FI MU

Faculty of Informatics
Masaryk University Brno

Estimating State Space Parameters

by

Radek Pelánek
Pavel Šimeček

FI MU Report Series **FIMU-RS-2008-01**

 January 2008

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/reports/`

Further information can be obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**

Estimating State Space Parameters

Radek Pelánek*
Faculty of Informatics
Masaryk University Brno, Czech Republic
pelanek@fi.muni.cz

Pavel Šimeček†
Faculty of Informatics
Masaryk University Brno, Czech Republic
xsimece1@fi.muni.cz

January 28, 2008

Abstract

We introduce the problem of estimation of state space parameters, argue that it is an interesting and practically relevant problem, and study several simple estimation techniques. Particularly, we focus on estimation of the number of reachable states. We study techniques based on sampling of the state space and techniques that employ data mining techniques (classification trees, neural networks) over parameters of breadth-first search. We show that even through the studied techniques are not able to produce exact estimates, it is possible to obtain useful information about a state space by sampling and to use this information to automate the verification process.

1 Introduction

Explicit model checking is a state-of-the-art technique for verification of asynchronous concurrent systems. This technique is based on construction of a reachable part of a

*Partially supported by GA ČR grant no. 201/07/P035.
†Partially supported by GA ČR grant no. 201/06/1338.

model state space. In this work we are concerned with techniques for estimation of state space parameters, particularly with estimation of the number of reachable states. Estimation of state space parameters is not a typical problem in verification. Nevertheless, we argue that it has a good motivation — there are even several independent reasons to study this problem.

Tuning of model checking algorithms. Estimation of state space parameters can be useful for tuning model checking algorithms, for example to select a suitable size of hash table or cache [10], [7] or to choose a proper I/O efficient graph exploration algorithm [11, 4].

Parameter estimations are particularly useful in the distributed environment. If we do not use enough workstations, then the computation runs out of memory. If we use too many workstations, then the performance deteriorates due to unnecessary communication overhead [14]. Estimate of the number of reachable states can help us choose the suitable number of workstations for a given model. Parameter estimations can also be used to select an appropriate reduction technique or to choose an algorithm from several competing ones [18]. Another application in a distributed environment is the selection of a suitable cycle detection algorithm. There are many distributed cycle detection algorithms which can be used for LTL verification of large models (see [2] for an overview); each of these algorithms is suitable for a certain class of graphs.

Information for users. The run-time of a model checker is often long. It would be very convenient for the user to have the estimation of the remaining time. Such an estimation is not just of practical interest, it has also psychological advantages — if users are informed about the remaining time, they are more willing to wait [13].

Selection of models for experiments. Experimentalists need a lot of models in order to convincingly show advantages of novel techniques and to properly evaluate known techniques. It is often desirable to have models with a specific state space size. Consider, for example, experiments with distributed algorithms and algorithms which are concerned with memory consumption (state space caching, external memory algorithms). For such experiments, it is needed to have models with a state space size around the limits of a single machine. It is laborious and computationally intensive to find models with specific state space size. Techniques for estimating the reachable state space size can significantly speed up this effort and thus lead to an improvement in experimental standards.

Our aims and contributions in this work are the following:

1. Study of simple techniques for estimation of state space parameters. Particularly, we study techniques based on sampling of the state space and techniques that employ data mining techniques (classification trees, neural networks) over parameters of breadth-first search.

2. Evaluation and comparison of techniques. We evaluate these techniques over a large benchmarks set and compare their performance.

3. Summary of techniques potential. We formulate what is reasonable to expect from studied techniques.

Note that most of the techniques presented in this work could be further improved by careful tuning and optimisation, but that is not the aim of the presented work.

Related Work The problem of estimating state space parameters did not receive much attention so far. Most of the relevant work deals only with techniques based on analysis of model structure and was not very successful. Watson and Desrochers [23] proposed a technique for estimation of the size of Petri nets based on a static analysis of the net. Estimation is quite accurate, but it is limited to a specific type of models. Chamillard et al. [6] proposed technique based on the linear regression measures given by static metrics of the model. However, the technique does not work very well. Peng and Makki [20] described a technique for comparison of two state space sizes; their technique is independent of the specification language. Sahoo et al. [22] use sampling of the state space to decide which BDD based reachability technique is the best for a given model.

2 Background

In this section we discuss the state space parameters that we are interested in. We also describe the experimental data and the sampling techniques that we have used for evaluation.

2.1 State Space Parameters

We view a state space as a simple directed graph $G = (V, E, v_0)$ with a set of vertices V, a set of directed edges $E \subseteq V \times V$, and a distinguished initial vertex v_0. Vertices are

states of the model, edges represent valid transitions between states. For our purposes we ignore any labeling of states or edges. We are concerned only with the reachable part of the state space.

An *average degree* of G is the ratio $|E|/|V|$. A *strongly connected component* (SCC) of G is a maximal set of states $C \subseteq V$ such that for each $u, v \in C$, the vertex v is reachable from u and vice versa. Let us consider the breadth-first search (BFS) from the initial vertex v_0. A *level* of the BFS with an index k is a set of states with distance from v_0 equal to k. The *BFS height* is the largest index of a non-empty level. An edge (u, v) is a *back level edge* if v belongs to a level with a lower or the same index as u.

2.2 Experimental Environment

Reported techniques are implemented in the DiVinE environment [3]. In order to evaluate estimation techniques, we perform experiments over the BEEM benchmark set [17]. The web portal of the benchmark set contains all the used models and it also presents state space properties of models.

