DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS
FOR COMBINATORIAL OPTIMZATION

Lecture 5

Construction Heuristics

Marco Chiarandini

Outline

Outline

Last Time

- Generic Methods
  (Branch and bound, Dynamic Programming,
  Linear Programming, Integer Programming)
  - Search Methods and Constraint Programming

Informed Complete Tree Search

A* search

- The priority assigned to a node x is determined by the function

$$f(x) = g(x) + h(x)$$

$g(x)$: cost of the path so far
$h(x)$: heuristic estimate of the minimal cost to reach the goal from x.

- It is optimal if $h(x)$ is an
  - admissible heuristic: *never overestimates* the cost to reach the goal
  - consistent: $h(n) \leq c(n, a, n') + h(n')$

A* best-first search

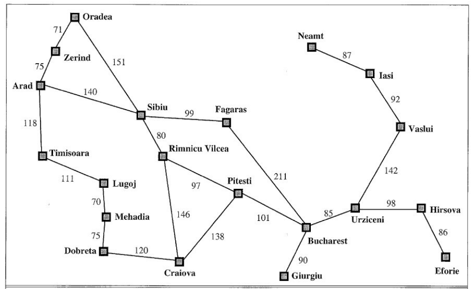


Figure 3.2 A simplified road map of part of Romania.

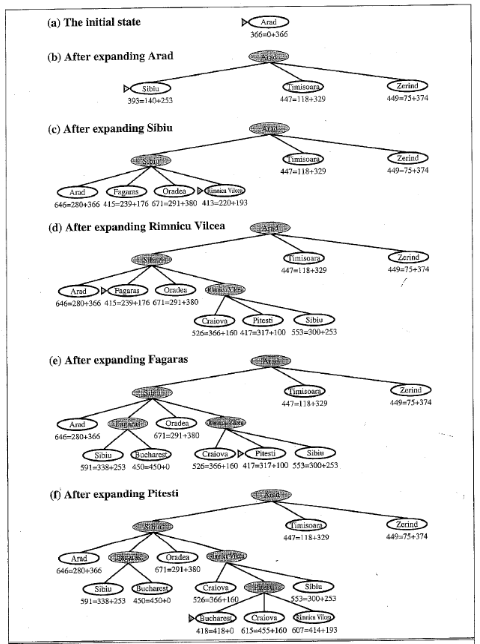


Figure 4.3 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 4.1.

A* search

Possible choices for admissible heuristic functions

- optimal solution to an easily solvable **relaxed problem**
- optimal solution to an easily solvable **subproblem**
- learning from experience by gathering statistics on state features
- preferred heuristics functions with higher values (provided they do not overestimate)
- if several heuristics available $h_1, h_2, \ldots, h_m$ and not clear which is the best then:

$$h(x) = \max\{h_1(x), \ldots, h_m(x)\}$$

A* search
Drawbacks

- Time complexity: In the worst case, the number of nodes expanded is exponential, but it is polynomial when the heuristic function h meets the following condition:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

h^* is the optimal heuristic, the exact cost of getting from x to the goal.

- Memory usage: In the worst case, it must remember an exponential number of nodes.
Several variants: including iterative deepening A* (IDA*), memory-bounded A* (MA*) and simplified memory bounded A* (SMA*) and recursive best-first search (RBFS)

Outline

Incomplete Search Paradigm

Heuristic: a common-sense rule (or set of rules) intended to increase the probability of solving some problem

Construction heuristics

They are closely related to tree search techniques but correspond to a single path from root to leaf

- search space = partial candidate solutions
- search step = extension with one or more solution components

```
Construction Heuristic (CH):
$s := \emptyset$
while $s$ is not a complete solution do
    choose a solution component $c$
    add the solution component to $s$
```

Systematic search is often better suited when ...

- proofs of insolubility or optimality are required;
- time constraints are not critical;
- problem-specific knowledge can be exploited.

Heuristics are often better suited when ...

- non linear constraints and non linear objective function;
- reasonably good solutions are required within a short time;
- problem-specific knowledge is rather limited.

Complementarity:
Local and systematic search can be fruitfully combined, *e.g.*, by using local search for finding solutions whose optimality is proved using systematic search.

