```cpp
#include "ionlib\log.h"
#include "ionlib\net.h"
#include "ionlib\genetic_algorithm.h"
#include <fstream>
#include <bitset>
#include <sstream>

int32_t signed_vector_to_int(std::vector<bool>::iterator first, std::vector<bool>::iterator end)
{
    LOGASSERT(end - first <= 32);
    uint32_t result = 0;
    for (std::vector<bool>::iterator it = first; it < end-1; ++it)
    {
        if (*it)
        {
            uint32_t offset = (uint32_t)(it - first);
            result |= 1 << offset;
        }
    }
    int32_t sign = (*(end-1)) ? -1 : 1;
    return sign * (int32_t)result;
}
class GANumOnes : public ion::GeneticAlgorithm
{
public:
    GANumOnes() = delete;
    GANumOnes(size_t num_members, size_t chromosome_length, double mutation_probability, double crossover_probability) : ion::GeneticAlgorithm(num_members, chromosome_length, mutation_probability, crossover_probability)
    {
        EvaluateMembers();
    }
    virtual void EvaluateMembers()
    {
        for (std::vector<std::vector<bool>>::iterator member_it = this->population_.begin(); member_it != this->population_.end(); ++member_it)
        {
            double fitness = 0.0;
            for (std::vector<bool>::iterator gene_it = member_it->begin(); gene_it != member_it->end(); ++gene_it)
            {
                if (*gene_it)
                {
                    fitness += 1.0 / member_it->size();
                }
            }
            this->fitness_[member_it - this->population_.begin()] = fitness;
            this->num_evaluations_++;
        }
    }
};

double dejong1(double x[3])
{
    return x[0]*x[0] + x[1]*x[1] + x[2]*x[2];
}

class GADejong1 : public ion::GeneticAlgorithm
{
public:
    GADejong1() = delete;
    GADejong1(size_t num_members, double mutation_probability, double crossover_probability) : ion::GeneticAlgorithm(num_members, num_chromosomes_*chromosome_length_, mutation_probability, crossover_probability)
    {
        double worst_x[3];
        worst_x[0] = worst_x[1] = worst_x[2] = -5.12;
        worst_fitness_ = dejong1(worst_x);
        EvaluateMembers();
    }
    virtual void EvaluateMembers()
    {
        for (std::vector<std::vector<bool>>::iterator member_it = this->population_.begin(); member_it != this->population_.end(); ++member_it)
        {
            this->num_evaluations_++;
            //convert to a value in range
            double x[num_chromosomes_];
            to_val(*member_it, x);
            //evaluate
            double raw_fitness = dejong1(x);
            //scale to [0.0,1.0]
            double fitness = (worst_fitness_ - raw_fitness) / worst_fitness_;
            LOGASSERT(fitness <= 1.0 && fitness >= 0.0);
            this->fitness_[member_it - population_.begin()] = fitness;
        }
    }
    static const uint32_t num_chromosomes_ = 3;
    static const uint32_t chromosome_length_ = 10;
    double worst_fitness_;
    void to_val(std::vector<bool> member, double x[num_chromosomes_])
    {
        for (uint32_t dim = 0; dim < num_chromosomes_; ++dim)
        {
            int32_t member_offset = signed_vector_to_int(member.begin() + dim*chromosome_length_, member.begin() + (dim + 1)*chromosome_length_);
            x[dim] = (double)member_offset / 100.0;
        }
    }
};
```

```cpp
double dejong2(double x[2])
{
    return 100 * pow(x[0]*x[0] - x[1], 2.0) + pow(1 - x[0], 2.0);
}

class GADejong2 : public ion::GeneticAlgorithm
{
public:
    GADejong2() = delete;
    GADejong2(size_t num_members, double mutation_probability, double crossover_probability) : ion::GeneticAlgorithm(num_members, num_chromosomes_*chromosome_length_, mutation_probability, crossover_probability)
    {
        double worst_x[2];
        worst_x[0] = worst_x[1] = -2.048;
        worst_fitness_ = dejong2(worst_x);
        EvaluateMembers();
    }
    virtual void EvaluateMembers()
    {
        for (std::vector<std::vector<bool>>::iterator member_it = this->population_.begin(); member_it != this->population_.end(); ++member_it)
        {