For the evaluation we use 160 models from the BEEM set. The state space size of used models is between 20,000 and 20,000,000 states (note that for the evaluation we use only models for which we are able generate the full state space). Used models are divided randomly into a training set and a testing set, each of them contains 80 models. Reported estimation techniques are trained (calibrated) on the training set and their success is measured over the testing set.

2.3 Sampling Techniques

All our estimation techniques are based on the following approach: we take a sample of the state space and according to the information collected by this sample we do the estimation. We use mainly the classical search techniques to make samples:

- breadth-first search (BFS) sample: first s states of BFS,
- depth-first search (DFS) sample: first s states of DFS (the size of stack was limited to 10,000 states during the search),
- random walk (RW) sample of size s: we run random walks through the state space until we find s states.

Beside these classical techniques, we also introduce a special kind of random walk. The aim of this technique is to traverse a small, but representative portion of the state space. The technique uses a hash function to decide which states should be stored and explored, therefore we denote it as *Hash-RW*. Unlike memoryless random walk, it uses a state repository to recognize visited states. To increase the probability that the walk visits states lying on various paths in various distances from the initial state, the search traverses through up to C states in a parallel fashion, where C is a given constant number.

In essence, this random walk works in a similar way as BFS. BFS works by levels: after generation of level i, level $i+1$ is generated using the successor function applied to states in level i. Hash-RW works in the same way, but it tries to traverse only a subset of states of size C from each level. The subset of states to traverse is determined by a special decision function. The function computes a hash of a given state and if it is smaller than a certain limit, it decides to store the state to the repository and the exploration queue. The limit is updated after computation of all successors of the last level to keep number of states in the next level close to C. The aim of this decision function is to reduce the number of traversed back level edges. In some cases the basic version of Hash-RW explores large portion of a state space. Therefore, we use additional finishing conditions; these conditions are described together with a specific usage of Hash-RW in Section 4.

3 Estimation of the Number of Reachable States

In this section we discuss techniques for estimating the number of reachable states of a given model. Our preliminary experiments showed that simple techniques are not able to produce accurate absolute estimates. Therefore, we classify models in three classes and try to estimate these classes. In the following we introduce the classification, discuss two approaches for the estimation of the classification (one based on sampling and one based on BFS parameters), and then we combine techniques and compare them.

3.1 Classification

Using simple techniques, it seems impossible to estimate the number of reachable states with a high accuracy. However, for practical applications this is not necessary. It is often sufficient to have an 'order of magnitude' estimate. Therefore, we introduce three

classes, which produce the order of magnitude estimate and study techniques for estimating the classification. This approach also simplifies evaluation and comparison of estimation techniques.

The classes are not defined in absolute terms (by number of states), but rather relatively: we suppose that a sample of a state space is generated and we define the classes with respect to how many times the total number of reachable states is larger than the sample. We have two reasons to adopt this approach. Firstly, it enables us to do meaningful experiments with estimation techniques on state spaces of different sizes. Secondly, the speed of state generation significantly differs for different models [17], i.e., the relative estimate is more useful for estimating the model checker run time.

For our experiments, we have used three classes which we believe have practical substantiation. Let R be the ratio of the total number of reachable states to the number of the taken sample:

- Class 1, $1 \leq R < 4$. Models in this class can be verified. It should be sufficient to just wait or to slightly tune the model checkers parameters (e.g., use a more appropriate hash table size, turn on a reduction technique).
- Class 2, $4 \leq R < 32$. To check a model in this class, it is necessary to use an aggressive reduction technique and/or distributed computation. It should be possible to verify the model as it is, but it may be a bit complicated.
- Class 3, $32 \leq R$. Models in this class seem out of reach (if the sample is large). It is probably necessary to apply abstraction to these models.

For calibration and evaluation of class estimation techniques we use training and test data of the following form: input = model + sample size, output = class. We have used several sample sizes for each model; both training and testing set contained approximately 320 inputs. Note that we work only with models for which we know the size of the state space, i.e., we know the correct class.

For each estimation technique we report: *success rate* — the ratio of inputs correctly classified, *major mistakes* — the ratio of inputs classified totally incorrectly (i.e., class 1 classified as class 3 or vice versa).

3.2 Techniques Based on Sampling

A straightforward approach for estimating the number of reachable states is the following:

1. Take two independent random samples of size s of reachable states.
2. Let the number of states which occur in both samples be x.
3. Estimated number of reachable states is s/x (in other words, the ratio x/s is expected to be close to s/n, where n is the number of reachable states).

However, this straightforward approach cannot be realized — without actually generating all reachable states, there is no way to obtain two independent random samples of reachable states. The straightforward way to obtain a random reachable state is to use random walk from the initial state. However, the chance of picking a state by this method is far from uniform [19], i.e., this method can not be used to obtain a random sample.

Nevertheless, the outlined method can be used even if samples are not completely independent and random. We use the BFS, DFS, and RW samples (as described in Section 2). Results in Fig. 1. show the relation between the ratio x/s and the correct estimate s/n. Results are show for all three combinations of the three sampling methods, these results are obtained on the training set. Based on the data from the training set we identify decision values which are used for the classification of the testing set (more specifically, the decision values were identified automatically with the use of R software [21]). For example, for the DFS x RW sampling method we use the following values:

- If $x/s >= 0.401$ then output 'C1'.
- If $x/s < 0.401 \wedge x/s >= 0.096$ then output 'C2'.
- If $x/s < 0.096$ then output 'C3'.

