CICLOPS: COMPUTATIONAL INTELLIGENCE COLLABORATIVE LABORATORY OF PANTOLOGICAL SOFTWARE

ES Peer, AP Engelbrecht, G Pampará, BS Masiye

Department of Computer Science, University of Pretoria
Pretoria, South Africa
espeer@cs.up.ac.za, engel@cs.up.ac.za, gpampara@cs.up.ac.za, bmasiye@cs.up.ac.za

ABSTRACT

This paper presents CiClops, which is a virtual laboratory for performing experiments, using Computational Intelligence (CI) algorithms, that scale over multiple workstations. Additionally, the paper introduces CIlib, which is an open source library of CI algorithms, currently containing mostly particle swarm optimization (PSO) and ant colony optimization (ACO) algorithms. The main purpose of CiClops is to specify CI algorithms to solve optimization problems, to schedule execution of large numbers of simulations on a cluster of workstations, and to archive all empirical data for analysis. The objective of this paper is to launch both CiClops and CIlib, and to emphasize to the CI (most specifically the swarm intelligence) research community the advantages of using these tools.

1. INTRODUCTION

The computational intelligence (CI) research field is broad and encompasses a very wide array of research interests. The formulation of a precise definition for Computational Intelligence (CI) and how it relates to the broader Artificial Intelligence (AI) field is a challenging task. Arguably, CI comprises of those paradigms in AI that relate to some kind of biological or naturally occurring system. General consensus suggests that these paradigms are neural networks, evolutionary computing, swarm intelligence and fuzzy systems [1, 2, 3, 7]. Neural networks are based on their biological counterparts in the human nervous system. Similarly, evolutionary computing draws heavily on the principles of Darwinian evolution observed in nature. Swarm intelligence, in turn, is modeled on the social behaviour of insects and the choreography of birds flocking. Finally, human reasoning using imprecise, or fuzzy, linguistic terms is approximated by fuzzy systems. Figure 1 shows these four primary branches of CI and illustrates that hybrids between the various paradigms are possible.

Research in these CI areas is by necessity empirical and as such dictates code implementation and simulation. It is in this vain that a software framework, CIlib and CiClops are proposed as a flexible, efficient and adaptable empirical CI research tool. This paper aims to introduce, motivate and explain the capabilities and implementation specifics of the framework.

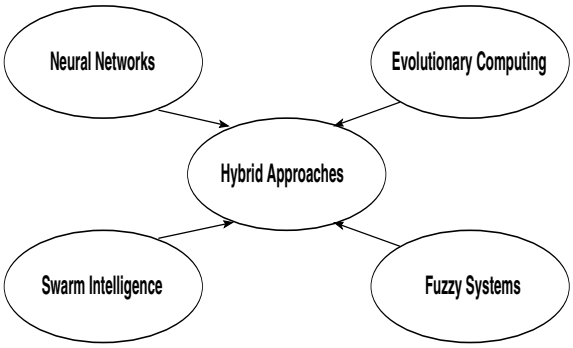

Figure 1. Computational intelligence paradigms

The rest of the paper is organised as follows: Section 2 provides a detailed motivation for the development of the CI framework. The CI library, CIlib, is discussed in Section 3. The simulation environment, CiClops, is introduced in Section 4.

2. MOTIVATION

The Computational Intelligence Research Group[1] at the University of Pretoria (CIRG@UP) has recently started an initiative to develop an extensive library of CI algorithms, mainly to facilitate the research efforts of the group. Additionally, a simulation and analysis environment has been developed to execute vast numbers of simulations which implement CI algorithms, and to process the large amounts of generated empirical data. The two tools have been called respectively, CIlib (Computational Intelligence library) and CiClops (Computational Intelligence Collaborative Laboratory of Pantological Software).

