

Стохастические методы оптимизации

Генетический алгоритм, метод роя частиц,
муравьиный алгоритм

Павел
Ломовицкий

$$\min_{\xi \in \Omega} f(\xi) - ?$$

$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$

$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$

а мы точно по
антиградиент
у крутим?..

эх, скорей
бы обед!

Проблема:

Не всегда есть
информация о
градиенте функции ∇f



$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$

$$\begin{aligned}\frac{\partial f_2^{\alpha,\varepsilon}}{\partial S^\eta} &= (p^\eta - p^\varepsilon) \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial S^\eta}, \\ \frac{\partial f_2^{\alpha,\varepsilon}}{\partial p^\varepsilon} &= \frac{\partial q_\alpha^\varepsilon}{\partial p^\varepsilon}, \quad \frac{\partial f_2^{\alpha,\varepsilon}}{\partial S^\varepsilon} = \frac{\partial q_\alpha^\varepsilon}{\partial S^\varepsilon}, \\ \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial p^{\varepsilon,\eta}} &= t_\alpha^{\varepsilon,\eta} \left(\frac{1}{\rho_\alpha^{\varepsilon,\eta}} \frac{\partial \rho_\alpha^{\varepsilon,\eta}}{\partial p^{\varepsilon,\eta}} - \frac{1}{\mu_\alpha^{\varepsilon,\eta}} \frac{\partial \mu_\alpha^{\varepsilon,\eta}}{\partial p^{\varepsilon,\eta}} \right), \\ \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial S^{\varepsilon,\eta}} &= \frac{k_\alpha^{\varepsilon,\eta} A_\alpha^{\varepsilon,\eta}}{h_\alpha^{\varepsilon,\eta}} \frac{\rho_\alpha^{\varepsilon,\eta}}{\mu_\alpha^{\varepsilon,\eta}} \frac{\partial k_\alpha^{\varepsilon,\eta}}{\partial S^{\varepsilon,\eta}} = \frac{t_\alpha^{\varepsilon,\eta}}{k_\alpha^{\varepsilon,\eta}} \frac{\partial k_\alpha^{\varepsilon,\eta}}{\partial S^{\varepsilon,\eta}}, \\ \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial p^\varepsilon} &= \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial p^{\varepsilon,\eta}} \frac{\partial p^{\varepsilon,\eta}}{\partial p^\varepsilon} + \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial S^{\varepsilon,\eta}} \frac{\partial S^{\varepsilon,\eta}}{\partial p^\varepsilon}, \\ \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial S^\varepsilon} &= \frac{\partial t_\alpha^{\varepsilon,\eta}}{\partial S^{\varepsilon,\eta}} \frac{\partial S^{\varepsilon,\eta}}{\partial S^\varepsilon}.\end{aligned}$$

Отчет
должен быть
готов на
этой неделе!

Проблема:

Не всегда возможно
посчитать ∇f за
разумное время



$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$

Если я легонько нажму
на эту красную
кнопочку, это будет
считаться малым
приращением?..



Проблема:

В модели могут быть
переменные, от
которых невозможно
взять градиент

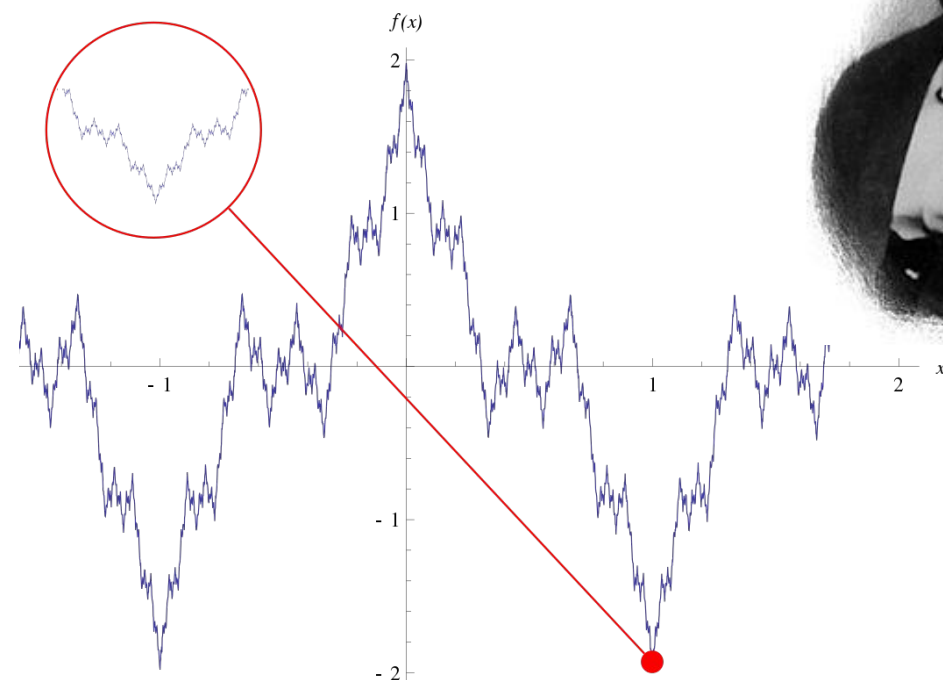
$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$

$$f(x) = \sum_{n=0}^{\infty} b^n \cos[a^n \pi x]$$



Проблема:

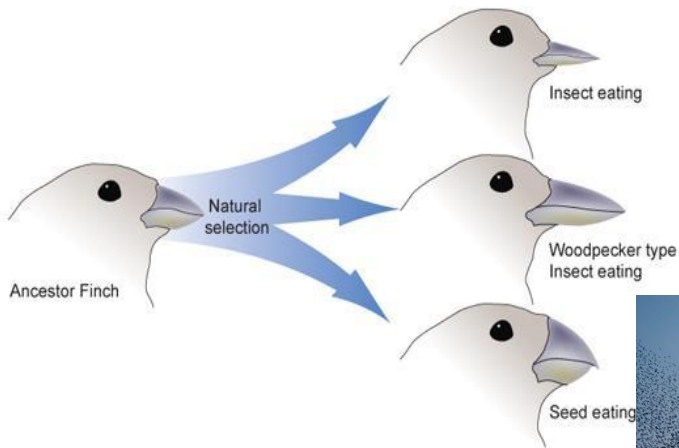
Вид оптимизируемой функции не позволяет эффективно применять градиенты



