Computational Techniques

Markus Leitner[1]

Department of Statistics and Operations Research
University of Vienna

WS 2015

[1] based on slides by Johannes Inführ, Christian Schauer (TU Wien)

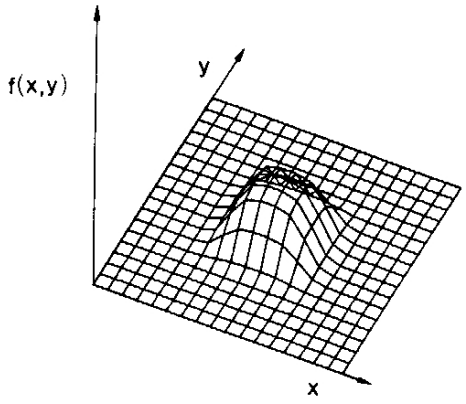
Part I

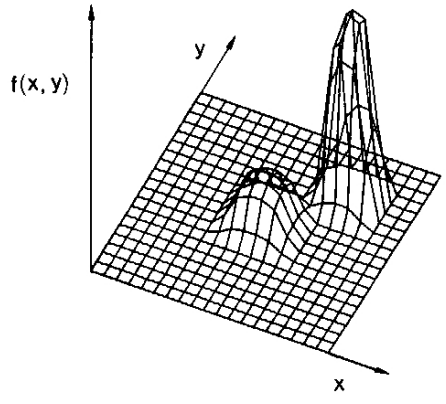
Introduction & Construction Heuristics

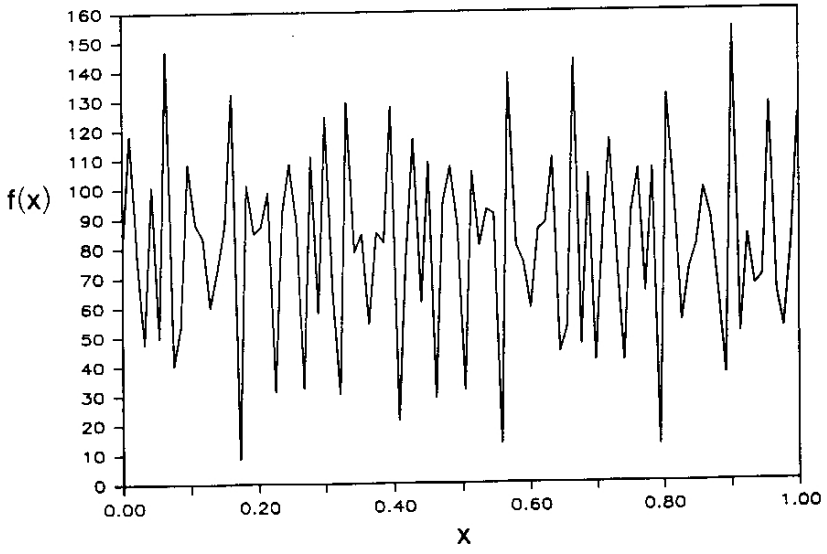
Types of optimization tasks

- **Combinatorial problems**
  - Discrete and finite set of solutions
  - *The lecture mainly focuses on this part*
- Continuous parameter optimization
- Optimization of problems with non-linear structures, e.g.,
  - Neural networks
  - Computer programs
  - Electronic circuits

Unimodal and multimodal continuous functions







Combinatorial optimization problems

Definition (Combinatorial optimization problem)

A **combinatorial optimization problem** is a set of **(problem-) instances**.

An instance is a pair (S, f), in which

- S is a finite set of all feasible solutions (**search space**) and
- $f : S \to \mathbb{R}$ is the **objective function** to minimize or maximize.

The task is to find a **(global) optimum solution**, i.e., an $x^* \in S$, so that $f(x^*) \leq f(x)$, $\forall x \in S$ (or $f(x^*) \geq f(x)$).

Examples for combinatorial optimization problems

- Shortest paths and tours
- Routing for delivery services
- Timetabling
- Process scheduling
- Optimal investments on limited budget
- Cutting and packing problems
- Network design

Example: Traveling Salesman Problem (TSP)

Definition (Traveling Salesman Problem)

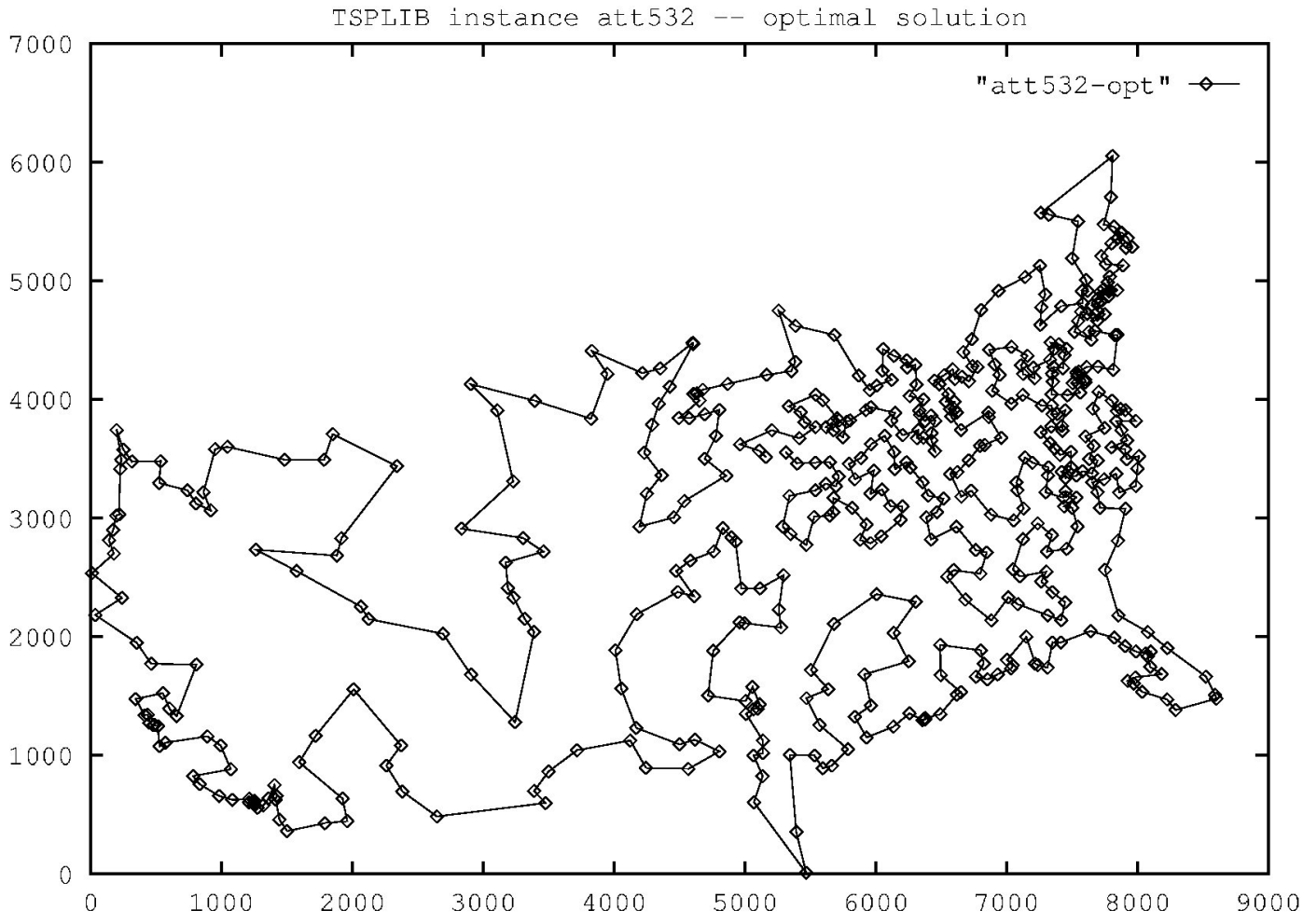
Given: a complete weighted graph $G(V, E)$
with edge costs $d_{i,j} \geq 0$, $\forall (i, j) \in E$

Wanted: a Hamiltonian cycle with minimum weight, i.e., the shortest tour that visits each vertex exactly once

Special cases:

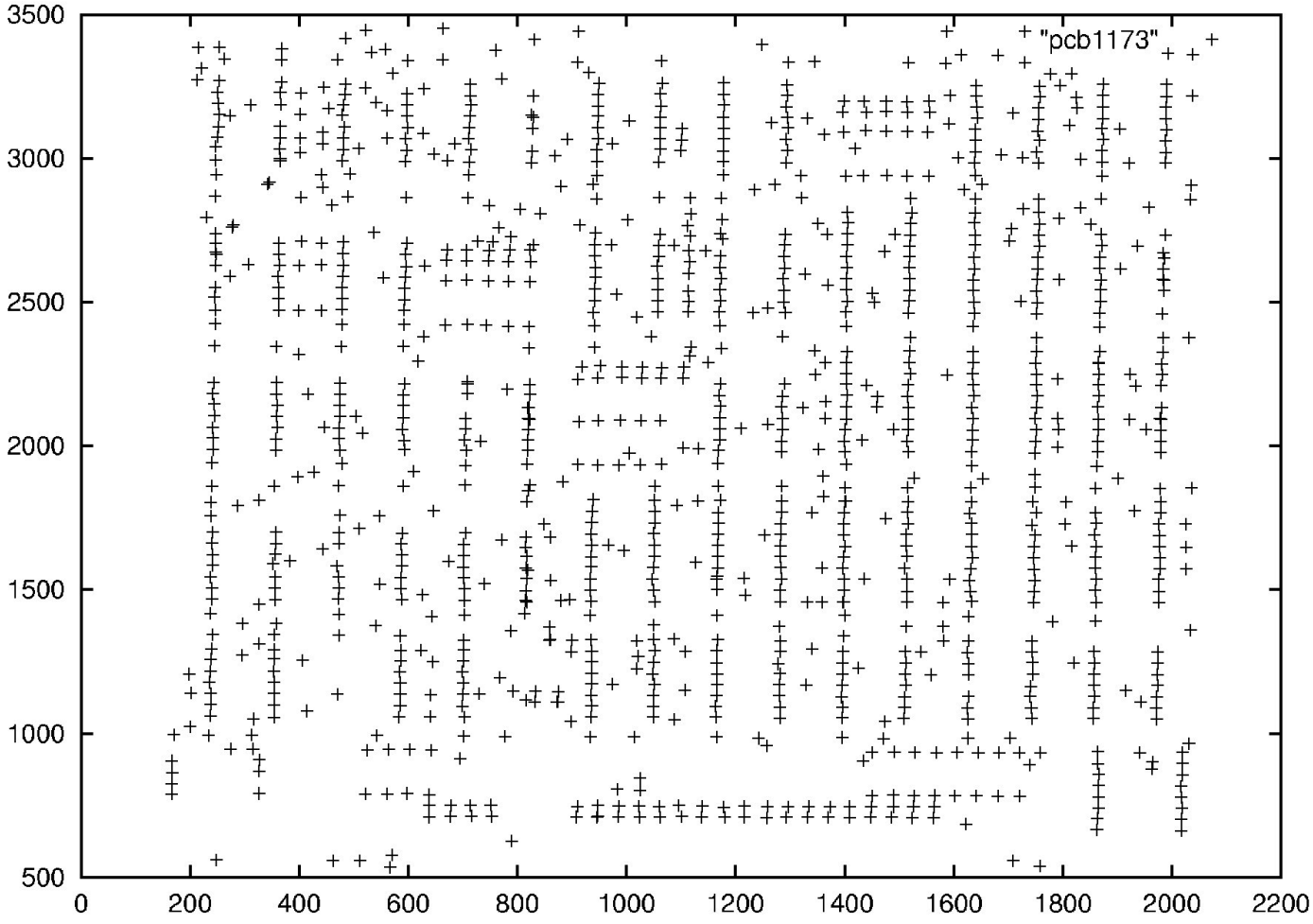
- Euclidean instances
- Symmetric TSP, i.e., $\forall i, j : \ d_{i,j} = d_{j,i}$
- Asymmetric TSP, i.e., $\exists i, j \mid d_{i,j} \neq d_{j,i}$
- ...

Example instance for TSP (1)



Shortest tour through 532 US cities

Example instance for TSP (2)



Instance of a drilling problem for a printed circuit board

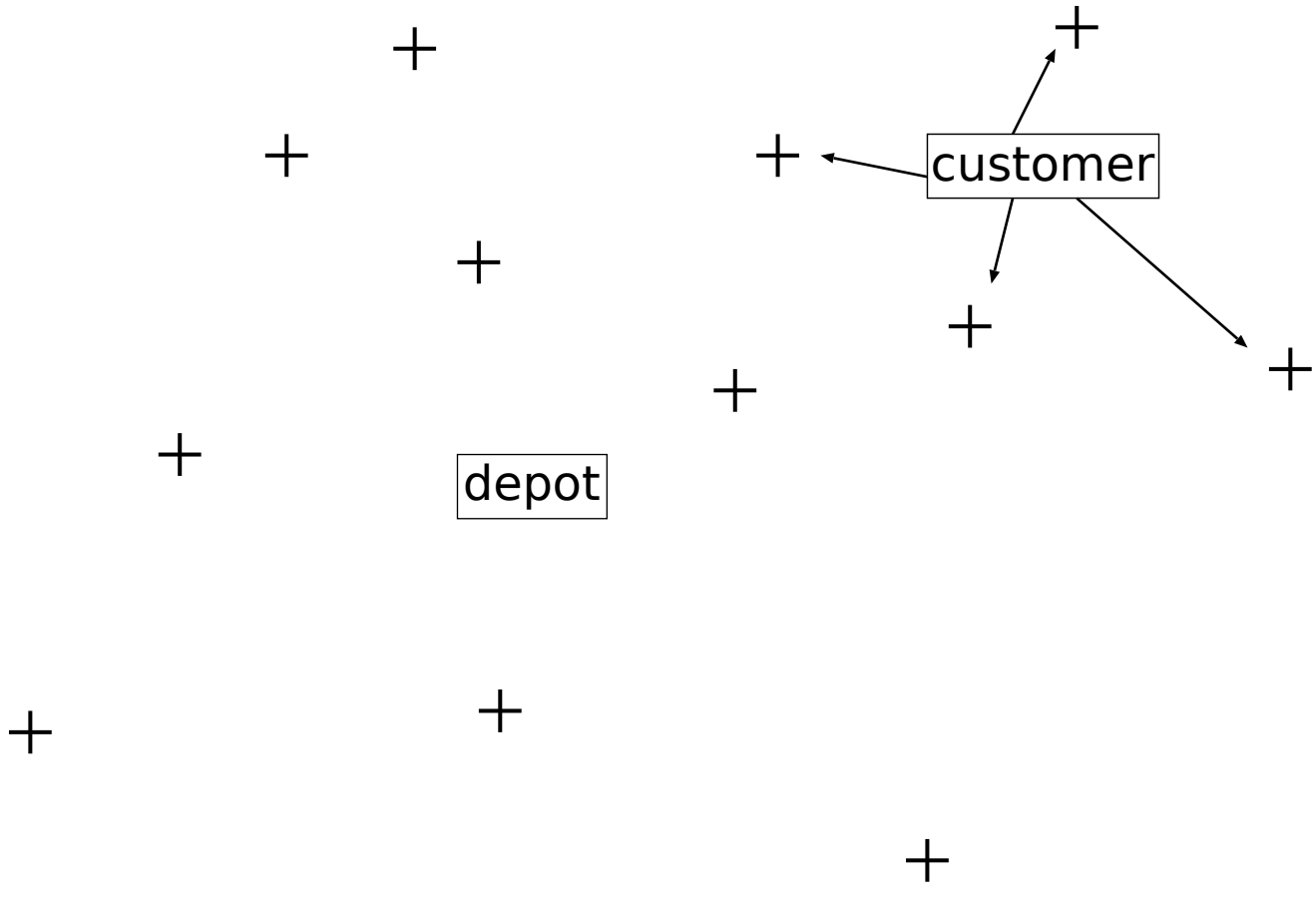
Practice oriented extension of TSP: Routing

Given: Soft drink company "Blue Cow"

Vehicle Routing Problem (VRP)

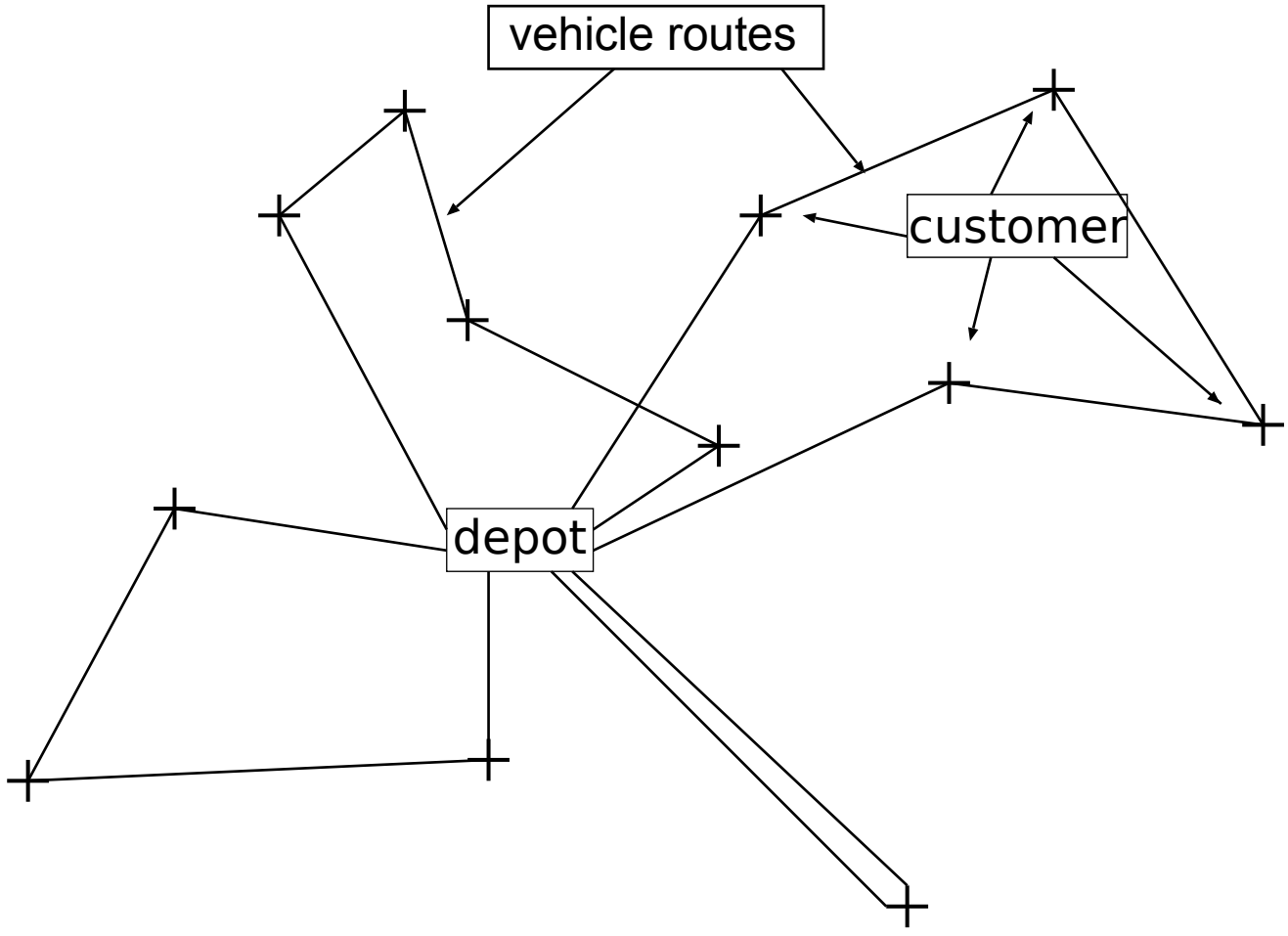
- Customers (e.g., bars) with given demands that must be satisfied
- There is only one depot (= starting and endpoint for the vehicles)
- Vehicle routes must be planned for the delivery taking the capacity restrictions of the vehicles into account.
- The objective is to minimize the number of routes and/or the overall distance of the routes.