[1] http://cirg.cs.up.ac.za

0-7803-8916-6/05/$20.00 ©2005 IEEE

Although CIlib and CiClops have initially been developed to serve the needs of CIRG@UP, a survey of recent PSO papers [4] have revealed a number of problems which motivated the group to develop systems that can be used by the wider CI (specifically the swarm intelligence) research community as effective collaborative tools. These problems include:

- **Duplication of effort**: In the restricted context of a research group, duplication of effort equates to lost productivity. In general, the science is better served if researchers can expend their efforts on developing new algorithms instead of writing implementations for software that already exists elsewhere. A collaborative code base can save researchers from reinventing the wheel.
- **Failure to take latest developments into account**: A collaborative code base increases awareness of what others are doing, in effect providing all participants with a more generalised view even though they specialise on their own specific work.
- **Insufficient testing on problems**: The No Free Lunch (NFL) theorem [5, 6] implies that algorithms should be tested on many problems to determine which problems they are best suited for, since all algorithms are on average equivalent when all possible problems are considered. Thus, large amounts of empirical data will need to be generated, which may have value if shared, to draw conclusions about the relative merit of different algorithms.
- **Poor parameter choices**: Good parameter choices for algorithms can be communicated as default values in a shared implementation platform. Also, a shared repository of simulation results can make researchers aware of the best results obtained for a given algorithm by other researchers.
- **Conflicting results**: A collaborative platform will undergo more stringent peer review, which will eliminate the occurrence of conflicting results reported on the same problem using the same algorithm. The code within a shared resource is likely to be far more reliable than throw away research code.
- **Invalid statistical inference**: Shared statistical analysis tools, which provide decision support for the best analysis method to use in a given context, can reduce the risk of researchers making incorrect assumptions about the applicability of statistical tests.

CIlib and CiClops have as objective to address these problems.

3. CILIB

CIlib, an open source library[2] is a software framework designed (and implemented in Java) to accommodate scientific research in CI, providing implementations for many CI algorithms, problems definitions, and a simulator for conducting experiments. The simulator receives an XML file which completely specifies the algorithm(s) to be executed, the problem (or set of problems) to be solved, and the performance criteria that need to be calculated. This section provides an overview of CIlib, and provides examples to illustrates its use. Section 3.1 summarises the goals of CIlib. An overview of the framework is given in Section 3.2, while example XML specifications are given in Section 3.3.

3.1. Goals

CIlib has the following high level goals:

- **Flexibility**: Design patterns are used to create a reusable framework capable of supporting the complexity of the CI field.
- **Experimentation**: The framework facilitates scientific experimentation, making it possible to measure any property of an algorithmic simulation.
- **Efficiency**: CI algorithms are usually computationally expensive. The framework trades-off fast implementations with clean object-oriented design (for maximum modularity).
- **Separability**: A clean separation of algorithms and problems is provided. Algorithms are also independent of scientific simulation and measurement components.
- **Reliability**: Clean object-oriented design and extensive unit testing reduce the chance of errors in code.
- **Collaboration**: The framework shares a common open source base, which allows other researchers to add to easily add to the code base.

3.2. CIlib Framework

`Algorithm` and `Problem` are the two main interfaces of the CIlib framework, used tp specify the algorithm to be executed and the optimisation problem to be solved. The logic for how the algorithm optimises the problem and the manner in which the problems are optimised are kept separate. The `Algorithm` interface, summarised in Figure 2, provides the needed operations for the execution of an algorithm. Algorithms have the following properties: (1) they are running or stopped, (2) they are initialised or uninitialised, and (3)

[2] http://cilib.sourceforge.net

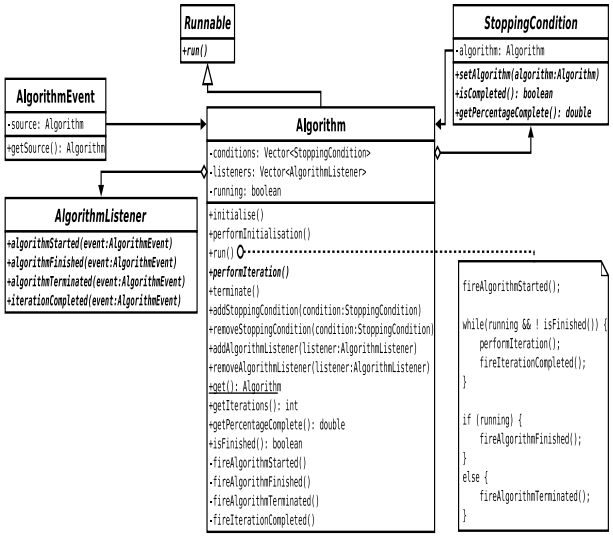

Figure 2. Algorithm Interface

they have a collection of `StoppingConditions`. Stopping conditions may assume a variety of forms, and CIlib facilitates this diversity. It is therefore possible to specify a range of terminal conditions, from maximum iterations to maximum fitness evaluations or minimum function optimization error or minimum swarm diameter.

