DM811

Heuristics for Combinatorial Optimization

Lecture 6

TSP

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. TSP

2. Code Speed Up

Outline

Construction Heuristics

Construction heuristics specific for TSP

- Heuristics that Grow Fragments
  - Nearest neighborhood heuristics
  - Double-Ended Nearest Neighbor heuristic
  - Multiple Fragment heuristic (aka, greedy heuristic)
- Heuristics that Grow Tours
  - Nearest Addition
  - Farthest Addition
  - Random Addition
  - Clarke-Wright savings heuristic
  - Nearest Insertion
  - Farthest Insertion
  - Random Insertion
- Heuristics based on Trees
  - Minimum spanning tree heuristic
  - Christofides' heuristics
  - Fast recursive partitioning heuristic

Construction Heuristics for TSP

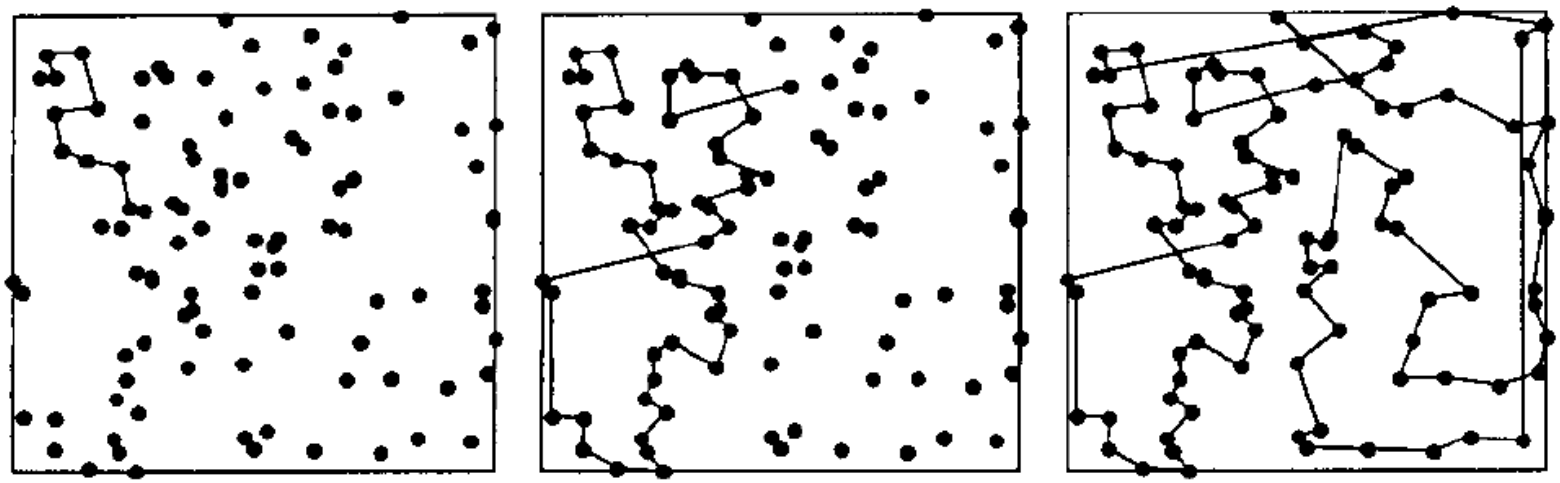
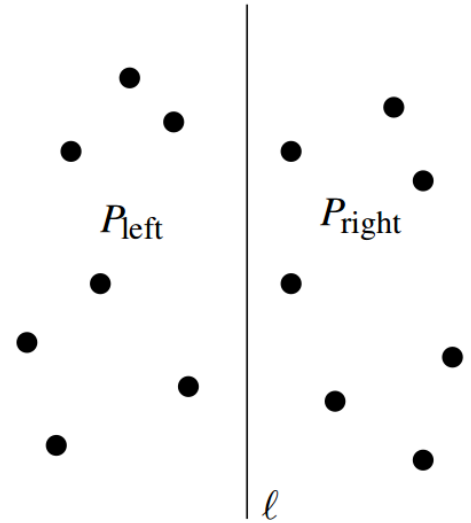


