Discrete Optimization - Heuristics

Geir Hasle

SINTEF ICT, Applied Mathematics, Oslo, Norway

University of Jyväskylä, Finland

eVITA Winter School 2009

Geilo, January 11.-16. 2009

Summary (slide 1)

- Discrete optimization problems are important
- Discrete optimization problems are often computationally hard
- Exact methods may take too long, will give guarantees
- Better to find a good solution to the real problem than the optimal problem to an overly idealized problem
- Local Search is a robust, simple and fast method
- Local Search gives few and weak guarantees
- Local Search is local, gets trapped in a local optimum

Summary (slide 2)

- Metaheuristics move on from local optima and explore larger parts of the solution space
- Metaheuristics are often based on local search
- Different strategies, many variants
- There is no free lunch
- This area is a lot of fun, many challenges
- Short road from theoretical to practical improvements

Outline

- 2-slide talk (thanks, François!)
- Background and Motivation
- Definition of Discrete Optimization Problems (DOP)
- Basic concepts
- Local Search
- Metaheuristics
- GUT
- No free lunch
- Future directions
- Summary

Literature

- **H. H. Hoos, T. Stützle: *Stochastic Local Search - Foundations and Applications*, ISBN 1-55860-872-9. Elsevier 2005.**
- C.C. Ribeiro, P. Hansen (editors): *Essays and Surveys in Metaheuristics.* ISBN 0-4020-7263-5. Kluwer 2003
- **F. Glover, G.A. Kochenberger (editors): *Handbook of Metaheuristics,* ISBN 0-7923-7520-3. Kluwer 2002.**
- S. Voss, D. Woodruff (eds): *Optimization Software Class libraries*. ISBN 1-4020-7002-0. Kluwer 2002.
- Z. Michalewicz, D. B. Fogel: *How to Solve It: Modern Heuristics*. ISBN 3540660615. Springer-Verlag 2000.
- S. Voss, S. Martello, I.H: Osman, C. Roucairol (editors): *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer 1999.
- **D. Corne, M. Dorigo, F. Glover (editors): *New Ideas in Optimization*. ISBN 007 709506 5. McGraw-Hill 1999.**
- L. A. Wolsey: *Integer Programming*. ISBN 0-471-28366-5. Wiley 1998.
- I.H. Osman, J.P. Kelly (editors): *Meta-Heuristics: Theory & Applications*. Kluwer 1996.
- **E. Aarts, J.K. Lenstra: *Local Search in Combinatorial Optimization*. ISBN 0-471-94822-5. Wiley 1997.**
- C. R. Reeves (editor): *Modern Heuristic Techniques for Combinatorial Problems*. ISBN 0-470-22079-1. Blackwell 1993.
- M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*. ISBN-0-7167-1045-5. Freeman 1979.

- *EU/ME The European chapter on metaheuristics* http://webhost.ua.ac.be/eume/
- Test problems
  - OR-LIBRARY http://www.brunel.ac.uk/depts/ma/research/jeb/info.html

Background and motivation

- Many real-world optimization problems involve discrete choices
- Operations Research (OR), Artificial Intelligence (AI)
- Discrete Optimization Problems are often computationally hard
- Real world problems need to be ”solved”
- Complexity Theory gives us bleak prospects regarding exact solutions
- The quest for optimality may have to be relaxed
- Having a good, approximate solution in time may be better than waiting forever for an optimal solution
- Modeling problem
- Optimization not the only aspect
- Response time requirements

- Heuristic methods

Real-life, important DOP

The Knapsack Problem

- n ”articles” *{1,...,n}* available for selection, weights c_i utilities v_i
- Knapsack with capacity *C*
- Find the selection of articles that maximizes total utility and obeys capacity

$$x_i = \begin{cases} 1 & \text{if article } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{s.t.}$$

$$\sum_{i=1}^{n} c_i x_i \leq C$$

$$x_i \in \{0,1\},\ i = 1,\ldots,n$$

Example: – Selection of projects

- You manage a large company
- Your employees have suggested a large number of projects
  - resource requirements
  - utility
- Fixed resource capacity
- Select projects that will maximize utility

- Strategic/tactical decision
- Discrete optimization problem

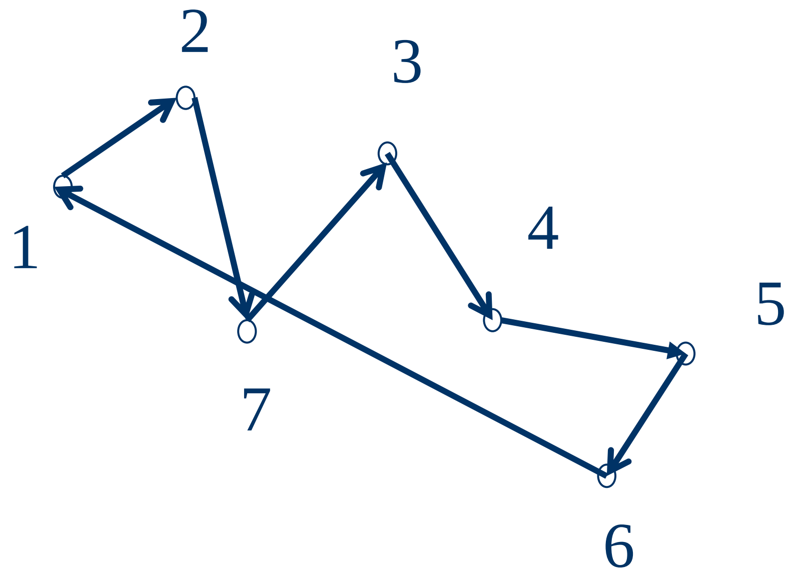
- *The Knapsack problem*

Example: - Transportation

- You have a courier company and a car
- You know your orders for tomorrow, pickup and delivery points
- You know the travel time between all points
- You want to finish as early as possible

- Operational decision
- Discrete optimization problem
- *The Traveling Salesman Problem*

Traveling Salesman Problem (TSP)



	1	2	3	4	5	6	7
1	0	**17**	18	23	23	23	23
2	2	0	88	32	28	27	**22**
3	27	33	0	**23**	37	43	23
4	33	73	14	0	**9**	23	19
5	29	65	26	65	0	**24**	25
6	**25**	99	29	35	43	0	33
7	83	40	**23**	43	77	73	0

Feasible (candidate) solution:

1 2 7 3 4 5 6 1 ; objective value (cost): 143

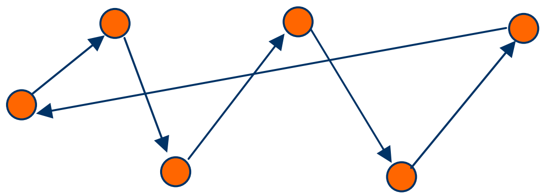
”Nearest neighbor” – example of a greedy heuristic - O(n^2)

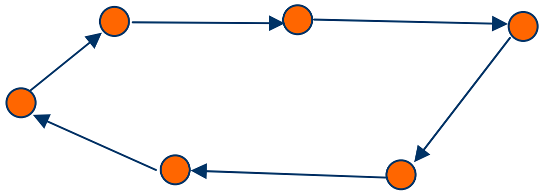
Calculation of the objective is O(n)

No constraints, except for the round trip requirement

The number of distinct round trips is (n-1)! (in the asymmetrical case)

Greed is good [Gordon Gekko 1987]





- A greedy heuristic rarely gives the optimal solution
- The Nearest Neighbor heuristic

Problem (types) and problem instances

- Example: TSP
- A type of concrete problems (instances)
- An instance is given by:
  - *n*: the number of cities
  - *A*: *nxn*-matrix of travel costs

Optimization Problem
- Mathematical formulation

- Decision variables with domains
- Objective function
- Constraints

- Mathematical program

$$\min \quad f(x_1,\ldots,x_n)$$
$$f_j(x_1,\ldots,x_n)=0 \quad j=1,\ldots,k$$
$$g_j(x_1,\ldots,x_n)\le 0 \quad j=1,\ldots,l$$
$$x_i \in \mathbb{R} \quad i=1,\ldots,n$$

$$\min \quad f(\mathbf{x})$$
$$\mathbf{x}\in\mathbf{S}$$

Linear Integer Program

$$\max \quad \zeta = \sum_{j=1}^{n} c_j x_j \quad \text{s.t.}$$
$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad i = 1,\dots,m$$
$$x_j \ge 0 \quad j = 1,\dots,n$$

$$\max \quad \zeta = \sum_{j=1}^{n} c_j x_j \quad \text{s.t.}$$
$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad i = 1,\dots,m$$
$$x_j \ge 0 \quad j = 1,\dots,n$$
$$x_i \in \mathbb{N}^0 \quad i \in I \subseteq \{1,\dots,n\}$$

- Mixed Integer Programs – MIP $\quad i \in I \subset \{1,\dots,n\}$
- Pure Integer Programs – IP, PIP $\quad i \in I = \{1,\dots,n\}$
- 0-1 programs $\quad x_i \in \{0,1\},\ i \in I$

(Linear) Integer Programming

- Many problems may be modeled as LP with integrality constraints
- Discrete choices, sequences, combinatorics, logic, ...
- Planning, scheduling, ...

- In general, IPs are computationally much more difficult to solve than LPs
- Often, the computing time for exact methods grow ”exponentially” with the size of the instance
- But not always ...

Definition – Discrete Optimization Problem

A Discrete Optimization Problem (DOP) is

- either a minimization or maximization problem
- specified by a set of problem instances

Definition – DOP instance

A **DOP instance** is a pair $(\mathbf{S}, f)$
where $\mathbf{S}$ is the set of **feasible solutions (the search space**) and $f : \mathbf{S} \to \mathbb{R}$ is the **objective (cost function).**

The goal is to find **a global optimum**:

$$s^* \in \mathbf{S} : f(s^*) \le f(s), \forall s \in \mathbf{S}$$

$f^* = f(s^*)$ (globally) optimal value

$\mathbf{S}^* = \left\{ s \in \mathbf{S} : f(s) = f^* \right\}$ (globally) optimal solutions

$\mathbf{X} \supseteq \mathbf{S}$ the **solution space**, also including infeasible **solutions**

Example 1: An asymmetrical TSP-instance

3 cities: 1, 2, 3

	1	**2**	**3**
1	0	15	32
2	13	0	3
3	2	17	0

$(\mathbf{S}, f)$

$$\min_{s \in \mathbf{S}} f(s)$$

$$\mathbf{S} = \{(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)\} \equiv \{s_1,\ldots,s_6\}$$

$$f(s_1) = 15+3+2 = 20$$

$$f(s_2) = 32+17+13 = 62$$

Observations

- In principle, the TSP is a very simple problem
- All feasible solutions can be represented by a permutation
- There is a finite number of solutions
- The objective is easy to compute for a given solution
- The British Museum algorithm: look everywhere
- The number of feasible solutions for an *n*-city (asymmetric) TSP is *(n-1)!*

Combinatorial explosion

10!	3628800	10^6
20!	2432902008176640000	10^{19}
50!	30414093201713378043612608166064768844377641568960512000000000000	10^{65}
100!	93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000	10^{159}

~ # atoms in our galaxy

# atoms in the universe ~10^{80}
# nanoseconds since Big Bang ~10^{26}

Example 2: The Knapsack Problem

- n ”articles” *{1,...,n}* available for selection, weights c_i utilities v_i
- Knapsack with capacity *C*
- Find the selection of articles that maximizes total utility and obeys capacity

$$x_i = \begin{cases} 1 & \text{if article } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{s.t.}$$

$$\sum_{i=1}^{n} c_i x_i \le C$$

$$x_i \in \{0,1\},\ i = 1,\dots,n$$

- how to specify/represent a problem instance?
- how to represent a solution?
- what are the sets **X** and **S**?

Example 1: 0-1 Knapsack-instance

Knapsack with capacity 100

10 ”articles” (projects, ...) 1,...,10

$(\mathbf{S}, f)$

$\max_{s \in \mathbf{S}} f(s)$

	1	2	3	4	5	6	7	8	9	10
Utility	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52

$$\mathbf{X} = \{0000000000, \ldots, 1111111111\} \equiv \{x_1, \ldots, x_{1024}\}$$

$$f(x_1) = 0$$

$$\vdots$$

$$f(x_{530}) = 117$$

$$\vdots$$

$$f(x_{1024}) = 464$$

$$f^* = f(x_{530}) = 117$$

$$x^* = \{0100100001\}$$

Comments on the definition of DOP

- ***S*** is rarely given explicitly, defined through constraints/relations
- ***S*** is often (small) subset of the total search space ***X***
- *f(s)* is rarely given explicitly, must be computed by a procedure
- there is often a compact representation of a problem instance and a (candidate) solution
- modelling is important
  - mathematical modelling
  - conceptual modelling
- a (candidate) solution is given by a valuation of the decision variables $(x_1,v_1), \ldots, (x_n,v_n)$ $\vec{x} \perp \vec{v}$
- often there are efficient (low polynomial) algorithms for checking feasibility (***S*** membership) and objective value for candidate solutions

DOP Applications

- Decision problems with discrete alternatives
- Synthesis problems
  - planning, scheduling
  - configuration, design
- Limited resources
- OR, AI
- Logistics, design, planlegging, robotics
- Geometry, Image analysis, Finance ...

Solution methods for DOP

- Exact methods that guarantee to find an (all) optimal solution(s)
  - generate and test, explicit enumeration
  - mathematical programming
- Approximation methods
  - with quality guarantees
  - heuristics

- Collaborative methods

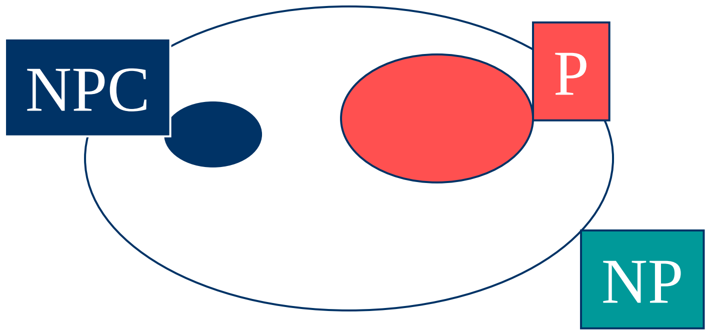
Computational Complexity Theory

- Computing time (memory requirements) for problem types
  - “the best” algorithm
  - over all instances
  - as function of problem size
- “Exponential” growth is cruel ...
- Parallel computing and general speed increase does not help much
- Problem type is considered tractable only of there is a polynomial time algorithm for it

- Worst case, pessimistic theory
- One problem instance is enough to deem a problem type as computationally intractable

Complexity classes of problem types

- Complexity classes
  - *P*
  - *NP*
  - *NP*-complete
- Cook's conjecture:



$$\boldsymbol{P} \neq \boldsymbol{NP} \text{ or } \boldsymbol{NP} \setminus \boldsymbol{P} \neq \varnothing$$

$LP \in P$	Kachian (1979)
$SAT \in NPC$	Cook (1971)
$TSP \in NPC$	Karp (1972)
$Knapsack \in NPC$	Karp (1972)

Motivation for heuristic DOP algorithms

- Computational Complexity theory
- Basic Computational Complexity Theory studies decision problems
- Close relation between decision problem and optimization problem
- The optimization equivalent is at least as hard as the decision variant
- ***NP***-complete decision problem -> ***NP***-hard optimization problem
- For ***NP***-hard DOPs there exist no polynomial time exact algorithm, unless ***P=NP***
- For some NP-hard DOPs there exist pseudo-polynomial, exact algorithms
  - The one-dimensional Knapsack problem is weakly ***NP***-hard
  - The multi-dimensional Knapsack problem is strongly ***NP***-hard
  - The TSP is strongly ***NP***-hard
- Alternatives
  - exact method
  - approximation method with performance guarantee
  - heuristic method (with no or weak a priori guarantees)
  - performance ratio of given approximation algorithm

$$R_A(I) = \frac{A(I)}{OPT(I)}$$

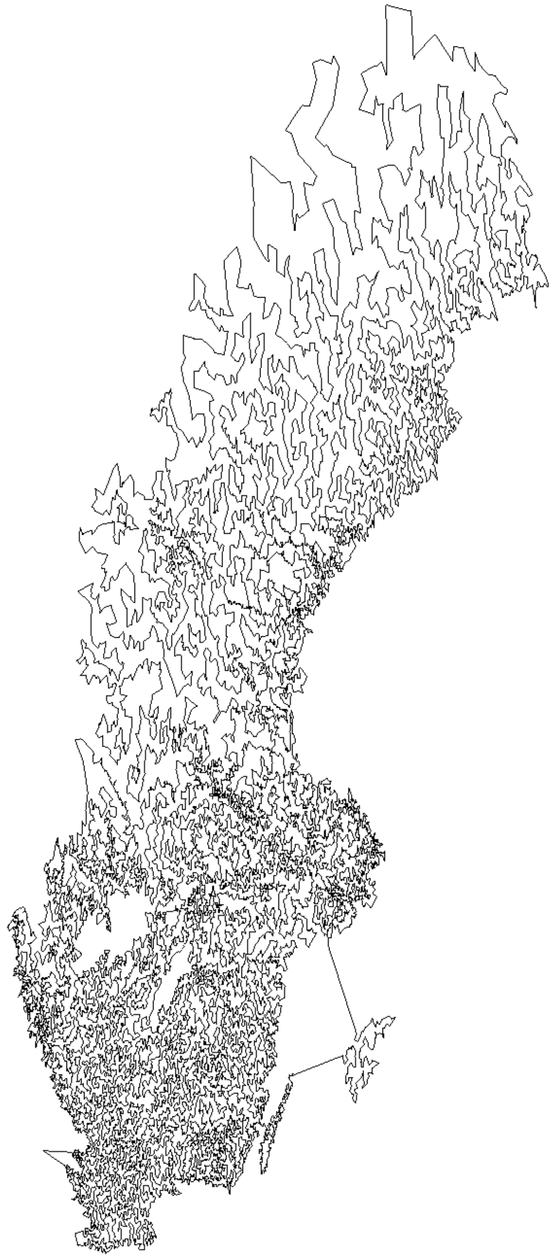
Some messages (1)

- Not all DOPs are ***NP***-hard, e.g., the Assignment Problem
- Even ***NP***-hard problems may be effectively solved
  - small instances
  - special structure
  - weakly ***NP***-hard
- Even large instances of strongly ***NP***-hard problems may be effectively solved to optimality
  - TSPs with a few hundred cities in a few seconds

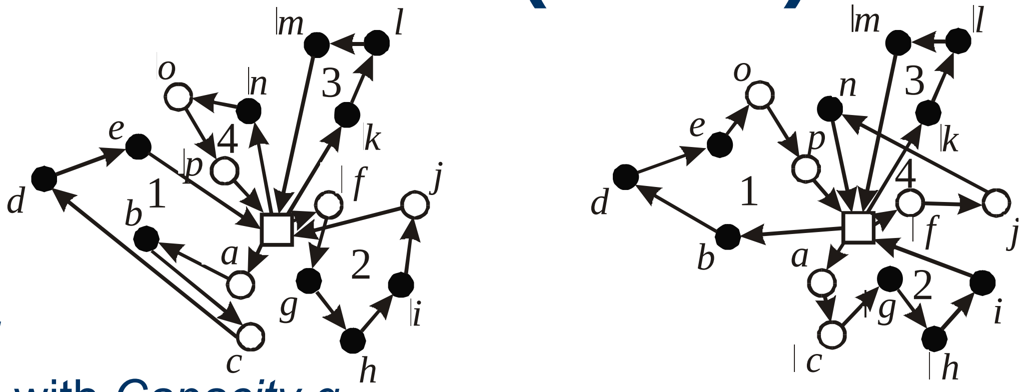
Large TSPs

- 24,978 Cities in Sweden
- 2004, The Concorde TSP solver
- 84.8 CPU years on a single
  Intel Xeon 2.8 GHz processor

- Largest TSP solved:
  85,900 Locations in a VLSI Application
- Challenge: World tour of 1,904,711 places
  best solution within 0.076% of optimum
- http://www.tsp.gatech.edu/



VRP with Capacity Constraints (CVRP)



- Graph *G*=(*N*,*A*)
  - *N*={0,…,*n*+1} *Nodes*
  - *0 Depot*, *i*≠*0 Customers*
  - *A*={(*i*,*j*): *i*,*j*∈*N*} Arcs
  - c_{ij} *>0 Transportation Costs*
- *Demand* d_i for each *Customer i*
- *V* set of identical *Vehicles* each with *Capacity q*
- Goal
  - Design a set of *Routes that* start and finish at the *Depot* - with *minimal Cost.*
  - Each *Customer* to be visited only once (no order splitting)
  - Total *Demand* for all *Customers* not to exceed *Capacity*
  - *Cost*: weighted sum of *Driving Cost* and *# Routes*
- DVRP – distance/time constraint on each route
- VRPTW – VRP with time windows
- Pickup and Delivery
  - Backhaul – VRPB(TW)
  - Pickup and delivery VRPPD(TW)
  - PDP

A mathematical model for VRPTW
(Network Flow Formulation)

$$\text{minimize} \quad \sum_{k\in V}\sum_{(i,j)\in A} c_{ij}x_{ij}^k \qquad (1)$$

minimize cost

subject to:

$$\sum_{k\in V}\sum_{i\in N} x_{ij}^k = 1, \qquad \forall j \in C \qquad (2)$$

each customer 1 time

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ij}^k \le q, \qquad \forall k \in V \qquad (3)$$

