

# Improving Procedures for Evaluation of Connectionist Context-Free Language Predictors\*

Henrik Jacobsson, Tom Ziemke

## Abstract

This paper shows how seemingly minor differences in training and evaluation procedures used in recent studies of recurrent neural networks as context free language predictors can lead to significant differences in apparent network performance. We therefore suggest standard evaluation procedures whose use would facilitate better reproducibility and comparability.

*Index Terms* — Context Free Language Prediction, Recurrent Neural Networks, Network Analysis

## 1 Introduction

A number of recent papers have investigated the use of Recurrent Neural Networks (RNNs) for predicting strings belonging to the class of the Context Free Language (CFL)  $\mathbf{a}^n\mathbf{b}^n$  and the Context Sensitive Language (CSL)  $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Each of these papers makes valuable contributions, but when we compared them, we noticed two problems: Firstly, sometimes a number of details of the evaluation method (for evaluating the generalization ability of the networks) were undocumented. Secondly, where details of evaluation were provided, minor differences between the methods used in different papers were found. This led us to carry out a series of experiments with the aim to systematically investigate whether these differences may affect the Estimated Generalization Ability (EGA) for a *given population* of RNNs. Such differences may be an indicator that the reproducibility and comparability of the generalization ability presented in these papers might be questioned.

In our experiments we have varied three aspects of the testing procedure in order to see how the EGA of the RNNs is affected. These aspects are: Firstly, the *string order*, i.e. the order in which strings of different lengths from the grammar  $\mathbf{a}^n\mathbf{b}^n$  are concatenated into the string which the RNN should predict. Secondly, the *maximum string length*, i.e. the highest value of  $n$  of the  $\mathbf{a}^n\mathbf{b}^n$  strings in the test set. The third aspect, *error tolerance* is the degree to which the network is allowed to make mistakes. The reason

that the two first aspects are important is that an RNN is a dynamical system with a potential sensitivity to its initial state which can be based on previous inputs. Variations of these three aspects exist in the above mentioned papers, but are in some cases just vaguely described, if at all. In addition to these three, other important aspects, such as the number of networks, number of repeated tests per network and basic definitions such as “success” are varied and in some cases quite vaguely described.

The structure of this paper is as follows: First the investigated papers are briefly summarized to give an overview of their experimental strategies. Then our experiments designed to evaluate the sensitivity of the EGA with respect to testing procedure are presented. The results of the survey and experiments are then fused into some concluding remarks and recommendations.

## 2 Background

The papers that present results of CFL and CSL predictions with RNNs and their testing approaches are summarized in Table 1. The architectures focused on in these papers were Simple Recurrent Networks (SRNs) [1, 2, 3, 4, 5, 6, 7], Sequential Cascaded Networks (SCNs) [6, 8, 10] and Long Short-Term Memory (LSTM) [9, 11]. The training algorithms used in these papers are either based on gradient descent [1, 2, 3, 4, 5, 6, 8, 9, 10, 11] and/or Evolutionary Hillclimbing (EH) [2, 6, 7]. There are, of course, many other important papers in the field of CFL/CSL prediction and related fields, but those not presenting quantitative studies of the generalization ability have been omitted as they have no direct bearing on our results. Other papers in the field of CFL- and CSL-prediction have also been omitted to make comparisons simpler, i.e. only  $\mathbf{a}^n\mathbf{b}^n$  and  $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$  papers are included.

The training and test set sizes used in the cited papers are presented in Table 1, as well as the ordering of strings in the test set. Where there has been any chance of misunderstanding the structure of the testing set/procedure, we have chosen not to make any assumptions. For example, when the test set is explained as “from depth 1 to 30” [1] or “strings up to  $n = 12$ ” [5] it may be implicit that the strings are ordered in an ascending order, but as no

---

\*Published 2003 in IEEE Transactions on Neural Networks, 14(4), 963-966.

explicit definition of string order is found, these papers are marked as being ambiguous about the test set order.

