Mirror Adaptive Random Testing

T.Y. Chen, F.C. Kuo*, R.G. Merkel, S.P. Ng
School of Information Technology, Swinburne University of Technology
Email: {tchen, dkuo, rmerkel, sng}@it.swin.edu.au

Abstract

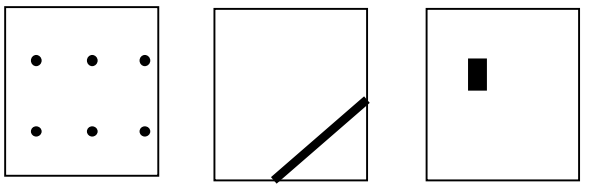
Adaptive random testing (ART) has recently been introduced to improve the fault-detection effectiveness of random testing (RT) for certain types of failure-causing patterns. However, ART requires extra computations to ensure an even spread of test cases, which may render ART to be less cost-effective than RT. In this paper, we introduce an innovative approach, namely Mirror Adaptive Random Testing (MART), to reduce these computations. Our simulation results clearly show that MART does improve the cost-effectiveness of ART.

Keywords Adaptive Random Testing, Random testing, Black box testing, Test case selection, Software testing

1. Introduction

Testing has always been considered to be an inevitable though expensive activity to ensure the quality of a software system. Generally speaking, a tester attempts to uncover as many faults as possible with the given testing resources. There are two common approaches to test case generation: black-box and white-box. Black-box testing selects test cases without referring to the structure of the program, while white-box testing selects test cases by referring to the structure of the program [14]. In this paper, our discussion focuses on some black-box techniques, namely random testing and some of its enhanced methods, under the assumption of uniform distribution of test cases. Since random testing is intuitively simple, easy to implement and also allows statistical reliability of the software system to be inferred, it is one of the commonly used testing techniques for practitioners [8, 10, 11, 13]. However, some researchers think that random testing is a poor method as it does not make use of any information to guide the generation of test cases [12].

* Corresponding Author



A. Point pattern B. Strip pattern C. Block pattern

Figure 1.The three types of failure-causing patterns

Chan et al. [1] have observed that failure-causing inputs form certain kinds of patterns. They have classified these failure-causing patterns into three categories: point, strip and block patterns. These patterns are illustrated in Figure 1, where we have assumed that the input domain is 2-dimensional.

A point pattern occurs when the failure-causing inputs are either stand alone inputs or cluster in very small regions. A strip pattern and a block pattern refer to those situations when the failure-causing inputs form the shape of a narrow strip and a block in the input domain, respectively. The following examples give sample C language program faults that yield these types of failure patterns.

Example 1. Point failure-causing pattern

```
int X, Y, Z;
do{
        scanf("Enter X, Y: %d, %d", X, Y);
} while (X <=0 || Y <= 0);
if (((X % 4) == 0) && ((Y % 6) ==0))
        Z = X / (2 * Y);
   /* the correct statement should be " Z = X / 2 * Y;"*/
  else
        Z = X * Y;
  printf("%d", Z);
```

In Example 1, the program's output will be incorrect when X and Y are divisible by 4 and 6, respectively. Therefore, the distribution of the failure-causing inputs forms a point pattern as schematically shown in Figure 1A.

Example 2. Strip failure-causing pattern

```
int X, Y, Z;
do{
        scanf("Enter X, Y: %d, %d", X, Y);
} while (X <=0 || Y <=0);
if ( 2 * X - Y > 10)
 /* the correct statement should be " if (2 * X -Y > 18)" */
        Z = X / 2 * Y;
else
        Z = X * Y;
printf("%d", Z);
```

In Example 2, the program will return incorrect outputs when X and Y satisfy the condition that $18 \geq 2*X - Y > 10$. In other words, failure-causing inputs lie between the lines $2*X - Y = 18$ and $2*X - Y = 10$, giving rise to the strip pattern as schematically shown in Figure 1B.

