Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 10

ПО КУРСУ

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Ведущий преподаватель

Ассистент Крашенинников Р. С.

СТУДЕНТ группы КС-36 Лупинос А. В.

Москва

2025

Задание

В рамках лабораторной работы необходимо решить ниже приведенную задачу.

Реализовать алгоритм отжига для поиска глобального оптимума (минимума) произвольной функции. В качестве примера использовать функцию:

$$F(x) = x^2 + 10 - 10 \cdot cos(2\pi x).$$

Описание алгоритма:

1. Задать начальное значение x_0 (можно выбрать случайно);
2. Изменить значение температуры с использованием функции $T(k)$, где k — номер итерации, получив температуру T_k;
3. Сгенерировать новую точку x_{k+1} для сравнения с текущей (возможна случайная генерация или использование функции, зависящей от температуры);
4. Вычислить значение искомой функции $F(x)$ в точке x_{k+1} и определить разность $\Delta F = F(x_{k+1}) - F(x_k)$;
5. Проверить вероятность принятия решения:

$$P(x_k, x_{k+1}) = \begin{cases} 1, & \text{если } \Delta F < 0, \\ \exp\left(-\frac{\Delta F}{T_k}\right), & \text{если } \Delta F \geq 0. \end{cases}$$

6. Проверить критерий завершения алгоритма (например, достижение заданной минимальной температуры).

Для реализации алгоритма применить вариант быстрого отжига, используя следующие математические формулы:

- Температура на итерации k:

$$T_k = \frac{T_0}{k},$$

где T_0 — начальная температура, заданная в начале алгоритма, а k — номер текущей итерации.

- Генерация новой точки:

$$x_{k+1} = x_k + T_k \cdot C(0, 1),$$

где x_k — текущая точка, T_k — текущая температура, а $C(0, 1)$ — случайное число, полученное с использованием распределения Коши с параметрами $x_0 = 0$ (центр) и $\gamma = 1$ (масштаб).

См. документацию по распределению Коши в C++: cauchy distribution.

После реализации требуется построить график зависимости времени нахождения решения от значения минимальной температуры, при этом на оси x отобразить инвертированную температуру $\frac{1}{T}$. Полученный график проанализировать.

Описание алгоритма

Эвристика — это подход к решению задач, который использует практические методы для получения приемлемых решений, когда точное решение требует слишком много времени или вычислительных ресурсов. Эвристика применяется в задачах, где традиционные методы, такие как полный перебор или аналитическое решение, слишком медленные или неосуществимы. Они основаны на интуитивных или эмпирических правилах, которые ускоряют процесс поиска решения.

Эвристический алгоритм — это алгоритм, который не гарантирует нахождение оптимального решения для всех возможных случаев, но, как правило, предоставляет достаточно хорошие решения за разумное время. Такие алгоритмы особенно полезны для сложных оптимизационных задач, где пространство поиска велико, а точное решение требует экспоненциального времени. Эвристические алгоритмы жертвуют математической строгостью ради практической эффективности.

Существует несколько типов эвристических алгоритмов, включая:

- Методы Монте-Карло: основаны на многократной случайной выборке для получения численных результатов; используются в симуляциях и оптимизации;
- Алгоритмы локального поиска: начинают с начального решения и итеративно переходят к соседним решениям, чтобы найти лучшее; примеры: восхождение на холм, табу-поиск;
- Имитация отжига: вероятностная техника, которая аппроксимирует глобальный оптимум функции, моделируя процесс охлаждения металлов.

Эвристические алгоритмы широко применяются в различных областях:

- Операционные исследования: планирование, маршрутизация, логистика;
- Искусственный интеллект: игры, машинное обучение, обработка естественного языка;

- Инженерный дизайн: оптимизация конструкций, размещение компонентов;
- Биоинформатика: анализ последовательностей ДНК, предсказание структуры белков;
- Финансы: оптимизация портфелей, прогнозирование.

Эвристические алгоритмы обладают следующими характеристиками:

- Отсутствие гарантии оптимальности: не всегда находят лучшее решение;
- Возможность пропуска решения: могут не найти решение, даже если оно существует;
- Вероятность ошибки: в некоторых случаях дают неверные результаты;
- Практическая эффективность: быстрее и удобнее для больших задач по сравнению с точными методами.

