LECTURE 2: OPTIMIZATION

COMPUTATIONAL INTELLIGENCE

VRIJE UNIVERSITEIT AMSTERDAM

OPTIMIZATION PROBLEMS

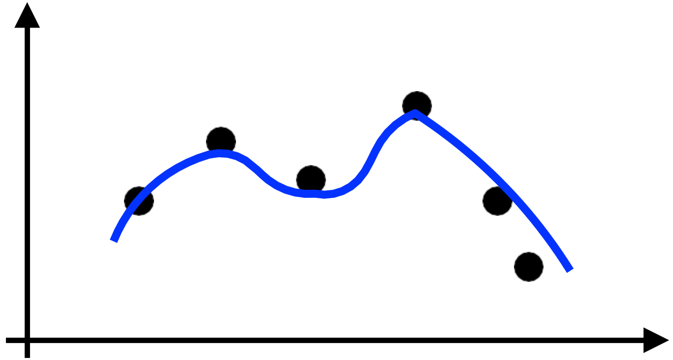
TRAVELING SALESMAN PROBLEM

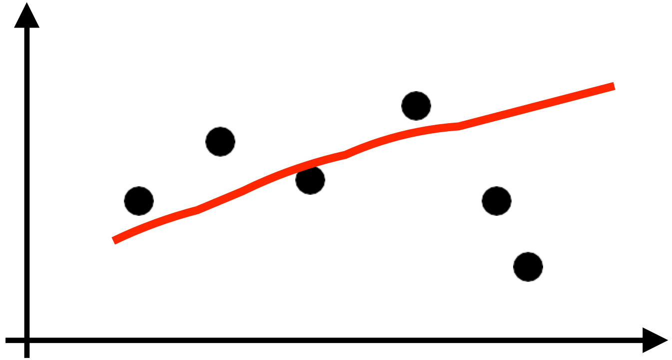
520km

745km

COVERAGE PROBLEM

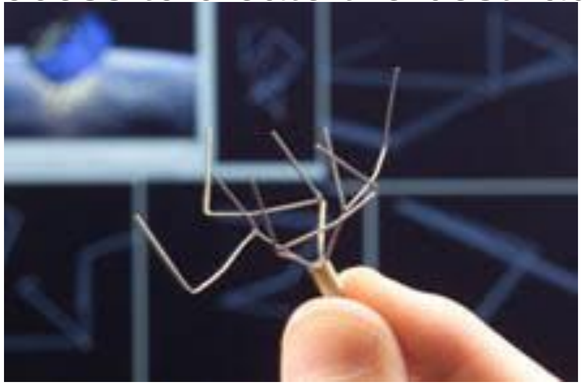
CURVE FITTING

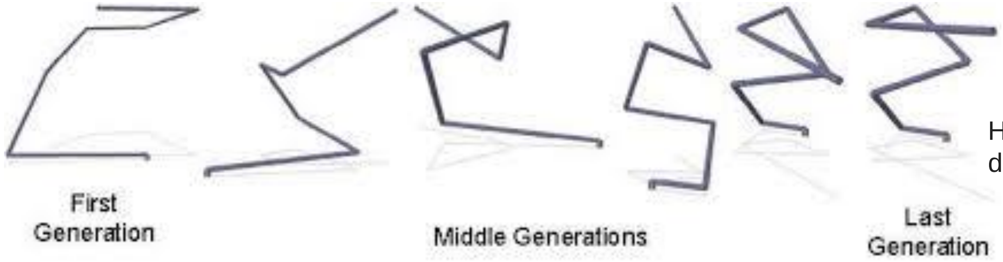




OPTIMIZATION OF ANTENNA SHAPE

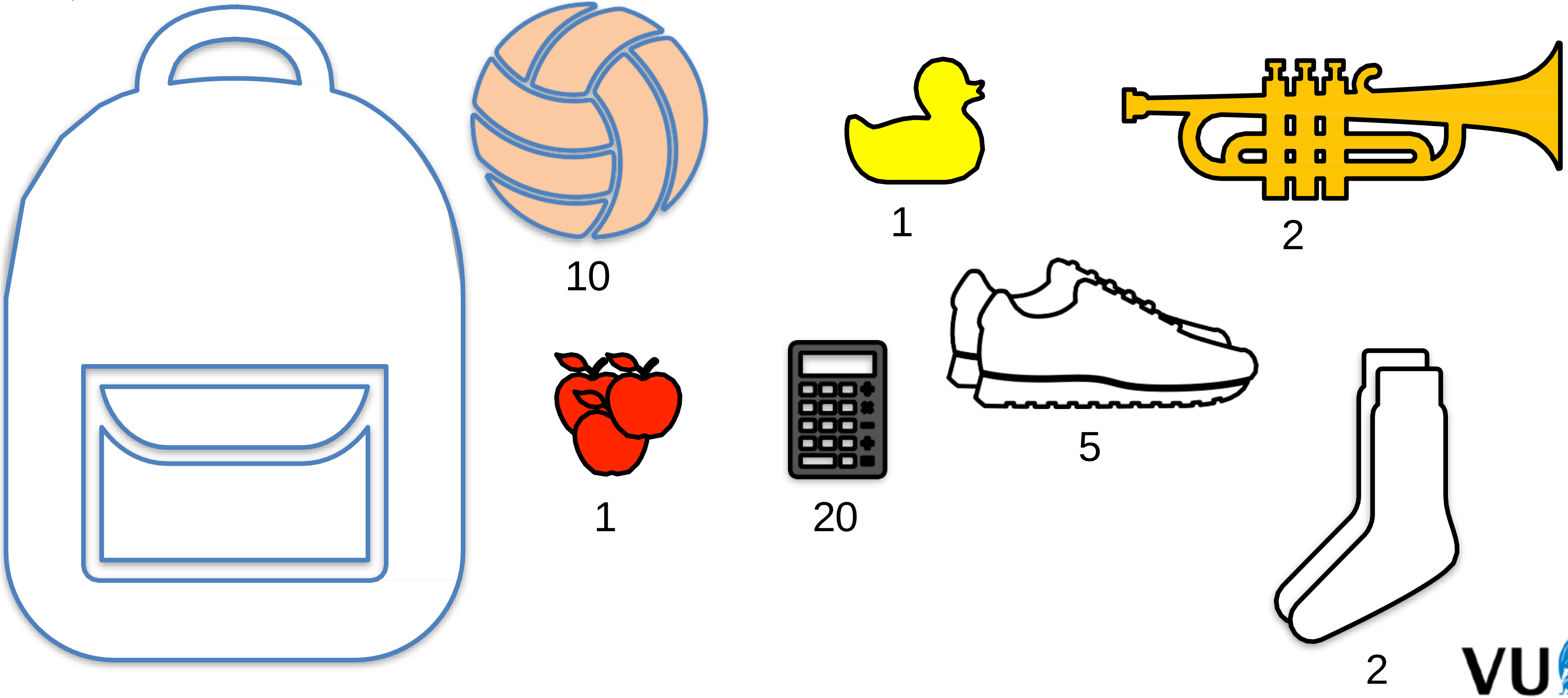
- The 2006 NASA ST5 spacecraft antenna that is found by an evolutionary process to create the best radiation pattern.



