DM811

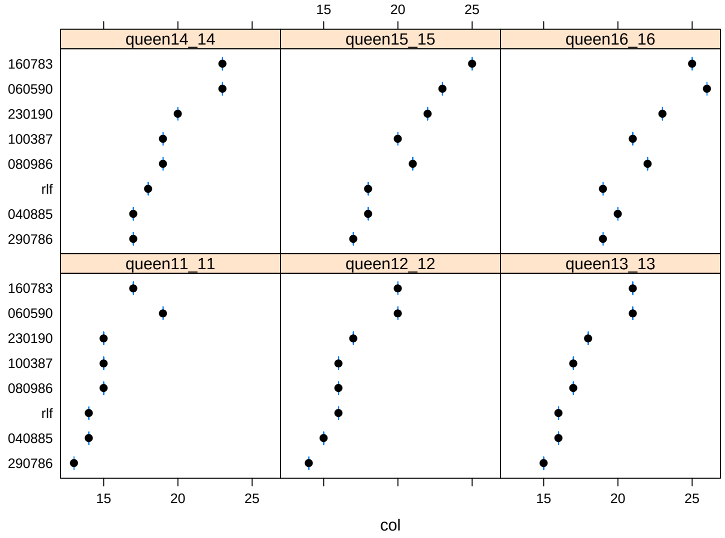
Heuristics for Combinatorial Optimization

Lecture 9
Local Search: Further Deepening Neighborhoods and Landscapes

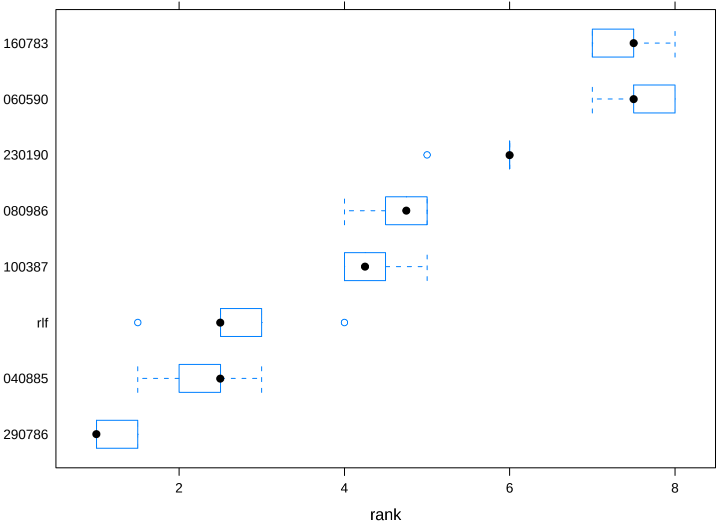
Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

