A Software Platform for Evolutionary Computation with Pluggable Parallelism and Quality Assurance

Pedro Evangelista[1,2], Jorge Pinho[1], Emanuel Gonçalves[1], Paulo Maia[1,2], João Luis Sobral[1], and Miguel Rocha[1]

[1] Department of Informatics / CCTC - University of Minho
jls@di.uminho.pt mrocha@di.uminho.pt
[2] IBB - Institute for Biotechnology and Bioengineering
Centre of Biological Engineering - University of Minho
Campus de Gualtar, 4710-057 Braga - PORTUGAL
ptiago@deb.uminho.pt paulo.maia@deb.uminho.pt

Abstract. This paper proposes the Java Evolutionary Computation Library (JECoLi), an adaptable, flexible, extensible and reliable software framework implementing metaheuristic optimization algorithms, using the Java programming language. JECoLi aims to offer a solution suited for the integration of Evolutionary Computation (EC)-based approaches in larger applications, and for the rapid and efficient benchmarking of EC algorithms in specific problems. Its main contributions are (i) the implementation of pluggable parallelization modules, independent from the EC algorithms, allowing the programs to adapt to the available hardware resources in a transparent way, without changing the base code; (ii) a flexible platform for software quality assurance that allows creating tests for the implemented features and for user-defined extensions. The library is freely available as an open-source project.

Keywords: Evolutionary Computation, Open-source software, Parallel Evolutionary Algorithms, Software Quality

1 Introduction

The field of EC has been rapidly growing in the last decades, addressing complex optimization problems in many scientific and technological areas. Also, there has been the proposal of several software platforms with varying sets of functionalities, implemented in distinct programming languages and systems. Although the authors acknowledge the existence of interesting platforms implementing EC approaches, none of the ones tested was able to simultaneously respond to the set of requirements underlying our needs, namely: (i) robustness, modularity and quality assurance in the process of developing EC based components for other systems and applications; (ii) efficiency in allowing the rapid benchmarking of distinct approaches in specific optimization tasks, mainly when taking advantage of the currently available parallel hardware.

This paper introduces the Java Evolutionary Computation Library (JECoLi), a reliable, adaptable, flexible, extensible and modular software framework that allows the implementation of metaheuristic optimization algorithms, using the Java programming language. The main contributions are its built-in capabilities to take full advantage of multicore, cluster and grid environments and the inclusion of a quality assurance platform, an added value in validating the software.

In this work, we show how independent parallelism models and platform mappings can be attached to the basic framework to take advantage of new computing platforms. Our objective is to support parallel processing allowing the user to specify the type of parallelization needed. The parallelization models are independent of the framework, and the user can specify the parallel execution model to meet specific computational methods and resources, taking advantage of the processing power of each target platform.

On the other hand, the quality assurance framework developed aims at providing a flexible way for both the developers of JECoLi and JECoLi-based applications to be able to validate the available features such as algorithms, operators, parallelization features, etc. This framework should allow to conduct software tests with minimal programming.

2 Main functionalities

JECoLi already includes a large set of EC methods: general purpose Evolutionary/ Genetic Algorithms, Simulated Annealing (SA), Differential Evolution, Genetic Programming and Linear GP, Grammatical Evolution, Cellular Automata GAs and Multi-objective Evolutionary Algorithms (NSGA II, SPEA2). Among the parameters that can be configured, we find the encoding scheme, the reproduction and selection operators, and the termination criteria.

In JECoLi, solutions can be encoded using different representations. A general purpose representation using linear chromosomes is available including binary, integer or real valued genes. Also, individuals can be encoded as permutations, sets and trees. Numerous reproduction operators are provided for each of these representations. It is also possible to add new user-defined representations and/ or reproduction operators.

A loose coupling is provided between optimization algorithms and problems allowing the easy integration with other software. New problems are added by the definition of a single class specifying the evaluation function. An adequate support is also given to the development of hybrid approaches, such as local optimization enriched Evolutionary Algorithms (EAs) (e.g. Memetic Algorithms), hybrid crossover operators and initial population enrichment with problem-specific heuristics.

The framework is developed in Java, being portable to the major systems. The project is open source, released under the GPL license. Extensive documentation (howto's, examples and a complete API), binaries and source code releases are available in the Wiki-based project's web site `http://darwin.di.uminho.pt/jecoli`.

