DM811
HEURISTICS AND LOCAL SEARCH ALGORITHMS FOR COMBINATORIAL OPTIMZATION

Lecture 8

Efficient Local Search

Marco Chiarandini

slides in part based on
`http://www.sls-book.net/`
H. Hoos and T. Stützle, 2005

Outline

Steiner Tree

Input: A graph $G = (V, E)$, a weight function $\omega : E \mapsto \mathbf{N}$, and a subset $U \subseteq V$.
Task: Find a Steiner tree, that is, a subtree $T = (V_T, E_T)$ of G that includes all the vertices of U and such that the sum of the weights of the edges in the subtree is minimal.

Outline

Efficiency vs Effectiveness

The performance of local search is determined by:

1. quality of local optima (effectiveness)
2. time to reach local optima (efficiency):
   A. time to move from one solution to the next
   B. number of solutions to reach local optima

Note:

- Local minima depend on g and neighborhood function $\mathcal{N}$.
- Larger neighborhoods $\mathcal{N}$ induce
  - neighborhood graphs with smaller diameter;
  - fewer local minima.

Ideal case: exact neighborhood, *i.e.*, neighborhood function 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 (to be assessed experimentally):

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

Speedups Techniques for Efficient Neighborhood Search

1) Incremental updates

2) Neighborhood pruning

Speedups in Neighborhood Examination

1) 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, $f(s)$ can be efficiently calculated from $f(s')$ 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.

2) Neighborhood Pruning

- **Idea:** Reduce size of neighborhoods by excluding neighbors that are likely (or guaranteed) not to yield improvements in f.
- **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.

Graph Coloring

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 function** $\mathcal{N}$**:** 1-exchange neighborhood
- **memory:** not used, *i.e.*, $M := \{0\}$
- **initialization:** uniform random choice from S, *i.e.*, $\texttt{init}\{\emptyset, \varphi'\} := 1/|S|$ 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.*, $\texttt{step}\{\varphi, \varphi'\} := 1/|I(\varphi)|$ if $s' \in I(\varphi)$, and 0 otherwise, where $I(\varphi) := \{\varphi' \mid \mathcal{N}(\varphi, \varphi') \wedge g(\varphi') < g(\varphi)\}$
- **termination**: when no improving neighbor is available

Local Search for 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

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

Look at implementation of local search for TSP by T. Stützle:
(from `http://www.sls-book.net/implementations.html`)

File: `http://www.imada.sdu.dk/~marco/Teaching/Fall2008/DM811/Lab/ls.c`

```
two_opt_b(tour);
two_opt_f(tour);
two_opt_best(tour);
two_opt_first(tour);
three_opt_first(tour);
```

LKH Helsgaun's implementation
`http://www.akira.ruc.dk/~keld/research/LKH/` (99 pages report)

Table 17.1 Cases for k-opt moves.

k	No. of Cases
2	1
3	4
4	20
5	148
6	1,358
7	15,104
8	198,144
9	2,998,656
10	51,290,496

[Appelgate Bixby, Chvátal, Cook, 2006]

Table 17.2 Computer-generated source code for k-opt moves.

k	No. of Lines
6	120,228
7	1,259,863
8	17,919,296

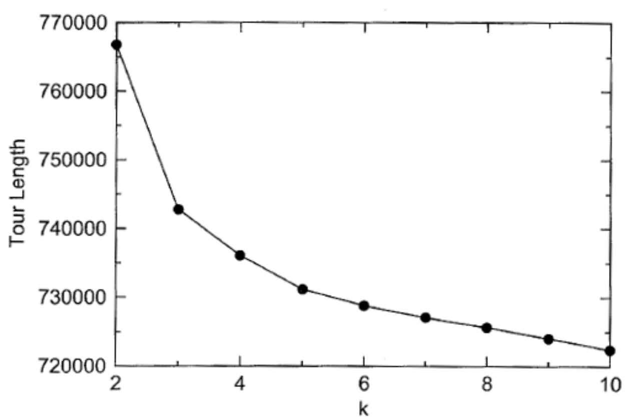
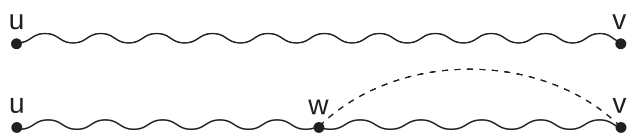


Figure 17.1 k-opt on a 10,000-city Euclidean TSP.

