Day 2: Optimization (continued)

Presenter: Auralee Edelen

Day 2

1st and 2nd Order Optimization Methods

1st Order Methods:

$$x_{n+1} = x_n - \alpha \nabla f'(x_n)$$

- Use information about the 1st derivative
- Gradient descent and variants (yesterday's lecture)

→ how to improve choices of step-size and direction?

(and in turn improve sample-efficiency in convergence)

1st and 2nd Order Optimization Methods

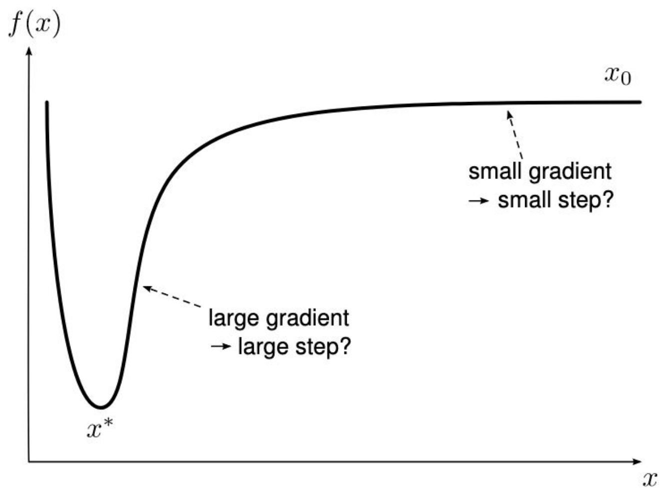
1st Order Methods:

- Use information about the 1st derivative
- Gradient descent and variants (yesterday's lecture)

$\longrightarrow$ how to improve choices of step-size and direction?

(and in turn improve sample-efficiency in convergence)

$$x_{n+1} = x_n - \alpha \nabla f'(x_n)$$



1st and 2nd Order Optimization Methods

1st Order Methods:

- Use information about the 1st derivative
- Gradient descent and variants (yesterday's lecture)

→ how to improve choices of step-size and direction?

(and in turn improve sample-efficiency in convergence)

2nd Order Methods:

- Use information about the curvature (2nd derivative)

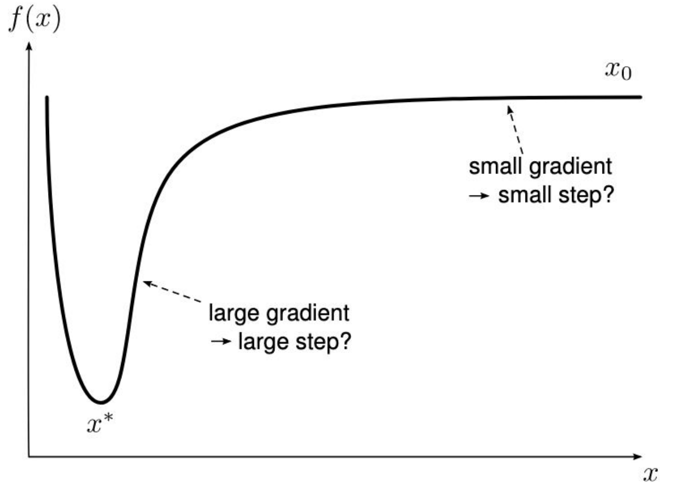
  → calculate or approximate the Hessian

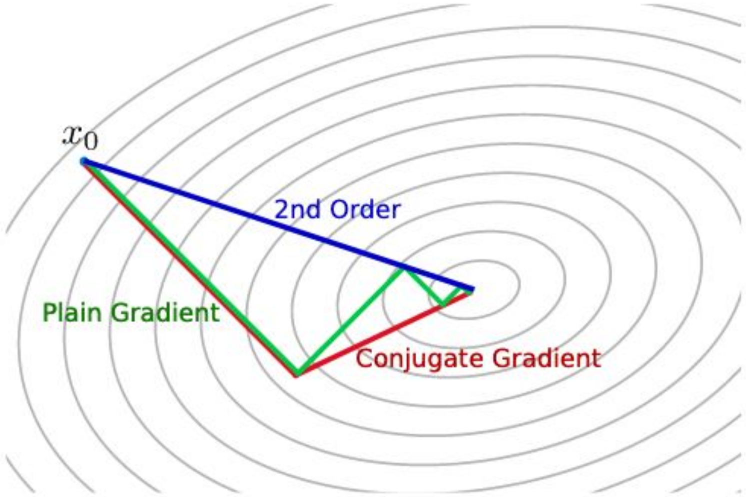
- Pro: Better direction and step-size than gradient descent
- Con: More costly per iteration to compute

"1.5 Order" Methods:

- In between 1st and 2nd order
- e.g. Powell's conjugate gradient method

$$x_{n+1} = x_n - \alpha \nabla f'(x_n)$$





Determining the next point: line search

Line search methods define a direction and then do optimization over that line

→ *often used as one step in a larger algorithm (e.g. for determining next point)*

→ *often do not require derivatives*

e.g. Brent's Method:

1. Bracket the minimum
2. Approximate parabola through successive points or use golden section search
3. Iterate

More details: Brent, R. P. Ch. 3-4 in [*Algorithms for Minimization Without Derivatives.*]
Englewood Cliffs, NJ: Prentice-Hall, 1973.

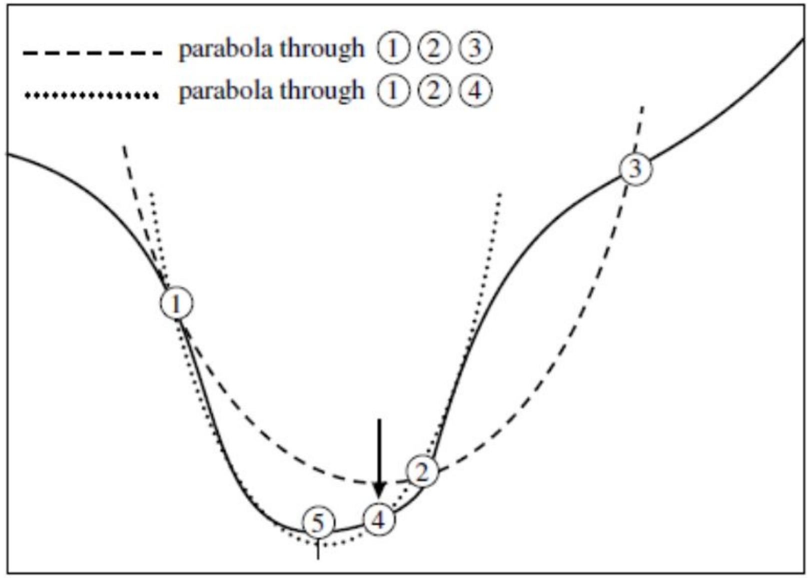
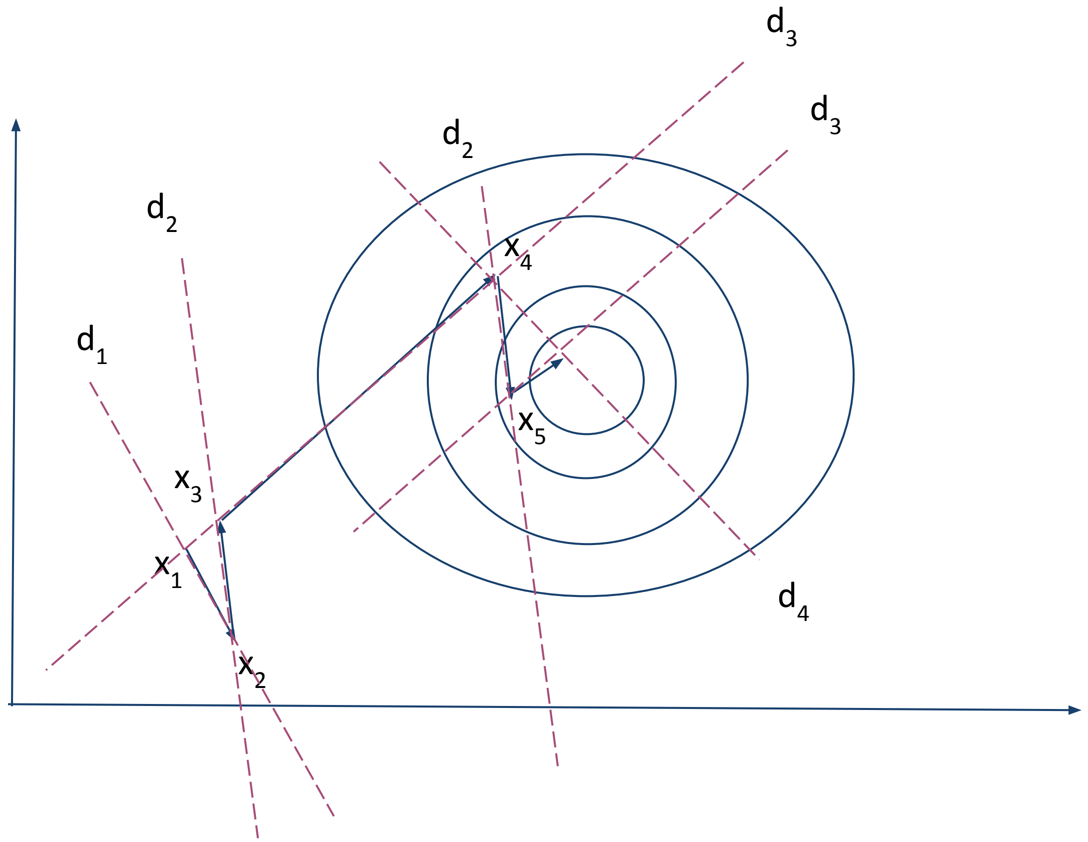


Illustration of Brent's method from Numerical Recipes

Example: Powell's Conjugate Gradient Method

Steps in Powell's Method:

1. Pick x_1 and two directions d_1 and d_2
2. Start at x_1 and do line search along d_1 to find minimum x_2
3. Start at x_2 and do line search along d_2 to find x_3
4. Connect x_1 to x_3 to define d_3
5. Start at x_3 and do line search along d_3 to find x_4
6. Start at x_4 and do line search along d_2 to find x_4

