**Title:**

**Comparative Analysis of Genetic Algorithms in Optimization Problems**

Under the guidance of

**Dr. Haider Banka**

(Associate Professor)

****

**Department of Computer Science and Engineering**

**INDIAN INSTITUTE OF TECHNOLOGY**

**(INDIAN SCHOOL OF MINES)**

**DHANBAD**

**By: Avinesh Pratap Singh**

**20JE0219**

**Index**

1. [**Problem Overview**](#ProblemOverview)
2. [**GA Working Principle**](#GAWorkingPrinciple)
3. [**NSGAII Working Principle**](#NSGAIIWorkingPrinciple)
4. [**Problem 5**](#Problem5)

**4.1-SGA**

**4.1.1-** [**Code**](#GACode5)

**4.1.2-** [**Input**](#gain5)

**4.1.3-** [**Output**](#gaout5)

**4.1.4-** [**Hands-On Calculations**](#gaho5)

**4.2-NSGAII**

**4.2.1-** [**Code**](#NSGACode5)

**4.2.2-** [**Input**](#NSGAinp5)

**4.2.3-** [**Output**](#NSGAout5)

**4.2.4-** [**Hands-On Calculations**](#NSGAho5)

1. [**Problem 6**](#Problem6)

**5.1-SGA**

**5.1.1-** [**Code**](#GACode6)

**5.1.2-** [**Input**](#GAinp6)

**5.1.3-** [**Output**](#GAhout6)

**5.1.4-** [**Hands-On Calculations**](#GAho6)

**5.2-NSGAII**

**5.2.1-** [**Code**](#NSGACode6)

**5.2.2-** [**Input**](#NSGAinp6)

**5.2.3-** [**Output**](#NSGAout6)

**5.2.4-** [**Hands-On Calculations**](#NSGAho6)

**SUBJECT - SOFT COMPUTING SUBJECT CODE - CSO505**

**NAME - Avinesh Pratap Singh**

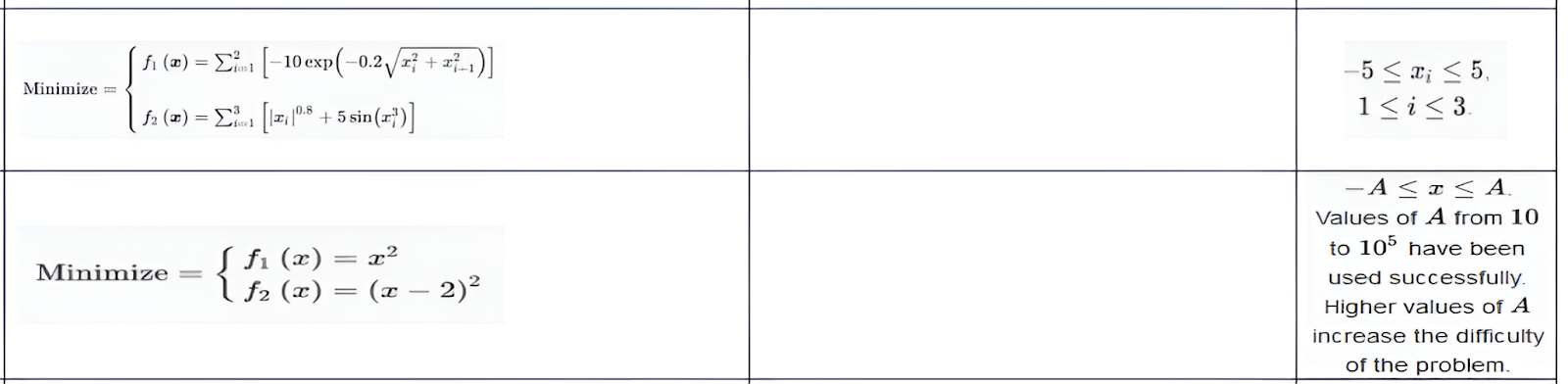
**ADMISSION No. - 20JE0219**

**Assignment Problem:**

Write programs in C/C++ only. One using simple GA and one using NSGA II. Each one should run on two benchmark functions as assigned in each group. The report to be submitted with the following format. A suitable Title of report with Name & Admission No. on it. Problem statements, Working Principle of the algorithm in your own English, codes with input and outputs, hands-on calculation to illustrate the whole process/simulation for at least two iterations.

**Problems Description:**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Constraints** | **Search Domain** |

****

**1. Introduction**

In this report, we explore the optimization capabilities of Genetic Algorithms (GA) and Non-dominated Sorting Genetic Algorithm II (NSGA-II) on benchmark functions. The primary objective is to compare the performance of these algorithms and analyse their efficiency in solving optimization problems.

**2. Working principle of Genetic Algorithms (GA):**

**2.1. Initialization:**

   - Initialize a population of candidate solutions randomly or using specific strategies.

   - Each candidate solution represents a potential solution to the optimization problem.

**2.2. Fitness Evaluation:**

   - Evaluate the fitness of each candidate solution in the population.

   - The fitness function quantifies how suitable each solution is based on problem-specific

criteria.

   - Higher fitness values indicate better solutions.

**2.3. Selection:**

   - Use selection mechanisms such as roulette wheel selection, tournament selection, or rank-

based selection to choose individuals for reproduction.

   - Individuals with higher fitness have a higher chance of being selected, mimicking the concept

of survival of the fittest.

**2.4. Crossover (Recombination):**

   - Perform crossover (also known as recombination) to create new candidate solutions.

   - Common crossover techniques include single-point crossover, multi-point crossover, and

uniform crossover.

   - Crossover combines genetic information from two parent solutions to produce offspring.

**2.5. Mutation:**

   - Apply mutation to introduce small random changes in the offspring's genetic information.

   - Mutation helps explore new areas of the solution space and prevents premature

convergence to suboptimal solutions.

**2.6. Replacement:**

   - Replace individuals in the current population with the offspring generated through crossover

and mutation.

   - Some strategies involve elitism, where the best-performing individuals from the current

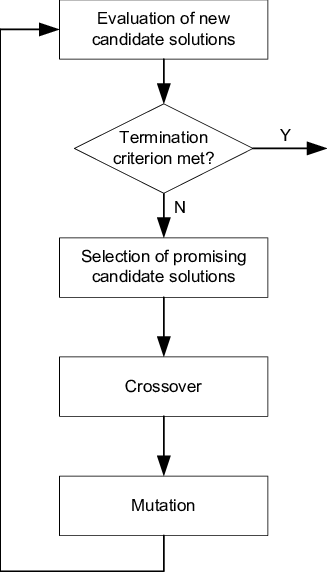
population are preserved.

**2.7. Termination:**

- Determine termination conditions such as reaching a maximum number of generations,

achieving a satisfactory fitness level, or reaching a computational limit.

   - Terminate the algorithm when the termination conditions are met.



**3. Working principle of Non-dominated Sorting Genetic Algorithm II (NSGA-II):**

**3.1. Initialization:**

   - Initialize a population of candidate solutions, similar to GA.

**3.2. Fitness Evaluation:**

   - Evaluate the fitness of each candidate solution using multiple objective functions (for multi-

objective optimization).

   - NSGA-II deals with optimizing solutions that are non-dominated, i.e., no other solution in the

population is better in all objectives and at least one objective is better.

**3.3. Non-dominated Sorting:**

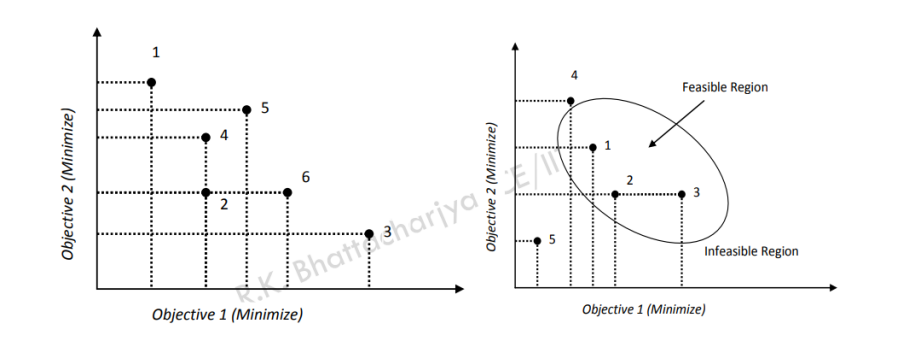
   - Perform non-dominated sorting to categorize solutions into different fronts based on their

dominance relationships.

   - A solution is non-dominated if there is no other solution in the population that is better in all

objectives.

   - Solutions in the first front are non-dominated and represent the Pareto front.



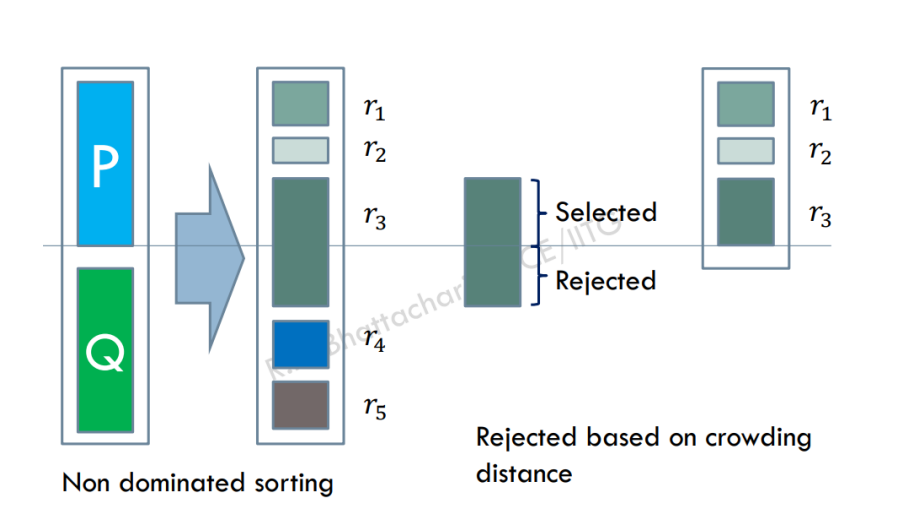
**3.4. Crowding Distance Calculation:**

   - Calculate the crowding distance for solutions in each front.

   - Crowding distance measures the density of solutions in the objective space.

   - Solutions with higher crowding distance are preferred to maintain diversity on the Pareto

front.



**3.5. Selection:**

   - Use a combination of non-dominated sorting and crowding distance to select individuals for

reproduction.

   - Solutions in less crowded regions of the Pareto front have a higher chance of being selected.

**3.6. Crossover and Mutation:**

- Apply crossover and mutation operators to create new offspring solutions.

   - Maintain diversity and balance between exploration and exploitation.

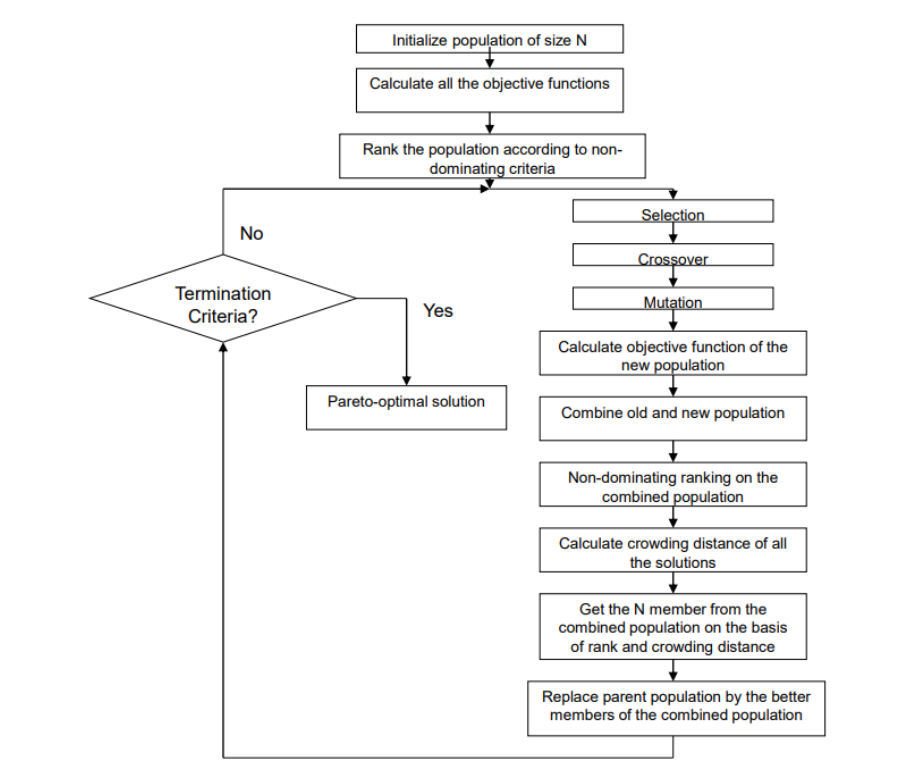
**3.7. Replacement:**

   - Replace individuals in the current population with offspring based on selection criteria.

   - Preserve non-dominated solutions and maintain diversity.

**3.8. Termination:**

   - Terminate the algorithm based on termination conditions similar to GA.



**4. Code:**

The code for GA and NSGAII is single file and structured format project (different header have

different work to do).

You can find the GitHub repository for 2nd one here:  
 <https://github.com/pratapavinesh/ga-and-nsgaii>

For visualising graphs in both type of the project please download the gnuplot from here:

<http://www.gnuplot.info/download.html>

I have also given the separate code but you can change the problem as well in same code.