Using this classification method, we classify the data in the testing set. Table 1. gives results. It shows that the best results are obtained using the intersection of DFS and RW samples — success rate is 72%.

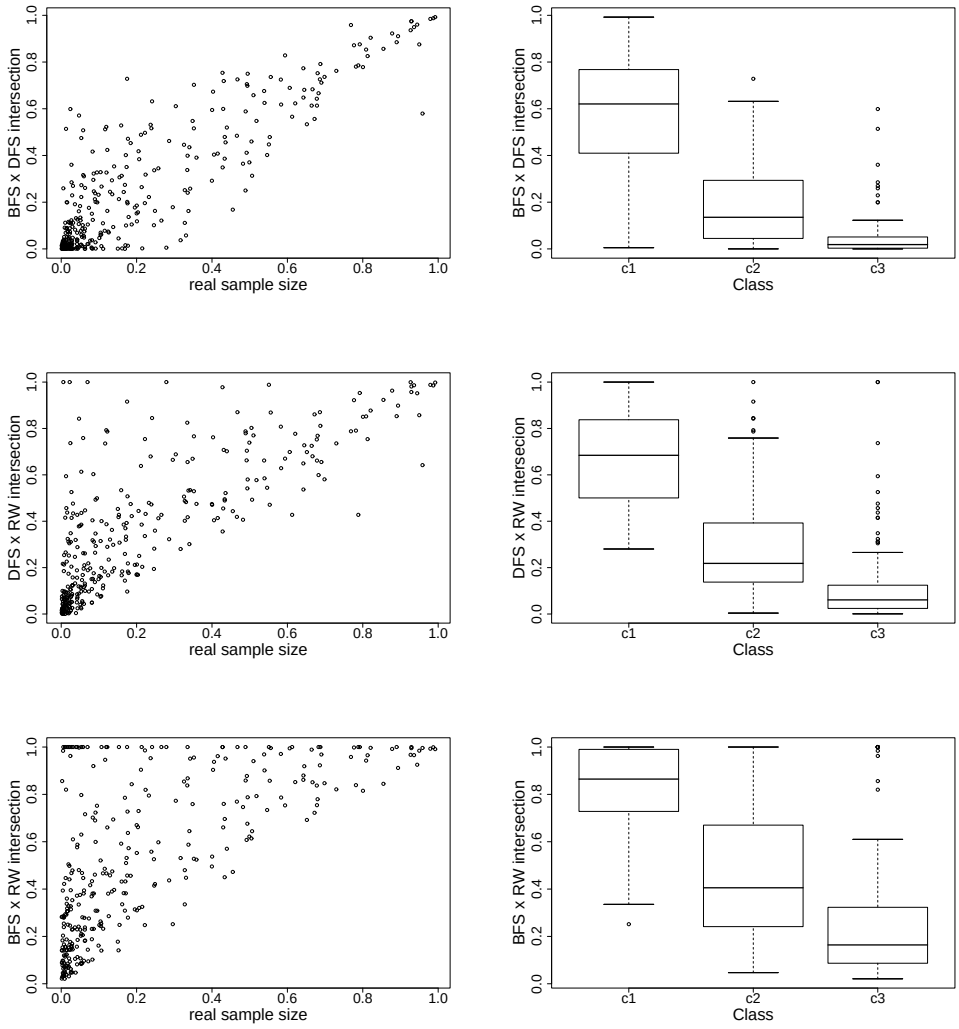

Figure 1: Figures on the left show scatter plot of the ratio of sample size to number of reachable states and the relative size of intersection of two samples. Figures on the right show relative sizes of intersections for each class using the boxplot method (the upper and lower lines are maximum and minimum values, the middle line is a median, the other two are quartiles; circles mark outliers).

BFS x DFS				DFS x RW				BFS x RW			
	E1	E2	E3		E1	E2	E3		E1	E2	E3
C1	31%	6%	0%	C1	34%	2%	0%	C1	25%	11%	0%
C2	8%	19%	12%	C2	8%	29%	3%	C2	4%	31%	4%
C3	3%	9%	12%	C3	3%	12%	9%	C3	3%	10%	11%
success rate 62%				success rate 72%				success rate 67%			
major mistakes 3%				major mistakes 3%				major mistakes 3%			

Table 1: Results of estimation techniques based on sampling. Rows (C1, C2, C3) are correct classifications, columns (E1, E2, E3) are estimated classifications. Results are given as percents, numbers are rounded.

3.3 Techniques Based on BFS Parameters

The sizes of levels follow a typical pattern. If we plot the number of states on a level against the index of a level we get a BFS level graph. Usually this graph has a 'bell-like' shape [16] (see Fig. 2. for several such graphs; more BFS level graphs are on the BEEM web page [17]). Our goal is to use the knowledge of this typical behaviour of breadth-first search and to estimate the size of the state space based on the first k BFS levels (k is determined by the size of a sample).

3.3.1 Estimation by Human

As the first step, we have performed experiments with classification by human (one of the authors). The human is shown a graph of first k levels (see Fig. 3. for examples) and based on this information estimates the classification. Results in Table 2. suggest that a human can perform classification reasonably well. This is an interesting observation — it suggests that the BFS level graph may be a useful output of a model checker during the state space generation. Note that this is a rather cheap operation both in terms of computation overhead and implementation effort.

