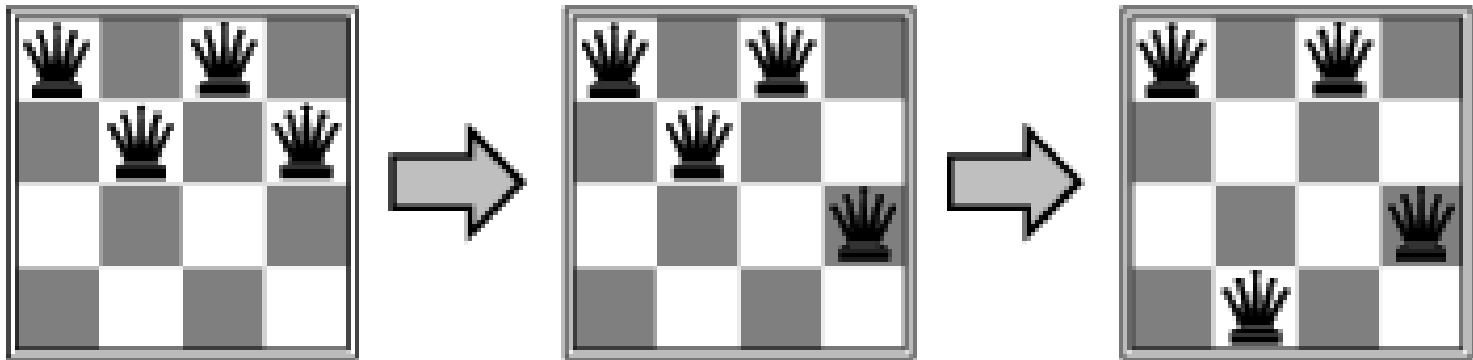


# Busca Local e Otimização

- Em muitos problemas, o caminho para a solução não é importante, e sim a solução obtida
  - Normalmente o estado guarda uma “configuração” do problema, que é modificado a cada iteração
  - Exemplos: problema das n-rainhas, caixeiro viajante, design de circuitos integrados, alocação de tarefas

# Exemplo: 8-Rainhas

- Dado um tabuleiro de Xadrez, colocar uma rainha em cada coluna de forma que elas não se ataquem
- Generalização n-rainhas (tabuleiro  $n \times n$ )