for changing the problem, please update the main function with corresponding problem.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem 5:**    C++ code for the SGA:    #ifndef MATH\_AUX\_\_  #define MATH\_AUX\_\_  #include <cstdlib>  #include <vector>  namespace MathAux  {  const double PI = 3.1415926;  const double EPS = 1.0e-14; // follow nsga-ii source code  inline double square(double n) { return n \* n; }  inline double random(double lb, double ub) { return lb + (static\_cast<double>(std::rand()) / RAND\_MAX) \* (ub - lb); }  // ASF(): achievement scalarization function  double ASF(const std::vector<double>& objs, const std::vector<double>& weight);  }  #endif  #ifndef BASE\_PROBLEM\_\_  #define BASE\_PROBLEM\_\_  #include <string>  #include <vector>  // ----------------------------------------------------------------------  // BProblem: the base class of problems (e.g. ZDT and DTLZ)  // ----------------------------------------------------------------------  class CIndividual;  class BProblem  {  public:  explicit BProblem(const std::string& name) :name\_(name) {}  virtual ~BProblem() {}  virtual std::size\_t num\_variables() const = 0;  virtual std::size\_t num\_objectives() const = 0;  virtual bool Evaluate(CIndividual\* indv) const = 0;  const std::string& name() const { return name\_; }  const std::vector<double>& lower\_bounds() const { return lbs\_; }  const std::vector<double>& upper\_bounds() const { return ubs\_; }  protected:  std::string name\_;  std::vector<double> lbs\_, // lower bounds of variables  ubs\_; // upper bounds of variables  };  #endif  #ifndef ASSIGNMENT\_PROBLEM\_\_  #define ASSIGNMENT\_PROBLEM\_\_  // #include "base\_problem.h"  #include <cstddef>  #include<iostream>  // ----------------------------------------------------------------------  class CProblemAssignment : public BProblem  {  public:  CProblemAssignment(std::size\_t M, std::size\_t k, const std::string& name, double lbs, double ubs);  virtual std::size\_t num\_variables() const { return k\_; }  virtual std::size\_t num\_objectives() const { return M\_; }  virtual bool Evaluate(CIndividual\* indv) const = 0;  protected:  std::size\_t M\_; // number of objectives  std::size\_t k\_; // number of variables  };  // ----------------------------------------------------------------------  // CProblemAssignment5  // ----------------------------------------------------------------------  class CProblemAssignment5 : public CProblemAssignment  {  public:  explicit CProblemAssignment5(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment5", lbs, ubs) {}  virtual bool Evaluate(CIndividual\* indv) const;  };  // ----------------------------------------------------------------------  // CProblemAssignment6  // ----------------------------------------------------------------------  class CProblemAssignment6 : public CProblemAssignment  {  public:  explicit CProblemAssignment6(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment6", lbs, ubs) {  //std::cout << "what happend !" << lbs << '\n';  }  virtual bool Evaluate(CIndividual\* indv) const;  };  #endif  #ifndef COMPARATOR\_\_  #define COMPARATOR\_\_  class CIndividual;  // ----------------------------------------------------------------------------------  // BComparator : the base class of comparison operators  // ----------------------------------------------------------------------------------  class BComparator  {  public:  virtual ~BComparator() {}  virtual bool operator()(const CIndividual& l, const CIndividual& r) const = 0;  };  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  class CParetoDominate : public BComparator  {  public:  virtual bool operator()(const CIndividual& l, const CIndividual& r) const;  };  extern CParetoDominate ParetoDominate;  #endif  #ifndef CROSSOVER\_\_  #define CROSSOVER\_\_  // ----------------------------------------------------------------------------------  // CSimulatedBinaryCrossover : simulated binary crossover (SBX)  // ----------------------------------------------------------------------------------  class CIndividual;  class CSimulatedBinaryCrossover  {  public:  explicit CSimulatedBinaryCrossover(double cr = 1.0, double eta = 30) :cr\_(cr), eta\_(eta) {} // NSGA-III (t-EC 2014) setting  void SetCrossoverRate(double cr) { cr\_ = cr; }  double CrossoverRate() const { return cr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2, double cr, double eta) const;  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2) const  {  return operator()(c1, c2, p1, p2, cr\_, eta\_);  }  private:  double get\_betaq(double rand, double alpha, double eta) const;  double cr\_; // crossover rate  double eta\_; // distribution index  };  #endif  #include <string>  #include <stdio.h>  #include <string>  #include <cstddef>  // ----------------------------------------------------------------------  // Gnuplot  //  // This is just a very simple interface to call gnuplot in the program.  // Now it seems to work only under windows + visual studio.  // ----------------------------------------------------------------------  class Gnuplot  {  public:  Gnuplot();  ~Gnuplot();  // prohibit copying (VS2012 does not support 'delete')  Gnuplot(const Gnuplot&);  Gnuplot& operator=(const Gnuplot&);  // send any command to gnuplot  void operator ()(const std::string& command);  void reset() { operator()("reset"); }  void replot() { operator()("replot"); }  void set\_title(const std::string& title);  void plot(const std::string& fname, std::size\_t x, std::size\_t y);  void splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z);  protected:  FILE\* gnuplotpipe;  };  #ifndef INDIVIDUAL\_\_  #define INDIVIDUAL\_\_  #include <vector>  #include <ostream>  // ----------------------------------------------------------------------  // CIndividual  // ----------------------------------------------------------------------  class BProblem;  class CIndividual  {  public:  typedef double TGene;  typedef std::vector<TGene> TDecVec;  typedef std::vector<double> TObjVec;  explicit CIndividual(std::size\_t num\_vars = 0, std::size\_t num\_objs = 0);  TDecVec& vars() { return variables\_; }  const TDecVec& vars() const { return variables\_; }  TObjVec& objs() { return objectives\_; }  const TObjVec& objs() const { return objectives\_; }  TObjVec& conv\_objs() { return converted\_objectives\_; }  const TObjVec& conv\_objs() const { return converted\_objectives\_; }  // if a target problem is set, memory will be allocated accordingly in the constructor  static void SetTargetProblem(const BProblem& p) { target\_problem\_ = &p; }  static const BProblem& TargetProblem();  private:  TDecVec variables\_;  TObjVec objectives\_;  TObjVec converted\_objectives\_;  static const BProblem\* target\_problem\_;  };  std::ostream& operator << (std::ostream& os, const CIndividual& indv);  #endif  #ifndef INITIALIZATION\_\_  #define INITIALIZATION\_\_  // ----------------------------------------------------------------------  // CRandomInitialization  // ----------------------------------------------------------------------  class CIndividual;  class CPopulation;  class BProblem;  class CRandomInitialization  {  public:  void operator()(CPopulation\* pop, const BProblem& prob) const;  void operator()(CIndividual\* indv, const BProblem& prob) const;  };  extern CRandomInitialization RandomInitialization;  #endif  #ifndef POPULATION\_\_  #define POPULATION\_\_  // #include "individual.h"  #include <vector>  class CPopulation  {  public:  explicit CPopulation(std::size\_t s = 0) :individuals\_(s) {}  CIndividual& operator[](std::size\_t i) { return individuals\_[i]; }  const CIndividual& operator[](std::size\_t i) const { return individuals\_[i]; }  std::size\_t size() const { return individuals\_.size(); }  void resize(size\_t t) { individuals\_.resize(t); }  void push\_back(const CIndividual& indv) { individuals\_.push\_back(indv); }  void clear() { individuals\_.clear(); }  private:  std::vector<CIndividual> individuals\_;  };  #endif  #ifndef ENVIRONMENTAL\_SELECTION\_\_  #define ENVIRONMENTAL\_SELECTION\_\_  #include <vector>  // ----------------------------------------------------------------------  // The environmental selection mechanism is the key innovation of  // the GA algorithm.  //  // Check Algorithm I in the original paper of NSGA-III.  // ----------------------------------------------------------------------  class CPopulation;  class CReferencePoint;  void SurvivorSelection(CPopulation\* pnext, // population in the next generation  CPopulation\* pcur, // population in the current generation  size\_t PopSize);  #endif  #ifndef LOG\_\_  #define LOG\_\_  #include <string>  #include <fstream> // Include <fstream> for std::ios\_base and std::ios\_base::openmode  class CPopulation;  class Gnuplot;  // Save a population into the designated file.  bool SaveToFile(const std::string& fname, const CPopulation& pop, std::ios\_base::openmode mode = std::ios\_base::app);  // Show a population by calling gnuplot.  bool ShowPopulation(Gnuplot& gplot, const CPopulation&, const std::string& legend = "pop");  #endif  #ifndef MUTATION\_\_  #define MUTATION\_\_  // ----------------------------------------------------------------------------------  // CPolynomialMutation : polynomial mutation  // ----------------------------------------------------------------------------------  class CIndividual;  class CPolynomialMutation  {  public:  explicit CPolynomialMutation(double mr = 0.0, double eta = 20) :mr\_(mr), eta\_(eta) {}  void SetMutationRate(double mr) { mr\_ = mr; }  double MutationRate() const { return mr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c, double mr, double eta) const;  bool operator()(CIndividual\* c) const  {  return operator()(c, mr\_, eta\_);  }  private:  double mr\_, // mutation rate  eta\_; // distribution index  };  #endif  #ifndef GA\_\_  #define GA\_\_  #include <cstddef>  #include <string>  // ----------------------------------------------------------------------------------  class BProblem;  class CPopulation;  class CGA  {  public:  explicit CGA(const std::string& inifile\_name = "");  void Solve(CPopulation\* solutions, const BProblem& prob);  const std::string& name() const { return name\_; }  private:  std::string name\_;  std::size\_t obj\_division\_p\_;  std::size\_t gen\_num\_;  double pc\_, // crossover rate  pm\_, // mutation rate  eta\_c\_, // eta in SBX  eta\_m\_; // eta in Polynomial Mutation  };  #endif  // #include "ga.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "initialization.h"  // #include "crossover.h"  // #include "mutation.h"  // #include "survivor\_selection.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include "windows.h" // for Sleep()  #include <vector>  #include <fstream>  #include<iostream>  using namespace std;  CGA::CGA(const string& inifile\_name) :  name\_("GA"),  obj\_division\_p\_(12),  gen\_num\_(1000),  pc\_(1.0), // default setting in GA  eta\_c\_(30), // default setting  eta\_m\_(20) // default setting  {  if (inifile\_name == "") return;  ifstream inifile(inifile\_name);  if (!inifile) return;  string dummy;  inifile >> dummy >> dummy >> name\_;  inifile >> dummy >> dummy >> obj\_division\_p\_;  inifile >> dummy >> dummy >> gen\_num\_;  inifile >> dummy >> dummy >> pc\_;  inifile >> dummy >> dummy >> eta\_c\_;  inifile >> dummy >> dummy >> eta\_m\_;  }  // ----------------------------------------------------------------------  void CGA::Solve(CPopulation\* solutions, const BProblem& problem)  {  CIndividual::SetTargetProblem(problem);  size\_t PopSize = 50;  while (PopSize % 4) PopSize += 1;  //CPopulation pop[2] = { CPopulation(PopSize) };  std::vector<CPopulation> pop(2, CPopulation(PopSize));  CSimulatedBinaryCrossover SBX(pc\_, eta\_c\_);  CPolynomialMutation PolyMut(1.0 / problem.num\_variables(), eta\_m\_);  Gnuplot gplot;  int cur = 0, next = 1;  //std::cout << problem.lower\_bounds()[0] << '\n';  RandomInitialization(&pop[cur], problem);  for (size\_t i = 0; i < PopSize; i += 1)  {  problem.Evaluate(&pop[cur][i]);  }  for (size\_t t = 0; t < gen\_num\_; t += 1)  {  pop[cur].resize(PopSize \* 2);  for (size\_t i = 0; i < PopSize; i += 2)  {  int father = rand() % PopSize,  mother = rand() % PopSize;  SBX(&pop[cur][PopSize + i], &pop[cur][PopSize + i + 1], pop[cur][father], pop[cur][mother]);  PolyMut(&pop[cur][PopSize + i]);  PolyMut(&pop[cur][PopSize + i + 1]);  problem.Evaluate(&pop[cur][PopSize + i]);  problem.Evaluate(&pop[cur][PopSize + i + 1]);  }  SurvivorSelection(&pop[next], &pop[cur], PopSize);  //ShowPopulation(gplot, pop[next], "pop"); Sleep(10);  std::swap(cur, next);  }  \*solutions = pop[cur];  }  // #include "comparator.h"  // #include "individual.h"  // ----------------------------------------------------------------------------------  CParetoDominate ParetoDominate;  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  bool CParetoDominate::operator()(const CIndividual& l, const CIndividual& r) const  {  bool better = false;  for (size\_t f = 0; f < l.objs().size(); f += 1)  {  if (l.objs()[f] > r.objs()[f])  return false;  else if (l.objs()[f] < r.objs()[f])  better = true;  }  return better;  }// CParetoDominate::operator()  // ----------------------------------------------------------------------------------  #include <vector>  // #include "math\_aux.h"  #include<limits>  using namespace std;  namespace MathAux  {  // ----------------------------------------------------------------------  // ASF: Achivement Scalarization Function  // ----------------------------------------------------------------------  double ASF(const vector<double> &objs, const vector<double> &weight)  {  double max\_ratio = -numeric\_limits<double>::max();  for (size\_t f=0; f<objs.size(); f+=1)  {  double w = weight[f]?weight[f]:0.00001;  max\_ratio = std::max(max\_ratio, objs[f]/w);  }  return max\_ratio;  }  // ---------------------------------------------------------------------  }// namespace MathAux  // #include "individual.h"  // #include "base\_problem.h"  using std::size\_t;  const BProblem\* CIndividual::target\_problem\_ = 0;  // ----------------------------------------------------------------------  CIndividual::CIndividual(std::size\_t num\_vars, std::size\_t num\_objs) :  variables\_(num\_vars),  objectives\_(num\_objs),  converted\_objectives\_(num\_objs)  {  if (target\_problem\_ != 0)  {  variables\_.resize(target\_problem\_->num\_variables());  objectives\_.resize(target\_problem\_->num\_objectives());  converted\_objectives\_.resize(target\_problem\_->num\_objectives());  }  }  // ----------------------------------------------------------------------  const BProblem& CIndividual::TargetProblem() { return \*target\_problem\_; }  // ----------------------------------------------------------------------  std::ostream& operator << (std::ostream& os, const CIndividual& indv)  {  for (size\_t i = 0; i < indv.vars().size(); i += 1)  {  os << indv.vars()[i] << ' ';  }  os << " => ";  for (size\_t f = 0; f < indv.objs().size(); f += 1)  {  os << indv.objs()[f] << ' ';  }  return os;  }  // ----------------------------------------------------------------------  // #include "initialization.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "math\_aux.h"  #include <cstddef>  #include<iostream>  using std::size\_t;  CRandomInitialization RandomInitialization;  void CRandomInitialization::operator()(CIndividual\* indv, const BProblem& prob) const  {  CIndividual::TDecVec& x = indv->vars();  x.resize(prob.num\_variables());  for (size\_t i = 0; i < x.size(); i += 1)  {  x[i] = MathAux::random(prob.lower\_bounds()[i], prob.upper\_bounds()[i]);  }    }  // ----------------------------------------------------------------------  void CRandomInitialization::operator()(CPopulation\* pop, const BProblem& prob) const  {  for (size\_t i = 0; i < pop->size(); i += 1)  {  (\*this)(&(\*pop)[i], prob);  }  }  // #include "crossover.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cmath>  #include <algorithm>  #include <cstddef>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function realcross() in crossover.c  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ----------------------------------------------------------------------  double CSimulatedBinaryCrossover::get\_betaq(double rand, double alpha, double eta) const  {  double betaq = 0.0;  if (rand <= (1.0 / alpha))  {  betaq = std::pow((rand \* alpha), (1.0 / (eta + 1.0)));  }  else  {  betaq = std::pow((1.0 / (2.0 - rand \* alpha)), (1.0 / (eta + 1.0)));  }  return betaq;  }  // ----------------------------------------------------------------------  bool CSimulatedBinaryCrossover::operator()(CIndividual\* child1,  CIndividual\* child2,  const CIndividual& parent1,  const CIndividual& parent2,  double cr,  double eta) const  {  \*child1 = parent1;  \*child2 = parent2;  if (MathAux::random(0.0, 1.0) > cr) return false; // not crossovered  CIndividual::TDecVec& c1 = child1->vars(), & c2 = child2->vars();  const CIndividual::TDecVec& p1 = parent1.vars(), & p2 = parent2.vars();  for (size\_t i = 0; i < c1.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) > 0.5) continue; // these two variables are not crossovered  if (std::fabs(p1[i] - p2[i]) <= MathAux::EPS) continue; // two values are the same  double y1 = std::min(p1[i], p2[i]),  y2 = std::max(p1[i], p2[i]);  double lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double rand = MathAux::random(0.0, 1.0);  // child 1  double beta = 1.0 + (2.0 \* (y1 - lb) / (y2 - y1)),  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  double betaq = get\_betaq(rand, alpha, eta);  c1[i] = 0.5 \* ((y1 + y2) - betaq \* (y2 - y1));  // child 2  beta = 1.0 + (2.0 \* (ub - y2) / (y2 - y1));  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  betaq = get\_betaq(rand, alpha, eta);  c2[i] = 0.5 \* ((y1 + y2) + betaq \* (y2 - y1));  // boundary checking  c1[i] = std::min(ub, std::max(lb, c1[i]));  c2[i] = std::min(ub, std::max(lb, c2[i]));  if (MathAux::random(0.0, 1.0) <= 0.5)  {  std::swap(c1[i], c2[i]);  }  }  return true;  }// CSimulatedBinaryCrossover  // #include "mutation.