DM63
HEURISTICS FOR
COMBINATORIAL OPTIMIZATION

Lecture 4
Local Search.
Implementation Issues and Search Landscapes.

Marco Chiarandini

Outline

1. Local Search
Implementation Examples
Neighborhoods Representations
Other Examples

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).

Iterative Improvement (2 OPT)

procedure TSP-2opt-first(s)
input: an initial candidate tour s ∈ S(∈)
output: a local optimum s ∈ S(π)
∆ = 0;
do
Improvement:=FALSE;
for i = 1 to n - 2 do
if i = 1 then n' = n - 1 else n' = n
for j = i + 2 to n' do
∆_{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 ∆_{ij} < 0 then
Update Tour(s_{ij});
Improvement:=TRUE;
end
end
until Improvement==TRUE;
end

DM63 – Heuristics for Combinatorial Optimization Problems

Note:
▶ Local minima depend on g and neighborhood relation N.
▶ Larger neighborhoods N induce
every 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.

Example: Random-order first improvement for the TSP
▶ Given: TSP instance G with vertices v_1, v_2, ..., v_n.
▶ search space: Hamiltonian cycles in G;
use standard 2-exchange neighborhood
▶ Initialization:
search position := fixed canonical path (v_1, v_2, ..., v_n, v_1)
P := random permutation of {1, 2, ..., n}
▶ Search steps: determined using first improvement w.r.t. f(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.

TSP data structures
- Tour representation:
  - determine pos of \( v \) in \( \pi \)
  - determine succ and prec
  - check whether \( u_i \) is visited between \( u_i \) and \( v_j \)
  - execute a k-exchange (reversal)
- Possible choices:
  - \( |V| < 1.000 \) array for \( \pi \) and \( \pi^{-1} \)
  - \( |V| < 1.000.000 \) two level tree
  - \( |V| > 1.000.000 \) splay tree
- Moreover static data structure:
  - priority lists
  - k-d trees

Solution Representations and Neighborhoods
Three different types of solution representations:
- Permutation
  - linear permutation: Single Machine Total Weighted Tardiness Problem
  - circular permutation: Traveling Salesman Problem
- Assignment: Graph Coloring Problem, SAT, CSP
- Set, Partition: Max Independent Set

A neighborhood function \( \mathcal{N} : S \rightarrow S \times S \) is also defined through an operator. An operator \( \Delta \) is a collection of operator functions \( \delta : S \rightarrow S \) such that
\[ s' \in \mathcal{N}(s) \iff \exists \delta \in \Delta \delta(s) = s' \]

Neighborhood Operators for Linear Permutations
- Swap operator
  \[ \Delta_S = \{ \delta^i_j | 1 \leq i < n \} \]
  \[ \delta^i_j(\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n) = (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_i, \pi_{i-1}, \ldots, \pi_n) \]
- Interchange operator
  \[ \Delta_I = \{ \delta'^i_j | 1 \leq i < j \leq n \} \]
  \[ \delta'^i_j(\pi) = (\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_j, \pi_{i+1}, \ldots, \pi_n) \]
- Insert operator
  \[ \Delta_I = \{ \delta''^i_j | 1 \leq i \leq n, 1 \leq j < n, j \neq i \} \]
  \[ \delta''^i_j(\pi) = \begin{cases} (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n) & i < j \\ (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n) & i > j \end{cases} \]

Neighborhood Operators for Circular Permutations
- Reversal (2-edge-exchange)
  \[ \Delta_R = \{ \delta^i_j | 1 \leq i < j \leq n \} \]
  \[ \delta^i_j(\pi) = \pi_1 \pi_2 \ldots \pi_{i-1} \pi_{i+1} \pi_{i+2} \ldots \pi_n \]
- Block moves (3-edge-exchange)
  \[ \Delta_B = \{ \delta^i_k | 1 \leq i < j < k \leq n \} \]
  \[ \delta^i_k(\pi) = \pi_1 \pi_2 \ldots \pi_{i-1} \pi_i \pi_{i+1} \pi_i \pi_{i+2} \ldots \pi_k \pi_{k+1} \pi_{k+2} \ldots \pi_n \]
- Short block move (Or-edge-exchange)
  \[ \Delta_B = \{ \delta''^i_j | 1 \leq i < j \leq n \} \]
  \[ \delta''^i_j(\pi) = \pi_1 \pi_2 \ldots \pi_{i-1} \pi_i \pi_{i+1} \pi_i \pi_{i+1} \pi_{i+2} \ldots \pi_k \pi_{k+1} \pi_{k+2} \ldots \pi_n \]

Discuss implementation of local search for TSP by Stützle:
http://www.sls-book.net/implementations.html
- two_opt_b(tour);
- two_opt_f(tour);
- two_opt_best(tour);
- two_opt_first(tour);
- three_opt_first(tour);

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
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
Neighborhood Operators for Assignments

An assignment can be represented as a mapping
\[ \sigma : \{X_1 \ldots X_n\} \to \{v : v \in D, |D| = k\} : \]
\[ \sigma = \{X_i = v_i, X_j = v_j \ldots\} \]

One exchange operator
\[ \Delta_{1E} = \{\delta_{1E}^i|1 \leq i \leq n, 1 \leq l \leq k\} \]
\[ \delta_{1E}^i(\sigma) = \{\sigma : \sigma'(X_i) = v_i and \sigma'(X_j) = \sigma(X_j) \forall j \neq i\} \]

Two exchange operator
\[ \Delta_{2E} = \{\delta_{2E}^i|1 < j < n\} \]
\[ \delta_{2E}^i(\sigma) = \{\sigma : \sigma'(X_i) = \sigma(X_j), \sigma'(X_j) = \sigma(X_i) and \sigma'(X_l) = \sigma(X_l) \forall i, j\} \]

Other Examples

Delta evaluations and neighborhood examinations in:
- Permutations
- SMTWTP
- Assignments
- SAT
- Sets
- Max Independent Set

SMTWTP
- Interchange: size \( \binom{n}{2} \) and \( O(|i - j|) \) evaluation each
  - first-improvement: \( p_m, n \)
  - possible use of auxiliary data structure to speed up the computation
- first-improvement: \( p_j, n \)
  - 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 \( O(n^2) \)

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., \( init(0, \varphi') := 1/|S| \)
  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 \( \varphi' \in I(\varphi) \),
    and 0 otherwise, where \( I(\varphi) := \{\varphi' | X(\varphi, \varphi') \wedge g(\varphi') < g(\varphi)\} \)
- termination: when no improving neighbor is available
  \( i.e., terminate(\varphi, T) := 1 \) if \( I(\varphi) = 0 \), and 0 otherwise.