Capacity

$$\sum_{j\in N} x_{0j}^k = 1, \qquad \forall k \in V \qquad (4)$$

k routes out of depot

$$\sum_{i\in N} x_{ih}^k - \sum_{j\in N} x_{hj}^k = 0, \qquad \forall h \in C, \ \forall k \in V \qquad (5)$$

flow balance for each customer

$$\sum_{i\in N} x_{i,n+1}^k = 1, \qquad \forall k \in V \qquad (6)$$

k routes into depot (redundant)

$$x_{ij}^k(s_i^k + t_{ij} - s_j^k) \le 0, \qquad \forall (i,j) \in A, \ \forall k \in V \qquad (7)$$

sequence and driving time

$$a_i \le s_i^k \le b_i, \qquad \forall i \in N, \ \forall k \in V \qquad (8)$$

arrival time in time window

$$x_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A, \ \forall k \in V \qquad (9)$$

arc (i,j) driven by vehicle k

Variables -arrival time

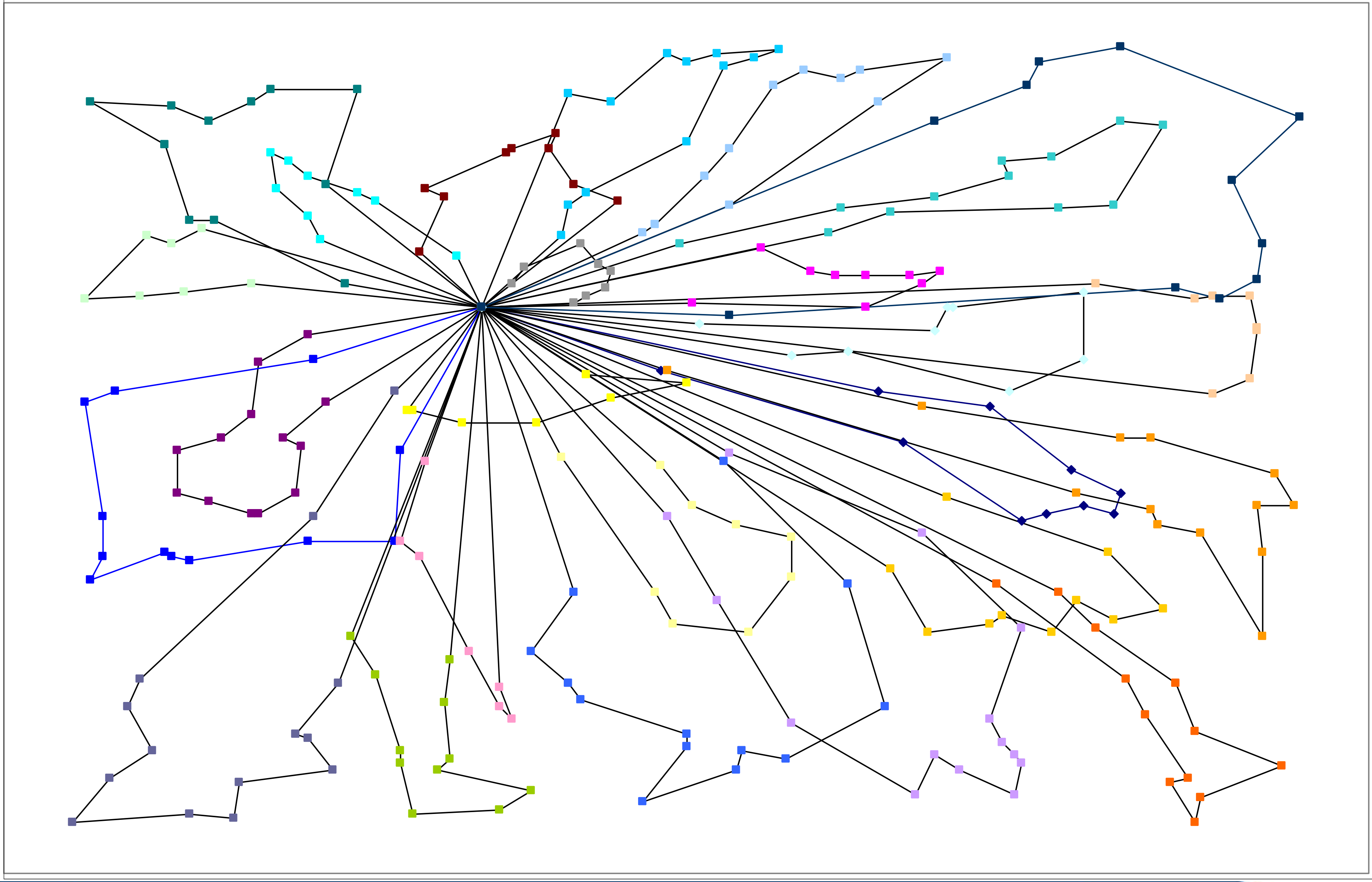
Arc Decision variables

Geir Hasle - eVITA Winter School 2009

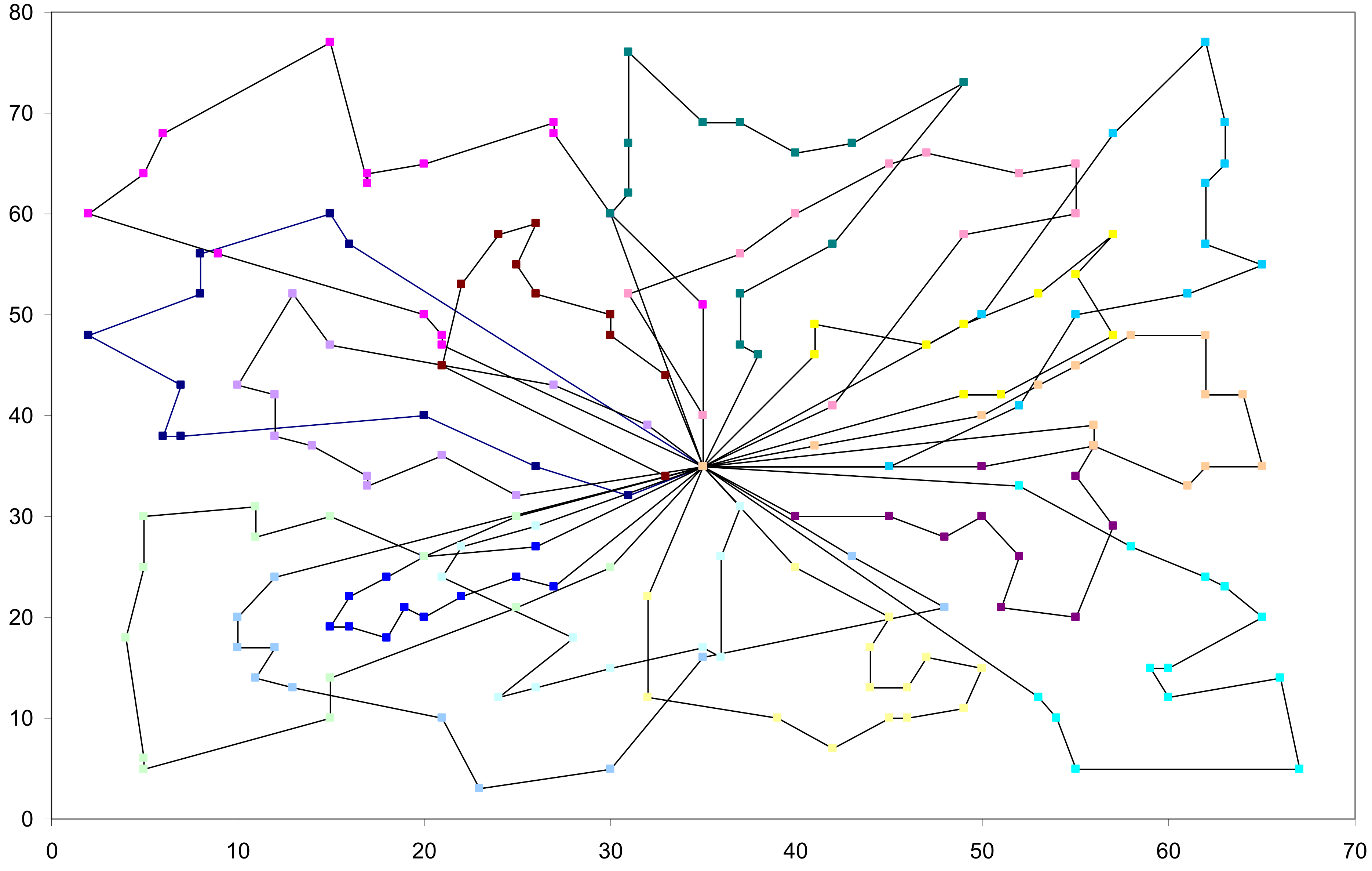
Complexity of VRP(TW) and State-of-the-art: Exact Methods

- Basic VRP (CVRP)
  - Strongly NP-hard
  - Branch & Bound + basic relaxations
  - Lagrange Relaxation
  - Set Partitioning, Column Generation
  - Branch & Cut
  - Consistently solve problem instances with 70 customers in reasonable time
- VRPTW: finding feasible solution is NP-complete
  - Dantzig-Wolfe decomposition, CG
    - subproblem: SPP med capacities and time windows
  - Lagrange Relaxation
  - Consistently solve problem instances with 100 customers in reasonable time
- Approximation Methods, Heuristics

G-n262-k25: 5685 vs. 6119, 5767 CPU s



M-n200-k16: First known feasible solution

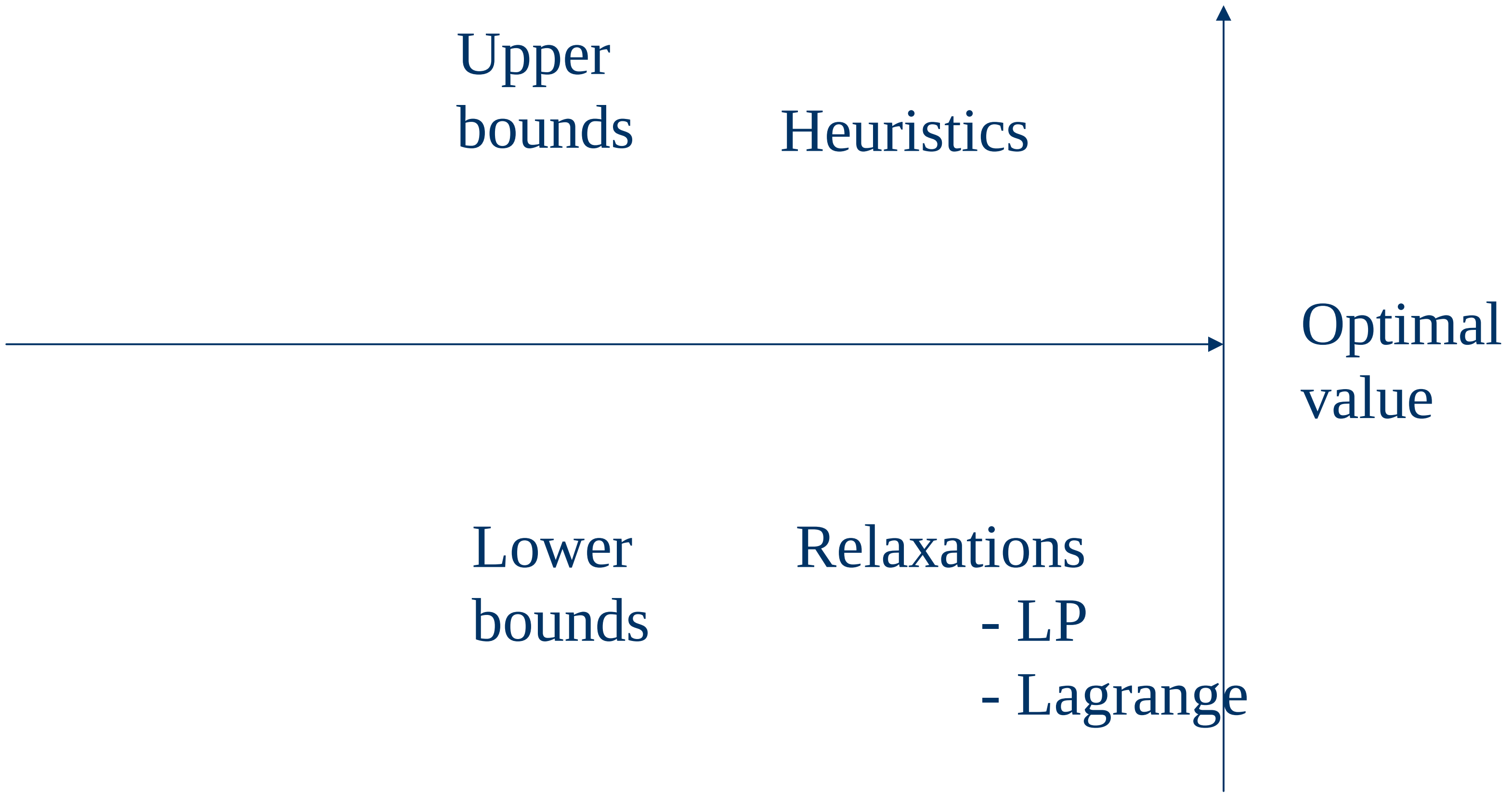


Some messages (2)

- DOPs should be analyzed
- Try exact methods
- Collaboration between exact and heuristic methods
  - a good, heuristic solution may jump-start and speed up an exact method
  - exact methods may give high quality bounds
  - true collaboration, asynchronous parallel algorithms

Quality Assessment

- upper and lower bounds (minimization)



Further motivation - heuristics

- In the real world
  - response requirements
  - instance size and response requirements may rule out exact methods
  - optimization is just one aspect
  - modelling challenges, what is the objective?
  - humans are satisficers, not optimizers [Herb Simon]
  - generic solver, all kinds of instances, robustness
- Heuristic methods are generally robust, few drastic assumptions

- Exact methods should not be disqualified a priori
- Cultures
  - mathematicians vs. engineers/pragmatists
  - OR vs. AI animosity
  - reconciliation

Exact methods for DOP

- DOPs typically have a finite # solutions
- Exact methods guarantee to find an optimal solution
- Response time?

- Good for solving limited size instances
- May be good for the instances in question
- Some (weakly) NP-hard problems are effectively solved, given assumptions on input data
- Basis for approximation methods
- Subproblems, reduced or relaxed problems

Heuristics - definitions

- **Wikipedia:** “Heuristics stand for strategies using readily accessible, though loosely applicable, information to control problem-solving in human beings and machines”.
- **Greek:** (h)eureka – “I have found it”, Archimedes 3rd century BC
- **Psychology:** Heuristics are simple, efficient rules, hard-coded by evolutionary processes or learned, which have been proposed to explain how people make decisions, come to judgments, and solve problems, typically when facing complex problems or incomplete information. Work well under most circumstances, but in certain cases lead to systematic cognitive biases.
- **Mathematics:** “How to solve it” [G. Polya 1957]. Guide to solution of mathematical problems.
- **AI:** Techniques that improve the efficiency of a search process often by sacrificing completeness
- **Computing science:** Algorithms that ignore whether the solution to the problem can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains, or intersects with, the solution of the more complex problem. Heuristics are typically used when there is no known way to find an optimal solution, or when it is desirable to give up finding the optimal solution for an improvement in run time.

Heuristics in Discrete Optimization

- Sacrificing the guarantee of finding the optimal solution
- Strong guarantees regarding solution quality vs. response time typically cannot be given

- General heuristics
  - strategies for traversing the Branch & Bound tree in MIP
- Greedy heuristics
- Special heuristics, exploiting problem structure
- Basic method: Local Search
- Better methods: Metaheuristics

How to find a DOP solution?

- Exact methods

- Earlier solution
- Trivial solution
- Random solution
- Constructive method
  - gradual build-up of solutions from scratch
  - greedy heuristic
- Solve simpler problem
  - remove or change constraints
  - modify objective

- Given a solution, modify it

Local Search and Meta-Heuristics

- Operate on a ”natural” representation of solutions
- The combinatorial object
- Search in the space of feasible solutions / all solutions (search space, solution space)

- Single solution: Trajectory based methods
- Multiple solutions: Population based methods

Local search for DOP

- Dates back to late 1950ies, TSP work
- Renaissance in the past 20 years
- Heuristic method
- Based on small modifications of given solution
- Ingredients:
  - Initial solution
  - Operator(s), Neighborhood(s)
  - Search strategy
  - Stop criterion
- Iterative method
- Anytime method

Example: TSP

Trivial solution:

1 2 3 4 5 6 7 (288)

Greedy construction:

1 3 5 7 6 4 2 (160)

	1	2	3	4	5	6	7
1	0	18	**17**	23	23	23	23
2	**2**	0	88	23	8	17	32
3	17	33	0	23	**7**	43	23
4	33	**73**	4	0	9	23	19
5	9	65	6	65	0	54	**23**
6	25	99	2	**15**	23	0	13
7	83	40	23	43	77	**23**	0

Example: 0-1 Knapsack

	1	2	3	4	5	6	7	8	9	10
Utility	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52

- Knapsack capacity 100
- 10 ”articles” (projects, ...) 1,...,10
- Trivial solution: empty knapsack, utility 0
- Greedy solution, add articles in descending utility sequence:
  - (0000010000), utility 85
  - Relative utility

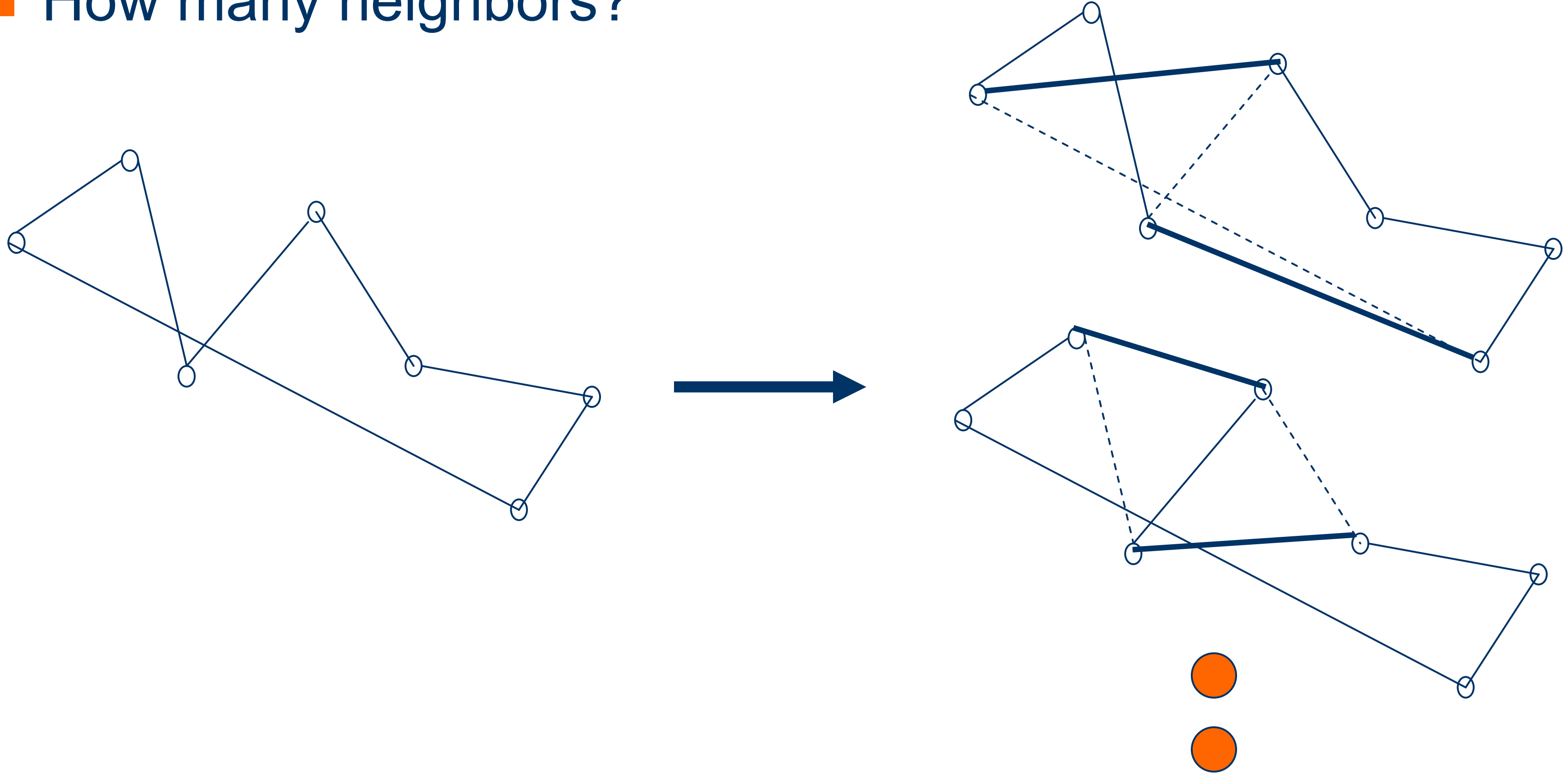
	1	2	3	4	5	6	7	8	9	10
Utility/size	0.93	**1.23**	0.98	0.86	**1.18**	0.89	0.77	0.89	0.82	**1.13**

Given a solution, how to find a better one?

