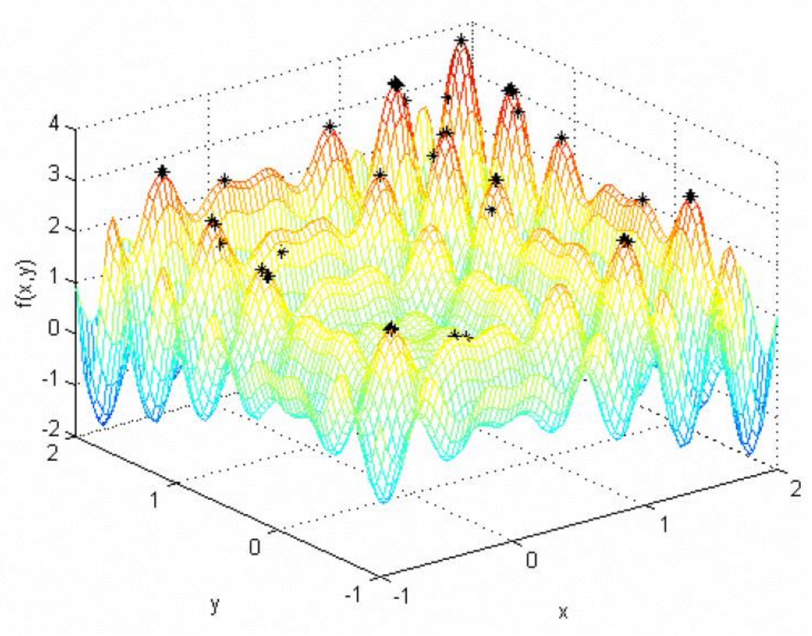
University of Ljubljana, Faculty of Computer and Information Science

Optimization and local search



Prof Dr Marko Robnik-Šikonja

Analysis of Algorithms and Heuristic Problem Solving
Version 2025

Search

- search is a basic problem solving mechanism
- many algorithms can be viewed as search algorithms
- problem states
- state space (reachable states form a graph, often searched as a tree)

$$\mathbf{S} = \{S; S_Z \xrightarrow{*} S\}$$

- connections between states
- a neighborhood generator N(S)

State space representation

- State space: $\boldsymbol{S} = \left\{ S; S_0 \xrightarrow{*} \mathrm{S} \right\}$
- Starting state: S_0
- Quality of a state: $q(S)$
- Global optimum: $S_{best} = \arg\min_{S \in \boldsymbol{S}} q(S)$
- Local optimum: $\boldsymbol{S}_{local} = \{S;\ \forall S \rightarrow S' : q(S) \leq q(S')\}$

Properties of local search

- Local search, LS;
- Local optimization, LO
- LS starts in a randomly generated state (solution) and tries to optimize it using local transformations
- The set of transformations determines the complexity of the algorithm
- Algorithm reruns return different solutions
- Ergo: repeat LS and return the overall best solution

