Scripting and Framework Integration in Heuristic Optimization Environments

Andreas Beham, Johannes Karder, Gabriel Kronberger,
Stefan Wagner, Michael Kommenda, Andreas Scheibenpflug
University of Applied Sciences Upper Austria
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232 Hagenberg, Austria
(abeham, jkarder, gkronber, swagner,
mkommend, ascheibe)@heuristiclab.com

Johannes Kepler University Linz
Institute for Formal Models and Verification
Altenberger Straße 69, 4040 Linz, Austria
(andreas.beham, michael.kommenda)
@students.jku.at

ABSTRACT

Rapid prototyping and testing of new ideas has been a major argument for evolutionary computation frameworks. These frameworks facilitate the application of evolutionary computation and allow experimenting with new and modified algorithms and problems by building on existing, well tested code. However, one could argue, that despite the many frameworks of the metaheuristics community, software packages such as MATLAB, GNU Octave, Scilab, or RStudio are quite popular. These software packages however are associated more closely with numerical analysis rather than evolutionary computation. In contrast to typical evolutionary computation frameworks which provide standard implementations of algorithms and problems, these popular frameworks provide a direct programming environment for the user and several helpful functions and mathematical operations. The user does not need to use traditional development tools such as a compiler or linker, but can implement, execute, and visualize his ideas directly within the environment. HeuristicLab has become a popular environment for heuristic optimization over the years, but has not been recognized as a programming environment so far. In this article we will describe new scripting capabilities created in HeuristicLab and give information on technical details of the implementation. Additionally, we show how the programming interface can be used to integrate further metaheuristic optimization frameworks in HeuristicLab.

Categories and Subject Descriptors

I.2.5 [**Artificial Intelligence**]: Programming Languages and Software

General Terms

Algorithms, Experimentation

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-2881-4/14/07 ...$15.00.
http://dx.doi.org/10.1145/2598394.2605690.

Keywords

HeuristicLab; Evolutionary Computation Frameworks; Metaheuristic Optimization Frameworks; Scripting

1. INTRODUCTION

The behavior and performance of metaheuristic optimization methods is often influenced by small details. To avoid confusion such methods are often shared with the community as a concrete implementation, for example in a given programming language. Because of the requirement to compare new algorithms with existing ones and an explosion of different metaheuristic algorithms, metaheuristic optimization (MOP) frameworks have emerged that contain implementations of such methods [9]. But while frameworks aim to provide mostly a well-designed API, software environments go one step further and define not only extension points for concrete applications, but provide a run-time environment in which the user can prototype and test new ideas.

HeuristicLab[1] has evolved over the years as an environment for heuristic optimization and became a popular tool for researchers and practitioners [10, 11]. Extensibility is a strong requirement of such an environment so that tools and methods to answer new research questions can be implemented. The benefit of the API and the reuse of existing methods, analysis and visualizations often outweighs the integration effort and can compensate for disadvantages such as a steeper learning curve [8]. Software environments attempt to mitigate the use of traditional development environments, i.e. IDE, compiler and linker. In HeuristicLab tasks such as algorithm prototyping, running and analyzing experiments can be performed in the run-time environment. Still, often when finding answers to new research questions, it is required to use traditional development tools to extend the functionality.

The ability to write and execute code while running HeuristicLab has been present since the release of version 3.3.0, but was restricted to operators that were executed inside an algorithm. Starting from HeuristicLab 3.3.10 programming directly within the environment has been significantly improved. This allows to prototype and test new ideas faster, especially for those users that are familiar with programming languages and the Microsoft .NET Framework. Additionally, tasks such as advanced experiment cre-

[1] `http://dev.heuristiclab.com`

ation could be more complex than what is already featured in the environment. Sometimes such advanced tasks can be integrated into the UI easily, but sometimes the additions would make the UI complex and confusing. With the addition of a scripting interface the user is able to implement a customized experiment generator that handles more complex requirements. Users are able to quickly write new algorithms for existing problems and reuse existing visualization and analysis features. An additional option that arises and which is further looked at in this paper concerns the integration of further MOP frameworks in HeuristicLab. Features, algorithms, and problems available in other frameworks are valuable and we aim to make these available within the HeuristicLab optimization environment.