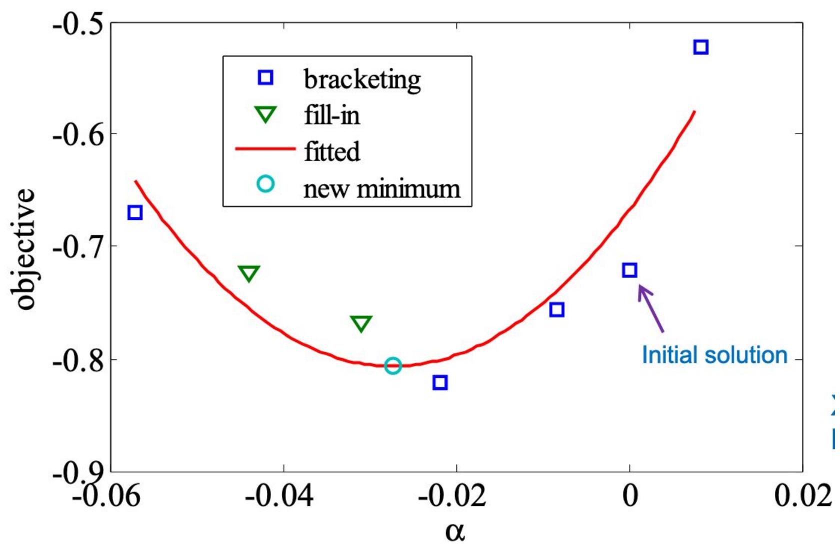


Does not require derivatives, efficient with regard to direction searched

Robust Conjugate Direction Search (RCDS)

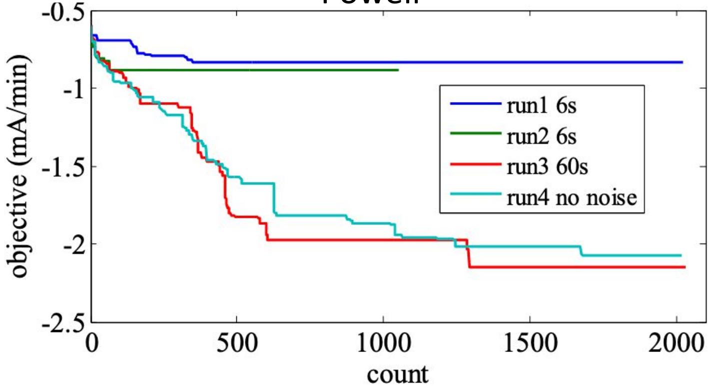
RCDS combines a noise-aware line search with Powell's method

- **Designed to deal with noisy online optimization in accelerators**
  → optimization algorithm needs to be sample-efficient, robust to noise
- Was developed for optimization of storage rings (e.g. dynamic aperture, emittance); has since been widely applied in accelerators
- Uses a random sample to find the bounds in each line search, ensuring these are above a specified noise level, and fills in values until an appropriate fit is obtained



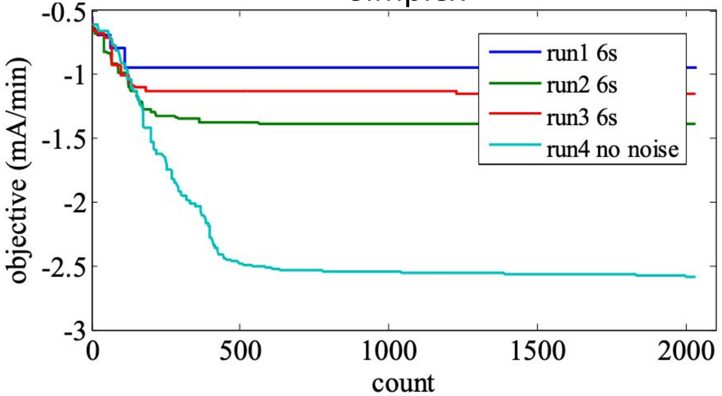
More details: X. Huang et al, Nucl. Instr. Methods, A 726 (2013) 77-83, https://www.slac.stanford.edu/pubs/slacpubs/15250/slac-pub-15414.pdf

Powell



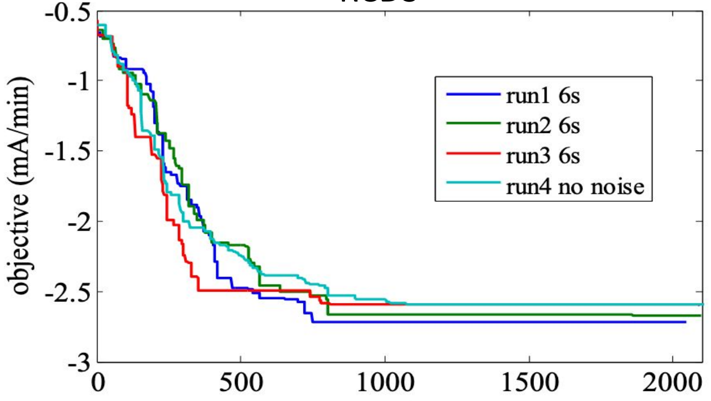
Powell's method not robust with noise

Simplex



simplex does not reach expected minima

RCDS



2nd Order Methods: Newton's method

Analogy to root finding

Taylor series approximation:

$$f(x) \approx f(a) + (x-a)f'(a)$$

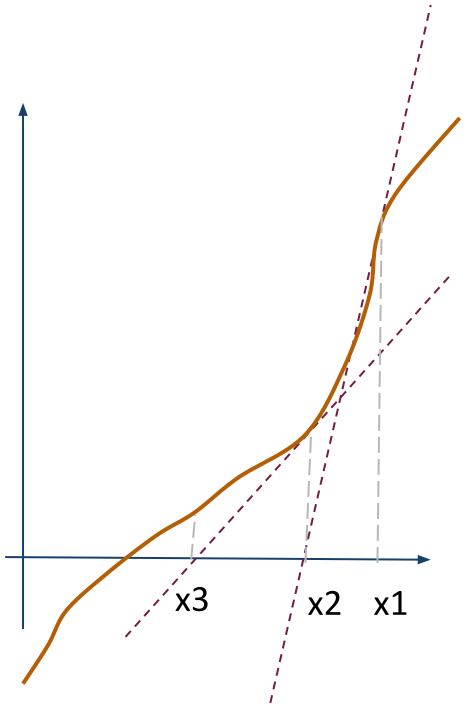
Set to 0 to find roots of function:

$$0 = f(a) + (x-a)f'(a)$$

$$x = a - \frac{f(a)}{f'(a)}$$

Iterate:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



2nd Order Methods: Newton's method

Analogy to root finding

Taylor series approximation:

$$f(x) \approx f(a) + (x-a)f'(a)$$

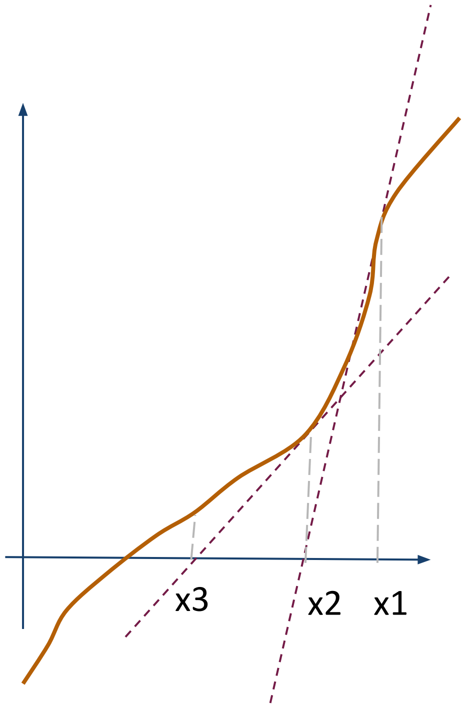
Set to 0 to find roots of function:

$$0 = f(a) + (x-a)f'(a)$$

$$x = a - \frac{f(a)}{f'(a)}$$

Iterate:

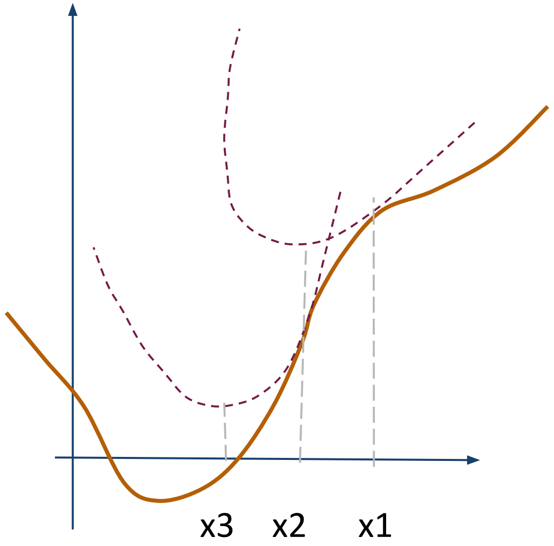
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



For optimization, take the derivative of the Taylor series

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

$$x_{n+1} = x_n - H^{-1}f(x_n)\nabla f(x_n)$$



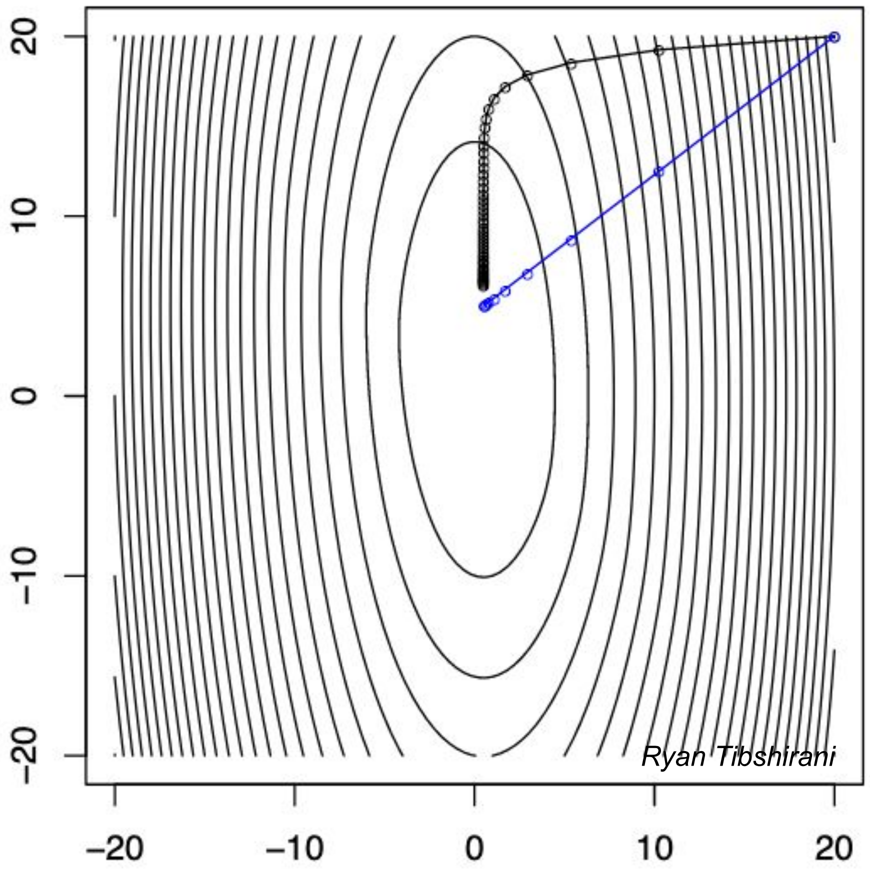
In essence: Newton's method iteratively approximates the function with a parabola

