



# »Лекционен курс »Интелигентни системи



Локално търсене >

# Обща характеристика на локално търсене

- » В много случаи **самото описание** на състоянията съдържа **цялата информация**, необходима за едно решение
  - > **Пътят**, по който е достигнато решението **не е съществен**
- » При тези случаи методите за **локално търсене** (итеративно подобрене) са много подходящи
- » **Подход:**
  - > Започваме с една **начална конфигурация**, която **итеративно подобряваме**
    - + Вместо систематично търсене на път



# Клас задачи

## » Приложимо за решаване на следните задачи

- > Разработване на интегрални схеми
- > Оборудване на производствени халета
- > Разписания за изпълнение на производствени договори, транспорт, ...
- > Автоматично програмиране
- > Оптимизация на телекомуникационни мрежи
- > Планиране на маршрути
- > Управление на портфолиа
- > Игри: 8-те царици, ...



# Локално търсене

- » Когато не ни интересува пътя, можем да използваме други алгоритми
  - > Които не се грижат за пътя
- » Те работят с **един единствен актуален възел**
  - > Вместо с много пътища
- » **Движат се само към съседите на този възел**
  - > Не „виждат“ зад съседите
- » **Не съхраняват пътища**
- » Въпреки, че **не са систематични** имат две съществени предимства
  - > **Необходима е малко памет**
  - > Могат да намират **смислени решения в големи (дори безкрайни) пространства**, за които систематичните алгоритми не са подходящи
- » Практически, използват се за решаване на **оптимизационни проблеми**
  - > Цел: най-доброто състояние спрямо целевата функция



# Алгоритми за локално търсене

## » Основни алгоритми от тази група

- > Алгоритъм на „катерача“
- > Симулирано закаляване
- > Локално търсене в лъчи
- > Генетични алгоритми

## » Методи, инсперирани от

- > Физиката (симулирано закаляване)
- > Еволюционната биология (генетични алгоритми)

