Изображение выглядит как эмблема, символ, герб, нашивка

Автоматически созданное описание

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №6**

**по дисциплине**

**«Системный анализ данных в системах поддержки принятия**

**решений»**

**Алгоритм роя светлячков**

Студент группы:ИКБО-04-22 \_\_Кликушин В.И.\_ *(Ф. И.О. студента)*

Преподаватель \_\_Железняк Л.М.\_\_

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc183883936)

[1 АЛГОРИТМ РОЯ СВЕТЛЯЧКОВ 4](#_Toc183883937)

[1.1 Описание алгоритма 4](#_Toc183883938)

[1.2 Постановка задачи 5](#_Toc183883939)

[1.3 Математическая модель 5](#_Toc183883940)

[1.4 Ручной расчёт 8](#_Toc183883941)

[1.5 Программная реализация 10](#_Toc183883942)

[ЗАКЛЮЧЕНИЕ 13](#_Toc183883943)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 14](#_Toc183883944)

[ПРИЛОЖЕНИЯ 15](#_Toc183883945)

# ВВЕДЕНИЕ

Алгоритм роя светлячков был предложен в 2007 году инженером и исследователем Синь-Шэ Янгом. Этот алгоритм относится к классу метаэвристических алгоритмов роевого интеллекта, ориентированных на оптимизацию функции.

В основе алгоритма лежит наблюдаемое в природе поведение рассматриваемых насекомых. Они излучают свет, который является механизмом коммуникации между особями: с его помощью они привлекают особей противоположного пола, сообщают о приближении хищников, привлекают добычу и так далее. Менее яркие светлячки перемещаются к более ярким, яркость одного светлячка, воспринимаемая другим, уменьшается при его удалении. Если светлячок не видит более яркого представителя роя, он перемещается хаотично.

Алгоритм роя светлячков нашел широкое применение в задачах, где требуется минимизация или максимизация сложных функций. Его успешно используют в инженерии, экономике, науке о данных и других областях, например, оптимизация маршрутов и логистики, настройка параметров нейронных сетей и машинного обучения, обработка изображений и восстановление сигналов, управление ресурсами и планирование задач.

Актуальность алгоритма обусловлена его способностью эффективно находить решения в сложных многомерных пространствах, где традиционные методы могут быть неприменимы из-за высокой вычислительной сложности или требований к непрерывности функции. Простота реализации и адаптация под широкий спектр задач делают алгоритм роя светлячков важным инструментом для оптимизации в современной науке и технике.

# 1 алгоритм РОЯ СВЕТЛЯЧКОВ

## 1.1 Описание алгоритма

Алгоритм роя светлячков использует искусственных агентов (светлячков), которые взаимодействуют друг с другом, перемещаясь в поисковом пространстве в направлении более качественных решений. Каждый светлячок характеризуется своей позицией, которая соответствует возможному решению задачи, и уровнем «свечения» (яркости), зависящим от значения целевой функции в данной позиции.

Оптимизация начинается с инициализации популяции светлячков, которые случайным образом размещаются в пространстве решений. Затем алгоритм выполняет итерации, каждая из которых включает следующие этапы:

1. Обновление яркости. Уровень свечения каждого светлячка пересчитывается в зависимости от значения целевой функции в его текущей позиции. Это позволяет определить, насколько «привлекателен» светлячок для других (Формула 1.3.2).
2. Формирование окрестности. Для каждого светлячка определяется список соседей — агентов, находящихся в пределах заданного радиуса, которые обладают более высокой яркостью (Формула 1.3.3).
3. Выбор соседа и перемещение. Светлячок перемещается в направлении соседа, вероятность выбора которого пропорциональна его яркости. Если в окрестности нет более ярких соседей, светлячок перемещается случайным образом (Формулы 1.3.4–1.3.6).
4. Модификация радиуса окрестности. Радиус видимости светлячка корректируется в зависимости от количества соседей, чтобы адаптироваться к текущей плотности агентов в пространстве (Формула 1.3.7).
5. Обновление глобального решения. Если положение какого-либо светлячка превосходит текущее лучшее значение, обновляется глобальное решение.

Процесс продолжается до тех пор, пока не выполнено условие остановки: либо достигнуто максимальное количество итераций, либо отсутствуют значительные улучшения в течение заданного числа шагов.

## 1.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом роя светлячков для нахождения приближённого глобального минимума функции.

Задачи: изучить алгоритм роя светлячков, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт итерации алгоритма, разработать программную реализацию алгоритма роя светлячков для задачи минимизации функции.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 1.2.1).