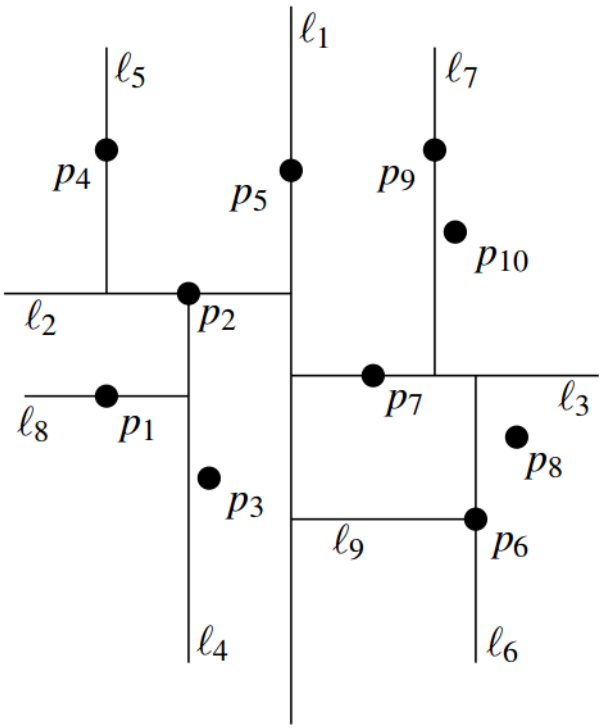
Figure 1. The Nearest Neighbor heuristic.

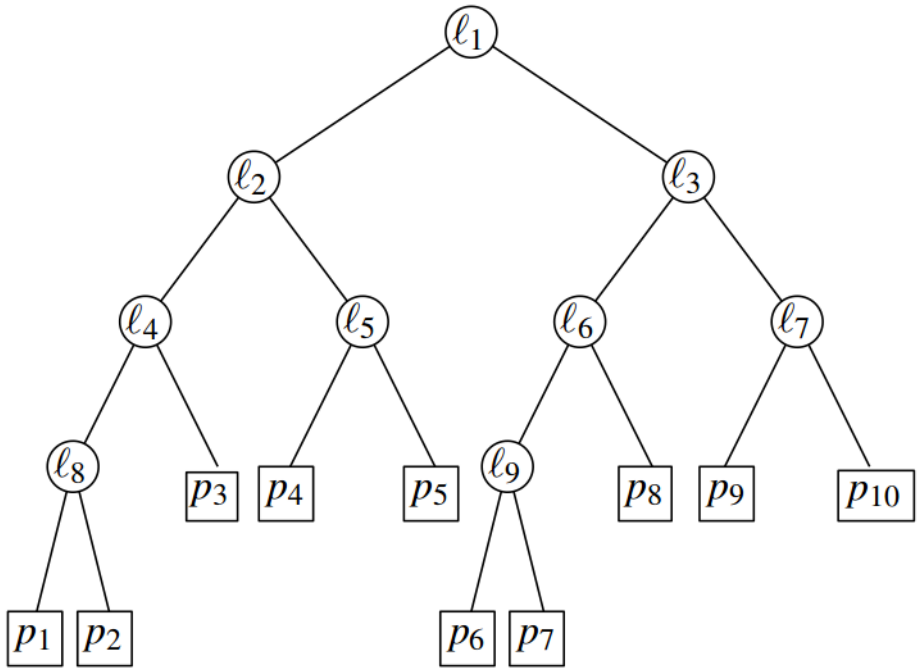
- In geometric instances: $NN < \frac{(\lceil \log N \rceil + 1)}{2} \cdot OPT$
- Double-Ended NN

```
Build(PtSet)
Perm[1]:=StartPt
DeletePt(Perm[1])
for i:=2 to N do
    Perm[i]:=NN(Perm[i-1])
    DeletePt(Perm[i])
```

