Simulation-Based Optimization with HeuristicLab: Practical Guidelines and Real-World Applications

Michael Affenzeller, Andreas Beham, Stefan Vonolfen, Erik Pitzer, Stephan M. Winkler, Stephan Hutterer, Michael Kommenda, Monika Kofler, Gabriel Kronberger and Stefan Wagner

Abstract Dynamic and stochastic problem environments are often difficult to model using standard problem formulations and algorithms. One way to model and then solve them is simulation-based optimization: Simulations are integrated into the optimization process in order to evaluate the quality of solution candidates and to identify optimized system configurations. Potential solutions are evaluated with a simulation model, which leads to new challenges regarding runtime performance, robustness, and distributed evaluation. In order to design, compare, and parameterize algorithmic approaches it is beneficial to use an optimization framework for algorithm design and evaluation. On the one hand, this chapter shows how arbitrary simulators can be coupled with the open-source HeuristicLab optimization framework. This coupling is implemented in a generic way so that the simulators act as external evaluators. On the other hand, we demonstrate how arbitrary optimizers available within HeuristicLab can be called from a simulator in order to perform complex optimization tasks within the simulation model. In order to illustrate the applicability of these approaches, real-world examples investigated by the authors are discussed. We show here application examples from different fields, namely logistics network design, vendor managed inventory routing, steel slab logistics, production optimization with dispatching rule scheduling, material flow simulation, and layout optimization.

M. Affenzeller (✉) · A. Beham · S. Vonolfen · E. Pitzer · S.M. Winkler · S. Hutterer · M. Kommenda · M. Kofler · G. Kronberger · S. Wagner
Heuristic and Evolutionary Algorithms Laboratory, School of Informatics, Communications and Media, University of Applied Sciences Upper Austria, Research Center Hagenberg, Softwarepark 11, 4232 Hagenberg, Austria
e-mail: michael.affenzeller@fh-hagenberg.at

M. Affenzeller · A. Beham · S. Vonolfen · S. Hutterer · M. Kofler
Institute for Formal Models and Verification, Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria

M. Mujica Mota et al. (eds.), *Applied Simulation and Optimization*,
DOI 10.1007/978-3-319-15033-8_1

1 Introduction

The field of simulation optimization [11, 16, 17, 20] is still a rather young flavor in operations research. One of its key enablers is the efficient utilization of parallel computing infrastructures which allows modeling and optimizing not only toy problems, but also more complex real-world scenarios from the field of production and logistics as well as other domains. So far, metaheuristic optimization approaches have been applied successfully in solving combinatorial optimization tasks such as vehicle routing, production scheduling, and layout optimization. However, when using standardized problem formulations, only singular aspects of the real-world can be modeled and optimized in a quite restrictive way—which is often not capable to represent the real-world and its complex interrelations and constraints appropriately.

Discrete event simulation approaches allow modeling complex and interrelated production and logistic scenarios in a more sophisticated and realistic way. However, the optimization capabilities of recent discrete simulation packages [17] are still quite limited and rather aimed to offer robust "broadband" optimizers which are not capable to explore the full optimization potential of concrete scenarios.

The approach presented in this chapter aims to couple a powerful meta-heuristic optimization framework offering a huge variety of optimization algorithms with diverse simulators acting as evaluators of solution candidates in a generic way. By this means, the user shall be enabled to choose and parameterize an appropriate optimization method in order to explore more optimization potential compared to when using built-in solvers (when available). Pursuing this approach we adhere to the *no free lunch theorem of optimization* [57] which postulates that a general purpose, universal optimization strategy cannot be implemented and that the only possibility for a strategy to outperform another one is to be more specialized to the structure of the tackled problem.

The generic approach described in this chapter is to couple diverse specific simulation models representing real-world scenarios with the open-source heuristic optimization framework HeuristicLab [53]. Google protocol buffers act as a generic interface between optimization algorithms and concrete simulation models that here act as an external evaluator.

From an algorithmic point of view, the main challenges of the proposed approach are algorithm selection and parameterization, runtime consumption, robustness, and stability of calculated solutions. One of the major issues in this context is the runtime consumption aspect: When solving combinatorial optimization problems, the evaluation of a solution candidate usually only takes small fractions of seconds, whereas in the context of simulation-based optimization the evaluation of a solution candidate might take several seconds or even minutes. For the combination of optimization and simulation it is necessary to scale back the complexity of both simulation and optimization to obtain good results in reasonable time. The requirement of this balance has led to a renaissance of optimization methods that require fewer evaluations such as evolution strategies [40] and simulated annealing [27].

Stochastic elements of the simulation model create the need for robustness analysis of solution candidates. This requires multiple evaluations and enhanced multi-objective fitness functions that also take into account robustness and stability of solution candidates. These additional requirements motivate on the one hand the use of massively parallel computing infrastructures for solution evaluation, and on the other hand the application of enhanced techniques for algorithm selection and parameterization in order to choose the appropriate degree of greediness of the algorithm, which is the basis for exploiting the achievable optimization potential.

This chapter is structured in the following way: Sect. 2 describes the technical basis and implementation of the coupling between simulation models and algorithms offered by HeuristicLab. Thus, for the reader this section also presents guidelines on how to couple simulation models with optimization frameworks. Section 3 summarizes several concrete application example scenarios, in which simulation-based optimization has been used in combination with HeuristicLab for solving real-world problems. Finally, Sect. 4 summarizes this chapter and points out the future research topics and challenges in the field of simulation-based optimization.

2 Methodology and Approach

Over the last decade, a great deal of research has been devoted to couple simulation models with optimization. In order to meet the increasing demand for optimization of simulation model parameters, commercial simulation software packages frequently offer integrated optimization, for example, OptQuest® [41] or the WITNESS® Optimizer [39].

These commercial software solutions frequently apply metaheuristic algorithms for optimization, for instance, genetic algorithms, evolution strategies, tabu search, simulated annealing, scatter search, or hill-climbers. Optimization interfaces usually show only a limited set of tunable algorithm parameters in order to simplify the user interface. These interfaces often do not expose characteristics of the optimization run, e.g., convergence behavior, but focus on statistical analysis of optimization results. This black box approach is certainly favorable with respect to robustness and usability of an embedded optimization tool, but it also limits the potential of the applied optimization method. Optimization environments on the other hand provide a more complex user interface, more algorithms, and allow analyzing the algorithm behavior in more detail in order to improve results or convergence speed.

But, parameter optimization is only one possibility for simulation-based optimization. As far as an embedded optimization approach is concerned commercial packages are not in widespread use. Simulation models that encounter decision problems which would be suited for optimization will have to include their own algorithms or link optimization frameworks into the model. We aim to describe in this section common interaction patterns between simulation and optimization, the HeuristicLab software architecture that is suitable for this kind of optimization, and the interfaces that mate simulation and optimization.

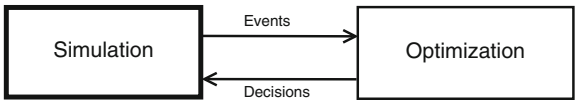

Fig. 1 Interaction pattern for control optimization. The simulation model (here shown *bold*) is the initiating part

2.1 Interaction Patterns Between Simulation and Optimization

Generally, two main interactions patterns, control optimization and parametric optimization can be identified [20]:

- **Control Optimization**: The optimization problem might arise within the simulation model, a decision has to be made given the state of the model.
- **Parametric Optimization**: The simulation model might act as a fitness function, which will take a number of parameters and calculate the resulting fitness value.

The main difference between these patterns is the role of the initiating part that steers the control flow. In this section we will also describe a third pattern which can be seen as a combination of the other two. The application of HeuristicLab has been successful for these patterns [38, 49] of which real-world examples will be given in Sect. 3.

Control Optimization

Control optimization is schematically depicted in Fig. 1. The optimization here is concerned with decision making in changing and uncertain environments. The simulated problem scenario changes over time as new events emerge and previous decisions get executed. It is not possible to undo or change decisions that have been implemented already. Sometimes, a rolling time horizon is allowed to plan ahead, in other cases only the actual situation may be taken into account for making decisions. This category can thus also be described as *online optimization*.

The simulation can be seen as a placeholder for a real-world environment and models the dynamics of the real system. The optimization or decision-making procedure has to react to these dynamics. Eventually, events from the simulation environment can be replaced with events from the real system. The simulation is mainly used in place of the real system to test and validate new optimization approaches. Experiments in the real system would be too costly and too slow to evaluate.

Parametric Optimization

Parametric optimization is illustrated in Fig. 2. A candidate solution (usually a parameter vector) is passed to the simulation which then returns a quality value by running the simulation model using the given parameter set. This approach can be applied when, for example, a closed-form representation of the evaluation function is not feasible because it contains complex stochastic elements or dynamic interactions. Especially, the application of metaheuristics has proven fruitful in this context [45].

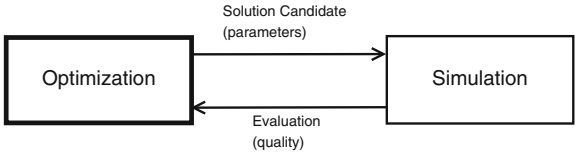

Fig. 2 Simulation-based parametric optimization. The optimization model is the initiating part (here shown *bold*)

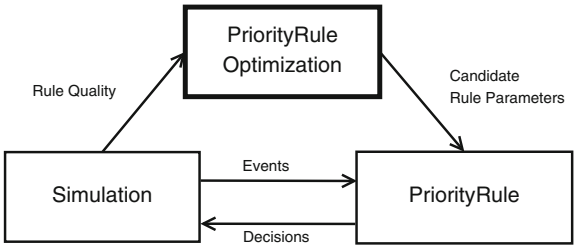

Fig. 3 Generation of priority rules combining parameter and control optimization. The control flow is steered by the priority rule optimization (here shown *bold*)

The candidate solutions are iteratively improved by the metaheuristic algorithm, often totalling a large number of simulation runs. This category can also be described as *offline optimization*. For stochastic simulation models the quality has to be seen as a random variable and often multiple simulation runs of the same parameter set need to be performed to estimate the expected quality.

Generation and Parameterization of Heuristic Policies

This approach combines parametric and control optimization. Policies can be seen as a control strategy that can be used for making decisions online. The simulation model uses such a policy according to the control optimization pattern. When events require a decision, the policy is called to make that decision. The policies themselves are improved offline using the quality that is returned by the simulation model. Policies should be designed such that they can efficiently compute the next decision based on the current situation in dynamic environments. Examples of such policies are priority rules that combine and weight different domain features to rank and prioritize a number of alternative decisions. Different representations for priority rules are possible, for example, vector and tree representations. Figure 3 illustrates this pattern schematically.