. (1.2.1)

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области .

## 1.3 Математическая модель

Алгоритм имеет следующие входные параметры: — коэффициент изменения радиуса окрестности; — коэффициент уменьшения уровня люциферина; — коэффициент изменения позиции; — начальный радиус окрестности; — максимальное количество итераций алгоритма; — размер популяции светлячков; — коэффициент привлекательности светлячков; — минимальные и максимальные границы пространства; — длина вектора позиции агента. Все параметры определены от нуля до единицы.

Позиция каждого светлячка инициализируется случайным образом (Формула 1.3.1).

(1.3.1)

где k – номер агента.

Изначально все светлячки имеют одинаковое количество люциферина *.* Радиус окрестности также инициализируется предварительно заданным значением *.*

Обновление уровня люциферина зависит от позиции агента в пространстве (значения его целевой функции). Вычисление уровня люциферина осуществляется по Формуле 1.3.2.

(1.3.2)

где *l* – количество люциферина;

*k* – номер агента;

*t* – номер итерации.

Каждый агент выбирает того агента внутри радиуса окрестности поиска, у которого уровень люциферина выше, чем его собственный. Окрестность светлячка определяется в соответствие с Формулой 1.3.3.

(1.3.3)

где – окрестность светлячка;

*m* – светлячок в окрестности светлячка *k*.

Таким образом, окрестность включает светлячков , которые находятся в пределах радиуса ( ), имеют уровень люциферина выше, чем у светлячка ().

Вычисление вероятности перемещения к соседям осуществляется по Формуле 1.3.4.

(1.3.4)

где – вычисленная вероятность движения светлячка к соседу .

Говоря иначе, показывает относительную привлекательность соседа по сравнению с остальными соседями.

Светлячок выбирает номер соседа в своей окрестности, используя метод колеса рулетки (Формула 1.3.5).

(1.3.5)

где – случайное число в интервале [0;1].

Обновленная позиция агента определяется по Формуле 1.3.6.

(1.3.6)

где – новая позиция агента;

В числителе второго слагаемого записан вектор смещения, направленный от текущей позиции светлячка к соседу . В знаменателе дроби записана длина этого вектора (евклидово расстояние). Это нормализующий фактор, чтобы перемещение происходило по направлению, но не зависело от расстояния.

Обновление радиуса окрестности осуществляется по Формуле 1.3.7.

(1.3.7)

где – обновленный радиус окрестности для светлячка ;

– желаемое количество соседей для светлячка ;

– текущее количество соседей светлячка  в его окрестности.

В конце итерации определяется наименьшее значение функции, обновляется глобальный минимум, если найдено значение, меньшее текущего глобального минимума.

## 1.4 Ручной расчёт

Коэффициент изменения радиуса окрестности ; коэффициент уменьшения уровня люциферина ; коэффициент изменения позиции ; начальный радиус окрестности ; размер популяции светлячков ; коэффициент привлекательности светлячков .

Светлячки случайным образом размещены в гиперпространстве поиска. Начальное количество люциферина для светлячков равно 0, начальный радиус равен 0,5. Ниже представлены начальные координаты светлячков:

В начале итерации обновляется количество люциферина у каждого светлячка по Формуле 1.3.2. Расчет количества люциферина представлен ниже:

Вычисляется множество соседей для каждого светлячка по Формуле 1.3.3. Расчет представлен ниже:

На первой итерации ни у одного из светлячков нет соседей.

Изменяется радиус окрестности светлячков в соответствие с Формулой 1.3.7. Расчет радиуса представлен ниже:

Лучшее глобальное значение функции на первой итерации равно 695,491. Точка, в которой достигается минимальное значение: ().

## 1.5 Программная реализация

Разработаны классы Firefly и FireflySwarm, которые реализуют отдельного светлячка и рой светлячков.

Код алгоритма роя светлячков для задачи поиска глобального минимума функции представлен в Приложении А.

Коэффициент изменения радиуса окрестности ; коэффициент уменьшения уровня люциферина ; коэффициент изменения позиции ; начальный радиус окрестности ; размер популяции светлячков ; коэффициент привлекательности светлячков ; максимальное количество итераций .

Результат работы алгоритма роя светлячков представлен на Рисунке 1.5.1.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Рисунок 1.5.1 – Результат работы алгоритма роя светлячков для задачи поиска глобального минимума функции

Для каждой итерации выводится её номер и глобальный текущий минимум функции.

