DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 3

# Other Example Problems and Local Search Components

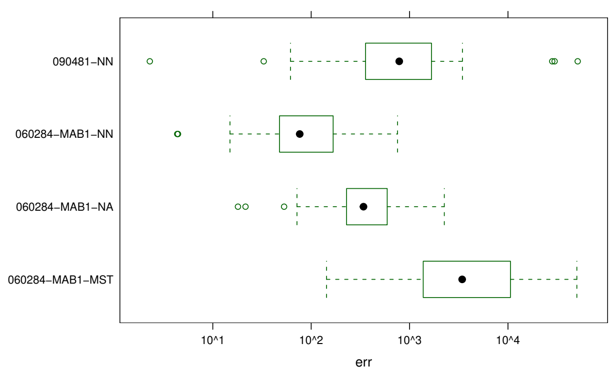
Marco Chiarandini

## Outline





## Results





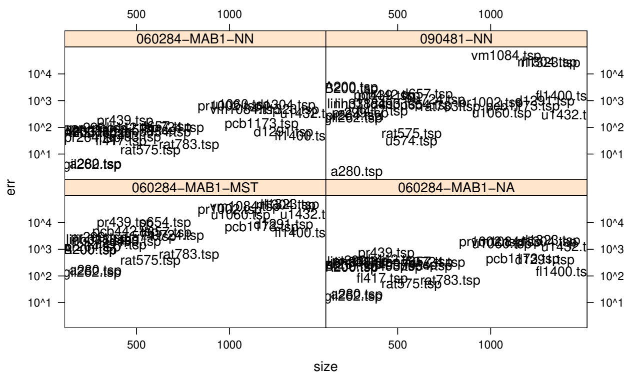

## Results

| Heuristic | median error |
|---|---|
| 060284-MAB1-MST | 3425.01 |
| 060284-MAB1-NA | 340.03 |
| 060284-MAB1-NN | 76.46 |
| 090481-NN | 785.27 |

In the literature: the NN has an error of 26%

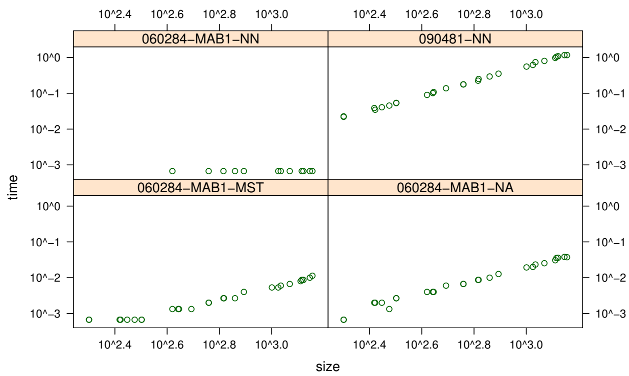


## Results






## Results






## So far...

- Traveling Salesman Problem (TSP)
- Vertex Coloring Problem (GCP)
- Propositional Satisfiability (SAT and MAX-SAT)



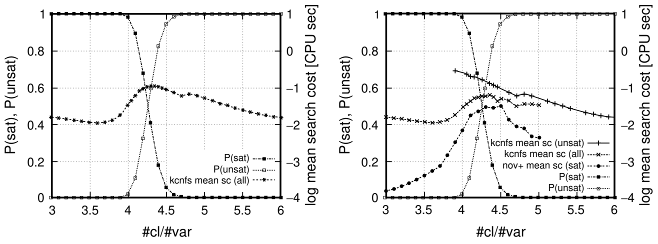
## The Propositional Satisfiability Problem

Polynomial simplifications of CNF formulae

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

## Phase Transition for 3-SAT

Random instances ⇒ $m$ clauses of $n$ uniformly chosen variables






## SAT Related Problems

- Propositional Validity Problem
- Satisfiability Problem for Quantified Boolean Formulae (QSAT)
- #SAT (countable and decision)



## Constraint Satisfaction Problem

- *Given:* a triple $\langle X, D, C\rangle$, where $X$ is a set of variable, $D$ is a domain of values, and $C$ is a set of constraints. Every constraint is a pair $\langle t, R\rangle$, where $t$ is a tuple of variables and $R$ is a relation (or a set of tuples of values; all these tuples having the same number of elements)

  An evaluation of the variables is a function from variables to $v : X \to D$. Such an evaluation satisfies a constraint $\langle (x_1, \dots, x_n), R\rangle$ if $(v(x_1), \dots, v(x_n)) \in R$.
- *Task:* A solution is an evaluation that satisfies all constraints.