Among these papers, we found three different test set orderings: *random*, *ascending* and *descending* order. Six out of eleven papers did not explicitly define the order of their test set. The maximum string length of the test set also varied among the papers. Furthermore, the details of the error tolerance were usually not discussed, i.e. it was actually quite unclear in some of the papers whether correct prediction *once* per string occurrence was enough to consider the prediction successful or if the network needed to *consistently* predict all strings correctly. It seems, however, that the former is most commonly used.

It may also be worth noting that two papers [9, 11] used slightly different domains,  $\mathbf{a}^n\mathbf{b}^n\mathbf{T}$  and  $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n\mathbf{T}$ , which strictly speaking are not the same languages as  $\mathbf{a}^n\mathbf{b}^n$  or  $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$ . The terminal symbol  $\mathbf{T}$  gives the network a mechanism for resetting its state in a more deterministic manner than otherwise. The comparison across these domains may therefore not be reliable. Considering only comparisons *within* the domains, however, the terminal symbol may in fact improve comparability due to the potential increase of determinism.

### 3 Experiments

The experiments presented in this paper are aimed towards evaluating whether the *string order*, *maximum string length* and *error tolerance* when testing RNN predictors affect the EGA significantly for given trained populations of networks. We therefore consider the training of the networks a secondary matter, i.e. no effort has been spent on finding optimal parameters for the EH. In effect, the results may not be comparable to other papers (a comparison that should not be done anyway). Instead the training should just be seen as a necessary step to generate populations of networks in which some effects of the testing parameters can be demonstrated.

#### 3.1 The Testing Procedure

The test set is determined by the string order and maximum string length. Three orderings of string are used; random, ascending, and descending. We let the maximum string length of the test set vary between 10 and 100. In each test, exactly 1000 strings of each length are included. The strings are concatenated into the sequence which the network is trained to predict.

The performance of the network is recorded for the 1000 strings of each length it receives. If we consider just one network we will have an estimate of the performance of the network on each individual string length. This performance is typically higher for short strings and lower for long strings. The performance is, however, not necessarily decreasing monotonically and a string with a high  $n$

may be predicted completely accurately, while the strings of length  $n - 1$  could at the same time be completely inaccurately predicted. We have chosen to record the maximum string length that the network processes correctly (string length is something which all previous papers have mentioned when talking about the generalization ability of their networks), but this measurement needs to take into account the nonmonotonic performance degradations for longer strings. The following definition will lead to such a measurement.

The *correctness*,  $c(n)$ , of a network in terms of predicting a given length is defined as

$$c(n) = \frac{\text{no. of correctly predicted strings of length } n}{\text{no. of strings of length } n} \quad (1)$$

where the total number of strings of length  $n$  in this case was 1000 for all  $n$  up to the maximal string length. A correctly predicted string means that at least the predictable part (i.e. not the first  $\mathbf{b}$ ) of the string is correctly predicted. As  $c(n)$  is not monotonically decreasing it can not be used directly to unambiguously define up to which string length the network is successful. In Table 2 an example of a string evaluation is shown. From this example it is clear that there is no obvious way to give statements of which maximum string length the network can handle. In the example, the network can handle all strings up to  $\mathbf{a}^6\mathbf{b}^6$  but fails on some of  $\mathbf{a}^7\mathbf{b}^7$ ,  $\mathbf{a}^{10}\mathbf{b}^{10}$  and  $\mathbf{a}^{11}\mathbf{b}^{11}$ . It can also handle all of  $\mathbf{a}^{13}\mathbf{b}^{13}$ , but none of  $\mathbf{a}^{12}\mathbf{b}^{12}$  or  $\mathbf{a}^{14}\mathbf{b}^{14}$ . Up to what string length should we then say that the network is performing correctly?

To solve this we introduce a recursive definition of correctness, reflecting that the performance on one string length depends also on the performance on all shorter string lengths. The *recursive correctness*,  $c_r(n)$ , is defined as:

$$\begin{aligned} c_r(1) &= c(1) \\ c_r(n) &= c_r(n - 1) \cdot c(n) \text{ for } n > 1 \end{aligned} \quad (2)$$

In the example of Table 2,  $c_r(n)$  is monotonically decreasing and only accepts string lengths for which previous string lengths also have been correctly predicted. The correctly predicted  $\mathbf{a}^{13}\mathbf{b}^{13}$  are now ignored since no correct predictions of  $\mathbf{a}^{12}\mathbf{b}^{12}$  were made.