Best-first search (aka greedy)

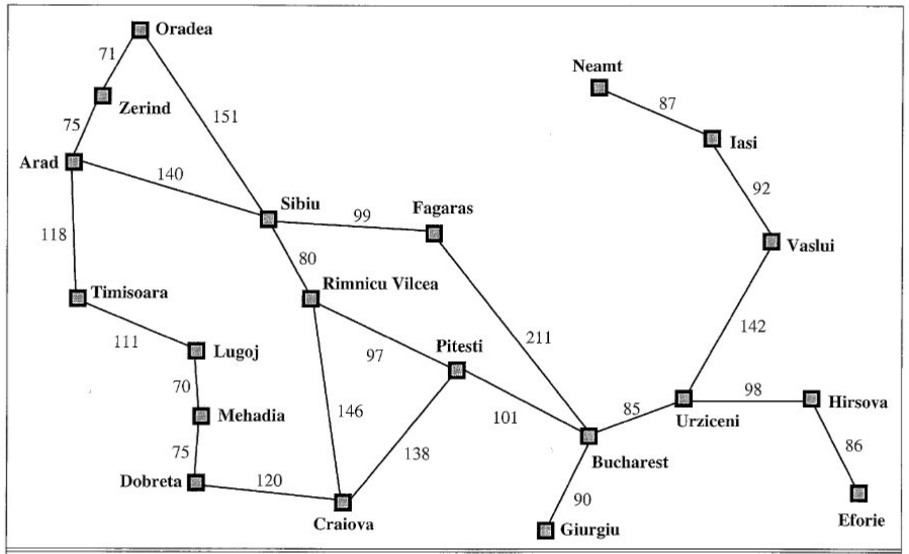


Figure 3.2 A simplified road map of part of Romania.

Best-first search (aka greedy)

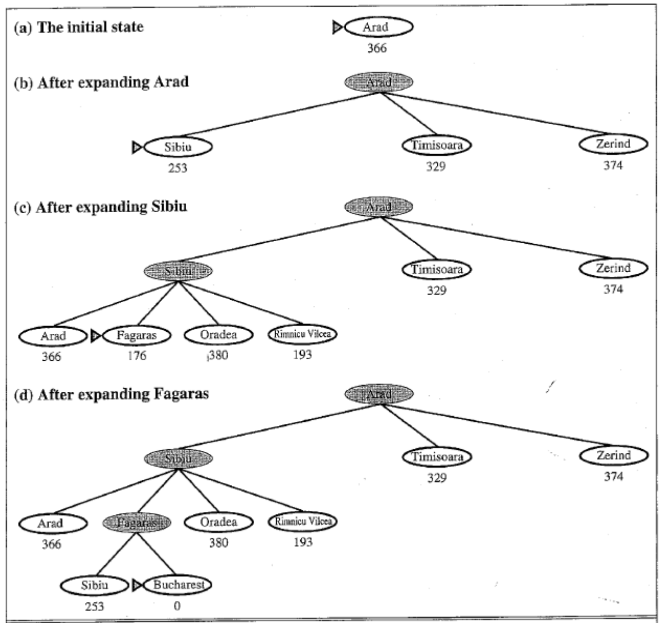
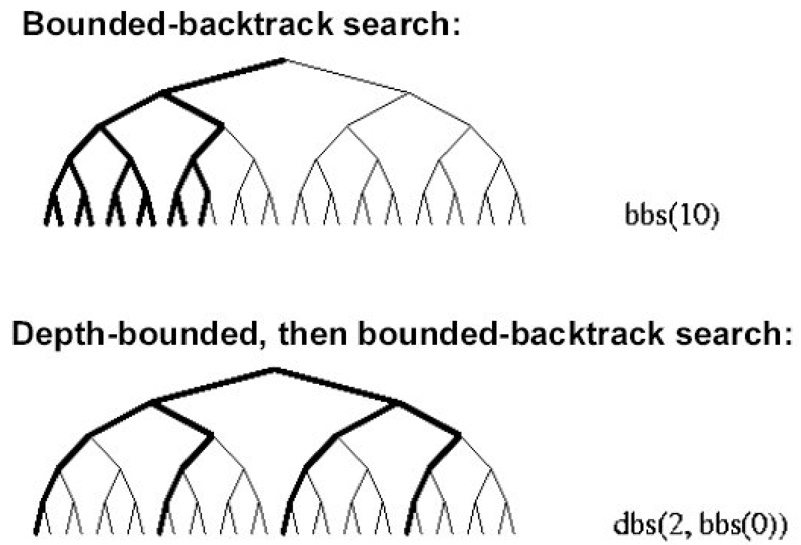


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

Greedy algorithms

- Strategy: always make the choice that is best at the moment.
- They are not generally guaranteed to find globally optimal solutions (but sometimes they do: Minimum Spanning Tree, Single Source Shortest Path, etc.)

