DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 6

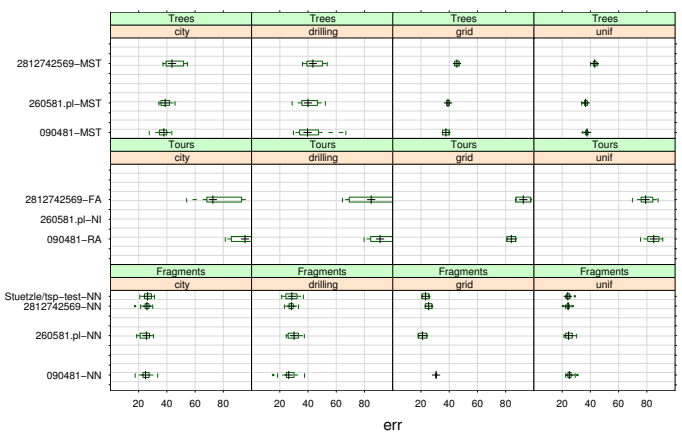
Lin-Kernighan Heuristic. Simulated Annealing

Marco Chiarandini

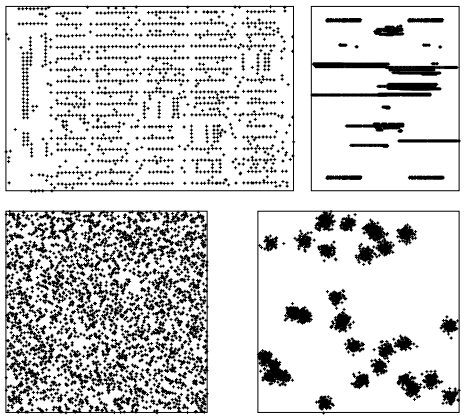
Outline

1. Competition
2. Variable Depth Search
3. Simulated Annealing

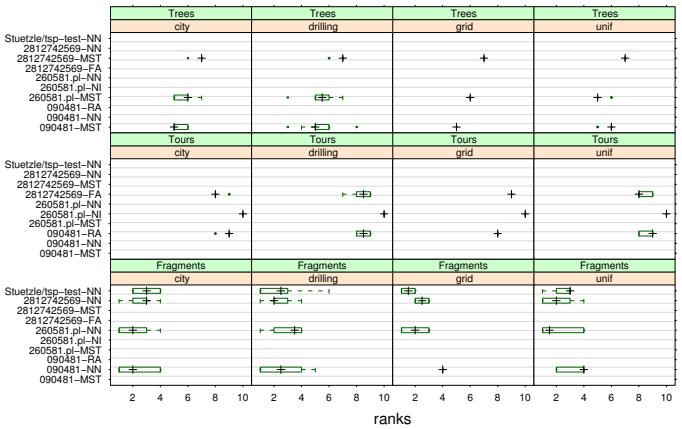
Results - Boxplots of Errors



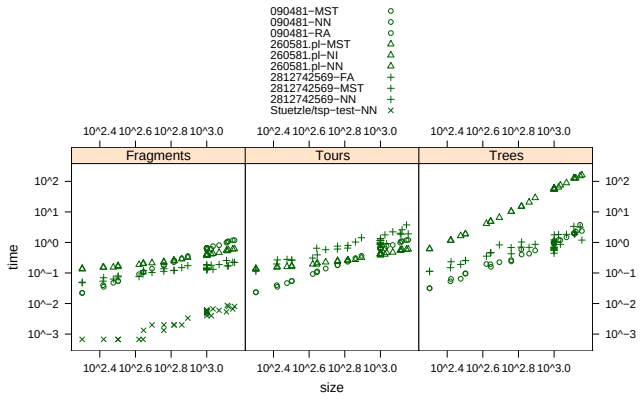
TSP: Benchmark Instances, Examples



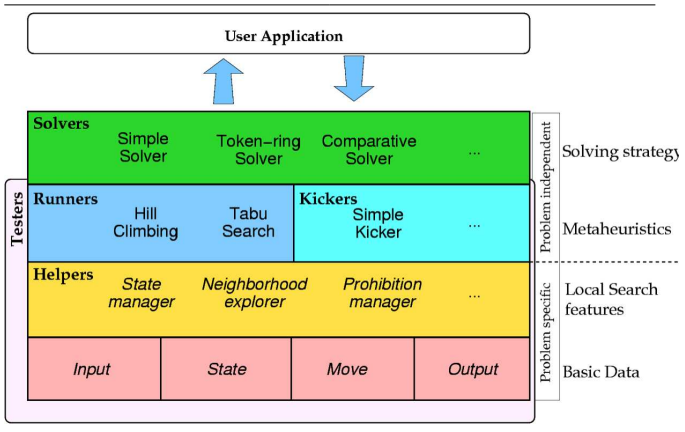
Results - Boxplots of Ranks



Results - Scatter Plots: size vs time



Software Framework for LS Methods



From EasyLocal++ by Schaerf and Di Gaspero (2003).