Визуализация процесса поиска представлена на Рисунках 1.5.2–1.5.3.

В верхней части графика выводится номер итерации, зелёные точки соответствуют светлячкам в гиперпространстве поиска решений. Светлячки с более высоким уровнем люциферина святятся ярче. На вертикальной оси - значение координаты по оси y, и на горизонтальной оси – значение координаты по оси x.

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Автоматически созданное описание

Рисунок 1.5.2 – Визуализация работы алгоритма роя светлячков на начальных итерациях

На Рисунке 1.5.2 видно, что светлячки распределены по всему гиперпространству поиска на начальных итерациях работы алгоритма.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.5.3 – Сходимость светлячков к глобальному минимуму

На Рисунке 1.5.3 светлячки сходятся к одной точке – глобальному минимуму рассматриваемой функции оптимизации.

Объединение светлячков в гиперпространстве поиска в один кучный рой говорит о сходимости алгоритма.

График сходимости алгоритма роя светлячков представлен на Рисунке 1.5.4.

Изображение выглядит как текст, снимок экрана, дисплей, число

Автоматически созданное описание

Рисунок 1.5.4 – График сходимости алгоритма роя светлячков

На горизонтальной оси отложен номер итерации, на вертикальной оси - глобальный текущий минимум функции.

По графику видно, что алгоритм находит приближённый глобальный минимум функции Гольдшейна-Прайса примерно спустя десять итераций работы.

# Заключение

В ходе выполнения данной работы изучен алгоритм роя светлячков, проведён его ручной расчёт одной итерации, а также разработана программа на языке Python для оптимизации функции Голдштейна-Прайса.

Основное преимущество алгоритма роя светлячков заключается в его способности эффективно находить решения сложных многомерных задач оптимизации. Это достигается благодаря принципу взаимодействия агентов (светлячков), которые движутся в направлении более перспективных областей пространства решений, ориентируясь на значение целевой функции. Такой подход позволяет комбинировать исследование новых областей пространства с углубленным поиском вокруг текущих лучших решений.

Реализованный алгоритм демонстрирует, как параметры, такие как радиус окрестности, коэффициент уменьшения люциферина, количество итераций и другие, влияют на сходимость и качество найденного решения. Визуализация работы алгоритма показала, как светлячки перемещаются в пространстве, группируясь вокруг глобального экстремума, что подтверждает его способность находить оптимальные решения в сложных ландшафтах целевой функции.

Таким образом, алгоритм роя светлячков является универсальным инструментом для решения задач оптимизации. Благодаря своей простоте, адаптивности и способности избегать локальных экстремумов, он нашел широкое применение в инженерии, науке о данных и других областях, требующих эффективного поиска решений в многомерных пространствах.

# Список информационных источников

1. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс] : Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
2. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
3. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
4. Firefly algorithm. [Электронный ресурс]: Википедия. – URL: https://en.wikipedia.org/wiki/Firefly\_algorithm (Дата обращения: 30.11.2024).

# приложения

Приложение А — Код реализации алгоритма роя светлячков.

### Приложение А

Код реализации алгоритма роя светлячков

Листинг А – Реализация алгоритма роя светлячков

import random

import math

from typing import List, Tuple, Callable

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

def goldstein\_price(x: float, y: float) -> float:

    '''

    Функция Голдштейна-Прайса для оптимизации.

    Параметры:

        x (float): Координата по оси x.

        y (float): Координата по оси y.

    Возвращает:

        float: Значение функции Голдштейна-Прайса.

    '''

    term1 = (1 + (x + y + 1) \*\* 2 \* (19 - 14 \* x + 3 \*

             x \*\* 2 - 14 \* y + 6 \* x \* y + 3 \* y \*\* 2))

    term2 = (30 + (2 \* x - 3 \* y) \*\* 2 \* (18 - 32 \* x + 12 \*

             x \*\* 2 + 48 \* y - 36 \* x \* y + 27 \* y \*\* 2))

    return term1 \* term2

class Firefly:

    def \_\_init\_\_(self, position: List[float], luciferin: float = 0.0, radius: float = 1.0):

        '''

        Инициализирует светлячка.

        Параметры:

            position (List[float]): Начальная позиция светлячка.

            luciferin (float): Уровень люциферина.

            radius (float): Радиус окрестности светлячка.

        '''

        self.position = position

        self.luciferin = luciferin

        self.radius = radius

    def update\_luciferin(self, function\_value: float, rho: float, gamma: float):

        '''

        Обновляет уровень люциферина светлячка.

