Data Mining for Simulation Algorithm Selection

Roland Ewald
Institute of Computer Science
Joachim Jungius Str. 10
18059 Rostock, Germany
roland.ewald@uni-rostock.de

Adelinde M. Uhrmacher
Institute of Computer Science
Joachim Jungius Str. 10
18059 Rostock, Germany
adelinde.uhrmacher@uni-rostock.de

Kaustav Saha*
Indian Institute of Technology
Kharagpur
Kharagpur - 721 302
India
kaustav.edu@gmail.com

ABSTRACT

While simulationists devise ever more efficient simulation algorithms for specific applications and infrastructures, the problem of automatically selecting the most appropriate one for a given problem has received little attention so far. One reason for this is the overwhelming amount of performance data that has to be analyzed for deriving suitable selection mechanisms. We address this problem with a framework for data mining on simulation performance data, which enables the evaluation of various data mining methods in this context. Such an evaluation is essential, as there is no best data mining algorithm for all kinds of simulation performance data. Once an effective data mining approach has been identified for a specific class of problems, its results can be used to select efficient algorithms for future simulation problems. This paper covers the components of the framework, the integration of external tools, and the re-formulation of the algorithm selection problem from a data mining perspective. Basic data mining strategies for algorithm selection are outlined, and a sample algorithm selection problem from Computational Biology is presented.

Categories and Subject Descriptors

I.6.7 [**Simulation Support Systems**]: Environments; I.2.6 [**Learning**]: Knowledge Acquisition

Keywords

Algorithm Selection, Data Mining, Simulation Performance Analysis

1. INTRODUCTION

In simulation, we often try to optimize our algorithms towards problem-dependent performance requirements. These are usually related to the specific objectives of the simulation study, the available hardware, or the modeling formalism that is used. While this is necessary to cope with increasingly complex simulation problems, such optimizations also lead to a strong dependency between the runtime behavior of an algorithm and the context in which it is used. Consequently, performance characteristics of previously developed algorithms can be hardly translated to other problem domains. A re-evaluation – and often, a re-optimization – of the algorithm in its new context becomes necessary.

An alternative to perpetually adjusting an implementation to the current requirements is the use of extensible simulation frameworks, such as JAMES II [19]. They allow to develop portfolios of algorithms that implement the same interface, which ensures their re-usability and exchangeability. For each new simulation problem, a user can now configure the system to use the algorithms that are deemed to be most efficient. This, however, leads to a new problem: the larger the set of eligible simulation algorithm configurations, the harder is the manual selection of optimal, or even 'reasonably good', configurations. This problem occurs whenever a simulation system is sufficiently flexible, even if the targeted audience has a strong simulation background [7].

The underlying *Algorithm Selection Problem* (ASP), which concerns the selection of an optimal algorithm for solving a given problem, has already been formulated in the 1970s [45]. A selection function has to be identified, which maps the features of a problem to an optimal algorithm for its solution (see section 2.1). How can we find such a selection function for simulation algorithms?

Selection functions might be manually deducible for small sets of well-known algorithms, e.g., by interpretation of experimental data or theoretical considerations – but large numbers of algorithms and uncertainty regarding their potential performance render such approaches infeasible in many cases. We argue that this problem can be solved by employing methods from statistical learning and artificial intelligence on large sets of performance data; a process known as knowledge discovery in databases, or *data mining*. Data mining is a broad, interdisciplinary field with a rich body of methods. The suitability of these methods depends significantly on the data set they are applied to; some problems may be solved with a simple linear regression, while others demand for more sophisticated techniques. As it is still unclear which data mining methods perform particularly well on which kind of simulation performance data, a system for the convenient generation of selection functions will have to provide several alternatives. We developed a framework for simulation performance data mining, SPDM, to address

*Kaustav Saha had an internship at the University of Rostock when working on this topic.

SIMUTools 2009 Rome, Italy

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

these needs. Its main objective is the creation and evaluation of selection functions by various data mining methods.

This paper consists of three parts. Sections 2 and 3 discuss some relevant concepts and related work. Then, sections 4 and 5 describe the SPDM framework and the tools we integrated. Section 6 illustrates the application of SPDM by presenting a real-world use case.

2. BACKGROUND

2.1 Algorithm Selection Problem

With the definition of the Algorithm Selection Problem [45], Rice formalized some fundamental concepts that help to clarify the challenging aspects of the problem and that need to be related to our specific domain, i.e., simulation. He defined a *problem space* $\mathbb{P}$, a *feature space* $\mathbb{F}$, and an *algorithm space* $\mathbb{A}$. The feature space contains elements that describe those aspects of a problem $x \in \mathbb{P}$ that are *relevant* for the selection of an appropriate algorithm $a \in \mathbb{A}$. Consequently, the *selection function* $S : \mathbb{F} \times \mathbb{R}^n \rightarrow \mathbb{A}$ maps the relevant features, instead of the actual problems, to algorithms from the algorithm space. It also takes into account user preferences from the *criteria space*, which is denoted by $\mathbb{R}^n$. A user criterion could, e.g., be the execution time of the algorithm. Note that the arbitrary dimension n of the criteria space allows for multi-faceted preferences, e.g., to select an algorithm that spares memory *and* exhibits a short execution time for the given problem. By using real numbers, the user can weight the importance of each criterion. User criteria are not only important for algorithm selection, but also for defining how performance can be measured. The performance mapping $p : \mathbb{A} \times \mathbb{P} \rightarrow \mathbb{R}^n$ defines the performance of algorithms for problems in $\mathbb{P}$ and each of the n criteria. A norm $g : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is now defined on this *performance measure space*, so that a single performance value $||p(a, x)|| = g(p(a, x), w)$ can be calculated from the given algorithm performance and the user's criteria weights $w \in \mathbb{R}^n$. The overall structure (after Rice [45]) is shown in figure 1.