~~$$\xi^{n+1} = \xi^n - \alpha^n \nabla f(\xi^n)$$~~

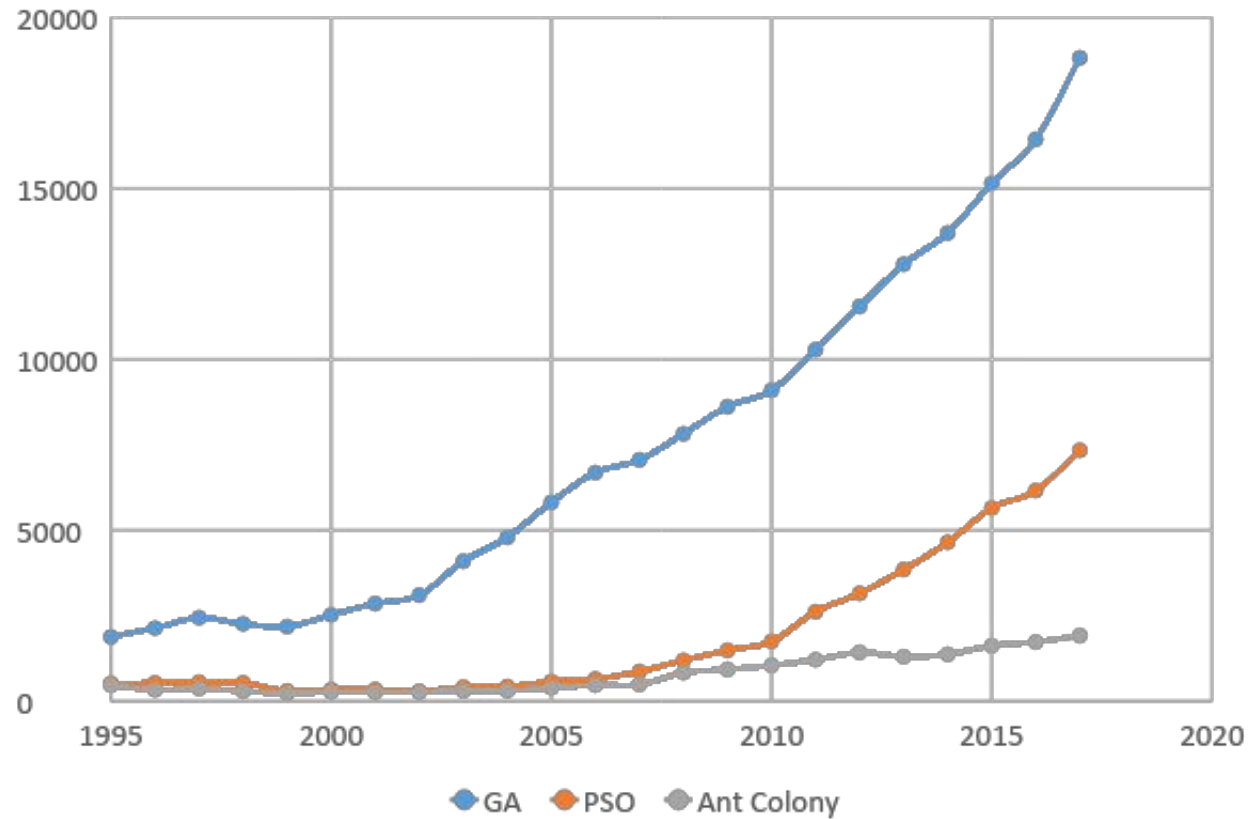
$$\xi^{n+1} = F(\xi^n, p)$$

Идеи подсмотрены у
роды!



Количество публикаций на Google Scholar: GA – 2,4 млн, PSO – 530 тыс., Ant Colony – 722 тыс.

На графике кол-во публикаций на ScienceDirect.com по годам



Eberhart R., Kennedy J. A new optimizer using particle swarm theory //Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on. – IEEE, 1995. – С. 39-43.

12223 цитирования

Kennedy J. Particle swarm optimization //Encyclopedia of machine learning. – Springer US, 2011. – С. 760-766.

48396 цитирований

Einstein A., Podolsky B., Rosen N. Can quantum-mechanical description of physical reality be considered complete? //Physical review. – 1935. – Т. 47. – №. 10. – С. 777.

16546 цитирований

PROS	CONS
Симулятор – «черный ящик»	Нет критерия сходимости
	Сходимость не строго монотонна
Разная природа параметров	Нужен опыт и инструменты, чтобы правильно настроить параметры
Намного быстрее перебора	
Exploration – поиск глобального оптимума	Результат зависит от целевой функции
Exploitation – локальная сходимость	Результат на совести пользователя
Легко параллелить и масштабировать	Вычислительная сложность от размерности задачи растет нелинейно
Выявляются ошибки в «черном ящике»	

Генетический алгоритм – один из первых эволюционных алгоритмов.

От вектора оптимизируемых параметров x переходим к понятию *хромосомы*. В хромосоме закодирована информация с помощью алфавита, например, двоичного или десятичного кода.

Аналогия с ДНК, состоящей из 4-х оснований: аденина (А), гуанина (Г), цитозина (Ц) и тимина (Т).

(00011001011000010000001101000111)
(25970371)

Набор хромосом, принадлежащий одной итерации оптимизационного алгоритма, называется *поколением*.

Сравнение хромосом

Вместо задачи $\min_{\xi \in \Omega} f(\xi)$ будем решать задачу вида $\max_{x \in \tilde{\Omega}} fit(x)$

$fit(x)$ – *fitness function*, целевая функция. Последующие действия над хромосомами будут зависеть от того, насколько они «приспособлены» относительно остальных.

При инициализации алгоритма создается первое поколение хромосом из допустимой области $\tilde{\Omega}$ (как правило, случайным образом), инициализируются все необходимые параметры, вычисляется целевая функция для каждой хромосомы.


Характерные операции, совершаемые в ГА на каждой итерации, операторы (genetic operators):

Elitism – перенос лучших хромосом в новое поколение без изменений

Crossover – формирование дочерней хромосомы из пары родительских

Mutation – небольшое изменение части хромосомы для поиска глобального оптимума

Elitism – одна или несколько хромосом переходят в новое поколение без изменений. Значение целевой функции для них уже известно, нет необходимости в пересчете. Благодаря данному оператору решение на итерации $n + 1$ будут как минимум не хуже решения, найденного к итерации n .

№	<i>fit</i>		№	<i>fit</i>
x_1^n	0,56		x_1^{n+1}	12,8
x_2^n	6,24		x_2^{n+1}	9,94
x_3^n	12,8		—	—
x_4^n	3,14		—	—
x_5^n	0,01		—	—
x_6^n	6,87		—	—
x_7^n	9,94		—	—
x_8^n	1,12		—	—

Для формирования оставшейся части поколения используем оператор скрещивания (crossover).

1. Исходя из некоторых соображений выбирается пара родительских хромосом. Например, выбор может быть полностью случайным или детерминированным – использовать информацию о целевой функции.

$$P_j = \alpha \frac{fit_j}{\sum_{k=1}^M fit_k} + (1 - \alpha) \frac{1}{M}$$

P_j – вероятность j -ой хромосомы из родительского поколения попасть в пару, fit_j – значение целевой функции для нее, M – число хромосом в поколении, α – коэффициент детерминированности.

Для формирования оставшейся части поколения используем оператор скрещивания (crossover).

