[SUPERCEDES AND REPLACES LOCAL RULE #IV]

# Supreme Court of Florida

No. AOSC07-38

IN RE: JUROR SELECTION PLAN: LEE COUNTY

## ADMINISTRATIVE ORDER

Section 40.225, Florida Statutes, provides for the selection of jurors to serve within the county by "mechanical, electronic, or electrical device." Pursuant to that section, a majority of judges within the circuit in which the county resides must approve, and the chief judge of the circuit must submit, a description of the method for selecting jurors to the Supreme Court of Florida. Section 40.225(3), Florida Statutes, charges the Supreme Court with the review and approval of the proposed juror selection process, hereinafter referred to as the "juror selection plan."

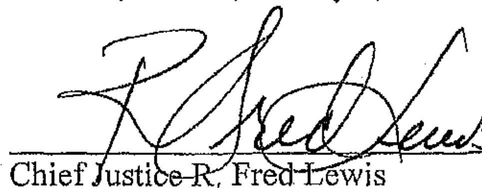
The use of technology in the selection of jurors has been customary within Florida for well over a decade, and the Supreme Court has developed standards necessary to ensure that juror selection plans satisfy statutory, methodological, and due process requirements. The Court has tasked the Office of the State Courts Administrator with evaluating proposed plans to determine their compliance with those standards.

On March 27, 2007, the Chief Judge of the Twentieth Judicial Circuit submitted the Lee County Juror Selection Plan (originally submitted as Twentieth Judicial Circuit Local Rule #IV Amended) for review and approval in accordance with section 40.225(2), Florida Statutes. The proposed plan reflects changes to both hardware and software used for juror pool selection in Lee County.

The Office of the State Courts Administrator has completed an extensive review of the proposed Lee County juror selection plan, including an evaluation of statutory, due process, statistical, and mathematical elements associated with selection of jury candidates. The plan meets established requirements for approval.

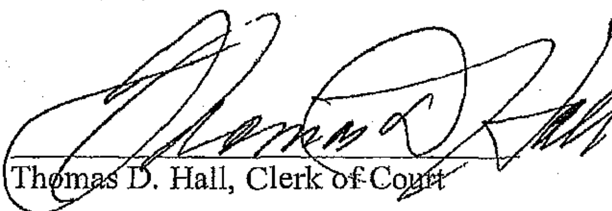
Accordingly, the attached Lee County Juror Selection Plan, submitted by Chief Judge Hugh Hayes on March 27, 2007, is hereby approved for use.

DONE AND ORDERED at Tallahassee, Florida, on July 9, 2007.

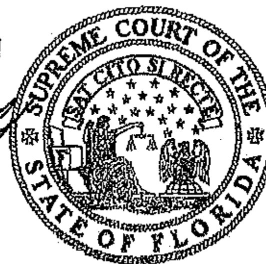


Chief Justice R. Fred Lewis

ATTEST:



Thomas D. Hall, Clerk of Court



**IN THE TWENTIETH JUDICIAL CIRCUIT IN AND FOR THE STATE OF FLORIDA**

**LOCAL RULE #IV**
**- Amended -**

**IN RE: SELECTION OF JURORS BY COMPUTER IN LEE COUNTY**

**WHEREAS,** the present method of selecting jurors can be expedited without additional expense or loss of the sanctity of random selection by the use of an electronic computer available for use by Lee County, and

**WHEREAS,** in accordance with Florida Statute § 40.011, the source of such selection is from the database of names from the Department of Highway Safety and Motor Vehicles, restricted to Lee County, which is in a computer compatible form and in the custody of the Lee County Clerk of the Circuit Court, and

**WHEREAS,** in accordance with Florida Statute § 40.011, the source of selection is also from the list of those whose names do not appear on the Department database, but who have filed with the Clerk of the Circuit Court an affidavit prescribed in the cited statute, it is therefore,

**RESOLVED** that the Rules of the Twentieth Judicial Circuit for procedure in all courts of Lee County in which jury trials are held shall be amended to include this additional Rule adopting the following alternative plan for the selection of persons for grand or petit jury service:

1. **EQUIPMENT**: The equipment used in the jury selection application will be a Dell PowerEdge 2850 Server with 2 Intel Xeon 3.2Ghz processors, 2 GB of memory and 140G of RAID 5 disk space running Windows 2003 Server operating system, located in the secured computer room of the Lee County Clerk of the Circuit Court.

2. **ALTERNATIVE METHOD OF SELECTING VENIRE:**

a. The source from which names shall be taken is the same as that which is described above in accordance with Florida Statute § 40.011. On a quarterly basis, the Lee County Clerk of the Circuit Court shall obtain a computerized listing of names from the Department of

Highway Safety and Motor Vehicles. The Clerk of the Circuit Court will protect the listing and tapes and keep them securely stored.

b. The Clerk of Circuit Court of Lee County is designated the official custodian of the computer records of the lists to be used in jury selection and shall ensure they are not accessible to anyone other than those directly involved in selection of venires, as herein provided. Functions of the Clerk of the Circuit Court may be performed by his deputies.

c. The entire list of drivers license holders, identification card holders, and those who have filed affidavits pursuant to Florida Statute § 40.011 (hereinafter "eligible jurors") may comprise the master jury list from which venires will be selected according to the provisions of paragraph 2(d) below. Alternatively, the Chief Judge or his or her designated representative, with the aid and assistance of the Clerk of the Circuit Court, may select the master jury list for the year by lot and at random from the entire list of eligible jurors using the method described in the attachments hereto.

d. The Clerk of the Circuit Court shall cause jury venires to be selected from the final jury list programmed into the Lee County computer using the method described in Attachment "A" (Method of Jury Selection For Lee County), Attachment "B" (Jury Selection Algorithm For Lee County) and Attachment "C" (JSI Juror Candidate List Processing Methodology), and in accordance with directions received from the Chief Judge or his or her designated representative.

e. The initial jury selection programming may exclude persons who have been previously excluded for physical infirmities or inability to comply with other non-correctable statutory qualifications. A detailed description as to how the list of excluded persons is developed and maintained is set forth in the attachments hereto.

f. Attachments "A", "B", and "C" shall be incorporated into this Local Rule as if fully set forth herein.

IN RE: SELECTION OF JURORS BY COMPUTER IN LEE COUNTY

STATE OF FLORIDA
COUNTY OF LEE

## CERTIFICATE

I HEREBY certify that, pursuant to Florida Statute § 40.225, a majority of judges authorized to conduct jury trials in Lee County, Florida, have consented to the use of the electronic system which is described in an attachment hereto, and requests the approval of the Supreme Court of Florida for the use of such system in Lee County, Florida.

DATED this 27th day of March, 2007.

Hugh D. Hayes
Chief Judge
Twentieth Judicial Circuit

History.- Local Rule #IV (March 20, 1998).

Attachment "A"

METHOD OF JURY SELECTION FOR LEE COUNTY

Lee County juror source list processing:

Candidates for Lee County petit and grand jury venires are drawn from a list of licensed Florida residents as provided by the Florida Department of Highway Safety and Motor Vehicles (i.e., drivers list) combined with a list of all residents indicating a desire to serve as a juror (i.e., affidavit list) in accordance with F.S. 40.011.

The drivers list is drawn quarterly by the Clerk of the Circuit Court for Lee County from the Department of Highway Safety and Motor Vehicles (DHSMV) via the Florida Association of Court Clerks (FACC) (see notes 1 and 2) and is maintained, by the Clerk, in accordance with F.S. 40.011 and 40.022.

The Clerk of the Circuit Court shall purge the final jury selection list of those persons statutorily excluded from juror service as specified in F.S. 40.013 and 40.022. Additionally, the Clerk of the Circuit Court shall, at least once a month, purge the jury selection lists of names of those persons:

1. Adjudicated mentally incompetent;
2. Convicted of a felony; or
3. Deceased.
4. Other persons as the Chief Judge directs.

All persons so excluded are compiled in a suppression list and provided to JSI to aid in the production of the final candidate list.

At least once a year, the Clerk of the Circuit Court for Lee County will provide the driver's list, the affidavit list and the suppression list, a minimum number of juror candidates required (not less that 250) and a set of seed values to Jury Systems Incorporated (JSI) who will perform the actual selection of candidate jurors. A detailed description of the selection process as performed by JSI is provided in Attachment "C". A description of the random number generator used in the selection process is provided in Attachment "B".

The seed values provided are used by JSI to initialize the random number generator which forms the basis of the mechanism by which juror candidates are selected "...by lot and at random ..." as per F.S. 40.225. The Clerk of the Circuit Court will provide JSI with two seed values such that the first seed value will be between 0 and 31,328 and the second value will be between 0 and 30,081.

These values are selected and calculated as follows:

Initial values will be drawn from the top ten most active stock index as published in The News-Press Business & Money Section on the Wednesday prior to the request to JSI. Starting with the first stock on the list, the value in the thousands digit of the trading volume will be used to create

the first seed value. Values will be read sequentially down the list (including leading zeros) until a number in the range of 0 to 99,999 is read. The second seed value will then be read from the next set of five active stock indexes exactly as the first. A copy of this page will be retained. For example, the following list would provide two seed values of 02604 and 45893

| Symbol | Volume | |
|---|---|---|
| F | 11,740,800 | 0 |
| LU | 8,592,800 | 2 |
| PFE | 7,576,400 | 6 |
| EMC | 7,190,500 | 0 |
| NT | 6,764,800 | 4 |
| MOT | 5,674,400 | 4 |
| GE | 5,525,900 | 5 |
| T | 5,108,500 | 8 |
| EWH | 4,969,200 | 9 |
| EWJ | 4,553,100 | 3 |

These seed values will be converted to the required range using the formula

$$(\frac{initial_value}{range_base} * (upper - lower + 1)) + lower$$

where intial_value is determined from the stock list as above, the range base is 100,000 (from the range 0 – 99,999), upper is the upper allowable limit of the seed value to be calculated and lower is the lower allowable limit of the seed value. All division is floating point division and the result is truncated to the next lower integer.