Метод имитации отжига — это метаэвристический алгоритм оптимизации, вдохновлённый процессом отжига в металлургии, где материал нагревают до высокой температуры и медленно охлаждают для уменьшения дефектов и достижения минимального энергетического состояния. Аналогично, алгоритм начинает с высокой "температуры", позволяя широко исследовать пространство решений, и постепенно снижает температуру, фокусируясь на улучшении решения.

Метод имитации отжига использует вероятностный подход для перехода между состояниями в пространстве поиска. Ключевой механизм — вероятность принятия нового решения, которая зависит от разницы в значениях целевой функции ΔF и текущей температуры T. Формула вероятности принятия представлена на рисунке 1.

$$P(x_k, x_{k+1}) = \begin{cases} 1, & \text{если } \Delta F < 0, \\ \exp\left(-\frac{\Delta F}{T_k}\right), & \text{если } \Delta F \geq 0. \end{cases}$$

Рис. 1 – Формула вероятности принятия нового решения

Если $\Delta F < 0$ (новое решение лучше), оно принимается автоматически. При высоких температурах алгоритм чаще принимает худшие решения, что позволяет избегать локальных минимумов. При низких температурах алгоритм становится более избирательным, приближаясь к оптимальному решению.

Этапы алгоритма имитации отжига:

1. Инициализация:
    a. Выбирается начальное состояние x_0 (случайно или заданное),
    b. Задаётся начальная температура T_0;
2. Итерации: пока текущая температура $T_k > T_{min}$:
    a. Генерируется новое состояние x_{k+1},
    b. Вычисляется разность $\Delta F = F(x_{k+1}) - F(x_k)$,
    c. Если $\Delta F < 0$, принимается x_{k+1},
    d. Если $\Delta F \geq 0$, x_{k+1} принимается с вероятностью $exp(-\frac{\Delta F}{T_k})$,
    e. Температура обновляется по заданному закону (в данном случае – быстрый отжиг, в котором график охлаждения имеет вид: $T_k = \frac{T_0}{k}$);
3. Завершение:
    a. Алгоритм останавливается при достижении минимальной температуры T_{min},
    b. Возвращается последнее найденное состояние.

Особенности алгоритма имитации отжига:

- Температурный параметр: контролирует баланс между исследованием и использованием пространства решений;
- Вероятность принятия: позволяет алгоритму избегать локальных минимумов;
- График охлаждения: определяет, как температура уменьшается с течением времени. Примеры: линейное, экспоненциальное, логарифмическое охлаждение.

Свойства алгоритма:

- Вероятностная сходимость: при правильной настройке параметров алгоритм может найти глобальный минимум с высокой вероятностью;
- Зависимость от параметров: эффективность зависит от начальной температуры, графика охлаждения и критерия остановки;
- Гибкость: применим к дискретным и непрерывным задачам оптимизации.

Преимущества метода отжига:

- Способность избегать локальных минимумов за счёт вероятностного принятия худших решений;
- Простота реализации по сравнению с другими глобальными методами оптимизации;
- Адаптивность к различным типам задач.

Алгоритм имитации отжига имеет следующие недостатки:

- Высокая вычислительная сложность при медленных графиках охлаждения;
- Необходимость тщательной настройки параметров (начальная температура, скорость охлаждения);
- Отсутствие гарантии нахождения глобального минимума, только вероятностная сходимость.

Временная сложность алгоритма зависит от числа итераций и стоимости вычисления целевой функции и генерации новых решений.

Алгоритм использует постоянное количество памяти (переменные для текущей точки x, температуры T, значений функции и генераторов случайных чисел). Таким образом, пространственная сложность — $O(1)$.

Быстрый отжиг является менее точным в общем случае, но быстрее для задач с выраженным глобальным минимумом, как заданная функция. Временная сложность: $O(\frac{T_0}{T_{min}})$, где T_0 — начальная температура, T_{min} — минимальная температура.

