JEO: Java Evolving Objects

M.G. Arenas[1], Brad Dolin[1,2], P.A. Castillo[1], I. Fdez. de Viana[3], J.J. Merelo[1], and G. Romero[1]

[1] Dpto de Arquitectura y Tecnología de los Computadores
Universidad de Granada
CP 18071 – Granada (Spain)
e-mail: {maribel,pedro,gustavo,jmerelo}@geneura.ugr.es
[2] Computer Science Department
Stanford University
Stanford, CA 94305 (USA)
e-mail: dolin@cs.stanford.edu
[3] Dpto de Ciencias de la Computación
Universidad de Granada
CP 18071 – Granada (Spain)
e-mail: ijfviana@ugr.es

Abstract. In this paper we introduce an Evolutionary Computation (EC) software system which addresses many of the current needs of the EC research and development community. Java Evolving Objects library (JEO) provides a flexible and powerful framework for developing EC experiments. The package makes extensive use of the object-oriented paradigm, so that new experiments can be easily implemented by reusing or extending various ready-made EC paradigms: Genetic Algorithms (GA), Genetic Programming (GP), and others. Furthermore, JEO is fully integrated into a distributed computing package and coded in platform-independent Java, enabling distributed use of computing resources, even in heteregenous networks.
After discussing previous work in the area of EC object oriented software design, we present JEO design, architecture, and implementation. We also present two sample EC problems - the Travelling Salesperson Problem (solved using a linear chromosome representation), and the Symbolic Regression Problem (represented in GP). Despite their different genomic representations, the problems share a good amount of code - demonstrating JEO's lack of a need for many problem-specific data structures. Furthermore, JEO is able to solve these problems seamlessly in a distributed computing environment.

1 Introduction

Evolutionary Computation (EC) is currently used for an incredibly diverse range of applications, from SQL sentences improvement [19] to routing using multi-agent systems [6]. As such, the research community needs more powerful and flexible tools every day. JEO (Java Evolving Objects) provides a flexible and powerful framework for developing EC experiments. With JEO, the researcher

can specify distinct types of EC experiments (Genetic Algorithms, Genetic Programming, Evolutionary Strategies, etc.) without having to learn a different tool for each paradigm. Furthermore, meticulous use of the object-oriented paradigm makes the package particularly amenable to extensibility and code re-use. Thus, different evolutionary paradigms can be easily compared with little programming and a reduced learning curve.

Besides this, JEO addresses the problem of the increased computational resources currently demanded from EC. Experiments and applications need, in many cases, not merely a single computer but the computational power of an entire network of computers. Since these networks are almost always heterogeneous, portability is an additional problem. DREAM (Distributed Resources Evolutionary Algorithm Machine) [22] is a European research project designed to provide the research community with a Peer-to-Peer (P2P) system for EC problems. DREAM consists of a code distribution system (the "distributed resource machine" [11][10]) and an evolutionary computation system (JEO), all written in platform-independent Java code. The fully-integrated package, as such, solves the distrubuted computation and portability requirements.

The paper is organised as follows. Section 2 reviews recent work in evolutionary computation frameworks. JEO design principles are explained in Section 3, while architecture and implementation are presented in Section 4. In Section 5 several commonly used problems are used to allow new JEO users to understand the philosophy and structure of the system. Finally, Section 6 presents some conclusions and ideas for future work. ideas for future work.

2 State of Art

JEO borrows from and improves upon many of the existing EC tools. Its main source of inspiration is the Evolving Objects tool (EO) [21][20][12][3]. EO is a C++ class library designed to evolve any complex structure. It is flexible enough to allow implementation of several problem types. However, EO is implemented in C++, and employs some rather arcane programming techniques - making for a steep learning curve - whereas JEO is implemented in pure Object-Oriented Paradigm (OOP) Java. In the latest versions, EO presents distributed computation features [3], but using these features in a heterogeneous computer network may be hindered by issues with code portability. JEO's Java implementation, on the other hand, is platform-independent.

GAlib [8] is another C++ Evolutionary Computation library. It uses Parallel Virtual Machine (PVM) for task distribution and is extensible and flexible (although less than EO - some features like the number and functionality of genetic operators are hardwired). Like EO, it presents portability problems. Also, GAlib is designed mainly for Genetic Algorithms experiments. There are other C++ Evolutionary Computation libraries like EvolC [25], although none are as complete and mature as EO or GAlib.