        Параметры:

            function\_value (float): Значение целевой функции в текущей позиции.

            rho (float): Коэффициент уменьшения люциферина.

            gamma (float): Коэффициент привлекательности светлячка.

        '''

        self.luciferin = (1 - rho) \* self.luciferin + \

            gamma \* (1 / function\_value)

    def move\_towards(self, other: 'Firefly', delta: float, bounds: List[Tuple[float, float]]):

        '''

        Перемещает светлячка в направлении другого более яркого светлячка.

        Параметры:

            other (Firefly): Другой светлячок, к которому перемещается текущий.

            delta (float): Коэффициент изменения позиции.

            bounds (List[Tuple[float, float]]): Границы пространства поиска.

        '''

        direction = [other.position[i] - self.position[i]

Продолжение Листинга А

                     for i in range(len(self.position))]

        distance = math.sqrt(sum(d \*\* 2 for d in direction))

        if distance > 0:

            normalized\_direction = [d / distance for d in direction]

            self.position = [

                min(max(

                    self.position[i] + delta \* normalized\_direction[i], bounds[i][0]), bounds[i][1])

                for i in range(len(self.position))

            ]

class FireflySwarm:

    def \_\_init\_\_(self, fitness\_function: Callable[..., float], bounds: List[Tuple[float, float]],

                 num\_fireflies: int, max\_iterations: int, beta: float, rho: float,

                 delta: float, gamma: float, initial\_radius: float):

        '''

        Инициализирует алгоритм роя светлячков.

        Параметры:

            fitness\_function (Callable[..., float]): Целевая функция для оптимизации.

            bounds (List[Tuple[float, float]]): Границы пространства поиска [(xmin, xmax), (ymin, ymax)].

            num\_fireflies (int): Количество светлячков.

            max\_iterations (int): Максимальное количество итераций.

            beta (float): Коэффициент изменения радиуса окрестности.

            rho (float): Коэффициент уменьшения уровня люциферина.

            delta (float): Коэффициент изменения позиции.

            gamma (float): Коэффициент увеличения люциферина.

            initial\_radius (float): Начальный радиус окрестности.

        '''

        self.fitness\_function = fitness\_function

        self.bounds = bounds

        self.num\_fireflies = num\_fireflies

        self.max\_iterations = max\_iterations

        self.beta = beta

        self.rho = rho

        self.delta = delta

        self.gamma = gamma

        self.initial\_radius = initial\_radius

        self.fireflies = []

        self.min\_radius = 0.1

        self.max\_radius = max(

            bounds[0][1] - bounds[0][0], bounds[1][1] - bounds[1][0])

        self.best\_values = []

        self.positions\_history = []

    def initialize\_fireflies(self):

        '''Инициализирует популяцию светлячков в случайных позициях.'''

        self.fireflies = [

            Firefly(

                position=[random.uniform(bounds[0], bounds[1])

                          for bounds in self.bounds],

                radius=self.initial\_radius

            )

            for \_ in range(self.num\_fireflies)

        ]

    def calculate\_neighbors(self, firefly: Firefly) -> List[Firefly]:

        '''

Продолжение Листинга А

        Вычисляет множество соседей светлячка.

        Параметры:

            firefly (Firefly): Текущий светлячок.

        Возвращает:

            List[Firefly]: Список соседей светлячка.

        '''

        return [

            other for other in self.fireflies

            if other is not firefly

            and math.dist(firefly.position, other.position) < firefly.radius

            and firefly.luciferin < other.luciferin

        ]

    def calculate\_probabilities(self, firefly: Firefly, neighbors: List[Firefly]) -> List[float]:

        '''

        Вычисляет вероятность перемещения к соседям.

        Параметры:

            firefly (Firefly): Текущий светлячок.

            neighbors (List[Firefly]): Список соседей.

        Возвращает:

            List[float]: Список вероятностей перемещения к каждому соседу.

        '''

        total\_difference = sum(

            other.luciferin - firefly.luciferin for other in neighbors)

        if total\_difference == 0:

            return [1 / len(neighbors)] \* len(neighbors)

        probabilities = [(other.luciferin - firefly.luciferin) /

                         total\_difference for other in neighbors]

        return probabilities

    def select\_neighbor(self, neighbors: List[Firefly], probabilities: List[float]) -> Firefly:

        '''

        Выбирает соседа на основе вероятностей методом рулетки.

        Параметры:

            neighbors (List[Firefly]): Список соседей.

            probabilities (List[float]): Список вероятностей.