1.1 Related Work

To the best of the authors' knowledge the integration of different MOP frameworks and direct reuse of algorithms and problems has not yet been described before. There have been some attempts at a generic and open data exchange between problem specific and algorithm specific parts. For instance, in the PISA project[2] an exchange was realized using text files, in GenOpt[3] a file-based exchange was realized to interconnect optimization and simulation, and also in HeuristicLab protocol-based data exchange has been implemented to communicate with external evaluation processes [1]. However, data exchange is only one aspect of interconnection. The ability to execute code written in other MOP frameworks requires an integration at the programming language level. Fortunately, a lot of progress has been made here by at least two projects that we aim to introduce and evaluate their usefulness in the context of bringing together different MOP frameworks. However, first we want to describe the scripting capabilities and how we allow writing and executing code within the running HeuristicLab application.

2. SCRIPTING IN HEURISTICLAB

A script in HeuristicLab can be created similar to how a problem or an algorithm would be created. Once a script is created three aspects of the script are presented in the GUI as can be seen in Figure 1:

- Script code
- Store for variables
- Console output log

Different scripts do not share these elements, so each script has access to its own console and variable store. The user can however use drag and drop to copy or link variables in the variables store from one script to another. Thus, different scripts can be created and applied to the same data. Unlike running algorithms in a typical MOP framework all variables remain present after the script has finished executing. These variables might then be stored together with the script code.

The language that is available for scripting as of version 3.3.10 is C#, which is natural for those familiar with the HeuristicLab API. Furthermore, a script type based on the Python programming language, which is popular among scientists, is under development and will be part of a future release. It is certainly desirable to include more script types with further languages, however the limitation is that all of these languages need to be compiled to or interpreted in the Microsoft Intermediate Language (MSIL) of the .NET Framework. We take a look at some options in Section 3.

C#-based Scripting

The class diagram that supports creating C# scripts is shown in Figure 2. The code is dynamically compiled to a class which derives from *CSharpScriptBase*. When the script is to be run, an object is instantiated, the variable store is passed to the instance, wrapped inside *Variables* and the *Main()* method of the *CSharpScript* is called.

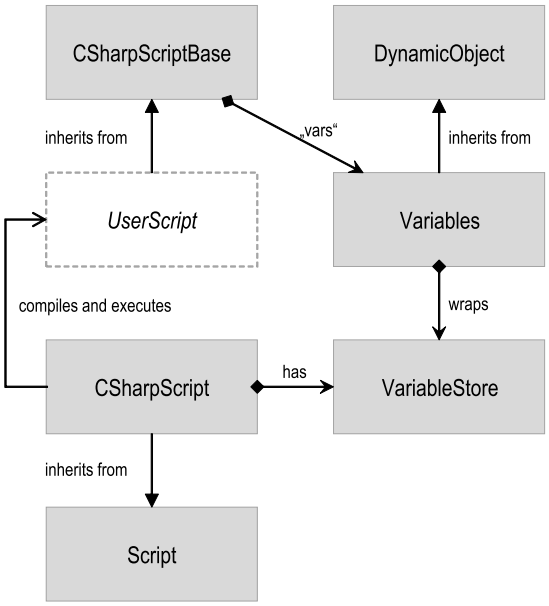

Figure 2: Two main classes provide scripting support: *Script* is the "design-time" class that contains the code, its subclass *CSharpScript* additionally contains the *VariableStore*, while *CSharpScriptBase* is the "run-time" class from which the compiled script derives.

Script Code

The code in a HeuristicLab script is presented as a simple string object. This object is updated through a code editor window that is based on SharpDevelop 3[4]. The editor features syntax highlighting and code completion for C# which are very valuable functions when developing scripts.

When the script is run it is compiled through the use of the *Microsoft.CSharp.CSharpCodeProvider* and further classes in the *System.CodeDom* namespace. Compile errors will be listed in a separate tab together with line and column number, and the compile error message. The script is executed in a separate thread which can be aborted in the case the user wants to prematurely terminate the script. When the user presses the stop button, the common language runtime

[2] http://www.tik.ee.ethz.ch/sop/pisa/
[3] http://simulationresearch.lbl.gov/GO/
[4] http://www.icsharpcode.net/opensource/sd/

