Методы стохастической оптимизации

Бочкарева Мария mibochkareva@edu.hse.ru

Кириллов Дмитрий dakirillov@edu.hse.ru

Содержание

1) Стохастическая оптимизация
2) Недостатки и преимущества
3) Обзор методов стохастической оптимизации
    a) Стохастические модификации градиентного спуска
    b) Метод Монте-Карло
    c) Имитация отжига
    d) Dynamically dimensioned search
    e) Генетический алгоритм
    f) Covariance Matrix Adaptation Evolution Strategy
4) Примеры задач

Задача стохастической оптимизации

Задача оптимизации является **стохастической**, если параметры, описывающие исследуемый объект (процесс, конструкцию и т.д.), носят вероятностный (случайный) характер.

Задачи стохастической оптимизации

Недостатки

- неспецифичность
- сложно доказать сходимость алгоритма

Преимущества

- скорость
- универсальность
- (обычно) малые требования к модели
- вычислительная легкость

Методы стохастической оптимизации

- Stochastic Gradient Descent
- Semi-Stochastic Gradient Descent
- Stochastic Average Gradient
- Метод Монте-Карло
- Имитация отжига
- Dynamically dimensioned search
- Генетический алгоритм
- Covariance Matrix Adaptation Evolution Strategy

Stochastic Gradient Descent (SGD)

$$f(x) = \sum_{n=1}^{N} f_n(x) \to \min$$

На каждом шаге случайно равномерно выбирается n

$$x_k = x_{k-1} - \mu \bigtriangledown_x f_n(x_{k-1})$$

Если функция выпуклая, градиент - Липшецевая функция и $\mu = \frac{const}{k}$, то алгоритм сходится к минимуму

Semi-Stochastic Gradient Descent (S2GD)

На каждом шаге выбирается $t \in \{1, 2, \ldots, T\}$ с вероятностью $\frac{1}{(1-\alpha\lambda)^k}$

$y = {}_1 x_{k-1}$

$g = \frac{1}{N} \nabla_x f(x)$

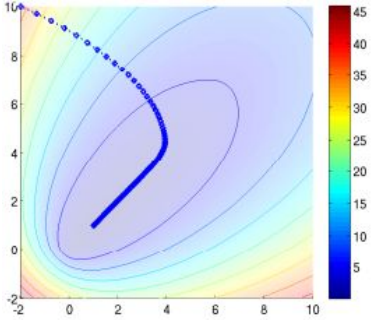
for $i \in \{1, 2, \ldots, t\}$:

Случайно равномерно выбрать n

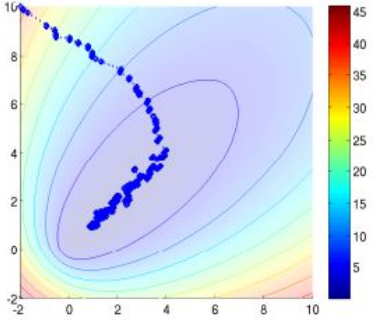
$$y = y - \mu\left(g - \nabla_x f_n(y) + \nabla_x f_n(x_{k-1})\right)$$