2.2 Software Architecture

HeuristicLab[1] [53] is a software environment for heuristic and evolutionary algorithms, developed and successively applied by members of the *Heuristic and Evolutionary Algorithms Laboratory*[2] since 2002. Being licensed under the GNU GPL,[3] HeuristicLab has a growing community of researchers and practitioners using the tool in a wide range of scientific as well as commercial areas. It provides a vast number of already implemented algorithms and problems for optimization and data analysis, an experiment designer, and support for algorithm and results analysis. Furthermore, a sophisticated graphical user interface distinguishes HeuristicLab from other heuristic optimization frameworks [35], which usually require substantial programming skills to extend algorithms and apply them to a given problem. HeuristicLab offers not only programming-based extensibility, but also allows to add and modify algorithms and problems using the graphical user interface and a graphical algorithm modeling language. In HeuristicLab, algorithms are described as operator graphs where an operator represents a node and the connections denote the execution flow. Changing or rearranging operators can be done by drag-and-drop without actually writing code [54]. The framework thereby enables users and practitioners to perform complex tasks such as algorithm development. The possibility to extend the framework on the code level remains, and software engineers benefit from the plugin-based architecture (Fig. 4) allowing them to develop custom algorithms, data structures for solution representations, or custom optimization problems. This has led to a significant level of code reuse across metaheuristic variants and gradually gives users an understanding of algorithm development [52, 55].

Base and Core Layer

The base layer contains plugins that provide essential functionality required by all other plugins of the above layers. Every plugin in HeuristicLab is based on the *PluginInfrastructure* which loads plugins and checks their dependencies. The base layer also includes the *Persistence* which allows to save and load files.

The core layer is situated atop the base layer and includes the algorithm modeling language. It contains core interfaces, data objects, parameters, operators, and engines. The algorithm modeling language uses operators to describe small, individual steps in an algorithm [52]. Algorithms are created by chaining together these operators. This is called an operator graph and *engines* are used to execute that graph by applying one operator after another sequentially. In general, the operators process data that is stored in the memory of an algorithm which is represented in form of a scope tree. Each scope can hold several variables, such as a quality value, the current iteration of an algorithm, or a complex data type like a solution candidate. If it contains subscopes, it can also represent a population. Operators can be applied on any level in the scope tree and may modify its structure as well as read and write variables. Some

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

[2] http://heal.heuristiclab.com.

[3] https://www.gnu.org/copyleft/gpl.html.

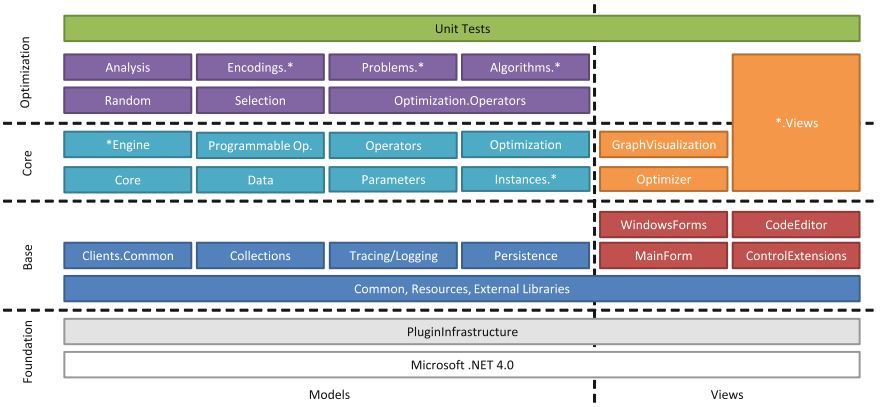

Fig. 4 Block diagram of the architecture of HeuristicLab with a separation into different logical layers. HeuristicLab is composed of a number of plugins each containing a defined set of functionality. Generally, each of the boxes in this figure represent a plugin, or multiple plugins if a "*" is added to the name. The *horizontal layers* are *base*, *core*, and *optimization*. The *vertical layers* show the separation between *models* and *views*. In the block diagram, a plugin always depends on the plugins on the layer below as well as the plugins to its *left* on the same layer

operators allow the application of subsequent operators on a number of sub-scopes in parallel. The *ParallelEngine* is able to execute those operators in different threads and further engines exist that allow to make use of distributed computing resources, such as HeuristicLab Hive [53]. Therefore, HeuristicLab provides an easy way to incorporate data-based parallelism into algorithms. To obtain the values of variables, operators specify *parameters* which can either directly contain a value or provide only the name of a variable which is used to find a match among the algorithm's parameters, the problem's parameters, and the scope tree. An example of such an operator is an evaluator that is applied on a solution scope. Evaluators typically read those variables that contain the solution encoding, those that provide the problem data and, after computing the fitness, add a quality variable to the scope. Similar to any other operator, it could also contain additional parameters that would, e.g., read the algorithm's state such as the current iteration or a collection which acts as another memory.

Optimization Layer

The topmost layer in the architecture includes the algorithms, problems, different standard encodings, and various other plugins for algorithm analysis and random number generation. HeuristicLab is shipped with several algorithms such as genetic algorithm [31], evolution strategy [10, 21], offspring selection genetic algorithm [2], local search, simulated annealing [27], tabu search [19], particle swarm optimization [26], and many more. Among the list of available problems in HeuristicLab are real-valued test functions, combinatorial problems such as the traveling salesman, vehicle routing, and the quadratic assignment problem, as well as data analysis problems such as regression and classification. The optimization layer also contains

analyzers that allow to study the performance of algorithms. Basic analyzers provide a quality progress, more sophisticated analyzers enable an inspection of the algorithms' behavior.

2.3 Interfacing with Simulation

In this section we describe three different strategies for performing simulation-based optimization with HeuristicLab. These strategies differ from each other with respect to their effort to set up the optimization and their possible applications.

- The first strategy (Sect. 2.3.1) is to create interfaces that couple HeuristicLab with specific simulation frameworks such as MATLAB or Scilab.
- Another possibility is to define a general inter-process communication protocol for data exchange that allows the coupling of arbitrary software with HeuristicLab (Sect. 2.3.2).
- The third strategy is to implement the simulation model within HeuristicLab, which results in the tightest coupling between the simulation model and the optimization algorithm (Sect. 2.3.3).

2.3.1 Specific Interface

A possibility to couple a simulation environment with an optimization framework is to provide a specific interface layer which is responsible for handling the communication between the optimization algorithm and the simulation model. Most simulation frameworks provide several ways to couple them with other applications using various kinds of technologies, ranging from direct calls to specialized application programming interfaces (API), component object model (COM) interfaces for interprocess communication, or web services. A drawback of specific interfaces is that they have to be implemented for each simulation environment and technology. However, they can be implemented in a generic way to execute arbitrary commands instead of running a specific simulation model.

HeuristicLab provides specific interfaces for MATLAB and Scilab out of the box. Both these frameworks excel at numeric computation, which makes them especially suited to perform continuous simulation. Furthermore, both frameworks provide specific modules to ease the development of simulation models. The interfaces for these two software systems have been implemented in a generic way allowing the execution of arbitrary scripts in the respective programming language. The MATLAB interface is based on the COM technology to enable communication with HeuristicLab, whereas the Scilab interface calls directly a native `C++` API.

A common problem in continuous simulation is the identification of appropriate parameter values to adapt the simulation model to the circumstances of the real-world—a use case is detailed in Sect. 3.6. Therefore, real-vector encoded parameter optimization problems with a coupling to MATLAB and Scilab have been imple-

```
//parameter assignment
p.m  = param1;
p.d1 = param2;
p.Fc = param3;

//import and execution of simulation model
importXcosDiagram('simulationModel.zcos');
xcos_simulate(scs_m,4);

//read original data and compute quality metric
originalValues = csvRead("data.csv",",",".");
simulatedValues = simulationModel.values(:,2);
quality = sum((values(:,1) - original(:,3))^2);
```

Fig. 5 Example of an evaluation script implemented in Scilab. The simulation is started using the defined model and the parameters given by the optimization algorithm (`param1` – `param3`), the quality is calculated as sum of squared differences between the output of the simulation model and previously measured values

mented in HeuristicLab. The evaluation of a solution candidate uses the aforementioned specific interfaces and executes a script in the simulation environment that calculates the quality of a solution candidate. As a result, every algorithm that can handle real-vector encoded solutions, such as for example evolution strategies, evolutionary algorithms, or simulated annealing, can be used to solve such parameter optimization problems.

A benefit of the chosen strategy is that the effort for configuration and programming is minimized for the user. The only part that has to be provided by the user to run the optimization is the evaluation script; the user can create this script in a familiar environment (MATLAB or Scilab) and does not have to learn the specifics of the HeuristicLab framework to execute the optimization. However, this minimal configuration effort comes with a price, namely that by default only real-valued parameters can be optimized with this approach.

Figure 5 shows an evaluation script for Scilab that parameterizes and runs a simulation model. Solution candidates, in this case combinations of real numbers (`param1` – `param3`), are generated by the algorithm and the quality of the parameter combination is calculated by the simulation framework. The here shown script loads an existing simulation model, sets its parameters to the values supplied by the algorithm, runs the simulation model, and finally extracts the quality of this given parameter vector by converting results of the simulation to a numerical value.

2.3.2 Generic Interface

A generic exchange protocol has been integrated in HeuristicLab that enables communication with external processes, such as simulators, and allows the encoding of several types of parameters. The protocol has been first described in [8]; in this section a summary will be given. It has also been enhanced with a cache which prevents the execution of simulation runs for solutions that have already been evaluated [38]. Since the execution of simulations can consume relatively large amounts of runtime,

```
message SolutionMessage {
  message DoubleVariable {
    required string name = 1;
    optional double data = 2;
  }
  message DoubleArrayVariable {
    required string name = 1;
    repeated double data = 2;
    optional int32 length = 3;
  }
  //... further sub-messages omitted ...
  required int32 solutionId = 1;
  repeated IntegerVariable integerVars = 2;
  repeated IntegerArrayVariable integerArrayVars = 3;
  repeated DoubleVariable doubleVars = 4;
  repeated DoubleArrayVariable doubleArrayVars = 5;
  repeated BoolVariable boolVars = 6;
  repeated BoolArrayVariable boolArrayVars = 7;
  repeated StringVariable stringVars = 8;
  repeated StringArrayVariable stringArrayVars = 9;
  repeated RawVariable rawVars = 10;
}
message QualityMessage {
  required int32 solutionId = 1;
  required double quality = 2;
  extensions 1000 to max;
}
```

Fig. 6 Definition of the generic interface messages in .proto format [8]

parallelization is a major aspect. The generic interface allows to specify several target machines that are running the given simulation with provided parameter settings and returns the results. The evaluation is thus distributed and allows even longer running simulation models to be optimized in reasonable time. The generic interface has been integrated in the form of a customizable problem definition in HeuristicLab and is explained in this section.