Problems are categorised into `OptimisationProblems`, as illustrated in Figure 3. Optimisation can either be a maximisation or a minimisation. Note that the interface allows for single-objective and multi-objective optimisation problems. Different problems are defined as subclasses of the interface `Function`. Each problem returns a `Fitness` which can then be used by the implemented algorithm.

Algorithms that solve optimisation problems follow the `OptimisationAlgorithm` interface as illustrated in Figure 4, which illustrates implementation for a particle swarm optimiser (PSO). Figure 5 summarises the specification of PSO implementations.

As an empirical platform, CIlib must have a facility for measurement of performance criteria, which is facilitated via the `Measurement` class. Measurements are the required statistics that can be gathered from an algorithm. These measurements are stored in a data base at specified intervals. Examples of measurements include fitness, time, swarm diameter, function optimization error measurement, iterations, fitness evaluations, restarts and percentage complete.

CIlib also provides for a very generic way of defining the characteristics of the search space. This is done via `Type` and `Domain` classes. Via these classes, it is possible to specify the domain as, e.g. R^{30} to indicate a 30-dimensional continuous-valued search space.

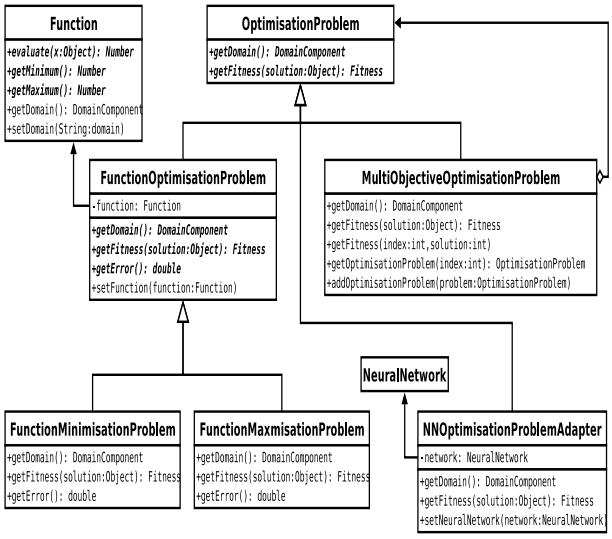

Figure 3. Problem Interface

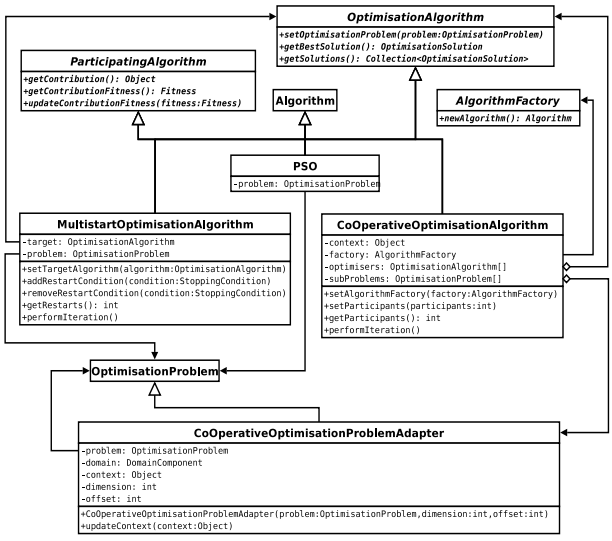

Figure 4. Optimisation Algorithm Interface

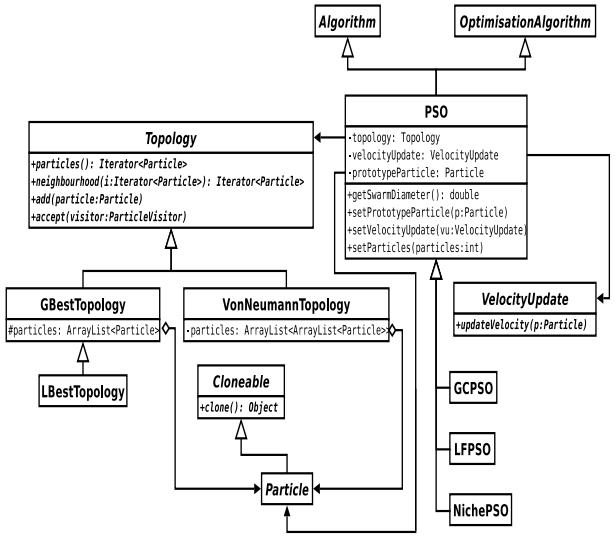

Figure 5. Particle Swarm Optimisation Interface

3.3. XML Specification Examples

CIlib provides a command line simulator for configuring and executing experiments. The simulator is essentially a XML object factory which allows for the specification of algorithms, problems and measurements. The XML object factory constructs, configures and composes the corresponding objects at run time according to an XML document.