$x_k = y$

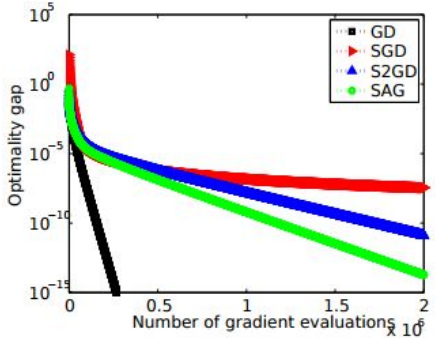
Сравнение методов градиентного спуска



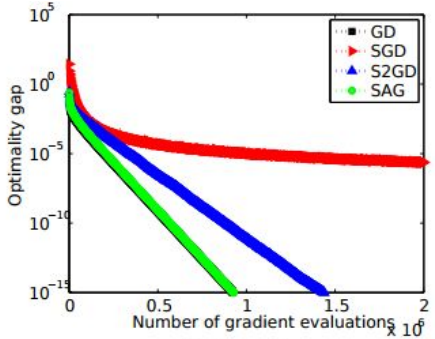
(a) Gradient descent



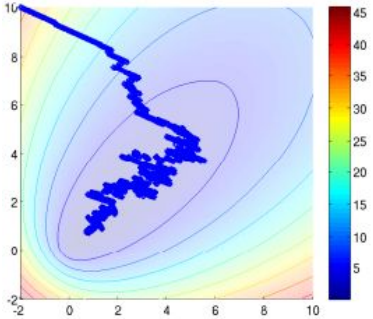
(b) Stochastic gradient descent



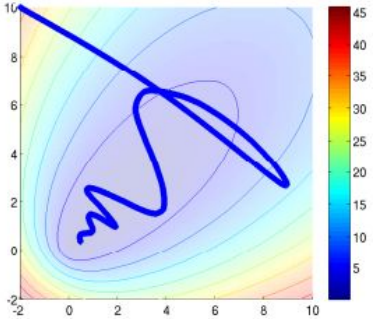
(a) $N = 10^2$



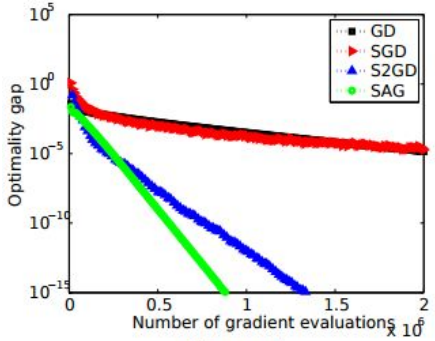
(b) $N = 10^3$



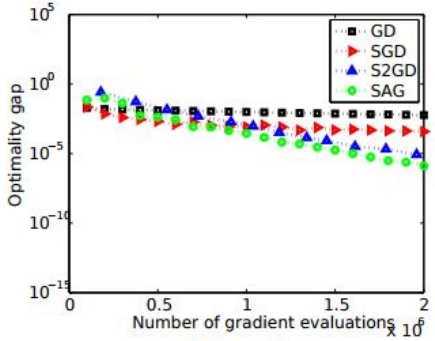
(c) Semi-stochastic gradient descent



(d) Stochastic average gradient



(c) $N = 10^4$



(d) $N = 10^5$

Stochastic Average Gradient (SAG)

Работает как случайный градиентный спуск

Но запоминает все ранее подсчитанные градиенты и на каждом шаге обновляет только одно слагаемое

На каждом шаге случайно равномерно выбирается n

$$x_k = x_{k-1} - \frac{\mu}{N}\left(\sum_{m \neq n} \nabla f_m(x_{k'}) + \nabla f_n(x_{k-1})\right) \forall k' < k$$

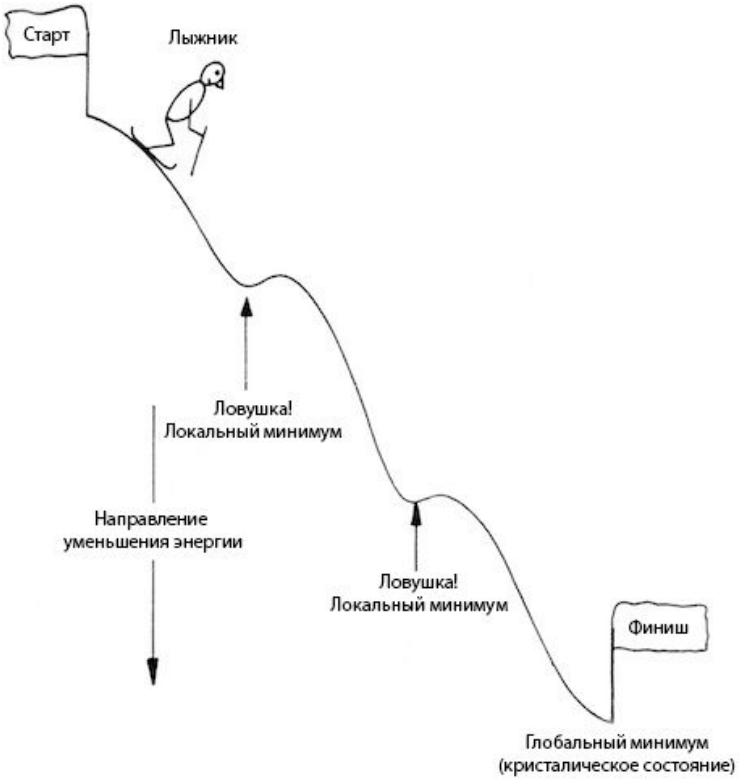
Метод Монте-Карло

На каждом шаге случайно выбираем точку **x**.