External Evaluation Problem

As the name implies, instances of this problem type have to be evaluated by an application that is external to HeuristicLab. This problem has no preconfigured representation or operators, but it can be customized. The *RealVectorEncoding* plugin contains operators that can be added if the simulation exposes real-valued parameters for optimization; if the parameters are discrete values, operators of the *IntegerVectorEncoding* plugin can be used to create and optimize the solutions. These encodings can also be used in combination if there are simulation parameters of both types.

Interoperability

In HeuristicLab, an evaluator that is applied on a solution scope calculates the solution's quality and adds this quality back to the scope. The evaluator of the *ExternalEvaluationProblem* collects a user specified set of variables, adds them to a *SolutionMessage*, and transmits this message to an external application. The evaluation operator then pauses and awaits the reply in form of a *QualityMessage*.

The quality value given in this message is then stored into the corresponding scope. This generic definition allows the use of many algorithms that are designed to optimize single-objective problems in HeuristicLab. To encode and transmit messages, the protocol buffer[4] framework has been used; the messsages' structure is shown in Fig. 6.

The protocol buffer format is designed to work with very compact files that are serialized, which minimizes transmission time. Furthermore, the serialization itself is also very quick. Google provides implementations of protocol buffers for Java, C++, and Python, and open-source ports also have been created for other languages such as C#, R, and many others.[5] The solution message buffer is a so-called "union type" protocol buffer, which is a very generic message for potentially unknown use cases. It includes fields for storing Boolean variables, integers, doubles, and strings as well as arrays of these types. In HeuristicLab, the *SolutionMessageBuilder* class translates the variables in a scope into variables in a *SolutionMessage*; this message builder can use custom converters for transcoding custom data types into a field of the solution message.

Parallelization and Caching

Parallelization is an effective means to reduce overall runtime if the necessary time to run a simulation model becomes very long. The overhead of the communication and the optimization procedure then become a negligible part. In HeuristicLab this is supported through the use of the above-mentioned *parallel engine*. This engine allows multiple evaluators to be executed concurrently, which in turn make use of multiple channels defined in the *Clients* parameter. In the background the *ThreadPool* in .NET is used to provide threads for efficient operations. To further decrease runtime an *EvaluationCache* and the respective evaluator can be used that hashes each visited solution and prevents further simulation runs. The cache can later be persisted to a file or exported as a comma-separated-values (CSV) file for further analysis [38].

Protocol Extensions

The *QualityMessage* can also be extended if more results or variables are included in the solution scope and shall be read and interpreted by an analyzer. This extension of the quality message can be achieved by creating a new message which extends the *QualityMessage*. Field numbers 1000 and higher can be used to declare extension fields. Figure 7 shows a .proto message definition that adds another field storing the number of repetitions.

Example Interface with AnyLogic 6

AnyLogic[6] is a simulation environment implemented in Java that allows to add Java code in various parts of the modeling process. In general, users are able to create model as *ActiveObjects* that might contain other *ActiveObjects*. A model can then be

[4] http://code.google.com/p/protobuf.

[5] http://code.google.com/p/protobuf/wiki/ThirdPartyAddOns.

[6] http://www.xjtek.com.

```
message MyResponse {
  extend HeuristicLab.Problems.ExternalEvaluation.QualityMessage {
    required int32 repetitions = 1000;
  }
}
```

Fig. 7 Extension of the quality message to return also the number of repetitions that have been performed

run in different experiments such as a *SimulationExperiment*. To couple the model with HeuristicLab and make use of the generic data-exchange interface, a special type of experiment is used. In AnyLogic the so-called *ParametersVariation* experiment allows to perform a set of simulation runs for certain parameters. These parameters can be varied automatically given certain bounds and a step size, and they can also be varied freely by an external program such as HeuristicLab. For this purpose a Java library *HL3ExternalEvaluation.jar* was added to the model, which is also available on the HeuristicLab website.[7] This Java library abstracts the data-exchange part and allows to set up the simulation model either as a push or a poll service for HeuristicLab. When choosing the push service, the model needs to implement an interface which is passed to the library; when opting for the poll service, the library can be polled for incoming solution messages and subsequently a quality can be returned.

2.3.3 Integrated Simulation and Optimization

While the methodologies presented in Sects. 2.3.1 and 2.3.2 are dedicated to parameter optimization of models that have been created using external simulation environments, an alternative approach is to implement simulation models directly in HeuristicLab and integrate them with optimization algorithms. The HeuristicLab architecture is generic in the sense that not only optimization algorithms, but arbitrary algorithms including simulations can be modeled. The applicability of this approach has been shown especially in the context of dynamic vehicle routing and various practical case studies [48, 50, 51].

The direct implementation of simulation models in HeuristicLab allows a tight coupling with optimization algorithms, which is beneficial in cases where efficient or sophisticated interactions between simulation and optimization are required. Integrative approaches require that the simulation framework and the optimization framework share a common platform or programming language. For HeuristicLab, this means that .NET based simulation frameworks such as Repast.net[8] or Sim#[9] should be utilized. This allows using the HeuristicLab API in simulation models and vice versa.

[7] http://dev.heuristiclab.com/howtos.

[8] http://repast.sourceforge.net.

[9] http://github.com/abeham/SimSharp.

When tackling control optimization problems an efficient and flexible integration is needed. In this case standard simulation software usually cannot be applied out of the box and only few generic and extensible modeling infrastructures exist for control optimization, e.g., dynamic vehicle routing [36]. Implementing control optimization problems with HeuristicLab requires that the simulation model references the HeuristicLab plugins. When running the model, the events that require a decision parameterize the corresponding optimization problem, the solver algorithm, and await the result. This can be done in a state where the simulation model is paused, or, in a real-time simulation, while the simulation model continues to run. The added complexity of real-time operation, such as the occurrence of changes while the optimization is running, has to be taken into account. Real-time simulation optimization requires a very tight combination of simulation and optimization that may require implementing customized optimization algorithms.

In parametric optimization problems, where the runtime of the simulation model is a critical and limiting factor, an integrated simulation and optimization approach reduces the inter-process communication overhead. Implementing such an approach requires the definition of a customized problem in HeuristicLab with a custom evaluator. In order to evaluate a candidate solution, the evaluator makes use of the simulation framework API, initializes the model, executes it, and creates the fitness values out of the model's performance indicators [7].

3 Real-World Examples

3.1 Simulation-Based Design of a European-Wide Logistics Network for Bio Residues

3.1.1 Problem Description

Increasing prices of fossil fuels and other nonrenewable energy sources have led to an increased interest in the development of alternatives. On the one hand, renewable energy sources, such as energy crops, are employed more often. On the other hand, we see an increased use of so-called second-generation bio-fuels which can be obtained by processing organic residues such as straw, wood chips, lifestock waste, or malt spent grains that do not compete with other food crops and have a more limited impact on greenhouse gas emissions [15, 25]. When processing these waste products, two problems are solved at the same time: Waste amounts are reduced, and precious resources are replenished. The inherent problem of this idea, however, is that it is typically rather uneconomical to further process or transport waste products, which is why they are considered waste in the first place, i.e., the effort invested in their transport is bigger than the expected revenue. The key to the mitigation of this problem is to increase the value density of these products by de-central and cheap

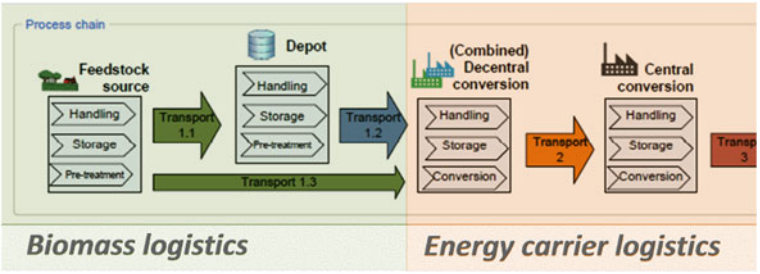

Fig. 8 Echelons in the logistics network

"upgrading" to intermediate energy carriers which can be transported over longer distances more economically.

This is why we have developed an optimized multi-echelon logistics network for the transport of feedstock, intermediate, and final products on a large scale within BioBoost.[10] In this research project, biomass potentials and key data on conversion facilities available in Europe have been compiled; this information forms the basis for planning a large logistics network. Suitable conversion plant locations and capacities as well as transport routes and product amounts are then optimized using simulation-based optimization: Many scenarios are iterated and evaluated, and metaheuristics are used to tune free variables so that the quality of the resulting logistics network is optimized. The approach described in this section encompasses several modeling optimizations to enable faster calculations.

3.1.2 Simulation Model

Several free variables are optimized: At each location, a certain *amount of feedstock* has to be obtained which is then *transported* to a *certain plant* for de-central processing. The intermediate energy carrier is then *transported* to a central plant where it is *converted* into heat, fuel, or other end products. This overall process is shown in Fig. 8.

The initial solution space size in a naive model would be determined by the number of free variables. In the case of the logistics network, the following factors have to be considered:

- Locations of feedstock sources and the utilization levels for each of these sources. In this case more than 1000 level 3 NUTS regions [14] have been used as possible source locations.
- Locations of the intermediate and central processing plants placed on any combination of more than 1000 regions.

[10] http://www.bioboost.eu.

- Connections of sources and targets in each echelon of the logistic network. The logistics network can be modeled as an adjacency matrix containing the links between source and target locations. This matrix would then have a size greater than $s = 1000 \times 1000$. Moreover, every subset of connections will then be a valid solution candidate giving a total of $s! = 8 \times 10^{5565708}$ possible routing networks for each type of feedstock.

The large size of this original solution space renders a direct solution of the problem infeasible. For this reason it was attempted to model the solution space so that it remains susceptible to optimization by providing a meaningful neighborhood definition, and at the same time removing as many unnecessary scenarios as possible.

