INTEGRATED SIMULATION AND OPTIMIZATION IN HEURISTICLAB

Andreas Beham[a, b], Gabriel Kronberger[a], Johannes Karder[a], Michael Kommenda[a, b], Andreas Scheibenpflug[a], Stefan Wagner[a], Michael Affenzeller[a, b]

[a]) Heuristic and Evolutionary Algorithms Laboratory
School of Informatics, Communication, and Media
University of Applied Sciences Upper Austria,
Softwarepark 11, 4232 Hagenberg, Austria

[b] Johannes Kepler University Linz
Institute for Formal Models and Verification
Altenberger Straße 69, 4040 Linz, Austria

[a] andreas.beham@heuristiclab.com, gabriel.kronberger@heuristiclab.com, johannes.karder@heuristiclab.com, michael.kommenda@heuristiclab.com, andreas.scheibenpflug@heuristiclab.com, stefan.wagner@heuristiclab.com, michael.affenzeller@heuristiclab.com

ABSTRACT
Process simulation has many applications that are closely related to optimization. Finding optimal steering parameters for the simulated processes is an activity in which the simulation model is often used as an evaluation function to an optimization procedure. Combining optimization and simulation has been achieved in the past already, however optimization procedures implemented in simulation software are often only black box solvers that are difficult to change, extend or parameterize. Optimization software frameworks on the other hand host a range of suitable algorithms, but often lack the ability to describe and run simulation models. Exchange protocols have been proposed in the past, however the interchange has still proven to be complex and work on simplification is ongoing. In this work, we want to pursue a different approach. We intend to integrate simulation capabilities into an optimization framework and thus want to better support applications for simulation-based optimization. We will describe a suitable generic simulation framework and its integration into HeuristicLab. A case study is presented as a demonstration of its usefulness.

Keywords: simulation, optimization

1. INTRODUCTION

Work on simulation-based optimization procedures is ongoing and has profited a lot from the steady rise in available computational power. Simulation models are complex functions that are time consuming to run. Often there needs to be a warm-up phase of the model until the process performance has been stabilized and the indicators can be computed. This further lengthens the timespan that the model has to cover and increases the amount of data that has to be processed. We have addressed the problem of simulation-based optimization (Affenzeller et al. 2007), its integration into HeuristicLab (Wagner et al. 2014) and further developments to generic and extensible protocols (Beham et al. 2012) with the aim of supporting parallel model evaluation in order to speed up the optimization. In this work the aim is to introduce simulation capabilities into the optimization software environment HeuristicLab. Instead of having to connect the optimization software with the simulation environment we want to allow writing and evaluating models in HeuristicLab more easily, especially those that are tied to optimization problems. This should facilitate the application of simulation-based optimization for smaller projects, academic examples, as well as more complex simulation models. For this purpose we have added a simulation core to the HeuristicLab framework that supports modeling and execution of simple and complex simulation models.

2. RELATED AND PREVIOUS WORK

This field has been actively researched for several years and a number of solutions have emerged. There are many commercial simulation frameworks that feature embedded optimizers, e.g. AnyLogic or FlexSim which include OptQuest (Laguna and Marti 2002). There are optimization frameworks that must be connected with simulation frameworks through generic interfaces such as HeuristicLab or GenOpt (Wetter 2001). Finally, software is available which allows performing simulation and optimization. For instance, MATLAB (http://www.matlab.com) includes SimuLink and also provides optimization algorithms in the global optimization toolbox. SciLab (http://www.scilab.org) is an open source environment for numerical computation that includes the Xcos simulator and features basic optimization algorithms such as genetic algorithms and simulated annealing. ECJ is another Java-based optimization framework that connects well with the

MASON simulation framework, both having the same creators. Finally, a large number of optimization frameworks exist which feature real-valued optimization, multi-objective problems and include classical operations research problems from the domain of production, logistics, and supply chain. However often functionality for integration with simulation environments is missing. A review can be found in (Parejo et al. 2012).

