Integration of Flexible Interfaces in Optimization Software Frameworks for Simulation-Based Optimization

Andreas Beham, Erik Pitzer,
Stefan Wagner, Michael Affenzeller
University of Applied Sciences Upper Austria
School of Informatics, Communication and
Media, Softwarepark 11, 4232 Hagenberg,
AUSTRIA
{andreas.beham, erik.pitzer,
stefan.wagner, michael.affenzeller}
@fh-hagenberg.at

Klaus Altendorfer,
Thomas Felberbauer, Martin Bäck
University of Applied Sciences Upper Austria
School of Management, Wehrgrabengasse 1-3,
4400 Steyr, AUSTRIA
{klaus.altendorfer,
thomas.felberbauer}@fh-steyr.at
martin.baeck@gmail.com

ABSTRACT

Optimization of simulation parameters is an important task in many different sciences where simulation is used to model and analyze complex processes and behaviors. In this work it is shown how users, such as researchers, students, and practitioners can benefit from the integration of data-exchange-interfaces in optimization software system. The development of such an interface enables users to couple their own systems and use preimplemented algorithms for their application. The interface description is based on a unified protocol buffer approach which can be ported to further frameworks and optimization software systems. The benefits of a modular architecture, such as in the HeuristicLab optimization environment, will be examined under the light of a successful integration. HeuristicLab is available on the web under the GPL license, its application to the optimization of planning and control systems in manufacturing environments will be shown as a case study in this work. The concrete subject of the case study is a production scenario where different control strategies are used to plan different products. The question is whether machines should be dedicated to a certain control strategy or whether the machines should be shared. The quality is measured by the achieved service level and amounted inventory costs.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Engineering—*Software Architectures*; D.2.12 [**Software**]: Software Engineering—*Interoperability*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; I.6.3 [**Computing Methodologies**]: Simulation and Modeling—*Applications*

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.

General Terms

Design, Experimentation

Keywords

HeuristicLab, metaheuristics, protocol buffers, simulation-based optimization

1. INTRODUCTION

The optimization of simulation parameters is not a new topic and has been discussed before [7, 6], however still there is a lack of widely available ready-to-use optimization systems that can be employed to perform the optimization task. Some simulation frameworks come with optimizers, such as OptQuest [9], included or provide a separate license to use them, but even in this case the included optimizer might not be well suited to solve the given problem. The cause for this can be found in the *no free lunch theorem* [22] which states that no algorithm can outperform any other algorithm on all problems. Due to the high heterogeneity of the actual simulation models and the meaning of their parameters, the problems can be of many different natures. In one case there might be only a single optimal solution that can be obtained through a basic local search algorithm, but in other cases the model response might be more complex and different strategies are required with different levels of diversification and intensification. In any case, the more solutions that need to be evaluated, the longer the optimization process will last. The selection of a suited optimization method and suited algorithm parameters is therefore crucial to find good parameters for the given simulation model.

1.1 Simulation Environments

A number of simulation environments and frameworks have been created and improved over time that provide the basis for developing a wide range of different simulation models. Among these are AnyLogic, Arena, or MatLab. Domain experts can make use of these frameworks that often provide a graphical development environment which abstracts the underlying programming language. This facilitates their use for people with no or only limited programming skills. These frameworks further provide many abstractions for database and file access, visualization, and include tools for user interface development and statisti-