Example 3. Block failure-causing pattern

```
int X, Y, Z;
do{
        scanf("Enter X, Y: %d, %d", X, Y);
} while (X <= 0 || Y <= 0);
if (( X >= 10 && X <= 11) && (Y >= 10 && Y <= 12))
        Z = X / (2 * Y);
      /*the correct statement should be "Z = X / 2* Y;"*/
else
        Z = X * Y;
printf("%d", Z);
```

In Example 3, the program's output will be incorrect when X and Y satisfy the conditions that $11 \geq X \geq 10$ and $12 \geq Y \geq 10$, respectively. Thus, failure-causing inputs fall into the region with boundaries, X = 11, X = 10, Y = 12 and Y = 10. This pattern of failure-causing inputs is known as a block pattern and is schematically shown in Figure 1C.

Chan et al. [1] have observed that strip and block patterns of failure-causing inputs are more common than point patterns in practice. Recently, Chen et al. [4, 5] have proposed the notion of adaptive random testing (ART) which is a failure-pattern-based random testing technique. In random testing, test cases are simply generated in a random manner. However, the randomly generated test cases may happen to be close to each other. In ART, test cases should be not only randomly selected but also widely spread. The rationale is that widely spread test cases are expected to have a greater chance of hitting the non-point failure-causing regions. However, extra computations may be required to ensure an even spread of test cases.

A simple ART implementation, known as the Fixed Size Candidate Set ART (FSCS-ART), is described as follows. Let $\boldsymbol{T} = \{T_1, T_2, \ldots, T_l\}$ denote the set of executed test cases and $\boldsymbol{C} = \{C_1, C_2, \ldots, C_k\}$ denote the set of candidate test cases of fixed size k. Initially, $\boldsymbol{T}$ is empty, and an input is randomly chosen as a test case. If this test case reveals a failure, stop the process, otherwise add it to $\boldsymbol{T}$. Then randomly generate $\boldsymbol{C}$ and select the candidate test case from $\boldsymbol{C}$, say C_i, which is furthest away from all executed test cases in $\boldsymbol{T}$, as the next test case for execution. If C_i does not reveal any failure, add C_i to $\boldsymbol{T}$. Repeat the whole process until either a failure is detected or testing resources are exhausted. For a program with an N-dimensional input domain, the furthest candidate test case can be chosen according to the Cartesian distance between the candidate and its closest neighbour in $\boldsymbol{T}$. The selection criterion adopted is to maximise the minimum Cartesian distance between the candidates and the test cases already executed. Let us denote the Cartesian distance between two test cases $P = (p_1, p_2, \ldots, p_N)$ and $Q = (q_1, q_2, \ldots, q_N)$ by $dist(P,Q) = \sqrt{\sum_{i=1}^{N} (p_i - q_i)^2}$. The best one out of the k random candidates $C_1, C_2, \ldots, C_k$ to be chosen as the next test case with respect to the already executed test cases $T_1, T_2, \ldots, T_l$ is the candidate Cj such that for any $r \in \{1, 2, \ldots, k\}$,

$$\min_{i=1}^{l} dist(C_j, T_i) \quad \geq \quad \min_{i=1}^{l} dist(C_r, T_i)$$

As the number of distance computations required is of order l^2, FSCS-ART may incur significant distance computation when l is large.

There are several commonly used metrics of fault detection effectiveness [4, 5, 7]. They include: P-measure which is defined as the probability of detecting at least one failure; E-measure which is defined as the expected number of detected failures; and F-measure which is defined as the number of test cases required to detect the first failure. In this paper, we have followed the practice of previous studies of ART to use the F-measure as the fault-detection effectiveness metric.

Previous studies have shown that the F-measure of ART may be as low as about half of the F-measure of random testing [4]. However, the savings in F-measure using ART may be outweighed by the associated extra computations, in particular for the situation where the cost of executing tests is low. We attempt to reduce these computations by proposing an innovative approach to improve the cost-effectiveness of ART.

In this paper, Section 2 describes our innovative approach. Section 3 reports the simulation and their results. Conclusion and discussion are presented in Section 4.

