6. Local search heuristics for TSP

- **Local search for TSP**
- **NN/Greedy/Christophides**
- **2OPT**
- **Lin-Kernighan**

```
LocalSearch(ProblemInstance x)
y := feasible solution to x;
while ∃ z ∈N(y): v(z)<v(y) do
    y := z;
od;
return y;
```

Heuristik: Vi har fundet på en algoritme, som ikke kan bevises teoretisk, således må vi teste den på rigtig data. Teorier, der ikke kan bevises. En slags guideline til at gøre noget.

Effektive approksimationsalgoritmer med gode worst case ratio'er findes typisk ikke for et NPC-problem. Vi kan med **local search** forbedre en fundet løsning, ved at søge i blandt kandidatløsninger i et "search space". LocalSearch kan skrives op således:

For en LocalSearch-algoritme skal vi overveje, hvilken løsning vi **starter** med, hvordan vi vælger den næste løsning **(naboer)** og hvornår vi stopper. I vores tilfælde stopper vi, når vi er trætte.

TSP: Givet N byer, så find den korteste tur imellem disse byer, således turen er minimeret, og hver by kun besøges én gang. Dette er et NPC-problem.

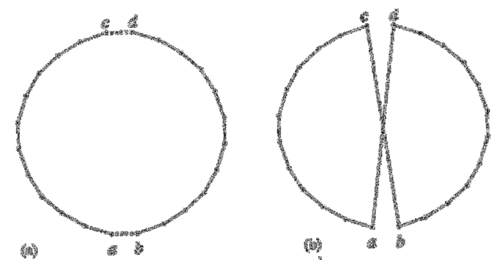
Start-turen kan findes med bl.a. en af følgende tre algoritmer:

Nearest Neighbor: En tilfældig by vælges, og så laver man en sti til den by, der er tættest på (og som endnu ikke er besøgt).
Greedy: Antager der er kanter imellem alle byer. Vælger først den korteste kant. Vælger herefter den næstkorteste, således at der ikke skabes en cykel, der er mindre end N byer og der ikke er en knude, der får en grad højere end 2.
Christophides: 1. Lav MST T 2. Kig på knuder med ulige grad, og lav en minmum perfect matching M i mellem disse. 3. Lav en ny graf, hvor kanterne fra M sættes ind i T. 4. Lav en Euler-tur fra roden. 5. Lav Euler-turen om til en Hamilton-tur ved at fjerne duplikater.

Held-Karp lower bound: LP-relaxation af TSP. Kører 2/3 af optimal.

Sammenligning af initielle tur: Ud fra empiriske data, kan vi se, at generelt laver Christophides den bedste tur, og Nearest Neighbor laver den dårligst. Derimellem ligger Greedy.

2-OPT: Man finder t_1 og t_2. Herefter finder man et punkt t_3 (og så er t_4 givet) således at $d(t_1, t_2) > d(t_2, t_3)$ eller $d(t_3, t_4) > d(t_4, t_1)$. Størrelsen af nabolaget $O(n^k)$.



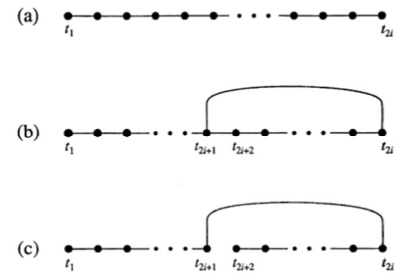
Optimeringer:

- Neighbor List: Liste med fx 20 nærmeste byer til hver by. Man kigger dermed efter t_3 som ligger før t_1 på nabo-listen for t_2.
- Don't look bits: Hvis der for en by b, ikke findes en optimering når $t_1 = b$, kigges der ikke på b igen, før en af de to nærmeste byer har ændret sig.

Ovenstående optimeringer gør muligvis turen lidt dårligere, men det vinder vi på tiden.

Tabu: Tillad "uphill" moves, for at komme ud af lokale optimale løsninger (ellers risikerer vi, at vi står og veksler mellem de to samme løsninger), så nærliggende løsninger kan findes. Undgå at vende tilbage til samme optimale løsning, ved at opsætte regler. Lav en liste med allerede sete løsninger, der forhindrer at de gentages.

Lin-Kernighan kombinerer nogle af de ting vi har gennemgået her. Generelt er det som Tabu, hvor vi sørger for, der bliver lavet uphill-moves. Betydeligt bedre resultater end 3-OPT, endda kun lidt langsommere.



LK-search (inner loop): Fixed t_1. For hvert $i = \{2, \dots, n\}$, lav et **LK-Move.**

LK-Move: Path p_i: Stien fra t_1 og t_{2i}. Kan ses som tour ved at tilføje en ny kant (t_1, t_{2i}). t_{2i} har en naboliste. Alle naboer, der ligger mellem t_{2i} og t_1 på p_i, hvor $d(t_1, t_{2i}) > d(t_{2i}, t_{i2+1})$ betragtes.

→ Ny kant (t_{2i}, t_{2i+1}) indsættes. Den nye path skal være et one-tree (ST + 1 kant), og være kortere end $P_i + (t_1, t_{2i})$.

→ Kanten (t_{2i+1}, t_{2i+2}) fjernes. Derved opnås ny path P_{i+1}.

→ Slettede og tilføjede kanter indsættes i en tabu-liste, som ingen grænse har.

→ Turen gemmes, hvis den er bedre end den tidligere sete i denne LK-search.

→ Vælg det næste kvalificerede nontabu move, som gør turen P_{i+1} bedst. Da P_{i+1} ikke nødvendigvis er kortere end P_i tillader dette ”uphill moves”.

LK-Search terminerer når der ikke er flere kvalificerede (forbedrende) nontabu-moves (sikrer maks. N moves).

Algoritmen:

1. Champion-tour: initielle tour.
2. Udvælg t_1 til t_5 og kør en 3-OPT på champion tour med disse valg.
   a. Hvis der ikke kan findes passende punkter, finder vi 3 punkter mere, og kører en 4-OPT
   b. Tabu-listen for LK-search fyldes med kanter fra 3- eller 4-OPT-movet.
3. Lav LK-search på denne tour.
   a. Hvis der findes en tour der er bedre end champion, færdigudføres det nuværende LK-search, og den nye tour er nu champion.
   b. Hvis der er et 2-OPT-move, som forbedrer turen for de valgte t_1 til t_4, (og LK-searchet ikke gjorde), så vælg denne tur som champion.
4. Gå til 2.

Terminerer når alle mulige valg for t_1 til t_5 (t_8)er prøvet af uden at få en bedre tour: Vi har fundet et lokalt optimum.

Iterated Lin-Kernighan: I stedet for at terminere når algoritmen ikke kan finde flere forbedrende ture, laver vi et 4-OPT double-bridge-move, og kører så algoritmen på denne tur. Vi stopper når vi ikke gider mere.