Example of a VRP instance



Given: depot und customers

Example of a VRP instance



Possible solution

Common practice-oriented extensions

Restrictions for vehicle/driver

- Capacity restrictions
- Maximum working hours
- Breaks for the drivers
- Different costs for the vehicles
- Time dependent tour costs

Common practice-oriented extensions (contd.)

Restrictions for customers

- Time windows from customers
- Delivery and collection at the customer can be stochastic
- Access restrictions for customers
- Priorities for the customers, if not all of them can be served
- Split delivery

"Difficult problems"

Some optimization problems are **difficult to solve**, because of

- Size of the search space
- Complexity of the objective function
- Additional constraints
- Noisy or imprecise objective function
- Time dependent objective function
- Multi-objective optimization

Multi-objective optimization

- Set of objective functions:

$$\min F(x) = (f_1(x), f_2(x), \ldots, f_n(x)), \ \forall x \in S, n \geq 2$$

- Usually, there is no solution x^*, which is optimal for all objectives
- Other concepts needed to consider optimality

  $\rightarrow$ New concept by Vilfredo Pareto in 1896

Multi-objective optimization

Definition (Pareto dominance)

An objective vector $u = (u_1, \ldots, u_n)$ is said to *dominate* $v = (v_1, \ldots, v_n)$ (denoted by $u \prec v$) iff:

$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i \wedge \exists i \in \{1, \ldots, n\} : u_i < v_i$$

No component of v is smaller than the corresponding component of u and at least one component of u is strictly smaller.

Multi-objective optimization

Definition (Pareto optimality)

A solution $x^* \in S$ is *Pareto optimal* if for every $x \in S$, $F(x)$ does not dominate $F(x^*)$, i.e., $F(x) \not\prec F(x^*)$

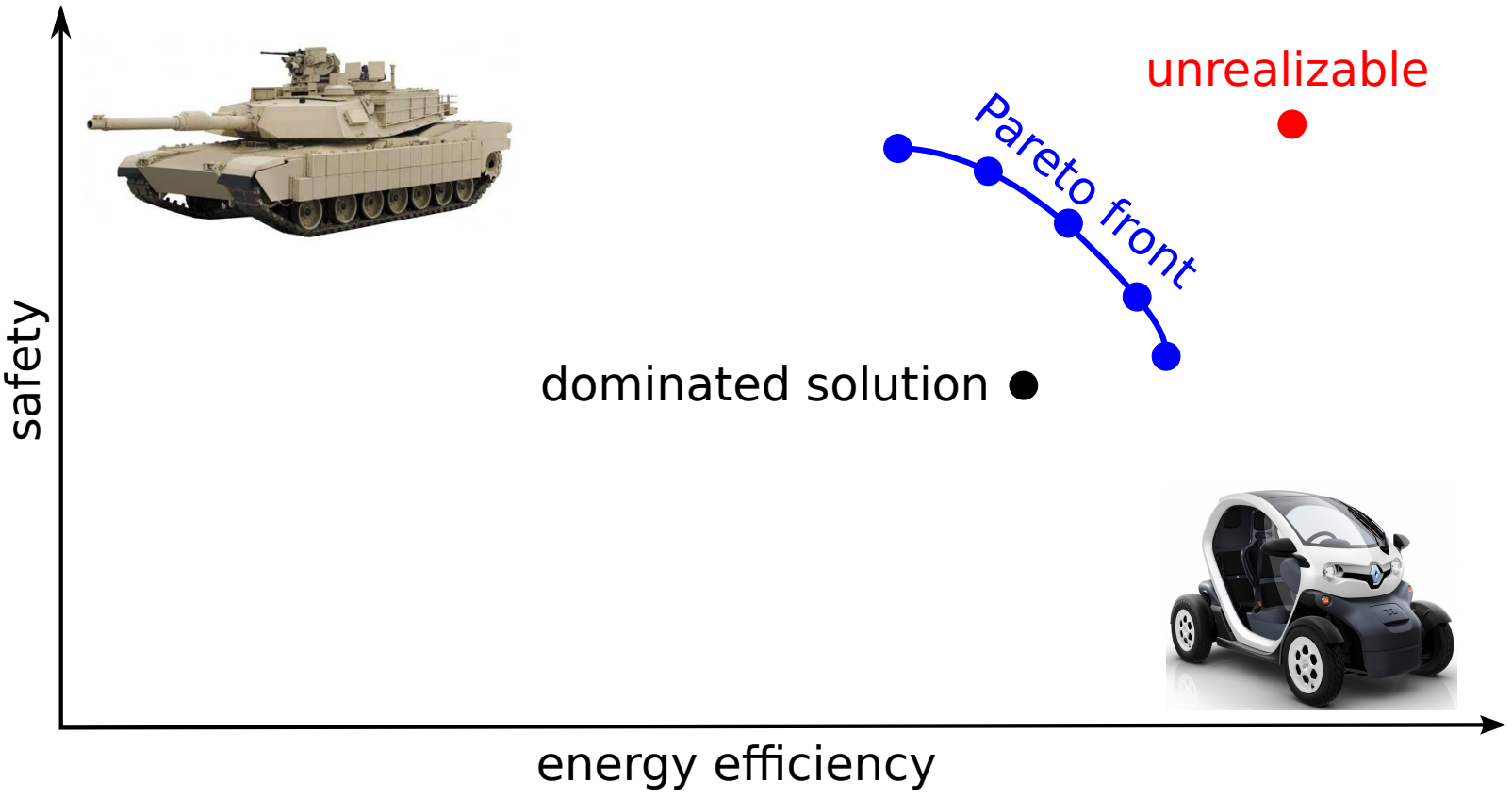
For a Pareto optimal solution it is impossible to improve the performance of a criterion without decrease the quality of at least another criterion.

Definition (Pareto front)

The set of Pareto optimal solutions is called the *Pareto front*.

Multi-objective optimization: Example

A customer wants to buy an energy efficient **and** safe vehicle



Solution space with Pareto front

Examples of typical optimization methods

Random search:

global method, generally applicable,
but mostly inefficient

Enumeration methods:

simple enumeration is inefficient in a larger search space
better: limited enumeration, dynamic programming

Branch and bound:

split search space recursively using divide and conquer principle,
until each part can either be solved or excluded

Gradient descent:

local method, derivation needed

Simplex algorithm:

for solving linear programs
(linear objective function, linear constraints, continuous variables)

Heuristic methods

Heuristic methods do not necessarily provide an optimum solution, but "hopefully" a good solution in reasonable time.

There are basically two types:

Construction heuristics:

- very problem specific approach
- usually fast and of simple design

Improvement heuristic:

- attached to a construction heuristic
- tries to improve a given solution by modifying it iteratively
- relies on rather problem independent principles
- often slower but with a better solution quality

What is a metaheuristic?

Definition (Metaheuristic (Wikipedia, October 2012))

"In computer science, metaheuristic designates a computational method that **optimizes a problem by iteratively trying to improve** a candidate solution with regard to a given measure of quality. Metaheuristics make **few or no assumptions about the problem** being optimized and can search very large spaces of candidate solutions. However, metaheuristics do **not guarantee an optimal solution** is ever found. Many metaheuristics implement some form of stochastic optimization."

Overview of methods (not exhaustive!)

- Construction heuristics
- Simple local search and its variants
- Greedy randomized adaptive search procedure (GRASP)
- Variable neighborhood search (VNS)
- Very large neighborhood search
- Simulated annealing
- Tabu search
- Guided local search
- Evolutionary algorithms (GA, ES, GP)
- Ant colony optimization (ACO)

No free lunch theorem

What is the very best method?

Wolpert and Macready (1996)

"All optimization methods perform over **all** problems equally well (or bad)."

This applies for random search, too!

→ **The performance of different optimization methods can only be compared with respect to a certain problem.**

Construction heuristics

Construction heuristics

Common scheme:

- start with an "empty" solution
- at each step add a new element to the solution
- Greedy method: Use a heuristic to select the locally best element. If an element is part of the solution it is never replaced by another.

procedure greedy construction heuristic

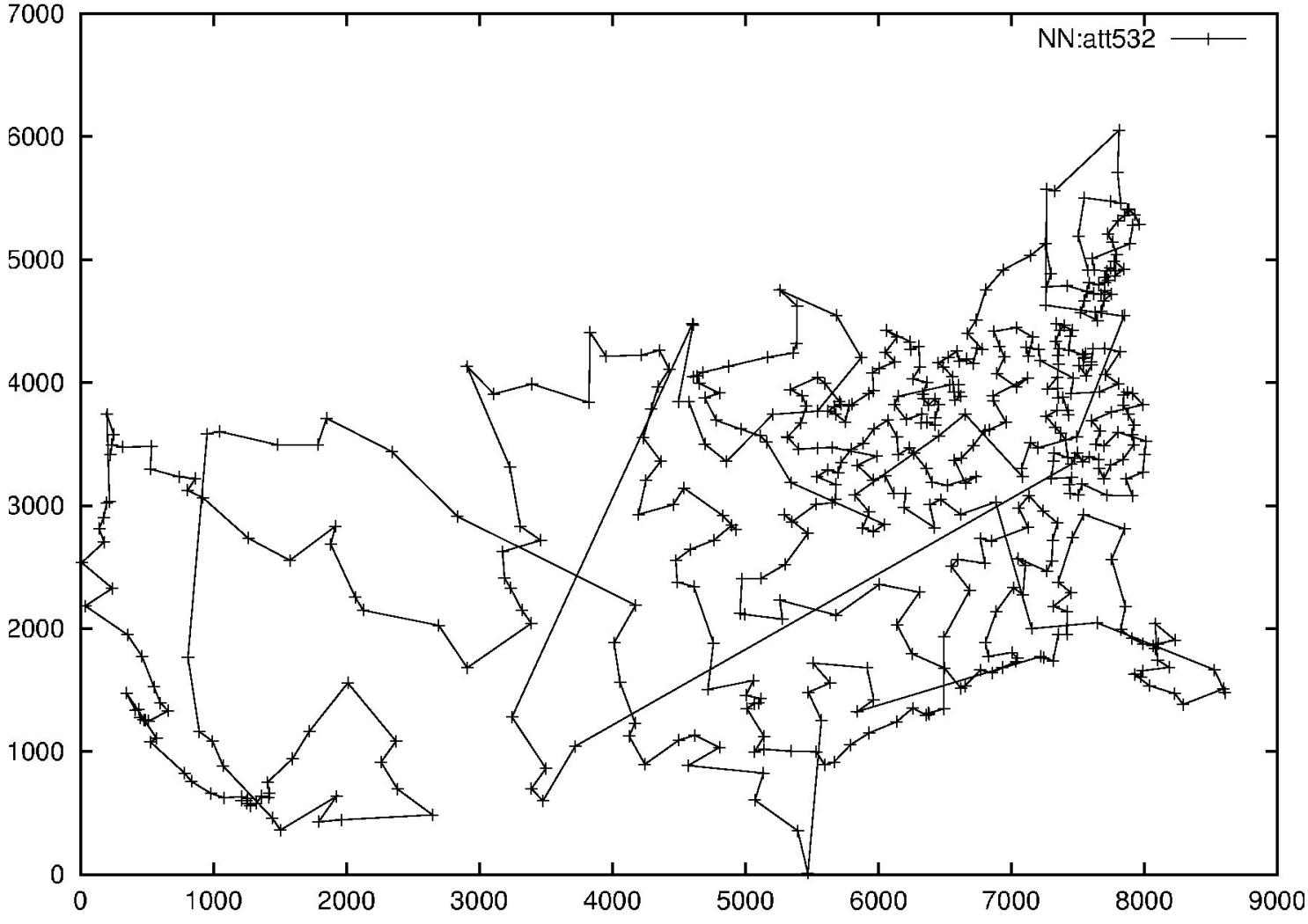
begin
  $x \leftarrow$ empty solution;
  while x is not a complete solution **do**
    $e \leftarrow$ currently the most beneficial extension for x;
    $x \leftarrow x \oplus e$;
  end;
end

Examples for construction heuristics

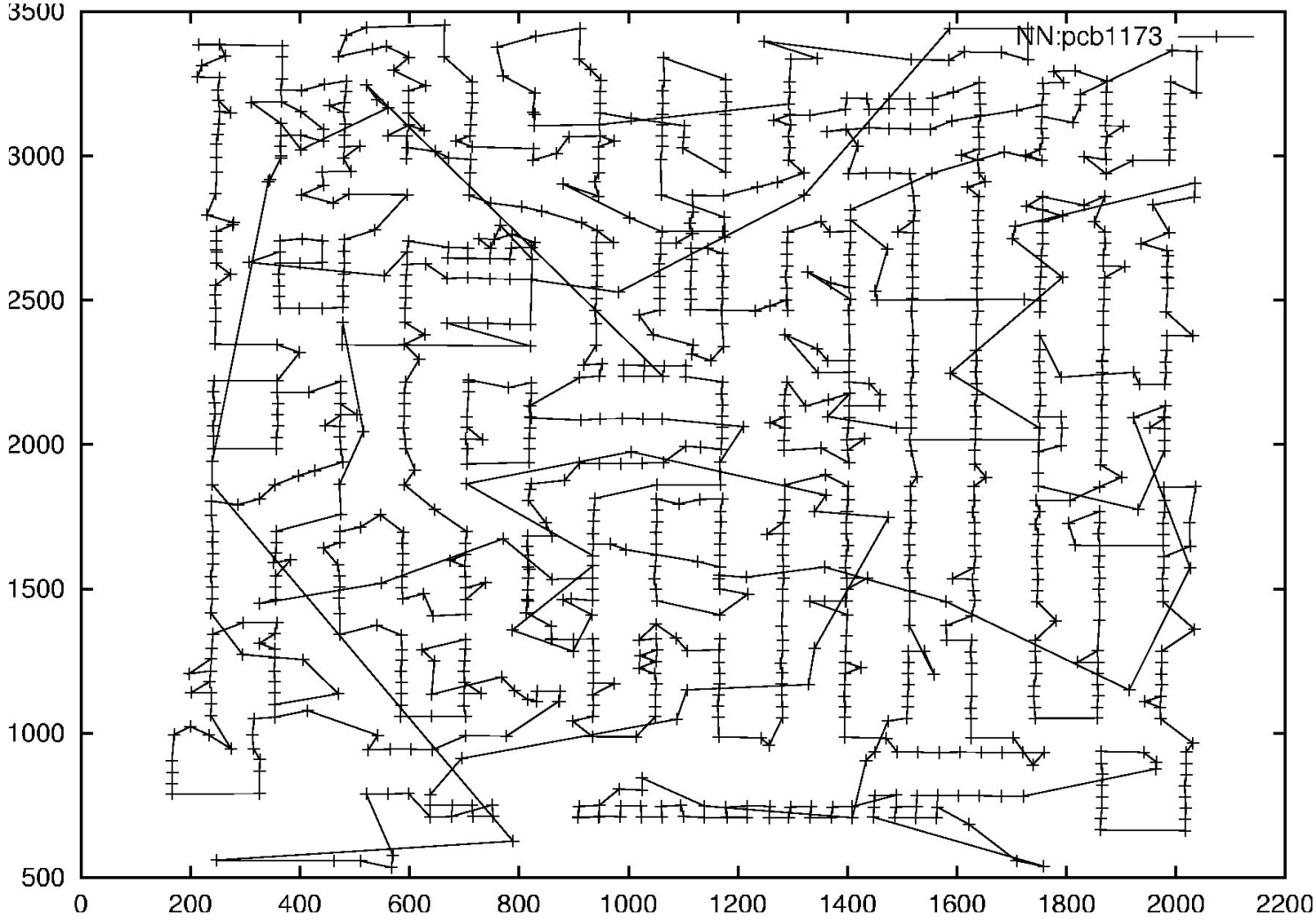
- Minimum spanning tree: Prim's und Kruskal's algorithms (greedy, compute the global optimum)
- 0/1-Knapsack problem: Greedy heuristics
- TSP:
  - Nearest neighbor heuristic (greedy)
  - Insertion heuristic (greedy)
  - Spanning tree heuristic (approx. ratio of 2 for Eucl. instances)
  - Christofides heuristic (approx. ratio of 1.5 for Eucl. instances)

Some of these algorithms can guarantee an approximation ratio
$\rightarrow$ **approximation algorithms**

Example for nearest neighbor heuristic (1)



Example for nearest neighbor heuristic (2)



Insertion heuristics for the TSP

- Start with a tour of 2 cities
  (or in case of Euclidean instance: convex hull)
- Add remaining cities stepwise until the tour is completed
- Selection strategies for the next city:
  - Random insertion: Insert city at random
  - Nearest insertion: Insert city with minimum distance to a city in the tour
  - Farthest insertion: Insert city whose minimum distance to a city in the tour is maximum
- Insertion strategies:
  - After the nearest city in the tour
  - Inducing minimum increase of tour length

Part II

Single Solution Based Metaheuristics

Simple Local Search and Variants

Simple local search

procedure local search

begin
  $x \leftarrow$ initial solution;
  repeat:
    choose an $x' \in N(x)$;
    if $f(x') \leq f(x)$ **then**
      $x \leftarrow x'$;
  until stopping criteria satisfied;
end

- $N(x)$: **Neighborhood of** x

(We always assume minimization.)

Elements of local search

- Definitions for solution representation and objective function $f(x)$
- Initial solution
- Neighborhood structure, i.e., which solutions are neighbors?
- Step function that defines the order the neighborhoods are searched and which neighbor is selected
- Stopping criteria

Neighborhood structure

Definition (Neighborhood structure)

A **neighborhood structure** is a function $N : S \to 2^S$,
that assigns to each solution $x \in S$ a set of **neighbors** $N(x) \subseteq S$.

$N(x)$ is also called the **neighborhood (of x)**.

- A neighbor is generated by the application of a **move** operator
- Possible visualization as neighborhood graph
- Size of neighborhood vs. complexity of the search

Example: Boolean satisfiability problem (SAT)

Definition (Satisfiability problem)

Given: A boolean logic formula in conjunctive normal form, e.g.,

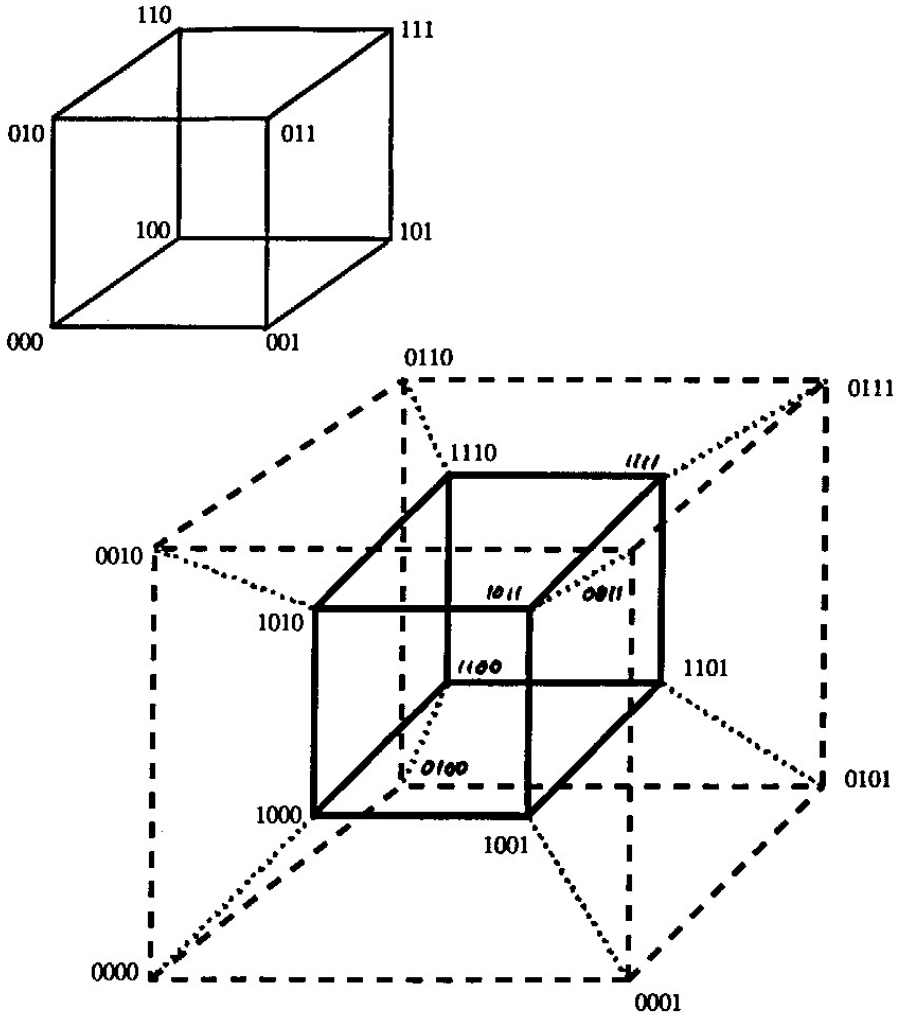
$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$$

Wanted: Assignment of 1 ("true") and 0 ("false") so that all clauses are satisfied.

