Bootstraps, Permutation Tests, and Sampling With and Without Replacement Orders of Magnitude Faster Using SAS®

John Douglas (J.D.) Opdyke*, President
DataMineIt, JDOpdyke@DataMineIt.com

Presented at Joint Statistical Meetings, Miami, Florida, 07/30-08/04, 2011

*The views presented herein are the views of the sole author, J.D. Opdyke, and do not necessarily reflect the views of other conference participants or discussants. All all calculations and computations were performed by J.D. Opdyke using SAS®.

Contents

1. Objective – Sampling Algorithms for "Big Data," all within SAS®
2. Results – Speed, All Else Equal: No Storage Required
3. Approach: Ask, "Which Things is SAS® Extremely Good At v. Other Packages?"
4. Exploiting Specific SAS® Speed Advantages:
   a. O(*N*) is better than O(*n*) (even when *N*>>*n*)??! Huh?!
   b. Sampling With (OPDY) and Without (OPDN) Replacement
   c. Keep it in Memory! No I/O
   d. Linear Time Complexity
5. Competing Approaches within SAS®:
   a. The PROCs: NPAR1WAY, SurveySelect, MultTest
   b. Array-Based Bebbington (1975)
   c. Hashing (Tables and Iterators)
   d. DA (Direct Access), Out-SM (Output Sort-Merge), A4.8 (Tille)
6. Conclusions: OPDY/OPDN
   - Speed (ORDERS OF MAGNITUDE FASTER (80 Sec. vs. 21.5 Hours))
   - Scalability (linear time complexity)
   - Robustness (Hashing and Procs Crash)
   - Generalizability (bootstrap multivariate models, permute any test statistic)
7. Acknowledgements & References

1. Objective: "Big Data" Sampling Algos within SAS®

Specific Objective:

Develop sampling algorithms specifically exploiting on SAS®'s fast dataset processing capabilities to implement bootstraps and permutation tests on "Big Data" without the prohibitive runtime constraints of existing SAS Procs (NPAR1WAY, SurveySelect, Multtest).

SPEED GOAL: Orders of Magnitude, all else equal. This is always needed by the likes of large banks and other financial institutions constantly running into the brick walls of runtime constraints: massive amounts of data + computationally intensive methods. Existing methods within SAS® simply cannot handle it. And SAS® is arguably faster than any other major statistical software package!

2. Results: Speed, All Else Equal

Relative (Real*) Runtimes: Challengers v. Bootstrap OPDY & Permutation Test OPDN

N (per Stratum)	#Strata	n = m	Challenger	vs. OPDY	vs. OPDY_Boot_FT1
10,000,000	12	2000	SurveySelect	218.3x	990.0x
10,000	12	2000	Hash Table + Hash Iterator	24.3x	28.9x
10,000	6	500	Hash Table + Hash Iterator	Challenger Crashed	Challenger Crashed
				vs. OPDN	vs. OPDN_Perm_FT1
7,500,000	6	2000	SurveySelect	242.0x	530.0x
1,000,000	12	2000	MultTest	685.1x	5,970.0x
7,500,000	2	2000	NPAR1WAY	353.0x	400.0x
10,000,000	2	500	NPAR1WAY	Challenger Crashed	Challenger Crashed
7,500,000	2	2000	Simultaneous Bebbington	201.0x	566.0x

*Relative CPU runtimes were very similar and are reported with complete simulation results in Opdyke (2010) and Opdyke (2011). OPDY_Boot_FT1 and OPDY_Perm_FT1 are the proprietary versions of published OPDY and OPDN.

2. Results: Speed, All Else Equal

Runtimes in Absolute Terms:

- **OPDY_Boot_FT1 Bootstraps in 78 seconds where Proc SurveySelect Bootstraps in 21.5 hours.**
- **OPDN_Perm_FT1 Conducts Permutation Tests in under 2 minutes where Proc MultTest Permutes in over 1 week.**

3. Approach: Ask, “Which Things is SAS® the Best At?”

SAS® is VERY FAST Compared to Other Statistical Packages at 3 Things:

1. **Reading in Datasets**
2. **Retaining Data Values Across Records**
3. **Looping on a Specific Record**

So Design Sampling Algorithms that Exploit These Advantages!

