Evolutionary Computation

2025/26

Master Artificial Intelligence

Arno Formella

Departamento de Informática
Escola Superior de Enxeñaría Informática
Universidade de Vigo

25/26

Local search methods

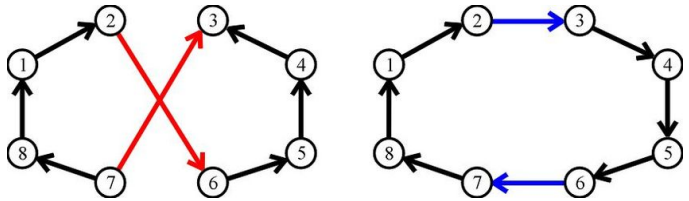
Further classic iterative optimization methods for minimization of real-valued functions that don't need gradient information, are:

- Nelder-Mead method (heuristic), converges to a stationary point (minimum, maximum, or saddle, i.e., gradient is zero)
- idea: shrink, reflect, and expand a simplex (triangle in 2D), by evaluating the objective function on corners and faces (edges in 2D)
- García-Palomares method, converges to a local minimum
- idea: explore the neighorhood according a random local spanning coordinate system and proceed at a point that has been found with a sufficiently steep descent (otherwise iterate with smaller tolerance)

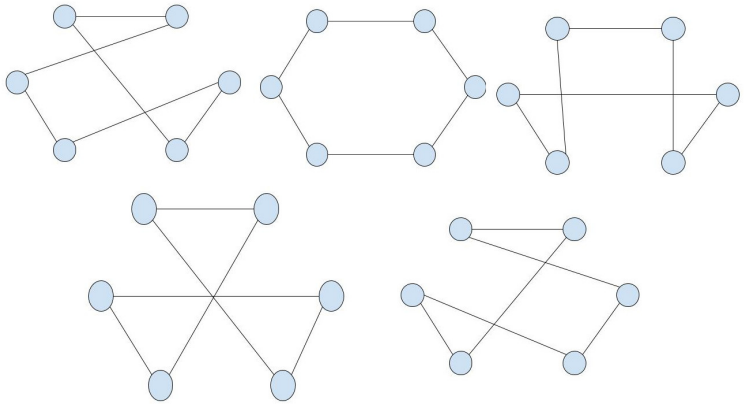
Local search for traveling salesperson problem

We have seen already the idea: define a local operation that changes a given tour into another tour

- 2-opt move:



- 3-opt move:



- Lin-Kernighan-heuristics (and efficient LKH by Helsgaun) is a combination of 2-opt, 3-opt, and rare *k*-opt moves (recall, still state-of-the-art to solve TSP)

Use of local search

Observe: local search methods can be used in any other optimization algorithm in order to (try to) converge to a local minimum.

That is exactly what LKH (Lin-Kernighan-Helsgaun, the very good implementation) does. It starts with a tour based on a minimal spanning tree. However:

- Can all tours be reached with 2-opt moves? (when starting with a certain initial tour) ...*still an open question*
- There are worst case scenarios where the 2-opt heuristics has exponential runtime until convergence.
- What about 3-opt, or *k*-opt, moves? ...*still an open question*