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cstddef>  #include <algorithm>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function real\_mutate\_ind() in mutation.c in  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ---------------------------------------------------------------------  bool CPolynomialMutation::operator()(CIndividual\* indv, double mr, double eta) const  {  //int j;  //double rnd, delta1, delta2, mut\_pow, deltaq;  //double y, yl, yu, val, xy;  bool mutated = false;  CIndividual::TDecVec& x = indv->vars();  for (size\_t i = 0; i < x.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) <= mr)  {  mutated = true;  double y = x[i],  lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double delta1 = (y - lb) / (ub - lb),  delta2 = (ub - y) / (ub - lb);  double mut\_pow = 1.0 / (eta + 1.0);  double rnd = MathAux::random(0.0, 1.0), deltaq = 0.0;  if (rnd <= 0.5)  {  double xy = 1.0 - delta1;  double val = 2.0 \* rnd + (1.0 - 2.0 \* rnd) \* (pow(xy, (eta + 1.0)));  deltaq = pow(val, mut\_pow) - 1.0;  }  else  {  double xy = 1.0 - delta2;  double val = 2.0 \* (1.0 - rnd) + 2.0 \* (rnd - 0.5) \* (pow(xy, (eta + 1.0)));  deltaq = 1.0 - (pow(val, mut\_pow));  }  y = y + deltaq \* (ub - lb);  y = std::min(ub, std::max(lb, y));  x[i] = y;  }  }  return mutated;  }// CPolynomialMutation  // #include "survivor\_selection.h"  // #include "population.h"  // #include "math\_aux.h"  #include <limits>  #include <algorithm>  using namespace std;  // ----------------------------------------------------------------------  // SurvivorSelection():  // ----------------------------------------------------------------------  void SurvivorSelection(CPopulation \*pnext, CPopulation \*pcur, size\_t PopSize)  {  CPopulation &cur = \*pcur, &next = \*pnext;  next.clear();  std::vector<size\_t> index;  for (int i = 0; i < cur.size(); i++)index.push\_back(i);  std::sort(index.begin(), index.end(), [&](size\_t a, size\_t b) {  return cur[a].objs()[0]+ cur[a].objs()[1] < cur[b].objs()[0]+ cur[b].objs()[1];  });    for (size\_t t = 0; t < PopSize ; t += 1)  {  next.push\_back(cur[index[t]]);  }  }  // ----------------------------------------------------------------------  // #include "gnuplot\_interface.h"  #include <iostream>  #include <sstream>  using namespace std;  // ---------------------------------------------------------  // Ref:  // http://user.frdm.info/ckhung/b/ma/gnuplot.php  // ---------------------------------------------------------  Gnuplot::Gnuplot()  {  // with -persist option you will see the windows as your program ends  //gnuplotpipe=\_popen("gnuplot -persist","w");  //without that option you will not see the window  // because I choose the terminal to output files so I don't want to see the window  gnuplotpipe = \_popen("gnuplot", "w");  if (!gnuplotpipe)  {  cerr << ("Gnuplot not found !");  }  }  // ---------------------------------------------------------  Gnuplot::~Gnuplot()  {  fprintf(gnuplotpipe, "exit\n");  \_pclose(gnuplotpipe);  }  // ---------------------------------------------------------  void Gnuplot::operator()(const string& command)  {  fprintf(gnuplotpipe, "%s\n", command.c\_str());  fflush(gnuplotpipe);  // flush is necessary, nothing gets plotted else  }  // ---------------------------------------------------------  void Gnuplot::set\_title(const std::string& title)  {  ostringstream ss;  ss << "set title \"" << title << "\"";  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::plot(const std::string& fname, std::size\_t x, std::size\_t y)  {  ostringstream ss;  ss << "plot \"" << fname << "\" using " << x << ":" << y;  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z)  {  ostringstream ss;  ss << "splot \"" << fname << "\" using " << x << ":" << y << ":" << z;  operator()(ss.str());  }  // ---------------------------------------------------------  // #include "assignment\_problem.h"  // #include "individual.h"  // #include "math\_aux.h"  #include <cmath>  #include <vector>  #include<iostream>  using std::size\_t;  using std::cos;  // ----------------------------------------------------------------------  // CProblemAssignment  // ----------------------------------------------------------------------  CProblemAssignment::CProblemAssignment(size\_t M, size\_t k, const std::string& name, double lbs, double ubs) :  BProblem(name),  M\_(M),  k\_(k)  {  lbs\_.resize( k\_, lbs); // lower bound  ubs\_.resize( k\_, ubs); // upper bound  }  bool CProblemAssignment5::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  for (size\_t i = 1; i < k\_; ++i) {  f[0] += -10 \* exp(-0.2 \* sqrt(MathAux::square(x[i - 1]) + MathAux::square(x[i])));  }  for (size\_t i = 0; i < k\_; ++i) {  f[1] += pow(abs(x[i]), 0.8) + 5 \* sin(pow(x[i], 3));  }    return true;  }  bool CProblemAssignment6::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  //std::cout << (x.size() == k\_)<<'\n';  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  f[0] = MathAux::square(x[0]);  f[1] = MathAux::square(x[0] - 2);  return true;  }  // #include "log.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  #include <fstream>  #include<iostream>  using namespace std;  #define OUTPUT\_DECISION\_VECTOR  bool SaveToFile(const std::string& fname, const CPopulation& pop, ios\_base::openmode mode)  {  ofstream ofile(fname.c\_str(), mode);  if (!ofile) return false;    for (size\_t i = 0; i < pop.size(); i += 1)  {  #ifdef OUTPUT\_DECISION\_VECTOR    for (size\_t j = 0; j < pop[i].vars().size(); j += 1)  {  ofile << pop[i].vars()[j] << ' ';  }  #endif  //std::cout << pop[i].objs().size() << endl;  for (size\_t f = 0; f < pop[i].objs().size(); f += 1)  {  ofile << pop[i].objs()[f] << ' ';  }    ofile << pop[i].objs()[0] + pop[i].objs()[1] << ' ';    ofile << endl;  }  ofile << endl;  return true;  }  // ----------------------------------------------------------------------  bool ShowPopulation(Gnuplot& gplot, const CPopulation& pop, const std::string& legend)  {  if (!SaveToFile(legend, pop, ios\_base::out)) return false;  size\_t n = 0;  #ifdef OUTPUT\_DECISION\_VECTOR  n = pop[0].vars().size();  #endif    if (pop[0].objs().size() == 2)  {  gplot.plot(legend, n + 1, n + 2);  return true;  }  else if (pop[0].objs().size() == 3)  {  gplot.splot(legend, n + 1, n + 2, n + 3);  return true;  }  else  return false;  }  // #include "assignment\_problem.h"  // #include "ga.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include <ctime>  #include <cstdlib>  #include <iostream>  // #include "individual.h"  // #include "math\_aux.h"  using namespace std;  int main()  {  CGA ga("ga");  CPopulation solutions;  Gnuplot gplot;  const size\_t NumRuns = 10;  BProblem\* problem5 = new CProblemAssignment5(2, 3,-5,5);  BProblem\* problem6 = new CProblemAssignment6(2, 1,-10,10);  BProblem\* problem = problem5;  //std::cout << problem->upper\_bounds()[0] << '\n';  for (size\_t r = 0; r < NumRuns; r += 1)  {  srand(r); cout << "Run Number: " << r << endl;  ga.Solve(&solutions, \*problem);  SaveToFile(ga.name() + "-" + problem->name() + ".txt", solutions);  ShowPopulation(gplot, solutions, "pop"); system("pause");  }  delete problem;  return 0;  }  Input: Terminal Input is not required Everything is Automated. If you want to change the parameters  please update the main function and ga function.  You can also add ga.ini file and write parameter values.  Output:  Values of F1 and F2 after max iteration          The Values after the max iteration (x1, x2, x3, f1, f2, f1+f2)    Hands On calculations:  Number of populations =4  Tournament parent selection  SBX crossover  Polynomial Mutation  Survivor selection Fitness propionate  Generation Number =1  Parent selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | g(x) | | 1 | -4.9884 | -2.64428 | 1.48152 | -8.68714 | 13.3724 | 4.68527 | | 2 | -4.25626 | -2.29759 | -1.39973 | -9.63955 | 1.66223 | -7.97732 | | 3 | -2.44713 | 4.8529 | -1.81082 | -6.92117 | 9.18446 | 2.26329 | | 4 | 4.34233 | -3.21375 | 3.5757 | -7.21749 | 9.56758 | 2.35009 |     Crossover     |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Father | Mother |  |  | c1 |  |  |  | c2 |  | |  |  |  | x1 | x2 | x3 |  | x1 | x2 | x3 | | 3 | 2 | 1 | -4.23878 | -2.28495 | -1.81082 | 2 | -2.46461 | 4.82254 | -1.39973 | | 3 | 3 | 3 | -2.44713 | 4.8529 | -1.81082 | 4 | -2.44713 | 4.8529 | -1.81082 |   Mutation   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  | Mutated Value |  |  |  |  | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | g(x) | | 1 | -4.23878 | -2.28495 | -1.81082 | -16.32 17 | 17.1229 | 0.802917 | | 2 | -1.80237 | 4.57723 | -1.39973 | -17.2175 | 13.0664 | -4.15109 | | 3 | -2.44713 | 4.8529 | -1.81082 | -13.8423 | 18.3689 | 4.52657 | | 4 | -1.70358 | 4.95464 | -1.81082 | -13.9098 | 26.3758 | 12.466 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | g(x) |  | Next Generation | |  |  | Old Population |  |  |  |  | | 1 | -4.9884 | -2.64428 | 1.48152 | 4.68527 | 1 | 2 | | 2 | -4.25626 | -2.29759 | -1.39973 | -7.97732 | 2 | 6 | | 3 | -2.44713 | 4.8529 | -1.81082 | 2.26329 | 3 | 5 | | 4 | 4.34233 | -3.21375 | 3.5757 | 2.35009 | 4 | 3 | |  |  | New Population |  |  |  |  | | 5 | -4.23878 | -2.28495 | -1.81082 | 0.802917 |  |  | | 6 | -1.80237 | 4.57723 | -1.39973 | -4.15109 |  |  | | 7 | -2.44713 | 4.8529 | -1.81082 | 4.52657 |  |  | | 8 | -1.70358 | 4.95464 | -1.81082 | 12.466 |  |  |   Generation Number =2  Parent selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | g(x) | | 1 | -4.25626 | -2.29759 | -1.39973 | -9.63955 | 1.66223 | -7.97732 | | 2 | -1.80237 | 4.57723 | -1.39973 | -17.2175 | 13.0664 | -4.15109 | | 3 | -4.23878 | -2.28495 | -1.81082 | -16.32 17 | 17.1229 | 0.802917 | | 4 | -2.44713 | 4.8529 | -1.81082 | -6.92117 | 9.18446 | 2.26329 |     Crossover     |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Father | Mother |  |  | c1 |  |  |  | c2 |  | |  |  |  | x1 | x2 | x3 |  | x1 | x2 | x3 | | 2 | 1 | 1 | -1.80237 | 4.5803 | -1.39973 | 2 | -4.25626 | -2.30231 | -1.39973 | | 1 | 3 | 3 | -4.25626 | -2.28487 | -1.39973 | 4 | -4.23878 | -2.29767 | -1.81082 |   Mutation   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  | Mutated Value |  |  |  |  | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | g(x) | | 1 | -1.80237 | 4.5803 | -1.39973 | -24.7911 | 24.3035 | -0.487555 | | 2 | -4.25626 | -2.30231 | -2.28626 | -18.6648 | 8.44429 | -10.2205 | | 3 | -4.25626 | -1.94899 | -1.39973 | -25.7147 | 24.8487 | -23.2843 | | 4 | -4.23878 | -2.28829 | -1.81082 | -25.7147 | 24.8487 | -0.866012 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | g(x) |  | Next Generation | |  |  | Old Population |  |  |  |  | | 1 | -4.25626 | -2.29759 | -1.39973 | -7.97732 | 1 | 7 | | 2 | -1.80237 | 4.57723 | -1.39973 | -4.15109 | 2 | 6 | | 3 | -4.23878 | -2.28495 | -1.81082 | 0.802917 | 3 | 1 | | 4 | -2.44713 | 4.8529 | -1.81082 | 2.26329 | 4 | 2 | |  |  | New Population |  |  |  |  | | 5 | -1.80237 | 4.5803 | -1.39973 | -0.487555 |  |  | | 6 | -4.25626 | -2.30231 | -2.28626 | -10.2205 |  |  | | 7 | -4.25626 | -1.94899 | -1.39973 | -23.2843 |  |  | | 8 | -4.23878 | -2.28829 | -1.81082 | -0.866012 |  |  |   Best Answer: x = [-4.25626 -1.94899 -1.39973]    C++ code for the NSGAII:  #ifndef MATH\_AUX\_\_  #define MATH\_AUX\_\_  #include <cstdlib>  #include <vector>  namespace MathAux  {  const double PI = 3.1415926;  const double EPS = 1.0e-14; // follow nsga-ii source code  inline double square(double n) { return n \* n; }  inline double random(double lb, double ub) { return lb + (static\_cast<double>(std::rand()) / RAND\_MAX) \* (ub - lb); }  // ASF(): achievement scalarization function  double ASF(const std::vector<double>& objs, const std::vector<double>& weight);  }  #endif  #ifndef BASE\_PROBLEM\_\_  #define BASE\_PROBLEM\_\_  #include <string>  #include <vector>  // ----------------------------------------------------------------------  // BProblem: the base class of problems (e.g. ZDT and DTLZ)  // ----------------------------------------------------------------------  class CIndividual;  class BProblem  {  public:  explicit BProblem(const std::string& name) :name\_(name) {}  virtual ~BProblem() {}  virtual std::size\_t num\_variables() const = 0;  virtual std::size\_t num\_objectives() const = 0;  virtual bool Evaluate(CIndividual\* indv) const = 0;  const std::string& name() const { return name\_; }  const std::vector<double>& lower\_bounds() const { return lbs\_; }  const std::vector<double>& upper\_bounds() const { return ubs\_; }  protected:  std::string name\_;  std::vector<double> lbs\_, // lower bounds of variables  ubs\_; // upper bounds of variables  };  #endif  #ifndef ASSIGNMENT\_PROBLEM\_\_  #define ASSIGNMENT\_PROBLEM\_\_  //#include "base\_problem.h"  #include <cstddef>  #include<iostream>  // ----------------------------------------------------------------------  class CProblemAssignment : public BProblem  {  public:  CProblemAssignment(std::size\_t M, std::size\_t k, const std::string& name, double lbs, double ubs);  virtual std::size\_t num\_variables() const { return k\_; }  virtual std::size\_t num\_objectives() const { return M\_; }  virtual bool Evaluate(CIndividual\* indv) const = 0;  protected:  std::size\_t M\_; // number of objectives  std::size\_t k\_; // number of variables  };  // ----------------------------------------------------------------------  // CProblemAssignment5  // ----------------------------------------------------------------------  class CProblemAssignment5 : public CProblemAssignment  {  public:  explicit CProblemAssignment5(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment5", lbs, ubs) {}  virtual bool Evaluate(CIndividual\* indv) const;  };  // ----------------------------------------------------------------------  // CProblemAssignment6  // ----------------------------------------------------------------------  class CProblemAssignment6 : public CProblemAssignment  {  public:  explicit CProblemAssignment6(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment6", lbs, ubs) {  //std::cout << "what happend !" << lbs << '\n';  }  virtual bool Evaluate(CIndividual\* indv) const;  };  #endif  #ifndef COMPARATOR\_\_  #define COMPARATOR\_\_  class CIndividual;  // ----------------------------------------------------------------------------------  // BComparator : the base class of comparison operators  // ----------------------------------------------------------------------------------  class BComparator  {  public:  virtual ~BComparator() {}  virtual bool operator()(const CIndividual& l, const CIndividual& r) const = 0;  };  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  class CParetoDominate : public BComparator  {  public:  virtual bool operator()(const CIndividual& l, const CIndividual& r) const;  };  extern CParetoDominate ParetoDominate;  #endif  #ifndef CROSSOVER\_\_  #define CROSSOVER\_\_  // ----------------------------------------------------------------------------------  // CSimulatedBinaryCrossover : simulated binary crossover (SBX)  // ----------------------------------------------------------------------------------  class CIndividual;  class CSimulatedBinaryCrossover  {  public:  explicit CSimulatedBinaryCrossover(double cr = 1.0, double eta = 30) :cr\_(cr), eta\_(eta) {} // NSGA-III (t-EC 2014) setting  void SetCrossoverRate(double cr) { cr\_ = cr; }  double CrossoverRate() const { return cr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2, double cr, double eta) const;  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2) const  {  return operator()(c1, c2, p1, p2, cr\_, eta\_);  }  private:  double get\_betaq(double rand, double alpha, double eta) const;  double cr\_; // crossover rate  double eta\_; // distribution index  };  #endif  #include <string>  #include <stdio.h>  #include <string>  #include <cstddef>  // ----------------------------------------------------------------------  // Gnuplot  //  // This is just a very simple interface to call gnuplot in the program.  // Now it seems to work only under windows + visual studio.  // ----------------------------------------------------------------------  class Gnuplot  {  public:  Gnuplot();  ~Gnuplot();  // prohibit copying (VS2012 does not support 'delete')  Gnuplot(const Gnuplot&);  Gnuplot& operator=(const Gnuplot&);  // send any command to gnuplot  void operator ()(const std::string& command);  void reset() { operator()("reset"); }  void replot() { operator()("replot"); }  void set\_title(const std::string& title);  void plot(const std::string& fname, std::size\_t x, std::size\_t y);  void splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z);  protected:  FILE\* gnuplotpipe;  };  #ifndef INDIVIDUAL\_\_  #define INDIVIDUAL\_\_  #include <vector>  #include <ostream>  // ----------------------------------------------------------------------  // CIndividual  // ----------------------------------------------------------------------  class BProblem;  class CIndividual  {  public:  typedef double TGene;  typedef std::vector<TGene> TDecVec;  typedef std::vector<double> TObjVec;  explicit CIndividual(std::size\_t num\_vars = 0, std::size\_t num\_objs = 0);  TDecVec& vars() { return variables\_; }  const TDecVec& vars() const { return variables\_; }  TObjVec& objs() { return objectives\_; }  const TObjVec& objs() const { return objectives\_; }  TObjVec& conv\_objs() { return converted\_objectives\_; }  const TObjVec& conv\_objs() const { return converted\_objectives\_; }  // if a target problem is set, memory will be allocated accordingly in the constructor  static void SetTargetProblem(const BProblem& p) { target\_problem\_ = &p; }  static const BProblem& TargetProblem();  private:  TDecVec variables\_;  TObjVec objectives\_;  TObjVec converted\_objectives\_;  static const BProblem\* target\_problem\_;  };  std::ostream& operator << (std::ostream& os, const CIndividual& indv);  #endif  #ifndef INITIALIZATION\_\_  #define INITIALIZATION\_\_  // ----------------------------------------------------------------------  // CRandomInitialization  // ----------------------------------------------------------------------  class CIndividual;  class CPopulation;  class BProblem;  class CRandomInitialization  {  public:  void operator()(CPopulation\* pop, const BProblem& prob) const;  void operator()(CIndividual\* indv, const BProblem& prob) const;  };  extern CRandomInitialization RandomInitialization;  #endif  #ifndef POPULATION\_\_  #define POPULATION\_\_  // #include "individual.h"  #include <vector>  class CPopulation  {  public:  explicit CPopulation(std::size\_t s = 0) :individuals\_(s) {}  CIndividual& operator[](std::size\_t i) { return individuals\_[i]; }  const CIndividual& operator[](std::size\_t i) const { return individuals\_[i]; }  std::size\_t size() const { return individuals\_.size(); }  void resize(size\_t t) { individuals\_.resize(t); }  void push\_back(const CIndividual& indv) { individuals\_.push\_back(indv); }  void clear() { individuals\_.clear(); }  private:  std::vector<CIndividual> individuals\_;  };  #endif  #ifndef ENVIRONMENTAL\_SELECTION\_\_  #define ENVIRONMENTAL\_SELECTION\_\_  #include <vector>  // ----------------------------------------------------------------------  // The environmental selection mechanism is the key innovation of  // the NSGA-III algorithm.  //  // Check Algorithm I in the original paper of NSGA-III.  // ----------------------------------------------------------------------  class CPopulation;  class CReferencePoint;  void SurvivorSelection(CPopulation \*pnext, // population in the next generation  CPopulation \*pcur, // population in the current generation  size\_t PopSize);  #endif  #ifndef NONDOMINATED\_SORT\_\_  #define NONDOMINATED\_SORT\_\_  #include <vector>  class BComparator;  class CPopulation;  class CNondominatedSort  {  public:  explicit CNondominatedSort(const BComparator &d):dominate(d) {}  // prohibit copying (VS2012 does not support 'delete')  CNondominatedSort(const CNondominatedSort &);  CNondominatedSort & operator= (const CNondominatedSort &);  typedef std::vector<std::size\_t> TFrontMembers; // a set of indices of individuals in a certain front  typedef std::vector<TFrontMembers> TFronts; // a set of fronts  std::pair< std::vector<size\_t>,TFronts> operator()(const CPopulation &pop) const;  private:  const BComparator &dominate;  };  extern CNondominatedSort NondominatedSort;  #endif  #ifndef LOG\_\_  #define LOG\_\_  #include <string>  #include <fstream> // Include <fstream> for std::ios\_base and std::ios\_base::openmode  class CPopulation;  class Gnuplot;  // Save a population into the designated file.  bool SaveToFile(const std::string& fname, const CPopulation& pop, std::ios\_base::openmode mode = std::ios\_base::app);  // Show a population by calling gnuplot.  bool ShowPopulation(Gnuplot& gplot, const CPopulation&, const std::string& legend = "pop");  #endif  #ifndef MUTATION\_\_  #define MUTATION\_\_  // ----------------------------------------------------------------------------------  // CPolynomialMutation : polynomial mutation  // ----------------------------------------------------------------------------------  class CIndividual;  class CPolynomialMutation  {  public:  explicit CPolynomialMutation(double mr = 0.0, double eta = 20) :mr\_(mr), eta\_(eta) {}  void SetMutationRate(double mr) { mr\_ = mr; }  double MutationRate() const { return mr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c, double mr, double eta) const;  bool operator()(CIndividual\* c) const  {  return operator()(c, mr\_, eta\_);  }  private:  double mr\_, // mutation rate  eta\_; // distribution index  };  #endif  #ifndef CROWDING\_DISTANCE\_H  #define CROWDING\_DISTANCE\_H  // #include "nondominated\_sort.h"  // #include "population.h"  #include <vector>  std:: vector<double> CalculateCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts);  void SortBasedOnCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts);  #endif  #ifndef NSGAII\_\_  #define NSGAII\_\_  #include <cstddef>  #include <string>  // ----------------------------------------------------------------------------------  // NSGAII  // Taken from NSGA III  // Deb and Jain, "An Evolutionary Many-Objective Optimization Algorithm Using  // Reference-point Based Non-dominated Sorting Approach, Part I: Solving Problems with  // Box Constraints," IEEE Transactions on Evolutionary Computation, to appear.  //  // http://dx.doi.org/10.1109/TEVC.2013.2281535  // ----------------------------------------------------------------------------------  class BProblem;  class CPopulation;  class CNSGAII  {  public:  explicit CNSGAII(const std::string& inifile\_name = "");  void Solve(CPopulation\* solutions, const BProblem& prob);  const std::string& name() const { return name\_; }  private:  std::string name\_;  std::size\_t obj\_division\_p\_;  std::size\_t gen\_num\_;  double pc\_, // crossover rate  pm\_, // mutation rate  eta\_c\_, // eta in SBX  eta\_m\_; // eta in Polynomial Mutation  };  #endif  // end of headers  // #include "nsgaii.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "initialization.h"  // #include "crossover.h"  // #include "mutation.h"  // #include "survivor\_selection.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include "windows.h" // for Sleep()  #include <vector>  #include <fstream>  #include<iostream>  using namespace std;  CNSGAII::CNSGAII(const string& inifile\_name) :  name\_("NSGAII"),  obj\_division\_p\_(12),  gen\_num\_(1000),  pc\_(1.0), // default setting in NSGA-II  eta\_c\_(30), // default setting  eta\_m\_(20) // default setting  {  if (inifile\_name == "") return;  ifstream inifile(inifile\_name);  if (!inifile) return;  string dummy;  inifile >> dummy >> dummy >> name\_;  inifile >> dummy >> dummy >> obj\_division\_p\_;  inifile >> dummy >> dummy >> gen\_num\_;  inifile >> dummy >> dummy >> pc\_;  inifile >> dummy >> dummy >> eta\_c\_;  inifile >> dummy >> dummy >> eta\_m\_;  }  // ----------------------------------------------------------------------  void CNSGAII::Solve(CPopulation\* solutions, const BProblem& problem)  {  CIndividual::SetTargetProblem(problem);  size\_t PopSize = 50;  while (PopSize % 4) PopSize += 1;  //CPopulation pop[2] = { CPopulation(PopSize) };  std::vector<CPopulation> pop(2, CPopulation(PopSize));  CSimulatedBinaryCrossover SBX(pc\_, eta\_c\_);  CPolynomialMutation PolyMut(1.0 / problem.num\_variables(), eta\_m\_);  Gnuplot gplot;  int cur = 0, next = 1;  //std::cout << problem.lower\_bounds()[0] << '\n';  RandomInitialization(&pop[cur], problem);  for (size\_t i = 0; i < PopSize; i += 1)  {  problem.Evaluate(&pop[cur][i]);  }  for (size\_t t = 0; t < gen\_num\_; t += 1)  {  pop[cur].resize(PopSize \* 2);  for (size\_t i = 0; i < PopSize; i += 2)  {  int father = rand() % PopSize,  mother = rand() % PopSize;  SBX(&pop[cur][PopSize + i], &pop[cur][PopSize + i + 1], pop[cur][father], pop[cur][mother]);  PolyMut(&pop[cur][PopSize + i]);  PolyMut(&pop[cur][PopSize + i + 1]);  problem.Evaluate(&pop[cur][PopSize + i]);  problem.Evaluate(&pop[cur][PopSize + i + 1]);  }  SurvivorSelection(&pop[next], &pop[cur], PopSize);  //ShowPopulation(gplot, pop[next], "pop"); Sleep(10);  std::swap(cur, next);  }  \*solutions = pop[cur];  }  // #include "comparator.h"  // #include "individual.h"  // ----------------------------------------------------------------------------------  CParetoDominate ParetoDominate;  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  bool CParetoDominate::operator()(const CIndividual& l, const CIndividual& r) const  {  bool better = false;  for (size\_t f = 0; f < l.objs().size(); f += 1)  {  if (l.objs()[f] > r.objs()[f])  return false;  else if (l.objs()[f] < r.objs()[f])  better = true;  }  return better;  }// CParetoDominate::operator()  // ----------------------------------------------------------------------------------  #include <vector>  #include <limits>  // #include "math\_aux.h"  using namespace std;  namespace MathAux  {  // ----------------------------------------------------------------------  // ASF: Achivement Scalarization Function  // ----------------------------------------------------------------------  double ASF(const vector<double> &objs, const vector<double> &weight)  {  double max\_ratio = -numeric\_limits<double>::max();  for (size\_t f=0; f<objs.size(); f+=1)  {  double w = weight[f]?weight[f]:0.00001;  max\_ratio = std::max(max\_ratio, objs[f]/w);  }  return max\_ratio;  }  // ---------------------------------------------------------------------  }// namespace MathAux  // #include "individual.h"  // #include "base\_problem.h"  using std::size\_t;  const BProblem\* CIndividual::target\_problem\_ = 0;  // ----------------------------------------------------------------------  CIndividual::CIndividual(std::size\_t num\_vars, std::size\_t num\_objs) :  variables\_(num\_vars),  objectives\_(num\_objs),  converted\_objectives\_(num\_objs)  {  if (target\_problem\_ != 0)  {  variables\_.resize(target\_problem\_->num\_variables());  objectives\_.resize(target\_problem\_->num\_objectives());  converted\_objectives\_.resize(target\_problem\_->num\_objectives());  }  }  // ----------------------------------------------------------------------  const BProblem& CIndividual::TargetProblem() { return \*target\_problem\_; }  // ----------------------------------------------------------------------  std::ostream& operator << (std::ostream& os, const CIndividual& indv)  {  for (size\_t i = 0; i < indv.vars().size(); i += 1)  {  os << indv.vars()[i] << ' ';  }  os << " => ";  for (size\_t f = 0; f < indv.objs().size(); f += 1)  {  os << indv.objs()[f] << ' ';  }  return os;  }  // ----------------------------------------------------------------------  // #include "initialization.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "math\_aux.h"  #include <cstddef>  using std::size\_t;  CRandomInitialization RandomInitialization;  void CRandomInitialization::operator()(CIndividual\* indv, const BProblem& prob) const  {  CIndividual::TDecVec& x = indv->vars();  x.resize(prob.num\_variables());  for (size\_t i = 0; i < x.size(); i += 1)  {  x[i] = MathAux::random(prob.lower\_bounds()[i], prob.upper\_bounds()[i]);  }  }  // ----------------------------------------------------------------------  void CRandomInitialization::operator()(CPopulation\* pop, const BProblem& prob) const  {  for (size\_t i = 0; i < pop->size(); i += 1)  {  (\*this)(&(\*pop)[i], prob);  }  }  // #include "crossover.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cmath>  #include <algorithm>  #include <cstddef>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function realcross() in crossover.c  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ----------------------------------------------------------------------  double CSimulatedBinaryCrossover::get\_betaq(double rand, double alpha, double eta) const  {  double betaq = 0.0;  if (rand <= (1.0 / alpha))  {  betaq = std::pow((rand \* alpha), (1.0 / (eta + 1.0)));  }  else  {  betaq = std::pow((1.0 / (2.0 - rand \* alpha)), (1.0 / (eta + 1.0)));  }  return betaq;  }  // ----------------------------------------------------------------------  bool CSimulatedBinaryCrossover::operator()(CIndividual\* child1,  CIndividual\* child2,  const CIndividual& parent1,  const CIndividual& parent2,  double cr,  double eta) const  {  \*child1 = parent1;  \*child2 = parent2;  if (MathAux::random(0.0, 1.0) > cr) return false; // not crossovered  CIndividual::TDecVec& c1 = child1->vars(), & c2 = child2->vars();  const CIndividual::TDecVec& p1 = parent1.vars(), & p2 = parent2.vars();  for (size\_t i = 0; i < c1.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) > 0.5) continue; // these two variables are not crossovered  if (std::fabs(p1[i] - p2[i]) <= MathAux::EPS) continue; // two values are the same  double y1 = std::min(p1[i], p2[i]),  y2 = std::max(p1[i], p2[i]);  double lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double rand = MathAux::random(0.0, 1.0);  // child 1  double beta = 1.0 + (2.0 \* (y1 - lb) / (y2 - y1)),  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  double betaq = get\_betaq(rand, alpha, eta);  c1[i] = 0.5 \* ((y1 + y2) - betaq \* (y2 - y1));  // child 2  beta = 1.0 + (2.0 \* (ub - y2) / (y2 - y1));  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  betaq = get\_betaq(rand, alpha, eta);  c2[i] = 0.5 \* ((y1 + y2) + betaq \* (y2 - y1));  // boundary checking  c1[i] = std::min(ub, std::max(lb, c1[i]));  c2[i] = std::min(ub, std::max(lb, c2[i]));  if (MathAux::random(0.0, 1.0) <= 0.5)  {  std::swap(c1[i], c2[i]);  }  }  return true;  }// CSimulatedBinaryCrossover  // #include "mutation.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cstddef>  #include <algorithm>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function real\_mutate\_ind() in mutation.c in  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ---------------------------------------------------------------------  bool CPolynomialMutation::operator()(CIndividual\* indv, double mr, double eta) const  {  //int j;  //double rnd, delta1, delta2, mut\_pow, deltaq;  //double y, yl, yu, val, xy;  bool mutated = false;  CIndividual::TDecVec& x = indv->vars();  for (size\_t i = 0; i < x.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) <= mr)  {  mutated = true;  double y = x[i],  lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double delta1 = (y - lb) / (ub - lb),  delta2 = (ub - y) / (ub - lb);  double mut\_pow = 1.0 / (eta + 1.0);  double rnd = MathAux::random(0.0, 1.0), deltaq = 0.0;  if (rnd <= 0.5)  {  double xy = 1.0 - delta1;  double val = 2.0 \* rnd + (1.0 - 2.0 \* rnd) \* (std::pow(xy, (eta + 1.0)));  deltaq = std::pow(val, mut\_pow) - 1.0;  }  else  {  double xy = 1.0 - delta2;  double val = 2.0 \* (1.0 - rnd) + 2.0 \* (rnd - 0.5) \* (std::pow(xy, (eta + 1.0)));  deltaq = 1.0 - (std::pow(val, mut\_pow));  }  y = y + deltaq \* (ub - lb);  y = std::min(ub, std::max(lb, y));  x[i] = y;  }  }  return mutated;  }// CPolynomialMutation  // #include "comparator.h"  // #include "nondominated\_sort.h"  // #include "population.h"  using namespace std;  CNondominatedSort NondominatedSort(ParetoDominate);  // ----------------------------------------------------------------------  std::pair< std::vector<size\_t>,std::vector< CNondominatedSort::TFrontMembers >> CNondominatedSort::operator()(const CPopulation &pop) const  {  CNondominatedSort::TFronts fronts;  size\_t num\_assigned\_individuals = 0;  size\_t rank = 1;  vector<size\_t> indv\_ranks(pop.size(), 0);  while (num\_assigned\_individuals < pop.size())  {  CNondominatedSort::TFrontMembers cur\_front;  for (size\_t i=0; i<pop.size(); i+=1)  {  if (indv\_ranks[i] > 0) continue; // already assigned a rank  bool be\_dominated = false;  for (size\_t j=0; j<cur\_front.size(); j+=1)  {  if ( dominate(pop[ cur\_front[j] ], pop[i]) ) // i is dominated  {  be\_dominated = true;  break;  }  else if ( dominate(pop[i], pop[ cur\_front[j] ]) ) // i dominates a member in the current front  {  cur\_front.erase(cur\_front.begin()+j);  j -= 1;  }  }  if (!be\_dominated)  {  cur\_front.push\_back(i);  }  }  for (size\_t i=0; i<cur\_front.size(); i+=1)  {  indv\_ranks[ cur\_front[i] ] = rank;  }  fronts.push\_back(cur\_front);  num\_assigned\_individuals += cur\_front.size();    rank += 1;  }  return { indv\_ranks,fronts };  }// CNondominatedSort::operator()  // ----------------------------------------------------------------------  // #include "crowding\_distance.h"  #include <iostream>  #include <limits>  #include <algorithm>  std::vector<double> CalculateCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts)  {  const size\_t num\_objs = cur[0].objs().size();  const size\_t num\_individuals = cur.size();  const size\_t num\_fronts = fronts.size();  std::vector<double> crowding\_distance(num\_individuals, 0.0);  for (size\_t f = 0; f < num\_objs; ++f)  {  for (size\_t i = 0; i < num\_fronts; ++i)  {  const size\_t front\_size = fronts[i].size();  if (front\_size <= 2)  {  // Skip fronts with 2 or fewer individuals as their crowding distance will be infinity  crowding\_distance[fronts[i][0]]=std::numeric\_limits<double>::infinity();  if(front\_size==2)crowding\_distance[fronts[i][1]]=std::numeric\_limits<double>::infinity();  continue;  }  std::sort(fronts[i].begin(), fronts[i].end(), [&](size\_t a, size\_t b) {  return cur[a].objs()[f] < cur[b].objs()[f];  });  crowding\_distance[fronts[i][0]] = crowding\_distance[fronts[i][front\_size - 1]] = std::numeric\_limits<double>::infinity();  const double obj\_range = cur[fronts[i][front\_size - 1]].objs()[f] - cur[fronts[i][0]].objs()[f];  for (size\_t j = 1; j < front\_size - 1; ++j)  {  const size\_t indv\_index = fronts[i][j];  crowding\_distance[indv\_index] += (cur[fronts[i][j + 1]].objs()[f] - cur[fronts[i][j - 1]].objs()[f]) / obj\_range;  }  }  }  return crowding\_distance;  }  void SortBasedOnCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts)  {  std::vector<double> crowding\_distance = CalculateCrowdingDistance(cur, fronts);  // Sort individuals within each front based on crowding distance  for (size\_t i = 0; i < fronts.size(); ++i)  {  CNondominatedSort::TFrontMembers& front = fronts[i];  std::sort(front.begin(), front.end(), [&](size\_t a, size\_t b) {  return crowding\_distance[a] > crowding\_distance[b]; // Sort in descending order of crowding distance  });  }    }  // #include "survivor\_selection.h"  // #include "population.h"  // #include "math\_aux.h"  // #include "nondominated\_sort.h"  // #include "crowding\_distance.h"  #include <limits>  #include <algorithm>  using namespace std;  // ----------------------------------------------------------------------  // SurvivorSelection():  // ----------------------------------------------------------------------  void SurvivorSelection(CPopulation \*pnext, CPopulation \*pcur, size\_t PopSize)  {  CPopulation &cur = \*pcur, &next = \*pnext;  next.clear();  // ---------- Step 4 in Algorithm 1: non-dominated sorting ----------  CNondominatedSort::TFronts fronts = NondominatedSort(cur).second;    SortBasedOnCrowdingDistance(cur, fronts);  for (size\_t t = 0; t < fronts.size() - 1; t += 1)  {  for (size\_t i = 0; i < fronts[t].size(); i += 1) {  if (next.size() >= PopSize) break;  next.push\_back(cur[fronts[t][i]]);  }  }  }  // ----------------------------------------------------------------------  // #include "gnuplot\_interface.h"  #include <iostream>  #include <sstream>  using namespace std;  // ---------------------------------------------------------  // Ref:  // http://user.frdm.info/ckhung/b/ma/gnuplot.php  // ---------------------------------------------------------  Gnuplot::Gnuplot()  {  // with -persist option you will see the windows as your program ends  //gnuplotpipe=\_popen("gnuplot -persist","w");  //without that option you will not see the window  // because I choose the terminal to output files so I don't want to see the window  gnuplotpipe = \_popen("gnuplot", "w");  if (!gnuplotpipe)  {  cerr << ("Gnuplot not found !");  }  }  // ---------------------------------------------------------  Gnuplot::~Gnuplot()  {  fprintf(gnuplotpipe, "exit\n");  \_pclose(gnuplotpipe);  }  // ---------------------------------------------------------  void Gnuplot::operator()(const string& command)  {  fprintf(gnuplotpipe, "%s\n", command.c\_str());  fflush(gnuplotpipe);  // flush is necessary, nothing gets plotted else  }  // ---------------------------------------------------------  void Gnuplot::set\_title(const std::string& title)  {  ostringstream ss;  ss << "set title \"" << title << "\"";  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::plot(const std::string& fname, std::size\_t x, std::size\_t y)  {  ostringstream ss;  ss << "plot \"" << fname << "\" using " << x << ":" << y;  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z)  {  ostringstream ss;  ss << "splot \"" << fname << "\" using " << x << ":" << y << ":" << z;  operator()(ss.str());  }  // ---------------------------------------------------------  // #include "assignment\_problem.h"  // #include "individual.h"  // #include "math\_aux.h"  #include <cmath>  #include <vector>  #include<iostream>  using std::size\_t;  using std::cos;  // ----------------------------------------------------------------------  // CProblemAssignment  // ----------------------------------------------------------------------  CProblemAssignment::CProblemAssignment(size\_t M, size\_t k, const std::string& name, double lbs, double ubs) :  BProblem(name),  M\_(M),  k\_(k)  {  lbs\_.resize( k\_, lbs); // lower bound  ubs\_.resize( k\_, ubs); // upper bound  }  bool CProblemAssignment5::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  for (size\_t i = 1; i < k\_; ++i) {  f[0] += -10 \* exp(-0.2 \* sqrt(MathAux::square(x[i - 1]) + MathAux::square(x[i])));  }  for (size\_t i = 0; i < k\_; ++i) {  f[1] += pow(abs(x[i]), 0.8) + 5 \* sin(pow(x[i], 3));  }    return true;  }  bool CProblemAssignment6::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  //std::cout << (x.size() == k\_)<<'\n';  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  f[0] = MathAux::square(x[0]);  f[1] = MathAux::square(x[0] - 2);  return true;  }  // #include "log.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  #include <fstream>  #include<iostream>  using namespace std;  #define OUTPUT\_DECISION\_VECTOR  bool SaveToFile(const std::string& fname, const CPopulation& pop, ios\_base::openmode mode)  {  ofstream ofile(fname.c\_str(), mode);  if (!ofile) return false;  for (size\_t i = 0; i < pop.size(); i += 1)  {  #ifdef OUTPUT\_DECISION\_VECTOR  for (size\_t j = 0; j < pop[i].vars().size(); j += 1)  {  ofile << pop[i].vars()[j] << ' ';  }  #endif  for (size\_t f = 0; f < pop[i].objs().size(); f += 1)  {  ofile << pop[i].objs()[f] << ' ';  }  ofile << endl;  }  ofile << endl;  return true;  }  // ----------------------------------------------------------------------  bool ShowPopulation(Gnuplot& gplot, const CPopulation& pop, const std::string& legend)  {  if (!SaveToFile(legend, pop, ios\_base::out)) return false;  size\_t n = 0;  #ifdef OUTPUT\_DECISION\_VECTOR  n = pop[0].vars().size();  #endif    if (pop[0].objs().size() == 2)  {  gplot.plot(legend, n + 1, n + 2);  return true;  }  else if (pop[0].objs().size() == 3)  {  gplot.splot(legend, n + 1, n + 2, n + 3);  return true;  }  else  return false;  }  // #include "assignment\_problem.h"  // #include "nsgaii.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include <ctime>  #include <cstdlib>  #include <iostream>  // #include "individual.h"  // #include "math\_aux.h"  using namespace std;  int main()  {  CNSGAII nsgaii("nsgaii");  CPopulation solutions;  Gnuplot gplot;  const size\_t NumRuns = 10;  BProblem\* problem5 = new CProblemAssignment5(2, 3,-5,5);  BProblem\* problem6 = new CProblemAssignment6(2, 1,-10,10);  BProblem\* problem = problem5;    for (size\_t r = 0; r < NumRuns; r += 1)  {  srand(r); cout << "Run Number: " << r << endl;  nsgaii.Solve(&solutions, \*problem);  SaveToFile(nsgaii.name() + "-" + problem->name() + ".txt", solutions);  ShowPopulation(gplot, solutions, "pop"); // system("pause");  }  delete problem;  return 0;  }  Input: Terminal Input is not required Everything is Automated. If you want to change the parameters  please update the main function and nsgaii function.  You can also add nsgaii.ini file and write parameter values.  Output:  Values of F1 and F2 after max iteration        The Values after the max iteration (x1, x2, x3, f1, f2)    Hands On calculations:  Number of populations =4  Tournament parent selection  SBX crossover  Polynomial Mutation  Survivor selection Based on Crowding Distance and Rank  Generation Number =1  Parent selection   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | | 1 | -4.9884 | -2.64428 | 1.48152 | -8.68714 | 13.3724 | | 2 | -4.25626 | -2.29759 | -1.39973 | -9.63955 | 1.66223 | | 3 | -2.44713 | 4.8529 | -1.81082 | -6.92117 | 9.18446 | | 4 | 4.34233 | -3.21375 | 3.5757 | -7.21749 | 9.56758 |     Crossover     |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Father | Mother |  |  | c1 |  |  |  | c2 |  | |  |  |  | x1 | x2 | x3 |  | x1 | x2 | x3 | | 3 | 2 | 1 | -4.23878 | -2.28495 | -1.81082 | 2 | -2.46461 | 4.82254 | -1.39973 | | 3 | 3 | 3 | -2.44713 | 4.8529 | -1.81082 | 4 | -2.44713 | 4.8529 | -1.81082 |   Mutation   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  | Mutated Value |  |  |  | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | | 1 | -4.23878 | -2.28495 | -1.81082 | -16.32 17 | 17.1229 | | 2 | -1.80237 | 4.57723 | -1.39973 | -17.2175 | 13.0664 | | 3 | -2.44713 | 4.8529 | -1.81082 | -13.8423 | 18.3689 | | 4 | -1.70358 | 4.95464 | -1.81082 | -13.9098 | 26.3758 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | Rank | Crowding Distance | Next Generation | | 1 | -4.9884 | -2.64428 | 1.48152 | 2 | 1.92021 | 2 | | 2 | -4.25626 | -2.29759 | -1.39973 | 1 | inf | 6 | | 3 | -2.44713 | 4.8529 | -1.81082 | 2 | inf | 3 | | 4 | 4.34233 | -3.21375 | 3.5757 | 2 | 0.715446 | 5 | | 5 | -4.23878 | -2.28495 | -1.81082 | 2 | inf |  | | 6 | -1.80237 | 4.57723 | -1.39973 | 1 | inf |  | | 7 | -2.44713 | 4.8529 | -1.81082 | 3 | inf |  | | 8 | -1.70358 | 4.95464 | -1.81082 | 3 | inf |  |   Generation Number =2  Parent selection   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | | 1 | -4.25626 | -2.29759 | -1.39973 | -9.63955 | 1.66223 | | 2 | -1.80237 | 4.57723 | -1.39973 | -17.2175 | 13.0664 | | 3 | -2.44713 | 4.8529 | -1.81082 | -6.92117 | 9.18446 | | 4 | -4.23878 | -2.28495 | -1.81082 | -16.32 17 | 17.1229 |     Crossover     |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Father | Mother |  |  | c1 |  |  |  | c2 |  | |  |  |  | x1 | x2 | x3 |  | x1 | x2 | x3 | | 2 | 1 | 1 | -1.80237 | 4.5803 | -1.39973 | 2 | -4.25626 | -2.30231 | -1.39973 | | 1 | 3 | 3 | -4.25626 | 4.8693 | -1.39973 | 4 | -2.44713 | -2.34261 | -1.81082 |   Mutation   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  | Mutated Value |  |  |  | | Sr. No. | x1 | x2 | x3 | f1(x) | f2(x) | | 1 | -1.80237 | 4.5803 | -1.39973 | -24.7911 | 24.3035 | | 2 | -4.25626 | -2.30231 | -2.28626 | -18.6648 | 8.44429 | | 3 | -4.25626 | 4.93193 | -1.39973 | -15.9437 | 5.5974 | | 4 | -2.44713 | -2.33324 | -1.81082 | -17.5459 | 11.4784 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | x2 | x3 | Rank | Crowding Distance | Next Generation | | 1 | -4.25626 | -2.29759 | -1.39973 | 1 | inf | 1 | | 2 | -1.80237 | 4.57723 | -1.39973 | 3 | inf | 5 | | 3 | -2.44713 | 4.8529 | -1.81082 | 2 | inf | 6 | | 4 | -4.23878 | -2.28495 | -1.81082 | 4 | inf | 7 | | 5 | -1.80237 | 4.5803 | -1.39973 | 1 | inf |  | | 6 | -4.25626 | -2.30231 | -2.28626 | 1 | 1.4101 |  | | 7 | -4.25626 | 4.93193 | -1.39973 | 1 | 0.89520 |  | | 8 | -2.44713 | -2.33324 | -1.81082 | 2 | inf |  |   Best Answer: x = [-4.25626 -2.29759 -1.39973] (Randomly selected one from pareto-optimal front)      **Problem 6:**        C++ code for the SGA:    #ifndef MATH\_AUX\_\_  #define MATH\_AUX\_\_  #include <cstdlib>  #include <vector>  namespace MathAux  {  const double PI = 3.1415926;  const double EPS = 1.0e-14; // follow nsga-ii source code  inline double square(double n) { return n \* n; }  inline double random(double lb, double ub) { return lb + (static\_cast<double>(std::rand()) / RAND\_MAX) \* (ub - lb); }  // ASF(): achievement scalarization function  double ASF(const std::vector<double>& objs, const std::vector<double>& weight);  }  #endif  #ifndef BASE\_PROBLEM\_\_  #define BASE\_PROBLEM\_\_  #include <string>  #include <vector>  // ----------------------------------------------------------------------  // BProblem: the base class of problems (e.g. ZDT and DTLZ)  // ----------------------------------------------------------------------  class CIndividual;  class BProblem  {  public:  explicit BProblem(const std::string& name) :name\_(name) {}  virtual ~BProblem() {}  virtual std::size\_t num\_variables() const = 0;  virtual std::size\_t num\_objectives() const = 0;  virtual bool Evaluate(CIndividual\* indv) const = 0;  const std::string& name() const { return name\_; }  const std::vector<double>& lower\_bounds() const { return lbs\_; }  const std::vector<double>& upper\_bounds() const { return ubs\_; }  protected:  std::string name\_;  std::vector<double> lbs\_, // lower bounds of variables  ubs\_; // upper bounds of variables  };  #endif  #ifndef ASSIGNMENT\_PROBLEM\_\_  #define ASSIGNMENT\_PROBLEM\_\_  // #include "base\_problem.h"  #include <cstddef>  #include<iostream>  // ----------------------------------------------------------------------  class CProblemAssignment : public BProblem  {  public:  CProblemAssignment(std::size\_t M, std::size\_t k, const std::string& name, double lbs, double ubs);  virtual std::size\_t num\_variables() const { return k\_; }  virtual std::size\_t num\_objectives() const { return M\_; }  virtual bool Evaluate(CIndividual\* indv) const = 0;  protected:  std::size\_t M\_; // number of objectives  std::size\_t k\_; // number of variables  };  // ----------------------------------------------------------------------  // CProblemAssignment5  // ----------------------------------------------------------------------  class CProblemAssignment5 : public CProblemAssignment  {  public:  explicit CProblemAssignment5(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment5", lbs, ubs) {}  virtual bool Evaluate(CIndividual\* indv) const;  };  // ----------------------------------------------------------------------  // CProblemAssignment6  // ----------------------------------------------------------------------  class CProblemAssignment6 : public CProblemAssignment  {  public:  explicit CProblemAssignment6(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment6", lbs, ubs) {  //std::cout << "what happend !" << lbs << '\n';  }  virtual bool Evaluate(CIndividual\* indv) const;  };  #endif  #ifndef COMPARATOR\_\_  #define COMPARATOR\_\_  class CIndividual;  // ----------------------------------------------------------------------------------  // BComparator : the base class of comparison operators  // ----------------------------------------------------------------------------------  class BComparator  {  public:  virtual ~BComparator() {}  virtual bool operator()(const CIndividual& l, const CIndividual& r) const = 0;  };  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  class CParetoDominate : public BComparator  {  public:  virtual bool operator()(const CIndividual& l, const CIndividual& r) const;  };  extern CParetoDominate ParetoDominate;  #endif  #ifndef CROSSOVER\_\_  #define CROSSOVER\_\_  // ----------------------------------------------------------------------------------  // CSimulatedBinaryCrossover : simulated binary crossover (SBX)  // ----------------------------------------------------------------------------------  class CIndividual;  class CSimulatedBinaryCrossover  {  public:  explicit CSimulatedBinaryCrossover(double cr = 1.0, double eta = 30) :cr\_(cr), eta\_(eta) {} // NSGA-III (t-EC 2014) setting  void SetCrossoverRate(double cr) { cr\_ = cr; }  double CrossoverRate() const { return cr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2, double cr, double eta) const;  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2) const  {  return operator()(c1, c2, p1, p2, cr\_, eta\_);  }  private:  double get\_betaq(double rand, double alpha, double eta) const;  double cr\_; // crossover rate  double eta\_; // distribution index  };  #endif  #include <string>  #include <stdio.h>  #include <string>  #include <cstddef>  // ----------------------------------------------------------------------  // Gnuplot  //  // This is just a very simple interface to call gnuplot in the program.  // Now it seems to work only under windows + visual studio.  // ----------------------------------------------------------------------  class Gnuplot  {  public:  Gnuplot();  ~Gnuplot();  // prohibit copying (VS2012 does not support 'delete')  Gnuplot(const Gnuplot&);  Gnuplot& operator=(const Gnuplot&);  // send any command to gnuplot  void operator ()(const std::string& command);  void reset() { operator()("reset"); }  void replot() { operator()("replot"); }  void set\_title(const std::string& title);  void plot(const std::string& fname, std::size\_t x, std::size\_t y);  void splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z);  protected:  FILE\* gnuplotpipe;  };  #ifndef INDIVIDUAL\_\_  #define INDIVIDUAL\_\_  #include <vector>  #include <ostream>  // ----------------------------------------------------------------------  // CIndividual  // ----------------------------------------------------------------------  class BProblem;  class CIndividual  {  public:  typedef double TGene;  typedef std::vector<TGene> TDecVec;  typedef std::vector<double> TObjVec;  explicit CIndividual(std::size\_t num\_vars = 0, std::size\_t num\_objs = 0);  TDecVec& vars() { return variables\_; }  const TDecVec& vars() const { return variables\_; }  TObjVec& objs() { return objectives\_; }  const TObjVec& objs() const { return objectives\_; }  TObjVec& conv\_objs() { return converted\_objectives\_; }  const TObjVec& conv\_objs() const { return converted\_objectives\_; }  // if a target problem is set, memory will be allocated accordingly in the constructor  static void SetTargetProblem(const BProblem& p) { target\_problem\_ = &p; }  static const BProblem& TargetProblem();  private:  TDecVec variables\_;  TObjVec objectives\_;  TObjVec converted\_objectives\_;  static const BProblem\* target\_problem\_;  };  std::ostream& operator << (std::ostream& os, const CIndividual& indv);  #endif  #ifndef INITIALIZATION\_\_  #define INITIALIZATION\_\_  // ----------------------------------------------------------------------  // CRandomInitialization  // ----------------------------------------------------------------------  class CIndividual;  class CPopulation;  class BProblem;  class CRandomInitialization  {  public:  void operator()(CPopulation\* pop, const BProblem& prob) const;  void operator()(CIndividual\* indv, const BProblem& prob) const;  };  extern CRandomInitialization RandomInitialization;  #endif  #ifndef POPULATION\_\_  #define POPULATION\_\_  // #include "individual.h"  #include <vector>  class CPopulation  {  public:  explicit CPopulation(std::size\_t s = 0) :individuals\_(s) {}  CIndividual& operator[](std::size\_t i) { return individuals\_[i]; }  const CIndividual& operator[](std::size\_t i) const { return individuals\_[i]; }  std::size\_t size() const { return individuals\_.size(); }  void resize(size\_t t) { individuals\_.resize(t); }  void push\_back(const CIndividual& indv) { individuals\_.push\_back(indv); }  void clear() { individuals\_.clear(); }  private:  std::vector<CIndividual> individuals\_;  };  #endif  #ifndef ENVIRONMENTAL\_SELECTION\_\_  #define ENVIRONMENTAL\_SELECTION\_\_  #include <vector>  // ----------------------------------------------------------------------  // The environmental selection mechanism is the key innovation of  // the GA algorithm.  //  // Check Algorithm I in the original paper of NSGA-III.  // ----------------------------------------------------------------------  class CPopulation;  class CReferencePoint;  void SurvivorSelection(CPopulation\* pnext, // population in the next generation  CPopulation\* pcur, // population in the current generation  size\_t PopSize);  #endif  #ifndef LOG\_\_  #define LOG\_\_  #include <string>  #include <fstream> // Include <fstream> for std::ios\_base and std::ios\_base::openmode  class CPopulation;  class Gnuplot;  // Save a population into the designated file.  bool SaveToFile(const std::string& fname, const CPopulation& pop, std::ios\_base::openmode mode = std::ios\_base::app);  // Show a population by calling gnuplot.  bool ShowPopulation(Gnuplot& gplot, const CPopulation&, const std::string& legend = "pop");  #endif  #ifndef MUTATION\_\_  #define MUTATION\_\_  // ----------------------------------------------------------------------------------  // CPolynomialMutation : polynomial mutation  // ----------------------------------------------------------------------------------  class CIndividual;  class CPolynomialMutation  {  public:  explicit CPolynomialMutation(double mr = 0.0, double eta = 20) :mr\_(mr), eta\_(eta) {}  void SetMutationRate(double mr) { mr\_ = mr; }  double MutationRate() const { return mr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c, double mr, double eta) const;  bool operator()(CIndividual\* c) const  {  return operator()(c, mr\_, eta\_);  }  private:  double mr\_, // mutation rate  eta\_; // distribution index  };  #endif  #ifndef GA\_\_  #define GA\_\_  #include <cstddef>  #include <string>  // ----------------------------------------------------------------------------------  class BProblem;  class CPopulation;  class CGA  {  public:  explicit CGA(const std::string& inifile\_name = "");  void Solve(CPopulation\* solutions, const BProblem& prob);  const std::string& name() const { return name\_; }  private:  std::string name\_;  std::size\_t obj\_division\_p\_;  std::size\_t gen\_num\_;  double pc\_, // crossover rate  pm\_, // mutation rate  eta\_c\_, // eta in SBX  eta\_m\_; // eta in Polynomial Mutation  };  #endif  // #include "ga.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "initialization.h"  // #include "crossover.h"  // #include "mutation.h"  // #include "survivor\_selection.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include "windows.h" // for Sleep()  #include <vector>  #include <fstream>  #include<iostream>  using namespace std;  CGA::CGA(const string& inifile\_name) :  name\_("GA"),  obj\_division\_p\_(12),  gen\_num\_(1000),  pc\_(1.0), // default setting in GA  eta\_c\_(30), // default setting  eta\_m\_(20) // default setting  {  if (inifile\_name == "") return;  ifstream inifile(inifile\_name);  if (!inifile) return;  string dummy;  inifile >> dummy >> dummy >> name\_;  inifile >> dummy >> dummy >> obj\_division\_p\_;  inifile >> dummy >> dummy >> gen\_num\_;  inifile >> dummy >> dummy >> pc\_;  inifile >> dummy >> dummy >> eta\_c\_;  inifile >> dummy >> dummy >> eta\_m\_;  }  // ----------------------------------------------------------------------  void CGA::Solve(CPopulation\* solutions, const BProblem& problem)  {  CIndividual::SetTargetProblem(problem);  size\_t PopSize = 50;  while (PopSize % 4) PopSize += 1;  //CPopulation pop[2] = { CPopulation(PopSize) };  std::vector<CPopulation> pop(2, CPopulation(PopSize));  CSimulatedBinaryCrossover SBX(pc\_, eta\_c\_);  CPolynomialMutation PolyMut(1.0 / problem.num\_variables(), eta\_m\_);  Gnuplot gplot;  int cur = 0, next = 1;  //std::cout << problem.lower\_bounds()[0] << '\n';  RandomInitialization(&pop[cur], problem);  for (size\_t i = 0; i < PopSize; i += 1)  {  problem.Evaluate(&pop[cur][i]);  }  for (size\_t t = 0; t < gen\_num\_; t += 1)  {  pop[cur].resize(PopSize \* 2);  for (size\_t i = 0; i < PopSize; i += 2)  {  int father = rand() % PopSize,  mother = rand() % PopSize;  SBX(&pop[cur][PopSize + i], &pop[cur][PopSize + i + 1], pop[cur][father], pop[cur][mother]);  PolyMut(&pop[cur][PopSize + i]);  PolyMut(&pop[cur][PopSize + i + 1]);  problem.Evaluate(&pop[cur][PopSize + i]);  problem.Evaluate(&pop[cur][PopSize + i + 1]);  }  SurvivorSelection(&pop[next], &pop[cur], PopSize);  //ShowPopulation(gplot, pop[next], "pop"); Sleep(10);  std::swap(cur, next);  }  \*solutions = pop[cur];  }  // #include "comparator.h"  // #include "individual.h"  // ----------------------------------------------------------------------------------  CParetoDominate ParetoDominate;  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  bool CParetoDominate::operator()(const CIndividual& l, const CIndividual& r) const  {  bool better = false;  for (size\_t f = 0; f < l.objs().size(); f += 1)  {  if (l.objs()[f] > r.objs()[f])  return false;  else if (l.objs()[f] < r.objs()[f])  better = true;  }  return better;  }// CParetoDominate::operator()  // ----------------------------------------------------------------------------------  #include <vector>  // #include "math\_aux.h"  #include<limits>  using namespace std;  namespace MathAux  {  // ----------------------------------------------------------------------  // ASF: Achivement Scalarization Function  // ----------------------------------------------------------------------  double ASF(const vector<double> &objs, const vector<double> &weight)  {  double max\_ratio = -numeric\_limits<double>::max();  for (size\_t f=0; f<objs.size(); f+=1)  {  double w = weight[f]?weight[f]:0.00001;  max\_ratio = std::max(max\_ratio, objs[f]/w);  }  return max\_ratio;  }  // ---------------------------------------------------------------------  }// namespace MathAux  // #include "individual.h"  // #include "base\_problem.h"  using std::size\_t;  const BProblem\* CIndividual::target\_problem\_ = 0;  // ----------------------------------------------------------------------  CIndividual::CIndividual(std::size\_t num\_vars, std::size\_t num\_objs) :  variables\_(num\_vars),  objectives\_(num\_objs),  converted\_objectives\_(num\_objs)  {  if (target\_problem\_ != 0)  {  variables\_.resize(target\_problem\_->num\_variables());  objectives\_.resize(target\_problem\_->num\_objectives());  converted\_objectives\_.resize(target\_problem\_->num\_objectives());  }  }  // ----------------------------------------------------------------------  const BProblem& CIndividual::TargetProblem() { return \*target\_problem\_; }  // ----------------------------------------------------------------------  std::ostream& operator << (std::ostream& os, const CIndividual& indv)  {  for (size\_t i = 0; i < indv.vars().size(); i += 1)  {  os << indv.vars()[i] << ' ';  }  os << " => ";  for (size\_t f = 0; f < indv.objs().size(); f += 1)  {  os << indv.objs()[f] << ' ';  }  return os;  }  // ----------------------------------------------------------------------  // #include "initialization.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "math\_aux.h"  #include <cstddef>  #include<iostream>  using std::size\_t;  CRandomInitialization RandomInitialization;  void CRandomInitialization::operator()(CIndividual\* indv, const BProblem& prob) const  {  CIndividual::TDecVec& x = indv->vars();  x.resize(prob.num\_variables());  for (size\_t i = 0; i < x.size(); i += 1)  {  x[i] = MathAux::random(prob.lower\_bounds()[i], prob.upper\_bounds()[i]);  }    }  // ----------------------------------------------------------------------  void CRandomInitialization::operator()(CPopulation\* pop, const BProblem& prob) const  {  for (size\_t i = 0; i < pop->size(); i += 1)  {  (\*this)(&(\*pop)[i], prob);  }  }  // #include "crossover.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cmath>  #include <algorithm>  #include <cstddef>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function realcross() in crossover.c  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ----------------------------------------------------------------------  double CSimulatedBinaryCrossover::get\_betaq(double rand, double alpha, double eta) const  {  double betaq = 0.0;  if (rand <= (1.0 / alpha))  {  betaq = std::pow((rand \* alpha), (1.0 / (eta + 1.0)));  }  else  {  betaq = std::pow((1.0 / (2.0 - rand \* alpha)), (1.0 / (eta + 1.0)));  }  return betaq;  }  // ----------------------------------------------------------------------  bool CSimulatedBinaryCrossover::operator()(CIndividual\* child1,  CIndividual\* child2,  const CIndividual& parent1,  const CIndividual& parent2,  double cr,  double eta) const  {  \*child1 = parent1;  \*child2 = parent2;  if (MathAux::random(0.0, 1.0) > cr) return false; // not crossovered  CIndividual::TDecVec& c1 = child1->vars(), & c2 = child2->vars();  const CIndividual::TDecVec& p1 = parent1.vars(), & p2 = parent2.vars();  for (size\_t i = 0; i < c1.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) > 0.5) continue; // these two variables are not crossovered  if (std::fabs(p1[i] - p2[i]) <= MathAux::EPS) continue; // two values are the same  double y1 = std::min(p1[i], p2[i]),  y2 = std::max(p1[i], p2[i]);  double lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double rand = MathAux::random(0.0, 1.0);  // child 1  double beta = 1.0 + (2.0 \* (y1 - lb) / (y2 - y1)),  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  double betaq = get\_betaq(rand, alpha, eta);  c1[i] = 0.5 \* ((y1 + y2) - betaq \* (y2 - y1));  // child 2  beta = 1.0 + (2.0 \* (ub - y2) / (y2 - y1));  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  betaq = get\_betaq(rand, alpha, eta);  c2[i] = 0.5 \* ((y1 + y2) + betaq \* (y2 - y1));  // boundary checking  c1[i] = std::min(ub, std::max(lb, c1[i]));  c2[i] = std::min(ub, std::max(lb, c2[i]));  if (MathAux::random(0.0, 1.0) <= 0.5)  {  std::swap(c1[i], c2[i]);  }  }  return true;  }// CSimulatedBinaryCrossover  // #include "mutation.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cstddef>  #include <algorithm>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function real\_mutate\_ind() in mutation.c in  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ---------------------------------------------------------------------  bool CPolynomialMutation::operator()(CIndividual\* indv, double mr, double eta) const  {  //int j;  //double rnd, delta1, delta2, mut\_pow, deltaq;  //double y, yl, yu, val, xy;  bool mutated = false;  CIndividual::TDecVec& x = indv->vars();  for (size\_t i = 0; i < x.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) <= mr)  {  mutated = true;  double y = x[i],  lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double delta1 = (y - lb) / (ub - lb),  delta2 = (ub - y) / (ub - lb);  double mut\_pow = 1.0 / (eta + 1.0);  double rnd = MathAux::random(0.0, 1.0), deltaq = 0.0;  if (rnd <= 0.5)  {  double xy = 1.0 - delta1;  double val = 2.0 \* rnd + (1.0 - 2.0 \* rnd) \* (pow(xy, (eta + 1.0)));  deltaq = pow(val, mut\_pow) - 1.0;  }  else  {  double xy = 1.0 - delta2;  double val = 2.0 \* (1.0 - rnd) + 2.0 \* (rnd - 0.5) \* (pow(xy, (eta + 1.0)));  deltaq = 1.0 - (pow(val, mut\_pow));  }  y = y + deltaq \* (ub - lb);  y = std::min(ub, std::max(lb, y));  x[i] = y;  }  }  return mutated;  }// CPolynomialMutation  // #include "survivor\_selection.h"  // #include "population.h"  // #include "math\_aux.h"  #include <limits>  #include <algorithm>  using namespace std;  // ----------------------------------------------------------------------  // SurvivorSelection():  // ----------------------------------------------------------------------  void SurvivorSelection(CPopulation \*pnext, CPopulation \*pcur, size\_t PopSize)  {  CPopulation &cur = \*pcur, &next = \*pnext;  next.clear();  std::vector<size\_t> index;  for (int i = 0; i < cur.size(); i++)index.push\_back(i);  std::sort(index.begin(), index.end(), [&](size\_t a, size\_t b) {  return cur[a].objs()[0]+ cur[a].objs()[1] < cur[b].objs()[0]+ cur[b].objs()[1];  });    for (size\_t t = 0; t < PopSize ; t += 1)  {  next.push\_back(cur[index[t]]);  }  }  // ----------------------------------------------------------------------  // #include "gnuplot\_interface.h"  #include <iostream>  #include <sstream>  using namespace std;  // ---------------------------------------------------------  // Ref:  // http://user.frdm.info/ckhung/b/ma/gnuplot.php  // ---------------------------------------------------------  Gnuplot::Gnuplot()  {  // with -persist option you will see the windows as your program ends  //gnuplotpipe=\_popen("gnuplot -persist","w");  //without that option you will not see the window  // because I choose the terminal to output files so I don't want to see the window  gnuplotpipe = \_popen("gnuplot", "w");  if (!gnuplotpipe)  {  cerr << ("Gnuplot not found !");  }  }  // ---------------------------------------------------------  Gnuplot::~Gnuplot()  {  fprintf(gnuplotpipe, "exit\n");  \_pclose(gnuplotpipe);  }  // ---------------------------------------------------------  void Gnuplot::operator()(const string& command)  {  fprintf(gnuplotpipe, "%s\n", command.c\_str());  fflush(gnuplotpipe);  // flush is necessary, nothing gets plotted else  }  // ---------------------------------------------------------  void Gnuplot::set\_title(const std::string& title)  {  ostringstream ss;  ss << "set title \"" << title << "\"";  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::plot(const std::string& fname, std::size\_t x, std::size\_t y)  {  ostringstream ss;  ss << "plot \"" << fname << "\" using " << x << ":" << y;  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z)  {  ostringstream ss;  ss << "splot \"" << fname << "\" using " << x << ":" << y << ":" << z;  operator()(ss.str());  }  // ---------------------------------------------------------  // #include "assignment\_problem.h"  // #include "individual.h"  // #include "math\_aux.h"  #include <cmath>  #include <vector>  #include<iostream>  using std::size\_t;  using std::cos;  // ----------------------------------------------------------------------  // CProblemAssignment  // ----------------------------------------------------------------------  CProblemAssignment::CProblemAssignment(size\_t M, size\_t k, const std::string& name, double lbs, double ubs) :  BProblem(name),  M\_(M),  k\_(k)  {  lbs\_.resize( k\_, lbs); // lower bound  ubs\_.resize( k\_, ubs); // upper bound  }  bool CProblemAssignment5::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  for (size\_t i = 1; i < k\_; ++i) {  f[0] += -10 \* exp(-0.2 \* sqrt(MathAux::square(x[i - 1]) + MathAux::square(x[i])));  }  for (size\_t i = 0; i < k\_; ++i) {  f[1] += pow(abs(x[i]), 0.8) + 5 \* sin(pow(x[i], 3));  }    return true;  }  bool CProblemAssignment6::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  //std::cout << (x.size() == k\_)<<'\n';  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  f[0] = MathAux::square(x[0]);  f[1] = MathAux::square(x[0] - 2);  return true;  }  // #include "log.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  #include <fstream>  #include<iostream>  using namespace std;  #define OUTPUT\_DECISION\_VECTOR  bool SaveToFile(const std::string& fname, const CPopulation& pop, ios\_base::openmode mode)  {  ofstream ofile(fname.c\_str(), mode);  if (!ofile) return false;    for (size\_t i = 0; i < pop.size(); i += 1)  {  #ifdef OUTPUT\_DECISION\_VECTOR    for (size\_t j = 0; j < pop[i].vars().size(); j += 1)  {  ofile << pop[i].vars()[j] << ' ';  }  #endif  //std::cout << pop[i].objs().size() << endl;  for (size\_t f = 0; f < pop[i].objs().size(); f += 1)  {  ofile << pop[i].objs()[f] << ' ';  }    ofile << pop[i].objs()[0] + pop[i].objs()[1] << ' ';    ofile << endl;  }  ofile << endl;  return true;  }  // ----------------------------------------------------------------------  bool ShowPopulation(Gnuplot& gplot, const CPopulation& pop, const std::string& legend)  {  if (!SaveToFile(legend, pop, ios\_base::out)) return false;  size\_t n = 0;  #ifdef OUTPUT\_DECISION\_VECTOR  n = pop[0].vars().size();  #endif    if (pop[0].objs().size() == 2)  {  gplot.plot(legend, n + 1, n + 2);  return true;  }  else if (pop[0].objs().size() == 3)  {  gplot.splot(legend, n + 1, n + 2, n + 3);  return true;  }  else  return false;  }  // #include "assignment\_problem.h"  // #include "ga.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include <ctime>  #include <cstdlib>  #include <iostream>  // #include "individual.h"  // #include "math\_aux.h"  using namespace std;  int main()  {  CGA ga("ga");  CPopulation solutions;  Gnuplot gplot;  const size\_t NumRuns = 10;  BProblem\* problem5 = new CProblemAssignment5(2, 3,-5,5);  BProblem\* problem6 = new CProblemAssignment6(2, 1,-10,10);  BProblem\* problem = problem6;  //std::cout << problem->upper\_bounds()[0] << '\n';  for (size\_t r = 0; r < NumRuns; r += 1)  {  srand(r); cout << "Run Number: " << r << endl;  ga.Solve(&solutions, \*problem);  SaveToFile(ga.name() + "-" + problem->name() + ".txt", solutions);  ShowPopulation(gplot, solutions, "pop"); system("pause");  }  delete problem;  return 0;  }  Input: Terminal Input is not required Everything is Automated. If you want to change the parameters  please update the main function and ga function.  You can also add ga.ini file and write parameter values.  