openGA user manual

A free C++ Genetic Algorithm library
Version: 1.0.2

Arash Mohammadi, *Institute for Intelligent Systems Research and Innovation (IISRI)*
arash.m at research.deakin.edu.au

This user documentation provides guidelines for using openGA , a free C++ library for Genetic Algorithm (GA) based optimization. This library is potential to run GA in single objective, multi-objective and interactive modes. OpenGA is highly flexible to customizations of users and avoids many limitations which MATLAB is currently suffering. The crossover and mutation operations are entirely under the control of the designer. Genes are not required to be presented via a vector. The evaluation of chromosomes are performed via multithreading implementation by default and the user can reject a chromosome after its evaluation.

Introduction

Motivation and aims

Despite currently there are some available C++ based GA libraries, the following raised my motivation to write this new GA library

- This code is aiming for GA problems when cost functions are expensive in terms of calculation. A thread pool is used to increase the speed of GA for chromosome evaluations.

- Separation of middle computation and final cost. Unfortunately, in many applications, the chromosomes need to run heavy simulations and they have to store valuable additional information into the cost beyond the final cost(s). This library, opens the hand of the designers to write their customized middle costs. Then, converting them to the final cost at the last stage.
- Rejection of chromosomes after computation. Another problem of many GA libraries including MATLAB is that they have a nonlinear condition, while the condition is not known unless heavy computation is performed. This computation can be no longer used for evaluation. In this GA library, the user code is able to reject a chromosome even after its heavy evaluation computation. In another term, the nonlinear condition can be mixed with evaluation.
- Flexible crossover and mutation. One of the problem of many GA libraries including MATLAB is to perform naive crossover and mutation. In this library, performing crossover and mutation is totally up to the opinion of the user.

User side code

The user side codes consist of the following

Settings: population, maximum generation number, etc.

Genes definition: chromosome data structure.

Middle cost definition: A temporary variable storing the results of related simulations. However, it needs to be finalized to be used as final objective(s).

Mapping genes: Generating genes from a given randomization function.

Evaluations: Genes are evaluated and converted into middle costs.

Mutation function: A function for custom genetic mutations. A scale called `shrink_scale` assist the programmer to confine the mutation range as the number of generations grow. This scale can be optionally defined by user via `set_shrink_scale` function.

Crossover function: A function for custom genetic crossover.

Total fitness: A function to summarize the middle cost to the final cost.

Report generation: A function to show/store the results of each generation.

Requirements

In the recent edition of openGA, the standard C++ libraries are sufficient and there is no dependency to any external library. Therefore the users can run openGA straightaway.

Compiler options: `-O3 -s -DNDEBUG -std=c++11 -pthread`

Linker options: `-pthread`

Settings

Optimization mode

The main setting is related to the problem mode which provides the following options:

- **`GA_MODE::SOGA`**: Single objective genetic algorithm
- **`GA_MODE::IGA`**: Interactive (single objective) genetic algorithm
- **`GA_MODE::NSGA_III`**: Multi-objective genetic algorithm (Nondominated sorting GA III)

Multithreading

Multithreading can improve or degrade the performance of optimization. It should be noted that threads impose extra overheads. The number of threads can be adjusted via `N_threads`. By default, this parameter is approximately equal to the supported hardware concurrent threads. In the multithreading mode, by default, each thread is given a chromosome to evaluate when they are free. If the evaluation process is very fast, it is more efficient that the thread responsibilities are divided at the beginning. This adjustment is possible by setting `dynamic_threading` to `false`.