3. INTEGRATED SIMULATION FRAMEWORK

One major requirement for choosing a simulation framework to integrate in HeuristicLab is the programming language. The framework has to be implemented in a language that can be used within the .Net framework. Otherwise, it is required to cross the language boundary by e.g. serializing to a common format which is precisely what was already realized in (Beham et al. 2012). These languages should ideally be native languages to the .Net Framework such as C#, F# or VB.Net.

From the many existing simulation frameworks that seem suited for integration we chose to go with SimPy (Matloff 2008), more specifically the current version SimPy 3, available at http://pypi.python.org/pypi/simpy/. In evaluating the framework we found that the models expressed with SimPy can be short, clear, and self-descriptive. Additionally, researchers have since created more specific layers to the rather slim core, e.g. ManPy "Manufacturing in Python" for manufacturing simulation (Dagkakis et al. 2013) which is developed within the EU FP7 funded project DREAM (http://dream-simulation.eu/). A model's processes can be described in SimPy by writing a program, however the model can be expressed without boilerplate code and without using a complex object-oriented design. As the name suggests the framework is written in Python and can be integrated in HeuristicLab via IronPython (http://ironpython .net/) as described in (Beham et al. 2014). However, we noted that some models are rather slow to execute using IronPython and so we created a C# port of SimPy which we termed Sim# and which is available on GitHub (https://github.com/abeham/SimSharp). An implementation of an m/m/1 queuing model in Sim# is given in Listing 1.

Processes in this framework are defined as .Net iterators, similarly to SimPy where they are described as Python generators. The process runs until a "yield return" statement is reached where it awaits the yielded event. After the event has been completed the process wakes up and continues to run until the next "yield return" statement or terminates. Termination of processes is also an event so that processes can wait on other processes. In the described case in Listing 1 the process is not yielded, thus process *MM1Q* does not wait on its termination. The environment maintains the current time as well as the event queue in which the events are inserted.

The Sim# framework makes it rather easy to describe simple models with very few lines of code - in SimPy it would be even slightly less code as it does not require variable declaration, types or visibility declarations. Having to deal with statically typed languages creates some overhead, however as Sim# models are compiled they are faster to execute, something which is relevant for simulation-based optimization.

```
static Environment env;
static Resource server;
static TimeSpan arrival_time = …
static TimeSpan proc_time = …
static TimeSpan simulation_time = …

static IEnumerable<Event> MM1Q() {
  while (true) {
    yield return env.TimeoutExponential(arrival_time);
    env.Process(Item());
  }
}

static IEnumerable<Event> Item() {
  using (var s = server.Request()) {
    yield return s;
    yield return env.TimeoutExponential(proc_time);
  }
}

public static void Main(string[] args) {
  env = new Environment(randomSeed: 42);
  server = new Resource(env, capacity: 1);
  env.Process(MM1Q());
  env.Run(simulation_time);
}
```

Listing 1: Queuing model implemented with Sim#.

4. SIMULATION OPTIMIZATION PROBLEM

Writing models in code is a very natural way for programmers and software developers. It is also natural for researchers in the field of mathematical optimization to express fitness functions and constraints as code, such as e.g. a linear program expressed in CPLEX. In a similar sense the integration of such models in HeuristicLab and the task of optimizing their parameters should also remain simple and powerful.

The main use of Sim# simulation models will be in place of an evaluation function. Instead of evaluating a mathematical expression for a given solution, the simulation model should be initialized and run. The performance criteria that result from the model can then be used to form a fitness value which should be either maximized or minimized by the algorithm.

To provide such an integration we have extended HeuristicLab with the introduction of a programmable optimization problem. This problem allows the user to specify the *configuration* in terms of binary, integer, real, and permutation parameters that have to be optimized. There can be an arbitrary number of parameters, each of them having a name that needs to be unique for this configuration. Integer and real parameters are defined in a certain interval, and for integer parameters a step size can be configured for coarse discretization. Permutation parameters define a length for each permutation and a type on how it may be interpreted (absolute, relative directed, or relative undirected). In a second method the

user can then specify the evaluation function which receives a *ParameterVector* that acts as a container for this configuration and that allows to access the parameters' values. The evaluation function also receives a random number generator which can be used for example for stochastic simulation models.