Output:    Values of F1 and F2 after Max iteration    The Values after the max iteration (x, f1, f2, f1+f2)      Hands On calculations:  Number of populations =4  Tournament parent selection  SBX crossover  Polynomial Mutation  Survivor selection Fitness propionate  Generation Number =1  Parent selection   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | g(x) | | 1 | -9.97681 | 99.5367 | 143.444 | 242.981 | | 2 | -5.28855 | 27.9688 | 53.123 | 81.0918 | | 3 | 2.96304 | 8.77962 | 0.92745 | 9.70707 | | 4 | -8.51253 | 72.4631 | 110.513 | 182.976 |     Crossover     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Father | Mother |  | c1 |  | c2 | |  |  |  | x1 |  | x1 | | 4 | 2 | 1 | -8.51253 | 2 | -5.28855 | | 3 | 2 | 3 | -5.20879 | 4 | 2.88328 |   Mutation   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Mutated Value |  |  |  | | Sr. No. | x1 | f1(x) | f2(x) | g(x) | | 1 | -6.67101 | 44.5024 | 75.1864 | 119.689 | | 2 | -4.12765 | 17.0375 | 37.5481 | 54.5855 | | 3 | -5.31273 | 28.2251 | 53.476 | 81.7011 | | 4 | 5.4853 | 30.0885 | 12.1473 | 42.2358 |   Survivor Selection   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Sr. No. | x1 | g(x) |  | Next Generation | |  | Old Population |  |  |  | | 1 | -9.97681 | 242.981 | 1 | 3 | | 2 | -5.28855 | 81.0918 | 2 | 8 | | 3 | 2.96304 | 9.70707 | 3 | 6 | | 4 | -8.51253 | 182.976 | 4 | 2 | |  | New Population |  |  |  | | 5 | -6.67101 | 119.689 |  |  | | 6 | -4.12765 | 54.5855 |  |  | | 7 | -5.31273 | 81.7011 |  |  | | 8 | 5.4853 | 42.2358 |  |  |   Generation Number =2  Parent selection   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | g(x) | | 1 | 2.96304 | 8.77962 | 0.92745 | 9.70707 | | 2 | 5.4853 | 30.0885 | 12.1473 | 42.2358 | | 3 | -4.12765 | 17.0375 | 37.5481 | 54.5855 | | 4 | -5.28855 | 27.9688 | 53.123 | 81.0918 |     Crossover     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Father | Mother |  | c1 |  | c2 | |  |  |  | x1 |  | x1 | | 4 | 2 | 1 | -4.10007 | 2 | -5.31613 | | 1 | 4 | 3 | 2.96304 | 4 | -5.28855 |   Mutation   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Mutated Value |  |  |  | | Sr. No. | x1 | f1(x) | f2(x) | g(x) | | 1 | -1.0203 | 1.04102 | 9.12222 | 10.1632 | | 2 | -8.16728 | 66.7044 | 103.373 | 170.078 | | 3 | 2.71579 | 7.37553 | 0.51236 | 7.88789 | | 4 | -7.84949 | 61.6146 | 97.0125 | 158.627 |   Survivor Selection   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Sr. No. | x1 | g(x) |  | Next Generation | |  | Old Population |  |  |  | | 1 | 2.96304 | 9.70707 | 1 | 7 | | 2 | 5.4853 | 42.2358 | 2 | 1 | | 3 | -4.12765 | 54.5855 | 3 | 5 | | 4 | -5.28855 | 81.0918 | 4 | 2 | |  | New Population |  |  |  | | 5 | -1.0203 | 10.1632 |  |  | | 6 | -8.16728 | 170.078 |  |  | | 7 | 2.71579 | 7.88789 |  |  | | 8 | -7.84949 | 158.627 |  |  |   Best Answer: x = [2.71579]  C++ code for the NSGAII:  #ifndef MATH\_AUX\_\_  #define MATH\_AUX\_\_  #include <cstdlib>  #include <vector>  namespace MathAux  {  const double PI = 3.1415926;  const double EPS = 1.0e-14; // follow nsga-ii source code  inline double square(double n) { return n \* n; }  inline double random(double lb, double ub) { return lb + (static\_cast<double>(std::rand()) / RAND\_MAX) \* (ub - lb); }  // ASF(): achievement scalarization function  double ASF(const std::vector<double>& objs, const std::vector<double>& weight);  }  #endif  #ifndef BASE\_PROBLEM\_\_  #define BASE\_PROBLEM\_\_  #include <string>  #include <vector>  // ----------------------------------------------------------------------  // BProblem: the base class of problems (e.g. ZDT and DTLZ)  // ----------------------------------------------------------------------  class CIndividual;  class BProblem  {  public:  explicit BProblem(const std::string& name) :name\_(name) {}  virtual ~BProblem() {}  virtual std::size\_t num\_variables() const = 0;  virtual std::size\_t num\_objectives() const = 0;  virtual bool Evaluate(CIndividual\* indv) const = 0;  const std::string& name() const { return name\_; }  const std::vector<double>& lower\_bounds() const { return lbs\_; }  const std::vector<double>& upper\_bounds() const { return ubs\_; }  protected:  std::string name\_;  std::vector<double> lbs\_, // lower bounds of variables  ubs\_; // upper bounds of variables  };  #endif  #ifndef ASSIGNMENT\_PROBLEM\_\_  #define ASSIGNMENT\_PROBLEM\_\_  //#include "base\_problem.h"  #include <cstddef>  #include<iostream>  // ----------------------------------------------------------------------  class CProblemAssignment : public BProblem  {  public:  CProblemAssignment(std::size\_t M, std::size\_t k, const std::string& name, double lbs, double ubs);  virtual std::size\_t num\_variables() const { return k\_; }  virtual std::size\_t num\_objectives() const { return M\_; }  virtual bool Evaluate(CIndividual\* indv) const = 0;  protected:  std::size\_t M\_; // number of objectives  std::size\_t k\_; // number of variables  };  // ----------------------------------------------------------------------  // CProblemAssignment5  // ----------------------------------------------------------------------  class CProblemAssignment5 : public CProblemAssignment  {  public:  explicit CProblemAssignment5(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment5", lbs, ubs) {}  virtual bool Evaluate(CIndividual\* indv) const;  };  // ----------------------------------------------------------------------  // CProblemAssignment6  // ----------------------------------------------------------------------  class CProblemAssignment6 : public CProblemAssignment  {  public:  explicit CProblemAssignment6(std::size\_t M, std::size\_t k, double lbs, double ubs) :CProblemAssignment(M, k, "Assignment6", lbs, ubs) {  //std::cout << "what happend !" << lbs << '\n';  }  virtual bool Evaluate(CIndividual\* indv) const;  };  #endif  #ifndef COMPARATOR\_\_  #define COMPARATOR\_\_  class CIndividual;  // ----------------------------------------------------------------------------------  // BComparator : the base class of comparison operators  // ----------------------------------------------------------------------------------  class BComparator  {  public:  virtual ~BComparator() {}  virtual bool operator()(const CIndividual& l, const CIndividual& r) const = 0;  };  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  class CParetoDominate : public BComparator  {  public:  virtual bool operator()(const CIndividual& l, const CIndividual& r) const;  };  extern CParetoDominate ParetoDominate;  #endif  #ifndef CROSSOVER\_\_  #define CROSSOVER\_\_  // ----------------------------------------------------------------------------------  // CSimulatedBinaryCrossover : simulated binary crossover (SBX)  // ----------------------------------------------------------------------------------  class CIndividual;  class CSimulatedBinaryCrossover  {  public:  explicit CSimulatedBinaryCrossover(double cr = 1.0, double eta = 30) :cr\_(cr), eta\_(eta) {} // NSGA-III (t-EC 2014) setting  void SetCrossoverRate(double cr) { cr\_ = cr; }  double CrossoverRate() const { return cr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2, double cr, double eta) const;  bool operator()(CIndividual\* c1, CIndividual\* c2, const CIndividual& p1, const CIndividual& p2) const  {  return operator()(c1, c2, p1, p2, cr\_, eta\_);  }  private:  double get\_betaq(double rand, double alpha, double eta) const;  double cr\_; // crossover rate  double eta\_; // distribution index  };  #endif  #include <string>  #include <stdio.h>  #include <string>  #include <cstddef>  // ----------------------------------------------------------------------  // Gnuplot  //  // This is just a very simple interface to call gnuplot in the program.  // Now it seems to work only under windows + visual studio.  // ----------------------------------------------------------------------  class Gnuplot  {  public:  Gnuplot();  ~Gnuplot();  // prohibit copying (VS2012 does not support 'delete')  Gnuplot(const Gnuplot&);  Gnuplot& operator=(const Gnuplot&);  // send any command to gnuplot  void operator ()(const std::string& command);  void reset() { operator()("reset"); }  void replot() { operator()("replot"); }  void set\_title(const std::string& title);  void plot(const std::string& fname, std::size\_t x, std::size\_t y);  void splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z);  protected:  FILE\* gnuplotpipe;  };  #ifndef INDIVIDUAL\_\_  #define INDIVIDUAL\_\_  #include <vector>  #include <ostream>  // ----------------------------------------------------------------------  // CIndividual  // ----------------------------------------------------------------------  class BProblem;  class CIndividual  {  public:  typedef double TGene;  typedef std::vector<TGene> TDecVec;  typedef std::vector<double> TObjVec;  explicit CIndividual(std::size\_t num\_vars = 0, std::size\_t num\_objs = 0);  TDecVec& vars() { return variables\_; }  const TDecVec& vars() const { return variables\_; }  TObjVec& objs() { return objectives\_; }  const TObjVec& objs() const { return objectives\_; }  TObjVec& conv\_objs() { return converted\_objectives\_; }  const TObjVec& conv\_objs() const { return converted\_objectives\_; }  // if a target problem is set, memory will be allocated accordingly in the constructor  static void SetTargetProblem(const BProblem& p) { target\_problem\_ = &p; }  static const BProblem& TargetProblem();  private:  TDecVec variables\_;  TObjVec objectives\_;  TObjVec converted\_objectives\_;  static const BProblem\* target\_problem\_;  };  std::ostream& operator << (std::ostream& os, const CIndividual& indv);  #endif  #ifndef INITIALIZATION\_\_  #define INITIALIZATION\_\_  // ----------------------------------------------------------------------  // CRandomInitialization  // ----------------------------------------------------------------------  class CIndividual;  class CPopulation;  class BProblem;  class CRandomInitialization  {  public:  void operator()(CPopulation\* pop, const BProblem& prob) const;  void operator()(CIndividual\* indv, const BProblem& prob) const;  };  extern CRandomInitialization RandomInitialization;  #endif  #ifndef POPULATION\_\_  #define POPULATION\_\_  // #include "individual.h"  #include <vector>  class CPopulation  {  public:  explicit CPopulation(std::size\_t s = 0) :individuals\_(s) {}  CIndividual& operator[](std::size\_t i) { return individuals\_[i]; }  const CIndividual& operator[](std::size\_t i) const { return individuals\_[i]; }  std::size\_t size() const { return individuals\_.size(); }  void resize(size\_t t) { individuals\_.resize(t); }  void push\_back(const CIndividual& indv) { individuals\_.push\_back(indv); }  void clear() { individuals\_.clear(); }  private:  std::vector<CIndividual> individuals\_;  };  #endif  #ifndef ENVIRONMENTAL\_SELECTION\_\_  #define ENVIRONMENTAL\_SELECTION\_\_  #include <vector>  // ----------------------------------------------------------------------  // The environmental selection mechanism is the key innovation of  // the NSGA-III algorithm.  //  // Check Algorithm I in the original paper of NSGA-III.  // ----------------------------------------------------------------------  class CPopulation;  class CReferencePoint;  void SurvivorSelection(CPopulation \*pnext, // population in the next generation  CPopulation \*pcur, // population in the current generation  size\_t PopSize);  #endif  #ifndef NONDOMINATED\_SORT\_\_  #define NONDOMINATED\_SORT\_\_  #include <vector>  class BComparator;  class CPopulation;  class CNondominatedSort  {  public:  explicit CNondominatedSort(const BComparator &d):dominate(d) {}  // prohibit copying (VS2012 does not support 'delete')  CNondominatedSort(const CNondominatedSort &);  CNondominatedSort & operator= (const CNondominatedSort &);  typedef std::vector<std::size\_t> TFrontMembers; // a set of indices of individuals in a certain front  typedef std::vector<TFrontMembers> TFronts; // a set of fronts  std::pair< std::vector<size\_t>,TFronts> operator()(const CPopulation &pop) const;  private:  const BComparator &dominate;  };  extern CNondominatedSort NondominatedSort;  #endif  #ifndef LOG\_\_  #define LOG\_\_  #include <string>  #include <fstream> // Include <fstream> for std::ios\_base and std::ios\_base::openmode  class CPopulation;  class Gnuplot;  // Save a population into the designated file.  bool SaveToFile(const std::string& fname, const CPopulation& pop, std::ios\_base::openmode mode = std::ios\_base::app);  // Show a population by calling gnuplot.  bool ShowPopulation(Gnuplot& gplot, const CPopulation&, const std::string& legend = "pop");  #endif  #ifndef MUTATION\_\_  #define MUTATION\_\_  // ----------------------------------------------------------------------------------  // CPolynomialMutation : polynomial mutation  // ----------------------------------------------------------------------------------  class CIndividual;  class CPolynomialMutation  {  public:  explicit CPolynomialMutation(double mr = 0.0, double eta = 20) :mr\_(mr), eta\_(eta) {}  void SetMutationRate(double mr) { mr\_ = mr; }  double MutationRate() const { return mr\_; }  void SetDistributionIndex(double eta) { eta\_ = eta; }  double DistributionIndex() const { return eta\_; }  bool operator()(CIndividual\* c, double mr, double eta) const;  bool operator()(CIndividual\* c) const  {  return operator()(c, mr\_, eta\_);  }  private:  double mr\_, // mutation rate  eta\_; // distribution index  };  #endif  #ifndef CROWDING\_DISTANCE\_H  #define CROWDING\_DISTANCE\_H  // #include "nondominated\_sort.h"  // #include "population.h"  #include <vector>  std:: vector<double> CalculateCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts);  void SortBasedOnCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts);  #endif  #ifndef NSGAII\_\_  #define NSGAII\_\_  #include <cstddef>  #include <string>  // ----------------------------------------------------------------------------------  // NSGAII  // Taken from NSGA III  // Deb and Jain, "An Evolutionary Many-Objective Optimization Algorithm Using  // Reference-point Based Non-dominated Sorting Approach, Part I: Solving Problems with  // Box Constraints," IEEE Transactions on Evolutionary Computation, to appear.  //  // http://dx.doi.org/10.1109/TEVC.2013.2281535  // ----------------------------------------------------------------------------------  class BProblem;  class CPopulation;  class CNSGAII  {  public:  explicit CNSGAII(const std::string& inifile\_name = "");  void Solve(CPopulation\* solutions, const BProblem& prob);  const std::string& name() const { return name\_; }  private:  std::string name\_;  std::size\_t obj\_division\_p\_;  std::size\_t gen\_num\_;  double pc\_, // crossover rate  pm\_, // mutation rate  eta\_c\_, // eta in SBX  eta\_m\_; // eta in Polynomial Mutation  };  #endif  // end of headers  // #include "nsgaii.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "initialization.h"  // #include "crossover.h"  // #include "mutation.h"  // #include "survivor\_selection.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include "windows.h" // for Sleep()  #include <vector>  #include <fstream>  #include<iostream>  using namespace std;  CNSGAII::CNSGAII(const string& inifile\_name) :  name\_("NSGAII"),  obj\_division\_p\_(12),  gen\_num\_(1000),  pc\_(1.0), // default setting in NSGA-II  eta\_c\_(30), // default setting  eta\_m\_(20) // default setting  {  if (inifile\_name == "") return;  ifstream inifile(inifile\_name);  if (!inifile) return;  string dummy;  inifile >> dummy >> dummy >> name\_;  inifile >> dummy >> dummy >> obj\_division\_p\_;  inifile >> dummy >> dummy >> gen\_num\_;  inifile >> dummy >> dummy >> pc\_;  inifile >> dummy >> dummy >> eta\_c\_;  inifile >> dummy >> dummy >> eta\_m\_;  }  // ----------------------------------------------------------------------  void CNSGAII::Solve(CPopulation\* solutions, const BProblem& problem)  {  CIndividual::SetTargetProblem(problem);  size\_t PopSize = 50;  while (PopSize % 4) PopSize += 1;  //CPopulation pop[2] = { CPopulation(PopSize) };  std::vector<CPopulation> pop(2, CPopulation(PopSize));  CSimulatedBinaryCrossover SBX(pc\_, eta\_c\_);  CPolynomialMutation PolyMut(1.0 / problem.num\_variables(), eta\_m\_);  Gnuplot gplot;  int cur = 0, next = 1;  //std::cout << problem.lower\_bounds()[0] << '\n';  RandomInitialization(&pop[cur], problem);  for (size\_t i = 0; i < PopSize; i += 1)  {  problem.Evaluate(&pop[cur][i]);  }  for (size\_t t = 0; t < gen\_num\_; t += 1)  {  pop[cur].resize(PopSize \* 2);  for (size\_t i = 0; i < PopSize; i += 2)  {  int father = rand() % PopSize,  mother = rand() % PopSize;  SBX(&pop[cur][PopSize + i], &pop[cur][PopSize + i + 1], pop[cur][father], pop[cur][mother]);  PolyMut(&pop[cur][PopSize + i]);  PolyMut(&pop[cur][PopSize + i + 1]);  problem.Evaluate(&pop[cur][PopSize + i]);  problem.Evaluate(&pop[cur][PopSize + i + 1]);  }  SurvivorSelection(&pop[next], &pop[cur], PopSize);  //ShowPopulation(gplot, pop[next], "pop"); Sleep(10);  std::swap(cur, next);  }  \*solutions = pop[cur];  }  // #include "comparator.h"  // #include "individual.h"  // ----------------------------------------------------------------------------------  CParetoDominate ParetoDominate;  // ----------------------------------------------------------------------------------  // CParetoDominate  // ----------------------------------------------------------------------------------  bool CParetoDominate::operator()(const CIndividual& l, const CIndividual& r) const  {  bool better = false;  for (size\_t f = 0; f < l.objs().size(); f += 1)  {  if (l.objs()[f] > r.objs()[f])  return false;  else if (l.objs()[f] < r.objs()[f])  better = true;  }  return better;  }// CParetoDominate::operator()  // ----------------------------------------------------------------------------------  #include <vector>  #include <limits>  // #include "math\_aux.h"  using namespace std;  namespace MathAux  {  // ----------------------------------------------------------------------  // ASF: Achivement Scalarization Function  // ----------------------------------------------------------------------  double ASF(const vector<double> &objs, const vector<double> &weight)  {  double max\_ratio = -numeric\_limits<double>::max();  for (size\_t f=0; f<objs.size(); f+=1)  {  double w = weight[f]?weight[f]:0.00001;  max\_ratio = std::max(max\_ratio, objs[f]/w);  }  return max\_ratio;  }  // ---------------------------------------------------------------------  }// namespace MathAux  // #include "individual.h"  // #include "base\_problem.h"  using std::size\_t;  const BProblem\* CIndividual::target\_problem\_ = 0;  // ----------------------------------------------------------------------  CIndividual::CIndividual(std::size\_t num\_vars, std::size\_t num\_objs) :  variables\_(num\_vars),  objectives\_(num\_objs),  converted\_objectives\_(num\_objs)  {  if (target\_problem\_ != 0)  {  variables\_.resize(target\_problem\_->num\_variables());  objectives\_.resize(target\_problem\_->num\_objectives());  converted\_objectives\_.resize(target\_problem\_->num\_objectives());  }  }  // ----------------------------------------------------------------------  const BProblem& CIndividual::TargetProblem() { return \*target\_problem\_; }  // ----------------------------------------------------------------------  std::ostream& operator << (std::ostream& os, const CIndividual& indv)  {  for (size\_t i = 0; i < indv.vars().size(); i += 1)  {  os << indv.vars()[i] << ' ';  }  os << " => ";  for (size\_t f = 0; f < indv.objs().size(); f += 1)  {  os << indv.objs()[f] << ' ';  }  return os;  }  // ----------------------------------------------------------------------  // #include "initialization.h"  // #include "base\_problem.h"  // #include "individual.h"  // #include "population.h"  // #include "math\_aux.h"  #include <cstddef>  using std::size\_t;  CRandomInitialization RandomInitialization;  void CRandomInitialization::operator()(CIndividual\* indv, const BProblem& prob) const  {  CIndividual::TDecVec& x = indv->vars();  x.resize(prob.num\_variables());  for (size\_t i = 0; i < x.size(); i += 1)  {  x[i] = MathAux::random(prob.lower\_bounds()[i], prob.upper\_bounds()[i]);  }  }  // ----------------------------------------------------------------------  void CRandomInitialization::operator()(CPopulation\* pop, const BProblem& prob) const  {  for (size\_t i = 0; i < pop->size(); i += 1)  {  (\*this)(&(\*pop)[i], prob);  }  }  // #include "crossover.