Если решение лучше всех ранее найденных, запоминаем его

Продолжаем предыдущие шаги либо пока не найдем достаточно хорошее решение, либо пока не сделаем слишком много итераций

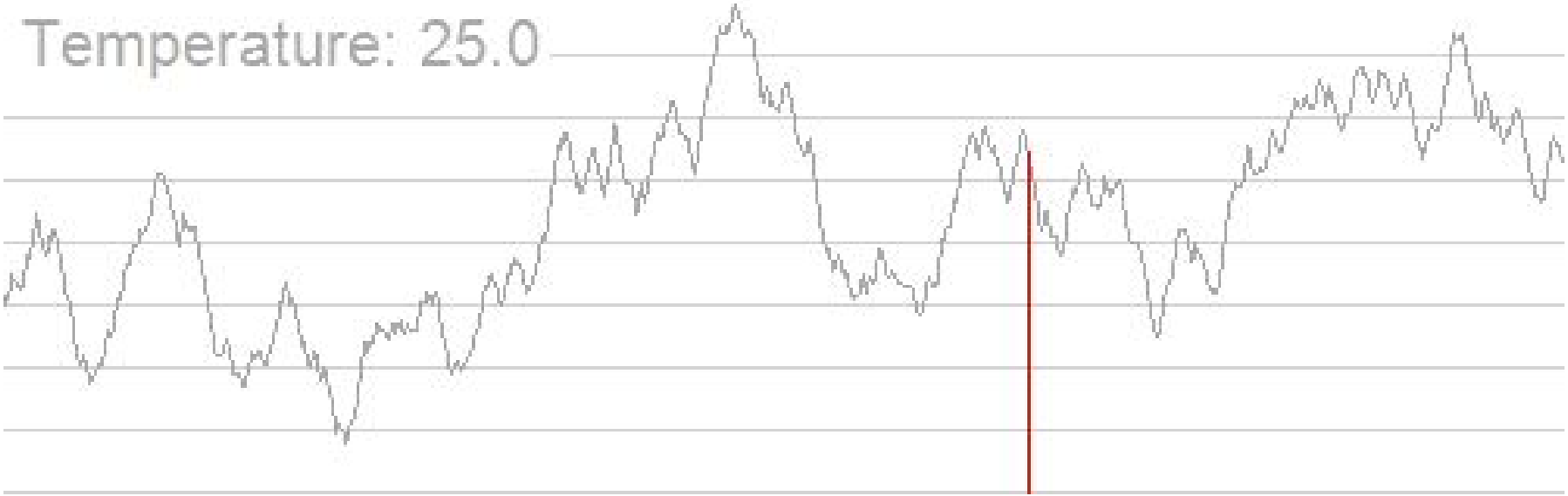
Алгоритм Имитации Отжига



Алгоритм Имитации Отжига

1. Случайным образом генерируем новое состояние; его распределение вероятности зависит от текущего состояния и текущей температуры
2. Принимаем или не принимаем сгенерированное состояние в качестве текущего; Вероятность принять решение зависит от разности функции в новом и текущего состояний и от температуры (чем выше температура, тем больше вероятность принять состояние хуже текущего)
3. если новое состояние не принято, генерируем другое и повторяем действия с третьего пункта, если принято — переходим к следующей итерации, понизив температуру.