Reactive Tabu Search

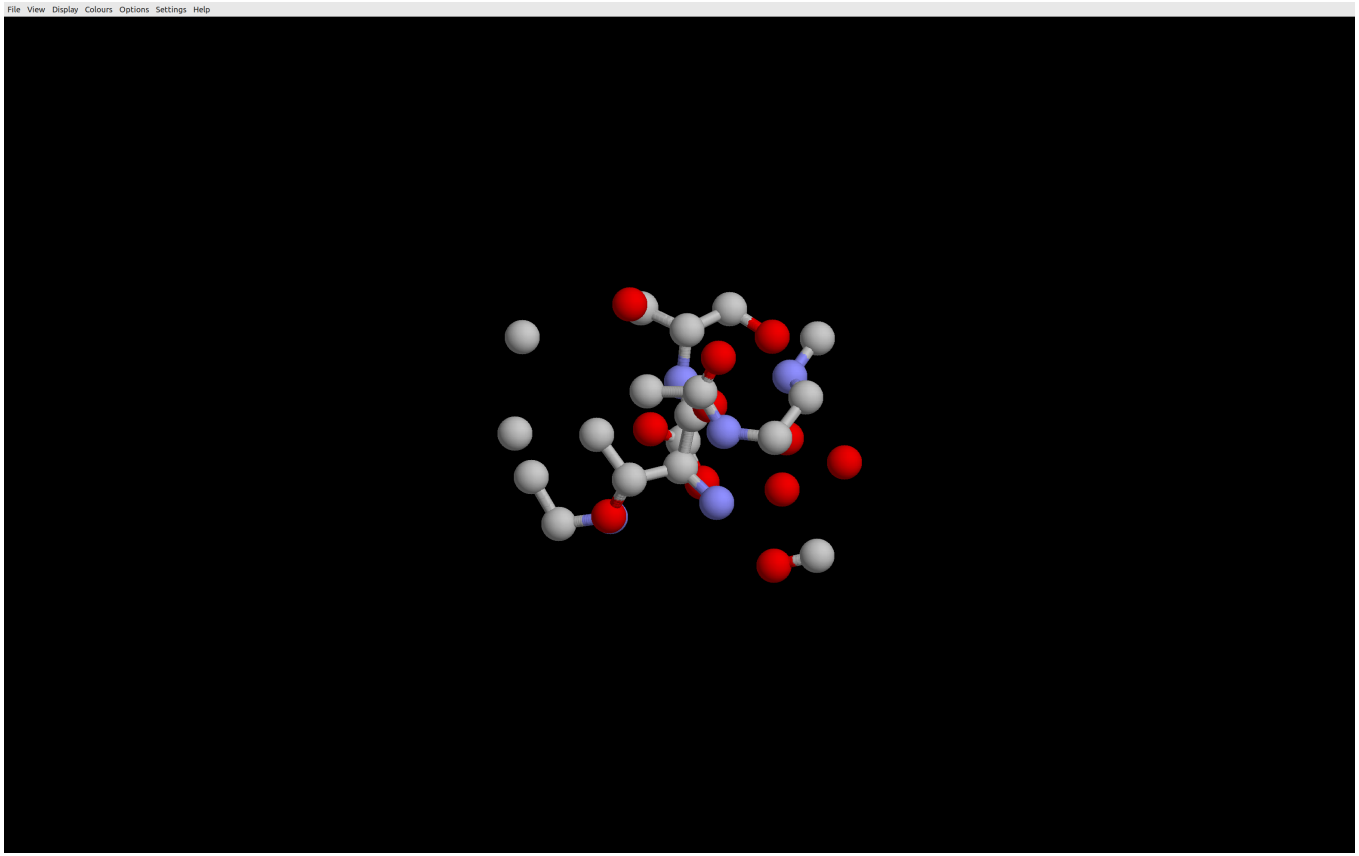
- start with a feasible solution (e.g., with some heuristics)
- search for possibilities to improve the current solution (e.g., search in the neighborhood)
- if we can improve: choose the best one or a random one
- if we cannot improve (i.e., trapped at a minimum):
  - search for possibilities to worsen the current solution
  - if we can escape: try again improvements
  - if we cannot escape: jump to another feasible solution

The Tabu criterion

- avoid repetitive movements taking advantage of a memory that stores forbidden intermediate solutions
  (or forbidden specific features of the current neighborhood search)
- i.e., mark certain movements as tabu for a certain number of iterations, i.e., the memory is volatile!
- reactive means that the tabu period is dynamically adapted

psm: point set match for proteins

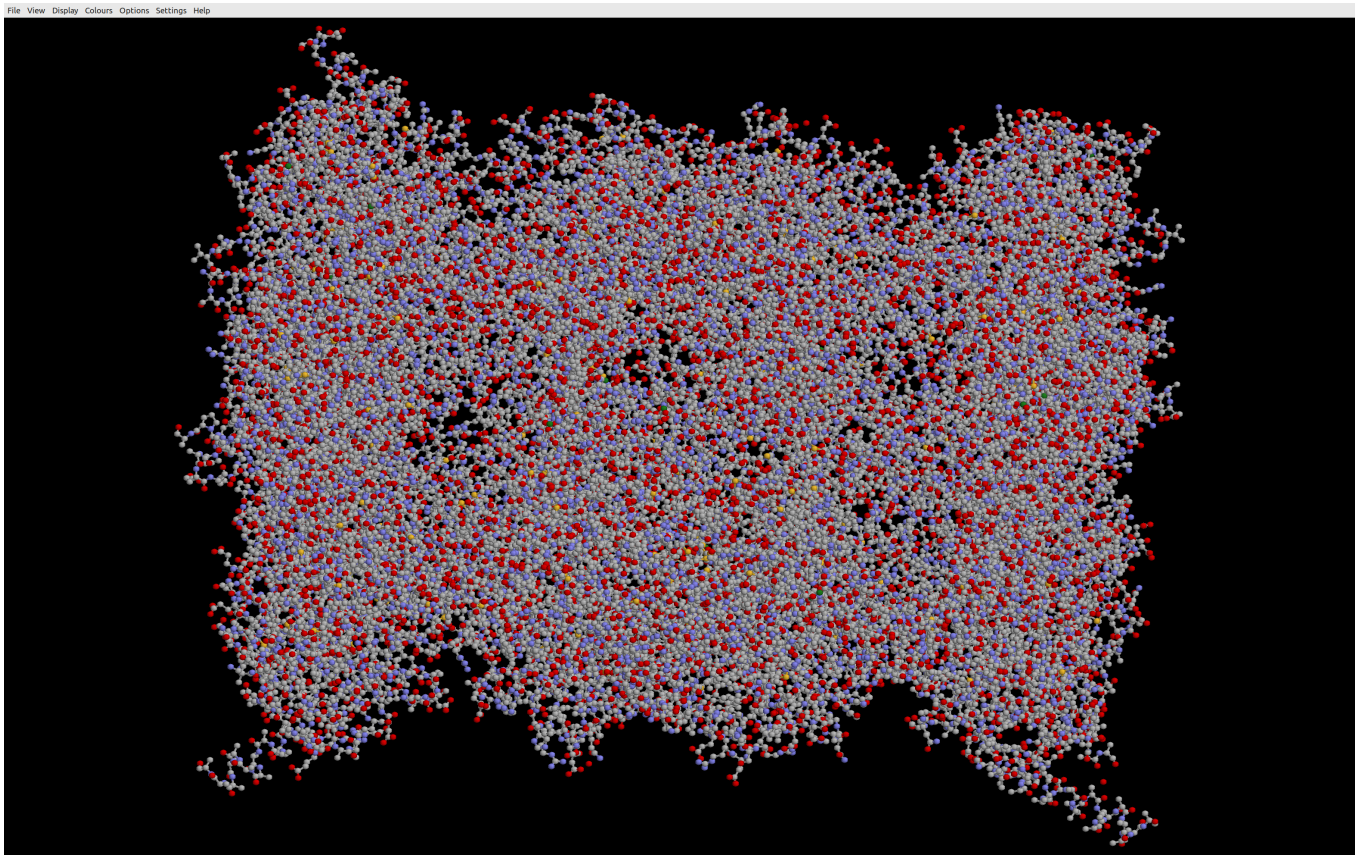
A template and graph based method with local search and use of domain specific knowledge for approximate match.