2nd Order Methods: Newton’s method

- Pro: better convergence per step than gradient descent
- Con: poor computational scalability $O(n^3)$

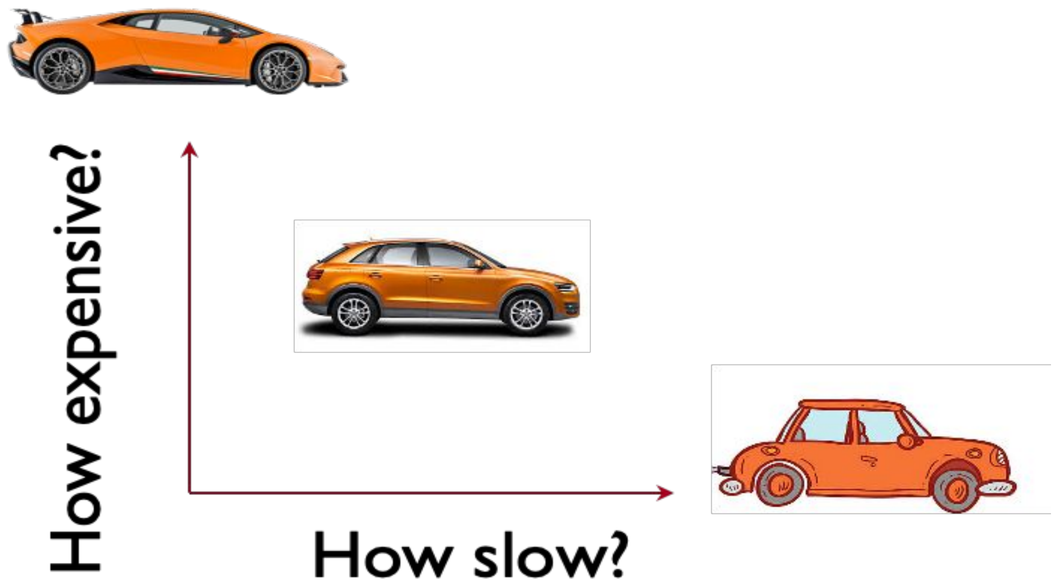
  $\longrightarrow$ *computing hessian and inverse*
- Quasi-Newton methods approximate the Hessian for better scalability (e.g. L-BFGS)