- Modification of given solution gives ”neighbor”
- A certain type of **operation** gives a set of neighbors: a **neighborhood**
- Evaluation
  - objective
  - feasibility

Example: TSP

- Operator: 2-opt
- How many neighbors?



Example: Knapsack

	1	2	3	4	5	6	7	8	9	10
Utility	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
	0	0	1	0	1	0	0	0	0	0

- We have a solution 0010100000 with value 73
- Simple operator (Flip): Change status of an element, i.e.,
  - if the article is in, take it out
  - if the article is out, put it in
- Some neighbors:
  - 0110100000 utility 105
  - 1010100000 utility 152, non-feasible
  - 0010000000 value 47
- *n* neighbors
- Other operators: 1-exchange, 2-exchange, ...

Definition: Neighborhood function

Let *(**S**,f)* be a DOP-instance. A neighborhood function is a mapping $N : \boldsymbol{S} \rightarrow 2^{\boldsymbol{S}}$

that, for a given solution $s \in \boldsymbol{S}$ defines
a **neighborhood** of solutions $\boldsymbol{N}(s) \subseteq \boldsymbol{S}$
that in some sense are ”close to” s
$t \in \boldsymbol{N}(s)$ is said to be a neighbor of s
relative to $\boldsymbol{N}$

Neighborhood operator

- Neighborhood functions are often defined through certain **generic operations** on a solution - **operator**
- Normally rather simple operations on key structures in the combinatorial object
  - removal of an element
  - addition of an element
  - exchange of two or more elements
- Multiple neighborhood functions - qualification by operator $N_\sigma(s), \sigma \in \Sigma$

Local Search (Neighborhood Search)

- Starting point: initial solution s_0
- Iterative search in neighborhoods for better solution
- Sequence/path of solutions $s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$

- Path is determined by
  - Initial solution
  - Neighborhood function
  - Acceptance strategy
  - Stop criterion

- What happens when the neighborhood contains no better solution?
- Local optimum

Definition: Local optimum

Let *(**S**,f)* be a DOP-instance, and let ***N*** be a neighborhood function. A solution $\hat{s}$ is a **local optimum** (minimum) w.r.t. ***N*** if:

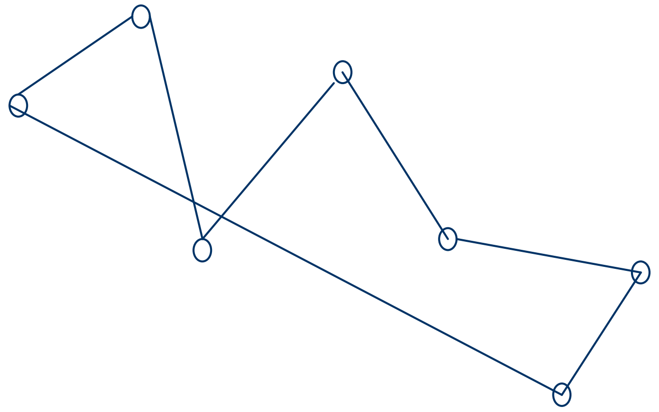
$$f(\hat{s}) \leq f(t), \forall t \in \boldsymbol{N}(\hat{s})$$

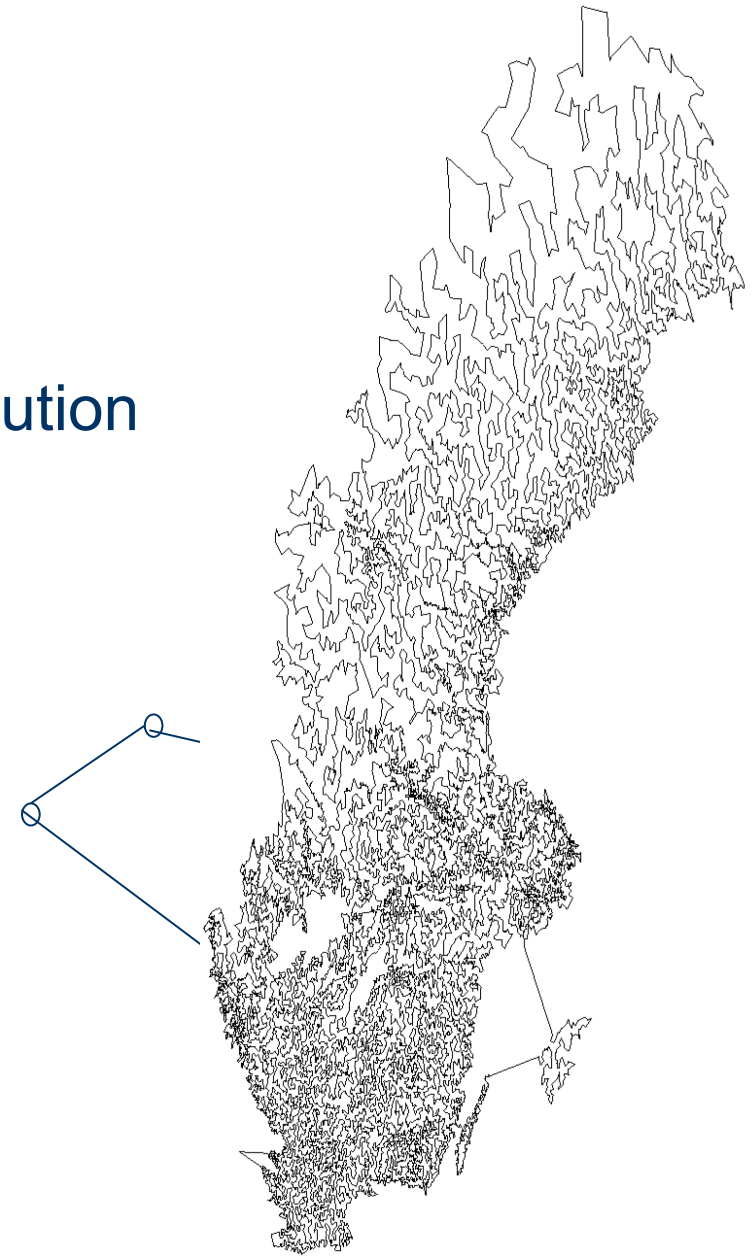
The set of locally optimal solutions: $\hat{\boldsymbol{S}}$

NB! Local optimality is relative to the neighborhood

Example: TSP

- Operator: 2-opt
- Local minimum (2-optimal) solution





Definition: Exact neighborhood

Let *(**S**,f)* be a DOP-instance and let
N be a neighborhood function. ***N*** is **exact** if:

$$\hat{S} \subseteq S^{*}$$

N is exact if local optima for the neighborhood ***N*** are also global optima.

Local Search (Neighborhood Search)

- Alternative strategies for exploring the neighborhood
- Different strategies will give different paths in the search space
- Accept the first (feasible) improving solution (”First Accept”)
- Complete exploration of the neighborhood
  - move to the **best improving** solution (”Steepest Descent”, ”Hill Climbing”, ”Iterative Improvement”)
  - always move to the best solution in the neighborhood, whether improving or not (”Best Neighbor”)

- Other strategies?

Local_Search (S,f,N,strategy)

```
*/ strategy is ”First Accept” or ”Best Accept”
current:=Init_Solution(S,f)
incumbent:=current                    */ best solution until now
local_optimum:=**false**
**while not** local_optimum **do**             */ other stop criteria may be envisaged
    (current,incumbent,local_optimum):=
        Search_the_Neighborhood (current,N(current),f,strategy,incumbent)
    **if** local_optimum **return** incumbent
**od**
```

Search_the_Neighborhood (current,Neighbors,f,strategy,incumbent)

```
best_neighbor:=current
for n in Neighbors do
        if f(n) < f(best_neighbor) then best_neighbor:=n fi  */ minimization
         if f(n) < f(incumbent) then
              if strategy=”First Accept” then
                 return (n,n,false) else
                 incumbent:=n                          */ strategy is ”Best Accept”
              fi
         fi
od
return (best_neighbor,incumbent,best_neighbor=current)
*/    Returns multiple value / structure:      (current,incumbent,local_optimum)
*/    Assuming that Neighbors are feasible
```

Observations and Structures

- LS with either ”First Accept” or ”Steepest Descent” stops in a local optimum (unless there are other stop criteria)
- If the neighborhood ***N*** is exact, Local Search with these strategies are exact optimization methods
- The neighborhood function ***N*** induces a directed graph, **the Neighborhood Graph** ***G_N* = *(X,A_N)*** where nodes are the members of the search space, and ***N*** defines the arcs:

$$\boldsymbol{A}_N = \left\{ \left(x, x' \right) : x \in \boldsymbol{X}, x' \in \boldsymbol{N}(x) \right\}$$

- Many neighborhood functions are symmetric $x' \in \boldsymbol{N}(x) \Rightarrow x \in \boldsymbol{N}(x') \; x, x' \in \boldsymbol{X}$
- A Local Search process defines a **trajectory** in the Neighborhood Graph

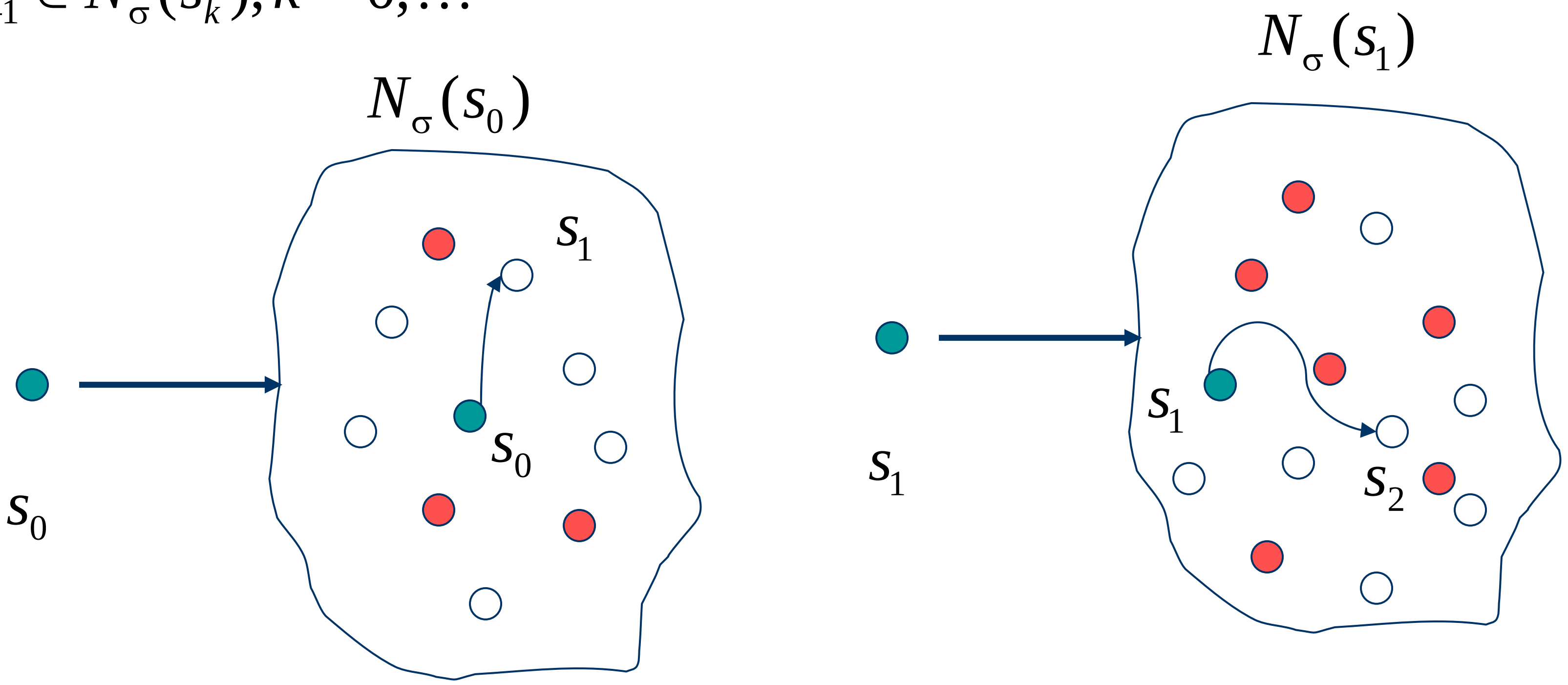
$$s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$$

- Associated with each node, there is a value that defines the ”topography” defined by the objective (or, more generally, an evaluation function) $f(x)$

- Search Landscape *(**S**,**N**,f)*

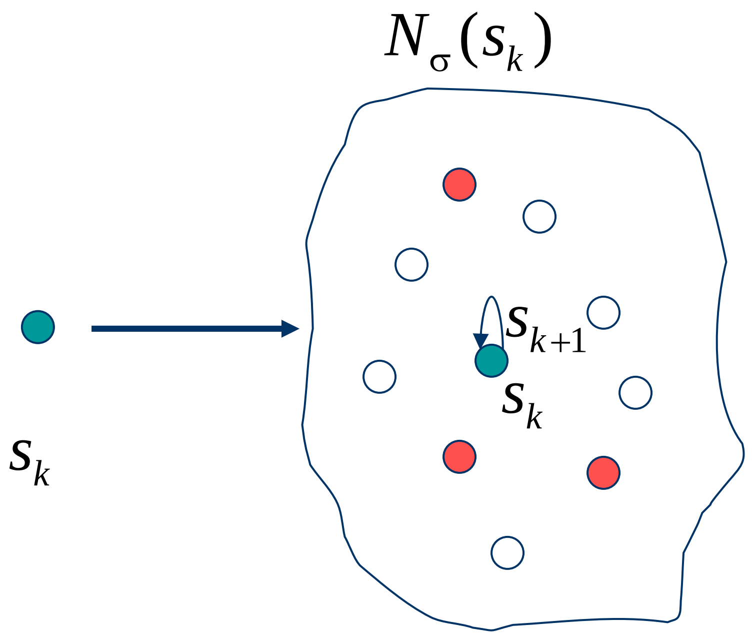
Traversing the Neighborhood Graph

$$s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$$



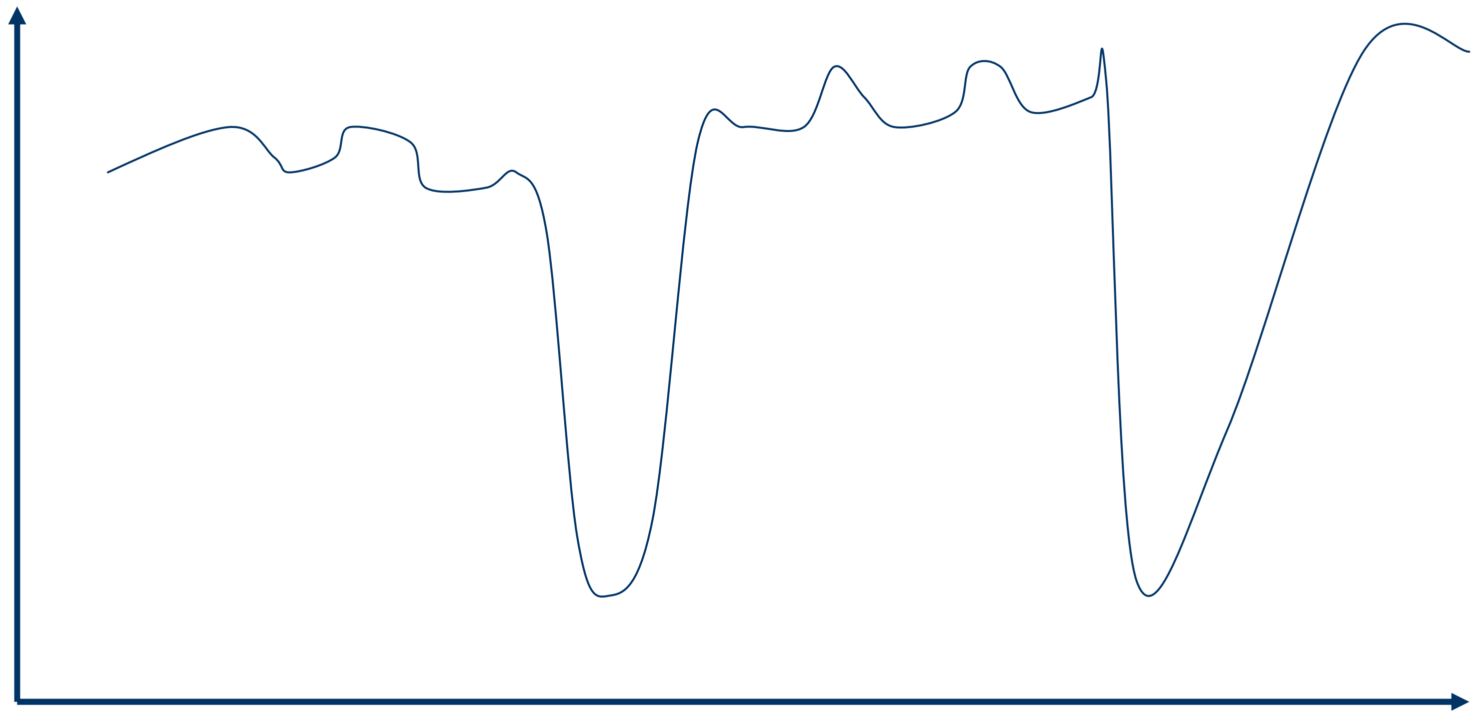
A *move* is the process of selecting a given solution in the neighborhood of the *current* solution, hence making it the current solution for the next iteration

Local Optimum



Search Landscapes
- Local and global optima

Objective value



Solution space

Simplex algorithm for LP as Local Search

- Simplex Phase I gives initial, feasible solution (if it exists)
- Phase II gives iterative improvement towards optimal solution (if it exists)
- The neighborhood is defined by the polyhedron
- The strategy is ”Iterative Improvement”
- The concrete moves are determined by pivoting rules
- The neighborhood is exact, i.e., Simplex is an exact optimization algorithm (for certain pivoting rules)

Local Search

- Main challenges
  - feasible region only, or the whole solution space?
  - design good neighborhoods
  - size, scalability, search landscape
  - initial solution
  - strategy
  - efficient evaluation of the neighborhoods
    - incremental evaluation of constraints
    - incremental evaluation of the objective (evaluation function)
  - stop criteria
  - performance guarantees

- The performance is typically much better than greedy heuristics

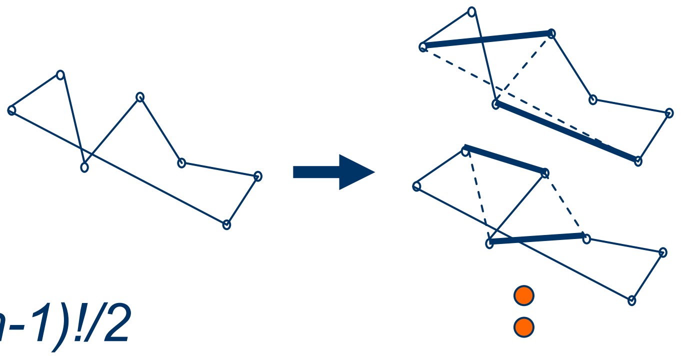
Design of neighborhood operators

- Based on natural attributes
- Neighborhood size, scalability
- Diameter: maximum # moves to get from one solution to another
- Connectivity

- Search complexity depends on Search Landscape

- Distance metrics
- Hamming distance, Edit distance

Example: Symmetric TSP and 2-opt



- Solution space cardinality *(n-1)!/2*
- Neighborhood cardinality *n(n-1)/2*
- Connected
- Diameter between *n/2* and *n-1* (still open)
- Simple move representation
- Objective Difference (delta evaluation) is simple and efficient
- Feasibility is no problem ...
- Generalization: *k*-opt
- *n*-opt is an exact neighborhood, Diameter is *1*, but ...

Diameter of 0-1 Knapsack problem with the ”Flip” neighborhood

$|\boldsymbol{X}| = 2^n$

- One is never more than *n* moves away from the optimal solution
- but the landscape you have to move through may be very bumpy ...