Variable Depth Search

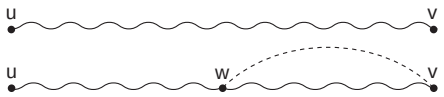
- **Key idea:** *Complex steps* in large neighborhoods = variable-length sequences of *simple steps* in small neighborhood.
- Use various *feasibility restrictions* on selection of simple search steps to limit time complexity of constructing complex steps.
- Perform Iterative Improvement w.r.t. complex steps.

Variable Depth Search (VDS):

determine initial candidate solution s
$\hat{t} := s$
While s is not locally optimal:
  Repeat:
    select best feasible neighbor t
    If $g(t) < g(\hat{t})$: $\hat{t} := t$
  Until construction of complex step has been completed
  $s := \hat{t}$

Example: The Lin-Kernighan (LK) Algorithm for the TSP (1)

- Complex search steps correspond to sequences of 2-exchange steps and are constructed from sequences of *Hamiltonian paths*
- *δ-path:* Hamiltonian path $p + 1$ edge connecting one end of p to interior node of p ('lasso' structure):

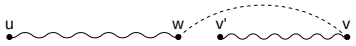


Basic LK exchange step:

- Start with Hamiltonian path $(u, \ldots, v)$:

- Obtain δ-path by adding an edge (v, w):

- Break cycle by removing edge (w, v'):



- *Note:* Hamiltonian path can be completed into Hamiltonian cycle by adding edge (v', u):

Construction of complex LK steps:

1. start with current candidate solution (Hamiltonian cycle) s; set $t^* := s$; set $p := s$
2. obtain δ-path p' by replacing one edge in p
3. consider Hamiltonian cycle t obtained from p by (uniquely) defined edge exchange
4. if $w(t) < w(t^*)$ then set $t^* := t$; $p := p'$; go to step 2
5. else accept t^* as new current candidate solution s

Note: This can be interpreted as sequence of 1-exchange steps that alternate between δ-paths and Hamiltonian cycles.

Additional mechanisms used by LK algorithm:

- *Pruning exact rule:* If a sequence of numbers has a positive sum, there is a cyclic permutation of these numbers such that every partial sum is positive.
  $\Rightarrow$ need to consider only gains whose partial sum is always positive
- *Tabu restriction:* Any edge that has been added cannot be removed and any edge that has been removed cannot be added in the same LK step.
- *Note:* This limits the number of simple steps in a complex LK step.
- *Limited form of backtracking* ensures that local minimum found by the algorithm is optimal w.r.t. standard 3-exchange neighborhood
- (For further details, see article)

Note:

Variable depth search algorithms have been very successful for other problems, including:

- the Graph Partitioning Problem [Kernighan and Lin, 1970];
- the Unconstrained Binary Quadratic Programming Problem [Merz and Freisleben, 2002];
- the Generalized Assignment Problem [Yagiura *et al.*, 1999].

'Simple' LS Methods

Goal:

Effectively escape from local minima of given evaluation function.

General approach:

For fixed neighborhood, use step function that permits *worsening search steps*.

Specific methods:

- Randomized Iterative Improvement
- Probabilistic Iterative Improvement
- Simulated Annealing
- Tabu Search
- Dynamic Local Search

Randomized Iterative Improvement

Key idea: In each search step, with a fixed probability perform an uninformed random walk step instead of an iterative improvement step.

Randomized Iterative Improvement (RII):

determine initial candidate solution s
While termination condition is not satisfied:
  With probability *wp*:
    choose a neighbor s' of s uniformly at random
  Otherwise:
    choose a neighbor s' of s such that $g(s') < g(s)$ or,
    if no such s' exists, choose s' such that $g(s')$ is minimal
  $s := s'$

Note:

- No need to terminate search when local minimum is encountered
  Instead: Bound number of search steps or CPU time from beginning of search or after last improvement.
- Probabilistic mechanism permits arbitrary long sequences of random walk steps
  Therefore: When run sufficiently long, RII is guaranteed to find (optimal) solution to any problem instance with arbitrarily high probability.
- A variant of RII has successfully been applied to SAT (GWSAT algorithm)
- A variant of GUWSAT, GWSAT [Selman et al., 1994], was at some point state-of-the-art for SAT.
- Generally, RII is often outperformed by more complex LS methods.