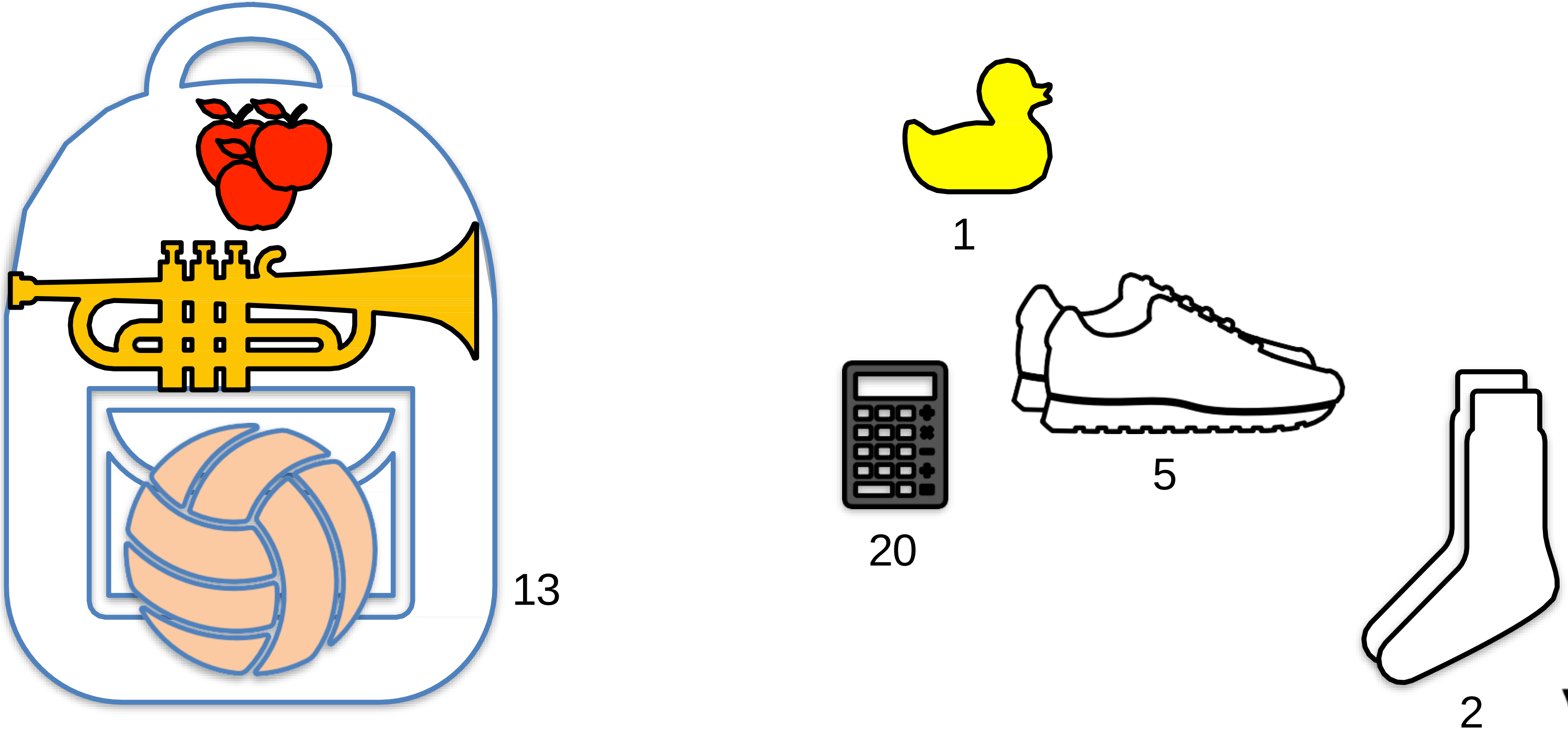


Hornby, G., Globus, A., Linden, D., & Lohn, J. (2006). Automated antenna design with evolutionary algorithms. In *Space 2006* (p. 7242).

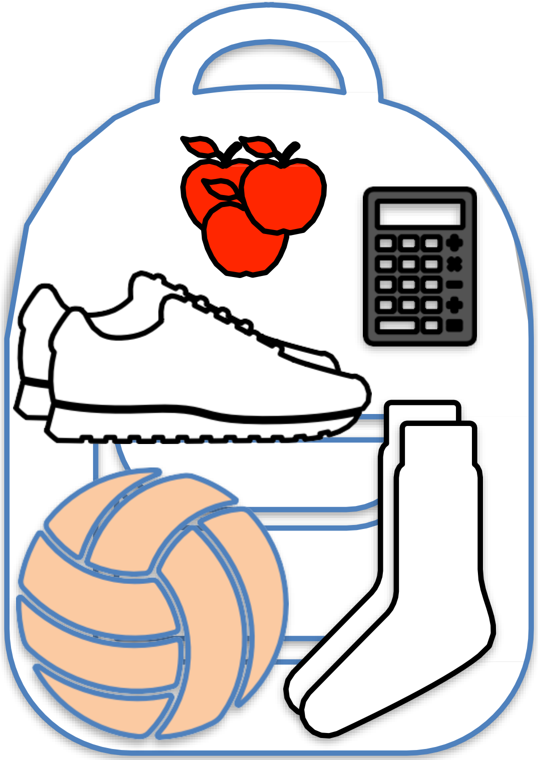
KNAPSACK PROBLEM



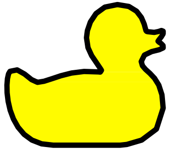
KNAPSACK PROBLEM



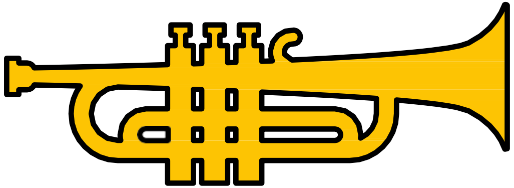
KNAPSACK PROBLEM



38



1



2

OPTIMIZATION

Find the **optimal solution** (min or max) from a given **set of possible solutions** $\mathbb{Y}$ that minimizes/maximizes given **objective function** $f(x)$.

