Open BEAGLE

A C++ Framework for your Favorite Evolutionary Algorithm

Christian Gagné, University of Lausanne, *christian.gagne@unil.ch*
Marc Parizeau, Université Laval, *parizeau@gel.ulaval.ca*

Numerous Evolutionary Computations (EC) software tools are now publicly available to the community – see for instance [1] and [2] for a listing of the most well known. The majority of these tools are specific to a particular EC flavor, however, only a few are truly generic EC softwares [3]. The highly diverse and adaptable nature of Evolutionary Algorithms (EA) make generic EC software tools a must-have for rapid prototyping of new approaches. As we all know, EC comprises a broad family of techniques where populations of solutions to problems are represented by some appropriate data structures (e.g. bit strings, real-valued vectors, trees, etc.) on which variation operators (e.g. mutation, crossover, etc.) are applied using iterative algorithms inspired from natural evolution. Different fitness measures can also be used, with one or several objectives, and it is possible to co-evolve several species of solutions, with different species represented by possibly different data structures.

To allow such great flexibility, these tools require a well-designed software architecture, generally attained using Object Oriented (OO) concepts [3]. Generic EC tools are thus significantly more complex than specialized tools, given all of the underlying mechanisms necessary for component replacement in representation, fitness, variation and selection operations, as well as evolutionary model. In the short-run, these mechanism may induce some cost to the user who is confronted with a somewhat steeper learning-curve. But we argue that, in the long-run, they also provide a very beneficial return on this investment, by allowing the efficient unification of different EC paradigms around a single flexible OO framework, which can provide elaborate additional facilities like dynamic configuration, logging and check-pointing of evolutions. Moreover, the maturing of a generic EC toolbox can, in the end, enable the construction of a black-box model, where components can be glued together without explicit programming, using a graphical interface and some scripting language.

The Open BEAGLE Framework

In 1999, the development of a small lightweight C++ library for Genetic Programming (GP) was started at Université Laval as a summer internship project. Three years later, in January 2002, after two complete rewrites, a generic C++ EC framework was publicly released as Open BEAGLE[1] [4]. Version 1.0 was released in July 2002, then version 2.0 in September 2003, and later on version 3.0 in October 2005.

While enabling most any EC paradigm through its generic mechanisms, Open BEAGLE currently provides direct support of the following major EA, through specialized layers:

- Bit string, integer-valued, and real-valued GA;
- Anisotropic self-adaptive ES and Covariance Matrix Adaptation ES (CMA-ES);
- Tree-based GP;
- Evolutionary multi-objective optimization (NSGA-II and NPGA2);
- Co-evolution through the use of multi-threading.

[1] BEAGLE refers to the name of the English vessel, HMS Beagle, on which Charles Darwin embarked for his famous voyage around the world. It also stands as a recursive acronym for: the Beagle Engine is an Advanced Genetic Learning Environment.

A general and extensible XML file format also allows the specification of EA configurations and parameters, logging of output and debugging information, and check-pointing of evolutions. With this file format, a web interface called BEAGLE Visualizer was designed to allow the viewing of basic evolution statistics using a standard web browser.

Another interesting derivative of Open BEAGLE is called *distributed* BEAGLE [4]. It enables the transparent distribution of the fitness evaluation tasks of any EA over a Beowulf cluster or a grid of workstations on a LAN. Distributed BEAGLE uses a master-slave architecture [5], where the master implements an abstract layer between evolution slaves that evolve a population to the next generation, and evaluation slaves that evaluate the fitness of individuals.

Code Example: OneMax

Despite the inherent complexity of a generic EC framework, the use of Open BEAGLE is relatively simple for a novice programmer. The different components have default values and policies that are suitable for most simple applications. The user is only required to define a fitness evaluation operator and a main method that initializes the different components of the framework. The following listing presents an evaluation operator implementation for the classical GA bit string example OneMax, which consists in searching for bit strings that have a maximum number of bits set to “1”.

```
 1. #include "beagle/GA.hpp"
 2. using namespace Beagle;
 3. class OneMaxEvalOp : public EvaluationOp {
 4. public:
 5. OneMaxEvalOp() : EvaluationOp("OneMaxEvalOp") { }
 6. virtual Fitness::Handle
 7.  evaluate(Individual& inIndividual, Context& ioContext)
 8.  {
 9.    GA::BitString::Handle lBitString =
10.      castHandleT<GA::BitString>(inIndividual[0]);
11.    unsigned int lCount = 0;
12.    for(unsigned int i=0; i<lBitString->size(); ++i)
13.      if((*lBitString)[i]) ++lCount;
14.    return new FitnessSimple(float(lCount));
15.  }
16. };
```

In this listing, line 5 is to construct a fitness operator named `OneMaxEvalOp`. Lines 6 to 15 corresponds to the function called to evaluate an individual fitness. Lines 9 and 10 cast the generic individual to evaluate into a bit string individual. Lines 11 to 13 count the number of ones in the bit string while line 14 returns the fitness measure, that is a single real value to maximize.