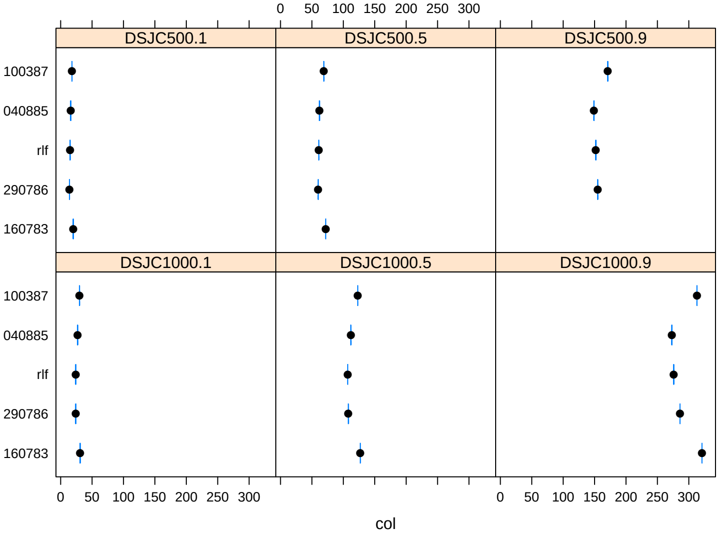
Results Assignment 2



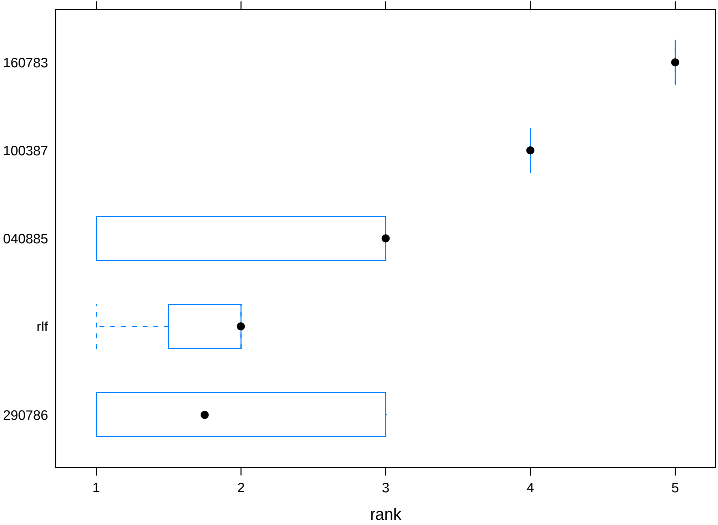
Results Assignment 2



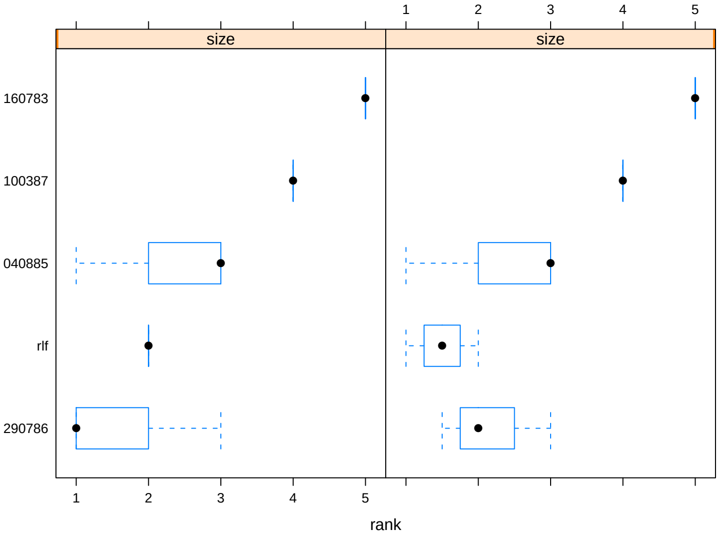
Results Assignment 2



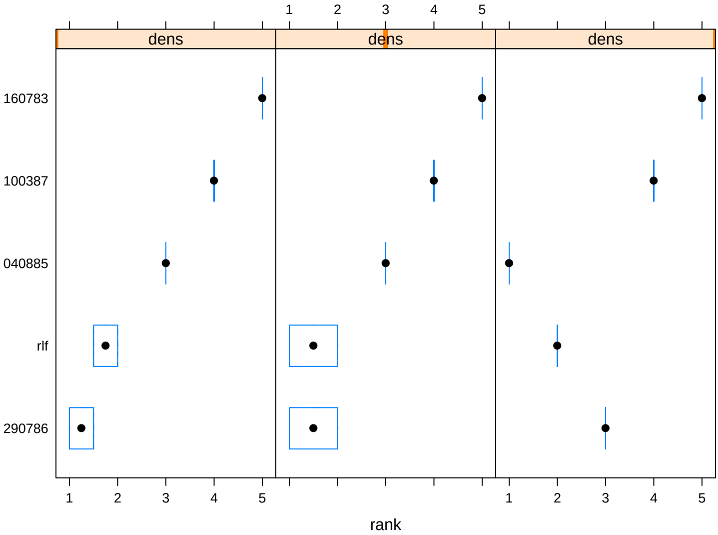
Results Assignment 2



Results Assignment 2



Results Assignment 2



Summary: Local Search Algorithms
(as in [Hoos, Stützle, 2005])

For given problem instance π:

1. search space S_π
2. neighborhood relation $\mathcal{N}_\pi \subseteq S_\pi \times S_\pi$
3. evaluation function $f_\pi : S \to \mathbf{R}$
4. set of memory states M_π
5. initialization function $\texttt{init} : \emptyset \to S_\pi \times M_\pi)$
6. step function $\texttt{step} : S_\pi \times M_\pi \to S_\pi \times M_\pi$
7. termination predicate $\texttt{terminate} : S_\pi \times M_\pi \to \{\top, \bot\}$

Outline

Examples, Resume

- Permutations
  - TSP
  - SMWTP
- Assignments
  - SAT
  - Coloring
  - Parallel machines
- Sets
  - Max Weighted Independent Set
  - Steiner tree

Single Machine Total Weighted Tardiness

Given: a set of n jobs $\{J_1, \dots, J_n\}$ to be processed on a single machine and for each job J_i a processing time p_i, a weight w_i and a due date d_i.

Task: Find a schedule that minimizes
the total weighted tardiness $\sum_{i=1}^{n} w_i \cdot T_i$
where $T_i = \max\{C_i - d_i, 0\}$ (C_i completion time of job J_i)

Example:

Job	J_1	J_2	J_3	J_4	J_5	J_6
Processing Time	3	2	2	3	4	3
Due date	6	13	4	9	7	17
Weight	2	3	1	5	1	2

Sequence $\phi = J_3, J_1, J_5, J_4, J_1, J_6$

Job	J_3	J_1	J_5	J_4	J_2	J_6
C_i	2	5	9	12	14	17
T_i	0	0	2	3	1	0
$w_i \cdot T_i$	0	0	2	15	3	0

Escaping Local Optima

Possibilities:

Escaping Local Optima

Possibilities:

- Enlarge the neighborhood

Escaping Local Optima

Possibilities:

- Enlarge the neighborhood
- Restart: re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)

Escaping Local Optima

Possibilities:

- Enlarge the neighborhood
- Restart: re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)
- Non-improving steps: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, *e.g.*, using minimally worsening steps.
  (Can lead to long walks in *plateaus*, *i.e.*, regions of search positions with identical evaluation function.) This is what Metaheuristics do.

Escaping Local Optima

Possibilities:

- Enlarge the neighborhood
- Restart: re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)
- Non-improving steps: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, *e.g.*, using minimally worsening steps.
  (Can lead to long walks in *plateaus*, *i.e.*, regions of search positions with identical evaluation function.) This is what Metaheuristics do.

Note: None of these mechanisms is guaranteed to always escape effectively from local optima.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims at greedily increasing solution quality, *e.g.*, by exploiting the evaluation function.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims at greedily increasing solution quality, *e.g.*, by exploiting the evaluation function.
- Diversification: aims at preventing search stagnation, that is, the search process getting trapped in confined regions.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims at greedily increasing solution quality, *e.g.*, by exploiting the evaluation function.
- Diversification: aims at preventing search stagnation, that is, the search process getting trapped in confined regions.

Examples:

- Iterative Improvement (II): *intensification* strategy.
- Uninformed Random Walk/Picking (URW/P): *diversification* strategy.

Diversification *vs* Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- Intensification: aims at greedily increasing solution quality, *e.g.*, by exploiting the evaluation function.
- Diversification: aims at preventing search stagnation, that is, the search process getting trapped in confined regions.