	SO	IGA	MO	Type	Default
problem_mode	✓	✓	✓	enum class	SOGA
multi_threading	✓	-	✓	bool	true
dynamic_threading	✓	-	✓	bool	true
N_threads	✓	-	✓	int	CPU cores
verbose	✓	✓	✓	bool	false
population	✓	✓	✓	uint	50
generation_max	✓	✓	✓	int	100
calculate_SO_total_fitness	✓	-	-	function	nullptr
calculate_IGA_total_fitness	-	✓	-	function	nullptr
calculate_MO_objectives	-	-	✓	function	nullptr
distribution _objective_reductions	-	-	✓	function	nullptr
init_genes	✓	✓	✓	function	nullptr
eval_genes	✓	-	✓	function	nullptr
eval_genes_IGA	-	✓	-	function	nullptr
crossover	✓	✓	✓	function	nullptr
mutate	✓	✓	✓	function	nullptr
set_shrink_scale	✓	✓	✓	function	nullptr
SO_report_generation	✓	✓	-	function	nullptr
MO_report_generation	-	-	✓	function	nullptr
custom_refresh	✓	-	✓	function	nullptr
elite_count	✓	✓	-	int	5
crossover_fraction	✓	✓	✓	double	0.7
mutation_fraction	✓	✓	✓	double	0.3
idle_delay_us	✓	✓	✓	long	1000
tol_stall_average	✓	✓	-	double	1e-4
average_stall_max	✓	✓	-	int	10
tol_stall_best	✓	✓	-	double	1e-6
best_stall_max	✓	✓	-	int	10
reference_vector_divisions	-	-	✓	uint	10
enable_reference_vectors	-	-	✓	bool	true

Stop reason

Genetic Algorithm may stop because of one of the following reasons:

- **`StopReason::MaxGenerations :`** Reaching the maximum generation number.
- **`StopReason::StallAverage :`** The average cost has not changed more than `tol_stall_average` for `average_stall_max` generation steps (only for single objective or interactive GA)
- **`StopReason::StallBest :`** The average cost has not changed more than `tol_stall_best` for `best_stall_max` generation steps (only for single objective or interactive GA)
- **`StopReason::UserRequest :`** user has requested for stopping GA by setting `user_request_stop` to `true`.

Single Objective Optimization

In single objective GA, each chromosome evaluation will be finalized into a single cost value.

Multi-Objective Optimization

In multi-objective GA, each evaluation does not lead into only a single cost but multiple objectives to be minimized. Therefore, the output of optimization is not a single chromosome as the best solution, but a set of nondominated solutions called pareto-optimal solutions.

The applied multi-objective GA is based on NSGA-III proposed by [Deb and Jain, 2014] [Jain and Deb, 2014].

Interactive Genetic Algorithm

Interactive Genetic Algorithm (IGA) is similar to the conventional GA except for the cost function is evaluated via human subjectivity [Takagi, 2001]. Interactive Evolutionary Algorithms (IEA) have applications in art [Dalvandi et al., 2010], fashion design [Kim and Cho, 2000], music [Tokui et al., 2000], graphic arts [Lewis, 2008] and architecture [Serag et al., 2008].

In this libary, there are several considerations for IGA:

- It is assumed, IGA is involved in a heavy computation via `eval_genes_IGA`. This function has access to the previous evaluated genes in the same generation and it can make decision based on them. For example if a solution is too close to the available solutions, it can be rejected. This function is no called to evaluate the middle cost of the generation elites again.
- Human evaluation is assumed to be applied in `calculate_IGA_total_fitness` function. This function is supposed to evaluate the final cost of the entire new generation based on human subjectivity.
- Sum of these three terms have to be exactly equal to `1.0` : `crossover_fraction`, `mutation_fraction` and `elit_fraction=elite_count/population`.

License

This library is free and distributed under Mozilla Public License Version 2.0.

Contact author

Any suggestion, recommendation, bug report and question related to this library is highly welcome. I may be also interested in involving in bigger projects. I am Arash Mohammadi and you can contact me via email

arash.m at research.deakin.edu.au

The source code of this library is available online at
`https://github.com/Arash-codedev/openGA`
Download link:
`https://github.com/Arash-codedev/openGA/archive/v1.0.2.zip`

If you have found this library useful for your work, please cite the following paper [Mohammadi et al., 2017]

Mohammadi, Arash, et al. "openGA, a C++ Genetic Algorithm library." Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on. IEEE, 2017.