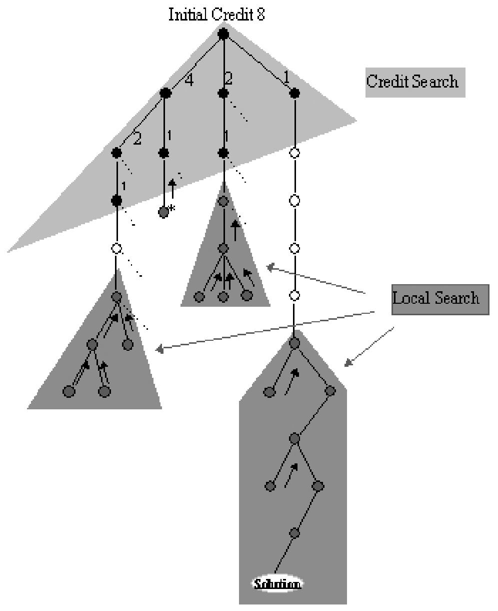
Bounded backtrack



Bounded-backtrack search:
bbs(10)
Depth-bounded, then bounded-backtrack search:
dbs(2, bbs(0))

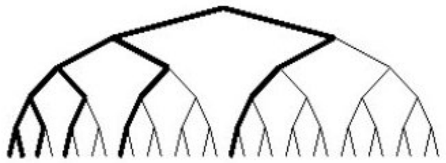
Credit-based search

- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, the distribution of credit among the children, and the amount of local backtracking at the bottom.



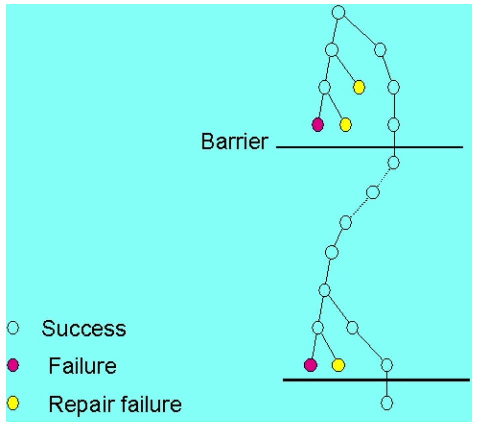
Limited Discrepancy Search (LDS)

- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Barrier Search

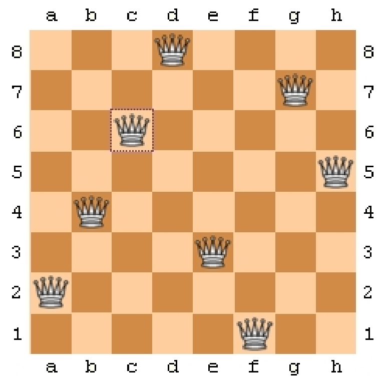
- Extension of LDS
- Key idea: we may encounter several, independent problems in our heuristic choice. Each of these problems can be overcome locally with a limited amount of backtracking.
- At each barrier start LDS-based backtracking



N-Queens problem

Input: A chessboard of size N × N

Task: Find a placement of n queens on the board such that no two queens are on the same row, column, or diagonal.



http://en.wikipedia.org/wiki/Eight_queens_puzzle

Examples of application of incomplete search methods:

http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm

N^2 Queens – queen graph coloring problem

Input: A chessboard of size N × N

Question: Given such a chessboard, is it possible to place N sets of N queens on the board so that no two queens of the same set are in the same row, column, or diagonal?