$$\min_{x \in \mathbb{X}} f(x)$$

$$\text{s.t. } x \in \mathbb{Y} \subseteq \mathbb{X}$$

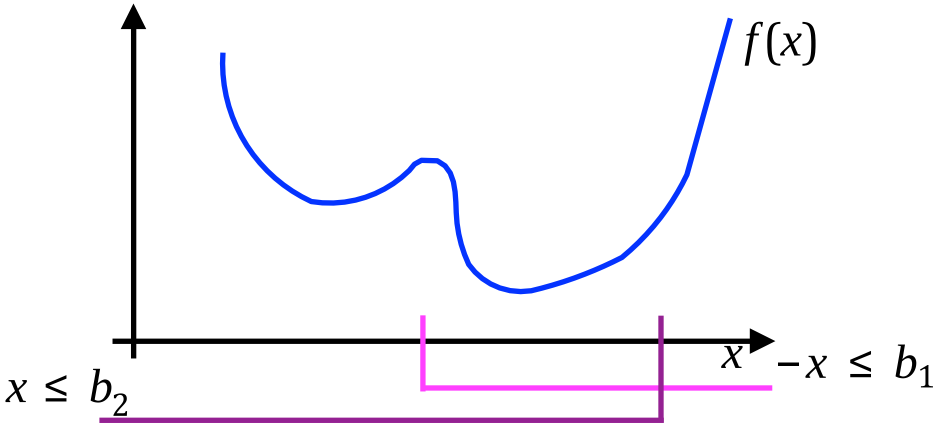
OPTIMIZATION

$x \in \mathbb{X}$ - **optimization variables** / parameters / unknowns

$f : \mathbb{X} \to \mathbb{R}$ - **objective function**

$c_i : \mathbb{X} \to \mathbb{R}$ - **constraint functions**

$\mathbb{X}$ - **search space**

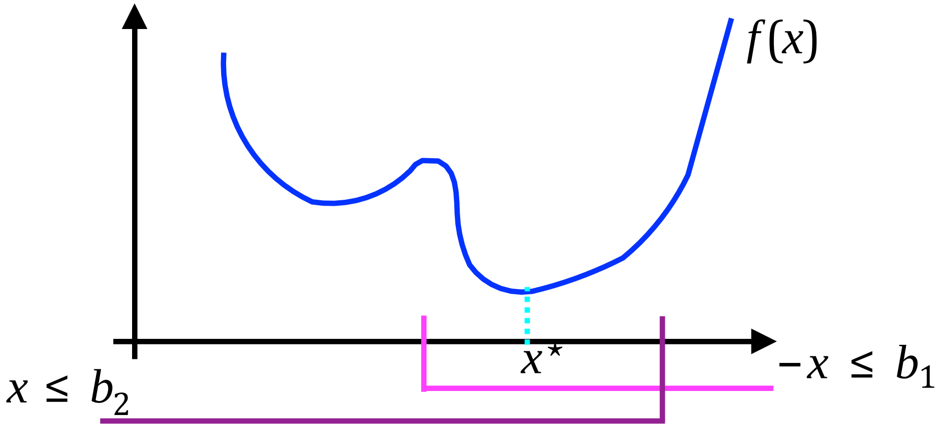


OPTIMIZATION

Formally:

$$\min_{x \in \mathbb{X}} \; f(x)$$

$$\text{s.t. } \forall_i \; c_i(x) \leq b_i$$

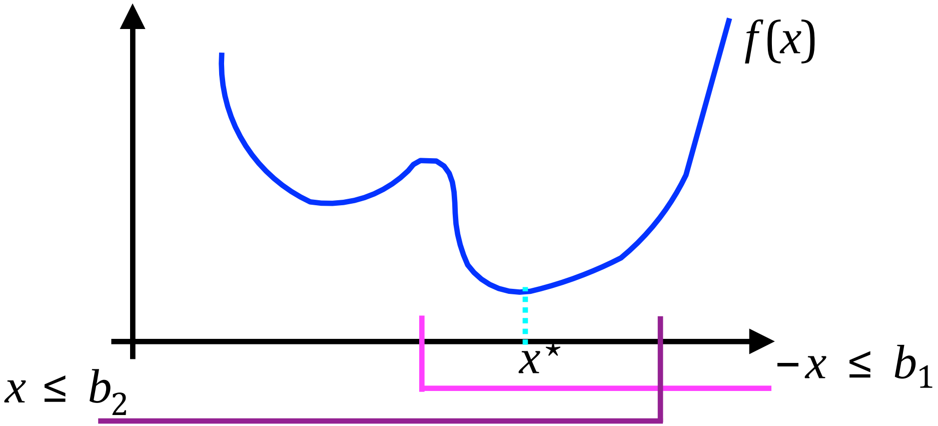


OPTIMIZATION

Formally:

$$\min_{x \in \mathbb{X}} f(x)$$

$$\text{s.t. } \forall_i \; c_i(x) \le b_i$$



Remarks:

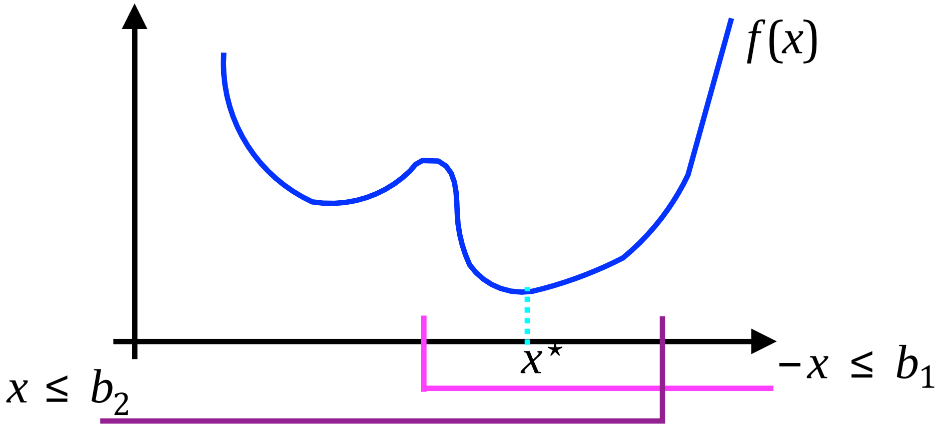
1) $\min f(x) = \max \{-f(x)\}$

OPTIMIZATION

Formally:

$$\min_{x \in \mathbb{X}} f(x)$$

$$\text{s.t. } \forall_i \; c_i(x) \leq b_i$$



Remarks:

1) $\min f(x) = \max \{-f(x)\}$

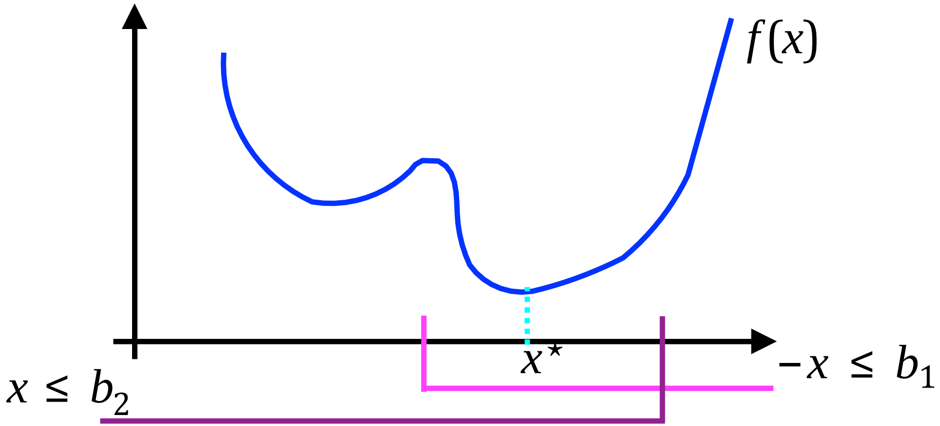
2) E.g., $\mathbb{X} = \mathbb{R}^D$, $\mathbb{X} = \{0,1\}^D$

OPTIMIZATION

Formally:

$$\min_{x \in \mathbb{X}} f(x)$$

$$\text{s.t. } \forall_i \; c_i(x) \leq b_i$$



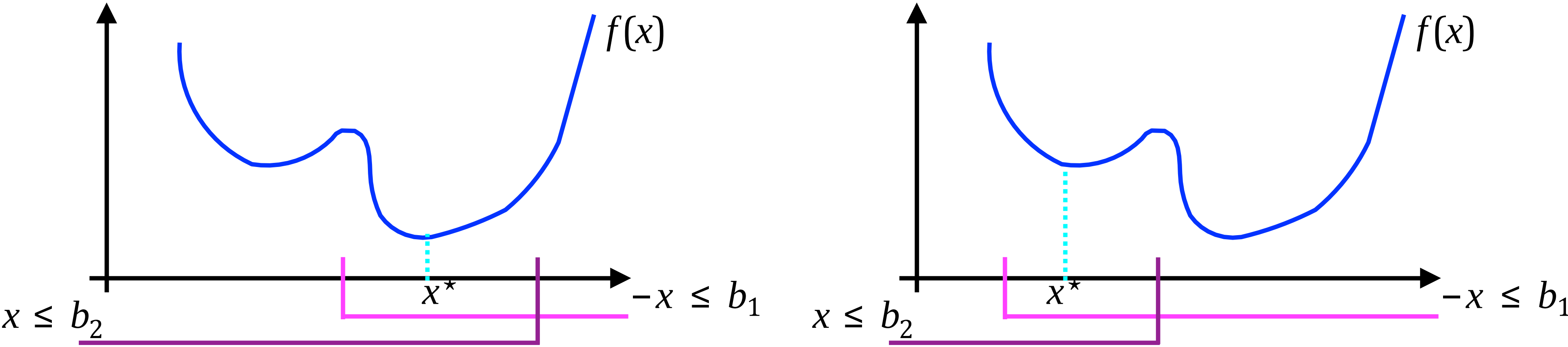
Remarks:

1) $\min f(x) = \max \{-f(x)\}$

2) E.g., $\mathbb{X} = \mathbb{R}^D$, $\mathbb{X} = \{0,1\}^D$

3) Optimal solution: $\forall_{x \in \mathbb{X}} \; f(x^{\star}) \leq f(x)$ and $\forall_i \; c_i(x^{\star}) \leq b_i$

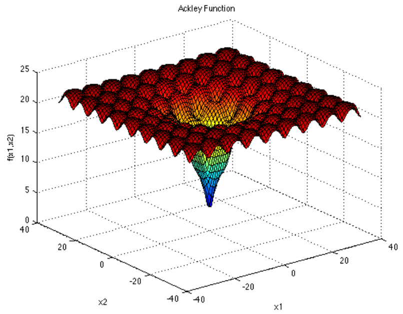
OPTIMIZATION

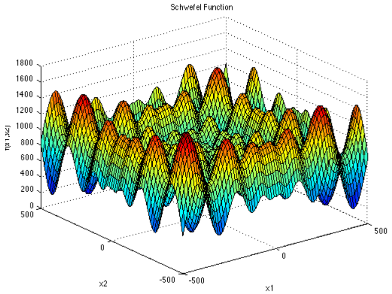


CONTINOUS OPTIMIZATION (BECHMARK FUNCTIONS)

Many Local Minima

1. Ackley Function
2. Bukin Function N. 6
3. Cross-in-Tray Function
4. Drop-Wave Function
5. Eggholder Function
6. Gramacy & Lee (2012) Function
7. Griewank Function
8. Holder Table Function
9. Langermann Function
10. Levy Function
11. Levy Function N. 13
12. Rastrigin Function
13. Schaffer Function N. 2
14. Schaffer Function N. 4
15. Schwefel Function
16. Shubert Function





Bowl-Shaped

17. Bohachevsky Functions
18. Perm Function 0, d, β
19. Rotated Hyper-Ellipsoid Function
20. Sphere Function
21. Sum of Different Powers Function
22. Sum Squares Function
23. Trid Function

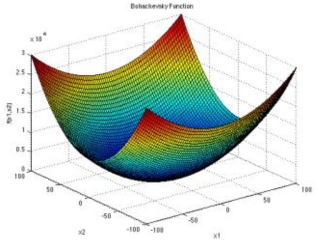
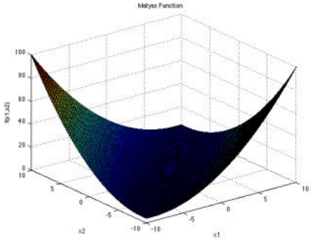


Plate-Shaped

24. Booth Function
25. Matyas Function
26. McCormick Function
27. Power Sum Function
28. Zakharov Function

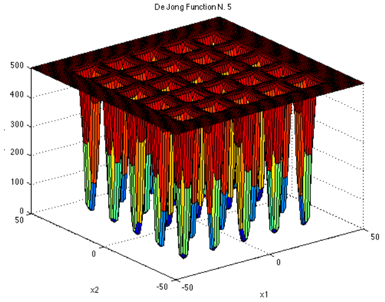


Valley-Shaped

29. Three-Hump Camel Function
30. Six-Hump Camel Function
31. Dixon-Price Function
32. Rosenbrock Function

Steep Ridges/Drops

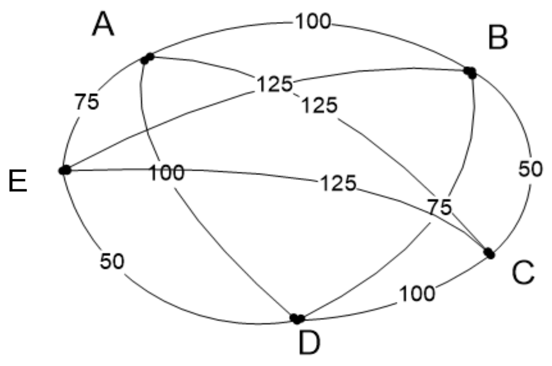
33. De Jong Function N. 5
34. Easom Function
35. Michalewicz Function

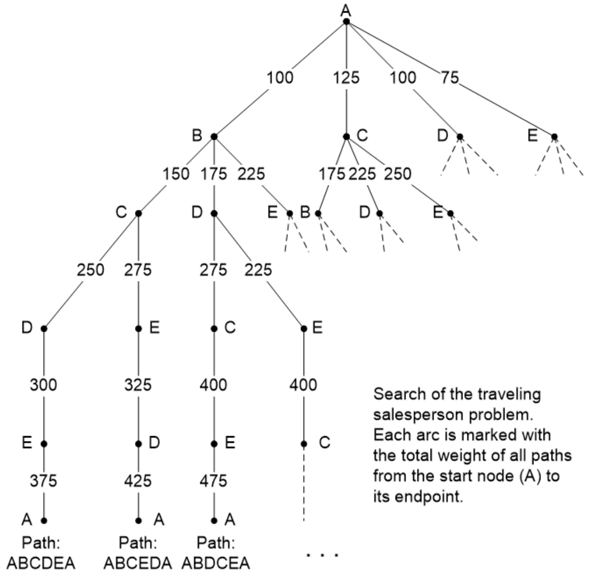


Source: https://www.sfu.ca/~ssurjano/optimization.html

DISCRETE OPTIMIZATION

Traveling Salesperson Problem (TSP)



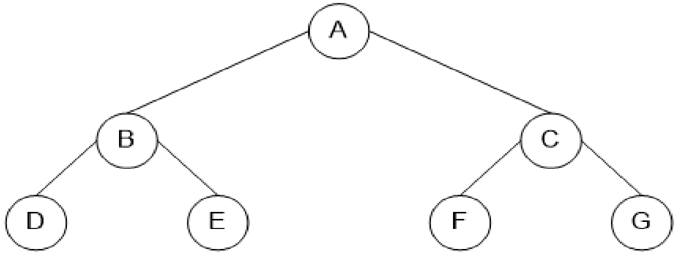


Search of the traveling salesperson problem. Each arc is marked with the total weight of all paths from the start node (A) to its endpoint.

- A graph with n vertices have **(n-1)!** paths

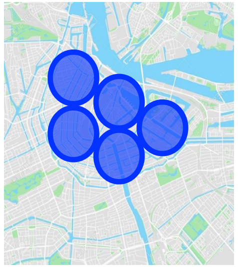
BLIND vs HEURISTIC SEARCH

- Blind search - no knowledge of the problem
  - Depth first search (DFS): A, B, D, E, C, F, G
  - Breadth first search (BFS): A, B, C, D, E, F, G
- Heuristic search - employ estimates of closeness to a goal
  - i.e. Beam Search, Best Search, Hill Climbing, etc..



