DM841

Discrete Optimization — Heuristics

Local Search Theory

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Neighborhood Operator

Goal: providing a formal description of neighborhood functions for the three main solution representations:

- Permutation
  - linear permutation: Single Machine Total Weighted Tardiness Problem
  - circular permutation: Traveling Salesman Problem
- Assignment: SAT, CSP
- Set, Partition: Max Independent Set

A neighborhood function $N : S \to 2^S$ is also defined through an operator.
An operator Δ is a collection of operator functions $\delta : S \to S$ such that

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

Permutations

S_n indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2 \ldots, n)$ is the identity permutation ι.

If $\pi \in \Pi(n)$ and $1 \leq i \leq n$ then:

- π_i is the element at position i
- $pos_\pi(i)$ is the position of element i

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

The permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')_i = \pi'(\pi(i))$

For each π there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$
$\pi^{-1}(i) = pos_\pi(i)$

$$\Delta_N \subset S_n$$

Linear Permutations

Swap operator

$$\Delta_S = \{\delta_S^i \mid 1 \leq i \leq n\}$$

$$\delta_S^i(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

Interchange operator

$$\Delta_X = \{\delta_X^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

($\equiv$ set of all transpositions)

Insert operator

$$\Delta_I = \{\delta_I^{ij} \mid 1 \leq i \leq n, 1 \leq j \leq n, j \neq i\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i > j \end{cases}$$

Circular Permutations

Reversal (2-edge-exchange)

$$\Delta_R = \{\delta_R^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_R^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1}\pi_j \ldots \pi_i\pi_{j+1} \ldots \pi_n)$$

Block moves (3-edge-exchange)

$$\Delta_B = \{\delta_B^{ijk} \mid 1 \leq i < j < k \leq n\}$$

$$\delta_B^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1}\pi_j \ldots \pi_k\pi_i \ldots \pi_{j-1}\pi_{k+1} \ldots \pi_n)$$

Short block move (Or-edge-exchange)

$$\Delta_{SB} = \{\delta_{SB}^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_{SB}^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1}\pi_j\pi_{j+1}\pi_{j+2}\pi_i \ldots \pi_{j-1}\pi_{j+3} \ldots \pi_n)$$

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

$$\Delta_{1E} = \{\delta_{1E}^{il} \mid 1 \leq i \leq n, 1 \leq l \leq k\}$$

$$\delta_{1E}^{il}(\sigma) = \{\sigma' : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \ \forall j \neq i\}$$

Two-exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_{2E}^{ij}(\sigma) = \{\sigma' : \sigma'(X_i) = \sigma(X_j), \ \sigma'(X_j) = \sigma(X_i) \ \text{ and } \ \sigma'(X_l) = \sigma(X_l) \ \forall l \neq i, j\}$$

Partitioning

An assignment can be represented as a partition of objects selected and not selected
$s : \{X\} \to \{C, \overline{C}\}$ (it can also be represented by a bit string)

One-addition operator

$$\Delta_{1E} = \{\delta_{1E}^{v} \mid v \in \bar{C}\}$$

$$\delta_{1E}^{v}(s) = \left\{s : C' = C \cup v \text{ and } \bar{C}' = \bar{C} \setminus v\right\}$$

One-deletion operator

$$\Delta_{1E} = \{\delta_{1E}^{v} \mid v \in C\}$$

$$\delta_{1E}^{v}(s) = \left\{s : C' = C \setminus v \text{ and } \bar{C}' = \bar{C} \cup v\right\}$$

Swap operator

$$\Delta_{1E} = \{\delta_{1E}^{v} \mid v \in C, u \in \bar{C}\}$$

$$\delta_{1E}^{v}(s) = \left\{s : C' = C \cup u \setminus v \text{ and } \bar{C}' = \bar{C} \cup v \setminus u\right\}$$

Distances

Set of paths in G_N with $s, s' \in S$:

$$\Phi(s,s') = \{(s_1,\ldots,s_h) \mid s_1 = s, s_h = s' \; \forall i : 1 \leq i \leq h-1, \langle s_i, s_{i+1}\rangle \in E(G_N)\}$$

If $\phi = (s_1,\ldots,s_h) \in \Phi(s,s')$ let $|\phi| = h$ be the length of the path; then the distance between any two solutions s, s' is the length of shortest path between s and s' in G_N:

$$d_{G_N}(s,s') = \min_{\phi \in \Phi(s,s')} |\Phi|$$

$\mathtt{diam}(G_N) = \max\{d_{G_N}(s,s') \mid s,s' \in S\}$ (= maximal distance between any two candidate solutions)
(= worst-case lower bound for number of search steps required for reaching (optimal) solutions)