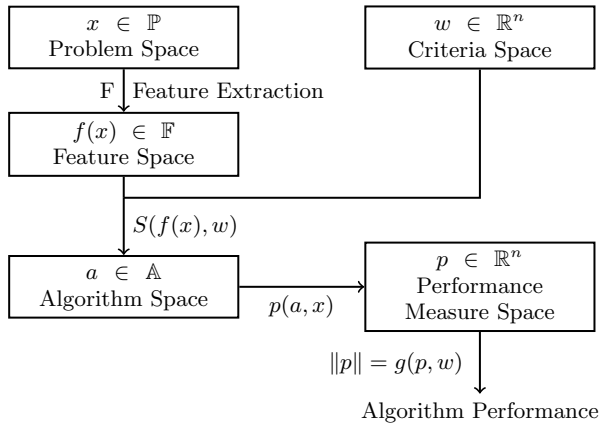

Figure 1: The algorithm selection problem. Feature extraction is expressed by a mapping $F : \mathbb{P} \rightarrow \mathbb{F}$, which extracts the features $f(x)$ from a given problem x.

In a simulation context, the problem space comprises not only the given model, but also the infrastructure to be used for its simulation. This is an important aspect, as many parallel simulation algorithms rely on specific computer or network architectures, e.g., the original GeorgiaTech TimeWarp simulator (GTW) was designed for cache-coherent shared memory multiprocessors [6]. Other algorithms may use specialized hardware accelerators, e.g., for molecular dynamics [46] or traffic simulation [42]. Extracted features of a simulation problem would therefore not only be model size, model structure, or formalism-specific model properties, but could also be – e.g., in case of a parallel and distributed simulation – the network topology, descriptions of available computing resources, and so on. Identifying the most relevant features of a problem is left to the developer of the selection mechanism and may pose a considerable challenge in itself [45].

There are several performance measures for simulation algorithms, e.g., resource consumption, execution time, and precision (in case of approximative methods). The set of algorithms can be represented by all eligible *configurations* of a simulation system for a given task (see section 2.4). The concrete definition of the ASP-related entities allows to store them in a well-structured data sink, a *performance database*, which in turn enables the application of automated data analysis.

2.2 Performance Data Storage

Performance databases are essential tools for the thorough experimental analysis of algorithms, as they support the experimenter in dealing with the typically vast amounts of performance data as easily as possible. Many performance data management systems have been developed, e.g., PerfDMF [24] and PDS [4]. PerfDMF is designed for detailed, large-scale performance analysis of complex parallel programs. PDS provides access to a central repository of benchmark results. In principle, both could be used to store simulation performance data, but they neither focus on the algorithm selection problem nor on the domain of simulation algorithms. They do not provide explicit representations of ASP entities, which are required for convenient analyses in this context. Additionally, the level of detail is usually inappropriate, e.g., PerfDMF focuses on function-wise performance measurements, i.e., profiling data, whereas only algorithms *as such* shall be investigated for the ASP. Finally, performance metrics like the accuracy of an approximative simulator are hard to integrate into such tools, as they are simply not tailored toward those demands. These reasons led to the development of a custom performance database that supports ASP-related inquiries for simulation systems. It is currently integrated with the general-purpose simulation system JAMES II (see section 2.4), but its structure is sufficiently general to support any other simulation system [7]. All necessary data for the mining tasks can be retrieved from this database, which is realized with Hibernate [1], JDBC [2], and MySQL [3].

2.3 Data Mining

The field of data mining is concerned with the extraction of knowledge from (potentially large) datasets. To do so, methods from statistical learning and its adjacent computing-focused discipline *machine learning* are applied [50, 17]. Knowledge can be gained from data in many ways; important data mining tasks are, e.g., clustering, association, feature

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

selection, classification, and regression [29]. As data mining is used here for *predicting* algorithm performance for a given simulation problem, we restrict the discussion to classification and regression. Classification means to predict the class of a so-called *problem instance*. For example, a concrete system configuration could be predicted to deliver `good-performance`, `medium-performance`, or `bad-performance` for a given a simulation problem. In this case, the problem instance would be defined by the configuration and the features of the simulation problem. Regression is a generalization of classification that supports an infinite number of (numerical) prediction classes, i.e., it can be used to predict the execution time of a configuration. For simplicity, we will refer to *both* kinds of prediction mechanisms as *predictors*.

A typical work-flow to generate a predictor involves the compilation of available data into a prescribed format, the training of the predictor, and finally its evaluation. Evaluation is usually done by estimating the error that is expected on a previously *unseen* set of data. This is problematic, as the quality of the predictor depends on the amount of available training data. Several methods can be applied to alleviate this problem, e.g., bootstrapping or cross-validation (cf. [50, p. 125 et sqq.]). Another problem closely related to the prediction error is the so-called *bias-variance trade-off* [50, 17]. The prediction error consists of three parts: irreducible error, error due to bias, and error due to variance [17, p. 197]. Let $\hat{f}(X)$ be a prediction function for data generated by the statistical model $Y = f(X) + \epsilon$ (with $E[\epsilon] = 0$ and $Var[\epsilon] = \sigma_\epsilon^2$). The expected error for an input $X = x_0$ can be decomposed as follows:

$$Err(x_0) = \sigma_\epsilon^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_o)) \qquad (1)$$

The irreducible error, σ_ϵ^2, is the prediction error even an optimal predictor, i.e., $f(X)$, would exhibit. It can be regarded as stochastic noise, e.g., the runtime of an algorithm usually varies and can therefore not be exactly predicted by any means. The squared *bias* is the error that is introduced by the *model* of the predictor, i.e., the underlying assumptions with respect to the characteristics of the prediction task. The *variance* is the error introduced by not training the predictor on all instances that are possible. The *bias-variance* trade-off means that by decreasing bias the variance is usually increased, and vice versa. Both bias and variance depend on *model complexity*: more complex models usually increase variance and reduce bias, simpler models usually increase bias and reduce variance [17, p. 37-38]. As the main objective is to reduce the *overall* prediction error, a good compromise between bias and variance has to be found, which is highly problem-specific. For this reason, *different* machine learning methods should be tested when it is unclear which method will perform best. This often holds for mining tasks on simulation performance data: it is usually unclear if and how algorithm performance can be inferred by certain model properties, or how the performances of available simulation algorithms relate to each other within the problem space. Furthermore, many machine learning schemes provide parameters to adjust them to the problem at hand, which might need to be explored as well. These circumstances motivate the development of a framework to evaluate various data mining methods on simulation performance data.