The configuration specifies the solution space, while the parameter vector denotes a concrete solution that the algorithm has identified and should be evaluated. An implementation of a function minimization problem in form of a programmable problem is given in Listing 2. In the evaluation function, any kind of C# code can be executed, including the initialization and execution of simulation models which will be the main interest of this paper.

```
public class Fmin : ISingleObjectiveProblemDefinition {
  public bool IsMaximizationProblem {
    get { return false; }
  }
  public Configuration GetConfiguration() {
    return new Configuration()
      .AddReal("r", length: 5, min: -10.0, max: 10.0);
  }
  public double Evaluate(IRandom random,
                         ParameterVector vector) {
    var quality = 0.0;
    quality = vector.Real("r").Select(x => x * x).Sum();
    return quality;
  }
}
```

Listing 2: Implementation of a simple function minimization problem in form of a programmable problem. The parameter configuration and the evaluation function need to be defined.

In addition to the objective function and the encoding, a number of manipulation operations must also be defined that perturb the solution. Different metaheuristics such as genetic algorithms require certain perturbation operators, e.g. crossover and mutation. These operators are automatically configured with default values and can be changed by the user.

This configuration is not trivial however and demands a closer look: The problem class reacts on each change to the script in that it tries to compile the script, instantiate the class and retrieve the configuration from the instance. For each of the parameters defined in this configuration, the problem will then create a solution creator and lookup the relevant operators. It will configure the solution creators in HeuristicLab to create the variable with the specified name and all the operators to modify that variable. There are two distinct cases to mention:

1. A single parameter type is chosen, e.g. only a vector of real values.
2. Multiple parameter types are combined, e.g. a vector of real values and a binary vector.

This distinction is important as in the first case the problem may be solved by algorithms which can handle the specific encoding. For a real vector, e.g. the CMAES implementation in HeuristicLab could be applied. In the second case the algorithms would need to support a combined encoding which is realized currently only for variants of genetic algorithms.

5. CASE STUDY: SUPPLY CHAIN

Supply chain management is concerned with the flow of goods. Along the supply chain excessive demands and backlog should be avoided, inventory costs and service times should remain low. The famous beer game (Hammond 1994) allows students to explore the effects of demand propagation in a supply chain that consists of 4 echelons and fixed customer demand at the end of the chain. The demand is usually constant for a few periods, but raises after a while. Typically the game is played by students on a table where they should experience the bullwhip effect (Lee et al. 1997) which means that the orders along the chain increasingly start to fluctuate. The factory has to cope with order sizes that bear no relation to the customer demand. This in turn creates a very costly supply chain as inventory levels and backlogs become very high. In this case study, the computer will control the orders using the well-known (S, s) order policy (Scarf 1993) and the parameters will be adjusted by the optimization algorithm. A bad setting of the parameters could result in high costs.

The supply chain consists of 4 echelons which are the factory, the regional distributor, the wholesaler, and the retailer. In each stage the orders will be placed to the next stage. The retailer orders from the wholesaler which in turn orders from the distributor which again orders from the factory. Customers arrive at the retailer and attempt to buy the goods or place an order in case the goods are not available. An order will have to be served as one delivery, order splitting and partial delivery is not considered.

The (S, s) order policy controls the minimum stock level at which an order will be placed (s), as well as the maximum amount of stock that should be kept (S). The difference between S and s basically determines the order size, although already placed orders and the backlog is taken into account. At any given time, if Stock + ExpectedDeliveries – BacklogOrders becomes smaller or equal to s a demand arises which will be placed to the next stage in the supply chain.