Besides projects in C++, there are also Java Evolutionary Computation tools such as JDEAL [7], ECJ [18], JRGP [2], DGP [4][5], GPSYS [23], MAFRA [14]

and GJGP [1]. However, only JDEAL and ECJ are comparable to JEO. The rest are designed only for the Genetic Programming paradigm, except MAFRA which is a specialized tool for implementing hybrid evolutionary algorithms (GAs and local search). So far, MAFRA only implements bitstrings.

JDEAL is a Java objects library for developing Evolutionary Computation experiments. It lets the user distribute some task through a heterogeneous computer network. It provides a robust statistics package, and makes extensive use of OOP. However, JDEAL communications is based on a master-slave schema that could be a problem if the user needs a scalable system. JEO, on the other hand, bases its communication schema in DRM [10][11]. DRM is a DREAM project module that provides a P2P system for experiment execution. The P2P system solves the scalability problem by using a graph structure where each node is effectively equal to the others: every node can send and receive the same messages, and the node structure is completely user-determined. Another important improvement is that JEO is designed for any Evolutionary Computation paradigm, not only for Evolution Strategies and Genetic programming, as is JDEAL.

ECJ [18] is an evolutionary computation framework developed in Java. The package is extremely feature-rich and has an open architecture, as does JEO. However, the island migration architecture suffers from the flaw that it is not completely P2P. Client islands rely on a server to continue operation; as the documentation states, "... if the server goes down, the clients do not continue operation; they will shut themselves down. This means that in general you can shut down an entire island model network just by killing the server process." The JEO migration model is completely P2P, meaning that if any machine goes down, the "hole" in the network does not affect the whole. There are additional restrictions placed on the islands in ECJ: they must have the same kind of subpopulations and species; each subpopulation must send the same number of migrants as any other subpopulation; and, migrants from a subpopulation in an island must only migrate to the same subpopulation in other islands. JEO places none of the above restrictions on subpopulation migration: the user can specify independent Immigrator and Emigrator objects which manage these tasks in any conceivable way.

3 DESIGN

Any useful tool is based on design principles which provide homogeneity, simplicity and power to the framework. JEO design is based on the following principles:

1. JEO is object-oriented. This programming paradigm enables programmers to create modules that do not need to be changed when a new type of object is added. This lets developers design and implement tools using pre-existing pieces.
2. JEO is platform independent. JEO has been developed to build EC experiments that can be executed in a distributed and heterogeneous virtual machine. JEO task distribution is based on the Island Model [9][17][26]. As

such, JEO experiments use a set of distributed Islands, exchanging information about partial or final problem solutions.

3. JEO allows diverse types of evolvable objects. Any evolvable object must complete a set of rules for mutation, crossover, etc. Interestingly, not only individuals can be programmed in this way: An operator, an evaluator, or any other object could implement these rules and therefore be evolved.
4. JEO presents a seamless view of the network as a computational resources pool. It builds a layer over the already implemented DRM [19] layer, so that the EC user must only deal with EC concepts, such as islands, operators, etc., as opposed to distributed computing concepts like serialization and remote method invocation.

4 Architecture and Implementation

JEO is programmed in Java. This programming language was selected because it provides platform independent software and is powerful enough for task distribution and data collection. The Java compiler used is jdk 1.3.1.

Regarding JEO architecture, extensive and meticulous use is made of the object-oriented paradigm (OOP) and design patterns. This allows for easy code re-use and extensibility. As demonstrated in the next section, even experiments which implement different EC paradigms make use of the same core classes.

JEO's OOP implementation allows the user, for example, to record any class variable, anywhere in the code, at any evolutionary interval (each generation, etc.). All one needs to do is initalize a `ValueExtractor` object and install it in the Recorder object. The user can even install the `ValueExtractor` object into an `Analyser` object first, which in turn gets installed into the Recorder, so that maximum values, minimum values, means - indeed, any measurable attribute of the value - can be recorded and outputted into a Comma Seperated Value (CSV) file.

Objects are grouped in Java packages, each of which groups some logically related classes. The most important include:

- `dream.evolution.genomes`: genomes to support the various EC paradigms.
- `dream.evolution.initers`: objects responsible for first-generation initialization.
- `dream.evolution.operators`: vast selection of evolutionary operators to perform mutation, crossover, selection, and other operations on the population.
- `dream.evolution.darwiners`: the evolutionary engine - objects which assess the fitness of each individual, and manage the size, form, and composition of each population.
- `dream.evolution.checkpointing`: objects which record and output statistical information during the run, and conditionally make use of this information to control or end the run.

