Алгоритмы оптимизации поиска

Популяционные алгоритмы

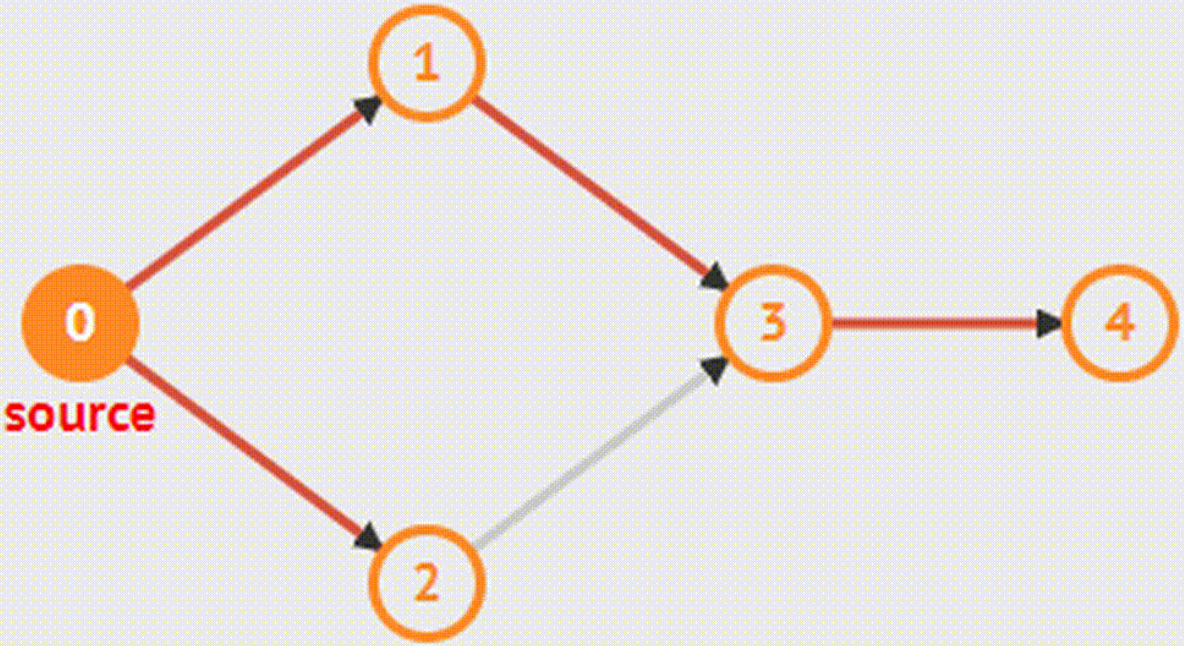
О чём речь? :)

- ✓ Искусственный интеллект
- ✓ Алгоритмы поиска
- ✓ Классические алгоритмы поиска
- ✓ Альтернатива классическим — популяционные поисковые алгоритмы

Поиск в ширину

✓ Только один движущийся объект,
решающий задачу — то есть
один программный поток,
или один так называемый — агент
— как принято в терминологии у людей в этой области