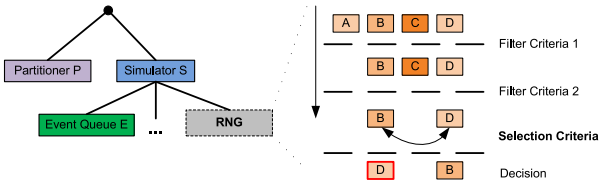

Figure 2: The factory filtering process in James II. Here, a partitioning algorithm, a simulation algorithm, and an event queue have already been selected; filtering is applied to identify a suitable random number generator (RNG). The result of all selection processes is a *selection tree* defining the configuration of the system.

2.4 James II

The general-purpose simulation system JAMES II [19] was designed to be extensible and flexible enough for the integration of new modeling formalisms, simulation algorithms, and auxiliary methods. It does so by providing a registry for managing *plug-ins*. Plug-ins are distinguished by their *plug-in type*, which allows to identify all plug-ins that can in principle be used for the same task – e.g., simulation, optimization, or random number generation. The selection of suitable algorithms is done by a filtering method that prevents, for example, that a numerical integrator is chosen to simulate a Petri Net. Filtering is done hierarchically, whenever an algorithm requests a plug-in from the registry to solve a particular sub-task. Figure 2 shows how a selection mechanism can be integrated into this filtering process without affecting the rest of the system. In the following, we will assume that a configuration can be represented as a list of name-value pairs, instead of an actual *selection tree* [7] as in figure 2. This is required to feed the data mining methods, as they usually cannot cope with hierarchical structures.

3. RELATED WORK

The need to analyze algorithm performance data by advanced techniques has often been highlighted in the field of *experimental algorithmics* [38], although this is hampered by several limiting factors. For example, it is usually impossible to translate performance findings from one hardware architecture to the other, e.g., because of differing memory hierarchies [32, 38, 39] – which makes it even more desirable to employ *automated* analysis techniques. Experimental algorithmic analysis also provides methods to obtain the necessary amount of data, e.g., by conducting *variance reduction* [37], and is concerned with experimentation methodology in general (e.g., [27]).

Systems such as PERFEXPLORER [23] or PROPHESY [47] are aimed at the analysis of performance data from large-scale parallel applications. They allow a detailed view on a given application's performance characteristics, which helps to identify bottlenecks and optimization potentials. Consequently, both systems work on much lower levels than our framework, which is focused on the *overall* performance of certain simulation system configurations in relation to each other, to select a good one automatically.

Machine learning has been used for algorithm selection before, most notably in *problem solving environments* such as PYTHIA II [21], in compiler optimization [51, 48, 49],

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

and in the field of artificial intelligence itself, where it is also known as *Meta-Learning* (see [10] for a good overview on AI-related algorithm selection). In [15], several machine learning methods are applied to sorting algorithms and an NP-hard probabilistic inference problem. For solving NP-hard problems, algorithm selection is often done with *algorithm portfolios* [14, 33, 10], an approach that is related to portfolio management from financial mathematics [35, 22]. Basically, an algorithm portfolio consists of a (usually rather small) set of available algorithms to solve a problem.

Vuduc et al. applied linear regression, a custom cost minimization scheme, and support vector machines to select from three implementations for fast matrix multiplication [48, 49]. They also distinguish between *data modeling*, i.e., predicting the runtime of each implementation, and *geometric modeling*, i.e., the identification of sub-regions in which one implementation dominates the others [49]. Both modeling approaches can be used for algorithm selection (see section 5).

Performance prediction is an ever re-occurring theme in simulation; the approaches range from analytical ones (e.g., [16, 41]), over hybrid ones (e.g., [34, 30, 8]), to purely empirical ones (e.g., [43]). Still, simulation performance predictors that employ machine learning are not so widely used yet. One exception is the work of Ferscha et al. [9], who apply a full factorial analysis to assess the sensitivity of parallel and distributed discrete-event simulations with respect to several synchronization schemes. However, machine learning has often been applied to *meta-modeling*, i.e., to train a predictor on the *outcomes* of a simulation, so that the parameter space of the model can be explored much faster than by simulating the original model. For example, neural networks can be used to efficiently predict hardware simulation results [25].

4. SPDM: SIMULATOR PERFORMANCE DATA MINING FRAMEWORK

4.1 Requirements

As motivated in section 2.3, a key requirement for a data mining framework aimed at simulation algorithm selection is to allow the comparison of *many* data mining methods. This is necessary because the optimal trade-off between bias and variance needs to be found, and the properties of simulator performance spaces are so heterogeneous (and often unknown) that it is impossible to select the one optimal learning method beforehand, for all selection tasks yet to come. Implementing and testing data mining schemes also involves a lot of effort, so another requirement of this framework is the easy integration of existing data mining toolkits.

Following our focus on algorithm selection, we will concentrate on *prediction* methods from data mining. Since prediction can be done by either classification or regression, i.e., predicting discrete or continuous classes (see section 2.3), both variants have to be supported. The data mining methods need to be applicable to performance data from various sources. They shall automatically construct selection functions for the algorithm selection problem (see section 2.1). In the following, the realization of such a selection function as a software component will also be called a *selector*. The general work-flow that the framework shall support is shown in figure 3.

The evaluation of selectors should provide both an estimation of the expected prediction error and a comparison between the selectors that have been used. The comparison of data miners needs to be *fair*, i.e., all selectors should be tested by the same methods. Such an analysis could reveal important aspects of the selection task at hand, e.g., a linear regression model that performs significantly worse than a general regression model hints at the non-linearity of the performance space under scrutiny (following the argumentation from [25]). Similarly, if all data miners fail, maybe the data set is insufficient to draw conclusions, or the selected problem features do not discriminate between relevant model classes.