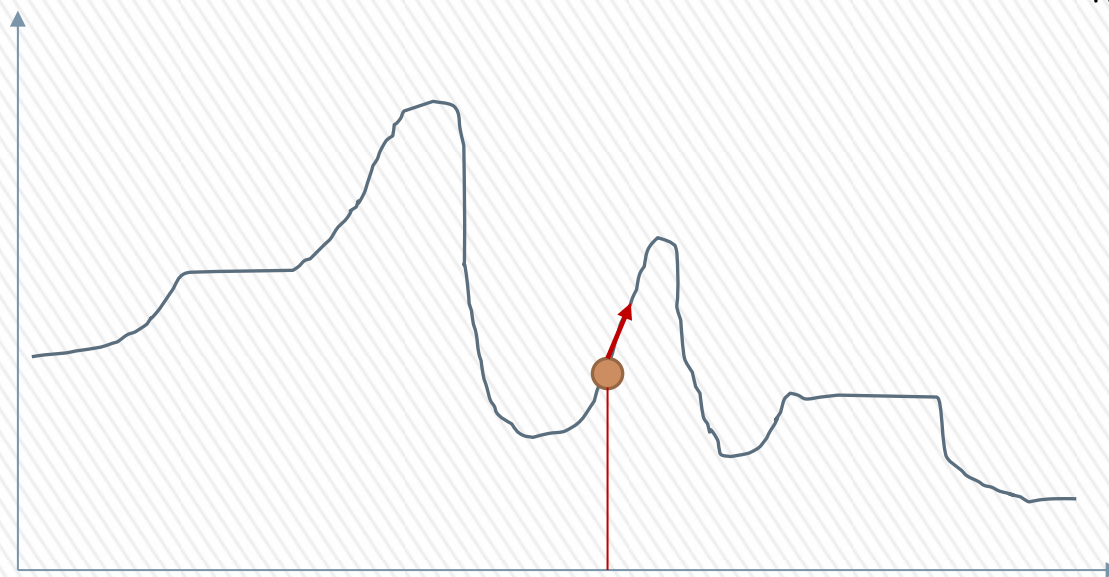




# Релеф в ПС



Целева функция



Актуално  
състояние

Пространство на  
състоянията

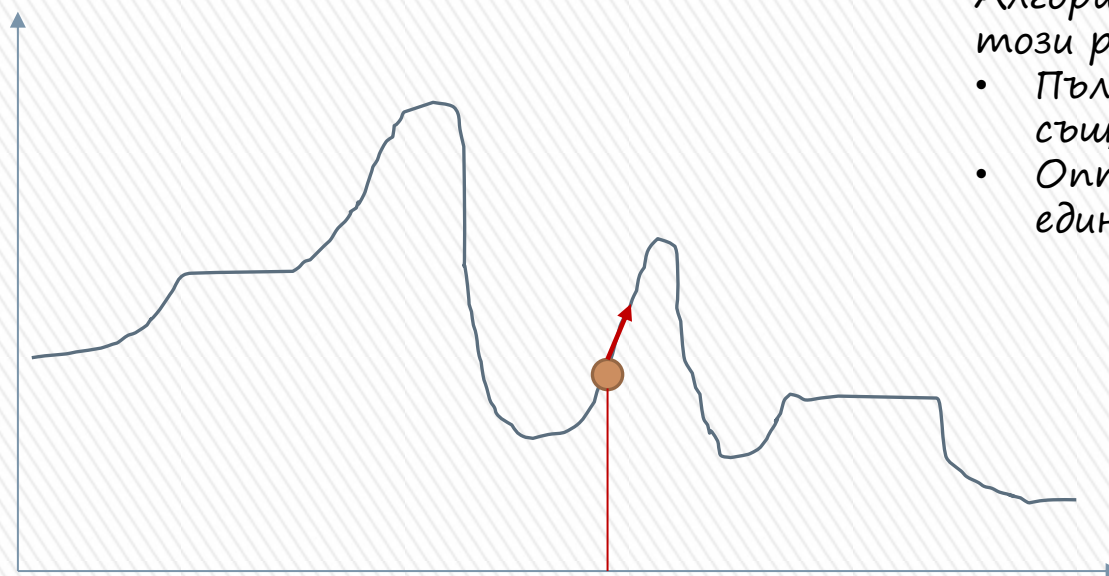
**Релеф:**

1. Позиция – дефинирана от състояние
2. Височина – дефинирана чрез стойността на:
  - Евристична функция на разходите – целта е да се намери най-ниската долина (глобален минимум)
  - Целева функция – целта е да се намери най-високия връх (глобален максимум)

# Релеф в ПС



Целева функция



Актуално  
състояние

Пространство на  
състоянията

Алгоритмите за локално търсене изследват този релеф:

- Пълен алгоритъм – намира винаги една съществуваща цел
- Оптимален алгоритъм – намира винаги един глобален минимум/максимум

# Метод на „катерача“

- » „Изкачване на Еверест в гъста мъгла при загуба на памет“
- » Итерации, протичащи в посока към **нарастваща стойност** (само нагоре)
  - > Прекъсва: намерен връх
    - + Никой съсед няма по-висока стойност
  - > **Не поддържа дърво на търсене**
  - > Не „вижда“ **извън непосредствените съседи** на актуалното състояние





# Метод на катерача (Hill Climbing)

```
function Hill-Climbing (problem) return състояние, което е локален максимум  
  current  $\leftarrow$  Make-Node(problem, Initial-State);  
  loop do  
    neighbor  $\leftarrow$  един current-наследник с най-голяма стойност;  
    if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE;  
    current  $\leftarrow$  neighbor;  
  end do
```

# Пример: 8 царици

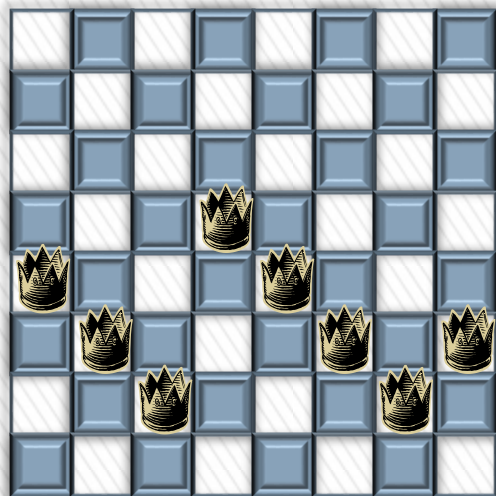
? Каква  $h$ ? Брой на цариците, които се нападат директно или индиректно

? Какви стойности за  $h$ ?

? Колко наследника на всяко състояние?

$$8 \times 7 = 56$$

$h =$  17



# Пример: 8 царици









? Каква  $h$ ? Брой на цариците, които се нападат директно или индиректно

? Какви стойности за  $h$ ?

? Колко наследника на всяко състояние?

