ParadisEO – PEO : Lesson 3

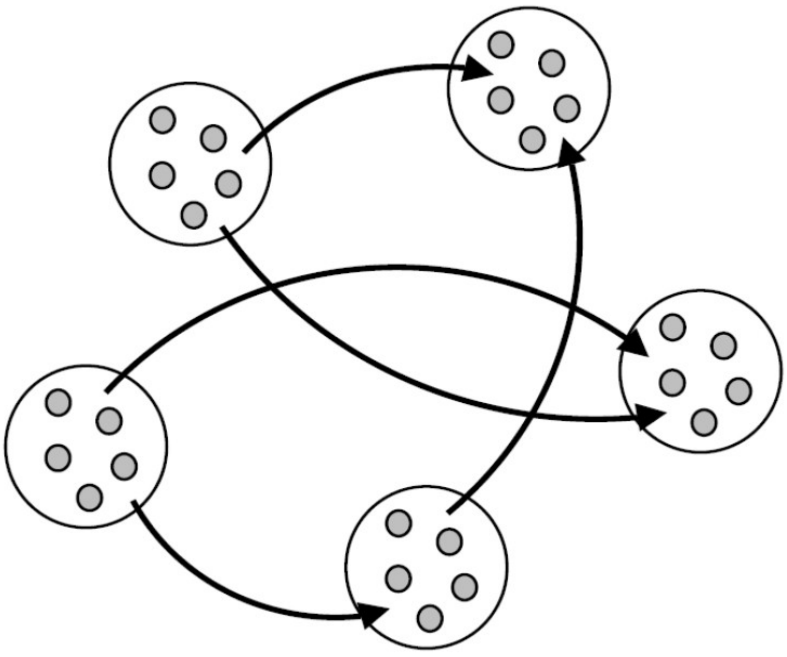
Note : All the components are not presented in this lesson (binary, topology, asynchronous or synchronous ...). To know the completeness of components refer to API documentation of ParadisEO – EO and ParadisEO – PEO.

Introduction

In the previous lessons strategies allowing for parallel evaluation of the population or transformation operators were presented. The included source code examples considered only one evolutionary algorithm (or one PSO) per application. While this model can be easily extended to include multiple evolutionary algorithms, no real interest in this approach exists as there would be no interaction between the concurrently executing algorithms. A more interesting scenario considers co-evolving algorithms disposed according to a given topology and which periodically exchange information in order to coordinate for improved convergence.

Explanations

A schematic representation of an arbitrary topological model with migrations between the islands is offered bellow:



Let's consider the following scenario - emigrations/immigrations should occur at every ten generations. In addition we would like to have control on selecting the individuals to emigrate as well as on the replacement process of the current individuals with the immigrant ones. In other words, constructing an insular migration model consists in (1) having a ring topological model including several evolutionary algorithms or PSO algorithms, (2) defining the migration frequency as well as the size of the migration (i.e. the number of individuals that emigrate) and (3) the selection and replacement strategies or the velocity strategy.

Requirements

Before to start this lesson 3, you should read and execute **Lesson 2**.

Of course, to execute the lesson 3, you should be in the directory of this lesson.

Problem

In the lesson 3 you can execute two different algorithms with a island model :

- Particle Swarm Optimization (PSO)
- Evolutionary Algorithm (EA)

The problem is the same in the two cases : minimizing the Rosenbrock function.