3.1.3 Optimization

A very strong simplification can be made by allowing only one target region per source region. This immediately reduces the number of choices per feedstock to approximately 10^9. However, the solution space for one complete scenario still comprises at least two echelons and at least three different feedstock types each of which contains a choice for source and target region, their connectedness, and the amount of acquired feedstock. Therefore, the solution space size is still in the area of around 10^{21} per product.

A second reduction of the solution space size can be achieved by eliminating variables or variable choices that would lead to solutions that can be guaranteed to be inferior. One naive possibility would be to allow only transport targets that are directly adjacent to the source region. The variant we have employed was to automatically select plant locations and capacities based on the transport targets of different feedstock types as this further reduces the number of free variables and, simultaneously, the solution space size to 10^{15} possibilities per feedstock type. After these transformations, the resulting solution space is manageable with current metaheuristic optimization methods.

In a third round, computationally expensive calculations of routes between source and target locations have been replaced with precalculated estimates to reduce the computation time. Furthermore, a speed-up has been achieved through aggregation into yearly calculations instead of step-wise event-driven simulation [30].

Finally, while the solution space is perceived as fixed for the whole optimization process, some combinations of variable choices can lead to meaningless, equal, or inferior results. For this purpose we have developed a mechanism that dynamically reduces the solution space only during the application of variation operators, hiding inferior options, which again leads to a significant runtime reduction.

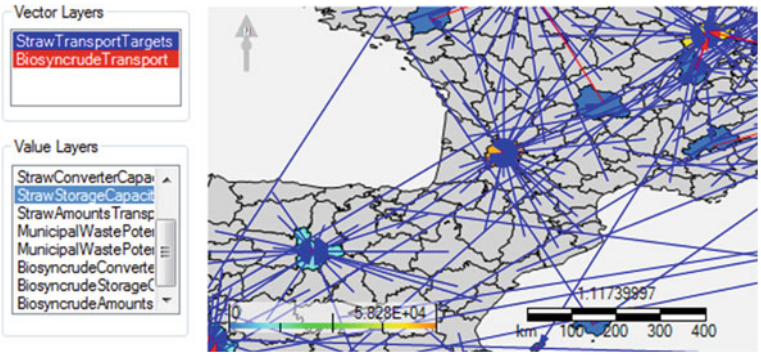

Fig. 9 Solution visualization within HeuristicLab

3.1.4 Conclusion

The simulation model has been implemented based on the HeuristicLab optimization environment [52, 53] where the evaluation of a two-echelon scenario takes about 1 ms on average for one feedstock type and meaningful optimization results are available after 2–12 h when executing the optimization on a single computer with a Core 2 Duo processor. Figure 9 shows a screenshot of the implementation in HeuristicLab with a visualization of the feedstock utilizations and transport vectors.

Using only a few simplifications and the powerful and flexible optimization algorithms available within HeuristicLab, an optimization task that initially seemed intractable has been reformulated to allow its optimization and might help to reduce the amount of waste while increasing the amount of energy available in the future.

3.2 Simulation-Based Priority Rule Optimization for Scheduling Production Systems

Problem Description

Scheduling plays a key role in industrial systems to ensure the efficient use of resources and timely completion of orders. Briefly summarized, a scheduling problem can be described as a set of jobs, each having a collection of operations that are tied to a machine. By specifying the start time of each operation a schedule is constructed. A number of different types of shops have been described in the literature such as job shop, flow shop, and open shop [37]. In the flow shop problem, the sequence of operations is the same for all jobs, while in the job shop, the sequence may be different for each job. In the open shop there is no predefined sequence. Generally, scheduling problems such as the job shop scheduling problem are NP hard as shown in [18, 37].

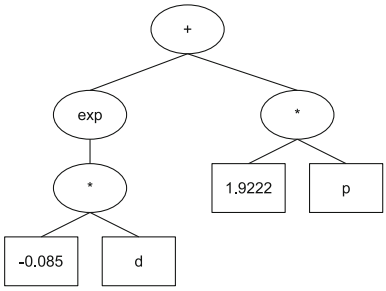

Fig. 10 Example of a tree-encoded priority rule. The variables d and p represent job properties, e.g., due date, processing time, batch size

When solving scheduling problems one can make a decision very early and *plan ahead* or rather late and *decide in time* as in the saying "we'll cross that bridge when we get to it." The first case can be seen as an *offline* approach to determine the optimal schedule in advance. This yields a high quality schedule, but requires high runtime and adaption in the face of changing conditions. The second case is denoted as an *online* approach, which considers the actual state of the system and which thus has to to make decisions much more quickly [3]. One popular method in such an online approach is the use of simple rules [34]: First-in-first-out, earliest-duedate-first, and many more have been proposed to rank and prioritize the pending order queue. A combination of simple heuristics to more complex priority rules, as can be seen in Fig. 10, allows creating tailored and customized rules for specific scheduling scenarios.

Simulation Model

Simulation is particularly suited to support the optimization of complex priority rules. When a machine becomes idle all items in its queue are ranked and the best ranked item is processed next. If a job is finished the performance metrics are updated. At the end of the simulation run the remaining jobs are also rated; this is important as otherwise "problematic" jobs, e.g., that require long setup, would potentially starve in the system and never get picked up.

The model itself describes the flow of the jobs, the entities available in the production system, as well as the interactions between them. Workers arrive at the production plant in the morning and pick up work. They will process jobs on the machines, make a pause, go for lunch, and continue to work. In the simulation model the decision of which item the workers are to pick up next is made using the priority rule as follows:

1. The set of possible decisions is constructed;
2. For each decision a dictionary is created that contains state variables, item characteristics, and more, see Table 1 for examples;
3. An interpreter reads the priority rule and computes a rank using the variables in the dictionary;
4. The set of decisions is sorted according to their rank;
5. The best ranked decision is implemented.

Table 1 State variables of a production system [38]

p…batch size of an order	n_r…number of remaining steps
p_r…remaining batch size after this step	l…queue length of a machine
d…job due date	s…setup required (± 1)
q…job quantity	Q…qualification
t…number of required tools	t_r…remaining processing time

Some situations require that multiple decisions should be made at the same time, e.g., selecting a worker and an item that she should process. This can be useful if the worker qualification or worker satisfaction is taken into account. In this case all combinations of workers and items constitute the set of possible decisions. However, as the evaluation of a decision is not without cost, this provides an additional performance hit and prolongs the simulation run. In the concrete studies in [5, 38] the interpretation of the priority rules was compiled to Microsoft intermediate language code prior to running the model in order to speed up the simulation runtime.

The performance metrics are translated into a fitness of the priority rule by a rating model. The formula may vary depending on the case, but timeliness, troughput, cash flow, or waste production are important factors. For instance, a linear step function can be designed to give a slight penalty for jobs that are finished too early and a heavier penalty for delayed delivery. It is advised to avoid highly discontinuous or flat transformation functions as this creates a very rough or very flat search space that may be difficult to optimize.

Optimization

Typically, the enterprise resource planning system of a production company provides the necessary data such as jobs, working plan, bill of materials, resources, due dates, processing times, and more to parameterize the simulation model. The importance of separating this data in training and test scenarios must be emphasized. Optimized rules may become highly specific to the scenario for which they have been trained and may not generalize very well. Simulating the optimized rules with data from test scenarios allows identifying generalizable rules that show good performance in both training and test. Additionally, maintaining an archive of rules allows exploring the Pareto front between complexity and quality; simple rules may not appear to perform well in the training, but might generalize better [38]. Such an archive may also be used to avoid re-evaluation of already known solutions. Simulation models of such production systems can become quite complex and, therefore it takes a few seconds to finish a run of several weeks of the production plant. Distributed evaluation of the simulation model is quite important in order to obtain good priority rules in reasonable time.

Using genetic programming [4, 29] tree-based priority rules can be generated and evolved to match the scenarios at hand. For this purpose trees are generated randomly and crossed with other trees in a population; better trees have a higher chance of being used for crossover, and therefore their features will prevail in the

following generations. While genetic programming traditionally employs a standard genetic algorithm, variants such as offspring selection genetic algorithm [1, 2] have worked quite well. Offspring selection introduces an additional selection step that discards offspring that are inferior to their parents.

Conclusion

Production systems benefit from suitable decision rules that are tuned to improve long-term goals such as timeliness or throughput. These rules take some time to optimize, but are quick to evaluate in the production system. Volatility in those systems is a concern that can be addressed through a continued reoptimization of these rules and by maintaining a comprehensive set of production scenarios that show many different characteristics. In practical implementations it is also important to assure that the inputs to the rules are available to the rule and that the data is accurate. For instance, if queue length is a highly relevant variable in such a rule, but it cannot be obtained in the real system the priority rule cannot be implemented. However, it would not be wise to omit these variables altogether as it might indicate that additional sensory data should be acquired.

3.3 Simulation-Based Optimization of Inventory Replenishment Rules

This section is based on a study that was previously published by Vonolfen et al. [48] and deals with the simulation-based generation of inventory replenishment rules for stochastic inventory routing problems. The evolved rules are evaluated and tested in the context of retailing based on real-world data with a large number of different products that are replenished at supermarkets. The methodology is an example of simulation-based priority rule generation using an integrated simulation optimization approach (as outlined in Sect. 2) applied to a real-world scenario.

Problem Description

The inventory routing problem (IRP) integrates inventory management and transportation. It is a mathematical model for the concept of vendor managed inventory (VMI) where the vendor has the responsibility for the replenishment of the customers, which requires information about the inventory levels to be available. According to Waller et al. [56], VMI was first popularized by Walmart and then implemented in various other companies. The goal is to minimize the inventory and transportation costs while maintaining a certain service level.

The IRP was first presented by Bell et al. [9] who considered the distribution of industrial gases; since then, many variants of the IRP have been studied. Moin and Salhi [32] provide an overview where they state that most problem formulations do not consider stochastic demand patterns and are deterministic. In contrast, the stochastic IRP (SIRP) considers product usages as probability distributions, but this also increases the problem complexity, which motivates a simulation-based approach.

The considered IRP here is a mixed formulation in which some of the customers choose VMI, while the other customers keep an order-based strategy. Multiple

products P are distributed from a central depot to a set of order-based customers (O) and a set of VMI-customers (N) using a homogeneous fleet of vehicles (V) with a known capacity C_v for each vehicle $v \in V$. Each VMI-customer $n \in N$ has a known storage capacity C_{np} for each product. The planning process is performed on discrete time steps t, which are days in real operation.