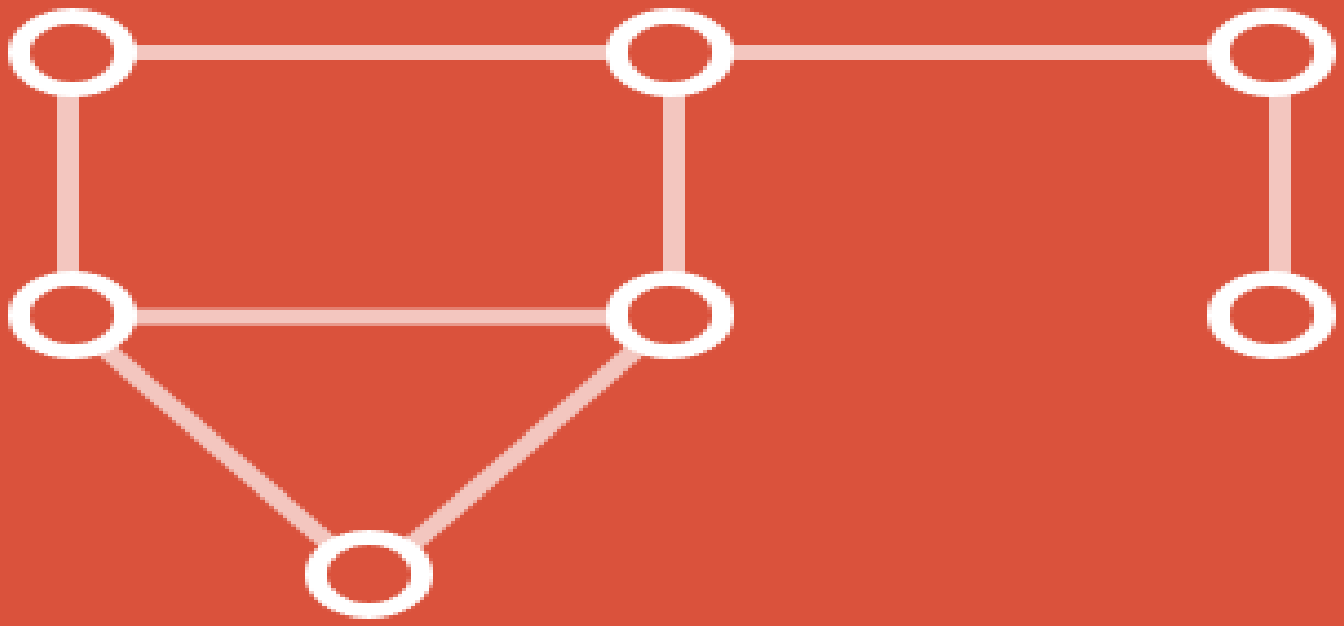
Поиск в ширину



Популяционный алгоритм

Популяционные алгоритмы называют ещё многоагентными, потому что они, в отличие от классических поисковых алгоритмов, ищут решение с помощью сразу нескольких агентов (совокупность «общающихся» между собой агентов называют роем)

Популяционный алгоритм



Классификация популяционных алгоритмов

- ✓ Детерминированные и стохастические (случайные)
- ✓ Статические и динамические
- ✓ Алгоритмы детерминированного поиска и стохастического (случайного) поиска

Популяционные = эвристические

Все популяционные алгоритмы называют эвристическими, то есть основанными на опыте, а не на строгом доказательстве верности их результата

Ещё разок об агентах

В качестве общего названия членов популяции используем термин **агент** (agent). В различных популяционных алгоритмах агенты называются индивидами, особями, частицами, муравьями, пчелами, бактериями и т.д.

Смысл названия

Поведение компьютерных агентов математических популяционных алгоритмов схоже с поведением особей разных популяций в живой природе.

Отсюда и идут заимствования названий, в том числе и самого названия алгоритма — популяционный алгоритм поиска.

Что конкретно мы ищем?

Определимся, с чем мы с Вами будем работать у нас в примерах дальше.

Мы должны **оптимизировать некую функцию**, выражающую смысл поставленной нами задачи.

Для этого упрощаем эту функцию, создавая некое её подобие, которое нам более удобно для реализации нашего алгоритма — **фитнес-функцию**.

Что конкретно мы ищем?

<...> Для этого упрощаем эту функцию, создавая некое её подобие, которое нам более удобно для реализации нашего алгоритма — **фитнес-функцию.**

Стратегически, в процессе миграции агенты движутся таким образом, чтобы приблизиться к глобальному экстремуму фитнес-функции.

Этапы популяционных алгоритмов

Общая схема популяционных алгоритмов включает в себя следующие этапы:

- **Инициализация популяции**. В области поиска тем или иным образом создаем некоторое число начальных приближений к искомому решению задачи – инициализируем популяцию агентов.

Этапы популяционных алгоритмов

- **Миграция агентов популяции**. С помощью некоторого набора миграционных операторов, специфических для каждого из популяционных алгоритмов, перемещаем агентов в области поиска таким образом, чтобы в конечном счете приблизиться к искомому экстремуму оптимизируемой функции.