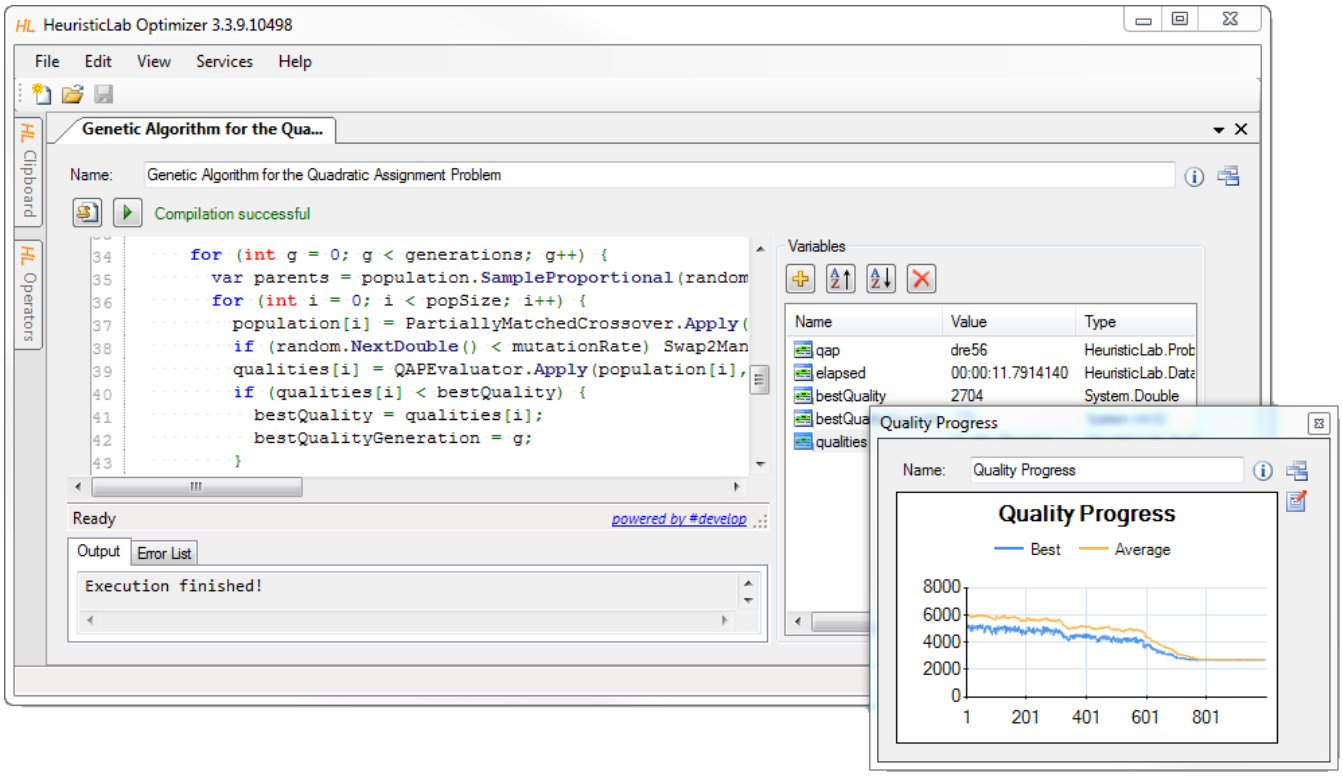

Figure 1: Screenshot of a C# script that executes a genetic algorithm on an instance of the quadratic assignment problem. The quality progress chart is shown in a separate window in the lower right.

throws a *ThreadAbortException*. Scripts can catch this exception in critical parts of the script in order to terminate gracefully, but care should be taken in order to terminate quickly. There is no method to force a thread to abort, a hanging thread can therefore only be stopped if the whole environment is restarted.

Variable Store

The variable store makes use of the dynamic language runtime (DLR) introduced in the Microsoft .NET Framework 4.0. This greatly facilitates access to variables as they can be defined by simply assigning a value to a name. In the background the variable store will create a new or overwrite an existing entry in a dictionary. The variable store can be accessed in scripts through the field *vars*. For instance, if the user writes `vars.x = 1`, a new variable x will be created in the store or if it already exists the value of x will be overwritten. The store displays the value of each variable as returned by the *ToString()* method.

Each variable has to have a unique name and a certain value. The value can be anything from primitive types such as int, double, DateTime, etc. to complex types such as DataTable or problem instances. Since basically any .NET object can be added as a variable, a problem with the serialization mechanism emerges. Any item in HeuristicLab and many .NET items can be written to a file, however instances of e.g. a Thread class cannot be serialized. Thus the variable store will indicate if it contains types which may not be serialized by an error icon next to the variable.

The variable store allows drag and drop behavior, so that an item in HeuristicLab can be dragged onto the variable store. The user could thus configure a problem instance in the GUI, drag it into the variable store and execute the script. When using drag and drop the user is given the option to name that variable, otherwise a default name is chosen.