Based on the experience with human estimation, we choose the following parameters as inputs for automatic classification methods (let k be the number of BFS levels in the sample):

- LA — ratio of size of the last explored BFS level to the average size of the first k levels;

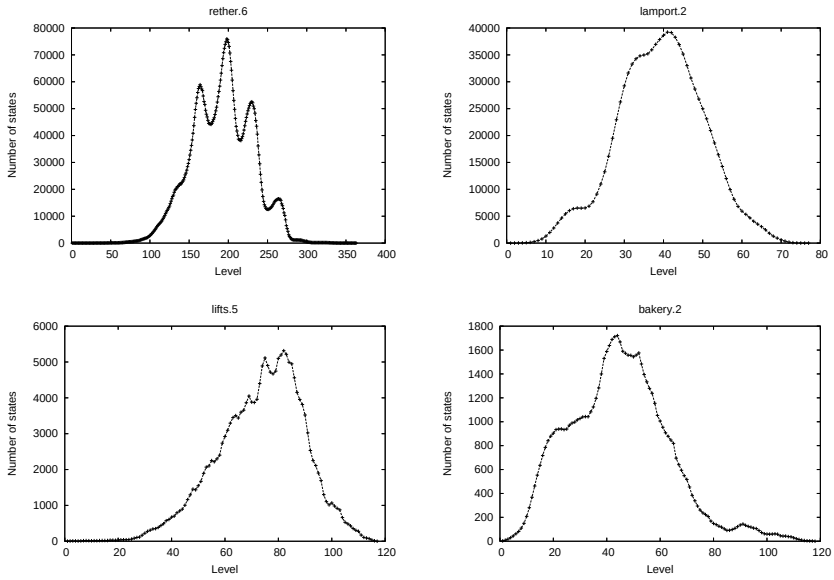

Figure 2: BFS level graphs. For simple models the curve is smooth and bell-like, for more complex models the graph is a bit ragged.

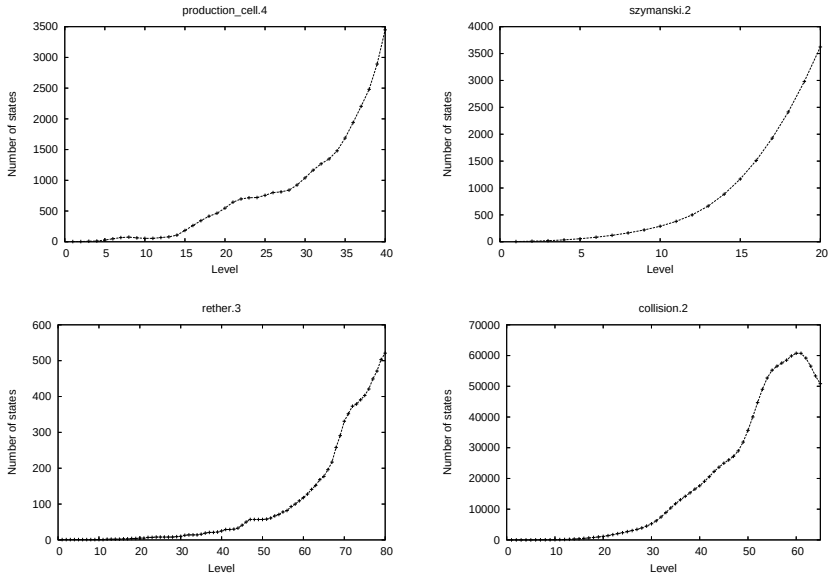

Figure 3: Examples of partial BFS level graphs as used for human classification.

- LM — ratio of size of the last level to the maximal size of a level in the first k levels;
- INF (inflexion) — boolean value, zero means that the difference of sizes of consecutive levels is increasing (up to k), i.e., there is no 'inflection point';
- HE (height estimate) — the ratio of k and an estimated BFS height; as an estimate of the BFS height we use the median height from the training set of models of the corresponding type (see Section 4.2).

3.3.2 Classification Tree

Classification tree [5] is a data mining technique used to predict membership of cases in the classes of a categorical dependent variable from their measurements on predictor variables. Classification tree is built through a binary recursive partitioning. Splitting conditions are determined automatically by an analysis of the training data.

Fig. 4. shows a classification tree constructed on our training data (the tree was constructed using R software [21]). Prediction results for the test data are given in Table 2.

3.3.3 Neural Network

Neural network is a machine-learning technique that simulates a network of communicating nerve cells. The network is a weighted graph, the learning is accomplished by modification of weights of edges. This process can be automated using a suitable learning algorithm.

We use a neural network with:

- 4 input neurons (parameters LA, LM, INF, and HE),
- 4 hidden neurons,
- 3 output neurons (one for each class).

For implementation we use FANN library [15]. The network is trained on the training data; we use the default learning algorithm of the FANN library. Results for the test data are in Table 2.