Bibtex:

```
@inproceedings{mohammadi2017openga,
title={openGA, a C++ Genetic Algorithm library},
author={Mohammadi, Arash and Houshyar Asadi, Shady Mohamed and Nelson, Kyle and
    Nahavandi, Saeid},
booktitle={Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference
    on},
pages={002051--002056},
year={2017},
organization={IEEE}
}
```

Sample codes

Single objective GA

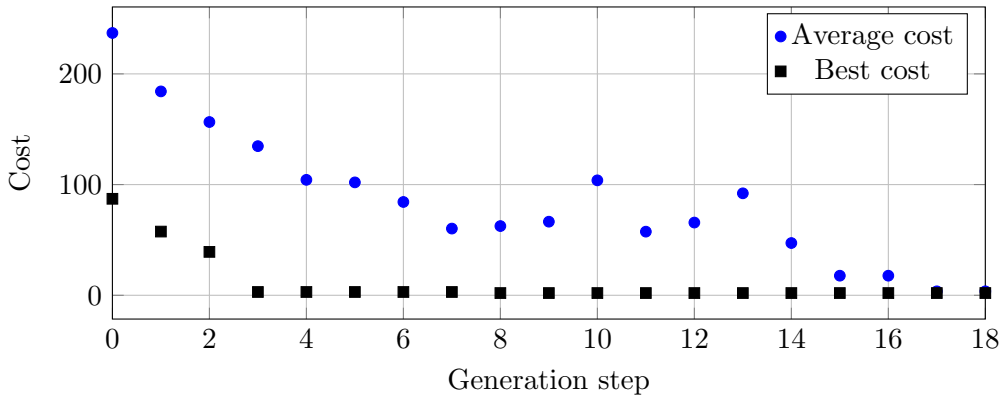


Figure 1: *Single objective GA convergence*

```
// This library is free and distributed under
// Mozilla Public License Version 2.0.

#include <string>
#include "genetic.hpp"
#include <fstream>

struct MyGenes
{
  double x;
  double y;

  std::string to_string() const
  {
    return
      "{x:"+std::to_string(x)+
      ", y:"+std::to_string(y)+
      "}";
  }
};

struct MyMiddleCost
{
  // This is where the results of simulation
  // is stored but not yet finalized.
  double cost_distance2;
  double cost_sqsin;
```

```
};

typedef EA::Genetic<MyGenes,MyMiddleCost> GA_Type;
typedef EA::GenerationType<MyGenes,MyMiddleCost> Generation_Type;

void init_genes(MyGenes& p,const std::function<double(void)> &rand)
{
  p.x=20.0*rand()-10.0;
  p.y=20.0*rand()-10.0;
}

bool eval_genes(
  const MyGenes& p,
  MyMiddleCost &c)
{
  double x=p.x;
  double y=p.y;
  // see the surface plot at:
  // https://academo.org/demos/3d-surface-plotter/?expression=x*x%2By
      *y%2B30.0*sin(x*100.0*sin(y)%2By*100.0*cos(x))%2B125%2B45.0*
      sqrt(x%2By)*sin((15.0*(x%2By))%2F(x*x%2By*y))&xRange=-10%2C%2
      B10&yRange=-10%2C%2B10&resolution=100
  //
  // the middle comupations of cost:
  if(x+y>0)
  {
    double predictable_noise=30.0*sin(x*100.0*sin(y)+y*100.0*cos(x));
    c.cost_distance2=x*x+y*y+predictable_noise;
    c.cost_sqsin=125+45.0*sqrt(x+y)*sin((15.0*(x+y))/(x*x+y*y));
    return true; // genes are accepted
  }
  else
    return false; // genes are rejected
}

MyGenes mutate(
  const MyGenes& X_base,
  const std::function<double(void)> &rand,
  double shrink_scale)
{
  MyGenes X_new;
  bool in_range_x,in_range_y;
  double loca_scale=shrink_scale;
  if(rand()<0.4)
    loca_scale*=loca_scale;
  else if(rand()<0.1)
    loca_scale=1.0;
  do{
    X_new=X_base;
```

```
    X_new.x+=0.2*(rand()-rand())*loca_scale;
    X_new.y+=0.2*(rand()-rand())*loca_scale;
    in_range_x= (X_new.x>=-10.0 && X_new.x<10.0);
    in_range_y= (X_new.y>=-10.0 && X_new.y<10.0);
  } while(!in_range_x || !in_range_y);
  return X_new;
}

MyGenes crossover(
  const MyGenes& X1,
  const MyGenes& X2,
  const std::function<double(void)> &rand)
{
  MyGenes X_new;
  double r;
  r=rand();
  X_new.x=r*X1.x+(1.0-r)*X2.x;
  r=rand();
  X_new.y=r*X1.y+(1.0-r)*X2.y;
  return X_new;
}

double calculate_SO_total_fitness(const GA_Type::thisChromosomeType &
    X)
{
  // finalize the cost
  double cost1,cost2;
  cost1=X.middle_costs.cost_distance2;
  cost2=X.middle_costs.cost_sqsin;
  return cost1+cost2;
}

std::ofstream output_file;

void SO_report_generation(
  int generation_number,
  const EA::GenerationType<MyGenes,MyMiddleCost> &last_generation,
  const MyGenes& best_genes)
{
  std::cout
    <<"Generation ["<<generation_number<<"], "
    <<"Best="<<last_generation.best_total_cost<<", "
    <<"Average="<<last_generation.average_cost<<", "
    <<"Best genes=("<<best_genes.to_string()<<")"<<", "
    <<"Exe_time="<<last_generation.exe_time
    <<std::endl;

  output_file
    <<generation_number<<"\t"
```

```
    <<best_genes.x<<"\t"
    <<best_genes.y<<"\t"
    <<last_generation.average_cost<<"\t"
    <<last_generation.best_total_cost<<"\n";
}

int main()
{
  output_file.open("./bin/result_so1.txt");
  output_file<<"step"<<"\t"<<"x_best"<<"\t"<<"y_best"<<"\t"<<"
      cost_avg"<<"\t"<<"cost_best"<<"\n";

  EA::Chronometer timer;
  timer.tic();

  GA_Type ga_obj;
  ga_obj.problem_mode= EA::GA_MODE::SOGA;
  ga_obj.multi_threading=true;
  ga_obj.idle_delay_us=1; // switch between threads quickly
  ga_obj.verbose=false;
  ga_obj.population=20;
  ga_obj.generation_max=1000;
  ga_obj.calculate_SO_total_fitness= calculate_SO_total_fitness;
  ga_obj.init_genes= init_genes;
  ga_obj.eval_genes= eval_genes;
  ga_obj.mutate= mutate;
  ga_obj.crossover= crossover;
  ga_obj.SO_report_generation= SO_report_generation;
  ga_obj.best_stall_max=10;
  ga_obj.elite_count=10;
  ga_obj.crossover_fraction=0.7;
  ga_obj.mutation_rate=0.4;
  ga_obj.solve();

  std::cout<<"The problem is optimized in "<<timer.toc()<<" seconds."
      <<std::endl;

  output_file.close();
  return 0;
}
```