If there is a default view for an item present in the variable store, that variable can be double clicked and the respective view for the value will open. Any changes made in that view will be applied to the variable's value. Items for which there is no view, ie. int, double, etc. cannot be changed in the GUI. However HeuristicLab has wrappers for these types such as IntValue or DoubleValue which may be viewed and edited [11].

Console Output

Scripts that need to output information to a console can write to the Console property. Although this property doesn't have the same functionality as the corresponding *System.Console*, it shares the write methods. We wanted to make it very similar so that existing pieces of code that should be run inside HeuristicLab can be quickly adapted. The console however does not feature input, therefore it is not possible to write interactive scripts.

Python-based Scripting

The *Python Script* in HeuristicLab is very similar to the C# script described above in that the variable store and the classes for handling console output are reused. The only difference is that the code is not compiled using .NET's CodeDom features, but executed with the help of the Iron-

Python[5] project. When the script is executed a *PyEngine* is instantiated. All HeuristicLab plugins are loaded into the engine and a *PyScope* is created which contains the variable store as variable *vars*. Thus, the access to the variable store is completely identical to the C# script. The user writes code which writes to or reads from *vars* or with the help of local variables that do not leave the PyScope. Console output redirection is achieved by adding an event handler to the engine's output *PyConsoleStream*. This raises another event of the *PythonScript* class to which the *PythonScriptView* listens. Python support via IronPython is added to HeuristicLab through a plugin which makes this feature generally available within the environment. The integration need not be performed for each script separately.

```
from System import *
from System.Linq import *
from HeuristicLab.Encodings.PermutationEncoding import *
from HeuristicLab.Problems.QuadraticAssignment import *
from HeuristicLab.Random import *

if not vars.Contains("qap"):
  print "qap is not defined!"

# Variables
N = vars.qap.Weights.Rows
w = vars.qap.Weights
d = vars.qap.Distances
popSize = 100
maxGen = 10000
mutRate = 0.05
rng = MersenneTwister()
start = DateTime.UtcNow
bestQuality = Double.MaxValue
bestQualityGeneration = 0
bestSolution = None

# Population Initialization
pop = [Permutation(PermutationTypes.Absolute, N, rng)] * popSize
qual = []
for i in range(0, popSize):
  qual.append(QAPEvaluator.Apply(pop[i], w, d))
  if qual[i] < bestQuality:
    bestQuality = qual[i]
    bestSolution = pop[i]

# Main Loop
for g in range(0, maxGen):
  par = Enumerable.ToArray(RandomEnumerable.SampleProportional(
    pop, rng, 2 * popSize, qual, True, True))
  for i in range(0, popSize):
    pop[i] = PartiallyMatchedCrossover.Apply(
      rng, par[i*2], par[i*2+1])
    if rng.NextDouble() < mutRate:
      Swap2Manipulator.Apply(rng, pop[i])
    qual[i] = QAPEvaluator.Apply(pop[i], w, d)
    if qual[i] < bestQuality:
      bestQuality = qual[i]
      bestQualityGeneration = g
      bestSolution = pop[i]

# Display Results
vars.elapsed = DateTime.UtcNow - start
vars.bestQuality = bestQuality
vars.bestQualityGeneration = bestQualityGeneration
vars.bestSolution = bestSolution
```

Figure 3: Genetic algorithm sample that solves a quadratic assignment problem written in Python and using the .NET and HeuristicLab API.

[5]http://ironpython.net/

3. FRAMEWORK REVIEW AND INTEGRATION

In a well-received survey on evolutionary computation frameworks ten different MOP frameworks are compared [8]. A large number of features are being analyzed in that study and an extensive comparison is being performed. Apart from the differences between frameworks, it is also obvious that many frameworks share features and algorithms are implemented many times anew. It is certainly understandable that an algorithm that is adapted to the peculiarities of a certain framework is easier to use and can make use of all the features, but at the same time we have to raise the question if it would not be possible to reuse features of one framework in another. We think that a promising future direction, at least for the HeuristicLab environment, would be the integration of different frameworks and to provide a development environment in which it is possible to run and experiment with algorithms of various frameworks. In this section we want to review and evaluate two frameworks, namely the Python-based DEAP[5] framework and the Java-based jMetal[4] framework. Both frameworks are licensed under the LGPL enabling an integration in HeuristicLab which is licensed under GPLv3.