REPRESENTATION

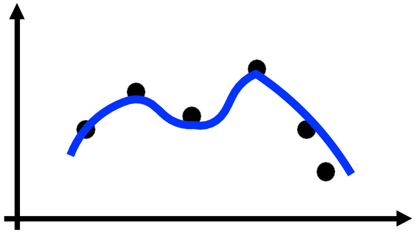
Coverage



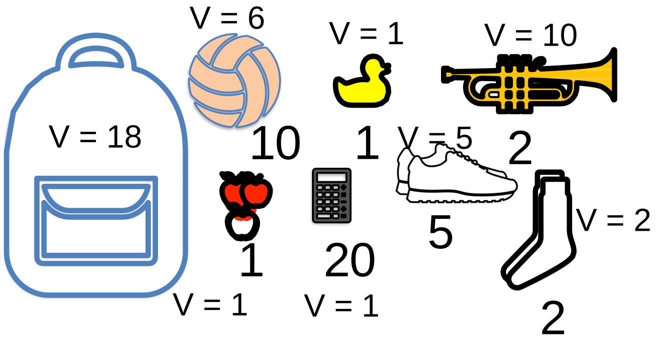
Design

TSP

Curve fitting



Knapsack



TAXONOMY OF OPTIMIZATION PROBLEMS

Convex / Non-convex

Constrained / Unconstrained

Optimization

Deterministic / Stochastic

Continuous / Discrete

Global / Local

OPTIMIZATION METHODS

TAXONOMY OF OPTIMIZATION METHODS

Optimization methods

Derivative-free methods (0th order methods)

Gradient-based methods (1st order methods)

Hessian-based methods (2nd order methods)

TAXONOMY OF OPTIMIZATION METHODS

Optimization methods

Derivative-free methods (0th order methods)	Gradient-based methods (1st order methods)	Hessian-based methods (2nd order methods)
Hill climbing Iterated local search …	Gradient descent ADAM …	Newton's method …

TAXONOMY OF OPTIMIZATION METHODS

Optimization methods

Derivative-free methods (0th order methods)	Gradient-based methods (1st order methods)	Hessian-based methods (2nd order methods)
Hill climbing	Gradient descent	Newton's method
Iterated local search	ADAM	…
…	…	

ITERATIVE OPTIMIZATION METHODS

- We are interested in **numerical** methods.
- Therefore, we consider **iterative optimization methods**.

ITERATIVE OPTIMIZATION METHODS

- We are interested in **numerical** methods.
- Therefore, we consider **iterative optimization methods**.
- The general procedure:

    1. Init x_0.
    2. Find new solution that fulfills constraints:

$$x_{t+1} = \Psi(x_t; f, \{c_i\}, \{x_0, \dots, x_{t-1}\})$$

    3. Go to 2 until STOP.

ITERATIVE OPTIMIZATION METHODS

- We are interested in **numerical** methods.
- Therefore, we consider **iterative optimization methods**.
- The general procedure:

  1. Init x_0.
  2. Find new solution that fulfills constraints:

$$x_{t+1} = \Psi(x_t; f, \{c_i\}, \{x_0, \ldots, x_{t-1}\})$$

  3. Go to 2 until STOP.

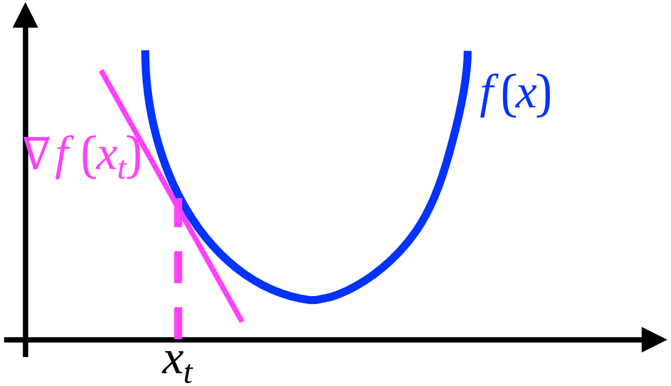
Crucial part is the formulation of $\Psi(\cdot)$.

GRADIENT DESCENT

- Derivatives of the objective function with respect to optimization variables = **gradient**:

$$\nabla f(x_t) = \begin{bmatrix} \frac{\partial f(x_t)}{\partial x_1} \\ \cdots \\ \frac{\partial f(x_t)}{\partial x_D} \end{bmatrix}$$

- Intuition behind the gradient:

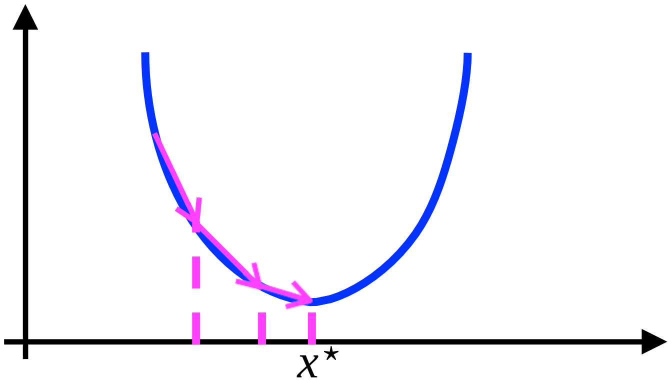


GRADIENT DESCENT

- We can use the information about the gradient to find new solutions

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

where $\alpha_t > 0$ is called the **step size** s.t. $\lim_{t \to \infty} \alpha_t = 0$.



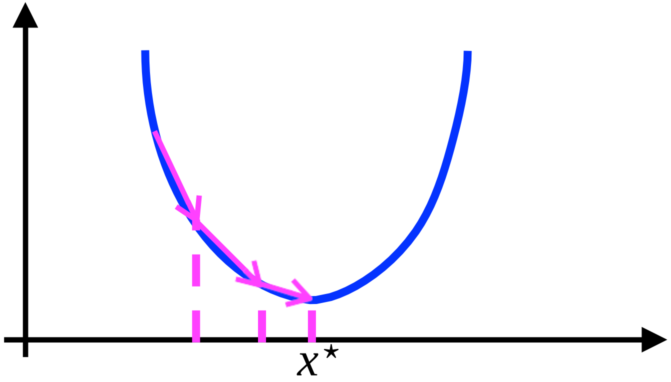
GRADIENT DESCENT

- We can use the information about the gradient to find new solutions

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

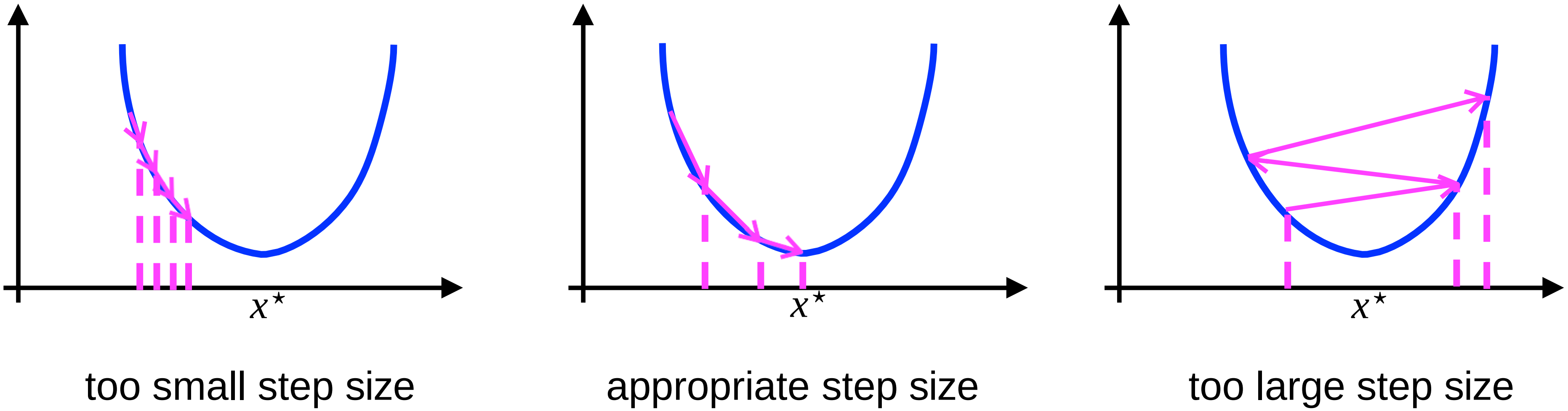
where $\alpha_t > 0$ is called the **step size** s.t. $\lim_{t\to\infty} \alpha_t = 0$.