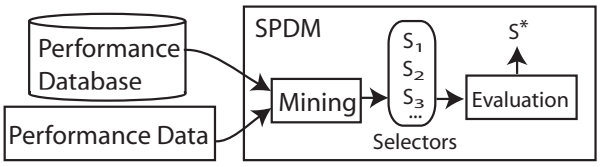

Figure 3: The SPDM work-flow: performance data is imported and various data mining schemes are applied to generate selectors. These will be evaluated, so that the best selector S^* can be identified. It can then be used for selecting algorithms in the simulation system that provided the data.

4.2 Implementation

SPDM uses the plug-in architecture of JAMES II in several ways. It basically applies the Strategy pattern [11, p. 315 et sqq.] to the work-flow blueprint outlined in section 4.1. A flexible data import layer is provided, which allows to translate available performance data into SPDM's internal representation. The representation consists of two classes: `PerformanceTuple` (see figure 4) represents a problem instance and therefore combines the `Features` of a simulation problem (e.g., certain model properties) and the `Configuration` that was used to solve the problem, i.e., the selection tree (see figure 2). The selection tree is flattened to a list of $(name, value)$ pairs. The name is taken from the plug-in type, e.g.,'`EventQueue`', and the value is the name of the chosen algorithm, e.g.,'`Heap`'. Parameters of the algorithms are handled similarly. Having equally named parameters or multiple algorithms of the same plug-in type requires an additional step for re-coding the names.

Finally, a `PerformanceTuple` includes a real-valued performance field representing $||p||$ (see section 2.1). Including p itself, i.e., a multi-dimensional performance measurement, is not suitable for many machine learning algorithms, which are tuned to predict a single outcome. Support for multi-faceted performance could be integrated by creating a selection function for each dimension and then combining the selections of all functions according to the user criteria. The second class for data representation, `PerfTupleMetaData`, provides information on the nature of the attributes defined by a set of `PerformanceTuple` instances, e.g., which values a qualitative attribute may have. Each name given in any `Features` or `Configuration` object is regarded as an attribute.

The other central entity in the framework is the `ISelector` interface (see figure 4), which represents the selectors to be generated. Its main method takes a `Features` instance

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

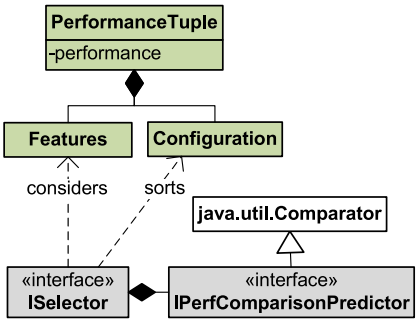

Figure 4: Structure of Performance Tuples and Selection Functions

describing the problem at hand and returns a list of `Configuration` instances, sorted by descending predicted performance. This list can then be used by a selection criteria to re-sort all available factories (see figure 2). Returning a sorted list instead of a single configuration is advantageous when it comes to the transparent integration of the selection process in the existing system – it is not known if the user installed all available plug-ins or restricts the filtering process to a certain subset of configurations. Since well-performing selection functions shall be persisted and deployed with the simulation system, `ISelector` extends `Serializable`. Sorting the available configurations in a list implies that two configurations can be compared, which allows to provide a default `ISelector` implementation and thereby reduce the implementation efforts for new selectors: merely an alternative `Comparator` implementation for configurations is required, which will then be used for sorting (cf. figure 4).

4.2.1 Data Import

Although this framework is intended to work on JAMES II performance data predominantly, importing performance data from arbitrary sources is an important feature. It allows to test and evaluate SPDM and its methods with external benchmark data, which ensures comparability with other toolkits. Therefore, a simple file format for reading performance data has been defined. As performance data from JAMES II is stored in a performance database (see section 2.2), another implementation of the corresponding `IDMDataImportManager` interface retrieves the data from the JAMES II performance database and translates it into performance tuples. Formerly imported data can be saved to an XML file and re-read with a corresponding import manager. Other data import schemes can be integrated similarly.

4.2.2 Selector Generation and Evaluation

The integration of existing toolkits is facilitated by the Adapter pattern [11, p. 139 et sqq.]. Any adapter for a data mining method has to implement the `ISelectorGenerator` interface. It defines a single function which takes the performance data for training and returns an instance of `ISelector`. As motivated in sections 2.3 and 4.1, another important task of the framework is the automated analysis of these generated selection functions by various methods. This task is divided into two sub-tasks: the *evaluation strategy* and the *performance measurement*. An evaluation strategy defines how the available data is divided into test and training data; it triggers the actual selector generation. A performance measurement, in turn, calculates the numerical error for a given selector and test data, i.e., it executes the generated `ISelector` instances. Different measurements of selector performance might be very helpful for advanced analyses: For example, simply counting the number of mis-predicted best configurations would punish 'near misses' overly hard – calculating the total extra runtime due to mis-predictions might be a much better estimator of future selector performance. For 'conservative' selectors in real-time simulation applications, which could focus on avoiding very bad configurations instead of always choosing an optimum, the maximal extra runtime imposed by a single prediction error might be even more suitable. Implementing different evaluation strategies is also desirable, as they have different strengths and weaknesses, e.g., depending on the amount of available data. Consequently, both kinds of components have been realized as plug-in types in JAMES II. So far we implemented cross-validation and bootstrapping as evaluation strategies, and several basic performance measurements. The overall structure of SPDM is summarized in figure 5.

5. METHODS FOR ALGORITHM SELECTOR GENERATION

Basically, any algorithm selection approach that follows the ideas from section 4.2 by providing a `Comparator` for `Configurations` (see figure 4) comes down to deciding which of two alternative configurations will perform better on a given problem, characterized by a set of features. The sorted list to be returned results from a succession of such decisions.

In principle, there are two ways to tackle this decision task. The *decision approach* focuses on the decision as such, i.e., it classifies tuples of the form

$$(features, conf_1, conf_2)$$

as $firstFaster$, $secondFaster$, or $equal$. Using such a representation executes the predictor only once per comparison, which saves computing time. However, this approach has a severe drawback: the number of tuples that need to be generated out of the available data grows quadratically with the number of available configurations – which could be several hundreds. In other words, this approach does not scale with the diversity of simulation system configurations.