Benefits of Framework Integration

The integration of frameworks is especially helpful when considering the need for comparison of different methods with each other. Methods that are implemented in a certain framework will have to be reimplemented in another framework for comparison purposes. Often this process takes a lot of time and is subject to certain coding standards enforced by the framework. This creates a barrier for comparing against a range of alternatives. Often the experimenting API and statistical evaluation of the results differs from framework to framework which makes it an arduous task to compare methods of different frameworks. There have been attempts at unifying parameterization of experiments [3], however this approach requires that the individual frameworks agree to a common description. So far, it has been difficult to enforce a standard in this context. In contrast, if the algorithms can be started and their results can be queried from a single framework, then the experiment analysis can also be unified in that framework. For instance, in HeuristicLab, each run of an optimization algorithm is stored in a *Run* object that contains dictionaries for the parameters and the results. It is possible to script the execution of an algorithm from a different framework, but generate and then compare those Run objects as results. Additionally, if the script produces the set of experiments that are calculated, this script can be shared together with the publication and the results can be reproduced by a reviewer.

DEAP

DEAP[6] 1.0.0, which was released in February 2014, is a relatively young framework developed at the Computer Vision and Systems Laboratory (CVSL) in the Faculty of Science and Engineering at Laval University, Québec, Canada. The first tag in the source tree dates back to 2010. It is implemented in Python which makes it attractive for prototyping new ideas as the IPython shell also provides a run-time environment. DEAP includes several algorithms such as genetic

[6]http://code.google.com/p/deap/

algorithms with arbitrary representations, genetic programming, evolution strategies, multi-objective algorithms such as NSGA-II[2] and SPEA2[12], co-evolution approaches, and many others.

jMetal

The jMetal 4.5 framework[7], which was releaed in January 2014, is implemented by Antonio J. Nebro and Juan J. Durillo from the Universidad Málaga. The framework is a bit more mature in that developments already started in 2004. It is a framework written in Java and which is especially targeted for solving multi-objective optimization problems. It includes a very large range of multi-objective algorithms which include NSGA-II variants, SPEA2, PAES[6], AbYSS[7], and many other algorithms. There are also a number of multi-objective problems implemented with and without constraints that are often used in testing as well as multi-objective quality indicators such as hypervolume metric, spread and generational distance. jMetal also features single-objective algorithms and problems such as genetic algorithms, evolution strategies, and many more.

We think that these, as well as many other frameworks are well suited for conducting research as they are created and maintained by researchers with a high visibility in the field of evolutionary computation. We think that new developments can be sparked in an attempt to bring these frameworks together. That is, we aim to provide the possibility to use the API of other metaheuristic frameworks from within the HeuristicLab optimization environment.

A major problem that we have to overcome in this goal is to cross the "language barrier". As described in [8], HeuristicLab is the only framework that is based on the .NET Framework. From the other considered frameworks in their study three are written in C++ and six frameworks are written in Java. This means that we cannot directly reference other frameworks and include them in a HeuristicLab plugin. However, a number of projects exist which make it easier to integrate different languages.

Dynamic Language Runtime

Microsoft laid the foundation for the integration of dynamically typed languages with the .NET Framework 4.0 and the inclusion of the dynamic language runtime (DLR). The DLR and the included hosting infrastructure is the base of the IronPython project. IronPython is an implementation of Python 2.7 for the .NET framework. IronPython was initially started and developed at Microsoft, but has been released to the public and is maintained by volunteers. The developments for IronPython 3 have started only recently. IronPython is available under the Apache 2.0 license which is compatible to the GPLv3 license.

Integration of the DEAP framework can be achieved by creating a transport plugin for the IronPython assemblies as well as the Microsoft.Scripting and Microsoft.Dynamic assemblies. When creating a python script the path to DEAP and the path to the IronPython *Libs* directory have to be added to the list of search paths before importing DEAP modules and running DEAP examples. Of course it is also possible to use the PythonEngine in a C# script to execute Python code. Data exchange between the Python code and the C# part can be achieved through the *PyScope* object which represents the global scope of the Python script and which can be read after executing the *PyEngine*.

We also encountered some problems in using these projects to integrate different frameworks. Using the famous Python libraries NumPy[8] and SciPy[9] in IronPython is rather experimental. We encountered several problems when trying to run DEAP algorithms that depend on NumPy using a .NET port of NumPy.

