DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 8

Experimental Analysis of ILS and Task 2
VLSN, Iterated Greedy

Marco Chiarandini

Outline

1. Task 2 – 2-opt for TSP
2. Metaheuristics
   - Very Large Scale Neighborhoods
   - Iterated Greedy
   - Multilevel Refinement
3. Exercises
   - Set Covering

Outline

1. Task 2 – 2-opt for TSP

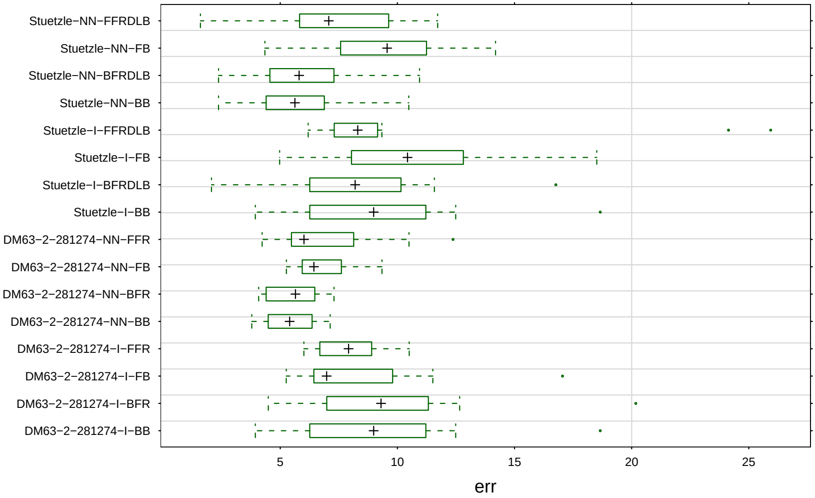
2. Metaheuristics
   - Very Large Scale Neighborhoods
   - Iterated Greedy
   - Multilevel Refinement

3. Exercises
   - Set Covering

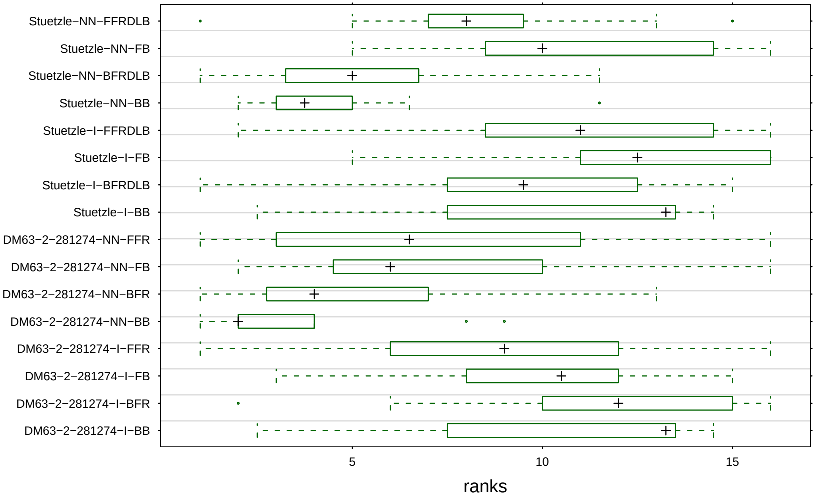
Numerical Results

Alg.	Err.	Size	sec.
DM63-2-281274-I-BB	8.99	657	171.50
DM63-2-281274-NN-BB	5.40	657	43.76
Stuetzle-I-BB	8.99	657	4.16
Stuetzle-NN-BB	5.63	657	1.07
DM63-2-281274-I-FB	6.98	657	200.07
DM63-2-281274-NN-FB	6.44	657	66.78
Stuetzle-I-FB	10.43	657	0.07
Stuetzle-NN-FB	9.56	657	0.08
DM63-2-281274-I-BFR	9.30	657	17.19
DM63-2-281274-I-FFR	7.92	657	4.78
DM63-2-281274-NN-BFR	5.65	657	1.35
DM63-2-281274-NN-FFR	6.01	657	1.03
Stuetzle-I-BFRDLB	8.20	657	0.57
Stuetzle-I-FFRDLB	8.31	657	0.00
Stuetzle-NN-BFRDLB	5.81	657	0.15
Stuetzle-NN-FFRDLB	7.07	657	0.00