Knapsack instance Idaho20

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{s.t.}$$

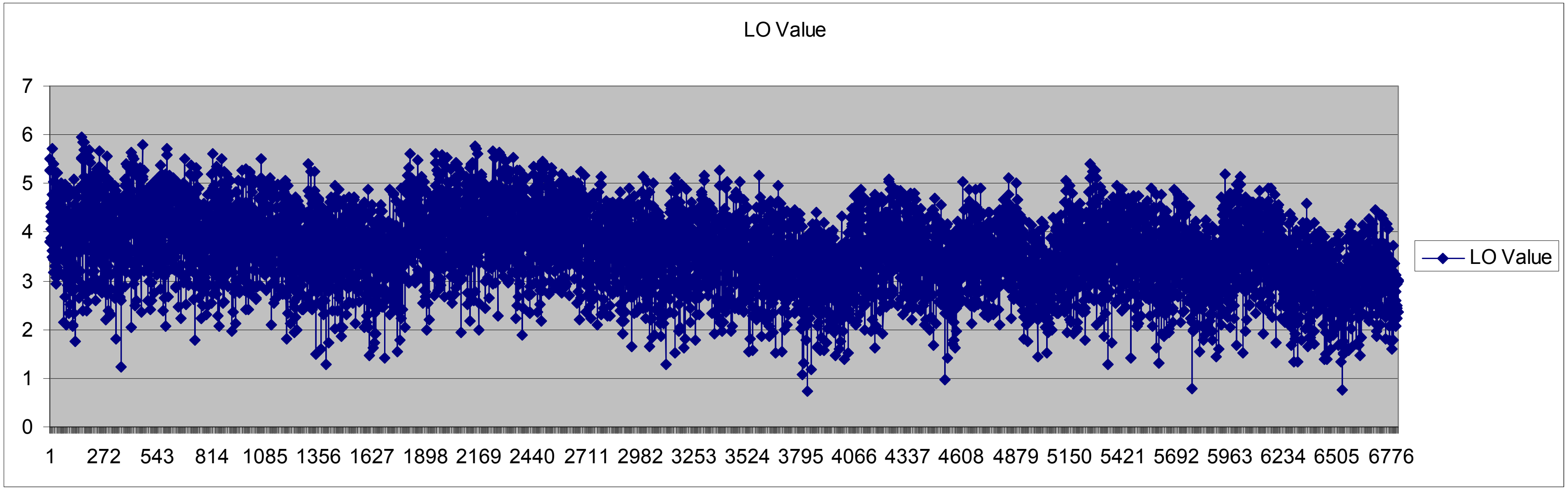
$$\sum_{j=1}^{n} c_i x_i \leq C$$

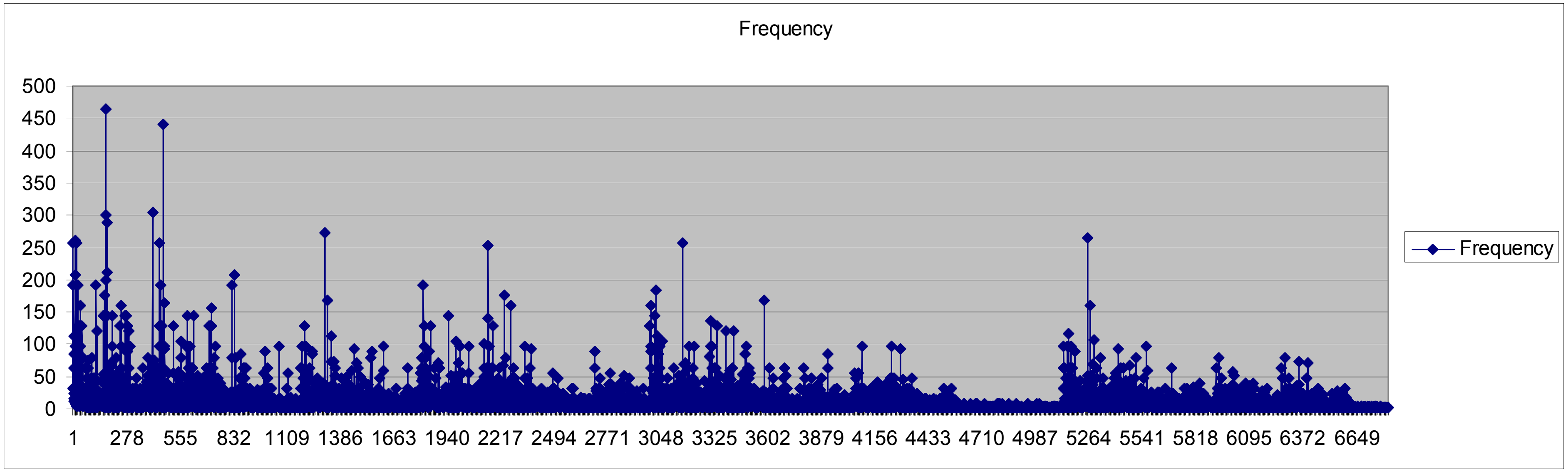
$$x_i \in \{0,1\}$$

Idaho20
n 20 **C** 2.5 **f*** 5.949363 **s*** 00110010101101001010 (slack 0.02)
v 0.751231 0.0562173 0.586932 0.695919 0.10468 0.242555 0.832725 0.00696871 0.828839 0.513085 0.704328 0.63044 0.556193 0.507427 0.159593 0.30589 0.573253 0.016571 0.5895 0.320655
c 0.703562 0.658012 0.194159 0.50693 0.372415 0.0674343 0.467352 0.132051 0.336674 0.790007 0.0390611 0.295304 0.530008 0.180224 0.116737 0.740043 0.440325 0.522951 0.0189656 0.725904

- 6810 local optima, value from 0.732882(1) to 5.949363(176)
- ”basin of attraction” size varies from 1 to 464

Search landscape for Idaho20





Knapsack-instance Idaho20ex

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{s.a.}$$

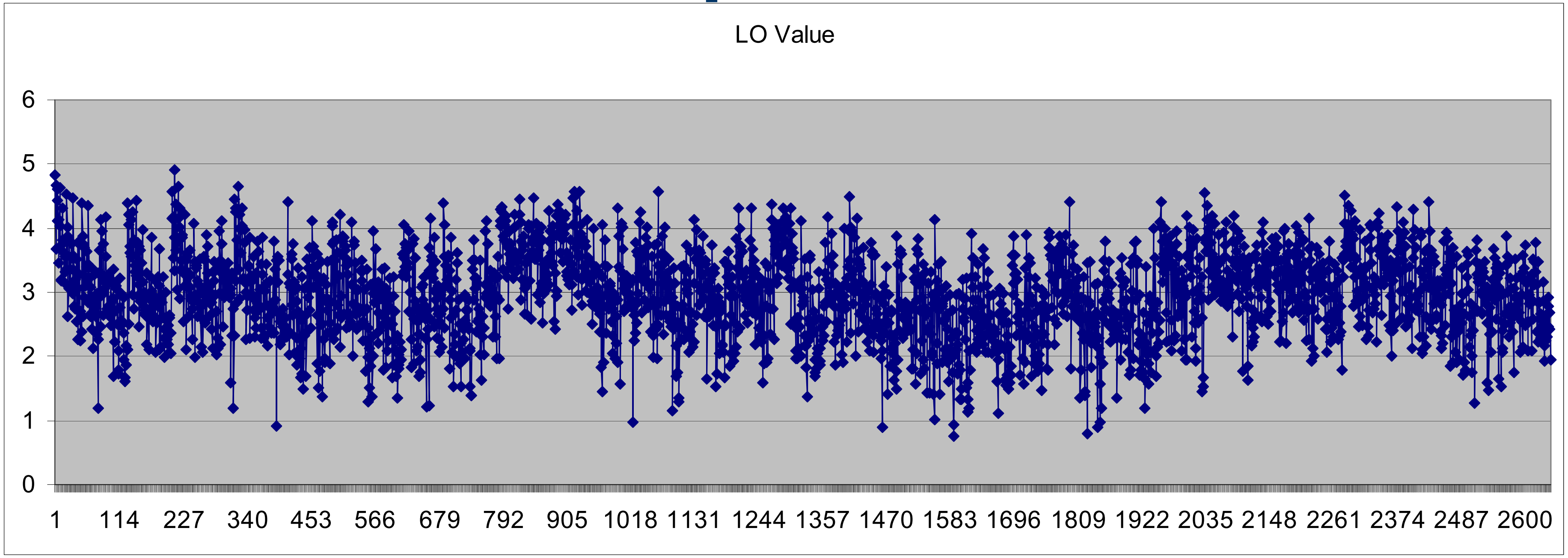
$$\sum_{j=1}^{n} c_i x_i \leq C$$

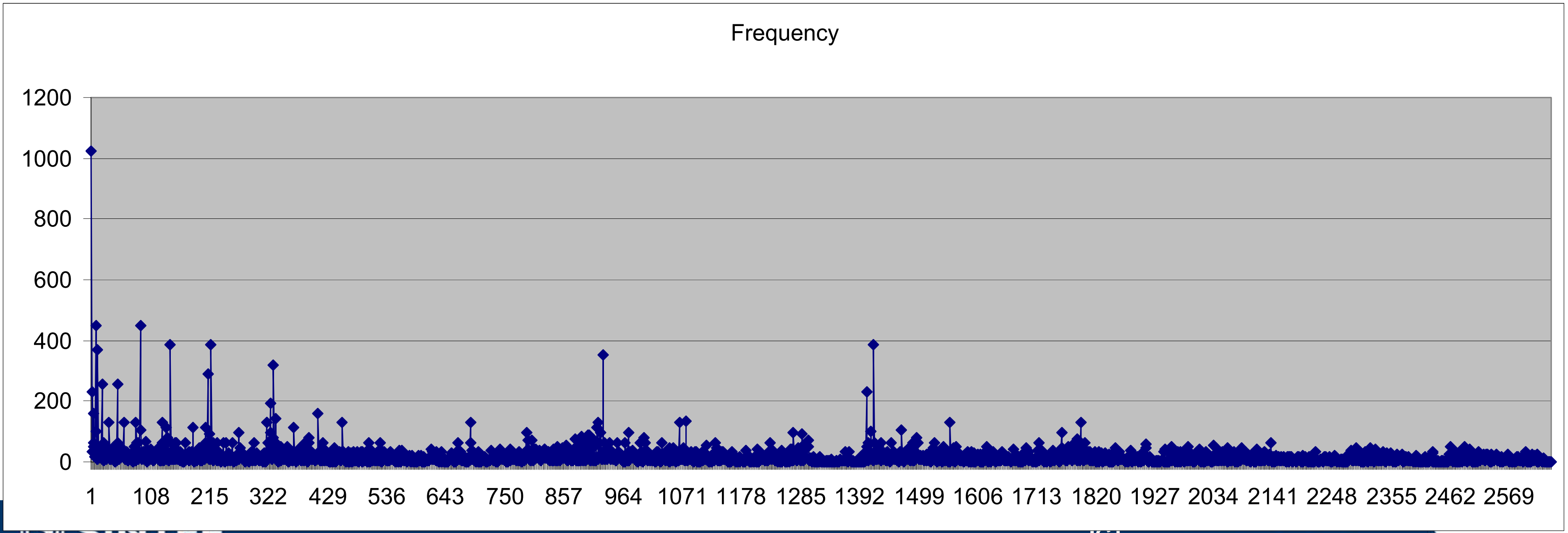
$$x_i \in \{0,1\}$$

Idaho20ex
n 20 **C** 2.5 **f*** 4.911716 **s*** 01001001010010011101 (slack 0.03)
v 0.703562 0.658012 0.194159 0.50693 0.372415 0.0674343 0.467352 0.132051 0.336674 0.790007 0.0390611 0.295304 0.530008 0.180224 0.116737 0.740043 0.440325 0.522951 0.0189656 0.725904
c 0.751231 0.0562173 0.586932 0.695919 0.10468 0.242555 0.832725 0.00696871 0.828839 0.513085 0.704328 0.63044 0.556193 0.507427 0.159593 0.30589 0.573253 0.016571 0.5895 0.320655

- 2644 local optima, value from 0.745181(1) to 4.911716(288)
- ”basin of attraction” size varies from 1 to 1024

Search Landscape Idaho20ex





Local Search

- Old idea, new developments over the past two decades
- Popular method for solving hard DOPs
- Generic and flexible
- No strong assumptions

- ”Anytime”-method
- Efficient, good quality solution quickly
- Performance depending on initial solution, neighborhood operator and strategy

- Exact methods preferable, if they solve the problem

Local Search

- The search landscape is typically complex
- Local optima may be far from global optima
- Local search is a local method
- Solution quality depends on initial solution, neighborhood, strategy

- ”Blind” and ”headstrong” method, no learning during search, no randomness

- No strong performance guarantees

What to do to move on from a local optimum to cover a larger part of the search space?

How to escape local optima in LS - some strategies

- Restart
- Random choice of move
- Allow moves to lower quality solutions
  - deterministic
  - prababilistic
- Memory
  - which solutions have been visited before?
  - diversify the search
  - (parts of) good solutions
  - intensify the search
- Change the search landscape
  - Change neighborhood
  - Change evaluation function

Metaheuristics (General heuristics)

- search strategies that escape local optima
- introduced early 1980ies
- considerable success in solving hard DOPs
- analogies from physics, biology, human brain, human problem solving
- a large number of variants
- some religions
- some confusion in the literature

Some metaheuristics

- **Simulated Annealing (SA)**
- Threshold Accepting (TA)
- **Genetic Algorithms (GA)**
- Memetic Algorithms (MA)
- Evolutionary Algorithms (EA)
- Differential Evolution (DE)
- Ant Colony Optimization (ACO)
- Particle Swarm Optimization (PSO)
- Immune Systems (IS)
- Tabu Search (TS)
- Scatter Search (SS)
- Path Relinking (PR)
- Guided Local Search (GLS)
- Greedy Randomized Adaptive Search (GRASP)
- Iterated Local Search (ILS)
- Very Large Neighborhood Search (VLNS)
- Variable Neighborhood Descent / Search (VND/VNS)
- Neural Networks (NN)

”Definition” of metaheuristics (Osman & Kelly)

A metaheuristic is
an iterative generation process
that guides an underlying heuristic
by combining
(in an intelligent way)
different strategies for exploring and exploiting solution spaces
(and learning strategies)
to find near-optimal solutions in an effective way

”Definition” of metaheuristics (Glover & Kochenberger)

Solution methods that utilize **interaction** between **local improvement procedures** (local search) and higher level **strategies** to **escape local optima** and ensure **robust search** in a solution space

Variant of LS: Random Search (RS)

”Brownian motion” - A borderline metaheuristic

```
Procedure Random_Search(f,N,Stop,initial)
begin
    current:=incumbent:=initial;
    while not Stop() do
    begin
        current:=Random_Solution(N(current))
        if f(current) < f(incumbent) then
        begin
            incumbent:=current;
        end
    end
    return incumbent;
end
```

Stop criteria?

Variant of RS: Random Descent
A borderline metaheuristic

```
Procedure Random_Descent(f,N,Stop,initial)
begin
    new_solution:=current:=incumbent:=initial
    while not Stop() do
    begin
          Neighbors:=N(current)
          while not Stop() and f(new_solution) >=f(current) do
          begin
                    new_solution:=Random_Solution(Neighbors)
          end
          current:=new_solution
          if f(current) < f(incumbent) then incumbent:=current
    end
    return incumbent
end
```

Metaheuristic strategies

- Goals
  - escape local optima
  - avoid loops
- Accept worse solutions
- Randomness

- Simulated Annealing (SA) utilizes these strategies

Simulated Annealing (SA)
Kirkpatrick et al. 1983 / Cerny 1985

- Inspired by work on statistical thermodynamics
  - the Metropolis algorithm 1953
  - simulation of energy changes in physical systems under cooling
- Used for DOP
- Built on LS (variant of Random Search/Random Descent)
- Trajectory-based method
- Simple to implement
- A lot of literature
- Used a lot, outside the metaheuristic community, probably too much ...
- Converges to a global optimum under weak assumptions
- Very slowly ....

SA - Analogies

Thermodynamics	DOP
System state	Solution
Energy	Cost
State change	Move
Temperature	Control parameter
Final state	Final solution

Simulated Annealing (SA)

May be expressed as strategy for move selection in basic LS:

```
**Procedure** Local_Search(Init_Sol,N,f,Strategy,Stop_Criterion)
*/ Strategy = SA
incumbent:=current:= Init_Sol()
**Repeat**
    current:=Select_SA_Neighbor(f,current,N(current),Stop_Criterion)
    **if** f(current)< f(incumbent) **then** incumbent :=current
**Until** Stop_Criterion**()**
**return** incumbent
```

Move selection in SA

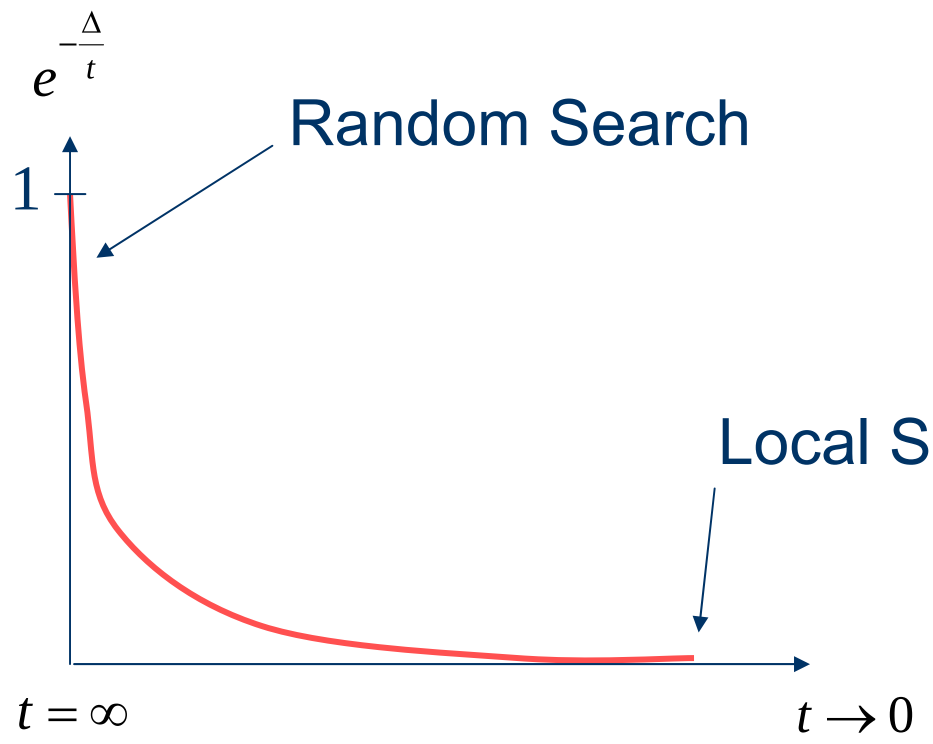
- Modified ”Random Descent”
- Select random solution in neighborhood
- Accept
    - unconditionally, if better
    - with a non-zero probability, if worse
- Probability determined by control parameter (temperature)

- Avoids getting stuck in local optimum
- Avoids looping

Move selection in SA

```
Procedure Select_SA_Neighbor
    (f,current,Neighbors,Stop_Criterion)
*/ Strategy is Simulated Annealing
begin
    i:=Random_Element(Neighbors)
    delta := f(i) - f(current) */ Could be improved ...
    if delta < 0 or Random(0,1) < exp(-delta/t) then
        return i
    else
        return current
end
```

SA
Acceptance of worse solutions



SA – higher level strategy

- initial control variable t_0 (high value)
- ”inner” stop criterion: # iterations with the same temperature
- temperature reduction
- ”cooling schedule” $t_{i+1} = \alpha(t_i)$
- stop criteria
  - minimum temperature
  - # iterations (without improvement)
  - time
  - user
- the procedure may be iterated
- efficiency is depending on parameters (optimization problem)
  - experiments, experience, (witch)craft, ...
  - over-fitting
  - self-adaptive, parameter-free methods ...
- selection of neighborhood is still important ...

SA – overall procedure

```
Procedure Simulated_Annealing
        (f,N,Stop_Criterion,t0,Nrep,Cool)
incumbent:=current:= Find_Initial_Solution()
t:=t0
Repeat
    for i:=1 to Nrep do                    */ Several iterations with one t value
    begin
        current :=Select_SA_Neighbor(f, current,N(sol),incumbent,t)
        if f(current) < f(incumbent) then incumbent:= current
    end
    t:=Cool(t)                             */ Cooling
Until Stop_Criterion()
return incumbent
```

Statistical analysis of SA

- Model: state transitions in search space
- Transition probabilities $[p_{ij}]$, only dependent on states
- Homogeneous Markov chain

When all transition probabilities are finite,
SA will converge to a stationary distribution
that is independent of the initial solution. When the temperature goes to zero, the distribution converges to a uniform distribution over
the global optima.