Example: Randomized Iterative Improvement for GCP

procedure *GUWGCP*$(F, wp, maxSteps)$
input: *a graph* G *and* k, *probability wp, integer maxSteps*
output: *a proper coloring* φ *for* G **or** $\emptyset$
choose coloring φ of G uniformly at random;
steps := 0;
while not(φ is not proper) **and** (*steps* < *maxSteps*) **do**
  with probability *wp* **do**
    select v in V and c in Γ uniformly at random;
  otherwise
    select v and c in V^c and Γ uniformly at random from those that decrease of number of unsat. edge constr. is max.;
  change color of v in φ;
  steps := *steps*+1;
end
if φ is proper for G **then return** φ
else return $\emptyset$
end
end *GUWGCP*

Probabilistic Iterative Improvement

Key idea: Accept worsening steps with probability that depends on respective deterioration in evaluation function value:
bigger deterioration ≅ smaller probability

Realization:

- Function $p(g, s)$: determines probability distribution over neighbors of s based on their values under evaluation function g.
- Let $step(s)(s') := p(g, s)(s')$.

Note:

- Behavior of PII crucially depends on choice of p.
- II and RII are special cases of PII.

Example: Metropolis PII for the TSP

- **Search space** S**:** set of all Hamiltonian cycles in given graph G.
- **Solution set:** same as S
- **Neighborhood relation** $N(s)$**:** 2-edge-exchange
- **Initialization:** an Hamiltonian cycle uniformly at random.
- **Step function:** implemented as 2-stage process:
  1. select neighbor $s' \in N(s)$ uniformly at random;
  2. accept as new search position with probability:

$$p(T, s, s') := \begin{cases} 1 & \text{if } f(s') \leq f(s) \\ \exp(\frac{f(s)-f(s')}{T}) & \text{otherwise} \end{cases}$$

  (*Metropolis condition*), where *temperature* parameter T controls likelihood of accepting worsening steps.
- **Termination:** upon exceeding given bound on run-time.

Inspired by statistical mechanics in matter physics:

- candidate solutions ≅ states of physical system
- evaluation function ≅ thermodynamic energy
- globally optimal solutions ≅ ground states
- parameter T ≅ physical temperature

Note: In physical process (*e.g.*, annealing of metals), perfect ground states are achieved by very slow lowering of temperature.

Simulated Annealing

Key idea: Vary temperature parameter, *i.e.*, probability of accepting worsening moves, in Probabilistic Iterative Improvement according to *annealing schedule* (aka *cooling schedule*).

Simulated Annealing (SA):
determine initial candidate solution s
set initial temperature T according to *annealing schedule*
While termination condition is not satisfied:
  While maintain same temperature T according to *annealing schedule*:
    probabilistically choose a neighbor s' of s using *proposal mechanism*
    If s' satisfies probabilistic *acceptance criterion* (depending on T): $s := s'$
  update T according to *annealing schedule*

Note:

- 2-stage step function based on
  - proposal mechanism (often uniform random choice from $N(s)$)
  - acceptance criterion (often *Metropolis condition*)
- Annealing schedule (function mapping run-time t onto temperature $T(t)$):
  - initial temperature T_0 (may depend on properties of given problem instance)
  - temperature update scheme (*e.g.*, linear cooling: $T_{i+1} = T_0(1 - i/I_{max})$, geometric cooling: $T_{i+1} = \alpha \cdot T_i$)
  - number of search steps to be performed at each temperature (often multiple of neighborhood size)
- Termination predicate: often based on *acceptance ratio*, *i.e.*, ratio of proposed *vs* accepted steps.

Example: Simulated Annealing for the TSP

Extension of previous PII algorithm for the TSP, with

- *proposal mechanism:* uniform random choice from 2-exchange neighborhood;
- *acceptance criterion:* Metropolis condition (always accept improving steps, accept worsening steps with probability $\exp[(f(s) - f(s'))/T]$);
- *annealing schedule*: geometric cooling $T := 0.95 \cdot T$ with $n \cdot (n-1)$ steps at each temperature (n = number of vertices in given graph), T_0 chosen such that 97% of proposed steps are accepted;
- *termination:* when for five successive temperature values no improvement in solution quality and acceptance ratio < 2%.

Improvements:

- neighborhood pruning (*e.g.*, candidate lists for TSP)
- greedy initialization (*e.g.*, by using NNH for the TSP)
- *low temperature starts* (to prevent good initial candidate solutions from being too easily destroyed by worsening steps)