The *error tolerance* is the quality demand on the network by the experimenter. The highest error tolerance corresponds to the experimenter being satisfied with the RNN correctly predicting strings only *at least once* and the lowest error tolerance is when the RNN needs to correctly predict *all strings*. Chalup and Blair [7] addressed the issue of error tolerance explicitly and defined “weak solutions” and “strong solutions” to correspond to networks satisfying the highest and lowest error tolerance requirements respectively. We adopt these terms in this paper. The EGA (using  $c_r(n)$ ) of the network in the example in Table 2 is then 6 if we consider only strong solutions, and 11 if we only require weak solutions.

Reference	Domain	Training set	Test set order	Test set
Wiles & Elman (1995) [1]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 12$	*	$1 \leq n \leq 30$
Tonkes et al. (1998) [2]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Rodriguez et al. (1999) [3]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 11$	asc	until failure
Tonkes & Wiles (1999) [4]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Bodén et al. (1999) [5]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 10$	*	$1 \leq n \leq 12$
Bodén et al. (2000) [6]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 10$	rand	*
Chalup & Blair (2000) [7]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 20$	rand	$1 \leq n \leq 20^{**}$
—”—	$\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n$	$1 \leq n \leq 20$	rand	$1 \leq n \leq 20^{**}$
Bodén & Wiles (2000) [8]	$\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n$	$1 \leq n \leq 10$	desc	$1 \leq n \leq$ “large $n$ ”
Gers & Schmidhuber (2001) [9]	$\mathbf{a}^n \mathbf{b}^n \mathbf{T}$	$1 \leq n \leq 10$ to $1 \leq n \leq 50$	*	$1 \leq n \leq 1000$
—”—	$\mathbf{a}^n \mathbf{b}^n \mathbf{c}^n \mathbf{T}$	$1 \leq n \leq 10$ to $1 \leq n \leq 50$	*	$1 \leq n \leq 500$
Bodén & Blair (2002) [10]	$\mathbf{a}^n \mathbf{b}^n$	$1 \leq n \leq 10$	*	*
Schmidhuber et al. (2002) [11]	Refers to the data in Gers & Schmidhuber (2001) [9]			

\*=not explicitly defined.

\*\*=incrementally tested during training.

Table 1: A summary of CFL and CSL prediction experiments using various neural network architectures.

$n$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$c(n)$	1.00	1.00	1.00	1.00	1.00	1.00	0.91	1.00	1.00	0.77	0.10	0.00	1.00	0.00
$c_r(n)$	1.00	1.00	1.00	1.00	1.00	1.00	0.91	0.91	0.91	0.70	0.07	0.00	0.00	0.00

Table 2: A realistic example of an evaluation of an RNN by using  $c(n)$  and  $c_r(n)$  of equation 1 and 2. If requiring a strong network, this network’s EGA is up to string length 6 and if only requiring a weak network, the EGA is 11.

### 3.2 Architecture & Training Algorithm

The network architecture used in our experiments is an SRN and the optimisation algorithm is an EH, see [6] for details. The fitness is proportional to the number of correctly predicted strings in a concatenated sequence of strings from  $\mathbf{a}^n \mathbf{b}^n$  with  $1 \leq n \leq 10$  where each string length occurred exactly three times (cf. the testing procedure in the previous section). Three separate fitness functions are used;  $F_{rand}$ ,  $F_{asc}$  and  $F_{desc}$  for random, ascending and descending string length order respectively, i.e. the only difference between the fitness functions is the ordering of the strings. It should be noted that the aim of the experiment is *not* to evaluate the differences between these populations but to evaluate how the EGA varies for these fixed populations under different testing strategies. The use of three different populations may reveal different effects the testing procedure may have on the estimated results. In fact *any* sufficiently large population would do as the goal basically is to show that there are populations for which testing procedure differences significantly affect the estimated performance.