For example, using the initial_value's selected above,

$$seed_1 = (\frac{02604}{100000} * (31328 - 0 + 1)) + 0 = 815.807 = 815$$

<u>and</u>

$$seed_2 = (\frac{45893}{100000} * (30081 - 0 + 1)) + 0 = 13805.532 = 13805$$

Juror candidates will be notified sequentially beginning with the first name on the final candidate list provided by JSI. The Clerk of the Circuit Court will select the number of names from the list required to fill current need in each notification cycle. The number selected will be the total number of jurors required minus any persons whose service comes due for this period as a result of a previously court approved postponement. These individuals are tracked separately. Subsequent juror call ups will begin with the first name following the last named used in the previous notification cycle.

If the final juror candidate list is exhausted before the next full candidate selection cycle, an additional list of names may be requested from JSI without repeating the source list processing as the JSI process randomizes the entire name list of eligible persons in Lee County regardless of the actual number of names requested by the Clerk of the Circuit Court. Thus, JSI has an additional pool of names over and above the number requested by the Clerk of the Circuit Court already processed and available.

To ensure the integrity and verifiability of the selection process, the Clerk of the Circuit Court will retain a copy of the full final juror candidate name list as developed by JSI along with the number of candidate names requested in each cycle and the initialization values provided to JSI in a restricted manner as per F.S. 40.02(2) and F.S. 40.221.

The Clerk of the Circuit Court will purge, monthly, the final juror candidate lists of those persons recently identified as felons by the Florida Department of Law Enforcement as per F.S. 40.022(4).

Notes:

1. The Florida Association of Court Clerks is merely a "pass-through" used by the Department of Highway Safety and Motor Vehicles to distribute the licensed driver file. The Florida Association of Court Clerks performs no manipulation, filtration, purging, etc. to the file received from the Department of Highway Safety and Motor Vehicles. The Department of Highway Safety and Motor Vehicles purges those on the list under the age of 18 and filters out all non-Lee County residents prior to forwarding the list to the Florida Association of Court Clerks.
2. All name lists are provided to JSI as ASCII text in no particular sequence or order
3. Jury Systems, Inc. does all of the processing of the data in-house using their own employees and equipment. The only exception to this is the optional NCOA (National Change of Address) process which is outsourced to an NCOA provider. If the NCOA option is requested, the file is then sent to an outsourced NCOA provider. The list is then returned to Jury Systems, Inc. and ready to be loaded into the Jury Plus System. Only the purged and most accurate source list is loaded into the Jury Plus System.

Attachment "C"

# JSI JUROR CANDIDATE LIST PROCESSING METHODOLOGY

The following are the steps that comprise JSI's Juror Source List process in Florida:

1. **CREATE EXTRACT FILE AND SUPPRESSION FILE**
   The Suppression File is created from JURY+ historical data. The Suppression File consists of records of jurors that have been permanently excused, temporarily excused, and exempt for other reasons according to client requirements.

   The Extract File is created from JURY+ Next Generation database. The Extract File consists of records of all jurors that have been loaded. From the Extract file, a Suppression file is created using Clients suppression requirements.

2. **CONVERT DRIVERS FILE TO STANDARD FORMAT**
   Every file processed must be in the *JURY+©* standard format (see attached document entitled CDF-Candidate Data File) so that the software understands where, within a record, the various data fields are located.

3. **CONVERT DECEASED FILE TO STANDARD FORMAT**
   Use the same method described in step 2.

4. **CREATE A MATCH KEY FOR EACH DRIVERS AND EXTRACT RECORD**
   The Match Key consists of the Drivers-ID

5. **SORT BOTH THE DRIVERS FILE AND EXTRACT FILE BY MATCH KEY**

6. **UPDATE DRIVERS FILE FROM EXTRACT FILE BY MATCH KEY**
   Every Drivers File record is compared to every Extract File record by Drivers-ID. When a match is found, the Voters-ID, SSN(Social Security Number), and PID(Person-ID) fields are checked for data. The fields that are without data will be replaced with their respective fields from the Extract File.

7. **CREATE A MATCH KEY FOR EACH DRIVERS AND EXTRACT RECORD**
   The Match Key consists of the Full Name, Street Address and Date of Birth

8. **SORT BOTH DRIVERS FILE AND EXTRACT FILE BY MATCH KEY**

9. **UPDATE DRIVERS FILE FROM EXTRACT FILE BY MATCH KEY**
   The programs consider two records with the same Name and Address to be duplicates only if:
   - The two birth dates are equal
   - If either birth date is blank or zeroes

Every Drivers File record is compared to every Extract File record by Match Key. When a match is found, the Drivers ID, SSN, and PID fields are checked for data. The fields that are without data will be replaced with their respective fields from the Extract File.
The Match Key build, sort, and update process are repeated using the Residential Address on both Files. If there is no Residential Address on any given record, the Mailing Address is used again.

10. **CREATE A MATCH KEY FOR EACH DRIVERS AND EXTRACT RECORD**
The Match Key consists of the Person-ID (PID)

11. **SORT BOTH THE DRIVERS FILE AND THE SUPPRESSION FILE BY MATCH KEY**

12. **REMOVE INTERNAL DUPLICATES WITHIN DRIVERS FILE BY MATCH KEY**
If the Person-ID of one record matches that of the next record on the file, the first record is discarded.

13. **SUPPRESSION MATCH BETWEEN THE DRIVERS AND SUPPRESSION FILES BY MATCH KEY**
Every Drivers File record is compared to every Suppression File record by PID. Drivers File records that are not on the Suppression File are written to the *new* Drivers File.

14. **CREATE A MATCH KEY FOR EACH DRIVERS AND SUPPRESSION RECORD**
The Match Key consists of the Full Name, Street Address and Date of Birth

15. **SORT BOTH THE SUPPRESSION AND DRIVERS FILE BY MATCH KEY**

16. **SUPPRESSION MATCH BETWEEN THE DRIVERS AND SUPPRESSION FILES BY MATCH KEY**
The programs consider two records with the same Name and Address to be duplicates only if:
- The two birth dates are equal
- If either birth date is blank or zeroes

Every Drivers File record is compared to every Suppression File record by Match Key. Drivers File records that are not on the Suppression File are written to the output Juror Load File.

The Match Key build, sort, and matching process are repeated using the Residential Address on the Suppression File and the Juror Load File. If there is no Residential Address on any given record, the Mailing Address is used again. This will eliminate duplicates where one file has a PO Box in the mailing address and the residential address in that field and the other file has the residential address in the mailing address field. A report of the eliminated duplicates is provided.

17. **ELIMINATE INTERNAL DUPLICATES ON THE JUROR LOAD FILE**

This process determines if there are any duplicates among the Juror Load records. If the Match Key of one record matches that of the next record on the file, the first record is discarded.

The Match Key build, sort, and matching process are repeated using the Residential Address on the Suppression File and the Juror Load File. If there is no Residential Address on any given record, the Mailing Address is used again. This will eliminate duplicates where one file has a PO Box in the mailing address and the residential address in that field and the other file has the residential address in the mailing address field. A report of the eliminated duplicates is provided.

**18. ELIMINATE JUROR LOAD RECORDS WITH INVALID ZIP CODES**

Each Juror Load record is evaluated as follows:

a) If the Zip Code of the Mailing Address **is blank** the record is rejected

b) If the Zip Code of the Residence Address **is not found** in the County's list of valid zip codes, the record is rejected.

c) If the Zip Code of the Residence Address **is found** in the County's list of valid zip codes, the record is then evaluated as follows:

- If the Zip Code of the Mailing Address **is found** in the County's list of valid zip codes the record is kept.
- If the Zip Code of the Mailing Address **is not found** in the County's list of valid zip codes, and the client has told us to reject out-of-County mailing addresses, the record is rejected – otherwise the record is kept.

d) If the Zip Code of the Residence Address **is blank**, the Mailing Address is evaluated as follows:

- If the Zip Code of the Mailing Address **is found** in the County's list of valid zip codes the record is kept.

**19. CREATE RANDOM NUMBERS AND JUROR IDS**

'N' number of random numbers are created where 'n' is sufficient in relation to the number of records on the Juror Load File. Each seven-digit random number is then prefixed with a two-digit value that represents the year (i.e. '24' is 2004 and '25' is 2005). See Attachment B, document entitled "JURY SELECTION ALGORITHM FOR LEE COUNTY."

**20. APPLY RANDOM NUMBERS TO RECORDS IN THE JUROR LOAD FILE**

Each record in the Juror Load File is assigned one of the random nine-digit Juror Identification Numbers.

**21. SORT THE JUROR LOAD FILE BY JUROR ID (JID)**

**22. CREATE SEQUENTIAL NUMBERS AND JUROR IDS**

As more and more clients choose to load jurors more than once per year it has become necessary to load randomly sorted jurors with a sequential JID because when random numbers are generated twice in the same year there is no guarantee of uniqueness between the occasions. That is, the unique JID 190001203 generated in January of 1999 has a chance of

being generated again in July of the same year. If 190001203 is generated and assigned to a juror in July and a record with that JID as its key is loaded into the JURY+ database, it will be rejected as a duplicate. Thus, once the Juror File is randomized using the random JID, a new sequential JID is applied to each record. In January the JIDs might run from 190000001 through 190050000 for 50,000 jurors. In July the starting JID would be 190050001 and for another 50,000 records would go through 190100000.