The *prediction approach*, in contrast, aims at predicting the performance of each configuration independently. The outcomes can then be compared, which would amount to two predictions per comparison. Still, the performance of each configuration only needs to be predicted *once* for sorting the list, so that the effort can be reduced by some caching within the comparator component. This approach has many advantageous aspects: Firstly, the number of tuples to be analyzed grows only linearly with the number of available configurations. Secondly, it can be realized by both (quantitative) prediction and classification, although prediction lends itself more naturally to the problem. A classification algorithm could simply classify algorithms to perform *good*, *medium*, or *bad* under the given circumstances, but this classification might become invalid – for example, if a new algorithm is much faster than the other algorithms on certain problems, so that the performance of the others has to be changed to *medium* or *bad* for these cases.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

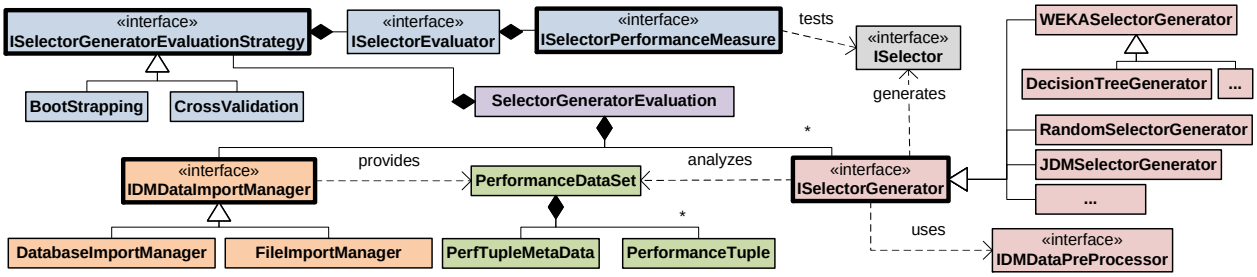

Figure 5: This UML chart depicts the central components of SPDM: data import (orange), data representation (green), selector generation (red), evaluation (blue), and the overall assessment strategy (purple). The final product to be generated are instances of the `ISelector` interface (grey). Interfaces marked with bold borders correspond to components that are exchangeable via the plug-in mechanism of James II.

Finally, quantitative prediction methods could help when searching for tipping points in the simulation space, i.e., the set of problem features for which an algorithm is likely to outperform the others – which in turn could steer experimentation to the 'most interesting' regions of the problem space. These are the main reasons why the current `ISelectorGenerator` implementations follow the prediction approach instead of the decision approach, but this could be easily implemented as well.

This distinction between the decision and the prediction approach reflects the distinction between geometric modeling and data modeling in [49], but is different in that the geometric modeling merely divides the feature space into regions dominated by a certain implementation, while the decision approach has to do this for any combination of two algorithms – it is unknown which implementations will be available to the user. This is the reason why geometric modeling works although the decision approach is troublesome.

Both ways of prediction could be augmented by general multi-model classifiers [50, p. 250 et sqq.], i.e., meta-learners that combine several predictors and learn when to use which. This is supported by SPDM, as the James II registry allows selector generators to discover other selector generators, which could then be used to create selectors that work together – the resulting selector ensemble just has to implement the `ISelector` interface as well. Realizing such kinds of selector generators is subject to future work. The next section briefly presents the data-mining tools we have integrated into SPDM so far.

5.1 Tool Integration

5.1.1 WEKA

The WEKA toolkit is a powerful and popular open-source machine learning software written in Java [50]. It contains several algorithms for all kinds of data-mining tasks; so far we have implemented generators based on WEKA's version of the *C4.5* decision tree algorithm, `J48`, as well as its implementation of Quinlan's *M5* model tree algorithm [44], `M5P`. Decision trees are only able to classify problems, but their structure is human readable and may reveal interesting patterns in the training data, e.g., the importance of different attributes. M5 model trees are related to decision trees but allow numerical prediction by a linear model. The model to be used is selected by a decision tree.

5.1.2 MLJ

MLJ is a set of *Machine Learning Tools in Java* [40]. It is a Java port of MLC++, a framework developed to ensure comparability and repeatability of machine learning results [31]. We integrated *ID3*, another decision tree algorithm and predecessor of *C4.5*, as well as a Naïve Bayes approach.

5.1.3 Joone

Joone is an engine for neural network simulation in Java [36]. Since neural networks are limited to real-valued input, our wrapper has to re-code all nominal attributes. Joone is highly flexible and allows various shapes of networks, transfer functions, and learning algorithms. This is reflected in our adapter, which is highly configurable as well. Changing the size of the model or the shape of the transfer function is important for finding the best trade-off between variance and bias, as described in section 2.3. Joone also supports multi-threading, which speeds up the costly training process.

5.1.4 JDM

We also started the integration of the official Java Data Mining interface, JDM [29], by following the guidelines from [20]. A general interface for Data Mining tools in Java has great appeal, as this could considerably reduce the efforts of integrating new tools. However, so far only a limited range of commercial toolkits support the JDM, although a new version of it is already under way [28]. We use the JDM implementation of KXEN Inc. to test our wrapper, but this is still work in progress.

6. USE CASE: STOCHASTIC SIMULATION ALGORITHMS

We now briefly describe a set of stochastic simulation algorithms that are commonly used in Computational Biology, and then illustrate what kinds of studies are possible when using the SPDM on their performance data. In [13], Gillespie proposed an alternative to simulating chemical reaction networks by ordinary differential equations (ODEs). While ODEs are suitable for simulating systems with many particles, they do not account for the stochastic effects that govern the behavior of chemical reaction networks with relatively few particles. Such networks can often be found in biology, e.g., in cell signaling pathways.