$$f(x_1,x_2)=100*(x_2-x_1^2)^2+(1-x_1)^2$$

The optimal is :

$$f(x_{1,}x_2)=0$$

with : $X=(1,1)$

PSO (mainPSO.cpp) :

```
#include <peo>
typedef eoRealParticle < double >Indi;
double f (const Indi & _indi)
{
    double sum;
    sum=_indi[1]-pow(_indi[0],2);
    sum=100*pow(sum,2);
    sum+=pow((1-_indi[0]),2);
    return (-sum);
}
int main (int __argc, char *__argv[])
{

// In this lesson, we define two algorithms of the PSO witch represents two
// islands.
// Of course, you can define more algorithms.

 // The parameters are common between the two algorithms.
 /*****************************************************************************/
    peo :: init( __argc, __argv );
    const unsigned int VEC_SIZE = 2;
    const unsigned int POP_SIZE = 20;
    const unsigned int NEIGHBORHOOD_SIZE= 6;
    const unsigned int MAX_GEN = 150;
    const double INIT_POSITION_MIN = -2.0;
    const double INIT_POSITION_MAX = 2.0;
    const double INIT_VELOCITY_MIN = -1.;
    const double INIT_VELOCITY_MAX = 1.;
    const double C1 = 0.5;
    const double C2 = 2.;
 // C3 is used for the calculation of one of the strategies of the island model.
    const double C3 = 2.;
 // MIG_FREQ define the frequency of the migration.
    const unsigned int  MIG_FREQ = 10; // The optimal value is 1 or 2 for the
                                       // component peoPSOVelocity.
    rng.reseed (time(0));
/******************************************************************************/

 // Define the topology of your island model
      RingTopology topologyMig;

 // First algorithm
 /*****************************************************************************/
    peoEvalFuncPSO<Indi, double, const Indi& > plainEval(f);
    peoSeqPopEval< Indi > eval(plainEval); // Here, the evaluation is sequential
    eoUniformGenerator < double >uGen (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
    eoUniformGenerator < double >sGen (INIT_VELOCITY_MIN, INIT_VELOCITY_MAX);
    eoVelocityInitFixedLength < Indi > veloRandom (VEC_SIZE, sGen);
    eoFirstIsBestInit < Indi > localInit;
    eoRealVectorBounds bndsFlight(VEC_SIZE,INIT_POSITION_MIN,INIT_POSITION_MAX);
    eoStandardFlight < Indi > flight(bndsFlight);
    eoPop < Indi > pop;
    pop.append (POP_SIZE, random);
    peoInitializer <Indi> init(eval,veloRandom,localInit,pop);
    eoLinearTopology<Indi> topology(NEIGHBORHOOD_SIZE);
    eoRealVectorBounds bnds(VEC_SIZE,INIT_VELOCITY_MIN,INIT_VELOCITY_MAX);
    eoStandardVelocity < Indi > velocity (topology,C1,C2,bnds);
    eoGenContinue < Indi > genContPara (MAX_GEN);
    eoCheckPoint<Indi> checkpoint(genContPara);
 // Specific implementation for the island model