$$8 \times 7 = 56$$

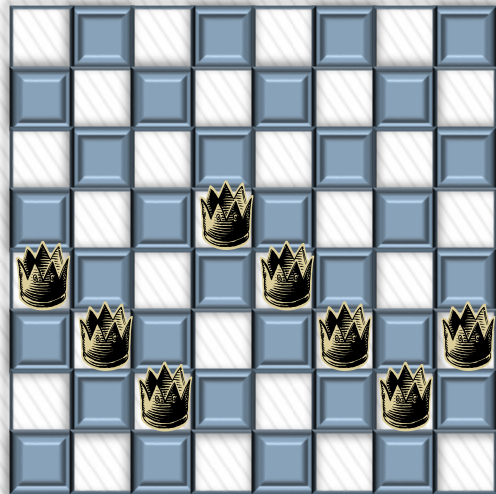
$h =$  17

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

# Пример: 8 царици

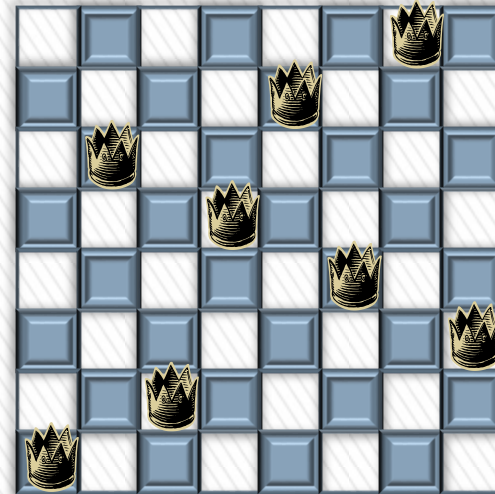
$h =$

17



$h =$

1



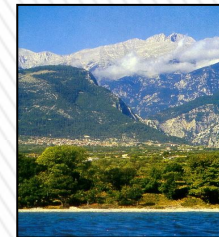
# Характеристика на метода

- » Нарича се още „**лакомо локално търсене**“
  - > Избира „добро“ съседно състояние, без да „мисли“ за продължението
- » Често **стига бързо** до едно решение
- » За примера (8 царици): необходими са само 5 стъпки за да достигне  $h=1$  (от  $h=17$ )
- » Недостатък:
  - > По различни причини често методът „затъва“



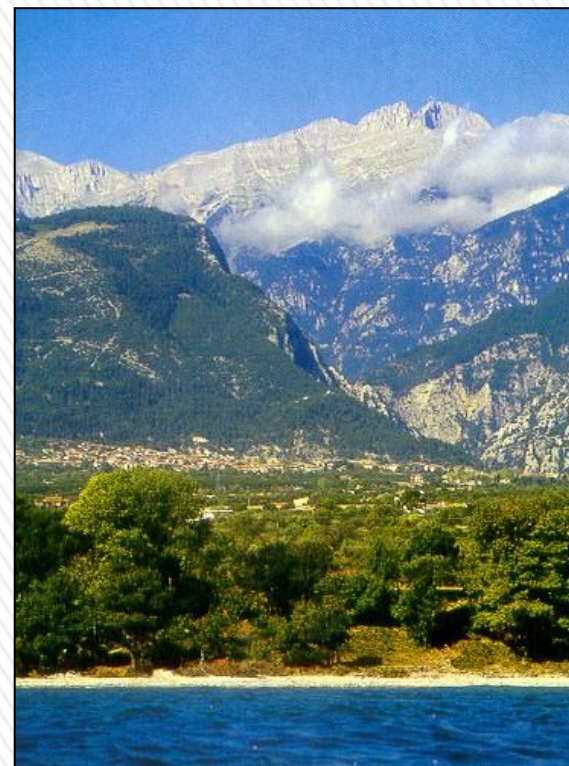


# Проблеми с Релефа



# Проблеми с метода

- » Предпланински проблем:
  - > най-стръмното изкачване води до локален оптимум ("съседен връх")
- » Проблем на платото:
  - > няма разлики в оценките
- » Хребетен проблем:
  - > предварително зададените посоки не разрешават изкачване



# Оценка на метода

## » При 8 царици

- > 14% решения, 86% „затъвания“ (неуспех)
- > Впечатляваща скорост: ПС с  $\approx 17$  милиона състояния ( $8^8$ )
  - + При успех използва средно 4 стъпки
  - + При неуспех средно 3 стъпки

## » Използване на „обходни“ пътища: напр., при 8 царици

- > При допустими 100 последователни „обхода“
- > Успехът се повишава от 14% на 94%



# Варианти на „катерача“

- » Съществуват различни варианти на метода
  - > Стохастичен „катерач“
    - + Избира случайно движение нагоре
    - + Вероятността на избора може да варира според „стръмнината“
  - > „Катерач“ с първи избор
    - + Случайно генериране на наследници, докато се генерира един „по-добър“ наследник на актуалното състояние



# Варианти на „катерача“

- > „Катерач“ със случаен нов старт
  - + Разгледаните досега методи са **непълни**
    - Не намират съществуваща цел понеже „затъват“ в някой локален максимум
  - + Изпълнява последователност от „изкачвания“ докато е намерена една цел
  - + **Пълен метод с вероятност, близка до 1**
    - Все някога ще генерира целево състояние като начално състояние
    - С вероятност **p** за успех, очакван брой нови стартове **1/p**, т.е. за 8 царици **p ≈ 0.14** или необходими около 7 итерации за постигане на една цел (6 грешни и 1 успешна)





# Обобщение на метода

- » Успехът на „катерача“ зависи в голяма степен от **вида на „релефа“**
  - > Когато съществуват малко локални максимуми и плата, тогава „катерачът“ със случаен нов старт намира много бързо добро решение
- » NP проблемите обикновено имат експоненциален брой локални максимуми, където могат да „затъват“



# Симулирано закаляване

- » „Катерачът“ – гарантирано **непълен** метод
  - > Понеже може да „затъне“ в локален максимум
- » Чисто случайно продължение – **пълен**, но изключително **неефективен**
- » Смислено комбиниране на двата метода
  - > За получаване пълнота и ефективност
- » „Симулирано закаляване“ е такъв алгоритъм

*В металургията: металите се загряват до високи температури, след което се оставят на постепенно охлаждане*



