden med maximal valens. Ta bort alla anslutande bågar.
- Minimal nodövertäckning: Finn maximal matchning, ta med alla ändnoder till bågarna i matchningen.
- Minimal bågövertäckning: Finn maximal matchning, fyll på med bågar till omatchade noder.



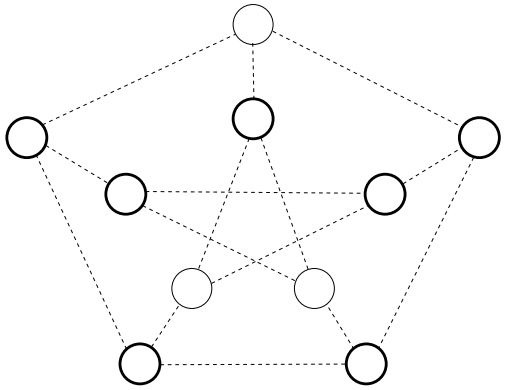
# Heuristik för maximal oberoende nodmängd

Ta med nästa nod om det går. Märk anslutande noder (får ej tas med).



## Heuristik för minimal nodövertäckning

Ta med noden med maximal valens. Ta bort alla anslutande bågar.





## Lokalsökning

Hoppa från en punkt till en bättre granne. Upprepa.

Definiera **omgivning** $N$ (närliggande punkter, grannar), mha avståndsmått samt avståndsgräns.

Exempel:
- Euklidiskt närmaste punkterna.
- Utbyte av element, t ex bågar, noder.
- Byte av 0 mot 1 och vv.

Genomsök omgivningen efter en bättre punkt.

0. Finn startpunkt $x^{(k)} \in X$. Sätt $k = 0$.
1. Genomsök $N(x^{(k)})$ efter en punkt $\bar{x}$ med $f(\bar{x}) < f(x^{(k)})$.
2. Om ingen finns: Stopp, $x^{(k)}$ är lokalt optimum.
3. Annars sätt $x^{(k+1)} = \bar{x}$. Sätt $k = k + 1$ och gå till 1.



## Lokalsökning

Varianter för bivillkor:

- Endast tillåtna punkter.
- Även otillåtna punkter. Lägg till straffunktion till målfunktionen.

Klättringsstrategier:

- Finn bästa punkten i hela omgivningen.
- Finn bästa punkten i delmängd av omgivningen.
- Finn första bättre punkten.
- Finn första bättre punkten i delmängd av omgivningen.



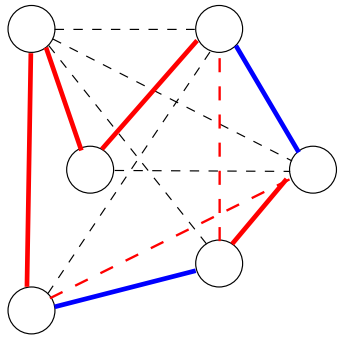
## Lokalsökning

Omgivning för handelsresandeproblemet:

2-byte: Ta bort två bågar och sätt dit två nya.

3-byte: Ta bort tre bågar och sätt dit tre nya.

Def: Lokalt optimum map viss omgivning. (T ex 2-bytes-optimum.)



## Utvidgningar av lokalsökning

**Itererad lokalsökning:** Gör många sökningar med olika startpunkter.

Hur välja startpunkter? Gammalt: Slumpmässigt.

Nyare: Baserat på tidigare optimum. Försök få annat opt.

**Variabel omgivning (Variable Neighborhood Search, VNS):**

Byt omgivning då och då.

Ett lokalt optimum i en omgivning är kanske inte ett lokalt optimum i en annan omgivning, men ett globalt optimum är lokalt optimum i alla omgivningar.

**Stora omgivningar (Very Large Scale Neighborhood Search, VLSN):**

Eftersom vi byter omgivning, kan man använda större omgivningar.



## Utvidgningar av lokalsökning

**"Förstör - reparera" (LNS, Large Neighborhood Search):**

Förstör en lösning genom att ta bort en viss del av den, och reparera den genom att sätta dit andra delar.

Ta bort delar slumpmässigt, och reparera med en girig heuristik.

**Girig slumpmässig adaptiv sökning (GRASP, Greedy Randomized Adaptive Search Procedure):**

Giriga slumpmässiga lösningar förbättras med lokalsökning.

Lösningar som är bra placeras i en kortare lista, ur vilken punkter slumpmässigt väljs.



## Tabusökning

Utvidgning av lokalsökning.

För att ej fastna i lokalt optimum: Tillåt temporär försämring.

Förbjud förflyttningar till nyss besökt grannar. Spara tidigare förflyttningar (besökta punkter) i en tabulista, av viss längd (kö).

1. Välj startpunkt, $x^{(0)} \in X$. Sätt $k = 0$. Börja med en tom tabulista.
2. Välj en sökmängd $A(x^{(k)}) \subseteq N(x^{(k)})$ av punkter som ej är tabu.
3. Finn nästa punkt, $x^{(k+1)}$, mha $\min_{x \in A(x^{(k)})} f(x)$.
4. Uppdatera tabulistan: Tillför $x^{(k)}$. Om listan är full, ta bort äldsta punkten.
5. Stoppa om $k > K$. Annars sätt $k = k + 1$ och gå till 2.