searching a 3D-structure (34 atoms) in a protein with certain admitted tolerance

psm: point set match for proteins

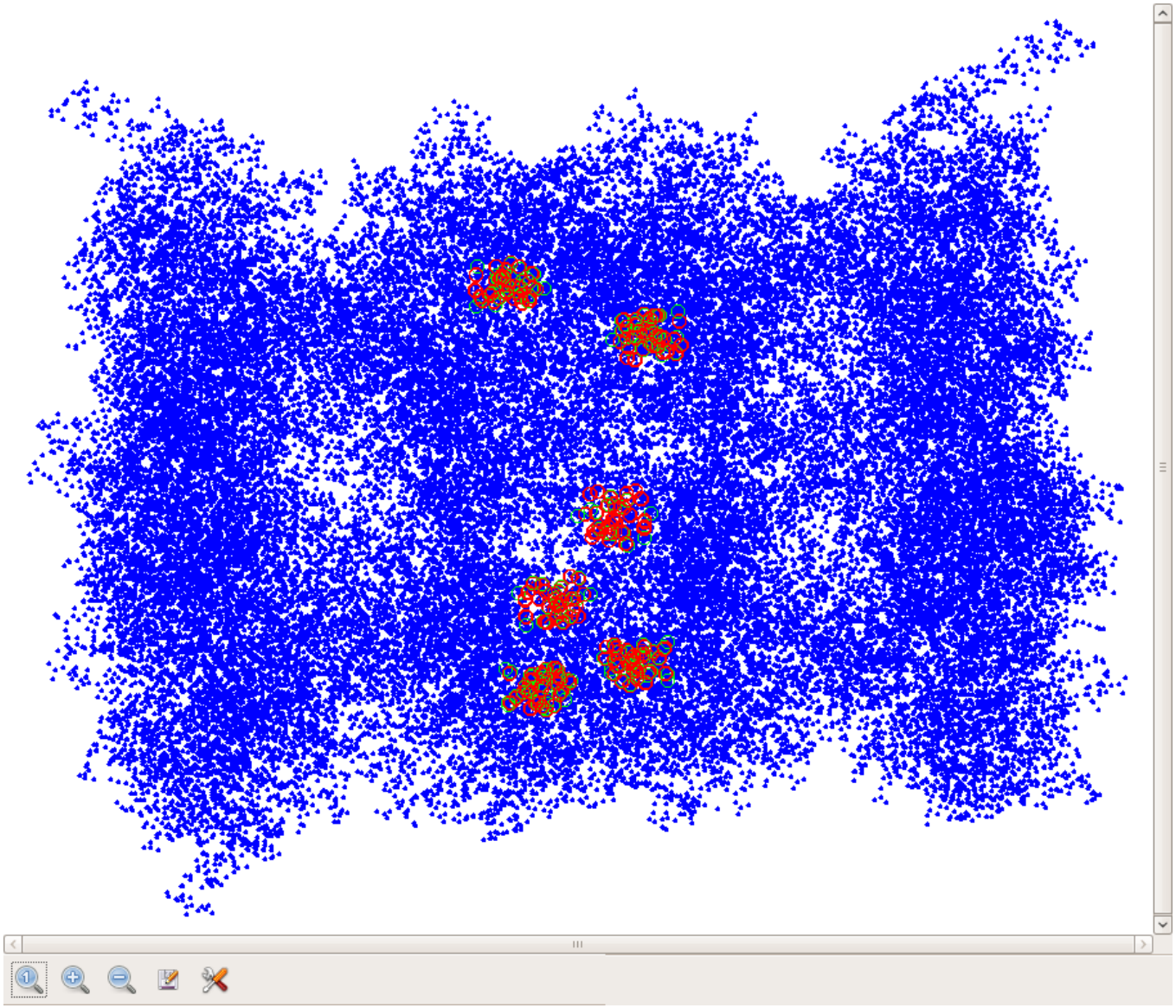
A template and graph based method with local search and use of domain specific knowledge for approximate match.