# Симулирано закаляване

```
function Simulated_Annealing(problem, schedule) return Goal_State
inputs: problem, един проблем
        schedule, функция на температурата във времето

current  $\leftarrow$  Make-Node(problem, Initial-State)

for t = 1 to  $\infty$  do
    T  $\leftarrow$  schedule(t)

    if T=0
    then return current

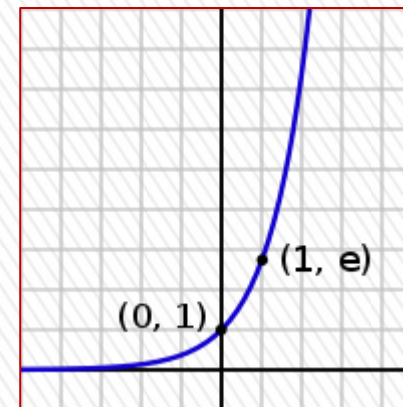
    next  $\leftarrow$  случайно избран наследник на current
     $\Delta E \leftarrow$  next.VALUE – current.VALUE

    if  $\Delta E > 0$ 
    then current  $\leftarrow$  next
    else current  $\leftarrow$  next с вероятност  $e^{\Delta E/T}$ 
```



# Експоненциална функция

- » В математиката, експоненциалната функция е тази функция, която е равна на собствената си производна
- » Бележи се с  $e^x$  (или  $\text{exp}(x)$ ), където  $e$  е Неперовото число (равно приблизително на 2.718)
- » Използва се за изразяване на функционална връзка, при която фиксирана промяна в абсолютната стойност на независимата променлива води до фиксирана пропорционална промяна
  - > Т.е. процентно увеличение или намаляване в стойността на функцията
- » Експоненциалната функция има широка употреба във физиката, химията и математиката



# Обяснение на метода

- » **Подобен** на метода на „катерача“
- » Вместо да избере „най-добрия“ наследник, приема **случайно избран наследник**
- » Ако избраният наследник подобрява ситуацията, **винаги** се приема
- » В противен случай, наследникът се приема **с вероятност по-малка от 1**
  - > Вероятността намалява експоненциално с „неприемливостта“ на движението – със стойност  $\Delta E$ , с която оценката е по-лоша
  - > Вероятността намалява също с падането на температурата
    - + В началото „лошите“ движения се появяват с по-голяма вероятност, когато  $T$  е висока и стават все по-невероятни когато  $T$  пада
- » Когато разписанието планира достатъчно бавно падане на температурата, алгоритъмът намира глобален оптимум с вероятност, близка до 1





# Приложение на метода

- » За първи път методът е използван в началото на 80-те години на 20 век за създаване на VLSI
  - > Проблем за рутиране на каналите
    - + Проблем на търсене с изключително висока комплексност
  - > Елементите трябва да бъдат поставени в чипа така, че да не се препокриват и да има място за свързващите ги проводници



# Локално търсене в лъчи

- » Използва **k** състояния вместо само едно
- » Начално състояние:
  - > Множество от **k** случайно генерирани състояния
- » Разширение:
  - > На всяка стъпка се генерират **всичките наследници** на всичките **k** състояния
- » Ако едно от тях е цел, алгоритъмът завършва
- » В противен случай **се избират** най-добрите **k** наследника от цялото множество наследници и процесът се повтаря



# Коментар

- » Търсенето в лъчи с  $k$  състояния **прилича на паралелно** вместо последователно изпълнение на  $k$  случайни нови стартирания
- » В същност двата алгоритъма **са много различни**
- » При търсене със случаен нов старт, всяко търсене се изпълнява независимо от другите
- » При търсенето в лъчи съществената информация се предава между паралелните нишки на търсене
  - > Практически състоянията, където се генерират най-добрите наследници споделят това с останалите
  - > Алгоритъмът прекъсва бързо нерезултатното търсене и концентрира ресурсите си там, където се постига най-големия напредък



# Генетични алгоритми

## » Идея

- > Състоянията-наследници се генерират като се комбинират две родителски състояния

## » Начално състояния: популация

- > Множество от **k** случайно генерирани състояния
- > Обикновено всяко състояние (индивид) се идентифицира посредством поредица от символи от някаква крайна азбука

## » Оценка

- > Всяко състояние се оценява посредством целева функция (фитнес-функция)
- > За по-добрите състояния тази функция връща по-високи стойности

## » Разширяване: репродукция

- > Използват се два оператора
  - + Кръстосване (Crossover)
  - + Мутация (Mutation)



# Ч. Дарвин: “The Origin of Species”

» В книгата Дарвин поддържа принципа на еволюцията посредством естествена селекция:

- > Всеки индивид е склонен да предава характерните си белези на потомците
- > Въпреки това, природата създава индивиди с различни белези
- > Най-годните индивиди (тези с най-благоприятни белези) са склонни да имат повече потомци
- > Това насочва популацията като цяло към фаворизираните белези
- > В един дълъг период, могат да се натрупат варианти, създавайки нови видове, които заемат екологични ниши

» От молекулярна гледна точка естествената селекция става възможна чрез видоизменение, което следва от:

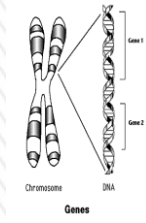
- > Кръстосване – събира съществуващите гени в нови комбинации
- > Мутация – създава нови, непознати досега гени