## Simulerad kylning

(Härma fysikaliskt förlopp: Långsam kylning av metall.)

Acceptera försämring med viss sannolikhet, $e^{-\Delta/T}$, där $T$ är "temperaturen" som långsamt minskas.

1. Välj en startpunkt, $x^{(0)} \in X$. Sätt $k = 0$.
Välj en starttemperatur, $T$, och en kylfaktor, $r : 0 < r < 1$.
2. Sätt $l = 0$.
   3. Välj en slumpmässig granne, $\bar{x} \in N(x^{(k)})$.
   4. Beräkna ändringen $\Delta = f(\bar{x}) - f(x^{(k)})$.
   5. Om $\Delta \leq 0$ (bättre), sätt $x^{(k+1)} = \bar{x}$. Gå till 5.
   6. Om $\Delta > 0$ (sämre), sätt $x^{(k+1)} = \bar{x}$ med sannolikheten $e^{-\Delta/T}$.
   7. Sätt $l = l + 1$. Om $l \leq L$, gå till 3.
8. Reducera temperaturen: Sätt $T = rT$.
9. Stoppa om systemet är "fruset". Annars sätt $k = k + 1$, $l = 0$, gå till 2.

## Myroptimering

Myror vandrar omkring slumpmässigt, tills de hittar mat.

När de återvänder till myrstacken, lämnar de ett doftspår på marken.

När andra myror stöter på detta doftspår, följer de spåret (för att öka sannolikheten att finna mat).

Fler och fler myror följer spåret och doftspåret blir starkare, så länge det finns mat där.

När det inte längre finns mat där, kommer doftspåret att försvinna.

Framgångsrika vägar kommer att användas mer och mer, medan andra vägar blir mer oanvända.

**Ant Colony Optimization**: Gör så för att hitta optimala vägar i stora nätverk.



## Genetiska algoritmer

(Härma biologiskt förlopp: Evolution av en population.)

En population av lösningar uppdateras genom vissa operationer, så att bra lösningar har större chans att överleva. Nya punkter bildas genom "crossover" (kombination av två "föräldrar" ger två "barn"), samt genom "immigration"/"mutation" (helt nya införs). De bästa ("eliten") från förra populationen sparas.

Följande steg genomgås:

**Evaluering:** Beräkning av kvaliteten av varje individ (målfunktionsvärde).

**Föräldraval:** Vissa par av lösningar (föräldrar) väljes, baseras på deras kvalitet.

**Crossover:** Varje föräldrapar kombineras till en eller två nya lösningar (barn).

**Mutation:** Några av barnen förändras slumpmässigt.

**Populationsurval:** Val av en ny population (baserat på kvalitet) genom att vissa barn ersätter lika många av den gamla populationen.



## Partikelsvärmoptimering

**Particle Swarm Optimization, PSO:**

En "svärm" av lösningar ("partiklar") uppdateras med enkla matematiska formler baserade på position och hastighet.

Jämför med fågelsvärmar som rör sig i skyn, som antas ha någon sorts "svärmintelligens" som leder dem rätt.

Rörelserna baseras på en partikels bästa position men också på hela svärmens bästa position.

När nya bästa positioner uppnås, påverkas svärmens rörelser.

Metoden prestanda beror på en balans mellan "exploration" och "exploitation", dvs. mellan att leta i nya områden och att intensifiera sökningen i ett lovande område.



## Sökmetod för ickedifferentierbar funktion

**Nelder-Meads metod**

1. Starta med $n+1$ (linjärt oberoende) punkter (en simplex).
2. Finn den sämsta punkten, $\tilde{x}$.
3. Finn mittpunkten av de $n$ bästa, $\bar{x}$.
4. Beräkna riktningen: $\Delta x = \bar{x} - \tilde{x}$.
5. Ersätt den sämsta punkten med $x = \bar{x} + \Delta x$. (Vippa simplexen.)
6. Gå till 2.

Extrafinesser (ändra form på simplexen):

- Om nya $x$ är bättre än bästa, gå dubbelt så långt, $x = \bar{x} + 2\Delta x$.
- Om nya $x$ är inte bättre än näst sämsta, gå hälften så långt, $x = \bar{x} + 0.5\Delta x$.
- Om nya $x$ är inte bättre än sämsta, gå hälften så långt bakåt, $x = \bar{x} - 0.5\Delta x$.
- Om inget ger en bättre punkt, flytta alla punkter närmare den bästa.

# Grön optimering

- Laddbar hybrid-elbil
- Utbyte av gamla vindkraftverk
- Utbyte av delar i vindkraftverk
- Ressträckor för handelsresande
- Design av försörjningskedja
- Placering av en oönskad anläggning
- Minimering av bränsleåtgång för tunga lastbilar
- Vintervägar
- Fordonsruttning för att minimera utsläpp
- Intelligent trafikflöde
- Habitatnätverksgenomsläpplighet
- Dimensionering av bussflotta
- Framtidens gruvor
- Åkeriets optimeringsproblem
- Optimering i lastbilsfarthållare
- Ruttplanering för hemtjänsten