Алгоритм Имитации Отжига



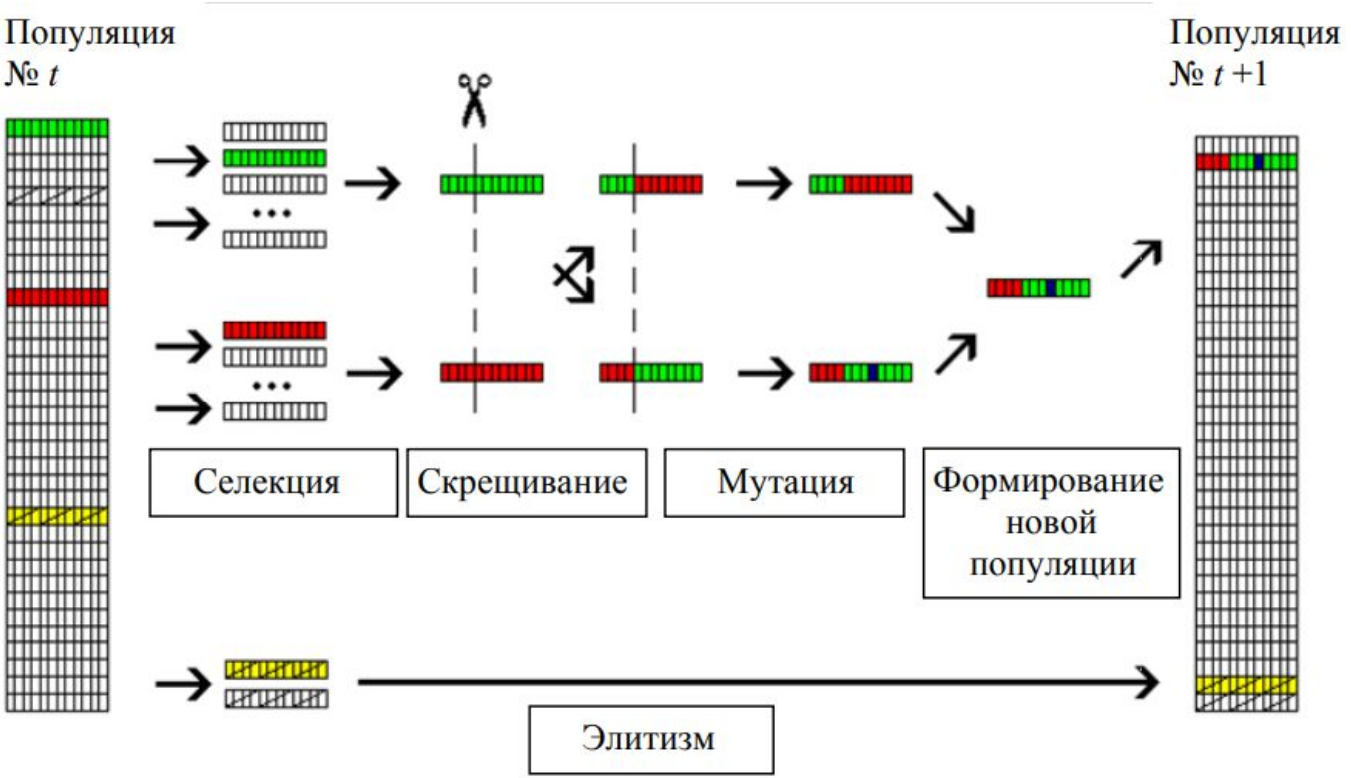
Dynamically dimensional search

Алгоритм вычисляет значение целевой функции ровно **m** раз

На **i**-ом шаге для каждого измерения выбирается, будет ли оно меняться. (Вероятность выбора для каждого измерения $P(i) = 1 - \frac{\ln i}{\ln m}$)

Координаты по выбранным измерениям варьируются, используя нормальное распределение

Генетический алгоритм

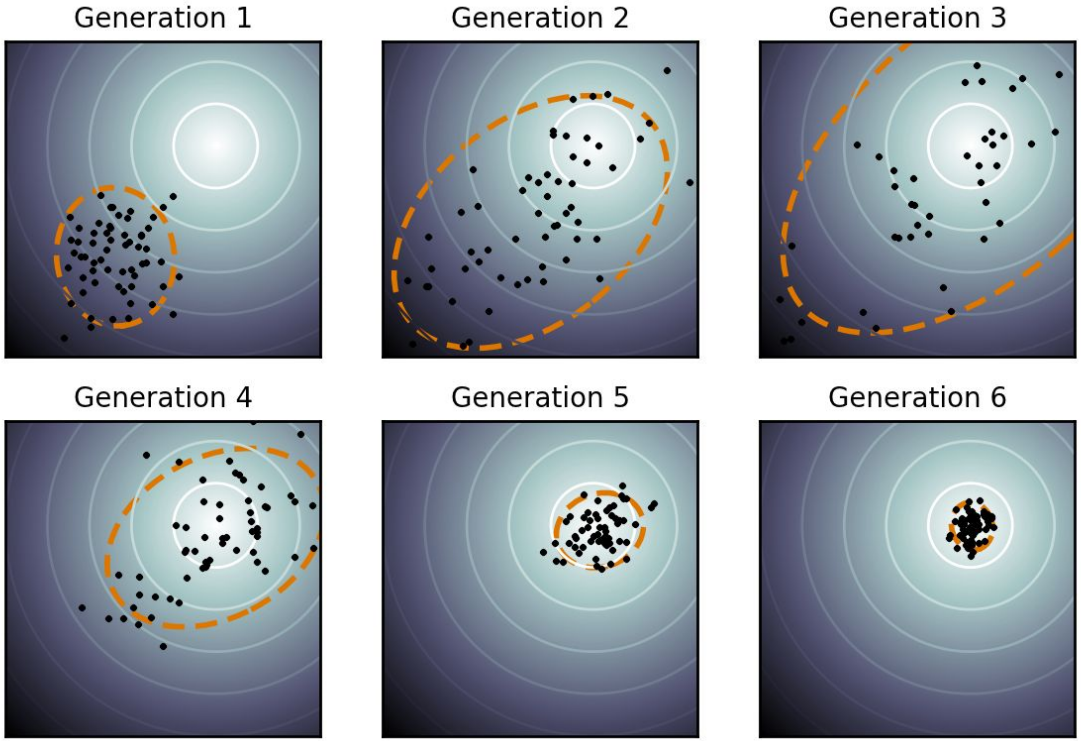


Covariance Matrix Adaptation Evolution Strategy

Метод непрерывной оптимизации второго порядка, подходящий для невыпуклых и нелинейных функций