Optimization variant:
MAX-SAT: Maximize the number of satisfied clauses

- Solution representation: Binary vector x
- Conceptually one of the simplest problems
- $\mathcal{NP}$-complete
- One of the most fundamental problems in computer science

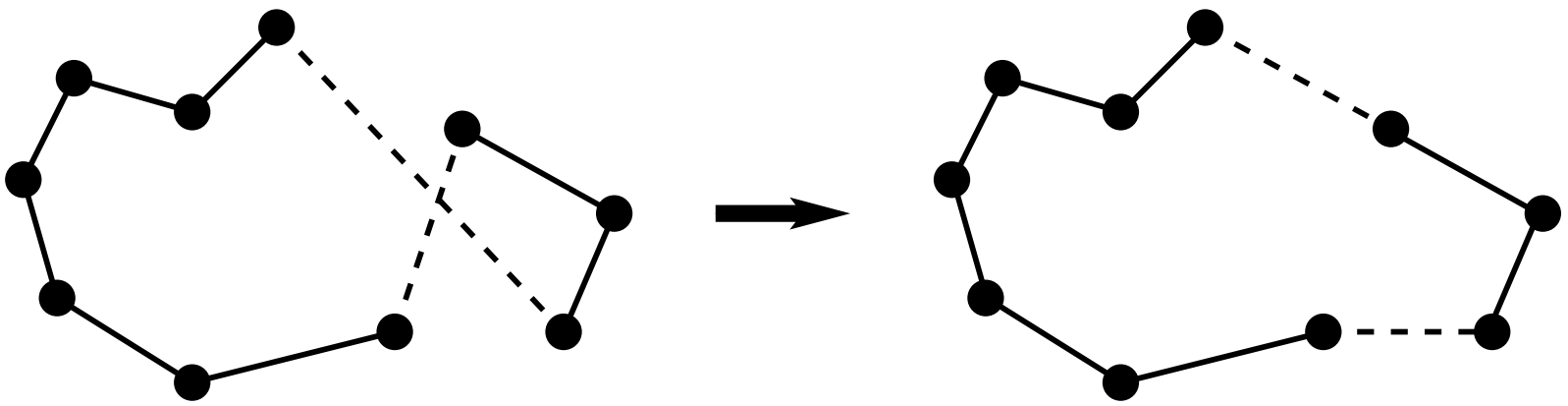
Example: Neighborhood structures of binary vectors



- ***k*-flip neighborhood:** solutions have a Hamming distance $\leq k$, i.e., they differ in at most k bits
- Neighborhood size: $O(n^k)$
- Picture: The search space of binary vectors of length n can be illustrated as an n-dimensional hypercube.

Example: Symmetric TSP

- Initial solution is any tour
- **k-exchange (k-opt) neighborhood:** tours that differ in at most k edges
- Neighborhood size: $O(n^k)$

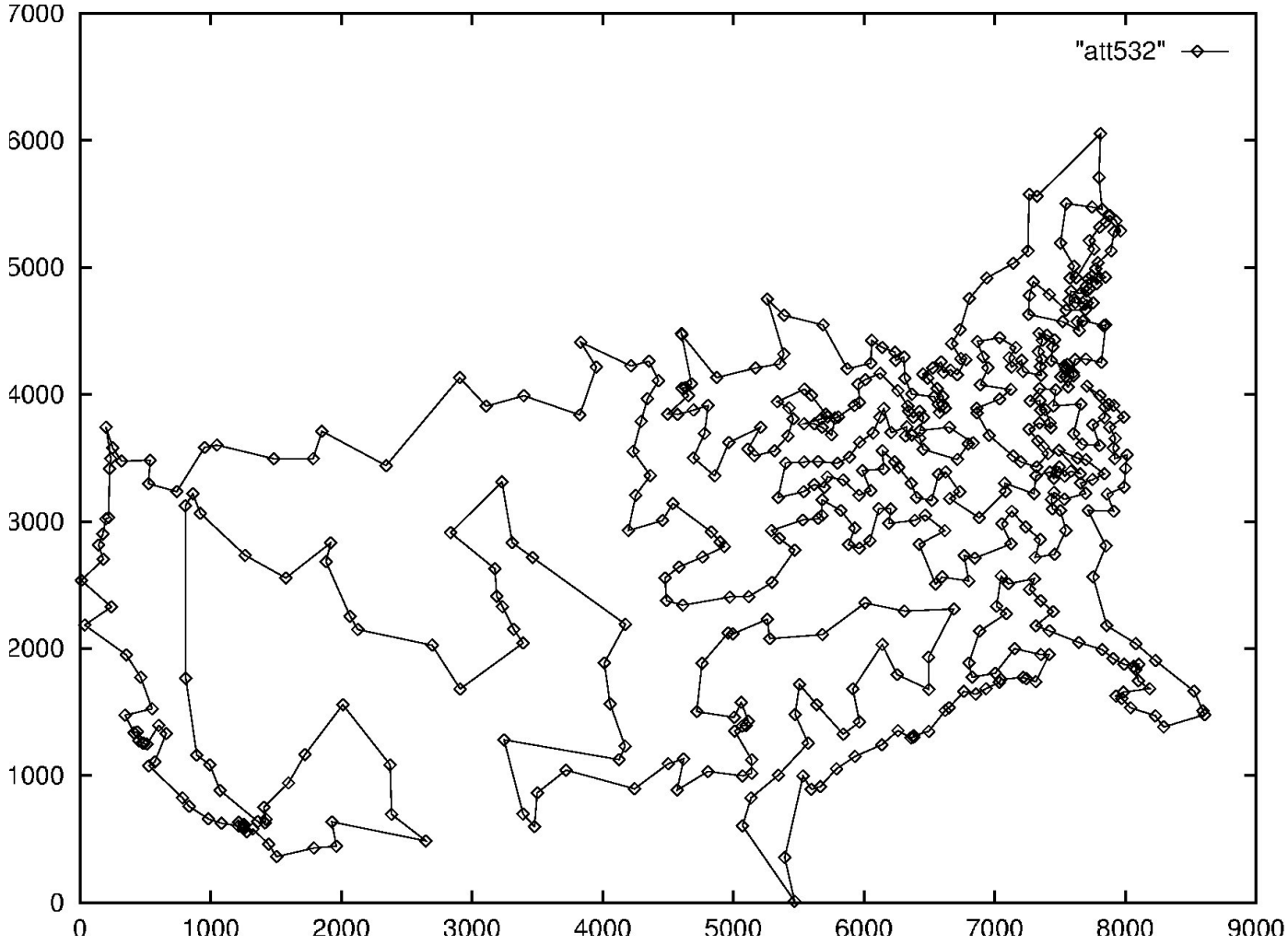


Example: 2-exchange move for the TSP

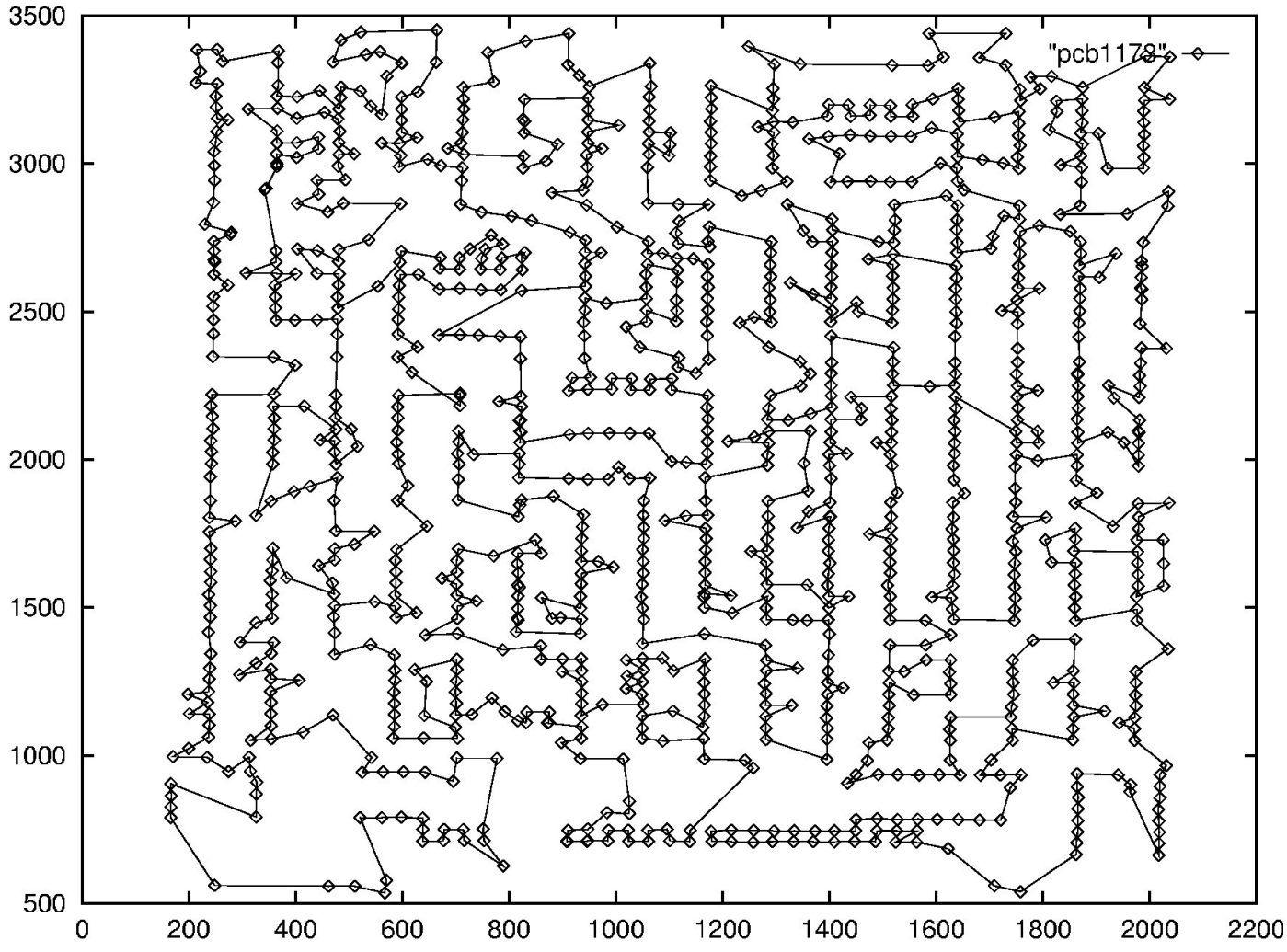
Example: Symmetric TSP (contd.)

- **k-opt solution**
  - $\rightarrow$ cannot be improved by the exchange of $\leq k$ edges
- with greater k the solution quality and running time increases
  - $\rightarrow$ 4-opt is already too "expensive" for practical use
- several other neighborhoods:
  - node-displacement
    - $\rightarrow$ better suited for asymmetric TSP
  - 2.5-opt
  - Lin-Kernighan

2-opt solution for TSP (1)



2-opt solution for TSP (2) – ("nearly 2-optimum")



Step function (choice of x')

Random neighbor:

Always generate a random neighbor from $N(x)$.

Next improvement:

Search $N(x)$ in a specified order, take first solution, which is better than x

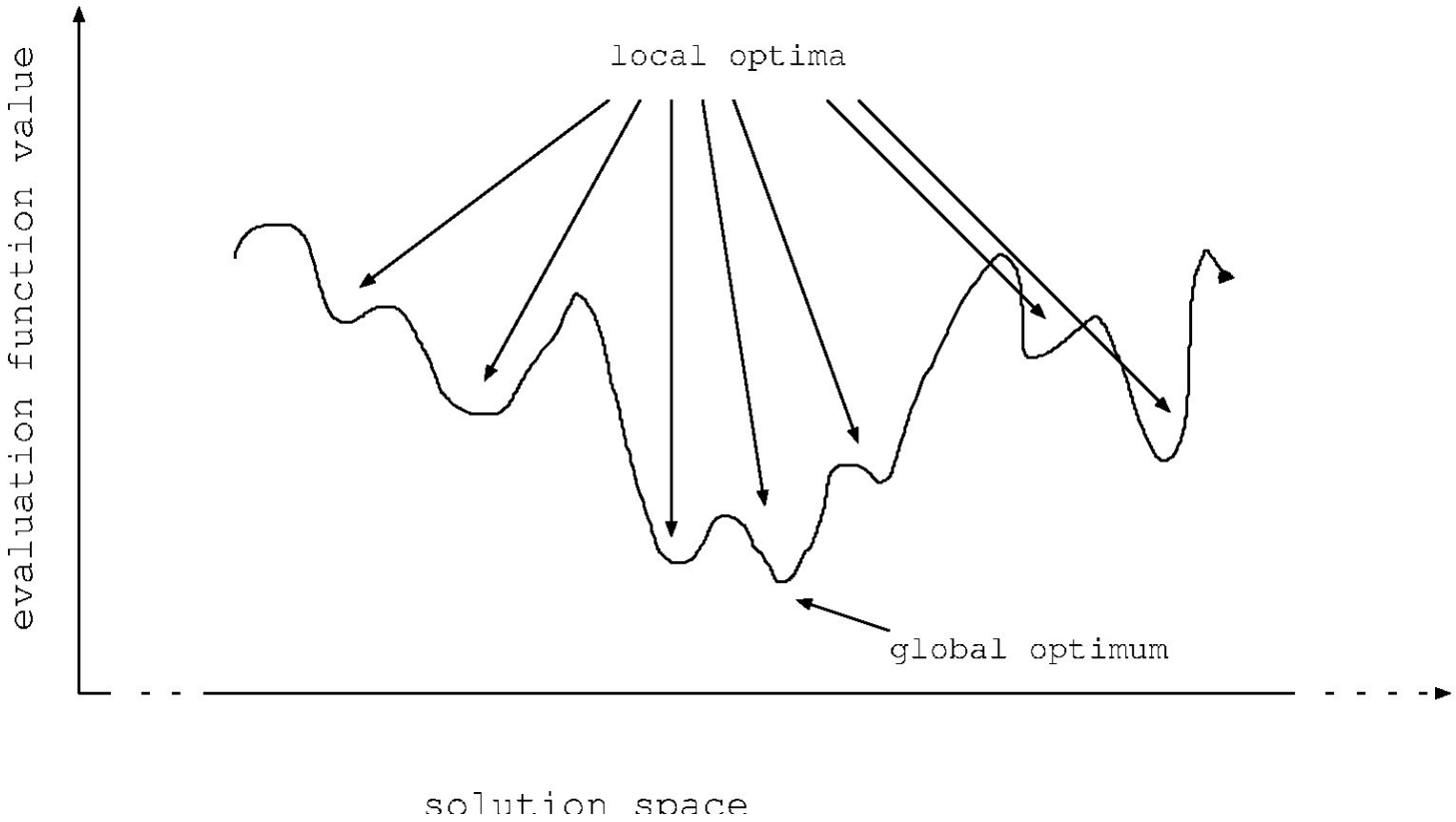
Best improvement:

Search through $N(x)$ completely and take the best neighbor

The selection process has great influence on the performance,
but no strategy is always better than the others!

Problem of local search

Simple local search usually finds solutions that are **local optima**:



Problem of local search

Definition

A **local minimum with respect to a neighborhood structure** N is a solution x' with: $f(x') \leq f(x)\ \forall x \in N(x')$.

But there could be an $x'' \in S$ with $f(x'') < f(x')$.

- Neighborhood structure defines, which solutions are local minima
- Global minima are special local minima
- What we want:
  Small exact neighborhoods where each local minimum is also a global minimum

Simple improvements

1. Enlarge neighborhood

- Neighborhoods with less and better local minima with larger **basins of attraction**:

  basin of attraction $A(x)$*:*
  Subset of all solutions for which the application of local search results in the local optimum x*.*

- Leads, generally, to better results but increases the running time drastically.

- E.g., k-exchange for the TSP: $O(n^k)$ neighbors
  in practice $k > 3$ is barely accomplishable or useful

2. Iterated local search (ILS)

procedure iterated local search

begin
  $x \leftarrow$ initial solution;
  $x \leftarrow$ local search (x);
  repeat:
    $x' \leftarrow$ **perturbation** (x);
    $x' \leftarrow$ local search (x');
    if $f(x') \leq f(x)$ **then**
      $x \leftarrow x'$;
  until stopping criteria satisfied;
end

Perturbation:
Random move in a larger neighborhood or
a series of random moves in smaller neighborhoods

3. Multistart local search

- Perform local search starting with different initial solutions
- Very simple improvement, but requires more time
- Effective if there are relatively few local minima or the global optimum has a large *basin of attraction*

Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Randomized Adaptive Search Procedure (GRASP)

Basic idea:

- A greedy construction heuristic is systematically randomized to generate a set of good initial solutions for a subsequent local search
- First approaches from Feo and Resende (1989)

GRASP

procedure GRASP

begin
  $x^* \leftarrow \emptyset$;
  repeat:
    $x \leftarrow$ randomized greedy heuristic;
    $x' \leftarrow$ local search (x);
    if $f(x') \leq f(x^*)$ **then**
      $x^* \leftarrow x'$;
  until stopping criteria satisfied;
  return x^*;
end

Randomization of greedy construction heuristics

procedure randomized greedy heuristic

begin
  $x \leftarrow$ empty solution;
  while x is no complete solution **do**
    build CL;
    build RCL from CL;
    $e \leftarrow$ random selection from RCL;
    $x \leftarrow x \oplus e$;
  end;
end

CL: **candidate list** = elements that can be included in partial solution
RCL: **restricted candidate list** = subset of CL with promising elements

Approaches to determine *RCL*

- Take the k cheapest solution components
- Let $c_{\min}$ be the minimum and $c_{\max}$ the maximum component costs within *CL*.
  Add $e \in CL$ to *RCL* if

$$c(e) \leq c_{\min} + \alpha \cdot (c_{\max} - c_{\min}), \quad \alpha \in [0, 1]$$

Note:

$\alpha = 0 \quad \hat{=} \quad$ greedy construction heuristic

$\alpha = 1 \quad \hat{=} \quad$ random solution construction

Subsequent local search

- typically *best improvement* step function

Benefits if local search is started from multiple randomized greedy solutions:

- Generally, a lower variance and a higher solution quality than performing local search on random solutions
- Local search usually requires few iterations
  - fast
  - in the same time more initial solutions can be improved
- Randomization and repetition leads to better results than performing the greedy heuristic and local search only once

Extensions for GRASP

- Select from *RCL* with Bias:
  - Components within *RCL* are selected with different probabilities, e.g., proportional to $1/c(e)$
- Automatic adjustment of α:
  - Set parameter randomly
  - Reactive GRASP
- Usage of a memory
  - Use an elite of good solutions to control the construction process
- Simple parallelization

Example: Quadratic assignment problem (QAP)

Definition (Quadratic assignment problem)

Given: n facilities, n locations

$D = (D_{i,j}) \in \mathbb{R}^{n \times n}$: distances between the locations

$F = (F_{i,j}) \in \mathbb{R}^{n \times n}$: flow between the facilities

Wanted: Assign all facilities to different locations, i.e.,
a permutation $x : \{1, \ldots, n\} \to \{1, \ldots, n\}$ with minimum sum of distances

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} F_{i,j} D_{x_i, x_j}$$

Note: The objective function is quadratic, hence the name.