black - gradient descent
blue - Newton’s method



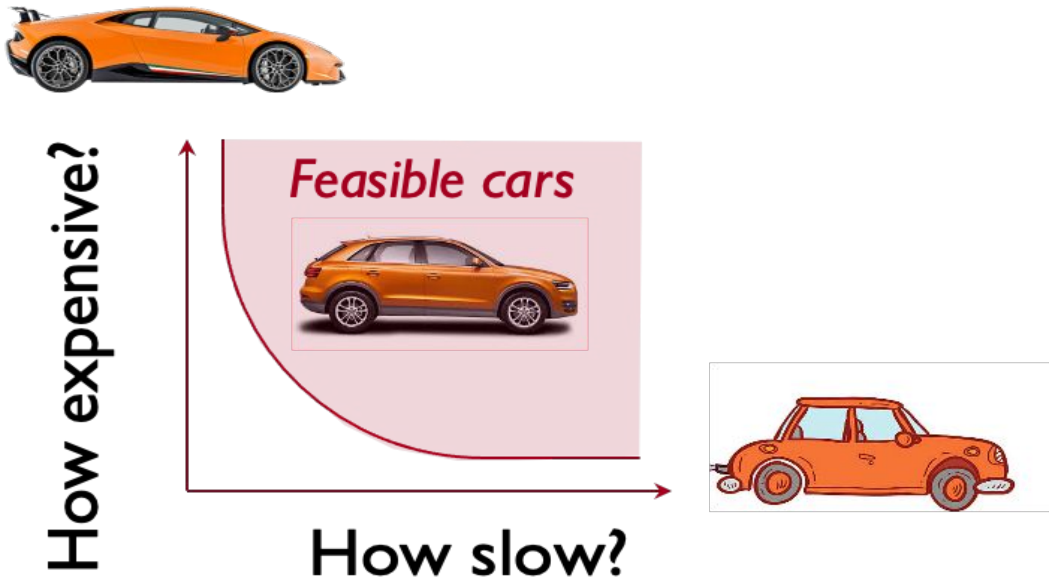
Multi-Objective Optimization: Intro

Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives



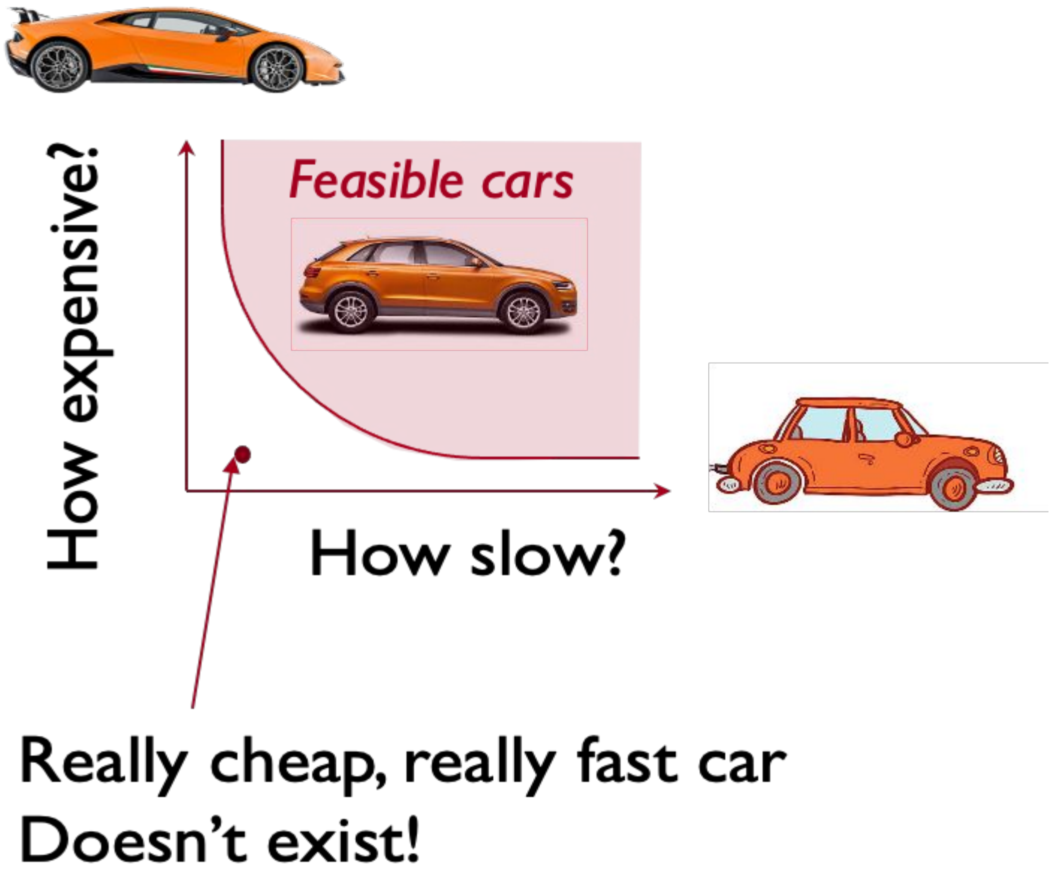
Multi-Objective Optimization: Intro

Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives



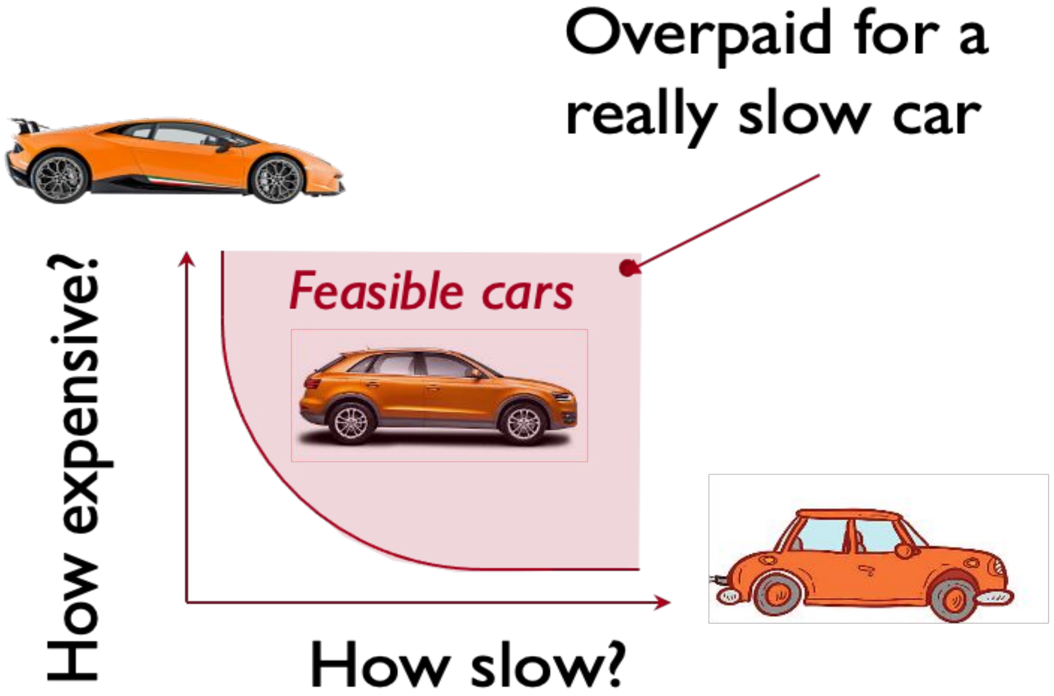
Multi-Objective Optimization: Intro

Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives



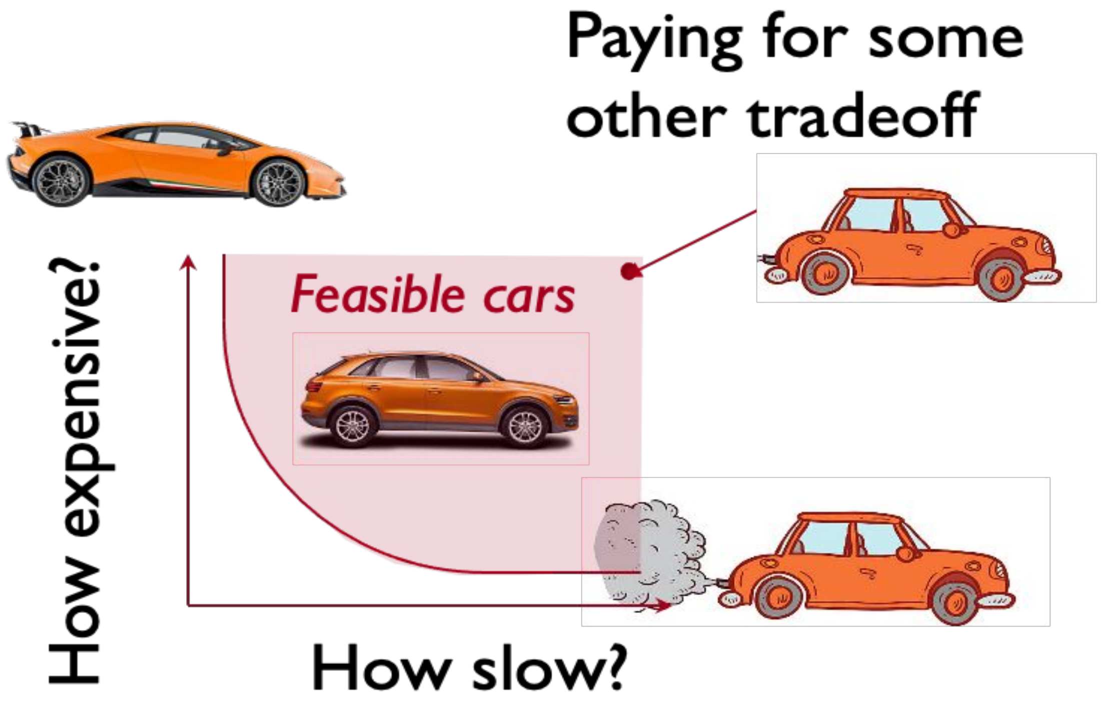
Multi-Objective Optimization: Intro

Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives



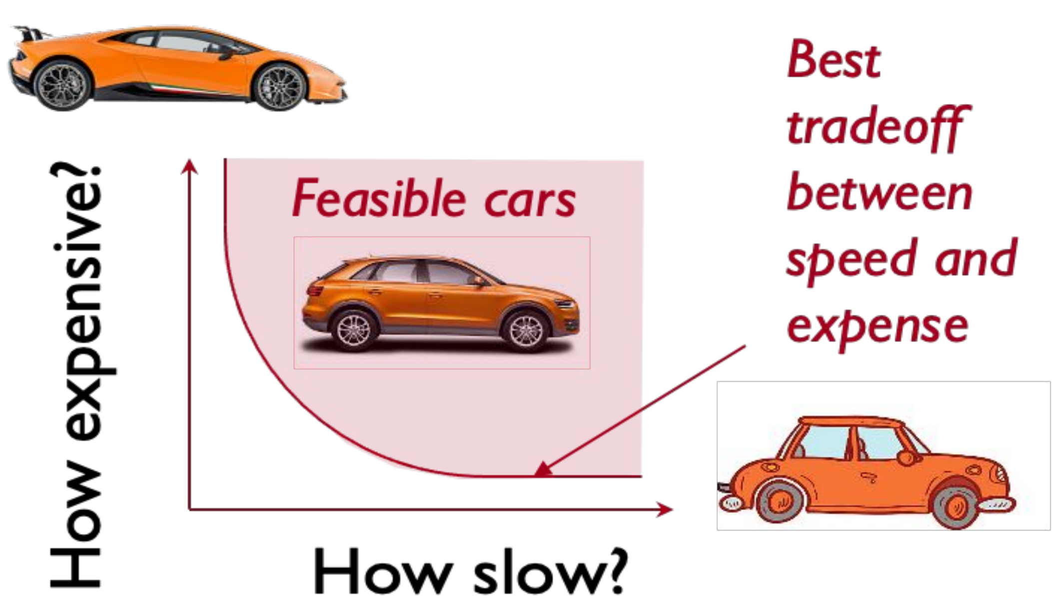
Multi-Objective Optimization: Intro

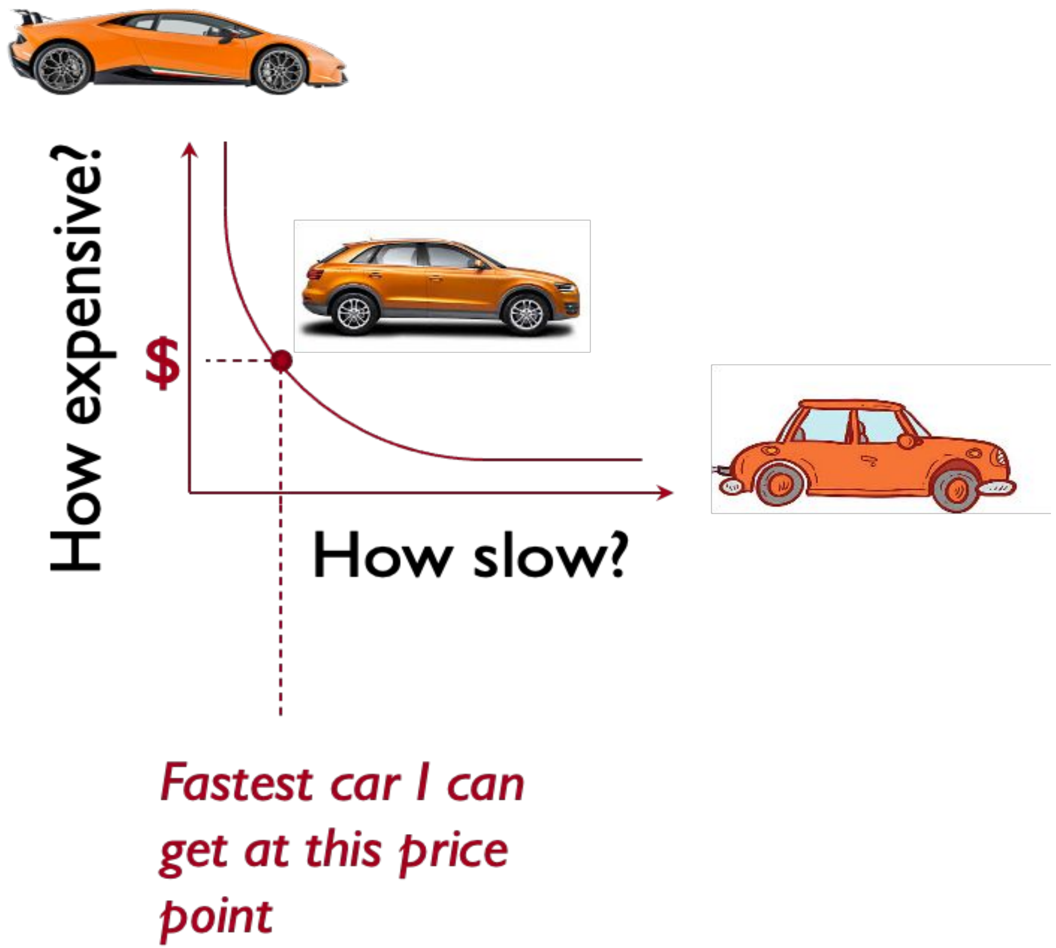
Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives



Multi-Objective Optimization: Intro

Instead of a single objective, in multi-objective optimization (MOO) we want to optimize multiple objectives

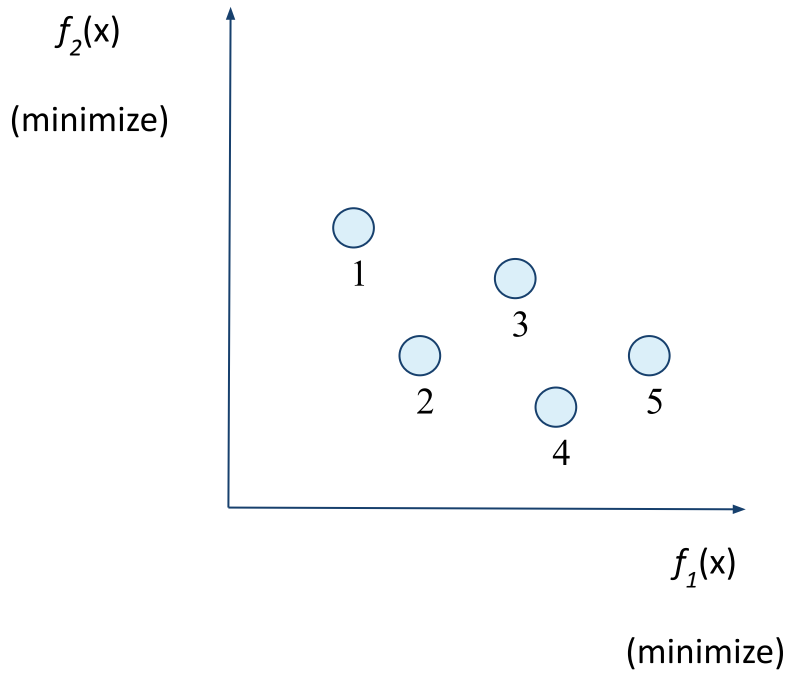




Multi-Objective Optimization: Dominance

Solutions in single-objective optimization are easy to compare by looking at the objective function values

