Алгоритм поиска кукушки с полетом Леви

Алгоритм:

Кукушка закладывает по одному яйцу за раз, и сбрасывает его в случайно выбранном гнезде;

Лучшие гнезда с высоким качеством яиц (решения) переносятся на следующие поколения;

Количество доступных хозяйских гнезд фиксировано, и хозяин может обнаружить инородное яйцо с вероятностью Pa ∈ [0, 1]. В этом случае птица-хозяин может либо выбросить яйцо из гнезда, либо покинуть гнездо, чтобы построить совершенно новое гнездо в новом месте.

Схему алгоритма можно представить в следующем виде:

Инициализируем популяцию S = (si, i ∈ [1 : |S|]) из |S| хозяйских гнёзд и кукушку, т.е. определяем начальные значения компонентов векторов Xi; i ∈ [1 : |S|], и вектор начального положения кукушки Xc;

Находим новое Xc, с помощью полётов Леви.

Случайным образом находим гнездо si: i ∈ [1 : |S|], и, если f(Xc) > f(Xi), заменяем яйцо в этом гнезде на яйцо кукушки, т.е. полагаем Xi = Xc;

С вероятностью pa удаляем из популяции некоторое число худших случайно выбранных гнезд и строим новые гнезда в местах определённых с помощью полётов Леви.

Если поколение достигло заданного предела, то заканчиваем алгоритм, иначе переходим ко второму шагу.

Полет Леви

Одной из наиболее мощных функций cuckoo search является использование Lévy flights для генерации новых решений-кандидатов (eggs). В соответствии с этим подходом новое решение-кандидат, $\mathbf{e}_i^{k+1}$ $(i \in [1, \ldots, N])$, создается путем возмущения текущего $\mathbf{e}_i^k$ изменением положения $\mathbf{c}_i$. Для получения $\mathbf{c}_i$ случайного шага $\mathbf{s}_i$ генерируется симметричным распределением Леви. Для получения $\mathbf{s}_i$ алгоритм Мантеньи [47] используется следующим образом:

$$\mathbf{s}_i = \frac{\mathbf{u}}{|\mathbf{v}|^{1/\beta}}, \tag{1}$$

где $\mathbf{u}$ $(\{u_1, \ldots, u_n\})$ и $\mathbf{v}$ $(\{v_1, \ldots, v_n\})$ являются n одномерными векторами и $\beta = 3/2$. Каждый элемент $\mathbf{u}$ и $\mathbf{v}$ вычисляется с учетом следующих нормальных распределений:

$$\begin{gathered} u \sim N\left(0, \sigma_u^2\right), \qquad v \sim N\left(0, \sigma_v^2\right), \\ \sigma_u = \left(\frac{\Gamma\left(1+\beta\right) \cdot \sin\left(\pi \cdot \beta/2\right)}{\Gamma\left(\left(1+\beta\right)/2\right) \cdot \beta \cdot 2^{(\beta-1)/2}} \right)^{1/\beta}, \qquad \sigma_v = 1, \end{gathered} \tag{2}$$

где $\Gamma(\cdot)$ представляет гамма-распределение. После $\mathbf{s}_i$ вычисления требуемое изменение позиции $\mathbf{c}_i$ вычисляется следующим образом:

$$\mathbf{c}_i = 0.01 \cdot \mathbf{s}_i \oplus \left(\mathbf{e}_i^k - \mathbf{e}^{\text{best}}\right), \tag{3}$$

где произведение $\oplus$ обозначает умножения по входам, тогда как $\mathbf{e}^{\text{best}}$ это лучшее решение (egg), виденное на данный момент с точки зрения его пригодности. Значение. Наконец, новое решение-кандидат, $\mathbf{e}_i^{k+1}$, вычисляется с помощью

$$\mathbf{e}_i^{k+1} = \mathbf{e}_i^k + \mathbf{c}_i. \tag{4}$$

Функции оптимизации

Тестирования функции от двух переменных была использована функция Бута:

$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$	$f(1,3) = 0$

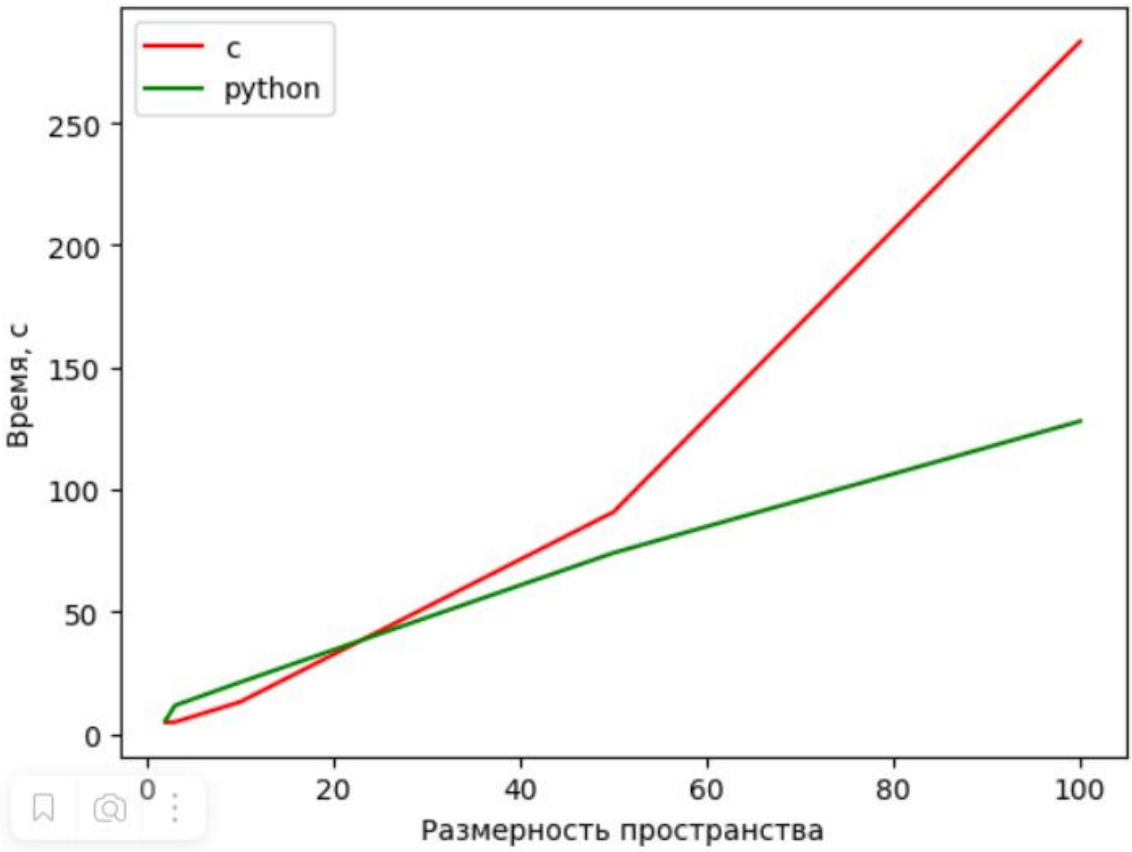
Для тестирования поиска минимума в пространстве выше 3х была применена функция Розенброка:

$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right]$	$\text{Min} = \begin{cases} n = 2 & \rightarrow \quad f(1,1) = 0, \\ n = 3 & \rightarrow \quad f(1,1,1) = 0, \\ n > 3 & \rightarrow \quad f(\underbrace{1, \ldots, 1}_{n \text{ times}}) = 0 \end{cases}$

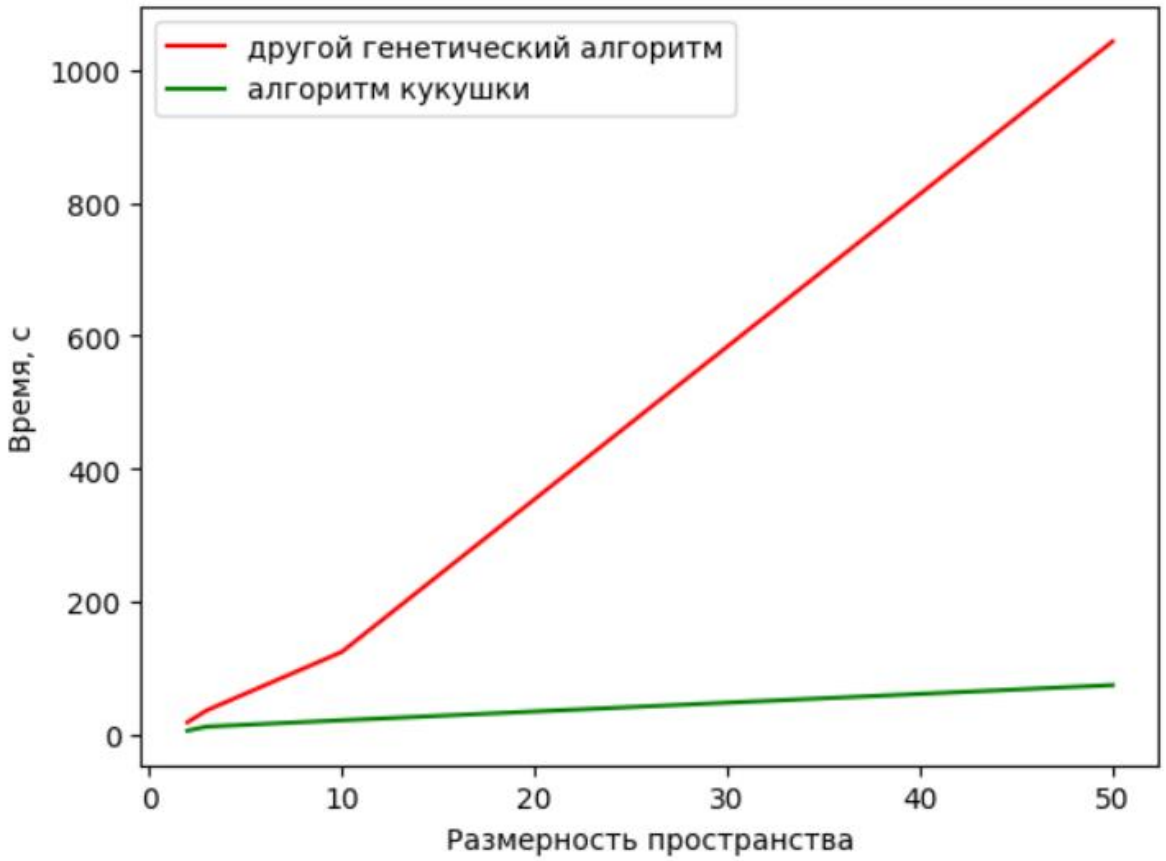
Результаты

Был реализован алгоритм поиска кукушки и протестирован на размерности 2, 3, 10, 50, 100.

Также было проведено сравнение времени выполнения алгоритма на языке Python и C.



И сравнение времени выполнения собственно реализованного алгоритма и генетического алгоритма из pypi.


другой генетический алгоритм
алгоритм кукушки
1000
800
600
400
200
0
Время, с
0
10
20
30
40
50
Размерность пространства