        Возвращает:

            Firefly: Выбранный сосед.

        '''

        cumulative\_probabilities = [

            sum(probabilities[:i + 1]) for i in range(len(probabilities))]

        rand = random.random()

        for i, prob in enumerate(cumulative\_probabilities):

            if rand <= prob:

                return neighbors[i]

    def adjust\_radius(self, firefly: Firefly, desired\_neighbors: int):

        '''

        Корректирует радиус окрестности светлячка.

        Параметры:

            firefly (Firefly): Текущий светлячок.

            desired\_neighbors (int): Целевое количество соседей.

        '''

        current\_neighbors = len(self.calculate\_neighbors(firefly))

        new\_radius = firefly.radius + self.beta \* \

            (desired\_neighbors - current\_neighbors)

        firefly.radius = min(self.max\_radius, max(self.min\_radius, new\_radius))

    def optimize(self) -> Tuple[List[float], float]:

Продолжение Листинга А

        '''

        Запускает процесс оптимизации.

        Возвращает:

            Tuple[List[float], float]: Лучшая позиция и значение целевой функции.

        '''

        self.initialize\_fireflies()

        best\_position = None

        best\_value = float('inf')

        desired\_neighbors = 5

        for iteration in range(self.max\_iterations):

            for firefly in self.fireflies:

                function\_value = self.fitness\_function(\*firefly.position)

                firefly.update\_luciferin(function\_value, self.rho, self.gamma)

            iteration\_positions = []

            for firefly in self.fireflies:

                neighbors = self.calculate\_neighbors(firefly)

                if neighbors:

                    probabilities = self.calculate\_probabilities(

                        firefly, neighbors)

                    selected\_neighbor = self.select\_neighbor(

                        neighbors, probabilities)

                    firefly.move\_towards(

                        selected\_neighbor, self.delta, self.bounds)

                iteration\_positions.append(firefly.position)

            self.positions\_history.append(iteration\_positions)

            for firefly in self.fireflies:

                self.adjust\_radius(firefly, desired\_neighbors)

            for firefly in self.fireflies:

                value = self.fitness\_function(\*firefly.position)

                if value < best\_value:

                    best\_value = value

                    best\_position = firefly.position

            self.best\_values.append(best\_value)

            print(f"Итерация {iteration +

                  1}: Лучший результат = {best\_value:.6f}")

        return best\_position, best\_value

    def plot\_history(self):

        '''Отображает график сходимости.'''

        plt.plot(self.best\_values)

        plt.title("Сходимость алгоритма роя светлячков")

        plt.xlabel("Итерации")

        plt.ylabel("Лучшее значение")

        plt.show()

    def visualize(self):

        '''Анимация перемещения светлячков с учётом яркости.'''

        fig, ax = plt.subplots()

        x\_min, x\_max = self.bounds[0]

        y\_min, y\_max = self.bounds[1]

        ax.set\_xlim(x\_min, x\_max)

        ax.set\_ylim(y\_min, y\_max)

        def update(frame):

Окончание Листинга А

            ax.clear()

            ax.set\_xlim(x\_min, x\_max)

            ax.set\_ylim(y\_min, y\_max)

            ax.set\_title(f"Итерация {frame + 1}")

            positions = self.positions\_history[frame]

            luciferin\_values = [firefly.luciferin for firefly in self.fireflies]

            max\_luciferin = max(luciferin\_values)

            min\_luciferin = min(luciferin\_values)

            normalized\_brightness = [

                (l - min\_luciferin) / (max\_luciferin - min\_luciferin + 1e-9)

                for l in luciferin\_values

            ]

            x\_coords, y\_coords = zip(\*positions)

            ax.scatter(

                x\_coords, y\_coords,

                c="green",

                s=10,

                alpha=normalized\_brightness

            )

        anim = FuncAnimation(

            fig, update, frames=len(self.positions\_history), blit=False, interval=500, repeat=False

        )

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    swarm = FireflySwarm(

        fitness\_function=goldstein\_price,

        bounds=[(-2, 2), (-2, 2)],

        num\_fireflies=100,

        max\_iterations=200,

        beta=0.6,

        rho=0.4,

        delta=0.25,

        gamma=1.0,

        initial\_radius=0.5

    )

    best\_position, best\_value = swarm.optimize()

    print(f"Оптимальное решение: {best\_position}, Значение: {best\_value:.6f}")

    swarm.plot\_history()

    swarm.visualize()