The model, as implemented in Sim# is given in Listing 4. Again, an advantage of Sim# and also SimPy is the textual representation of models which makes it easy to include them together with the publications, e.g. in an appendix if the model is larger.

```
class SupplyChain : SimSharp.Environment {
  public Echelon Factory;
  public Echelon Distributor;
  public Echelon Wholesaler;
  public Echelon Retailer;
  private bool mainSeason;

  public SupplyChain(DateTime start, IntegerVector sS)
    : base(start) {
    Factory = new Echelon(this, 15, null,
                          sS[3], sS[3] + sS[7]);
```

```
    Distributor = new Echelon(this, 15, Factory,
                              sS[2], sS[2] + sS[6]);
    Wholesaler = new Echelon(this, 15, Distributor,
                              sS[1], sS[1] + sS[5]);
    Retailer = new Echelon(this, 15, Wholesaler,
                              sS[0], sS[0] + sS[4]);

    Process(Season());
    Process(Simulation());
  }

  IEnumerable<Event> Simulation() {
    for (int i = 0; i < 50; i++) {
      Process(Round());
      yield return TimeoutD(1);
    }
  }

  IEnumerable<Event> Round() {
    var demand = mainSeason ? 9 : 5;
    var beer = Process(Retailer.Deliver(demand));
    Retailer.CalculateCosts();
    Wholesaler.CalculateCosts();
    Distributor.CalculateCosts();
    Factory.CalculateCosts();
    Process(Retailer.Order());
    Process(Wholesaler.Order());
    Process(Distributor.Order());
    Process(Factory.Order());

    yield return beer;
  }

  IEnumerable<Event> Season() {
    yield return TimeoutD(5);
    mainSeason = true;
  }
}

class Echelon {
  SupplyChain env;
  Echelon supplier;
  double s, S;
  public Container Stock;
  public double BacklogOrders, BacklogDeliveries;
  public double BacklogCosts, InventoryCosts;

  public Echelon(SupplyChain env, double initialStock,
Echelon supplier, double s, double S) {
    this.env = env;
    this.Stock = new Container(env, initial: initialStock);
    this.supplier = supplier;
    this.s = s;
    this.S = S;
  }

  public IEnumerable<Event> Order() {
    var newStock = Stock.Level + BacklogDeliveries -
                   BacklogOrders;
    if (newStock <= s) {
      var d = S - newStock;
      if (d > 0) {
        if (supplier != null) {
          var shipment = env.Process(supplier.Deliver(d));
          BacklogDeliveries += d;
          yield return shipment;
          var delivery = (ContainerGet)shipment.Value;
          Stock.Put(delivery.Amount);
        } else { // factory
          BacklogDeliveries += d;
          yield return env.TimeoutD(2); // produce
          Stock.Put(d);
        }
      }
      BacklogDeliveries -= d;
    }
  }

  public IEnumerable<Event> Deliver(double amount) {
    BacklogOrders += amount;
    var delivery = Stock.Get(amount);
    yield return delivery;
    BacklogOrders -= amount;
    yield return env.TimeoutD(2); // transport
    env.ActiveProcess.Succeed(delivery);
  }

  public void CalculateCosts() {
    BacklogCosts += BacklogOrders;
    InventoryCosts += 0.5 * Stock.Level;
  }
}
```

Listing 3: The model of the beer game supply chain implemented in Sim#. This is the full model description, the s and S parameters are given in an IntegerVector which is a HeuristicLab datastructure.

6. OPTIMIZATION

The solution to this parameterization problem can be described in terms of 8 integer parameters. We chose the first 4 items to represent s while the last 4 items denote $S - s$. This is chosen in order to avoid a constrained optimization problem, because obviously it must hold that $s \leq S$. The bounds have been chosen as [2;100] for the first 4 items and [0;100] for the last 4 items. As a step size we chose to go with 2 so that only even values would be evaluated. We have a set up a genetic algorithm to optimize these parameters using a population size of 100. We then proceeded to vary several of the other parameters of the genetic algorithm in order to identify a good combination. These parameters and their variations are shown in Table 1. Because the parameters are integers and some of these operators are known from the real-valued domain, the results are rounded to the nearest feasible integer (also taking step size into consideration).

Table 1: Parameter variation for the genetic algorithm.