Note: with permutations it is easy to see that:

$$d_{G_N}(\pi,\pi') = d_{G_N}(\pi^{-1}\cdot\pi',\iota)$$

Distances for Linear Permutation Representations

- Swap neighborhood operator
  computable in $O(n^2)$ by the precedence based distance metric:
  $d_S(\pi, \pi') = \#\{\langle i, j\rangle | 1 \leq i < j \leq n, pos_{\pi'}(\pi_j) < pos_{\pi'}(\pi_i)\}$.
  $\texttt{diam}(G_N) = n(n-1)/2$

- Interchange neighborhood operator
  Computable in $O(n) + O(n)$ since $d_X(\pi, \pi') = d_X(\pi^{-1} \cdot \pi', \iota) = n - c(\pi^{-1} \cdot \pi')$
  $c(\pi)$ is the number of disjoint cycles that decompose a permutation.
  $\texttt{diam}(G_{N_X}) = n - 1$

- Insert neighborhood operator
  Computable in $O(n) + O(n\,log(n))$ since $d_I(\pi, \pi') = d_I(\pi^{-1} \cdot \pi', \iota) = n - |lis(\pi^{-1} \cdot \pi')|$ where $lis(\pi)$ denotes the length of the longest increasing subsequence.
  $\texttt{diam}(G_{N_I}) = n - 1$

Distances for Circular Permutation Representations

- Reversal neighborhood operator
  sorting by reversal is known to be NP-hard
  surrogate in TSP: bond distance

- Block moves neighborhood operator
  unknown whether it is NP-hard but there does not exist a proved polynomial-time algorithm

Distances for Assignment Representations

- Hamming Distance

- An assignment can be seen as a partition of n in k mutually exclusive non-empty subsets

  One-exchange neighborhood operator
  The *partition-distance* $d_{1E}(\mathcal{P}, \mathcal{P}')$ between two partitions $\mathcal{P}$ and $\mathcal{P}'$ is the minimum number of elements that must be moved between subsets in $\mathcal{P}$ so that the resulting partition equals $\mathcal{P}'$.

  The partition-distance can be computed in polynomial time by solving an assignment problem. Given the assignment matrix M where in each cell (i, j) it is $|S_i \cap S'_j|$ with $S_i \in \mathcal{P}$ and $S'_j \in \mathcal{P}'$ and defined $A(\mathcal{P}, \mathcal{P}')$ the assignment of maximal sum then it is $d_{1E}(\mathcal{P}, \mathcal{P}') = n - A(\mathcal{P}, \mathcal{P}')$

Example: Search space size and diameter for the TSP

- Search space size = $(n-1)!/2$
- Insert neighborhood
  size = $(n-3)n$
  diameter = $n-2$
- 2-exchange neighborhood
  size = $\binom{n}{2} = n \cdot (n-1)/2$
  diameter in $[n/2, n-2]$
- 3-exchange neighborhood
  size = $\binom{n}{3} = n \cdot (n-1) \cdot (n-2)/6$
  diameter in $[n/3, n-1]$

Let N_1 and N_2 be two different neighborhood functions for the same instance (S, f, π) of a combinatorial optimization problem.

If for all solutions $s \in S$ we have $N_1(s) \subseteq N_2(s)$ then we say that N_2 dominates N_1

Example:

In TSP, 1-insert is dominated by 3-exchange.
(1-insert corresponds to 3-exchange and there are 3-exchanges that are not 1-insert)

Search Landscape

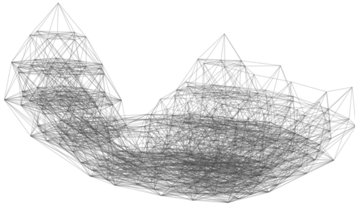
Given:

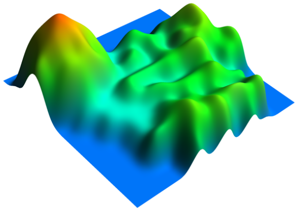
- Problem instance π
- Search space S_π
- Neighborhood function $N : S \subseteq 2^S$
- Evaluation function $f_\pi : S \to \mathrm{R}$

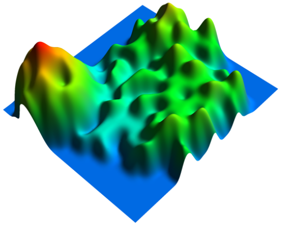
Definition:

The **search landscape** L is the vertex-labeled neighborhood graph given by the triplet $\mathcal{L} = \langle S, N, f \rangle$.

