Paradiseo: MoeoLesson2

Paradiseo-MOEO Lesson 2 : Implement NSGA-II for the SCH1 problem

Introduction

We describe here a complete methodology to design and implement an easy-to-use NSGA-II algorithm using ParadisEO-MOEO applied to the SCH1
academic problem (Schaffer's bi-objective poblem 1). All the sources and programs required for this lesson are provided in the ParadisEO-MOEO package (located in the *paradiseo-moeo/tutorial/Lesson2* directory). The main file explained here is *Sch1.cpp*.

Fitness functions

The goal is to minimize the following functions:

- f1(x)=x^2
- f2(x)=(x-2)^2

We have one decision variable x, and two objective functions f1 and f2 that we want to minimize. We define a *Sch1ObjectiveVectorTraits* class:

```
class Sch1ObjectiveVectorTraits : public moeoObjectiveVectorTraits
{
public:static bool minimizing (int i) { return true; }
static unsigned nObjectives () { return 2; }
};
```

Note that *Sch1ObjectiveVectorTraits* must extend the ParadisEO-MOEO class *moeoObjectiveVectorTraits*.

The next step consists in defining the type of our individuals. Assuming our individuals are coded using a real-coded genotype, we need an *moeoRealObjectiveVector* templatized within *Sch1ObjectiveVectorTraits*:

```
typedef moeoRealObjectiveVector < Sch1ObjectiveVectorTraits > Sch1ObjectiveVector;
class Sch1 : public moeoRealVector < Sch1ObjectiveVector, double, double >
{
public:
Sch1() : moeoRealVector < Sch1ObjectiveVector, double, double > (1) {}
};
```

Then, we give the content of the fitness functions thanks to an *moeoEvalFunc* class based on *Sch1* :

```
class Sch1Eval : public moeoEvalFunc < Sch1 >
{
public:
void operator () (Sch1 & _sch1)
{
if (_sch1.invalidObjectiveVector())
{
Sch1ObjectiveVector objVec;
double x = _sch1[0];
objVec[0] = x * x;
objVec[1] = (x - 2.0) * (x - 2.0);
_sch1.objectiveVector(objVec);
```

```
            }
        }
    };
```

Crossover and mutation operators

Many variation operators are available within ParadisEO-EO for this kind of representation. We choose here a *eoQuadCloneOp* as the crossover operator and a *eoUniformMutation* as the mutation operator.

```
double M_EPSILON = 0.01;
eoQuadCloneOp < Sch1 > xover;
eoUniformMutation < Sch1 > mutation (M_EPSILON);
```

Initial Population

The *eoPop* class represents a population of individuals. Many initializers are available within ParadisEO-EO, we use a *eoRealInitBounded*, that generates individuals real genotype at random into a specified range.

```
unsigned POP_SIZE = 20; // population size
eoRealVectorBounds bounds (1, 0.0, 2.0); // range [0, 2]
eoRealInitBounded < Sch1 > init (bounds); // initializer
eoPop < Sch1 > pop (POP_SIZE, init);
```

The NSGA-II declaration

The last parameters we need to provide are :

```
unsigned MAX_GEN = 100; // number of generations before stopping
double P_CROSS = 0.25; // crossover probability
double P_MUT = 0.35; // mutation probability
Sch1Eval eval; // global evaluation object
// definition of NSGA-II for our problem
moeoNSGAII < Sch1 > nsgaII (MAX_GEN, eval, xover, P_CROSS, mutation, P_MUT);
```

Run it

Now we are ready to apply NSGA-II on the initial population:

```
nsgaII (pop);
```

Extract Pareto front from the final population

At the end, the population "pop" contains both the Pareto front found by NSGA-II and maybe some individuals from next fronts too. So we have to extract the Pareto front from it. This is done by using a *moeoArchive* object like this:

```
moeoArchive < Sch1 > arch;
arch.update (pop);
```

Conclusion

For any question or further information, please contact **paradiseo-help@lists.gforge.inria.fr**.