Не требует дифференцируемости функции, использует метод максимального правдоподобия для улучшения значений

Covariance Matrix Adaptation Evolution Strategy



Covariance Matrix Adaptation Evolution Strategy

Поддерживает среднее значение **m**, матрицу ковариации **C** для распределения точек и длину шага**σ**

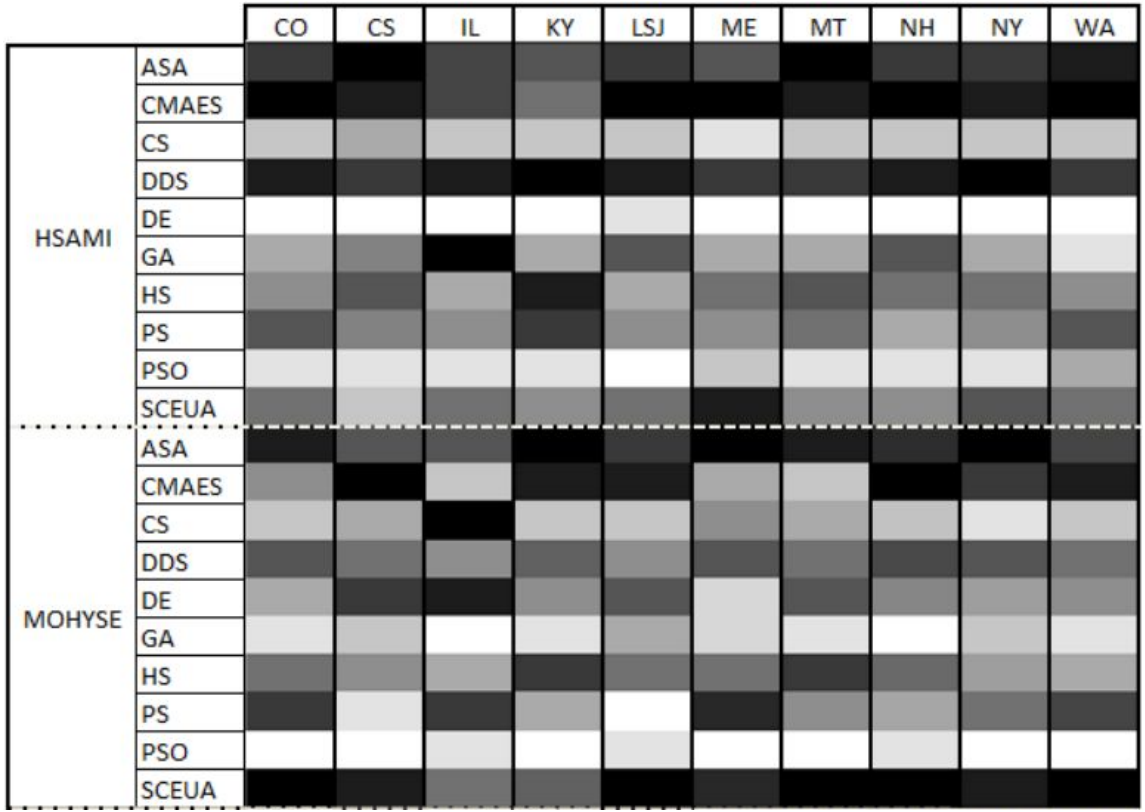
Выбираются точки нового поколения: $\boldsymbol{x}_i = \boldsymbol{m} + \sigma \boldsymbol{y}_i, \quad \boldsymbol{y}_i \sim \mathcal{N}_i(\mathbf{0}, \mathbf{C}), \quad \text{for } i = 1, \ldots, \lambda$

Точки сортируются по возрастанию значений целевой функции на них.