0	5	9	6	3	8	4	1	10	11	7	2
7	11	4	2	1	6	10	3	0	8	9	5
8	1	10	9	5	2	0	7	11	6	3	4
10	0	3	8	7	11	9	5	4	1	2	6
5	6	11	4	2	1	3	0	8	9	10	7
11	7	0	1	10	4	8	6	3	2	5	9
2	8	6	3	9	5	7	11	1	10	4	0
3	4	5	0	11	10	6	9	2	7	8	1
9	2	1	10	4	7	5	8	6	3	0	11
4	10	7	11	0	3	1	2	9	5	6	8
6	3	2	5	8	9	11	4	7	0	1	10
1	9	8	7	6	0	2	10	5	4	11	3

The answer is yes $\iff$ the graph has coloring number N.
The graph is N-colourable whenever $N \mod 6$ is 1 or 5 (but the condition is only sufficient and not necessary)

http://users.encs.concordia.ca/~chvatal/queengraphs.html

Randomization in Tree Search Methods

- Dynamical selection of solution components in construction or choice points in backtracking.
- Randomization of construction method or selection of choice points in backtracking while still maintaining the method complete $\rightsquigarrow$ *randomized systematic search*.
- Randomization can also be used in incomplete search

Outline

Rollout/Pilot Method

Derived from A*

- Each candidate solution is a collection of m components $s = (s_1, s_2, \ldots, s_m)$.
- Master process add components sequentially to a partial solution $S_k = (s_1, s_2, \ldots s_k)$
- At the k-th iteration the master process evaluates seemly feasible components to add based on a look-ahead strategy based on heuristic algorithms.
- The evaluation function $H(S_{k+1})$ is determined by sub-heuristics that complete the solution starting from S_k
- Sub-heuristics are combined in $H(S_{k+1})$ by
  - weighted sum
  - minimal value

Speed-ups:

- halt whenever cost of current partial solution exceeds current upper bound
- evaluate only a fraction of possible components

Beam Search

Possible extension of tree based construction heuristics:

- maintains a set B of bw (beam width) partial candidate solutions
- at each iteration extend each solution from B in fw (filter width) possible ways
- rank each $bw \times fw$ candidate solutions and take the best bw partial solutions
- complete candidate solutions obtained by B are maintained in B_f
- Stop when no partial solution in B is to be extend

Iterated Greedy

Key idea: use greedy construction

- alternation of Construction and Deconstruction phases
- an acceptance criterion decides whether the search continues from the new or from the old solution.

Iterated Greedy (IG):

```
determine initial candidate solution $s$
while termination criterion is not satisfied do
    $r := s$
    greedily destruct part of $s$
    greedily reconstruct the missing part of $s$
    based on acceptance criterion,
       keep $s$ or revert to $s := r$
```

Extension: Squeaky Wheel

Key idea: solutions can reveal problem structure which maybe worth to exploit.

Use a greedy heuristic repeatedly by prioritizing the elements that create troubles.

Squeaky Wheel

- Constructor: greedy algorithm on a sequence of problem elements.
- Analyzer: assign a penalty to problem elements that contribute to flaws in the current solution.
- Prioritizer: uses the penalties to modify the previous sequence of problem elements. Elements with high penalty are moved toward the front.

Hybridize with subsidiary perturbative search

Example: on the SMTWTP

Greedy Randomized Adaptive Search Procedure (GRASP)

Key Idea: Combine randomized constructive search with subsequent perturbative search.

Motivation:

- Candidate solutions obtained from construction heuristics can often be substantially improved by perturbative search.
- Perturbative search methods typically often require substantially fewer steps to reach high-quality solutions
  when initialized using greedy constructive search rather than random picking.
- By iterating cycles of constructive + perturbative search, further performance improvements can be achieved.

Greedy Randomized "Adaptive" Search Procedure (GRASP):

```
While termination criterion is not satisfied:
|  generate candidate solution s using
|      subsidiary greedy randomized constructive search
|  perform subsidiary perturbative search on s
```

Note:

- Randomization in *constructive search* ensures that a large number of good starting points for *subsidiary perturbative search* is obtained.
- Constructive search in GRASP is 'adaptive' (or dynamic): Heuristic value of solution component to be added to given partial candidate solution r may depend on solution components present in r.
- Variants of GRASP without perturbative search phase (aka *semi-greedy heuristics*) typically do not reach the performance of GRASP with perturbative search.