2. Mirror Adaptive Random Testing

Randomness and an even spread of test cases are the core characteristics of ART. However, the computations that are required to ensure an even spread of test cases may be expensive, as in the case of FSCS-ART. To address this problem, we introduce the notion of *Mirror Adaptive Random Testing* (MART). In MART, the input domain of the program to be tested is divided into m disjoint subdomains. One subdomain is designated as the original subdomain, and the ART algorithm is only applied to this original subdomain. If the selected test case in the original subdomain is executed and failure is not detected, this test case is *mirrored* successively into other subdomains, known as *mirror subdomains*. This is achieved by means of *mirror functions*, which generate (m-1) distinct test cases, one within each of the mirror subdomains. If none of these (m-1) test cases reveals a failure, resume the process of ART in the original subdomain.

The way of dividing the input domain in MART is referred as *mirror partitioning* in this paper. As a pilot study, the input domain is partitioned into disjoint subdomains with equal size and shape. Some examples of mirror partitioning of a 2-dimensional input domain are illustrated in Figure 2.

We denote a mirror partitioning in terms of the number of partitions formed along each dimension of the input domain. For the leftmost mirror partitioning in Figure 2, the input domain is vertically partitioned in the middle producing two sections along the X-axis, and there is no subdivision on the Y-axis. In this paper, we adopt the notation X2Y1 to denote this particular mirror partitioning. In general, XiYj denotes the mirror partitioning in which the input domain is partitioned into $i \times j$ subdomains such that there are i and j equal sections along the X and Y axes respectively.

For the X2Y1 configuration in Figure 2, we use the notation $\{(0, 0), (u, v)\}$ to represent the entire input domain where (0, 0) and (u, v) are respectively the leftmost bottom corner and the rightmost top corner of the 2-dimensional rectangle. Without loss of generality, we have assumed that the leftmost bottom corner is always at the origin. After partitioning the input domain into two equal halves along the X-direction, the two disjoint subdomains D_1 and D_2, denoted by $\{(0, 0), (u/2, v)\}$ and $\{(u/2, 0), (u, v)\}$, respectively, are created. Suppose D_1 is designated as the original subdomain. Then, ART is applied to D_1 to generate test cases, and a mirror function is used to generate corresponding test cases, known as images, in the mirror subdomain D_2.

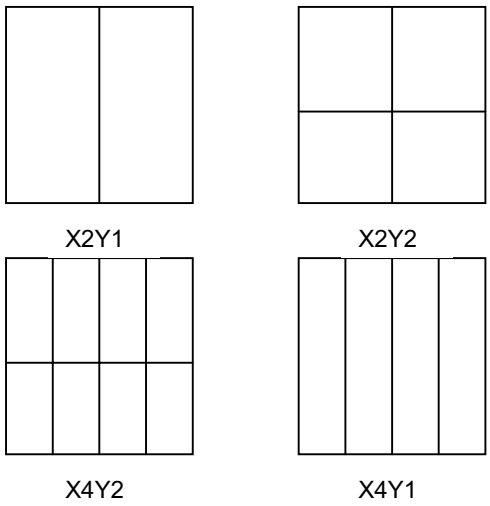
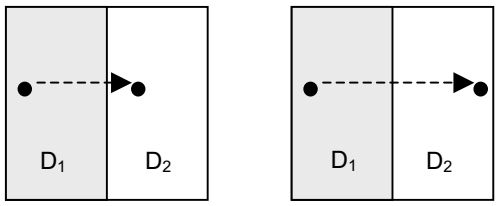


Figure 2. Some simple ways of mirror partitioning

There are many potential mirror functions. One simple function to achieve the *mirrored* effect is to linearly translate the test case from one subdomain to another, that is, Translate(x, y) is defined as $(x + (u/2), y)$. Another mirror function is a simple reflection with respect to the vertical line at $x = u/2$, that is, Reflect$(x, y) = (u - x, y)$. Figure 3 illustrates the mapping of test cases by Translate and Reflect in X2Y1 MART. Both of these mirror functions were used in our simulations to investigate their effects on the fault-detection effectiveness of MART.