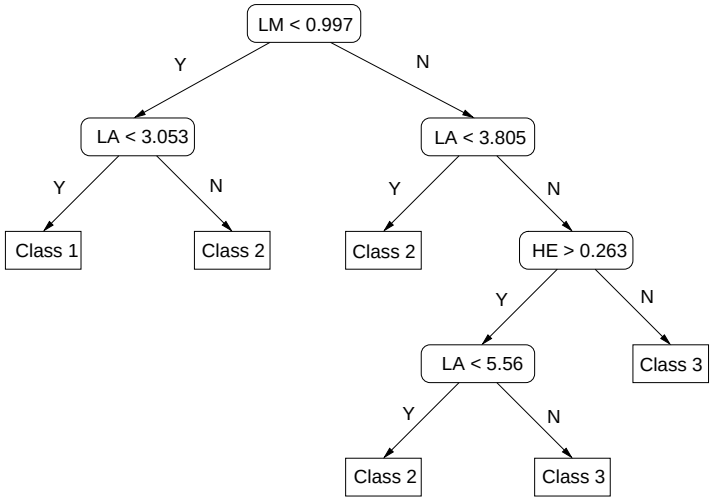

Figure 4: Classification tree based on BFS parameters.

	human				classification tree				neural net		
	E1	E2	E3		E1	E2	E3		E1	E2	E3
C1	21%	18%	2%	C1	16%	20%	1%	C1	23%	13%	1%
C2	2%	18%	13%	C2	2%	24%	13%	C2	7%	25%	7%
C3	3%	6%	17%	C3	0%	10%	14%	C3	2%	14%	7%
success rate 56%				success rate 54%				success rate 55%			
major mistakes 5%				major mistakes 1%				major mistakes 3%			

Table 2: Results of estimation techniques based on BFS parameters on on the test data. Rows (C1, C2, C3) are correct classifications, columns (E1, E2, E3) are estimated classifications. Results are given as percents, numbers are rounded.

3.4 Combinations and Comparison

Finally, we combine estimates produced by the five above presented automated techniques (three sampling techniques, classification tree, and neural network). The combined estimate is obtained in the following way:

- If four techniques agree on the estimate, we output the given class.
- If the estimates are divided between two neighbouring classes, we output 'undecided estimate' (E12, E23).
- Otherwise, we output 'don't know' (DN).

	E1	E12	E2	E23	E3	DN
C1	18%	15%	2%	0%	0%	1%
C2	1%	4%	19%	7%	0%	8%
C3	0%	1%	4%	9%	5%	5%

Table 3: Combined estimations from the five presented automated techniques. 'Exy' means that the estimate is between classes x and y, DN means don't know.

Table 3. presents results obtained in this way. We see that there are no major mistakes, the number of misses is low (7%), and at the same time the number of undecided and don't know results is reasonable.

Let us compare the studied techniques. Better results can be achieved using the sampling approach: 70% success rate compared to 55% success rate of the BFS based techniques. However, the sampling approach is less practical: it requires at least two samples of the state space and it cannot be easily used on-the-fly (during the state space generation). Techniques based on BFS parameters are less precise, but they can be used very easily during the BFS traversal of the state space — after the traversal of each BFS level we just plug the data into the prefabricated classification tree or neural network. Moreover, we suppose that it should be possible to further improve BFS based techniques by considering more parameters for decision and by incorporating domain specific information (i.e., by training techniques on similar data as they will be applied to).

4 Estimation of Other Parameters

In this section we study techniques for estimation of some other state space parameters: the average degree, the BFS height, the number of back level edges, and the size of the largest SCC.

4.1 Average Degree

The average degree usually corresponds to the amount of non-determinism in the system, which substantially influences usefulness of partial order reduction [9]. Average degree can also be understood as a quotient of the number of edges and the number of