Examples:

- Iterative Improvement (II): *intensification* strategy.
- Uninformed Random Walk/Picking (URW/P): *diversification* strategy.

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.

Outline

Computational Complexity of LS

For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

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

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

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 $\mathcal{PLS}$.

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.

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

Outline

Learning goals of this section

- Review basic formal and theoretical concepts
- Learn about techniques and goals of experimental search space analysis
- Develop intuition on features of local search that may guide the design of LS algorithms

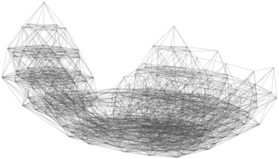
Definitions

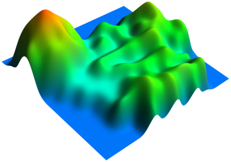
- Search space S
- Neighborhood function $\mathcal{N} : S \subseteq 2^S$
- Evaluation function $f_\pi : S \to \mathbf{R}$
- Problem instance π

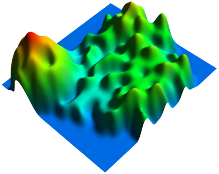
Definition:

The **search landscape** L is the vertex-labeled neighborhood graph given by the triplet $\mathcal{L} = (S_\pi, N_\pi, f_\pi)$.

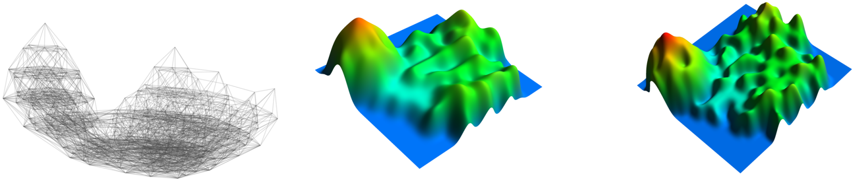
Search Landscape







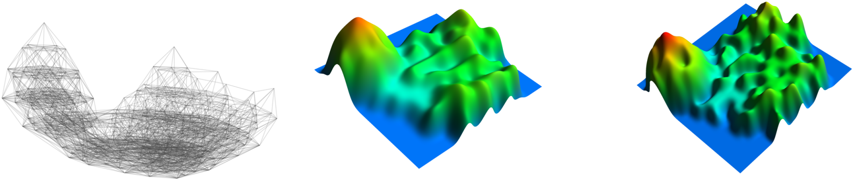
Search Landscape



Transition Graph of Iterative Improvement

Given $\mathcal{L} = (S_\pi, N_\pi, f_\pi)$, 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.

Search Landscape



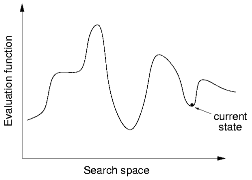
Transition Graph of Iterative Improvement

Given $\mathcal{L} = (S_\pi, N_\pi, f_\pi)$, 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.

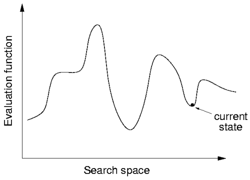
Ideal visualization of landscapes principles

- Simplified landscape representation

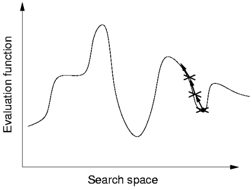


Ideal visualization of landscapes principles

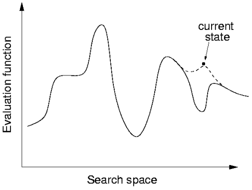
- Simplified landscape representation



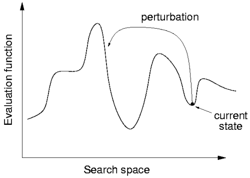
- Tabu Search



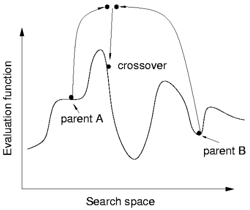
- Guided Local Search



- Iterated Local Search



- Evolutionary Alg.



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 $\mathcal{N}_2$ dominates $\mathcal{N}_1$)

Outline

Neighborhood Operator

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

- Permutation
  - linear permutation:
  - circular permutation:
- Assignment:
- Set, Partition:

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: Graph Coloring Problem, SAT, CSP
- Set, Partition: Max Independent Set

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: Graph Coloring Problem, SAT, CSP
- Set, Partition: Max Independent Set

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

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

Permutations

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

Permutations

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

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

Permutations

$\Pi(n)$ 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

Permutations

$\Pi(n)$ 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 $\pi(i) = \pi_i$

Permutations

$\Pi(n)$ 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 $(\pi \cdot \pi')_i = \pi'(\pi(i))$

Permutations

$\Pi(n)$ 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 $\pi^{-1} \cdot \pi = \iota$

Permutations

$\Pi(n)$ 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 $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi$$

Linear Permutations

Swap operator

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

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

Interchange operator

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

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

($\equiv$ set of all transpositions)

Insert operator

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

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

Circular Permutations

Reversal (2-edge-exchange)

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

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

Block moves (3-edge-exchange)

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

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

Short block move (Or-edge-exchange)

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

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

Assignments

An assignment can be represented as a mapping $\sigma : \{X_1 \ldots X_n\} \rightarrow \{v : v \in D, |D| = k\}$:

$$\sigma = \{X_i = v_i, X_j = v_j, \ldots\}$$

One-exchange operator

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

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

Two-exchange operator

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

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

Partitioning

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

One-addition operator

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

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

One-deletion operator

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

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

Swap operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in C, u \in \overline{C}\}$$