Search Landscape





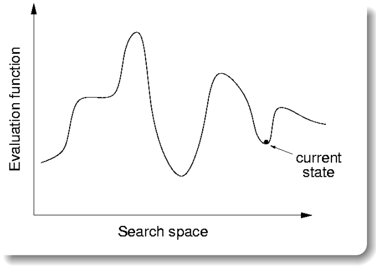


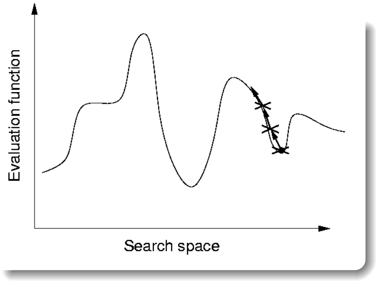
Transition Graph of Iterative Improvement

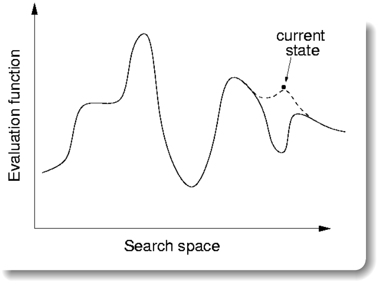
Given $\mathcal{L} = \langle S, N, f \rangle$, the transition graph of iterative improvement is a directed acyclic subgraph obtained from $\mathcal{L}$ by deleting all arcs (i, j) for which it holds that the cost of solution j is worse than or equal to the cost of solution i.

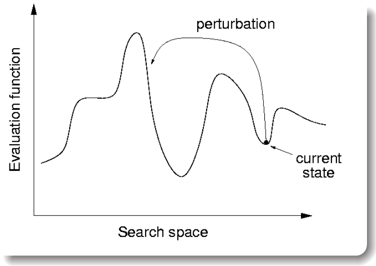
It can be defined for other algorithms as well and it plays a central role in the theoretical analysis of proofs of convergence.

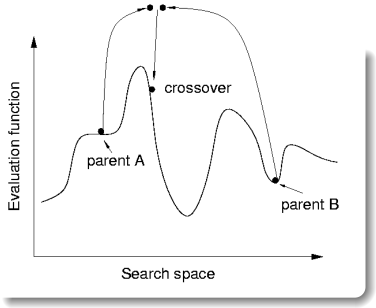
Idealized visualization of landscapes principles











Fundamental Properties

The behavior and performance of an LS algorithm on a given problem instance crucially depends on properties of the respective search landscape.

Simple properties:

- search space size $|S|$
- reachability: solution j is reachable from solution i if neighborhood graph has a path from i to j.
    - strongly connected neighborhood graph
    - weakly optimally connected neighborhood graph
- distance between solutions
- neighborhood size (ie, degree of vertices in neigh. graph)
- cost of fully examining the neighborhood
- relation between different neighborhood functions
  (if $N_1(s) \subseteq N_2(s)$ forall $s \in S$ then N_2 dominates N_1)

Outline

Computational Complexity of LS

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 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 LS

$\mathcal{PLS}$-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.

Outline

Comments to Last Year Submissions

```
- Focus on relevant aspects, not on trivial and known features. For
  example, explaining the Code Framework is not necessary as we all know
  about it. The algorithmic sketch must be on a relevant and original
  procedure. What you choose to describe and to show algorithmically
  will also be used to decide the grade.
- Be formal and do not use terms like "stupid". Do not tell about lack
  of time (everybody always lacks of time anyway).
- Do not make speculations but try to support your claims by
  experimental or analytic evidence.
- Define the notation that you use
- The calculation of the Delta by incremental updates is a requirement
- Recognize the algorithms that you end up implementing and give them
  name.
- Random restart is really a basic algorithm and you have to do better
  than that.
```

- You will get credit for how involved the method is and for the amount of work done.

- Check whether your algorithm has chances for ending in a loop. That would be bad.

- Your algorithms must be 100% reproducible by only reading the report.

- Make a clear list of the algorithms you tested and give names to the algorithms you are describing. In this way it becomes clearer what you are precisely referring to. When writing the name of the algorithms use a different style, for example, sanserif or slanted, etc.

- Remember to give the big O analysis of the main procedures, that is, constructing a solution, initializing the data structures, evaluating a move, deciding a step in the local search and updating the data structures (ie, ensuring invariants).

- Write the report keeping in mind that the reader will be the external censor. He/She is acknowledged about local search and heuristics but has not been at the lectures and hence does not know what we have precisely discussed about.

- Consider carefully the focus of your description and algorithmic sketches. This choice is alone providing to the examiners an indication of the level reached in this course. Reporting general sketches that have been seen in class is not a smart choice. A more appropriate choice is showing the specialized, non-trivial procedures that you have developed, that may indicate the originality and depth of thought in your work.

- Make it possible to distinguish entities in your plots to both readers that will print in colors and to those that will print in black and white.

- Leave a space before opening a parenthesis. Example: "Heuristics(DM841)" is wrong. "Heuristics (DM841)" is correct.