## Definition: Local Search Algorithm

For given problem instance $\pi$:

- search space $S(\pi)$
- solution set $S'(\pi) \subseteq S(\pi)$
- neighborhood relation $N(\pi) \subseteq S(\pi) \times S(\pi)$
- set of memory states $M(\pi)$
- initialization function *init* : $\emptyset \mapsto \mathcal{D}(S(\pi) \times M(\pi))$
- step function *step* : $S(\pi) \times M(\pi) \mapsto \mathcal{D}(S(\pi) \times M(\pi))$
- termination predicate *terminate* : $S(\pi) \times M(\pi) \mapsto \mathcal{D}(\{\top, \bot\})$



## Example: Uninformed random walk for $k$-col

- **search space** $S$**:** set of all $k$-colorings of $G$
- **solution set** $S'$**:** set of all proper $k$-colorings of $G$
- **neighborhood relation** $N$**:** *1-exchange neighborhood*, *i.e.*, colorings are neighbors under $N$ iff they differ in the color at one vertex
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from $\Gamma$, *i.e.*, $init(\varphi', m) := 1/|S|$ for all coloring $\varphi'$ and memory states $m$
- **step function:** uniform random choice from current neighborhood, *i.e.*, $step(\varphi, m)(\varphi', m) := 1/|N(\varphi)|$ for all assignments $\varphi$ and memory states $m$, where $N(a) := \{\varphi' \in S \mid N(\varphi, \varphi')\}$ is the set of all neighbors of $\varphi$.
- **termination:** when $terminate(\varphi, m) := 1$ if $\varphi$ is a feasible coloring of $G$, and $0$ otherwise.



## Definition: LS Algorithm Components (continued)

### Note:

- Procedural versions of $init$, $step$ and $terminate$ implement sampling from respective probability distributions.
- Memory state $m$ can consist of multiple independent attributes, *i.e.*, $M(\pi) := M_1 \times M_2 \times \ldots \times M_{l(\pi)}$.
- LS algorithms realize *Markov processes*:
  behavior in any search state $(s, m)$ depends only on current position $s$ and (limited) memory $m$.



## Definition: LS Algorithm Components (continued)

### Search Space

Defined by the solution representation:

- permutations
- assignment vectors
- sets or lists



## Definition: LS Algorithm Components (continued)

Neighborhood relation (structure): $N : S \times S \to \{T, F\}$ or $N \subseteq S \times S$

- neighborhood (set) of candidate solution $s$: $N(s) := \{s' \in S \mid N(s, s')\}$
- neighborhood graph of problem instance $\pi$: $G_N(\pi) := (S(\pi), N(\pi))$

*Note:* Diameter of $G_N$ = worst-case lower bound for number of search steps required for reaching (optimal) solutions

### Example:

$k$-col instance with $n$ vertices, 1-exchange neighborhood:
$G_N$ = $n$-dimensional hypercube; diameter of $G_N = n$.

A neighborhood function $N : S \to S \times S$ is also defined through an operator. An operator $\Delta$ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(S) \iff \exists \delta \in \Delta | \delta(s) = s'$$

## Definition

$k$-exchange neighborhood: candidate solutions $s, s'$ are neighbors iff $s$ differs from $s'$ in at most $k$ solution components

## Examples:

- 1-exchange (flip) neighborhood for $k$-col
  (solution components = single vertex assignments)
- 2-exchange neighborhood for TSP
  (solution components = edges in given graph)



## Definition: LS Algorithm Components (continued)

Step function

- Search step (or move): pair of search positions $s, s'$ for which $s'$ can be reached from $s$ in one step, *i.e.*, $N(s, s')$ and $step(s, m)(s', m') > 0$ for some memory states $m, m' \in M$.
- Search trajectory: finite sequence of search positions $(s_0, s_1, \ldots, s_k)$ such that $(s_{i-1}, s_i)$ is a *search step* for any $i \in \{1, \ldots, k\}$ and the probability of initializing the search at $s_0$ is greater zero, *i.e.*, $init(s_0, m) > 0$ for some memory state $m \in M$.
- Search strategy: specified by *init* and *step* function; to some extent independent of problem instance and other components of LS algorithm.
  - Random
  - Evaluation function
  - ...