Data Structures







- Construction in $O(n \log n)$ time and $O(n)$ space
- Range search: reports the leaves from a split node.
- `Delete(PointNum)` amortized constant time
- `NearestNeighbor(PointNum)` bottom-up search
  visit nodes + compute distances
  $A + BN^C$, $A > 0, B < 0, -1 < C < 0$ (expected constant time) if no deletions happened and data uniform
- `FixedRadiusNearestNeighbor(PointNum, Radius, function)`
- `BallSearch(PointNum, function)` ball centered at point
- `SetRadius(PointNum, float Radius)`
- `SphereOfInfluence(PointNum, float Radius)` ball centered at point with given radius

Construction Heuristics for TSP

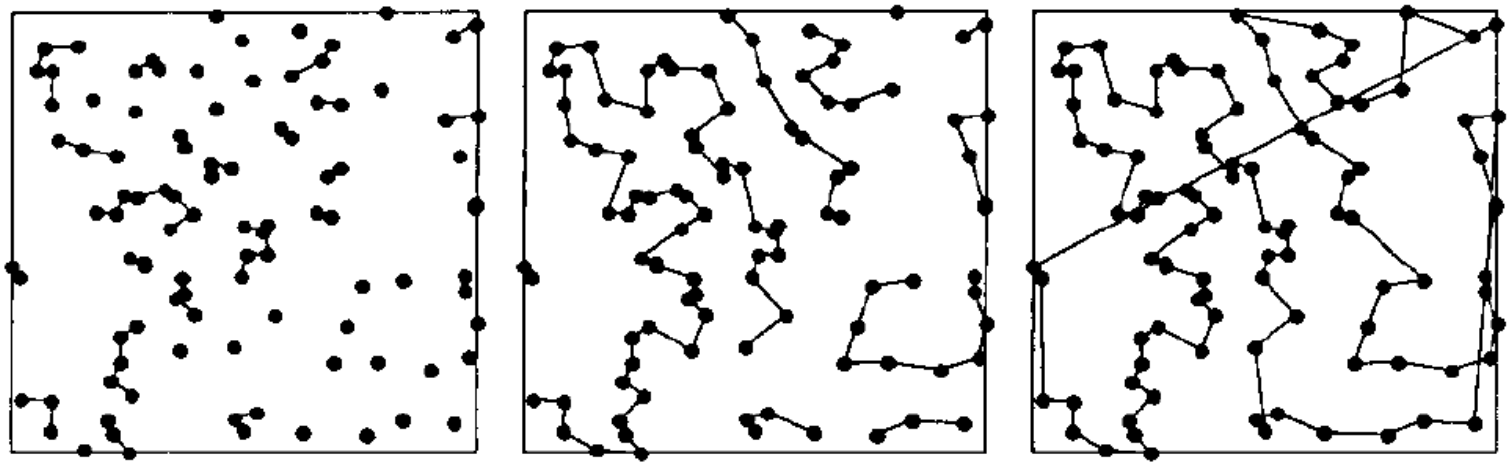


Figure 5. The Multiple Fragment heuristic.

- $O(\sqrt{N})$ approximation

- Array `Degree` num. of tour edges
- K-d tree for nearest neighbor searching (only eligible nodes)
- Array `NNLink` containing index to nearest neighbor of i not in the fragment of i
- Priority queue (heap) with nearest neighbor links
- Array `Tail` link to the other tail of current fragments.