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

Outline

Distances

Set of paths in $\mathcal{L}$ with $s, s' \in S$:

$$\Phi(s, s') = \{(s_1, \ldots, s_h) | s_1 = s, s_h = s' \ \forall i : 1 \leq i \leq h-1, \langle s_i, s_{i+1} \rangle \in E_{\mathcal{L}}\}$$

Distances

Set of paths in $\mathcal{L}$ with $s, s' \in S$:

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

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 $\mathcal{L}$:

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

Distances

Set of paths in $\mathcal{L}$ with $s, s' \in S$:

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

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 $\mathcal{L}$:

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

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

Distances

Set of paths in $\mathcal{L}$ with $s, s' \in S$:

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

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 $\mathcal{L}$:

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

$\mathtt{diam}(\mathcal{L}) = \max\{d_{\mathcal{N}}(s, s') \mid s, s' \in S\}$ (= 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:

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

Distances for Linear Permutation Representations

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

- 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.
  $\mathtt{diam}(G_{\mathcal{N}_X}) = n - 1$

- 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.
  $\mathtt{diam}(G_{\mathcal{N}_I}) = n - 1$

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 =

Example: Search space size and diameter for the TSP

- Search space size = $(n-1)!/2$

Example: Search space size and diameter for the TSP

- Search space size = $(n-1)!/2$
- Insert neighborhood
  size =

Example: Search space size and diameter for the TSP

- Search space size = $(n-1)!/2$
- Insert neighborhood
  size = $(n-3)n$
  diameter =

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 =

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

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 =

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

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 $[n/3, n-1]$

Example: Search space size and diameter for SAT

SAT instance with n variables, 1-flip neighborhood:
$G_{\mathcal{N}} = n$-dimensional hypercube; diameter of $G_{\mathcal{N}} = n$.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ 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 $\mathcal{N}_2$ dominates $\mathcal{N}_1$

Example:

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

Outline

Other Search Space Properties

- number of (optimal) solutions $|S'|$, solution density $|S'|/|S|$
- distribution of solutions within the neighborhood graph

Other Search Space Properties

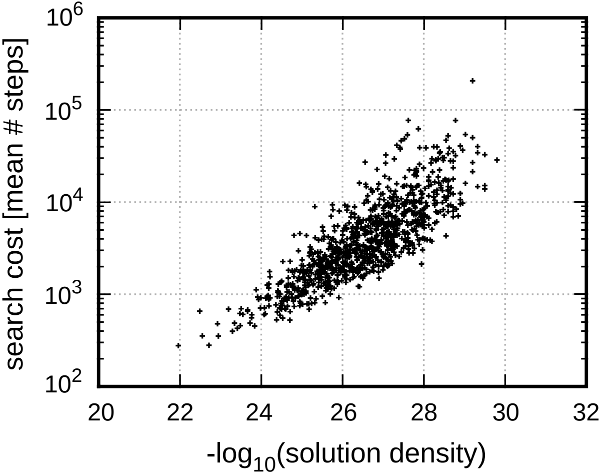
- number of (optimal) solutions $|S'|$, solution density $|S'|/|S|$
- distribution of solutions within the neighborhood graph

Solution densities and distributions can generally be determined by:

- exhaustive enumeration;
- sampling methods;
- counting algorithms (often variants of complete algorithms).

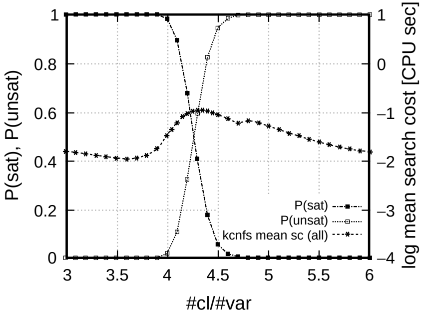
Example: Correlation between solution density and search cost for GWSAT over set of hard Random-3-SAT instances:

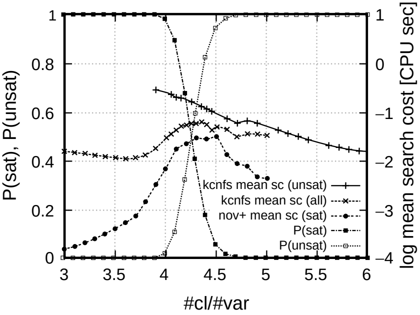
The less solutions, the harder to find them



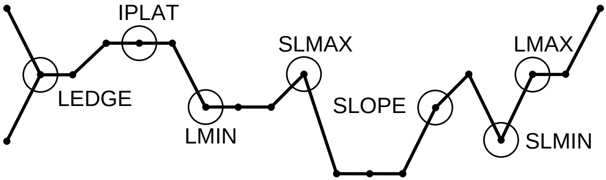
Phase Transition for 3-SAT

Random instances ⇝ *m* clauses of *n* uniformly chosen variables





Classification of search positions



position type	>	=	<
SLMIN (strict local min)	+	−	−
LMIN (local min)	+	+	−
IPLAT (interior plateau)	−	+	−
SLOPE	+	−	+
LEDGE	+	+	+
LMAX (local max)	−	+	+
SLMAX (strict local max)	−	−	+

"+" = present, "–" absent; table entries refer to neighbors with larger (">") , equal ("="), and smaller ("<") evaluation function values

Example: Complete distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
`uf20-91/easy`	13.05	0%	0.11%	0%
`uf20-91/medium`	83.25	$< 0.01\%$	0.13%	0%
`uf20-91/hard`	563.94	$< 0.01\%$	0.16%	0%