- Statistical guarantee
- In practice: exponential computing time needed to guarantee optimum

SA i practice

- heuristic approximation algorithm
- performance strongly dependent on cooling schedule
- rules of thumb
  - large # iterations, few temperatures
  - small # iterations, many temperatures

SA in practice – Cooling schedule

- geometric sequence often works well

$$t_{i+1} = at_i, \quad i = 0, \ldots, K \quad a < 1 \quad (0.8 - 0.99)$$

- vary # repetitions and *a*, adaptation
- computational experiments

SA – Cooling schedule

- # repetitions and reduction rate should reflect search landscape
- Tuned to maximum difference between solution values
- Adaptive # repetitions
  - more repetitions for lower temperatures
  - acceptance rate, maximum limit
- Very low temperatures are unnecessary (Local Search)
- Overall cooling rate more important than the specific cooling function

SA – Decisions

- Goal: High quality solution in short time
- Search space: only feasible solutions?
- Neighborhood
- Evaluation function
- Cooling schedule

SA – Computational efficiency aspects

- Random choice of neighbor
  - neighborhood reduction, good candidates
- Evaluation of objective (evaluation function)
  - difference without full calculation
  - approximation
- Evaluation of constraints (evaluation function)
- Move acceptance
  - calculating the exponential function takes time
  - drawing random numbers take time
- Parallel computing
  - fine-grained, in Local Search
  - coarse-grained: multiple SA searches

SA – Modifications and extensions

- Probabilistic
  - Acceptance probability
  - Approximation of (exponential) function / table
  - Approximation of cost function
  - Few temperatures
  - Restart
- Deterministic
  - Threshold Accepting (TA), Dueck and Scheuer 1990
  - Record-to-Record Travel (RTR), The Great Deluge Algorithm (GDA), Dueck 1993

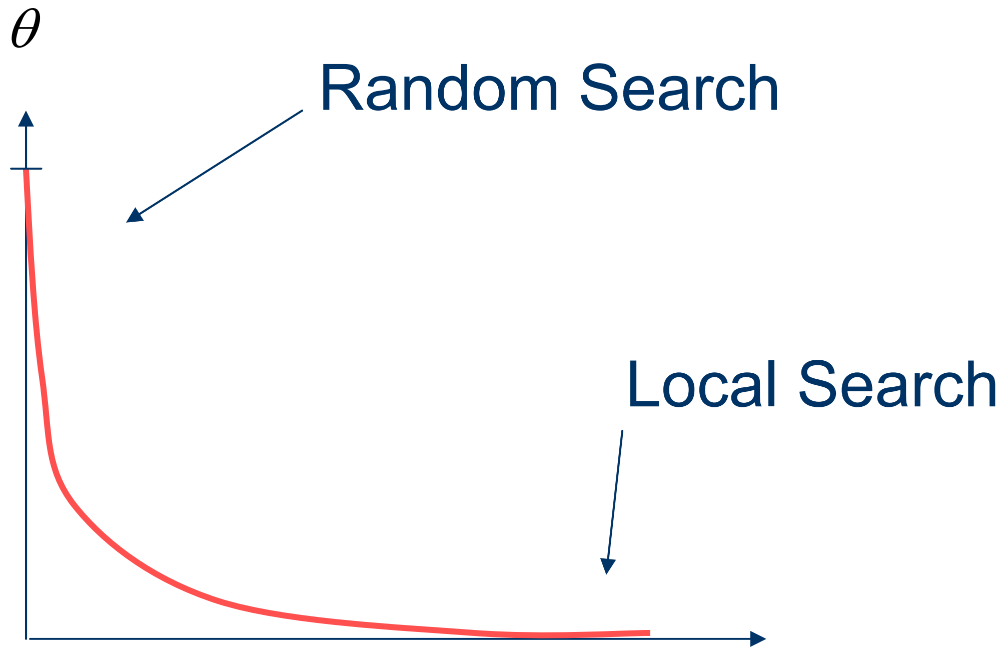
Deterministic Annealing - Threshold Accepting (TA)

```
Procedure Select_TA_Neighbor (f,current,Neighbors,incumbent,theta1)
*/ Strategy is Threshold Accepting
begin
    i:=Random_Element(Neighbors)
    delta := f(i) - f(current)
    */ SA: if delta < 0 or Random(0,1) < exp(-delta/t)
    if delta < theta1         */ Positive Threshold w.r.t. current
    then return i
end
```

Deterministic Annealing
- Record-to-Record Travel (RRT)

```
Procedure Select_RRT_Neighbor
    (f,current,Neighbors,incumbent,theta2)
*/ Strategy is Record-to-Record Travel
begin
    i:=Random_Element(Neighbors)
    */ SA, TA:  delta := f(i) - f(current)
    */ SA: if delta < 0 or Random(0,1) < exp(-delta/t)
     if f(i) < theta2*f(incumbent)                */ theta2 > 1
    then return i
end
```

TA, RTR: Cooling schedule of tolerance



Tabu Search (TS)
F. Glover / P. Hansen 1986

- Fred Glover 1986: ”Future paths for integer programming and links to artificial intelligence”
- Pierre Hansen 1986: ”The Steepest Ascent/Mildest Descent Heuristic for Combinatorial Optimization”
- DOP research – OR and AI
- Barrier methods, search in infeasible space
- Surrogate constraints
- Cutting plane methods
- Automated learning, cognitive science

Tabu (Taboo)

”Banned because of moral, taste, or risk”

Tabu Search: Search guidance towards otherwise inaccessible areas of the search space by use of restrictions

- Principles for intelligent problem solving
- Structures that exploit history (”learning”)

Tabu Search – Main ideas

- Trajectory-based method, based on Local Search
- Seeks to allow local optima by allowing non-improving moves
- Aggressive: Always move to best solution in neighborhood
- Looping problem, particularly for symmetric neighborhoods
- Use of **memory** to
  - avoid loops (short term memory)
  - diversify the search (long term memory)
  - intensify the search (long term memory)
  - General strategy to control ”inner” heuristics (LS, ...)
- *Metaheuristic* (Glover)

Basic Tabu Search

- LS with ”Best Neighbor” strategy
- Always move to new neighbor (”aggressive LS”)
- But: some neighbors are **tabu**
- Tabu status defined by **tabu-criteria**
- However: some tabu moves are **admissible**
  - **admissibility criteria**
  - typical example: new incumbent
- Short term memory: **Tabu List**

Tabu Restrictions

- defined on properties of solutions or moves – **attributes**
- how often – or how recent (frequency, recency) has the attribute been involved in (generating) a solution
- data structure: tabu list

Local_Search (S,f,N,strategy)

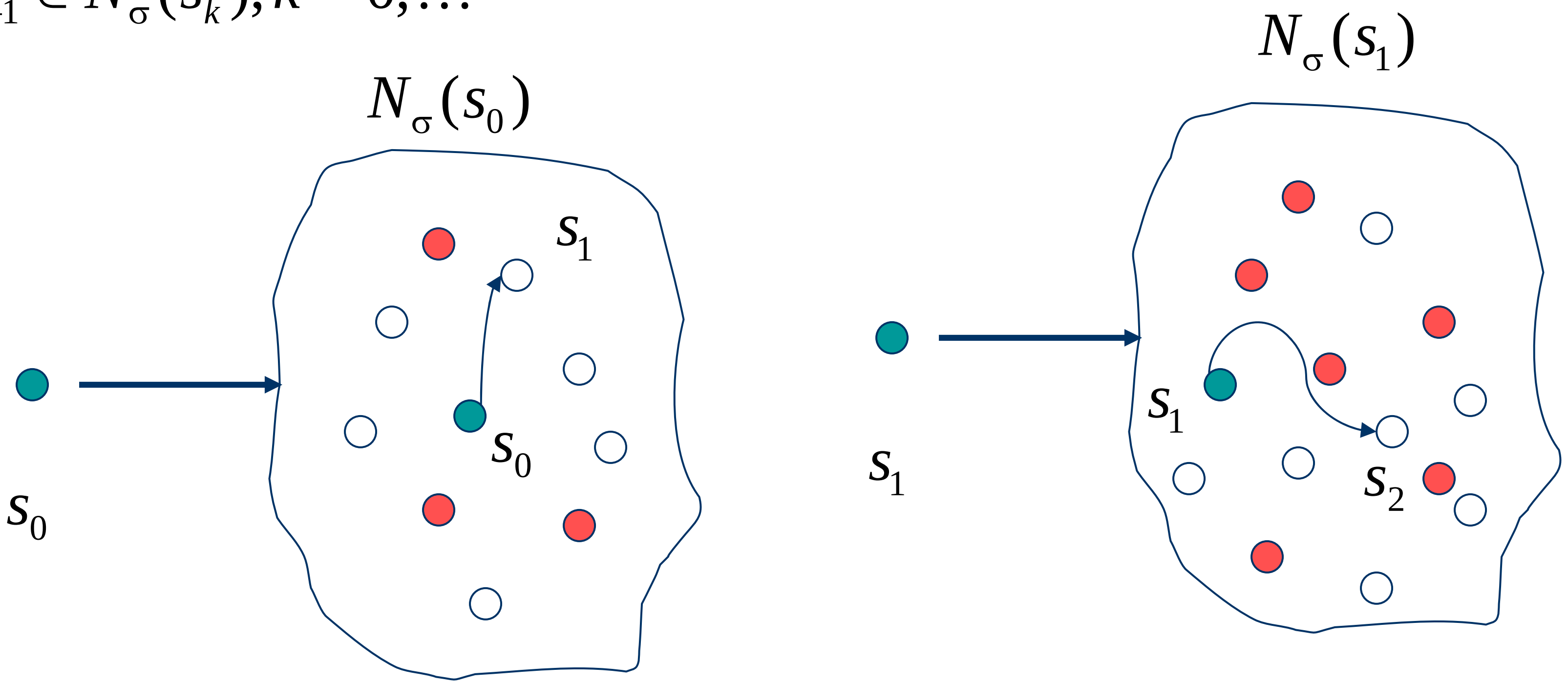
```
incumbent:=current=Init_Solution(S)
local_optimum:=false
while not local_optimum do
    (current,incumbent,local_optimum):=
        Search_the_Neighborhood (current,N,f,strategy,incumbent)
    if local_optimum return incumbent
od
```

Search_the_Neighborhood (current,N,f,strategy,incumbent)

```
best_neighbor:=current
neighbors=N(current)
for i in neighbors do
      if f(i) < f(best_neighbor) then best_neighbor:=i
      if f(i) < f(incumbent) then
            if strategy=”First Accept” then
               return (i,i,false) else
               incumbent:=i                      */ strategy is ”Best Accept”
            fi
        fi
od
return (best_neighbor,incumbent,best_neighbor=current)
*/          (current,incumbent,local_optimum)
```

Traversing the Neighborhood Graph

$$s_{k+1} \in N_\sigma(s_k), k = 0, \ldots$$



A *move* is the process of selecting a given solution in the neighborhood of the *current* solution, hence making it the current solution for the next iteration

Local_Search (S,f,N,'Basic_Tabu_Search')

```
incumbent:=current:=Init_Solution(S)
*/ best solution until now, "champion"
*/ local_optimum:=false
while not Stopping_Criterion() do
        current:=Search_the_Neighborhood
            (current,N,f,Basic_Tabu_Search,incumbent)
        if f(current) < f(incumbent) then incumbent:=current
        */ if local_optimum return incumbent
od
return incumbent
```

Search_the_Neighborhood (current,N,f,strategy,incumbent)

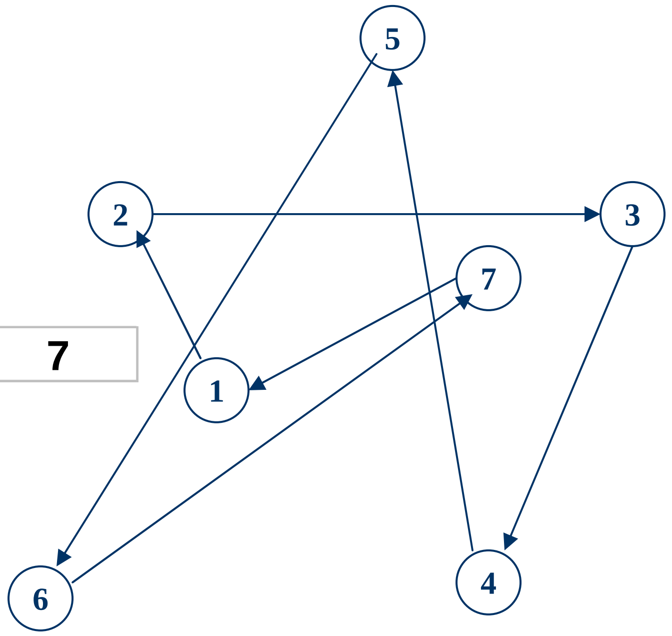
```
*/ Strategy=Basic_Tabu_Search
*/ best_neighbor:=current
best_acceptable_neighbor:=Really_Bad_Solution()
Neighbors=N(current)
for i in Neighbors do
    if f(i) < f(best_acceptable_neighbor)
    and (not Tabu(i,Tabu_List) or Admissible(i)) then
        best_acceptable_neighbor:=i
od
Update_Tabu_List(best_acceptable_neighbor,Tabu_List)
return best_acceptable_neighbor
```

Example: TSP

- Representation: permutation vector
- Operator: pairwise exchange

$$(i, j) \quad i < j \quad i, j \in [1, n]$$

1	**2**	**3**	**4**	**5**	**6**	**7**



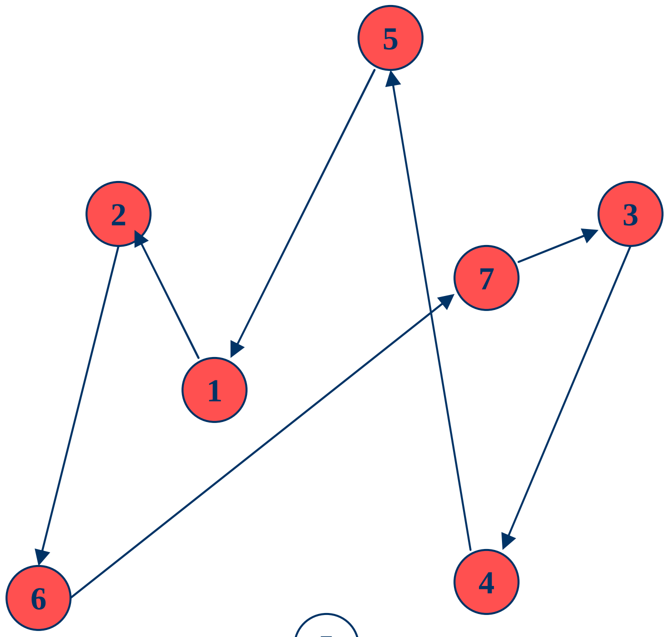
TSP-example

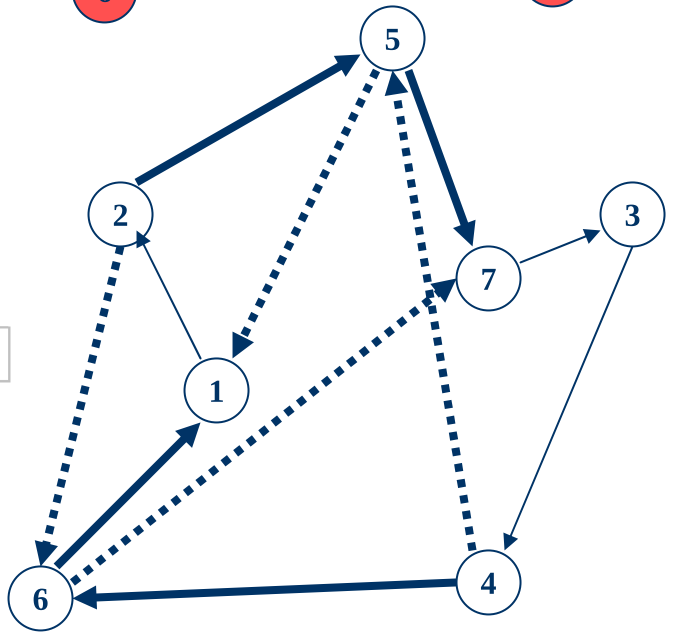
1-exchange in permutation vector

2	6	7	3	4	5	1

Move: *Exchange(5,6)*

2	5	7	3	4	6	1





TSP-example 1-exchange

- Neighborhood cardinality: $\binom{n}{2}$
- For every move: *move value* $\Delta_{k+1} = f(i_{k+1}) - f(i_k),\ i_{k+1} \in N(i_k)$
- Choice of tabu restriction
  - attribute: city involved in **move**
  - tabu to perform moves that involve cities that have recently have been involved
  - for the past *k* iterations
  - *k*=3 (”tabu tenure”)
- Choice of aspiration criterion
  - the classical one ...

Tabu criterion and tabu list

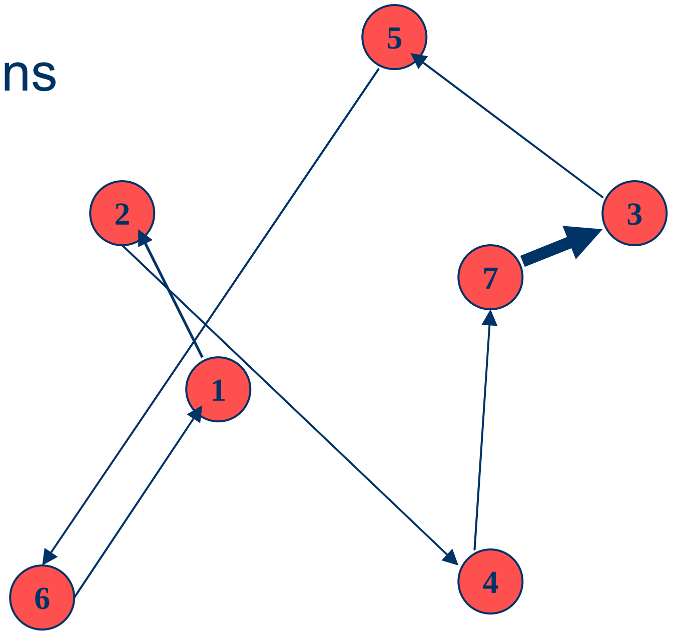
- TSP-example

- Tabu Attribute: the pair of cities involved in a move
- Tabu Criterion: move that involves the same pair
- Tabu tenure: 3
- Data structure: triangular table, # iterations until move becomes legal
- Update after each move

	2	3	4	5	6	7
1	0	2	0	0	0	0
2		0	3	0	0	0
3			0	0	0	0
4				1	0	0
5					0	0
6						0

Alternative tabu criteria / attributes
- TSP-example

- Do not operate on given cities
- Do not operate on cities in certain positions in vector
- Edge/Arc based criteria
  - Edge has often occured in good solutions
  - Edge lengths
  - Edge in/out
- For permutation problems:
  - attributes related to previous/next relations often work well



1	2	3	4	5	6	7
2	4	7	3	5	6	1

Candidate list of moves
- TSP-example

1	2	3	4	5	6	7
2	4	7	3	5	6	1

Current solution

Cost: 200

Move	Value
1,3	-2
2,3	-1
3,6	1
1,7	2
1,6	4

Candidate list

TSP-example 1-exchange

- Iteration 0/1

1	2	3	4	5	6	7
2	5	7	3	4	6	1

Current solution

Cost: 234

Tabu list	2	3	4	5	6	7
1	0	0	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	0	0	0
				5	0	0
					6	0

Move	Value
4,5	-34
4,7	-4
3,6	-2
2,3	0
1,4	4

$\longleftarrow$ Select move *(4,5)*

Candidate list

TSP-example
- Iteration 1 (after Exchange (4,5))

Move	Value
4,5	-34
4,7	-4
3,6	-2
2,3	0
4,1	4

Candidate list with selected move

1	2	3	4	5	6	7
2	4	7	3	5	6	1

New Current

Cost: 200

Tabu list	2	3	4	5	6	7
1	0	0	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	3	0	0
				5	0	0
					6	0

TSP-example

- Iteration 2

Current solution

1	2	3	4	5	6	7
2	4	7	1	5	6	3

Cost: 198

Cost: 200

Move	Value
1,3	-2
2,3	-1
3,6	1
1,7	2
1,6	4

← Select move (1,3)

New candidate list

Tabulist	2	3	4	5	6	7
1	0	3	0	0	0	0
	2	0	0	0	0	0
		3	0	0	0	0
			4	3	0	0
				5	0	0
					6	0

TSP-example

- Iteration 3

1	2	3	4	5	6	7
4	2	7	1	5	6	3

Current

Cost: 202

Move	Value
4,5	2
2,4	4
6,7	6
4,5	7
3,5	9

Tabu!

Select move (2,4) (deteriorating)

Tabu list	2	3	4	5	6	7
1	0	2	0	0	0	0
	2	0	3	0	0	0
		3	0	0	0	0
			4	1	0	0
				5	0	0
					6	0

TSP-example
- Iteration 4

1	2	3	4	5	6	7
4	2	7	1	5	6	3

Current

Cost: 196

Move	Value
4,5	-6
5,3	-2
7,1	0
1,3	3
2,6	6

$\longleftarrow$ Tabu, but Aspiration Criterion says OK. Select (4,5)

Tabu list	2	3	4	5	6	7
1	0	1	0	0	0	0
	2	0	2	0	0	0
		3	0	0	0	0
			4	3	0	0
				5	0	0
					6	0

Observations

- In the example 3 of 21 moves are tabu
- Stronger tabu criteria are achieved by
  - increasing tabu tenure
  - strengthening the tabu restriction
- Dynamic tabu tenure (“Reactive Tabu Search”) often works better than static
- Tabu-list requires space (why not store full solutions instead of attributes?)
- In the example: the tabu criterion is based on **recency, short term memory**
- Long term memory: normally based on **frequency**

TSP-example
- Frequency based long term memory

- Typically utilized to diversify search
- Is often activated when the search stagnates (no improving moves for a long time)
- Typical mechanism for long-term diversification strategies: **Penalty** for moves that have been frequently used
- Augmented move evaluation

Tabu status (recency)

	1	2	3	4	5	6	7	
1			2					
2				3				
3	3							
4	1	5			1			
5		4		4				
6			1		2			
7	4			3				

Frequency of moves

Tabu Search - Main Ideas

- Less use of randomization (than SA)
- “Intelligent” search must be based on more systematic guidance
- Emphasis on flexible memory structures
- Neighborhoods are in effect modified on the basis of short term memory (one excludes solutions that are tabu)
- Memory of good solutions (or parts of them), e.g. good local optima, particularly in long term strategies
- Use of search history to modify evaluation of solutions/moves
- TS may be combined with penalties for constraint violations (a la Lagrange-relaxation)
- Strategic oscillation
  - intensification and diversification
  - feasible space and infeasible space

Tabu Search – More ideas, and practice

- “Aggressive search”: move on – select good neighbor
- Computational effort and scalability remedies (general)
  - delta-evaluation
  - approximation of cost function
  - identify good candidates in neighborhood
  - candidate list of moves, extensions

- Most TS-implementations are deterministic
- Probabilistic Tabu Search
  - moves are chosen probabilistically, but based on TS principles
- Most TS-implementations are simple
  - basic TS with short term memory, static tabu tenure
  - potential for improvement ...