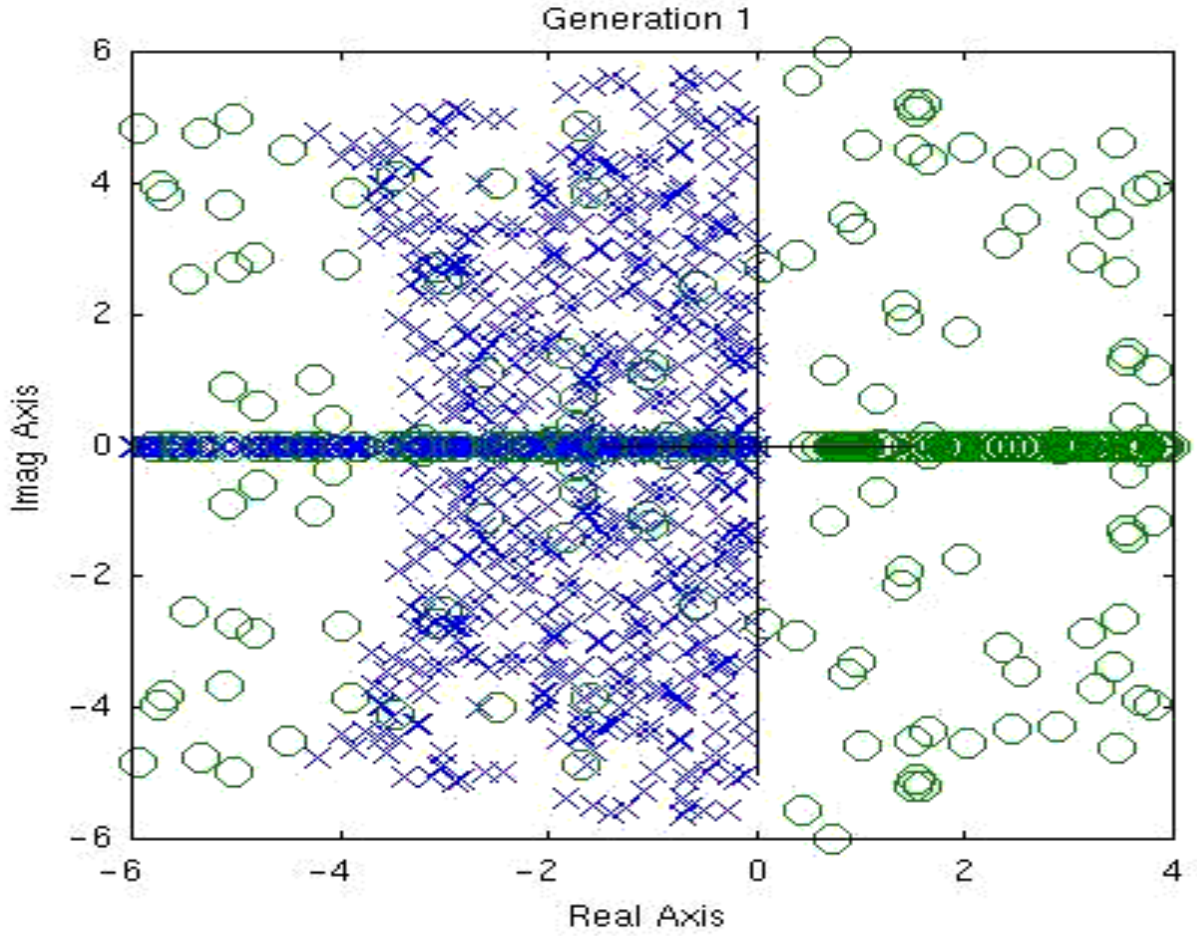
Этапы популяционных алгоритмов

- **Завершение поиска**. Проверяем выполнение условий окончания итераций, и если они выполнены, завершаем вычисления, принимая лучшее из найденных положений агентов популяции за приближенное решение задачи. Если указанные условия не выполнены, возвращаемся к выполнению этапа 2.

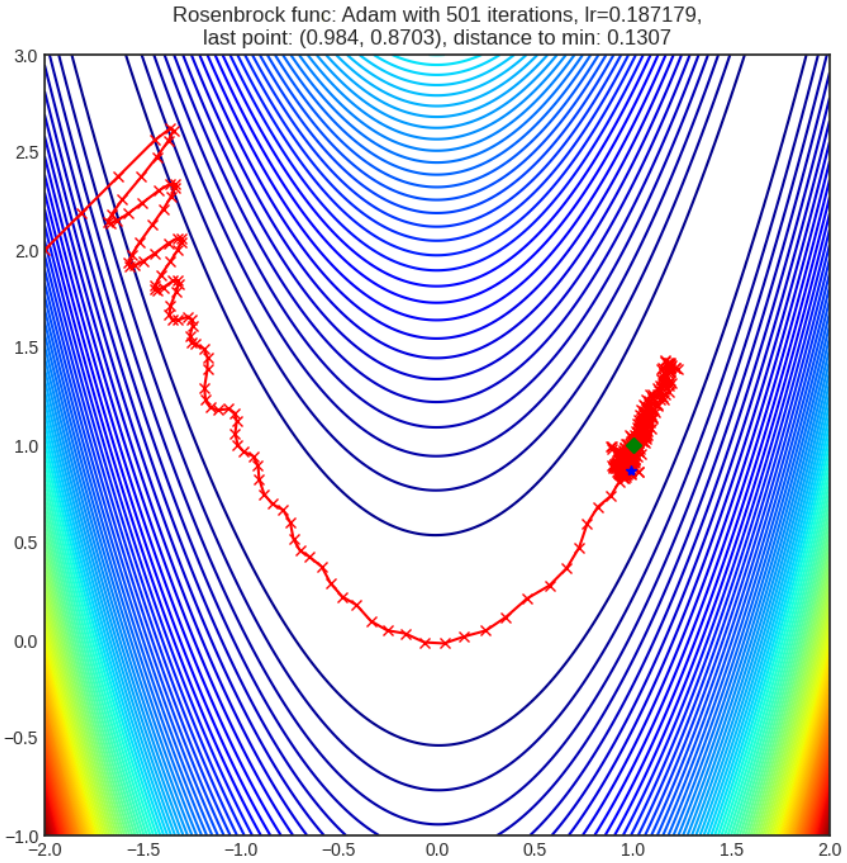
Свойства агентов

- **Автономность** — агенты движутся в пространстве поиска, хотя бы частично, независимо друг от друга.
- **Стохастичность** — процесс миграции агентов содержит случайную компоненту.
- **Ограниченность представления** — каждый из агентов популяции обладает информацией лишь об исследуемой им части области поиска и, быть может, об окружении некоторых других агентов.
- **Децентрализация** — отсутствие агентов, управляющих процессом поиска в целом.
- **Коммуникабельность** — агенты тем или иным способом могут обмениваться между собой информацией о топологии (ландшафте) оптимизируемой функции, выявленной в процессе исследования своей части области поиска.