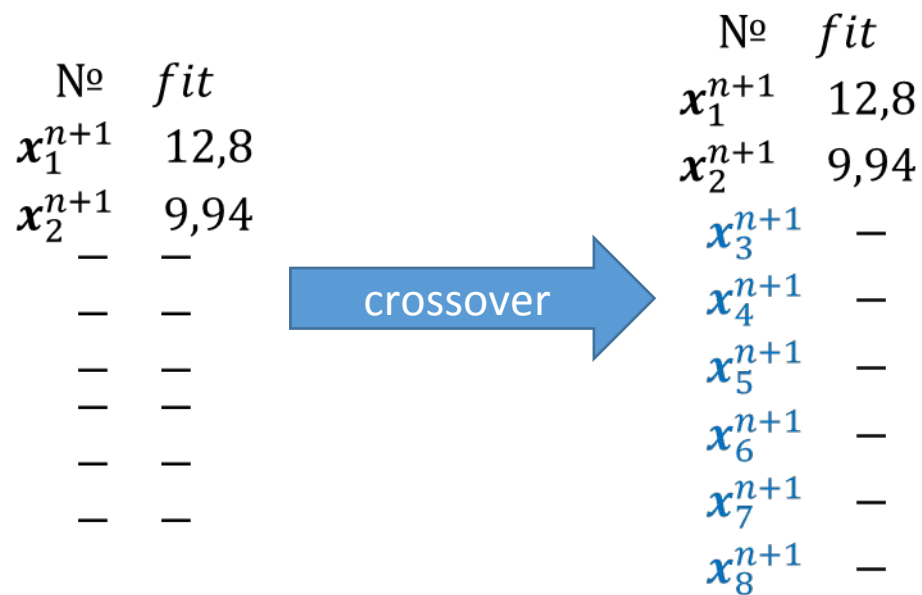
2. Получаем новую хромосому путем скрещивания. Возможные варианты:

$\begin{pmatrix} 25970371 \\ 31560058 \end{pmatrix} \xrightarrow{\text{crossover}} (25960058)$

$\begin{pmatrix} 25970371 \\ 31560058 \end{pmatrix} \xrightarrow{\text{crossover}} (21570051)$

Для формирования оставшейся части поколения используем оператор скрещивания (crossover).

3. Записываем полученные хромосомы в новое поколение



Хромосомы из дочернего поколения подвергаются мутациям, чтобы обеспечить вариативность поиска и иметь возможность алгоритму выйти из локального оптимума.

Важно: «элитные» частицы не подвергаются мутации, т.к. иначе мы можем потерять решение с высоким fitness value.

(25960058)  mutation (25960758)

(21570051)  mutation (41510021)

После того, как дочернее поколение окончательно сформировалось, для всех частиц в нем вычисляется значение целевой функции. Далее оно становится родительским и начинается следующая итерация генетического алгоритма.

№	<i>fit</i>		№	<i>fit</i>		№	<i>fit</i>
x_1^{n+1}	12,8		x_1^{n+1}	12,8		x_1^{n+1}	12,8
x_2^{n+1}	9,94		x_2^{n+1}	9,94		x_2^{n+1}	9,94
x_3^{n+1}	—		x_3^{n+1}	—		x_3^{n+1}	7,62
x_4^{n+1}	—	mutation	x_4^{n+1}	—	fitness evaluation	x_4^{n+1}	1,89
x_5^{n+1}	—		x_5^{n+1}	—		x_5^{n+1}	19,6
x_6^{n+1}	—		x_6^{n+1}	—		x_6^{n+1}	11,4
x_7^{n+1}	—		x_7^{n+1}	—		x_7^{n+1}	0,03
x_8^{n+1}	—		x_8^{n+1}	—		x_8^{n+1}	9,54

```
initialization
while !termination_condition
  for i = 1 : nElite
    newGeneration[i] <- oldGeneration[i]
  end
  for i = nElite+1 : M
    x1,x2 <- choosePair(oldGeneration)
    newGeneration[i] <- crossover(x1,x2)
    if mutation_criterion
      mutate(newGeneration[i])
    end
  end
  end
  oldGeneration <- newGeneration
  newGeneration <- ∅

  evaluateFitnessValues(oldGeneration)
  sortByFitnessValues(oldGeneration)
  bestChromosome <- oldGeneration[i]
end
return fit(bestChromosome)
```

```
initialization
while !termination_condition
  for i = 1 : nElite
    newGeneration[i] <- oldGeneration[i]
  end
  for i = nElite+1 : M
    x1, x2 <- choosePair(oldGeneration)
    newGeneration[i] <- crossover(x1, x2)
    if mutation_criterion
      mutate(newGeneration[i])
    end
  end
  oldGeneration <- newGeneration
  newGeneration <- ∅

  evaluateFitnessValues(oldGeneration)
  sortByFitnessValues(oldGeneration)
  bestChromosome <- oldGeneration[i]
end
return fit(bestChromosome)
```

Каким его выбрать?

Сколько взять?

Размер

поколения?

Как выбрать пару?

Как проводить мутацию?

А может, мутацию до
кроссовера?

Насколько сильно?

Метод роя частиц воспроизводит поведение стаи птиц, роя насекомых или косяка рыб, исследующих территорию в поисках пищи. Рой является децентрализованной и самоорганизующейся системой.

Будем называть кодированный вектор оптимизируемых параметров x *частицей*. Понятие целевой функции останется без изменений. Частицы на одной итерации будем называть *роем*. Все частицы в рое движутся по простому закону движения:

$$x^{n+1} = x^n + v^{n+1}, v - \text{скорость частицы.}$$

В памяти компьютера на каждой итерации хранятся x , v и x_{best} - лучшее (с точки зрения значения целевой функции) положение частицы за все итерации.

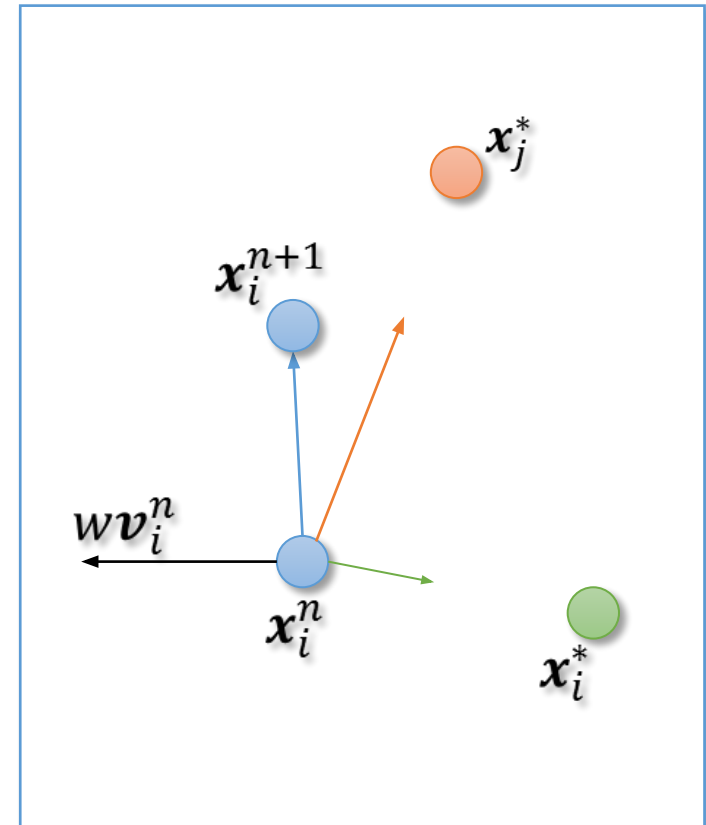
Скорость \mathbf{v}_i^{n+1} складывается из *инерциальной*, *когнитивной* и *социальной* компонент:

$$\mathbf{v}_i^{n+1} = w\mathbf{v}_i^n + c_1\mathbf{r}_1(\mathbf{x}_i^* - \mathbf{x}_i^n) + c_2\mathbf{r}_2(\mathbf{x}_j^* - \mathbf{x}_i^n)$$

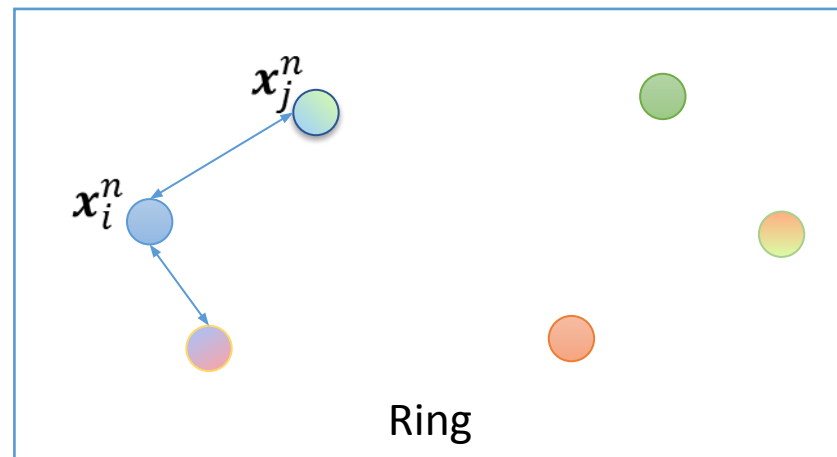
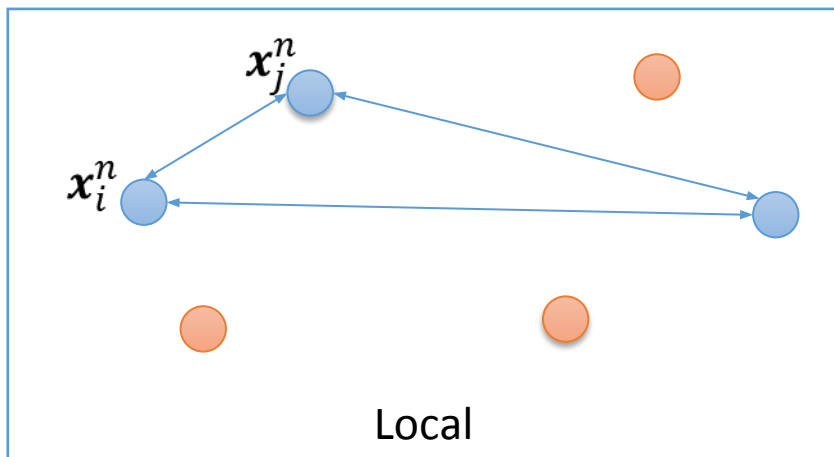
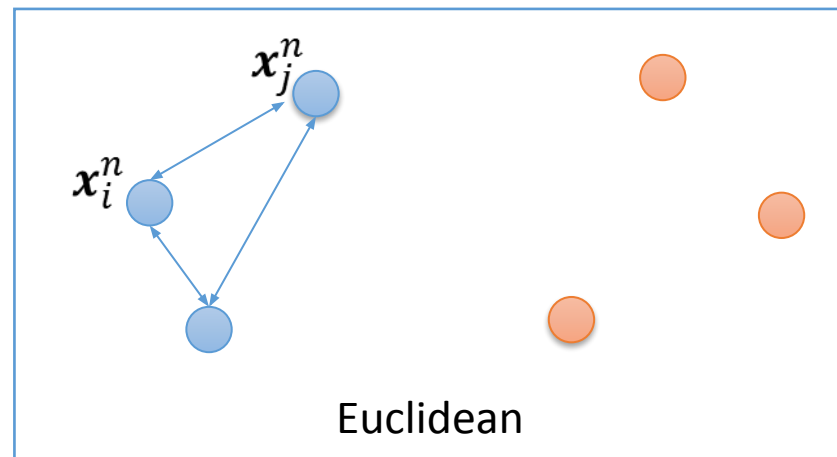
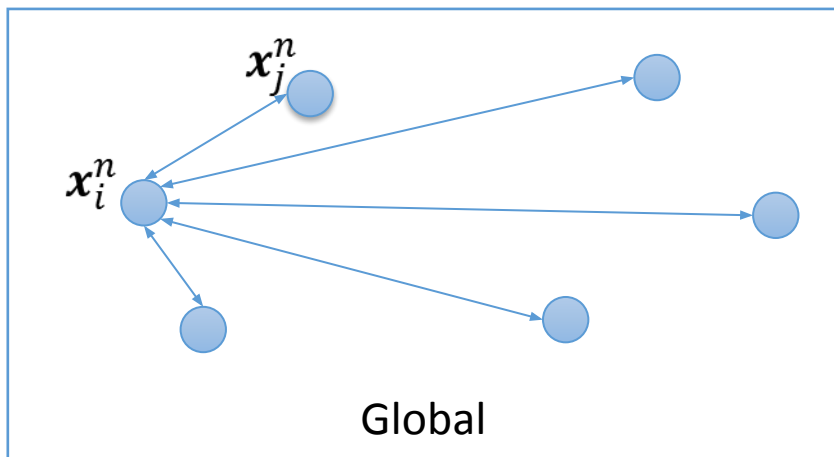
w, c_1, c_2 – весовые коэффициенты, $\mathbf{r}_1, \mathbf{r}_2$ – вектора случайных величин от 0 до 1.

\mathbf{x}_i^* – лучшее решение, найденное частицей i за все итерации.

\mathbf{x}_j^* – лучшее решение, найденное всеми частицами из *окрестности* i -ой частицы за все итерации.