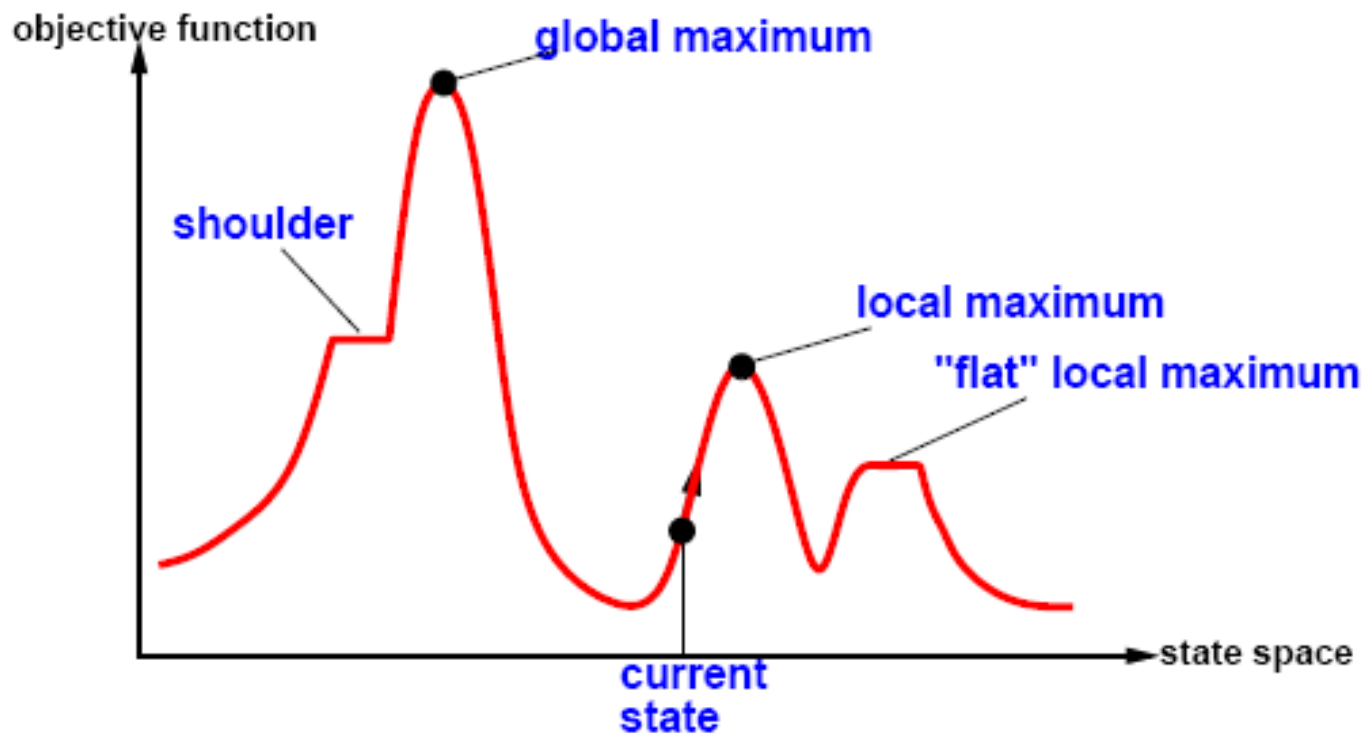
$n$ :	1	2	3	4	5	6	7	8	9	10	11	12	13	14
soluções:	1	0	0	1	2	1	6	12	46	92	341	1,787	9,233	45,752

# Busca Local e Otimização

- Algoritmos de busca local
  - Guardam o estado corrente
  - Investigam (geralmente) os vizinhos em busca de um estado melhor até achar a solução
    - Aperfeiçoam o estado corrente
  - Podem ser usados para resolver problemas de otimização
    - Maximizar ou minimizar uma função objetivo
  - Podem ser usados também para problemas tradicionais de busca, guardando o caminho.

# Busca Local e Otimização

- “Paisagem” dos estados (space *landscape*)
  - $(x)$  é o espaço de estados
  - $(y)$  é a função objetivo











# Hill Climbing

*“Subindo o everest na neblina com amnésia”*

- Move na direção do incremento da função, terminando quando acha um pico
- Guloso: *steepest ascent*
- Problemas
  - Máximos Locais
  - Platôs
    - Máximo local plano
    - *Shoulder* (ombro).

# Exemplo: 8-Rainhas

- Estado: posição das 8 rainhas, uma em cada coluna
  - $8 \times 7 = 56$  sucessores
- Função de avaliação:  $h$  número de pares de rainhas se atacando
- No tabuleiro ao lado,  $h = 17$
- Melhor “vizinho”  $h=12$
- Com 5 jogadas atinge  $h=1$
- Com estados iniciais gerados aleatoriamente, hill climbing fica preso 86% das vezes
- 4 movimentos em média