A joint probability distribution P^w_{np} is given for each weekday $w \in [0..6]$, customer n and product p from which the product consumption can be sampled for a given time step during the simulation. The inventory level x^t_{np} at a customer of a certain product can be measured every day. The order-based customers place fixed orders $d^t_{op} \in D^t_f$ which they derive from an ordering strategy that is not influenced by the vendor.

Each day, the vendor replenishment d^t_{np} and customer orders d^t_{op} are combined into vehicle routes R^t. The objective is to minimize the required vehicle fleet size $|M|$ as well as the driven distance $d = \sum L_r$ while maintaining the service level by preventing out-of-stock situations where x^t_{np} falls below a certain safety stock.

The motivation to apply simulation-based priority rule generation stems from the high problem complexity resulting from the practical case study performed in cooperation with an Austrian retailer. The considered scenario consists of 84 supermarkets which are served from a central depot. In total, the supermarkets serve 5113 different fast-moving consumer goods that have stochastic demand distributions with high fluctuations during a week. The supermarkets are served with a fleet of homogeneous vehicles. A main challenge is to flatten the peaks in the resource usage and to balance it more evenly to achieve constant resource usage. This is complicated by the very limited storage capabilities at the individual supermarkets, while each individual product contributes to the service quality since out-of-stock situations may lead to a potential loss of revenue.

Simulation Model

The simulation model developed for this inventory replenishment problem is an agent-based formulation consisting of vendor, customer, and vehicle agents. We have implemented the simulation model based on Repast.net[11] agent-based simulation framework and integrated it into HeuristicLab.

The interactions between the agents are illustrated in Fig. 11: Each customer agent has an inventory which is updated by sampling from the demand distributions. Depending on whether the customer has a vendor-managed inventory or places the orders by itself, different interactions with the vendor occur. In the case of VMI, the vendor uses a replenishment policy to determine the replenished demands. In the case of order-based customers, the customer uses an order policy and the vendor has no access to the inventory. In that case, a classical threshold-based order strategy is used which keeps a certain security buffer.

For each day of operation, the accumulated demands are converted into a standard capacitated vehicle routing problem (CVRP) instance which can be solved with any VRP algorithm available in HeuristicLab. The calculated tours are then executed by

[11] http://repast.sourceforge.net.

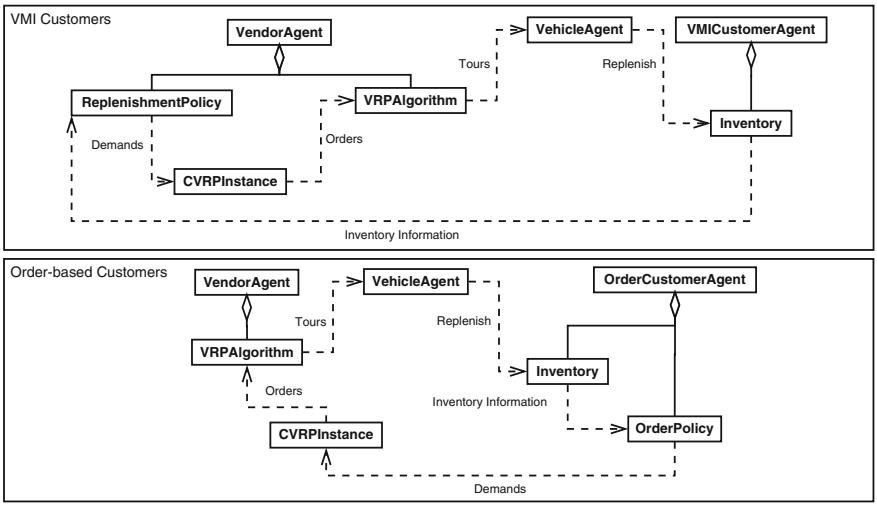

Fig. 11 Interactions between the agents for VMI and order-based customers

vehicle agents which execute the tours and replenish the inventories of the customers. This results in a two-stage approach where first the replenished goods are determined and then the tours are calculated for each day of operation.

Optimization

The aim is to automatically evolve inventory replenishment policies that are able to balance the resource usage with fluctuating demands maintaining a given service quality by preventing out-of-stock situations for each individual product. In order to reach this goal we apply simulation-based generation of priority rules as described in Sect. 2.

The inventory replenishment policy has the main goal of constant resource utilization with fluctuating demand distributions. It consists of two priority rules: The first rule is responsible for choosing a set of customers that should be visited, the second rule determines the amount of replenishment for each product at these customers. This two-stage approach aims at taking into consideration both the routing (e.g., avoiding to visit customers that are geographically far away) and replenishment (e.g., avoiding out-of-stock situations). The description of the policies is based on Vonolfen et al. [48] where the reader is referred to for details.

The replenishment policy is represented as a real-valued vector that consists of general parameters and parameters of the first and second priority rule. The two general parameters are *CapacityUtilization* and *PriorityThreshold*. The capacity utilization parameter determines the capacity that should be used over time to replenish the customers. This capacity is used as a basis for the replenishment rule and aims at constant resource utilization. The *PriorityThreshold* parameter determines the minimum priority a customer must have to be considered for replenishment to avoid unnecessary detours.

The first priority rule calculates a priority for each customer n to be replenished by weighting $m = 6$ factors f_{ni} with an parameterized weight a_i to a priority score $p_n = (\sum_{i=1}^{m} f_{ni} * a_i)/m$. The factors considered for a given customer n are (as discussed in detail in [48]):

f_{n1} : The minimum expected amount of days a stock-out will occur
f_{n2} : The average expected amount of days for stock-outs
f_{n3} : Number of days since the last delivery
f_{n4} : Total inventory size
f_{n5} : Minimum detour to incorporate the customer in existing routes
f_{n6} : Geographic isolation

The second priority rule determines the amount of a product that should be delivered to a given customer. If the expected days a customer would run out of a product falls below a certain *RefillThreshold* (derived from the stochastic product consumption rate information) or the available stock falls below a certain *RefillBarrier* (predefined amount of safety stock) it is refilled to a certain level determined by the *RefillFactor* which defines this level as a ratio of the maximum storage capacity for the respective product.

In total, the parameter vector for the replenishment strategy to be optimized contains 11 parameters: *CapacityUtilization*, *PriorityThreshold*, $a_i (1 \leq i \leq 6)$, *RefillThreshold*, *RefillBarrier*, and *RefillFactor*. These parameters are optimized using an evolution strategy as an priority rule optimization algorithm. Each candidate solution is evaluated by running a simulation run of a time frame of 60 days and evaluating the resource usage as well as the service quality. For each day of the simulation, the replenishment amounts are calculated using the parameterized replenishment strategy and the resulting tours are optimized using a routing algorithm (push-forward insertion heuristic) implemented in HeuristicLab. After each day, the inventory level at each customer is reduced by using the predefined demand distribution.

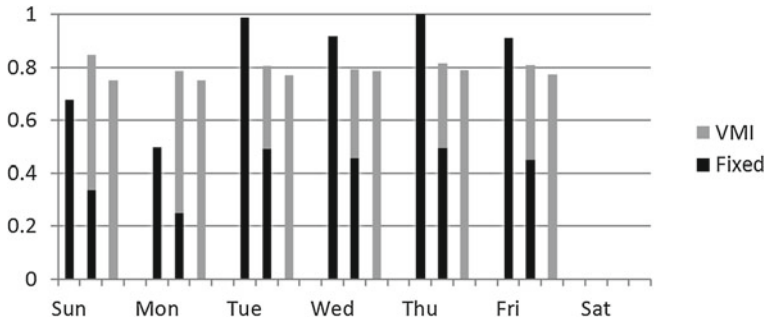

Fig. 12 Average resource utilization over different weekdays for different scenarios [48]. The x-axis represents the weekday and the y-axis the relative resource utilization. The resource utilization is divided for the fixed and VMI customers

Conclusion

As illustrated in Fig. 12, we are able to balance the resource utilization using a vendor managed inventory. When the customers place the orders themselves using a classical threshold-based order strategy, the fluctuations in demands over weekdays lead to a corresponding fluctuation in resource usage, which is undesirable for determining needed capacities. The biggest smoothing effect could be achieved if all customers chose to apply a VMI; however, even if only part of the customers are switched to a VMI, this effect can also be observed.

As a practical guideline for applying simulation optimization it can be concluded that simulation-based priority rule generation is a powerful method for making operational decisions in high-dimensional and stochastic problem environments. By modeling different scenarios and evaluating the optimization potentials, simulation-based optimization aids in making tactical and strategical decisions.

3.4 Simulation Optimization of Transport Activities Within Steel Slab Logistics

This section deals with transport optimization within steel slab logistics and is based on a previously published study of Vonolfen et al. [50]. The main aim is to evaluate optimization potentials in the transportation of steel slabs in terms of throughput maximization within cold-charge. The core of the approach is a detailed simulation model considering constraints of the cold-charge steel production process. According to the classification presented in Sect. 2, it is an example of parameter optimization where the parameters are the transport sequence in this case and the simulation model is integrated with the route optimization algorithm in HeuristicLab.

Problem Description

Steel production is a multi-stage process and is generally geographically distributed and energy as well as capital intensive [43]. The typical production process starts with raw materials and the melted steel produced in the furnace is transformed into slabs at the continuous casting machine. The slabs are then rolled into plates or coils in the rolling mill. Generally, there are two pathways for steel slabs. In the hot-charge process they are transported directly from the caster to the rolling mill, while in the cold-charge process a slab yard is used as an intermediate buffer storage.

This work focuses on scheduling the transport activities within the cold-charge process which are linked to the lifecycle of a steel slab illustrated in Fig. 13. Within cold-charge three transport activities can be identified: transportation from the caster to the slab yard, transportation to processing aggregates, and transportation to the rolling mill. Scheduling the individual transport activities is not a trivial task, since upstream and downstream processes of steel production have to be considered and transport links them together.

In the presented case study, straddle carriers transport the slabs and the activities are scheduled by a human expert. The straddle carriers can carry up to 105 tons

which usually correspond to around four slabs. An outside slab yard is used as an intermediate storage which decouples casting and rolling. The slab yard consists of several fields which are organized in lanes each containing several stacks of slabs. If a slab that lies beneath other slabs is retrieved, shuffling operations have to occur. At the continuous caster as well as at the processing aggregates and the rolling mill, stacks are retrieved and stored in stacks at handover places.