When failure-causing inputs cluster together in the form of block or strip patterns, ART has a smaller F-measure than that of random testing [4]. In MART, the ART process is used to select a test case in the original subdomain and a one-to-one mapping is then successively applied to generate images in the mirror subdomains. In the two previously presented examples of mirror functions, test cases generated by linear translation mapping are simply replications of those generated by ART in the original subdomain, whilst test cases generated by reflection mapping are mirrored



Mapping of test case by Translate( )

Mapping of test case by Reflect( )

Figure 3. Simple mirror functions for mapping of test cases

replications of those generated by ART in the original subdomain. Effectively, ART has been applied to all mirror subdomains. It may be argued that the same patterns of selected test cases in each subdomain may collectively violate the intuition of spreading test cases evenly across the entire input domain. We argue that such a violation, if any, can be kept to a minimum by using only a small number of mirror subdomains in MART. To validate our argument, simulations are conducted to investigate the effectiveness of MART under different conditions. Details of the simulation results are discussed in Section 3.

As previously discussed, in FSCS-ART, every test case except the first one requires distance computations. However, in MART, distance computations are only required to generate test cases in the original subdomain.

In FSCS-ART which uses a candidate set **C** of size k, the number of distance calculations performed in a run of n test cases is

$$k \sum_{i=1}^{n} i = \frac{kn(n+1)}{2}$$

Hence, the distance calculations in FSCS-ART are of the order of n^2.

On the other hand, for MART using the FSCS-ART algorithm with candidate set of size k and m mirror subdomains, the number of distance calculations performed in a run of n test cases (without loss of generality, assume n is a multiple of m) is

$$k \sum_{i=1}^{\frac{n}{m}} i = \frac{kn(n+m)}{2m^2}$$

In other words, MART needs only about $1/m^2$ as many distance calculations as the FSCS-ART when both have executed the same number of test cases.

Increasing the number of mirror subdomains will decrease the number of distance computations. However, more mirror subdomains may lead to higher F-measure, equivalently, less effective testing. This can be illustrated by the following simple example.

Example 4

Consider a square input domain with failure rate θ, which is partitioned into m square subdomains as shown in Figure 4.

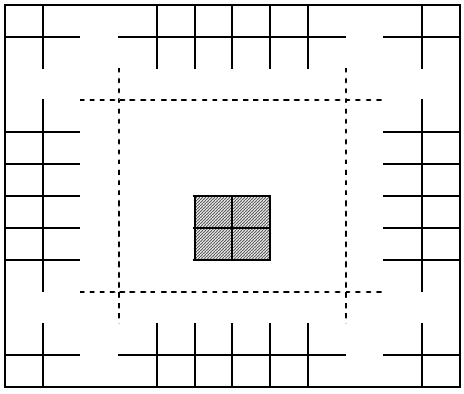


Figure 4. A simple example of very large number of subdomains

Assume that the failure pattern is a square block and coincides exactly with 4 subdomains which are shaded as shown in Figure 4. Then, $\theta = 4/m$. Further assume that the subdomains are sequentially selected for testing. In this case, the expected F-measure of MART is $m/2$ because on average, half of the m squares need to be chosen for testing prior to detecting a failure. However, when random testing is applied on the entire input domain, the expected F-measure is $1/\theta = m/4$. In other words, MART's expected F-measure is double that of random testing in such a situation. Hence, it may not be cost effective to perform MART in this case. This example also illustrates the expected F-measure of MART depends on the subdomain selection method.

3. Simulation – Comparison between MART and FSCS-ART

Simulations were designed and performed to investigate fault-detection effectiveness of MART under various conditions. In our simulation analysis, we assumed that the input domain was a 2-dimensional square. In each simulation, a failure-causing region of failure rate θ was randomly assigned within the input domain with a selected mirror partitioning. MART, as described in Section 2, was applied and its F-measure was recorded. The process was repeated until we could obtain a statistically reliable mean value of the F-measure. Using the central limit theorem [9], the sample size required to estimate the mean of F-measure with an accuracy range of ± r% and a confidence level of $(1 - \alpha) \times 100\%$ is given by

$$S = \left(\frac{100 \cdot z \cdot \sigma}{r \cdot \mu} \right)^2$$

where z is the normal variate of the desired confidence level, μ is the population mean and σ is the population standard deviation.