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

# Hill Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum  
  current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)  
  loop do  
    neighbor  $\leftarrow$  a highest-valued successor of current  
    if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE  
    current  $\leftarrow$  neighbor
```

# Hill Climbing

## ■ Melhorias

### □ Movimentos laterais

- Cuidados para não entrar em loop
- Melhoria para 94% de sucesso no 8-rainhas, com média de 21 movimentos para o sucesso e 64 para a falha

### □ Hill Climbing Estocástico

- Escolhe aleatoriamente entre os sucessores que incrementam a função de avaliação, ou seja, não pega necessariamente o melhor sucessor
- Variação: *first-choice hill climbing*



# Hill Climbing

- Melhorias

- Recomeço Aleatório

- Se falhar, tenta de novo começando em outro estado gerado aleatoriamente
    - Probabilisticamente completo:
    - se  $p$  é a chance de sucesso, são necessários  $1/p$  recomeços para achar o gol
    - Costuma funcionar bem

- O sucesso do Hill Climbing depende fortemente da forma do espaço de estados

# Simulated Annealing

- Alusão ao processo de “temperar” metais
- Similar ao hill climbing, mas faz movimentos aleatórios para sair de mínimos locais (considerando um processo de minimização)
- Aceita movimentos “ruins” (que incrementam a função objetivo), reduzindo a sua frequência e amplitude ao longo do tempo
  - Probabilidade decresce de acordo com  $\Delta E$ 
    - $\Delta E$  é a diferença entre a “energia” dos estados
  - Aceita decrementos maiores no início
    - “Temperatura” diminui ao longo do tempo

# Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to “temperature”

  current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
  for  $t = 1$  to  $\infty$  do
     $T \leftarrow$  schedule( $t$ )
    if  $T = 0$  then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  next.VALUE – current.VALUE
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

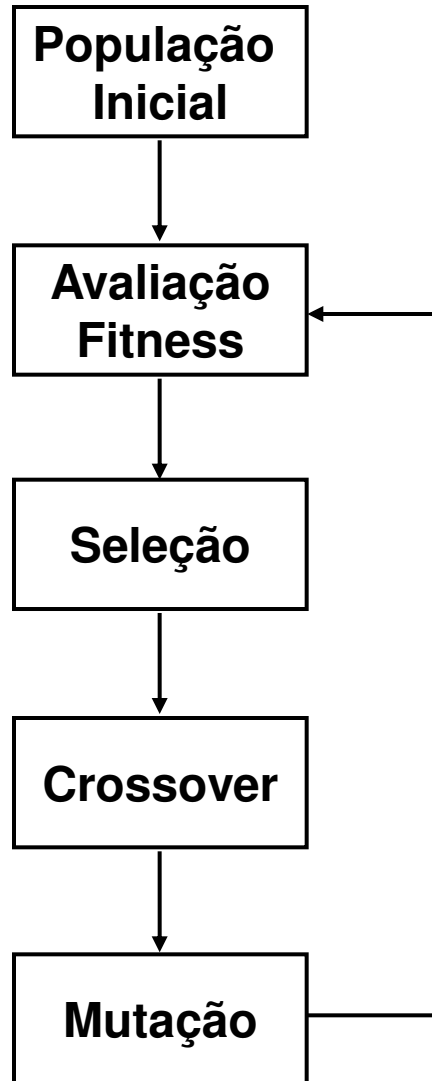
# Local Beam Search

- Investiga  $k$  estados simultaneamente
  - Começa com  $k$  estados aleatórios
  - A cada passo, escolhe  $k$  sucessores
- As buscas em paralelo trocam informação
  - Pode gerar uma concentração de buscas na mesma região do espaço de estados

# Algoritmos Genéticos

- São processos de otimização, no qual “indivíduos” são combinados e os “inferiores” são descartados