– `dream.evolution.migration`: migration and immigration controllers (note that actual network communications code is outside of the `dream.evolution` JEO package, in other dream.* packages)

Each logical concept has its own Java interface. Using Java interfaces makes JEO extensible by providing the user with three ways to use one object: the user may directly utilize some pre-existing JEO interface implementation, extend or modify an existing JEO implementation by sub-classing, or implement the object interface with a new class. Since in Java multiple inheritance is only possible via the use of interfaces, JEO permits, for example, a single object to serve as both an operator and an evolvable individual.

JEO includes some abstract concepts, each of which is represented by an interface. The principal logical concepts include:

1. InfoHabitant: In traditional Evolutionary Computation, an individual represents only a problem solution or part of the problem solution. However, JEO considers individuals as live inhabitants of a virtual world, using "InfoHabitant" as a name for this concept. The main InfoHabitant function is to provide a way to solve a problem (with the genome encoding the solution). But JEO extends this narrow, passive definition to allow for the possibility of "active" InfoHabitants. As such, an InfoHabitant also lives in a virtual world and has the ability to make decisions, and to think about any subject. An InfoHabitant may include a brain, which may be as complicated as an Artificial Neural Network or as simple as a set of decision rules. Of course, in traditional EC situations, the user need not concern himself with these concepts, but this view is more general and leaves room for the development of new algorithms.
2. Island: Each one of the independent processes that is executed to solve an Evolutionary Computation problem is called an Island. An Island is associated with only one CPU, but a CPU may have more than one Island. A problem is solved using a set of Islands that evolve a set of Environments. Each Island can run the same EC algorithm or a set of different ones. This corresponds also to the classical "deme" concept.
3. Environment: An Environment groups some InfoHabitants within the Island population. Each InfoHabitant group has some common features and can be modified with the same operators. If the problem solution can be represented using one class of InfoHabitants, the user usually needs only one Environment in the Island. If the problem solution needs more than one InfoHabitant to be represented, this problem is a coevolution problem and the user usually needs more than one Environment per Island.
4. Breeder: The Breeder's function is to apply variation operators to an Environment of InfoHabitants to produce an Environment offspring population; a Breeder is applied each generation and an Environment usually uses a single Breeder.
5. Migrator: Migrator carries out InfoHabitant movements between any two Islands that are evolving for the same experiment. Movements allow distribution of possible solutions throughout the Island network.

5 Experiments

We provide two experiments, taken from different paradigms within EC, to demonstrate the flexibility of the system.

5.1 Simple TSP

Consider a traveling salesperson who starts from his home city, visits each of a set of n cities exactly once, and then returns home. The salesperson wants to travel this circuit in the shortest possible distance. The Traveling Salesperson Problem [16][24] consists of finding a permutation of the n cities, called a tour, which minimizes the total tour distance. The version we implement here is symmetric in that the distance from city i to j is the same as that from city j to i. This experiment uses a linear integer chromosome to represent the tour with a specialized local search optimization.

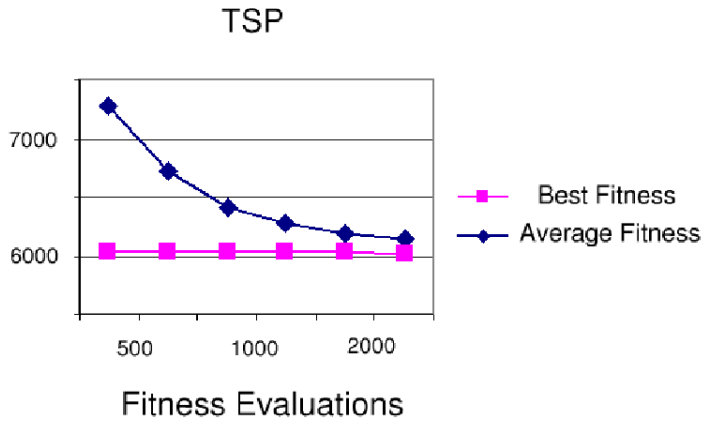

Fig. 1. Results of a typical node for the Travelling Salesperson Problem experiment. Best fitness of 7013 is achieved in generation 5.

Fitness is a double which measures the total tour distance. Selection is standard Roulette Wheel Selection. Two standard operators manipulate the genomes: OX-Order-Crossover [15] (with 0.80 probability), and ExchangeMutator (with 0.10 probability), which simply exchanges two genes in the city's tour. The remaining 10 percent of individuals are copied without modification. There is also a local search procedure, particular to the TSP problem, applied to each individual in the population (a type of "informed mutation") [8].