3. Approach: Ask, “Which Things is SAS® the Best At?”

Combine 1. and 2. in order to fill an array and accomplish 3. USING TEMPORARY ARRAYS!

1. **There is no faster way to fill an array in SAS®, from scratch, than to read-in values as the dataset is read in, record by record (things like Proc Transpose are SLOW and crash on large arrays).**
2. **TEMPORARY Arrays retain values across records automatically and save HUGE amounts of memory by avoiding assigning names to all the array cells**
3. **<u>For YEARS NOW</u>, TEMPORARY ARRAYS HAVE HAD NO 32,000 CELL/VARIABLE LIMIT! Only 2GB RAM allowed 125 million cells!**
4. **Also, NO STORAGE REQUIRED!! So no storage constraints, and no I/O so MUCH FASTER EXECUTION.**

3. Approach: Ask, "Which Things is SAS® the Best At?"

So once a dataset (column) has efficiently spilled into a TEMPORARY array (row), perform calculations with FAST LOOPING.

1. **Since TEMPORARY arrays canNOT be saved to dataset, the users will never crash the code accidentally**
2. **Each column can be read-in by BY VARIABLE combinations, and the calculation values saved at the BY VARIABLE level**
3. **This is FASTER than "DOW" looping (see Dorfman, and Opdyke, 2011).**
4. **When CI's for bootstraps or p-values for permutation tests need to be calculated and saved, use TEMPORARY arrays to hold the m statistics from the m samples, and calculate either by looping on the m cells of the TEMPORARY arrays. The only thing that needs be saved is the final p-value/confidence interval (CI).**

3. Approach: Ask, "Which Things is SAS® the Best At?"

Note that in SAS®, if minimizing real runtimes are the practical concern, then often O(N) algos are better than (faster than) O(n) algos, even when N>>n! Why? Because its MUCH faster to simply read-in the entire dataset than it is to try to sub-select specific records.

So theoretical time complexity alone should not drive algorithm development and implementation in SAS®.

4. Exploiting the SAS® Speed Advantages

Sampling With Replacement - OPDY:

Once the large TEMPORARY array is efficiently filled, simply sample with replacement using using OPDY ("One-Pass, Duplicates? Yes").

For example, a bootstrap on means simply would be:

```
array bmeans{<# of bootstrap samples>} _TEMPORARY_;
array temp{<size of dataset or strata>} _TEMPORARY_;
do m=1 to num_bsmps;
        x=0;
        do n=1 to <bootstrap sample size>;
              x = temp[floor(ranuni(-1)*freq) + 1] + x;
        end;
        bmeans[m] = x/<bootstrap sample size>;
end;
```

4. Exploiting the SAS® Speed Advantages

Keeping the bootstrap values in a TEMPORARY array can speed things up by several multiples, and the final CI's and/or bootstrap statistic can be output, or saved in macro variables if using a data _null_ (which further speeds things up as less memory is held aside for a dataset to be output in a data step).

4. Exploiting the SAS® Speed Advantages

Sampling Without Replacement - OPDN:

Once the large TEMPORARY array is efficiently filled, simply sample with replacement using OPDN ("One-Pass, Duplicates? No").

Goodman & Hedetniemi (1982) is PERFECT for this purpose, but not noted in the statistics literature (for example, it is not cited in Tillé (2006), an authoritative statistical sampling source. But Pesarin (2000) does identify it, if not cite the source.)

An example for a permutation test of the mean is presented below, with two versions of implementation.

4. Exploiting the SAS® Speed Advantages

OPDN implementation #1 of Goodman & Hedetniemi (1982) for Permutation Tests:

***** temp[ ] is the array filled with all the data values, for current stratum, of the variable being permuted**
***** psums[ ] is the array containing the permutation sample statistic values for every permutation sample**

```
DO m = 1 to #permutation tests
    x ← 0
    tot_FREQ_hold ← # records in current stratum
    tot_FREQ ← tot_FREQ_hold
    do n = 1 to # records in smaller of Control and Treatment samples
            cell ← uniform random variate on 1 to tot_FREQ
            x   ← temp[cell] + x
            hold ← temp[cell]
            temp[cell] ← temp[tot_FREQ]
            temp[tot_FREQ] ← hold
            tot_FREQ ← tot_FREQ -1
    end;
    psums[m] ← x
END;
```