In multi-objective optimization, solutions are evaluated by the **dominance** wrt the combinations of objectives



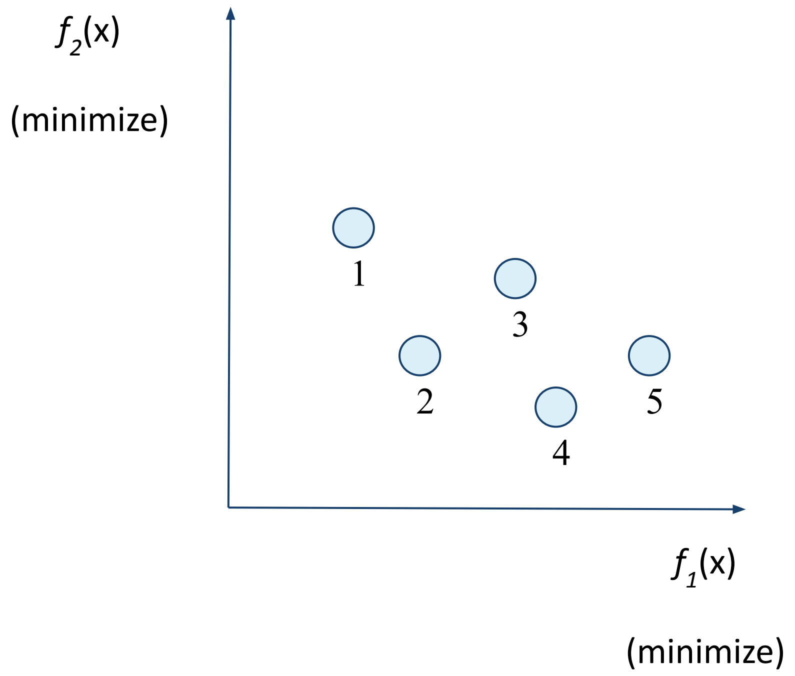
Solution x^1 is said to dominate solution x^2 if both of the following are true:

1. x^1 is no worse than x^2 for all objectives
2. x^1 is strictly better than x^2 in at least one objective

Multi-Objective Optimization: Dominance

Solutions in single-objective optimization are easy to compare by looking at the objective function values

In multi-objective optimization, solutions are evaluated by the **dominance** wrt the combinations of objectives



Solution x^1 is said to dominate solution x^2 if both of the following are true:

1. x^1 is no worse than x^2 for all objectives
2. x^1 is strictly better than x^2 in at least one objective

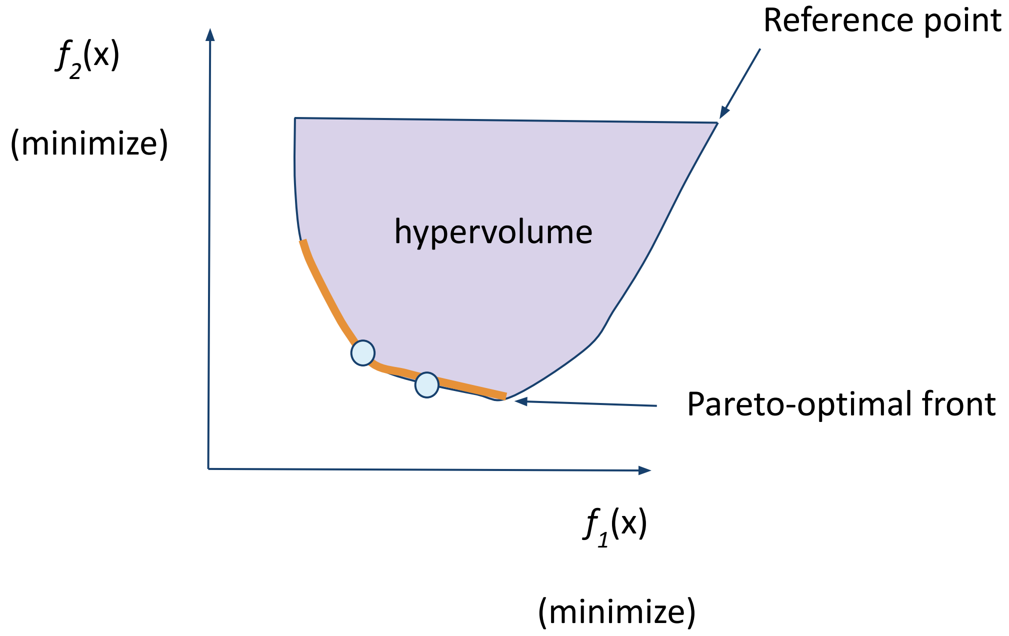
In this example: 2 dominates 3   4 dominates 5   2 dominates 5   Neither 1 nor 2 dominate each other

Multi-Objective Optimization: Pareto Front

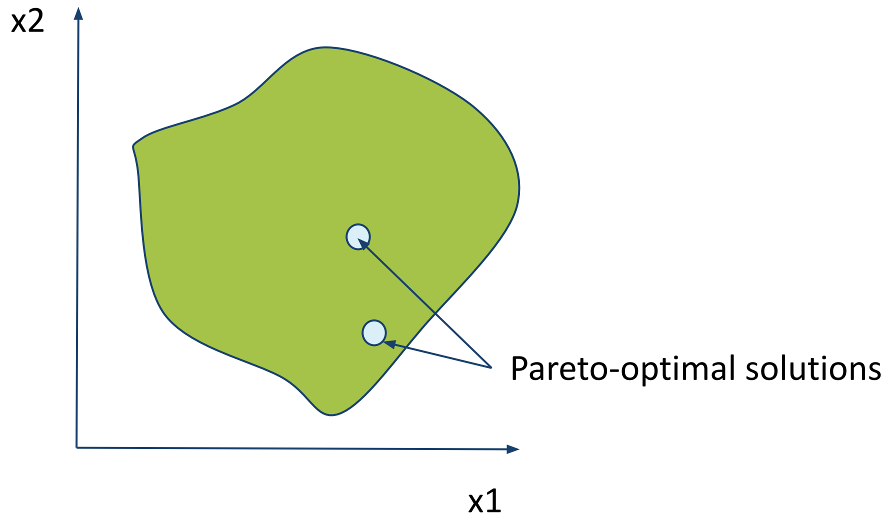
- **Non-dominated solution set:** *all solutions which are not dominated*
- **Pareto-optimal set:** *non-dominated solution set over the entire decision space*
- **Pareto-optimal front:** *boundary mapped out by Pareto-optimal set*

→ ***For any point on the Pareto front, one cannot improve the value of one objective without reducing another***

feasible objective space



feasible design (or "decision") space



Example shown is 2D for visualization, but can in principle go up to N-D

Multi-Objective Optimization: Pareto Front

Can also have more complicated Pareto fronts that provide additional challenges (e.g. disconnected regions)

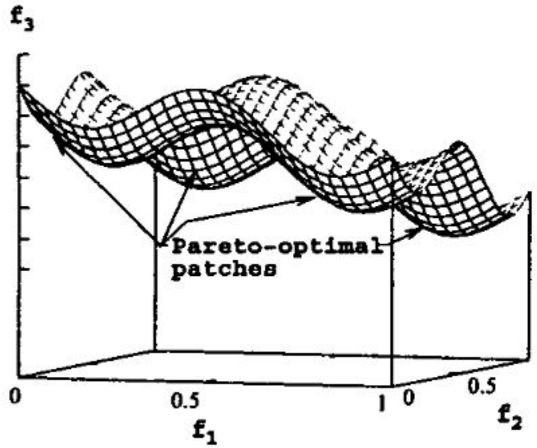
Numerous standard "test functions" are used to assess optimization problems (including multi-objective and constrained optimization)

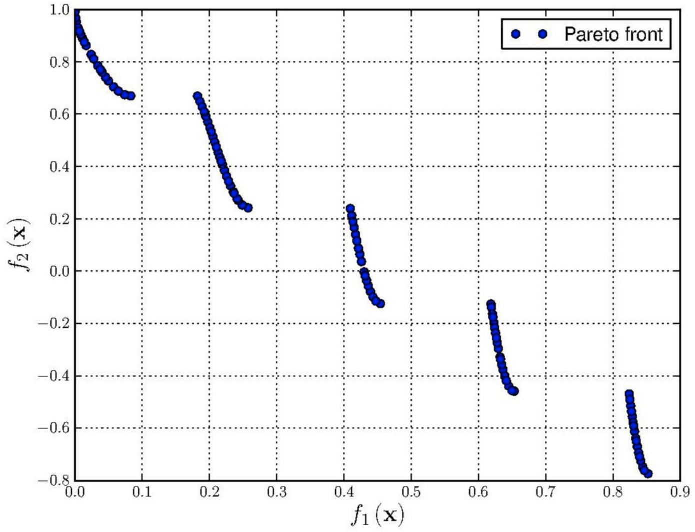
e.g. see:

Deb et al., Scalable multi-objective optimization test problems, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1007032

Husband et al., A review of multiobjective test problems and a scalable test problem toolkit, https://ieeexplore.ieee.org/document/1705400

Wikipedia page on test functions for optimization: https://en.wikipedia.org/wiki/Test_functions_for_optimization





ZDT-3

https://en.wikipedia.org/wiki/File:Zitzler-Deb-Thiele%27s_function_3.pdf

Multi-Objective Optimization: Scalarization

Scalarization: convert multiple objectives to a single objective

$$\sum_{i=1}^{k} w_i f_i(x)$$ for k objectives

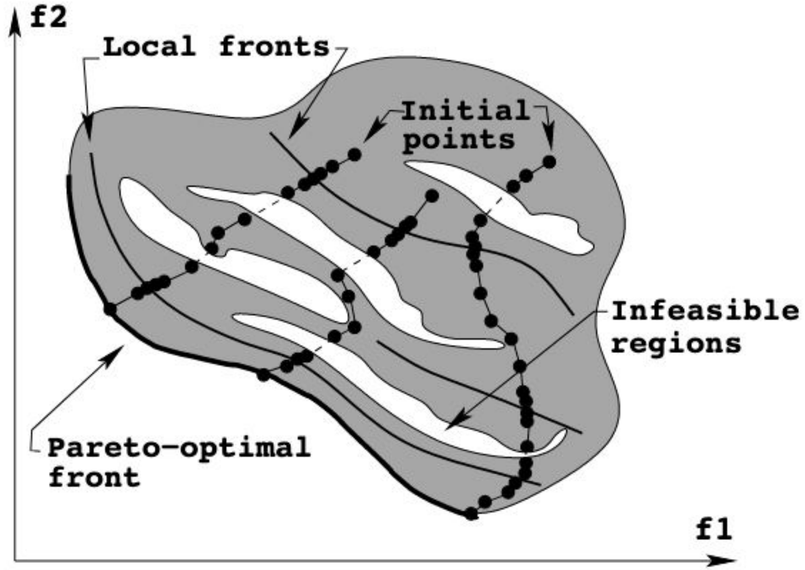
(linear scalarization)