Restricted candidate lists (RCLs)

- Each step of *constructive search* adds a solution component selected uniformly at random from a restricted candidate list (RCL).
- RCLs are constructed in each step using a *heuristic function* h.
  - RCLs based on cardinality restriction comprise the k best-ranked solution components. (k is a parameter of the algorithm.)
  - RCLs based on value restriction comprise all solution components l for which $h(l) \leq h_{min} + \alpha \cdot (h_{max} - h_{min})$, where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

Example: GRASP for SAT [Resende and Feo, 1996]

- **Given:** CNF formula F over variables $x_1, \ldots, x_n$
- **Subsidiary constructive search:**
  - start from empty variable assignment
  - in each step, add one atomic assignment (*i.e.*, assignment of a truth value to a currently unassigned variable)
  - heuristic function $h(i, v)$:= number of clauses that become satisfied as a consequence of assigning $x_i := v$
  - RCLs based on cardinality restriction (contain fixed number k of atomic assignments with largest heuristic values)
- **Subsidiary perturbative search:**
  - iterative best improvement using 1-flip neighborhood
  - terminates when model has been found or given number of steps has been exceeded

GRASP has been applied to many combinatorial problems, including:

- SAT, MAX-SAT
- various scheduling problems

Extensions and improvements of GRASP:

- reactive GRASP (*e.g.*, dynamic adaptation of α during search)

Adaptive Iterated Construction Search

Key Idea: Alternate construction and perturbative local search phases as in GRASP, exploiting experience gained during the search process.

Realisation:

- Associate *weights* with possible decisions made during constructive search.
- Initialize all weights to some small value τ_0 at beginning of search process.
- After every cycle (= constructive + perturbative local search phase), update weights based on solution quality and solution components of current candidate solution.

Adaptive Iterated Construction Search (AICS):

```
initialise weights
While termination criterion is not satisfied:
  | generate candidate solution $s$ using
  |     subsidiary randomized constructive search
  | perform subsidiary local search on $s$
  | adapt weights based on $s$
```

Subsidiary constructive search:

- The solution component to be added in each step of *constructive search* is based on *weights* and heuristic function h.
- h can be standard heuristic function as, *e.g.*, used by greedy construction heuristics, GRASP or tree search.
- It is often useful to design solution component selection in constructive search such that any solution component may be chosen (at least with some small probability) irrespective of its weight and heuristic value.

Subsidiary perturbative local search:

- As in GRASP, perturbative local search phase is typically important for achieving good performance.
- Can be based on Iterative Improvement or more advanced LS method (the latter often results in better performance).
- Tradeoff between computation time used in construction phase *vs* local search phase (typically optimized empirically, depends on problem domain).

Weight updating mechanism:

- Typical mechanism: increase weights of all solution components contained in candidate solution obtained from local search.
- Can also use aspects of search history; *e.g.*, current *incumbent candidate solution* can be used as basis for weight update for additional intensification.

Example: A simple AICS algorithm for the TSP (1)

(Based on Ant System for the TSP [Dorigo *et al.*, 1991].)

- Search space and solution set as usual (all Hamiltonian cycles in given graph G).
- Associate weight τ_{ij} with each edge (i, j) in G.
- Use heuristic values $\eta_{ij} := 1/w((i, j))$.
- Initialize all weights to a small value τ_0 (parameter).
- *Constructive search* starts with randomly chosen vertex and iteratively extends partial round trip ϕ by selecting vertex not contained in ϕ with probability

$$\frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N'(i)} [\tau_{il}]^\alpha \cdot [\eta_{ij}]^\beta}$$

Example: A simple AICS algorithm for the TSP (2)

- *Subsidiary local search* = iterative improvement based on standard 2-exchange neighborhood (until local minimum is reached).
- *Weight update* according to

$$\tau_{ij} := (1 - \rho) \cdot \tau_{ij} + \Delta(i, j, s')$$

  where $\Delta(i, j, s') := 1/f(s')$, if edge (i, j) is contained in the cycle represented by s', and 0 otherwise.
- Criterion for weight increase is based on intuition that edges contained in short round trips should be preferably used in subsequent constructions.