Наглядный пример работы популяционного алгоритма



Наглядный пример работы популяционного алгоритма



Алгоритмы, вдохновлённые неживой природой

Поведение агентов популяционных алгоритмов, как мы с Вами обсудили чуть раньше, имеет аналогию в живой природе. Кроме этого, агенты алгоритмов могут быть похожи и на поведение нескольких **неживых** объектов, объединённых в одну группу.

К примеру, музыкальная **гармония** — это совокупность одновременно звучащих звуков. И звуки в этом случае, это и есть агенты, а гармония этих звуков — это популяция.

Алгоритмы, вдохновлённые неживой природой

Алгоритм поиска гармонии (Harmony Search algorithm, **HS**) предложен в 2001 г. Гимом (Z. W. Geem).

Алгоритм вдохновлен процессом поиска музыкантами гармонии в музыке. Ситуации идеальной гармонии звуков алгоритм HS ставит в соответствие глобальный экстремум в задаче многомерной оптимизации, а процессу импровизации музыканта - процедуру поиска этого экстремума.

Алгоритмы, вдохновлённые неживой природой

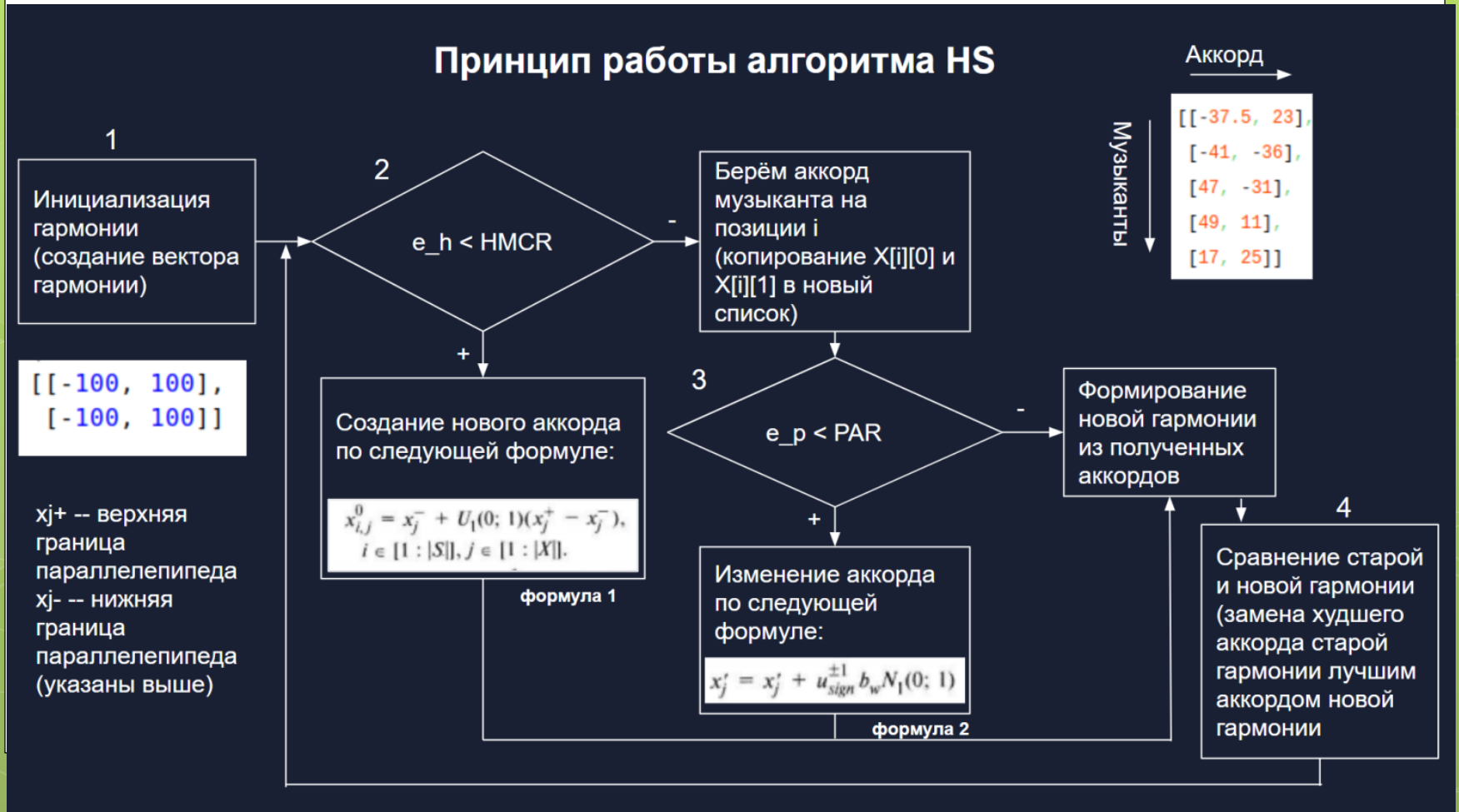
Рассмотрим задачу глобальной условной минимизации в гиперпараллелепипеде Π. Музыкантам ставим в соответствие индивидов $\boldsymbol{s_i}$, а оркестру — популяцию $\boldsymbol{S = \{s_i, i \in [1 : |S|]\}}$. Аккорду, который берет музыкант $\boldsymbol{s_i}$ в данный момент времени, сопоставляем значение вектора варьируемых параметров $\boldsymbol{X_i}$. Гармонию звуков формализует глобальный минимум фитнес-функции $\boldsymbol{\varphi(X)}$. Совокупность текущих координат $\boldsymbol{X_i}$, $\boldsymbol{i \in [1 : |S|]}$ образует $(|\boldsymbol{S}| \times |\boldsymbol{X}|)$-матрицу памяти гармоний (harmony memory) H^m.