Parameter	Variation
Elites	0, 1
Crossover	Discrete, BLX-0.75-0.25, Heuristic, 1-point
Mutator	Uniform, Normal (σ=2)
Selector	Roulette, 2-Tournament
Muation rate	0.05, 0.15

The total number of variations was 64 and each possible configuration was evaluated 10 times. The experiment was run on an Intel Core i7 with 4 physical (8 virtual) cores each running at 2.6 Ghz. Using parallel evaluation of the simulation models an average run of the GA took about 17 seconds to evaluate in which around 100.000 simulation runs have been performed. Executing

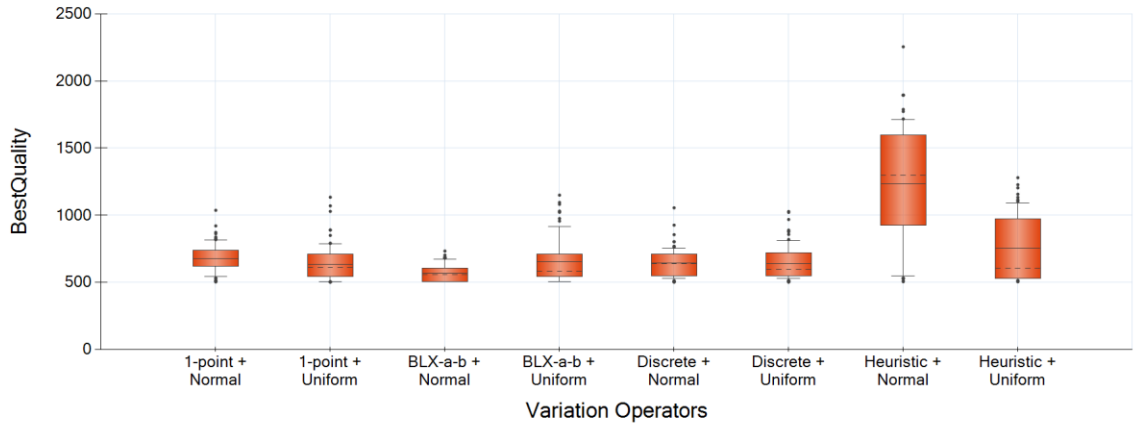

Figure 1: Boxplot of different combinations of crossover and mutation operators for optimizing the order policy parameters.

the algorithm sequentially took about 45 seconds. Parallelizing the evaluation thus presented a nice speed up which HeuristicLab offers out of the box.

We did find a strong dependence on the performance with respect to the crossover and mutator operator that is used. Over the different settings for Elites, Selector, and mutation rates the combination of BLX-a-b and Normal mutation proved most favorably. Figure 1 shows results from this experiment. Each combination of crossover and mutator is given on the x-axis. Because normality cannot be assumed in the present data we used non-parametric statistical significance tests to further evaluate these results. The Kruskal-Wallis analysis of variance is a non-parametric variant of ANOVA which can be used if there is a significant difference among all tested groups. For our experiment we obtained a p-value of 3.01E-30, which suggests a significant difference in the groups. A two tailed Mann-Whitney-U test on the best and second best combination results in a p-value of 0.0004 which also indicates a statistically significant difference.

7. CONCLUSIONS

We have shown a new and different approach of mating simulation and optimization in a common framework. The introduced optimization problem allows to define a more complex encoding. We have shown a simple, but powerful and fast simulation framework that can be used to describe models which are run inside an evaluation function.

ACKNOWLEDGMENTS

The work described in this paper was done within the K-Project Heuristic Optimization in Production and Logistics (HOPL) sponsored by the Austrian Research Promotion Agency (FFG).

REFERENCES

Affenzeller, M., Kronberger, G., Winkler, S., Ionescu, M., Wagner, S., 2007. Heuristic Optimization Methods for the Tuning of Input Parameters of Simulation Models. In *Proceedings of I3M 2007*, DIPTEM University of Genova, 278-283.

Affenzeller, M., Winkler, S., Wagner, S., Beham, A., 2009. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*. Chapman & Hall/CRC. ISBN 978-1584886297.

Beham, A., Pitzer, E., Wagner, S., Affenzeller, M., Altendorfer, K., Felberbauer, T., Bäck, M., 2012. 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*, July, Philadelphia, PA, USA, 125-132.