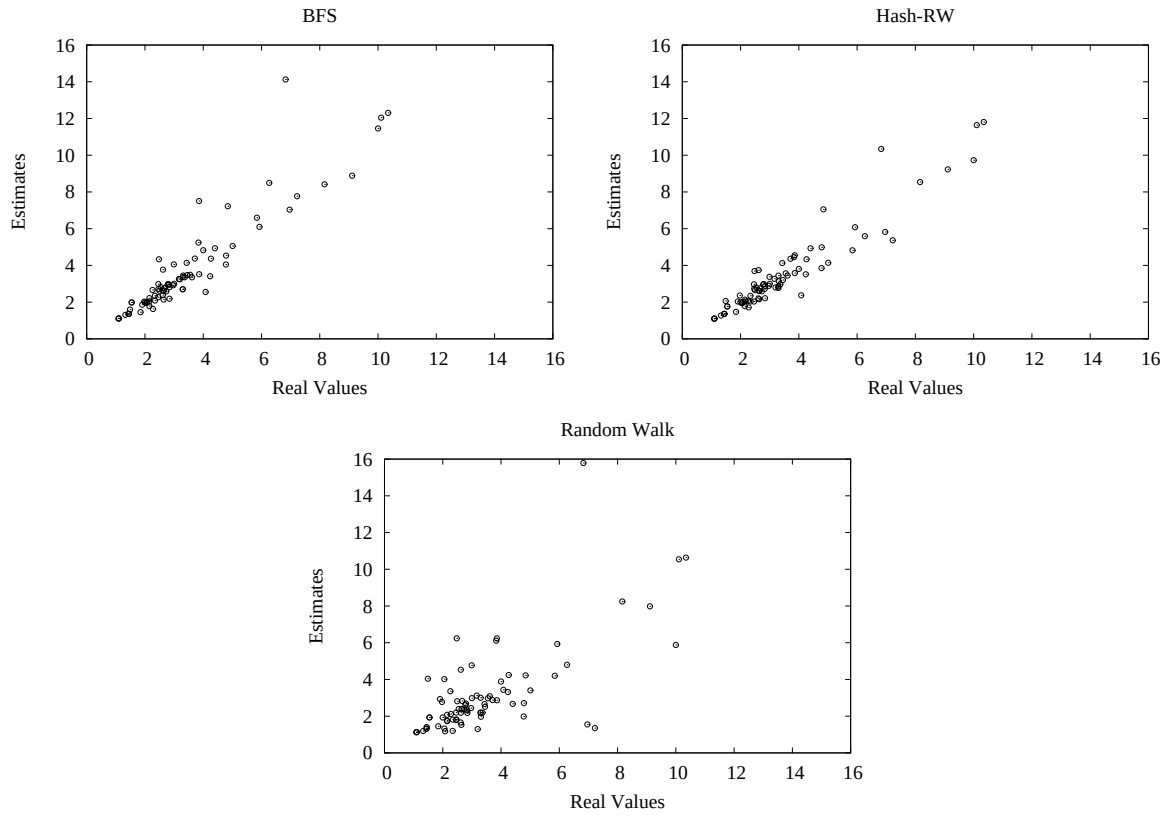

Figure 5: Graphs show an average degree from the whole state space (Real value) and an average degree from a given sample (Estimate).

states. Since a complexity of many graph algorithms depends on a number of edges, estimation of graph edges amount is crucial.

We found out that the vertex degree is almost evenly spread among all vertices of the graph. Therefore, it should be possible to estimate the average degree from quite a small sample of the state space. Fig. 5. shows estimation of the average vertex degree gained by exploration of 5% of the state space. The figure shows results computed by three sampling techniques: BFS, Hash-RW (with $C = 500$) and a simple random walk. For each technique we moreover compute ratios of an estimate and a real value of an average degree and study their distribution. The best results are provided by Hash-RW (standard deviation of the ratio is 0.17). BFS also provides good results (standard deviation is 0.23). Estimates produced by the random walk are poor (standard deviation is 0.44).

We conclude that average degree can be estimated quite easily, nevertheless it matters which technique is used for sampling.

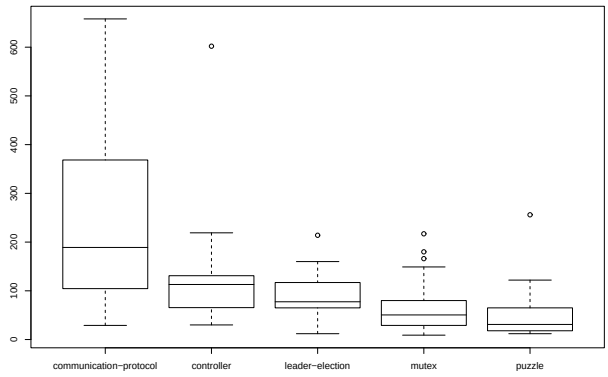

Figure 6: BFS heights sorted by the type of the model. Results displayed using the boxplot method (see Fig. 1. for description).

4.2 BFS Height

Estimation of BFS height can be used for estimation of state space size (see Section 3.3). It can also be used to tune several verification techniques: setting depth limit for random walk search [19] and explicit bounded search [12] ; or setting parameters for techniques using stratified caching [8].

Models in our benchmark have BFS heights mostly between 20 and 600. This interval can be further specified if we restrict to a certain type of models. Fig. 6. shows BFS heights of models according to their type (in Section 3.3 we use median values as BFS height estimates).

We provide also an estimation technique based on sampling. First, we reduce BFS height estimation to the estimation of the largest BFS level index. Since we expect bell-like shape of the BFS level graph (see Section 3.3), the largest level has an index equal approximately to the half of the BFS height of a state space. Hence, we can estimate the index of the largest level and multiply it by two.

To identify the index of the largest level, we use Hash-RW with a special finishing condition. The basic idea of the finishing condition exploits estimation of BFS level widths from Hash-RW level widths and numbers of Hash-RW levels successors. While absolute values of BFS level sizes estimates are quite inaccurate, their relative sizes are preserved — if real BFS level sizes are growing with higher index, estimated sizes are usually also growing and if real sizes are falling, estimated sizes are also falling. Due to these estimates we are able to recognize local maximums in the graph of BFS levels

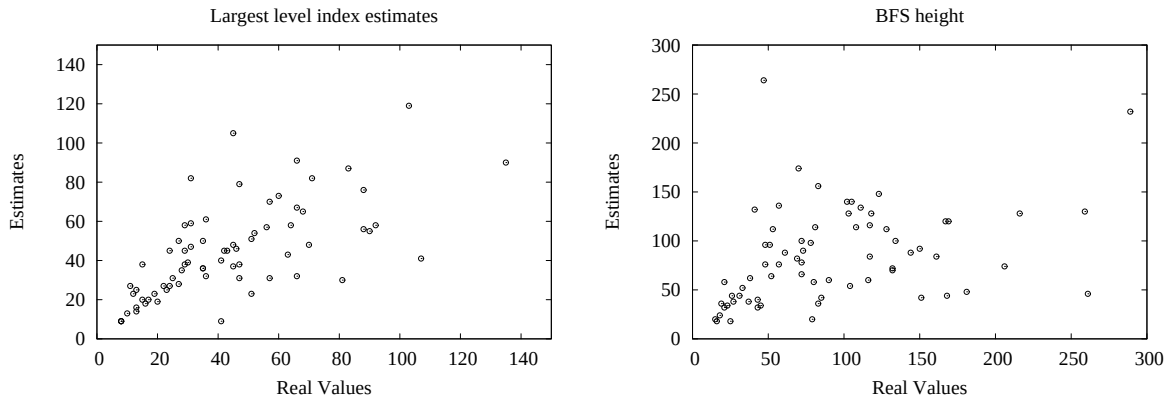

Figure 7: Estimation of the index of the largest level and BFS height (8 dots are outside the displayed area).

using Hash-RW. We ignore small local maximums (specified by a constant which is derived from the training set — the local maximum is small if it is up to 20% larger than the last BFS level size estimate) and the search is stopped when a big local maximum is identified. During the reported experiments Hash-RW explored 6.2% of the state space on average.