the protein has 50000 atoms

psm: point set match for proteins

psm found, for instance, 6 locations:



things to take into account

the search space and/or the objective function can be:

discrete	or	continous
total	or	partial
simple	or	complex
explicite	or	implicite
modelado	or	experimental
linear	or	non-linear
convex	or	non-convex
differentiable	or	non–differentiable
constrained	or	unconstrained
static	or	dynamic
single-objective	or	multi-objective

We have seen already a lot of examples of all kind.

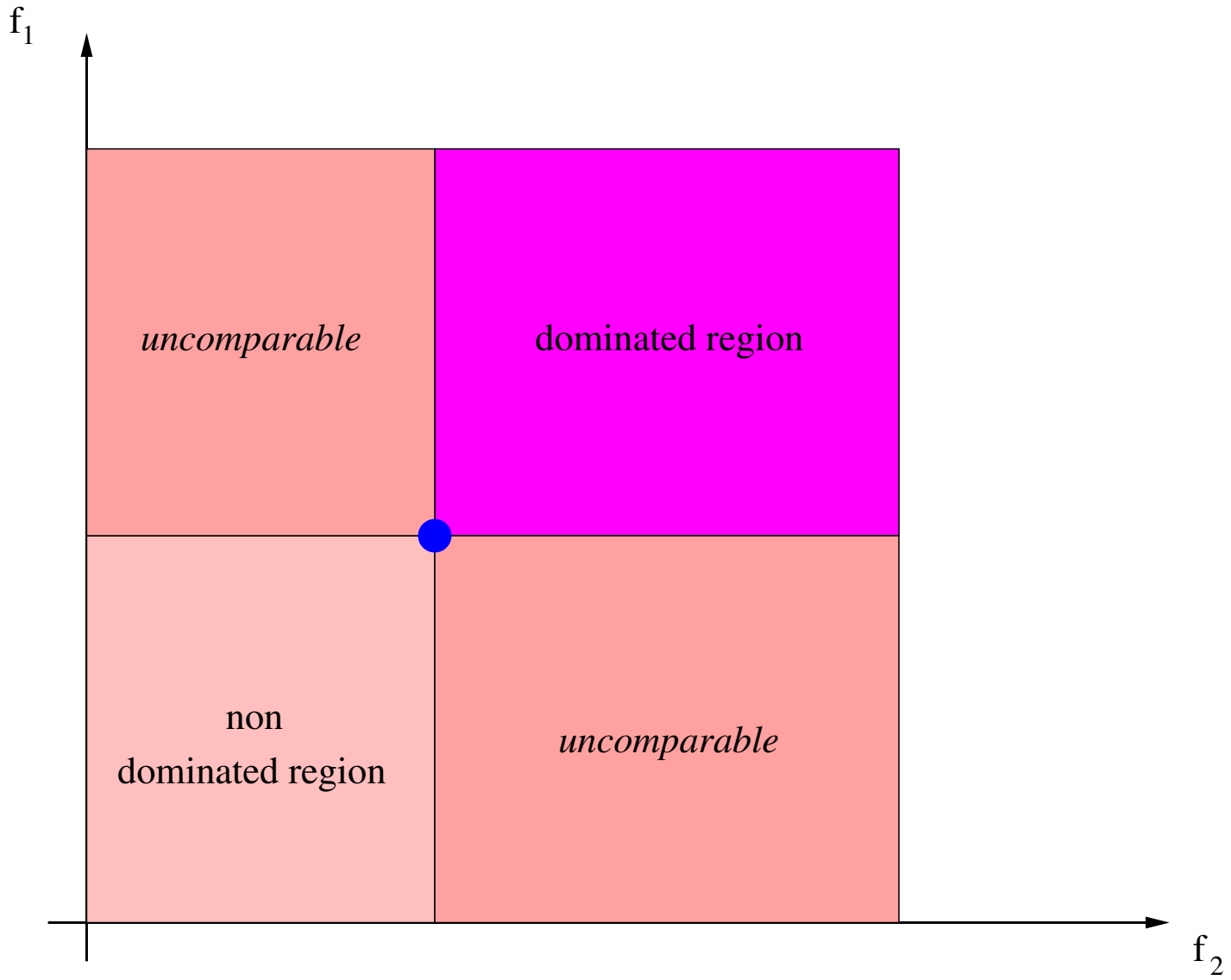
multi-objective optimization

- given a search space $\mathbb{X}$ (called as well *search domain* or *problem space*) and
- a set of k functions f_i (bounded from below) from the search space to the real numbers (or at least a totally ordered set) e.g. $f_i : \mathbb{X} \longrightarrow \mathbb{R}$,