            this->num_evaluations_++;
            //convert to a value in range
            double x[num_chromosomes_];
            to_val(*member_it, x);
            //evaluate
            double raw_fitness = dejong2(x);
            //scale to [0.0,1.0]
            double fitness = (worst_fitness_ - raw_fitness) / worst_fitness_;
            LOGASSERT(fitness <= 1.0 && fitness >= 0.0);
            this->fitness_[member_it - population_.begin()] = fitness;
        }
    }
    static const uint32_t num_chromosomes_ = 2;
    static const uint32_t chromosome_length_ = 12;
    double worst_fitness_;
    void to_val(std::vector<bool> member, double x[num_chromosomes_])
    {
        for (uint32_t dim = 0; dim < num_chromosomes_; ++dim)
        {
            int32_t member_offset = signed_vector_to_int(member.begin() + dim*chromosome_length_, member.begin() + (dim + 1)*chromosome_length_);
            x[dim] = (double)member_offset / 1000.0;
        }
    }
};

double dejong3(double x[5])
```

```cpp
{
    return (double)((int32_t)x[0] + (int32_t)x[1] + (int32_t)x[2] + (int32_t)x[3] + (int32_t)x[4]);
}

class GADejong3 : public ion::GeneticAlgorithm
{
public:
    GADejong3() = delete;
    GADejong3(size_t num_members, double mutation_probability, double crossover_probability) : ion::GeneticAlgorithm(num_members, num_chromosomes_*chromosome_length_, mutation_probability, crossover_probability)
    {
        double worst_x[5];
        worst_x[0] = worst_x[1] = worst_x[2] = worst_x[3] = worst_x[4] = 5.12;
        worst_fitness_ = dejong3(worst_x);
        EvaluateMembers();
    }
    virtual void EvaluateMembers()
    {
        for (std::vector<std::vector<bool>>::iterator member_it = this->population_.begin(); member_it != this->population_.end(); ++member_it)
        {
            this->num_evaluations_++;
            //convert to a value in range
            double x[num_chromosomes_];
            to_val(*member_it, x);
            //evaluate
            double raw_fitness = dejong3(x) + worst_fitness_;
            //scale to [0.0,1.0]
            double fitness = (worst_fitness_*2 - raw_fitness) / (2*worst_fitness_);
            LOGASSERT(fitness <= 1.0 && fitness >= 0.0);
            this->fitness_[member_it - population_.begin()] = fitness;
        }
    }
    static const uint32_t num_chromosomes_ = 5;
    static const uint32_t chromosome_length_ = 10;
    double worst_fitness_;
    void to_val(std::vector<bool> member, double x[num_chromosomes_])
    {
        for (uint32_t dim = 0; dim < num_chromosomes_; ++dim)
        {
            int32_t member_offset = signed_vector_to_int(member.begin() + dim*chromosome_length_, member.begin() + (dim + 1)*chromosome_length_);
            x[dim] = (double)member_offset / 100.0;
        }
    }
};

```

```cpp
double dejong4(double x[30])
{
    double result = 0.0;
    for (uint32_t i = 0; i < 30; ++i)
    {
        double random_number = ion::random_normal_distribution(0.0, 1.0);
        result += i * pow(x[i], 4) + random_number;
    }
    return result;
}

class GADejong4 : public ion::GeneticAlgorithm
{
public:
    GADejong4() = delete;
    GADejong4(size_t num_members, double mutation_probability, double crossover_probability) : ion::GeneticAlgorithm(num_members, num_chromosomes_*chromosome_length_, mutation_probability, crossover_probability)
    {
        //note that we can't actually define a worst X for this function since it is random, however it is extremely unlikey we would exceed this value
        double worst_x[30];
        for (uint32_t x_index = 0; x_index < 30; ++x_index)
        {
            worst_x[x_index] = 1.28;
        }
        worst_fitness_ = dejong4(worst_x);
        EvaluateMembers();
    }
    virtual void EvaluateMembers()
    {
        for (std::vector<std::vector<bool>>::iterator member_it = this->population_.begin(); member_it != this->population_.end(); ++member_it)
        {
            this->num_evaluations_++;
            //convert to a value in range
            double x[num_chromosomes_];