Tabu Search - Generic procedure

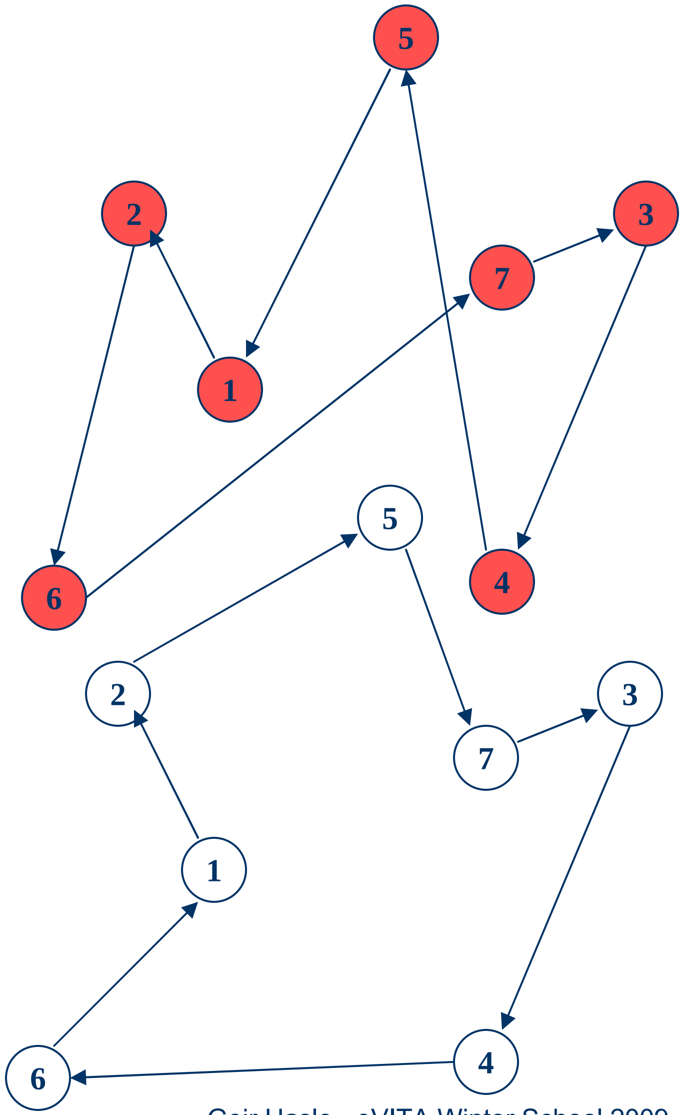
```
Find initial solution x
Initialize memory H
While not Stop_Criterion()
    Find (with limited resources)
    Candidate_List_N(x) in N(x)
    Select (with limited resources)
    x' = argmin(c(H,x), x in Candidate_List_N(x))
    x= x'
    H=Update(H)
end
```

Tabu Search – Attributes

- Attribute: Property of solution or move
- May be based on any aspect of the solution or move
- Basis for definition of tabu restrictions
- A move may change more than one attribute

Example: TSP

- Attribute based on edges in tour
  - *A1*: Edge is added to solution
  - *A2*: Edge is removed from solution
- *Exchange*-move in permutation vector:
  - 4 edges are removed
  - 4 edges are added
  - *Exchange*(5,6)
    - A1:(2,5),(5,7),(4,6),(6,1)
    - A2:(2,6),(6,7),(4,5),(5,1)
- *Exchange* is *O(n*^{2}*)* segment of the *O(n*^{4}*) 4-opt* neighborhood



Use of attributes in tabu restrictions

- Assume that the move $x^{(k)}$-> $x^{(k+1)}$ involves the attribute *A*
- Normal tabu restriction:
  Tabu to perform move that reverses the status of *A*
- TSP-example:
  - the attributes are edges
  - current move introduces edge (2,5): $y_{2,5}$(0->1)
  - moves that remove edge (2,5): $y_{2,5}$(1->0) are tabu (for some iterations)

Tabu tenure – the (witch)craft

- Static
  - t=7
  - t=√n where n is a measure of problem size
- Dynamic (and randomized)
  - t ∈ [5,11]
  - t ∈ [$^{.9}$√n, $^{1.1}$√n]
- Depending on attribute
  - TSP
  - edge-attribute, tabu criterion both on edge in / edge out
  - fewer edges in than out (n vs. n^2-n)
  - same tabu tenure would be unbalanced
- Self-adaptation

Aspiration criteria

- Classical aspiration criterion:
  Accept tabu move that will give a new incumbent
- Other relevant criteria are based on:
  - solution quality
  - degree of feasibility
  - strength of tabu
  - degree of change: *Influence* of move
- High influence move may be important to follow when close to local optimum, search stagnates

- Distance metrics for solutions
  - Hamming-distance between strings
    - *h*(1011101,1001001) = 2
    - *h*(2173896,2233796) = 3
    - h("**toned**”,"**roses**”) = 3
  - More general, sequences: Edit Distance (Levenshtein)
    - insertion, removal, transposition

Frequency based memory

- Complementary to short term memory
- Long term strategies
- Frequency measures
  - *residence*-based
  - *transition*-based

- TSP-example
  - how often has a given edge (triplet, ...) been included in the current solution? (*residence*-based)
  - how often has the in/out status of the edge been changed? (*transition*-based)

Intensification and diversification

- *Intensification:* Aggressive prioritization of good solution attributes
  - short term: based directly on attributes
  - long term: use of **elite solutions**, parts of **elite solutions** (vocabularies)
- *Diversification:* Spreading of search, prioritization of moves that give solutions with new composition of attributes

- Strategic oscillation

Intensification and diversification
- basic mechanisms

- use of frequency based memory
- select solution from subset ***R*** of ***S*** *(or **X**)*
- diversification:
  - ***R*** is chosen to contain a large part of the solutions generated so far (e.g., all local optima)
- intensification :
  - ***R*** is chosen as a small set of elite solutions that
    - to a large degree have identical attributes
    - have a small distance in solution space
    - cluster analysis
    - path relinking

Path relinking

- Assuming that new good solutions are found on the path between two good solutions
- Select two elite solutions *x'* and *x''*
- Find (shortest) path in the solution graph between them
  x' -> x'(1)-> ... x'(r)= x''
- Select one or more of the intermediate solutions as end points for path relinking

Punishment and encouragement
Whip and carrot

- Augmented move evaluation, in addition to objective
- Carrot for intensification is whip for diversification, and vice versa

- Diversification
  - moves to solutions that have attributes with high frequency are penalized
  - TSP-example: $g(x)=f(x)+w_1\Sigma\omega_{ij}$
- Intensification
  - moves to solutions that have attributes that are frequent among elite solutions are encouraged
  - TSP-example: $g(x)=f(x)-w_2\Sigma\gamma_{ij}$

Candidate list

- Strategy to limit computational effort in evaluating neighborhoods
- Limited subset of moves that seem promising
  - approximate move evaluation (evaluation function, constraints)
  - heuristic selection based on attributes (TSP edges)
  - randomness
- Candidate list may be expanded
- Candidate list may be reused

- Parallel processing is another strategy ...

- General idea in Local Search

Tabu Search - Summary

- Inspired from math. progr., AI/cognitive science
- Focus on memory/learning rather than random choice
- A lot of ideas, more a framework than a concrete metaheuristic
- Based on “aggressive” local search, acceptance of worse solutions
- Candidate list strategies for cheaper neighborhood evaluation
- Short term memory
  - avoid reversal of moves and repetition
  - attributes, tabu criteria, tabu list
- Aspiration criteria - tabus are there to be broken ...
- Long term, frequency based memory for diversification and intensification
- Path relinking
- Strategic oscillation
- Complexity?

- Good results for many hard DOPs
- Worshippers, as for most other metaheuristics, communities and congregations

Guided Local Search (GLS) (E. Tsang, C. Voudouris 1995)

- Project for solving Constraint Satisfaction Problems (CSP), early 90ies (E. Tsang, C. Voudouris, A. Davenport)
- GENET (neural network)
- Development of GENET from CSP to ‘Partial CSP’ satisfaction $\Rightarrow$ optimization
- the Tunnelling Algorithm’ (94) -> GLS (95)

GLS - Main ideas

- General strategy for guidance of local search/”inner” heuristics: metaheuristic
- Local search (trajectory based)
- Penalizes undesired properties (‘features’) of solutions
- Focuses on promising parts of the solution space
- Seeks to escape local optima through a dynamic, augmented evaluation function (objective + penalty term)
- Memory
- Changing the search landscape
- LS to local minimum, update of penalty term, LS to local minimum, update of penalty term, ...

Features

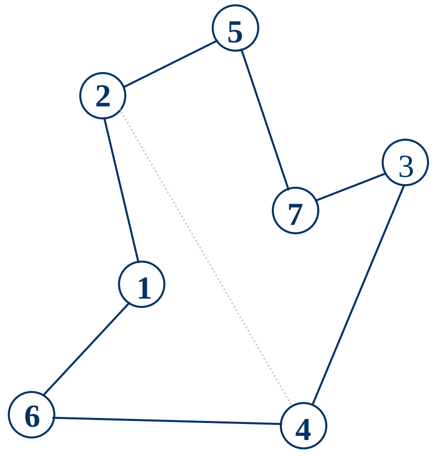
- GLS focuses on characteristic (non-trivial) solution **features**
- Features are problem dependent
- Features have a **cost**
- Costly features will be avoided
- Indicator function

$$I_i(s) = \begin{cases} 1 & \text{, if } s \text{ has feature } i \\ 0 & \text{, otherwise} \end{cases} \quad , s \in \boldsymbol{S}$$

Feature example: TSP

- A solution consists of a number of edges
- **Edge** is a good choice as a basic feature structure
  - Either in or out
  - Cost: edge cost (length)
- Feature set

$$\mathbf{E}=\{e_{ij}\},\ i=1\ldots N,\ j=i+1,\ldots,N,\ i\neq j$$



- Feature costs given by distance matrix

$$\mathbf{C} = [d_{ij}],\ i=1,\ldots,N,\ j=1,\ldots,N$$

Augmented objective

- Feature set E={ e_i }, i=1,…,G
- Indicator function

$$I_i(s) = \begin{cases} 1 & \text{if } s \text{ contains } \mathrm{e}_i \\ 0 & \text{otherwise} \end{cases}, \quad s \in \boldsymbol{S}$$

- Penalty vector **p**=[p_i], i=1…G, # times feature e_i has been penalized
- Penalty factor λ

$$f'(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- Cost vector **c**=[c_i], c_i > 0, i=1…G, cost of feature e_i

Augmented objective - Comments

$$f'(s) = f(s) + \lambda \sum_{i=1}^{G} I_i(s) p_i$$

- λ determines influence of penalty
  - Low value: intensification
  - High value: diversification
- Initially, $p_i=0 \ \forall \ i$
- In local minima, the feature(s) with maximum **utility** are penalized

  $u_i(s_{min}, e_i) = I_{i^*} c_i/(1+p_i)$

- These feature(s) are penalised: $p_i = p_i+1$
  - ⇨ Diversification: different features are penalized
  - ⇨ high cost features are penalized more often

Note: Only local minimum features are penalized

GLS(**S**, **f**, λ, **I**, **c**, G, stopCriterion, **N**)

```
{
    int k := 0;    *number of GLS-iterations*
     $s^* := s_k$ := InitialSolution(S);    *get initial solution*
    set all $p_i := 0$;    *set all penalties*
    while (stopCriterion not satisfied) do {
        $f' := f + \lambda * \sum(I_i * p_i)$;
        $s_{k+1} :=$ *Local_Search* ($s_k$, f', N, “best accept”);  *local minimum*
        for (i:=1 to G)
            $u_i := I_i(s_{k+1}) * c_i /(1+p_i)$;
        for each i such that $u_i$ is maximum do
            $p_i := p_i + 1$;
        k = k+1;
        if ($f(s_{k+1}) < f(s^*)$)
            $s^* := s_{k+1}$;            *new incumbent*
    }
    return s*;
}
```

Comments

- Augmented objective in Local Search
  - s_{k+1} := Local_Search (s_k, f', **N**, “best accept”);
- Local Search strategy may well be ”first accept”, ..
- Variants of *Local_Search* may be used
- Delta-evaluation of moves must take penalties into account
- If all features are penalized equally many times *f'(s)* gives the same landscape as *f(s)*
- Resetting penalty vectors

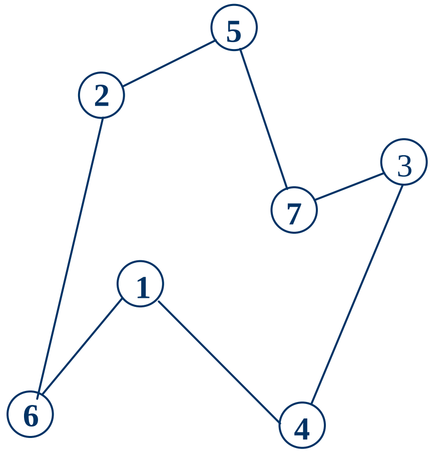
- Avoids bad features, how about encouragement of good features?

λ Values

- GLS seems to be rather robust regarding λ values
- General advice: fraction of value of local minimum
- Tsang & Voudouris:
  - TSP : $\lambda = a_* f(s_{min})/n$, $a \in [0,1]$ problem dependent
  - QAP: $\lambda = a_* f(s_{min})/n^2$, $a \in [0.2,1]$ problem dependent
  - For other problems they report absolute values depending on problem size

GLS - example : TSP

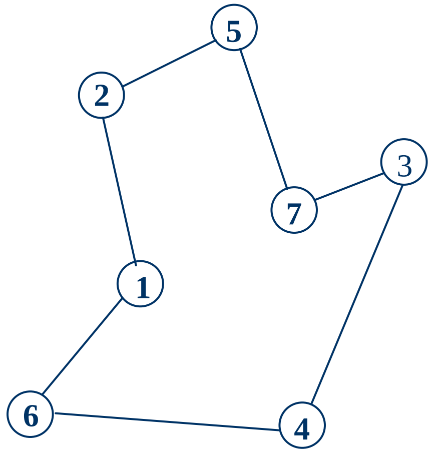
- Feature: edge
- Feature cost: length
- e_{26} will be punished:
- Augmented objective is $f(s)$ if e_{26} is out, $f(s)+\lambda$ if e_{26} is in



	1	2	3	4	5	6	7
1		0	0	0	0	0	0
2			0	0	0	1	0
3				0	0	0	0
4					0	0	0
5						0	0
6							0

GLS - example : TSP

- After next LS, e_{34} will be penalized



	1	2	3	4	5	6	7
1		0	0	0	0	0	0
2			0	0	0	1	0
3				1	0	0	0
4					0	0	0
5						0	0
6							0

$$f'(s) = \begin{cases} f(s) & , e_{26}, e_{34} \notin s \\ f(s)+\lambda & , e_{26} \in s \text{ or } e_{34} \in s \\ f(s)+2\lambda & , e_{26}, e_{34} \in s \end{cases}$$

GLS vs. SA

- SA
  - Local optima avoided by uphill moves
  - Looping avoided by randomness
  - High temperatures give bad solutions
  - Low temperatures: convergence to local minimum
  - The cooling schedule is critical and problem dependent
- GLS visits local minima, but will escape
  - No uphill moves, but changes landscape
  - Deterministic (but probabilistic elements are easily added)

GLS vs. Tabu Search

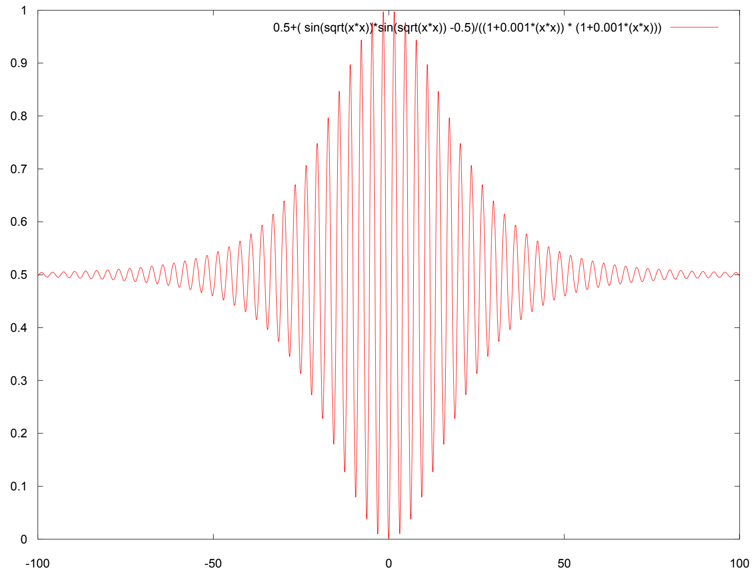
- Similarities ...
- Some arguments, from the GLS community ...
- TS utilizes frequency based (long term) memory used to penalize features that are often present (for diversification)
- GLS utilizes memory (p_i) throughout the search, not in phases
- GLS penalizes on the basis of both cost and frequency
- TS penalizes only on the basis of frequency, may avoid “good” features
- GLS avoids this by utilizing domain knowledge (c_i)
- In GLS the probability for a feature to be penalized is reduced according to the number of times it has been penalized before
- $u_i(s_{min}, e_i) = I_{i^*} c_i/(1+p_i)$

Fast Local Search

- Heuristic limitation of neighborhoods (a la Candidate Lists)
- Idea
  - partition of neighborhood into sub-neighborhoods
  - Status sub-neighborhoods: active or non-active
  - Only search in active sub-neighborhoods
  - Association of properties to sub-neighborhoods
    - property ⇔ neighborhood that change status of this property
- General idea, particularly well suited for GLS

Extensions

- Limited duration of penalties
- Decreasing penalties
- Rewards
- Automatic setting of λ
- Alternative utility-functions that determine which features will be penalized



- GLS for continuous, nonlinear optimization

$$F6(x,y) = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{\left[1+0.001(x^2+y^2)\right]^2}$$

Genetic Algorithms (GA)

- Rechenberg, Schwefel 1960-1970
- Holland et al. 1970ies
- Function optimization
- AI (games, pattern matching, ...)
- OR
- Basic idea
  - intelligent exploration of search spaces based on randomness
  - parallelism
  - analogies from evolutionary biology

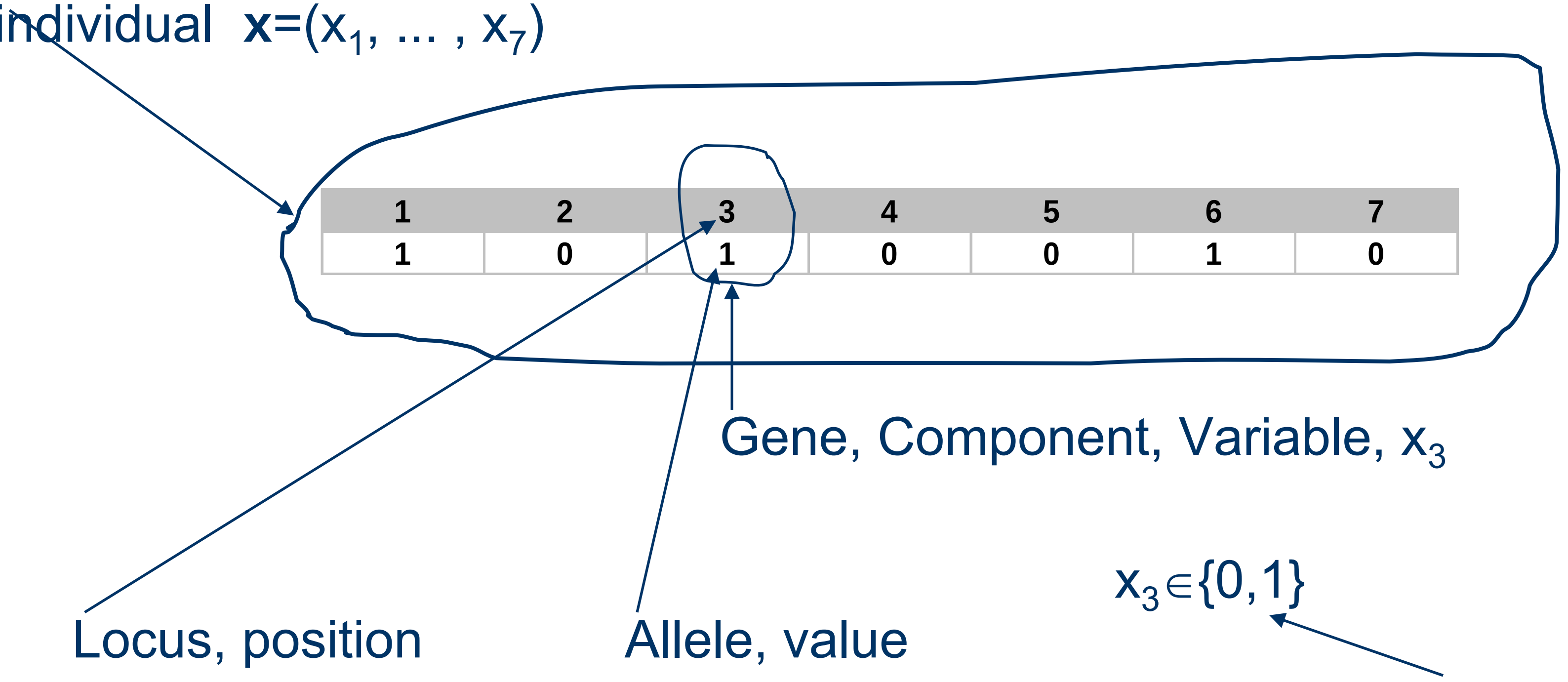
GA – Analogies with biology

- Representation of complex objects by vector of simple elements
- Chromosomes
- Selective breeding
- Darwinistic evolution

- Classical GA for DOP: Binary encoding

Classical GA: Binary chromosomes

Chromosome, (component) vector, string, solution, individual **x**=(x_1, ... , x_7)



1	2	3	4	5	6	7
1	0	1	0	0	1	0

Gene, Component, Variable, x_3

$x_3 \in \{0,1\}$

Locus, position

Allele, value

Alleles, domain

Genotype, Phenotype, Population

- Genotype
  - chromosome
  - collection of chromosomes
  - coded string, collection of coded strings
- Phenotype
  - the physical expression
  - attributes of a (collection of) solutions
- Population – a set of solutions

Genetic operators

- Manipulate chromosomes/solutions
- Mutation: Unary operator
- Crossover: Binary (or n-ary) operator
- Inversion
- ...