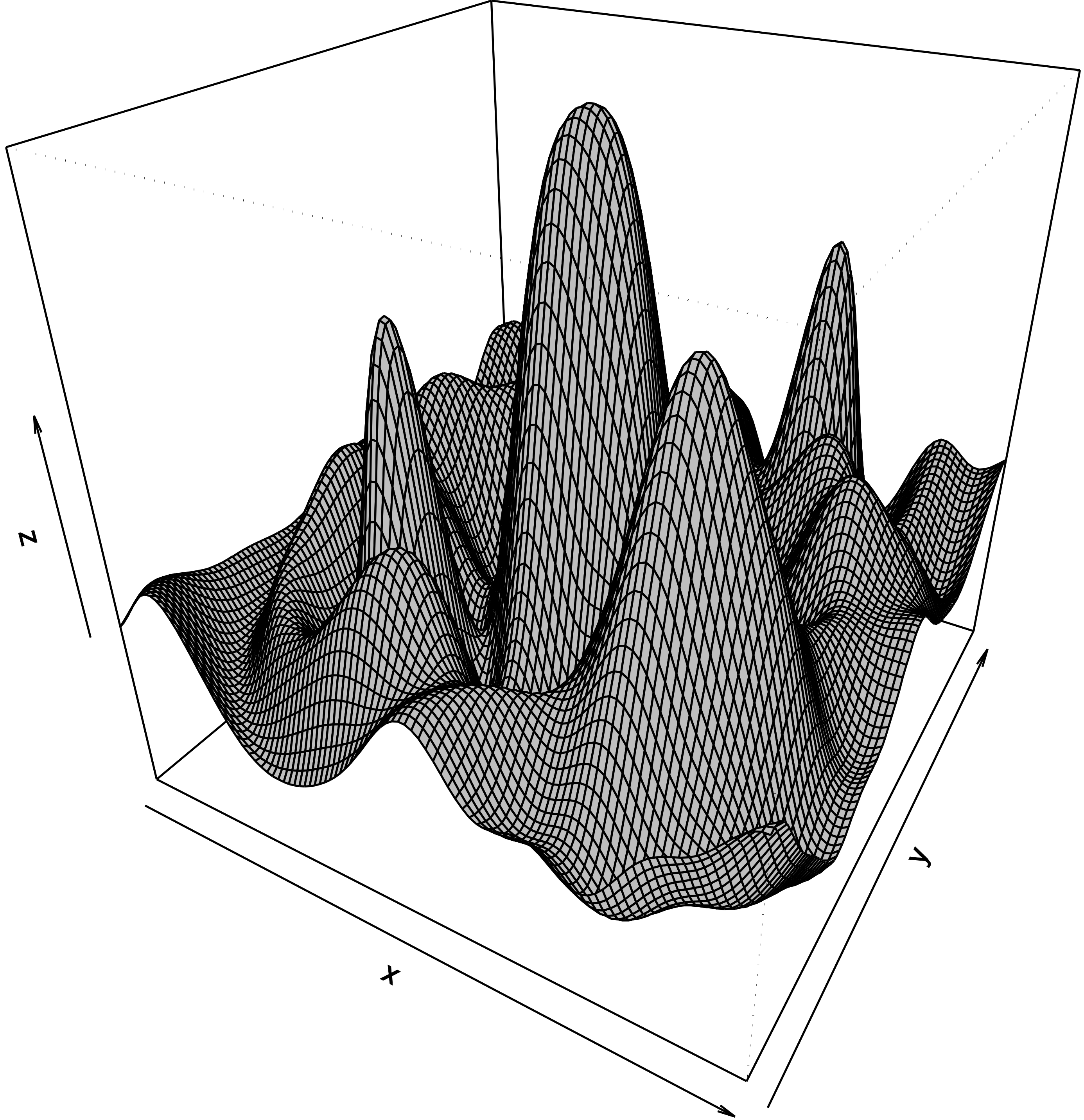
LS basic scheme

```
LS(S_0) { // S_0 is a starting state
  S = S_m = S_0
  do {
    N(S) = {S'; S_0 → S'}
    S = arg min_{S' ∈ N(So)} q(S')
    if ( q(S) < q(S_m) )
      S_m = S
    else
      break ;
  } while (true) ;
  return( S_m ) ;
}
```