Multi-objective GA

```
// This library is free and distributed under
// Mozilla Public License Version 2.0.

#include <string>
```

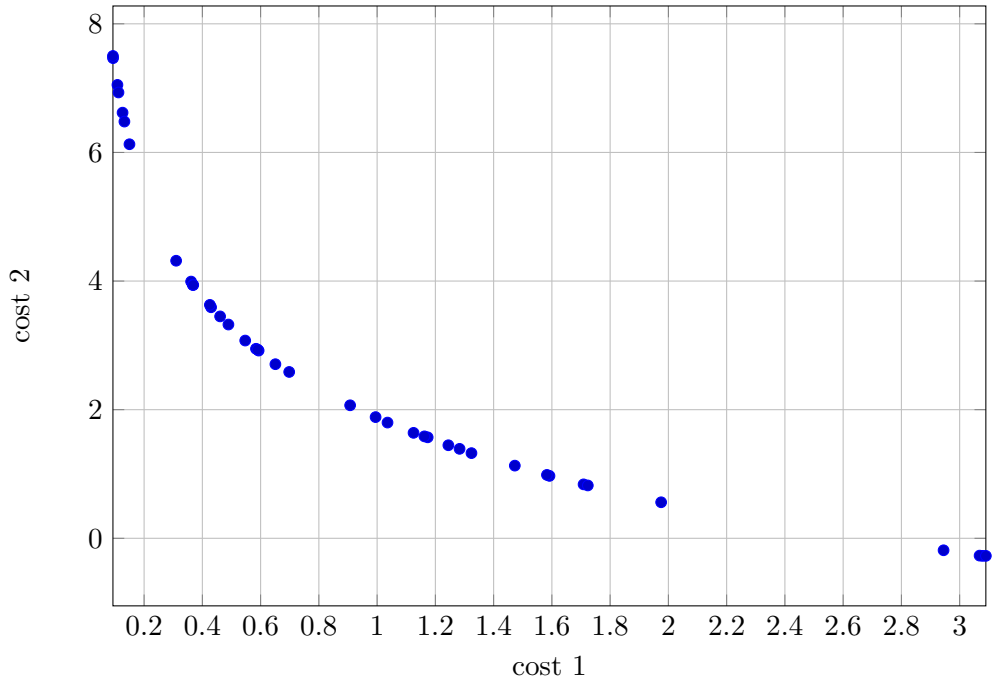


Figure 2: *The MO-GA pareto-front*

```
#include <iostream>
#include <fstream>
#include "genetic.hpp"

struct MyGenes
{
  double x;
  double y;

  std::string to_string() const
  {
    return
      "{x:"+std::to_string(x)+
      ", y:"+std::to_string(y)+
      "}";
  }
};

struct MyMiddleCost
{
  // This is where the results of simulation
  // is stored but not yet finalized.
```

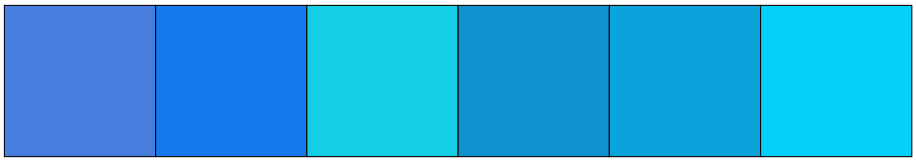
```
  double cost_A;
  double cost_B;
};

typedef EA::Genetic<MyGenes,MyMiddleCost> GA_Type;
typedef EA::GenerationType<MyGenes,MyMiddleCost> Generation_Type;

void init_genes(MyGenes& p,const std::function<double(void)> &rand)
{
  p.x=10.0*rand();
  p.y=10.0*rand();
}

bool eval_genes(
  const MyGenes& p,
  MyMiddleCost &c)
{
  double x=p.x;
  double y=p.y;
  // the middle comupations of cost:
  c.cost_A=log(1.0+x*sqrt(x*y));
  c.cost_B=98.0-100.0*(1.0-1.0/(1.0+y*sqrt(x*y)));
  return true; // genes are accepted
}

MyGenes mutate(
  const MyGenes& X_base,
  const std::function<double(void)> &rand,
  double shrink_scale)
{
  MyGenes X_new;
  double loca_scale=shrink_scale;
  if(rand()<0.4)
    loca_scale*=loca_scale;
  else if(rand()<0.1)
    loca_scale=1.0;
  bool in_range_x,in_range_y;
  double local_scale=shrink_scale;
  if(rand()<0.4)
    local_scale*=local_scale;
  else if(rand()<0.1)
    local_scale=1.0;
  do{
    X_new=X_base;
    X_new.x+=0.2*(rand()-rand())*local_scale;
    X_new.y+=0.2*(rand()-rand())*local_scale;
    in_range_x= (X_new.x>=0.0 && X_new.x<10.0);
    in_range_y= (X_new.y>=0.0 && X_new.y<10.0);
  } while(!in_range_x || !in_range_y);
```

```
  return X_new;
}

MyGenes crossover(
  const MyGenes& X1,
  const MyGenes& X2,
  const std::function<double(void)> &rand)
{
  MyGenes X_new;
  double r;
  r=rand();
  X_new.x=r*X1.x+(1.0-r)*X2.x;
  r=rand();
  X_new.y=r*X1.y+(1.0-r)*X2.y;
  return X_new;
}

std::vector<double> calculate_MO_objectives(const GA_Type::