Assessment of individuals

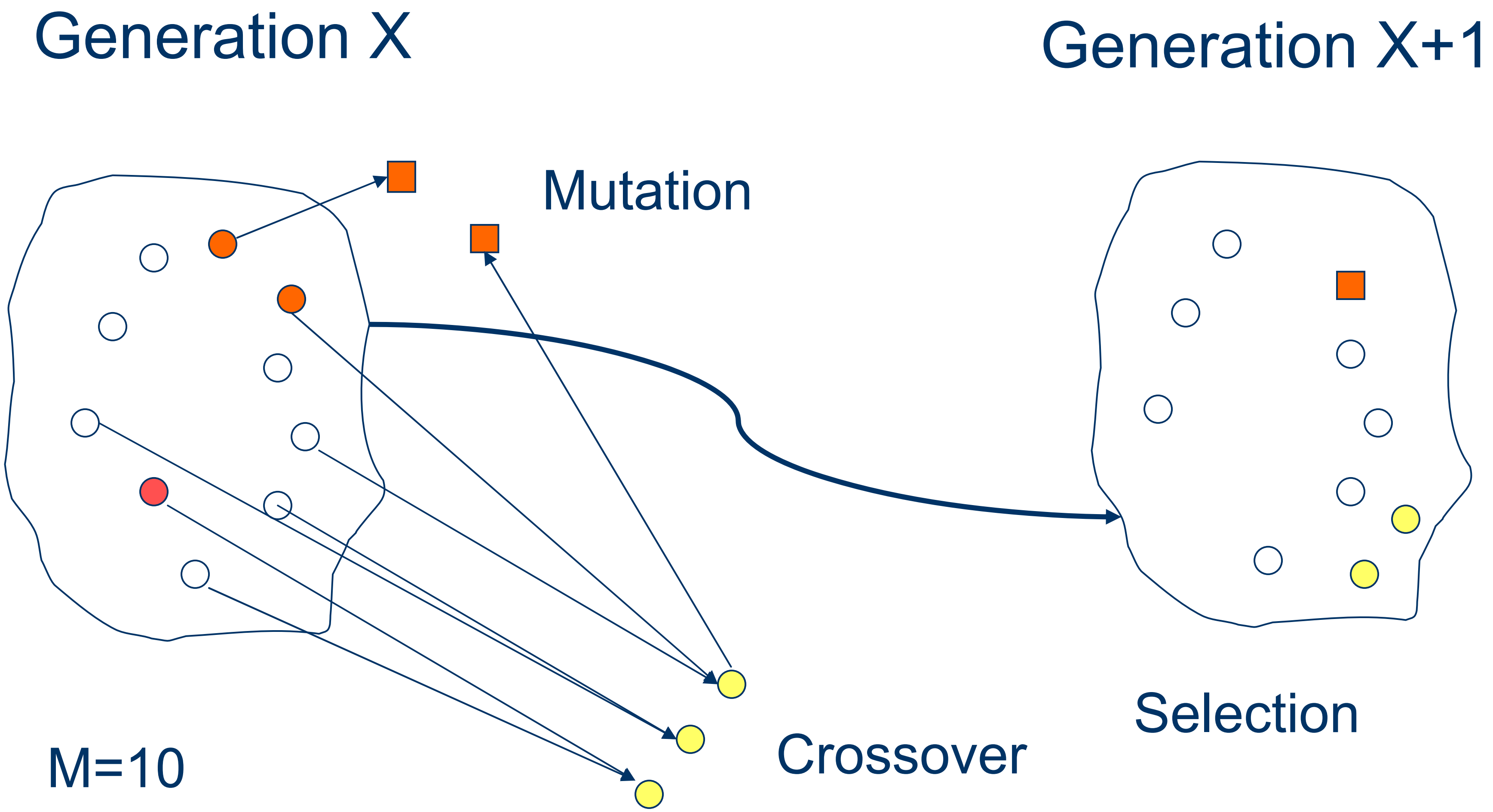
- ”Fitness”
- Related to objective for DOP
- Maximized
- Used in selection (”Survival of the fittest”)
- Fitness is normally normalized

$$f : S \to [0,1]$$

GA - Evolution

- *N* **generations** of populations
- For each generation (step in evolution)
  - selection of individuals for genetic operators
  - formation of new individuals
  - selection of individuals that will survive
- Population size (typically fixed) *M*

GA - Evolution



Classical GA for DOP: Binary chromosomes

1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
0	1	1	1	0	0	1

- Function optimization
  - chromosome reflects binary encoding of real number
- DOP, e.g. TSP
  - binary encoding of solution
  - more direct representation often better
    (e.g. sequential representation)

GA - Mutation

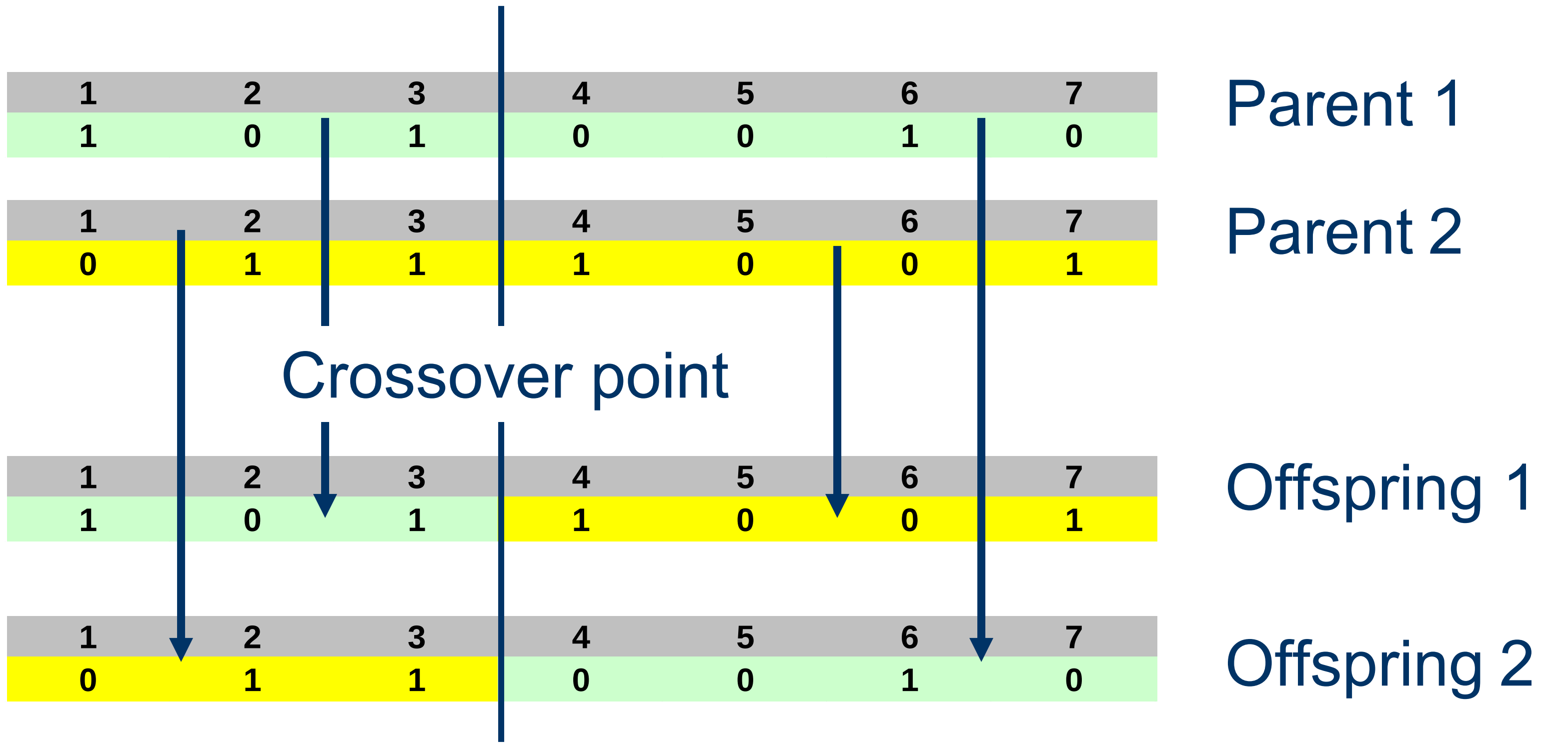
1	2	3	4	5	6	7
1	0	1	0	0	1	0

$$\downarrow$$

1	2	3	4	5	6	7
1	0	1	1	0	1	0

GA - Classical crossover (1-point, 2 individuals)

- One parent selected on the basis of fitness
- The other parent selected randomly
- Random choice of crossover point



GA - Classical crossover cont'd.

- Random individual in population exchanged for one of the offspring
- Reproduction as long as you have energy
- Sequence of almost equal populations
- More elitist alternative
  - the most fit individual selected as one parent
  - crossover until M offspring have been born
  - change the whole population
- Lots of other alternatives ...
- Basic GA with classical crossover and mutation often works well

GA – standard reproduction plan

- Fixed polulation size
- Standard 1-point 2-individual crossover
- Mutation
  - standard: offspring are mutated
  - a certain (small) probability that a certain gene is mutated

Theoretical analysis of GA (Holland)

- Schema/ schemata: subsets of similar chromosomes
- same alleles (values) in certain loci (variables)
- wildcards

1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
0	1	1	1	0	0	1

- A given chromosome belongs to multiple schema
- How many?

1	2	3	4	5	6	7
*	*	1	*	0	*	*

Schema - fitness

- Every time we evaluate fitness to a chromosome, information on average fitness to each of the associated schemata is gathered
- A populasjon may contain M 2^n schemata
- In practice: overlap

GA – Intrinsic parallelism

- A number of schemata are evaluated in parallel
- How many?
- Under reasonable assumptions: $O(M^3)$
- Application of genetic operators will change the fitness of the schemata that are represented in the population
- Schema theorem

Length and order of schema

1	2	3	4	5	6	7
*	*	1	*	0	*	*

- *Length*: Distance between first and last defined position
- *Order*: # defined position
- Example has both length and order 2

Fitness-ratio

Ratio between average fitness of a scheme *e(S)* and average fitness of the population *P*

$$e(S) = \frac{\dfrac{\sum_{s\in S} f(s)}{N(S)}}{\dfrac{\sum_{s\in P} f(s)}{M}}$$

Lemma 1

Under a reproduction plan where a parent is selected by fitness, the expected # instances of a schema *S* in generation *t+1* is

$$E(S,t+1)=e(S,t)N(S,t)$$

where e*(S,t)* is the fitness-ratio for schema *S* and *N(S,t)* is # instances of *S* in generation *t*

Lemma 2

If crossover is used in generation *t* with probability P_c for a schema *S* of length *l(S)*, then the probability *P(S,t+1)* that *S* will be represented in generation *t+1* is bounded by

$$P\left(S,t+1\right) \geq 1 - P_c \frac{l(S)}{n-1}\left(1 - P(S,t)\right)$$

Probability that the other parent does not belong to schema *S*

Probability that *S* is destroyed by crossover

Lemma 3

If mutation is used in generation *t,* with probability P_m that an arbitrary bit is mutated, for a schema *S* with order *k(S),* then the probability that *S* will be represented in generation *t+1* will be bounded by:

$$P(S,t+1) = (1-P_m)^{k(S)} \geq 1 - P_m k(S)$$

Probability that *S* survives mutation

The schema theorem

In GA with standard reproduction plan, where the probability for crossover and mutation is P_c og P_m, respectively, and schema *S* with order *k(S)* and length *l(S)* has a fitness-ratio *e(S,t)* in generation *t*, then the expected # representantives for schema *S* in generation *t+1* is bounded by:

$$E\left(S,t+1\right) \geq \left[1 - P_c \frac{l(S)}{n-1}\left(1 - P(S,t)\right) - P_m k(S)\right] e(S,t) N(S,t)$$

S survives crossover and mutation

Corollary

The representation of schema ***S*** will increase on average if

$$e(S,t) \geq 1 + P_c \frac{l(S)}{n-1} + P_m k(S)$$

Short, low order schemata will increase their representation, assuming that their fitness-ratio is somewhat larger than 1, while longer, high-order schemata have to work harder to survive.

GA – Building block hypothesis

- Goldberg (1989)
- Short, low order schemata are combined to increasingly better solutions

Later development of theory

- Schema theorem for
  - uniform choice of parents in crossover
  - selection of both parents based on fitness
- Exact expressions in the schema theorem
- Analysis by Walsh-functions (from signal analysis)
- Generalization of schema – form
  - design of desired operators

- Fitness landscape analysis

GA – Observations, Extensions, and Modifications

- many knobs to turn
- a lot of literature, chaotic
- somewhat unclear terminology
- modifications
  - population
  - encoding
  - operators
  - hybridization
  - parallelization

GA – Evaluation of performance

- 3 important measures
  - based on objective *f(x)*
  - index *t* (”time”) indicates when solution has been generated
- Best solution so far $f^*(x_t)$
- On-line $$f^{online}(T) = \frac{\sum_{t=1}^{T} f(x_t)}{T}$$
- Off-line $$f^{offline}(T) = \frac{\sum_{t=1}^{T} f^*(x_t)}{T}$$

GA – Population size

- Small populations – low coverage
- Large populations – computationally demanding
- Optimal size increases exponentially with string length in binary encodings
- Rule of thumb: 30
- between *n* and *2n* (Alander)

GA – Initial population

- normal: random solutions
- alternative: ”seeds”: high quality solutions
  - quicker convergence
  - premature convergence

GA – Selection mechanisms

- Generation gap/overlap vs total exchange of population
- Incremental selection works well in many applications

Alternative setup

- A part of the population is selected for reproduction
- Offspring replace randomly selected individuals
- May work better

- Duplication of solution should be avoided
- The incumbent should survive
- Elitisme/Selective death

GA – Fitness

- Objective function untouched rarely the perfect choice
- Naive measure tends to give convergence to similar solutions, premature convergence
- Scaling
  - limited competition in early generations
  - increasing competition over time

$$f(x) = \alpha g(x) + \beta$$

GA – Fitness/Selection

- Use ranks rather than objective
- Tournament selection
  - random selection of groups
  - the most fit in each group are selected

GA – Operators

- Mutation – keeps diversity
  - mutation rate normally not so critical
- Crossover – essential
  - effective, particularly in the early phase
  - selective choice of crossover point
- Multi-point crossover

GA – Generalized crossover

- Bit-string determines where genes should be fetched

1	2	3	4	5	6	7	
1	0	1	0	0	1	0	P1

1	2	3	4	5	6	7
1	1	1	0	0	1	0

1	2	3	4	5	6	7	
0	1	1	1	0	0	1	P0

1	2	3	4	5	6	7	
1	0	1	1	0	1	1	Offspring

GA – Inversion

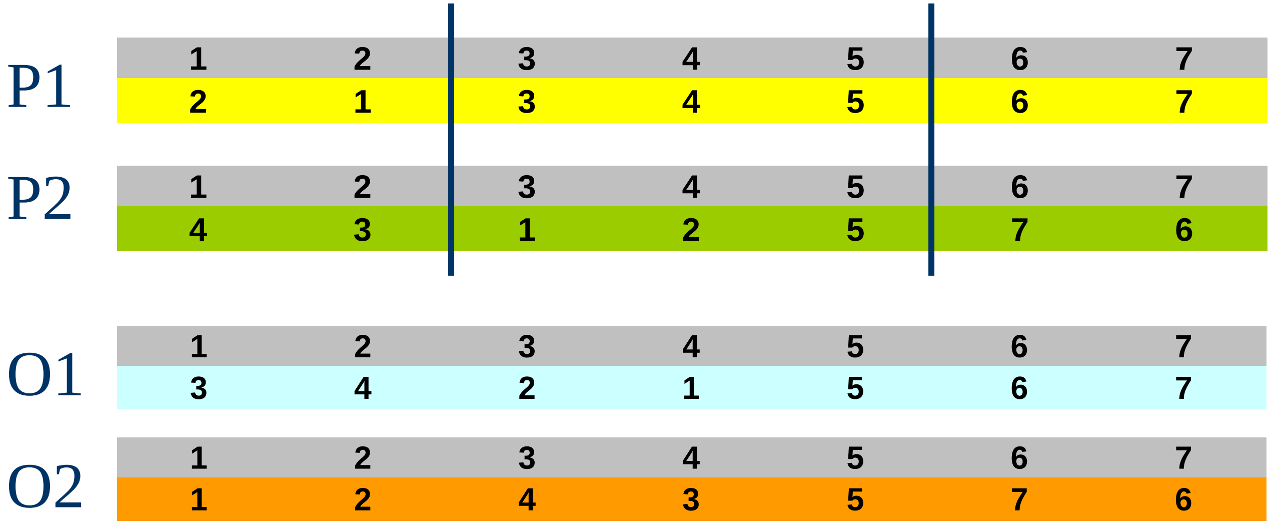
1	2	3	4	5	6	7
0	1	1	1	0	0	1

$$\downarrow$$

1	2	3	4	5	6	7
1	0	0	0	1	1	0

GA – Encoding and representation

- Non-binary encoding
- Sequence representation
- PMX (Partially Mapped Crossover)
  - 2-point, identifies segment for definition of permutation
  - parents define exchanges
  - permutation of each parent gives two children



P1	1	2	3	4	5	6	7
	2	1	3	4	5	6	7
P2	1	2	3	4	5	6	7
	4	3	1	2	5	7	6
O1	1	2	3	4	5	6	7
	3	4	2	1	5	6	7
O2	1	2	3	4	5	6	7
	1	2	4	3	5	7	6

$3 \leftrightarrow 1$

$4 \leftrightarrow 2$

$5 \leftrightarrow 5$

GA - Hybridization

- Strength and weakness of GA: domain independence
- Constraints may be a challenge - penalties
- Hybridization
  - Seeding, good individuals in initial population
  - ”Local search” on individuals
  - Combination with other metaheuristics

GA - parallelization

- Fitness-evaluation
- Fine grained parallelization
  - every solution its own process
  - (a)synchronous parallelization
- Coarse grained parallelization
  - sub-populations
  - the island model

GA - Summary

- Inspired by biological evolution
- Population of solutions that evolves
- Genetic operators
- Randomness
- Domain independence – encoding
- Lacks utilization of problem structure
- Intrinsic parallelism – schema, vocabularies
- Robust, but danger of premature convergence
- Good diversification, lacks somewhat in intensification
- Hybrid with local search: Memetic algorithms

Memetic - meme

- Introduced by Richard Dawkins (1976): ”The Selfish Gene”
- Analogous to gene, in cultural evolution

”Examples of **memes** are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or building arches.”

Trends
GA –> Evolutionary Algorithms

- More direct representation of solution
  - more natural encoding of solutions
  - specialized crossover- and mutation operators
- More intensification
  - local search (”special mutation operators”) to local optimum
  - hybridization of e.g. SA or TA to get deeper
  - Memetic algorithms
- Parallel computing

- 1-population, no crossover EA ...

Ant Colony Optimization – ACO

(Marco Dorigo 1992, Marc Reimann)

- Population based method
- Inspirered by the"collective learning" and communication of ants through **pheromones,** i.e. chemicals released by an organism into its environment enabling it to communicate with other members of its own species
- Multiple "agents" (ants) construct solutions by:
  - construction heuristics
  - random choice
  - information from other agents

Ants (cont´d.)