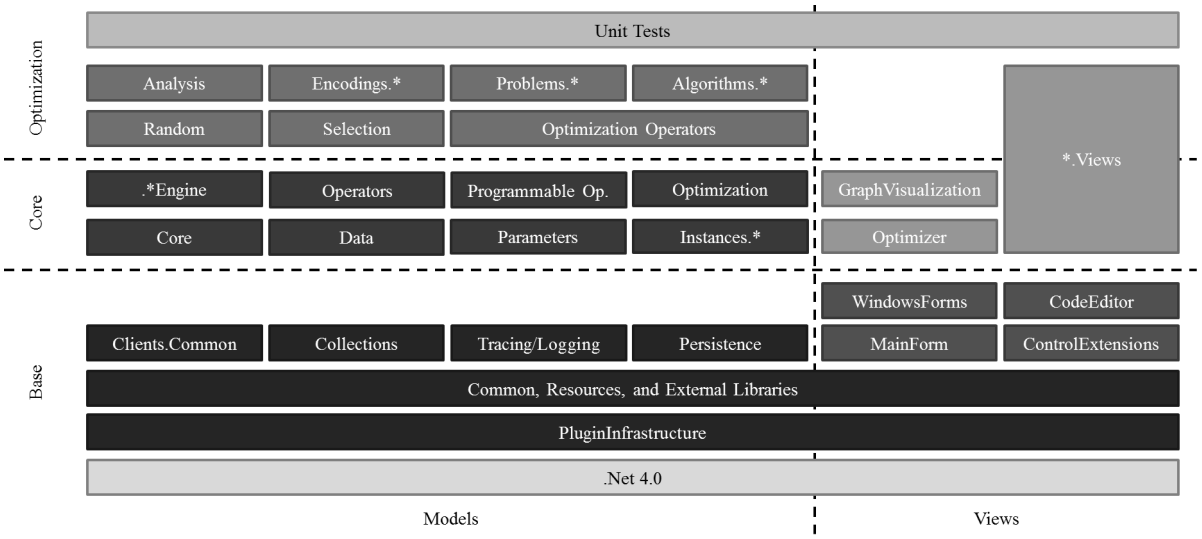


Figure 1: Block diagram of the architecture in HeuristicLab 3.3 with a separation into the different logical layers.

cal analysis. A number of different modeling methodologies may also be supported such as the development of discrete-event-based or agent-based simulation models, and/or models based on system dynamics. Naturally, models can also be written from scratch using a general purpose programming language, but the aforementioned environments offer a lot of functionality for students, researchers, and practitioners and are therefore popular starting points.

1.2 Optimization Environments

Optimization methods, especially metaheuristics, are often developed by researchers that are not experts in the application domain of a simulation model. Implementations of these algorithms may be available in a certain general purpose programming language, but often tailored to a certain problem. In the emergence and further development of optimization frameworks these methods have been generalized and abstracted to work with a number of different problems. These frameworks bundle several methods, provide a rich functionality to build upon and a sound base that is created to software engineering standards, see [16] for a comparison. HeuristicLab [19, 20] is one such environment that provides a diverse set of provided algorithms and a graphical user interface (GUI). The graphical interface allows for a more natural interaction with the software and lowers the barrier for users to apply and design optimization methods, much like AnyLogic, Arena, MatLab, etc. provide a graphical environment to create and run simulation models. In HeuristicLab, algorithms can be modeled visually through the use of a modeling language and their performance can be studied in extensive experiments. Problems can be described in the graphical environment by combining existing representations with a user-defined evaluation function. A compiler and an integrated development environment (IDE) is only necessary should new features be added in new plugins. HeuristicLab 3.3 is written in C# and was released to the public in 2010 under the GNU General Public License (GPL) v3.0.

2. HEURISTICLAB ARCHITECTURE

The architecture of HeuristicLab is composed of a multitude of different plugins. Generally, each of the boxes in Figure 1 represent an assembly, or multiple assemblies if a "*" is added to the name. The architecture consists of three horizontal layers and two vertical layers. The horizontal layers are *base*, *core*, and *optimization*. The vertical layers split the horizontal layers into *models* on the one hand and *views* on the other hand. The dependencies in this block diagram are modeled only roughly, but the general idea is that assemblies depend on those below and those to the left. The architecture is generally split sharply into assemblies that contain user interface (UI) elements and assemblies that contain model elements. Therefore HeuristicLab can also be used in environments that do not support or need a user interface.

Base Layer

The base layer contains plugins which provide essential functionality. Every plugin in HeuristicLab is based on the `PluginInfrastructure` which provides services such as loading assemblies and checking the plugin dependencies. An important class in this assembly is the `ApplicationManager` singleton which can lookup types, instantiate them, and thus provides the base for a loosely coupled architecture. The base layer also houses the `Persistence` which allows to store and restore object graphs in a compressed XML file.