Important Elements

- Exploit the locality inherent in the problem to solve it (NN search, Fixed-radius search, ball search)
- Search time modelled by a function $A + BN^C$
- Number of searches
- Priority queue of links to nearest neighbors

Addition Heuristics

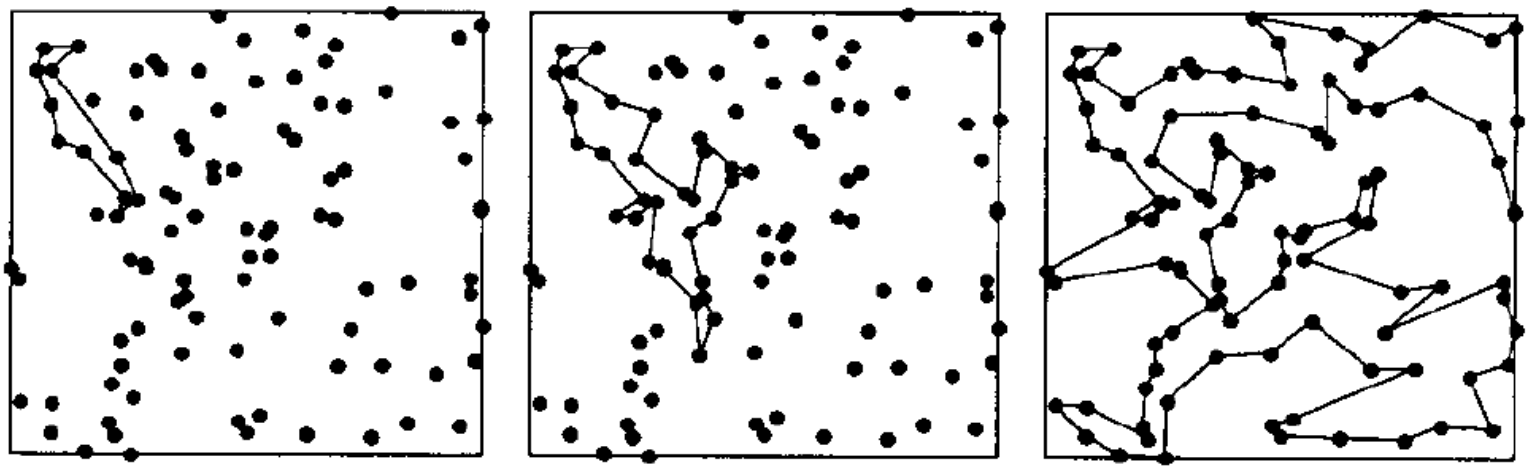


Figure 8. The Nearest Addition heuristic.

Tour maintained as a doubly-linked list

Addition Heuristics

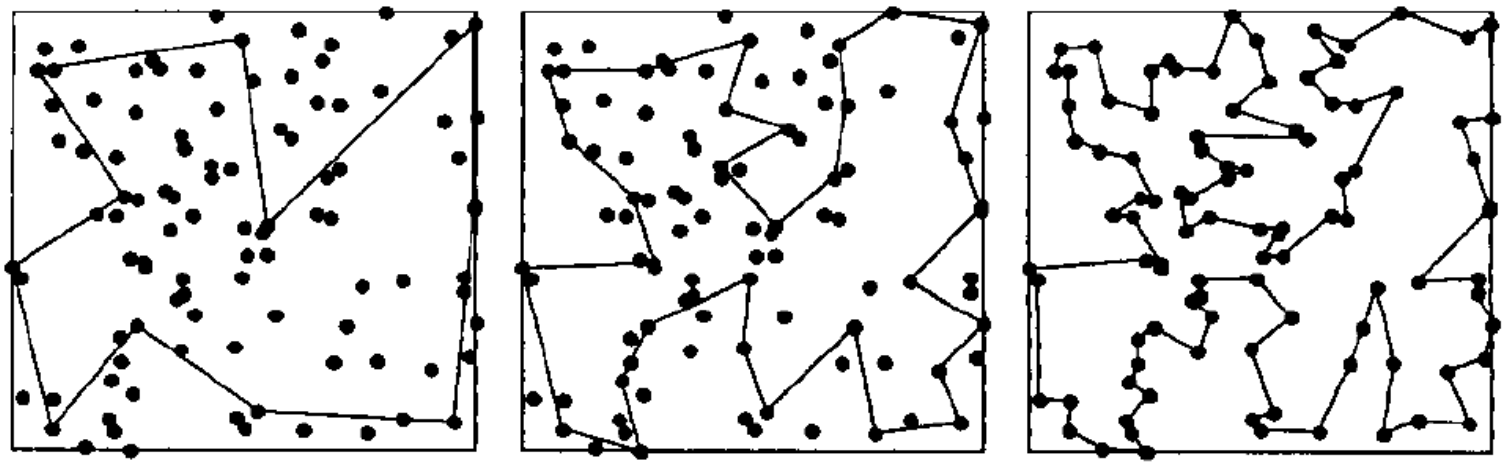


Figure 11. The Farthest Addition heuristic.