Figure 6 is an example configuration for the CIlib simulator, using a standard PSO with a linear decreasing inertia component to find the minimum of the spherical function on its default domain of "R(-100,100)^30", given by:

$$f(\mathbf{x}) = \sum_{i=1}^{30} x_i^2, \text{ with } x_i \in \{\mathbb{R} \mid -100 \leq x_i \leq 100\} \quad (1)$$

while measuring the number of iterations and function optimisation error, by default every 100 iterations, and outputting the results to a file named "inertia.txt". By default, the simulator repeats the experiment 30 times, actually it runs them in parallel threads, outputting all the results to the same file, where they can be later analysed. The simulation engine parses the XML document for a `</simulator>` tag, specifying a single algorithm on a given problem, each time measuring certain properties. Applicable tag names are governed by the properties available in the source code at run time. The set of permissible tags may be found in the Java reflection API generated by Javadoc. Primitive typed properties and strings are set by enclosing them in tags. For example, in Figure 6

```
<minimumValue>0.25</minimumValue>
<maximumValue>1.0</maximumValue>
```

set minimumValue and minimumValue to 0.25 and 1.0 respectively.

Figure 7, in turn, illustrates another slightly more complex configuration file. This example demonstrates how portions of the document can be reused by making use of ID references. Typically, more descriptive identifiers than "A", "B", "M" and "S" would be used, they were shorten here purely for formatting reasons. Note that it is immaterial that multiple algorithms and simulations are specified within `<algorithms/>` and `<simulations/>` elements. The simulator merely searches for simulation elements and follows any identity links to their targets, irrespective of where they are defined in the document. Further, the sample demonstrates two short hand ways to set properties. Primitive and string valued properties can be specified directly as attributes in the parent element instead of nesting them as separate elements. Alternatively, they can be specified using the `value` attribute of their own property tags instead of placing the value in the body of the element. Properties in reused portions of the document can be overridden where they are referenced. For example, the same measurement suite configuration is used to output to two different file names. In addition, the measurement suite has two additional properties: i) the number of repetitions of the experiment, or samples; and ii) the resolution, which specifies how often results are written to file.

Figure 9 illustrates how an ant colony optimization algorithm, the ant system in this case, can be specified to solve the traveling salesman problem. The specification is based on the `ACO` interface.

4. CICLOPS

The main motivation for CiClops, which is currently not available as an open source project, is to provide a system to schedule simulations on a computer cluster, to archive all generated data, and to provide tools for the visualisation and analysis of empirical data. One of the tools is to maintain an up-to-date ranking of all algorithms in CIlib, based on defined criteria. The main objectives of CiClops are:

- **Scalability**: An arbitrary number of simulations per experiment is allowed, with simulations scheduled over multiple workstations.
- **Simulation repository**: Complete simulation results are stored for all algorithms and all problems. This eliminates expensive re-computations for new research studies. Simulation data keep track of dependencies on code and data sets, so that if dependencies change, results are automatically recalculated to ensure consistency and correctness.
- **Statistical analysis tools**: The aim is to implement and provide decision support for sound statistical hy-

```
<simulator>
  <simulation>
    <algorithm class="PSO.PSO">
      <addStoppingCondition class="StoppingCondition.MaximumIterations"/>
      <velocityUpdate class="PSO.StandardVelocityUpdate">
        <inertiaComponent class="PSO.LinearDecreasingValue">
          <minimumValue>0.25</minimumValue>
          <maximumValue>1.0</maximumValue>
        </inertiaComponent>
      </velocityUpdate>
    </algorithm>
    <problem class="Problem.FunctionMinimisationProblem">
      <function class="Functions.Spherical"/>
    </problem>
    <measurements class="Simulator.MeasurementSuite">
      <file>inertia.txt</file>
      <addMeasurement class="Measurement.Iterations"/>
      <addMeasurement class="Measurement.FunctionOptimisationError"/>
    </measurements>
  </simulation>
</simulator>
```

Figure 6. Simple PSO Specifi cation

pothesis testing. Built-in tools will also be provided for data visualization.