Соседство частиц (топологии окрестностей) определяет, каким образом они будут обмениваться информацией. Самый популярный вариант – global, он же gbest, все частицы взаимодействуют со всеми.



$$\begin{aligned} \mathbf{x}^{n+1} &= \mathbf{x}^n + \mathbf{v}^{n+1} \\ \mathbf{v}_i^{n+1} &= w\mathbf{v}_i^n + c_1\mathbf{r}_1(\mathbf{x}_i^* - \mathbf{x}_i^n) + c_2\mathbf{r}_2(\mathbf{x}_j^* - \mathbf{x}_i^n) \end{aligned}$$

Параметры w , c_1 и c_2 , а так же выбранная топология, определяют движение роя. Нам хотелось бы избежать его «схлопывания» или «разлета», а также (по возможности) стремления к глобальному оптимуму.

Примеры работ, в которых изучается поведение системы и даются рекомендации по устойчивости:

Trelea I. C. The particle swarm optimization algorithm: convergence analysis and parameter selection //Information processing letters. – 2003. – Т. 85. – №. 6. – С. 317-325.

Clerc M., Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space //IEEE transactions on Evolutionary Computation. – 2002. – Т. 6. – №. 1. – С. 58-73.

Основная идея: пусть при стремлении $n \rightarrow \infty$ вместо случайных величин можно взять матожидание, все лучшие положения x_i^* будут соответствовать положению глобального оптимума g^* . Тогда алгоритм станет детерминированным, в одномерном случае запишется как

$$\begin{pmatrix} x^{n+1} \\ v^{n+1} \end{pmatrix} = \begin{pmatrix} 1-b & w \\ -b & w \end{pmatrix} \begin{pmatrix} x^n \\ v^n \end{pmatrix} + \begin{pmatrix} b \\ b \end{pmatrix} g^*, b = w \frac{c_1 + c_2}{2}.$$

Исследуя собственные значения матрицы $A = \begin{pmatrix} 1-b & w \\ -b & w \end{pmatrix}$ можно получить соотношения на w, c_1, c_2 при которых динамическая система будет вести себя определенным образом в окрестности глобального оптимума.

Исходя из данного анализа, можно вывести некоторые рекомендации, в литературе такой алгоритм будет называться PSO with *constriction coefficients*.

$$\begin{aligned} \mathbf{v}_i^{n+1} &= w\mathbf{v}_i^n + c_1\mathbf{r}_1(\mathbf{x}_i^* - \mathbf{x}_i^n) + c_2\mathbf{r}_2(\mathbf{x}_j^* - \mathbf{x}_i^n) \\ \mathbf{v}_i^{n+1} &= w[\mathbf{v}_i^n + \tilde{c}_1\mathbf{r}_1(\mathbf{x}_i^* - \mathbf{x}_i^n) + \tilde{c}_2\mathbf{r}_2(\mathbf{x}_j^* - \mathbf{x}_i^n)] \end{aligned}$$

$$w = \frac{2\kappa}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \varphi = \tilde{c}_1 + \tilde{c}_2 > 4, \kappa \in [0,1]$$

Как правило, $\kappa = 1$, $\tilde{c}_1 = \tilde{c}_2 = 2,05$, $\varphi = 4,1$ и

$$w \approx 0,7298 \quad c_1 = c_2 \approx 1,4962$$

Алгоритм с данными параметрами хорошо работает на большинстве задач при топологии gbest.

```
initialization
while !termination_condition
    for i = 1 : nNeighborhoods
        updateNeighborhood(i)
    end
    for i = 1 : nParticles
        xbest <- bestSolution[i]
        v_c <- c1*rand*(xbest - x[i]) // cognitive
        k <- findNeighborhood(i)
        nbest <- bestSolutionInNeighborhood(k)
        v_s <- c2*rand*(nbest - x[i]) // social
        v[i] <- w*v[i] + v_c + v_s
    end
    for i = 1 : nParticles
        x[i] = x[i] + v[i]
        if fit(x[i]) > bestFitness[i]
            bestFitness[i] <- fit(x[i])
            bestSolution[i] <- x[i]
        end
    end
end
end
n = argmax(bestFitness)
return bestSolution[n]
```

```
initialization
while !termination_condition
  for i = 1 : nNeighborhoods
    updateNeighborhood(i)
  end
  for i = 1 : nParticles
    xbest <- bestSolution[i]
    v_c <- c1*rand*(xbest - x[i]) // cognitive
    k <- findNeighborhood(i)
    nbest <- bestSolutionInNeighborhood(k)
    v_s <- c2*rand*(nbest - x[i]) // social
    v[i] <- w*v[i] + v_c + v_s
  end
  for i = 1 : nParticles
    x[i] = x[i] + v[i]
    if fit(x[i]) > bestFitness[i]
      bestFitness[i] <- fit(x[i])
      bestSolution[i] <- x[i]
    end
  end
end
end
n = argmax(bestFitness)
return bestSolution[n]
```

Каким его выбрать?

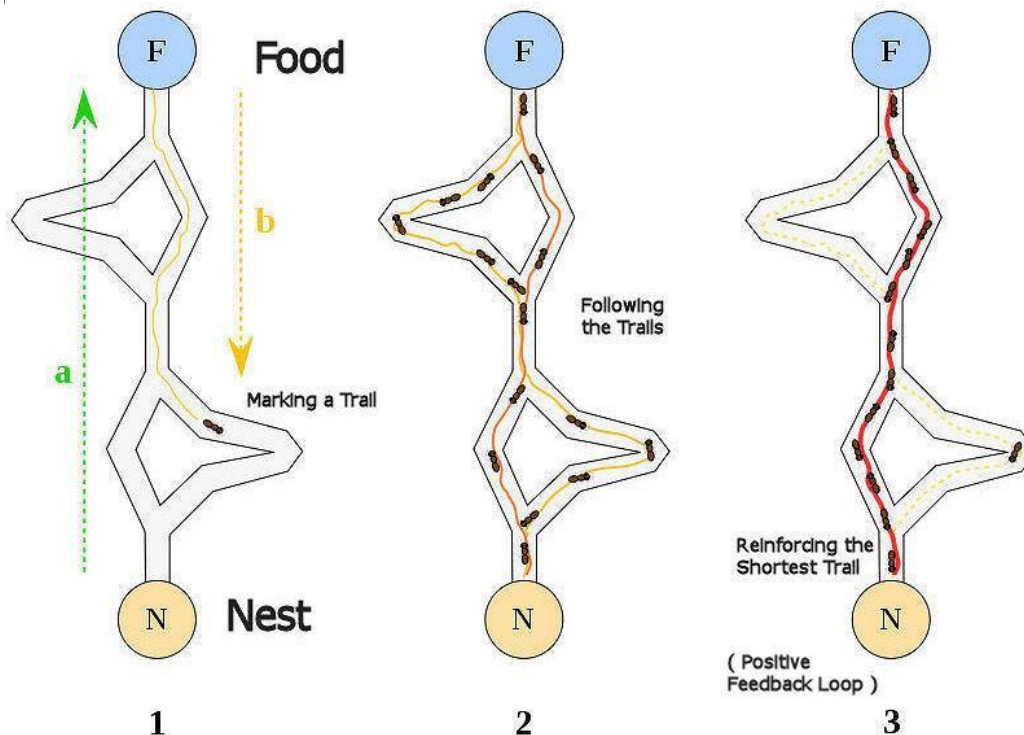
Как задать окрестности?