Addition Heuristics

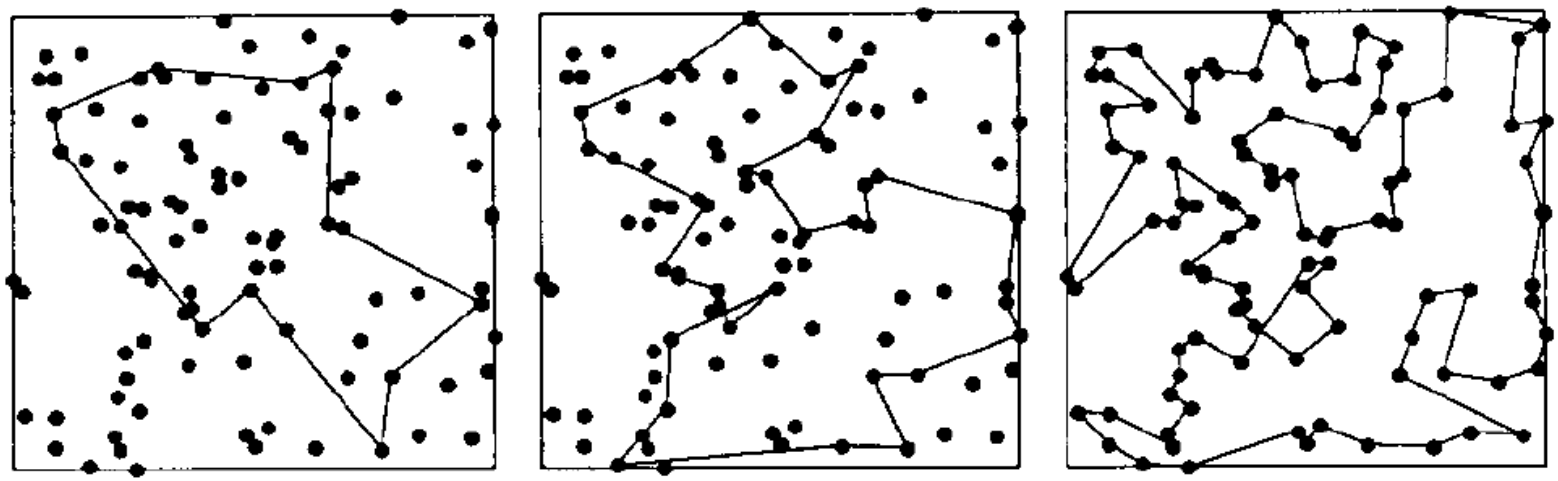
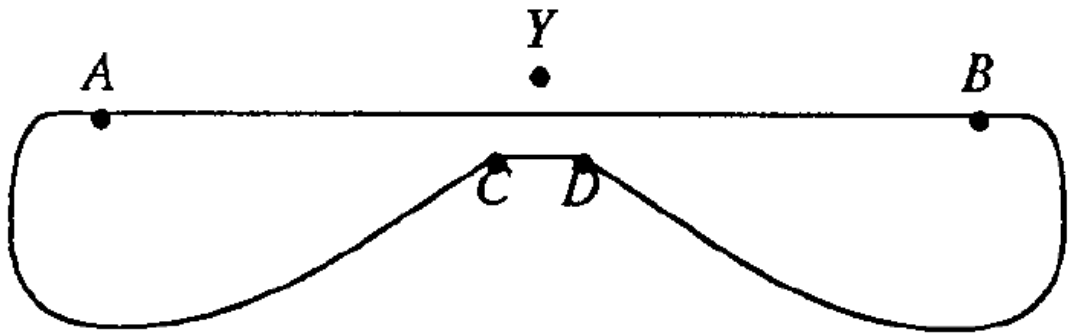


Figure 14. The Random Addition heuristic.