- find an element $x^\star \in \mathbb{X}$ such that $f_i(x^\star) \leq f_i(x)$ for all $x \in \mathbb{X}$ and all k functions f_i
- maybe there is no point in $\mathbb{X}$ that minimizes all the k functions simultaneously, then we look for Pareto-optimal solutions, i.e., solutions that cannot be improved without worsening at least one of the other objective functions

Pareto front

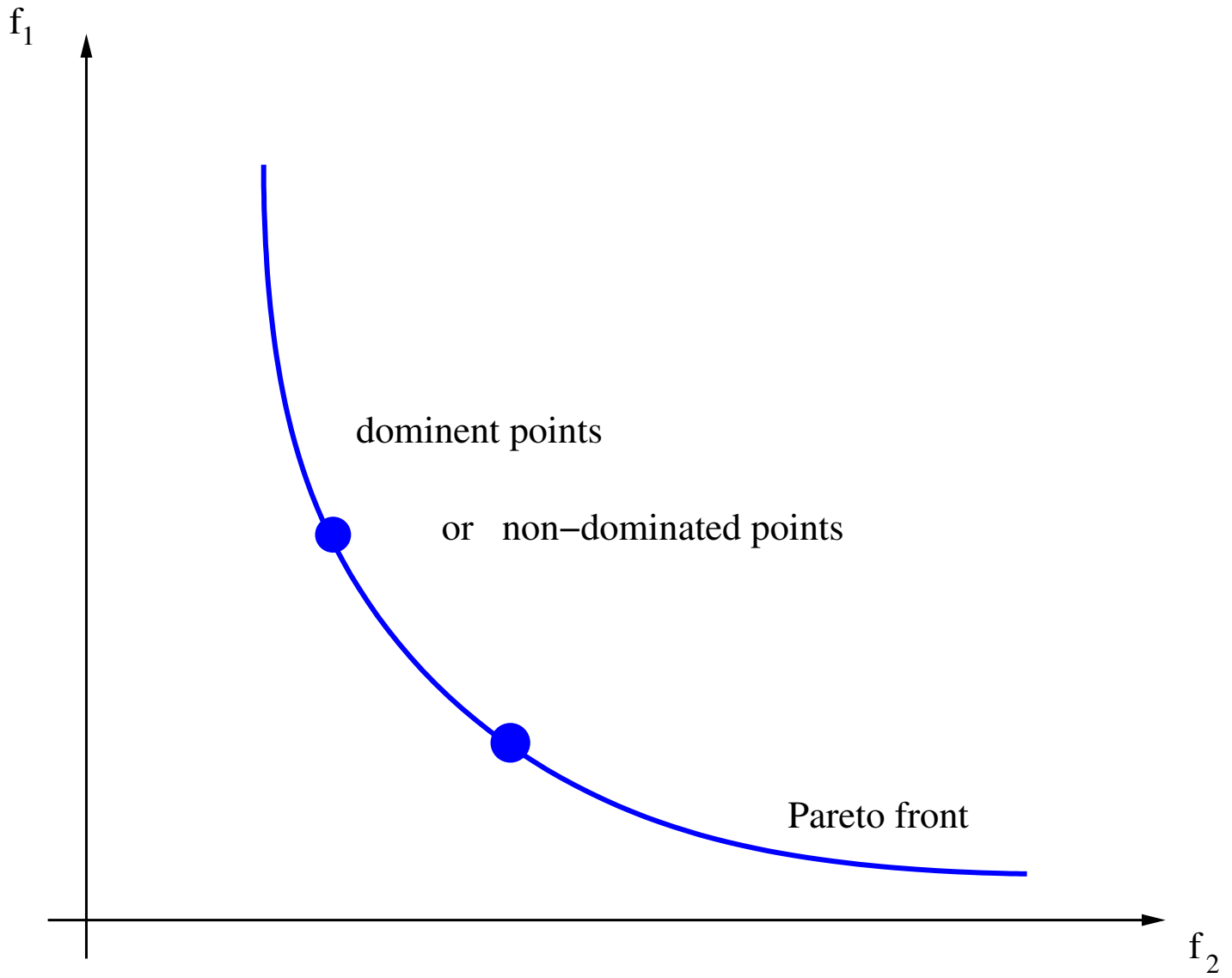
- remind we have more than one independent objective function
- Pareto optimal (global): every other component for all other solutions is worse (or equal)
  (other names are: efficient points or non-interior points)
- Pareto optimal (local): every other component for all other solutions in a local neighborhood is worse (or equal)
- hence, the Pareto front describes the trade-off between the different objectives
- the Pareto front consist of the points in the search space that are not dominated by any other point