- Cada indivíduo é um estado
- Inspirados, de certa forma, na teoria de evolução das espécies
  - *“The survival of the fittest”*
- “Computação Evolutiva”

# Algoritmos Genéticos



# Algoritmos Genéticos

## ■ População inicial

- Normalmente os indivíduos são inicializados aleatoriamente
  - O estado é composto por “genes”
  - Uso de strings ou arrays para representação  
Ex. Como um DNA: CGATTTAA...

## ■ Avaliação do Fitness

- Alguma função de avaliação que indica quão bom é aquele estado

## ■ Seleção

- Indivíduos com melhor avaliação são escolhidos para reprodução
- Uso de probabilidades ou *Thresholds*

# Algoritmos Genéticos

## ■ Crossover

- ❑ Mistura-se os genes de dois ou mais pais que foram selecionados
  - Ex. **CGATTAGC** x **GATACAGG** = **CGATCAGG**
- ❑ A próxima geração irá conter uma mistura de genes dos melhores indivíduos

## ■ Mutação

- ❑ Alguns genes são mudados aleatoriamente
- ❑ Exploração do espaço de estados
- ❑ Garante que novas características que não estão presentes nos pais possam aparecer nos filhos
- ❑ Se são boas características, irão permanecer nas gerações futuras, senão serão eliminadas na seleção