Example: GRASP for QAP

(Li, Pardalos, Resende, 1994)

- Greedy function: costs of interaction between new facility/location pairs and the already existing pairs in the partial solution
- *RCL* size is restricted through k
- Select facility/location pair randomly from *RCL*
- 2-exchange local search
  (i.e., neighborhood: exchange two facilities)

Example: Boolean satisfiability problem (SAT)

Definition (Satisfiability problem)

Given: A boolean logic formula in conjunctive normal form, e.g.,

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$$

Wanted: Assignment of 1 ("true") and 0 ("false") so that all clauses are satisfied.

Optimization variant:
MAX-SAT: Maximize the number of satisfied clauses

- Solution representation: Binary vector x
- Conceptually one of the simplest problems
- $\mathcal{NP}$-complete
- One of the most fundamental problems in computer science

Example: GRASP for SAT

(Resende, Feo, 1996)

- Greedy function: number of unsatisfied clauses that would be satisfied after instantiating a variable x_i with 1 or 0
  - Let Φ_i^+: set of additionally satisfied clauses, if $x_i = 1$
  - Let Φ_i^-: set of additionally satisfied clauses, if $x_i = 0$
- *RCL* restrictions:
  - Let $\Phi^* = \max_{i=1,\ldots,n}\{|\Phi_i^+|, |\Phi_i^-| \,|\, x_i \text{ not instantiated yet}\}$
  - $|\Phi_i^+| \geq \alpha \cdot \Phi^* \;\rightarrow\; (x_i = 1) \in RCL$
  - $|\Phi_i^-| \geq \alpha \cdot \Phi^* \;\rightarrow\; (x_i = 0) \in RCL$
  - E.g., $\alpha = 0.5$

Example: GRASP for SAT (contd.)

- Selection from *RCL*:
  - If an unsatisfied clause contains only one non instantiated variable, then try to satisfy this clause
  - Otherwise, select an element from *RCL* randomly
- Local Search:
  - 1-flip neighborhood structure
  - Best improvement

Conclusions for GRASP

- A “simple to implement” extension of construction heuristics
- Few parameters
- Many applications
- Often relatively fast while producing relatively good solutions

Variable Neighborhood Search (VNS)

Variable neighborhood search

(Hansen and Mladenovic, 1997)

This method takes advantage of the following properties:

1. A local optimum with respect to one neighborhood structure is not necessarily a local optimum with respect to another.

2. A global optimum is a local optimum with respect to all possible neighborhood structures.

3. For many problems local optima are relatively close together.

Variable neighborhood descent (VND)

Neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}$ are changed systematically.

procedure variable neighborhood descent (x)

begin
  $l \leftarrow 1$;
  repeat:
    find an x' with $f(x') \leq f(x''),\ \forall x'' \in \mathcal{N}_l(x)$;
    if $f(x') < f(x)$ **then**
      $x \leftarrow x'$;
      $l \leftarrow 1$;
    else
      $l \leftarrow l + 1$;
  until $l > l_{\max}$;
  return x;
end

Variable neighborhood descent (contd.)

- Step function: usually *best* or *next improvement*
- A solution will be generated that is a **local optimum according to all neighborhood structures**.
- Order of neighborhood structures:
  - mostly, according to the size or complexity of the evaluation
  - or randomly
- Example: Vehicle routing problem (VRP)
  - $\mathcal{N}_1(x)$: 2-exchange within a single tour
  - $\mathcal{N}_2(x)$: Move city from one tour to another
  - $\mathcal{N}_3(x)$: Exchange two cities between two tours
  - $\mathcal{N}_4(x)$: 3-exchange within a single tour
  - ...

Reduced variable neighborhood search (RVNS)

- RVNS is a framework to **escape local optima**.
- The basic idea is to select neighbors randomly from systematically increasing neighborhoods.
- Again, there is a pool of different neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}$, but with the following properties:
  - There are also significantly larger neighborhoods, which cannot be fully searched in general
  - The application order is typically ranked following the increasing complexity or according to the "structural distance between the new solutions and the initial solution".

procedure reduced VNS (x)

begin
  repeat:
    $k \leftarrow 1$;
    repeat:
      shaking: generate a random $x' \in \mathcal{N}_k(x)$;
      if $f(x') < f(x)$ **then**
        $x \leftarrow x'$;
        $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
    until $k > k_{\max}$;
  until stopping criteria satisfied;
  return x;
end

About the neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}$

- At least theoretically, to avoid getting stuck in a local optimum the union of all $\mathcal{N}_1(x), \ldots, \mathcal{N}_{k_{\max}}(x)$ should cover the feasible search space S:

$$S \subseteq \mathcal{N}_1(x) \cup \mathcal{N}_2(x) \cup \cdots \cup \mathcal{N}_{k_{\max}}, \ \forall x \in S$$

- The individual neighborhoods can but need not be disjoint.
- Often nested neighborhoods are the better choice:

$$\mathcal{N}_1(x) \subset \mathcal{N}_2(x) \subset \cdots \subset \mathcal{N}_{k_{\max}}, \forall x \in S$$

  This can be achieved for instance by implementing the shaking in neighborhood $\mathcal{N}_k$ as k consecutive moves in neighborhood $\mathcal{N}_1$.

Basic Variable Neighborhood Search (BVNS)

RVNS is extended by a simple local search:

procedure Basic VNS (x)

begin
  repeat:
    $k \leftarrow 1$;
    repeat:
      shaking: generate a random $x' \in N_k(x)$;
      $x' \leftarrow$ **local search**(x')**;**
      if $f(x') < f(x)$ **then**
        $x \leftarrow x'$; $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
    until $k > k_{\max}$;
  until stopping criteria satisfied;
  return x;
end

General variable neighborhood search (GVNS)

- Combines BVNS with VND as local search method.
- The neighborhood structures of VNS ($N_1, \ldots, N_{k_{\max}}$) and of VND ($\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}$) are not identical!

procedure general VNS (x)

begin
  repeat:
    $k \leftarrow 1$;
    repeat:
      shaking: generate a random $x' \in N_k(x)$;
      // $x' \leftarrow$ local search using VND (x'):
        $l \leftarrow 1$;
        repeat:
          find an x'' with $f(x'') \leq f(x'''),\ \forall x''' \in \mathcal{N}_l(x')$;
          if $f(x'') < f(x')$ **then**
            $x' \leftarrow x''$; $l \leftarrow 1$;
          else
            $l \leftarrow l + 1$;
        until $l > l_{\max}$;
      if $f(x') < f(x)$ **then**
        $x \leftarrow x'$; $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
    until $k > k_{\max}$;
  until stopping criteria satisfied;
  return x;
end

Variable neighborhood decomposition search (VNDS)

Variation or extension of (basic) VNS.

procedure VNDS (x)

begin
  repeat:
    $k \leftarrow 1$;
    repeat:
      shaking: generate a random $x' \in N_k(x)$;
      $Y \leftarrow$ Set of variables where x is different from x';
      $x' \leftarrow$ local search or improvement heuristic(x'), where the variables $x \setminus Y$ are fixed, i.e., only variables in Y are optimized;
      if $f(x') < f(x)$ **then**
        $x \leftarrow x'$; $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
    until $k > k_{\max}$;
  until stopping criteria satisfied;
  return x;
end

Simulated Annealing

Simulated annealing (SA)

- First approach in local search to escape local optima (Kirkpatrick et al. 1983)
- Inspired by the physical process of annealing to obtain a strong crystalline structure
  $\rightarrow$ **nature inspired method**
- Basic idea:
  Even worse solutions are accepted with a certain probability.
- Usually, is based on **random neighbor step function.**

procedure simulated annealing

begin
  $t \leftarrow 0$;
  $T \leftarrow T_{\text{init}}$;
  $x \leftarrow$ initial solution;
  repeat:
    repeat:
      choose a $x' \in N(x)$ randomly;
      if x' is better than x **then**
        $x \leftarrow x'$;
      else
        if $P < e^{-|f(x')-f(x)|/T}$ **then**
          $x \leftarrow x'$;
      $t \leftarrow t + 1$;
    until equilibrium condition satisfied;
    $T \leftarrow g(T, t)$;
  until stopping criteria satisfied;
end

Metropolis-criterion:

$$P < e^{\frac{-|f(x')-f(x)|}{T}}$$

$P \ldots$ random number $\in [0, 1)$

Annealing:

Temperature T
is slowly degraded

Parameters for the cooling schedule

- Initial temperature T_{init}
  - e.g., based on known bounds: $T_{\text{init}} = f_{\text{max}} - f_{\text{min}}$
  - e.g., so that $\approx 3\%$ of the moves are rejected at the beginning
- Cooling schedule $g(T, t)$
  - **Geometric cooling:**
    $g(T, t) = T \cdot \alpha$, $\alpha < 1$, (e.g., $\alpha = 0.95$)
- Number of iterations to satisfy equilibrium condition
  - e.g., multiple of the neighborhood size
- Stopping criteria
  - e.g., reaching a final temperature
  - or no improvement over τ levels of temperature

Example: simulated annealing for the TSP

- Simple implementation:

  (Johnson, McGeoch, 1997)

  - Start with a random tour
  - 2-exchange neighborhood
  - Cooling schedule:
    - T_{init} so that $\approx 3\%$ of moves are rejected
    - Geometric cooling, $\alpha = 0.95$
    - Iterations to satisfy equilibrium: $n(n-1)$
    - Stopping criteria: 5 temperature levels without improvement

$\rightarrow$ relatively poor results

Example: simulated annealing for the TSP (contd.)

- Significant improvements
  - Use construction heuristic for initial solution
  - Restriction of the neighborhood to promising moves
- Nevertheless
  Not competitive compared to leading TSP methods.

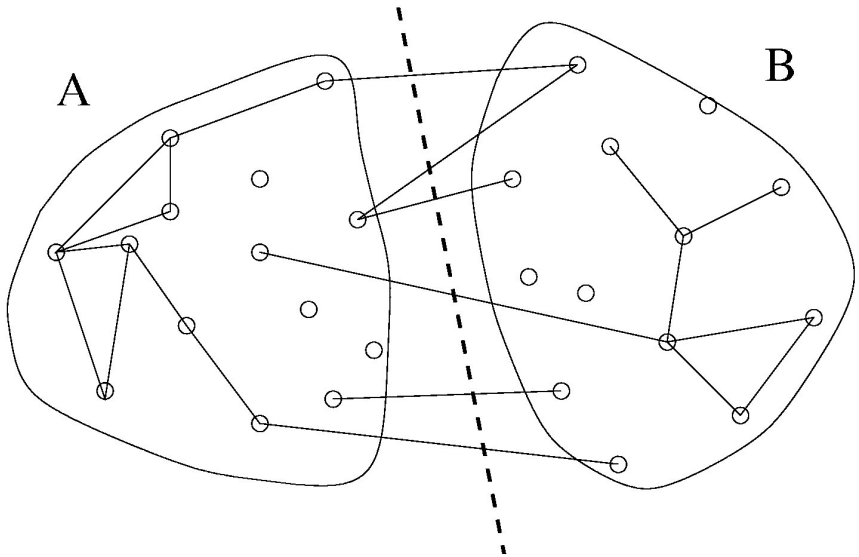
Example: SA for graph bipartitioning

Definition (graph bipartitioning)

Given: graph $G = (V, E)$

Wanted: partitioning of G into two vertex sets V_1, V_2,
with $|V_1| = |V_2|$, $V_1 \cap V_2 = \emptyset$, and $V_1 \cup V_2 = V$,

minimize $|\{(u, v) \in E \mid u \in V_1 \wedge v \in V_2\}|$



Example: SA for graph bipartitioning (contd.)

- Solution representation:

  Characteristic vector $x = (x_1, \ldots, x_n)$, $n = |V|$
  $\rightarrow$ node i is assigned to set V_{x_i}

- Simulated annealing (Johnson et al., 1989)
  - 2-node-exchange neighborhood
  - Random initial solution
  - Similar cooling schedule as for the TSP
  - One of the first applications of SA

Example: SA for graph partitioning (contd.)

- Improvements:
  - **Restriction of the neighborhood and permission of infeasible solutions**
  - *Modified objective function:*
    $f(V_1, V_2) = |\{(u, v) \in E \mid u \in V_1 \wedge v \in V_2\}| + \gamma(|V_1| - |V_2|)^2$
    γ: imbalance factor
  - *New neighborhood:*

    Move single node into the other set
    - $|N(x)| = n$ instead of $n^2/4$
- Comparison with Kernighan-Lin heuristic for this problem:
  - SA better for random graphs
  - Kernighan-Lin better for Euclidean graphs

Enhancements of SA

- Nonmonotonic cooling schedule
  - *"Reheating"*
- Dynamic vs. static cooling schedule
  - e.g., equilibrium state satisfied if no improvement was found after a certain number of iterations
- Deterministic investigation of the neighborhood
- Combination with other methods
- Parallelization

Conclusions for SA

- SA has been applied to several hundred applications.
- SA is one of the most commonly used and also theoretically most studied metaheuristics.
- Often it is simple to implement.
- SA delivers good results for many problems, but usually no excellence results compared to other leading methods.
- SA is relatively time consuming.

Tabu Search

Basic idea of tabu search (TS)

(Glover 1986; Hansen 1986: "steepest ascent/mildest descent")

- Based on a **memory (history)** of the previous optimization process, which is used to escape local optima, i.e., the **tabu list**
- **Avoids cycles** by forbidding the revisit of solutions in the memory
- Often **deterministic driven search**
- Often **best neighbor step function**:
  at each step the best allowed neighbor is accepted,
  even if this solution is worse than the current

Tabu search – basic algorithm

procedure tabu search

```
begin
   TL ← ∅; // tabu list of length t_L
   x ← initial solution;
   repeat:
      X' ← subset of N(x) regarding TL;
      x' ← best solution in X';
      add x' to TL;
      remove element from TL, which is older than t_L iterations;
      x ← x';
      if x is best solution found so far then
         store x as best solution;
   until stopping criteria satisfied;
end
```

Memory: the tabu list

- Store complete solutions explicitly
- Alternative: Store solution attributes (i.e., **tabu attributes**) of visited solutions
  $\rightarrow$ solutions are forbidden (**tabu**), if they contain tabu attributes
- Often, tabu attributes are represented as attributes of moves; the reverse moves stored in the tabu list are forbidden for the next t_L iterations
- Important parameter: **length of tabu list** t_L
- Often several tabu lists for different tabu attributes are used

Parameter: length of tabu list

- Choice of t_L is often very critical!
- Too short tabu lists would allow cycles
- Too long tabu lists would forbid many possible solutions and thus greatly restrict the search
- Appropriate length usually is problem specific, must be determined experimentally, or
  - Choose randomly from an interval every few iterations
  - Lifespan of tabu attributes is set randomly for each move

Aspiration criteria

- Sometimes a not visited solution could be forbidden (i.e., its attributes are in the tabu list) even if it is very good.
- **Aspiration criteria** overwrite the tabu status of "interesting" solutions.
- **Example** for often used aspiration criteria:
  - A forbidden solution is better than the best found solution.

Example: Tabu search for MAX-SAT

(Hansen, Jaumard, 1990; Selman, Kautz, 1994)

Neighborhood structure: 1-flip neighborhood

Tabu attributes: modified variables

Tabu criterion: changing a variable is forbidden for t_L iterations

Aspiration criteria: if a move leads to a new best solution then overwrite tabu status.

Additional "tie-breaking" criterion: If there are several best solutions, then break this "tie" in favor of the one with the variable, which violates the most clauses.

Example: TS for the graph coloring problem

Definition (graph coloring problem)

Given: graph $G = (V, E)$

Wanted: to each node $v \in V$ assign a color $x_v \in \{1, \ldots, k\}$, so that $x_u \neq x_v, \ \forall (u, v) \in E$

Optimization variants:

- Minimize the number of needed colors k
- Minimize the number of "violated" edges

Example: TS for the graph coloring problem (contd.)

Evaluation criterion: minimize number of "violated" edges

Neighborhood: change color of one node

Tabu attributes: pairs (v, j) with $v \in V$, $j \in \{1, \ldots, k\}$,
i.e, color assignment j to node v

Tabu criterion: if move $(v, j) \rightarrow (v, j')$ is applied, then attribute (v, j) is forbidden for t_L iterations

Aspiration criterion: if move results in a new best solution, then overwrite tabu status

Restriction of the neighborhood: consider only nodes for color assignments, which are involved in a constraint violation

Reactive tabu search

(Battiti und Tecchiolli, 1994)

- Originally developed for QAP and 0/1-knapsack problem
- **Length of tabu list is adaptive:**
  - Increased by constant factor, if cycle is detected
  - Let m be the average cycle length
  - Reduction by constant factor, if last increase was more than m iterations ago
- **Diversificationmechanism:**
  - Is performed, if the number of revisited solutions, exceeds a predefined threshold
  - Execute a number of random steps (proportional to m)

Attribute based hill climber (ABHC)

(Whittley und Smith, 2004)

- Originally developed for TSP and QAP
- Uses **regional aspiration** (vs. *global aspiration* for TS)
  - Value of an attribute is equivalent to the best solution found, which contains the attribute (or infinite).
  - Solution is accepted, if it is better than the value of at least one of its attributes
- Advantage: parameter free
- Competitive to TS and GLS

Multi level memory

Short term memory

- Essential part of TS, which was described previously

Medium term memory

- **Intensification** of the search
- "Restart" from *elite solutions*, e.g., with empty tabu list
- Solution attributes, which are part of several elite solutions, are fixed
- And several other strategies

Multi level memory (contd.)

Long term memory

- **Diversification** of the search
- Turns focus on hardly investigated regions of the search space
- Establishes solution attributes, which are used rarely
  - Used once followed by a "restart"
  - Continuous, by penalizing often- and rewarding rarely-used attributes
- And several other strategies (e.g., strategic oscillation)

Other variants of memory

In general, the following variants of memory can be used and exploited through appropriate rules

recency-based: When was an attribute used the last time?

frequency-based: How often was an attribute used?

quality-based: What is the observed influence of an attribute on the objective function value?

influence-based: What is the influence of an attribute on the solution structure and feasibility?

Conclusions for TS

- Many different strategies for the
  - memory
  - diversification
  - intensification
- Many successful applications
- Often excellent results and relatively fast
- Usually, a relatively time consuming fine tuning is necessary

Part III

Analysis of Metaheuristics

Testing Metaheuristics

Testing

To compare different optimization methods for a certain problem usually empiric tests are needed.

Performance measure:

- Solution quality
- Running time
- Memory consumption
- Robustness (with respect to multiple runs for the same instance, different types of instances, ...)

Testing (contd.)

Particularly when using stochastic methods you have to consider:

- Never conclude from the results of a single test run on the average behavior!
- Always use the results of several runs (≥ 30) and compute the mean (median) and the standard deviation.
- Never conclude from one problem instance on the average behavior of all instances!

If it is assumed that a method X provides better results (on average) than a method Y, then a **statistical test** should be performed to confirm this assumption.

Examples

- Student t-test (assumption that results are normally distributed!)
- Wilcoxon rank-sum test

Search space analysis: search landscapes

The analysis of the search space structure can provide information about the difficulties to search for the optimum and give hints for suitable methods and the choice of parameters.

- We define a **distance** $d(x, y)$ between two solutions $x, y \in S$:
  - often the minimum amount of moves to convert x into y
  - but also a more problem specific measure, e.g., the number of different edges for two TSP tours

- We imagine the search space as a **(search) landscape** with mountains, plains, hills, craters, valleys etc.;
  the height corresponds to the quality $f(x)$ of a solution x

Important characteristics of search landscapes

What characteristics might be instructive?