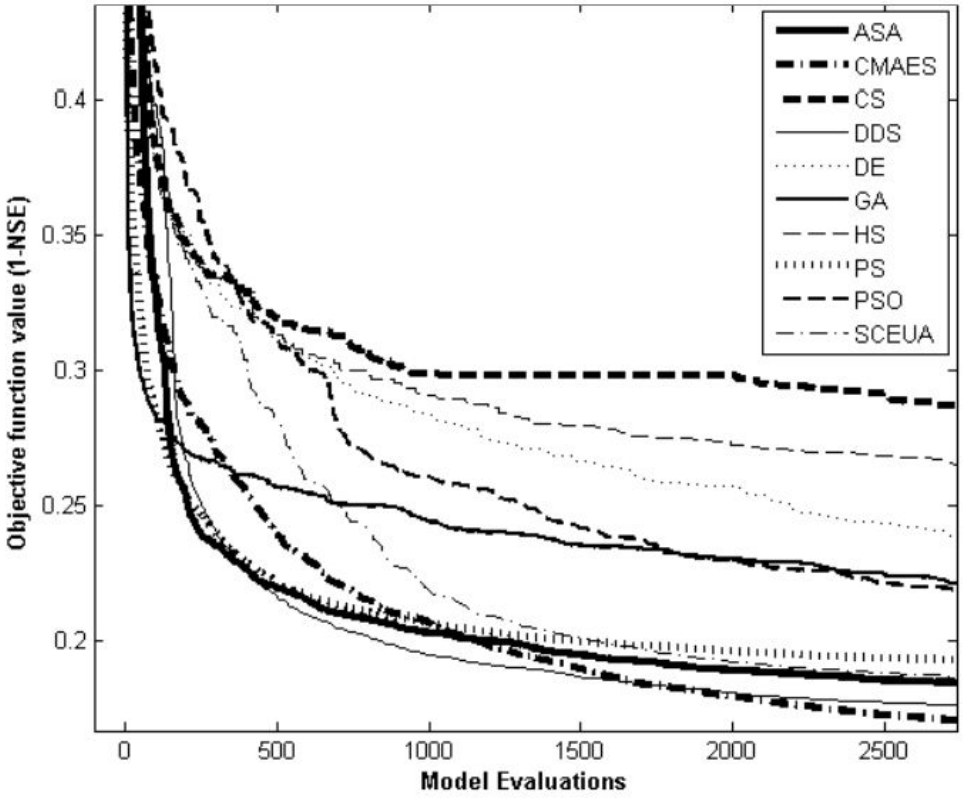
Обновляется среднее: $\boldsymbol{m} \leftarrow \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}$

Обновляется матрица ковариации **C** и **σ**

Сравнение алгоритмов



Сравнение алгоритмов



Примеры задач: случайная строка

Интересный эксперимент: возьмём любое предложение, например Шекспировскую строку: *Methinks it is like a weasel* и случайную строку такой же длины: *wdltmnlt dtjbkwirzrezlmqco p* и начнем вносить в неё случайные изменения. Через сколько поколений эта случайная строка превратится в Шекспировскую строку, если выживать будут лишь потомки более похожие на Шекспировскую?

Решение: случайная строка

Входные данные: строка s

Выходные данные: натуральное число N, равное количеству поколений необходимых для преобразования случайной строки длины len(s) в строку s

Что в нашем случае **мутация**?

Под мутацией строки *S* мы понимаем замену одного случайно выбранного символа из строки *S* на другой произвольный символ алфавита. В данной задаче мы используем только символы нижнего регистра латиницы и пробел.

Что такое **изначальная популяция**?

Это случайная строка равная по длине входной строке *S*.

Что такое **потомки** (offsprings)?

Пусть мы зафиксировали количество мутаций одной строки на константу *k*, тогда потомки — это *k* мутаций каждой строки текущего поколения.

Что такое **выжившие** (survivors)?

Пусть мы зафиксировали размер популяции на константу h, тогда выжившие — это h строк максимально похожих на входную строку S.

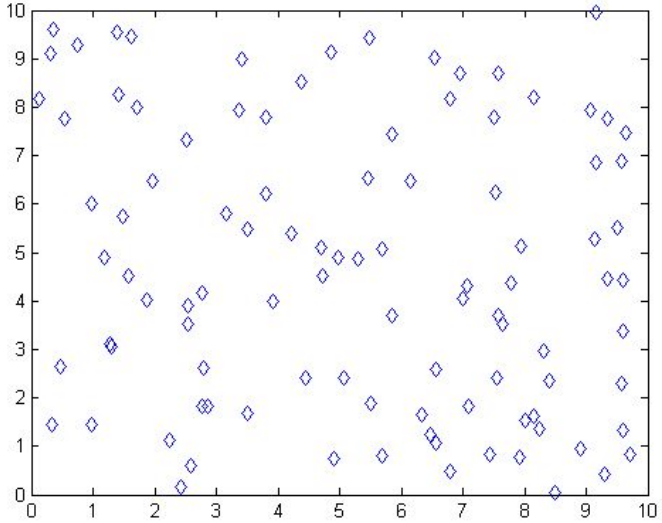
Примеры задач: задача коммивояжера

Представьте себе, что Вы странствующий торговец и хотите предложить свой товар жителям каждого города в стране. Путешествия отнимают много сил и времени, поэтому логично, что Вы хотите составить свой маршрут таким образом, чтобы расстояние, которое предстоит преодолеть, было минимальным. Этот маршрут и предстоит отыскать.