    thisChromosomeType &X)
{
  return {
    X.middle_costs.cost_A,
    X.middle_costs.cost_B
  };
}

std::vector<double> distribution_objective_reductions(const std::
    vector<double> &objs)
{
  return objs;
}

void MO_report_generation(
  int generation_number,
  const EA::GenerationType<MyGenes,MyMiddleCost> &last_generation,
  const std::vector<unsigned int>& pareto_front)
{
  (void) last_generation;

  std::cout<<"Generation ["<<generation_number<<"], ";
  std::cout<<"Pareto-Front {";
  for(unsigned int i=0;i<pareto_front.size();i++)
  {
    std::cout<<(i>0?",":"");
    std::cout<<pareto_front[i];
  }
  std::cout<<"}"<<std::endl;
}
```

```
void save_results(const GA_Type &ga_obj)
{
  std::ofstream output_file;
  output_file.open("./bin/result_mo1.txt");
  output_file<<"N"<<"\t"<<"x"<<"\t"<<"y"<<"\t"<<"cost1"<<"\t"<<"cost2
      "<<"\n";
  std::vector<unsigned int> paretofront_indices=ga_obj.
      last_generation.fronts[0];
  for(unsigned int i:paretofront_indices)
  {
    const auto &X=ga_obj.last_generation.chromosomes[i];
    output_file
      <<i<<"\t"
      <<X.genes.x<<"\t"
      <<X.genes.y<<"\t"
      <<X.middle_costs.cost_A<<"\t"
      <<X.middle_costs.cost_B<<"\n";

  }
  output_file.close();
}

int main()
{
  EA::Chronometer timer;
  timer.tic();

  GA_Type ga_obj;
  ga_obj.problem_mode= EA::GA_MODE::NSGA_III;
  ga_obj.multi_threading=true;
  ga_obj.idle_delay_us=1; // switch between threads quickly
  ga_obj.verbose=false;
  ga_obj.population=40;
  ga_obj.generation_max=100;
  ga_obj.calculate_MO_objectives= calculate_MO_objectives;
  ga_obj.init_genes=init_genes;
  ga_obj.eval_genes=eval_genes;
  ga_obj.distribution_objective_reductions=
      distribution_objective_reductions;
  ga_obj.mutate=mutate;
  ga_obj.crossover=crossover;
  ga_obj.MO_report_generation=MO_report_generation;
  ga_obj.crossover_fraction=0.7;
  ga_obj.mutation_rate=0.4;
  ga_obj.solve();

  std::cout<<"The problem is optimized in "<<timer.toc()<<" seconds."
      <<std::endl;
```

```
  save_results(ga_obj);
  return 0;
}
```