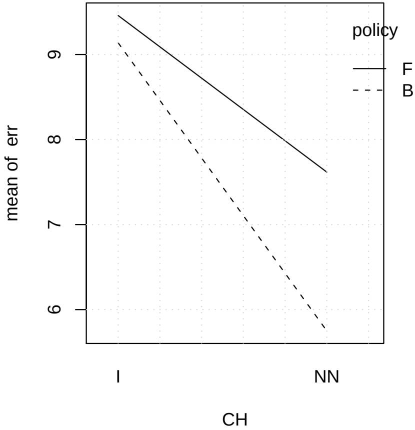
Boxplots of Errors

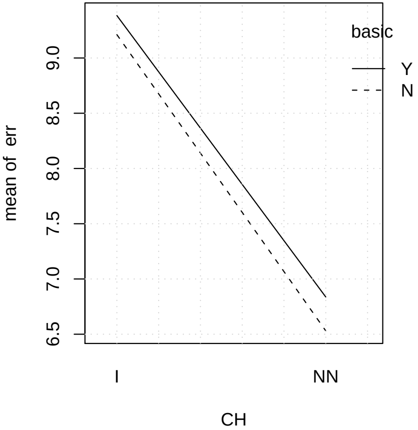


Boxplots of Ranks

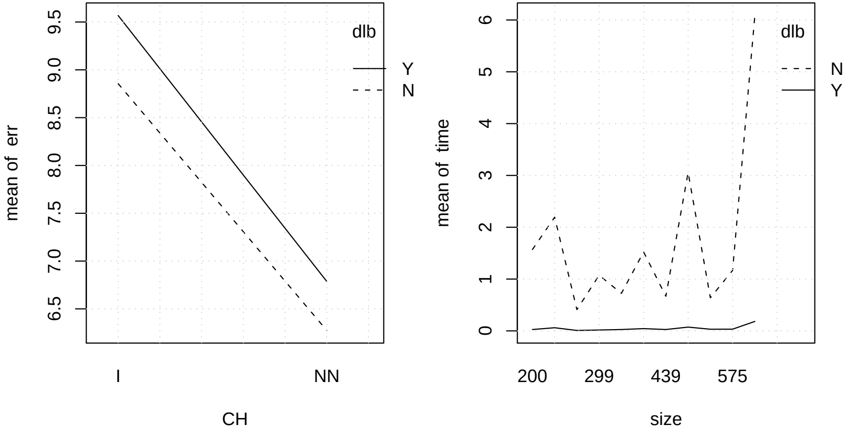


Interaction Plots

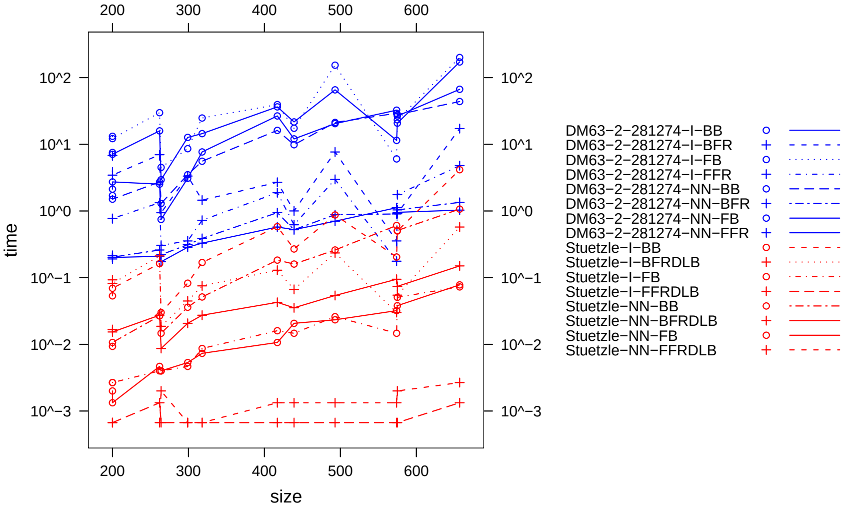




Interaction Plots



Line Plot: time vs size



Program Profiling

- Plot the development of
  - best visited solution quality
  - current solution quality

  over time and compare with other features of the algorithm.

- Profile time consumption per program components
  under Linux: `gprof`
  1. add flag -pg in compilation
  2. run the program
  3. `gprof program-file > a.txt`

Outline

So far...