**23. APPLY SEQUENTIAL NUMBERS TO RECORDS IN THE JUROR LOAD FILE**

**24. SELECT THE REQUESTED NUMBER OF RECORDS**

***If the Juror Load File contains 132,428 records but the client only wants 50,000, the first 50,000 are selected and written to the final output Juror Load File. The balance of the Juror Load File is kept in the data vault at Jury Systems Incorporated. Due the low cost of hard disk space these days, many clients have opted to load their entire Juror Load File into JURY+.***

**25. SEED VALUES**

There are two seed values that are typed in as parameter values. They are a maximum of 5 digits a piece. Different values are passed than that of the previous loads' seed values. Jury Systems, Inc. arbitrarily creates the values.

**26. VERIFICATION**

Jury Systems, Inc will take the names of the potential jurors that have been loaded into the NextGen application and download them onto a Compact Disc. Jury Systems, Inc. will then write the seed values used in the process on the outside of the Compact Disc and forward them to the Lee County Clerk of the Circuit Court.

*Sample Statistics*

Client ID: **DVL** Name: **Duval, FL** Period: **Load *2004***
Episode: **100632**

| **Raw Files Conv** | **# of recs** | **# good** | **# bad**** | |
|---|---|---|---|---|
| (primary) Drivers: | | | | Note: Voters and Drivers fields are filled in only when we are performing a Merge/Purge, otherwise use the OTHER fields!!! |
| Voters: | | | | |
| Other: | 593,955 | 593,955 | | |
| *Comments:* | | | | |

| **Input Files** | **# of recs** | **# of DMV** | **# of ROV** | **# of other** | **# miss src** | **# of inval** | **Data Conv.** |
|---|---|---|---|---|---|---|---|
| Drivers: | | | | | | | |
| Voters: | | | | | | | |
| Other: | 593,955 | 593,955 | | | | | 593,955 |
| Suppression: | 122,826 | 101,517 | 20,786 | 523 | | | 122,826 |
| *Comments:* | NGSupp=11,659; ClassicSupp=110,684; Dead File=483 | | | | | | |

| **Special Dup Elim** | **# of recs** | **# of DMV** | **# of ROV** | **# of other** | **# miss src** | **# of inval** | |
|---|---|---|---|---|---|---|---|
| Recs suppressed: | | | | | | | |
| Output: | | | | | | | |
| *N/Aor Comments:* | N/A | | | | | | |

| **Merge/Purge** | **# of recs** | **# of DMV** | **# of ROV** | **# of other** | **# miss src** | **# of inval** | |
|---|---|---|---|---|---|---|---|
| Drivers: | | | | | | | |
| Voters: | | | | | | | |
| Other: | | | | | | | |
| Mail Dups found: | | | | | | | |
| Resi Dups found: | | | | | | | |
| **Voters/Other Out:** | | | | | | | |
| Merged Master: | | | | | | | |
| *N/Aor Comments:* | N/A | | | | | | |

| **PersonID Dup Elim** | **# of recs** | **# of DMV** | **# of ROV** | **# of other** | **# miss src** | **# of inval** | |
|---|---|---|---|---|---|---|---|
| Internal Dup found: | 5 | 5 | | | | | |
| Recs suppressed: | 8,947 | 8,947 | | | | | |
| Recs passed: | 585,003 | 585,003 | | | | | |

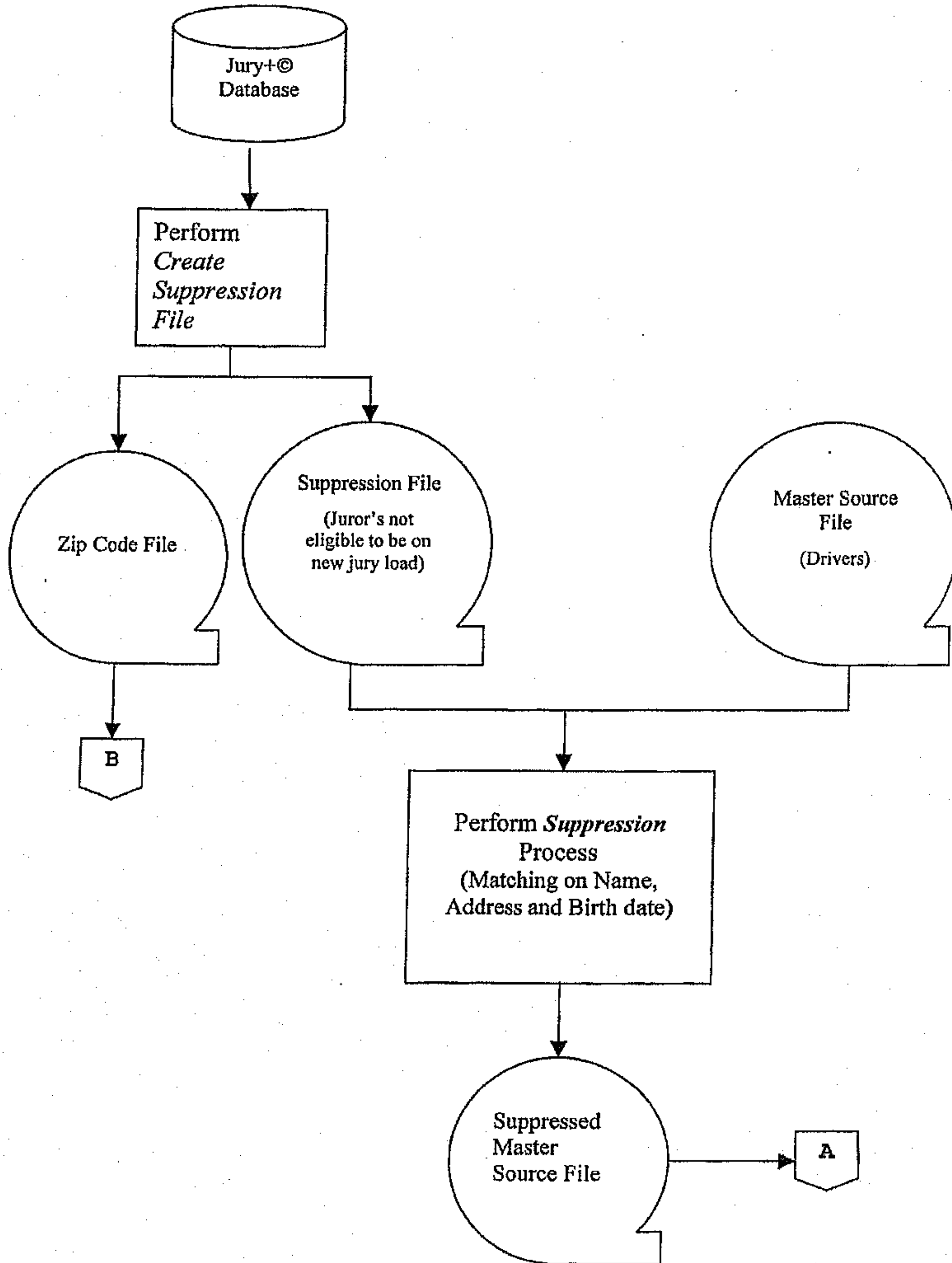
| **Special Dup Elim** | **# of recs** | **# of DMV** | **# of ROV** | **# of other** | **# miss src** | **# of inval** | |
|---|---|---|---|---|---|---|---|
| Recs suppressed: | 55,718 | 55,718 | | | | | |
| Output: | 529,285 | 529,285 | | | | | |
| *N/Aor Comments:* | Matching by SSN | | | | | | |

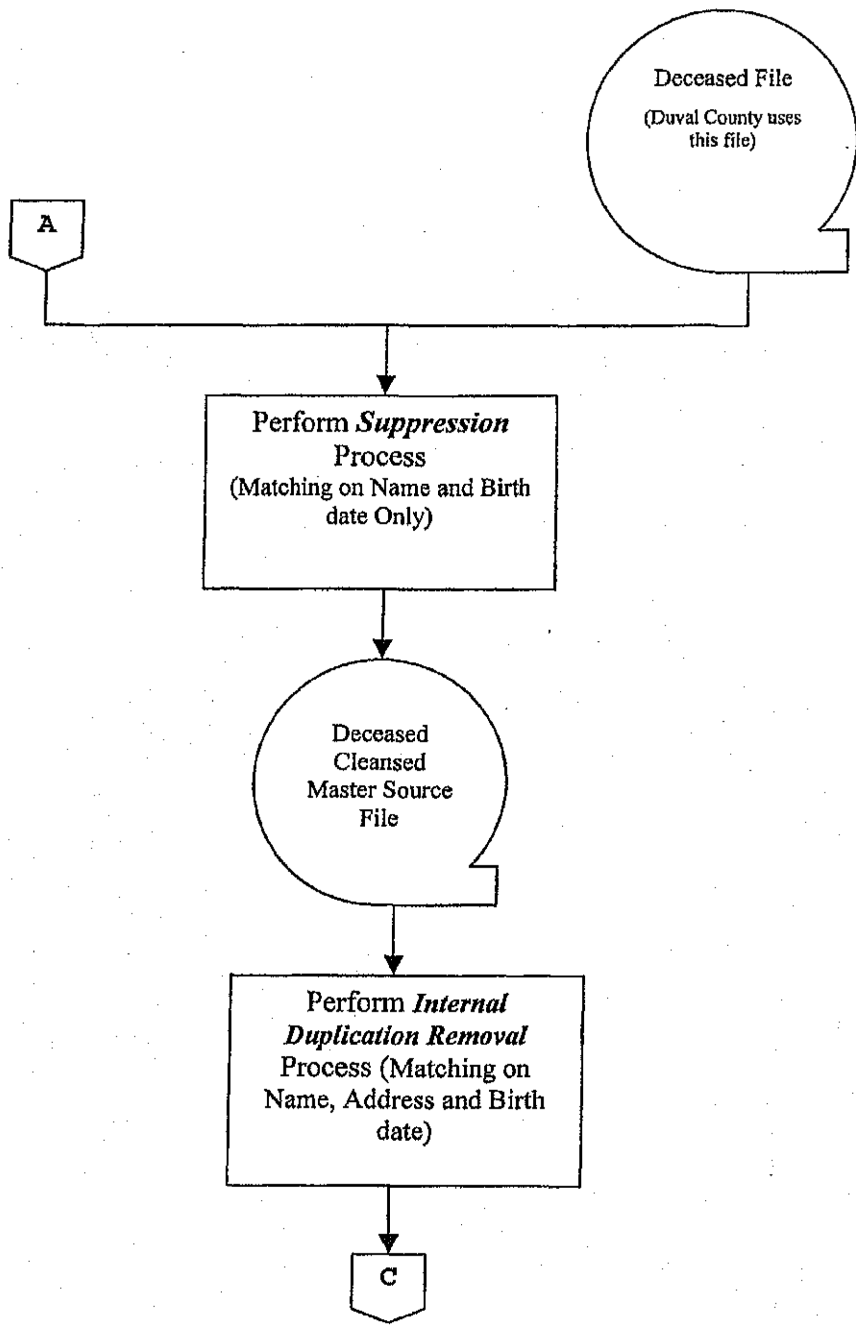
| Supression Stats | # of recs | # of DMV | # of ROV | # of other | # miss src | # of inval | |
|---|---|---|---|---|---|---|---|
| Mail Recs supp: | 6,279 | 6,279 | | | | | |
| Mail Recs passed: | 523,006 | 523,006 | | | | | |
| Resi Recs supp: | 68 | 68 | | | | | |
| Resi Recs passed: | 522,938 | 522,938 | | | | | |