4. Exploiting the SAS® Speed Advantages

OPDN implementation #2 of Goodman & Hedetniemi (1982) for Permutation Tests:

*** temp[ ] is the array filled with all the data values, for current stratum, of the variable being permuted
*** psums[ ] is the array containing the permutation sample statistic values for every permutation sample

```
1.    DO m = 1 to #permutation tests
2.        tot_FREQ_hold ← # records in current stratum
3.        tot_FREQ ← tot_FREQ_hold
4.        do n = 1 to # records in smaller of Control and Treatment samples
5.                cell ← uniform random variate on 1 to tot_FREQ
6.                hold ← temp[cell]
7.                temp[cell] ← temp[tot_FREQ]
8.                temp[tot_FREQ] ← hold
9.                tot_FREQ ← tot_FREQ -1
10.       end;
11.       psums[m] ← sum(temp[tot_FREQ] to temp[tot_FREQ_hold])
12.   END;
```

4. Exploiting the SAS® Speed Advantages

Note that Goodman & Hedetniemi (1982) uses only a single array for ALL the permutation samples: since the initial order of the array cells doesn't matter for random sampling, the array is left as it was from the previous sample when beginning to sample the next permutation sample. This is an extremely efficient use of the large TEMPORARY array containing the entire dataset/strata of values.

4. Exploiting the SAS® Speed Advantages

Keep it in Memory: No I/O:
Note that neither OPDY nor OPDN write to disk: the entire algorithm is executed in memory, which GREATLY increases execution speed. And SAS®'s memory management of TEMPORARY arrays is second to none: OPDY and OPDN can handle datasets (technically, strata size) orders of magnitude larger than can Proc NPAR1WAY AND Hash Tables, both of which crash on datasets OPDY and OPDN handle easily.

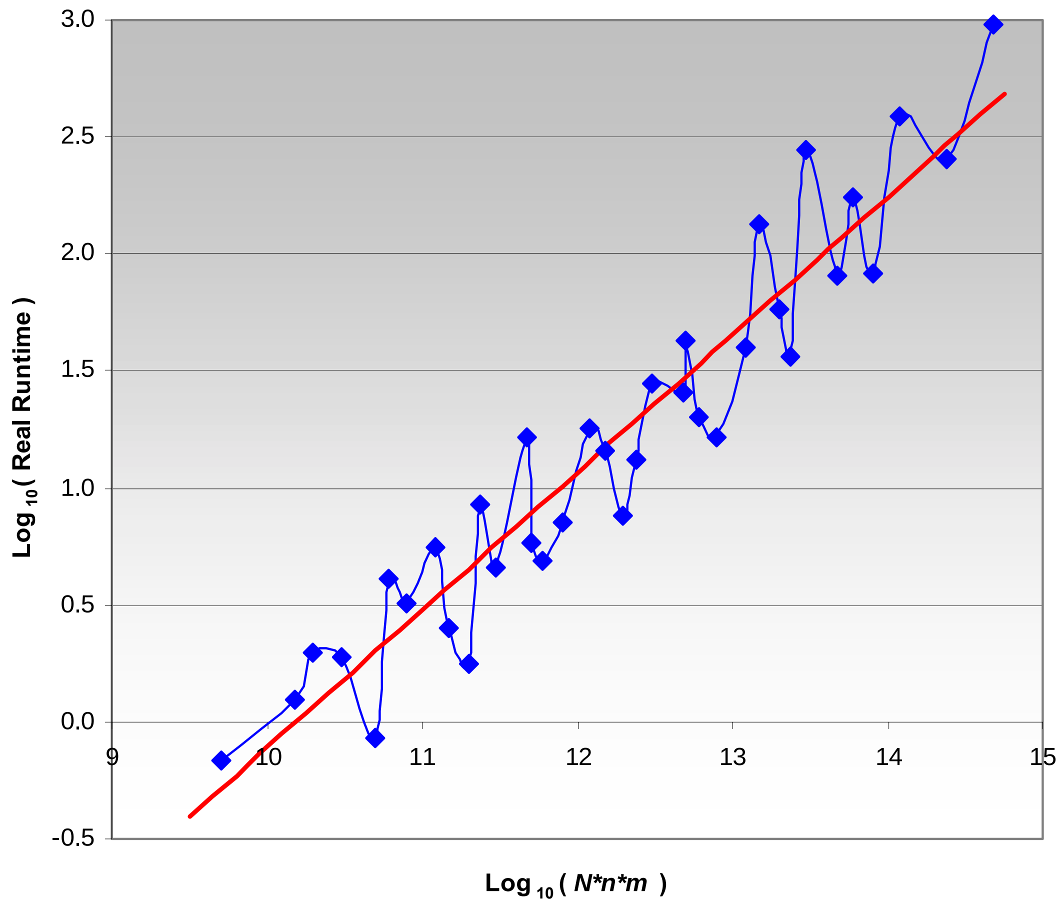
With only 2GB RAM, TEMPORARY arrays of 125 million cells can be used in SAS® v.9.2 before crashing.