Simulation Model

Simulation optimization is applied to evaluate the optimization potential in scheduling the three types of transport activities optimizing total throughput while considering all relevant operational constraints. The motivation to use a simulation model are the dynamic interactions between the individual activities which would be difficult to represent as a static model. There are several operational constraints for the individual transport activities (cf. [50]):

- Shuffling constraints concern the retrieval and storage of slabs at handover places and the slab yard. No shuffling is possible at handover places which means only the topmost slabs at handover stacks can be retrieved. In the slab yard, only a single straddle carrier can operate at an individual handover place.
- Rolling constraints ensure the correct transportation of slabs to the rolling mill. A certain security buffer of slabs scheduled to be rolled has to be present at the rolling mill which cannot be underrun. Additionally, the rolling sequence has to be followed to a certain degree, otherwise the cranes at the rolling mill have to reshuffle the slabs.
- Temporal constraints concern availability of straddle carriers where each straddle carrier has scheduled maintenance tasks as well as driver breaks. Additionally, the processing schedule has to be considered which means slabs have to arrive earlier than their scheduled processing time.
- Capacity constraints are that each carrier can transport up to 105 tons and slabs must have similar dimensions to be transported together.

The simulation is carried out in discrete time steps where each step represents a minute of operation. The three types of events that occur are actions performed by straddle carriers, update of handover places by simulated movements caused by upstream and downstream activities and the rolling of slabs using the predefined rolling sequence.

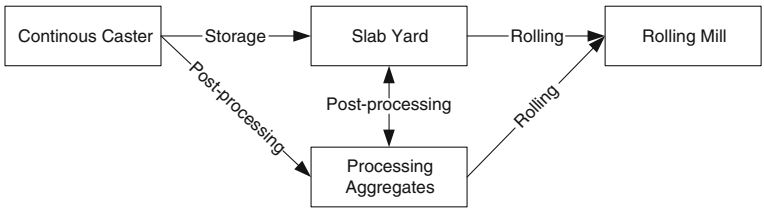

Fig. 13 Lifecycle of a slab in the cold-charge steel production process (cf. [50])

Optimization

For the optimization of the transport schedules, a unified tabu search heuristic is used which can be applied to several routing problems and was presented for pickup and delivery problems by Cordeau and Laporte [12]. By neighborhood exploration it systematically improves the current solution using a tabu list to prevent cycling. The basic neighborhood operation is moving a request to another vehicle at the best possible insertion position. Moving back the request to the original vehicle is made tabu for a given number of iterations.

The quality of a given schedule is evaluated with a simulation run where key values concerning throughput and constraint violations are considered. The key values are combined in a single quality value (cf. [50]):

$$\begin{aligned} quality = {} & travelTime + shufflingTime \\ & + \alpha * shufflingViolations + \beta * rollingViolations \\ & + \gamma * temporalViolations + \delta * capacityViolations \end{aligned} \tag{1}$$

The objective is to maximize the total throughput and thus to minimize the total travel and shuffling time of the straddle carriers. Constraint violations are penalized using penalty factors which are adapted during the search process. If the current solution is feasible, the penalty is decreased and if it is infeasible it is increased. This allows the algorithm to move through infeasible regions of the search space.

During an iteration, a large number of possible insertion positions have to be evaluated. Since a detailed simulation is very time-consuming a combination of a static evaluation and a simulation evaluation is used. The full simulation is only run for interesting insertion positions while the static evaluation serves as a lower bound. However, the static evaluation does not consider the dynamic interactions which include the shuffling constraints and the interactions between the straddle carriers (locking of rows). In preliminary experiments, the best found tradeoff between runtime and achieved quality was to evaluate the best 10 insertion positions for each neighborhood operation using a full simulation.

Conclusion

After exporting 10 historical shifts and comparing the optimized schedule with the schedule created by the domain expert, and by analyzing the optimized schedules, the main bottleneck that was identified were the shuffling operations in the slab yard. The algorithmic solution works around this problem by performing more trips to the storage area and thus avoiding stacking and shuffling operations. The effort to create temporary stacks, which are required when picking multiple slabs at once from the yard, is reduced significantly. The reduction of capacity utilization is compensated by minimizing the time traveled empty by sequencing the individual trips more efficiently.

To tackle the main identified bottleneck of shuffling operations, an efficient storage assignment is needed that considers the rolling schedule [28]. Also, when creating the

rolling schedule the current storage assignment in the slab yard should be considered to reduce shuffling which is known as the slab stack shuffling problem [44]. In the long term, a holistic model should be created that combines the optimization of casting, transport, storage, and rolling due to strong interdependencies.

In terms of practical guidelines, this real-world case study has shown that bottlenecks in the process can be identified by applying simulation optimization. The optimized schedules are evaluated and compared with original schedules using a simulation model to identify potential process improvements. When considering a process where the individual activities are strongly interconnected, it makes sense to create a holistic model instead of optimizing the activities independently. A possible approach is to integrate several individual simulation and optimization models [51].

3.5 Material Flow Simulation and Layout Optimization

Problem Description

The design of manufacturing plants is a complex process where several criteria determine the feasibility and suitability of a certain arrangement. The floor layout contains the storage zones, paths, and workstations required to store, transport, and process the materials into intermediate or end products. Simulating the processes on the computer presents several insights onto the performance of the future plant and may guide the planners' decisions. Often, a large amount of data is available when rearranging the internals of an existing plant which can be obtained from enterprise resource planning (ERP) systems. In designing new plants from the ground up, some assumptions will have to be made. A good overview of facility layout problems is given in [13].

Simulation Model

Common to most ERP systems is the notion of a *job*, which is split into one or more *operations* which in turn demand zero or more *resources*. The outcome of a job is either an end or intermediate product which is either shipped to the customer or placed in the company warehouse. Operations describe basic tasks and are executed in a certain order. Operations are not limited to production tasks; they may also include management tasks such as monitoring or coaching [6]. In a simulation model the material flows can be calculated by taking into account the production plan, that is, the start and processing times of the operations and their assignment to a machine. In rearrangement tasks it is possible to rely on past data if the future outlook is similar, but quite a large amount of data must be accommodated as the time period can stretch over one or several years. In a busy plant this means that hundreds of thousands operations have to be considered.

Three different kinds of flows can be identified when running a simulation model that can later be combined to form a similarity matrix between the locations [6]:

1. A *sequential flow* occurs when there is a transition from one operation to the next.

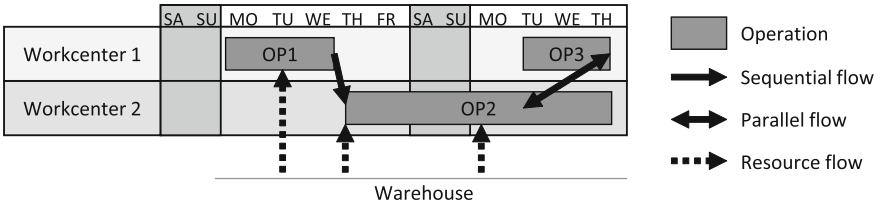

Fig. 14 Examples of sequential, parallel, and resource flows on three operations in a job

2. A *parallel flow* arises when an operation starts while another operation of the same job is still running. Parallel flows are less common than sequential flows, but they can indicate strong collaboration if an assembly part requires the cooperation of several work centers.
3. A *resource flow* occurs when an operation demands materials. This demand can then be satisfied directly through a flow from the last operation of a producing job or by an unknown source, typically the warehouse.

These flow types are visualized in Fig. 14. Sequential and parallel flows represent the flows of products or intermediate products and are passed on or shared between operations. Resource flows represent the flows of items used in the manufacturing of the products. In a layout optimization scenario the sequential and parallel flows would indicate a closeness due to the sequence in the manufacturing process. Resource flows indicate requirements for buffer capacities or closeness to the storage area of which they receive the material. However, resource flows also indicate supplier-producer relationships within the plant.

The strength of a flow is calculated as the sum of weighted transitions between two operations. These weights can be the occurrence or the number of materials that are transported or a number of containers. A special case during flow simulation also occurs in 1:N and N:1 transitions, i.e., when an operation has several possible successors or predecessors. In such cases it is possible to specify additional data such as process flowcharts or to either split and combine or duplicate the handling events. An appropriate assumption depends largely on the problem scenario. It is necessary to discuss and decide on these possibilities in the preparation stage if the strength of the flow should be a valid approximation of the closeness between two work centers.

Optimization

The faced optimization problem here is to arrange a set of rectangular shapes R on a flat surface G such that they lie completely within a boundary polygon P with p_i being the points of the closed polygon. The problem further contains the set B of fixed blocks, which are immobile locations in the layout. Each shape in R represents a work center and is specified by the location of the center coordinates, the dimensions of the rectangle, and a rotation, e.g., in 90° intervals. Each shape in B is specified by the lower left and upper right points. Finally, the matrix F specifying the flow strength is given as a $N \times N$ matrix with $N = |R|$. Elements of this matrix are called f_{ij} and denote the strength of the flow from i to j.

A solution to this problem specifies the location as x and y position, the width w, and rotation of each shape in R. The solution thus can be encoded in the form of multiple vectors of integer values if G is assumed to be discrete. Two vectors $\mathbf{x}$ and $\mathbf{y}$ encode the location on the plane, one vector $\mathbf{w}$ denotes the width—the area A_i remains fixed, so the height is given as $\frac{A_i}{w}$, and the last vector denotes the rotation $\boldsymbol{\omega}$. The distance matrix D with elements d_{ij} between all shapes $i, j \in R$ represents the shortest Manhattan-distance between the rectangles' edges.

The main fitness characteristic, the flow-distance-fitness Q_{flow} is defined as

$$Q_{flow} = \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} * f_{ij} \tag{2}$$

The second fitness characteristic, the relayouting costs $Q_{relayout}$, represent the costs of transforming the given layout into the optimized layout. For this purpose each shape $i \in R$ can be attributed with a movement cost mm_i that depends on the distance that the shape is moved as well as a fixed cost ms_i, e.g., for packing and unpacking or calibration. For this purpose a vector of transition distances t_i is calculated that contains the Manhattan-distance between initial and optimized location.

$$Q_{relayout} = \sum_{i=1}^{N} x_i * (ms_i + mm_i * t_i) \tag{3}$$

where x_i is a decision variable that is 1 if $t_i > 0$ and 0 otherwise.