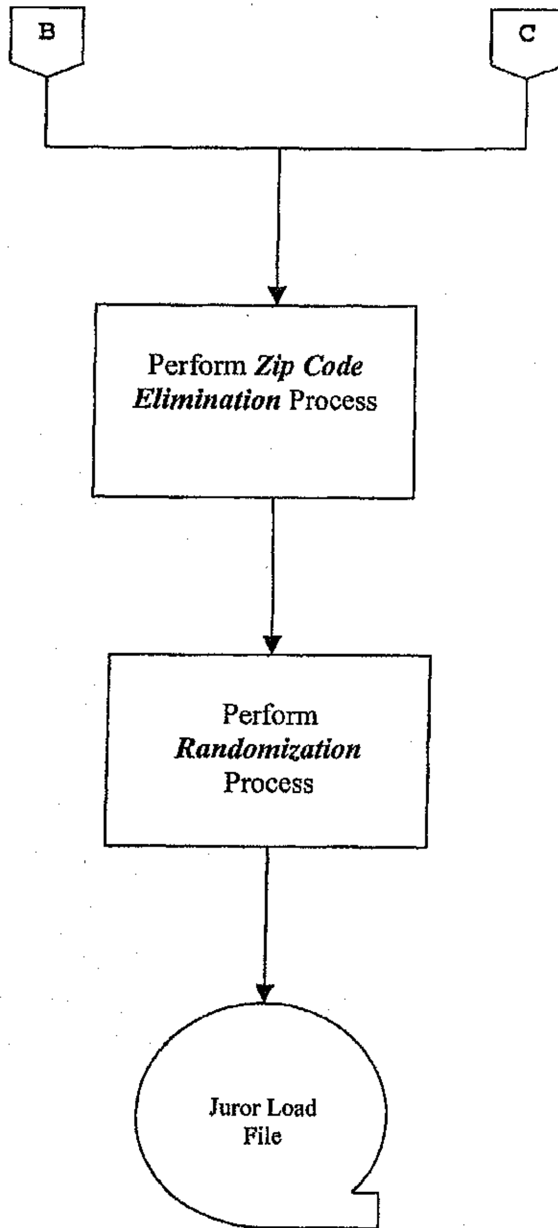
| Cleansing Stats | # of recs | # of DMV | # of ROV | # of other | # miss src | # of inval | |
|---|---|---|---|---|---|---|---|
| Resi Int Dup found: | 86 | 86 | | | | | |
| Mail Int Dup found: | | | | | | | |
| Zip Code process: | 1,190 | 1,190 | | | | | |
| Cleansed Master: | 521,662 | 521,662 | | | | | |
| *N/A* or Split for NCOA: | | | | | | | N/A |
| Balance of Split: | | | | | | | |
| *Comments:* | | | | | | | |

| Load Results | # of recs | # of DMV | # of ROV | # of other | # miss src | # of inval | |
|---|---|---|---|---|---|---|---|
| Total Loaded: | 80,000 | 80,000 | | | | | |
| Bad Zips: | | | | | | | |
| No First or Last Nm: | | | | | | | |
| Balance File: | 441,662 | 441,662 | | | | | |
| *Date:* | 3/29/04 | | **Source File Total (ROV/DMV/OTHER) =** | | | | 593,955 |

**Juror Source List Processing Flow**




Deceased File
(Duval County uses this file)
A
Perform *Suppression* Process
(Matching on Name and Birth date Only)
Deceased Cleansed Master Source File
Perform *Internal Duplication Removal* Process (Matching on Name, Address and Birth date)
C


B
C
Perform Zip Code Elimination Process
Perform Randomization Process
Juror Load File

Attachment "B"

JURY SELECTION ALGORITHM FOR LEE COUNTY

JURY SYSTEMS
INCORPORATED

# JURY+ *Next Generation*

# Universal Random Generator

# Detailed Design & Functionality

## Notice

Techniques contained in this document are considered proprietary to Jury Systems Incorporated. They may not be revealed or released to any party without the express written consent of Jury Systems Incorporated.

This material may not be copied or reproduced in any form without the express written permission of Jury Systems Incorporated.

©October 2006

## CONTENTS

1. Introduction ........ 16

2. The Definition of the Universal Random Number Generator ........ 17

3. Development of the Universal Random Number Generator ........ 18

4. JURY+ use of the Universal Random Number Generator ........ 19

5. Logic Specifications for the Universal Random Number Generator ........ 20

6. Validation of the Universal Random Number Generator ........ 22

I. APPENDIX A - The Universal Random Number Generator Program (C-Language Version) ........ 23

II. APPENDIX B - The JSI Implementation of the Universal Random Number Generator (COBOL-Language Version) ........ 27

III. APPENDIX C – Toward a Universal Random Number Generator By George Marsaglia and Arif Saman ........ 34

## 1. Introduction

This document describes the theory and structure of the random number generator that is used by the JURY+ Jury Management System to perform those jury management business functions that require randomization.

The random number generator employed by the JURY+ software is the "Universal" generator which appeared in an article written by George Marsaglia and Arif Zaman who are part of the "Supercomputer Computations Research Institute and Department of Statistics" at The Florida State University, Tallahassee. Also contributing to the article was Wai Wan Tsang a member of the "Department of Computer Science" at the University of Hong Kong.

The article (titled: "Toward a Universal Random Number Generator" is included in its entirety as an appendix to this document.

## 2. The Definition of the Universal Random Number Generator

The Universal generator algorithm is a combination of a Fibonacci sequence (with lags of 97 and 33, and operation "subtraction plus one, modulo one") and an "arithmetic sequence" (using subtraction).

It passes ALL of the tests for random number generators and has a period of $2^{144}$ and is completely portable (gives bit identical results on all machines with at least 24-bit mantissas in the floating point representation).

The Universal random number generator employed by Jury Systems Incorporated in its JURY+ application software is a true, exact implementation of the algorithm defined in "Toward a Universal Random Number Generator" and thus all randomness tests for that process published in statistical literature applies to the JURY+ implementation.

## 3. Development of the Universal Random Number Generator

In June 2006, the Florida State AOC required that all randomization for purposes of jury selection be accomplished using the Universal Random Number generator described in an article titled "Towards a Universal Random Number Generator" by George Marsaglia. The Universal Generator is a combination generator. It combines two different generators, the first of which takes two user seed values, converts them into 4 seed values and generates a sequence of 97 random numbers. These number become "seed" values for the second random generator which uses them in a combination process to combine the series of random numbers, producing a "Universal" value.

Previously, Jury Systems Incorporated used the "Marsaglia" random number generator which is a feedback shift register (FSR) method to generate uniform random numbers between 0 and 1, inclusive. The method was named for and based upon the idea of George Marsaglia (1965) who developed a coupled random number generator called super duper. Super duper couples a multiplicative-congruential generator with an FSR generator. This generator was subjected to extensive testing by Rand Laboratories and shown to pass all randomness tests for all sample sizes likely to be encountered in the JURY selection process. This routine is a 2-seed routine, in that each number in a random sequence is provided based on two seed values.

Using the referenced article and a published C-language implementation of the Universal random generator (both of which are included as an appendix to this document), Jury Systems Incorporated created a version of the routine for integration into its JURY+ application for use in Florida and any other site that may desire it. The JSI implementation was done in August of 2006 and is a COBOL version of the routine. A copy of the JSI implementation is also included in an appendix.

## 4. JURY+ use of the Universal Random Number Generator

Wherever randomness is requisite in the JURY+ application, the Universal generator is employed. Those application functionalities include the following:

- Source List Processing
  When source lists are processed to supply juror names to JURY+, each member of the list is assigned a random number. The list is then sorted by the random number (known as a Juror Identification Number - JID) and the first 'n' records are selected per client requirements.