## Uninformed Random Picking

- $N := S \times S$
- does not use memory
- $init, step$: uniform random choice from $S$,
  *i.e.*, for all $s, s' \in S$, $init(s) := step(s)(s') := 1/|S|$

## Uninformed Random Walk

- does not use memory
- $init$: uniform random choice from $S$
- $step$: uniform random choice from current neighborhood, *i.e.*, for all $s, s' \in S$, $step(s)(s') := 1/|N(s)|$ if $N(s, s')$, and $0$ otherwise

*Note:* These uninformed LS strategies are quite ineffective, but play a role in combination with more directed search strategies.



## Evaluation function:

- function $g(\pi) : S(\pi) \mapsto \mathbb{R}$ that maps candidate solutions of a given problem instance $\pi$ onto real numbers, such that global optima correspond to solutions of $\pi$;
- used for ranking or assessing neighbors of current search position to provide guidance to search process.

## Evaluation *vs* objective functions:

- *Evaluation function*: part of LS algorithm.
- *Objective function*: integral part of optimization problem.
- Some LS methods use evaluation functions different from given objective function (*e.g.*, dynamic local search).



## Iterative Improvement (II)

- does not use memory
- $init$: uniform random choice from $S$
- $step$: uniform random choice from improving neighbors,
  *i.e.*, $step(s)(s') := 1/|I(s)|$ if $s' \in I(s)$, and $0$ otherwise,
  where $I(s) := \{s' \in S \mid N(s, s') \wedge g(s') < g(s)\}$
- terminates when no improving neighbor available
  (to be revisited later)
- different variants through modifications of step function
  (to be revisited later)

*Note:* II is also known as *iterative descent* or *hill-climbing*.



## Example: Iterative Improvement for $k$-col

- **search space** $S$**:** set of all $k$-colorings of $G$
- **solution set** $S'$**:** set of all proper $k$-coloring of $F$
- **neighborhood relation** $N$**:** 1-exchange neighborhood
  (as in Uninformed Random Walk)
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from $S$, *i.e.*, $init()(\varphi') := 1/|S|$ for all colorings $\varphi'$
- **step function**:
  - **evaluation function:** $g(\varphi) :=$ number of edges in $G$ whose ending vertices are assigned the same color under assignment $\varphi$
    (*Note:* $g(\varphi) = 0$ iff $\varphi$ is a proper coloring of $G$.)
  - **move mechanism:** uniform random choice from improving neighbors,
    *i.e.*, $step(\varphi)(\varphi') := 1/|I(\varphi)|$ if $s' \in I(\varphi)$,
    and $0$ otherwise, where $I(\varphi) := \{\varphi' \mid N(\varphi, \varphi') \wedge g(\varphi') < g(\varphi)\}$
- **termination**: when no improving neighbor is available
  *i.e.*, $terminate(\varphi)(\top) := 1$ if $I(\varphi) = \emptyset$, and $0$ otherwise.



## Incremental updates (aka delta evaluations)

- **Key idea:** calculate *effects of differences* between current search position $s$ and neighbors $s'$ on evaluation function value.
- Evaluation function values often consist of *independent contributions of solution components*; hence, $g(s)$ can be efficiently calculated from $g(s')$ by differences between $s$ and $s'$ in terms of solution components.
- Typically crucial for the efficient implementation of II algorithms (and other LS techniques).



## Example: Incremental updates for TSP

- solution components = edges of given graph $G$
- standard 2-exchange neighborhood, *i.e.*, neighboring round trips $p$, $p'$ differ in two edges
- $w(p') := w(p)$ − edges in $p$ but not in $p'$
  \+ edges in $p'$ but not in $p$

*Note:* Constant time (4 arithmetic operations), compared to linear time ($n$ arithmetic operations for graph with $n$ vertices) for computing $w(p')$ from scratch.

Definition:

- Local minimum: search position without improving neighbors w.r.t. given evaluation function $g$ and neighborhood $N$, *i.e.*, position $s \in S$ such that $g(s) \leq g(s')$ for all $s' \in N(s)$.
- Strict local minimum: search position $s \in S$ such that $g(s) < g(s')$ for all $s' \in N(s)$.
- *Local maxima* and *strict local maxima*: defined analogously.



Note:

- Local minima depend on $g$ and neighborhood relation $N$.
- Larger neighborhoods $N(s)$ induce
  - neighborhood graphs with smaller diameter;
  - fewer local minima.

Ideal case: exact neighborhood, *i.e.*, neighborhood relation for which any local optimum is also guaranteed to be a global optimum.