Example: Complete distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
`uf20-91/easy`	13.05	0%	0.11%	0%
`uf20-91/medium`	83.25	$< 0.01\%$	0.13%	0%
`uf20-91/hard`	563.94	$< 0.01\%$	0.16%	0%

instance	SLOPE	LEDGE	LMAX	SLMAX
`uf20-91/easy`	0.59%	99.27%	0.04%	$< 0.01\%$
`uf20-91/medium`	0.31%	99.40%	0.06%	$< 0.01\%$
`uf20-91/hard`	0.56%	99.23%	0.05%	$< 0.01\%$

Example: Complete distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
uf20-91/easy	13.05	0%	0.11%	0%
uf20-91/medium	83.25	$< 0.01\%$	0.13%	0%
uf20-91/hard	563.94	$< 0.01\%$	0.16%	0%

instance	SLOPE	LEDGE	LMAX	SLMAX
uf20-91/easy	0.59%	99.27%	0.04%	$< 0.01\%$
uf20-91/medium	0.31%	99.40%	0.06%	$< 0.01\%$
uf20-91/hard	0.56%	99.23%	0.05%	$< 0.01\%$

(based on exhaustive enumeration of search space;
sc refers to search cost for GWSAT)

Example: Sampled distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
`uf50-218/medium`	615.25	0%	47.29%	0%
`uf100-430/medium`	3 410.45	0%	43.89%	0%
`uf150-645/medium`	10 231.89	0%	41.95%	0%

Example: Sampled distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
uf50-218/medium	615.25	0%	47.29%	0%
uf100-430/medium	3 410.45	0%	43.89%	0%
uf150-645/medium	10 231.89	0%	41.95%	0%

instance	SLOPE	LEDGE	LMAX	SLMAX
uf50-218/medium	$< 0.01\%$	52.71%	0%	0%
uf100-430/medium	0%	56.11%	0%	0%
uf150-645/medium	0%	58.05%	0%	0%

Example: Sampled distribution of position types for hard Random-3-SAT instances

instance	*avg sc*	SLMIN	LMIN	IPLAT
`uf50-218/medium`	615.25	0%	47.29%	0%
`uf100-430/medium`	3 410.45	0%	43.89%	0%
`uf150-645/medium`	10 231.89	0%	41.95%	0%

instance	SLOPE	LEDGE	LMAX	SLMAX
`uf50-218/medium`	$< 0.01\%$	52.71%	0%	0%
`uf100-430/medium`	0%	56.11%	0%	0%
`uf150-645/medium`	0%	58.05%	0%	0%

(based on sampling along GWSAT trajectories;
sc refers to search cost for GWSAT)

Local Minima

Note: Local minima prevent local search progress.

Local Minima

Note: Local minima prevent local search progress.

Simple properties of local minima:

- *number of local minima:* $|lmin|$, *local minima density* $|lmin|/|S|$

Local Minima

Note: Local minima prevent local search progress.

Simple properties of local minima:

- *number of local minima:* $|lmin|$, *local minima density* $|lmin|/|S|$
- *localization of local minima:* distribution of local minima within the neighborhood graph

Local Minima

Note: Local minima prevent local search progress.

Simple properties of local minima:

- *number of local minima:* $|lmin|$, *local minima density* $|lmin|/|S|$
- *localization of local minima:* distribution of local minima within the neighborhood graph

Problem: Determining these measures typically requires exhaustive enumeration of search space.

Local Minima

Note: Local minima prevent local search progress.

Simple properties of local minima:

- *number of local minima:* $|lmin|$, *local minima density* $|lmin|/|S|$
- *localization of local minima:* distribution of local minima within the neighborhood graph

Problem: Determining these measures typically requires exhaustive enumeration of search space.

$\leadsto$ Approximation based on sampling or estimation from other measures (such as autocorrelation measures, see below).

Example: Distribution of local minima for the TSP

Goal: Empirical analysis of distribution of local minima for Euclidean TSP instances.

Example: Distribution of local minima for the TSP

Goal: Empirical analysis of distribution of local minima for Euclidean TSP instances.

Experimental approach:

- Sample sets of local optima of three TSPLIB instances using multiple independent runs of two TSP algorithms (3-opt, ILS).

Example: Distribution of local minima for the TSP

Goal: Empirical analysis of distribution of local minima for Euclidean TSP instances.

Experimental approach:

- Sample sets of local optima of three TSPLIB instances using multiple independent runs of two TSP algorithms (3-opt, ILS).
- Measure pairwise distances between local minima (using *bond distance* = number of edges in which two given tours differ).

Example: Distribution of local minima for the TSP

Goal: Empirical analysis of distribution of local minima for Euclidean TSP instances.

Experimental approach:

- Sample sets of local optima of three TSPLIB instances using multiple independent runs of two TSP algorithms (3-opt, ILS).
- Measure pairwise distances between local minima (using *bond distance* = number of edges in which two given tours differ).
- Sample set of purportedly globally optimal tours using multiple independent runs of high-performance TSP algorithm.

Example: Distribution of local minima for the TSP

Goal: Empirical analysis of distribution of local minima for Euclidean TSP instances.

Experimental approach:

- Sample sets of local optima of three TSPLIB instances using multiple independent runs of two TSP algorithms (3-opt, ILS).
- Measure pairwise distances between local minima (using *bond distance* = number of edges in which two given tours differ).
- Sample set of purportedly globally optimal tours using multiple independent runs of high-performance TSP algorithm.
- Measure minimal pairwise distances between local minima and respective closest optimal tour (using bond distance).