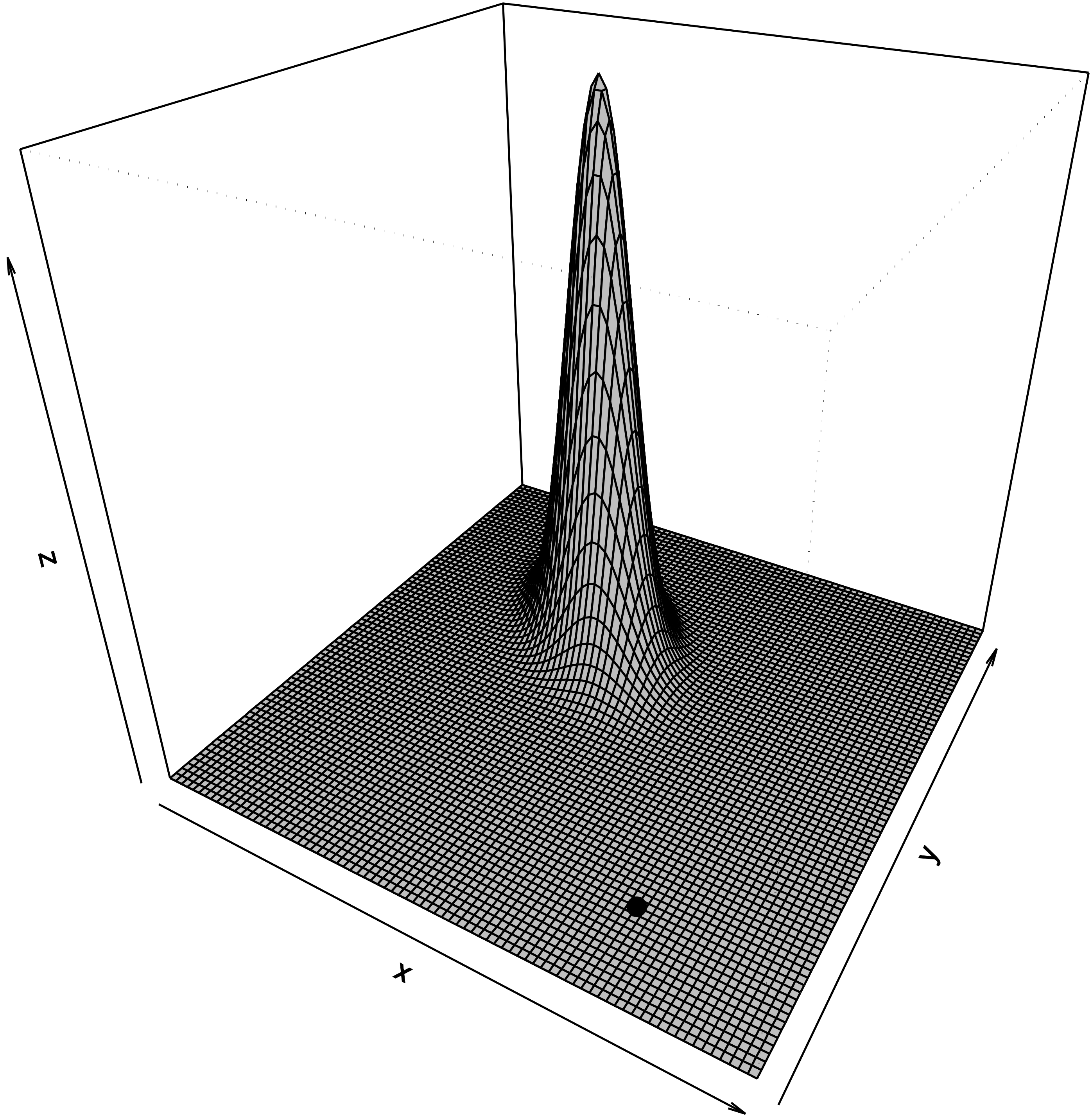
LS problems

- local and global extremes,
- plato,
- ridge

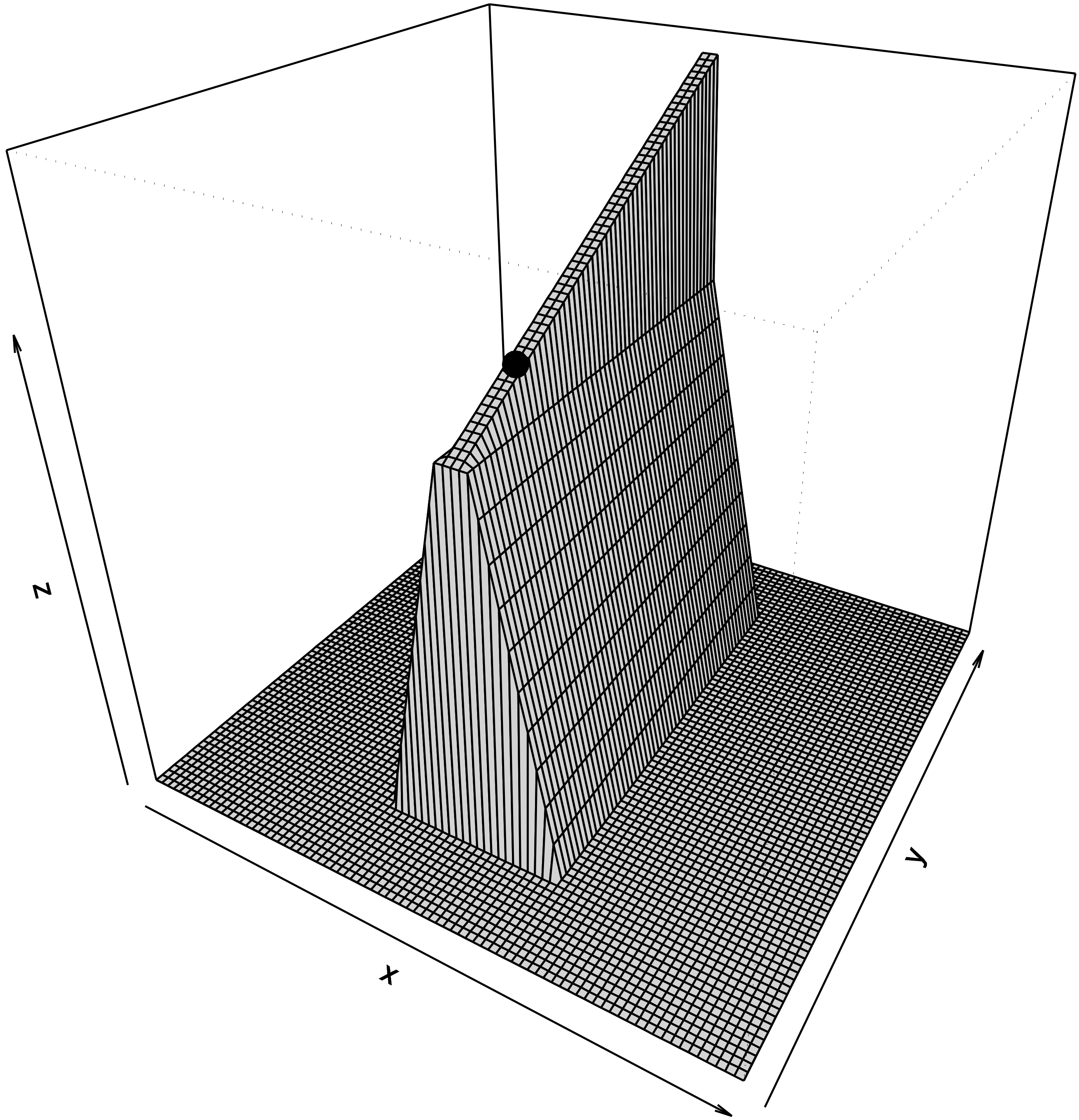
Local extremes



Plato



Ridge