3 Overall architecture and development principles

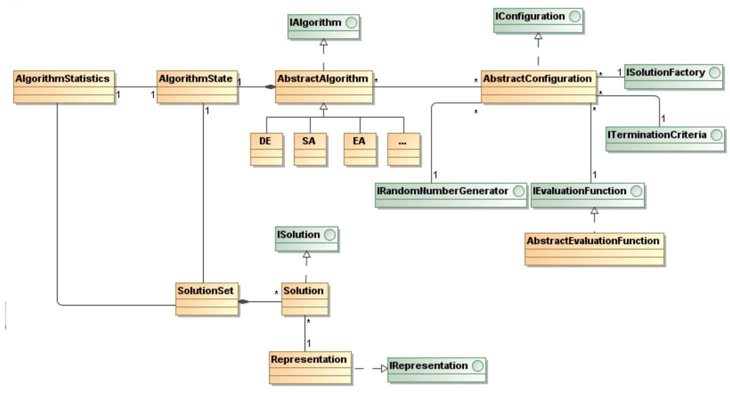

Fig. 1. Architecture of the JECoLi: main classes.

The JECoLi architecture was built to be an adaptable, flexible, extensible, reliable and modular software platform. In Figure 1 a simplified class diagram is shown, containing the main entities. The *Algorithm* abstraction represents the optimization method. The information common to all algorithms was abstracted in the *AbstractConfiguration* class. The algorithm dependent configuration details are stored in the concrete implementation. When an algorithm starts executing, it checks all setup components, executes a specific initialization routine, and finally executes a series of iterations. During the execution, each algorithm holds an internal state that stores the current solution set, results and statistics collected during the previous iterations. Several alternative termination criteria can be defined (e.g. number of generations, function evaluations, CPU time).

Classes representing individuals implement interface *ISolution*, including the genome (with a specific representation) and the fitness value(s). Implementation of specific representations follow the *IRepresentation* interface. The solution factory design pattern was used to allow building new solutions and copying existing ones. These factories are a centralized location for restrictions applied to a specific genome. Populations are implemented by the *SolutionSet* that implements lists of solutions with enhanced functionalities.

An evaluation function is responsible for decoding an individual to a particular problem domain and providing for fitness evaluation. This makes the connection between the problem and the algorithm domains. One of the major aims was the extensibility of the platform, implemented by defining contracts (using interfaces) for specific components enabling the addition of new algorithms, representations, operators, termination criteria, etc.

4 Parallelization of the library

The parallelization strategy is based on three key characteristics:

– Non-invasive - parallelization should have minimal impact on the original code; using/ developing functionalities in the base JECoLi does not require the developer to have knowledge about the parallelism models/mappings;
– Localized - parallelization code should be localized in well defined modules;
– Pluggable - the original code can execute when parallelism modules are added/ removed, i.e. parallelism modules are included on request.

Thus, the parallelization of the library consists of (i) the base JECoLi; (ii) a set of non-invasive and pluggable modules that adapt the base framework to support parallel execution and a set of mappings that adapt the supported models of parallel execution to each target platform; (iii) a tool that composes the base framework with modules for parallel execution and platform mappings, according to the user requests. Two distinct conceptual models for the parallelization of EAs are implemented: (i) the island model; and (ii) the parallel solution evaluation. These models do not imply any specific parallel environment and can be executed even in sequential architectures.

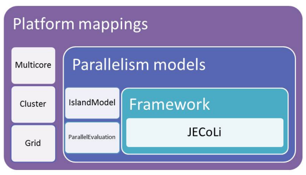

Fig. 2. Architecture of the Parallelization of JECoLi.

The parallelization of JECoLi is based on a three-layer architecture depicted in Figure 2. The features were made based on layers: the first layer is the original JECoLi; the second contains the parallelism models to use, and the last one introduces the mapping of the parallel behavior to the execution platform. The parallelism model layer contains the parallel models presented above: island model and parallel execution of the evaluation. Three distinct modules were implemented in this layer: *Parallel Eval*, *BuildIslandModel* and *Abstract Migration*, implementing the parallelization features that encapsulate platform independent behaviour. The first implements the model related to the parallel execution of the evaluation of solutions, while the second and third are related to the implementation of the island model. The mappings layer contains all the specific code to be loaded regarding a specific parallel platform. It encapsulates the mapping of the parallelism models to meet specific target parallel environments: single machine with multicore processor, clusters and grid environments.

The implementation of the modules resorted to the use of Aspect Oriented Programming (AOP) [2], a programming technique used to encapsulate crosscutting phenomenona into specific modules of the programs. Technical details of the full implementation of the modules are provided in the web site documentation.