- Distribution (means, variance) of the **objective function values**
- Number and distribution of the **local minima**, their quality and relative location to the global optima
- Topology, size and variance of the **basins of attraction**
- **Ruggedness** of the search landscape; different measures can be found in the literature
- **Plateaus** and their size; *Neutrality*: degree about neighbors having the same quality

Fitness-distance correlation (FDC) analysis

Practical method that provides an indicator of how much *"the improvement of the solution quality leads to the global optimum"*.

Precisely:

- Determination of the (linear) correlation between the solution quality and the distance to a global optimum

Procedure:

- Choose a "typical" problem instance, for which all global optima are known.
- Generate systematically all solutions or k solutions randomly; determine the quality f_i and the distance to the nearest global optimum d_i.

- From the fitness-distance data a **correlation coefficient** is computed:

$$\rho_{f,d} = \frac{\sigma_{f,d}}{\sigma_f \cdot \sigma_d} = \frac{\sum_{i=1}^{k}(f_i - \overline{f})(d_i - \overline{d})}{\sqrt{\sum_{i=1}^{k}(f_i - \overline{f})^2 \sum_{i=1}^{k}(d_i - \overline{d})^2}}$$

- It is $-1 \leq \rho_{f,d} \leq 1$

Result:

- Strong correlation
  ($\rho_{f,d} \approx 1$ at minimization and $\rho_{f,d} \approx -1$ at maximization):
  - Points out that an increase in quality, in general, leads to the optimum.

  $\rightarrow$ "simple problem"

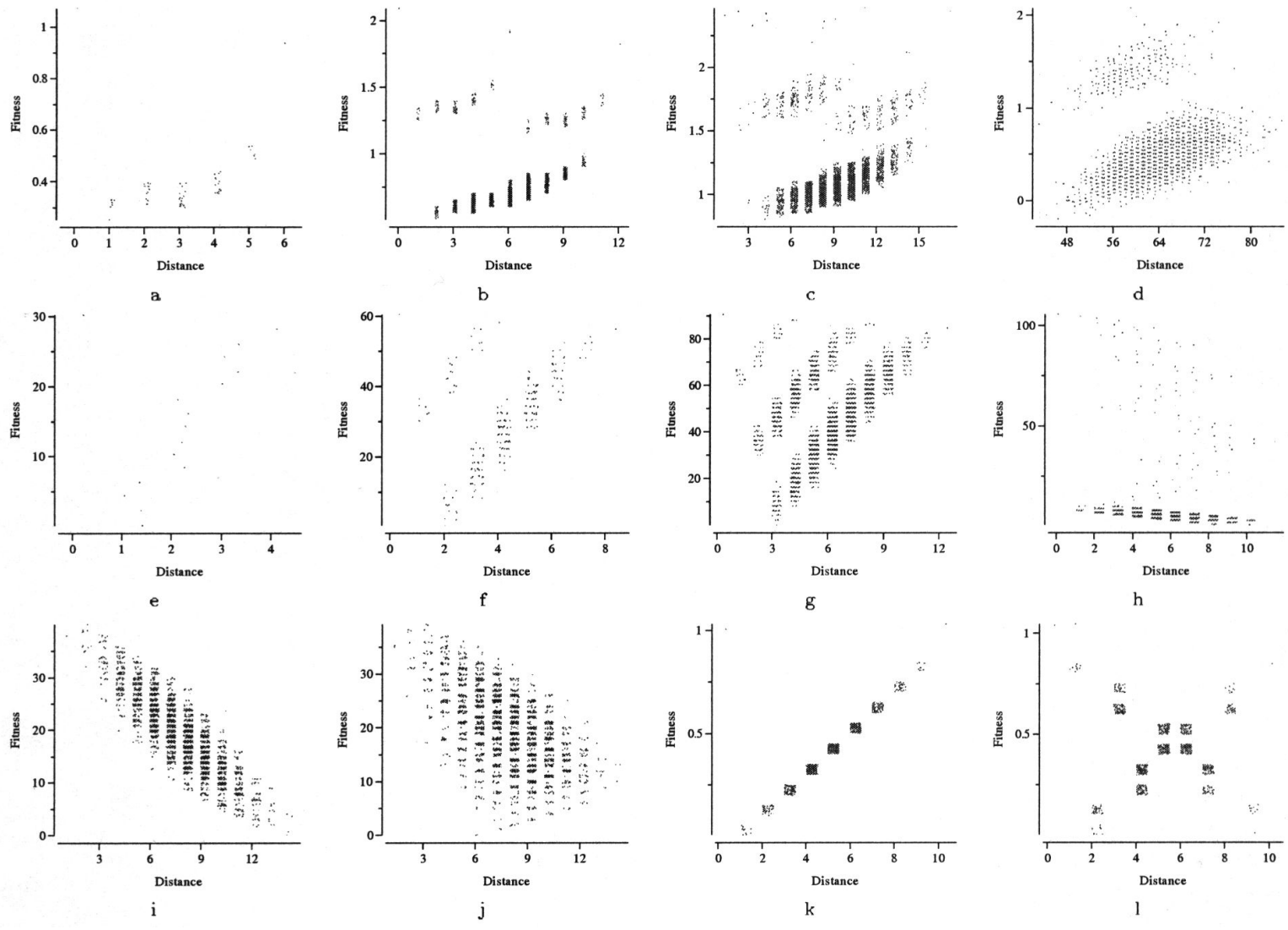
- Weak Correlation
  ($\rho_{f,d} \approx 0$):
  - Solutions with a high fitness are, in general, not close to an optimum.

  $\rightarrow$ "difficult problem"
  - Local search methods are often not more efficient than random search.

- Negative correlation
  ($\rho_{f,d} < 0$ at minimization and $\rho_{f,d} > 0$ at maximization):
  - An increase in quality leads, in general, *away* from the nearest optimum.

  $\rightarrow$ "*misleading* problem"
  - Random search can be more efficient than local search!

  - The correlation coefficient alone can sometimes be misleading. Sometimes X–Y plots can be more meaningful.
  - Often, the FDC analysis is performed for local optimal solutions, i.e., every solution is generated by simple local search.



Examples for X–Y plots

Part IV

Population Based Metaheuristics

Evolutionary Algorithms

Computer-implemented systems for problem solving where the key elements are computable models of natural, evolutionary processes

Charles Darwin, 1859:

On the Origin of Species

- Inheritance with Changes (Mutation)
- Natural Selection (Selection)

Gregor Johann Mendel, 1822–1884:

- Mendel's Laws of Heredity (Inheritance)

Basic Structure of EAs

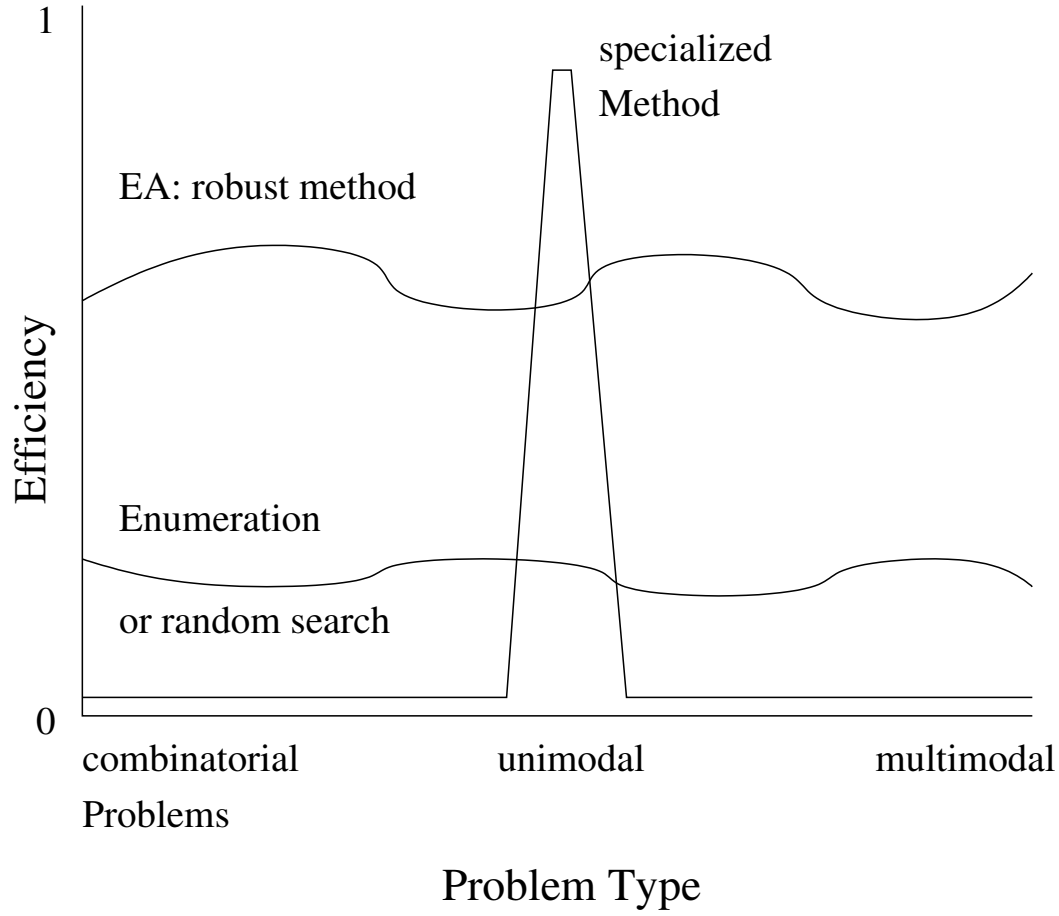
(or any population based approach in general)

procedure EA
begin
  Initialize(P); //P: population
  Evaluate(P);
  repeat:
    $Q \leftarrow$ GenerateNewSolutionsByVariation(P);
    Evaluate(Q);
    $P \leftarrow$ SelectBetterSolutions(P, Q);
end

Differences from Previous Methods

- Instead of working with a single solution, a whole population of candidate solutions is used
- Randomization is usually very important during initialization, variation and selection
- The basic principle is widely applicable
- Not only useful for combinatorial problems, but also for continuous parameter optimization and optimization of non-linear structures

- Robust: EAs are useful for a diverse set of problem types. In most cases, specialized methods are only able to achieve better results when considering some specific problems.



Important EA-Categories

- Genetic Algorithms
- Evolution Strategies
- Genetic Programming

The whole research field is called *Evolutionary Computing* nowadays.

Genetic Algorithms

Genetic Algorithms

Idea of J. H. Holland
from the 70s.

Advanced by (among others):
D. E. Goldberg,
K. A. De Jong,
Z. Michalewicz

Canonical GA

procedure GA
begin
  $t \leftarrow 0$;
  initialize($P(t)$);
  evaluate($P(t)$);
  while (**not** termination-condition) **do**
    $t \leftarrow t + 1$;
    $Q_s(t) \leftarrow$ select($P(t-1)$);
    $Q_r(t) \leftarrow$ recombine($Q_s(t)$);
    $Q_m(t) \leftarrow$ mutate($Q_r(t)$);
    $P(t) \leftarrow$ replace($P(t-1), Q_m(t)$);
    evaluate($P(t)$);
  done
end

Representation

Potential solutions have to be represented by a suitable data structure.
Some nomenclature was taken from the biological evolution theory:

Chromosome (Individual):
: contains all parameters for a candidate solution in encoded form

Gene:
: a chromosome is a collection of genes (often a vector);
every gene can assume a value of its domain

Allele:

assumed value of a gene

Genotype:

encoded form of a solution (=chromosome)

Phenotype:

decoded form of a solution;

the genotype can be seen as the phenotype's building plan

Population:

set of the GA's individuals

Initialization

The aim of population **initialization** is to create a **very diverse set of solutions**, so solution creation is mostly purely random or a randomized heuristic is utilized.

Fitness function $f(i)$
= Function that evaluates all individuals (Evaluation)

From now on it is assumed that:

higher fitness $\hat{=}$ better solution
lower fitness $\hat{=}$ worse solution
highest fitness $\hat{=}$ sought after optimal solution

Selection

= Commonly a randomized selection of parents to create the next generation

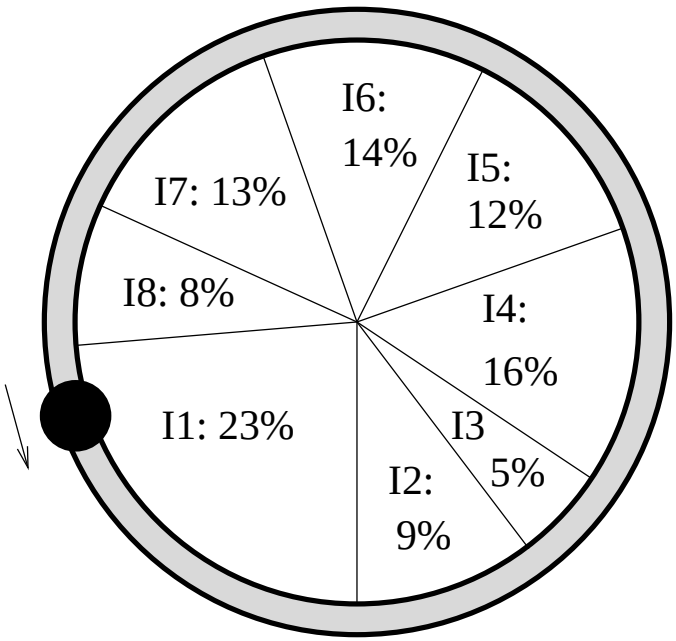
Following the principle of natural selection:

- Select better individuals with higher probability (more often) than worse individuals

Selection "pushes" the population in the direction of better solutions in a GA.

Fitness Proportional Selection

(roulette-wheel selection)



fitness proportional selection

Probability $p_s(i)$ for individual i to be selected (n... size of the population):

$$p_s(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

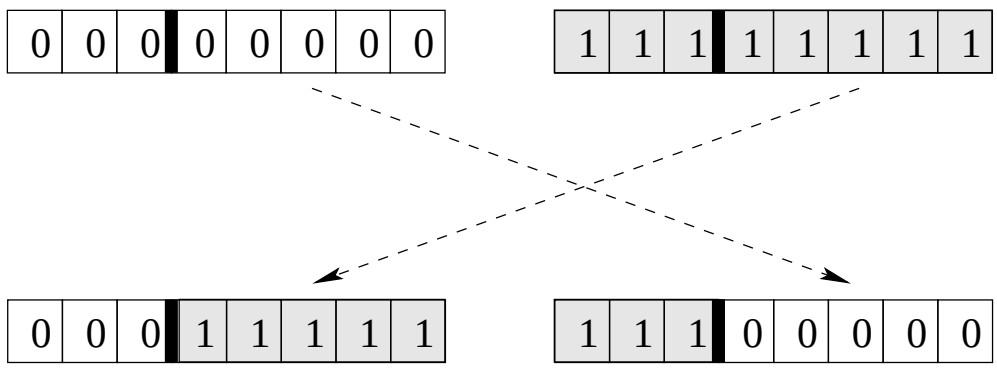
where $f(j) \geq 0$ and $\sum_{j=1}^{n} f(j) > 0$

Recombination

= *primary operator*

- Recombination creates new individuals by using the selected parents.
- A new individual (*offspring*) should contain the attributes of its parents. (*heritability*)

Example:



1-point crossover

- Recombination is either performed for all selected individuals or for a big fraction of them (e.g. $p_c = 0.8$).
- The selection of the parents ($\neq$ selection phase) and *crossover points* is purely random.

Mutation

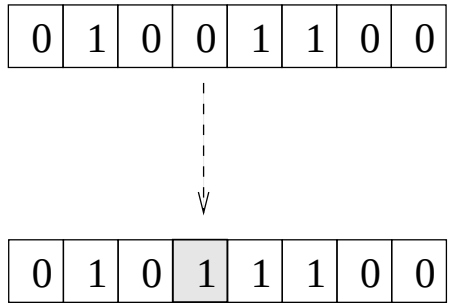
= *secondary operator*

Small, random changes, comparable with a *move* during local search.

Is used to seed new or lost gene material (alleles) into the population.

Example:

Every gene gets a new, random value with a small probability. Probability of mutation per gene for instance $p_m = 1/n$, $n \ldots$ chromosome length



Mutation

Replacement

- Selecting the survivors from the previous population and the new individuals
- The size of the population stays the same (usually)
- Two possible extremes:
  - *Generational repl.*: new individuals completely replace the old population
  - *Steady-state repl.*: only one new individual is produced and inserted into the population
- Everything in between is also possible

An example from daily life

Assume you have inherited a fast-food chain consisting of 4 restaurants and are completely inexperienced. To maximize profits, the following parameters have to be optimized:

O – Offering : The amount of different meals you offer can be small (0), medium (1) or large (2).

P – Price : Prices can be low (0), medium (1) or high (2).

Q – Quality : Your offerings can be of low (0), medium (1) or high (2) quality.

S – Service : There can be self-service (0), basic (1) or advanced (2) service.

Population: 4 restaurants
Generation: 1 month
Fitness: Profit at the end of the month
Encoding: OPQS

Size of the search space $= 3^4 = 81$

Initial population

Random initialization:

P	$f(i)$	$p_s(i)$	sel.
1: 0210	180	0.22	1×
2: 2002	120	0.14	0×
3: 1121	310	0.36	2×
4: 2110	240	0.28	1×
	$\Sigma = 850$		

1. Generation

Q_s	P	$f(i)$	$p_s(i)$	sel.
4: 21|10	1: 2121	230	0.21	1×
3: 11|21	2: 1110	250	0.23	1×
3: 1|121	3: 1210	160	0.15	0×
1: 0|210	4: 0121	440	0.41	2×
		$\Sigma = 1080$		

2. Generation

Q_s	P	$f(i)$	$p_s(i)$	sel.
4: 012|1	1: 0120	$\underline{520}$	0.37	2×
2: 111|0	2: 1111	170	0.12	0×
1: 21|21	3: 21$\underline{0}$1	260	0.18	1×
4: 01|21	4: 0121	440	0.32	1×
		$\Sigma = 1390$		

Mutation at individual 3

Best strategy (1: 0120):

small offering,
medium prices,
high quality,
self-service

Schema Theorem

The Schema Theorem – Trying to explain the success of GAs

l:	Number of genes in an individual
A:	Domain of the genes
k:	Cardinality of A
	(e.g. $A = \{0, 1\}$, $k = 2$)
$P(t)$:	Population at time t
n:	Size of the population

Size of the search space $= k^l$

Schemata

Schema H:

Mask which describes a subset of individuals with the same alleles at specific positions

Representation: A is extended to A_+ by adding the wildcard symbol "*".

Example:

$A_+ = \{0, 1, *\}$

*H=*000 represents 0000, 1000*
*H=*11* represents 0110, 0111, 1110, 1111*

Number of different schemata: $(k+1)^l$

Every individual contains 2^l schemata.

$P(t)$ contains at least 2^l at most $n \cdot 2^l$ schemata (depending on the diversity of the population).

Order of a schema $o(H)$:

Number of non-wildcard alleles

Defining length of a schema $\delta(H)$:

Distance between the left-most and right-most non-wildcard allele

Example:

H=*011*1***, $o(H)$=*4*, $\delta(H)$=*4*
H=***0*****, $o(H)$=*1*, $\delta(H)$=*0*
H=***0***1*, $o(H)$=*2*, $\delta(H)$=*4*

Fitness of a schema $f(H, t)$:

Average fitness of all individuals in $P(t)$ representing H

A schema H represents a *subspace of order* $l - o(H)$.