»

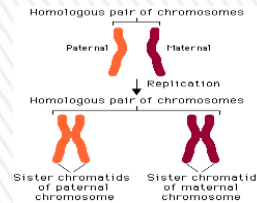




# Понятия: хромозоми, гени, хомология, диплоид, хаплоид, кръстосване, мутация



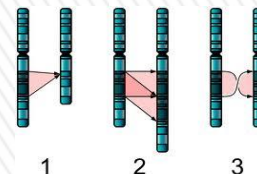
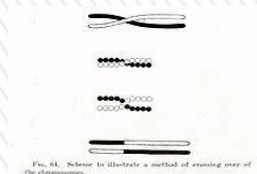
- Ядрата на високо развитите растения и животни съдържат хромозоми
- От своя страна хромозомите съдържат гени - фактори за отличителните белези



- Обикновено хромозомите се чифтосват и образуват хомологии



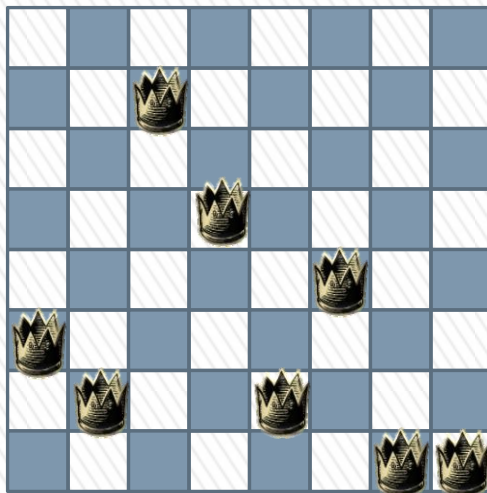
- Клетки, които съдържат чифтосани хомологии се наричат диплоиди



- Кръстосване, мутация

# Пример: 8 царици

- ? Каква фитнес-функция?
- ? Стойност на фитнес-функцията за решението и за примера?



Фитнес-функция: Брой на двойките царици, които не се нападат

- Решение:

28

- За примера:

23

# Пример: 8 царици

Начална  
популация

2 4 7 4 8 5 5 2

3 2 7 5 2 4 1 1

2 4 4 1 5 1 2 4

3 2 5 4 3 2 1 3

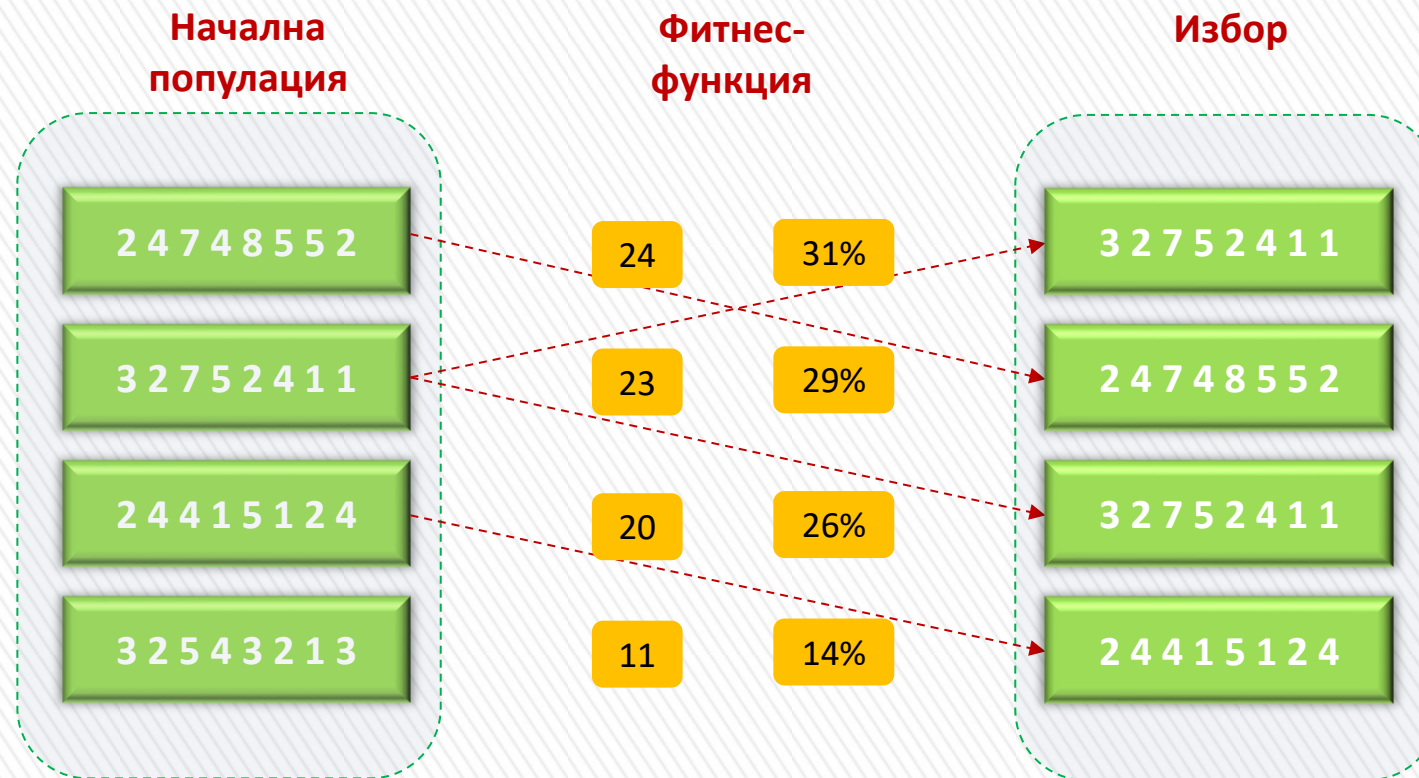
Представяне: (колона, ред)