Core Layer

On top of the base layer lies the core layer that includes the algorithm modeling language. It is described by core interfaces, data objects, parameters, operators, and engines. The interplay of these elements is shown in Figure 2. The important concept is the separation of individual steps in the form of small operators [19]. These are small enough to represent an essential step in an algorithm and usually large enough to contain more than just a single statement. Operators can be linked to form an operator graph which in turn describes an algorithm. An *engine* is then used to execute that graph in that it applies one operator after another

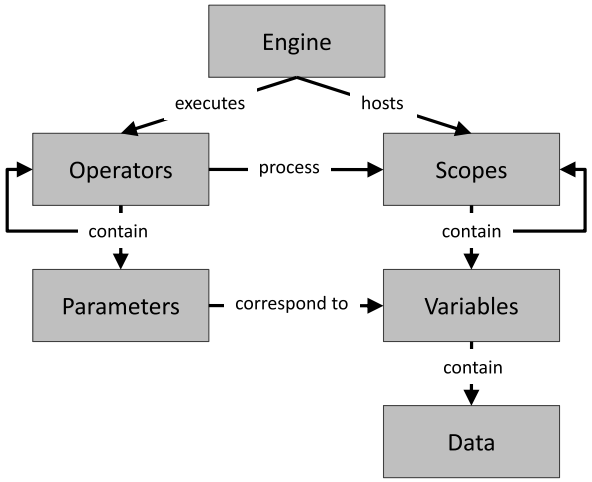


Figure 2: Interaction in the core layer in HeuristicLab 3.3.

sequentially by starting from a predefined initial operator. In general, the operators process data that is stored in the scope tree. Each scope can hold several variables and may be viewed as representing a solution. If it contains sub-scopes, it may also be seen as a population. An operator can be applied on any level in the scope tree and may read and write variables or modify the tree structure. Some operators may declare that each of the subsequent operators are applied to a number of sub-scopes in parallel. The `ParallelEngine` can execute those operators in different threads and further specialized engines allow to make use of distributed computing resources, such as HeuristicLab Hive. To obtain variables operators contain *parameters* which can contain a value themselves, or provide only the name of a variable that is then looked up in the algorithm's parameters, the problem's parameters, and the scope tree. One such operator typically is an evaluation operator that is applied on a solution scope. It reads the variables that constitute the problem representation, the problem's parameters and adds a quality variable to that scope with the fitness score. As any other operator it could also contain additional parameters that would e.g. read the algorithm's state such as the actual iteration or a collection which acts as a memory.

Optimization Layer

Finally, on top of the core layer lie the algorithm and problem models, the encodings, and various other plugins for algorithm analysis and random distributions. Several algorithms are already implemented such as genetic algorithm [15], evolution strategy [4], offspring selection genetic algorithm [1], local search, simulated annealing [14], tabu search [8], particle swarm optimization [13], and many more. The problems include real-valued test functions, and 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 a set of *analyzers* which allow to gain insights into the performance of an algorithm. Basic analyzers provide a quality progress, but more advanced analyzers enable to inspect the algorithms' behavior.

2.1 Integration of Simulation-Based Optimization

Generally, two different cases of simulation-based optimization can be identified. In the first case the simulation model acts as a fitness function, it will take a number of parameters and calculate the resulting fitness value. In the second case, the optimization problem occurs within the simulation model itself. For example, the simulation of a production facility might require to solve a scheduling problem to determine the best order of job executions which in turn requires the integration of an optimization approach. HeuristicLab has been used in both cases [17, 18] successfully, but while the first case has been generalized and abstracted as shown in this work, the second case still requires a tighter coupling with the simulation model. The generalization and abstraction of the second case could be a topic for future work.

External Evaluation Problem

Simulation-based optimization in HeuristicLab has been integrated in the form of the `ExternalEvaluationProblem`. As the name implies it assumes the evaluation is taking place in another application. This problem has no predefined representation or operators, instead the user can customize the problem according to her needs. If the actual simulation-based optimization tasks can be represented by a set of real-valued parameters, the `RealVectorEncoding` plugin and its operators can be added to the problem. If instead the parameters are integer values, the `IntegerVectorEncoding` plugin can be used to create, and modify the solutions. Both encodings can also be combined if the problem parameters are mixed. A screenshot of the problem configuration view is given in Figure 4. In the following the parameters of this problem are explained.

`BestKnownQuality & BestKnownSolution:` These parameters are used and updated by certain analyzers and remember the best quality that has been found so far as well as the corresponding best solution.

`Cache:` Together with the appropriate evaluation operator this parameter can be used to add an evaluation cache. The cache can be used to store already seen configurations and their corresponding quality so that these need not be simulated again.

`Clients:` Contains a list of clients in the form of communication channels. At least one must be specified, but the list can contain multiple channels if the simulation model is run on multiple machines.