5 Quality assurance framework

A generic test framework was engineered in order to validate JECoLi's functionalities. The main aim is to verify the correctness of the distinct components. The tests treat each component of the library as a black-box entity with a predefined set of inputs and a desired output. These are defined in order to cover all the possible outcomes of an entity for a certain input even if the library component only returns a partial result set. If the outcome of an entity is contained within the full result set the test is considered valid.

A domain specific language (DSL) with an LL(1) grammar was conceived to create *JUnit* tests [1] for the distinct components (algorithms, operators and parallelization methods). This language captures the tests configuration skeleton. Examples of configurations of specific tests are provided in the project web site.

Also, a test framework was developed encompassing a class hierarchy that is given in the web site documentation. The implementation of the abstract classes in the lower level of the class hierarchy only needs to detail component configuration information. These structures allow to capture seamlessly the test structure and to implement it in a domain specific language without the need to program in Java to add new tests for existing components.

6 Applications

The library has already been used in several research projects, described in the web site. Here, we highlight a few of these examples that have led to the development of software applications:

– Metabolic engineering: the aim is microbial strain optimization, using *in silico* simulations based on genome-scale metabolic models. EAs and SA using a set-based representation have been used [5] and implemented as part of the OptFlux software platform (`www.optflux.org`) [4], that represents a reference open-source software platform.
– Fermentation optimization: aims at the numerical optimization of feeding profiles in fed-batch fermentation processes in bioreactors [3]. EAs with a real value representations and several variants of DE were tested and evaluated. The implementation of these approaches was included in a software application named OptFerm (`darwin.di.uminho.pt/optferm`).
– Network traffic engineering: the aim is to implement methods to improve the weights of intra-domain routing protocols [6]. Two approaches were used, both using an integer representation: single objective EAs with linear weighting objective functions and multiobjective EAs. These methods have been incorporated into an application - NetOpt (`darwin.di.uminho.pt/netopt`).

7 Conclusions

This paper described the framework JECoLi, a Java-based platform for the implementation of metaheuristic methods. The main strengths of this platform rely on its efficiency, portability, flexibility, modularity and extensibility which make it ideal to support larger applications that need to include an optimization engine based on EC methods. The main focus of this library has been to support this embedding capabilities also by making available a quality assurance framework, including a domain specific language for building unit tests, that allows the full coverage of the implemented features.

In terms of computational efficiency, one of the major concerns has been the development of modules to support parallelism. The parallelization of JECoLi was pursued in a pluggable, non-invasive and localized way, thus allowing the original library to evolve with new features being added and also to allow the automatic adaptation of the code to distinct scenarios in terms of hardware resources (multicore, cluster, grid).

Most of the future work includes the improvement of the existing functionalities. Also, we aim to develop new capabilities including new algorithms, representations or selection/reproduction operators.

Acknowledgments

This work is supported by project PTDC/EIA-EIA/115176/2009, funded by Portuguese FCT and Programa COMPETE.

References

1. Y. Cheon and G. Leavens. A simple and practical approach to unit testing: The JML and JUnit way. *ECOOP 2002, Object-Oriented Programming*, pages 1789–1901, 2006.
2. G.J. Kiczales, J.O. Lamping, C.V. Lopes, J.J. Hugunin, E.A. Hilsdale, and C. Boyapati. Aspect-oriented programming, October 15 2002. US Patent 6,467,086.
3. R. Mendes, I. Rocha, E. Ferreira, and M. Rocha. A comparison of algorithms for the optimization of fermentation processes. In *2006 IEEE Congress on Evolutionary Computation*, pages 7371–7378, Vancouver, BC, Canada, jul 2006.
4. I. Rocha, P. Maia, P. Evangelista, P. Vilaça, S. Soares, J. P. Pinto, J. Nielsen, K.R. Patil, E.C. Ferreira, and M. Rocha. Optflux: an open-source software platform for in silico metabolic engineering. *BMC Systems Biology*, 4(45), 2010.
5. M. Rocha, P. Maia, R. Mendes, E.C. Ferreira, K. Patil, J. Nielsen, and I. Rocha. Natural computation meta-heuristics for the in silico optimization of microbial strains. *BMC Bioinformatics*, 9(499), 2008.
6. P. Sousa, M. Rocha, M. Rio, and P. Cortez. Efficient OSPF Weight Allocation for Intra-domain QoS Optimization. In *Lecture Notes in Computer Science 4268, 16. Autonomic Principles of IP Operations and Management*, pages 37–48. Springer, 2006.