- **Ease of use**: An easy to use graphical user interface is provided to specify simulations. The user interface updates automatically based on changes in the CIlib code base.
- **Security**: A granular permission system is provided to keep data private if so desired, and to ensure data integrity.

Using a graphical user interface, the researcher specifies the algorithm(s) to execute, the problem(s) to solve and the performance measures to be calculated.

The architectural overview of CiClops is given in Figure 10. There are three main components in CiClops:

- **The CiClops code base**: This is the heart of CiClops and it is used in conducting the actual experiments. This replaces the role of the command line simulator (which receives the XML simulation specification). CiClops periodically updates its version of CIlib from the the latest version stored in the CVS repository at SourceForge.
- **A cluster of workstations**: Each cluster node, or worker, consists of a light weight stub which executes tasks, taking the form of CIlib simulations, on behalf of the CiClops server. Workers always execute simulations using the latest available version of the CIlib

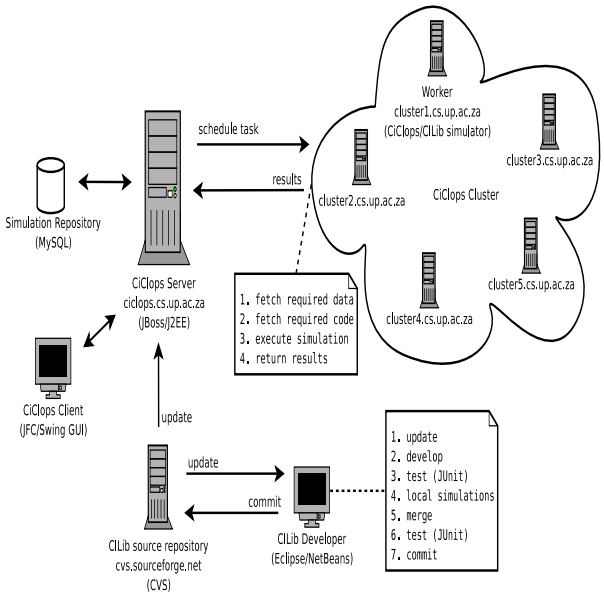

Figure 10. CiClops Architectural Overview

```
<simulator>
  <algorithms>
    <algorithm id="A" class="Algorithm.CoOperativeOptimisationAlgorithm">
      <algorithmFactory class="XML.XMLAlgorithmFactory">
        <algorithm idref="B"/>
      </algorithmFactory>
      <participants value="10"/>
    </algorithm>
    <algorithm id="B" class="PSO.PSO">
      <topology class="PSO.VonNeumannTopology"/>
      <addStoppingCondition class="StoppingCondition.MaximumIterations"/>
    </algorithm>
  </algorithms>
  <problem id="S" class="Problem.FunctionMinimisationProblem">
    <function class="Functions.Spherical" domain="R(-50,50)^100"/>
  </problem>
  <measurements id="M" class="Simulator.MeasurementSuite" samples="50">
    <addMeasurement class="Measurement.FitnessEvaluations"/>
    <addMeasurement class="Measurement.FunctionOptimisationError"/>
  </measurements>
  <simulations>
    <simulation>
      <algorithm idref="A"/>
      <problem idref="S"/>
      <measurements idref="M" file="data/cpso.txt"/>
    </simulation>
    <simulation>
      <algorithm idref="B"/>
      <problem idref="S"/>
      <measurements idref="M" file="data/pso.txt"/>
    </simulation>
  </simulations>
</simulator>
```

Figure 7. More Complex PSO Specification

classes and any data sets by means of remote class loading and efficient local caching of data sets.

- **A central server and data store**: The CiClops server is implemented as a J2EE application and deployed on the open source JBoss application server. The back-end data store is a MySQL relational database, although the J2EE persistence framework makes this largely irrelevant to the application, affecting only the deployment descriptor, which is generated automatically using XDoclet. The server is responsible for configuring experiments, scheduling tasks on the cluster, archiving simulation results and performing statistical analysis on the results. The load balancing services provided by the J2EE container means that CiClops can also be scaled up to multiple servers if and when the load of many workers becomes too high for one server to handle. Finally, some kind of user interface is required to interact with the system. Presently, this is provided in the form of a rich JFC/Swing based GUI client, with a view to providing a web based front end in the future. Fortunately, this should not be difficult to accomplish, since all the CiClops application logic is executed on the server, lying within the application tier of the J2EE framework.