Описание выполнения задачи

Для реализации задачи была написана программа на C++:

```
#include <iostream>

#include <cmath>

#include <random>

#include <chrono>

#include <fstream>

#include <vector>


using namespace std;


const double PI = 3.14159265358979323846;  // Число Пы


// Целевая функция

double F(double x) {

    return x * x + 10 - 10 * cos(2 * PI * x);

}


// Алгоритм симуляции отжига

double SimulatedAnnealing(double x0, double T0, double Tmin, mt19937& gen) {

    double x = x0;  // Начальная координата

    double f = F(x);  // Значение в начальной координате

    double T = T0;  // Начальная температура

    int k = 1;  // Итерация
```

```
    // Распределение Коши для генерации новых координат
    cauchy_distribution<double> cauchy(0.0, 1.0);
    // Вероятность принятия решения
    uniform_real_distribution<double> uniform(0.0, 1.0);

    while (T > Tmin) {
        T = T0 / k;  // Температура на итерации
        // Генерация новой координаты
        double x_new = x + T * cauchy(gen);
        double f_new = F(x_new);  // Значение в новой координате
        // Разность между значением функции в новой и прошлой координатах
        double dF = f_new - f;
        // Условие принятия решения
        if (dF < 0 || uniform(gen) < exp(-dF / T)) {
            x = x_new;
            f = f_new;
        }

        k++;
    }

    return x;
}

int main() {
    random_device rd;
```

```
mt19937 gen(rd());

double x0 = 1.0;  // Заданная начальная координата
double T0 = 1000.0;  // Начальная температура

// Критерий завершения
double Tmin_start = 1e-1;  // Верхняя граница погрешности
double Tmin_end = 1e-10;  // Нижняя граница погрешности
double step = 10.0;  // Шаг для расчета погрешностей

vector<double> Tmin_values;
double T = Tmin_start;
while (T >= Tmin_end) {
    Tmin_values.push_back(T);
    T /= step;
}
cout << "Generated T: ";
for (double T_value : Tmin_values) {
    cout << T_value << " ";
}
cout << endl;

ofstream out("results.csv");
out << "Tmin,1 / Tmin,Время (мс),x,F(x)\n";

int i = 1;  // Номер текущей погрешности
```

```
int N = Tmin_values.size();  // Количество погрешностей

for (double Tmin : Tmin_values) {
    double inverse_Tmin = 1.0 / Tmin;  // Для графиков

    // Вывод прогресса в консоль
    cout << i << "/" << N
         << ": Tmin = " << Tmin << "..." << endl;

    gen.seed(rd());
    auto start = chrono::high_resolution_clock::now();
    double x_result = SimulatedAnnealing(x0, T0, Tmin, gen);
    auto end = chrono::high_resolution_clock::now();
    double time_ms = chrono::duration<double, milli>(end - start).count();

    cout << "Time: " << time_ms << " ms\n" << endl;

    out << Tmin << "," << inverse_Tmin << "," << time_ms << "," << x_result
        << "," << F(x_result) << "\n";

    i++;  // Следующая погрешность
}

out.close();
cout << "Results saved to results.csv\n";
```

```
    return 0;
}
```

Программа реализует алгоритм имитации отжига для поиска глобального минимума функции $F(x) = x^2 + 10 - 10 \cdot cos(2\pi x)$ и проводит эксперименты для оценки зависимости времени выполнения от минимальной температуры T_{min}.

Основные функции программы:

- Оптимизация функции: поиск точки *x*, минимизирующей функцию *F*(*x*), с использованием метода имитации отжига (быстрый отжиг);
- Эксперименты с разными T_{min}: измерение времени выполнения для значений T_{min} от 10^{-1} до 10^{-10}, а также запись найденных значений *x* и *F*(*x*). Результаты сохраняются в файл results.csv;
- Анализ зависимости: для каждого T_{min} записывается инвертированная температура $\frac{1}{T_{min}}$, время выполнения, найденная точка *x* и значение функции *F*(*x*), что позволяет построить график зависимости времени от $\frac{1}{T_{min}}$.

Специфические элементы реализации:

- Функция SimulatedAnnealing: реализует метод имитации отжига с использованием быстрого отжига. Начальная точка $x_0 = 5.0$, начальная температура $T_0 = 1000$. Температура на каждой итерации вычисляется как $T_k = \frac{T_0}{k}$, где *k* — номер итерации;
- Генерация новых точек: новая точка x_{new} генерируется с использованием распределения Коши: $x_{new} = x + T_k \cdot C(0, 1)$, где $C(0, 1)$ — случайное число из распределения Коши с центром 0 и масштабом 1 (реализовано через cauchy_distribution);
- Вероятность принятия: для новой точки вычисляется разность значений функции $\Delta F = F(x_{new}) - F(x)$. Если $\Delta F < 0$, точка принимается; если

$\Delta F \geq 0$, точка принимается с вероятностью $exp(-\frac{\Delta F}{T_k})$, что реализовано с помощью uniform_real_distribution для генерации случайного числа от 0 до 1;

- Критерий остановки: алгоритм завершается, когда температура T_k становится меньше или равной заданной минимальной температуры T_{min};
- Экспериментальный цикл: программа перебирает значения T_{min} от 10^{-1} до 10^{-10}, уменьшая T_{min} в 10 раз на каждом шаге. Для каждого T_{min} выполняется оптимизация, замеряется время с помощью chrono::high_resolution_clock, и результаты записываются в results.csv;
- Случайность: используются генераторы random_device и mt19937 для обеспечения качественной случайности при генерации новых точек и принятии решений;
- Вывод прогресса: для удобства отслеживания программа выводит в консоль текущий T_{min}, прогресс эксперимента (например, "1/10") и время выполнения каждого шага.

