Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 10

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

Реализовать алгоритм отжига для поиска глобального оптимума(минимума) произвольной функции. В качестве примера взять функцию F(x) = x ^ 2 + 10 - 10 * cos (2 * pi * x) Сам алгоритм выглядит следующим образом:

- Задать начальное значение (можно выбирать случайно)
- Изменить значение температуры при помощи заданной функции T(k), где k это номер итерации, получив температуру T[k]
- Сгенерировать новую точку x[k+1], с которой будет сравниваться текущий вариант(возможна случайная генерация, или использование какой либо функции от температуры)
- Вычислить значение искомой функции F(x) в точке x[k+1] и вычислить разницу между F(x[k + 1]) - F(x[k]) = dF
- Проверка решения на вероятность принятий: { 1 при dF < 0 P (x[k] , x[k+1]) = { { exp(-dF / T[k])
- Проверяем критерий завершения, критерием является некоторая температура окончания. Воспользуемся вариантом быстрого отжига: T(k) = T[0] / k A(x) = x + T * C(0,1), где C это случайно число сгенерированное при помощи распределения коши.
- После реализации, требуется построить график зависимости времени нахождения решения от значения минимальной температуры, при это ось x пусть будет инвертированной температурой, полученный график проанализировать.

Описание метода/модели.

Алгоритм отжига

Алгоритм имитации отжига — это метаэвристический алгоритм оптимизации, основанный на аналогии с процессом отжига металлов (медленного охлаждения для улучшения структуры). Он используется для нахождения приближённого решения сложных задач оптимизации, особенно в ситуациях, где градиентные методы неприменимы.

Принцип работы:

1. Начальная конфигурация:

Выбирается начальное решение (случайное или эвристическое).

2. Выбор соседнего решения:

Генерируется случайное изменение текущего решения.

3. Оценка решения:

Вычисляется функция стоимости (энергии) E(x) для нового решения.

4. Принятие или отклонение нового решения:

Если новое решение лучше (E(new) <E(old)), оно принимается.

Если хуже, принимается с вероятностью:

где T- "температура", управляющая вероятностью принятия плохих решений.

5. Уменьшение температуры (т)

Постепенно снижается по закону, например: T=Toια,

αε (0,1)

При низкой температуре алгоритм становится жадным и почти всегда принимает только улучшения.

6. Завершение алгоритма

Останавливается, когда температура слишком низкая или достигнуто стабильное решение.

Характеристики и применение:

- Имеет шанс выйти из локального минимума благодаря временному принятию худших решений.
- Применяется в задачах комбинаторной оптимизации, таких как:
- Задача коммивояжёра
- Раскраска графов
- Оптимизация расписаний
- Размещение электронных схем
- Сложность алгоритма зависит от параметров, но в среднем работает за O(N log N).

Плюсы и минусы:

- Гибкость (работает в любых задачах с функцией оценки).
- Простота реализации.
- Выбор параметров (температуры, шага) сложен.
- Не гарантирует нахождение глобального минимума (но даёт хорошее приближённое решение).

Алгоритм имитации отжига полезен там, где точные методы слишком дорогие, а случайный поиск слишком неэффективен.