- Juror Summonsing
  When it is necessary to summon jurors to a specific court and date, the full set of eligible jurors is assigned a random number. The list is sorted and the first 'n' number of jurors are selected.

- Panel Selection
  When requests for juror panels are received at the assembly room, the user initiates a computer program to create a panel of the requested size. The computer program provides a randomly ordered list of jurors available for service at that moment.

  Each juror in the pool is assigned a random number. Once all jurors have thus been assigned a temporary unique number, the list is ordered by that number. Once the panel jurors are selected from this list, the panel jurors are re-randomized and a Case Information Sheet listing them is produced assuring that each juror has an equal opportunity to be the first seated for voir dire.

- Reporting
  Many of the JURY+ reports allow the user to select list of jurors that re ordered randomly.

## 5. Logic Specifications for the Universal Random Number Generator

The Universal Generator is a combination generator in that it combines two different random generators to provide a random series that passes every randomness test. The principal component of the two has a very long period, about $10^{36}$. It is a lagged-Fibonacci generator based on the binary operation x times y on reals x and y.

The Fibonacci generator has an extremely long period and appears to be suitably random based on results of stringent tests that were applied to it. However, there is one test which it fails: the "birthday-spacings test. In order to get a generator that passes all of the stringent tests the first generator is combined with a second generator.

The choice of the second generator is a simple arithmetic sequence for the prime modulus $2^{24}-3 = 16777213$.

Detailed information regarding the theory behind the Universal Random number generator is provided in the published article included as appendix C of this document. The article provides a Fortran language version of the algorithm.

Sometime after the original article appeared, the Fortran program was converted into a "C" programming language implementation and published. The "C" version is included as an appendix to this document.

For implementation into JURY+, JSI developed a COBOL language implementation of the Universal generator by duplicating the logic published in the "C" program. The JSI version is also included in an Appendix to this document.

To the greatest extent possible, the variable names used in the JSI version directly correspond to identically named variables in the "C" implementation. This makes the comparison of the two sets of logic much more straight forward.

A review of the "COBOL" version shows that there are two entry points (distinct processes) in the Universal Algorithm. The first entry point is a routine "1000-set-seeds" (this corresponds to the subroutine called "RMARIN" in the "C" version.

The 1000-set-seeds routine implements the "first" random generator in the Universal process. Using two seed values supplied by the user, it creates four seed values and uses them to create a Fibonacci sequence of 97 random numbers. This series of random numbers is used in the creation of the Universal random number.

Additionally this routine creates a representation of a second sequence (initially set to 362436/16777216 and referenced by variables "C", "CD", "CM"). The Fibonacci series and this series are combined (in the random number generation routine below) to create a Universal Random number.

The second routine is 2000-Gen-Rand (this corresponds to the "RANMAR" routine in the "C" program). This routine generates a Universal Random number by combining the two sequences (series) set up in the 1000-set-seeds routine.

First, two entries from the Fibonacci series (referenced with variables "I" and "J") are subtracted from each other, (the first time a Universal random number is requested the two entries referenced are 97 and 33 respectively) giving the basis for our Universal random number.

After the basis calculated, it replaces the Fibonacci number referenced by "I" (in preparation for the next time a universal number is needed). Then, both references ("I" and "J") are decremented. When either of the reference indicators ("I" or "J") reach zero, they are reset to their initial value (97 and 33 respectively). Thus the series of 97 numbers is processed in a circular fashion. (All of this is in preparation for the next request for a Universal random number).

Finally, the next number in the second series (which was initialized in 1000-set-seeds and are represented by variables "C", "CD", and "CM") is computed and combined with the basis random number (via subtraction). The result is returned to the calling program as a "Universal" random number.

## 6. Validation of the Universal Random Number Generator

As indicated in the "Towards a Universal Random Number", the statistical "randomness" of this routine has been thoroughly tested and documented. It is also clearly explained that an appropriately coded algorithm, regardless of the language it is written in or the computer it is executed on, produces exactly the same "random" sequence when given identical seed values.

Thus, the validation (and thus proof of randomness) becomes one of showing that two different implementation produce the same known results when given appropriate seed values.

The JSI implementation produces the same results as the program on which it was modeled. The "C" version of the program indicates the following test to insure a properly functioning Universal random number generator algorithm:

```
Use IJ = 1802 & KL = 9373 to test the random number generator. The
 subroutine RANMAR should be used to generate 20000 random numbers.
 Then display the next six random numbers generated multiplied by
4096*4096
 If the random number generator is working properly, the random numbers
 should be:
           6533892.0  14220222.0   7275067.0
           6172232.0   8354498.0  10633180.0
```

These are exactly the results produced by calling the JSIRAND1 routine with seed value 1082 and 9373 and viewing the 20001 through 20006th random numbers..

# I. APPENDIX A - The Universal Random Number Generator Program (C-Language Version)

```
/************************************************************************
/************************************************************************
 This random number generator originally appeared in "Toward a Universal
 Random Number Generator" by George Marsaglia and Arif Zaman.
 Florida State University Report: FSU-SCRI-87-50 (1987)

 It was later modified by F. James and published in "A Review of Pseudo-
 random Number Generators"

 Converted from FORTRAN to C by Phil Lintell, James F. Hickling
 Management Consultants Ltd, Aug. 14, 1989.

 THIS IS THE BEST KNOWN RANDOM NUMBER GENERATOR AVAILABLE.
       (However, a newly discovered technique can yield
         a period of 10^600. But that is still in the development stage.)

 It passes ALL of the tests for random number generators and has a period
   of 2^144, is completely portable (gives bit identical results on all
   machines with at least 24-bit mantissas in the floating point
   representation).

 The algorithm is a combination of a Fibonacci sequence (with lags of 97
   and 33, and operation "subtraction plus one, modulo one") and an
   "arithmetic sequence" (using subtraction).

 On a Vax 11/780, this random number generator can produce a number in
    13 microseconds.
************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define TRUE    1
#define FALSE   0

float u[97], c, cd, cm;
int i97, j97, test;

int rmarin(int ij, int kl);
int ranmar(float rvec[], int len);

/************************************************************************
 This is the initialization routine for the random number generator RANMAR()
 NOTE: The seed variables can have values between:    0 <= IJ <= 31328
                                                      0 <= KL <= 30081
 The random number sequences created by these two seeds are of sufficient
 length to complete an entire calculation with. For example, if several
 different groups are working on different parts of the same calculation,
 each group could be assigned its own IJ seed. This would leave each group
 with 30000 choices for the second seed. That is to say, this random
 number generator can create 900 million different subsequences -- with
 each subsequence having a length of approximately 10^30.
```

```
 Use IJ = 1802 & KL = 9373 to test the random number generator. The
 subroutine RANMAR should be used to generate 20000 random numbers.
 Then display the next six random numbers generated multiplied by 4096*4096
 If the random number generator is working properly, the random numbers
 should be:
           6533892.0  14220222.0   7275067.0
           6172232.0   8354498.0  10633180.0
***********************************************************************/

int rmarin(int ij, int kl)
{

        float s, t;
        int i, j, k, l, m;
        int ii, jj;

        /* Change FALSE to TRUE in the next statement to test the
           random routine.*/

        test = TRUE;
        IF ( ( IJ < 0 || IJ > 31328 ) ||
                ( KL < 0 || KL > 30081 ) )
        {
                printf ("RMARIN: The first random number seed must have a "
                         "value between 0 and 31328\n");
                printf ("        The second random number seed must have a "
                         "value between 0 and 30081");
                return 1;
        }

        i = (int)fmod(ij/177.0, 177.0) + 2;
        j = (int)fmod((double)ij, 177.0) + 2;
        k = (int)fmod(kl/169.0, 178.0) + 1;
        l = (int)fmod((double)kl, 169.0);

        for ( ii=0; ii<=96; ii++ )
        {
                s = (float)0.0;
                t = (float)0.5;
                for ( jj=0; jj<=23; jj++ )
                {
                        m = (int)fmod( fmod((double)i*j,179.0)*k , 179.0 );
                        i = j;
                        j = k;
                        k = m;
                        l = (int)fmod( 53.0*l+1.0 , 169.0 );
                        if ( fmod((double)l*m,64.0) >= 32)
                                s = s + t;
                        t = (float)(0.5 * t);
                }
                u[ii] = s;
        }
```

```
        c  = (float)(  362436.0 / 16777216.0);
        cd = (float)( 7654321.0 / 16777216.0);
        cm = (float)(16777213.0 / 16777216.0);

        i97 = 96;
        j97 = 32;

        test = TRUE;

        return 0;
}


int ranmar(float rvec[], int len)
{
        float uni;
        int ivec;

        if ( !test )
                printf ("RANMAR: Call the initialization routine (RMARIN) "
                        "before calling RANMAR.\n");
                return 1;
        }
        for ( ivec=0; ivec < len; ivec++)
        {
                uni = u[i97] - u[j97];
                if ( uni < 0.0F )
                        uni = uni + 1.0;
                u[i97] = uni;
                i97--;
                if ( i97 < 0 )
                        i97 = 96;
                j97--;
                if ( j97 < 0 )
                        j97 = 96;
                c = c - cd;
                if ( c < 0.0F )
                        c = c + cm;
                uni = uni - c;
                if ( uni < 0.0F )
                        uni = uni + 1.0;
                rvec[ivec] = uni;
        }
        return 0;
}
```