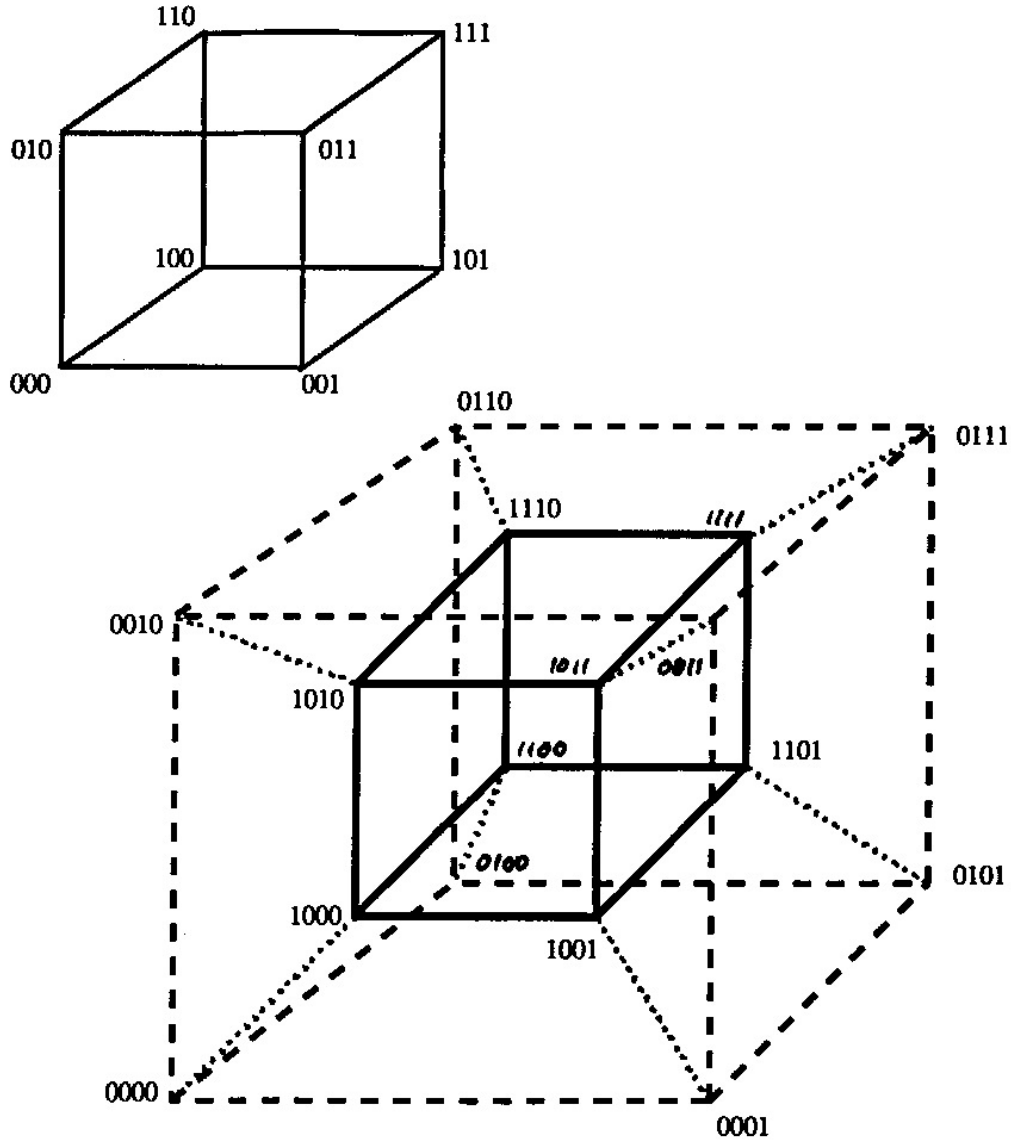
Example:

3-dim.:

$o(H) = 1$ Plane
$o(H) = 2$ Edge
$o(H) = 3$ Corner

4-dim.:

$o(H) = 1$ Cube
$o(H) = 2$ Plane



3D und 4D Hypercubes

Selection

Frequency of schema H in population $P(t)$: $m(H, P(t))$

$$m(H, Q_s(t+1)) =$$
$$m(H, P(t)) \cdot n \frac{f(H,t)}{\sum_{j=1}^{n} f(j)} =$$
$$m(H, P(t)) \frac{f(H,t)}{\overline{f}} \quad \text{where} \quad \overline{f} = \frac{\sum_{j=1}^{n} f(j)}{n}$$

The effect of selection is that schemata with above-average fitness proliferate while schemata with below-average fitness grow sparse.

Rate of increase for schemata:

Let H be a schema with above-average fitness $f(H,t) = \overline{f} + c\overline{f}$:

$$\begin{aligned} m(H, Q_s(t+1)) &= m(H, P(t)) \frac{\overline{f} + c\overline{f}}{\overline{f}} \\ &= m(H, P(t)) \cdot (1 + c) \end{aligned}$$

Considering all previous generations *with assumed constant* $\overline{f}$ results in a geometric series:

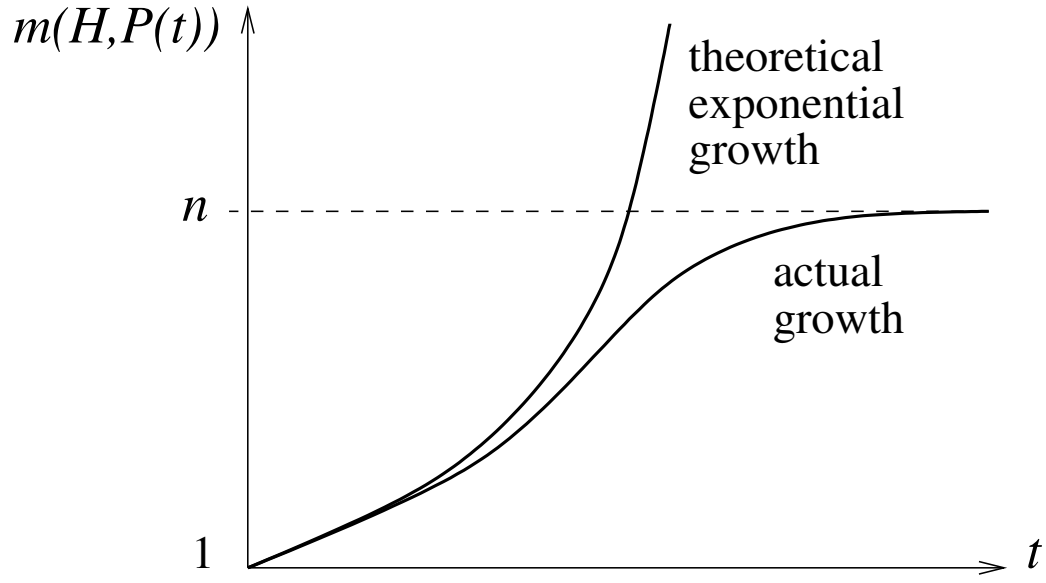
$$m(H, Q_s(t)) = m(H, P(0)) \cdot (1 + c)^t$$

It follows that selection results in an *exponentially* rising/falling number of above-/below-average schemata.

Beware:

In this discussion we disregarded the fact that the size of the population is limited and that $\overline{f}$ grows with rising number of iterations!

These observations are only valid approximations for the immediately following generations.



Theoretical and actual rise in schema frequency

Recombination

Crossover is used to introduce new schemata, but it also destroys existing ones.

Probability p_{surv} of a schema surviving a crossover:

H=0*****1,	*destroyed in most cases*
H=**01***,	*survives in most cases*
H=**0****,	*survives always*

The defining length $\delta(H)$ is relevant.

Probability, that the crossover point lies in the range of the defining length and the schema gets destroyed with high probability:

$$p_{dest} = \frac{\delta(H)}{l-1}$$

Lower bound for p_{surv} with crossover probability p_c:

$$p_{surv} \geq 1 - p_c \cdot \frac{\delta(H)}{l-1}$$

Combining selection and crossover:

$$m(H, Q_r(t+1)) \geq$$
$$m(H, P(t)) \cdot \frac{f(H,t)}{\overline{f}} \left(1 - p_c \cdot \frac{\delta(H)}{l-1}\right)$$

The amount of schemata with above average fitness and short defining length will grow exponentially.

Mutation

A schema survives mutation, when none of the fixed genes are mutated:

$$p_{surv} = (1 - p_m)^{o(H)}$$

For very small p_m holds:

$$p_{surv} \approx 1 - o(H) \cdot p_m$$

It follows that:

$$\begin{aligned} & m(H, P(t+1)) \geq \\ & \qquad m(H, P(t)) \cdot \frac{f(H,t)}{\overline{f}} \left(1 - p_c \cdot \frac{\delta(H)}{l-1}\right) (1 - o(H)p_m) \\ \approx \quad & m(H, P(t)) \cdot \frac{f(H,t)}{\overline{f}} \left(1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H)p_m\right) \end{aligned}$$

Schema-Theorem

Adding mutation: Lower order $\rightarrow$ higher survival probability

Schema-Theorem

The amount of schemata in the population with high fitness, short defining length and low order grows exponentially.

Building-Block-Hypothesis

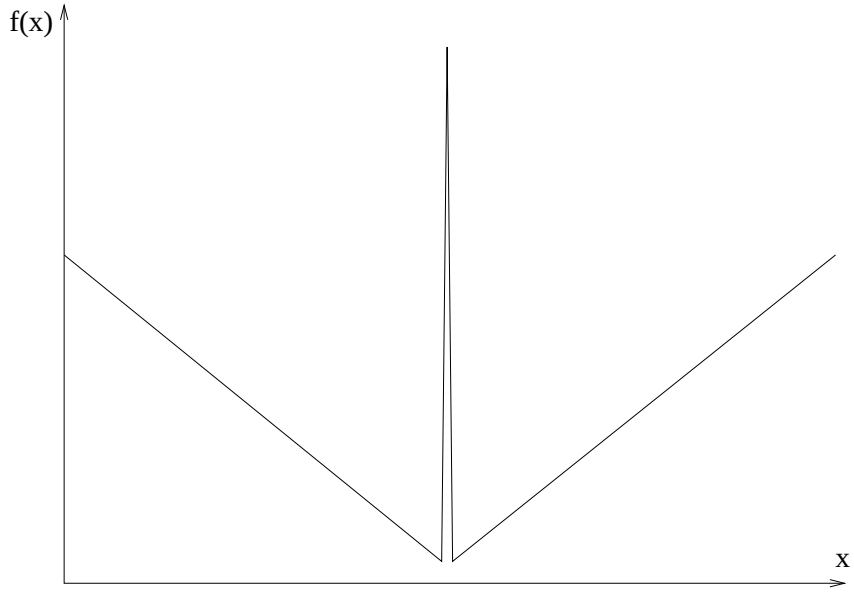
Schemata of **high fitness**, **short defining length** and **low order** are called **building blocks**.
By assembling building blocks, GAs execute an efficient, in some sense optimal, search for the optimal solution.

Corollaries:

- Encoding has to be adapted to the problem
- Sequence of genes plays a crucial role, strongly dependent genes should be placed next to each other

Deceptive Problems

Functions which deceive a GA have an *isolated optimum*.
Search for a needle in a haystack.



Function with isolated optimum

Building blocks set the GA on the wrong track.

Minimal Deceptive Problem:

Schemata with $o(H) = 2$:

Schema	Fitness
H_{00}=***0*****0*	f_{00}
H_{01}=***0*****1*	f_{01}
H_{10}=***1*****0*	f_{10}
H_{11}=***1*****1*	f_{11}

Assume:

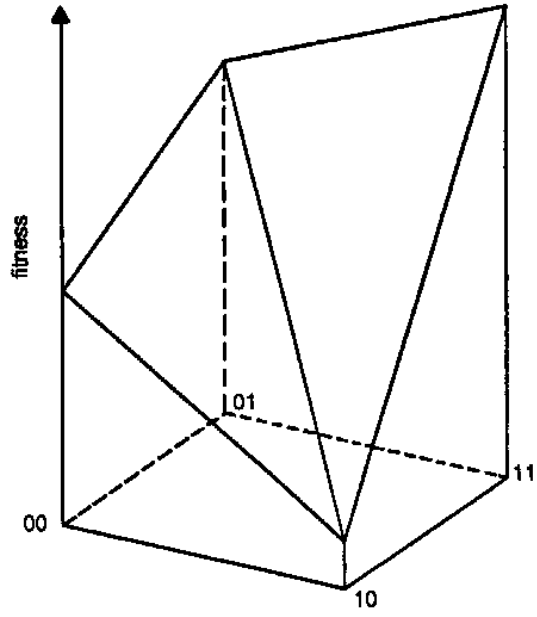
$$f_{11} > f_{00}, \quad f_{11} > f_{01}, \quad f_{11} > f_{10}$$

Schemata with $o(H) = 1$:

$$f_{0*} = \frac{f_{00}+f_{01}}{2} \qquad f_{*0} = \frac{f_{00}+f_{10}}{2}$$
$$f_{1*} = \frac{f_{10}+f_{11}}{2} \qquad f_{*1} = \frac{f_{01}+f_{11}}{2}$$

The problem is deceptive if

$$f_{0*} > f_{1*} \quad \text{or} \quad f_{*0} > f_{*1}$$



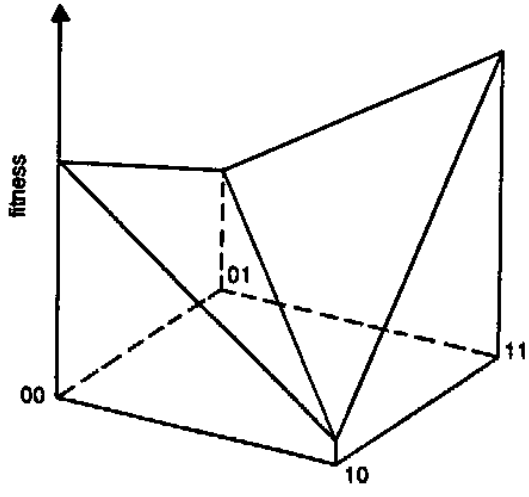
Type I

We have a closer look at the first case:

$$\frac{f_{00} + f_{01}}{2} > \frac{f_{10} + f_{11}}{2}$$

Two cases:

Type I:	$f_{01} > f_{00}$	not very critical
Type II:	$f_{00} \geq f_{01}$	very critical



Type II

Necessary condition that an EA can work efficiently:

Locality

Small changes in the genotype have to correspond (at least with high probability) to small changes in phenotype and fitness, and vice versa!

Otherwise an EA is not more efficient than random search!

Scaling

Scaling

There is an important distinction between:

Evaluation function $g(i)$ **(raw fitness)**:

Function to evaluate an individual (with a domain suitable for the problem)

Fitness function $f(i) = scale(g(i))$:

Calculates values which are suitable for selection and is derived from $g(i)$ by means of **scaling**

Properties:

- $f(i)$ is calculable for all individuals i
- $f(i) \geq 0$
- $f(i) > f(j)$ means:
  Individual i is better than individual j

Minimization Problems

$$f(i) = \begin{cases} C_{max} - g(i) & \text{if} \quad g(i) < C_{max} \\ 0 & \text{otherwise} \end{cases}$$

$g(i)$ **can be negative:**

$$f(i) = \begin{cases} g(i) + C_{min} & \text{if} \quad g(i) + C_{min} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Selection pressure

How much are better individuals preferred?
One possibility of quantifying the selection pressure is using the ratio

$$S = \frac{f_{\max}}{f_{\text{av}}}$$

where $f_{\max}$ is the fitness of the best solution and f_{av} the average fitness of the population.

Selection pressure too high:

Good individuals are preferred too much:

- they multiply rapidly ($\rightarrow$ *superindividual*)
- diversity contained in the population is reduced
- premature convergence of the GA towards a *local optimum*

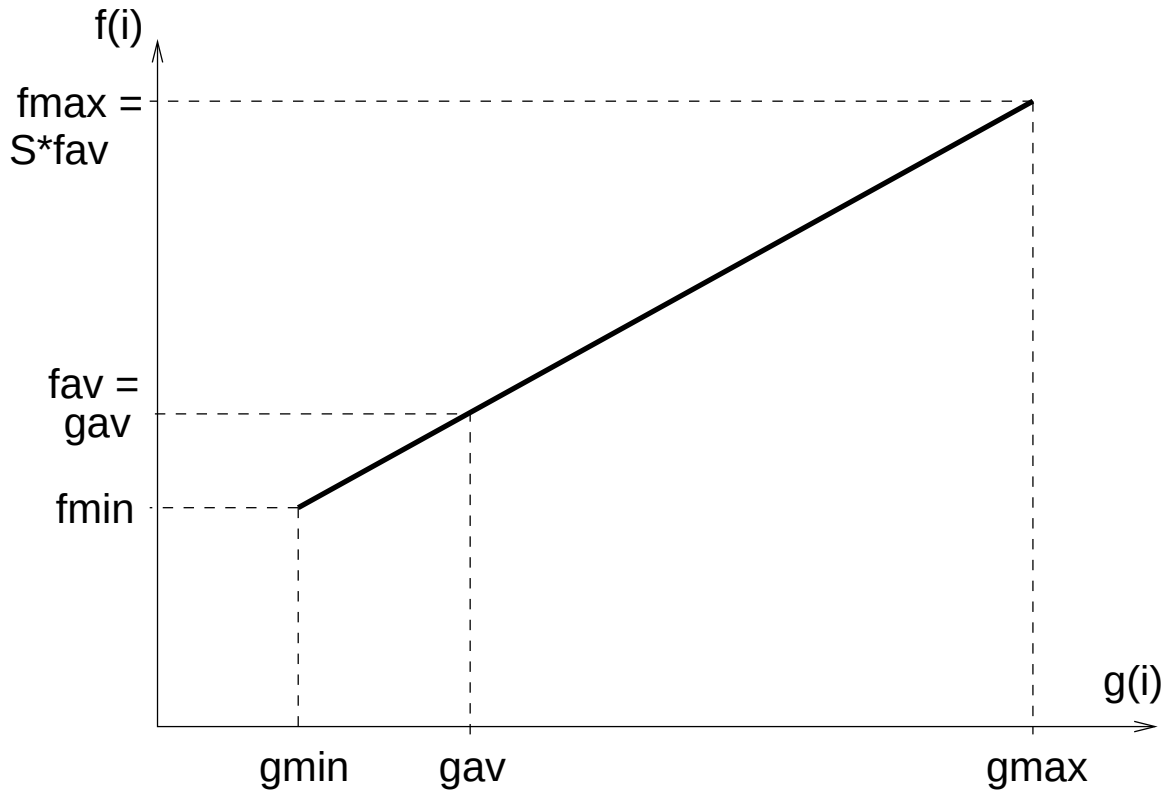
Selection pressure too low:

Good individuals are barely preferred:

- they spread very slowly in the population (if at all)
- bad individuals remain in the population
- algorithm degenerates to random search
- GA converges very slowly or doesn't converge

Linear Scaling

Linear scaling of the evaluation function: $f(i) = a \cdot g(i) - b$



Linear Scaling

It has to hold that $\overline{f}=\overline{g}$

We want the best individual to be selected S times on average for the next generation:

$$f_{max} = S \cdot \overline{f}$$

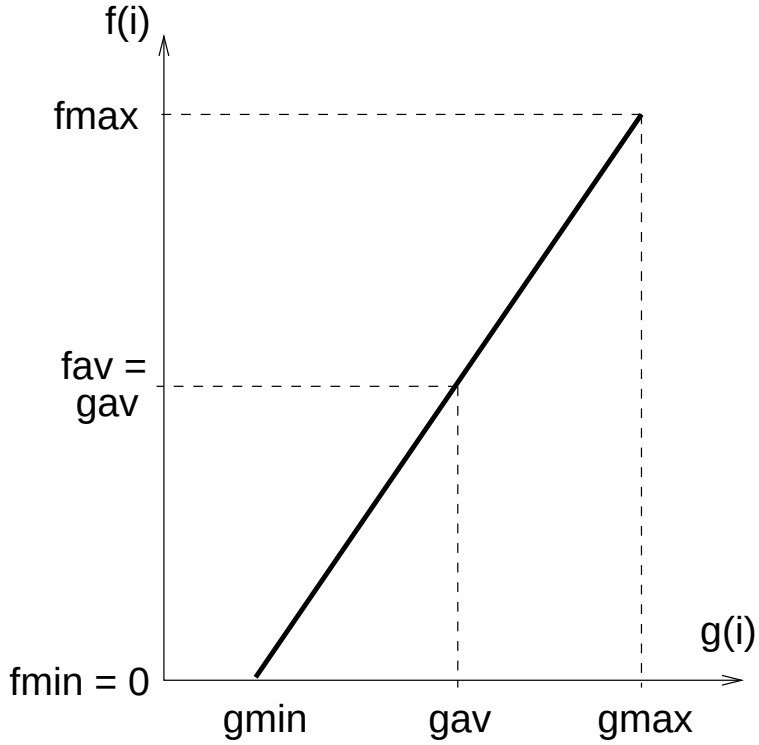
(appropriate values for $n = 50$ to 100: $S = 1.2$ to 2)

It follows:

$$a = \frac{S\overline{f} - \overline{f}}{g_{max} - \overline{f}} \qquad b = a \cdot \overline{f} - \overline{f}$$

If the worst evaluation g_{min} would result in a negative f_{min}, a and b are chosen so that $f_{min} = 0$ instead of $f_{max} = S\overline{f}$:

$$a = \frac{\overline{f}}{\overline{f} - g_{min}} \qquad b = a \cdot \overline{f} - \overline{f}$$



Linear Scaling: Avoiding negative values

Additional Selection Methods

Rank Selection

- Individuals are sorted according to their fitness
- For selection only the position according to this sequence (= *rank*) is used

Relative differences in quality have no influence

$\rightarrow$ useful when the distribution of fitness values is problematic (e.g. superindividuals)

There are a lot of variants:

Linear Ranking

Rank i:
0... best, $n-1$... worst individual

Given: $\alpha = p_s(0)\, n$ and $\beta = p_s(n-1)\, n$

Every individual has a probability of being selected proportional to its rank:

$$p_s(i) = \frac{\alpha + i\frac{\beta-\alpha}{n-1}}{n}$$

The actual selection is done by fitness proportional selection!