Constraints can be modeled as a penalty that is added to each evaluated layout. The first penalty $C_{overlap}$ deals with the constraint of nonoverlapping elements; this can occur frequently as an element's location is modified in a manner that does not consider feasibility. The second penalty $C_{boundary}$ puts a penalty on shapes that lie at least partly outside P by summing the area that falls beyond the boundary weighted with the distance to the boundary. The third constraint $C_{distance}$ is violated when the distance of two shapes is smaller or larger than the bounds specified on their mutual distance. The last constraint C_{aspect} adds a penalty to solutions in which the aspect ratio of the shape is outside the given bound. The fitness value is then computed as weighted sum of the qualities and constraints as formulated by

$$\begin{aligned} fitness = {} & \alpha_1 * Q_{flow} + \alpha_2 * Q_{relayout} \\ & + \alpha_3 * C_{overlap} + \alpha_4 * C_{boundary} + \alpha_5 * C_{distance} + \alpha_6 * C_{aspect} \end{aligned} \tag{4}$$

Conclusions

Optimizing such a layout presents many alternatives into possible rearrangements with a strong influence of the connectedness between work centers. The layouts solved by the current model do not lead to immediate practical layouts, as many real-world issues, such as infrastructure connections, security, social, and legal

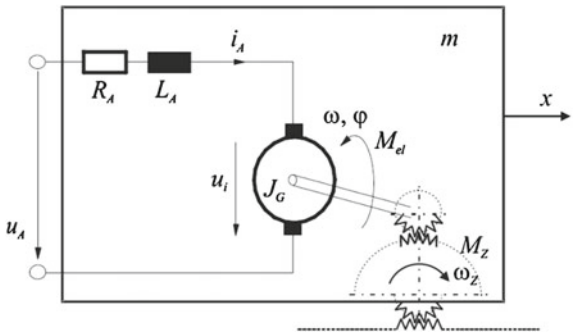

Fig. 15 Schematic picture of the electric cart system

requirements are not taken into account. Still, in a planning process together with a human planner starting points can be identified and good arrangements may be found that have not been thought of before.

3.6 Parameter Optimization of Continuous Simulation Models

Problem Description

Parameter identification in this context refers to the identification of the best possible parameter values of a simulation model: A simulation model has to be adapted to the concrete circumstances of the modeled system in order to match the real-world as exactly as possible. If this adaptations were not performed, the whole effort of creating the simulation model would be pointless because its predictions would be inaccurate. For example, the friction coefficients of a cart vary depending on the surface it is moved on and have to be adapted in the according simulation model. The only prerequisite of this parameter optimization approach is that the structure of the simulation model has to match the system which is modeled since otherwise, regardless of the effort used for parameter optimization, an adaptation to the real system would certainly fail.

Simulation Model

As a reference application we here consider the well-known cart system with an electric motor, where the vehicle mass m_1, the linear friction coefficient d_1, and the static friction coefficient F_C are unknown but constant. The simulation model is implemented in Scilab/XCos with the three free parameters m_1, d_1 and F_C, which have to be identified based on a known current U_A and measurements of the position x of the cart. Figure 15 schematically shows the electric motor and the corresponding differential equations of the continuous simulation model are displayed below.

$$\dot{x} = x$$
$$\dot{v} = -\frac{d_1}{\tilde{m}} \cdot v - \frac{1}{\tilde{m}} \cdot F_C \cdot sign(v) + \frac{k_m \cdot n}{r \cdot \tilde{m}} \cdot i_A$$
$$\dot{i_A} = -\frac{k_m \cdot n}{L_A \cdot r} \cdot v - \frac{R_A}{L_A} \cdot i_A + \frac{u_A}{L_A}$$
$$\tilde{m} = m_1 + J_A \cdot \left(\frac{n}{r}\right)^2$$

Furthermore, a simulation model of a simplified cart system and of a cart with a pendulum attached were used to test the suitability of the approach.

Optimization

In the context of parameter optimization, HeuristicLab is used to identify to parameter values of continuous simulation models which are implemented in Scilab/Xcos. Therefore, a generic coupling between HeuristicLab and Scilab has been implemented (Sect. 2.3.1) that allows the execution of arbitrary Scilab scripts. When a parameter optimization problem for simulation models should be solved, the script is responsible for executing the simulation model with suggested parameter values and calculating a quality value. The quality value expresses the accordance between the results of the simulation model with the currently used parameter values and the observed measurements in the real-world. Most of the times the sum of the squared errors at predefined time steps is calculated and used as quality value.

Every algorithm which is able to handle real-vector encoded problems could be used to solve this parameter optimization problem for continuous simulation models. HeuristicLab provides several algorithms which are suitable for this task: Genetic algorithms, evolution strategies, simulated annealing, etc. However, it was observed that the best results regarding solution quality, convergence speed, and robustness were obtained using the covariance matrix adaptation evolution strategy (CMA-ES) [21].

Conclusion

The presented approach for parameter identification has the great advantage that no information about the simulation model is needed, as the only information exchanged is a parameter vector generated by the optimization algorithm and its according quality value calculated by the simulation model. Therefore, a whole new range of optimization algorithms become applicable and one can refrain from implementing parameter optimization algorithms anew.

3.7 Electric Power System Optimization with Policy Functions

Problem Description

The electric power systems research society early identified the necessity of optimization both for planning and operation tasks, formulations such as the optimal power flow (OPF) problem shape this research domain ever since [33]. At the same

time, technological changes to electric power grids challenge new methods, requiring optimization in both dynamic as well as uncertain systems. In this context, heuristic optimization methods have evolved which are capable of managing many of those upcoming needs. Simulation optimization with metaheuristics provides a promising ground for handling uncertain dynamic problems, which makes it attractive to dynamic stochastic optimal power flow (DSOPF) issues [22, 23].

Such multi-period considerations (i.e., dynamic problem formulations) are often necessary for control issues [42, 46, 47], where optimal decisions have to be made over time while satisfying constraints robustly under stochastic conditions. Especially in future smart electric grids, among others the control of huge amounts of distributed devices (e.g., for the sake of load control) is a crucial challenge. Here, the optimal control of electric vehicles' charging processes has been identified as one of the hot spots in actual smart grids research. It can be formulated as a DSOPF problem and shall here be considered as an exemplary case.

The main idea of controlled charging is that some kind of central or decentral control influences the individual charging behavior of each single electric vehicle (EV) within a given fleet. Common objectives are system-wide peak-load avoidance, correlated charging with renewable supply, and in general the protection of existing distribution grid equipment. Such charging control decisions would need to be derived online (e.g., when an EV reaches a charging infrastructure) through consideration of the system's actual state (e.g. the EV's actual battery state-of-charge, the power grid condition, or the current supply from solar/wind power plants). Hence, similar to the approach of priority-rule optimization, a general function is needed that provides (near-) optimal charging decisions at runtime.

This is the aim of so-called policy function approximation, where an analytic function shall be identified that returns a control decision given a state without the need for embedded optimization. As demonstrated in detail in [23], such a policy function can be approximated using simulation-based metaheuristic optimization.

Simulation Model

Charging control decisions need to consider the power grid's point of view on the one hand (e.g., for avoidance of peak load values, satisfaction of secure power grid operation), but additionally have to satisfy the end-users' needs (recharge the needed energy for the next tour). While especially in modern considerations the uncertain supply provided by wind and solar power plants has to be included into load-control formulations, the resulting simulation model needs to contain three parts: the load flow simulation for deriving the grid's physical state, the traffic simulation that mimics the users' EV usage, and finally the renewable supply simulation that probabilistically describes the uncertain power injection from solar or wind power plants.

In order to derive a valid charging control decision from a given state, a policy function has to consider information from all parts and finally derive the resulting real-valued charging power for the respective EV. This principle is shown in detail in Fig. 16.

While EV-specific parameters concern the EV's driving behavior and charging demand, including its residence time at the actual charging station or its likelihood

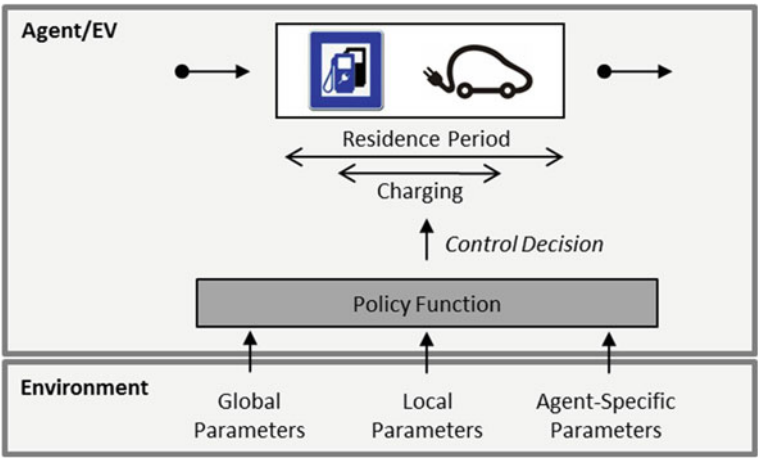

Fig. 16 Principle of policy function based control and simulation

of getting parked at another charging spot later on, local parameters also consider other EVs immediately affecting the local situation in the power grid. For example, if the power grid is stressed locally because of a high amount of EVs charging at the same grid node, their charging power may has to be reduced in the next time step in order to avoid critical power flow conditions. Finally, global parameters consider information describing the entire system's state, such as the total load to the distribution grid, total expected supply provided by renewable sources, and financial aspects considering costs of electrical power supply. Of these information entities, the policy function finally derives the approximate optimal charging decision for a given EV at a defined time step. While the mentioned parameters deliver specific information for each EV, the same policy function can be applied for all EVs in a fleet and still lead to individual decisions.

Optimization

In order to find such policy functions, genetic programming (GP) provides a fruitful method for function approximation that does not need a-priori knowledge on the aspired mathematical function, but only has to know the input variables (parameters as given above) as well as a specific grammar for combining them. Applying a metaheuristic search process (based on a genetic algorithm), GP searches for high-performance policies within a space of analytic functions. Similar to the application of priority rule optimization in the job dispatching example described in the previous section, formula trees are evolved by GP where leafs represent input variables describing the system's state (parameters as given in Fig. 16) that are combined by given mathematical operators incorporated by inner nodes. This kind of solution representation allows the evolution of arbitrary analytic functions without knowing their structure beforehand, which overcomes a severe restriction of existing works on policy function approximation in the literature.

Conclusion

Dynamic optimization with policy functions has the great advantage that it avoids the necessity for computing a specific solution to each state the dynamic system exhibits over time. Instead, an approximate optimal function is optimized offline that takes a system's state and returns control actions online. Unifying this approach with simulation optimization, the treatment of both dynamic and uncertain problems is enabled. Furthermore, the usage of GP for function approximation avoids the need for defining a function's structure beforehand, and thus overcomes a major shortcoming of policy function approximation described in the literature. While we consider here the application of EV charging control as in [23], the same methodology has been applied successfully to applications such as generation unit scheduling in power grids [24].