Dominant points and regions



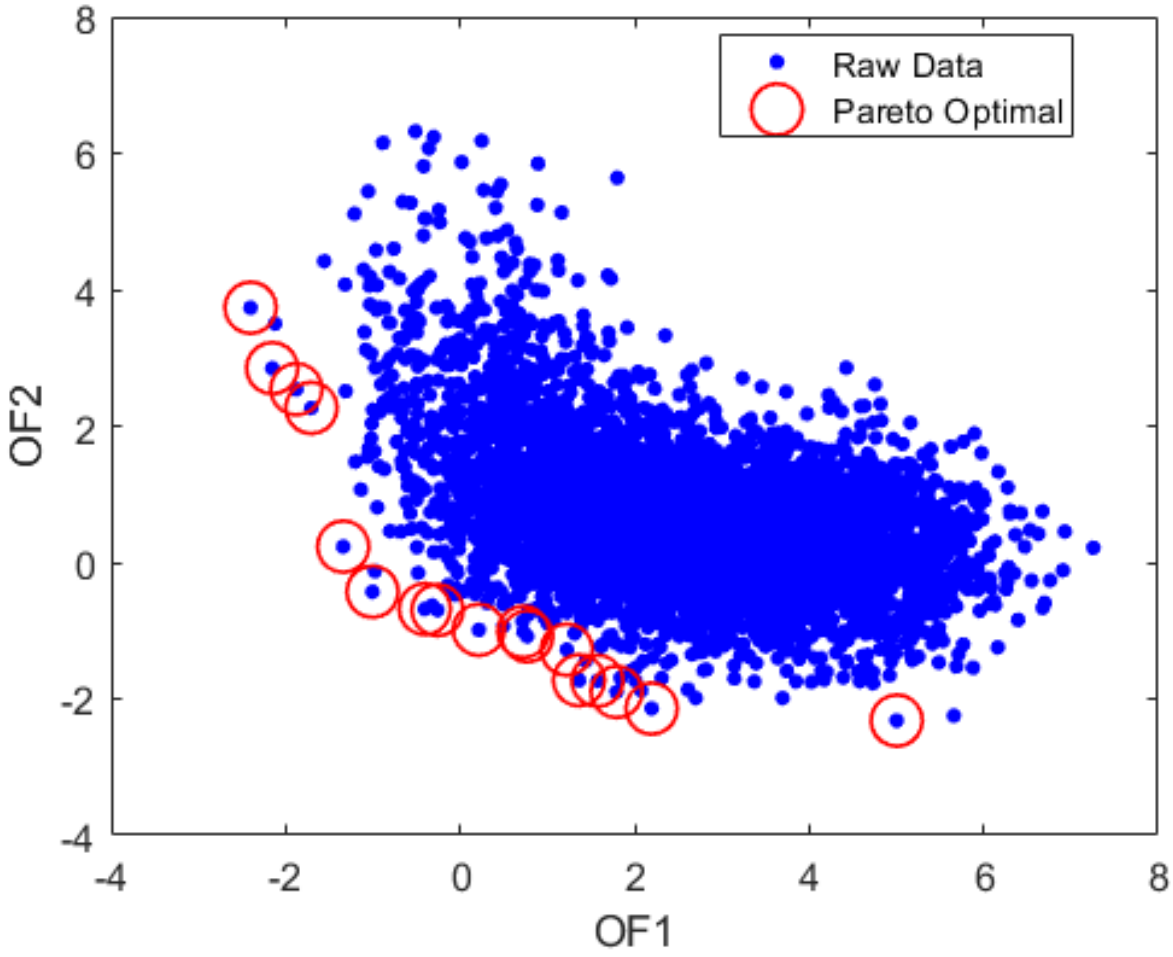
Pareto front

Example of a Pareto front with 2 marked non-dominated points:



Pareto front

Example of a Pareto front for 2 objectives (on measured data):



Multi–objective optimization

How can we find a somewhat good solution to the optimization problem with more than one objective function?

- convex combination of the objectives
- homotopic techniques, i.e., compute the entire Pareto front, for instance with a population based algorithm, and select later...
  (to obtain the Pareto front one might explore the coefficient space of the convex combination)
- goal programming, i.e., fixed values for all objectives and minimize the distance of all objectives to the predefined goals (according to some convenient distance metric)

Multi–objective optimization (continued)

- priority optimization, i.e., fix thresholds for all but one objective function beforehand and optimize above the threshold according to the most important one
- priorization (multi-level) programming, i.e., optimize according to a predefined ordering of the objective functions.
- fixed trade-off, i.e., find the point in the Pareto front that is tangent to a certain hyperplane (especially usefull when Pareto front is convex and low dimensional).