Each Island contains a single population of 500 individuals, and each run is terminated either when a solution is found or when we reach the maximum num-

ber of generations, 100. We use 10 such Islands, arranged in the simple "token-ring" structure. Each island sends its best individual, at the end of each generation, to the next Island in the ring. We use the city data file ulysses22.tsp (available from `http://www.iwr.uni?heidelberg.de/groups/comopt/software/TSPLIB95/tsp`). Results for a typical Island are shown in figure 1.

5.2 Symbolic Regression

We implement a simple symbolic regression experiment as well, this time using Genetic Programming functionality. The goal of symbolic regression (e.g., Koza [13]), is to find a tree-based function (i.e., a Lisp "S-expression") which very nearly approximates a given set of points. Here, we attempt to approximate the function:

$$Y = x^4 + x^3 + x^2 + x \tag{1}$$

Each expression can be constructed from the arithmetic operators $+$, $-$, $*$, `ProtectedModulus`, `ProtectedDivision`, Sin, Cos, `ProtectedExponent`, and `ProtectedRLog` [13]. We also include a terminal which stands for the independent variable, x.

Fitness is calculated as the total absolute error of the evolved function, as measured at 20 evenly spaced test points in the domain $[-1, 1]$.

We use Kozas ramped half-and-half method for initial tree generation, with an initial maximum tree depth of 6. The maximum tree depth after operator application is 17; if the resultant tree is larger, we attempt the operator again with a different set of individuals. Operators include: tree crossover with 0.90 probability (with internal node selection probability of 0.90 and leaf node selection probability of 0.10), and reproduction (copy without modification) with 0.10 probability. Tournament selection is used, with a tournament size of 3.

Each Island contains a population of size 500, and runs are terminated either when a solution is found, or 100 generations have been evolved. There are 10 such Islands, with the same structure as used in the TSP experiment, above. Results for a typical Island are shown in figure 2.

5.3 Experiment Specification in JEO

It is noteworthy that even while these experiments have different solution objectives, different fitness measures, and indeed even use genome representations and operators from different EC paradigms, the amount of code sharing is extensive. Indeed, JEO is designed such that the only code that needs to be written is that which is particular to the given experiment.

As such, the "evolutionary engine" - population storage, selection and replacement mechanisms, statistics gathering and reporting, inter-island migration - is utilized "as is" by both experiments. Each experiment specification class need only include EC parameters, and simple initialization code of ready-made

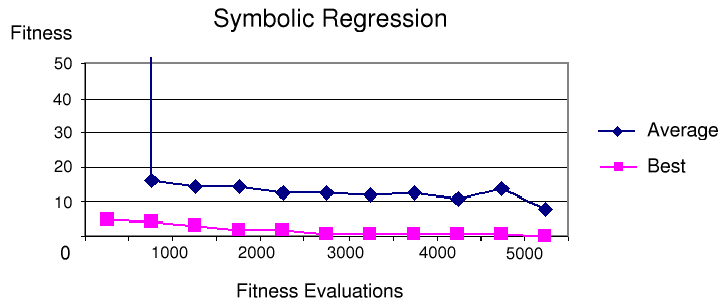

Fig. 2. Results of a typical node for the Symbolic Regression experiment. Average fitness in generation 0, off the graph, is $3.09E13$. Best fitness of 0 is achieved at generation 10.

JEO genomes and operators. The fitness function, and of course any experiment-specific operators, are the only logical concepts that must be implemented by the user. Code can be downloaded from `http://sourceforge.net/projects/dr-ea-m`.

6 Conclusions and Future Work

We have described the design principles, architecture and implementation of Java Evolving Objects - an extensible, platform-independent Evolutionary Computation software package capable of performing distributed computation. As proof of concept, we have presented two experiments from different EC paradigms - one using a linear chromosome and one from GP - which benefit from substantial code sharing.

JEO is, of course, still a work in progress. Future goals include the implementation of other EC paradigms (EA, GEP, etc.) as well as experiments with the "active" sort of InfoHabitant discussed above.

Acknowledgements

This work is supported by *Distributed Resources Evolutionary Algorithm Machine* (DREAM IST-1999-12679) project. This work is funded as part of the European commission Information Society Technologies Programme (Future and Emerging Technologies). The authors have sole responsibility for this work: it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein. Brad Dolin is supported by a Fulbright Grant. Thanks to the PUFO (*Prediccin Universal Financiera On-Line* CICYT TIC 1999-0550) and *INTAS* (INTAS-9730950) projects for their collaboration.

References

[1] Robert Baruch. Groovy java genetic programming. https://sourceforge.net/projects/jgprog.