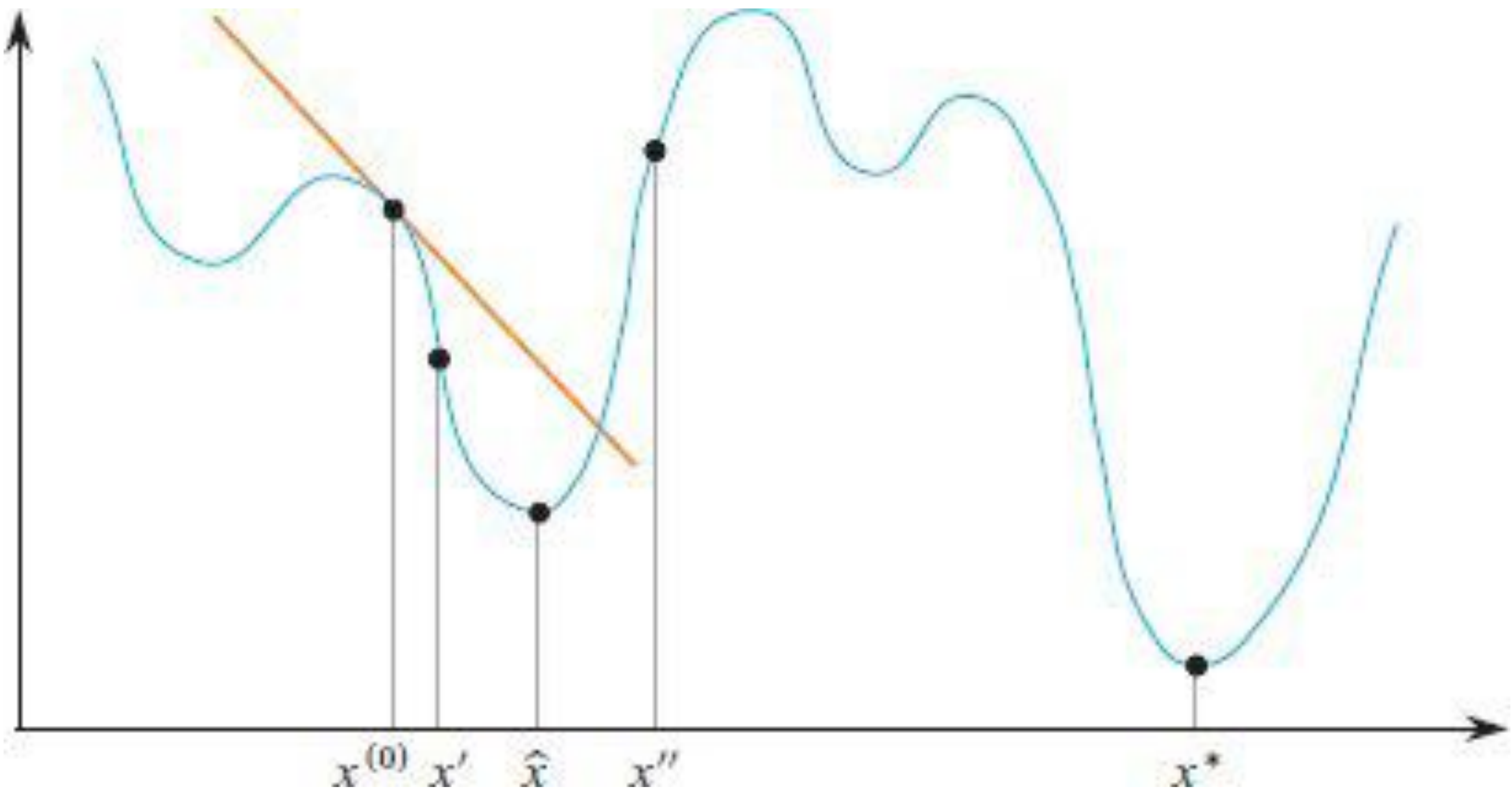
Gradient descent (GD)

- Gradient descent is an efficient local optimization in $\mathbb{R}^n$
- Local minimum of function *f:* $\mathbb{R}^n \to \mathbb{R}$ is a point ***x*** for which *f(x) ≤ f(x')* for all ***x'*** that are "near" ***x***
- Gradient $\nabla f(x)$ is a function ∇f: $\mathbb{R}^n \to \mathbb{R}^n$ comprising *n* partial derivatives:

$$\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})$$

- The GD minimization moves in the direction of -$\nabla f(x)$

Ilustration of GD



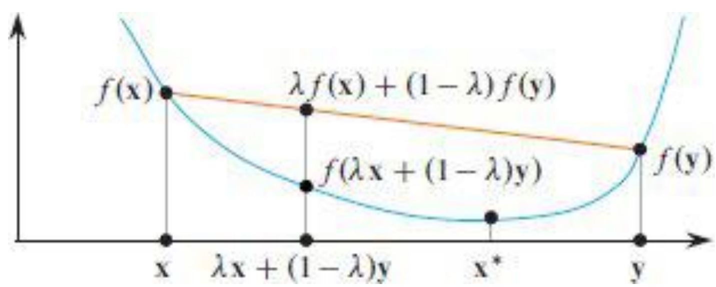
GD algorithm

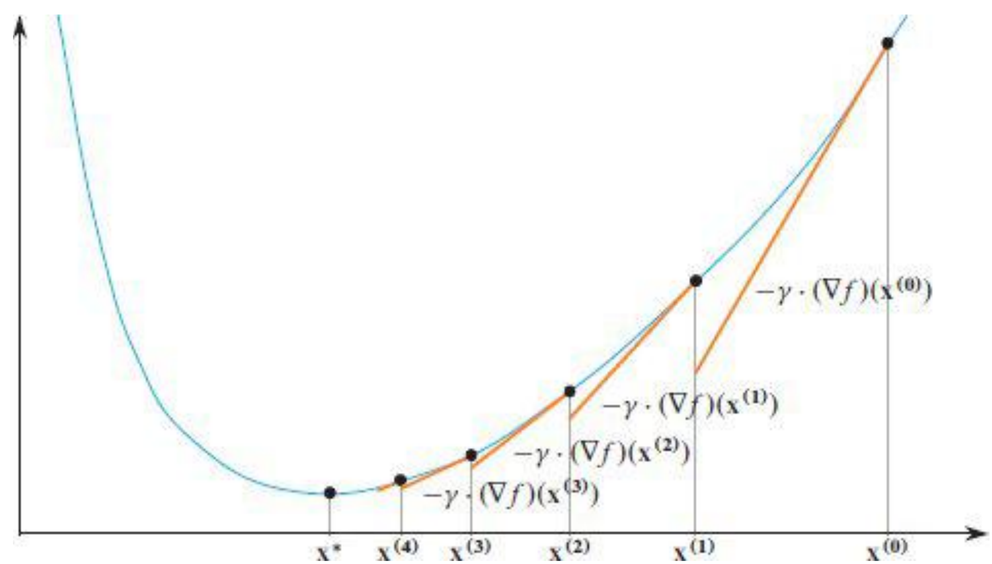
```
GRADIENT-DESCENT(f, x0, γ, T) {
 // function f,  initial value x0, fixed step size γ, number of steps T
 x_best = x = x0 ; // n-dimensional vectors, initially set to the initial value
 f_best = f_x = f(x_best) ;
 for t = 0 to T – 1 do {
   x_next = x – γ  ·  ∇f(x);  // ∇f(x), x, and x_next are n-dimensional
   f_next = f(x_next)
   if (f_next < f_x)
     x_best = x_next ;
   x = x_next ;
   f_x = f_next ;
 }
 return x_best ;
}
```

GD for convex functions

- For convex *f,* the GD finds the global optimum
- Function *f*: $\mathbb{R}^n \rightarrow \mathbb{R}$ is convex if for all x, y ∈ $\mathbb{R}^n$ and for all 0 ≤ λ ≤ 1, we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$





Metropolis algorithm and simulated annealing

- Generalizations of greedy LS
- If a better neighbor exists, move to it
- Otherwise, choose a random neighbor, but accept better neighbors with a larger probability
- Decrease the probability of acceptance with time
- In time, stochastic search turns into deterministic LS

Metropoli - Physics background

- Idea from thermodynamics – trying to find a state with the minimum energy
- Boltzmann distribution law says that the probability of a system being in a state with energy E_i is proportional to:

$$P(E_i) = e^{-\frac{E_i}{kT}},$$

where T is a temperature and k a positive constant.

- Therefore, the probability of a low energy state is larger if the temperature is lower
- To reach a low energy state, i.e. a nice crystal (optimal state), the molten matter has to be cooled slowly
- Cooling too fast gives a suboptimal state (imperfect crystal). The slower we cool the matter, the more probable we get a nice crystal (but the algorithm will be slower).