Более формально: необходимо найти кратчайший путь, проходящий через каждый город и заканчивающийся в точке отправления.

Решение: задача коммивояжера

Для начала давайте определимся с данными. Пусть наши города случайным образом разбросаны в квадрате 10x10. Каждый город, соответственно, представлен парой координат. Всего 100 городов.



Литература

Лопатин А. С. Метод отжига //Стохастическая оптимизация в информатике. – 2005. – Т. 1. – №. 1. – С. 133.

Matrenin, Pavel & Sekaev, Vg & Grif, M.G.. (2016). Методы стохастической оптимизации.

Arsenault R. et al. Comparison of stochastic optimization algorithms in hydrological model calibration //Journal of Hydrologic Engineering. – 2013. – Т. 19. – №. 7. – С. 1374-1384.

Papamakarios G. Comparison of Modern Stochastic Optimization Algorithms. – 2014.

Stochastic Gradient Descent

Algorithm 2.2: Stochastic gradient descent

Input: initial step size $\alpha_0 > 0$, decay parameter $\lambda > 0$, $\mathbf{w}_0$

for $k \in \{0, 1, \ldots\}$ **do**
  $\alpha_k = \alpha_0 (1 + \alpha_0 \lambda k)^{-1}$
  choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
  $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f_n (\mathbf{w}_k)$
end for

Output: $\mathbf{w}_k$

Algorithm 2.4: Stochastic average gradient

Input: step size $\alpha > 0$, $\mathbf{w}_0$

initialize $\mathbf{g}_n = \mathbf{0}$ for all $n \in \{1, 2, \ldots, N\}$
for $k \in \{0, 1, \ldots\}$ **do**
  choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \frac{\alpha}{N} \left[\sum_{n' \neq n} \mathbf{g}_{n'} + \nabla f_n (\mathbf{w}_k) \right]$
  $\mathbf{g}_n \leftarrow \nabla f_n (\mathbf{w}_k)$
end for

Output: $\mathbf{w}_k$

Algorithm 2.3: Semi-stochastic gradient descent

Input: step size $\alpha > 0$, maximum # of inner iterations $T > 0$, $\lambda \geq 0$, $\mathbf{w}_0$

for $k \in \{0, 1, \ldots\}$ **do**
  $\mathbf{g} \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla f_n (\mathbf{w}_k)$
  $\mathbf{y} \leftarrow \mathbf{w}_k$
  choose $t \in \{1, 2, \ldots, T\}$ with probability $\propto (1 - \lambda\alpha)^{-t}$
  for $i \in \{1, 2, \ldots, t\}$ **do**
    choose $n \in \{1, 2, \ldots, N\}$ uniformly at random
    $\mathbf{y} \leftarrow \mathbf{y} - \alpha \left[\mathbf{g} - \nabla f_n (\mathbf{y}) + \nabla f_n (\mathbf{w}_k) \right]$
  end for
  $\mathbf{w}_{k+1} \leftarrow \mathbf{y}$
end for

Output: $\mathbf{w}_k$

Сверхбыстрый отжиг

$$\begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix}, \text{ где } x_i \in [A_i, B_i].$$

$$g_T(y) = \prod_{i=1}^{D} \frac{1}{2(|y_i| + T_i)\ln(1 + 1/T_i)} \equiv \prod_{i=1}^{D} g_{(i;T)}(y_i), \quad y_i \in [-1, 1].$$

$$x_i' = x_i + z_i(B_i - A_i)$$

$$z_i = \operatorname{sgn}(\alpha_i - \frac{1}{2}) T_i((1 + 1/T_i)^{|2\alpha_i - 1|} - 1), \alpha_i - i.i.c.\ \alpha_i \sim U[0; 1]$$

$$T_i(k) = T_{(i;0)} \exp(-c_i k^{1/D}), \quad c_i > 0 \quad c_i = m_i \exp(-n_i/D).$$

DDS pseudocode

STEP 1. Define DDS inputs:

- neighborhood perturbation size parameter, r (0.2 is default)
- maximum # of function evaluations, m
- vectors of lower, $\boldsymbol{x}^{min}$, and upper, $\boldsymbol{x}^{max}$, bounds for all D decision variables
- initial solution, $\boldsymbol{x}^0 = [x_1, \ldots, x_D]$

STEP 2. Set counter to 1, i = 1, and evaluate objective function F at initial solution, $F(\boldsymbol{x}^0)$:

- $F_{best} = F(\boldsymbol{x}^0)$, and $\boldsymbol{x}^{best} = \boldsymbol{x}^0$