`Evaluator:` This operator is used to collect the required variables, packs them into a message, and transmits them to one of the clients. If the cached evaluator is used the cache will be filled and the quality of previously seen configurations will be taken directly from the cache.

`Maximization:` This parameter determines if the received quality values should be maximized or minimized.

`Operators:` This list holds all operators that can modify and process solutions such as for example crossover and mutation operators. Any operator added to the list can be used in an algorithm and certain algorithms might require certain operators to work.

`SolutionCreator:` This operator is used to create the initial solution, typically it randomly initializes a vector of a certain length and within certain bounds.

Interoperability

The representation of solutions as scope objects which contain an arbitrary number of variables and the organization of scopes in a tree provides an opportunity for integrating a generic data exchange mechanism. Typically, an evaluator is applied on the solution scope and calculates the quality based on some variables that it would expect therein. The evaluator in the ExternalEvaluationProblem will however collect a user specified set of variables in the solution scope, and adds them to the `SolutionMessage`. This message is then transmitted to the external application for evaluation. The evaluation operator then waits for a reply in form of the `QualityMessage` which contains the quality value and which can be inserted into the solution scope again. This allows to use any algorithm that can optimize single-objective problems in general to optimize the ExternalEvaluationProblem. The messages in this case are protocol buffers[1] which are defined in a .proto file. The structure of these messages is shown in Figure 3.

The protocol buffer specification in form of the message definitions is used by a specific implementation to generate a tailored serializer and deserializer class for each message. The format is designed for very compact serialized files that do not impose a large communication overhead and the serialization process is quick due to the efficiency of the specific serialization classes. Implementations of protocol buffers are provided by Google for Java, C++, and Python, but many developers have provided open source ports for other languages such as C#, Clojure, Objective C, R, and many others[2]. The solution message buffer is a so called "union type", that means it provides fields for many different data types, but not all of them need to be used. In particular there are fields for storing Boolean, integers, doubles, and strings, as well as arrays of these types, and there is also a field for storing bytes. Which data type is stored in which field is again customizable. HeuristicLab uses a `SolutionMessageBuilder` class to convert the variables in the scope to variables in the solution message. This message builder is flexible and can be extended to use custom converters, so if the user adds a special representation to HeuristicLab a converter can be provided to store that representation in a message. By default, if the designer of an ExternalEvaluationProblem would use an integer vector, it would be stored in an integer array variable in the solution message. The simulation model can then extract the variable and use it to set its parameters. The protocol buffer is also extensible in that new optional fields may be added at a later date. Finally, transmission to the client is also abstracted in the form of *channels*. The default channel is based on the transmission control protocol (TCP) which will start a connection to a network socket that is opened by the simulation software. The messages are then exchanged over this channel.

Parallelization and Caching

If the required time to execute a simulation model becomes very long, users might want to parallelize the simulation by running the model on multiple computers. In HeuristicLab this is easily possible through the use of the *parallel engine*. The parallel engine allows multiple evaluation operators to be executed concurrently which in turn can make use of multiple channels defined in the `Clients` parameter. It makes use of the `ThreadPool` in .NET which manages the available threads for efficient operations. To further speed up the optimization the user can add aforementioned `EvaluationCache` and the respective evaluator. The cache can be persisted to a file or exported as a comma-separated-values (CSV) file for later analysis [17].

The application of these features in HeuristicLab on a simulation-based optimization task in a production planning and control scenario will be discussed in the next section.

[1] `http://code.google.com/p/protobuf`

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

```
message SolutionMessage {
  message IntegerVariable {
    required string name = 1;
    optional int32 data = 2;
  }
  message IntegerArrayVariable {
    required string name = 1;
    repeated int32 data = 2;
    optional int32 length = 3;
  }
  //... further sub-messages omitted for brevity ...
  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;
}
```

Figure 3: Definition of the generic interface messages.