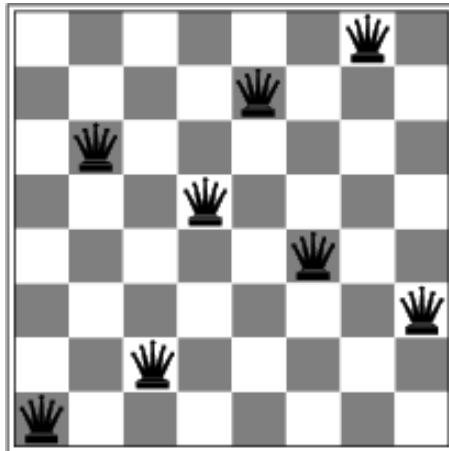


# Exemplo: 8-Queens

## ■ Representação (Genes)

- 8 números, contendo a linha onde está a rainha em cada uma das 8 colunas

16257483



## ■ Fitness Function

- Número de pares que **não** se atacam (0..28)

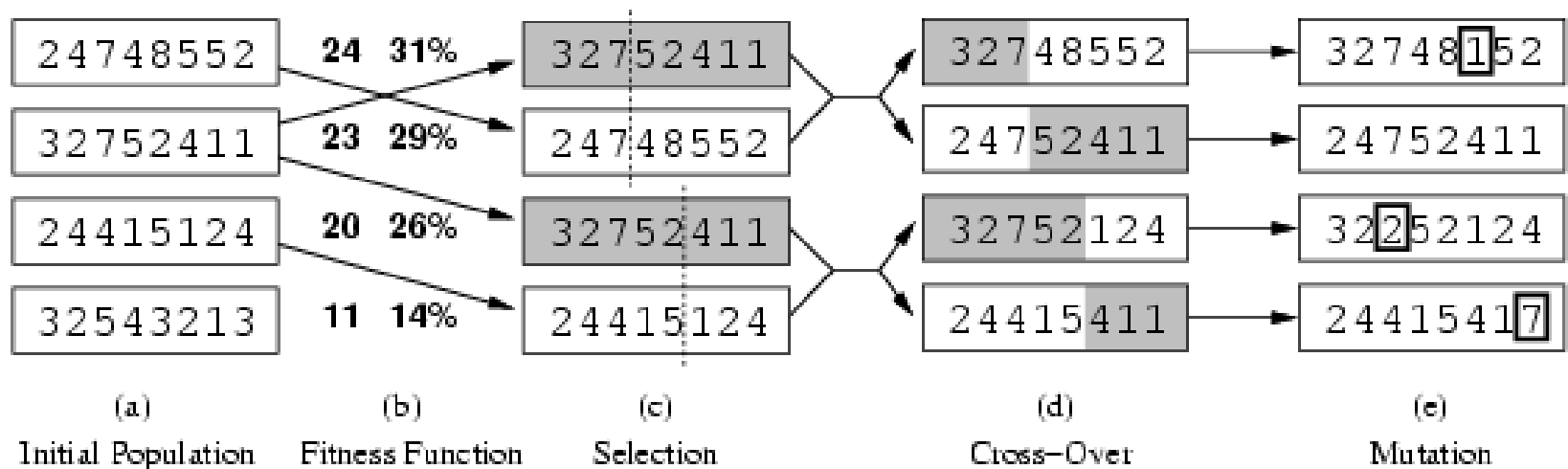
# Exemplo: 8-Queens

## ■ Seleção

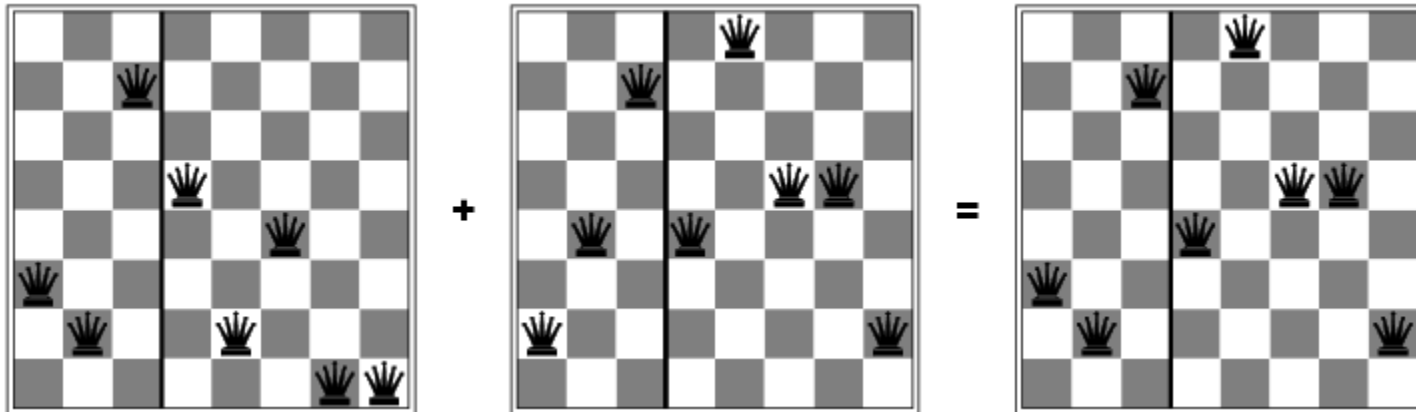
- Escolhe-se os indivíduos proporcionalmente ao valor da função

## ■ Crossover

- Cada par de pais gera dois filhos com  $k$  genes do pai e  $n-k$  da mãe ( $k$  é o *crossover point*)



# Exemplo: 8-Queens



# Algumas Observações

- Várias formas de fazer a evolução
- Estocástico... Necessário *tuning*
- Pode ser computacionalmente intensivo
- Facilmente paralelizável
- Simples de implementar
  - Mas definir a função de avaliação pode ser difícil
- Tem sido muito usado em diversos problemas

# *Particle Swarm Optimization (PSO)*

- Inspiração Biológica – *Swarm Intelligence*
- Várias “partículas” representando possíveis estados navegam pelo o espaço de estados
- O “vetor velocidade” das partículas é alterado com base na ponderação entre a melhor solução já encontrada pela partícula (*pbest*) e pela melhor solução encontrada pelas partículas vizinhas (*lbest*)