Pro: simple to implement

Con: for a given set of weights one gets only one solution

$\longrightarrow$ Need to solve optimization problem multiple times with scan through different weights to obtain the Pareto-optimal front

Can be time-consuming and difficult for each optimization to navigate to the front $\longrightarrow$ *motivation for population based methods (parallel search)*

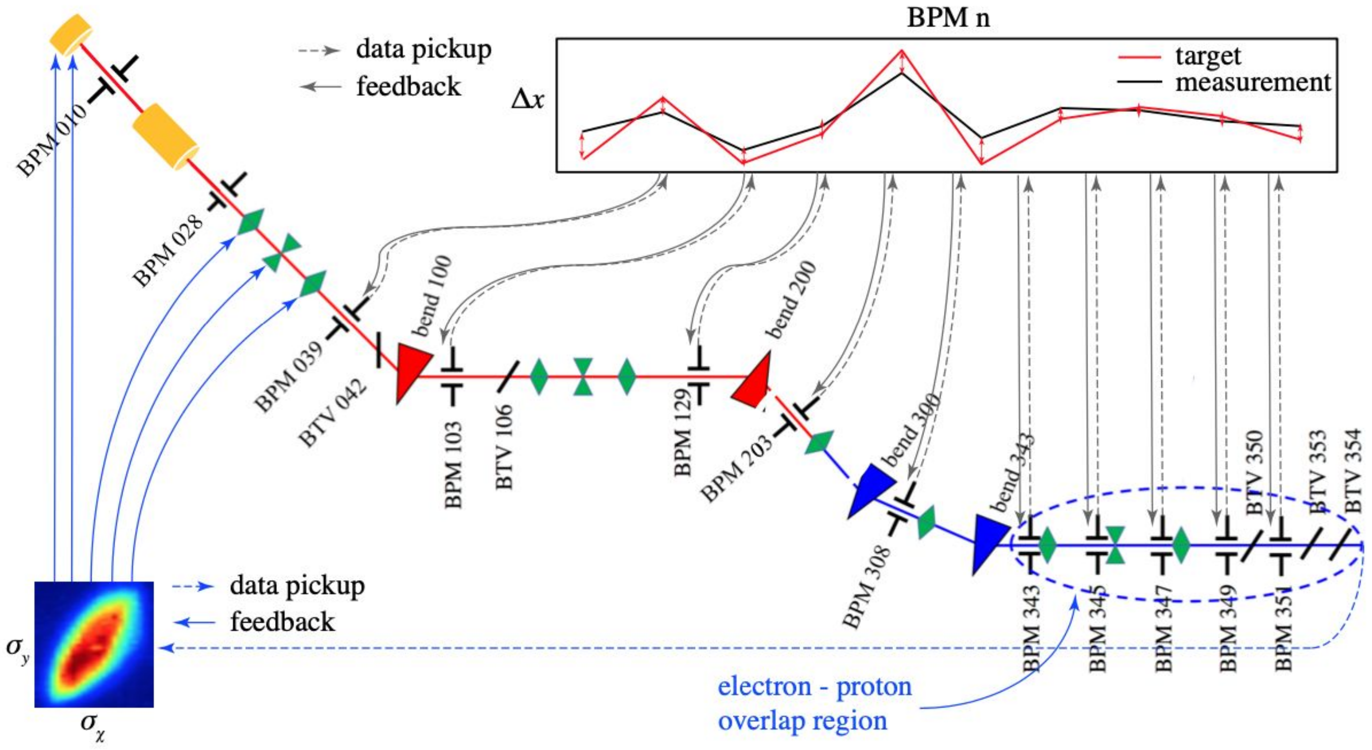


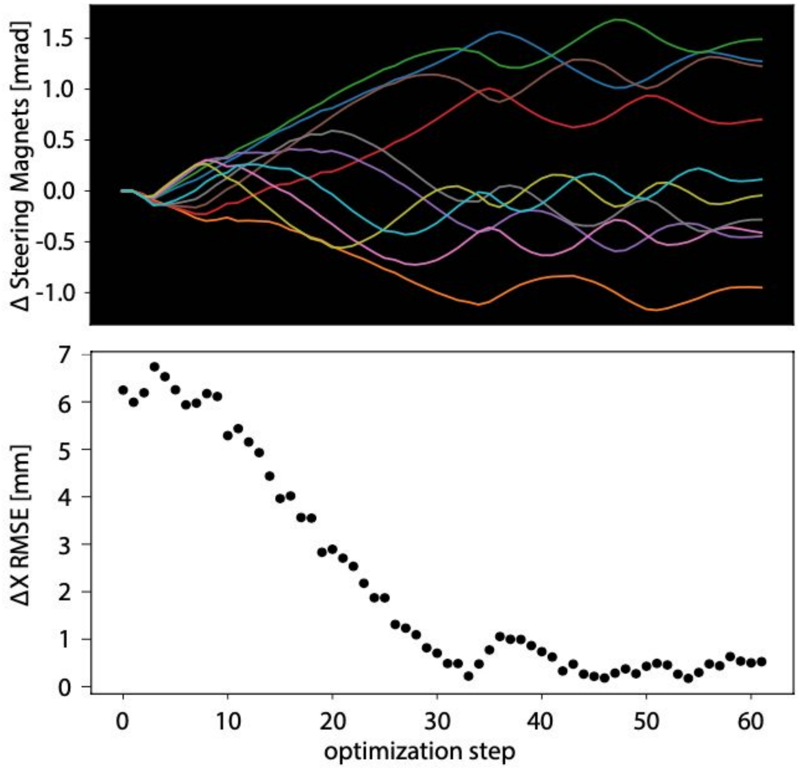
K. Deb et al

Example of MOO with scalarization: AWAKE electron beam line

Goal: maintain beam on a target trajectory while minimizing beam size

Uses multi-objective optimization with scalarization. In this case, no need to find the Pareto front!





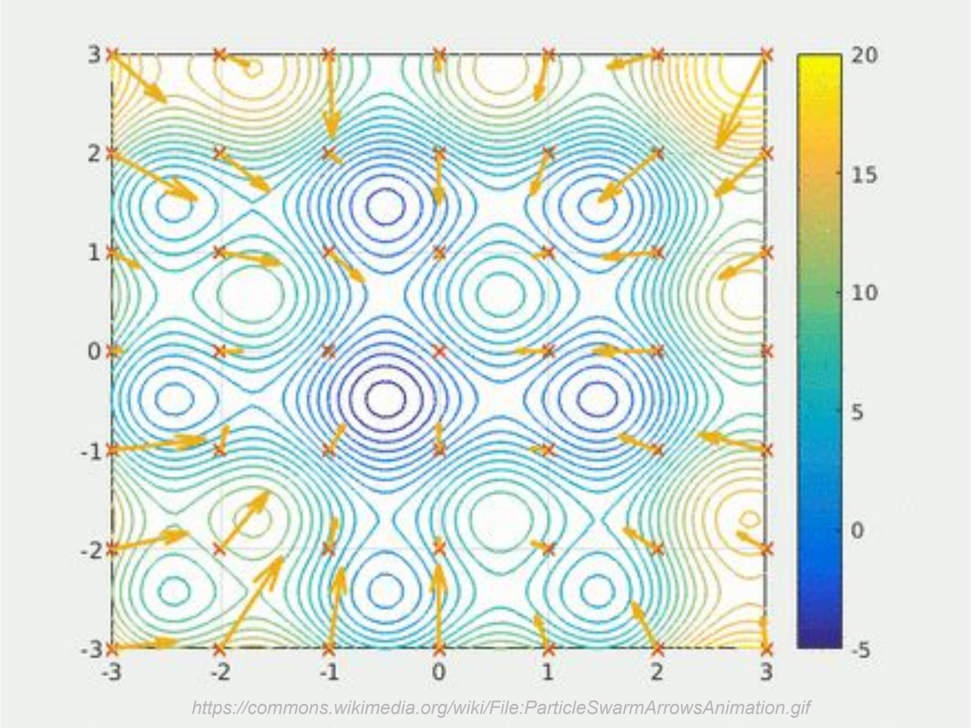
Scheinker et al., "Online Multi-Objective Particle Accelerator Optimization of the AWAKE Electron Beam Line for Simultaneous Emittance and Orbit Control" (2020)
https://arxiv.org/abs/2003.11155v1

Multi-Objective Optimization: Population-based Metaheuristics

“meta-heuristic” → uses general principles for moving toward good solutions vs. deterministic update rules

Population-based algorithms:

- Typically does not use gradients (though some variants do) → useful in situations where gradient is expensive to calculate or estimate
- Population approach: multiple solutions generated in a single iteration
  - Inherently parallel search
  - Important for multi-objective / multi-modal problems
- Uses stochastic operators (rather than deterministic ones)
- Evolutionary computation and swarm intelligence are two major categories



Example of particle swarm optimization (PSO)

Evolutionary Algorithms

Genetic algorithms

Differential evolution

Covariance matrix adaptation evolution strategy (CMA-ES)