Алгоритмы, вдохновлённые неживой природой

Шаги гармонического поиска:

- ✓ **Инициализируем алгоритм.**
- ✓ **Формируем вектор гармонии.**
- ✓ **Выполняем пошаговую настройку вектора гармонии.**
- ✓ **Обновляем матрицу памяти гармоний.**
- ✓ **Завершаем итерации, если условие окончания итераций выполнено; возвращаемся к шагу 2 в противном случае.**

Алгоритмы, вдохновлённые неживой природой



Алгоритмы, вдохновлённые неживой природой

```python
import random
import numpy as np
import matplotlib.pyplot as plt

HMCR = 0.7  # вероятность выбора из гармоник в памяти
PAR = 0.5  # вероятность модификации
BW = 0.04
counter = 0

harmony = [[-37.5, 23],
           [-41, -36],
           [47, -31],
           [49, 11],
           [17, 25]]

n = int(input("Iteration number = "))
dimensions = [[-100, 100],
              [-100, 100]]

res_plot = []

def fitness_function(v):
    return v[0] * v[0] + v[1] * v[1]  # нахождение фитнесс функции

```

Первые два этапа:

инициализация и создание гармонии

Алгоритмы, вдохновлённые неживой природой

Третий этап:

настройка одного вектора гармонии

```
24
25
26  while counter < n:
27      worst_i = 0  # индекс максимального элемента в старой гармонии (для редактирования элемента списка)
28      best_i = 0  # индекс минимального элемента в новой гармонии (для считывания конкретного элемента в списке)
29
30      i1 = np.random.randint(0, len(harmony), size=1)[0]
31      X1 = [None] * len(harmony[i1])
32      X1_flags = [False] * len(X1)
33      for j in range(len(harmony[i1])):
34          if random.random() < HMCR:  # e_h < HMCR
35              X1[j] = harmony[i1][j]
36              X1_flags[j] = True
37          else:
38              U1 = random.random()
39              X1[j] = dimensions[j][0] + U1 * (dimensions[j][1] - dimensions[j][0])
40
41      for j in range(len(X1)):
42          if X1_flags[j] and random.random() < PAR:  # e_p < PAR
43              N1 = np.random.normal()
44              u_sign = random.choice([-1, 1])
45              X1[j] = X1[j] + u_sign * BW * N1
46
```

Алгоритмы, вдохновлённые неживой природой

```
47  # нахождение худшего элемента в старой гармонии
48  worst_result = -65000
49  for i in range(len(harmony)):
50      res = fitness_function(harmony[i])
51      if res > worst_result:
52          worst_result = res
53          worst_i = i
54
55  if fitness_function(X1) < worst_result:
56      harmony[worst_i] = X1
57
58  counter += 1
59  min_i = -1
60  min_result = 65000
61  for i in range(len(harmony)):
62      res = fitness_function(harmony[i])
63      if res < min_result:
64          min_result = res
65          min_i = i
66
67  print(harmony[min_i])
68  res_plot.append(harmony[min_i])
69  print(min_result)
```

Четвёртый этап:

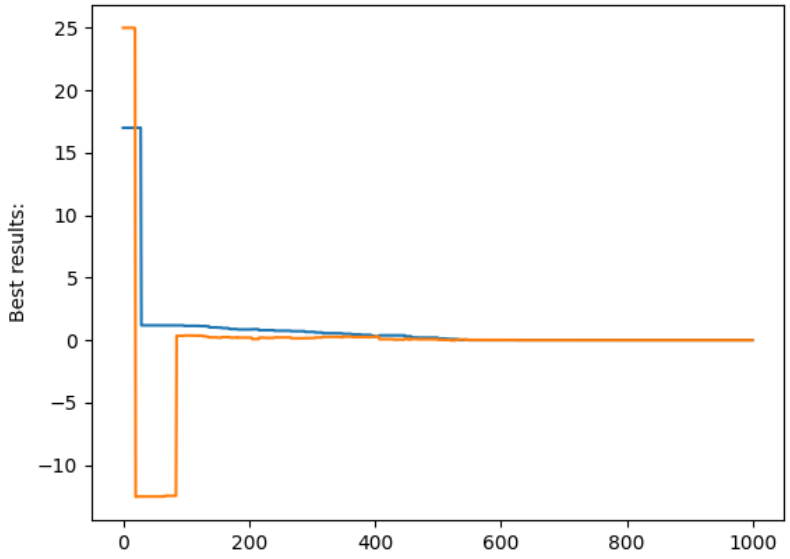
обновляем матрицу гармонии

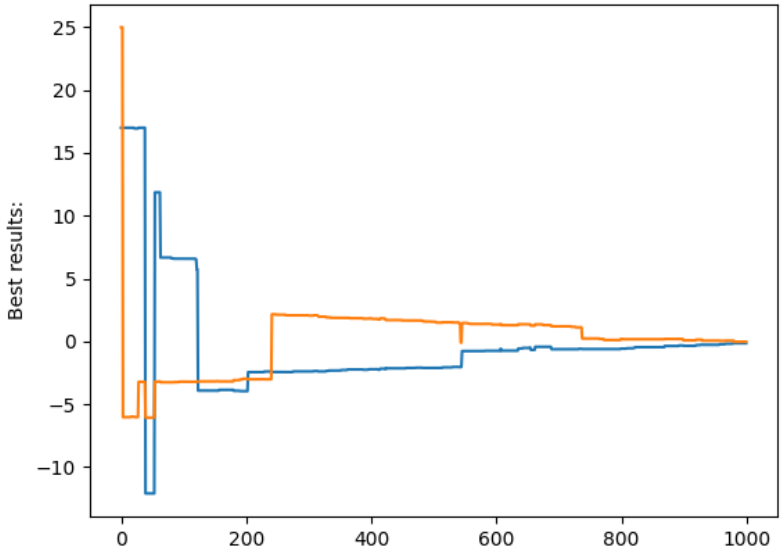
Алгоритмы, вдохновлённые неживой природой

Пятым и последним этапом мы проверяем условие завершения алгоритма: если нужное кол-во итераций достигнуто, то мы завершаем цикл и выходим из программы; если же условие не выполняется — возвращаемся к третьему этапу (делаем следующую итерацию в цикле).

Алгоритмы, вдохновлённые неживой природой

Результаты выполнения алгоритма HS:





Алгоритмы, вдохновлённые живой природой

Есть ещё один вид популяционных алгоритмов оптимизации, который называется бактериальной оптимизацией.

Известны примеры использования таких алгоритмов для решения прикладных задач из различных предметных областей, например, задачи оптимального управления динамической системой, задачи обучения нейронной сети и т. д. Принято считать, что метод бактериальной оптимизации предложен в работе Пассино (K. M Passino) в 2002 г.

Алгоритмы, вдохновлённые живой природой

Поведение бактерий обусловлено механизмом, который называется **бактериальным хемотаксисом** (bacterial chemotaxis) и представляет собой двигательную реакцию микроорганизмов на химический раздражитель. Данный механизм позволяет бактерии двигаться по направлениям **к аттрактантам** (чаще всего, питательным веществам) и **от репеллентов** (потенциально вредных для бактерии веществ).

В контексте задачи поисковой оптимизации, бактериальный хемотаксис можно также интерпретировать как механизм оптимизации использования бактерией известных пищевых ресурсов и поиска новых, потенциально **более ценных областей**.

Алгоритмы, вдохновлённые живой природой

```python
import numpy as np
import random
import matplotlib.pyplot as plt
import math

res_plot = [
    [[],[]],
    [[],[]],
    [[],[]],
    [[],[]],
    [[],[]],
    [[],[]]
]

def fitness_function(x):
    return (3*x*x-49)  # нахождение фитнесс функции

h = 0.1  # шаг
e = 0.4  # общая вероятность (при вычислениях)
x_k1 = -50  # нижняя граница области появления бактерий
x_k2 = 50  # верхняя граница области появления бактерий
t = 500  # число итераций (шагов хемотаксиса)
n = 6  # кол-во бактерий в нашей популяции
```

```python
X = [
    [60,-55],
    [50,20],
    [-30,-45],
    [0,45],
    [-50,-50],
    [-30,0]
] # координаты 5-ти бактерий
V = [
    [1,0],
    [0,-1],
    [-1,1],
    [0,1],
    [1,1],
    [-1,-1]
] # направляющие векторы всех наших бактерий

def norma(Vi): # вычисление нормы вектора
    if (max(Vi) != 0):
        return max(Vi)
    return 1;

## Определение начальных координат бактерий:
for i in range(n):
```