Optimization with restrictions (or constraints)

In many application there are restrictions (or constraints) that limit the optimization process:

- find an element $x^\star \in \mathbb{X}$ such that $f_i(x^\star) \leq f_i(x)$ for all $x \in \mathbb{X}$ and all k functions f_i and
- a certain number L of inequality constraints $g_j(x) \geq 0$ (for all $j \in [1:L]$) are fulfilled, and
- a certain number E of equality constraints $h_n(x) = 0$ (for all $n \in [1:N]$) are fulfilled.

Simple constraints are for instance so-called box-constraints, i.e., the search space is confined in each dimension by an interval.

Such box constraints are often handled separately in the optimization packages.

Optimization with restrictions (or constraints)

- The inequality and equality constraints again might be linear or non-linear functions.
- Due to the restrictions there might arise points (elements of the search space) during the optimization process which are unfeasible, i.e., no valid objective function values can be computed (or even the objective function cannot be computed at all).
- Sometimes even trying to find some feasible solution is already a very complex task (for instance: for TSP with time windows the problem is already NP-hard).

Optimization with restrictions (or constraints)

To tackle constraints there are two main classical approaches:

- use of penalties, i.e., assign *sufficiently large* value(s) to the objective function(s)
- use of interior methods, i.e., make sure not to leave the feasible region
  (main idea: use the fulfillment of the constraints as additional objective function building a so-called *barrier function* with an additional parameter μ that is continuously shrinked to reach zero)

Multi–objective optimization with evolutionary methods

- Evolutionary methods can approximate the Pareto front in parallel (with the help of the diversity among the individuals).
- For instance particle swarm systems varying the weights of a convex combination periodically during the iterations.
- For instance a genetic algorithm can hold a population that tries to converge (in some sense uniformly) towards the Pareto front.

multi-objective evolutionary algorithms (MOEA)

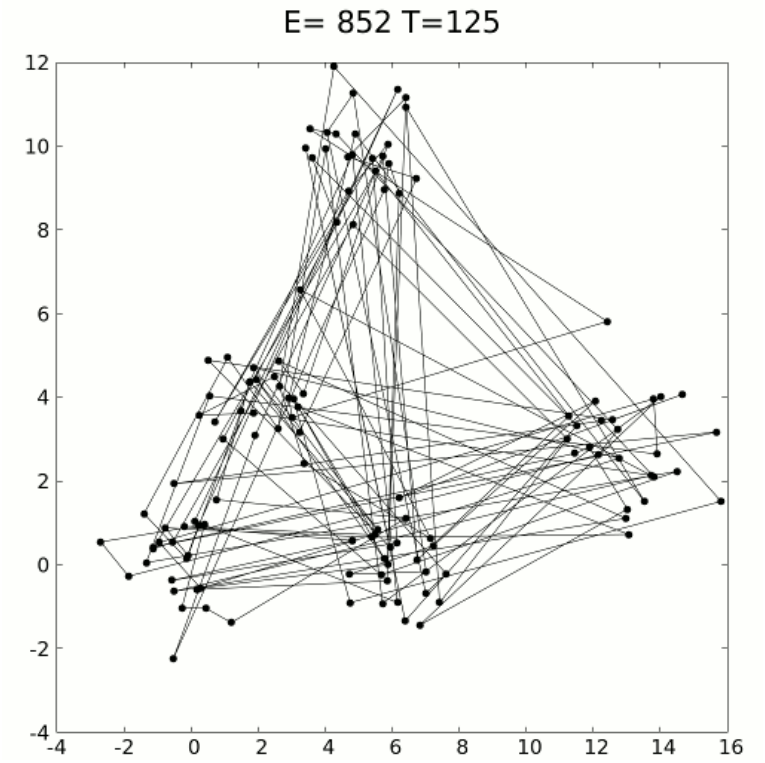
- VEGA, vector evaluated genetic algorithm
  Idea: in the selection process parts of the mating parents are selected according to each objective function
- MSGA, multi-sexual genetic algorithm
  Idea: individuals are marked as belonging to a certain objective function, ranking is used to select parents, only differently marked individuals are used to generate children
- NSGA, non-dominant sorting genetic algorithm
  Idea: sort individuals according to their dominance, and design the selection according the dominance classes (i.e., work with several fronts, intending to converge eventually to the Pareto front.

multi-objective evolutionary algorithms (MOEA)

- NSGA-II, including elitism, dominant individuals are preserved in population, clustering is avoided
- SPEA, Strength Pareto Evolutionary algorithm
  Idea: maintain a fixed set of best individuals while guaranteeing that they are spread over the Pareto front without to much clustering

Simulated Annealing

The name and idea stem from the slow and repeated heating and cooling process of certain materials (usually alloys of different metals with addings, e.g., iron+carbon) until certain properties are achieved.



https://en.wikipedia.org/wiki/Simulated_annealing

Simulated Annealing

- Explores a neighbor in the search space, whenever
  - there is an improvement in the objective function at the neighbor or
  - it is not worse than a temperature dependent threshold with some probability.
- Let Δf be the difference between current solution f and neighbor solution f_n.
- Let T be a temperature.
- Then the neighbor is accepted whenever either $\Delta f < 0$ (we are going downhill)
  or if $e^{-\Delta f/T} > r$, for r being a random value in $[0 : 1]$ (we are going uphill).
- With increasing iteration rounds (Monte Carlo iterations where the temperature is held constant) the temperature is reduced, hence, the threshold converges towards zero.

