人工智能 上机实验报告

**学号：** 1908090117 **姓名： 李泉志 所在系：** 信息学院 **班级：** 191班

**实验名称： 遗传算法的实现与应用 实验日期 2021年11月4日**

**实验指导教师**  **李 辉**   **实验机房 D2-602**

------------------------------------------------------------------------------------------------------

1. 实验目的：

(1)理解遗传算法的基本过程，掌握其选择-复制、交叉、变异三种运算。

(2)掌握基本遗传算法、粒子群算法，会建立适应度函数，通过算法的迭代实现优化问题。

2. 实验内容及要求：

(1)采用遗传算法求解以下函数的最大值：

其中

(2)采用基本粒子群算法求解以下函数的最小值：

(在实现过程中体会迭代步数、粒子群规模、学习因子、惯性权重对结果的影响)

3．**算法设计**（编程思路或流程图或关键源代码）

代码仓库：https://github.com/ZlycerQan/AIExperiments

// genetic\_algorithm.cpp

#include <random>

#include <iostream>

#include <vector>

#include <bitset>

#include <chrono>

#include <algorithm>

int identification() {

    std::cout << "+------------+-----------+" << '\n';

    std::cout << "| 1908090117 | Liquanzhi |" << '\n';

    std::cout << "+------------+-----------+" << '\n';

    return 1;

}

int me = identification();

using flo = double;

constexpr int NUMBER\_PERCISION = 4;

constexpr int NUMBER\_EXPANSION\_VALUE = pow(10, NUMBER\_PERCISION);

constexpr int NUMBER\_OF\_EVOLUTION = 10000;

constexpr int INITIAL\_NUMBER = 400;

constexpr flo LOWER\_BOUND = -2.048;

constexpr flo UPPER\_BOUND = 2.048;

constexpr flo CROSS\_LEFT\_GENE\_PROBABILITY = 0.2;

constexpr flo CROSS\_RIGHT\_GENE\_PROBABILITY = 0.2;

constexpr flo MUTATE\_LEFT\_GENE\_PROBABILITY = 0.07;

constexpr flo MUTATE\_RIGHT\_GENE\_PROBABILITY = 0.07;

constexpr flo CROSS\_PROBABILITY = CROSS\_LEFT\_GENE\_PROBABILITY + CROSS\_RIGHT\_GENE\_PROBABILITY;

constexpr flo MUTATE\_PROBABILITY = MUTATE\_LEFT\_GENE\_PROBABILITY + MUTATE\_RIGHT\_GENE\_PROBABILITY;

constexpr int BIT\_LENGTH = log2((UPPER\_BOUND - LOWER\_BOUND) \* NUMBER\_EXPANSION\_VALUE) + 1;

constexpr int BORDER = LOWER\_BOUND < 0 ? -LOWER\_BOUND \* NUMBER\_EXPANSION\_VALUE : 0;

using gene = std::bitset<BIT\_LENGTH>;

gene encode(const flo& value) {

    return gene(value \* NUMBER\_EXPANSION\_VALUE + BORDER);

}

flo decode(const gene& value) {

    return (static\_cast<flo>(value.to\_ulong()) - BORDER) / NUMBER\_EXPANSION\_VALUE;

}

using chromosome = std::pair<gene, gene>;

using group = std::vector<chromosome>;

inline flo f(const chromosome& x) {

    const flo x1 = decode(x.first);

    const flo x2 = decode(x.second);

    return 100.0 \* (x1 \* x1 - x2) \* (x1 \* x2 - x2) + (1 - x1) \* (1 - x1);

}

std::mt19937 rng = std::mt19937(std::chrono::steady\_clock::now().time\_since\_epoch().count());

std::uniform\_real\_distribution<> get\_random(const flo& l, const flo& r) {

    return std::uniform\_real\_distribution<>(l, r);

}

inline bool contains(const gene& x) {

    flo value = decode(x);

    return value >= LOWER\_BOUND && value <= UPPER\_BOUND;

}

void select(group& popu) {

    const int n = popu.size();

    auto gen = get\_random(0, 1);

    std::vector<flo> adapt;

    flo sum = 0;

    for (const chromosome& i : popu) {

        adapt.emplace\_back(f(i));

        sum += adapt.back();

    }

    for (flo& i : adapt) {

        i = i / sum;

    }

    for (int i = 1; i < n; ++ i) {

        adapt[i] = adapt[i - 1] + adapt[i];

    }

    group result = popu;

    for (int i = 0; i < n; ++ i) {

        popu[i] = result[std::upper\_bound(adapt.begin(), adapt.end(), gen(rng)) - adapt.begin()];

    }

}

void cross\_gene(gene& l, gene& r, int lp, int rp = BIT\_LENGTH) {;

    for (int i = lp; i < rp; ++ i) {

        int p = l[i], q = r[i];

        l.set(i, q);

        r.set(i, p);