Hence, Gillespie introduced two stochastic methods, the

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

First Reaction Method and the Direct Method (DM), to simulate such networks. Both are basically simulating the same continuous-time Markov chain and hence produce equivalent results, but they differ algorithmically [13]. Later, the First Reaction Method was enhanced by Gibson and Bruck [12], who introduced their variant as the Next Reaction Method (NRM). The NRM maintains additional data structures to speed up the simulation, e.g., it relies on an efficient event queue implementation. In [5], Cao et al. apply similar enhancements to the Direct Method, which leads to the Optimized Direct Method. We refer to [26] for a detailed comparison of the actual algorithms, as their theoretical advantages and disadvantages are not relevant in this context. In fact, SPDM should allow for an automated *empirical* assessment of the implementations under scrutiny.

JAMES II provides implementations of the Direct Method and the Next Reaction Method, as well as several event queues [18]. We chose ten event queue variants and combined each with the NRM implementation. Together with the Direct Method, which does not rely on any additional data structure, we therefore have eleven configurations of the simulation system to test. These configurations represent our algorithm space $\mathbb{A}$ (cf. figure 1). The problem space $\mathbb{P}$ was defined by the *CCS* model from [26], a synthetic benchmark model that is parameterizable with respect to the number of chemical species involved, the number of reactions per species, and the number of reactants per reaction, which determines the degree of interdependence between the reactions. Hence, $\mathbb{P}$ is three-dimensional in this case: there were between one and ten species interacting by one to fifty reactions, each reaction having one to ten reactants, i.e., species participating in the reaction. The last parameter also adjusts the number of products per reaction, as all reactions in *CCS* have the same number of reactants and products.

Our feature space $\mathbb{F}$ has three dimensions as well: we selected the number of species, the number of reactions, and the average number of reactants and products per reaction as features. Features have to be easy to compute, as they need to be extracted at run-time from each simulation problem to which algorithm selection shall be applied (see figure 1). The features we selected merely require a single pass through the model and hence can be computed in $O(n)$, as illustrated by the simplified sample code in algorithm 1.

The execution time was taken as a performance measure. Our overall data set consisted of 59 problems that were selected randomly from $\mathbb{P}$, and all eleven configurations were applied to each problem, which results in 649 performance tuples. To rule out side-effects from external load or the operating system, each application of a configuration to a problem was replicated ten times, i.e., this data set represents 6490 simulation runs. The performance tuples were constructed using the average execution time. The overall work-flow followed is depicted in figure 7: while also a manual algorithm selection requires to define benchmark models, to run experiments, and to store the results (e.g., in a database), using SPDM also requires to program at least one feature extractor (cf. algorithm 1) and to configure the framework regarding input data and selector generators to be tested. As a result, SPDM returns the performance of the selectors it generated and evaluated. In contrast, a manual analysis has to deal with the raw performance data.

Algorithm 1 Feature extraction for SSA

```
public class SSAFeatExtr extends
    FeatureExtractor {

public Map extractFeature(ISimulationProblem
    prob) {
 //...(+ 5 lines model creation from problem)
 double avgDeg = 0;
 List reacts = model.getReactions();
 for (IGillespieReaction r : reacts)
   avgDeg += r.getChangeVec().length;
 avgDeg = avgDeg / reacts.size();

 Map f = new HashMap();
 f.put("#Species", model.getStateVec().length);
 f.put("#Reactions", reacts.size());
 f.put("#AvgReactProdDeg", avgDeg);
 return f;
 }
}
```

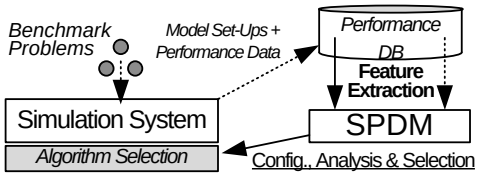

Figure 7: Overall work-flow of using SPDM.

Results.

Figure 6 shows a *truth map* of the *CCS* data, a visualization inspired by [49]: it plots the best configurations as a point cloud in the feature space. A single combination of algorithms, NRM with a simple list-based event queue implementation, dominates a large part of the feature space. The only exception is a region with few species and few reactions, as can be seen in the middle plot of figure 6.

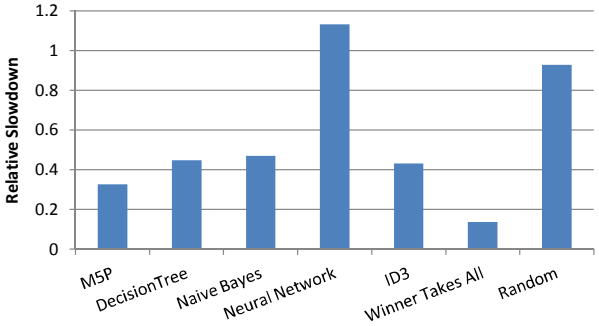

Figure 8: Performance of data mining schemes on *CCS* performance data, each averaged over 1000 iterations of 0.632 bootstrapping.

Figure 8 shows the results of some classifiers that have been applied to the data set. The performance was measured by letting each generated selector sort all configurations for a (formerly unseen) combination of features from the test set. Then, the real-world execution time of the configuration that was predicted to be the fasted, t_{pred}, is

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

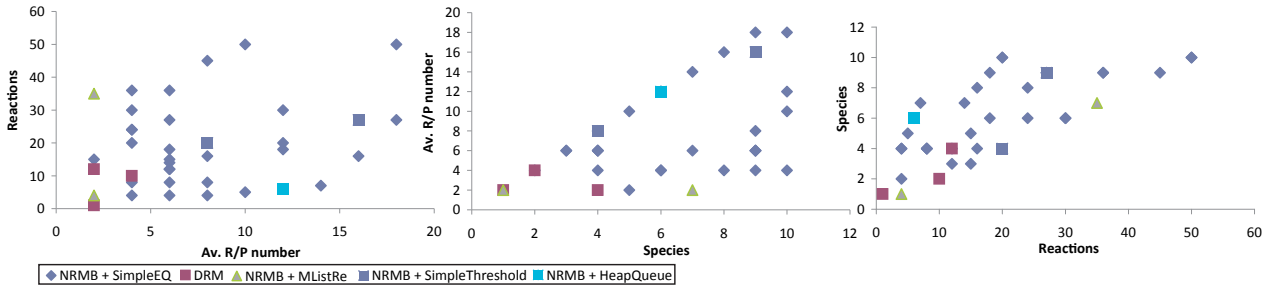

Figure 6: Truth map for the *CCS* data set. The Next Reaction Method using a simple event queue implementation (SimpleEQ) dominates a large region of the problem space. Several distinct problems have equal features, hence there are less than 59 points per plot. Only five configurations (out of eleven) could outperform all others on at least one simulation problem.