3. SIMULATION OF MANUFACTURING CONTROL STRATEGIES

Production planning and control has a major impact on the performance of manufacturing systems. Mostly key performance indicators like inventory, tardiness and service level are measured to identify the logistical performance of such systems. One of the research questions treated concerning production planning and control is the parameterization of these methods. Looking at the available literature shows that research currently focuses on analytical models [2, 3]. Such models have the advantage that optimal solutions can be identified. However, their disadvantage is the simple structure of production systems these models have to assume. To identify good parameters for production planning and control, simulation studies can be applied which are able to model more complex and therefore more practically relevant production systems. For dispatching rule development this method has already broadly been applied [21, 5]. Recently simulation is also applied to identify good production planning and control parameters like lotsize and production lead time [12] whereby in [11] a simulation framework to support this approach is provided. Motivated by a practical case, in this paper a production system is studied with some materials being produced to stock and others being produced to order. In detail the practically relevant prob-

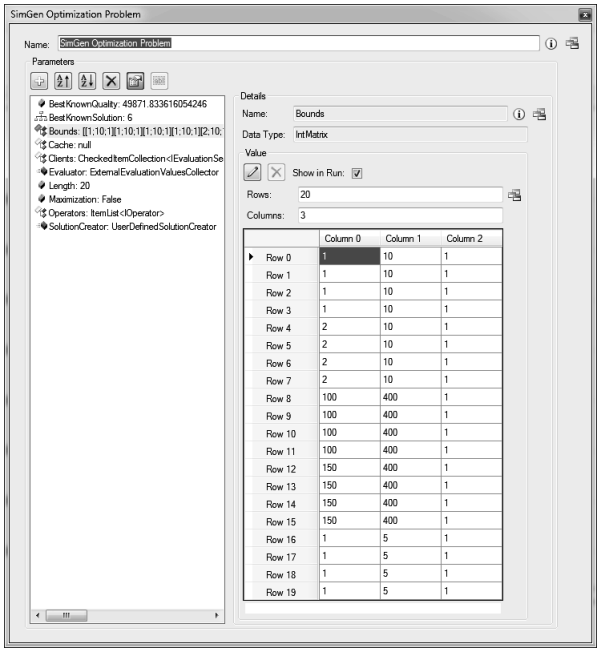


Figure 4: Example of a simulation-based optimization problem configuration in HeuristicLab 3.3 with multiple bounds shown for each dimension of an integer parameter vector. The first column denotes the minimum value, the second column the maximum value, and the step size can be given in the third column.

lem is how to combine these materials in one production system. The complexity of the problem makes it analytically intractable and therefore the simulation-based optimization approach becomes a suitable option. For the simulation model generation, the simulation framework developed in [11] is applied in this paper.

3.1 Model Description

The modeled production system follows a flow shop structure inspired by a production company operating in the automotive sector. The flow shop consists of six machines (M1-M6), which are arranged in three machine groups which can be seen as production, assembling and packaging. In each group there are two individual machines which are identical, so it is possible to produce all products on both machines within the machine groups.

The production planning and control methods for the materials are material requirements planning (MRP) as well as kanban. Half of the finished goods (items) are MRP planned and have a demand with a variation coefficient of 0.5. The second half of the items are kanban controlled and have a variation coefficient of 0.25. The question to answer is how should the planned lead times, safety stocks, and fixed order periods (FOP) for MRP and the number and size of kanban containers be set to minimize inventory and tardiness costs with respect to a predefined production system capacity. The orders were chosen such that the system utilization is approximately 85%.

The basic function of MRP is to plan material requirements. The target of MRP is to schedule jobs and purchase orders to satisfy material requirements generated by external demand. Kanban on the other hand is a pull system and triggered by demand. If a kanban is empty the upstream workstation gets a signal to reproduce the used material in the amount of the kanban size. For a more detailed description of the methods refer to [10].

Market Scenarios

Two different scenarios are considered in this work which shall be optimized and compared. The scenarios differ in the machine allocation policy of the production system which is illustrated in Figure 6a and 6b. While in the first scenario each machine group dedicates a machine to either MRP or kanban controlled materials, the second scenario shares the machines between these strategies. In both scenarios there are four kinds of items, four materials and two raw materials. There are customer orders for a total of 3000 parts per month of products 10 and 12, and 4500 parts per month for products 11 and 13. The actual amount per order is lognormal distributed with the values given in Table 1. The customer required lead time is also lognormal distributed, but with a mean value of 10 days and a standard deviation of 1.4 days for all finished products.