- beam search
- greedy randomized adaptive search procedure
- variable neighborhood search (and extensions)
- variable depth search (and Lin-Kernighan heuristic for TSP)
- ejection chains and dynasearch
- randomized iterative improvement (min-conflict heuristic, novelty)
- probabilistic iterative improvement (metropolis algorithm)
- simulated annealing (and noising method, threshold method, old bachelor acceptance)
- tabu search
- dynamic local search (guided local search)
- iterated local search

Very Large Scale Neighborhoods

Key idea: use very large neighborhoods that can be searched very efficiently (preferably in polynomial time) or are searched heuristically

General framework to Variable Depth Search, Dynasearch and Ejection Chains

Three parts needed in a neighborhood search technique:

- Neighborhood structure: exponentially large neighborhoods
- Method for searching: polynomial time search algorithm
- Method for selecting: select the optimal solution in the neighborhood

VLSN allows to use the literature on polynomial time algorithms

Examples of VLSN Search:

- based on Dynamic Programming or Network Flows
  - dynasearch and variants (ex. SMTWTP)
  - assignment based neighborhoods (ex. TSP)
  - cyclic exchange neighborhood (ex. VRP)
- based on polynomially solvable special cases of hard combinatorial optimization problems
  - Pyramidal tours
  - Halin Graphs

$\Rightarrow$ Idea: turn a special case into a neighborhood

Note

- they can provide a form of look ahead
- they can be effective but must be tested empirically

Iterated Greedy

Key idea: use greedy construction

- replace Local Search and Perturbation Phase in Iterated Local Search by Construction and Deconstruction phases
- an acceptance criterion decide whether the search continue 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.

Can be seen as a case of GRASP or Iterated Greedy

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

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 π_0 into π_i, $i = 0, \ldots, k$ successive non degenerate problems
  $i = k$
  determine an initial candidate solution for π_k
  repeat
    $i = i - 1$
    extend the solution found in π_{i+1} to π_i
    use subsidiary perturbative search to refine the solution on π_i
  until $i \geq 0$;

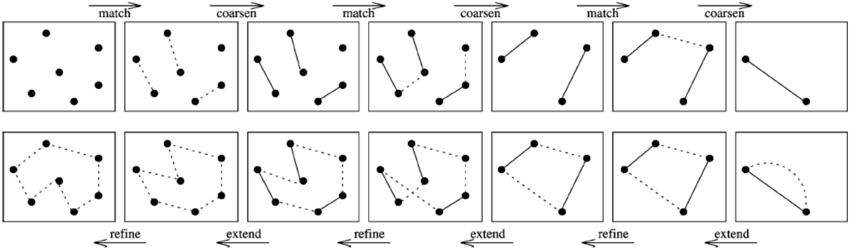
Example: Multilevel Refinement for TSP

Coarsen: fix some edges and contract vertices

Solve: matching
(always match vertices with 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

1. Task 2 – 2-opt for TSP

2. Metaheuristics
   - Very Large Scale Neighborhoods
   - Iterated Greedy
   - Multilevel Refinement

3. Exercises
   - Set Covering

Example Problems: So far...

- Traveling Salesman Problem (TSP)
- Vertex Coloring Problem (GCP)
- Propositional Satisfiability (SAT and MAX-SAT)
- Constraint Satisfaction Problem (CSP and MAX-CSP)
- The Single Machine Total Weighted Tardiness Problem (SMTWTP)
- The p-median Problem
- Quadratic Assignment Problem

Set Covering Problem

Input: a finite set X and a family $\mathcal{F}$ of subsets of X such that every element of X belongs to at least one subset in $\mathcal{F}$:

$$X = \cup_{S \in \mathcal{F}} S$$

Task: Find a minimum cost subset $\mathcal{C}$ of $\mathcal{F}$ whose members cover all X:

$$\min \quad \textstyle\sum_{S \in \mathcal{C}} w(S)$$

$$\text{such that} \quad X = \cup_{S \in \mathcal{C}} S \qquad (1)$$

Any $\mathcal{C}$ satisfying (1) is said to cover X

Covering, Partitioning, Packing

Set Covering

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
& \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \\
& x_j \in \{0,1\}
\end{aligned}
$$

Set Partitioning

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
& \sum_{j=1}^{n} a_{ij} x_j = 1 \quad \forall i \\
& x_j \in \{0,1\}
\end{aligned}
$$

Set Packing

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} c_j x_j \\
& \sum_{j=1}^{n} a_{ij} x_j \leq 1 \quad \forall i \\
& x_j \in \{0,1\}
\end{aligned}
$$

Plan: How to Tackle a Problem

- Formulate clearly the problem
- Understand the combinatorial hardness of the problem
- Model the problem and recognise possible similar problems
- Search in the literature (that is, on the Internet) for:
  - complexity results (is the problem NP-hard?)
  - solution algorithms for the original problem
  - solution algorithms for the simplified problem
- Design possible solution algorithms
- Test experimentally with the goals of:
  - configuring
  - tuning parameters
  - comparing
  - studing the behaviour (scaling and optimal deviation)