## II. APPENDIX B - The JSI Implementation of the Universal Random Number Generator (COBOL-Language Version)

```
 2 IDENTIFICATION DIVISION.
 3 PROGRAM-ID.     JSIRAND1.
 4 AUTHOR.         Jury Systems Incorporated.
 5 DATE-WRITTEN.   June 2006.
 6 DATE-COMPILED.  10-Nov-06 06:53.
 8*****************************************************************
 9*(c)*************************************************************
10*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
11*ÝÝÝÝÝÝÝÝ         ÝÝÝ   ÝÝÝ   ÝÝÝ        ÝÝÝ   ÝÝÝ   ÝÝÝÝÝÝÝÝ[R]Ý*
12*ÝÝÝÝÝÝÝÝÝÝÝ   ÝÝÝÝÝ   ÝÝÝ   ÝÝÝ   ÝÝÝ   ÝÝÝ   Ý   ÝÝÝÝÝ   ÝÝÝÝÝÝÝ*
13*ÝÝÝÝÝÝÝÝÝÝ   ÝÝÝÝÝ   ÝÝÝ   ÝÝÝ   ÝÝ   ÝÝÝÝÝ     ÝÝÝÝÝÝÝ   ÝÝÝÝÝÝÝ*
14*ÝÝÝ   ÝÝÝ   ÝÝÝÝÝ   ÝÝÝ   ÝÝÝ      ÝÝÝÝÝÝÝ   ÝÝÝÝÝ          ÝÝÝ*
15*ÝÝÝ   ÝÝ   ÝÝÝÝÝÝ   ÝÝ   ÝÝÝ   ÝÝ   ÝÝÝÝÝÝ   ÝÝÝÝÝÝÝÝÝÝ   ÝÝÝÝÝÝÝ*
16*ÝÝÝÝ      ÝÝÝÝÝÝÝÝ      ÝÝÝ   ÝÝÝ   ÝÝÝÝÝ   ÝÝÝÝÝÝÝÝÝÝÝ   ÝÝÝÝÝÝÝ*
17*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
18*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
19*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ N E X T   G E N E R A T I O N ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
20*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
21*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝ*
22*ÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝÝFrom Jury Systems IncorporatedÝ*
23*****************************************************************
24*****************************************************************
25**  COPYRIGHT (C) 1998,      by Jury Systems Incorporated        **
26**  All rights reserved.  An unpublished work.                   **
27**                                                               **
28**  TRADE SECRETS NOTICE: ALL RIGHTS RESERVED.  This material    **
29**  contains the valuable properties and trade secrets of Jury   **
30**  Systems Incorporated, a California corporation, embodying    **
31**  substantial creative efforts and confidential information,   **
32**  ideas and expressions, no part of which may be reproduced    **
33**  or transmitted in any form or by any means, electronic,      **
34**  mechanical, or otherwise, including photographic and         **
35**  recording, or in connection with any information storage     **
36**  or retrieval system without written permission from          **
37**  Jury Systems Incorporated.                                   **
38**                                                               **
39*****************************************************************
40*****************************************************************
41**                                                               **
42**  PROGRAM DESCRIPTION:                                         **
43**  ====================                                         **
44**                                                               **
45**  JSIRAND1  -  Produce a uniform, normal and exponential       **
46**               random number.                                  **
47**                                                               **
48**    This routine uses logic of a 4 seed Universal random       **
49**    number generateor as described and documented by          **
50**    George Marsaglia and adopted by the State of Florida.      **
51**                                                               **
52**    The MicroFocus calling sequence is as follows:            **
53**                                                               **
54**    CALL  JSIRAND1 using  JSIRAND-PARM-BLK.                    **
55**                                                               **
56**---------------------- Modification Log ----------------------**
```

```
 57** mm/dd/yy uuu - xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  **
 58**                                                                **
 59******************************************************************
 60 ENVIRONMENT DIVISION.
 61/
 62 CONFIGURATION SECTION.
 63 SOURCE-COMPUTER.
 64      IBM-PC.
 65 OBJECT-COMPUTER.
 66      IBM-PC.
 67/
 68 INPUT-OUTPUT SECTION.
 69 FILE-CONTROL.
 70
 71 DATA DIVISION.
 72 FILE SECTION.
 73
 74******************************************************************
 75***      W O R K I N G - S T O R A G E   S E C T I O N        ***
 76******************************************************************
 77 WORKING-STORAGE SECTION.
 78
 79*************
 80*** Constants
 81*************
 82
 83 78  78-dflt-seed-IJ            value 1082.
 84 78  78-dflt-seed-KL            value 9793.
 85
 86*************
 87*** The following working storage data names for the various work
 88*** fields are taken directly from the document provided by the
 89*** State of Florida. This makes logic comparison between the two
 90*** program and document simpler.
 91***
 92*************
 93
 94 01  workIntegers.
 95      05  I          pic x(04) comp-5  value 0.
 96      05  II         pic x(04) comp-5  value 0.
 97      05  J          pic x(04) comp-5  value 0.
 98      05  K          pic x(04) comp-5  value 0.
 99      05  L          pic x(04) comp-5  value 0.
100      05  M          pic x(04) comp-5  value 0.
101      05  IJ         pic x(04) comp-5  value 0.
102      05  KL         pic x(04) comp-5  value 0.
103      05  i-temp     pic x(04) comp-5  value 0.
104
105 01  floatValues.
106      05  U          comp-1  occurs  97.
107      05  C          comp-1.
108      05  CD1        comp-1.
109      05  CM1        comp-1.
110      05  S          comp-1.
```

```
  111       05  T          comp-1.
  112       05  UNI        comp-1.
  113
  114
  115/
  116*****************************************************************
  117***          L I N K A G E        S E C T I O N              ***
  118*****************************************************************
  119 LINKAGE SECTION.
  120
* 121      COPY  "JSIRAND.UWS".
  122*-------------------------------------------------------------*
  123*                                                             *
  124*                        JSIRAND.UWS                          *
  125*                                                             *
  126*  (c) Jury Systems Incorporated, 1998    All rights reserved.  *
  127*-------------------------------------------------------------*
  128*                                                             *
  129*   This copylib defines data fields needed as arguments for  *
  130*   the JSIRAND  subroutine.                                  *
  131*                                                             *
  132*- - - - - - - - - - - Modifications - - - - - - - - - - - - -*
  133* mm/dd/yy uuu - xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx *
  134*                                                             *
  135* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  136
  137 78  JSIRAND-78-SUB-NM               VALUE  'JSIRAND'.
  138 78  JSIRAND-78-RAND1-SUB-NM         VALUE  'JSIRAND1'.
  139
  140 01  JSIRAND-PARM-BLK.
  141     05  JSIRAND-FC-FUNC-CODE   PIC 9(01)          VALUE ZERO.
  142         88  JSIRAND-FC-SET-SEEDS                  VALUE 1.
  143         88  JSIRAND-FC-GEN-RAND                   VALUE ZERO.
  144     05  JSIRAND-SEED1          PIC 9(05) COMP-5   VALUE ZERO.
  145     05  JSIRAND-SEED2          PIC 9(05) COMP-5   VALUE ZERO.
  146     05  JSIRAND-NBR-AREA.
  147         10  JSIRAND-RAND-NBR   PIC 9(07).
  148         10  FILLER             PIC 9(01).
  149     05  FILLER  REDEFINES  JSIRAND-NBR-AREA.
  150         10  JSIRAND-RAND-NBR-8  PIC 9(08).
  151     05  JSIRAND-NOTUSED         PIC X(08)       VALUE SPACES.
  152
  153***   End of JSIRAND.UWS   ***
  154
  155/
  156*****************************************************************
  157***          P R O C E D U R E    D I V I S I O N             ***
  158*****************************************************************
  159 PROCEDURE DIVISION    USING  JSIRAND-PARM-BLK.
  160
  161 0000-MAIN.
  162*---------*
  163***
  164*** Either generate a uniform random number or initialize the
```

```
165*** seed values depending upon function request.
166***
167      IF  JSIRAND-FC-SET-SEEDS
168          PERFORM  1000-SET-SEEDS
169          SET  JSIRAND-FC-GEN-RAND    TO  TRUE
170      ELSE
171          PERFORM  2000-GEN-RAND
172      END-IF
173      CONTINUE.
174
175
176 0000-PGM-EXIT.
177*-------------*
178***
179
180      EXIT PROGRAM.
181
182/
183 1000-SET-SEEDS.
184*--------------*
185***
186*** Initialize starting values from seed values passed in
187*** by user.
188***
189
190      MOVE  JSIRAND-SEED1      TO  IJ
191      MOVE  JSIRAND-SEED2      TO  KL
192
193      *> Insure seeds are in proper range
194      if  IJ < 0 or IJ > 31238
195          move  78-dflt-seed-IJ  to  IJ
196      end-if
197      if  KL < 0 or KL > 30081
198          move  78-dflt-seed-KL  to  KL
199      end-if
200
201      *> Take 2 user input "seed" values and convert to 4 seeds
202      compute I = function mod(IJ/177, 177) + 2
203      compute J = function mod(IJ, 177) + 2
204      compute K = function mod(KL/169, 178) + 1
205      compute L = function mod(kL, 169)
206
207
208      *> Fill a 97 element array with random distribution
209      PERFORM  VARYING  II  FROM 1 BY 1
210        UNTIL  II > 97
211
212          MOVE  ZERO  TO  S
213          MOVE  0.5   TO  T
214          PERFORM  24 TIMES
215              COMPUTE  M =
216                       FUNCTION MOD(FUNCTION MOD(I*J, 179) * K, 179)
217              MOVE J TO I
218              MOVE K TO J
```

```
219                MOVE M TO K
220                COMPUTE  L =
221                        FUNCTION MOD (53 * L + 1, 169)
222                IF  FUNCTION MOD (L * M, 64) >= 32
223                    COMPUTE  S = S + T
224                END-IF
225                COMPUTE  T = 0.5 * T
226            END-PERFORM
227            MOVE  S   TO  U(II)
228        END-PERFORM
229        COMPUTE  C     = 362436 / 16777216
230        COMPUTE  CD1   = 7654321 / 16777216
231        COMPUTE  CM1   = 16777213 / 16777216
232
233        *> Initialize  I and J which will be used as indexes
234        *> by the generation module.
235        MOVE  97      TO I
236        MOVE  33      TO J
237        CONTINUE.
238
239
240/
241 2000-GEN-RAND.
242*-------------*
243***
244*** Generate a uniform, normal and exponential random number
245***
246***  Be sure that the "set-seed" function has already been called
247***
248        IF  I = ZERO  OR  J = ZERO
249            PERFORM  1000-SET-SEEDS
250        END-IF
251
252***
253*** Use the previously generated table to create the new number
254***
255        COMPUTE UNI = U(I) - U(J)
256        IF  UNI < ZERO
257            ADD   1       TO  UNI
258        END-IF
259
260        *> Put the current Random number into the table for use in
261        *> later iterations...
262        MOVE  UNI  TO  U(I)
263
264        *> Recycle I and J when they reach the beginning of the
265        *> random array
266        SUBTRACT  1  FROM  I
267        SUBTRACT  1  FROM  J
268        IF  I = 0
269            MOVE  97  TO  I
270        END-IF
271        IF  J = 0
272            MOVE  97  TO  J
```

```
273      END-IF
274
275      COMPUTE C = C - CD1
276      IF  C  <  0
277          COMPUTE C = C + CM1
278      END-IF
279      COMPUTE  UNI = UNI - C
280      IF  UNI <  0
281          ADD  1     TO  UNI
282      END-IF
283
284      *> Shift fractional bits left 24 times giving whole decimal
285      *> number.
286      COMPUTE  JSIRAND-RAND-NBR-8  = (4096 * (4096 * UNI))
287
288      *> Compute 7 digit random number by dropping leas significant
289      *> digit and rounding
290      COMPUTE  JSIRAND-RAND-NBR ROUNDED
291            = (4096 * (4096 * UNI)) / 10
292
293
294      CONTINUE.
295
296***  End of jsirand2.CBL  ***
```