5. CONCLUSIONS

The paper introduced a flexible, peer reviewed collaborative framework for CI empirical work. The benefits accruing from the use of, and continual assimilation of new algorithms into the framework, have been discussed. Although reported usage of the framework is within CIRG@UP, it is

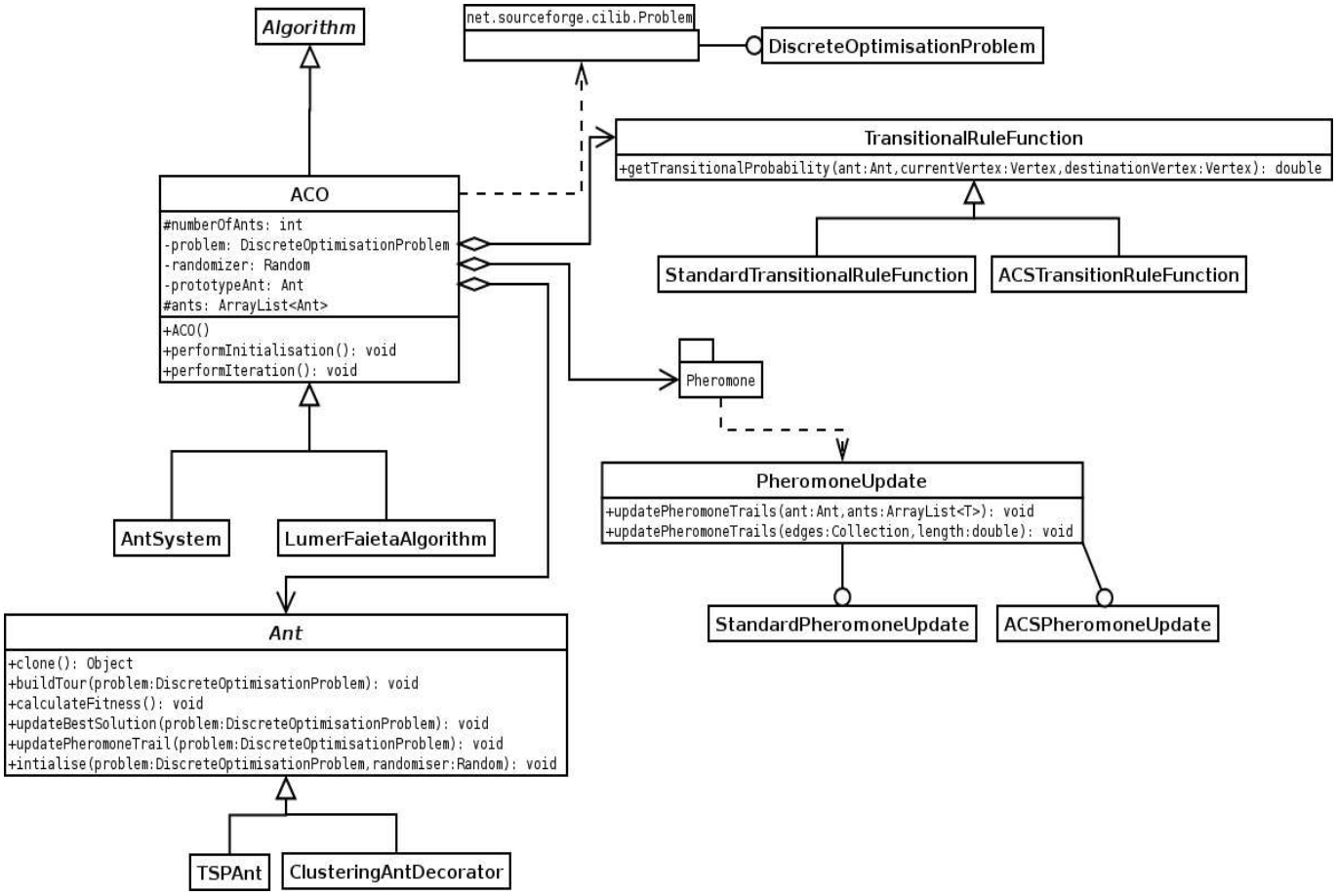

Figure 8. Ant Colony Optimisation Interface

hoped that the CI community will take advantage of, and also contribute to the growth and maturity of CIlib. It is also hoped that in the future the following may be closely looked at:

- **The role of open source in collaborative research**: Open source clearly has benefits for collaborative software development. Its role should be studied further, to identify and quantify critical success factors when using open source as a means to facilitate collaborative research, so that these factors may be applied to other projects. If and when CIlib becomes successful as a collaborative tool beyond the borders of the CIRG@UP, it can be analysed as a case study to achieve this goal.
- **PSO Taxonomy and characterisation of optimisation problems**: A solid foundation for performing empirical studies is provided by the combination of CIlib and CiClops. In this light, the original goal of creating a PSO taxonomy and empirically testing PSOs should be revisited. Further, a method of characterizing optimisation problems should be investigated to determine the type of problems for which a particular optimisation algorithm is best suited.
- **MathML for specifying benchmark functions**: Benchmark functions in CIlib are implemented using a separate class for each function, resulting in a very large number of classes and no way to define new functions without resorting to writing code. MathML, an XML grammar for defining mathematical expressions, should be investigated as an alternative. A primary concern will be the efficiency of this approach, since benchmark functions are typically executed in tight loops. One possibility worth investigating is compiling MathML function descriptions directly into Java byte code at run time so that they become the equivalent of classes.
- **Mining simulation data**: CiClops has the potential to generate large volumes of simulation data. Data mining techniques should be investigated to determine trends in simulation data. In cases where the underlying data mining algorithms are based on CI tech-

```
<simulator>
  <algorithms>
    <algorithm id="astsp" class="ACO.ASTSP" numberAnts="6" tau="0.000001">
      <prototypeAnt class="ACO.TSPAnt">
        <transitionRuleFunction class="ACO.StandardTransitionRuleFunction"
               alpha="1.0" beta="5.0"/>
        <pheromoneUpdate class="ACO.Pheromone.StandardPheromoneUpdate"
               rho="0.5" e="5.0" Q="100.0"/>
      </prototypeAnt>
      <addStoppingCondition class="StoppingCondition.MaximumIterations"
               iterations="9"/>
    </algorithm>
  </algorithms>
  <problems>
    <problem id="TSP" class="ACO.TSPProblem">
      <dataSet class="Simulator.LocalDataSet" file="data/pr107.tsp"/>
    </problem>
  </problems>
  <measurements id="measurements" class="Simulator.MeasurementSuite"
               samples="1" resolution="10">
    <addMeasurement class="ACO.GraphMeasurementSolutionLength" />
    <addMeasurement class="ACO.GraphMeasurementSolution" />
  </measurements>
  <simulations>
    <simulation>
      <algorithm idref="astsp" />
      <problem idref="TSP" />
      <measurements idref="measurements" file="data/results-aco-astsp.txt" />
    </simulation>
  </simulations>
</simulator>
```

Figure 9. Ant System Algorithm to Solve the Traveling Salesman Problem

niques, as many are, an interesting question of whether CI techniques be applied recursively to make sense of CI simulation results can be answered.

Readers who think consider to make use of CIlib, or who want to contribute code, are invited to contact the second author, AP Engelbrecht.

6. REFERENCES

[1] RC Eberhart, P Simpson, R Dobbins, *Computational Intelligence PC Tools*, AP Professional, 1996.

[2] AP Engelbrecht, *Computational Intelligence: An Introduction*, Wiley& Sons, 2002.

[3] W Pedrycz, *Computational Intelligence: An Introduction*, CRC Press, 1998.

[4] ES Peer, AP Engelbrecht, F van den Bergh, *CIRG@UP OptiBench: A Statistically Sound Framework for Benchmarking Optimisation Algorithms*, Proceedings of the IEEE Congress on Evolutionary Computation, pp 2386-2392, 2003, Canberra, Australia.

[5] DH Wolpert, WG Macready, *No Free Lunch Theorems for Search*, Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

[6] DH Wolpert, WG Macready, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, 4:67–82, 1997.

[7] H. Zimmermann, G Tselentis, M van Someren, G Dounias, *Advances in Computational Intelligence and Learning: Methods and Applications*, Kluwer Academic Publishers, 2002.