            to_val(*member_it, x);
            //evaluate
            double raw_fitness = dejong4(x) + worst_fitness_;
            //scale to [0.0,1.0]
            double fitness = (worst_fitness_ * 2 - raw_fitness) / (2 * worst_fitness_);
            LOGASSERT(fitness <= 1.0 && fitness >= 0.0);
            this->fitness_[member_it - population_.begin()] = fitness;
        }
    }
    static const uint32_t num_chromosomes_ = 30;
    static const uint32_t chromosome_length_ = 8;
    double worst_fitness_;
    void to_val(std::vector<bool> member, double x[num_chromosomes_])
    {
        for (uint32_t dim = 0; dim < num_chromosomes_; ++dim)
        {
            int32_t member_offset = signed_vector_to_int(member.begin() + dim*chromosome_length_, member.begin() + (dim + 1)*chromosome_length_);
            x[dim] = (double)member_offset / 100.0;
        }
    }
};

void ExecuteGa(uint32_t population_size, double mutation_rate, double crossover_rate)
{

    std::ofstream fout;
    uint32_t dejong_num = 4;
    std::stringstream filename;
    filename << "DJ" << dejong_num << "_pop" << population_size << "_mut" << mutation_rate << "_xover" << crossover_rate << ".csv";
    fout.open(filename.str());
    fout << "Generation,Min,Max,Mean,Evals" << std::endl;
    double max_fitness[5000] = { 0 };
    double min_fitness[5000] = { 0 };
    double avg_fitness[5000] = { 0 };
    double num_evals[5000] = { 0 };
    double num_hits[5000] = { 0 };
    for (uint32_t trial = 0; trial < 30; ++trial)
    {
        //Change this next line to switch between functions
        GADejong4 algo(population_size, mutation_rate, crossover_rate);
        uint32_t generation = 0;
        max_fitness[generation] += algo.GetMaxFitness();
        min_fitness[generation] += algo.GetMinFitness();
        avg_fitness[generation] += algo.GetAverageFitness();
        num_evals[generation] += algo.GetNumEvals();
        num_hits[generation]++;
        for (generation = 1; algo.GetMaxFitness() < 0.99999999 && generation < 5000; ++generation)
        {
            algo.NextGeneration();
            max_fitness[generation] += algo.GetMaxFitness();
            min_fitness[generation] += algo.GetMinFitness();
            avg_fitness[generation] += algo.GetAverageFitness();
            num_evals[generation] += algo.GetNumEvals();
            num_hits[generation]++;
        }
        LOGINFO("Completed trial %u", trial);
    }
    //scale all of the computed values
    for (uint32_t generation_index = 0; generation_index < 5000; +
        +generation_index)
    {
        if (num_hits[generation_index] == 0)
        {
            break;
        }
        max_fitness[generation_index] /= num_hits[generation_index];
        min_fitness[generation_index] /= num_hits[generation_index];
        avg_fitness[generation_index] /= num_hits[generation_index];
        num_evals[generation_index] /= num_hits[generation_index];
        fout << generation_index << "," << min_fitness[generation_index] << "," << max_fitness[generation_index] << "," << avg_fitness[generation_index] << "," << num_evals[generation_index] << std::endl;
    }
    fout.close();
}

int main(int argc, char* argv[])
{
    ion::Error result = ion::InitSockets();
    ion::LogInit("genetic_algorithm");
    //open a file for logging results
    uint32_t population_set[3] = { 50, 100, 150 };
    double mutation_set[3] = { 0.0001, 0.001, 0.01 };
    double crossover_set[3] = { 0.2, 0.67, 0.99 };
    for (uint32_t pop_choice = 0; pop_choice < 3; ++pop_choice)
    {
        for (uint32_t mutation_choice = 0; mutation_choice < 3; ++mutation_choice)
        {
            for (uint32_t crossover_choice = 0; crossover_choice < 3; ++crossover_choice)
            {
                ExecuteGa(population_set[pop_choice], mutation_set[mutation_choice], crossover_set[crossover_choice]);
                LOGINFO("Completed pop %d, mutation %d, crossover %d", pop_choice, mutation_choice, crossover_choice);
            }
        }
    }
    return 0;
}
```