STEP 3. Randomly select J of the D decision variables for inclusion in neighborhood, $\{N\}$:

- calculate probability each decision variable is included in $\{N\}$ as a function of the current iteration count: $P(i) = 1 - \ln(i)/\ln(m)$
- FOR d = 1, … D decision variables, add d to $\{N\}$ with probability P
- IF $\{N\}$ empty, select one random d for $\{N\}$

STEP 4. FOR j = 1, …, J decision variables in $\{N\}$, perturb x_j^{best} using a standard normal random variable, N(0,1), reflecting at decision variable bounds if necessary:

- $x_j^{new} = x_j^{best} + \sigma_j N(0,1)$, where $\sigma_j = r(x_j^{max} - x_j^{min})$
- IF $x_j^{new} < x_j^{min}$, reflect perturbation:
  - $x_j^{new} = x_j^{min} + (x_j^{min} - x_j^{new})$
  - IF $x_j^{new} > x_j^{max}$, set $x_j^{new} = x_j^{min}$
- IF $x_j^{new} > x_j^{max}$, reflect perturbation:
  - $x_j^{new} = x_j^{max} - (x_j^{new} - x_j^{max})$
  - IF $x_j^{new} < x_j^{min}$, set $x_j^{new} = x_j^{max}$

STEP 5. Evaluate $F(\boldsymbol{x}^{new})$ and update current best solution if necessary:

- IF $F(\boldsymbol{x}^{new}) \leq F_{best}$, update new best solution:
  - $F_{best} = F(\boldsymbol{x}^{new})$ and $\boldsymbol{x}^{best} = \boldsymbol{x}^{new}$

STEP 6. Update iteration count, i = i+1, and check stopping criterion:

- IF i = m, STOP, print output (e.g. F_{best} & $\boldsymbol{x}^{best}$)
- ELSE go to STEP 3

CMAES

Input: $\boldsymbol{m} \in \mathbb{R}^n$, $\sigma \in \mathbb{R}_+$, λ
Initialize: $\mathbf{C} = \mathbf{I}$, and $\boldsymbol{p}_\mathbf{c} = \mathbf{0}$, $\boldsymbol{p}_\sigma = \mathbf{0}$,
Set: $c_\mathbf{c} \approx 4/n$, $c_\sigma \approx 4/n$, $c_1 \approx 2/n^2$, $c_\mu \approx \mu_w/n^2$, $c_1 + c_\mu \leq 1$, $d_\sigma \approx 1 + \sqrt{\frac{\mu_w}{n}}$,
and $w_{i=1\ldots\lambda}$ such that $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \approx 0.3\,\lambda$
While not terminate

$$\boldsymbol{x}_i = \boldsymbol{m} + \sigma\,\boldsymbol{y}_i, \quad \boldsymbol{y}_i \sim \mathcal{N}_i(\mathbf{0}, \mathbf{C}), \quad \text{for } i = 1, \ldots, \lambda \qquad \text{sampling}$$

$$\boldsymbol{m} \leftarrow \sum_{i=1}^{\mu} w_i\,\boldsymbol{x}_{i:\lambda} = \boldsymbol{m} + \sigma\boldsymbol{y}_w \quad \text{where } \boldsymbol{y}_w = \sum_{i=1}^{\mu} w_i\,\boldsymbol{y}_{i:\lambda} \qquad \text{update mean}$$

$$\boldsymbol{p}_\mathbf{c} \leftarrow (1 - c_\mathbf{c})\,\boldsymbol{p}_\mathbf{c} + \mathbb{1}_{\{\|\boldsymbol{p}_\sigma\| < 1.5\sqrt{n}\}} \sqrt{1 - (1 - c_\mathbf{c})^2}\,\sqrt{\mu_w}\,\boldsymbol{y}_w \qquad \text{cumulation for } \mathbf{C}$$

$$\boldsymbol{p}_\sigma \leftarrow (1 - c_\sigma)\,\boldsymbol{p}_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\,\sqrt{\mu_w}\,\mathbf{C}^{-\frac{1}{2}}\boldsymbol{y}_w \qquad \text{cumulation for } \sigma$$

$$\mathbf{C} \leftarrow (1 - c_1 - c_\mu)\,\mathbf{C} + c_1\,\boldsymbol{p}_\mathbf{c}\boldsymbol{p}_\mathbf{c}^{\mathrm{T}} + c_\mu \sum_{i=1}^{\mu} w_i\,\boldsymbol{y}_{i:\lambda}\boldsymbol{y}_{i:\lambda}^{\mathrm{T}} \qquad \text{update } \mathbf{C}$$

$$\sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1\right)\right) \qquad \text{update of } \sigma$$