($\alpha_t \equiv \alpha$ works fine in practice)

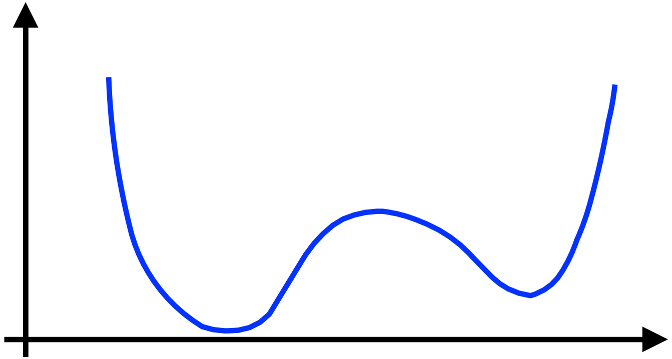


GRADIENT DESCENT

- Choosing the value of α_t is extremely important!



GRADIENT DESCENT



- It can stuck in **local minima**.
  Possible solution: run multiple times.

- It works only for **continuous** spaces.

- It works only for **smooth** & **differentiable** functions.

- BUT: It’s **easy** to use and it could be **automated** (AutoGrad)!

DERIVATIVE-FREE METHODS

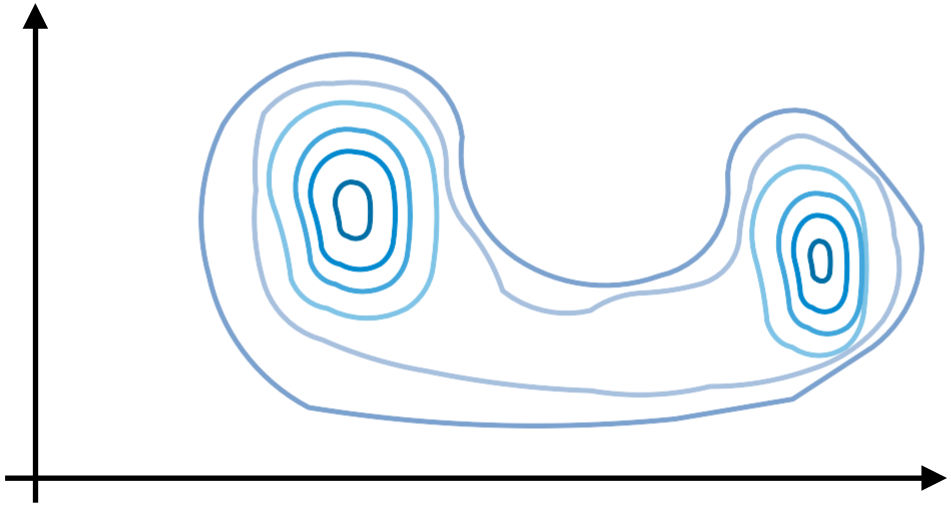
- Very often we **cannot** calculate gradients:
  - objective function is **non-differentiable**;
  - objective function is unknown, but we can query it (**blackbox**);
  - the search space is **discrete**.

- The question is: How to formulate $\Psi(\cdot)$ now?

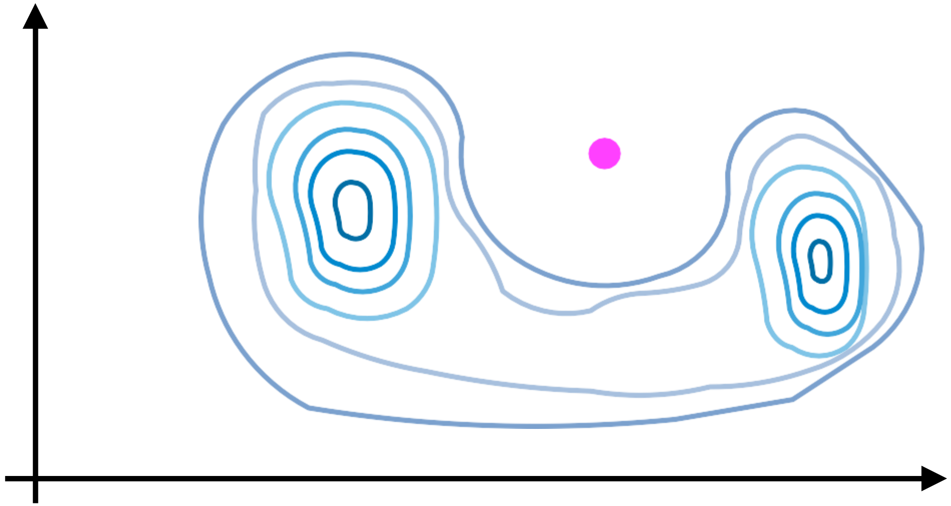
RANDOM SEARCH

- Pick a solution at **random**!
- It works pretty well on low-dimensional problems.
- BUT:
  - it **fails** in high-dimensional problems (curse of dimensionality);
  - requires **a lot** of evaluations of the objective function;
  - it does **not** take into account current and past solutions.