In all our simulations, confidence level and accuracy range were chosen as 95% and ± 5% respectively. In other words, simulations were repeated until the sample mean of F-measure was accurate within 5% of its value at 95% confidence level. Since μ and σ were unknown, the mean ($\bar{x}$) and standard deviation (s) of the collected F-measure data were used instead respectively. For 95% confidence, z = 1.96. Hence, S could be evaluated as follows:

$$S = \left(\frac{100 \cdot 1.96 \cdot s}{5 \cdot \bar{x}} \right)^2$$

3.1. Effect of Types of Failure Patterns

In this series of simulations, the performances of X2Y1 MART for block, strip and point failure-causing patterns were investigated using a failure rate of 0.001. Hence, the F-measures for random testing are expected to be 1000. Both mirror functions, Translate and Reflect were used in the simulations. T-map and R-map in Tables 1 to 3 denote the Translate and Reflect mirror functions respectively.

For a block failure pattern, a square was used as the failure region. Table 1 shows the performances of FSCS-ART and X2Y1 MART for block failure patterns. The average F-measure of FSCS-ART is 633. For X2Y1 MART, the average F-measures using Translate and Reflect mirror functions are 635 and 638, respectively. Hence, X2Y1 MART is as effective as FSCS-ART with respect to F-measure, but its distance computations are only about one quarter of those for FSCS-ART.

For a strip failure pattern, a narrow strip of total size equivalent to a failure rate of 0.001 was used as the failure pattern, and the results are presented in Table 2. The average F-measure of FSCS-ART is 782, whilst the average F-measures of X2Y1 MART using Translate and Reflect mirror functions are 787 and 796, respectively. The simulation results again show that X2Y1 MART performs as effectively as FSCS-ART with respect to F-measure but with only about one quarter of distance calculations as required in FSCS-ART.

In our simulations for point pattern, 10 circular regions were randomly located in the input domain without overlapping each other. The total size of the circles is equivalent to the failure rate of 0.001. Hence, the expected F-measure for random testing is 1000 as in the previous two simulations. As shown in Table 3, the average F-measure of FSCS-ART is 927, while the average F-measure of X2Y1 MART are 938 and 928 for Translate and Reflect mirror functions, respectively. X2Y1 MART and FSCS-ART still have comparable F-measures but X2Y1 MART's distance computations are only about a quarter of those for FSCS-ART.

In summary, our simulations have shown that the F-measure of X2Y1 MART is very similar to that of FSCS-ART, but with an approximately 75% reduction in distance computations. As expected, FSCS-ART and MART perform the best when the failure-causing inputs are of block patterns.

3.2. Effects of Mirror Partitionings and Failure Rates

In this series of simulations, the failure-detection effectiveness of MART was investigated for different mirror partitionings and different failure rates. All simulations in this series of analyses used a square block failure region randomly selected within the input domain. Several simple mirror partitionings were chosen to investigate the effects of different numbers of mirror partitions along both X and Y axes on the MART performance. In each specified MART partitioning, the simulation was repeated with a range of failure rate from 0.0001 to 0.05. As in Section 3.1, each simulation was conducted by using both the Translate and Reflect mirror functions. The results were shown in Tables 4 to 9.

The data in Tables 4 to 9 consistently show that for the mirror partitionings and the failure rates under study, MART has comparable F-measures as FSCS-ART but requiring fewer distance computations. Furthermore, neither the translation mapping nor the reflection mapping always delivers a smaller F-measure than the other.