Metropolis algorithm

```
Metropolis(S_0, T) { // S_0 is a starting state, T is a temperature
   S = S_m = S_0 ;
   do {
      select S' randomly from neighborhood N(S) = {S'; S → S'}
      if ( q(S') < q(S_m) )
         S_m = S' ;
      if ( q(S') < q(S) )
         S = S' ; // move
      else
         with probability e^(-(q(S')-q(S))/T) make a move S = S'

} while (! stopping condition) ;
   return( S_m ) ;
}
```

$$e^{\frac{-(q(S')-q(S))}{T}}$$

Simulated annealing (SA) - the idea

- Use the idea of finding low energy states to introduce stochastic element to LS
- Next state is selected stochastically
- Better neighbors are selected with higher probability
- Use temperature as a knob for stochastic behavior
- Larger temperature implies larger probability for acceptance of worse neighbor and vice versa
- With T = 0, the algorithm is deterministic

SA search

- Start with a random state S
- Select random neighbor S’
- If q(S’) < q(S), move to S’ with probability 1
- Otherwise move to S’ with probability

$$P(S \to S') = e^{\frac{-(q(S') - q(S))}{T}}$$

Annealing

- Decrease temperature while it is not close to zero
- Slower decreasing will cause searching of a larger portion of the search space and will increase the probability of the optimal state
- Usually, a geometrical rule to decrease temperature is used

$$T' = \lambda T, \qquad 0 < \lambda < 1$$

- Typically: $\lambda = 0.95$
- End with a deterministic LS

Algorithm SA

SA(S_0, λ, T) { // S_0 is a starting state, λ is annealing schedule, T is the starting temperature

  S = S_m= S_0 ;

  do {

    randomly select S' from N(S) = {S'; S → S'}

    if (q(S') < q(S_m))

      S_m = S' ;

    if (q(S') < q(S))

      S = S' ; // move

    else {

      with probability $e^{\frac{-(q(S')-q(S))}{T}}$ make a move S = S'

      T = λ T ;

    }

  } **while** (! stopping criterion) ;

  S_m = LS(S_m) ; // end with pure LS

  return(S_m) ;

}

Max-cut and LS

- state space representation
- define neighborhoods
- proof of LS being a 2-approximation algorithm

Max-cut algorithm with LS

```
Max-Cut-Local (G, w) {
   Pick a random node partition (A, B)

   while (∃ improving node v) {
      if (v is in A) move v to B
      else           move v to A
   }

   return (A, B)
}
```

Neighborhood selection

- Large enough not to stop too fast in a local extreme
- Small enough not to be too computationally expensive
- An example: K-L heuristics for max-cut

Best reponse dynamics

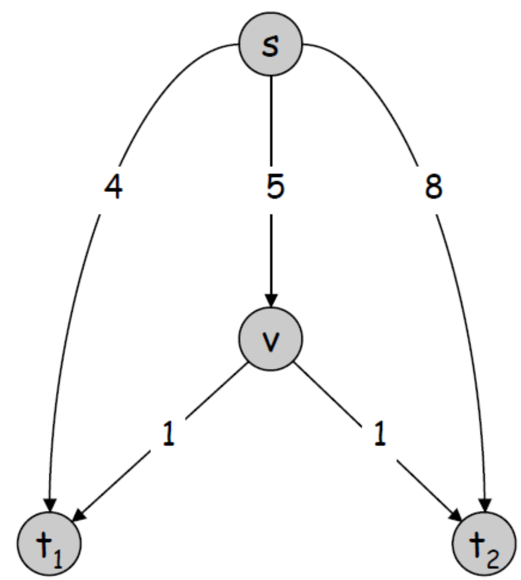
- Multicast routing problem
- Each agents searches the best solution for himself (selfishness)
- Stability of solutions and Nash equilibrium
- Relation to local search
- Social choice
- Price of stability
- Based on J. Kleinberg, E. Tardos: Algorithm Design. Pearson, 2006 (chapter 12)

Multicast Routing

Multicast routing. Given a directed graph $G = (V, E)$ with edge costs $c_e \geq 0$, a source node s, and k agents located at terminal nodes $t_1, \ldots, t_k$. Agent j must construct a path P_j from node s to its terminal t_j.

Fair share. If x agents use edge e, they each pay c_e / x.