Swarm Intelligence

Particle swarm optimization

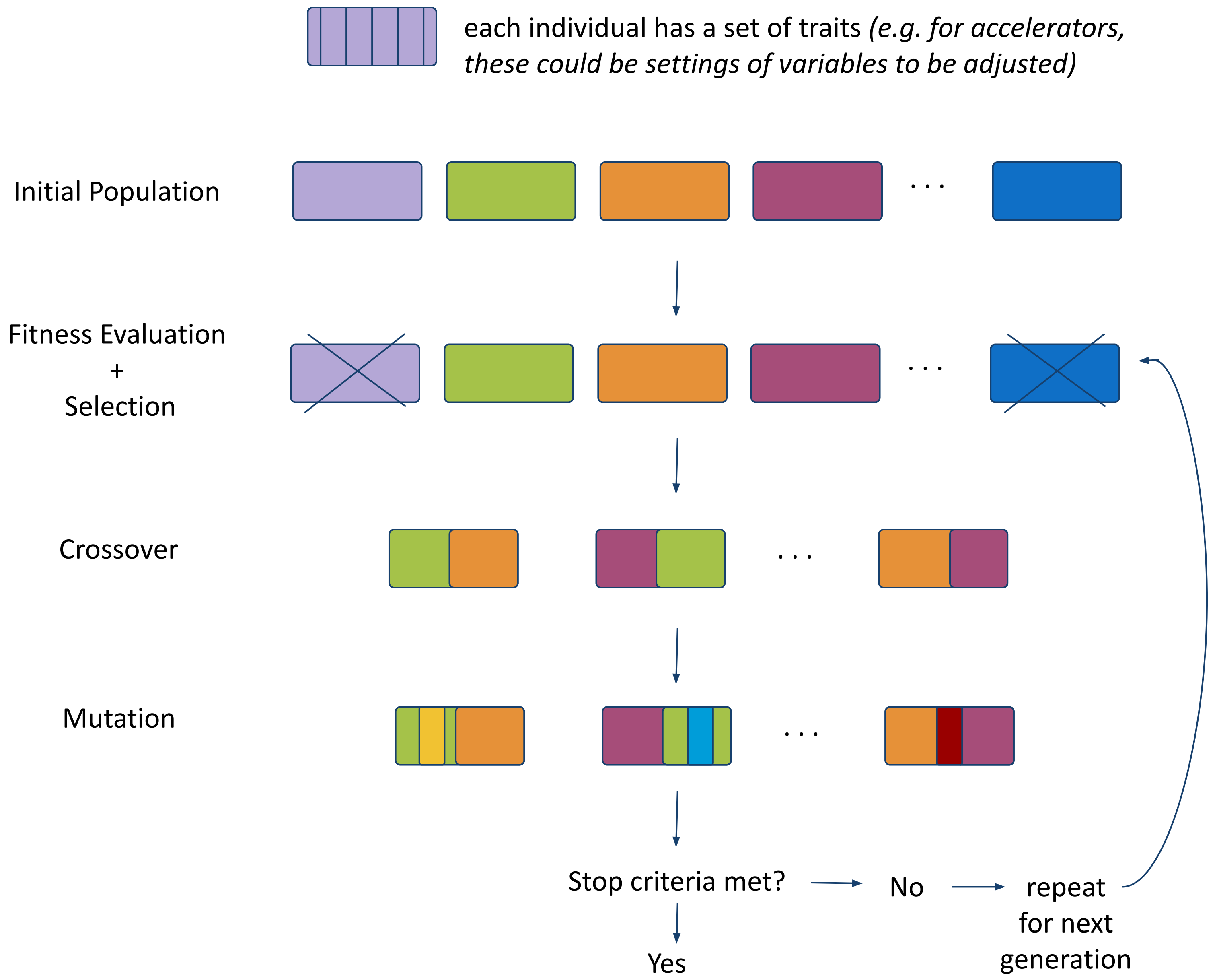
Ant colony optimization

Artificial bee colony optimization

Multi-Objective Optimization: Genetic Algorithms

Genetic algorithms are inspired by genetic evolution in biology:

1. Initial population of N individuals with a variety of traits *(e.g. random initial inputs)*
2. **Evaluation and selection** of individuals for the next population *(based on "fitness," or how good the solution is by some criteria)*
3. **Crossover** → combining the traits of selected parent individuals to produce "offspring" individuals
4. **Mutation** → altering some traits of individuals
5. **Repeat** steps 2-4 until stopping criteria is met *(often max number of "generations" or convergence criteria -- e.g. solution distance or hypervolume increase)*



Multi-Objective Optimization: Genetic Algorithms

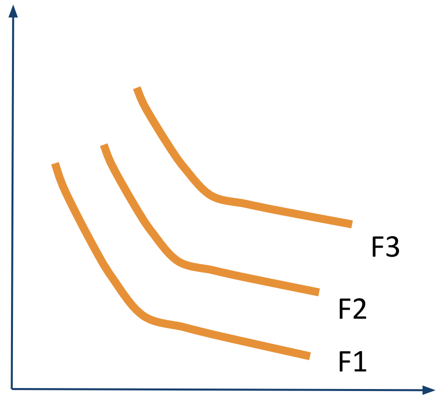
Fitness Evaluation and Selection:

- Many algorithms
- NSGA-II is a popular choice

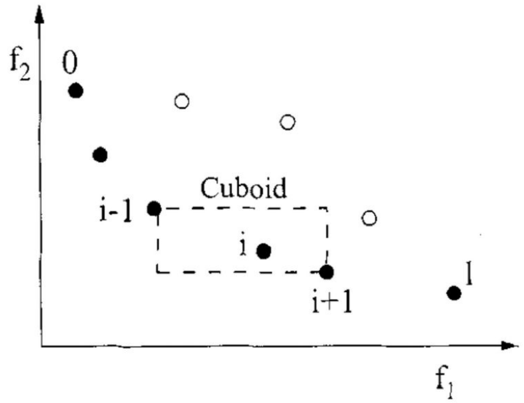
NSGA-II:

1. Non-dominated sorting on parent + offspring population, sorted into fronts
2. Create new population from front ranking
3. Sort according to crowding distance (how close solutions are to one another) → less-dense is preferable
4. Create new population from crowded-tournament selection (front rank and crowding distance)
5. Conduct crossover and mutation
6. Repeat

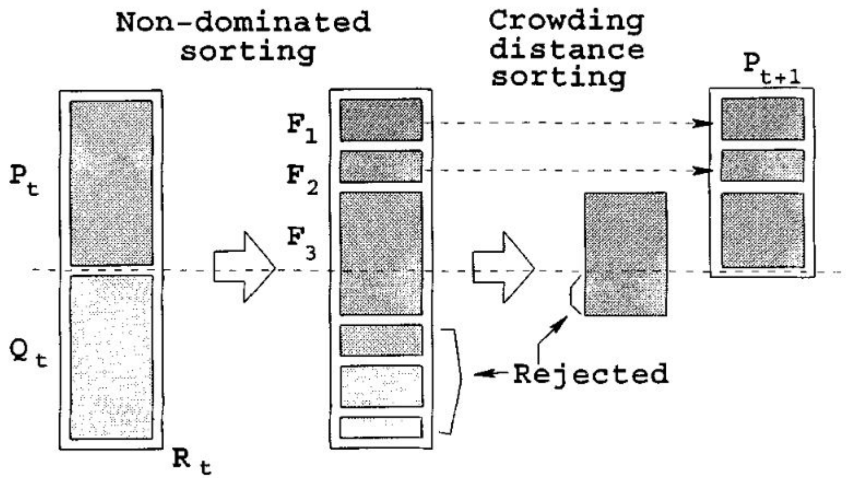
Preserves diversity, allows best solutions to propagate ("elitist principle")



ranking of fronts



crowding distance:
average side length of cuboid



original NSGA-II paper: K. Deb et al., https://ieeexplore.ieee.org/document/996017

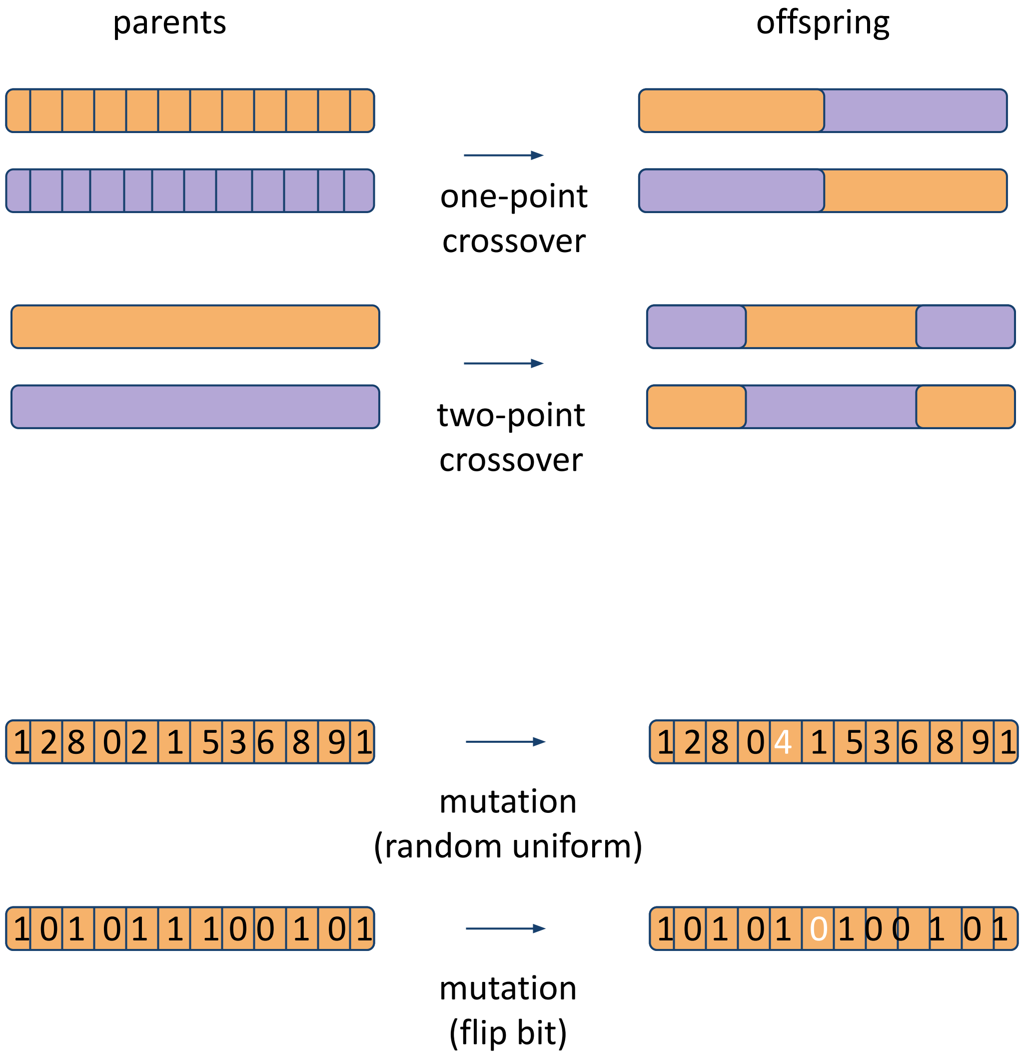
Multi-Objective Optimization: Genetic Algorithms