Insertion Heuristics

Motivation:

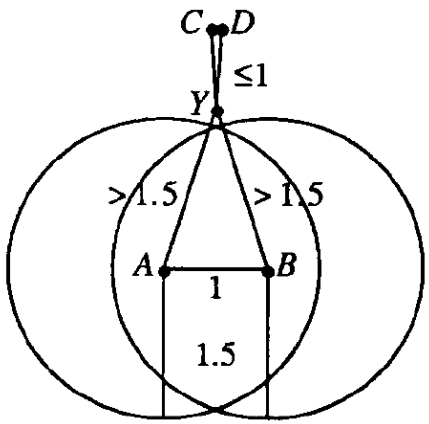


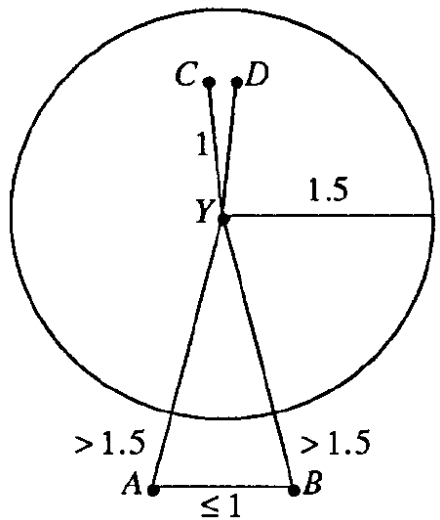
Theorem

Y not yet in tour
C nearest neighbor of Y
D neighbor of C in tour that minimize $C(Y, CD)$
The tour edges with minimal expansion is:

- *the nearest neigbhor edge CD*
- *the edge AB such that A is in $\texttt{NNBall}(Y, 1.5 \cdot e_{\min})$, $e_{\min}$ shorest edge from Y*
- *the edge AB such that Y is in $\texttt{SphereOfInfluence}(A, 1.5 \cdot e_{\max})$, $e_{\max}$ longest edge from A scale 1.5*