Multilevel Refinement

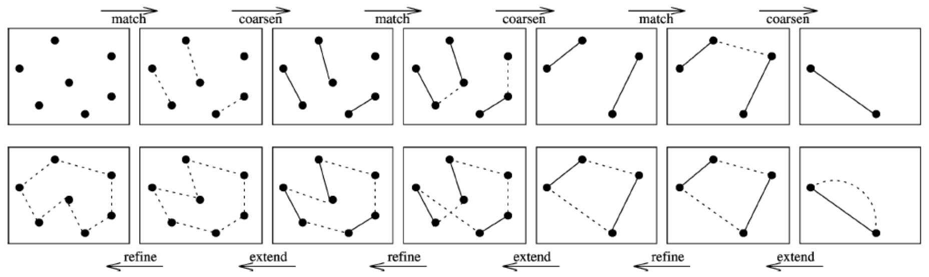
Key idea: make the problem recursively less refined creating a hierarchy of approximations of the original problem.

- an initial solution is found on the original problem or at a refined level
- solutions are iteratively refined at each level
- use of projection operators to transfer the solution from one level to another

```
Multilevel Refinement
while Termination criterion is not satisfied do
    coarse the problem $\pi_0$ into $\pi_i$, $i = 0, \dots, k$ successive non degenerate problems
    $i = k$
    determine an initial candidate solution for $\pi_k$
    repeat
        $i = i - 1$
        extend the solution found in $\pi_{i+1}$ to $\pi_i$
        use subsidiary perturbative search to refine the solution on $\pi_i$
    until $i \geq 0$ ;
```

Example: Multilevel Refinement for TSP

Coarsen:	fix some edges and contract vertices
Solve:	matching (always match vertices with the nearest unmatched neighbors)
Extend:	uncontract vertices
Refine:	LK heuristic



Note

- crucial point: the solution to each refined problem must contain a solution of the original problem (even if it is a poor solution)

Applications to

- Graph Partitioning
- Traveling Salesman
- Graph Coloring

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
  - Nearest Insertion
  - Farthest Insertion
  - Random Insertion
  - Clarke-Wright savings heuristic
- Heuristics based on Trees
  - Minimum span tree heuristic
  - Christofides' heuristics
  - Fast recursive partitioning heuristic

Construction Heuristics for TSP

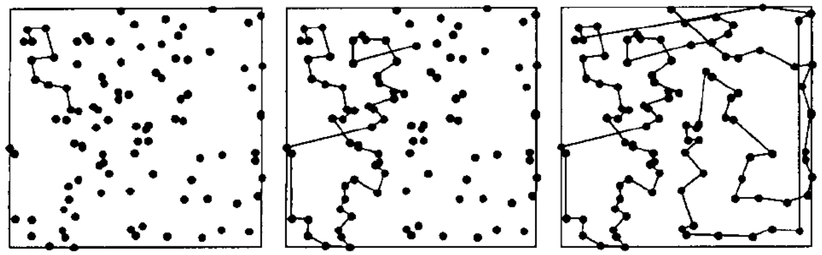


Figure 1. The Nearest Neighbor heuristic.

Construction Heuristics for TSP

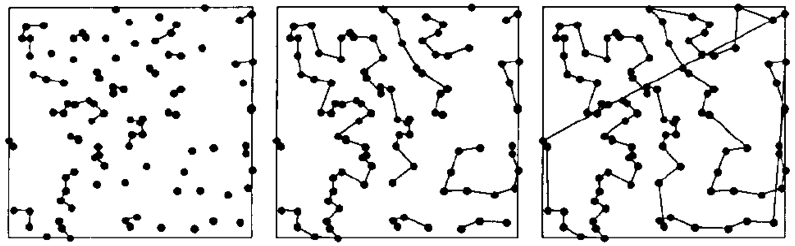


Figure 5. The Multiple Fragment heuristic.

Construction Heuristics for TSP

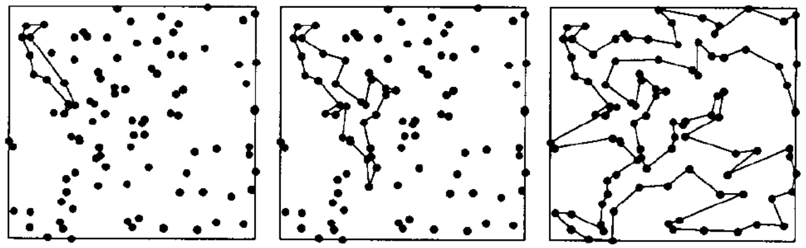


Figure 8. The Nearest Addition heuristic.