Като поредица от цели числа:

- Колона: позиция
- Ред: число

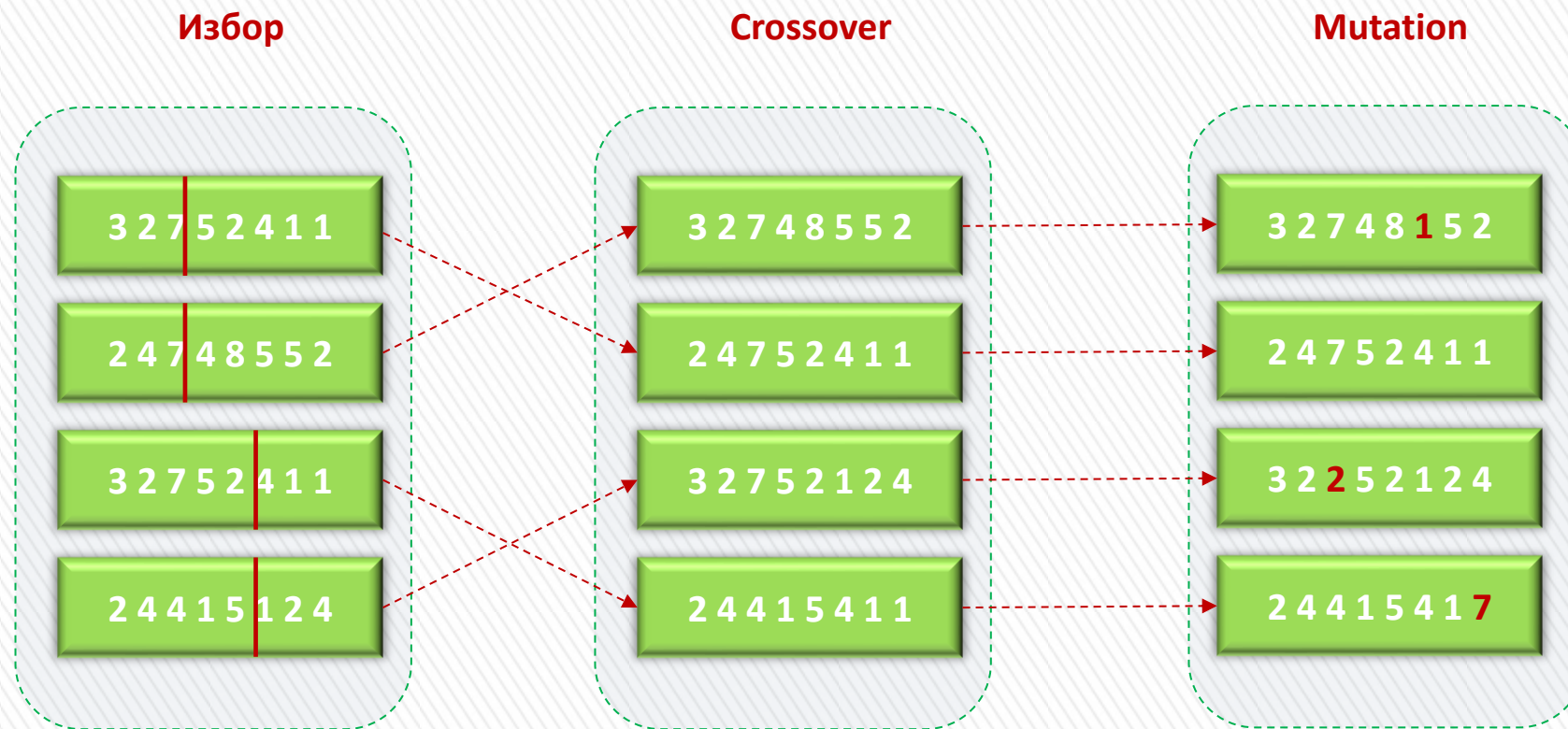
Фитнес-функция: 24, 23, 20 и 11

# Пример: 8 царици



Случаен избор за репродукция на две двойки, в зависимост от вероятностите

# Пример: 8 царици

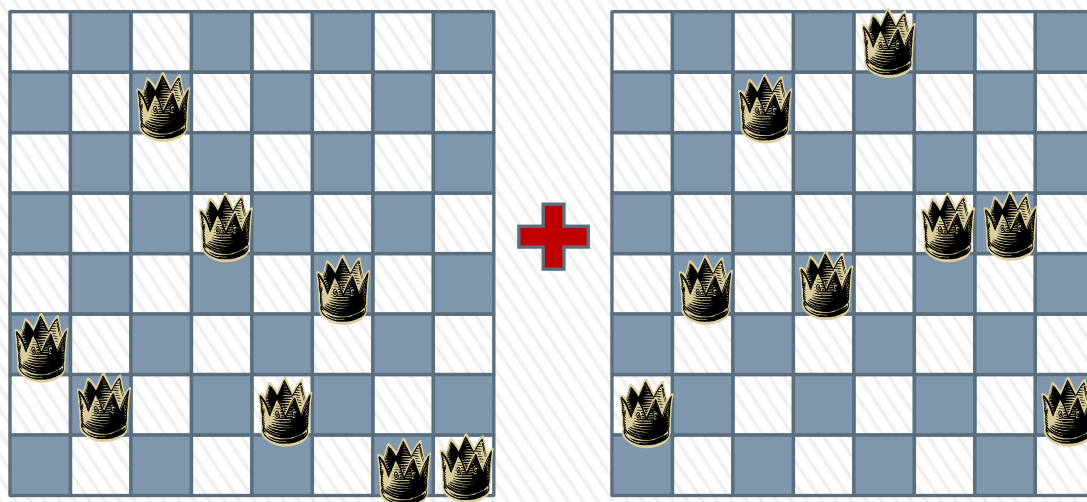