Now, the following listing presents the associated main routine for the OneMax problem.

```
 1. #include <cstdlib>
 2. #include <iostream>
 3. #include "beagle/GA.hpp"
 4. #include "OneMaxEvalOp.hpp"
 5. using namespace Beagle;
 6. int main(int argc, char** argv) {
 7.   try {
 8.     GA::BitString::Alloc::Handle
 9.       lBSAlloc = new GA::BitString::Alloc;
10.     Vivarium::Handle lVivarium = new Vivarium(lBSAlloc);
11.     OneMaxEvalOp::Handle lEvalOp = new OneMaxEvalOp;
12.     const unsigned int lNumberOfBits = 20;
13.     GA::EvolverBitString lEvolver(lEvalOp, lNumberOfBits);
14.     System::Handle lSystem = new System;
15.     lEvolver.initialize(lSystem, argc, argv);
16.     lEvolver.evolve(lVivarium);
17.   }
18.   catch(Exception& inException) {
19.     inException.terminate(std::cerr);
20.   }
21.   return 0;
22. }
```

Lines 8, 9, and 10 build a bit string population. Line 11 instantiates the fitness evaluation operator defined above. Lines 12 and 13 define a bit string GA evolver where individuals are initialized as a string of 20 bits each. Line 14 creates the evolution system while line 15 initializes the evolver and the evolution system, parses the command line, and reads configuration files. Finally, the evolution is launched at line 16. The entire routine is in a try-catch block in order to intercept exceptions which may be thrown by Open BEAGLE, if a problem is detected at runtime.

Different configurations of the evolutionary algorithm is possible. For example, if the user wants to use a steady-state GA instead of the default generational model, he must define a XML configuration file similar to the following one.

```
<?xml version="1.0"?>
<Beagle>
  <Evolver>
    <BootStrapSet>
      <GA-InitBitStrOp/>
      <OneMaxEvalOp/>
      <StatsCalcFitnessSimpleOp/>
    </BootStrapSet>
    <MainLoopSet>
      <SteadyStateOp>
        <OneMaxEvalOp>
          <GA-CrossoverOnePointBitStrOp
            matingpb="ga.cx1p.prob">
            <SelectTournamentOp/>
            <SelectTournamentOp/>
          </GA-CrossoverOnePointBitStrOp>
        </OneMaxEvalOp>
        <OneMaxEvalOp>
          <GA-MutationFlipBitStrOp
            mutationpb="ga.mutflip.indpb">
            <SelectTournamentOp/>
          </GA-MutationFlipBitStrOp>
        </OneMaxEvalOp>
        <SelectTournamentOp repropb="ec.repro.prob"/>
      </SteadyStateOp>
      <StatsCalcFitnessSimpleOp/>
      <TermMaxGenOp/>
      <MilestoneWriteOp/>
    </MainLoopSet>
  </Evolver>
</Beagle>
```

No re-compilation is necessary, the user only needs to execute the program with a command-line option referring to the previous configuration file. This example, as well as many others, are packaged together with the source code of Open BEAGLE.

Conclusion

Open BEAGLE is an open source LGPL framework for EC, freely available on the Web [4]. Written in C++, it is adaptable, portable, and quite efficient. Given its open and generic nature, it can be used to federate software development for EC, using an ever-growing library of components and tools, some of which have already been donated by different researchers from different institutions around the world. Through this newsletter, the authors hope that new EC researchers can join the pool of Open BEAGLE users and, eventually, become BEAGLE developers that contribute in their modest way to the progress of our community.

Bibliography

[1] John Eikenberry. GNU/Linux AI and Alife HOWTO. http://zhar.net/howto/html.

[2] EvoWeb Software Listing. http://evonet.lri.fr/evoweb/resources/software.

[3] Christian Gagné and Marc Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–174, April 2006.

[4] Christian Gagné and Marc Parizeau. Open BEAGLE W3 page. http://beagle.gel.ulaval.ca.

[5] Marc Dubreuil, Christian Gagné, and Marc Parizeau. Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 36(1):229–235, February 2006.