1	2	1 pays	2 pays
outer	outer	4	8
outer	middle	4	5 + 1
middle	outer	5 + 1	8
middle	middle	5/2 + 1	5/2 + 1



Nash Equilibrium

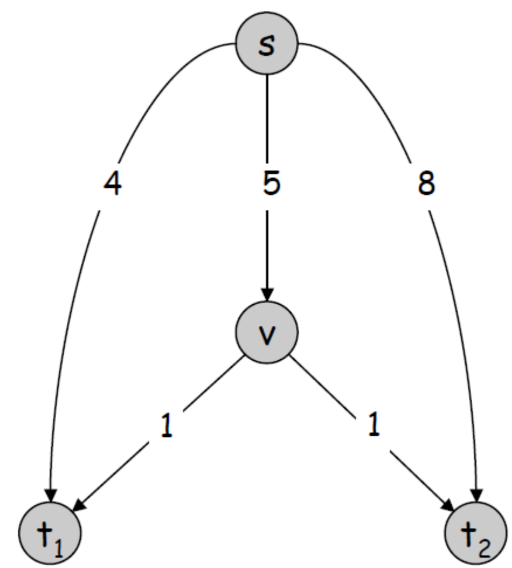
Best response dynamics. Each agent is continually prepared to improve its solution in response to changes made by other agents.

Nash equilibrium. Solution where no agent has an incentive to switch.

Fundamental question. When do Nash equilibria exist?

Ex:

- Two agents start with outer paths.
- Agent 1 has no incentive to switch paths (since $4 < 5 + 1$), but agent 2 does (since $8 > 5 + 1$).
- Once this happens, agent 1 prefers middle path (since $4 > 5/2 + 1$).
- Both agents using middle path is a Nash equilibrium.



Directing multiple agents

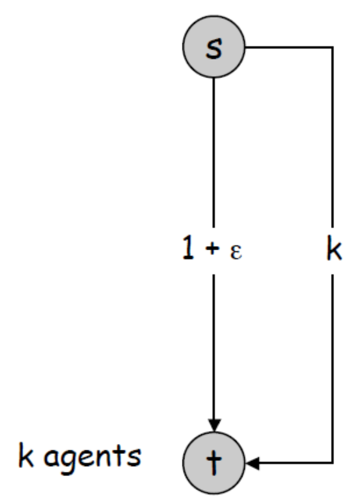
```
Best-Response-Dynamics(G, c) {
   Pick a path for each agent

   while (not a Nash equilibrium) {
      Pick an agent i who can improve by switching paths
      Switch path of agent i
   }
}
```

- provable that the algorithm reaches the Nash equilibrium
- we define a function which strictly decreases in each step

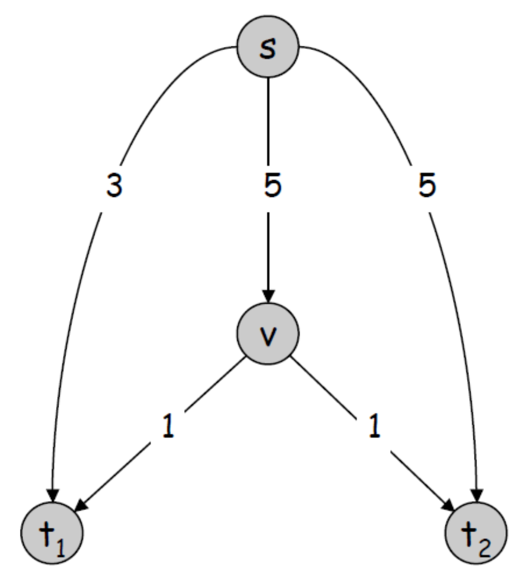
Socially Optimum

Social optimum. Minimizes total cost to all agent.

Observation. In general, there can be many Nash equilibria. Even when its unique, it does not necessarily equal the social optimum.



Social optimum = $1 + \varepsilon$
Nash equilibrium A = $1 + \varepsilon$
Nash equilibrium B = k



Social optimum = 7
Unique Nash equilibrium = 8

Price of Stability

Price of stability. Ratio of best Nash equilibrium to social optimum.

Fundamental question. What is price of stability?

Ex: Price of stability = $\Theta(\log k)$.
Social optimum. Everyone takes bottom paths.
Unique Nash equilibrium. Everyone takes top paths.
Price of stability. $H(k) / (1 + \varepsilon)$.

$1 + 1/2 + \ldots + 1/k$