Сколько частиц взять?

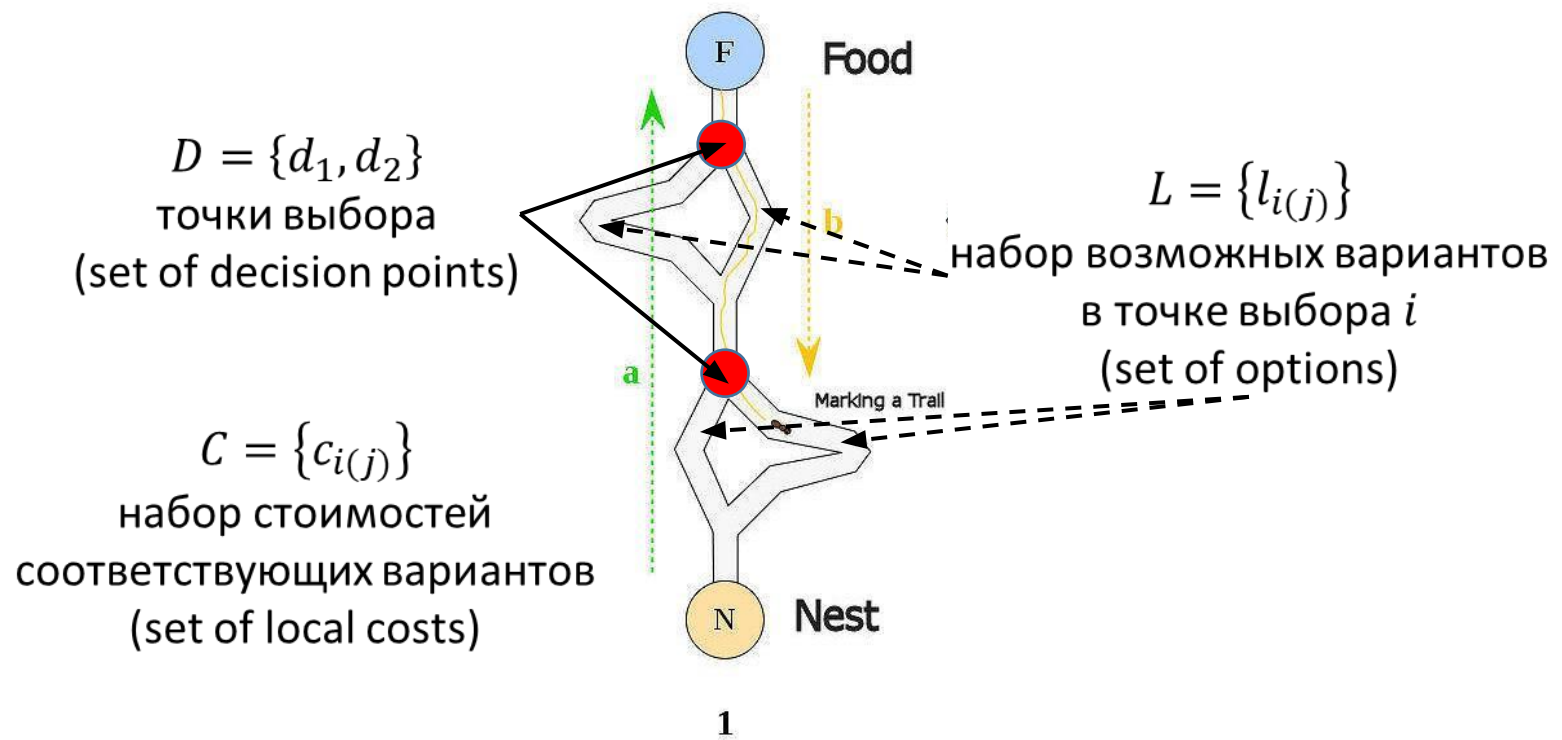
Как выбрать
коэффициенты?

Алгоритм оптимизации подражанием муравьиной колонии, муравьиный алгоритм (Ant Colony Optimization) описывает поведение колонии муравьев, отправляющихся из своего жилища к источнику пищи и обратно.

Цель – минимизация пройденного пути



В процессе возвращения от источника пищи муравей несколько раз делает *выбор* – в какую сторону ему пойти.



На каждой итерации алгоритма M муравьев проходит свой путь, формирующийся случайным образом по обновляющемуся распределению вероятностей. В литературе один такой проход (т.е. выбор параметров и вычисление целевой функции) обычно называют циклом.

Вероятность того, что будет выбран вариант $l_{i(j)}$ в цикле k на итерации t :

$$p_{i(j)}(k, t) = \frac{[\tau_{i(j)}(t)]^\alpha [\eta_{i(j)}]^\beta}{\sum_{i(j)} [\tau_{i(j)}(t)]^\alpha [\eta_{i(j)}]^\beta}$$

$\tau_{i(j)}(t)$ – концентрация *феромонов*

$\eta_{i(j)}$ – *видимость* варианта (локальная эвристика)

α, β – некоторые коэффициенты

В самой простой постановке $\alpha = 1, \beta = 0, \eta_{i(j)} = 1/c_{i(j)}$

$$p_{i(j)}(k, t) = \frac{[\tau_{i(j)}(t)]^\alpha [\eta_{i(j)}]^\beta}{\sum_{i(j)} [\tau_{i(j)}(t)]^\alpha [\eta_{i(j)}]^\beta}$$

На самой первой итерации алгоритма все $\tau_{i(j)}$ равны между собой. Если $\beta = 0$, что соответствует случаю «отсутствия зрения» у муравьев, то и выбор происходит равновероятно в каждом цикле. После формирования списка $x_k(t)$ вычисляется целевая функция $f(x_k(t))$.