.NET Java Virtual Machine

A further project that is of high interest for our goals is the IKVM[10] project which is an implementation of Java for the .NET Framework. IKVM provides the ability to convert Java class and jar files to .NET assemblies. These assemblies then reference a port of OpenJDK to the .NET Framework which provides the Java API. IKVM allows both .NET developers to use Java classes and Java developers to use .NET classes. However, Java developers that seek to link against .NET assemblies first have to create stub classes against which Java can compile against. IKVM however does not provide a Java compiler, so e.g. Oracle's Java SDK has to be used. IKVM is public software that is free to use and free to modify. OpenJDK is licensed under GPL and LGPL.

Integration of jMetal could be achieved using IKVM. First we created a .NET assembly using the ikvm compiler *ikvmc*. We excluded classes with external references and which were not necessary for programming against the framework such as the NUnit tests. Then we created a transport plugin for the IKVM and OpenJDK assemblies and also for the generated jMetal assembly. This plugin is available as an add-on[11] to the HeuristicLab development trunk and may be part of a future release. It shall be noted that this enables users only to program against the API of jMetal. The algorithms in jMetal are not available as HeuristicLab algorithms so far. This would require a much deeper integration which is beyond the scope of this article.

C and C++

The integration of C code is generally possible in C# by using platform invoke (PInvoke). Methods of C DLLs can be called by defining those methods in C# as *extern* and attributing them with *DLLImport*. This requires that the method signature given in C# is the same as in the DLL. However, PInvoke cannot be used to marshal native C++ classes and also the use of PInvoke is not without dangers. We have experienced, that an exception inside a native function cannot be catched in managed code and the whole environment crashes immediately and without error notification.

To make native C++ code available within HeuristicLab one would need to write a managed C++ wrapper class for all the native classes. The managed C++ DLL could then be referenced in a C# project, i.e. a HeuristicLab plugin. Another option would be to create *extern "C"* wrapper functions for creating and destroying objects and all of its public methods. An excellent article is available on codeproject[12]. Of course this is much more effort than the approaches stated

[7] `http://jmetal.sourceforge.net/`

[8] `http://www.numpy.org/`

[9] `http://www.scipy.org/`

[10] `http://www.ikvm.net/`

[11] `http://dev.heuristiclab.com/trac/hl/core/browser/branches/jMetal`

[12] `http://www.codeproject.com/Articles/18032/How-to-Marshal-a-C-Class`

```
ManualResetEvent mutex = new ManualResetEvent(false);

public override void Main() {
  var ga = new GeneticAlgorithm();
  ga.MaximumGenerations.Value = 50;
  ga.PopulationSize.Value = 10;
  ga.Problem = new TravelingSalesmanProblem();

  var experiment = new Experiment();
  for (int i = 0; i < 5; i++) {
    experiment.Optimizers.Add(new BatchRun() {
    Optimizer = (IOptimizer)ga.Clone(),
    Repetitions = 10 });
    ga.PopulationSize.Value *= 2;
  }

  experiment.ExecutionStateChanged += OnStateChanged;
  experiment.Start();
  mutex.WaitOne();
  MainFormManager.MainForm.ShowContent(experiment);
  MainFormManager.MainForm.ShowContent(experiment.Runs,
    typeof(RunCollectionTableView));
}

private void OnStateChanged(object sender, EventArgs e) {
  if (((IExecutable)sender).ExecutionState
    == ExecutionState.Stopped) mutex.Set();
}
```

Figure 4: Part of a script that shows GUI automation capabilities. It runs an experiment and displays the relevant results in a bubble chart.

above as the wrappers would need to be written for each framework anew.

Framework Design for Integration

In experimenting and evaluating the interplay of different frameworks we already noticed a few design decisions that make it difficult for frameworks to interoperate. In this section we want to state two specific points that are worth considering. The discussion may seem shallow, but an in-depth examination would require an article of its own. The aim here is to spark a discussion on framework design from the point of view of reuse in other frameworks.