Table 1: Market for the small size scenario

Item		10	11	12	13
Orders / month	[pcs]	3000	4500	3000	4500
Order amount μ	[pcs]	100	150	100	150
Order amount σ	[pcs]	50	75	25	37.5

Bill of Material

Figure 5 shows the bill of material (BOM) for the scenarios and the LLC (low level code). LLC 0 combines all finished goods which are arranged into two product groups called product group PG1 and PG2. The arrows indicate which item is required to produce which other item and the cardinality in the arrows states the required number of items. The materials in LLC 3 are purchased materials in this case that are always available and need not be taken into considerations for the planning. The items in PG1 as well as item 20 and 30 are controlled by MRP, whereas the items in PG2, as well as item 21 and 31 are kanban controlled.

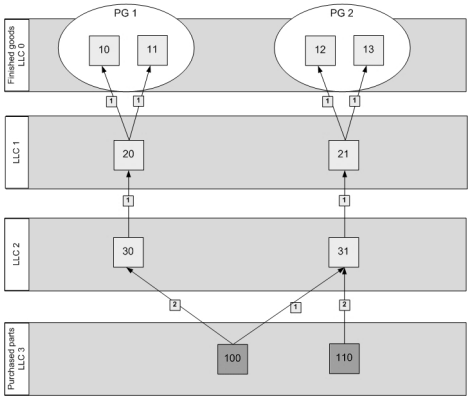
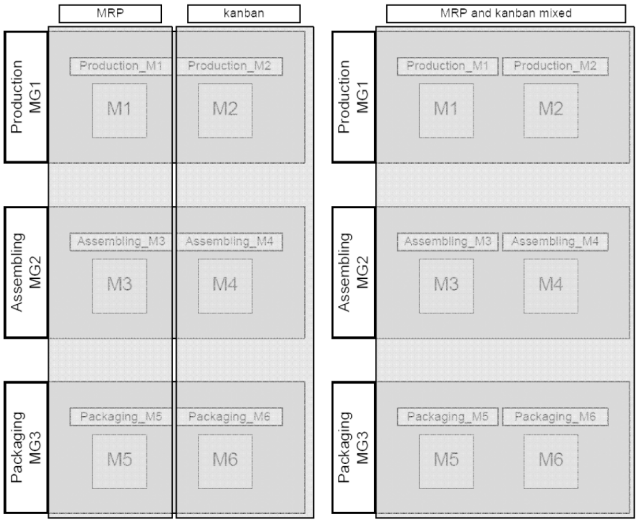


Figure 5: Bill of material

Machine Allocation Policies and Routing

Naturally, the different allocation scenarios also affect the routing. In the case where the production is mixed items 1X are produced on machine group MG3, items 2X are produced on machine group MG2, and items 3X are produced on machine group MG1. In the segmented case the machine groups are split according to the control strategy. Items 10 and 11 are produced only on M5, item 20 is produced on M3, and item 30 is produced on M1. While the kanban controlled items 12, 13, 21, and 31 are produced on the respective other machine in the machine groups. The processing and set up times are equal for all item and machine combinations and are given in Table 2. Generally, the orders are sorted by an earliest due date (EDD) dispatching rule.



(a) segmented (b) mixed

Figure 6: Different machine allocation policies.

Table 2: Processing and set up times for all items on all workstations. Set up times occur if a production order is processed on a machine that differs from the previously processed order. Otherwise no set up time is required.

		μ	σ
Processing	[min]	4.86	2.43
Set up	[min]	6	0.15

4. COUPLING WITH HEURISTICLAB

The simulation model described in the previous section is implemented in AnyLogic[3] 6. AnyLogic is written in Java and allows to add Java code in various parts of the modelling process. Generally one creates a model in form of an `ActiveObject` which might contain many other ActiveObjects. The model can then be 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

```
ExperimentInitialization();
for (e = 0 to Experiments) do
  BeforeExperiment();
  for (i = 0 to Iterations) do
    for (r = 0 to Replications) do
      BeforeSimulationRun();
      // run the model
      AfterSimulationRun();
    end for
    AfterIteration();
  end for
end for
```

Figure 7: Pseudo code of the ParametersVariation experiment in AnyLogic 6

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, but can also be varied freely such as by HeuristicLab. For this purpose a small Java library was added to the model, which is also available on the HeuristicLab website[4]. The Java library abstracts the data-exchange part and allows to set the simulation model up as either a push or a poll service for HeuristicLab. In the push service the model needs to implement an interface which is passed to the library, in the poll service the library can be polled for incoming solution messages and a quality can be returned. The ParametersVariation experiment in AnyLogic has a few steps which perfectly allow to integrate the communication requirements as can be seen in Figure 7. In the `ExperimentInitialization` section the library is used to create a new poll service on a given TCP port. In `BeforeExperiment` the quality is initialized, the solution message is polled and the process blocks until a message is received. In `BeforeSimulationRun` the parameters are assigned from the received values, and in `AfterSimulationRun` the performance indicators are calculated and added to the quality. Finally in `AfterIteration` the average quality is calculated, the library is called to transmit the value back to HeuristicLab, the quality is reset, and a new message is polled.