4 Conclusion

The fruitful combination of simulation and optimization provides mutual benefits for each field. On the one hand simulation engineers are able to improve their models using optimized parameters. On the other hand, optimization experts are able to model systems that are much closer to the real-world. Yet, often the initiating part in simulation-based optimization is the solver with the simulator being merely an evaluation function. However, a growing number of cases emerge where simulation will be used to describe optimization in dynamic environments. These cases are highly interesting from the point of view of optimization as it creates a setup that is much closer to real-world applicability. As in the simulated environment, optimization algorithms in live systems have to deal with changing conditions, uncertainty about the future, and have to make one decision at a time. In the future it will become more and more interesting to study and improve these algorithms using simulated environments.

Interfacing between simulation and optimization has also been a topic that was much discussed. Interprocess communication and different programming languages provide technical difficulties. We have described specific as well as generic interfaces that can be used to overcome these difficulties and allow exchanging candidate solution data as well as a quality feedback. We have motivated how the HeuristicLab architecture is highly suited for these tasks and given more insight into the implementation of these interfaces. The topic of integrated simulation and optimization has also been discussed and is highly relevant for future activities. In several real-world examples we have aimed to describe successful applications that may be interesting and motivating to do further research. These examples can also be seen as guidelines for a generic approach in simulation-based optimization.

Due to the steady increase of available parallel computing resources, the authors are convinced that the simulation-based optimization approach has high potential to model interrelated decision situations, leading again to a more holistic view of production and logistics optimization. The emerging fields *internet of things* and *cyber-physical systems*, which are a matter of recent research in production and

logistics optimization, are expected to benefit from the availability of such enhanced simulation optimization approaches in the future.

Acknowledgments Part of the work described in this chapter were sponsored by the *European Regional Development Fund* and by *Upper Austrian public funds* (within the the Regio 13 program - project 4EMobility), by the *Austrian Research Promotion Agency (FFG)* (within the the Josef Ressel Centre for Heuristic Optimization, the project "NPR" #829679, and the K-project "HOPL" #843532), by the *University of Applied Sciences Upper Austria* (within the basic research program), and by the *seventh framework programme* (within the project BioBoost). HeuristicLab is developed by the Heuristic and Evolutionary Algorithm Laboratory and can be downloaded from the official HeuristicLab homepage http://dev.heuristiclab.com.

References

1. M. Affenzeller and S. Wagner. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, editors, *Adaptive and Natural Computing Algorithms*, Springer Computer Series, pages 218–221. Springer, 2005.
2. M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*. Numerical Insights. CRC Press, 2009.
3. S. Albers. Better bounds for scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.
4. W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1998.
5. A. Beham, M. Kofler, S. Wagner, M. Affenzeller, H. Heiss, and M. Vorderwinkler. Enhanced priority rule synthesis with waiting conditions. In *22nd European Modeling and Simulation Symposium EMSS 2010*, 2010.
6. A. Beham, M. Kofler, S. Wagner, M. Affenzeller, and W. Puchner. Using erp-driven flow analysis to optimize a constrained facility layout problem. In *22nd European Modeling and Simulation Symposium EMSS 2010*, pages 71–76, 2010.
7. A. Beham, G. K. Kronberger, J. Karder, M. Kommenda, A. Scheibenpflug, S. Wagner, and M. Affenzeller. Integrated simulation and optimization in heuristiclab. In *Proceedings of the 26th European Modeling and Simulation Symposium EMSS 2014*, Bordeaux, France, September 2014.
8. 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.
9. W. Bell, L. Dalberto, M. Fisher, A. Greenfield, R. Jaikumar, P. Kedia, R. Mack, and P. Prutzman. Improving the distribution of industrial gases with an online computerized routing and scheduling optimizer. *Interfaces*, 13:4–23, 1983.
10. H.-G. Beyer and H.-P. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002.
11. Y. Carson and A. Maria. Simulation optimization: methods and applications. In *Proceedings of the 29th conference on Winter simulation*, pages 118–126. IEEE Computer Society, 1997.
12. J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.
13. A. Drira, H. Pierreval, and S. Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007.
14. Eurostat, European Union. Nomenclature of territorial units for statistics.

15. G. Evans. International biofuels strategy project. liquid transport biofuels - technology status report, nnfcc 08-017. Technical report, National Non-Food Crops Centre, 2008.
16. M. Fu, F. Glover, and J. April. Simulation optimization: A review, new developments, and applications. In *Proceedings of the 2005 Winter Simulation Conference*, pages 83–95, 2005.
17. M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS J. on Computing*, 14(3):192–215, Summer 2002.
18. M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, May 1976.
19. F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
20. A. Gosavi. *Simulation-based optimization: parametric optimization techniques and reinforcement learning*, volume 25. Springer, 2003.
21. N. Hansen. The CMA evolution strategy: a comparing review. In J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.
22. S. Hutterer and M. Affenzeller. Probabilistic electric vehicle charging optimized with genetic algorithms and a two-stage sampling scheme. *International Journal of Energy Optimization and Engineering*, 2:1–15, 2013.
23. S. Hutterer, M. Affenzeller, and F. Auinger. Evolutionary computation enabled controlled charging for e-mobility aggregators. In *Proceedings of the IEEE Symposium Series on Computational Intelligence, Workshop on Computational Intelligence Applications in Smart Grid (IEEE CIASG 2013*, pages 115–121, 2013.
24. S. Hutterer, S. Vonolfen, and M. Affenzeller. Genetic programming enabled evolution of control policies for dynamic stochastic optimal power flow. In *Companion Publication of the 2013 Genetic and Evolutionary Computation Conference*, pages 1529–1536, 2013.
25. O. R. Inderwildi and D. A. King. Quo vadis biofuels? *Energy Environ. Sci.*, 2:343–346, 2009.
26. J. Kennedy and R. C. Eberhardt. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press, 1995.
27. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
28. M. Kofler, A. Beham, S. Vonolfen, S. Wagner, and M. Affenzeller. Modelling and optimizing storage assignment in a steel slab yard. In *Proceedings of the 4th IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2013)*, pages 101–106, Smolenice, Slovakia, September 2012.
29. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
30. A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 2007.
31. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1999.
32. N. Moin and S. Salhi. Inventory routing problems: a logistical overview. *Journal of the Operational Research Society*, 58:1185–1194, 2007.
33. J. Momoh. *Electric Power System Applications of Optimization*. CRC / Taylor & Francis, 2009.
34. S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, Jan-Feb 1977.
35. J. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561, 2012.
36. V. Pillac, C. Guéret, and A. L. Medaglia. An event-driven optimization framework for dynamic vehicle routing. *Decision Support Systems*, 2012.
37. M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice-Hall, 1995.
38. E. Pitzer, A. Beham, M. Affenzeller, H. Heiss, and M. Vorderwinkler. Production fine planning using a solution archive of priority rules. In *Proceedings of the IEEE 3rd International Symposium on Logistics and Industrial Informatics (Lindi 2011)*, pages 111–116, Budapest, Hungary, August 2011.

39. i. Rawles. The WITNESS toolbox - A tutorial. In D. Medeiros, E. Watson, J. Carson, and M. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 223–226, 1998.
40. I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.
41. D. Sadowski and V. Bapat. The Arena product family: Enterprise modeling solutions. In P. Farrington, H. Nembhard, D. Sturrock, and G. Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 159–166, 1999.
42. E. Sortomme, M. M. Hindi, S. D. J. McPherson, and M. Venkata. Coordinated charging of plug-in hybrid electric vehicles to minimize distribution system losses. *IEEE Transactions on Smart Grid*, 2:198–205, 2011.
43. L. Tang, J. Liu, A. Rong, and Z. Yang. A review of planning and scheduling systems and methods for integrated steel production. *European Journal of Operational Research*, 133(1):1–20, 2001.
44. L. Tang, J. Liu, A. Rong, and Z. Yang. Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules. *International Journal of Production Research*, 40(7):1583–1595, 2002.
45. E. Tekin and I. Sabuncuoglu. Simulation optimization: A comprehensive review on theory and applications. *IIE Transactions*, 36(11):1067–1081, 2004.
46. G. K. Venayagamoorthy. Dynamic, stochastic, computational, and scalable technologies for smart grids. *IEEE Computational Intelligence Magazine*, 6:22–35, 2011.
47. J. G. Vlachogiannis. Probabilistic constrained load flow considering integration of wind power generation and electric vehicles. *IEEE Transactions on Power Systems*, 24:1808–1817, 2009.
48. S. Vonolfen, M. Affenzeller, A. Beham, E. Lengauer, and S. Wagner. Simulation-based evolution of resupply and routing policies in rich vendor-managed inventory scenarios. *Central European Journal of Operations Research*, 21(2):379–400, March 2013.
49. S. Vonolfen, M. Affenzeller, A. Beham, S. Wagner, and E. Lengauer. Simulation-based evolution of municipal glass-waste collection strategies utilizing electric trucks. In *Proceedings of the IEEE 3rd International Symposium on Logistics and Industrial Informatics (Lindi 2011)*, pages 177–182, August 2011.
50. S. Vonolfen, A. Beham, M. Kofler, M. Affenzeller, and K. Dörner. Simulation-based optimization of transport activities within cold charge steel production. In *Proceedings of the 5th IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2013)*, pages 67–73, Wildau, Germany, September 2013.
51. S. Vonolfen, M. Kofler, A. Beham, M. Affenzeller, and W. Achleitner. Optimizing assembly line supply by integrating warehouse picking and forklift routing using simulation. In *Proceedings of the Winter Simulation Conference*, page 339. Winter Simulation Conference, 2012.
52. S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Johannes Kepler University, Linz, Austria, 2009.
53. 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.
54. S. Wagner, G. Kronberger, A. Beham, S. Winkler, and M. Affenzeller. Modeling of heuristic optimization algorithms. In *Proceedings of the 20th European Modeling and Simulation Symposium*, pages 106–111. DIPTEM University of Genova, 2008.
55. S. Wagner, S. Winkler, R. Braune, G. Kronberger, A. Beham, and M. Affenzeller. Benefits of plugin-based heuristic optimization software systems. In R. Moreno-Diaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2007*, volume 4739 of *Lecture Notes in Computer Science*, pages 747–754. Springer, 2007.
56. M. Waller, M. Johnson, and T. Davis. Vendor-management inventory in the retail supply chain. *Journal of Business Logistics*, 20:181–203, 1999.
57. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.