Custom Data Types

The use of custom data types on the one hand eases manipulation in one framework, but is a stumbling block for the integration. Data has to be converted from one data type to the other which requires lengthy pieces of code and challenges run-time performance. In jMetal 4.5 the *Solution* class is a data type that links various parts of the framework such as the *Problem*, *SolutionType*, and *Variable* class. This class is often used in public interfaces of operators. Therefore in an integrated approach one would have to provide an additional implementation of, e.g. the problem, as extensions to jMetal objects. The public interfaces cannot be used unless the solution is reconstructed. In DEAP 1.0.0 there is a creator for dynamically creating classes which many algorithms use to create a class *Individual*. This individual is then the expected contract of many operators also impeding data reuse. In HeuristicLab there are also objects, but often they wrap a single simple datatype such as the *RealVector* or *DoubleMatrix*. Additionally, there are static methods in many places which provide a very simple API. In Figure 3 it can be seen that the problem variable *qap* only encapsulates the weights and distance matrices, but does not contain any logic. Proportional selection is implemented for 'IEnumerable' which is the base of arrays, lists, dictionaries, etc. However, there are also parts in HeuristicLab that still need to be improved. Historically, the objects in HeuristicLab have been designed around the operator paradigm. Some of these operators provide a static interface as well as an interface to the operator execution API, others lack a static API and therefore cannot be used as easily in a script. Recommendation: Create APIs that use the most basic data types available. Compose objects into classes, but decompose them before handing them to relevant functions. This enables better reuse of those functions in other frameworks.

Composition of Algorithms

Additionally, when integrating frameworks, extension by composition is probably the more promising approach than by inheritance. If a framework provides only an algorithm which implements most of the code in a single method it is difficult to inject code from other frameworks and alter or analyze the behavior. The more algorithms can be composed of modular parts of the framework, the more possibilities for combining parts from different frameworks. In DEAP 1.0.0 the algorithms are mostly combinations of simpler operations. The operations are implemented as methods and some of them provide more or less functionality. The methods are linked into operations through a toolbox which combines the method reference and several parameters into a partial function. In contrast in jMetal 4.5 and HeuristicLab 3.3.9 there is an *Algorithm* class that is used for representing algorithms. While the algorithms in jMetal are rather monolithic, the algorithms in HeuristicLab again are composed of smaller operators and can be adapted by modifying the operator graph. Still, it would currently be difficult to integrate a crossover of another framework into a HeuristicLab algorithm. In jMetal, most of the code is grouped within an *execute()* method which is not optimal with respect to code reuse. Recommendation: Use inheritance or sub-classing only when necessary. Instead of deep inheritance hierarchies it would be better to create a shallow layer of full-featured classes that can be extended through composition.

4. EXAMPLES

Not only can scripts be used to perform small tasks or prototype entire algorithms, scripts can also be used to open views and show them to the user. Figure 4 shows a script which creates a new experiment, runs it and displays the results in a table view.

Other kind of automated tasks, which do not require to use the MainForm API could be to load a set of files from a directory, perform some calculation and save that file again. This could be useful if a certain value should be calculated in a run, e.g. the parameter of exponential growth or decay in the quality progress chart. If the quality chart is contained within the run, but this value was not calculated, a script could be used to calculate it later on.

Figure 3 shows an example of a genetic algorithm written in Python that uses the HeuristicLab API to be applied to the quadratic assignment problem (QAP). The code calls static methods which perform the necessary operations for a genetic algorithm: selection, crossover, mutation, and evaluation. We also compared the run-time difference between the Python script and a C# script implementation of that

Table 1: Time measured in seconds to run a genetic algorithm with 10,000 generations and a population size of 100 on different QAP instances.

Problem Size	C# Script	Python Script	Difference
19	4.65	6.51	+40.0%
25	6.92	8.80	+27.2%
36	13.08	14.97	+14.4%
50	24.16	26.45	+9.5%
100	95.12	97.39	+2.4%
150	208.89	211.58	+1.3%

algorithm. The algorithm in Figure 3 was used in the given configuration and applied to problems of different sizes. It can be seen in Table 2 that the Python engine is a little bit slower, but as the problem dimension increases the difference vanishes. All performance tests in this article were conducted on an Intel Core i7 with 2.6 Ghz, the run-times are given in seconds and averaged over 10 executions.

Figure 5 shows an example of using the SPEA2 algorithm defined in the jMetal framework to solve a multi-objective test function problem. The algorithm is executed inside a C# script which will create a new run object after every execution. Again it is interesting to evaluate the performance when that algorithm is executed inside HeuristicLab, when it is executed using IKVM's Java Virtual Machine (JVM) and when using Oracle's JVM. We modified the algorithm given in Figure 5 and measured the time before *settings.configure()* and after *algorithm.execute()*.

Table 2 shows the results of this performance test. Interestingly, a fast way to execute the jMetal SPEA2 algorithm on the ZDT1 multi-objective test function is using the jMetal plugin in HeuristicLab and running it with a C# script. Using Oracle's JVM (version 1.7.0u55 and 1.8.0u5) is comparable to using the IKVM.NET virtual machine (version 7.2.4630.5).