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

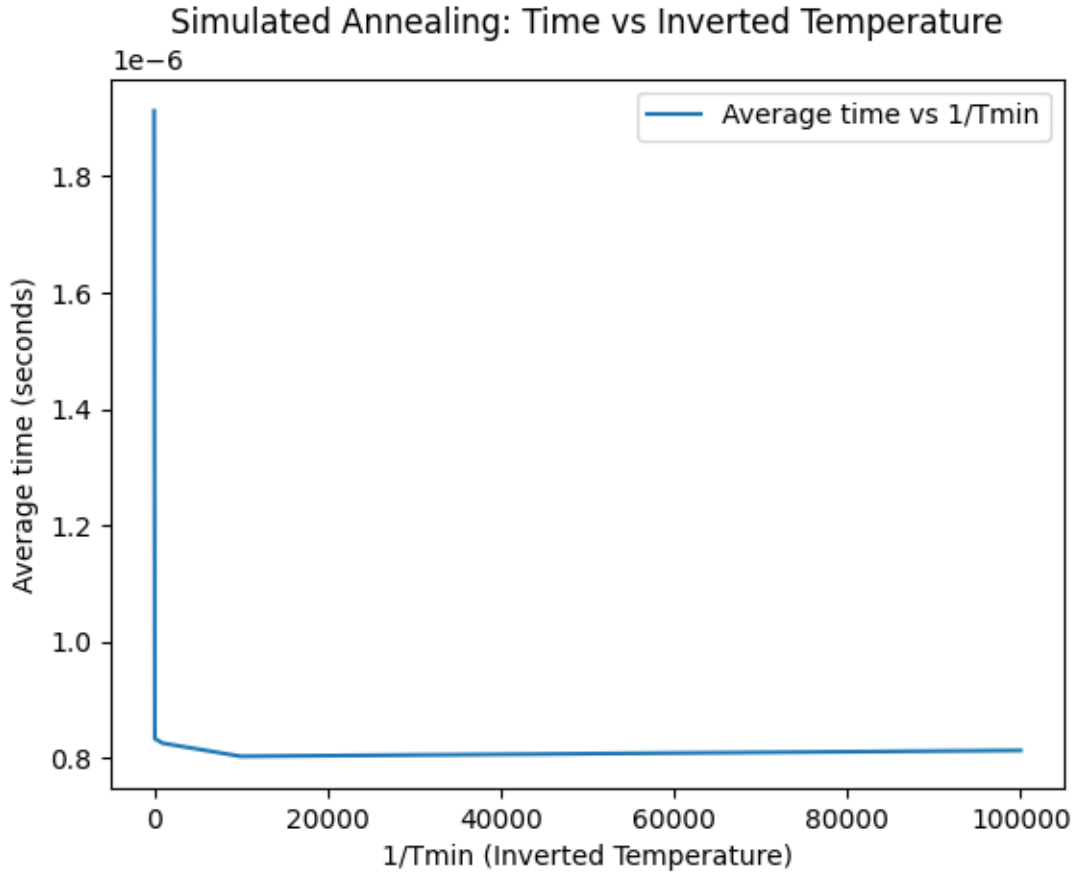
Блок кода отвечающий за реализацию алгоритма отжига:

```
    def simulated_annealing(T0, Tmin, max_iter):
        # Начальная случайная точка
        x_current = random.uniform(-5, 5)
        F_current = objective_function(x_current)
        T = T0
        k = 1
        # Регистрируем время на каждой температуре
        times = []
        while T > Tmin and k <= max_iter:
            start_time = time.time()  # Начало замера времени
            # Генерация новой точки с учетом температуры
            x_new = x_current + T * np.random.standard_cauchy()  # заменили random на numpy
            F_new = objective_function(x_new)
            # Разница между значениями функции
            dF = F_new - F_current
            # Если dF < 0, принимаем новую точку
            if dF < 0 or random.random() < math.exp(-dF / T):
                x_current = x_new
                F_current = F_new
            # Охлаждаем температуру
            T = T0 / k
            k += 1
            # Окончание замера времени
            end_time = time.time()
            # Записываем время работы на каждом шаге
            times.append(end_time - start_time)
        return x_current, F_current, times
```

Блоки кода отвечающие за тестирование:

```
    # Тестирование для разных Tmin
    def test_minimum_temperature():
        Tmin_values = [0.1, 0.01, 0.001, 0.0001, 0.00001]
        T0 = 1000
        max_iter = 10000
        times_results = []
        temperatures = []
        for Tmin in Tmin_values:
            _, _, times = simulated_annealing(T0, Tmin, max_iter)
            times_results.append(np.mean(times))  # усредненное время на каждом Tmin
            temperatures.append(1 / Tmin)  # инвертированная температура как ось X
        # Построение графика зависимости времени от инвертированной температуры
        plt.plot(temperatures, times_results, label="Average time vs 1/Tmin")
        plt.xlabel("1/Tmin (Inverted Temperature)")
        plt.ylabel("Average time (seconds)")
        plt.title("Simulated Annealing: Time vs Inverted Temperature")
        plt.legend()
        plt.show()
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже.



Заключение.

Алгоритм имитации отжига (Simulated Annealing) — мощный метод приближённой оптимизации, позволяющий избежать локальных минимумов за счёт контролируемого случайного поиска.

- Достоинства: Простая реализация, подходит для сложных задач, эффективен при правильной настройке.
- Недостатки: Зависит от параметров (температура, скорость охлаждения), не гарантирует нахождение глобального оптимума.

Производительность зависит от количества итераций и стратегии охлаждения (O(N log N)).

Хорош для комбинаторных задач (коммивояжёр, раскраска графов, маршрутизация).

Требует настройки параметров для хороших результатов.