    }

    if (!contains(l) || !contains(r)) {

        for (int i = lp; i < rp; ++ i) {

            int p = l[i], q = r[i];

            l.set(i, q);

            r.set(i, p);

        }

    }

}

void cross\_chromosome(chromosome& lhs, chromosome& rhs) {

    auto gen = get\_random(0, 1);

    if (gen(rng) <= CROSS\_LEFT\_GENE\_PROBABILITY) {

        int p = gen(rng) \* BIT\_LENGTH;

        cross\_gene(lhs.first, rhs.first, p);

    }

    if (gen(rng) <= CROSS\_RIGHT\_GENE\_PROBABILITY) {

        int p = gen(rng) \* BIT\_LENGTH;

        cross\_gene(lhs.second, rhs.second, p);

    }

}

void cross(group& popu) {

    auto gen = get\_random(0, 1);

    std::shuffle(popu.begin(), popu.end(), rng);

    for (int i = 0; i < (int)popu.size(); i += 2) {

        if (gen(rng) <= CROSS\_PROBABILITY) {

            cross\_chromosome(popu[i], popu[i + 1]);

        }

    }

}

void mutate\_chromosome(chromosome& x) {

    auto gen = get\_random(0, 1);

    if (gen(rng) <= MUTATE\_LEFT\_GENE\_PROBABILITY) {

        int p = gen(rng) \* BIT\_LENGTH;

        x.first.flip(p);

        if (!contains(x.first)) {

            x.first.flip(p);

        }

    }

    if (gen(rng) <= MUTATE\_RIGHT\_GENE\_PROBABILITY) {

        int p = gen(rng) \* BIT\_LENGTH;

        x.second.flip(p);

        if (!contains(x.second)) {

            x.second.flip(p);

        }

    }

}

void mutate(group& popu) {

    auto gen = get\_random(0, 1);

    for (chromosome& i : popu) {

        if (gen(rng) <= MUTATE\_PROBABILITY) {

            mutate\_chromosome(i);

        }

    }

}

void run\_genetic\_algorithm() {

    auto gen = get\_random(LOWER\_BOUND, UPPER\_BOUND);

    group popu;

    for (int i = 1; i <= INITIAL\_NUMBER; ++ i) {

        popu.emplace\_back(encode(gen(rng)), encode(gen(rng)));

    }

    flo ans = 0;

    chromosome res;

    for (int cas = 1; cas <= NUMBER\_OF\_EVOLUTION; ++ cas) {

        std::vector<flo> adapt(popu.size());

        select(popu);

        cross(popu);

        mutate(popu);

        for (const chromosome& i : popu) {

            if (f(i) > ans) {

                ans = f(i);

                res = i;

            }

        }

    }

    std::cout << "x1: " << decode(res.first) << " x2: " << decode(res.second) << '\n';

    std::cout << "ans: " << ans;

}

int main() {

    run\_genetic\_algorithm();

    return 0;

}

// pso\_algorithm.cpp

#include <iostream>

#include <array>

#include <random>

#include <chrono>

#include <iomanip>

int identification() {

    std::cout << "+------------+-----------+" << '\n';

    std::cout << "| 1908090117 | Liquanzhi |" << '\n';

    std::cout << "+------------+-----------+" << '\n';

    return 1;

}

int me = identification();

using flo = double;

constexpr int NUMBER\_OF\_EVOLUTION = 862353;

constexpr int X\_NUMBER = 30;

constexpr int PARTICLE\_NUMBER = 30;

constexpr flo UPPER\_BOUND = 2;

constexpr flo LOWER\_BOUND = 0;

constexpr flo C1 = 0.05;

constexpr flo C2 = 0.05;

constexpr flo V\_MAX = 0.0005;

using particle = std::array<flo, X\_NUMBER>;

using swarm = std::array<particle, PARTICLE\_NUMBER>;

using velocity = std::array<flo, X\_NUMBER>;

std::mt19937 rng = std::mt19937(std::chrono::steady\_clock::now().time\_since\_epoch().count());

std::uniform\_real\_distribution<> get\_real\_random(const flo& l, const flo& r) {

    return std::uniform\_real\_distribution<>(l, r);

}

swarm create\_initial\_swarm() {

    swarm result;

    auto gen  = get\_real\_random(LOWER\_BOUND, UPPER\_BOUND);

    for (int i = 0; i < PARTICLE\_NUMBER; ++ i) {

        particle x;

        for (int j = 0; j < X\_NUMBER; ++ j) {

            x[j] = gen(rng);

        }

        result[i] = std::move(x);

    }

    return result;