Empirical results:

Instance	avg sq [%]	avg d_{lmin}	avg d_{opt}
	Results for 3-opt		
rat783	3.45	197.8	185.9
pr1002	3.58	242.0	208.6
pcb1173	4.81	274.6	246.0

Empirical results:

Instance	avg sq [%]	avg d_{lmin}	avg d_{opt}
Results for 3-opt			
rat783	3.45	197.8	185.9
pr1002	3.58	242.0	208.6
pcb1173	4.81	274.6	246.0
Results for ILS algorithm			
rat783	0.92	142.2	123.1
pr1002	0.85	177.2	143.2
pcb1173	1.05	177.4	151.8

Empirical results:

Instance	avg *sq* [%]	avg d_{lmin}	avg d_{opt}
	Results for 3-opt		
rat783	3.45	197.8	185.9
pr1002	3.58	242.0	208.6
pcb1173	4.81	274.6	246.0
	Results for ILS algorithm		
rat783	0.92	142.2	123.1
pr1002	0.85	177.2	143.2
pcb1173	1.05	177.4	151.8

(based on local minima collected from 1 000/200 runs of 3-opt/ILS)
avg *sq* [%]: average solution quality expressed in percentage deviation from optimal solution

Interpretation:

- Average distance between local minima is small compared to maximal possible bond distance, n.

  ⇝ *Local minima are concentrated in a relatively small region of the search space.*

Interpretation:

- Average distance between local minima is small compared to maximal possible bond distance, n.

  $\rightsquigarrow$ *Local minima are concentrated in a relatively small region of the search space.*

- Average distance between local minima is slightly larger than distance to closest global optimum.

  $\rightsquigarrow$ *Optimal solutions are located centrally in region of high local minima density.*

Interpretation:

- Average distance between local minima is small compared to maximal possible bond distance, n.

  $\rightsquigarrow$ *Local minima are concentrated in a relatively small region of the search space.*

- Average distance between local minima is slightly larger than distance to closest global optimum.

  $\rightsquigarrow$ *Optimal solutions are located centrally in region of high local minima density.*

- Higher-quality local minima found by ILS tend to be closer to each other and the closest global optima compared to those determined by 3-opt.

  $\rightsquigarrow$ *Higher-quality local minima tend to be concentrated in smaller regions of the search space.*

Interpretation:

- Average distance between local minima is small compared to maximal possible bond distance, n.

  $\rightsquigarrow$ *Local minima are concentrated in a relatively small region of the search space.*

- Average distance between local minima is slightly larger than distance to closest global optimum.

  $\rightsquigarrow$ *Optimal solutions are located centrally in region of high local minima density.*

- Higher-quality local minima found by ILS tend to be closer to each other and the closest global optima compared to those determined by 3-opt.

  $\rightsquigarrow$ *Higher-quality local minima tend to be concentrated in smaller regions of the search space.*

Interpretation:

- Average distance between local minima is small compared to maximal possible bond distance, n.

  ⇝ *Local minima are concentrated in a relatively small region of the search space.*

- Average distance between local minima is slightly larger than distance to closest global optimum.

  ⇝ *Optimal solutions are located centrally in region of high local minima density.*

- Higher-quality local minima found by ILS tend to be closer to each other and the closest global optima compared to those determined by 3-opt.

  ⇝ *Higher-quality local minima tend to be concentrated in smaller regions of the search space.*

Note: These results are fairly typical for many types of TSP instances and instances of other combinatorial problems.
In many cases, local optima tend to be clustered; this is reflected in multi-modal distributions of pairwise distances between local minima.

Fitness-Distance Correlation (FDC)

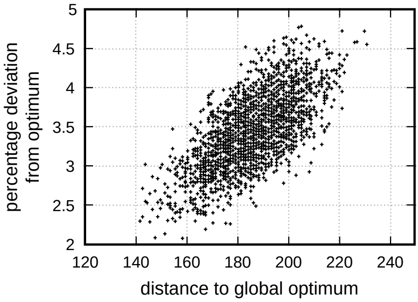
Idea: Analyze correlation between solution quality (fitness) g of candidate solutions and distance d to (closest) optimal solution.

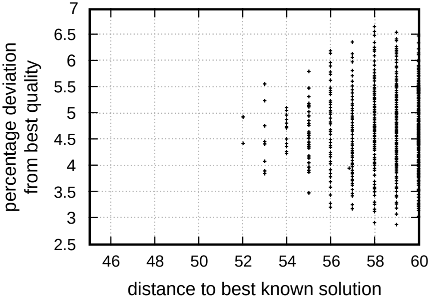
Measure for FDC: *empirical correlation coefficient* r_{fdc}.

Fitness-distance plots, *i.e.*, scatter plots of the (g_i, d_i) pairs underlying an estimate of r_{fdc}, are often useful to graphically illustrate fitness distance correlations.

- The FDC coefficient, r_{fdc} depends on the given neighborhood relation.
- r_{fdc} is calculated based on a sample of m candidate solutions (typically: set of local optima found over multiple runs of an iterative improvement algorithm).

Example: FDC plot for TSPLIB instance `rat783`, based on 2500 local optima obtained from a 3-opt algorithm





High FDC (r_{fdc} close to one):

- 'Big valley' structure of landscape provides guidance for local search;
- search initialization: high-quality candidate solutions provide good starting points;

High FDC (r_{fdc} close to one):

- 'Big valley' structure of landscape provides guidance for local search;
- search initialization: high-quality candidate solutions provide good starting points;
- search diversification: (weak) perturbation is better than restart;

High FDC (r_{fdc} close to one):

- 'Big valley' structure of landscape provides guidance for local search;
- search initialization: high-quality candidate solutions provide good starting points;
- search diversification: (weak) perturbation is better than restart;
- typical, *e.g.*, for TSP.

High FDC (r_{fdc} close to one):

- 'Big valley' structure of landscape provides guidance for local search;
- search initialization: high-quality candidate solutions provide good starting points;
- search diversification: (weak) perturbation is better than restart;
- typical, *e.g.*, for TSP.