Finally, both algorithms are SCALABLE, as the time complexity of both is linear runtime (see Graphs 1 and 2 below). This is not the case for Proc MultTest, Proc NPAR1WAY, or Proc SurveySelect.

4. Exploiting the SAS® Speed Advantages

Log10(Real Runtime) = -5.99228 + 0.588164 * Log10(*N*n*m*) (where *N* = all *N* across strata) (1)

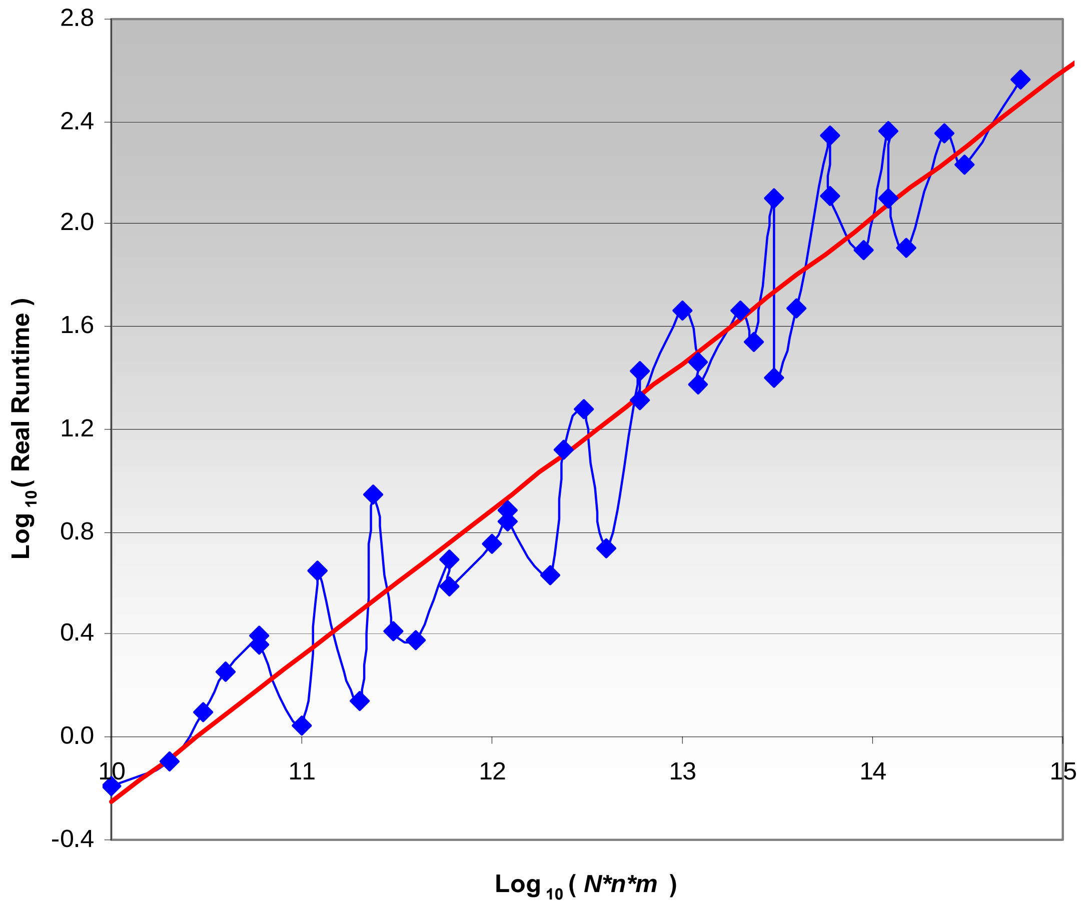
Graph 1: OPDY Real Runtime by *N*n*m* (*N* = all strata)