Simulated Annealing cooling schemes

- linear cooling: $T = T_0 - \eta k$
  with T_0 an initial temperature, k the iteration number and η some free parameter, being constant during optimization.
  (To avoid negative temperature: $T = \max(T_0 - \eta k, T_{\min})$)
- exponential cooling: $T = aT$
  with $a \in [0.8, 1)$ typically; slower cooling the a close to 1.
- inverse cooling: $T = T/(1+\beta T)$
  with β being a small constant (e.g. $\beta = 0.001$).
- logarithmic cooling: $T = T_0/\log k$
  with c a suitable constant.
  (Used as well a generalization: $T = T_0/\log(k+d)$).
  Not really practical in applications but was used to prove convergence to global minimum under certain conditions.
- inverse linear cooling: $T = T_0/k$.
  Not really practical in applications but was used to prove convergence to global minimum under certain conditions.

Simulated Annealing cooling schemes

- It seems that any monotonely decreasing function, that is neither too steep at the beginning and neither too slow decreasing in the end, might serve.
- Cooling can be implemented differently in each dimension in multi-dimensional problems.
- The entire process can be restarted from another location in search space (Monte Carlo approach around inner simulated annealing process).

Simulated Annealing vs. Tabu Search

- Both methods belong to the random path methods:
  - there is only one individual
  - the moves in the search space explore the neighborhood in a random manner
  - a move is accepted whenever
    - TS: a random, but not tabu, move among the improving once in first place and non-improving onces in second place.
    - SA: a random move that fulfills the annealing condition
- Tabu search is, as improvements are prefered, a local search method that finds local minima;
  which might not be the case for simulated annealing (you need to add some final local search approach).

memetic algorithms

The population based algorithms usually do not perform local optimization of the individuals.

Including such an approach is called a memetic algorithm:

- use an adequate local search strategy (e.g., steepest decent, iterated local search, etc.) to improve the fitness of the individuals
- this improvement of an individual is also called *individual learning*
- the result of the local optimization:
  - might change the genotype of the individual (Lamarckian learning), i.e., the changed individual participates in the genetic algorithm,
  - or might not change the genotype of the individual (Baldwinian learning), i.e., the population is not changed

memetic algorithms efficiency considerations

- the local search can be restricted to a certain part of the population, for instance, the best ranked ones
- the local search can be restricted to be performed only after a certain amount of iterations in the overall genetic algorithm

biogeography-based optimization

- The biogeography-based optimization uses the idea of populations evolving at islands that once in a while exchange individuals via migration.
- Implemented as a further metaheuristic in the framework of genetic algorithms (especially for separable objective functions, where optimization of their convex combination is an option).
- Can be applied to particle swarm optimization as well (I have found implementation examples, e.g., from 2017).

much more nature-inspired work done

Without going into details, there is a huge bunch of other nature-inpired optimization heuristics (see introduction as well):

- ant lion optimization
- artificial bee colonies
- bat algorithm
- dragonfly algorithm
- firefly optimization
- grasshopper optimization
- grey wolf optimizer
- whale optimization
- invasive weed optimization
- cristalization of materials
- great deluge algorithm
- gravitational search algorithm
- etc. etc. etc.

issues to be considered

The following issues should be considered when implementing and using a certain optimization approach for a given problem:

- implementation difficulty
- number of parameters to be adjusted
- population size
- number of objective function evaluations
- convergence velocity (convergence profile)
- handling of local optima (stucking)
- handling of restrictions
- statistically correct evaluation on benchmark problems and Monte Carlo runs
- is there a Las Vegas condition available?

efficiency issues to be considered

The following efficiency aspects might be useful in a certain implementation of an optimization approach on a given platform:

- use of parallelization
- use of caching
- use of efficient data structures
- use of adequate precision in calculations
- use of approximation algorithms (especially in early phases for the objective function computation)
- use of good random number generators

hyper-parameter tuning

Many of the different heuristics seen so far have a certain number of free parameters that must be fixed when starting the optimization, but

- We might use another optimization algorithm that tries to find a good parameter set for the original algorithm.
- For instance, we run the algorithm on a suitable benchmark suite to obtain good free parameters, and then run the optimization with these settings on the real problems we are interested to solve.
- This process can be iterated, i.e., each use of the optimizer might adapt to a certain degree its free parameters.
- Eventually, we arrive at an online-algorithm which adapts its parameters with each problem it is faced to.
- We might try to find a correlation between good free parameters and certain problem characteristics.

last but not least

that’s it, folks ... thanks. questions?