DM811

Heuristics for Combinatorial Optimization

Lecture 19

Very Large Scale Neighborhoods

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Course Overview

1. Combinatorial Optimization, Methods and Models
2. General overview
3. Solver System and Working Environment
4. Construction Heuristics
5. Local Search: Components, Basic Algorithms
6. Local Search: Neighborhoods and Search Landscape
7. Efficient Local Search: Incremental Updates and Neighborhood Pruning
8. Stochastic Local Search & Metaheuristics
9. Methods for the Analysis of Experimental Results
10. Configuration Tools: F-race
11. Very Large Scale Neighborhoods

Examples: GCP, CSP, TSP, SAT, MaxIndSet, SMTWP, Steiner Tree, p-median, set covering, bin packing

Very Large Scale Neighborhoods

Small neighborhoods:

- might be short-sighted
- need many steps to traverse the search space

Large neighborhoods

- introduce large modifications to reach higher quality solutions
- allows to traverse the search space in few steps

Very Large Scale Neighborhoods

Small neighborhoods:

- might be short-sighted
- need many steps to traverse the search space

Large neighborhoods

- introduce large modifications to reach higher quality solutions
- allows to traverse the search space in few steps

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

Very large scale neighborhood search

1. define an exponentially large neighborhood
   (though, $O(n^3)$ might already be large)
2. define a polynomial time search algorithm to search the neighborhood
   (= solve the neighborhood search problem, NSP)
   - exactly (leads to a best improvement strategy)
   - heuristically (some improving moves might be missed)

Examples of VLSN Search

[Ahuja, Ergun, Orlin, Punnen, 2002]

- based on concatenation of simple moves
  - Variable Depth Search (TSP, GP)
  - Ejection Chains
- based on Dynamic Programming or Network Flows
  - Dynasearch (ex. SMTWTP)
  - Weighted Matching based neighborhoods (ex. TSP)
  - Cyclic exchange neighborhood (ex. VRP)
  - Shortest path
- based on polynomially solvable special cases of hard combinatorial optimization problems
  - Pyramidal tours
  - Halin Graphs

➤ Idea: turn a special case into a neighborhood
VLSN allows to use the literature on polynomial time algorithms

Outline

Variable Depth Search

- **Key idea:** *Complex steps* in large neighborhoods = variable-length sequences of *simple steps* in small neighborhood.

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.

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

- **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

- **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 **do**
  repeat
    select best feasible neighbor t
    if $g(t) < g(\hat{t})$ **then** $\hat{t} := t$
    $s := \hat{t}$
  until construction of complex step has been completed ;

Graph Partitioning

Graph Partitioning

Given: $G = (V, E)$, weighted function $\omega : V \to \mathbf{R}$, a positive number p: $0 < w_i \leq p, \forall i$ and a connectivity matrix $C = [c_{ij}] \in \mathbf{R}^{|V| \times |V|}$.

Task: A k-partition of G, $V_1, V_2, \ldots, V_k$: $\bigcup_{i=1}^{n} V_i = G$ such that:

- it is admissible, ie, $|V_i| \leq p$ for all i and
- it has minimum cost, ie, the sum of c_{ij}, i, j that belong to different subsets is mimimal

VLSN for the Traveling Salesman Problem

- *k*-exchange heuristics
  - 2-opt [Flood, 1956, Croes, 1958]
  - 2.5-opt or 2H-opt
  - Or-opt [Or, 1976]
  - 3-opt [Block, 1958]
  - *k*-opt [Lin 1965]

VLSN for the Traveling Salesman Problem

- k-exchange heuristics
  - 2-opt [Flood, 1956, Croes, 1958]
  - 2.5-opt or 2H-opt
  - Or-opt [Or, 1976]
  - 3-opt [Block, 1958]
  - k-opt [Lin 1965]

- complex neighborhoods
  - Lin-Kernighan [Lin and Kernighan, 1965]
  - Helsgaun's Lin-Kernighan
  - Dynasearch
  - Ejection chains approach