The evolutionary algorithm was run for 10,000 generations with a mutation rate of  $\sigma = 1.0$  and a population size of 100 of which 20 were selected as elite. The elite group was saved to the next generation and was the group from which new networks were generated. 120 runs were carried out for each fitness function with different random seeds and the best network of each successful end-population was saved for further analysis. A population was deemed successful if at least one of its networks correctly predicted (the predictable part of) *all* strings in the training set.

## 4 Results

### 4.1 Training Results

Of the 120 experiments with each of the three fitness functions  $F_{rand}$ ,  $F_{asc}$  and  $F_{desc}$  the number of successful (in terms of correctly predicting the entire training set) runs were 114, 75 and 76 respectively. All the statistics will be based on the best network of each successful population. It is worth noting that the success rate is much higher for

$F_{rand}$  than for  $F_{asc}$  and  $F_{desc}$ . This is probably due to higher sensitivity to local optima for the deterministic fitness functions. Subsequent experiments (not documented here) indicated that for higher values of the mutation parameter,  $\sigma$ , this problem vanishes.

## 4.2 Estimated Generalization Abilities

The resulting EGA of networks generated with the three fitness functions tested under different conditions are shown in Table 3. The maximum correctly predicted string length of each successful network was calculated according to equations 1 and 2 as in the example in Table 2.

### 4.2.1 The Effect of Error Tolerance Level

The effect of demanding weak or strong networks is clearest when the networks are tested on strings in a random order. The EGA is half or lower for the strong solutions given a high enough maximum string length of the test set. The error tolerance effect is still there with a test set in ascending order, but weaker.

Interestingly, the error tolerance has virtually no effect at all when testing on strings in a descending order. We speculate that this is due to the RNN gradually receiving simpler and simpler strings, resulting in the exact same behaviour every time, i.e. the network either correctly predicts all strings of a specific length or none at all.

One should keep in mind that, as the test set has 1000 replicas of each string length, strong solutions correctly predict 1000 out of 1000 strings, whereas weak solutions need only predict 1 out of 1000 correctly. In our opinion, this makes strong solutions much more interesting.

### 4.2.2 Effects of Maximum String Length

The effect of the maximum string length ( $N$  in Table 3) differs depending on test set order, error tolerance and fitness function. When only considering strong solutions and random test set order, a higher  $N$  leads to a significantly lower EGA for all networks. The opposite seems to be true for most weak solutions for all test set orderings and networks. For ascending test set order, the degrading performance for higher values of  $N$  is not as clear as when testing on randomly ordered strings. For tests on strings in descending order,  $N$  has no degrading effect.

### 4.2.3 The Effects of String Order

String order is perhaps one of the more interesting aspects of the testing procedures, as there were three distinct orderings found in previous work while most papers did not describe this aspect of testing explicitly. In our experiments, string order played two roles, in the training and testing of networks. The networks trained on the different training sets can be clearly ranked in terms of perfor-

mance. Networks trained on  $F_{rand}$  are clearly better than  $F_{asc}$  which is clearly better than  $F_{desc}$ .

A ranking of the test sets is not as straightforward. Considering only strong solutions it is, however, clear that a randomly ordered test set is tougher than the ascending order which is in turn tougher than the descending order. For weak solutions the randomly ordered test set gives the highest results. This is not surprising as weak solutions need only 1 out of 1000 strings correctly predicted of every string length and a randomly ordered set gives the network a higher variety of initial states of which some may lead to a correct prediction.

It is interesting to see that, as a validation of the network training, all networks handle their training sets perfectly and that the networks trained with  $F_{rand}$  also handle the other training sets perfectly. Networks trained on randomly ordered strings thus seem more robust.

Although the results of the randomly ordered test set seem to be most sensitive to the other parameters (i.e. string length and error tolerance), in our opinion, this test provides the most interesting results, as the network will be tested more rigorously.