4.1 Experiment Design

The system utilization for all experiments is about 85%. The performance measures are the overall costs separated in inventory and tardiness costs. The fitness value after each iteration is calculated as the mean value of the overall costs measured in Euro. The tardiness costs are corrected by the end of the simulation run. All customer orders that cannot be satisfied when the simulation model ends are penalized as being tardy to avoid situations with seemingly low costs, but only due to excessively delaying customer orders. The inventory cost rate to the tardiness costs rate is chosen to be 1:20 so that tardiness costs are penalized stronger. In the experiments one year with 360 days is analyzed whereby each run is replicated 5 times to account for stochastic variance.

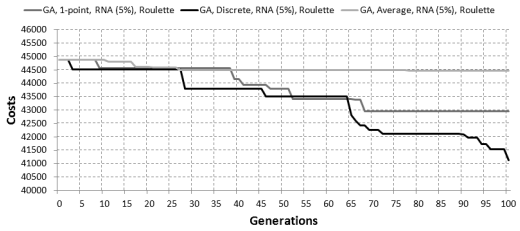
[3]http://www.xjtek.com

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

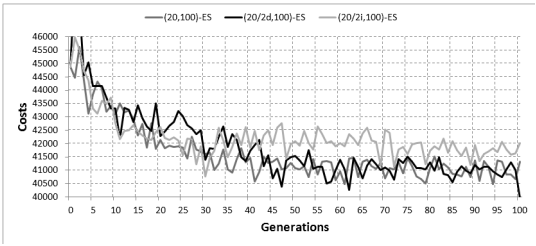
5. RESULTS

The results have been computed by applying different algorithms on the simulation model in the scenario where the production is segmented as well as in the mixed case. Figure 8 shows a comparison of the quality performance in the segmented case with various configurations of genetic algorithms (GA) and evolution strategies (ES). For the GA, "1-point" denotes a single point crossover, "Discrete" is a discrete crossover choosing each gene randomly from one of the parents, and "Average" denotes an averaging of the genes. The mutation operator, termed "RNA", was chosen to slightly perturb a solution by adding a normal distributed value to each gene and rounding the result. For the ES there were runs attempted with comma selection without recombination as well as with average (intermediate) and discrete (dominant) recombination. Every algorithm configuration was run with the same seed so that the same initial population was used. The algorithms were set to evaluate 10,000 solutions and run times for a complete run varied from 16 hours on a modern quad-core with 3.2Ghz to 24 hours on an older laptop with 1.87Ghz. The cache was active in all runs, but emptied before each new run. Some runs of the genetic algorithm benefited from the cache in that about 1-10% of the configurations were revisited and needed not to be reevaluated. This reduced the runtime to about 2 hours on average. The cache for the evolution strategy typically showed no or an insignificant number of hits meaning also that it found a larger number of unique solutions. In general in this scenario, the evolution strategy was able to find better solutions than the genetic algorithm. The quality progress of the ES showed the typical exponential shape in contrast to the progress of the GA which suggests that the discovery of better solutions is a more random event. Further considerations of the quality progress shows that variations in the mixed scenario were less common and the search could progress more smoothly, while in the segmented scenario the quality variations between the generations were higher. This indicates that the mixed scenario is better to optimize and possibly more robust to parameter variations.

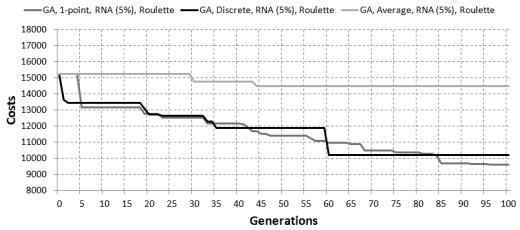
The best quality / least costs that could be achieved were about 40,000 in the segmented allocation scenario and close to 9000 in the mixed allocation scenario. This indicates that sharing of the machines might be beneficial when considering the rather small manufacturing environment under approximately 85% utilization. The generalization of these findings to real-world scenarios needs of course a much more thorough analysis under various utilization levels and also with more complex and bigger manufacturing scenarios.