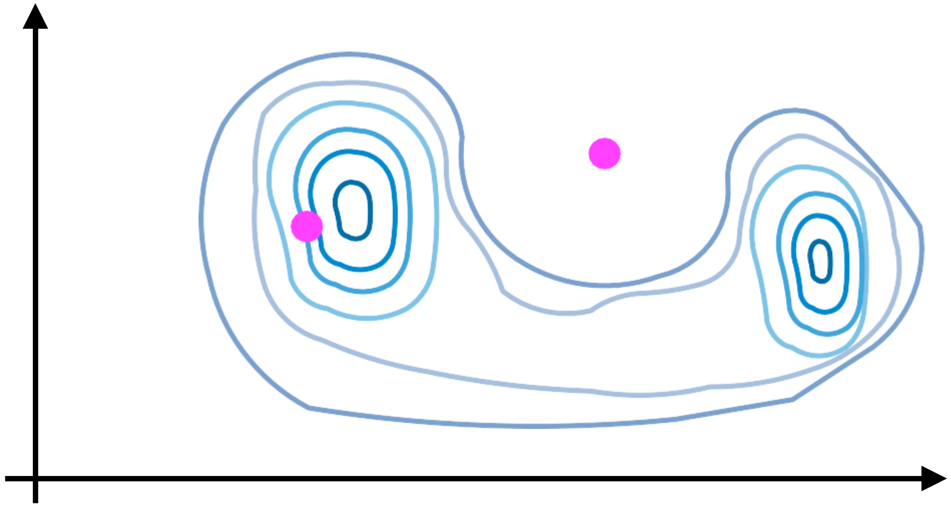
RANDOM SEARCH



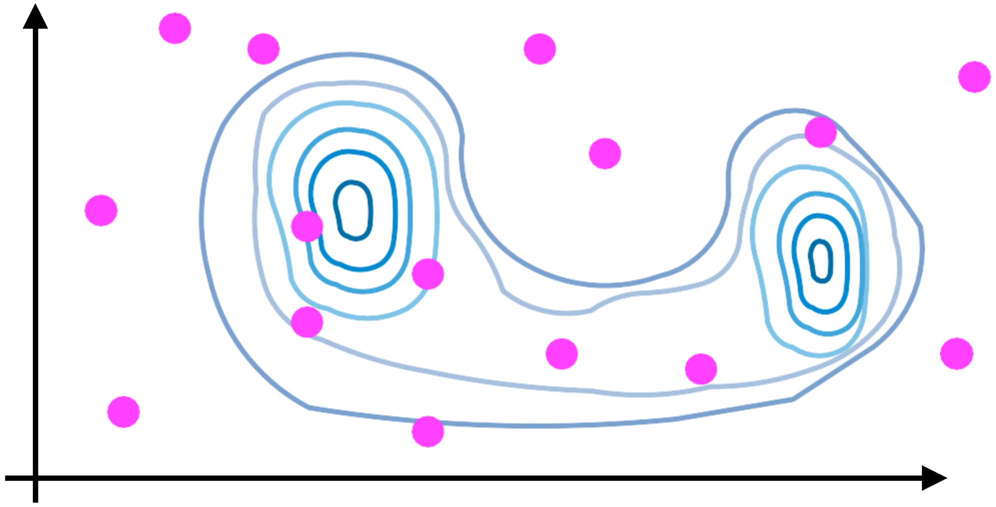
RANDOM SEARCH



RANDOM SEARCH



RANDOM SEARCH



(NEGATIVE) HILL CLIMBING

- The update step $\Psi(\cdot)$:
  1. Pick the *d*-th variable.
  2. Change its value.

     E.g.: use derivative information, replace value with other dimension.
  3. Check whether the objective has lower value.

     If yes, accept the new value.
  4. Go to 1 until STOP.

Wanderer above the Sea of Fog, Painting by Caspar David Friedrich, 1818

(NEGATIVE) HILL CLIMBING

- The update step $\Psi(\cdot)$:
  1. Pick the *d*-th variable.
  2. Change its value.

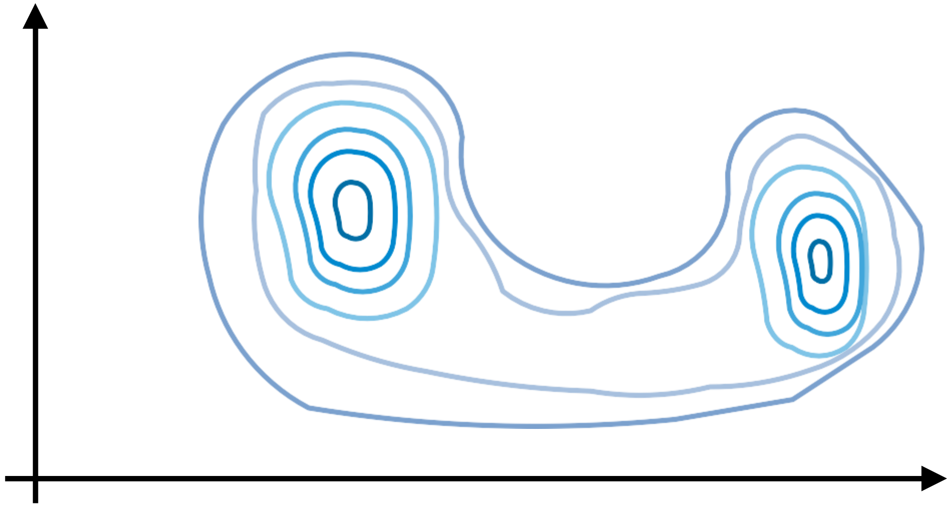
     Coordinate descent

     E.g. use derivative information, replace value with other dimension.
  3. Check whether the objective has lower value.

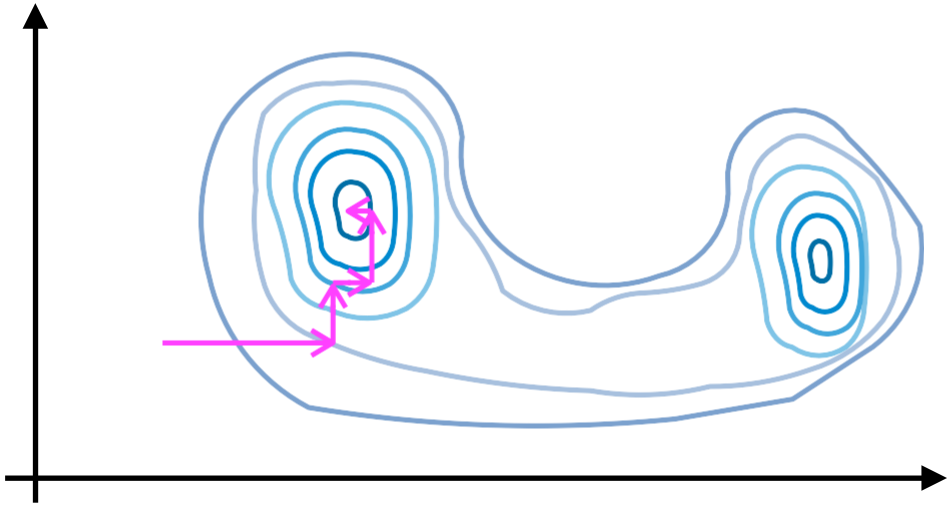
     If yes, accept the new value.
  4. Go to 1 until STOP.