Low FDC (r_{fdc} close to zero):

- global structure of landscape does not provide guidance for local search;

High FDC (r_{fdc} close to one):

- ‘Big valley’ structure of landscape provides guidance for local search;
- search initialization: high-quality candidate solutions provide good starting points;
- search diversification: (weak) perturbation is better than restart;
- typical, *e.g.*, for TSP.

Low FDC (r_{fdc} close to zero):

- global structure of landscape does not provide guidance for local search;
- typical for very hard combinatorial problems, such as certain types of QAP (Quadratic Assignment Problem) instances.

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;
- analysis of problem and problem instance difficulty.

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;
- analysis of problem and problem instance difficulty.

Limitations and short-comings:

- *a posteriori* method, requires set of (optimal) solutions,

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;
- analysis of problem and problem instance difficulty.

Limitations and short-comings:

- *a posteriori* method, requires set of (optimal) solutions, **but:** results often generalize to larger instance classes;

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;
- analysis of problem and problem instance difficulty.

Limitations and short-comings:

- *a posteriori* method, requires set of (optimal) solutions,
  but: results often generalize to larger instance classes;
- optimal solutions are often not known, using best known solutions can lead to erroneous results;

Applications of fitness-distance analysis:

- algorithm design: use of strong intensification (including initialization) and relatively weak diversification mechanisms;
- comparison of effectiveness of neighborhood relations;
- analysis of problem and problem instance difficulty.

Limitations and short-comings:

- *a posteriori* method, requires set of (optimal) solutions,
  but: results often generalize to larger instance classes;
- optimal solutions are often not known, using best known solutions can lead to erroneous results;
- can give misleading results when used as the sole basis for assessing problem or instance difficulty.

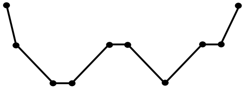
Ruggedness

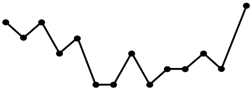
Idea: Rugged search landscapes, *i.e.*, landscapes with high variability in evaluation function value between neighboring search positions, are hard to search.

Ruggedness

Idea: Rugged search landscapes, *i.e.*, landscapes with high variability in evaluation function value between neighboring search positions, are hard to search.

Example: Smooth *vs* rugged search landscape





Ruggedness

Idea: Rugged search landscapes, *i.e.*, landscapes with high variability in evaluation function value between neighboring search positions, are hard to search.

Example: Smooth *vs* rugged search landscape

Note: Landscape ruggedness is closely related to local minima density: rugged landscapes tend to have many local minima.

The ruggedness of a landscape L can be measured by means of the empirical autocorrelation function $r(i)$:

$$r(i) := \frac{1/(m-i) \cdot \sum_{k=1}^{m-i} (g_k - \bar{g}) \cdot (g_{k+i} - \bar{g})}{1/m \cdot \sum_{k=1}^{m} (g_k - \bar{g})^2}$$

where $g_1, \ldots g_m$ are evaluation function values sampled along an uninformed random walk in L.

Note: $r(i)$ depends on the given neighborhood relation.

- Empirical autocorrelation analysis is computationally cheap compared to, *e.g.*, fitness-distance analysis.
- (Bounds on) AC can be theoretically derived in many cases, *e.g.*, the TSP with the 2-exchange neighborhood.
- There are other measures of ruggedness, such as *empirical autocorrelation coefficient* and *(empirical) correlation length*.

High AC (close to one):

- "smooth" landscape;
- evaluation function values for neighboring candidate solutions are close on average;

High AC (close to one):

- “smooth” landscape;
- evaluation function values for neighboring candidate solutions are close on average;
- low local minima density;

High AC (close to one):

- "smooth" landscape;
- evaluation function values for neighboring candidate solutions are close on average;
- low local minima density;
- problem typically relatively easy for local search.

High AC (close to one):

- "smooth" landscape;
- evaluation function values for neighboring candidate solutions are close on average;
- low local minima density;
- problem typically relatively easy for local search.

Low AC (close to zero):

- very rugged landscape;
- evaluation function values for neighboring candidate solutions are almost uncorrelated;

High AC (close to one):

- "smooth" landscape;
- evaluation function values for neighboring candidate solutions are close on average;
- low local minima density;
- problem typically relatively easy for local search.

Low AC (close to zero):

- very rugged landscape;
- evaluation function values for neighboring candidate solutions are almost uncorrelated;
- high local minima density;

High AC (close to one):

- "smooth" landscape;
- evaluation function values for neighboring candidate solutions are close on average;
- low local minima density;
- problem typically relatively easy for local search.

Low AC (close to zero):

- very rugged landscape;
- evaluation function values for neighboring candidate solutions are almost uncorrelated;
- high local minima density;
- problem typically relatively hard for local search.

Note:

- Measures of ruggedness, such as AC, are often insufficient for distinguishing between the hardness of individual problem instances;

Note:

- Measures of ruggedness, such as AC, are often insufficient for distinguishing between the hardness of individual problem instances;
- but they can be useful for
  - analyzing differences between neighborhood relations for a given problem,

Note:

- Measures of ruggedness, such as AC, are often insufficient for distinguishing between the hardness of individual problem instances;
- but they can be useful for
  - analyzing differences between neighborhood relations for a given problem,
  - studying the impact of parameter settings of a given SLS algorithm on its behavior,

Note:

- Measures of ruggedness, such as AC, are often insufficient for distinguishing between the hardness of individual problem instances;
- but they can be useful for
  - analyzing differences between neighborhood relations for a given problem,
  - studying the impact of parameter settings of a given SLS algorithm on its behavior,
  - classifying the difficulty of combinatorial problems.