Sum of probabilities:

$$\begin{aligned}\sum_{i=0}^{n-1} p_s(i) &= \alpha + \frac{\beta-\alpha}{n(n-1)} \sum_{i=0}^{n-1} i \\ &= \frac{1}{2}(\alpha+\beta)\end{aligned}$$

Since $\sum_{i=0}^{n-1} p_s(i) = 1$ has to hold:

$$\rightarrow \quad \alpha = 2 - \beta$$

For instance: $\alpha = 2, \quad \beta = 0$

Tournament Selection

To select an individual, k random (uniformly distributed) individuals are chosen and the best one of those is selected.

k controls the selection pressure.

Advantages:

- Applicable in situations where a single individual cannot (or only with much difficulty) be evaluated independently of other individuals.
  Example: A winning strategy for a game has to be developed.
- An efficient implementation is possible, which is interesting for very large populations.

Selection probabilities:
$k = 1$ (uniformly distributed random selection):

$$p_s(i) = 1/n$$

$k = 2$:

$$p_s(i) = \left(1 - \left(1 - \frac{1}{n}\right)^2\right) \cdot \left(1 - \frac{i}{n}\right)$$

any k:

$$p_s(i) = \left(1 - \left(1 - \frac{1}{n}\right)^k\right) \cdot \left(1 - \frac{i}{n}\right)^{k-1}$$

The expected number of offspring for an individual is asymptotically equal to linear ranking for $k = 2$.

Weighted Tournament Selection

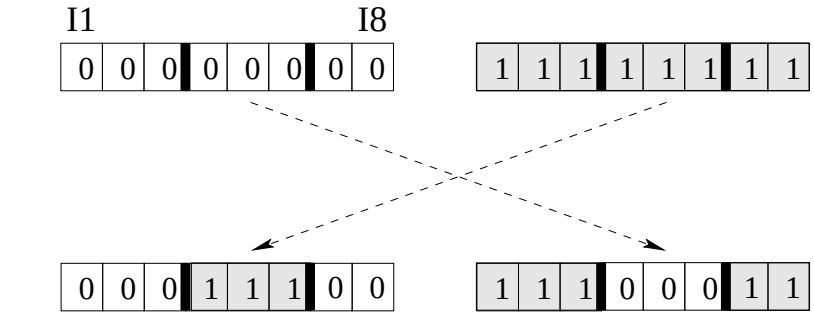
To have a more fine-grained control over the selection pressure:

Instead of always choosing the best individual among k, it is only selected with probability p_1, the second best with probability $p_2 < p_1$.

It hast to hold that: $\sum_{i=1}^{k} p_i = 1$

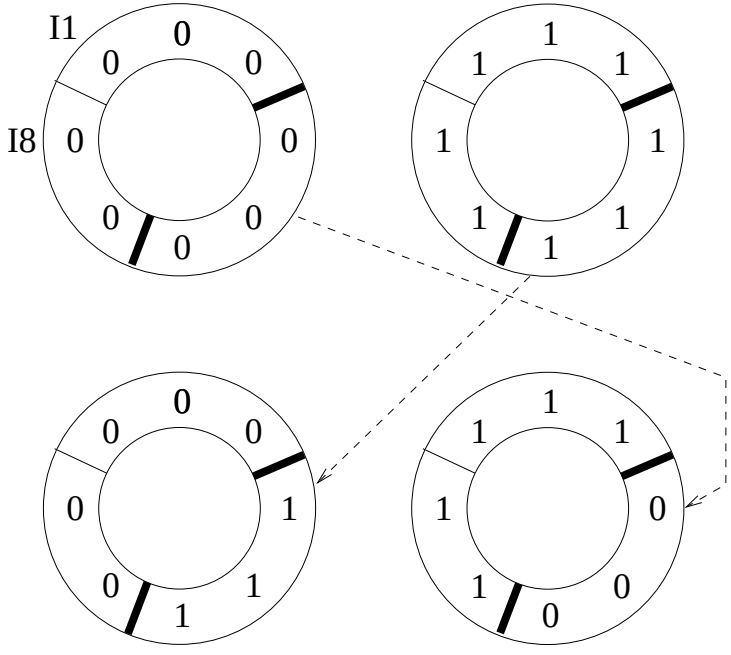
Additional Crossover Methods

2-point Crossover



The first and last gene are treated as if they were neighbors.

$\rightarrow$ the string can be interpreted as ring



2-point Crossover

Multipoint Crossover

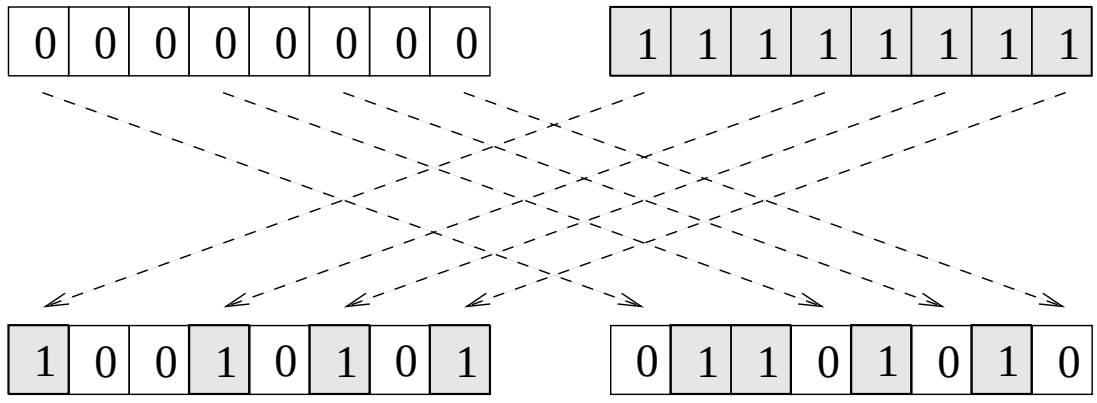
Generalization of the 1/2-point crossover

The string can be interpreted as ring if the number of crossover points is even.

In general, there are $\binom{l}{cp}$ possibilities with cp crossover points.

Generally, more crossover points have the effect that fewer schemata survive and that they are more mixed up.

Uniform Crossover



Uniform Crossover

For every gene there is a separate decision which of its parents alleles it receives.

The defining length and the sequence of genes is irrelevant for the survival probability of schemata. The only determining factor is the order:

$$p_{surv} \geq (1/2)^{o(H)}$$

Further Approaches to Increase Efficiency

Elitism:
Independently of selection, the k best individuals of the parent generation are inserted into the new population unchanged.

Properties:

- Fitness of the best individual is monotonically increasing.
- Bigger k means faster convergence.
- Method is susceptible to super individuals and gets trapped in local optima more easily.
- Concentration on local search (instead of global seach)

$\rightarrow$ Well suited for unimodal functions.

Further Approaches to Increase Efficiency

Overlapping Populations:

- Per generation only $G \cdot n$ individuals ($G \in (0, 1]$, *generation gap*) are newly created.
- $n - G \cdot n$ individuals are copied (uniform random selection) from the parent population.

Advantages:

- Beneficial for "learning while performing"
- For large populations: Memory savings

Extreme Case:
Create only *one* individual and replace an old one with it
$\rightarrow$ *steady state GA* (*non-generational GA*)
(following the principles of nature more closely)

Further Approaches to Increase Efficiency

Crowding Model:
Population is split into *CF* (*crowding factor*) groups.

Space for a new individual is created in the following way:

- A group is selected uniformly at random
- The most *similar* individual in the group (compared to the new individual) is deleted.

To measure similarity, use e.g. hamming-distance

Advantages:

- The diversity in a group is preserved
- Fights super individuals
- Very suited for complicated, multimodal functions.

Constraints

Constraints

Very often some **constraints** have to be fulfilled in order for a solution to be valid.

Example: Multidimensional Knapsack Problem (MKP)

Given: n objects with values $v_1, \ldots, v_n$,
m resources with capacities $R_1, \ldots, R_m$,
for every object $i = 1, \ldots, n$ and every resource $j = 1, \ldots, m$: resource requirements $r_{i,j}$

Find: Incidence vector $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$

$$\text{maximize} \quad \sum_{i=1}^{n} v_i \, x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} r_{i,j} \, x_i \leq R_j \quad \forall j = 1, \ldots, m$$

Constraint Handling

Often the best approach:

Choose encoding, initialisation and operators for a problem so that all possible individuals represent *valid* solutions (i.e. all constraints are always satisfied).

This is not always possible.

$\rightarrow$ Invalid candidate solutions get assigned a bad fitness value to stop them from propagating.

Penalization

Easy approach: Invalid solutions have a fitness of 0.

If there are a lot more invalid individuals than valid individuals in the population: GA will probably not find a good solution!

$\Rightarrow$ There should be some distinction based on the degree of constraint violation.

The common approach is to add a *penalty* term to the fitness function:

$$f'(i) = f(i) - \sum_{j=1}^{m} \gamma_j \cdot \Psi_j(i)$$

γ_j . . . penalty coefficients, $\Psi_j(i)$ . . . penalty functions

Penalization

Choice of γ_j (and $\Psi_j(i)$) is often not straight forward!

Optimal values for γ_j can depend on the current population!

Stepwise Adaption of Weights (SAW):

- Initialize all γ_j with small starting values.
- After a specific number of generations t_{update}:
  Increase γ_j for all constraints j which are not satisfied in the currently best solution: $\gamma_j \leftarrow \gamma_j + \Delta r$

Repairing

A problem specific **"repair"–algorithm** modifies invalid individuals, so that they become valid.

Those changes can be applied:

- only temporarily during evaluation (*Baldwin effect*)
  $\rightarrow$ successors don't profit (directly).
- by replacing the genotype (*Lamarckian inheritance*)
  $\rightarrow$ changes the distribution of schemata.

From experiments:
Change genotype in $\approx 5\%$ of cases.

Permutation Encodings

A lot of combinatorial problems can be represented by **permutations**:
$\Pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$

A problem specific **decoding heuristic** transforms the genotype into the phenotype.

e.g. MKP:

procedure First-Fit (Π)
begin
  $x = (0, \ldots, 0)$;
  for $i = 1, \ldots, n$ **do**
    $x_{\Pi[i]} = 1$;
    if $\exists j \mid \sum_{i=1}^{n} r_{i,j}\, x_i > R_j$ **then**
      $x_{\Pi[i]} = 0$;
end

Example: Traveling Salesman Problem (TSP)

Find a tour that visits each of n cities exactly once.

For this problem, a permutation naturally corresponds to the sequence in which the cities are visited.

Problem:
Classic mutation and recombination operators do not create valid permutations.

$\rightarrow$ New operators are needed.

Mutation Operators for Permutations

Reciprocal Exchange:
Two alleles are exchanged.

Insertion:
An allele is selected and inserted at another location, the range between old and new location is moved.

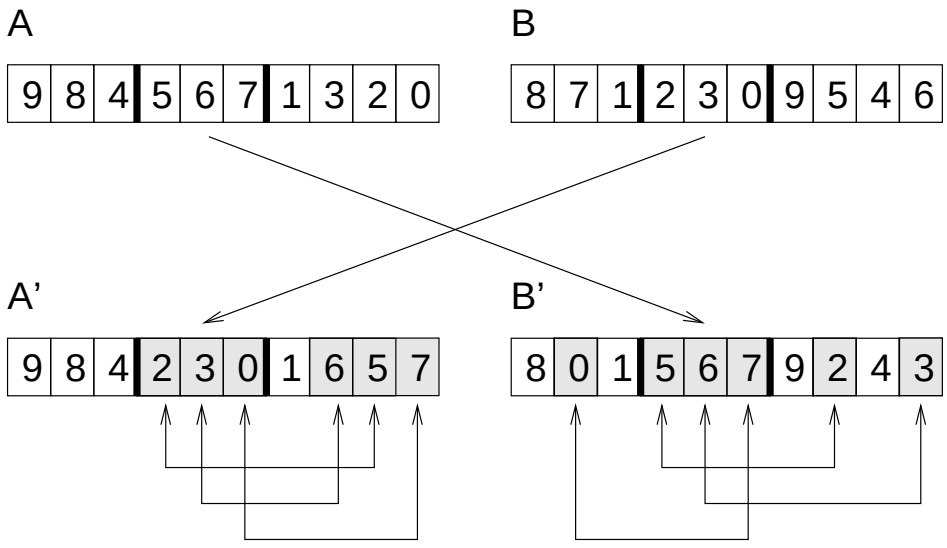
Displacement:
A substring is selected and inserted at another location.

Inversion:
A random range is mirrored.

Recombination Operators for Permutations

Partially Matched Crossover – PMX

1. Chose crossover range by randomly selecting two gene positions
2. $A' \leftarrow A, \quad B' \leftarrow B$
3. For all genes i in the crossover range:
   - 3.1 Find location of B_i in A' $(= A'_j)$
   - 3.2 Swap A'_j and A'_i
   - 3.3 Find location of A_i in B' $(= B'_k)$
   - 3.4 Swap B'_k and B'_i



Partially Matched Crossover

Properties:

- Crossover range is swapped exactly.
- Remaining genes and their positions are preserved "as good as possible".
- Absolute positions and sequence of genes are partially passed down.
- For TSP: moderate performance

Order Crossover – OX

1. Randomly determine a crossover range:

```
A  = 9 8 4|5 6 7|1 3 2 0
B  = 8 7 1|2 3 0|9 5 4 6
```

2. Delete genes in crossover range of A in B and vice versa:

```
A' = 9 8 4|5 6 7|1 _ _ _
B' = 8 _ 1|2 3 0|9 _ 4 _
```

3. Rearrange genes:
   0.1 Genes from inside the crossover range
   0.2 Free space
   0.3 All remaining genes (beginning at the end of the crossover range)

```
A' = 5 6 7 _ _ _ 1 9 8 4
B' = 2 3 0 _ _ _ 9 4 8 1
```

Order Crossover – OX

4. Fill the free space with the genes from the crossover range of the other parent:

```
A' = 5 6 7 2 3 0 1 9 8 4
B' = 2 3 0 5 6 7 9 4 8 1
```

Properties:

- Relative order of the genes is passed on to a wide extent
- Absolute gene positions are not preserved
- For TSP: good test results

Cycle Crossover – CX

Basic condition:

Every gene is copied from its parent while *preserving its position*.

Initial situation:

```
A  = 9 8 2 1 7 4 5 0 6 3
B  = 1 2 3 4 5 6 7 8 9 0
```

1. For A' copy first gene of A:

   A' = 9 _ _ _ _ _ _ _ _ _

2. To fulfill the basic condition, a whole *cycle* has to be copied from A:

   A' = 9 _ _ 1 _ 4 _ _ 6 _

3. All free locations are filled by genes from B:

   A' = 9 2 3 1 5 4 7 8 6 0

4. The same is done for B':

   B' = 1 8 2 4 7 6 5 0 9 3

Properties:

- Absolute positions of the genes are preserved
- Order of genes is not preserved
- For TSP: suboptimal results

Uniform Order Based Crossover – UOBX

1. Random decision at every position, whether the gene is copied from the first parent or remains free.
2. The free positions are filled with the missing genes, their order is determined by the second parent.

Properties:

- Order and absolute position are partially preserved
- For TSP: suboptimal results

Edge Recombination Crossover – ERX

TSP: Connections between cities (= *edges*) essential!

- Child is created in a way that preserves the edges
- Direction of the edges is irrelevant

Parents:

```
A = 1 2 3 4 5 6 7 8 9
B = 4 1 2 8 7 6 9 3 5
```

1. Use parents to create edge lists for all cities:

```
1:  9, 2, 4
2:  1, 3, 8
3:  2, 4, 9, 5
4:  3, 5, 1
5:  4, 6, 3
6:  5, 7, 9
7:  6, 8
8:  7, 9, 2
9:  8, 1, 6, 3
```

2. Choose starting city c (e.g. 1. city of 1. parent, here $c = 1$); remove c from edge list

3. Select from edge list of c the city with the shortest edge list
   - If there are multiple possibilities with arc lists of equal length: Random selection among them
   - No possible cities (*edge failure*): Randomly choose a city that has not been visited yet (rarely necessary)

4. Chosen city now is current city c; remove it from all edge lists

5. While tour is incomplete, go to 3.

Possible solution: `A' = 1 4 5 6 7 8 2 3 9`

Improvement:

- Edges present in both parents are marked and preferred:

```
1:  9, #2, 4
2:  #1, 3, 8
3:  2, 4, 9, 5
4:  3, #5, 1
5:  #4, 6, 3
6:  5, #7, 9
7:  #6, #8
8:  #7, 9, 2
9:  8, 1, 6, 3
```

Possible solution: `A' = 1 2 8 7 6 9 3 4 5`
Especially for the "blind" TSP one of the best techniques!

Hybrid Approach with Heuristic

(by Grefenstette)

1. Choose starting city c randomly
2. From 2 parents select 4 edges adjacent to c;
   disregard edges to already visited cities
3. Choose one of the selected edges:
   - Probability distribution *inversely proportional to costs*
   - No edges available:
     Random selection among all unvisited cities
4. Chosen edge determines new current city c
5. While tour is incomplete, go to 2.

$\rightarrow$ Even better results by using this heuristic.

Hybrid Genetic Algorithms

Hybrid Genetic Algorithms

If problem specific knowledge is available, it is often useful to combine a GA/EA with a local optimization method, to find better solutions faster.

GA:

global view, finds "hills"

local optimization:

hill climbing finds "top of hills"

1. Local optimization at the beginning

- Create initial population randomly
- Locally improve some or all solutions by hill climbing

2. Local improvements after EA ("polishing")

- Find a good solution with EA
- Improve it (or the k best) locally

3. Local improvement before fitness evaluation

- Perform some local improvement steps before fitness evaluation

If local search is very time-consuming: only execute for k best individuals or only execute a predetermined number of steps.

Hybrid GAs following this approach are also called **Memetic Algorithms**.

Memetic Algorithm

- Introduced by Pablo Moscato 1989,
  also called: hybrid EAs, Lamarck EAs
- Mainly combination of EAs and other methods
- gene $\rightarrow$ meme, individual $\rightarrow$ agent
- individual improvements and population based cooperation
- MA not necessarily EA working with locally *optimal* solutions, rather with locally *optimized* solutions

$\Rightarrow$ Very successful when applied to real world problems!

4. Intelligent recombination / mutation

See heuristic edge-crossover for TSP.

5. Intelligent initialization

The solutions contained in the initial population of the EA are created by a heuristic.

IMPORTANT: Enough diversity has to be guaranteed!

6. Intelligent decoder

The phenotype is generated from the genotype with an "intelligent decoder", see for example the decoder for MKP.