- Typically, exact neighborhoods are too large to be searched effectively (exponential in size of problem instance).
- *But:* exceptions exist, *e.g.*, polynomially searchable neighborhood in Simplex Algorithm for linear programming.



Trade-off:

- Using larger neighborhoods can improve performance of II (and other LS methods).
- *But:* time required for determining improving search steps increases with neighborhood size.

More general trade-off:

Effectiveness *vs* Efficiency (= time complexity of search steps).



In II, various mechanisms (pivoting rules) can be used for choosing improving neighbor in each step:

- Best Improvement (aka *gradient descent*, *steepest descent*, *greedy hill-climbing*): Choose maximally improving neighbor, *i.e.*, randomly select from $I^*(s) := \{s' \in N(s) \mid g(s') = g^*\}$, where $g^* := \min\{g(s') \mid s' \in N(s)\}$.

  *Note:* Requires evaluation of all neighbors in each step.

- First Improvement: Evaluate neighbors in fixed order, choose first improving step encountered.

  *Note:* Can be much more efficient than Best Improvement; order of evaluation can have significant impact on performance.



Example: Random-order first improvement for the TSP

- **Given:** TSP instance $G$ with vertices $v_1, v_2, \ldots, v_n$.
- search space: Hamiltonian cycles in $G$; use standard 2-exchange neighborhood
- **Initialization:**

  search position := fixed canonical path $(v_1, v_2, \ldots, v_n, v_1)$
  $P$ := random permutation of {1,2, . . . , n}

- **Search steps:** determined using first improvement w.r.t. $g(p)$ = weight of path $p$, evaluating neighbors in order of $P$ (does not change throughout search)
- **Termination:** when no improving search step possible (local minimum)



Speed-up Techniques: Neighborhood Pruning

- *Idea:* Reduce size of neighborhoods by excluding neighbors that are likely (or guaranteed) not to yield improvements in $g$.
- *Note:* Crucial for large neighborhoods, but can be also very useful for small neighborhoods (*e.g.*, linear in instance size).

Example: Heuristic candidate lists for the TSP

- *Intuition:* High-quality solutions likely include short edges.
- Candidate list of vertex $v$: list of $v$'s nearest neighbors (limited number), sorted according to increasing edge weights.
- Search steps (*e.g.*, 2-exchange moves) always involve edges to elements of candidate lists.
- Significant impact on performance of LS algorithms for the TSP.



## Perturbative Search on the Traveling Salesman Problem

- $k$-exchange heuristics
  - 2-opt
  - 2.5-opt
  - Or-opt
  - 3-opt
- complex neighborhoods
  - Lin-Kernighan
  - Helsgaun's Lin-Kernighan
  - Dynasearch
  - ejection chains approach

Implementations exploit speed-up techniques

- neighborhood pruning: fixed radius nearest neighborhood search
- neighborhood lists: restrict exchanges to most interesting candidates
- don't look bits: focus perturbative search to "interesting" part
- sophisticated data structures



## Iterative Improvement (2 OPT)

**procedure** *TSP-2opt-first*($s$)
  **input:** *an initial candidate tour* $s \in S(\in)$
  **output:** *a local optimum* $s \in S(\pi)$
  $\Delta = 0$;
  **for** $i = 1$ **to** $n - 2$ **do**
  **if** $i = 1$ **then** $n' = n - 1$ **else** $n' = n$
    **for** $j = i + 2$ **to** $n'$ **do**
      $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
      **if** $\Delta_{ij} < 0$ **then**
        *UpdateTour(s,i,j)*
    **end**
  **end**
**end** *TSP-2opt-first*

In total $\binom{n}{2}$ possible moves

## Computational Complexity of Local Search (1)

For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

Complexity class $\mathcal{PLS}$: class of problems for which a local search algorithm exists with polynomial time complexity for:

- search initialization
- any single search step, including computation of any evaluation function value

For any problem in $\mathcal{PLS}$ ...

- local optimality can be verified in polynomial time
- improving search steps can be computed in polynomial time
- *but:* finding local optima may require super-polynomial time



## Computational Complexity of Local Search (2)

$\mathcal{PLS}$-complete: Among the most difficult problems in $\mathcal{PLS}$; if for any of these problems local optima can be found in polynomial time, the same would hold for all problems in $\mathcal{PLS}$.

Some complexity results:

- TSP with $k$-exchange neighborhood with $k > 3$ is $\mathcal{PLS}$-complete.
- TSP with 2- or 3-exchange neighborhood is in $\mathcal{PLS}$, but $\mathcal{PLS}$-completeness is unknown.