Interactive GA



```
// This library is free and distributed under
// Mozilla Public License Version 2.0.

#include <string>
#include "genetic.hpp"
#include "gui.hpp"
#include <fstream>

struct MyGenes
{
  double R,G,B;

  std::string to_string() const
  {
    const unsigned red = (unsigned)R;
    const unsigned green = (unsigned)G;
    const unsigned blue = (unsigned)B;
    char hexstr[16];
    snprintf(hexstr,sizeof(hexstr),"%02x%02x%02x",red,green,blue);
    std::string retstr=hexstr;
    return retstr;
  }
};

struct MyMiddleCost
{
  double R,G,B;
  double cost_user_score;
};

typedef EA::Genetic<MyGenes,MyMiddleCost> GA_Type;
```

```
typedef EA::GenerationType<MyGenes,MyMiddleCost> Generation_Type;

void init_genes(MyGenes& p,const std::function<double(void)> &rand)
{
  p.R=255.0*rand();
  p.G=255.0*rand();
  p.B=255.0*rand();
}

bool eval_genes_IGA(
  const MyGenes& p,
  MyMiddleCost &c,
  const EA::GenerationType<MyGenes,MyMiddleCost>&)
{
  c.R=p.R;
  c.G=p.G;
  c.B=p.B;
  return true; // genes are accepted
}

MyGenes mutate(
  const MyGenes& X_base,
  const std::function<double(void)> &rand,
  double shrink_scale)
{
  MyGenes X_new;
  (void) shrink_scale;
  bool in_range_R,in_range_G,in_range_B;
  do{
    X_new=X_base;
    X_new.R+=100*(rand()-rand());
    X_new.G+=100*(rand()-rand());
    X_new.B+=100*(rand()-rand());
    in_range_R= (X_new.R>=0.0 && X_new.R<255.0);
    in_range_G= (X_new.G>=0.0 && X_new.G<255.0);
    in_range_B= (X_new.B>=0.0 && X_new.B<255.0);
  } while(!in_range_R || !in_range_G || !in_range_B);
  return X_new;
}

MyGenes crossover(
  const MyGenes& X1,
  const MyGenes& X2,
  const std::function<double(void)> &rand)
{
  MyGenes X_new;
  double r;
  r=rand();
  X_new.R=r*X1.R+(1.0-r)*X2.R;
```

```
  r=rand();
  X_new.G=r*X1.G+(1.0-r)*X2.G;
  r=rand();
  X_new.B=r*X1.B+(1.0-r)*X2.B;
  return X_new;
}

void calculate_IGA_total_fitness(GA_Type::thisGenerationType &g)
{


  for(unsigned int i=0;i<g.chromosomes.size();i++)
  {
    GA_Type::thisChromosomeType &X=g.chromosomes[i];
    // X.total_cost=100.0-X.middle_costs.cost_user_score;
    gui_subject_R=X.middle_costs.R;
    gui_subject_G=X.middle_costs.G;
    gui_subject_B=X.middle_costs.B;
    refresh_gui();
    refresh_gui();
    std::cout<<"How much do you like this ("<<X.genes.to_string()<<")
        blue color (0-100%)? ";
    std::cin>>X.middle_costs.cost_user_score;
    X.total_cost=100.0-X.middle_costs.cost_user_score;
  }
}

std::ofstream output_file;

void SO_report_generation(
  int generation_number,
  const EA::GenerationType<MyGenes,MyMiddleCost> &last_generation,
  const MyGenes& best_genes)
{
  std::cout
    <<"Generation ["<<generation_number<<"], "
    <<"Best="<<100.0-last_generation.best_total_cost<<", "
    <<"Average="<<100.0-last_generation.average_cost<<", "
    <<"Best genes=("<<best_genes.to_string()<<")"<<", "
    <<"Exe_time="<<last_generation.exe_time
    <<std::endl;

  output_file
    <<generation_number<<"\t"
    <<best_genes.to_string()<<"\t"
    <<100.0-last_generation.average_cost<<"\t"
    <<100.0-last_generation.best_total_cost<<"\n";
}
```

```
int main()
{
  output_file.open("./bin/result_iga1.txt");
  output_file<<"step"<<"\t"<<"color_best"<<"\t"<<"cost_avg"<<"\t"<<"
      cost_best"<<"\n";
  init_gui();

  GA_Type ga_obj;
  ga_obj.problem_mode= EA::GA_MODE::IGA;
  ga_obj.verbose=false;
  ga_obj.population=15;
  ga_obj.generation_max=20;
  ga_obj.calculate_IGA_total_fitness= calculate_IGA_total_fitness;
  ga_obj.init_genes= init_genes;
  ga_obj.eval_genes_IGA= eval_genes_IGA;
  ga_obj.mutate= mutate;
  ga_obj.crossover= crossover;
  ga_obj.SO_report_generation= SO_report_generation;
  ga_obj.elite_count=3;
  double non_elit_fraction=1-double(ga_obj.elite_count)/double(ga_obj
      .population);
  ga_obj.crossover_fraction=non_elit_fraction;
  ga_obj.mutation_rate=0.1;
  ga_obj.solve();

  output_file.close();
  return 0;
}
```