Plateaux

Plateaux, *i.e.*, ‘flat’ regions in the search landscape

Plateaux

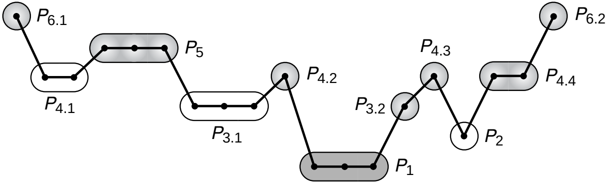
Plateaux, *i.e.*, ‘flat’ regions in the search landscape

Intuition: Plateaux can impede search progress due to lack of guidance by the evaluation function.

Plateaux

Plateaux, *i.e.*, ‘flat’ regions in the search landscape

Intuition: Plateaux can impede search progress due to lack of guidance by the evaluation function.



Definitions

- **Region:** connected set of search positions.
- **Border of region R:** set of search positions with at least one direct neighbor outside of R (**border positions**).
- **Plateau region:** region in which all positions have the same level, *i.e.*, evaluation function value, l.
- **Plateau:** maximally extended plateau region, *i.e.*, plateau region in which no border position has any direct neighbors at the plateau level l.
- **Solution plateau:** Plateau that consists entirely of solutions of the given problem instance.
- **Exit of plateau region R:** direct neighbor s of a border position of R with lower level than plateau level l.
- **Open / closed plateau:** plateau with / without exits.

Measures of plateau structure:

- *plateau diameter* = diameter of corresponding subgraph of $G_{\mathcal{N}}$
- *plateau width* = maximal distance of any plateau position to the respective closest border position
- *number of exits*, *exit density*
- *distribution of exits within a plateau*, *exit distance distribution* (in particular: avg./max. distance to closest exit)

Some plateau structure results for SAT:

- Plateaux typically don't have an interior, *i.e.*, almost every position is on the border.

Some plateau structure results for SAT:

- Plateaux typically don't have an interior, *i.e.*, almost every position is on the border.
- The diameter of plateaux, particularly at higher levels, is comparable to the diameter of search space. (In particular: plateaux tend to span large parts of the search space, but are quite well connected internally.)

Some plateau structure results for SAT:

- Plateaux typically don't have an interior, *i.e.*, almost every position is on the border.
- The diameter of plateaux, particularly at higher levels, is comparable to the diameter of search space. (In particular: plateaux tend to span large parts of the search space, but are quite well connected internally.)
- For open plateaux, exits tend to be clustered, but the average exit distance is typically relatively small.

Barriers and Basins

Observation:

The *difficulty of escaping* from closed plateaux or strict local minima is related to the *height of the barrier*, *i.e.*, the difference in evaluation function, that needs to be overcome in order to reach better search positions:

Barriers and Basins

Observation:

The *difficulty of escaping* from closed plateaux or strict local minima is related to the *height of the barrier*, *i.e.*, the difference in evaluation function, that needs to be overcome in order to reach better search positions:

Higher barriers are typically more difficult to overcome (this holds, *e.g.*, for Probabilistic Iterative Improvement or Simulated Annealing).

Definitions:

- Positions s, s' are *mutually accessible at level* l iff there is a path connecting s' and s in the neighborhood graph that visits only positions t with $g(t) \leq l$.

Definitions:

- Positions s, s' are *mutually accessible at level* l iff there is a path connecting s' and s in the neighborhood graph that visits only positions t with $g(t) \leq l$.
- The *barrier level between positions* s, s', $bl(s, s')$ is the lowest level l at which s' and s' are mutually accessible;

Definitions:

- Positions s, s' are *mutually accessible at level* l
iff there is a path connecting s' and s in the neighborhood graph that visits only positions t with $g(t) \leq l$.

- The *barrier level between positions* s, s', $bl(s, s')$
is the lowest level l at which s' and s' are mutually accessible;
the difference between the level of s and $bl(s, s')$ is called
the *barrier height between* s *and* s'.

Definitions:

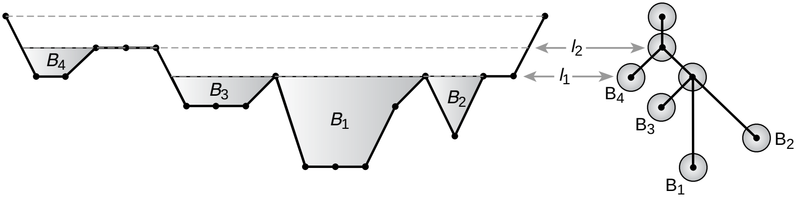
- Positions s, s' are *mutually accessible at level* l iff there is a path connecting s' and s in the neighborhood graph that visits only positions t with $g(t) \leq l$.

- The *barrier level between positions* s, s', $bl(s, s')$ is the lowest level l at which s' and s' are mutually accessible;

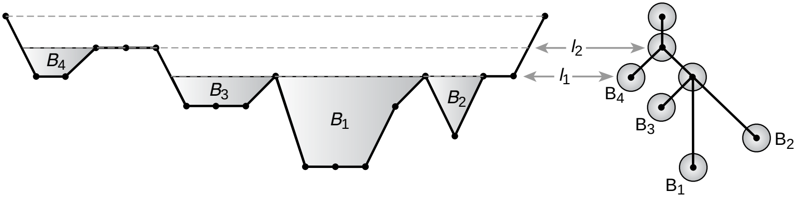
  the difference between the level of s and $bl(s, s')$ is called the *barrier height between* s *and* s'.

- **Basins**, *i.e.*, maximal (connected) regions of search positions below a given level, form an important basis for characterizing search space structure.

Example: Basins in a simple search landscape and corresponding basin tree



Example: Basins in a simple search landscape and corresponding basin tree



Note: The basin tree only represents basins just below the critical levels at which neighboring basins are joined (by a *saddle*).