## III. APPENDIX C – Toward a Universal Random Number Generator By George Marsaglia and Arif Saman

Statistics & Probability Letters 8 (1990) 35–39
North-Holland January 1990

# TOWARD A UNIVERSAL RANDOM NUMBER GENERATOR

George MARSAGLIA and Arif ZAMAN

*Supercomputer Computations Research Institute and Department of Statistics, The Florida State University, Tallahassee, FL 32306, USA*

Wai Wan TSANG

*Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong*

Received December 1987
Revised June 1988

*Abstract:* This article describes an approach towards a random number generator that passes all of the stringent tests for randomness we have put to it, and that is able to produce exactly the same sequence of uniform random variables in a wide variety of computers, including TRS80, Apple, Macintosh, Commodore, Kaypro, IBM PC, AT, PC and AT clones, Sun, Vax, IBM 360/370, 3090, Amdahl, CDC Cyber and even 205 and ETA supercomputers.

*Keywords:* Random number generator.

## 1. Introduction

An essential property of a random number generator is that it produce a satisfactorily random sequence of numbers. Increasingly sophisticated uses have raised questions about the suitability of many of the commonly available generators (see, for example, Marsaglia, 1986) Another shortcoming in many, indeed most, random number generators is they are not able to produce the same sequence of variables in a wide variety of computers. Such a requirement seems essential for an experimental science that lacks standardized equipment for verifying results.

We address these deficiencies here, suggesting a combination generator tailored particularly for reproducibility in all CPU's with at least 16-bit integer arithmetic. The random numbers themselves are reals with 24-bit fractions, uniform on [0, 1). We provide a suggested Fortran implementation of this "universal" generator, together with suggested sample output with which one may verify that a particular computer produces exactly the same bit patterns as the computers enumerated in the abstract. The Fortran code is so straightforward that versions may be readily written for other languages; so far, correspondents have written or confirmed results for Basic, Fortran, Pascal, C, Modula II and Ada versions.

A list of desirable properties for a random number generator might include:

(1) *Randomness.* Provides a sequence of independent uniform random variables suitable for all reasonable applications. In particular, passes all the latest tests for randomness and independence.

(2) *Long period.* Able to produce, without repeating the initial sequence, all of the random variables for the huge samples that current computer speeds make possible.

(3) *Efficiency.* Execution is rapid, with modest memory requirements.

(4) *Repeatability.* Initial conditions (seed values) completely determine the resulting sequence of random variables.

(5) *Portability.* Identical sequences of random variables may be produced in a wide variety of computers, for given starting values.

(6) *Homogeneity.* All subsets of bits of the

0167-7152/90/$3.50 © 1990, Elsevier Science Publishers B.V. (North-Holland)

numbers must be random, from the most- to the least-significant bits.

## 2. Choice of the method

We seek a generator that has all of these desirable properties. (All? Well, almost all; the generator we propose falls short on *efficiency*, for it is slower than some of the standard, simple, machine-dependent generators. But all of the standard generators fail one or more of the stringent tests for randomness. See Marsaglia, 1986.)

Our choice is a combination generator. It combines two different generators. The principal component of the two has a very long period, about $10^{36}$. It is a lagged-Fibonacci generator based on the binary operation $x \cdot y$ on reals $x$ and $y$ defined by

$$x \cdot y = \{\text{if } x \geqslant y \text{ then } x - y, \text{ else } x - y + 1\}.$$

Ultimately, we require a sequence of reals on $[0, 1)$: $U_1, U_2, U_3, \ldots$, each with a 24-bit fraction. We choose 24 bits because it is the most common fraction size for single-precision reals and because the operation $x \cdot y$ can be carried out exactly, with no loss of bits, in most computers—those with reals having fractions of 24 or more bits.

This choice allows us to use a lagged-Fibonacci generator, designated $F(r, s, \cdot)$, as the basic component of our universal generator. It provides a sequence of reals by means of the operation $x \cdot y$:

$$x_1, x_2, x_3, \ldots, \quad \text{with } x_n = x_{n-r} \cdot x_{n-s}.$$

The lags $r$ and $s$ are chosen so that the sequence is satisfactorily random and has a very long period. If the initial (seed) values, $x_1, x_2, \ldots, x_r$ are each 24-bit fractions, $x_i = I_i/2^{24}$, then the resulting sequence, generated by $x_n = x_{n-r} \cdot x_{n-s}$, will produce a sequence with period and structure identical to that of the corresponding sequence of integers,

$$I_1, I_2, I_3, \ldots, \quad \text{with } I_n = I_{n-r} - I_{n-s} \bmod 2^{24}.$$

For suitable choices of the lags $r$ and $s$ the period of the sequence is $(2^{24} - 1) \times 2^{r-1}$. The need to choose $r$ large for long period and randomness must be balanced with the resulting memory costs: a table of the $r$ most recent $x$ values must be stored. We have chosen $r = 97$, $s = 33$. The resulting cost of 97 storage locations for the circular list needed to implement the generator seems reasonable. A few hundred memory locations more or less is no longer the problem it used to be. The period of the resulting generator is $(2^{24} - 1) \times 2^{96}$, about $2^{120}$, which we boost to $2^{144}$ by the other part of the combination generator, described below. Methods for establishing periods for lagged-Fibonacci generators are given in Marsaglia and Tsay (1985).

## 3. The second part of the combination

We now turn to choice of a generator to combine with the $F(97, 33, \cdot)$ chosen above. We are not content with that generator alone, even though it has an extremely long period and appears to be suitably random from the stringent tests we have applied to it. But it fails one of the tests: the birthday-spacings test. A typical version of this test goes as follows: let each of the generated values $x_1, x_2, \ldots$, represent a "birthday" in a "year" of, say, $2^{24}$ days. Choose, say, $m = 512$ birthdays, $x_1, x_2, \ldots, x_m$. Sort these to get $x_{(1)} \leqslant x_{(2)} \leqslant \cdots \leqslant x_{(m)}$. Form spacings $y_1 = x_{(1)}$, $y_2 = x_{(2)} - x_{(1)}$, $y_3 = x_{(3)} - x_{(2)}, \ldots, y_m = x_{(m)} - x_{(m-1)}$. Sort the spacings, getting $y_{(1)} \leqslant y_{(2)} \leqslant \cdots \leqslant y_{(m)}$. The test statistic is $J$, the number of duplicate values in the sorted spacings, i.e., initialize $J \leftarrow 0$ then for $i = 2$ to $m$, put $J \leftarrow J + 1$ if $y_{(i)} = y_{(i-1)}$. The resulting $J$ should have a Poisson distribution with mean $\lambda = m^3/(4n) = m^3/2^{26}$.

Lagged-Fibonacci generators $F(r, s, \cdot)$ fail this test, unless the lag $r$ is more than 500 or the binary operation $\cdot$ is multiplication for odd integers mod $2^k$. The count $J$, the number of duplicate spacings, is only asymptotically Poisson distributed, requiring that $n$, the length of the year, be large. Applications of the birthday spacings test typically choose $n$ to be 100000 or more—for example, using the leftmost 18 or more bits of the random number to provide a "birthday".