Proof: $C(Y, CD) \leq 2D(Y, C)$





Construction Heuristics for TSP

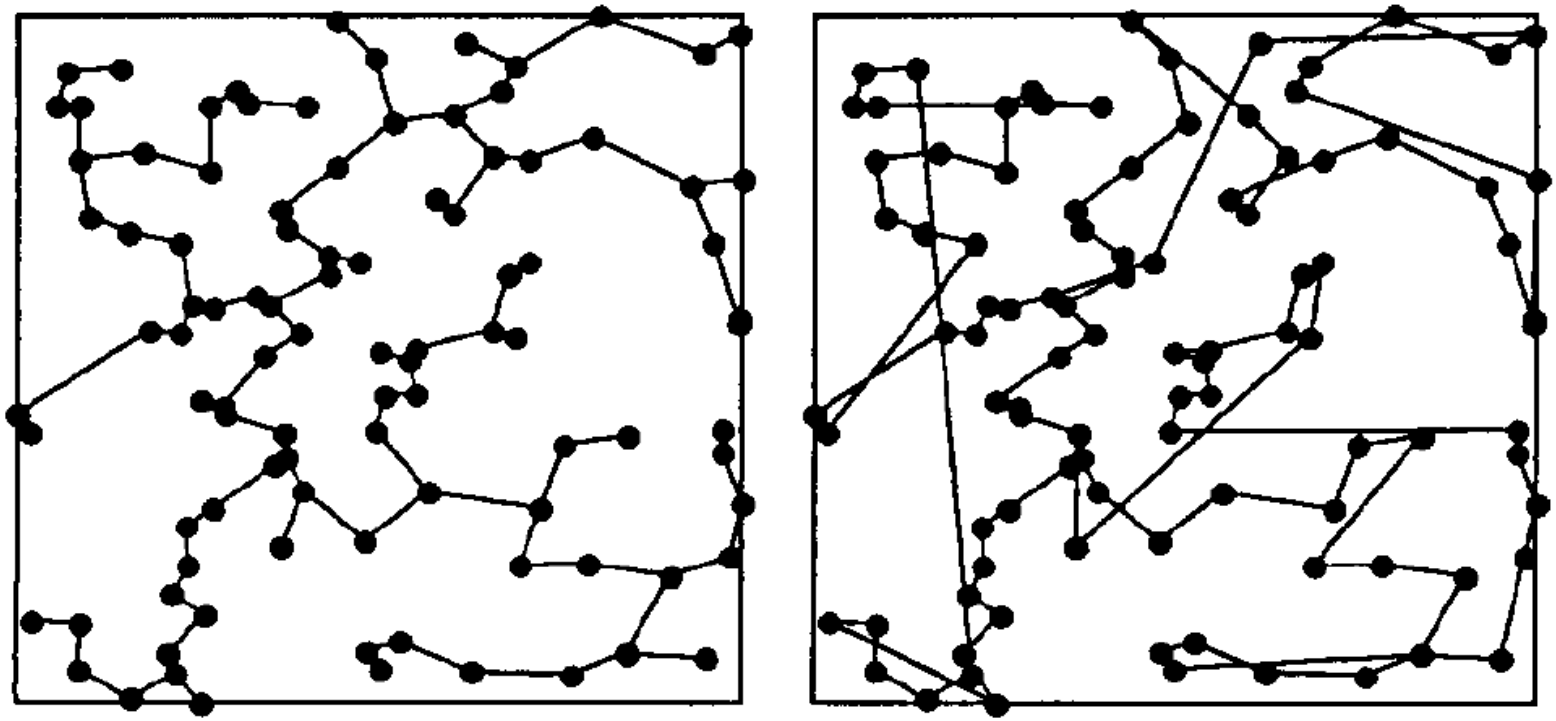


Figure 18. The Minimum Spanning Tree heuristic.