    eoPeriodicContinue< Indi > mig_cont( MIG_FREQ );
    peoPSOSelect<Indi> mig_selec(topology);
    eoSelectNumber< Indi > mig_select(mig_selec);
// If you want to use a replacement stategy :
//                peoPSOReplacement<Indi> mig_replace;
// If you want to use a consideration of the migration in the calculation of the
//velocity :
//               peoPSOVelocity<Indi> mig_replace(C3,velocity);
      peoPSOReplacement<Indi> mig_replace;
/*******************************************************************************/

 // Second algorithm (on the same model but with others names)
/*******************************************************************************/
    peoEvalFuncPSO<Indi, double, const Indi& > plainEval2(f);
    peoSeqPopEval< Indi > eval2(plainEval2);
    eoUniformGenerator < double >uGen2 (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random2 (VEC_SIZE, uGen2);
    eoUniformGenerator < double >sGen2 (INIT_VELOCITY_MIN, INIT_VELOCITY_MAX);
    eoVelocityInitFixedLength < Indi > veloRandom2 (VEC_SIZE, sGen2);
    eoFirstIsBestInit < Indi > localInit2;
    eoRealVectorBounds
    bndsFlight2(VEC_SIZE,INIT_POSITION_MIN,INIT_POSITION_MAX);
    eoStandardFlight < Indi > flight2(bndsFlight2);
    eoPop < Indi > pop2;
    pop2.append (POP_SIZE, random2);
    peoInitializer <Indi> init2(eval2,veloRandom2,localInit2,pop2);
    eoLinearTopology<Indi> topology2(NEIGHBORHOOD_SIZE);
    eoRealVectorBounds bnds2(VEC_SIZE,INIT_VELOCITY_MIN,INIT_VELOCITY_MAX);
    eoStandardVelocity < Indi > velocity2 (topology2,C1,C2,bnds2);
    eoGenContinue < Indi > genContPara2 (MAX_GEN);
    eoCheckPoint<Indi> checkpoint2(genContPara2);
    eoPeriodicContinue< Indi > mig_cont2( MIG_FREQ );
    peoPSOSelect<Indi> mig_selec2(topology2);
    eoSelectNumber< Indi > mig_select2(mig_selec2);
    peoPSOReplacement<Indi> mig_replace2;
/*******************************************************************************/

 // Define the communication between the islands
      peoAsyncIslandMig< Indi > mig( mig_cont, mig_select, mig_replace,
topologyMig, pop, pop);
      checkpoint.add( mig );
      peoAsyncIslandMig< Indi > mig2( mig_cont2, mig_select2, mig_replace2,
topologyMig, pop2, pop2);
      checkpoint2.add( mig2 );
 // Initialization of the algorithms
    peoPSO < Indi > psa(init,checkpoint, eval, velocity, flight);
    mig.setOwner( psa );
    psa(pop);
    peoPSO < Indi > psa2(init2,checkpoint2, eval2, velocity2, flight2);
    mig2.setOwner( psa2 );
    psa2(pop2);
    peo :: run();
    peo :: finalize();
    if(getNodeRank()==1)
    {
      std::cout << "Population 1 :\n" << pop << std::endl;
      std::cout << "Population 2 :\n" << pop2 << std::endl;
    }
}
```