Further example of "intelligent decoding": Weight-Coding

(interesting for complex combinatorial problems, with constraints)

- Take a heuristic for the problem at hand.
- The chromosome is a vector of (continuous) parameters.
- For phenotype construction and evaluation, the values of the chromosome are used to temporarily change the current problem (**Biasing**).
- The heuristic is used to solve the modified problem.
- The generated solution is the phenotype and used for evaluation with respect to the original problem.

Ant Colony Optimization

Ant Colony Optimization

... umbrella term for optimization algorithms inspired by the search for food of ant colonies

Emergence

... a system presents phenomena on a global level, which could not have been predicted by just understanding the components of the system on a lower level.

Feedback often plays a crucial role.

Figure : ripple marks in sand develop out of (in the beginning) minuscule bumps interacting with air turbulences created by them (positive feedback loop)

Swarm Intelligence

- emergent phenomenon
- decentralized organization
- collective behaviour / collective intelligence
- super organism

Examples: fish, birds, termites, ants, human brain

$\Rightarrow$ Examples of optimization algorithms inspired by swarm intelligence:

- Particle Swarm Optimization (PSO)
- Artificial Immune Systems (AIS)
- Ant Colony Optimization (ACO)
- ...

Self-organization

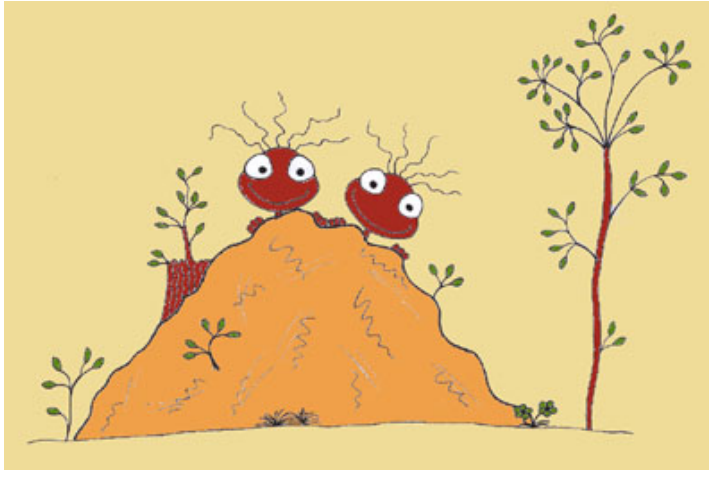
> *"Self-organization is a process in which a pattern at the global level of a system emerges solely from numerous interactions among lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern."* Camazine et al., *Self-Organization in Biological Systems, p.8*

Mechanisms:

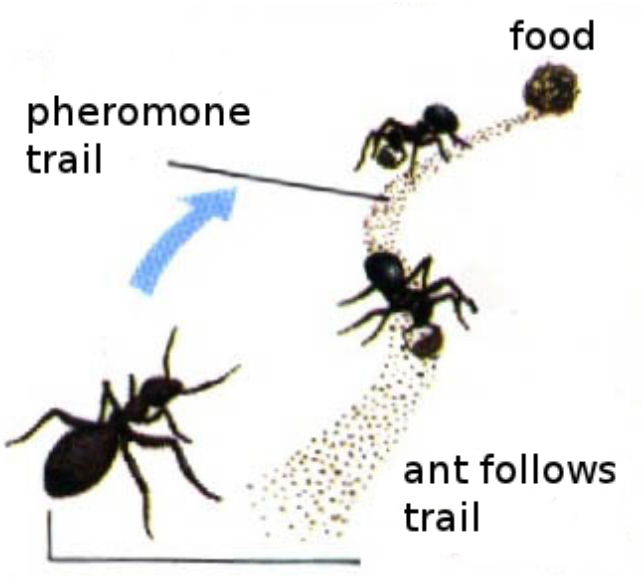
- Feedback
- **Stigmergy:** Communication in a decentralized system by modifying the surroundings
- decentralized control
- heterarchy
- random fluctuations ($\rightarrow$ makes new structures possible)

Ant Colony

- thousands of ants (biggest known colony: billions of ants!)
- consisting mainly of: queen(s), workers, drones
- higher developed species use far more differentiated schemes for division of labour
- solves complex optimization problems (e.g. shortest path to a source of food)



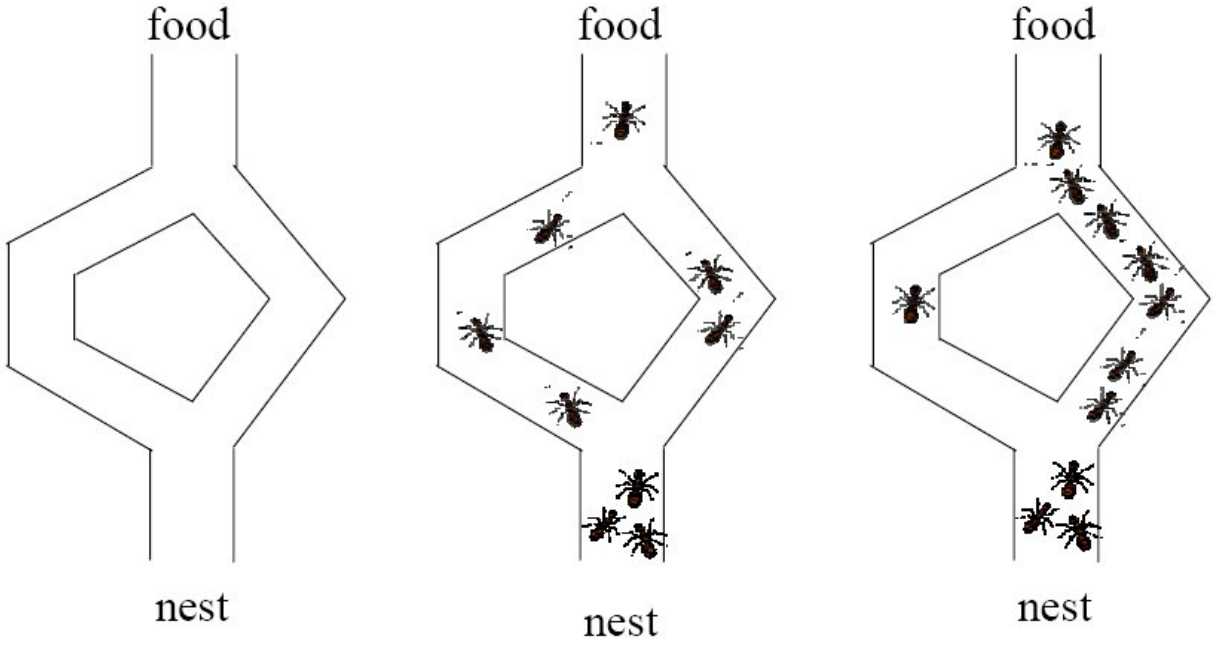
Pheromones



- Pheromones: secreted chemical factor
- differ from species to species
- can mark paths, for instance towards food
- higher intensity $\Rightarrow$ higher probability of choosing the path
- deviations from the path are also possible $\Rightarrow$ finding new possible paths
- $\Rightarrow$ nearly every ant uses the optimal path without global information

Pheromones (2)

The pheromone trail on the optimal (shortest) path increases in strength faster than on the longer paths, because the ants on the shortest path can travel it more often in the same amount of time when transporting food to the ant hill.



ACO: Principles

- **ACO (Ant Colony Optimization)** = umbrella term for some (similar) ant-based algorithms
- developed by Marco Dorigo, 1991
- mainly used for combinatorial optimization problems, but there are also approaches for continuous parameter optimization
- principle: constructive metaheuristic, borrowing principles of an ant colony, i.e. individuals construct/modify a solution based on
  1. global information (stigmergy by means of pheromones) and
  2. local information (ant "knows" its immediate neighborhood).

Travelling Salesman Problem (TSP)

We will now discuss different ACO variants applied to the Travelling Salesman Problem (TSP)

Def. TSP:
Given: complete directed graph $G = (V, A)$.
Find: shortest tour using all cities, so that each city is visited exactly once.

Basic Idea:
Ants travel through the **construction graph** (G in this case) and leave pheromones. During execution of the algorithm, better and better tours emerge.

The Fundamental Algorithm

Repeat steps 1-3 until termination criterion is satisfied:

1. Ants **create a solution** by using local information and pheromones
2. **Pheromone update** calculates the new pheromone levels depending on previous pheromone levels, evaporation and newly added pheromones
3. **Daemon Actions (optional):** Operations which require global information and cannot be executed by ants, for instance improvement heuristics,...

Ant System (AS)

- **Discrete timesteps t:**
  In every timestep, all ants create a complete solution.
- **Ants: (artificial ants)** Index $k = 1, \ldots, m$
- **Pheromone-matrix $\tau_{ij}(t)$:** Intensity of the pheromones on edge from vertex i to vertex j in the construction graph.
- **Matrix for local information η_{ij}:**
  TSP: "Visibility" (=inverse distance) of the city ($\eta_{ij} = 1/d_{ij}$)

Ant System (AS)

Deciding on the next city in a vertex:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{otherwise} \end{cases}$$

- **Parameters α and β** control the ratio between pheromones and local information.
  $\beta = 0$: fast convergence to a (random) path
  $\alpha = 0$: classic greedy approach (nearest neighbor heuristic)
- **Neighborhood $\mathcal{N}_i^k$:** every ant has a list with the valid neighbors.
  TSP: List of unvisited cities

AS: Pheromone Update

Reinforce used paths

$$\tau_{ij}(t) = \tau_{ij}(t) + \triangle\tau_{ij}(t)$$

where

$$\triangle\tau_{ij}(t) = \sum_{k=1}^{m} \triangle\tau_{ij}^{k}(t)$$

with

$$\triangle\tau_{ij}^{k}(t) = \begin{cases} 1/\mathrm{length}(T^k(t)) & \text{if}(i,j) \in T^k(t) \\ 0 & \text{otherwise} \end{cases}$$

T^k ... constructed tour of ant k

AS: Pheromone Update

Evaporation

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t)$$

- initial small pheromone value: $\tau_0 \geq 0$
- ρ: pheromone evaporation rate $(0 < \rho \leq 1)$

AS: Algorithm

procedure local search

begin
  for $t \leftarrow 1, \ldots, t_{\max}$
    for each *ant* $k = 1, \ldots, m$
      Choose initial city
      for each *unvisited city i*
        Choose a city according to p_{ij}^k
    Apply pheromones to paths
    Evaporate pheromones
end

Note: Experiments show that letting each ant *start at a random city* leads to better results.

AS is not very efficient for bigger instances...

Further AS Variants

... there are a lot of ACO variants, for instance:

- Elitist AS (EAS)
- Rank-Based AS (AS_{rank})
- Ant-Q
- $\mathcal{MAX} - \mathcal{MIN}$ AS
- Ant Colony Systems (ACS)
- CACO
- CIAC
- Approximate Nondeterministic Tree Search (ANTS)
- API
- Hypercube-Framework

Elitist AS (EAS)

Dorigo 1992

Idea: Ant which has found the currently best solution (index *bs*, "best-so-far") applies more pheromones

$$\tau_{ij}(t) = \tau_{ij}(t) + \sum_{k=1}^{m} \triangle\tau_{ij}^{k}(t) + e \cdot \triangle\tau_{ij}^{\mathrm{bs}}$$

$$\triangle\tau_{ij}^{\mathrm{bs}} = \begin{cases} 1/\sum_{(k,l)\in \mathrm{bs}} d_{kl} & (i,j) \in \mathrm{T}^{\mathrm{bs}} \\ 0 & \text{otherwise} \end{cases}$$

- shorter tours are found
- less iterations are required

Rank-Based AS

Bullnheimer 1999

- Like EAS, but ants with higher rank r (according to their solution evaluation) apply more pheromones.
- Only the best $(w-1)$ ants and the ant with the currently best tour influence the pheromone update.

$$\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1} (w-r) \triangle \tau_{ij}^{r} + w \cdot \triangle \tau_{ij}^{\mathrm{bs}}$$

where $\triangle \tau_{ij}^{r} = 1/\text{length of tour of ant } r$
and $\triangle \tau_{ij}^{\mathrm{bs}} = 1/\text{length of best found tour.}$

$\Rightarrow$ Somewhat better results than EAS.

$\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)

Stützle, Hoos (1997, 2000)

- Pheromon-Update only by the "most successful" ant
- Limits for pheromone values: $[\tau_{\min}, \tau_{\max}]$
  Avoids premature stagnation
- Initialization: $\tau_{\max}$ (very exploratory in the beginning!)
- Reinitialization when stagnating (indicators: statistic or time since last improvement)
- proportional updates of paths: weaker reinforcement of strongly marked trails. (*Pheromone smoothing*)

$\Rightarrow$ $\mathcal{MM}$AS is one of the best researched and most successful ACO algorithms.

Ant Colony System (ACS)

Dorigo, Gambardella (1997)

- originally developed for higher dimensional problems
- parameter q_0 ($0 \leq q_0 \leq 1$) controls diversification/intensification
- Choice of next city j:

$$j = \begin{cases} argmax_{l \in \mathcal{N}_i^k} \left[\tau_{il} \cdot (\eta_{il})^\beta \right] & \text{if } q \leq q_0 \\ N & \text{if } q > q_0 \end{cases}$$

where q is a uniformly distributed random variable in $[0, 1]$, and $N \in \mathcal{N}_i^k$ a city selected from the neighborhood according to the probabilities

$$p_{ij}^k = \frac{\tau_{ij} \cdot \eta_{ij}^\beta}{\sum_{l \in \mathcal{N}_i^k} \tau_{il} \cdot (\eta_{il})^\beta}$$

ACS (2)

- $q > q_0$: Selection like AS $\rightarrow$ *Diversification*
- $q \leq q_0$: Utilizing already gained information $\rightarrow$ *Intensification*
- Candidate lists for heuristically selecting the next city $\Rightarrow$ improves both solution quality and runtime!
- Pheromone Update:
  1. local update (immediately after using the edge):

$$\tau_{ij}(t) = (1 - \xi) \cdot \tau_{ij}(t) + \xi \cdot \tau_0$$

     with parameter $\xi \approx 0.1$.

  2. global update:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \triangle\tau_{ij}^{\text{bs}} \quad \forall (i,j) \in T^{\text{bs}}$$

ACS (3)

- Local update reduces the pheromones on edge (i, j) immediately after it has been used.
- Increases utilization of unused edges $\Rightarrow$ avoids stagnation.
- Due to local updates, there is a pronounced difference if ants build their tours in *parallel* or *sequentially*. Parallel turns out to be advantageous.
- Implicit limits for pheromone values!

Remarks

The problem representation plays a crucial role

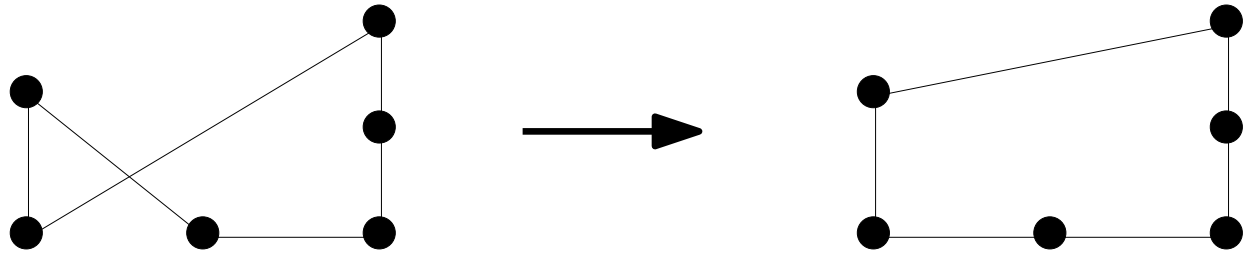
1. $\tau_{ij} \equiv$ Interest of visiting j after city i.
2. $\tau_{ij} \equiv$ Visit i as j-th city.

Note: The second variant is bad, because not all symmetries of the problem are utilized. There is also low locality.

Daemon Actions

Additionally, utilization of improvement heuristics is possible. Those usually take *global properties* of the problem instance into account.

- 2-exchange neighborhood



- k-exchange neighborhood (2.5-opt $\mathcal{O}(n^2)$ and 3-opt $\mathcal{O}(n^3)$ are successful for the TSP)
- When using local search (LS), the heuristic information η may not be necessary any more!
- Tour $c_1 \Longrightarrow^{\mathrm{LS}} c_2$; Pheromone update according to evaluation of c_2.
  1. Approach of Lamarck: Update of edges of c_2
  2. Approach of Darvin: Update of edges of c_1

Comparing different ACO variants

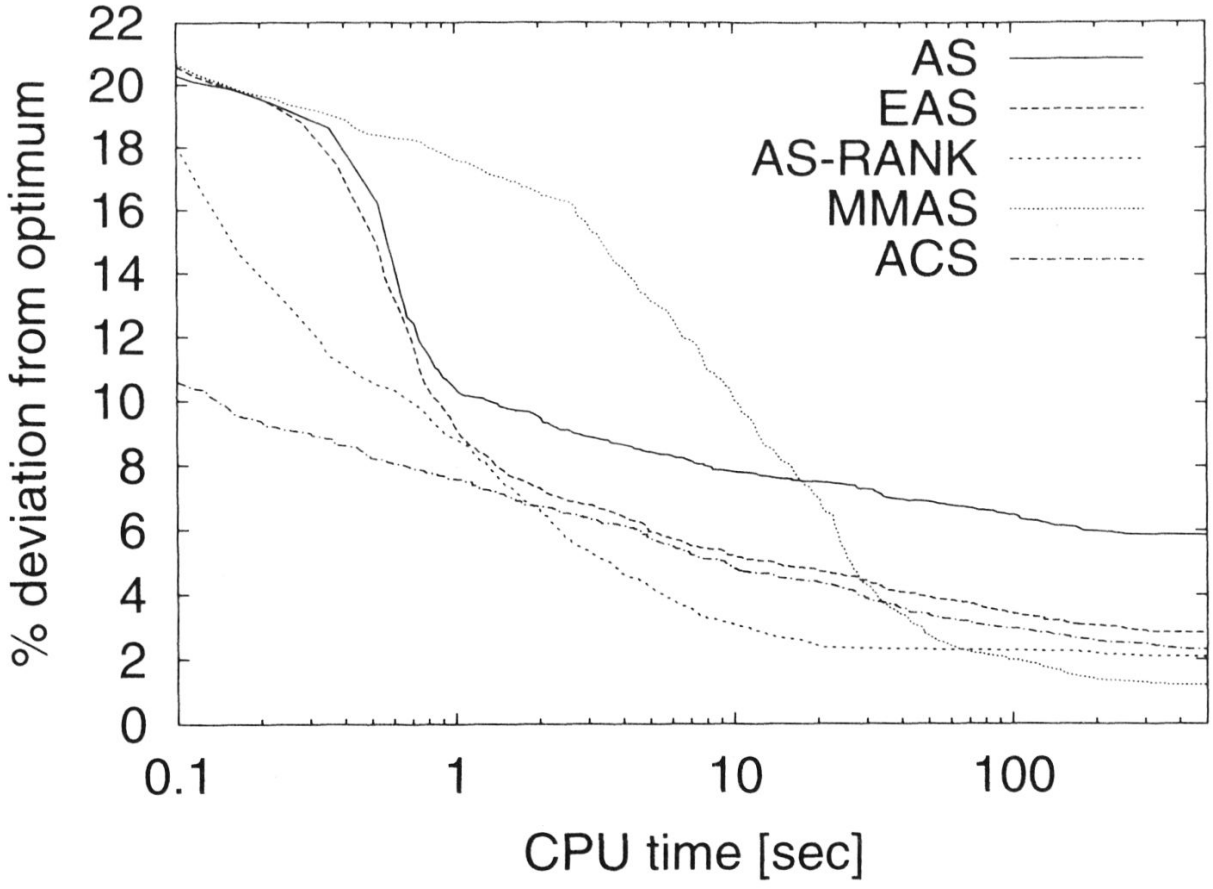


Figure : Time to Target plot for different ACO variants. Source: "Ant Colony Optimization" (Dorigo, 2003)