Случаен избор на точката  
на кръстосване (за всяка  
двойка)

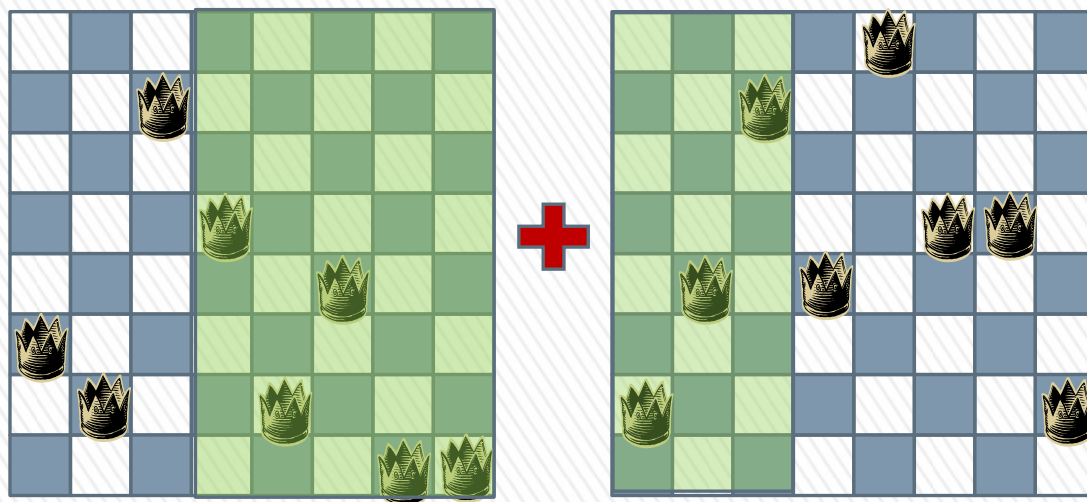
Всяка позиция може да бъде случайно  
избрана с малка независима  
вероятност за мутация



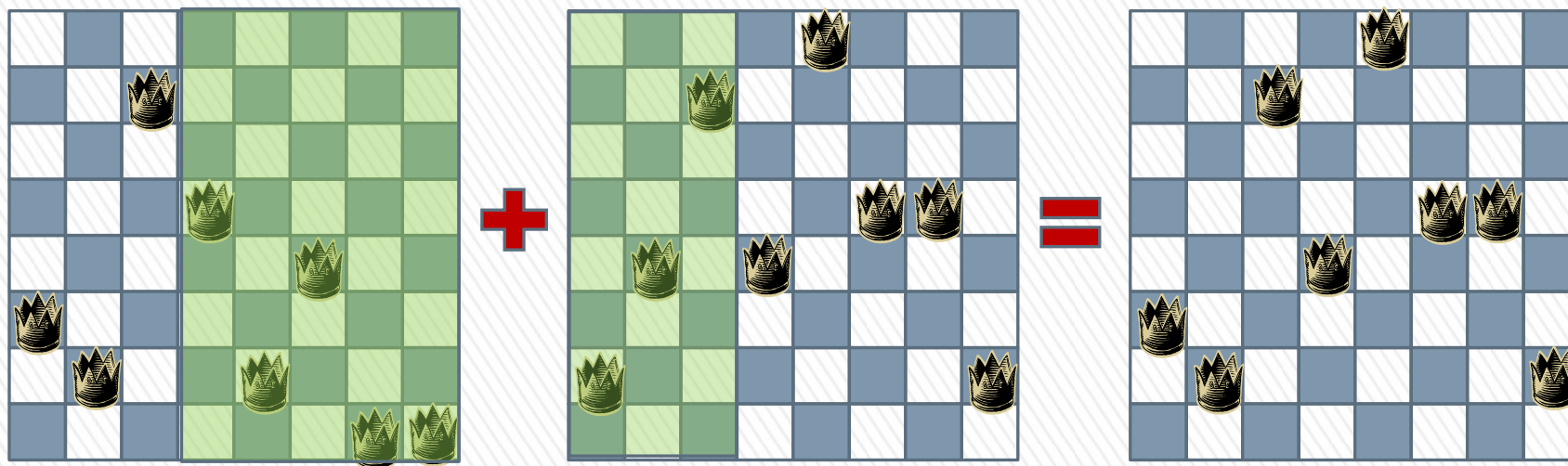
# Пример: 8 царици (кръстосване)