Construction Heuristics for TSP

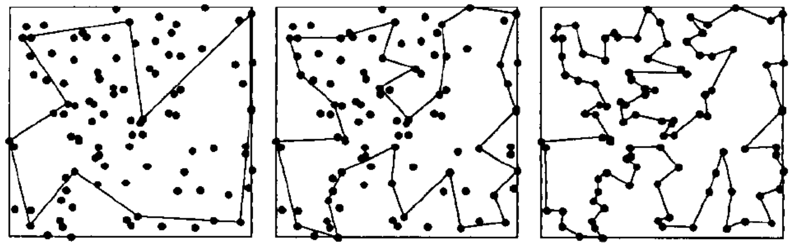


Figure 11. The Farthest Addition heuristic.

Construction Heuristics for TSP

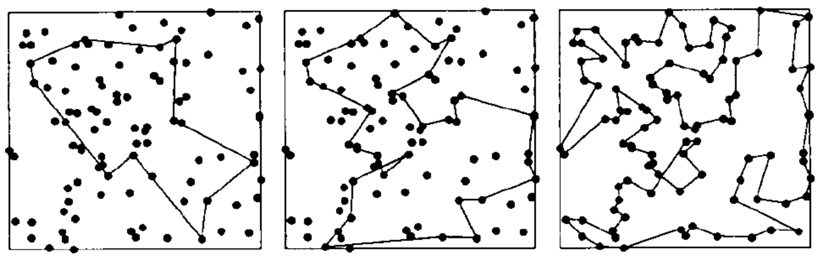


Figure 14. The Random Addition heuristic.

Construction Heuristics for TSP

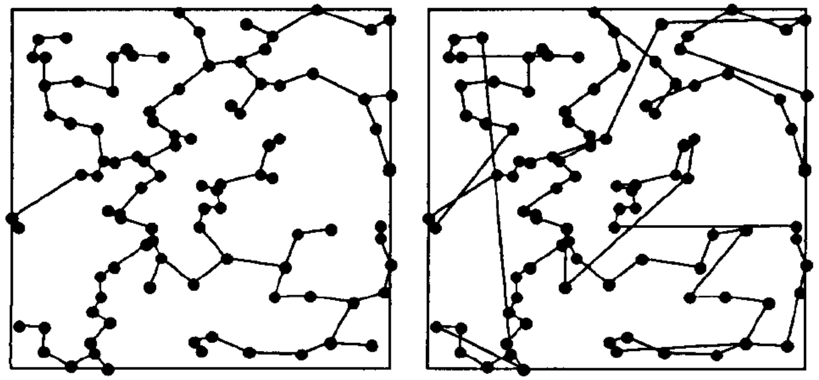


Figure 18. The Minimum Spanning Tree heuristic.

Construction Heuristics for TSP

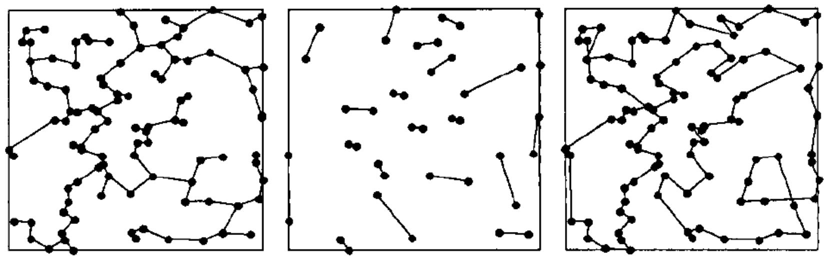


Figure 19. Christofides' heuristic.

Outline

Preprocessing Rules

Polynomial simplifications

SAT (CNF formulae)

- elimination of duplicates
- elimination tautological clauses
- elimination subsumed clauses
- elimination clauses with pure literals
- elimination unit clauses and unit propagation

k-coloring

- Remove under-constrained nodes
- Remove subsumed nodes
- Merge nodes that must have the same color

Outline

Software Development: Extreme Programming & Scrum

Planning

Release planning creates the schedule // Make frequent small releases // The project is divided into iterations

Designing

Simplicity // No functionality is added early // Refactor: eliminate unused functionality and redundancy

Coding

Code must be written to agreed standards // Code the unit test first // All production code is pair programmed // Leave optimization till last // No overtime

Testing

All code must have unit tests // All code must pass all unit tests before it can be released // When a bug is found tests are created