[2] Pietro Berkes and Samuele Pedroni. Jrgp. Available from http://jrgp.sourceforge.net.

[3] J. G. Castellano, P.A Castillo, J. J. Merelo, and G. Romero. Paralelizacion de evolving library usando mpi. *In XII Jornadas de Paralelismo. ISBN: 84-9705-043-6, Valencia*, pages 265–270, September 2001.

[4] Fuey Sian Chong. A java based distributed approach to genetic programming on the internet. *In Procedings of Evolutionary Computation and Parallel Processing*, I:163–166, July 1999.

[5] Fuey Sian Chong. A java based distributed approach to genetic programming on the internet. *In Proceedings of Genetic And Evolutionary Computation Conference, ISBN 1-55860-611-4*, II, July 1999.

[6] Postgres Community. Postgres evolution. available from. Available from http://postgresql.lerner.co.il/devol-corner/docs/postgres/geqo-pg-intro.html.

[7] Joao Costa, Nuno Lopes, and Pedro Silva. Jdeal, the java distributed evolutionary algorithms library. Available from http://laseeb.ist.utl.pt/sw/jdeal.

[8] B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problem. In IEEE Press, editor, *In Proceedings of the IEEE international Conference on Evolutionary computation (ICEC'96)*, pages 616–621, 1996.

[9] D. E. Golberg and E. Cant-Paz. Modeling idealized bounding cases of parallel genetic algorithms. In Morgan Kaufmann, editor, *Proceedings of the Second Annual Conference of Genetic Programming*, pages 353–361, 1997.

[10] Mark Jelasity, Mike Preub, and Ben Paechter. A scalable and robust framework for distributed application. *2002 Congress on Evolutionary Computation*, May 2002.

[11] Mark Jelasity, Mike Preub, Maarten van Steen, and Ben Paechter. Maintaining connectivity in a scalable and robust distributed environment. *In 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, May 2002.

[12] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: a general purpose evolutionary computation library. *In Procedings Evolution Artificielle 2001*, 2001.

[13] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[14] Natalio Krasnogor and Jim Smith. MAFRA: A java memetic algorithms framework. In William Hart, Natalio Krasnogor, and Jim Smith, editors, *2000 Genetic And Evolutionary Computation Conference, First International Workshop On Memetic Algorithms - Workshop Proceedings*, pages 125–130, Las Vegas, Nevada, USA, Agoust 2000. citeseer.nj.nec.com/krasnogor00mafra.html.

[15] Davis L. Applying adaptive algorithms to epistatic domains. In *In Proceedings of the International Joint Conference on Artifical Intelligence*, pages 162–164, 1985.

[16] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons Ltd., 1985.

[17] M. R. Leuze, C. B. Pettey, and J.J. Grefenstette. A parallel genetic algorithms. In J. J. Grafenstette, editor, *Proceedings of the second International Conference on Genetic Algorithms*, pages 155–162, 1987.

[18] Sean Luke. A java-based evolutionary computation and genetic programming research system. Available from http://www.cs.umd.edu/projects/plus/ec/ecj.

[19] Sinclair M.C. Minimum cost wavelength-path routing and wavelength allocation using a genetic-algorithm/heuristc hybrid approach. *IEEE Proceedings Communications*, 146(1):1–7, Febreary 1999.

[20] J. J. Merelo, M. G. Arenas, J. Carpio, P.A. Castillo, V. M. Rivas, G. Romero, and M. Schoenauer. Evolving objects. In *Proceedings JCIS 2000 (Joint Conference on Information Sciences)*, volume I, pages 1083–1086, 2000.

[21] J. J. Merelo, Maarten Keijzer, and Marc Schoenauer. Eo evolutionary computation framework. Available from http://eodev.sourceforge.net.

[22] Ben Paechter, Thomas Baech, Marc Schoenauer, Michele Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. Dream distributed resource evolutionary algorithm machine. In *Proceedings of the Congress on Evolutionary Computation 2000*, volume 2, pages 951–958, 2000. Available from http://dr-ea-m.sourceforge.net.

[23] Adil Qureshi. A java genetic programming system. Available from http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html.

[24] G. Reinelt. *The traveling salesman: Computational solutions for TSP applications.* Springer Verlag, 1994. LNCS 840.

[25] Marc Schoenauer. Evolc. Available from http://www.eark.polytechnique.fr/EvolC.html.

[26] R. Tanese. Parallel genetic algorithms for hypercube. In J. J. Grafenstette, editor, *Proceedings of the second International Conference on Genetic Algorithms*, pages 177–184, 1987.