}

flo f(const particle& x) {

    flo result = 0;

    for (const flo& i : x) {

        result += i \* i;

    }

    return result;

}

void run\_pso() {

    flo result = 1e9;

    particle result\_pos;

    auto gen = get\_real\_random(0, 1);

    swarm sw = create\_initial\_swarm();

    std::array<flo, PARTICLE\_NUMBER> p\_value;

    std::array<particle, PARTICLE\_NUMBER> p\_pos;

    std::array<velocity, PARTICLE\_NUMBER> v;

    for (int i = 0; i < PARTICLE\_NUMBER; ++ i) {

        p\_value[i] = f(sw[i]);

        p\_pos[i] = sw[i];

        v[i].fill(0);

    }

    for (int evo = 1; evo <= NUMBER\_OF\_EVOLUTION; ++ evo) {

        particle g\_pos;

        flo g\_value = 1e9;

        for (int i = 0; i < PARTICLE\_NUMBER; ++ i) {

            flo y = f(sw[i]);

            if (y < result) {

                result = y;

                result\_pos = sw[i];

            }

            if (y < p\_value[i]) {

                p\_value[i] = y;

                p\_pos[i] = sw[i];

            }

            if (y < g\_value) {

                g\_value = y;

                g\_pos = sw[i];

            }

        }

        for (int i = 0; i < PARTICLE\_NUMBER; ++ i) {

            for (int j = 0; j < X\_NUMBER; ++ j) {

                v[i][j] += C1 \* gen(rng) \* (p\_pos[i][j] - sw[i][j]) + C2 \* gen(rng) \* (g\_pos[j] - sw[i][j]);

                if (v[i][j] >= V\_MAX) {

                    v[i][j] = V\_MAX;

                }

                if (v[i][j] <= -V\_MAX) {

                    v[i][j] = -V\_MAX;

                }

                sw[i][j] += v[i][j];

                if (sw[i][j] > UPPER\_BOUND) {

                    sw[i][j] = LOWER\_BOUND + fmod(sw[i][j] - LOWER\_BOUND, UPPER\_BOUND - LOWER\_BOUND);

                } else if (sw[i][j] < LOWER\_BOUND) {

                    sw[i][j] = UPPER\_BOUND - fmod(UPPER\_BOUND - sw[i][j], UPPER\_BOUND - LOWER\_BOUND);

                }

            }

        }

    }

    for (int i = 0; i < X\_NUMBER; ++ i) {

        std::cout << "x" << i + 1 << ": " << std::fixed << std::setprecision(2) << result\_pos[i] << '\n';

    }

    printf("%.2lf", result);

}

int main() {

    run\_pso();

    return 0;

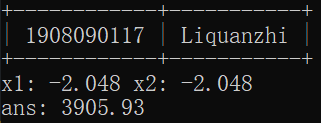
}

1. **程序调试**（实验数据记录——根据程序要求输入几组不同数据，记录程序运行结果，并分析结果,分析程序运行中出现的主要错误。或对其他程序环境的使用情况的记录。注：必须认真书写）

（1）

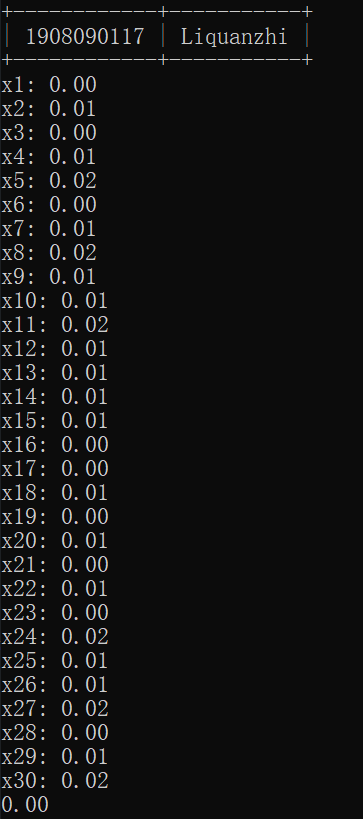
算法中采用的精度为4位，初始种群数量为400，左染色体交叉概率为0.2，右染色体交叉概率为0.2，左染色体变异概率为0.07，右染色体变异概率为0.07，迭代次数为1000次。

编码方案采用将所有数字首先扩大10000倍，舍弃小数部分，当作整数进行二进制编码。



（2）

算法中采用的精度为2位，初始粒子群大小为30，采用的搜索边界为上界2，下界0，学习因子c1为0.05，c2为0.05，速度最大限制为0.005，迭代次数为862353次。



5．**讨论**（通过实验的一些体会、学会的知识和技能等）

通过此次实验了解到了随机算法调节参数的一些方法，体会到了算法中不同的参数有着什么样的影响，调整参数时要兼顾精度与效率。