Fig. 7. shows estimates of the index of the largest BFS level and the BFS height. It is apparent that computing the BFS height as a double of the index of the largest BFS level brings an additional inaccuracy, but estimates are still reasonable. To evaluate the estimates we again compute ratios of estimates and correct values and their standard deviations. For the index of the largest level the standard deviation is 0.72; for the BFS height the standard deviation is 0.84. This means that BFS height estimates usually do not exceed double of the real height and they are rarely lower than one fifth of the real height.

4.3 Back Level Edges

Some algorithms work more efficiently on models with no or only few back level edges [1], [24]. Therefore, it can be useful to know a ratio of back level edges from all edges in the graph.

Back level edge is a notion closely connected to BFS. There is a straightforward method to determine how much back level edges is among all transitions — to compute the number of all edges A and back level edges B in a sample given by BFS limited to 5% of the state space. Then we estimate a ratio of back level edges as B/A. Since the

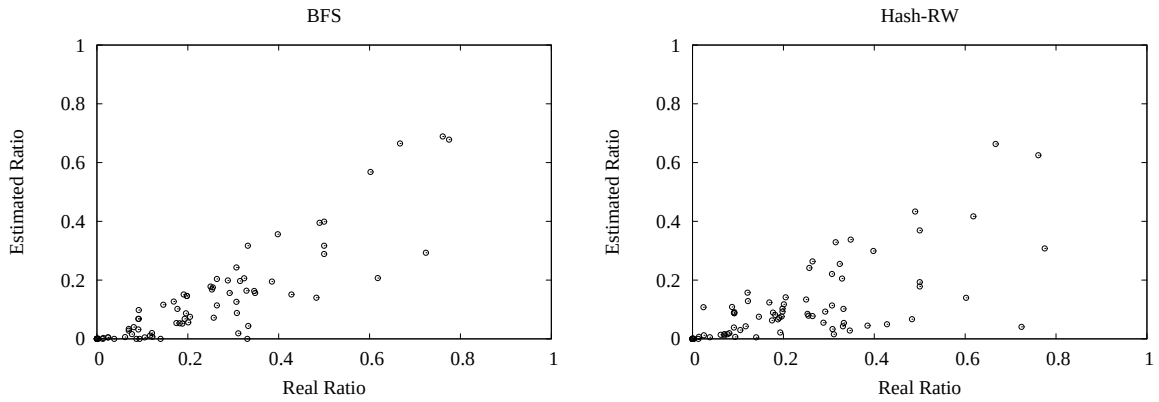

Figure 8: Estimation of back level edges ratio.

number of back level edges is naturally growing as more of the state space is explored, our estimates are practically always below the real values (see Fig. 8.) and the standard deviation of the ratio of estimated and real values is 0.42.

Hash-RW can be used as well. The standard deviation of the ratio is worse (0.58), but the technique can find back level edges unreachable by limited BFS, since back level edges are often hidden in high BFS levels. Consequently, Hash-RW is more successful in deciding, whether the given state space has any back level edges or not (90% success rate for BFS, 99% success rate for Hash-RW).

4.4 Size of the Largest SCC

State spaces have a specific structure of strongly connected components [16], particularly with respect to the size of the largest SCC. In [18], we identify three main classes of state spaces: acyclic state spaces, state spaces with small SCCs, and state spaces with one large strongly connected component (more than 50% states). This classification is relevant for example for selection of a distributed cycle detection algorithm [2] or an SCC detection algorithm.

For estimating this classification we use simple random walk exploration [19]. We run 100 independent random walks through the state space. Each random walk starts at the initial state and is limited to at most 2000 steps. During the walk we store visited states, i.e., path through the state space. If a state is revisited then a cycle is detected and its length can be easily computed. At the end of the procedure, we return the length of the longest detected cycle.

Using training data we identified the following bounds for estimation:

	estimated acyclic	estimated small components	estimated large component
acyclic	28%	0%	0%
small components	0%	16%	4%
large component	0%	18%	34%

Table 4: Results for SCC structure estimation technique.

- If the longest detected cycle is zero (i.e., no cycle is detected) then we estimate that the state space is acyclic.
- If the longest detected cycle is shorter then 69 then we estimate that the state space contains only small components.
- Otherwise, we estimate that the state space contains one large component.

Results of this estimation technique over testing data are in Table 4. The method can safely distinguish between acyclic and cyclic state spaces, models with large component are sometimes wrongly classified as models with small components.

5 Conclusions and Future Work

In this work we study simple techniques for estimation of state space parameters. Particularly, we focus on techniques based on sampling of the state space. We employ breadth-first search sample, depth-first search sample, random walk, and a novel hash-RW technique. The main messages of our work are:

- Estimation of state space parameters is an interesting problem with applications particularly in the distributed environment.
- Some parameters are easy to estimate (e.g., the average degree), other parameters are rather difficult to estimate (e.g., the number of states, the number of back level edges).
- Selection of a sampling technique matters. Each sampling technique is suitable for estimation of different parameters.
- It seems not reasonable to expect accurate estimates of the number of reachable states. However, when we restrict to three estimate classes and combine several

methods, we can get reasonable and useful results. Particularly, it is possible to safely distinguish between huge state spaces and state spaces only slightly larger than a taken sample.