  - Inspirado nas *Flocking behaviors* de pássaros

# Busca local em espaços contínuos

- Ambientes contínuos são comuns em diversas aplicações
  - Robótica
  - Visão
  - Aprendizado
- Função objetivo definida em termos de variáveis contínuas
- Em ambientes contínuos a função sucessora pode retornar um número infinito de estados

# Busca local em espaços contínuos

- Gradiente da função

- Em duas dimensões:

$$\nabla f = \frac{\partial f}{\partial x} \hat{x} + \frac{\partial f}{\partial y} \hat{y}$$

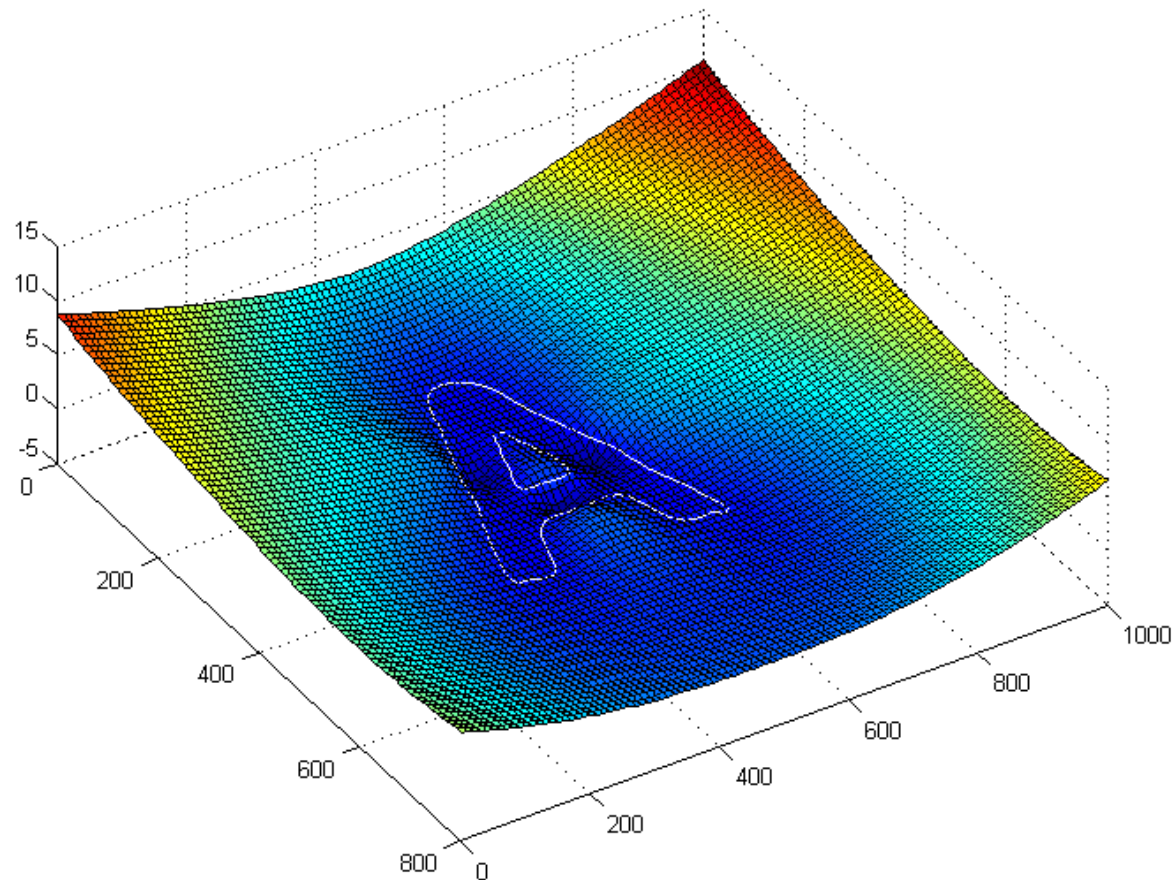
- Aponta para o maior crescimento da função
  - Descida do gradiente leva ao mínimo

- $\nabla f = 0$  normalmente não pode ser resolvida em uma forma fechada

- Busca local:  $X = X + \alpha \nabla f$
  - Método de Newton

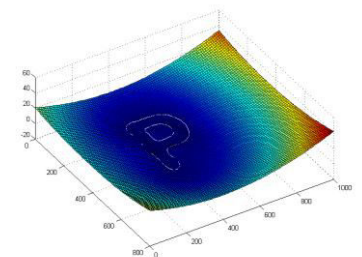
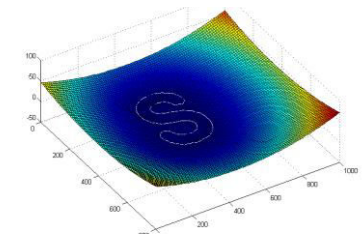
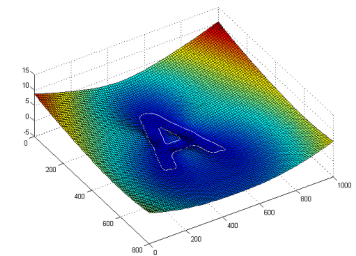
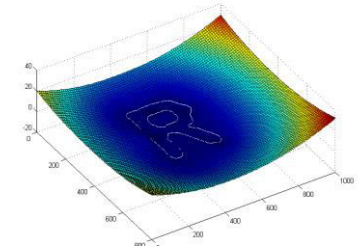
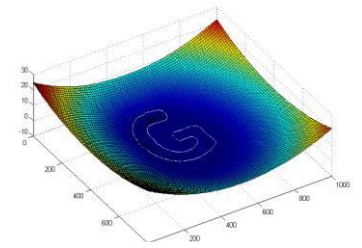
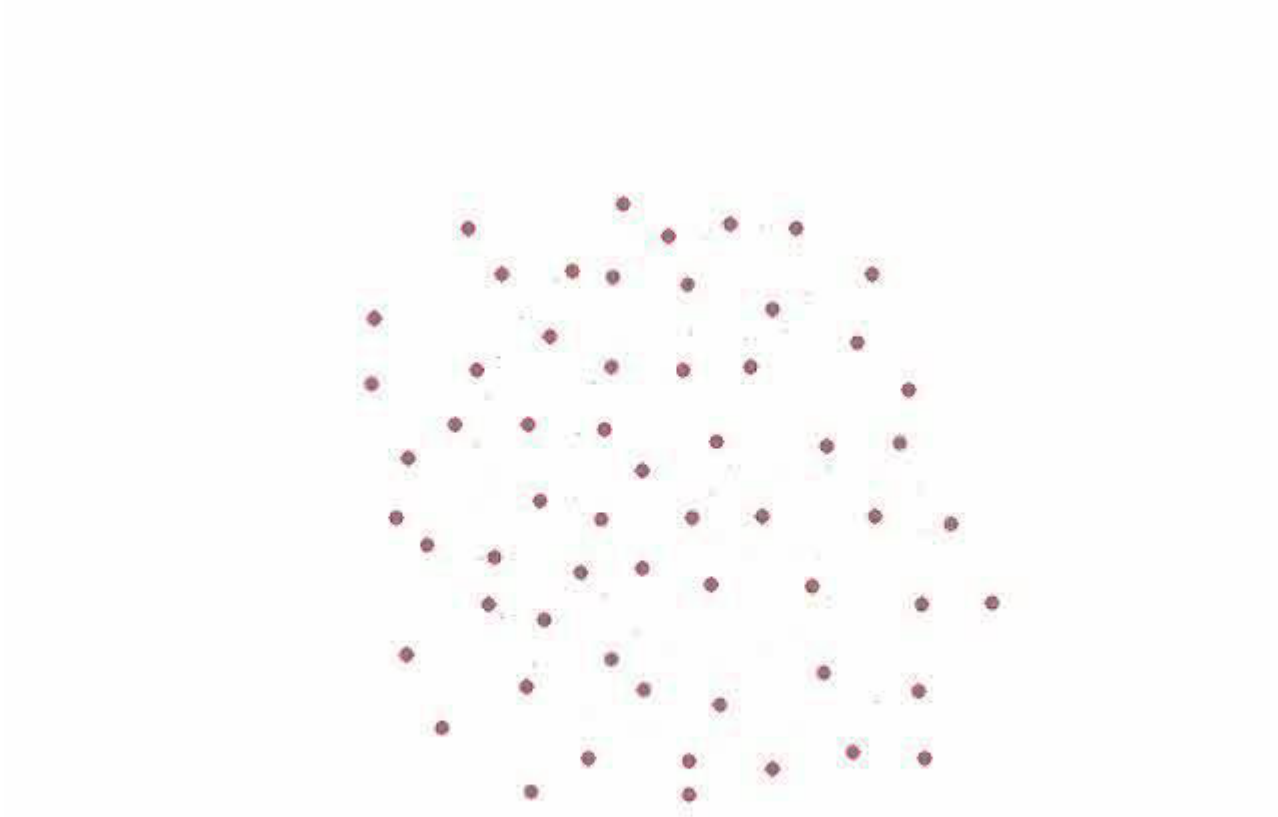
# Swarming c/ Funções Implícitas

Exemplo “físico” de uma descida de gradiente





# Swarming c/ Funções Implícitas



# Swarming c/ Funções Implícitas