Wanderer above the Sea of Fog, Painting by Caspar David Friedrich, 1818

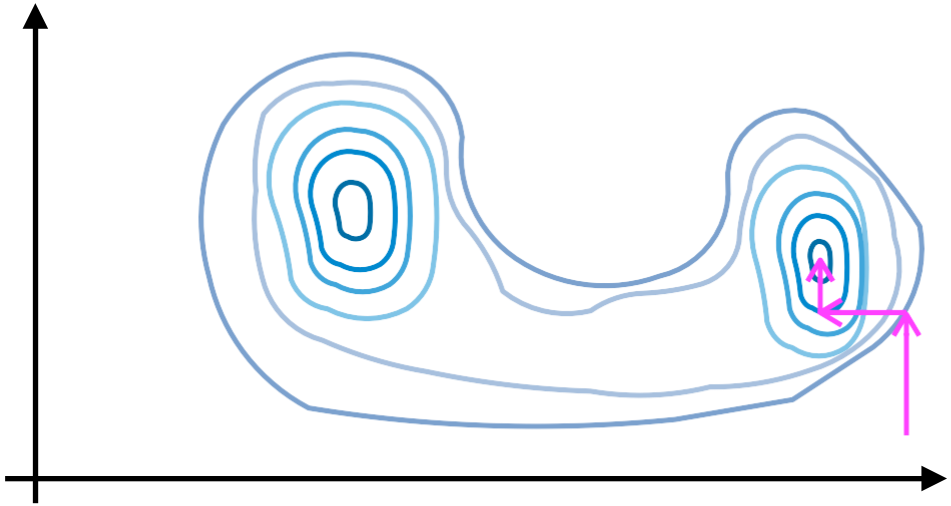
(NEGATIVE) HILL CLIMBING



(NEGATIVE) HILL CLIMBING



(NEGATIVE) HILL CLIMBING



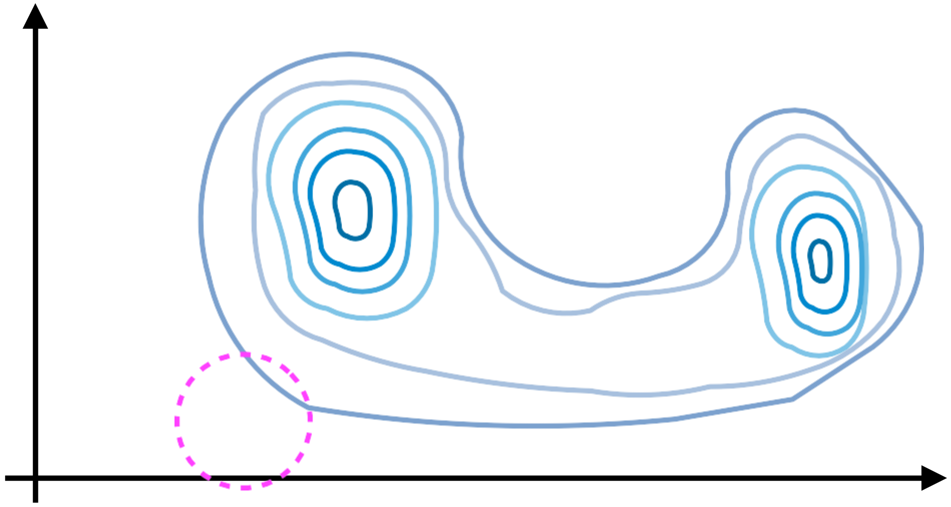
LOCAL SEARCH

- The update step $\Psi(\cdot)$:
  1. (Search) Find the best solution for the neighborhood of the current best solution.
  2. Set the new solution as the best current solution.
  3. Go to 1 until STOP.

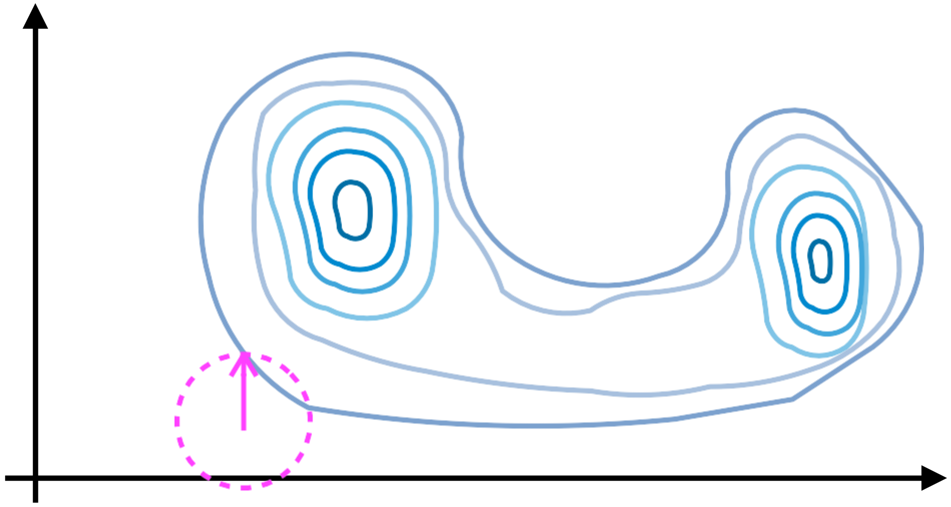
LOCAL SEARCH

- The update step $\Psi(\cdot)$:
  1. (Search) Find the best solution for the neighborhood of the current best solution.
  2. Set the new solution as the best current solution.
  3. Go to 1 until STOP.
- In practice, we need to search effectively the neighborhood, e.g.:
  - choose randomly some points;
  - pick the best performing value of a single dimension.

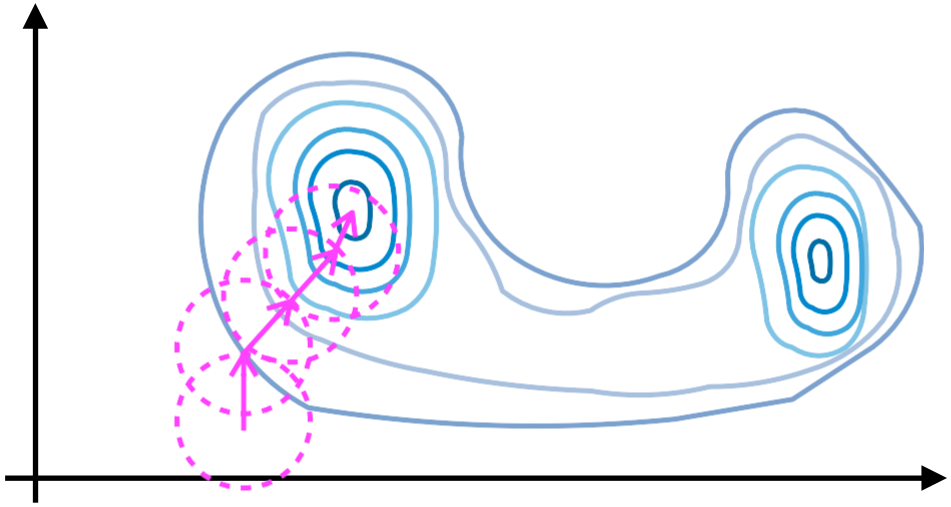
LOCAL SEARCH



LOCAL SEARCH



LOCAL SEARCH



ITERATED LOCAL SEARCH (GLOBAL SEARCH)

- The update step $\Psi(\cdot)$:
  1. (Search) Find the best solution for the neighborhood of the current best solution.
  2. Set the new solution as the best current solution.
  3. Go to 1 until STOP.
  4. $(\mathrm{Perturb})$ Perturb the best solution found in previous steps.
  5. Go to 1 until STOP.

ITERATED LOCAL SEARCH (GLOBAL SEARCH)

- The update step $\Psi(\cdot)$:
  1. (Search) Find the best solution for the neighborhood of the current best solution.
  2. Set the new solution as the best current solution.
  3. Go to 1 until STOP.
  4. (Perturb) Perturb the best solution found in previous steps.
  5. Go to 1 until STOP.

How to perturb? Starting from multiple inits could give similar results.

LOCAL vs GLOBAL SEARCH

- Two components to any search algorithm: **exploitation** and **exploration**.
- ***Exploitation*** - Once one good solution is found, examine its neighbors to determine if a better solution is present (good solutions are likely to lie close to one another).
- ***Exploration*** - Often a better solution may lie in an unexplored region of the state space, so do not remain in one small region.
- An ideal search algorithm must strike the <u>proper balance between these two conflicting strategies</u>.

DERIVATIVE-FREE METHODS

- **Easy** to implement.
- Require (almost) **no extra knowledge** (e.g., calculus).
- Work both for **continuous** and **discrete** problems.
- BUT they could be **very slow**, require **many evaluations**, **no guarantees**.
- Other approaches:
  - **Population-based methods**: take advantage of multiple solutions.
  - **Bayesian optimization**: surrogate model + uncertainty + optimization

DERIVATIVE-FREE METHODS

- **Easy** to implement.
- Require (almost) **no extra knowledge** (e.g., calculus).
- Work both for **continuous** and **discrete** problems.
- BUT they could be **very slow**, require **many evaluations**, **no guarantees**.
- Other approaches:
  - **Population-based methods**: take advantage of multiple solutions.
  - **Bayesian optimization**: surrogate model + uncertainty + optimization

May require gradients

Thank you!

VU VRIJE UNIVERSITEIT AMSTERDAM

EXTRA READING

Nocedal & Wright, “Numerical Optimization”, Springer

Audet & Hare, “Derivative-Free and Blackbox Optimization”, Springer