4. Exploiting the SAS® Speed Advantages

Log10(Real Runtime) = -5.95291 + 0.57001* Log10(*N*n*m*) (where *N* = all *N* across strata) (2)

Graph 2: OPDN Real Runtime by *N*n*m* (*N* = all strata)



© J.D. Opdyke

5. Competing Approaches within SAS®

For bootstraps:

- Proc SurveySelect
- A4.8, cited in Tillé (2006), proof derived in Opdyke (2011)
- Hash Table with Hash Iterator
- DA (direct-access)
- Out-SM (output, sort, merge)

For permutation tests:

- The Procs: SurveySelect, NPAR1WAY, MultTest
- Array-based Bebbington (1975)

2. MLE vs. Robust Statistics: Point-Counterpoint

Bebbington (1975):

```
1.  Initialize: Let $i \leftarrow 0$, $N' \leftarrow N + 1$, $n' \leftarrow n$
2.  $i \leftarrow i + 1$
3.  If $n' = 0$, STOP Algorithm
4.  Visit Data Record $i$
5.  $N' \leftarrow N' - 1$
6.  Generate Uniform Random Variate $u \sim$ Uniform[0, 1]
7.  If $u > n' / N'$, Go To 2.
    Otherwise,      Output Record $i$ into Sample
                    $n' \leftarrow n' - 1$
                    Go To 2.
```

Execute the algorithm above m times simultaneously, on each record, using an m-dimensional array.

2. MLE vs. Robust Statistics: Point-Counterpoint

A4.8 (cited in Tillé, 2006, proof provided in Opdyke, 2011):

```
1. Initialize: Let $i \leftarrow 0$, $N' \leftarrow N + 1$, $n' \leftarrow n$
2. $i \leftarrow i + 1$
3. If $n' = 0$, STOP Algorithm
4. Visit Data Record $i$
5. $N' \leftarrow N' - 1$
6. Generate Binomial Random Variate $b \sim$ Binomial($n'$, $p \leftarrow 1/N'$ )*
7. If $b = 0$, Go To 2.
   Otherwise,     Output Record $i$ into Sample $b$ times
                  $n' \leftarrow n' - b$
                  Go To 2.
```

Execute the algorithm above m times simultaneously, on each record, using an m-dimensional array.

* Using A4.8, p will never equal zero. If $p = 1$ (meaning the end of the stratum (dataset) is reached and $i = N$, $N' = 1$, and $n' = 0$) before all n items are sampled, the rand function b=rand('binomial',p,n') in SAS® assigns a value of n' to b, which is correct for A4.8.

5. Competing Approaches within SAS®

The only difference between Bebbington (1975) and A4.8 is the density determining whether, and the number of times, the observation is selected: the former, for sampling without replacement, selects at most one time using a uniform pseudo-random number generator; the latter, for sampling with replacement, selects zero or more times using a binomial pseudo-random number generator.

Bebbington is slightly more competitive because the uniform pseudo-random number generator is faster than the binomial pseudo-random number generator, which prevents A4.8 from being a viable competitor.

5. Competing Approaches within SAS®

Hashing is fast, because it is memory-based, but it runs into memory constraints, crashing on datasets/strata well under an order of magnitude smaller than those OPDY can handle.

DA is an aging, very slow, essentially obsolete method: the "POINT=" Direct Access option on the SET statement can be used on small datsets, but it is not viable for modern, computationally intensive methods requiring large amounts of resampling.

Out-SM is inadequate, too, but for different reasons: it becomes prohibitively slow because of all the I/O required. Outputting large numbers of bootstrap-sample observations, sorting them, and then merging them back on to the original data by observation id# is slow, unwieldy and resource-intensive.

So DA, and especially Out-SM, are essentially useless under "Big Data" conditions; Hashing is fast, but cannot scale to "Big Data," either.