# Пример: 8 царици (кръстосване)



# Пример: 8 царици (кръстосване)



# Генетичен алгоритъм

```
function Genetic-Algorithm (population, Fitness-Fn) return един индивид
  inputs: population, a set of individuals
         Fitness-Fn, a fitness function
  repeat
    new_population  $\leftarrow \emptyset$ 
    for i = 1 to Size(population) do
      x  $\leftarrow$  Random-Selection(population, Fitness-Fn)
      y  $\leftarrow$  Random-Selection(population, Fitness-Fn)
      child  $\leftarrow$  Reproduce(x, y)
      if (small probability) then child  $\leftarrow$  Mutate(child)
      child add to new_population
    population  $\leftarrow$  new_population
  until един индивид достатъчно годеен или изразходвано много време
  return най-добрия индивид в population относно Fitness-FN
```

# Генетичен алгоритъм: reproduce

```
function Reproduce (x,y) return един индивид
    input: x, y , родителски индивиди

     $n \leftarrow \text{Length}(x)$ 
     $c \leftarrow \text{Random-number between 1 and } n$ 

    return Append(Substring(x, 1, c), Substring(y, c+1, n))
```



# Пример

Родител 1:

1	1	1	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---

Родител 2:

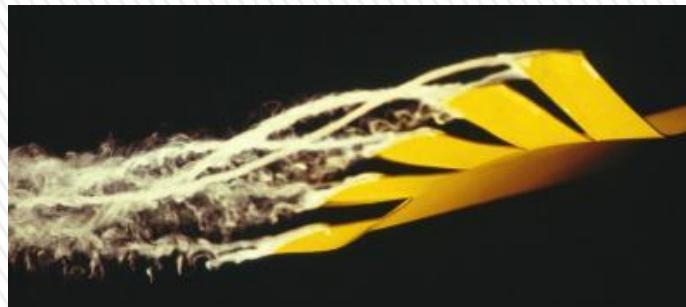
0	0	1	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---

1	1	1	0	1	0	0	0	1	1
0	0	1	1	0	0	1	1	0	0

# Приложения: Бионика

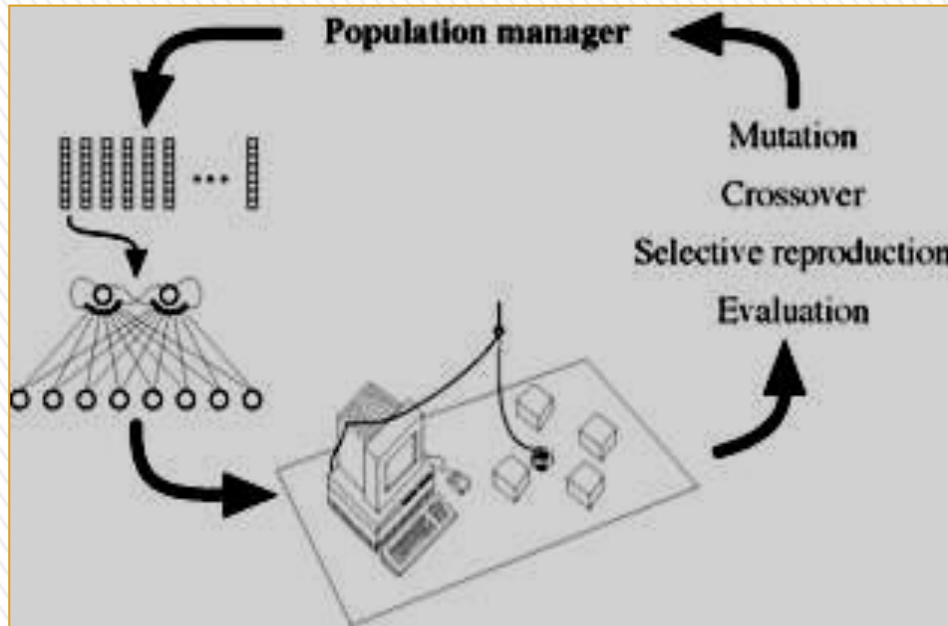
TU Berlin (Ingo Rechenberg)

<http://lautaro.bionik.tu-berlin.de/institut/s2foshow/>



# Приложения

„Artificial Life“: Golem-Project Cornell University



- » *Genetically Organized Lifelike Electro Mechanics*
- » *This is the first time robots have been robotically designed and robotically fabricated*





Благодаря за вниманието!