Table 2: Time measured in seconds to execute the SPEA2 algorithm implementation in jMetal 4.5 on the ZDT1 problem using a maximum of 50,000 evaluated solutions.

PopSize	HeuristicLab	IKVM	JVM 1.7	JVM 1.8
100	6.20	6.82	6.67	6.55
500	8.66	9.32	9.34	9.25
1000	13.36	14.16	14.41	14.26

5. CONCLUSIONS

We have described the integration of scripting in HeuristicLab and have evaluated possibilities of integrating metaheuristic optimization frameworks in the HeuristicLab optimization environment which may be a highly interesting path for future development. The reuse of algorithms of different frameworks is still an experimental task which may raise many questions regarding a suitable design of those frameworks. Each framework itself is certainly a well-rounded package, but there is a big potential in combining the strengths of these frameworks in one environment. We have evaluated two frameworks that allow interoperation of C# with Python and Java so that algorithms implemented in these languages can be reused in HeuristicLab. We have further made two performance comparisons, which on the one hand show that for higher dimensional problems the performance difference between C# and Python becomes irrelevant and on the other hand show that compiling the jMetal library with IKVM.NET and running algorithms from a C# script in HeuristicLab is not a slow solution at all. This would make it a feasible option to do performance and results comparison in a common framework.

```
using System;
using System.Linq;
using HeuristicLab.Analysis;
using HeuristicLab.Common;
using jmetal.core;
using jmetal.experiments.settings;
using jmetal.qualityIndicator.fastHypervolume;

public class UserScript
  : HeuristicLab.Scripting.UserScriptBase {
  public override void Main() {
    var settings = new SPEA2_Settings("ZDT1");
    settings.maxEvaluations_ = 50000;
    settings.populationSize_ = 200;
    settings.crossoverProbability_ = 0.8;
    settings.mutationProbability_ = 0.1;
    var algorithm = settings.configure();
    var numObj = algorithm.getProblem().getNumberOfObjectives();
    var solutionSet = algorithm.execute();

    var solutionIterator = solutionSet.iterator();
    var front = new ScatterPlot();
    var row = new ScatterPlotDataRow();
    front.Rows.Add(row);

    while (solutionIterator.hasNext()) {
      var solution = (Solution)solutionIterator.next();
      var objectives = Enumerable.Range(0, numObj)
        .Select(x => solution.getObjective(x)).ToArray();
      row.Points.Add(
        new Point2D<double>(objectives[0], objectives[1]));
      Console.WriteLine(string.Join("; ", objectives));
    }
    vars.front = front;
    vars.HV = new FastHypervolume()
      .computeHypervolume(solutionSet);
  }
}
```

Figure 5: SPEA2 that solves the ZDT1 problem using both the jMetal and HeuristicLab API.

6. ACKNOWLEDGMENTS

The work described in this paper was done partly within the Josef Ressel Centre for Heuristic Optimization (*Heureka!*) sponsored by the Austrian Research Promotion Agency (FFG).

7. REFERENCES

[1] A. Beham, E. Pitzer, S. Wagner, M. Affenzeller, K. Altendorfer, T. Felberbauer, and M. Bäck. Integration of flexible interfaces in optimization software frameworks for simulation-based optimization. In *Companion Publication of the 2012 Genetic and Evolutionary Computation Conference, GECCO'12 Companion*, pages 125–132, Philadelphia, PA, USA, July 2012.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[3] F. Dobslaw. Input: The intelligent parameter utilization tool. In *Companion Publication of the 2012 Genetic and Evolutionary Computation Conference, GECCO'12 Companion*, pages 149–156, Philadelphia, PA, USA, July 2012.

[4] J. J. Durillo and A. J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[5] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[6] J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[7] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham. Abyss: Adapting scatter search to multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 12(4):439–457, 2008.

[8] J. A. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561, 2012.

[9] S. Voß and D. L. Woodruff. *Optimization software class libraries*. Springer, 2002.

[10] S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Johannes Kepler University, Linz, Austria, 2009.

[11] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller. *Advanced Methods and Applications in Computational Intelligence*, volume 6 of *Topics in Intelligent Engineering and Informatics*, chapter Architecture and Design of the HeuristicLab Optimization Environment, pages 197–261. Springer, 2014.

[12] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In K. Giannakoglou et al., editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE), 2002.