## 5 Discussion and Conclusions

It is clear from table 3 that changes in the testing procedure render significantly different results. These effects are also not consistent for the three populations and can therefore at this stage not be predicted. These results are not surprising, as it is well known that initial conditions may affect the behavior of dynamical systems, and hence affect the performance of RNNs, a subset of dynamical systems. The cited papers, implicitly or explicitly, touch the dynamical nature of RNNs, but in the construction or description of the experimental setup this important issue often does not receive much attention. All papers describe the architectures and algorithmic details of the learning techniques quite thoroughly and present insightful, detailed analyses of individual networks. But without a proper description of the testing procedures used to generate quantitative results, reproducibility and comparability are lost. Three papers also make cross paper comparisons [9, 12, 11] in the domain of these papers, comparisons that, due to the problems pointed out here, may be questioned. For the same reasons, it would also not make sense to compare our results to those of any other paper using different testing procedures.

Some practical recommendations for future research in this area: Train and test sets should be ordered randomly to give both robust networks and a thorough testing of these networks. Only strong networks (or perhaps a slightly relaxed version of “strong”, e.g. 90-99% correct) should be considered. A network solving a task only (at least) once is far less interesting than those solving it consistently. Since the results also indicate that the maximum

string length in the test set has a significant effect on the results the *expected* performance may affect the *measured* performance directly, since the maximum string length in the test set will probably be chosen based on the expected performance. Hence, the maximum string length in the test set should be varied, perhaps starting with a low value and then increasing stepwise.

What can be learned from this is that to guarantee reproducibility, the description of the generation of testable objects has to be complemented with a description of the testing procedure applied to these objects. In the cited papers the architectures, training procedures and analysis of individual RNNs came out mostly crystal clear to the reader, while some crucial details of the testing methods did less so. So our final, and most important recommendation, is to recognize that the analysis tools are as important a part of the data generation as the networks themselves.

## References

- [1] J. Wiles and J.L Elman. Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent neural networks. In *Proc. of the 17:th Annual Conf. of the Cogn. Sci. Soc.*, pages 482–487. Cambridge MA: MIT Press, 1995.
- [2] B. Tonkes, A. Blair, and Janet Wiles. Inductive bias in context-free language learning. In *Proc. of the 9:th Australian Conference on Neural Networks*, pages 52–56, 1998.
- [3] P. Rodriguez, J. Wiles, and J. L. Elman. A recurrent network that learns to count. *Connection Science*, 11:5–40, 1999.
- [4] B. Tonkes and J. Wiles. Learning a context-free task with a recurrent neural network: An analysis of stability. In R. Heath, B. Hayes, A. Heathcote, and C. Hooker, editors, *Proc. of the 4:th Biennial Conference of the Australasian Cognitive Science Society (OzCogSci97)*, 1999.
- [5] M. Bodén, J. Wiles, B. Tonkes, and A. Blair. Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *Proceedings of ICANN 99*, pages 359–364, Edinburgh, 1999.
- [6] M. Bodén, H. Jacobsson, and T. Ziemke. Evolving context-free language predictors. In *Proc. of GECCO*, pages 1033–1040. Morgan Kaufmann, 2000.
- [7] S. K. Chalup and A. D. Blair. First order recurrent neural networks learn to predict a mildly context-sensitive language. Technical Report ISBN 0-7259-1109-3, Dept. of Computer Science and Software Engineering, The University of Newcastle, 2000.
- [8] M. Bodén and J. Wiles. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, 12(3/4):196–210, 2000.
- [9] F. A. Gers and J. Schmidhuber. LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12(6), 2001.
- [10] M. Bodén and A. Blair. Learning the dynamics of embedded clauses. *Applied Intelligence: Special issue on natural language and machine learning*, in press.
- [11] J. Schmidhuber, F. Gers, and D. Eck. Learning nonregular languages: A comparison of Simple Recurrent Networks and LSTM. *Neural Computation*, 14(9):2039–2041, 2002.
- [12] M. Bodén and J. Wiles. On learning context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 13(2):491–493, 2002.