4. Discussion and Conclusions

Previous investigations demonstrate that ART uses fewer test cases to detect the first failure than RT. It has been suggested that if RT was chosen as the testing method, ART may be used instead. However, extra computations which may be required in ART to ensure an even spread of test cases, may render ART to be less cost-effective than RT. We have proposed the Mirror Adaptive Random Testing (MART) as an innovative approach to reduce these computations. In MART, ART algorithm is only applied to the original subdomain; and a simple mirror function is used instead to generate test cases in the mirror subdomains. We

have applied this approach to an ART implementation, namely FSCS-ART; and conducted a series of simulations to investigate whether MART is as good at detecting failures as FSCS-ART. The results of our preliminary simulation show that MART performs as effectively as FSCS-ART with respect to F-measure but with only about one quarter of distance calculations as required in FSCS-ART. It shows that MART could be more cost effective than FSCS-ART and hence be used instead. Although our discussions and simulations of MART have so far only referred to FSCS-ART, the technique of *mirroring* is obviously applicable to other implementations of ART, such as restricted random testing [2, 3]. We are currently investigating the integration of mirroring and restricted random testing.

When applying the technique of mirroring to any implementation of ART, we need to ensure that the design of mirroring also preserves the degree of evenness in the spread of test cases that exists in the original subdomain. Clearly, mirror functions play a key role in preserving the even spreading of test cases in the mirror subdomains. If the test cases generated by any ART implementation are evenly distributed in the original subdomain and the mirror function used does not change their relative spacing, test cases generated by the mirror function would be expected to be evenly spread in the mirror subdomains as in the original subdomain. Research on mirror functions is now being undertaken.

As shown in Example 4, the number of mirror subdomains has a significant impact on the F-measure. In this study, we have restricted our simulations to a small number of mirror subdomains because the use of many mirror subdomains may possibly be in conflict with the goal of having an even spread of test cases across the entire input domain. Detailed analysis of the effect of the number of mirror subdomains on the performance of MART is beyond the scope of this paper and is currently under investigation.

Chen et al. [5] have recently pointed out that there exist some interesting relationships between ART and the Proportional Sampling Strategy (PSS). PSS is a test allocation scheme that guarantees to have an equal or higher probability of detecting at least one failure (P-measure) than random testing [6]. Since MART can be viewed as a combination of PSS and ART, it may be interesting to further investigate the relationship between PSS and MART.

Acknowledgement

The authors would like to thank Jim Sykes for his invaluable comments on the manuscript of this paper.

References

[1] F.T. Chan, T.Y. Chen, I.K. Mak, and Y.T. Yu, "Proportional Sampling Strategy: Guidelines for Software Testing Practitioners", *Information and Software Technology*, Vol.38, No.12, pp.775-782, 1996.

[2] K.P. Chan, T.Y. Chen, and D. Towey, "Restricted Random Testing", *Proceedings of the 7th European Conference on Software Quality*, pp321-330, 2002.

[3] K.P. Chan, T.Y. Chen, and D. Towey, "Normalized Restricted Random Testing", (accepted to appear in the *Proceedings of the Eighth International Conference on Reliable Software Technologies*, 2003)

[4] T.Y. Chen, H. Leung, and I.K. Mak, "Adaptive Random Testing", *submitted for publication*.

[5] T.Y. Chen, T.H. Tse, and Y.T. Yu, "Proportional sampling strategies: a compendium and some insights", *The Journal of Systems and Software*, Vol.58, pp.65-81, 2001.

[6] T.Y. Chen and Y.T. Yu, "On the Relationship Between Partition and Random Testing", *IEEE Transactions on Software Engineering*, Vol. 20, No. 12, pp.977-980, 1994.

[7] T.Y. Chen and Y.T. Yu, "On the Expected number of failures detected by subdomain testing and random testing", *IEEE Transactions on Software Engineering*, Vol. 22, No. 2, pp.109-119, 1996.

[8] R. Cobb and H.D. Mills, "Engineering Software under Statistical Quality Control", *IEEE Software* Vol. 7, pp.44-56, 1990.

[9] B.S. Everitt, *The Cambridge Dictionary of Statistics*, Cambridge University Press, 1998.

[10] R. Hamlet, "Random Testing", *Encyclopaedia of Software Engineering*, edited by J. Marciniak, Wiley, pp.970-978, 1994.

[11] H.D. Mills, M. Dyer, and R.C. Linger, "Cleanroom Software Engineering", *IEEE Software*, Vol. 3, pp.19-24, 1986.

[12] G. Myers, *The Art of Software Testing*, New York: John Wiley & Sons, 1979.