There are several directions for the future work. In this work we restricted our attention to simple techniques. It should be possible to get better estimates by optimizing presented techniques, by parameter tuning, and by incorporating domain specific information.

As an output for a user of a model checker, it would be useful to have on-the-fly estimates of the number of states. Such estimates would be updated regularly during the search (e.g., after the traversal of each BFS level). It would be interesting to have an on-the-fly estimate as an absolute number and to study whether (how fast) the estimate converges to the correct value.

Finally, our long term goal is to use parameter estimates for automation of the verification process [18], i.e., for selection of verification techniques, algorithms, and parameters values.

References

[1] J. Barnat, L. Brim, and J. Chaloupka. Parallel breadth-first search LTL model-checking. In *Proc. 18th IEEE International Conference on Automated Software Engineering*, pages 106–115. IEEE Computer Society, 2003.

[2] J. Barnat, L. Brim, and I. Černá. Cluster-Based LTL Model Checking of Large Systems. In *FMCO'05*, volume 4111 of *LNCS*, pages 259–279. Springer, 2006.

[3] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Rockai, and P. Šimeček. DiVinE - A Tool for Distributed Verification. In *CAV'06*, volume 4144 of *LNCS*, pages 278–281. Springer, 2006. The tool is available at `http://anna.fi.muni.cz/divine`.

[4] Jiri Barnat, Lubos Brim, and Pavel Simecek. I/O Efficient Accepting Cycle Detection. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 281–293. Springer, 2007.

[5] L. Breiman. *Classification and Regression Trees*. CRC Press, 1984.

[6] A. Chamillard. An Empirical Comparison of Static Concurrency Analysis Techniques, 1996.

[7] P. C. Dillinger and P. Manolios. Enhanced Probabilistic Verification with 3Spin and 3Murphi. In *Proc. of SPIN Workshop*, volume 3639 of *LNCS*, pages 272–276. Springer, 2005.

[8] J. Geldenhuys. State Caching Reconsidered. In *Proc. of SPIN Workshop*, volume 2989 of *LNCS*, pages 23–39. Springer, 2004.

[9] P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer, 1996.

[10] M. Hammer and M. Weber. "To Store or not to Store" Reloaded: Reclaiming Memory on Demand. In *Formal Methods for Industrial Critical Systems (FMICS'06)*, 2006. To appear.

[11] Irit Katriel and Ulrich Meyer. Elementary Graph Algorithms in External Memory. In *Algorithms for Memory Hierarchies*, volume 2625 of *Lecture Notes in Computer Science*, pages 62–84, Berlin, Germany, 2003. Springer.

[12] P. Krčál. Distributed Explicit Bounded LTL Model Checking. In *Proc. of Parallel and Distributed Methods in verifiCation (PDMC'03)*, volume 89 of *ENTCS*. Elsevier, 2003.

[13] D. H. Maister. The Psychology of Waiting Lines. In J. A. Czepiel, M. R. Solomon, and C. Suprenant, editors, *The Service Encounter*. Lexington Books, 1985.

[14] Richard P. Martin, Amin M. Vahdat, David E. Culler, and Thomas E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. *SIGARCH Comput. Archit. News*, 25(2):85–97, 1997.

[15] S. Nissen. Implementation of a fast artificial neural network library. Technical report, Department of Computer Science, University of Copenhagen, 2003.

[16] R. Pelánek. Typical Structural Properties of State Spaces. In *Proc. of SPIN Workshop*, volume 2989 of *LNCS*, pages 5–22. Springer, 2004.

[17] R. Pelánek. Web Portal for Benchmarking Explicit Model Checkers. Technical Report FIMU-RS-2006-03, Masaryk University Brno, 2006. `http://anna.fi.muni.cz/models`.

[18] R. Pelánek. Model Classifications and Automated Verification. In *Proc. of Formal Methods for Industrial Critical Systems (FMICS'07)*, 2007. To appear.

[19] R. Pelánek, T. Hanžl, I. Černá, and L. Brim. Enhancing Random Walk State Space Exploration. In *Proc. of Formal Methods for Industrial Critical Systems (FMICS'05)*, pages 98–105. ACM Press, 2005.

[20] W. Peng and K. Makki. On Reachability Analysis of Communicating Finite State Machines. In *Proc. of International Conference on Computer Communications and Networks (ICCCN '95)*, page 58, Washington, DC, USA, 1995. IEEE Computer Society.

[21] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[22] D. Sahoo, J. Jain, S. K. Iyer, D. Dill, and E. A. Emerson. Predictive Reachability Using a Sample-Based Approach. In *Proc. of Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *LNCS*, pages 388–392. Springer, 2005.

[23] J. F. Watson and A. A. Desrochers. A Bottom-Up Algorithm for State-Space Size Estimation of Petri Nets. In *Proc. of International Conference Robotics and Automation (ICRA'93)*, volume 1, pages 592–597. IEEE Computer Society Press, 1993.

[24] Rong Zhou and Eric A. Hansen. Breadth-First Heuristic Search. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 92–100. AAAI, 2004.