- Social insects
- Self organizing collective behavior
- Complex dynamics emerges from simple individuals
  - „emergent behaviour“
  - „swarm intelligence“

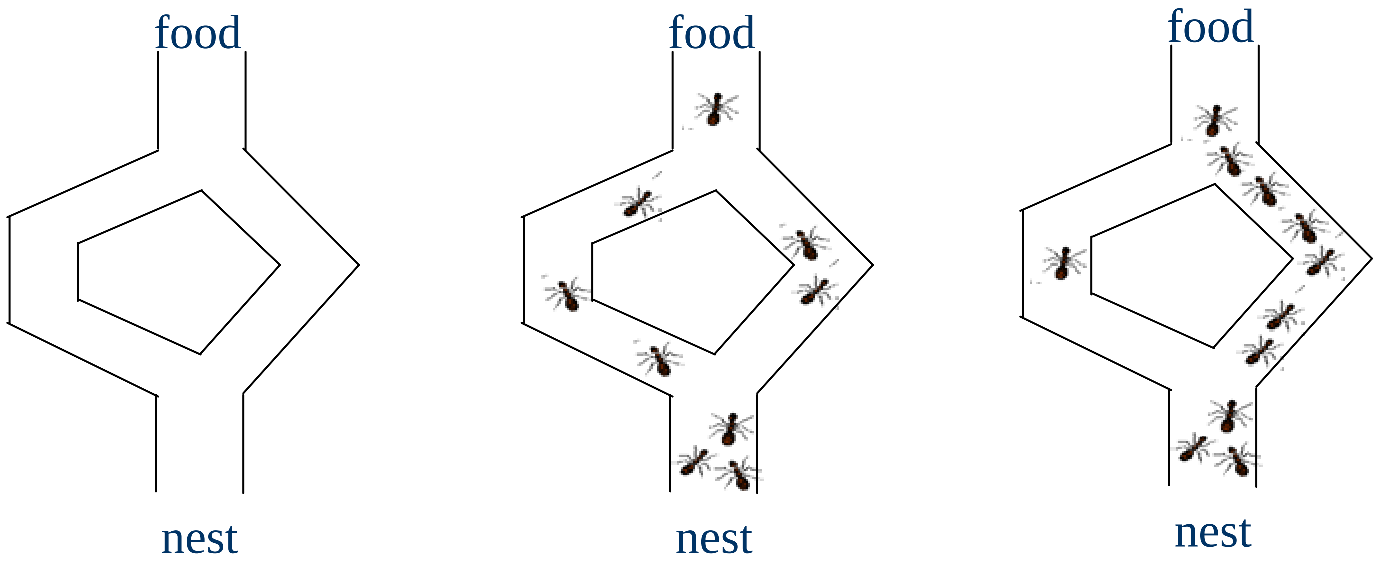
- Interesting social phenomena (Work sharing, Allocation of tasks, Organization of „graveyards“)
- Transport management, finding food (e.g. Lasius niger)

Ants and paths

- Formation and following of paths by simple, local rules
- Every ant legger lays a pheromone trail from nest to food source and back
- Pheromone evaporates
- Ants follow pheromone trails of other ants, according to intensity
- Strong paths get stronger, weak paths get weaker
- Positive feedback, reinforcement learning

Ants – Path creation and path following

- „Binary bridge“-experiment
- Transportation optimization

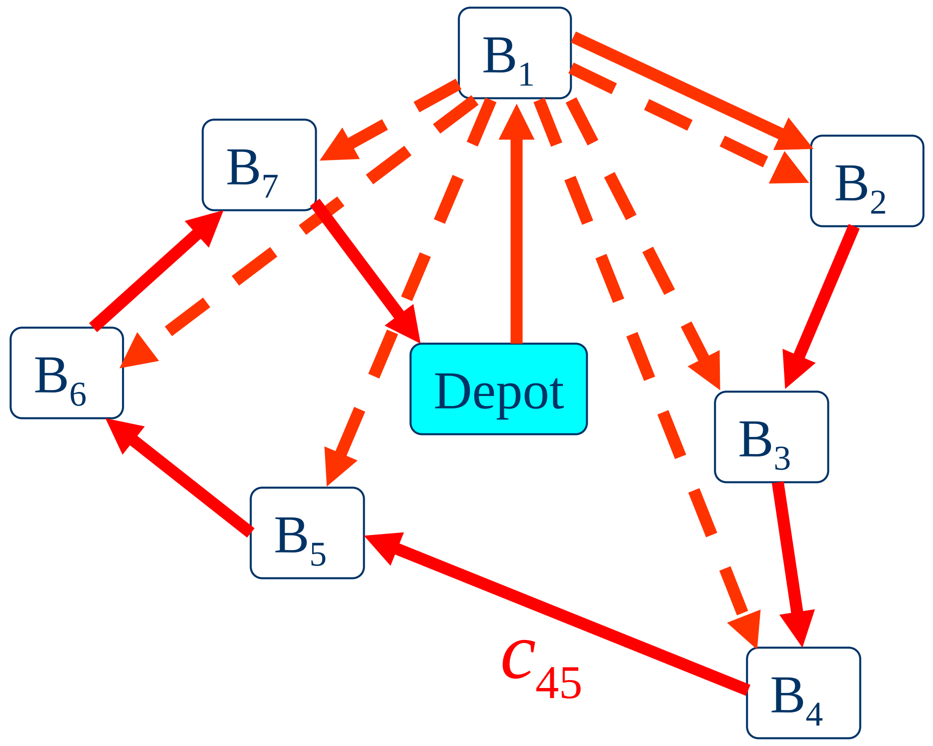


Ant Colony Optimization (ACO) for DOP

- Introduced by Dorigo, Maniezzo & Colorni 1992
- Population based metaheuristic
- Every „ant“ in the population **constructs** a solution
- When they all are finished, a **memory** (artificial pheromone) is updated
- Construction and memory update is repeated until stop criterion is satisfied

Ant Colony Optimization (ACO) for DOP
- Ant System (AS)

- First ACO-method (Dorigo 1992)
- Construction (for every ant)
- Greedy heuristic
- Probabilistic decisions
- e.g. TSP (first problem)
- Construction mechanism
  - „Nearest neighbor“
  - randomness



Ant System - Construction

Local decisions under construction are based on:

- a constructive heuristic (greedy) rule
- a local quality criterion *σ (a priori heuristic information)*
- an adaptive memory (a dynamic, global quality criterion τ)
- randomness

$$p_{ij} = \begin{cases} \dfrac{[\sigma_{ij}]^{\beta} \cdot [\tau_{ij}]^{\alpha}}{\sum_{h \in \Omega_i} [\sigma_{ih}]^{\beta} \cdot [\tau_{ih}]^{\alpha}} & \text{if} \quad h \in \Omega_i \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma_{ij} = \frac{1}{c_{ij}} \quad \mathit{for} \quad \mathit{TSP}$$

where Ω_i is the set of feasible alternatives

Ant System

Update of dynamic informasjon (pheromone)

TSP-example: All edges are updated for each iteration, for each ant

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \sum_{m\in M} \Delta\tau_{ij}^{(m)}, \quad 0 < \rho \le 1$$

$(1-\rho)\cdot\tau_{ij}$ is pheromone on edge (i,j) after evaporation

$\sum_{m\in M} \Delta\tau_{ij}^{m}$ is the added pheromone on edge (i,j)

$$\Delta\tau_{ij}^{m} = \begin{cases} \dfrac{1}{f(s^m)}, & \text{if} \quad (i,j)\in s^m \\ 0 & otherwise \end{cases}$$

Ant System - developments

- Performance of AS not impressive ...
- Biological analogy partly abandoned
- 3 main developments
  - Ant Colony System (ACS)
  - Max-Min Ant System
  - Rank Based Ant System
- All include local search for intensification
- Convergence proofs

- ACS
  - Modified global and local update
  - Global and local pheromone
  - Elitism: only „the best ant“ gets to update **global** pheromone
  - Different randomization, two-step probabilistic choice
    - greedy
    - probabilistic among feasible alternatives

Constructive heuristics

- Greedy construction
  - degree of locality in the decisions
  - informed choice vs. computing time
- Sequence based construction
  - Criticality
  - Static vs. dynamic evaluation of criticality
- Parallel vs. sequential construction
- Seeds
- Combination with search
  - local search
  - systematic

- ”Squeaky Wheel” optimization

Simple remedy to continue from local optima

- Restart from a new solution
- Multi-start methods

Simplest multi-start method

- Random Restart, RR
- Local search (e.g. with Steepest Descent) to local optimum
- Choose random start solution
- Iterate

- Simple metaheuristic
- Blindfolded helicopter skiing
- Embarassingly parallel

- Distance metric may be useful

Alternative pseudo code

LS with ”Steepest Descent”

```
Procedure Local_Search_SD(init_sol,N,f)
current:=init_sol
new_current:=Best_Neighbor(current,N,f)
*/ Assuming Best_Neighbor returns current
*/ if there is no improving move
while not f(new_current)=f(current) do
    current:=new_current
    new_current:= Best_Neighbor(current,N,f)
od
return current                    ; Local optimum
```

Random Restart – RR

```
Procedure Random_Restart (S,N,f,Stop_Criterion)
current:=Init_Solution(S)
incumbent:=current                              */ best solution until now
while not Stop_Criterion() do
        local_optimum:=Local_Search_SD(current,N,f)
        if f(local_optimum) < f(incumbent) then
                incumbent:= local_optimum
        fi
        current:=Random_Init_Solution(S)
od
return incumbent */ best solution until now
```

Greedy Randomized Adaptive Search (GRASP)

- Variant of Random Restart
- Construction with random choice
- Somewhat similar to Ant Colony Optimization, but trajectory based
- Limited candidate list for extension of partial solution

GRASP

```
Procedure GRASP (Max_Iterations)
incumbent:=Bad_Solution()
for k:=1 to Max_Iterations do
          current:=Local_Search(.....
                              Greedy_Randomized_Construction(...))
          if f(current) < f(incumbent) then
                    incumbent:= current
          fi
od
return incumbent */ best solution until now
```

GRASP – Randomized Construction

```
Procedure Greedy_Randomized_Construction(...)
partial_solution := Empty_Solution()
while not Complete(solution) do
    Restricted_Candidate_List
                    :=Evaluate_Incremental_Costs(solution)
    partial_solution:=Extend_Solution(partial_solution,Random_Element(Restricted_Candidate_List))
od
return solution
```

GRASP – Restricted Candidate List

- Restrictions may be
  - Rank based (”the 10 best alternatives”)
  - Value based (”all elements with incremental cost no greater than threshhold”)

$$c(e) \in \left[c^{\min}, c^{\min} + \alpha\left(c^{\max} - c^{\min}\right)\right], \quad \alpha \in [0,1]$$

- Reactive (self-adaptive) GRASP: automatic adjustment
- Perturbation of cost function (noising)
- Extension with Path Relinking
- Preprosessor to GA

”Squeaky Wheel Optimization” D. Joslin and D. Clements 1999

- In a squeaking machine you first lubricate the squeaking parts
- Based on constructive heuristic where
  - Solution is built by successive augmentation of partial solution with new elements with a (greedy) heuristic
  - The **sequence of unserviced elements** is important (priority of elements)
  - There is a measure of how ”pleased” an element is in the final solution
- Change the priority sequence of elements
- Repetition

Squeaky Wheel Optimization

```
Procedure Squeaky_Wheel_Optimization(f)
Element_Priority_List
      :=Determine_Element_Priorities() ; Static prioritization
incumbent:=Some_Feasible_Solution()
while not Stop_Criterion() do
      solution:= Empty_Solution()
      while not Complete(solution) do
            solution:=Augment_Solution(solution,
                                            Element_Priority_List)
      od
      if f(solution)< f(incumbent) then incumbent:=solution
       Element_Priority_List
            :=Update_Element_Priorities(Element_Priority_List,solution)
od
return solution
```

Variable Neighborhood Search (VNS) P. Hansen and N. Mladenovic 1999

- Local optimum is relative to neighborhood
- A local optimum w.r.t. one neighborhood is not necessarily a local optimum w.r.t. another
- A globalt optimum is a et lokal optimum for all neighborhoods
- Local optima may be close to one another
- Basic idea in VNS:
  - Systematic variation of neighborhoods
- Structure of neighborhoods, often ordered by size

$$N_{\mathrm{k}}, \quad \mathrm{k} = 1, \ldots \mathrm{k}_{\max}$$

Variable Neighborhood Descent (VND)

```
Procedure VND (N[1..kmax])
incumbent:=Initial_Solution() ; best solution until now
while not Stop() do
    restart: for k:=1 to kmax do
        local_optimum:=Some_Local_Search(N(k),incumbent)
        ; Variants: First Improve, Best Neighbor, SA, TS ...
        if f(local_optimum)< f(incumbent) then
            incumbent:=local_optimum
            if not Stop() goto restart fi
        fi
    od
od
return incumbent
```

Variable Neighborhood Search (VNS)

```
Procedure VNS (N[1..kmax])
incumbent:=Initial_Solution()
while not Stop() do
    restart: for k:=1 to kmax do
        current:=Random_Element(N(k),incumbent)
        local_optimum:=Some_Local_Search(N(k),current)
        ; Variants: First Improve, Best Neighbor, SA, TS, VND ...
        if f(local_optimum)< f(incumbent) then
            incumbent:=local_optimum
            if not Stop() goto restart
        fi
    od
od
return incumbent
```

VNS

- ”Local search” in VNS may be VND
- VNS is computationally demanding for large instances
- Reduced local search: VNDS
- Hybridization
  - Tabu search
  - GRASP
  - ...

Iterated Local Search (Lourenço et al.)

- Based on (variant of) local search
- Lokal search gives a mapping

$$\mathrm{LS}: \boldsymbol{S} \rightarrow \hat{\boldsymbol{S}}$$

- Different initial solutions gives different local optima (not really, the mapping is surjective)
- Random restart will eventually find a global optimum ...
- To speed up things, it would be nice to use recursion

$$\mathrm{LS}: \hat{\boldsymbol{S}} \rightarrow \hat{\hat{\boldsymbol{S}}}$$

Iterated Local Search

- Try to iterate in such a way that we do not enter the same ”Basin of attraction”
- Perturbation of the current solution
- Diversification

Iterated Local Search

```
Procedure Iterated_Local_Search(N,f)
incumbent:=current:=Local_Search(Initial_Solution(),N,f)
while not Stop_Criterion() do
    new_start:=Perturbation(current,history)
    new_local_optimum:= Local_Search(new_start,N,f)
    if f(new_local_optimum)<f(incumbent) then
            incumbent:=new_local_optimum
    fi
    if Accept(new_local_optimum,incumbent,history) then
            current:= new_local_optimum
    fi
od
return incumbent
```

Iterated Local Search - Perturbation

- Move in higher order neighborhood
- Ruin and recreate
- Random perturbation
- Focused perturbation
- ”Noising” – change problem data, find local optimum
- Distance measures to cut off search

Iterated Local Search - Perturbation

- How big?
- Too small: same ”Basin of attraction”
- Too big: ”Random restart”
- Varying size

(Very) Large Neighborhood Search (P. Shaw 1999)

- Local Search based methods are often too local
- Good diversification is essential
- Large Neigborhood Search

Ruin and Recreate (Schrimpf et al. 2000)

- Find a good, complete solution
- Take away a substantial number of commitments (5%-40%)
  - Alternative “ruiners”
  - Randomly
  - “Similar” commitments
- Reconstruct
  - Alternative recreators
  - Cheapest insertion
  - Regret-based insertion
- Accept new solution if
  - better
  - Threshold Acceptance
  - Simulated Annealing
- Iterate, until no progress
- Learning, self-adaptation

Categorization of metaheuristics

- Trajectory vs. population
- Stochastic vs. deterministic
- Memoryless vs. ”memorable”
- Restart vs. one shot
- Penalty-based

”GUT” of Metaheuristics?

- Which mechanisms work well?
- local search
- restart
- randomness
- uphill moves
- memory
- penalty
- diversification
- ruin and recreate

- populasjon
- ....

”No Free Lunch”-theorem (Wolpert & MacReady, 1995)

Informal description:

When we average over all instances of a given problem, all search algorithms have the same average performance.

For any search algorithm, you have to pay for any performance superiority for a given set of instances with a performance inferiority in others.

”No Free Lunch”-theorem (Wolpert & MacReady, 1995)

- When we average the performance of a search algorithm over all possible search landscapes, it has no better performance than random search.
- To achieve high performance, a search algorithm needs to utilize domain knowledge

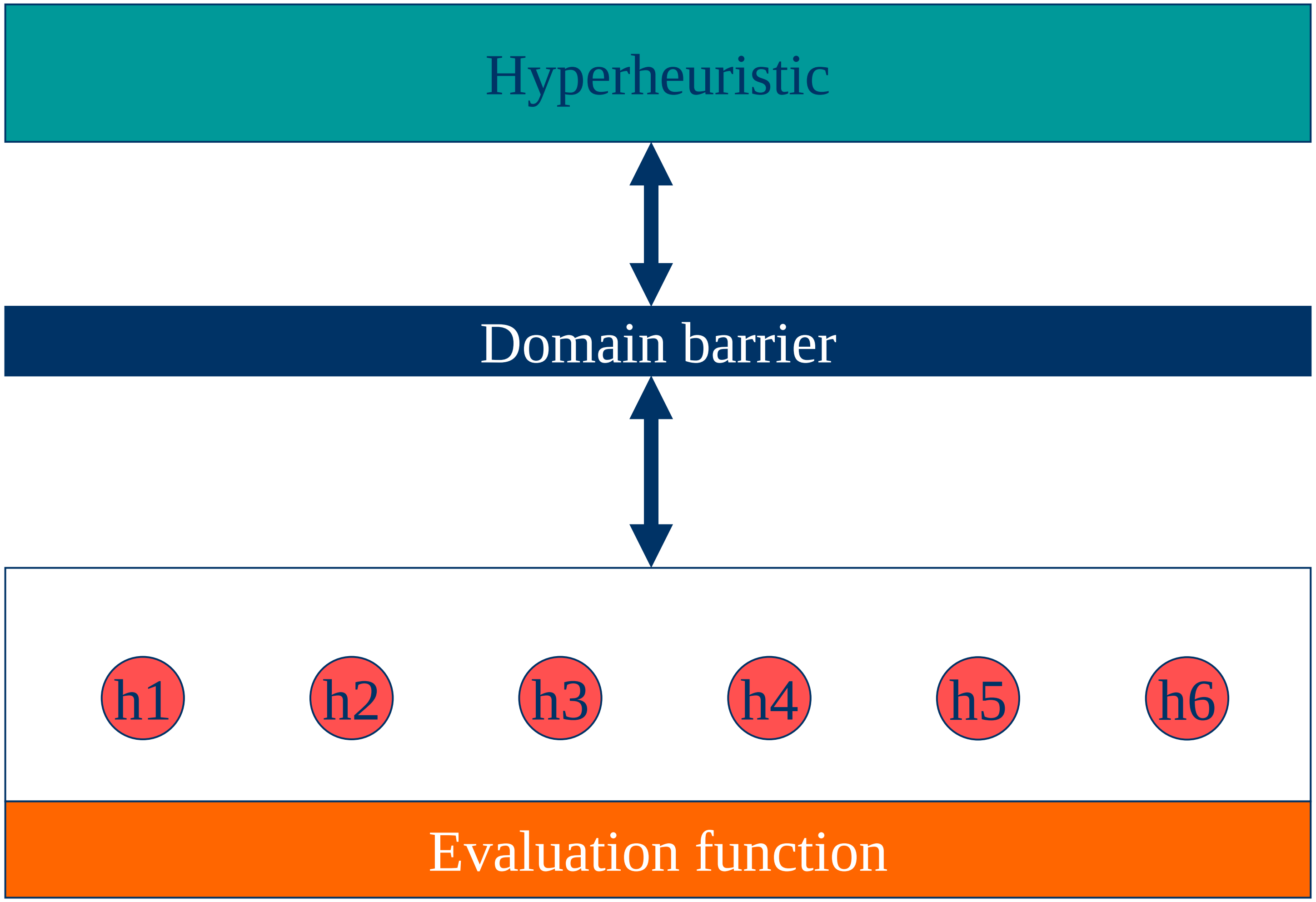
GUT for metaheuristics

- ”Beauty contest” of different search algorithms
- Working mechanisms
- Analysis, learning, opportunistic reasoning
  - investigation of search space
  - analyse of problem solving history / status
  - choice of appropriate technique

Hyperheuristics

- Metaheuristics are not generic
- There is no globally best metaheuristic
- Hyperheuristics
  - General search techniques
  - Based on (meta)heuristics
  - Use of (meta)heuristics to select (meta)heuristic during search
  - no use of domain knowledge

Hyperheuristics



Hyperheuristics

- General information to hyperheuristic from each (meta)heuristic
- CPU time
- Merit, improvement of objective over time
- How long since last active

Topics for future research

- Hybrid methods
- Combination of exact and heuristic methods
- Parallel algorithms
- Collaborating solvers
- Self-adaptive methods, hyperheuristics

- Theoretical understanding of search algorithms

Summary (slide 1)

- Discrete optimization problems are important
- Discrete optimization problems are often computationally hard
- Exact methods may take too long, will give guarantees
- Better to find a good solution to the real problem than the optimal problem to an overly idealized problem
- Local Search is a robust, simple and fast method
- Local Search gives few and weak guarantees
- Local Search is local, gets trapped in a local optimum

Summary (slide 2)

- Metaheuristics move on from local optima and explore larger parts of the solution space
- Metaheuristics are often based on local search
- Different strategies
  - stochastic search
  - allow uphill moves
  - memory structures
  - penalties, changing the landscape
  - combining vocabularies of good solutions
  - vary between neighborhoods
  - restart
  - ruin and recreate
- There is no free lunch
- This area is a lot of fun, many challenges
- Short road from theoretical to practical improvements

Discrete Optimization - Heuristics

Geir Hasle

SINTEF ICT, Applied Mathematics

University of Jyväskylä, Finland