$N$	Networks trained on $F_{rand}$ (114 RNNs)				Networks trained on $F_{asc}$ (75 RNNs)				Networks trained on $F_{desc}$ (76 RNNs)			
	strong		weak		strong		weak		strong		weak	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
<b>Test set in random order</b>												
10	10.00 (0.00)	10	10.00 (0.00)	10	6.79 (0.45)	10	10.00 (0.00)	10	4.07 (0.48)	10	10.00 (0.00)	10
15	8.95 (0.45)	15	12.09 (0.16)	15	5.11 (0.62)	15	12.13 (0.26)	15	3.46 (0.56)	15	12.36 (0.21)	15
20	7.77 (0.53)	20	12.48 (0.22)	20	3.33 (0.58)	17	13.12 (0.31)	20	1.58 (0.37)	14	12.96 (0.36)	20
25	6.51 (0.50)	20	12.55 (0.23)	23	2.72 (0.54)	17	13.53 (0.36)	25	1.37 (0.35)	14	13.55 (0.38)	25
50	5.81 (0.48)	20	12.63 (0.24)	23	2.00 (0.45)	17	14.19 (0.61)	49	1.32 (0.35)	14	14.53 (0.59)	36
100	5.81 (0.48)	20	12.62 (0.24)	23	2.00 (0.45)	17	14.08 (0.60)	49	1.33 (0.35)	14	14.04 (0.49)	30
<b>Test set in ascending order</b>												
10	10.00 (0.00)	10	10.00 (0.00)	10	10.00 (0.00)	10	10.00 (0.00)	10	6.49 (0.49)	10	8.80 (0.31)	10
15	10.84 (0.34)	15	11.71 (0.16)	15	9.24 (0.61)	15	11.43 (0.28)	15	6.49 (0.64)	15	8.78 (0.52)	15
20	11.10 (0.37)	20	11.97 (0.22)	20	8.88 (0.66)	20	11.65 (0.33)	20	5.16 (0.63)	20	8.75 (0.56)	20
25	10.26 (0.44)	21	11.98 (0.22)	21	7.32 (0.75)	25	11.71 (0.36)	25	5.26 (0.67)	21	8.59 (0.58)	21
50	10.80 (0.38)	20	11.98 (0.22)	21	8.07 (0.67)	19	12.00 (0.54)	45	5.00 (0.64)	21	8.51 (0.59)	21
100	10.80 (0.38)	20	11.98 (0.22)	21	8.07 (0.67)	19	12.00 (0.54)	45	5.00 (0.64)	21	8.51 (0.59)	21
<b>Test set in descending order</b>												
10	10.00 (0.00)	10	10.00 (0.00)	10	9.33 (0.22)	10	9.36 (0.22)	10	10.00 (0.00)	10	10.00 (0.00)	10
15	11.64 (0.16)	15	11.64 (0.16)	15	10.97 (0.36)	15	10.97 (0.36)	15	10.50 (0.24)	15	10.50 (0.24)	15
20	11.89 (0.22)	20	11.89 (0.22)	20	11.08 (0.45)	20	11.08 (0.45)	20	10.51 (0.32)	20	10.51 (0.32)	20
25	11.90 (0.22)	21	11.90 (0.22)	21	11.16 (0.46)	25	11.16 (0.46)	25	10.63 (0.33)	23	10.63 (0.33)	23
50	11.90 (0.22)	21	11.90 (0.22)	21	11.43 (0.62)	45	11.43 (0.62)	45	10.61 (0.32)	21	10.61 (0.32)	21
100	11.90 (0.22)	21	11.90 (0.22)	21	11.43 (0.62)	45	11.43 (0.62)	45	10.61 (0.32)	21	10.61 (0.32)	21

Table 3: The average, standard deviation (in parentheses), and maximum length the networks was deemed to process correctly. The performance is evaluated on networks generated with the three different fitness functions,  $F_{rand}$ ,  $F_{asc}$  and  $F_{desc}$ . The results are separated into the three different test sets and results for weak and strong solutions are presented separately. The results for different maximum string lengths  $N$  are also shown separately.