Выводы

В ходе работы был реализован алгоритм имитации отжига для поиска минимума функции $F(x) = x^2 + 10 - 10 \cdot cos(2\pi x)$. Был проведен эксперимент для получения зависимости времени нахождения решения от значения T_{min} от 10^{-1} до 10^{-10}. Результат работы программы представлен в таблице 1. На основе полученных данных был построен график зависимости времени нахождения решения от значения минимальной температуры (рисунок 2).

Tmin	1 / Tmin	Время (мс)	x	F(x)
0,1	10	3,13095	0,0106757	0,0226024
0,01	100	26,8587	-0,000915731	0,000166364
0,001	1000	252,593	-0,00280207	0,00155765
0,0001	10000	2361,08	0,000684671	9,30E-05
1,00E-05	100000	22520,1	-5,40E-05	5,79E-07
1,00E-06	1,00E+06	216895	8,81E-06	1,54E-08
1,00E-07	1,00E+07	464139	1,32E-05	3,46E-08
1,00E-08	1,00E+08	465000	1,52E-05	4,57E-08
1,00E-09	1,00E+09	469389	-1,49E-05	4,42E-08
1,00E-10	1,00E+10	471886	-1,80E-05	6,45E-08

Таблица 1 – Результаты работы программы

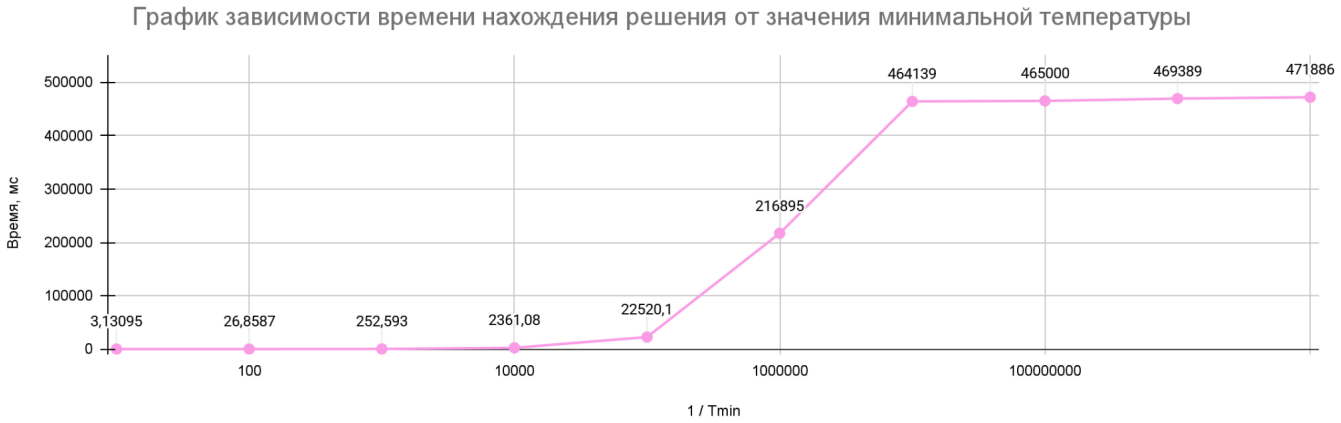

Рисунок 2 – График зависимости времени нахождения решения от значения минимальной температуры

Для наглядности график был построен по инвертированной температуре $\frac{1}{T_{min}}$ и с использованием логарифмической шкалы на оси X. Анализ таблицы и графика показал, что время работы алгоритма увеличивается с уменьшением T_{min} не линейно: до 10^{-6} рост составляет примерно 9 – 10 раз на каждое уменьшение T_{min}, а после 10^{-6} график становится почти плоским, увеличиваясь всего в ~2,2 раза до 471886 мс при $T_{min} = 10^{-10}$, что связано с особенностями быстрого отжига и случайностью шагов. Точность решения после $T_{min} = 10^{-6}$ перестает уменьшаться на порядок.