Detailed discussion of the test and test results will appear elsewhere, but here are results of a typical test on four different generators (see Table 1): two lagged-Fibonacci generators using subtrac-

Table 1
A birthday-spacings test for four generators

| duplicate spacings | number expected | $F(97, 33, -)$ observed | $F(55, 24, -)$ observed | $F(97, 33, *)$ observed | congruential observed |
|---|---|---|---|---|---|
| 0 | 36.79 | 41 | 29 | 41 | 36 |
| 1 | 36.79 | 16 | 14 | 33 | 37 |
| 2 | 18.39 | 18 | 34 | 20 | 20 |
| ≥ 3 | 8.03 | 25 | 23 | 6 | 7 |
| chi-square for 3 d.f. | | 48.1 | 56.91 | 1.53 | 0.29 |
| probability of better fit | | 1.0000 | 1.0000 | 0.432 | 0.33 |

tion, a lagged-Fibonacci generator using multiplication on odd integers, and a popular congruential generator, $x_n = 69069x_{n-1}$, all for modulus $2^{32}$. The leftmost 25 bits are used to choose one of 512 birthdays. Thus $n = 2^{25}$ and $m = 2^9$, so $J$ should be Poisson distributed with $\lambda = m^3/(4n) = 1$. Of the four, only the $F(97, 33, *)$ and the congruential generator pass. The two lagged-Fibonacci generators using subtraction fail the test. Their duplicate-spacing counts are far from Poisson distributed, and remain so, whatever the choice of seed values, (and for a wide variety of choices of $n$, $m$ and lags $r$, $s$ as well).

In order to get a generator that passes *all* the stringent tests we have applied, we have resorted to combining the $F(97, 33, \cdot)$ generator with a second generator. Combining different generators has strong theoretical support (see Marsaglia, 1986).

Our choice of the second generator is a simple arithmetic sequence for the prime modulus $2^{24} - 3 = 16777213$. For an initial integer $I$, subsequent integers are $I - k$, $I - 2k$, $I - 3k, \ldots,$ mod 16777213. This may be implemented in 24-bit reals, again with no bits lost, by letting the initial value be, say $c = 362436/17666216$, then forming successive 24-bit reals by the operation $c \circ d$, defined as

$$c \circ d = \{\text{if } c \geqslant d \text{ then } c - d, \quad \text{else } c - d + 16777213/16777216\}.$$

Here $d$ is some convenient 24-bit rational, say $d = 7654321/16777216$. The resulting sequence has period $2^{24} - 3$, and while it is far too regular for use alone, it serves, when combined by means of the $\cdot$ operation with the $F(97, 33, \cdot)$ sequence, to provide a composite sequence that meets all of the criteria mentioned in the introduction, except for efficiency. All of the operations in the combination generator are simple and efficient, and the generation part is quite simple, but the setup procedure, setting the initial 97 $x$ values, is more complicated than the generating procedure. We now turn to details of implementation.

## 4. Implementation

We have two binary operations, each able to produce exact arithmetic on reals with 24-bit fractions:

$$x \cdot y = \{\text{if } x \geqslant y \text{ then } x - y, \text{ else } x - y + 1\},$$

$$c \circ d = \{\text{if } c \geqslant d \text{ then } c - d, \quad \text{else } c - d + 16777213/16777216\}.$$

We require computer instructions that will generate two sequences:

$x_1, x_2, x_3, \ldots, x_{97}, x_{98}, \ldots,$
with $x_n = x_{n-97} \cdot x_{n-33}$,
$c_1, c_2, c_3, \ldots,$
with $c_n = c_{n-1} \circ (7654321/16777216)$.

Then produce the combined sequence

$U_1, U_2, U_3, \ldots,$ with $U_n = x_n \cdot c_n$.

The $c$ sequence requires only one initial value, which we arbitrarily set to $c_1 = 362436/16777216$. The $x$ sequence requires 97 initial (seed) values, each a real of the form $I/16777216$, with $0 \leqslant I \leqslant 16777215$. The main problem in implementing the universal generator is in finding a suitable way to set the 97 initial values, a way that is both random and consistent from one computer to another.

Table 2
Fortran subprograms for initializing and calling UNI

```
      SUBROUTINE RSTART (I, J, K, L)
      REAL U(97)
      COMMON /SET1/ U, C, CD, CM, IP, JP
      DO 2 II = 1,97
      S = 0.
      T = .5
        DO 3 JJ = 1,24
        M = MOD (MOD(I*J, 179)*K, 179)
        I = J
        J = K
        K = M
        L = MOD(53*L + 1, 169)
        IF(MOD(L*M, 64).GE.32) S = S+T
3       T = .5*T
2     U(II) = S
      C = 362436./16777216.
      CD = 7654321./167777216.
      CM = 16777213./16777216.
      IP = 97
      JP = 33
      RETURN
      END
```

```
      FUNCTION UNI()
C***  FIRST CALL RSTART (I, J, K, L)
C***  WITH I, J, K, L INTEGERS
C***  FROM 1 TO 168, NOT ALL 1
C***  NOTE: RSTART CHANGES I, J, K, L
C***  SO BE CAREFUL IF YOU REUSE
C***  THEM IN THE CALLING PROGRAM.
      REAL U(97)
      COMMON /SET1/ U, C, CD, CM, IP, JP
      UNI = U(IP) - U(JP)
      IF(UNI.LT.0.) UNI = UNI + 1.
      U(IP) = UNI
      IP = IP - 1
        IF(IP.EQ.0) IP = 97
      JP = JP - 1
      IF(JP.EQ.0) JP = 97
      C = C - CD
      IF(C.LT.0.) C = C + CM
      UNI = UNI - C
      IF(UNI.LT.0) UNI = UNI + 1
      RETURN
      END
```

The $F(97, 33, -\bmod 1)$ generator is quite robust, in that it gives good results even for bad initial values. Nonetheless, we feel that the initial table should itself be filled by means of a good generator, one that need not be fast because it is used only for the setup. Of course, we might ask that the user provide 97 seed values, each with an exact 24-bit fraction, but that seems too great a burden. After considerable experimentation, we recommend the following procedure:

Assign values bit-by-bit to the initial table $U(1), U(2), \ldots, U(97)$ with a random sequence of bits $b_1, b_2, b_3, \ldots$. Thus $U(1) = 0.b_1 b_2 \ldots b_{24}$, $U(2) = 0.b_{25} b_{26} \ldots b_{48}$ and so on. The sequence of bits is generated by combining two different generators, each suitable for exact implementation in any computer: one a 3-lag Fibonacci generator using multiplication, the other an ordinary congruential generator for modulus 169.

The two sequences that are combined to produce bits $b_1, b_2, b_3, \ldots$, are:

$y_1, y_2, y_3, y_4, \ldots,$
 with $y_n = y_{n-3} \times y_{n-2} \times y_{n-1} \bmod 179,$
$z_1, z_2, z_3, z_4, \ldots,$
 with $z_n = 53 z_{n-1} + 1 \bmod 169.$

Then $b_i$ in the sequence of bits is formed as the sixth bit of the product $y_i z_i$, using operations which may be carried out in most programming languages: $b_i =$ (if $y_i z_i \bmod 64 < 32$ then 0, else 1).

Choosing the small moduli 179 and 169 ensures that arithmetic will be exact in all computers, after which combining the two generators by multiplication and bit extraction stays within the range of 16-bit integer arithmetic. The result is a sequence of bits that passes extensive tests for randomness, and thus seems well suited for initializing a universal generator.

The user's burden is reduced to providing three seed values for the 3-lag Fibonacci sequence, and one seed value for the congruential sequence $z_n = 53 z_{n-1} + 1 \bmod 169$. For Fortran implementations (see Table 2) of the universal generator, we recommend that a table U(1), ..., U(97) be shared, in (labelled) COMMON, with a setup routine, say RSTART(I, J, K, L) and the function subprogram, UNI(), that returns the required uniform variate. An alternative approach is to have a single subprogram that includes an entry for the setup procedure, but not all Fortran compilers allow multiple entries to a subprogram. The seed values for

the setup are I, J, K and L. Here I, J, K must be in the range 1 to 178, and not all 1, while L may be any integer from 0 to 168. If (positive) integer values are assigned to I, J, K, L outside the specified ranges, the generator will still be satisfactory, but may not produce exactly the same bit patterns in different computers, because of uncertainties when integer operations involve more than 15 bits.

To use the generator, one must first call RSTART(I, J, K, L) to set up the table in labelled common, then get subsequent uniform random variables by using UNI() in an expression as, for example, in X = UNI( ) or Y = 2.*UNI( ) − ALOG(UNI( )), etc.

## 5. Verifying the universality

We now suggest a short Fortran program for verifying that the universal generator will produce exactly the same 24-bit reals that other computers produce. Conversion to an equivalent Basic, Pascal or other program should be transparent, but those who wish to may get the setup, generating and verification programs for various languages by writing to the authors.

Assume then that you have implemented the UNI routine with its RSTART setup procedure in your computer. Running the short program of Table 3, or an equivalent, should produce the output as shown in Table 4.

If it does, you will almost certainly have a universal random number generator that passes all the standard tests, and all the latest—more stringent—tests for randomness, has an incredibly long period, about $2^{144}$, and, for given RSTART values I, J, K, L, produces the same sequence of 24-bit reals as do almost all other commonly-used computers.

Good luck.

Table 3

```
      CALL RSTART(12, 34, 56, 78)
      DO 6 II = 1, 20005
      X = UNI()
  6   IF(II.gt.20000)
        print 21, (MOD(INT(X*16.**I), 16), I = 1, 7)
 21   FORMAT(8X, 7I3)
      END
```

Table 4

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 3 | 11 | 3 | 0 | 4 | 0 |
| 13 | 8 | 15 | 11 | 11 | 14 | 0 |
| 6 | 15 | 0 | 2 | 3 | 11 | 0 |
| 5 | 14 | 2 | 14 | 4 | 8 | 0 |
| 7 | 15 | 7 | 10 | 12 | 2 | 0 |

## References

Marsaglia, G. (1986), A current view of random number generators, *Computer Science and Statistics: Proc. 16th Symp. Interface, Atlanta, March 1984* (Elsevier Science Publishers, Amsterdam).

Marsaglia, G. and L.H. Tsay (1985), Matrices and the structure of random number sequences, *Linear Algebra Appl.* 67, 147–156.