По окончании итерации (M циклов) происходит *обновление феромонов*:

$$\tau_{i(j)}(t + 1) = \rho \tau_{i(j)}(t) + \Delta \tau_{i(j)}(t)$$

ρ – коэффициент испарения, $\rho < 1$

Существует много способов вычислять добавку $\Delta\tau_{i(j)}$. Основная идея в том, что решения с меньшим значением целевой функции должны вносить больший вклад, чтобы на следующей итерации вероятность пойти по более короткому пути увеличилась.

$$\Delta\tau_{i(j)} = \sum_{k=1}^M \Delta\tau_{i(j)}^k$$

$\Delta\tau_{i(j)} = \Delta\tau_{i(j)}^{k^*}, k^*$ – цикл с лучшим решением на итерации t

$$\Delta\tau_{i(j)}^k = \begin{cases} \frac{R}{f(\mathbf{x}_k)}, & \text{если } l_{i(j)} \text{ был выбран в цикле } k, \\ 0, & \text{иначе.} \end{cases}$$



```
initialization
p <- formP(tau,eta,alpha,beta)
while !termination_condition
  deltaTau <-  $\emptyset$ 
  for i = 1 : M
    x <- choosePath(p)
    cost <- f(x)
    if cost < bestCost
      bestCost <- cost
    end
    updateDTau(deltaTau,cost)
  end
  tau <- rho*tau + deltaTau
  p <- formP(tau,eta,alpha,beta)
end
return bestCost
```

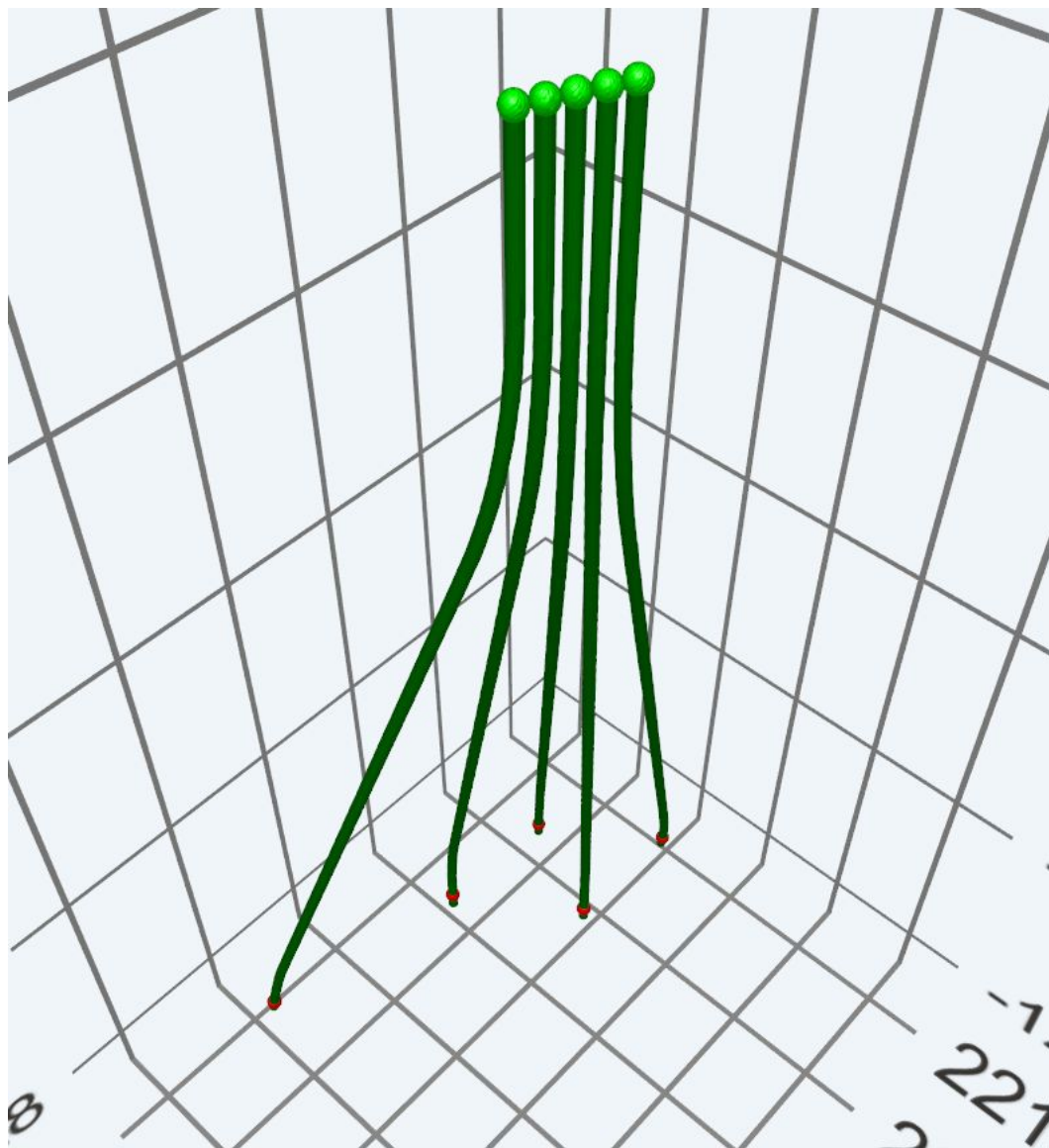
```
initialization
p <- formP(tau,eta,alpha,beta)
while !termination_condition
  deltaTau <- 0
  for i = 1 : M
    x <- choosePath(p)
    cost <- f(x)
    if cost < bestCost
      bestCost <- cost
    end
  end
  updateDTau(deltaTau,cost)
end
tau <- rho*tau + deltaTau
p <- formP(tau,eta,alpha,beta)
end
return bestCost
```

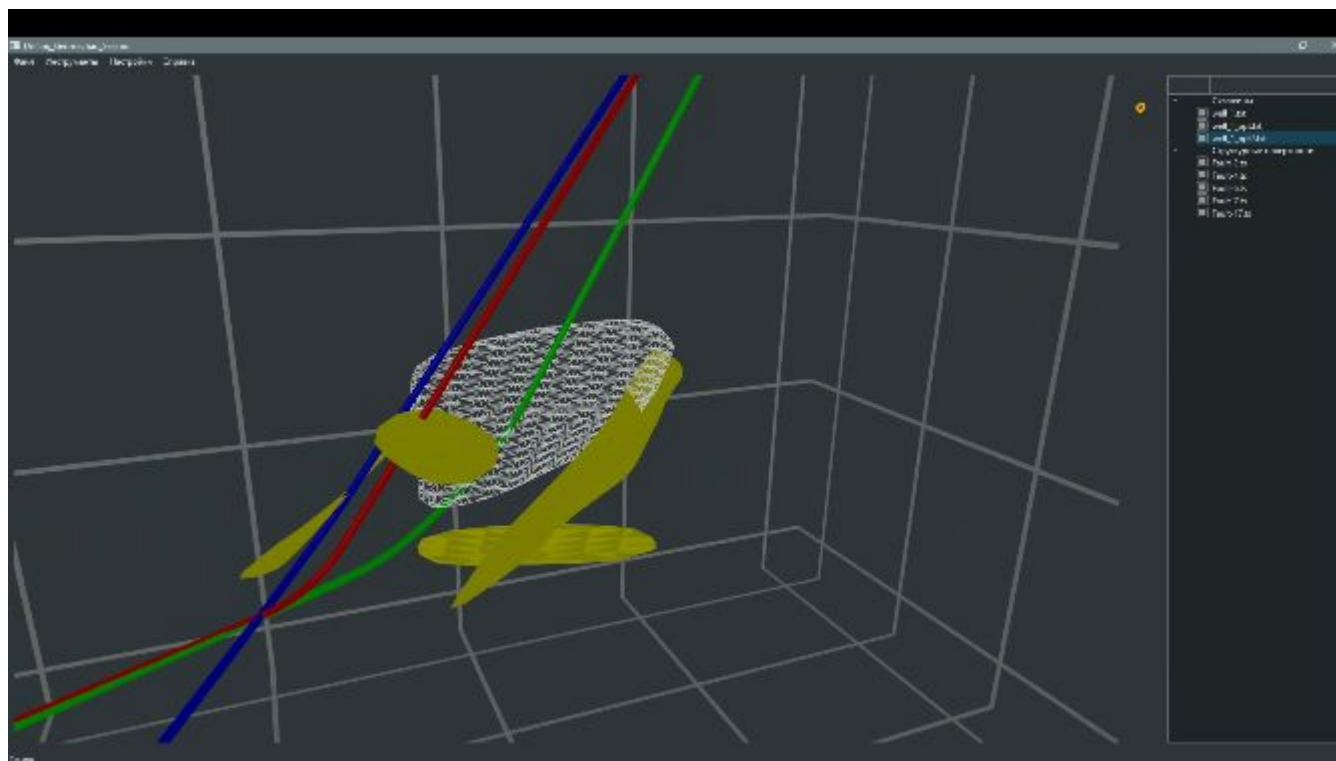
Каким его выбрать?