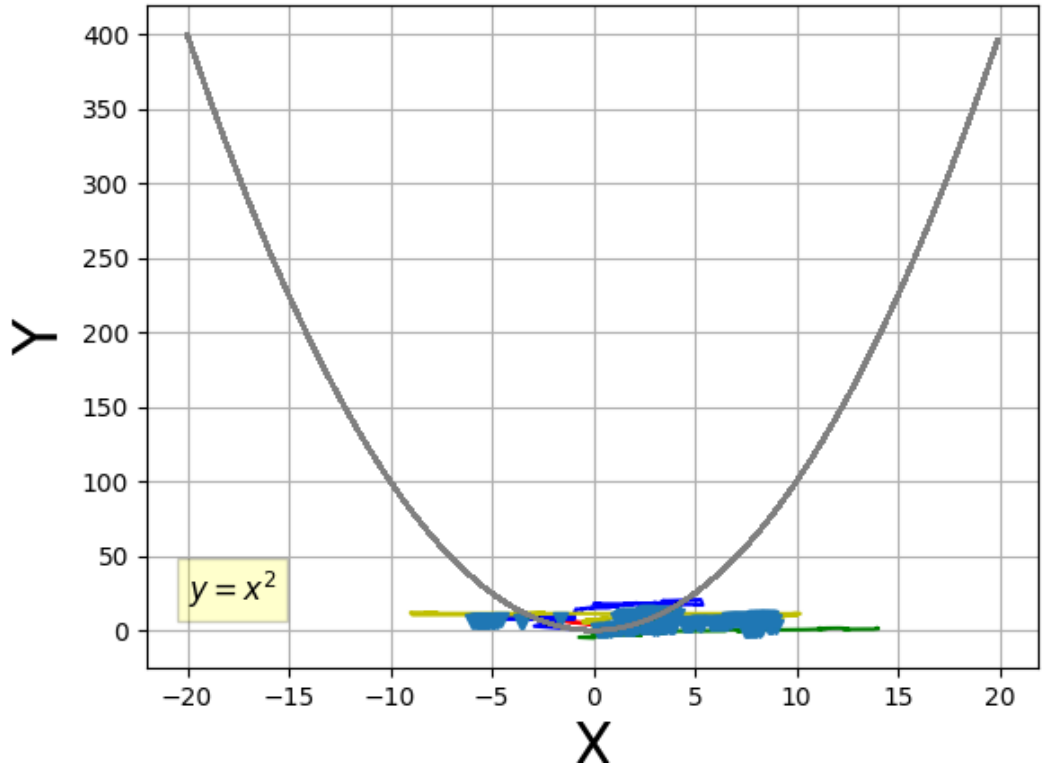
Алгоритмы, вдохновлённые живой природой

```
47  ## Определение начальных координат бактерий:
48  for i in range(n):
49      X[i][0] = X[i][0] + h * V[i][0]/norma(V[i])
50      X[i][1] = X[i][1] + h * V[i][1]/norma(V[i])
51
52  for i in range(t):
53      ## Репродукция на итерациях с номером, кратным десяти:
54      for j in range(n):
55          if (i%10 == 0) and (i != 0):
56              health = []
57              for k in range(n):
58                  s = 0
59                  for m in range(len(res_plot[k][0])):
60                      s += res_plot[k][1][m]
61                  health.append([k,s])
62              health.sort(key = lambda x : x[1])
63              new_agents = n//2
64              for k in range(n - new_agents, n):
65                  rand = random.randint(0, n - new_agents - 1);
66                  X[health[k][0]][0] = X[rand][0];
67                  X[health[k][0]][1] = X[rand][1];
68
```

```
69          ## Если j-я бактерия ушла от линии графика
70          ## достаточно далеко (более чем на пять единиц),
71          ## то меняем направляющий вектор (а значит направление этой бактерии)
72          ## случайным образом:
73          if ((X[j][1] - fitness_function(X[j][0]) < -1) or (X[j][1] - fitness_function(X[j][0]) > 1)):
74              V[j][0] = random.random() * random.choice([-1,1])  # компоненты направляющего вектора могут быть от -1 до 1
75              V[j][1] = random.random() * random.choice([-1,1])
76
77          ## Передвигаем j-ую бактерию на следующую позицию
78          ## согласно направляющему вектору:
79          X[j][0] = X[j][0] + h * V[j][0]/norma(V[j])
80          X[j][1] = X[j][1] + h * V[j][1]/norma(V[j])
81
82          ## Добавляем очередную точку нахождения j-й бактерии
83          ## для последующего построения линии её движения:
84          res_plot[j][0].append(X[j][0])
85          res_plot[j][1].append(X[j][1])
86
87  ## Построение графика движения всех бактерий:
88  for i in range(n):
89      style = 'r'
90      rand = random.random()
91      if (i == 1):
92          style = 'g'
```