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cmath>  #include <algorithm>  #include <cstddef>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function realcross() in crossover.c  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ----------------------------------------------------------------------  double CSimulatedBinaryCrossover::get\_betaq(double rand, double alpha, double eta) const  {  double betaq = 0.0;  if (rand <= (1.0 / alpha))  {  betaq = std::pow((rand \* alpha), (1.0 / (eta + 1.0)));  }  else  {  betaq = std::pow((1.0 / (2.0 - rand \* alpha)), (1.0 / (eta + 1.0)));  }  return betaq;  }  // ----------------------------------------------------------------------  bool CSimulatedBinaryCrossover::operator()(CIndividual\* child1,  CIndividual\* child2,  const CIndividual& parent1,  const CIndividual& parent2,  double cr,  double eta) const  {  \*child1 = parent1;  \*child2 = parent2;  if (MathAux::random(0.0, 1.0) > cr) return false; // not crossovered  CIndividual::TDecVec& c1 = child1->vars(), & c2 = child2->vars();  const CIndividual::TDecVec& p1 = parent1.vars(), & p2 = parent2.vars();  for (size\_t i = 0; i < c1.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) > 0.5) continue; // these two variables are not crossovered  if (std::fabs(p1[i] - p2[i]) <= MathAux::EPS) continue; // two values are the same  double y1 = std::min(p1[i], p2[i]),  y2 = std::max(p1[i], p2[i]);  double lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double rand = MathAux::random(0.0, 1.0);  // child 1  double beta = 1.0 + (2.0 \* (y1 - lb) / (y2 - y1)),  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  double betaq = get\_betaq(rand, alpha, eta);  c1[i] = 0.5 \* ((y1 + y2) - betaq \* (y2 - y1));  // child 2  beta = 1.0 + (2.0 \* (ub - y2) / (y2 - y1));  alpha = 2.0 - std::pow(beta, -(eta + 1.0));  betaq = get\_betaq(rand, alpha, eta);  c2[i] = 0.5 \* ((y1 + y2) + betaq \* (y2 - y1));  // boundary checking  c1[i] = std::min(ub, std::max(lb, c1[i]));  c2[i] = std::min(ub, std::max(lb, c2[i]));  if (MathAux::random(0.0, 1.0) <= 0.5)  {  std::swap(c1[i], c2[i]);  }  }  return true;  }// CSimulatedBinaryCrossover  // #include "mutation.h"  // #include "individual.h"  // #include "math\_aux.h"  // #include "base\_problem.h"  #include <cstddef>  #include <algorithm>  using std::size\_t;  // ----------------------------------------------------------------------  // The implementation was adapted from the code of function real\_mutate\_ind() in mutation.c in  // http://www.iitk.ac.in/kangal/codes/nsga2/nsga2-gnuplot-v1.1.6.tar.gz  //  // ref: http://www.slideshare.net/paskorn/simulated-binary-crossover-presentation#  // ---------------------------------------------------------------------  bool CPolynomialMutation::operator()(CIndividual\* indv, double mr, double eta) const  {  //int j;  //double rnd, delta1, delta2, mut\_pow, deltaq;  //double y, yl, yu, val, xy;  bool mutated = false;  CIndividual::TDecVec& x = indv->vars();  for (size\_t i = 0; i < x.size(); i += 1)  {  if (MathAux::random(0.0, 1.0) <= mr)  {  mutated = true;  double y = x[i],  lb = CIndividual::TargetProblem().lower\_bounds()[i],  ub = CIndividual::TargetProblem().upper\_bounds()[i];  double delta1 = (y - lb) / (ub - lb),  delta2 = (ub - y) / (ub - lb);  double mut\_pow = 1.0 / (eta + 1.0);  double rnd = MathAux::random(0.0, 1.0), deltaq = 0.0;  if (rnd <= 0.5)  {  double xy = 1.0 - delta1;  double val = 2.0 \* rnd + (1.0 - 2.0 \* rnd) \* (std::pow(xy, (eta + 1.0)));  deltaq = std::pow(val, mut\_pow) - 1.0;  }  else  {  double xy = 1.0 - delta2;  double val = 2.0 \* (1.0 - rnd) + 2.0 \* (rnd - 0.5) \* (std::pow(xy, (eta + 1.0)));  deltaq = 1.0 - (std::pow(val, mut\_pow));  }  y = y + deltaq \* (ub - lb);  y = std::min(ub, std::max(lb, y));  x[i] = y;  }  }  return mutated;  }// CPolynomialMutation  // #include "comparator.h"  // #include "nondominated\_sort.h"  // #include "population.h"  using namespace std;  CNondominatedSort NondominatedSort(ParetoDominate);  // ----------------------------------------------------------------------  std::pair< std::vector<size\_t>,std::vector< CNondominatedSort::TFrontMembers >> CNondominatedSort::operator()(const CPopulation &pop) const  {  CNondominatedSort::TFronts fronts;  size\_t num\_assigned\_individuals = 0;  size\_t rank = 1;  vector<size\_t> indv\_ranks(pop.size(), 0);  while (num\_assigned\_individuals < pop.size())  {  CNondominatedSort::TFrontMembers cur\_front;  for (size\_t i=0; i<pop.size(); i+=1)  {  if (indv\_ranks[i] > 0) continue; // already assigned a rank  bool be\_dominated = false;  for (size\_t j=0; j<cur\_front.size(); j+=1)  {  if ( dominate(pop[ cur\_front[j] ], pop[i]) ) // i is dominated  {  be\_dominated = true;  break;  }  else if ( dominate(pop[i], pop[ cur\_front[j] ]) ) // i dominates a member in the current front  {  cur\_front.erase(cur\_front.begin()+j);  j -= 1;  }  }  if (!be\_dominated)  {  cur\_front.push\_back(i);  }  }  for (size\_t i=0; i<cur\_front.size(); i+=1)  {  indv\_ranks[ cur\_front[i] ] = rank;  }  fronts.push\_back(cur\_front);  num\_assigned\_individuals += cur\_front.size();    rank += 1;  }  return { indv\_ranks,fronts };  }// CNondominatedSort::operator()  // ----------------------------------------------------------------------  // #include "crowding\_distance.h"  #include <iostream>  #include <limits>  #include <algorithm>  std::vector<double> CalculateCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts)  {  const size\_t num\_objs = cur[0].objs().size();  const size\_t num\_individuals = cur.size();  const size\_t num\_fronts = fronts.size();  std::vector<double> crowding\_distance(num\_individuals, 0.0);  for (size\_t f = 0; f < num\_objs; ++f)  {  for (size\_t i = 0; i < num\_fronts; ++i)  {  const size\_t front\_size = fronts[i].size();  if (front\_size <= 2)  {  // Skip fronts with 2 or fewer individuals as their crowding distance will be infinity  crowding\_distance[fronts[i][0]]=std::numeric\_limits<double>::infinity();  if(front\_size==2)crowding\_distance[fronts[i][1]]=std::numeric\_limits<double>::infinity();  continue;  }  std::sort(fronts[i].begin(), fronts[i].end(), [&](size\_t a, size\_t b) {  return cur[a].objs()[f] < cur[b].objs()[f];  });  crowding\_distance[fronts[i][0]] = crowding\_distance[fronts[i][front\_size - 1]] = std::numeric\_limits<double>::infinity();  const double obj\_range = cur[fronts[i][front\_size - 1]].objs()[f] - cur[fronts[i][0]].objs()[f];  for (size\_t j = 1; j < front\_size - 1; ++j)  {  const size\_t indv\_index = fronts[i][j];  crowding\_distance[indv\_index] += (cur[fronts[i][j + 1]].objs()[f] - cur[fronts[i][j - 1]].objs()[f]) / obj\_range;  }  }  }  return crowding\_distance;  }  void SortBasedOnCrowdingDistance(CPopulation& cur, CNondominatedSort::TFronts& fronts)  {  std::vector<double> crowding\_distance = CalculateCrowdingDistance(cur, fronts);  // Sort individuals within each front based on crowding distance  for (size\_t i = 0; i < fronts.size(); ++i)  {  CNondominatedSort::TFrontMembers& front = fronts[i];  std::sort(front.begin(), front.end(), [&](size\_t a, size\_t b) {  return crowding\_distance[a] > crowding\_distance[b]; // Sort in descending order of crowding distance  });  }    }  // #include "survivor\_selection.h"  // #include "population.h"  // #include "math\_aux.h"  // #include "nondominated\_sort.h"  // #include "crowding\_distance.h"  #include <limits>  #include <algorithm>  using namespace std;  // ----------------------------------------------------------------------  // SurvivorSelection():  // ----------------------------------------------------------------------  void SurvivorSelection(CPopulation \*pnext, CPopulation \*pcur, size\_t PopSize)  {  CPopulation &cur = \*pcur, &next = \*pnext;  next.clear();  // ---------- Step 4 in Algorithm 1: non-dominated sorting ----------  CNondominatedSort::TFronts fronts = NondominatedSort(cur).second;    SortBasedOnCrowdingDistance(cur, fronts);  for (size\_t t = 0; t < fronts.size() - 1; t += 1)  {  for (size\_t i = 0; i < fronts[t].size(); i += 1) {  if (next.size() >= PopSize) break;  next.push\_back(cur[fronts[t][i]]);  }  }  }  // ----------------------------------------------------------------------  // #include "gnuplot\_interface.h"  #include <iostream>  #include <sstream>  using namespace std;  // ---------------------------------------------------------  // Ref:  // http://user.frdm.info/ckhung/b/ma/gnuplot.php  // ---------------------------------------------------------  Gnuplot::Gnuplot()  {  // with -persist option you will see the windows as your program ends  //gnuplotpipe=\_popen("gnuplot -persist","w");  //without that option you will not see the window  // because I choose the terminal to output files so I don't want to see the window  gnuplotpipe = \_popen("gnuplot", "w");  if (!gnuplotpipe)  {  cerr << ("Gnuplot not found !");  }  }  // ---------------------------------------------------------  Gnuplot::~Gnuplot()  {  fprintf(gnuplotpipe, "exit\n");  \_pclose(gnuplotpipe);  }  // ---------------------------------------------------------  void Gnuplot::operator()(const string& command)  {  fprintf(gnuplotpipe, "%s\n", command.c\_str());  fflush(gnuplotpipe);  // flush is necessary, nothing gets plotted else  }  // ---------------------------------------------------------  void Gnuplot::set\_title(const std::string& title)  {  ostringstream ss;  ss << "set title \"" << title << "\"";  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::plot(const std::string& fname, std::size\_t x, std::size\_t y)  {  ostringstream ss;  ss << "plot \"" << fname << "\" using " << x << ":" << y;  operator()(ss.str());  }  // ---------------------------------------------------------  void Gnuplot::splot(const std::string& fname, std::size\_t x, std::size\_t y, std::size\_t z)  {  ostringstream ss;  ss << "splot \"" << fname << "\" using " << x << ":" << y << ":" << z;  operator()(ss.str());  }  // ---------------------------------------------------------  // #include "assignment\_problem.h"  // #include "individual.h"  // #include "math\_aux.h"  #include <cmath>  #include <vector>  #include<iostream>  using std::size\_t;  using std::cos;  // ----------------------------------------------------------------------  // CProblemAssignment  // ----------------------------------------------------------------------  CProblemAssignment::CProblemAssignment(size\_t M, size\_t k, const std::string& name, double lbs, double ubs) :  BProblem(name),  M\_(M),  k\_(k)  {  lbs\_.resize( k\_, lbs); // lower bound  ubs\_.resize( k\_, ubs); // upper bound  }  bool CProblemAssignment5::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  for (size\_t i = 1; i < k\_; ++i) {  f[0] += -10 \* exp(-0.2 \* sqrt(MathAux::square(x[i - 1]) + MathAux::square(x[i])));  }  for (size\_t i = 0; i < k\_; ++i) {  f[1] += pow(abs(x[i]), 0.8) + 5 \* sin(pow(x[i], 3));  }    return true;  }  bool CProblemAssignment6::Evaluate(CIndividual\* indv) const  {  CIndividual::TDecVec& x = indv->vars();  CIndividual::TObjVec& f = indv->objs();  //std::cout << (x.size() == k\_)<<'\n';  if (x.size() != k\_) return false; // #variables does not match  f.resize(M\_, 0);  f[0] = MathAux::square(x[0]);  f[1] = MathAux::square(x[0] - 2);  return true;  }  // #include "log.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  #include <fstream>  #include<iostream>  using namespace std;  #define OUTPUT\_DECISION\_VECTOR  bool SaveToFile(const std::string& fname, const CPopulation& pop, ios\_base::openmode mode)  {  ofstream ofile(fname.c\_str(), mode);  if (!ofile) return false;  for (size\_t i = 0; i < pop.size(); i += 1)  {  #ifdef OUTPUT\_DECISION\_VECTOR  for (size\_t j = 0; j < pop[i].vars().size(); j += 1)  {  ofile << pop[i].vars()[j] << ' ';  }  #endif  for (size\_t f = 0; f < pop[i].objs().size(); f += 1)  {  ofile << pop[i].objs()[f] << ' ';  }  ofile << endl;  }  ofile << endl;  return true;  }  // ----------------------------------------------------------------------  bool ShowPopulation(Gnuplot& gplot, const CPopulation& pop, const std::string& legend)  {  if (!SaveToFile(legend, pop, ios\_base::out)) return false;  size\_t n = 0;  #ifdef OUTPUT\_DECISION\_VECTOR  n = pop[0].vars().size();  #endif    if (pop[0].objs().size() == 2)  {  gplot.plot(legend, n + 1, n + 2);  return true;  }  else if (pop[0].objs().size() == 3)  {  gplot.splot(legend, n + 1, n + 2, n + 3);  return true;  }  else  return false;  }  // #include "assignment\_problem.h"  // #include "nsgaii.h"  // #include "population.h"  // #include "gnuplot\_interface.h"  // #include "log.h"  #include <ctime>  #include <cstdlib>  #include <iostream>  // #include "individual.h"  // #include "math\_aux.h"  using namespace std;  int main()  {  CNSGAII nsgaii("nsgaii");  CPopulation solutions;  Gnuplot gplot;  const size\_t NumRuns = 10;  BProblem\* problem5 = new CProblemAssignment5(2, 3,-5,5);  BProblem\* problem6 = new CProblemAssignment6(2, 1,-10,10);  BProblem\* problem = problem6;    for (size\_t r = 0; r < NumRuns; r += 1)  {  srand(r); cout << "Run Number: " << r << endl;  nsgaii.Solve(&solutions, \*problem);  SaveToFile(nsgaii.name() + "-" + problem->name() + ".txt", solutions);  ShowPopulation(gplot, solutions, "pop"); // system("pause");  }  delete problem;  return 0;  }  Input: Terminal Input is not required Everything is Automated. If you want to change the parameters  please update the main function and nsgaii function.  You can also add nsgaii.ini file and write parameter values.  Output:  Values of F1 and F2 after Max iteration      The Values after the max iteration (x, f1, f2)      Hands On calculations:  Number of populations =4  Tournament parent selection  SBX crossover  Polynomial Mutation  Survivor selection Based on Crowding Distance and Rank  Generation Number =1  Parent selection   |  |  |  |  | | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | | 1 | -9.97681 | 99.5367 | 143.444 | | 2 | -5.28855 | 27.9688 | 53.123 | | 3 | 2.96304 | 8.77962 | 0.92745 | | 4 | -8.51253 | 72.4631 | 110.513 |     Crossover     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Father | Mother |  | c1 |  | c2 | |  |  |  | x1 |  | x1 | | 4 | 2 | 1 | -8.51253 | 2 | -5.28855 | | 3 | 2 | 3 | -5.20879 | 4 | 2.88328 |   Mutation   |  |  |  |  | | --- | --- | --- | --- | |  | Mutated Value |  |  | | Sr. No. | x1 | f1(x) | f2(x) | | 1 | -6.67101 | 44.5024 | 75.1864 | | 2 | -4.12765 | 17.0375 | 37.5481 | | 3 | -5.31273 | 28.2251 | 53.476 | | 4 | 5.4853 | 30.0885 | 12.1473 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | Rank | Crowding Distance | Next Generation | | 1 | -9.97681 | 8.77962 | 0.92745 | 7 | inf | 3 | | 2 | -5.28855 | 17.0375 | 37.5481 | 3 | inf | 6 | | 3 | 2.96304 | 30.0885 | 12.1473 | 1 | inf | 8 | | 4 | -8.51253 | 27.9688 | 53.123 | 6 | inf | 2 | | 5 | -6.67101 | 75.0785 | 44.4194 | 5 | inf |  | | 6 | -4.12765 | 70.0454 | 107.523 | 2 | inf |  | | 7 | -5.31273 | 7.37553 | 0.51236 | 4 | inf |  | | 8 | 5.4853 | 61.6146 | 97.0125 | 2 | inf |  |   Generation Number =2  Parent selection     |  |  |  |  | | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | | 1 | 2.96304 | 8.77962 | 0.92745 | | 2 | -4.12765 | 17.0375 | 37.5481 | | 3 | 5.4853 | 30.0885 | 12.1473 | | 4 | -5.28855 | 27.9688 | 53.123 |   Crossover   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Father | Mother |  | c1 |  | c2 | |  |  |  | x1 |  | x1 | | 4 | 3 | 1 | 5.74123 | 2 | -5.54449 | | 1 | 4 | 3 | 2.96304 | 4 | -5.28855 |   Mutation   |  |  |  |  | | --- | --- | --- | --- | |  | Mutated Value |  |  | | Sr. No. | x1 | f1(x) | f2(x) | | 1 | 8.66478 | 75.0785 | 44.4194 | | 2 | -8.36931 | 70.0454 | 107.523 | | 3 | 2.71579 | 7.37553 | 0.51236 | | 4 | -7.84949 | 61.6146 | 97.0125 |   Survivor Selection   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Sr. No. | x1 | f1(x) | f2(x) | Rank | Crowding Distance | Next Generation | | 1 | -9.97681 | 8.77962 | 0.92745 | 2 | inf | 7 | | 2 | -5.28855 | 17.0375 | 37.5481 | 3 | inf | 1 | | 3 | 2.96304 | 30.0885 | 12.1473 | 3 | inf | 2 | | 4 | -8.51253 | 27.9688 | 53.123 | 4 | inf | 3 | | 5 | 8.66478 | 75.0785 | 44.4194 | 4 | inf |  | | 6 | -8.36931 | 70.0454 | 107.523 | 6 | inf |  | | 7 | 2.71579 | 7.37553 | 0.51236 | 1 | inf |  | | 8 | -7.84949 | 61.6146 | 97.0125 | 5 | inf |  |   Best Answer: x = [2.71579] (Randomly selected from pareto-optimal front) |