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

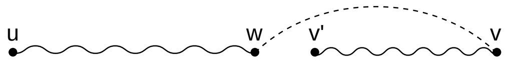


Basic LK exchange step:

- 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 remains 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 original article)

TSP data structures

Tour representation:

- `reverse`(a, b)
- `succ`
- `prec`
- `sequence(a,b,c)` – check whether b is within a and b

Possible choices:

- $|V| < 1.000$ array for π and π^{-1}
- $|V| < 1.000.000$ two level tree
- $|V| > 1.000.000$ splay tree

Moreover static data structure:

- priority lists
- k-d trees

SMTWTP

- Interchange: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - first-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$

Two-Dimensional Packing Problems

Two dimensional bin packing

Given: A set $L = (a_1, a_2, \ldots, a_n)$ of n rectangular *items*, each with a width w_j and a height h_j and an unlimited number of identical rectangular bins of width W and height H.
Task: Allocate all the items into a minimum number of bins, such that the original orientation is respected (no rotation of the items is allowed).

Two dimensional strip packing

Given: A set $L = (a_1, a_2, \ldots, a_n)$ of n rectangular *items*, each with a width w_j and a height h_j and a bin of width W and infinite height (*a strip*).
Task: Allocate all the items into the strip by minimizing the used height and such that the original orientation is respected (no rotation of the items is allowed).

Two dimensional cutting stock

Each item has a profit $p_j > 0$ and the task is to select a subset of items to be packed in a single finite bin that maximizes the total selected profit.

Quadratic Assignment Problem

- **Given:** n locations with a matrix $D = [d_{ij}] \in \mathbb{R}^{n \times n}$ of distances and n units with a matrix $F = [f_{kl}] \in \mathbb{R}^{n \times n}$ of flows between them
- **Task:** Find the assignment σ of units to locations that minimize the sum of product between flows and distances, ie,

$$\min_{\sigma \in \Sigma} \sum_{i,j} f_{ij} d_{\sigma(i)\sigma(j)}$$

Applications: hospital layout; keyboard layout

Example: QAP

$$D = \begin{pmatrix} 0 & 4 & 3 & 2 & 1 \\ 4 & 0 & 3 & 2 & 1 \\ 3 & 3 & 0 & 2 & 1 \\ 2 & 2 & 2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 3 & 4 \\ 2 & 2 & 0 & 3 & 4 \\ 3 & 3 & 3 & 0 & 4 \\ 4 & 4 & 4 & 4 & 0 \end{pmatrix}$$

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

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

Delta evaluation

Evaluation of 2-exchange $\{r, s\}$ can be done in $O(n)$

$$
\begin{aligned}
\Delta(\psi, r, s) = {} & b_{rr} \cdot (a_{\psi_s \psi_s} - a_{\psi_r \psi_r}) + b_{rs} \cdot (a_{\psi_s \psi_r} - a_{\psi_r \psi_s}) + \\
& b_{sr} \cdot (a_{\psi_r \psi_s} - a_{\psi_s \psi_r}) + b_{ss} \cdot (a_{\psi_r \psi_r} - a_{\psi_s \psi_s}) + \\
\sum_{k=1, k \neq r,s}^{n} & (b_{kr} \cdot (a_{\psi_k \psi_s} - a_{\psi_k \psi_r}) + b_{ks} \cdot (a_{\psi_k \psi_r} - a_{\psi_k \psi_s}) + \\
& b_{rk} \cdot (a_{\psi_s \psi_k} - a_{\psi_r \psi_k}) + b_{sk} \cdot (a_{\psi_r \psi_k} - a_{\psi_s \psi_k}))
\end{aligned}
$$