EA (mainEA.cpp) :

```
#include <peo>
#include <es.h>
typedef eoReal<double> Indi;
double f (const Indi & _indi)
{
    double sum;
    sum=_indi[1]-pow(_indi[0],2);
    sum=100*pow(sum,2);
    sum+=pow((1-_indi[0]),2);
    return (-sum);
}
int main (int __argc, char *__argv[])
{

    peo :: init( __argc, __argv );
    const unsigned int VEC_SIZE = 2;
    const unsigned int POP_SIZE = 20;
    const unsigned int MAX_GEN = 300;
    const double INIT_POSITION_MIN = -2.0;
    const double INIT_POSITION_MAX = 2.0;
    const float CROSS_RATE = 0.8;
    const double EPSILON = 0.01;
    const float MUT_RATE = 0.3;
 // MIG_FREQ define the frequency of the migration.
    const unsigned int  MIG_FREQ = 10;
 // MIG_SIZE define the size of each migration.
    const unsigned int  MIG_SIZE = 5;
    rng.reseed (time(0));
 // Define the topology of your island model
         RingTopology topology;

 // First algorithm
 /*****************************************************************************/
    eoGenContinue < Indi > genContPara (MAX_GEN);
    eoCombinedContinue <Indi> continuatorPara (genContPara);
    eoCheckPoint<Indi> checkpoint(continuatorPara);
    peoEvalFunc<Indi> plainEval(f);
    peoSeqPopEval< Indi > eval(plainEval);// Here, the evaluation is sequential
    eoUniformGenerator < double >uGen (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random (VEC_SIZE, uGen);
    eoRankingSelect<Indi> selectionStrategy;
    eoSelectNumber<Indi> select(selectionStrategy,POP_SIZE);
    eoSegmentCrossover<Indi> crossover;
    eoUniformMutation<Indi>  mutation(EPSILON);
    eoSGATransform<Indi> transform(crossover,CROSS_RATE,mutation,MUT_RATE);
    peoSeqTransform<Indi> eaTransform(transform); // Here, the transformation is
                                                  // sequential
    eoPlusReplacement<Indi> replace;
    eoPop < Indi > pop;
    pop.append (POP_SIZE, random);
    eoPeriodicContinue <Indi> mig_cont( MIG_FREQ ); // Migration occurs
                                                    // periodically
    eoRandomSelect<Indi> mig_select_one; // Emigrants are randomly selected
    eoSelectNumber<Indi> mig_select (mig_select_one,MIG_SIZE);
    eoPlusReplacement<Indi> mig_replace; // Immigrants replace the worse
                                         // individuals

/******************************************************************************/
// Second algorithm (on the same model but with others names)
 /*****************************************************************************/
    eoGenContinue < Indi > genContPara2 (MAX_GEN);
    eoCombinedContinue <Indi> continuatorPara2 (genContPara2);
    eoCheckPoint<Indi> checkpoint2(continuatorPara2);
    peoEvalFunc<Indi> plainEval2(f);
    peoSeqPopEval< Indi > eval2(plainEval2);
    eoUniformGenerator < double >uGen2 (INIT_POSITION_MIN, INIT_POSITION_MAX);
    eoInitFixedLength < Indi > random2 (VEC_SIZE, uGen2);
    eoRankingSelect<Indi> selectionStrategy2;
    eoSelectNumber<Indi> select2(selectionStrategy2,POP_SIZE);
    eoSegmentCrossover<Indi> crossover2;
    eoUniformMutation<Indi>  mutation2(EPSILON);
    eoSGATransform<Indi> transform2(crossover2,CROSS_RATE,mutation2,MUT_RATE);
    peoSeqTransform<Indi> eaTransform2(transform2);
    eoPlusReplacement<Indi> replace2;
    eoPop < Indi > pop2;
    pop2.append (POP_SIZE, random2);
    eoPeriodicContinue <Indi> mig_cont2( MIG_FREQ );
    eoRandomSelect<Indi> mig_select_one2;
    eoSelectNumber<Indi> mig_select2 (mig_select_one2,MIG_SIZE);
    eoPlusReplacement<Indi> mig_replace2;

/******************************************************************************/

// You can choose between :
//
// - Synchronous communication :
// peoSyncIslandMig<Indi> mig(MIG_FREQ,mig_select,mig_replace,topology,pop,pop);
// - Asynchronous communication :
//peoAsyncIslandMig<Indi> mig(mig_cont,mig_select,mig_replace,topology,pop,pop);
// With a grid, you should use an asynchronous communication
         peoAsyncIslandMig<Indi>
         mig(mig_cont,mig_select,mig_replace,topology,pop,pop2);
         checkpoint.add(mig);
         peoAsyncIslandMig<Indi>
         mig2(mig_cont2,mig_select2,mig_replace2,topology,pop2,pop);
         checkpoint2.add(mig2);
// Initialization of the algorithms
         peoEA<Indi> Algo(checkpoint,eval,select,eaTransform,replace);
         mig.setOwner(Algo);
         Algo(pop);
         peoEA<Indi> Algo2(checkpoint2,eval2,select2,eaTransform2,replace2);
         mig2.setOwner(Algo2);
         Algo2(pop2);

    peo :: run();
    peo :: finalize();
    if(getNodeRank()==1)
    {
            std::cout << "Final population 1 :\n" << pop << std::endl;
            std::cout << "Final population 2 :\n" << pop2 << std::endl;
    }
}
```

Launching the program

Your file should be called mainPSO.cpp or mainEA.cpp - please make sure you do not rename the file (we will be using a pre-built makefile, thus you are required not to change the file names). Please make sure you are in the paradiseo-peo/tutorial/build/Lesson3 directory - you should open a console and you should change your current directory to the one of Lesson3.

<u>Compilation :</u>

- make
- make install

<u>Execution (ie Technical Introduction):</u>

```
mpiexec -n 4 ./pso @param
or
mpiexec -n 4 ./ea @param
```