[13] R.A. Thayer, M. Lipow, and E.C. Nelson, *Software Reliability,* Amsterdam, The Netherlands: North-Holland, 1978.

[14] L.J. White, "Software Testing and Verification", *Advances in Computers*, Vol. 26, pp.335-391, 1987.

Appendix

Table 1. Effectiveness of FSCS-ART and X2Y1 MART for block failure-causing pattern with failure rate of 0.001

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	873		633		477		601		665	
X2Y1 MART	816	886	635	638	463	484	603	606	606	670

Table 2. Effectiveness of FSCS-ART and X2Y1 MART for strip failure-causing pattern with failure rate of 0.001

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	1476		782		766		743		821	
X2Y1 MART	1502	1408	787	796	778	762	748	757	827	836

Table 3. Effectiveness of FSCS-ART and X2Y1 MART for point failure-causing pattern with failure rate of 0.001

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	1243		927		834		881		974	
X2Y1 MART	1345	1182	938	928	876	813	891	882	984	975

Table 4. Performance of FSCS-ART and MART with failure rate of 0.0001
Failure rate = 0.0001, expected F-measure for random testing = 10000

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	903		6162		4722		5855		6470	
X2Y1 MART	835	859	6293	6377	4638	4767	5979	6058	6608	6696
X3Y1 MART	945	898	6309	6160	4947	4707	5994	5852	6625	6467
X1Y2 MART	910	878	6293	6380	4836	4819	5979	6061	6607	6699
X1Y3 MART	912	1018	6112	6239	4708	5073	5807	5927	6418	6550

Table 5. Performance of FSCS-ART and MART with failure rate of 0.0005
Failure rate = 0.0005, expected F-measure for random testing = 2000

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	878		1231		931		1170		1293	
X2Y1 MART	863	889	1306	1281	977	974	1241	1217	1371	1345
X3Y1 MART	846	928	1239	1324	919	1028	1177	1258	1300	1390
X1Y2 MART	925	879	1238	1294	960	979	1177	1230	1300	1359
X1Y3 MART	923	917	1304	1310	1010	1011	1239	1245	1370	1376

Table 6. Performance of FSCS-ART and MART with failure rate of 0.001
Failure rate = 0.001, expected F-measure for random testing = 1000

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	873		633		477		601		665	
X2Y1 MART	816	886	635	638	463	484	603	606	667	670
X3Y1 MART	960	874	631	653	498	492	600	621	663	686
X1Y2 MART	784	909	626	687	446	528	594	652	657	721
X1Y3 MART	950	904	649	656	510	503	616	623	681	689

Table 7. Performance of FSCS-ART and MART with failure rate of 0.005
Failure rate = 0.005, expected F-measure for random testing = 200

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	852		135		100		128		142	
X2Y1 MART	878	861	126	133	95	99	119	126	132	139
X3Y1 MART	828	872	138	132	101	99	131	125	145	138
X1Y2 MART	909	873	130	137	100	103	124	130	137	144
X1Y3 MART	874	746	134	141	101	98	128	134	141	148

Table 8. Performance of FSCS-ART and MART with failure rate of 0.01
Failure rate = 0.01, expected F-measure for random testing = 100

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	934		68		53		64		71	
X2Y1 MART	836	900	71	67	52	52	67	64	75	71
X3Y1 MART	879	782	70	76	53	54	67	72	74	79
X1Y2 MART	898	805	67	70	51	50	64	66	70	73
X1Y3 MART	793	825	72	76	52	55	68	72	75	80

Table 9. Performance of FSCS-ART and MART with failure rate of 0.05
Failure rate = 0.05, expected F-measure for random testing = 20

	No. of Run		Average F-measure		Std Dev		95% Confidence			
							-		+	
	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map	T-map	R-map
FSCS-ART	682		15		10		15		16	
X2Y1 MART	770	749	16	17	11	12	15	16	17	17
X3Y1 MART	824	838	16	18	12	13	15	17	17	19
X1Y2 MART	775	773	16	16	11	11	15	15	16	16
X1Y3 MART	863	816	15	17	12	13	15	16	16	18