Crossover

- Mixes parent individuals' traits
- Can be at one point or multiple points
- *Aids convergence toward pareto-optimal population*
-

Mutation

- With some probability, modify traits of an individual
- Can adjust trait in many ways: binary, random uniform from range, etc
- *Increases diversity, and thus the probability of finding global optimum*
- *Can slow convergence (especially if hyperparameters set mutation too high)*



Multi-Objective Optimization: Genetic Algorithms

Commonly thought of as a global optimization method, but is not guaranteed to find the global optimum in practice:

- Population size, crossover, mutation are hyperparameters
- Lack of diversity can lead to “stalling” of the front

Mainly used offline for design optimization:

- Requires many function evaluations (sample inefficient)
- Computationally expensive: sorting in fitness evaluation and selection step can be expensive
- Leverages high performance computing resources to support parallel evaluation of solutions
- Sampling in GAs is not conducive to practical limitations in online optimization (e.g. desirable to move settings smoothly)

Typical use in accelerators:

- 2-3 competing objectives
- Only the most relevant variables
- Population sizes around 100 - 500 are usually sufficient
- *Generally use low fidelity simulations first, then re-start from relatively converged population with high fidelity sims*

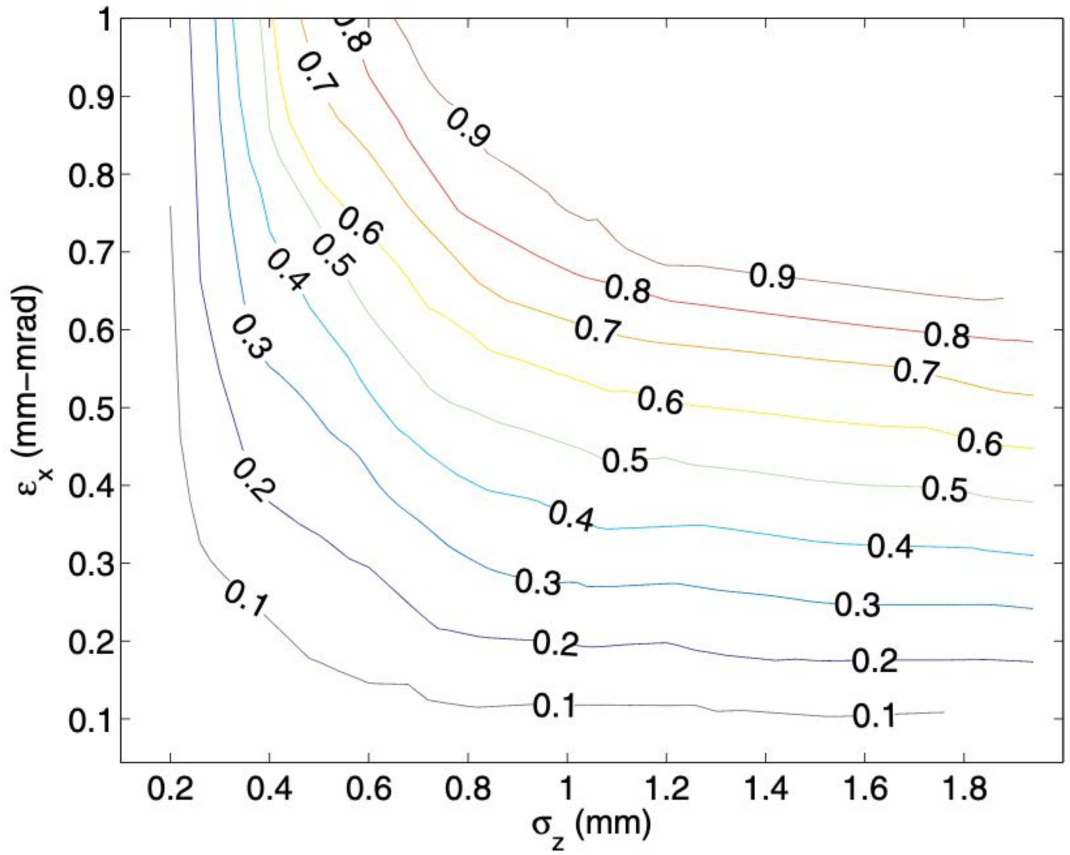
Example of MOO with GAs: Injector Optimization

“Multivariate optimization of a high brightness dc gun photoinjector”
Bazarov and Sinclair, 2005:
https://journals.aps.org/prab/pdf/10.1103/PhysRevSTAB.8.034202

→ *early application of GAs in the design of a photoinjector for an energy recovery linac*

> “In particular, we show how the use of multiobjective evolutionary algorithms helped us address the following questions: What is the optimal transverse and longitudinal shape of the laser pulse? How high should the gun voltage be for good injector performance? How does the thermal emittance of the photocathode affect the final emittance? What are the trade-offs between bunch length, emittance, and bunch charge? ”

22 variables: laser spot, duration, longitudinal and transverse profiles, cavity phases and amplitudes, element positions, etc.



Emittance and bunch length at different charges (nC), 10^5 simulations

Genetic Algorithms in Context

GAs are not explicitly using "learned" information from previous samples, but are inspired by nature and have the flavor of Artificial Intelligence

Artificial Intelligence (AI)

- *How to enable machines to exhibit aspects of "intelligence"*
- *knowledge, learning, planning, reasoning, perception*

Machine Learning (ML)

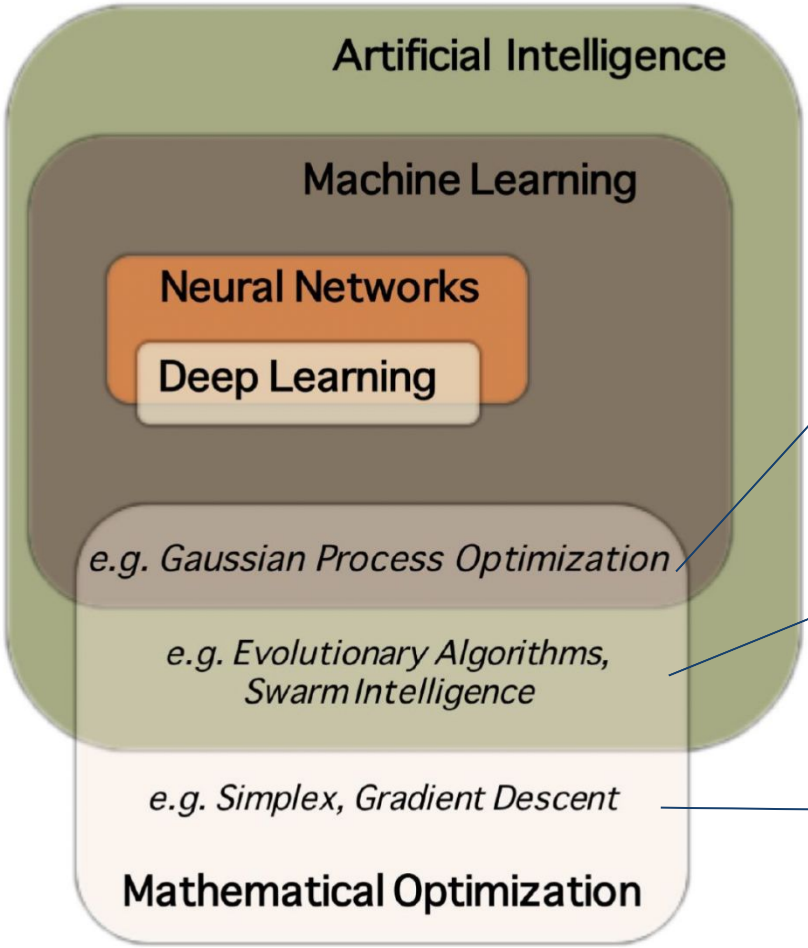
- *Use learned representations to complete tasks without being explicitly programmed*
- Tasks: Regression, Classification, Dimensionality Reduction, etc.

Neural Networks (NNs)

- *Class of ML structures that use many connected processing units to learn input/output maps (used to be called "connectionism")*

Deep Learning (DL)

- *Learning hierarchical representations*
- Right now, largely synonymous with deep (many-layered) NNs



In general:

Iteratively learn a system model to guide the search

Use some inspiration from intelligent behavior in nature, but do not learn system representations

Iterative methods that do not learn representations of the system

Summary

Needs for online optimization in accelerators:

- Sample-efficient (as few calls to machine as possible)
- Robust to noise
- Desirable not to numerically estimate the gradient

⟶ Methods like RCDS combine noise robustness with standard optimization methods (e.g. Powell's method) that choose step size and direction more efficiently than gradient descent

⟶ 2nd order methods (Newton and Quasi-Newton) in principle could be more sample-efficient in convergence but are more computationally expensive per iteration

Multi-objective optimization:

- Essential tool for examining parameter tradeoffs in accelerators
- Used for both optimization and characterization (+ extensive use in design optimization)

⟶ Genetic Algorithms are extensively used in the accelerator community

⟶ MOO with GAs is most often used offline due to computational expense and sample-inefficiently, rather than online

⟶ Scalarization can be used with any optimization algorithm for online use (but generally without finding Pareto front)

Teaser for next lectures: ML methods can help get around the limitations of these standard approaches!

Additional Resources

- Mitchell, Introduction to Genetic Algorithms: https://mitpress.mit.edu/books/introduction-genetic-algorithms
- Mitchell, Evolutionary Computation: https://melaniemitchell.me/PapersContent/ARES1999.pdf
- Fletcher, *Practical Methods of Optimization* (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8
- Schewchuk, Introduction to Conjugate Gradient without All the Agonizing Pain, https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf
- Introduction to the conjugate gradient method: https://folk.idi.ntnu.no/elster/tdt24/tdt24-f09/cg.pdf
- Numerical Recipes, 3rd Edition, https://g.co/kgs/k3ZxLB
- Chong and Zak, Introduction to Optimization, https://www.amazon.com/Introduction-Optimization-Edwin-K-Chong/dp/1118279018