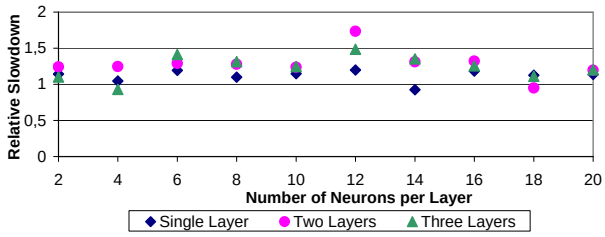

Figure 9: Effect of neural network parameters (50 bootstrapping iterations): Neither number of layers nor number of neurons have a positive effect on selector performance. This motivates the exploration of other parameters (learning algorithm, type of synapse, etc.), or changing the internal structure of the selector, e.g., by generating a single neural network per configuration (instead of one large one).

compared to the execution time of the fastest configuration for the given set of features, t_{best}. The relative slowdown can then be computed easily: $\frac{t_{pred}}{t_{best}} - 1$. Since $t_{pred} \geq t_{best}$, the slowdown will always be greater than 0, which would be the optimal outcome. All numbers have been averaged over 1000 iterations of 0.632 bootstrapping.

The performance of the tested methods differs greatly: while the neural network performs *worse* than a random algorithm selection, the decision tree methods (J48, ID3) and the Naïve Bayes classifier can perform significantly better, albeit still far away from the optimum. As another benchmark, we implemented a `WinnerTakesAll` selector that simply ranks all configurations according to their overall performance on the training set. It does not consider any problem features and selects a fixed configuration order. No data mining method can outperform it in this example, which suggests that the sample data set is too homogeneous or too small to allow effective data mining. Although this result might seem unsatisfying in that the data mining techniques applied in this specific case are not superior to a much simpler method, it nicely illustrates why a framework like SPDM can be of great use. Firstly, SPDM *shows* that a certain configuration is indeed so dominant for the given problem that none of the tested mining methods yields better results than a very simple technique. Secondly, SPDM also showed *which* of the applied mining techniques have returned the best results and could therefore outperform `WinnerTakesAll` selection when more performance data is available. Thirdly, the results also give an estimate of the overall speed-up to be expected by an intelligent selection criteria when compared to the average case – here, this would still be $\approx 25\%$ of the execution time (for `M5P`, see figure 8).

Also note that these results are preliminary – the performance measure only considers the configuration predicted to be best and none of the algorithms was particularly tuned towards the input. These results should merely illustrate how SPDM allows us to apply a *broad range* of methods in a convenient manner, as the selection of a suitable approach is non-trivial and fine-tuning requires additional efforts. In fact, SPDM can be easily configured to tune the parameters of a data mining scheme automatically. This allows, for example, to explore the various aspects of neural network parameterization, as depicted in figure 9. It is also important to notice that none of the results from above can guarantee that the generated selectors work well in real life: perhaps the selected features allow to predict the performance of the *CCS* benchmark model, but do not translate to other models (with the same features)? Other features would have to be chosen if that was the case. This aspect has still to be investigated for the stochastic simulation algorithms and the generated selectors.

7. CONCLUSIONS

In this paper, we motivated the use of data mining methods to automatically analyze simulator performance data and thereby tackle the algorithm selection problem. We introduced a general data mining framework for this task, SPDM, and described its central entities, as well as the specific requirements that led to them. SPDM is built on the simulation system JAMES II, but can process performance data from other systems as well. Finally, we outlined different data mining strategies in the context of algorithm selection and presented a typical use case for SPDM, the generation of selection functions for stochastic simulation algorithms.

As the use case in section 6 illustrates, algorithm selection may sometimes concern only a subset of the configuration space, e.g., if only one or two configurations dominate the others – but to detect such situations will be greatly facilitated by an automated performance data analysis. More-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

over, the gain of selecting the better of both dominant configurations might still be worth the effort, especially if the effort of trying out various data mining schemes is reduced to a few minutes, as intended by the SPDM framework.

While SPDM itself is up and running, further work needs to be done until a fully-automated algorithm selection for JAMES II is in place. Most importantly, the work-flow of parameterizing a set of benchmark models, exploring the simulation space, selecting some features, and *interactively* exploring the data mining results is still in its infancy. To investigate the applicability of a given data mining method to algorithm selection, much more performance data and additional sets of simulation algorithms are needed.

8. ACKNOWLEDGMENTS

This research is sponsored by the German research foundation (DFG). We thank KXEN Inc. for providing us with a temporary academic license for the KXEN Analytic Framework.

9. REFERENCES

[1] Hibernate: http://www.hibernate.org/.

[2] JDBC: http://java.sun.com/javase/technologies/database.

[3] MySQL: http://www.mysql.com.

[4] M. W. Berry, J. J. Dongarra, and B. H. Larose. The development and implementation of a performance database server. Technical report, University of Tennessee, Knoxville, TN, USA, 1993.

[5] Y. Cao, H. Li, and L. Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The Journal of Chemical Physics*, 121(9):4059–4067, 2004.

[6] S. R. Das, R. M. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. GTW: A Time Warp System for Shared Memory Multiprocessors. In *Winter Simulation Conference*, Proceedings of the 1994 Winter Simulation Conference, 1994.

[7] R. Ewald, J. Himmelspach, and A. M. Uhrmacher. An algorithm selection approach for simulation systems. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS) 2008*, volume 22, pages 91–98, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[8] R. Ewald, J. Himmelspach, A. M. Uhrmacher, D. Chen, and G. K. Theodoropoulos. A simulation approach to facilitate parallel and distributed discrete-event simulator development. In *Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2006*, pages 209–218, Washington, DC, USA, 2006. IEEE Computer Society.