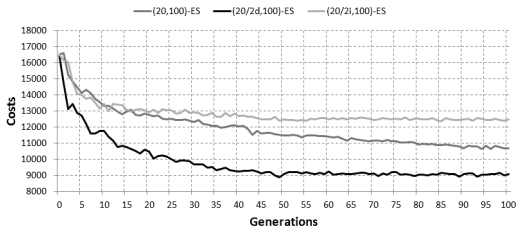
(a) genetic algorithm in the segmented scenario



(b) evolution strategy in the segmented scenario



(c) genetic algorithm in the mixed scenario



(d) evolution strategy in the mixed scenario

Figure 8: Performance comparison of algorithms with respect to the current best quality progress in different scenarios.

6. CONCLUSIONS

Metaheuristic software environments have a great benefit in that users can make use of existing algorithms and problems. Problem data can be entered into the available models and a range of different methods can be quickly employed to find the best solution. Nevertheless, the development of additional problems, possibly also making use of further problem-oriented frameworks, often requires users to deal with deeper parts both environments and may require a certain programming skill. The integration of generic data exchange methods, such as protocol buffers eases the application of optimization and allows users to provide a fitness function in a way suitable for them. For this purpose a data-exchange protocol was introduced that is based on two protocol buffer messages and it has been shown how the integration was performed within the HeuristicLab architecture as well as in the AnyLogic simulation environment. Due to the generic nature of these messages and the broad availability of protocol buffer implementations the results could also be transferred to further frameworks and environments. Finally, results were shown from a simulation-based optimization study that benefited from the described protocol.

7. ACKNOWLEDGMENTS

This work was supported by Austrian Research Promotion Agency (FFG) within the Josef Ressel Centre "Heureka!".

8. REFERENCES

[1] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*. Numerical Insights. CRC Press, 2009.

[2] K. Altendorfer. *Capacity and Inventory Planning for Make-to-Order Production Systems - The Impact of a Customer Required Lead Time Distribution*. PhD thesis, University of Vienna, 2011.

[3] K. Altendorfer and S. Minner. Simultaneous optimization of capacity and planned lead time in a two-stage production system with different customer due dates. *European Journal of Operational Research*, 213(1):134–146, 2011.

[4] H.-G. Beyer and H.-P. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002.

[5] R. Dabbas, J. Fowler, D. Rollier, and D. McCarville. Multiple response optimization using mixture-designed experiments and desirability functions in semiconductor scheduling. *International Journal of Production Research*, 41(5):939, 2003.

[6] 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.

[7] M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS J. on Computing*, 14(3):192–215, Summer 2002.

[8] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[9] F. Glover, J. P. Kelly, and M. Laguna. New advances for wedding optimization and simulation. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 255–260, 1999.

[10] W. Hopp and M. Spearman. *Factory Physics*. Mc Graw Hill / Irwin: Boston, 2008.

[11] A. Hübl, K. Altendorfer, H. Jodlbauer, M. Gansterer, and R. Hartl. Flexible model for analyzing production systems with discrete event simulation. In *Proceedings of the 2011 Winter Simulation Conference*, pages 1559–1570, Phoenix, Arizona, U.S.A, December 2011.

[12] H. Jodlbauer and A. Huber. Service-level performance of mrp, kanban, conwip and dbr due to parameter stability and environmental robustness. *International Journal of Production Research*, 46(8):2179–2195, 2008.

[13] 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.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[15] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition, 1999.

[16] J. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16(3):527–561, 2011.

[17] 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, August, 2011.

[18] 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.

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

[20] S. Wagner, A. Beham, G. K. Kronberger, M. Kommenda, E. Pitzer, M. Kofler, S. Vonolfen, S. M. Winkler, V. Dorfer, and M. Affenzeller. Heuristiclab 3.3: A unified approach to metaheuristic optimization. In *Actas del séptimo congreso español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2010)*, page 8, Valencia, Spain, September 2010.

[21] A. Waikar, B. Sarker, and A. Lal. A comparative study of some priority dispatching rules under different shop loads. *Production Planning & Control*, 6(4):301–310, 1995.

[22] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.