$$MST \leq 2 \cdot OPT$$

Construction Heuristics for TSP

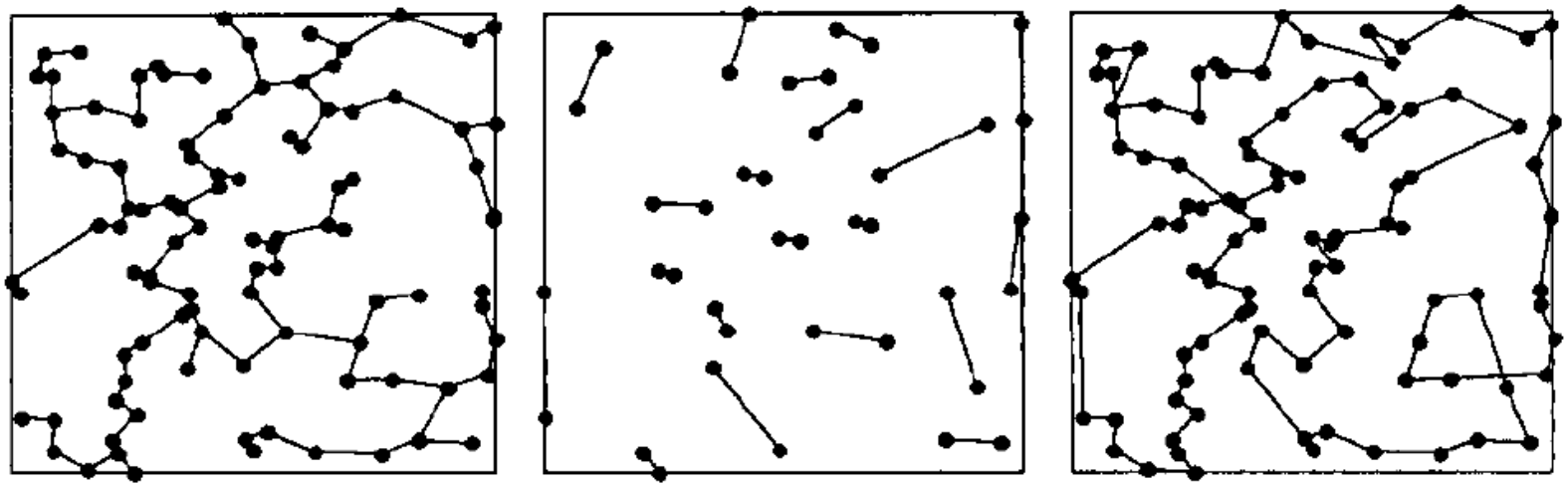


Figure 19. Christofides' heuristic.

$CH \leq \frac{3}{2} \cdot OPT$ tight and best known

Complete Algorithms and Lower Bounds

Reference Results

- Branch & cut algorithms (Concorde: http://www.tsp.gatech.edu/)
  - cutting planes + branching
  - use LP-relaxation for lower bounding schemes
  - effective heuristics for upper bounds

Solution times with Concorde

Instance	No. nodes	CPU time (secs)
att532	7	109.52
rat783	1	37.88
pcb1173	19	468.27
fl1577	7	6705.04
d2105	169	11179253.91
pr2392	1	116.86
rl5934	205	588936.85
usa13509	9539	ca. 4 years
d15112	164569	ca. 22 years
s24978	167263	84.8 CPU years

- Lower bounds: (within less than one percent of optimum for random Euclidean, up to two percent for TSPLIB instances)

Outline

Where do speedups come from?

Where can maximum speedup be achieved?
How much speedup should you expect?

Code Tuning

- Caution: proceed carefully! Let the optimizing compiler do its work!
  - optimizing flags (C++ `-O3`, java http://java.sun.com/developer/onlineTraining/Programming/JDCBook/perfTech.html)
  - just-in-time-compilation: it converts code at runtime prior to executing it natively, for example bytecode into native machine code. (in java done by deafult – to disable `-Djava.compiler=NONE` – in C++ possible via `llvm-g++` http://vmakarov.fedorapeople.org/spec/2011/llvmgcc64.html)
- Cache aware (`-m32` vs `-m64`)
- Profiling (java: `java -Xrunhprof:cpu=times prog` information on the time spent in each method of the program written to `java.hprof.txt`. C++: `gprof`, instruments, http://visualvm.java.net/)

- Expression Rules: Recode for smaller instruction counts.
- Loop and procedure rules: Recode to avoid loop or procedure call overhead.
- Hidden costs of high-level languages
- String comparisons in C: proportional to length of the string, not constant
- Object construction / de-allocation: very expensive
- Matrix access: row-major order $\neq$ column-major order
- Exploit algebraic identities
- Avoid unnecessary computations inside the loops

Where Speedups Come From?

McGeoch reports conventional wisdom, based on studies in the literature.

- Concurrency is tricky: bad -7x to good 500x
- Classic algorithms: to 1trillion and beyond
- Data-aware: up to 100x
- Memory-aware: up to 20x
- Algorithm tricks: up to 200x
- Code tuning: up to 10x
- Change platforms: up to 10x

In Optimization

- In IP
  bounding and cutting techniques
- CP solvers
  filtering techniques
- LS solvers
  moves and incremental evaluation machinery

Relevant Literature

Bentley, **Writing Efficient Programs; Programming Pearls** (Chapter 8 Code Tuning)

Kernighan and Pike, **The Practice of Programming** (Chapter 7 Performance).
Shirazi, **Java Performance Tuning**, O'Reilly

McCluskey, **Thirty ways to improve the performance of your Java program.** Manuscript and website: www.glenmccl.com/jperf

Randal E. Bryant e David R. O'Hallaron: **Computer Systems: A Programmer's Perspective**, Prentice Hall, 2003, (Chapter 5)