[9] A. Ferscha, J. Johnson, and S. J. Turner. Distributed simulation performance data mining. *Future Generation Computer Systems*, pages 157–174, September 2001.

[10] M. Gagliolo and J. Schmidhuber. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, August 2006.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.

[12] M. A. Gibson and J. Bruck. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *J. Chem. Physics*, 104:1876–1889, 2000.

[13] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81(25), 1977.

[14] C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In D. Geiger and P. P. Shenoy, editors, *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 190–197, San Francisco, August January–March 1997. Morgan Kaufmann Publishers.

[15] H. Guo. *Algorithm Selection for Sorting and Probabilistic Inference: A Machine-Learning Approach*. PhD thesis, Kansas State University, 2003.

[16] A. Gupta, I. F. Akyldiz, and R. M. Fujimoto. Performance Analysis of Time Warp With Multiple Homogeneous Processors. *IEEE Trans. on Softw. Eng., Special Section on Parallel Systems Performance*, 17(10), October 1991.

[17] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[18] J. Himmelspach and A. M. Uhrmacher. The event queue problem and pdevs. In *Proceedings of the SpringSim '07, DEVS Integrative M&S Symposium*, pages 257–264. SCS, 2007.

[19] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Proceedings of the 40th Annual Simulation Symposium*, pages 137–143. IEEE Computer Society, 2007.

[20] M. F. Hornick, E. Marcade, and S. Venkayala. *Java Data Mining: Strategy, Standard, and Practice. A Practical Guide for Architecture, Design, and Implementation*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2007.

[21] E. N. Houstis, A. Catlin, J. Rice, V. Verykios, N. Ramakrishnan, and C. Houstis. Pythia ii: A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, 26(2):227–253, 2000.

[22] B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.

[23] K. A. Huck and A. D. Malony. PerfExplorer: A performance data mining framework for large-scale parallel computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 41, 2005.

[24] K. A. Huck, A. D. Malony, R. Bell, and A. Morris. Design and implementation of a parallel performance data management framework. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing*, pages 473–482, Washington, DC, USA, 2005. IEEE Computer Society.

[25] E. Ipek, S. A. Mckee, K. Singh, R. Caruana, B. R. de Supinski, and M. Schulz. Efficient architectural design space exploration via predictive modeling. *ACM Trans. Archit. Code Optim.*, 4(4):1–34, January 2008.

[26] M. Jeschke and R. Ewald. Large-scale design space

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659

exploration of ssa. In *Computational Methods in Systems Biology, International Conference, CMSB 2008, Rostock, Germany, October 12-15, 2008, Proceedings*, 2008.

[27] D. Johnson. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, chapter A theoretician's guide to the experimental analysis of algorithms, pages 215–250. American Mathematical Society, Oxford, United States, February 2003.

[28] JSR-247 Expert Group. Java(tm) Specification Request 247: Java(tm) Data Mining (JDM) 2.0, September 2006.

[29] JSR-73 Expert Group. Java(tm) Specification Request 73: Java(tm) data mining (JDM), July 2004.

[30] Z. Juhasz, S. Turner, K. Kuntner, and M. Gerzson. A Trace-based Performance Prediction Tool for Parallel Discrete Event Simulation. In *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2002*, 2002.

[31] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. MLC++: a machine learning library in C++. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, pages 740–743, 1994.

[32] A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 370–379, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[33] K. Leyton-Brown, E. Nudelman, G. Andrew, J. Mcfadden, and Y. Shoham. A portfolio approach to algorithm selection. In *IJCAI*, 2003.

[34] J. Liu, D. M. Nicol, B. J. Premore, and A. L. Poplawski. Performance Prediction of a Parallel Simulator. In *Workshop on Parallel and Distributed Simulation*, pages 156–164, 1999.

[35] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[36] P. Marrone. *Java Object Oriented Neural Engine: The Complete Guide*, January 2007.

[37] C. McGeoch. Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. *ACM Comput. Surv.*, 24(2):195–212, 1992.

[38] C. C. McGeoch. Experimental analysis of algorithms. *Notices of the AMS*, 48(3):304–311, March 2001.

[39] C. C. McGeoch. Experimental algorithmics. *Communications of the ACM*, 50(11):27–31, November 2007.

[40] MLJ: Machine Learning Tools in Java (MLJ). http://sourceforge.net/projects/mldev/. Accessed July 28, 2008.

[41] D. M. Nicol. Scalability, locality, partitioning and synchronization PDES. In *Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 5–11. IEEE Computer Society, 1998.

[42] K. S. Perumalla. Efficient execution on GPUs of field-based vehicular mobility models. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS) 2008*, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[43] K. S. Perumalla, R. M. Fujimoto, P. J. Thakare, S. Pande, H. Karimabadi, Y. Omelchenko, and J. Driscoll. Performance Prediction of Large-Scale Parallel Discrete Event Models of Physical Systems. In *Winter Simulation Conference 2005*, 2005.

[44] J. R. Quinlan. Learning with continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[45] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[46] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, R. C. Ho, D. J. Lerardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. Mcleavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang. Anton, a special-purpose machine for molecular dynamics simulation. *Commun. ACM*, 51(7):91–97, July 2008.

[47] V. Taylor, X. Wu, and R. Stevens. Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications. *SIGMETRICS Perform. Eval. Rev.*, 30(4):13–18, March 2003.

[48] R. Vuduc, J. Demmel, and J. Bilmes. Statistical models for automatic performance tuning. In *Computational Science – ICCS 2001*, pages 117–126. Springer Berlin / Heidelberg, 2001.

[49] R. Vuduc, J. W. Demmel, and J. A. Bilmes. Statistical models for empirical search-based performance tuning. *Int. J. High Perform. Comput. Appl.*, 18(1):65–94, February 2004.

[50] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.

[51] H. Yu, D. Zhang, and L. Rauchwerger. An adaptive algorithm selection framework. In *Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*, pages 278–289, 2004.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5659
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5659