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*

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

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

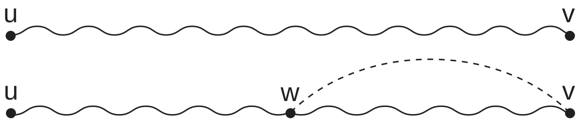
- Complex search steps correspond to sequences of 2-exchange steps and are constructed from sequences of *Hamiltonian paths*

- δ*-path:* Hamiltonian path p + 1 edge connecting one end of p to interior node of p

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 $(u, \ldots, v)$:

Basic LK exchange step:

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

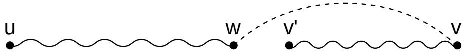
- Obtain δ-path by adding an edge (v, w):

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'):

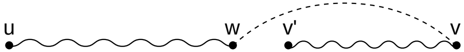


Basic LK exchange step:

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

- 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 $p := s$

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

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

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

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
   else accept t^* as new current candidate solution s

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
   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.

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.
  ➡ 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)

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

Outline

Ejection Chains

- Attempt to use large neighborhoods without examining them exhaustively
- Sequences of successive steps each influenced by the precedent and determined by myopic choices
- Limited in length
- Local optimality in the large neighborhood is not guaranteed.

Ejection Chains

- Attempt to use large neighborhoods without examining them exhaustively
- Sequences of successive steps each influenced by the precedent and determined by myopic choices
- Limited in length
- Local optimality in the large neighborhood is not guaranteed.

Example (on TSP):
successive 2-exchanges where each exchange involves one edge of the previous exchange

Example (on GCP):
successive 1-exchanges: a vertex v_1 changes color from $\varphi(v_1) = c_1$ to c_2, in turn forcing some vertex v_2 with color $\varphi(v_2) = c_2$ to change to another color c_3 (which may be different or equal to c_1) and again forcing a vertex v_3 with color $\varphi(v_3) = c_3$ to change to color c_4.

Outline

Dynasearch

- Iterative improvement method based on building complex search steps from combinations of mutually independent search steps
- Mutually independent search steps do not interfere with each other wrt effect on evaluation function and feasibility of candidate solutions.

Example: Independent 2-exchange steps for the TSP:

Therefore: Overall effect of complex search step = sum of effects of constituting simple steps;
complex search steps maintain feasibility of candidate solutions.

- **Key idea:** Efficiently find optimal combination of mutually independent simple search steps using *Dynamic Programming*.

Outline

1. Variable Depth Search
2. Ejection Chains
3. Dynasearch
4. Weighted Matching Neighborhoods
5. Cyclic Exchange Neighborhoods

Weighted Matching Neighborhoods

- **Key idea** use basic polynomial time algorithms, example: weighted matching in bipartied graphs, shortest path, minimum spanning tree.
- Neighborhood defined by finding a minimum cost matching on a (non-)bipartite improvement graph

Example (TSP)
Neighborhood: Eject k nodes and reinsert them optimally

Outline

Cyclic Exchange Neighborhoods

- Possible for problems where solution can be represented as form of partitioning

Cyclic Exchange Neighborhoods

- Possible for problems where solution can be represented as form of partitioning
- Definition of a partitioning problem:
  Given: a set W of n elements, a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of subsets of W, such that $W = T_1 \cup \ldots \cup T_k$ and $T_i \cap T_j = \emptyset$, and a cost function $c : \mathcal{T} \rightarrow \mathbf{R}$:
  Task: Find another partition $\mathcal{T}'$ of W by means of single exchanges between the sets such that

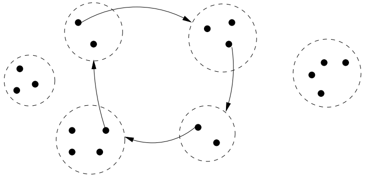
$$\min \sum_{i=1}^{k} c(T_i)$$

Cyclic Exchange Neighborhoods

- Possible for problems where solution can be represented as form of partitioning
- Definition of a partitioning problem:
  Given: a set W of n elements, a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of subsets of W, such that $W = T_1 \cup \ldots \cup T_k$ and $T_i \cap T_j = \emptyset$, and a cost function $c : \mathcal{T} \to \mathbf{R}$:
  Task: Find another partition $\mathcal{T}'$ of W by means of single exchanges between the sets such that

$$\min \sum_{i=1}^{k} c(T_i)$$

- Cyclic exchange:

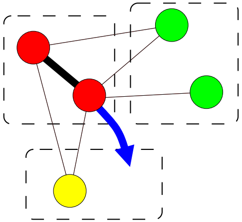


Neighborhood search

- Define an improvement graph
- Solve the relative
  - Subset Disjoint *Negative* Cost Cycle Problem
  - Subset Disjoint *Minimum* Cost Cycle Problem

Example (GCP)

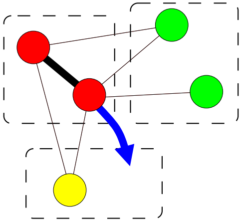
Neighborhood Structures: Very Large Scale Neighborhood



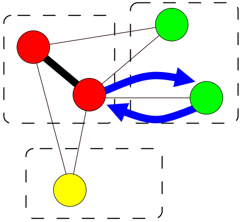
One Exchange

Example (GCP)

Neighborhood Structures: Very Large Scale Neighborhood



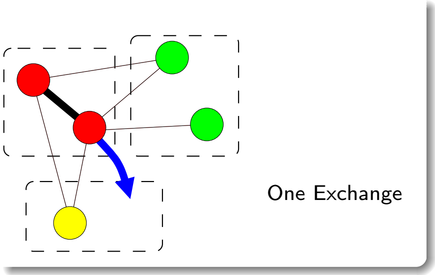
One Exchange

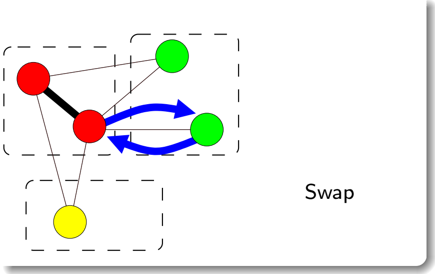


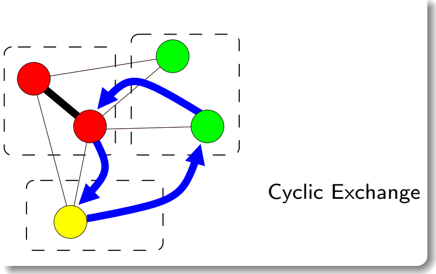
Swap

Example (GCP)

Neighborhood Structures: Very Large Scale Neighborhood

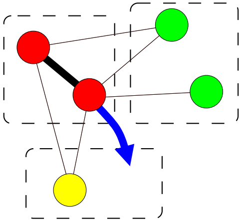




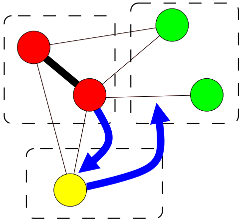


Example (GCP)

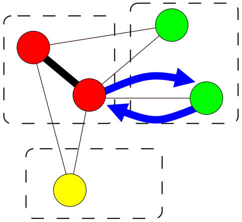
Neighborhood Structures: Very Large Scale Neighborhood



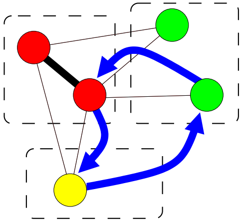
One Exchange



Path Exchange



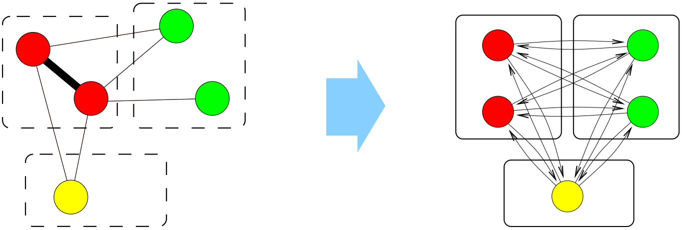
Swap



Cyclic Exchange

Example (GCP)
Examination of the Very Large Scale Neighborhood

Exponential size but can be searched efficiently



Improvement Graph

A Subset Disjoint Negative Cost Cycle Problem in the Improvement Graph can be solved by dynamic programming in $\mathcal{O}(|V|^2 2^k |D'|)$.
Yet, heuristic rules can be adopted to reduce the complexity to $\mathcal{O}(|V'|^2)$