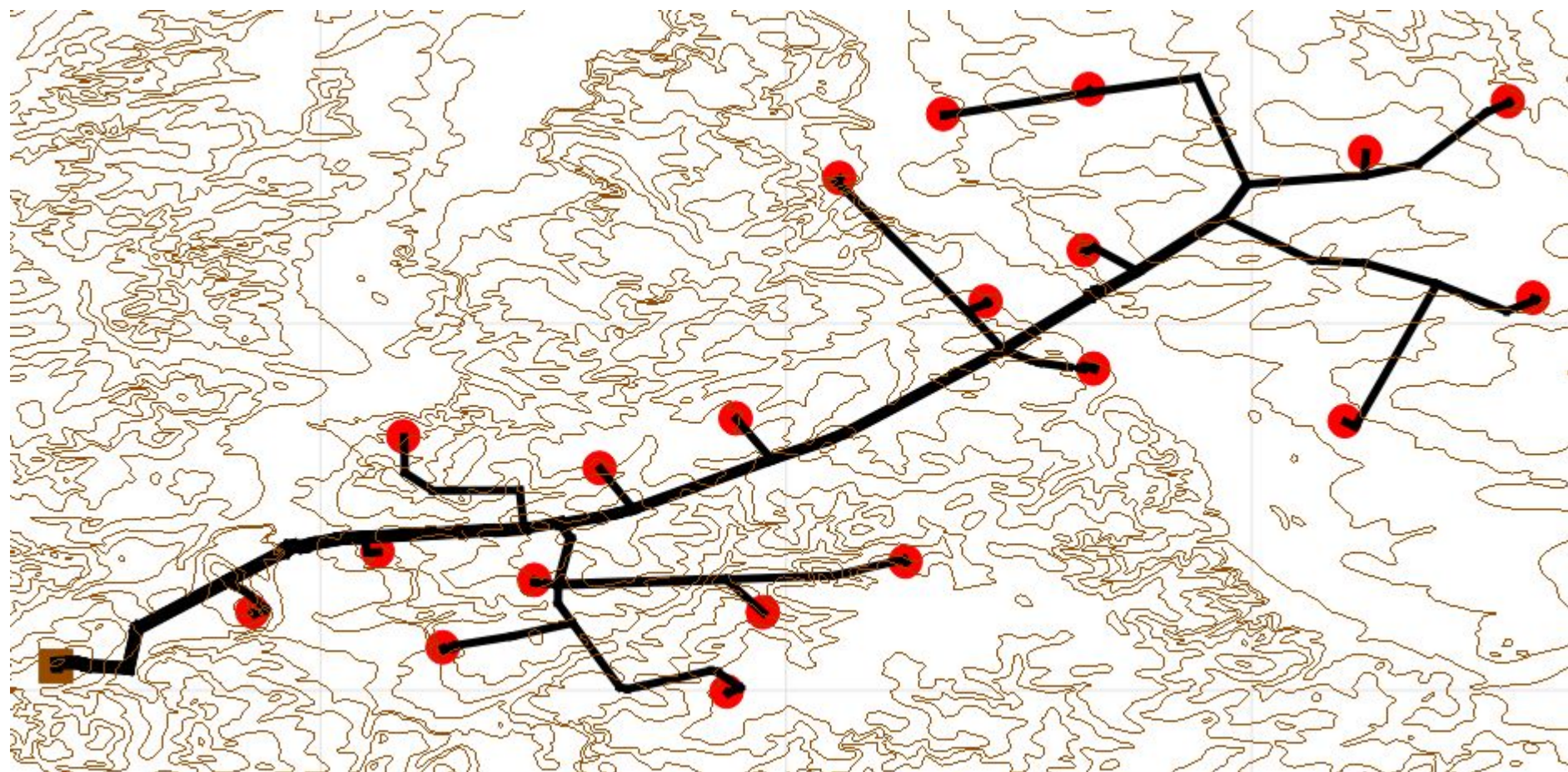
Сколько взять?

Как выбрать
параметры?

Какой способ выбрать?







1. Whitley D. A genetic algorithm tutorial //Statistics and computing. – 1994. – Т. 4. – №. 2. – С. 65-85.
2. Kennedy J. Particle swarm optimization //Encyclopedia of machine learning. – Springer US, 2011. – С. 760-766.
3. Dorigo M., Blum C. Ant colony optimization theory: A survey //Theoretical computer science. – 2005. – Т. 344. – №. 2-3. – С. 243-278.

Bonus: [youtube.com/watch?v=GOFws_hhZs8](https://www.youtube.com/watch?v=GOFws_hhZs8)

Очень много реализаций стохастических методов с примерами: <http://yarpiz.com>

Каждому из вас дается реализация алгоритма PSO в простом виде на языке MATLAB. Кому не нравится MATLAB – покажу, где скачать его на Python. Кому не нравится Python – ищите сами.

Алгоритм можно модифицировать **как угодно**, есть 2 ограничения:

- алгоритм **не знает ничего** о целевой функции
- число вызовов строго фиксировано – **50000**

Каждый должен будет придумать 3 максимально заковыристые тестовые функции (по аналогии с продемонстрированными сегодня), на которых ваша версия алгоритма будет находить оптимум, а версии соперников – нет. **Размерность** задачи равна **10**, пространство поиска $\Omega = [-5; 5] \times [-5; 5] \times \dots \times [-5; 5]$.

Во **вторник 27 февраля** будет проведен баттл по олимпийской системе. Победитель получает респект и 10 за домашнее задание.