5. Competing Approaches within SAS®

Proc SurveySelect is surprisingly slow, given that it is a relatively new procedure. Proc NPAR1WAY is faster, due to more efficient use of memory, but the price it pays is the same tradeoff as Hashing: it crashes on datasets/strata well under an order of magnitude smaller than those OPDY can handle. And Proc MultTest, the oldest of the three, is also the slowest of the three, because it is much more I/O intensive.

5. Competing Approaches within SAS®

Note that while OPDY and OPDN execute on datasets/strata much larger than Hashing and Proc NPAR1WAY can handle, they actually have theoretically unlimited dataset size: they are only limited by the size of the largest stratum in the dataset. So if a truly massive dataset was comprised of a large number of strata with fairly large, but not massive numbers of observations in each, the other methods would fail, but OPDY and OPDN would not.

6. Conclusions

- SCALABLE: No other algorithms or Procs in SAS® are at all scalable as are OPDY and OPDN for executing Bootstraps and Permutation Tests on “Big Data”.
- FASTER: They are both ORDERS OF MAGNITUDE FASTER than all other algorithms/Procs when datasets are at least of modest size, which is the only time that speed matters anyway.
- MORE ROBUST: And both can handle strata orders of magnitude larger than all other methods before those either crash, or become prohibitively slow, since they do not have linear time complexity as do OPDY and OPDN. Theoretically, dataset size is unlimited for these algorithms.

6. Conclusions

- GENERALIZABILITY: Both OPDY and OPDN also are completely generalizable: OPDY can execute bootstraps on multivariate models (DataMineIt has a version of OPDY_Boot_FT1 that does this), and OPDN can be modified to execute permutation tests using virtually any test statistic.
- No other algorithms/Procs in SAS can handle the challenge of applying computationally intensive resampling methods to "BIG DATA" as do OPDY and OPDN (and their proprietary versions, OPDY_Boot_FT1 and OPDN_Perm_FT1)

2. Results: Speed, All Else Equal

Relative (Real*) Runtimes: Challengers v. Bootstrap OPDY & Permutation Test OPDN

N (per Stratum)	#Strata	n = m	Challenger	vs. OPDY	vs. OPDY_Boot_FT1
10,000,000	12	2000	SurveySelect	**218.3x**	**990.0x**
10,000	12	2000	Hash Table + Hash Iterator	**24.3x**	**28.9x**
10,000	6	500	Hash Table + Hash Iterator	**Challenger Crashed**	**Challenger Crashed**
				vs. OPDN	**vs. OPDN_Perm_FT1**
7,500,000	6	2000	SurveySelect	**242.0x**	**530.0x**
1,000,000	12	2000	MultTest	**685.1x**	**5,970.0x**
7,500,000	2	2000	NPAR1WAY	**353.0x**	**400.0x**
10,000,000	2	500	NPAR1WAY	**Challenger Crashed**	**Challenger Crashed**
7,500,000	2	2000	Simultaneous Bebbington	**201.0x**	**566.0x**

***Relative CPU runtimes were very similar and are reported with complete simulation results in Opdyke (2010) and Opdyke (2011). OPDY_Boot_FT1 and OPDY_Perm_FT1 are the proprietary versions of published OPDY and OPDN.**

12. References

- Bebbington, A. (1975), "A Simple Method of Drawing a Sample Without Replacement," *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, Vol. 24, No. 1, 136.
- Dorfman, P., "The DOW-Loop Unrolled," Paper BB-13.
- Goodman, S. & S. Hedetniemi (1977), *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York.
- Opdyke, J.D. (2010), "Much Faster Bootstraps Using SAS®," *InterStat*, October, 2010.
- Opdyke, J.D. (2011), "Permutation Tests (and Sampling Without Replacement) Orders of Magnitude Faster Using SAS®," *InterStat*, January, 2011.
- Pesarin, F. (2001), *Multivariate Permutation Tests with Applications in Biostatistics*, John Wiley & Sons, Ltd., New York.
- Tillé, Y. (2006), *Sampling Algorithms*, New York, NY, Springer.

Acknowledgments:

I sincerely thank Nicole Ann Johnson Opdyke and Toyo Johnson for their support and belief that SAS® could produce a better bootstrap and a better permutation test.

J.D. Opdyke
President, DataMineIt
JDOpdyke@DataMineIt.com
www.DataMineIt.com

Providing statistical consulting and risk analytics to the banking, credit, and consulting sectors.