Алгоритмы, вдохновлённые живой природой

Первый тест: **f(x) = x^2**



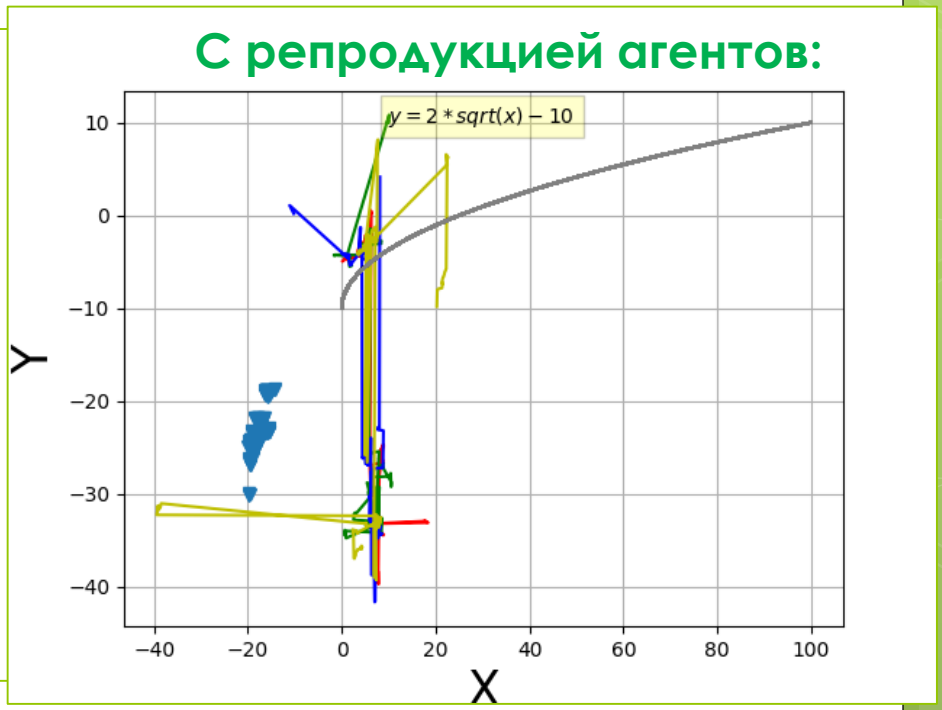
Простейший алгоритм с бактериями: без этапа репродукции бактерий, всего пять особей в популяции и 200 итераций цикла

Алгоритмы, вдохновлённые живой природой

Второй тест: **f(x) = $2\sqrt{x} - 10$**

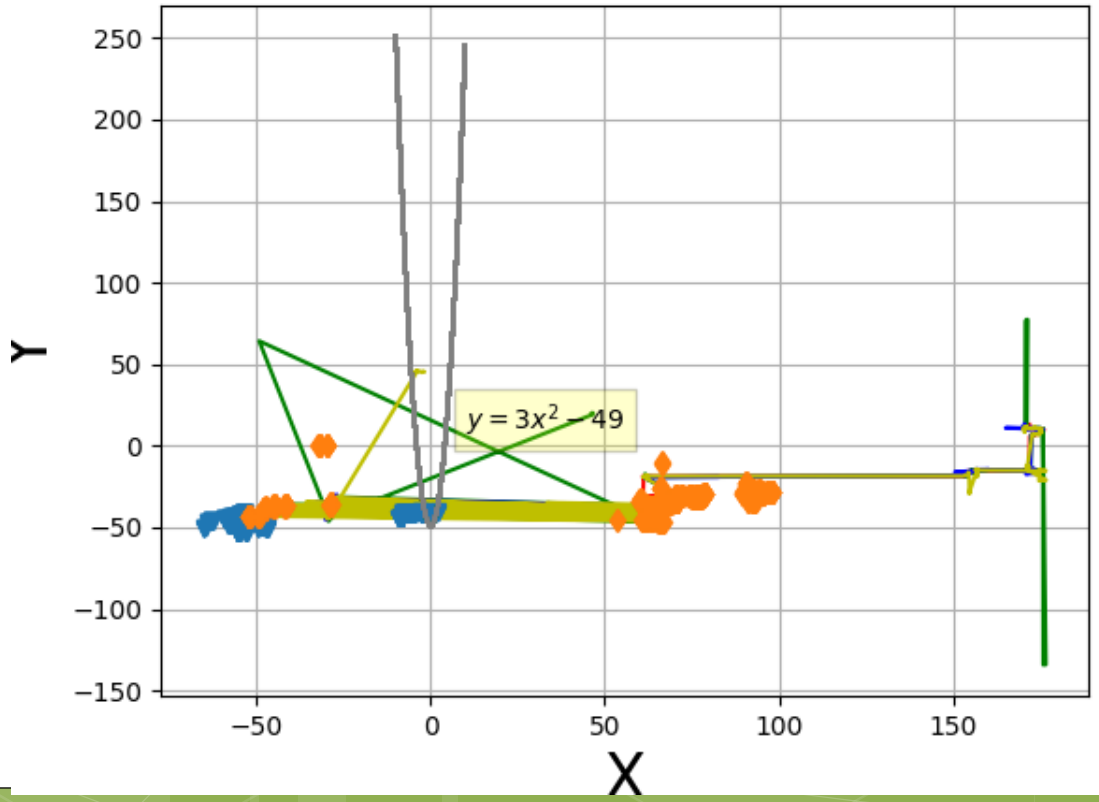
Без этапа репродукции:

С репродукцией агентов:



Алгоритмы, вдохновлённые живой природой

Третий тест: $f(x) = 2x^2 - 49$



Список использованных источников

- Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. - 2-е изд. - Москва : Издательство МГГУ им. Н. Э. Баумана, 2017. -446, [2] с. : ил.
- Копнём поглубже: сравниваем популярные алгоритмы оптимизации с менее известными // Хабр URL: https://habr.com/ru/company/prequel/blog/568496/ (дата обращения: 15.09.2021).
- Другие полезные материалы, предоставленные преподавателем.

Спасибо Вам за внимание!