Beham, A., Karder J., Kronberger, G., Wagner, S., Kommenda, M., Scheibenpflug, S., 2014. Scripting and Framework Integration in Heuristic Optimization Environments. In: *Companion Publication of the 2014 Genetic and Evolutionary Computation Conference, GECCO'14 Companion*, July, Vancouver, CA.

Dagkakis, G., Heavey, C., Robin, S., Perrin, J., 2013. ManPy: An Open-Source Layer of DES Manufacturing Objects Implemented in SimPy. In: *Proceedings of the 2013 8th EUROSIM Congress on Modelling and Simulation*, September, Cardiff, Wales, UK, 357-363.

Hammond, J.H., 1994. "Beer Game, The: Board Version." Harvard Business School Background Note 694-104, June 1994 (revised October 1999).

Laguna, M., Marti, R., 2002. The OptQuest callable library. In *Optimization software class libraries* 193-218. Springer US.

Lee, H.L., Padmanabhan, V., Whang, S., 1997. The Bullwhip Effect in Supply Chains. *Sloan Management Review* 38 (3), 93–102.

Matloff, N., 2008. Introduction to Discrete-Event Simulation and the SimPy Language. *Davis, CA. Dept of Computer Science. University of California at Davis.* Available from: http://heather.cs.ucdavis.edu [accessed 9 July 2014]

Parejo J. A., Ruiz-Cortés A., Lozano, S., Fernandez, P., 2012. Metaheuristic optimization frameworks: a

survey and benchmarking. *Soft Computing*, 16(3), 527-561.

Scarf, H., 1993. The Optimality of (S, s) Policies in the Dynamic Inventory Problem. *Optimal pricing, inflation, and the cost of price adjustment*, 49-56.

Wagner, S., Kronberger, G., Beham, A., Kommenda, M., Scheibenpflug, A., Pitzer, E., Vonolfen, S., Kofler, M., Winkler, S., Dorfer, V., Affenzeller, M., 2014. Architecture and Design of the HeuristicLab Optimization Environment. In *Advanced Methods and Applications in Computational Intelligence, Topics in Intelligent Engineering and Informatics Series*, 197-261. Springer

Wetter, M., 2001. GenOpt - A generic optimization program. In *Proceedings of the 7th IBPSA Conference*, 601-608, August, Rio de Janeiro, Brazil.

AUTHORS BIOGRAPHY

ANDREAS BEHAM received his Master in computer science in 2007 from JKU Linz, Austria, and is a research associate at the Research Center Hagenberg. His research interests include metaheuristic methods applied to combinatorial and simulation-based problems. He is a member of the HeuristicLab architects team.

GABRIEL KRONBERGER received his PhD in engineering sciences in 2010 from JKU Linz, Austria, and is a professor at the campus Hagenberg. His research interests include genetic programming, machine learning, and data mining. He is a member of the HeuristicLab architects team.

JOHANNES KARDER is pursuing his master study of software engineering at the University of Applied Sciences Upper Austria and is a research associate at the Research Center Hagenberg. He is a member of the HeuristicLab development team.

MICHAEL KOMMENDA received his Master in bioinformatics in 2007 from the University of Applied Sciences Upper Austria, and is a research associate at the Research Center Hagenberg. His research interests include genetic programming and data mining. He is a member of the HeuristicLab architects team.

ANDREAS SCHEIBENPFLUG received his Master in software engineering in 2011 from the University of Applied Sciences Upper Austria, and is a research associate at the Research Center Hagenberg. His research interests include parallel and distributed computing. He is a member of the HeuristicLab architects team.

STEFAN WAGNER received his PhD in technical sciences in 2009 from the Johannes Kepler University Linz, Austria. He is a professor at the University of Applied Sciences Upper Austria, Campus Hagenberg. He is the project manager and head developer of the HeuristicLab optimization environment.

MICHAEL AFFENZELLER has published several papers, journal articles and books dealing with theoretical and practical aspects of evolutionary computation, genetic algorithms, and meta-heuristics in general. In 2001 he received his PhD in engineering sciences and in 2004 he received his habilitation in applied systems engineering, both from the Johannes Kepler University of Linz, Austria. Michael Affenzeller is professor at UAS, Campus Hagenberg.