References

[Dalvandi et al., 2010] Dalvandi, A., Behbahani, P. A., and DiPaola, S. (2010). Exploring persian rug design using a computational evolutionary approach. In *EVA*.

[Deb and Jain, 2014] Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601.

[Jain and Deb, 2014] Jain, H. and Deb, K. (2014). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. *IEEE Trans. Evolutionary Computation*, 18(4):602–622.

[Kim and Cho, 2000] Kim, H.-S. and Cho, S.-B. (2000). Application of interactive genetic algorithm to fashion design. *Engineering applications of artificial intelligence*, 13(6):635–644.

[Lewis, 2008] Lewis, M. (2008). Evolutionary visual art and design. In *The art of artificial evolution*, pages 3–37. Springer.

[Mohammadi et al., 2017] Mohammadi, A., Houshyar Asadi, S. M., Nelson, K., and Nahavandi, S. (2017). openga, a c++ genetic algorithm library. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pages 002051–002056. IEEE.

[Serag et al., 2008] Serag, A., Ono, S., and Nakayama, S. (2008). Using interactive evolutionary computation to generate creative building designs. *Artificial life and Robotics*, 13(1):246–250.

[Takagi, 2001] Takagi, H. (2001). Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296.

[Tokui et al., 2000] Tokui, N., Iba, H., et al. (2000). Music composition with interactive evolutionary computation. In *Proceedings of the 3rd international conference on generative art*, volume 17, pages 215–226.