1. Call READ_INSTANCE() to read input, compute an upper bound *c** on the optimal solution value, and return the average neighborhood size *N*.
2. Call INITIAL_SOLUTION() to generate an initial solution *S* and return *c* = *cost* (*S*).
3. Choose an initial temperature *T* > 0 so that in what follows the *changes*/*trials* ratio starts out approximately equal to *INITPROB*.
4. Set *freezecount* = 0.
5. While *freezecount* < *FREEZE_LIM* (i.e., while not yet "frozen") do the following:
   - 5.1 Set *changes* = *trials* = 0.
     While *trials* < *SIZEFACTOR*·*N* and *changes* < *CUTOFF*·*N*, do the following:
     - 5.1.1 Set *trials* = *trials* + 1.
     - 5.1.2 Call NEXT_CHANGE() to generate a random neighbor *S'* of *S* and return *c'* = *cost* (*S'*).
     - 5.1.3 Let Δ = *c'* − *c*.
     - 5.1.4 If Δ ≤ 0 (downhill move), Set *changes* = *changes* + 1 and *c* = *c'*. Call CHANGE_SOLN() to set *S* = *S'* and, if *S'* is feasible and *cost* (*S'*) < *c**, to set *S** = *S'* and *c** = *cost* (*S'*).
     - 5.1.5 If Δ > 0 (uphill move), Choose a random number *r* in [0,1]. If *r* ≤ $e^{-\Delta/T}$ (i.e., with probability $e^{-\Delta/T}$), Set *changes* = *changes* + 1 and *c* = *c'*. Call CHANGE_SOLN().
   - 5.2 Set *T* = *TEMPFACTOR* · *T* (reduce temperature). If *c** was changed during 5.1, set *freezecount* = 0. If *changes*/*trials* < *MINPERCENT*, set *freezecount* = *freezecount* + 1.
6. Call FINAL_SOLN() to output *S**.

'Convergence' result for SA:

Under certain conditions (extremely slow cooling), any sufficiently long trajectory of SA is guaranteed to end in an optimal solution [Geman and Geman, 1984; Hajek, 1998].

Note:

- Practical relevance for combinatorial problem solving is very limited (impractical nature of necessary conditions)
- In combinatorial problem solving, *ending* in optimal solution is typically unimportant, but *finding* optimal solution during the search is (even if it is encountered only once)!

Markov Chains

A Markov chain is collection of random variables X_t (where the index t runs through 0, 1, ...) having the property that, given the present, the future is conditionally independent of the past.

Formally,

$$P(X_t = j | X_0 = i_0, X_1 = i_1, ..., X_{t-1} = i_{t-1}) = P(X_t = j | X_{t-1} = i_{t-1})$$

Related Approaches (1)

Noising Method
Perturb the objective function by adding random noise.
The noise is gradually reduced to zero during algorithm's run.

Threshold Method
Removes the probabilistic nature of the acceptance criterion

$$p_k(\Delta(s, s')) = \begin{cases} 1 & \Delta(s, s') \leq Q_k \\ 0 & \text{otherwise} \end{cases}$$

Q_k deterministic, non-increasing step function in k.
Suggested: $Q_k = Q_0(1 - i/I_{MAX})$

Related Approaches (2)

Critics to SA:
The annealing schedule strongly depends on

- the time bound
- the search landscape and hence on the single instance

Evidence that there are search landscapes for which optimal annealing schedules are non-monotone [Hajek and Sasaki, Althofer and Koschnick, Hu, Kahng and Tsao].

Old Bachelor Acceptance
Dwindling expectations

$$Q_{i+1} = \begin{cases} Q_i + incr(Q_i) & \text{if failed acceptance of } s' \\ Q_i - decr(Q_i) & \text{if } s' \text{ accepted} \end{cases}$$

- $decr(Q_i) = incr(Q_i) = T_0/M$
- $Q_i = \left((\frac{age}{a})^b - 1\right) \cdot \Delta \cdot \left(1 - \frac{i}{M}\right)^c$
- ... (self-tuning, non-monotonic)

Quadratic Assignment Problem

- *Given:* n locations with a matrix $\mathbf{A} = [a_{ij}] \in \Re^{n \times n}$ of distances and n objects with a matrix $\mathbf{B} = [b_{kl}] \in \Re^{n \times n}$ of flows between them
- *Task:* Find the assignment φ of objects to locations that minimize the sum of product between flows and distances, ie,

$$f(\varphi) = \sum b_{ij} a_{\varphi(i)\varphi(j)}$$

Applications: hospital layout; keyboard layout

Example: QAP

$$\mathbf{A} = \begin{pmatrix} 0 & 4 & 3 & 2 & 1 \\ 4 & 0 & 3 & 2 & 1 \\ 3 & 3 & 0 & 2 & 1 \\ 2 & 2 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 3 & 4 \\ 2 & 2 & 0 & 3 & 4 \\ 3 & 3 & 3 & 0 & 4 \\ 4 & 4 & 4 & 4 & 0 \end{pmatrix}$$

The optimal solution is $\varphi = (1, 2, 3, 4, 5)$, that is,
facility 1 is assigned to location 1,
facility 2 is assigned to location 2, etc.

The value of $f(\varphi)$ is 100.

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
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $init()(\varphi') := 1/|S|$ for all colorings φ'
- **step function**:
  - **evaluation function:** $g(\varphi) :=$ number of edges in G whose ending vertices are assigned the same color under assignment φ (*Note:* $g(\varphi) = 0$ iff φ 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.