United States Department of Agriculture

National Agricultural Statistics Service

Research and
Development Division
Washington DC 20250

RDD Research Report
Number RDD-09-07

October 2009

Comparison of the Speed of the 2007 Census of Agriculture Donor Search Method Against Five Alternative Techniques

Matthew E. Gregg
Michael Hogye

EXECUTIVE SUMMARY

Since assuming responsibility for conducting the Census of Agriculture in 1997, the National Agricultural Statistics Service (NASS) has worked continuously to improve the processing capabilities of the census edit system. Following the 2002 Census of Agriculture, NASS realized that it needed to improve the speed and reliability of the edit system for the 2007 Census. The subroutine that performed item level imputation was one component of the edit that was targeted for improvement, and it underwent extensive redesign between 2002 and 2007.

The imputation routine utilized several techniques that were new for 2007. These techniques included stratifying all potential donors into groups called profiles; running the donor program as a continually running program, known in computer terminology as a daemon, using SAS/Share to mediate interprocess communication between an edit job and a donor daemon; and storing donor data in temporary arrays in SAS to provide optimal access to the data. The new imputation program achieved the speed goals set forth during the 2007 Census planning phase. Due to the time constraints involved in getting the new system in place, however, no effort was made to benchmark the speed of new donor delivery methodology.

In this report, the speed of searching through temporary arrays is compared with five alternative techniques. The use of temporary arrays requires that multiple searches are conducted within a single data step. The alternative methods conduct multiple searches by starting and stopping the data step for each search. The key measure of speed used was the CPU time spent on the search. The results of the study show that searching for a record in temporary arrays, the current operational method, had the fastest response times among the methods tested.

RECOMMENDATIONS

1. NASS should continue to use donor daemons and temporary arrays as the means of donor delivery for the 2012 Census of Agriculture.

2. In cases where large datasets are repeatedly subset on the same variable, SAS indexes or compressed indexes should be considered to improve the speed of data retrieval.

TABLE OF CONTENTS

1. INTRODUCTION ……...1

2. MEASURING RESPONSE TIMES ……..…3

2.1 Response Time Data ……3
2.2 Collecting Response Time Data …….....3

3. METHODS ……...…4

3.1 Search Methods ……4
3.2 Test Methods ……5

4. RESULTS ……..…6

5. DISCUSSION ……..9

6. RECOMMENDATIONS ……...11

7. REFERENCES ……...11

APPENDIX A: An Example of the Use of ARM Macros and Temporary Arrays …….12

APPENDIX B: Summary Tables …….14

APPENDIX C: Box Plots of Search Methods …….22

APPENDIX D: Plots of Search Methods ……....28

Comparison of the Speed of the 2007 Census of Agriculture Donor Search Method Against Five Alternative Techniques

Matthew E. Gregg, Michael Hogye[1]

Abstract

The National Agricultural Statistics Service (NASS) conducts a census of agriculture every five years. A major focus after the 2002 Census was to improve the speed and efficiency of the edit and imputation system for the 2007 Census. The imputation routine developed for 2007 achieved the speed goals set forth during the 2007 Census planning phase, but no effort was made to benchmark the new procedure against other SAS search techniques.

This study compared the search method used in the 2007 Census of Agriculture imputation routine with five alternative SAS techniques for finding a record in a SAS dataset. The results showed that the temporary array technique used in the 2007 Census had the fastest response times with the least variation among the techniques tested.

Key Words: SAS, donor, imputation, daemon, nearest neighbor

1. INTRODUCTION

Prior to assuming responsibility for the census of agriculture in 1997, the National Agricultural Statistics Service (NASS) had little experience in the use of donor imputation to correct data at the item level in records where specific data fields were found to be in error. As part of an effort to reduce analyst review and more fully automate the editing process, nearest neighbor imputation was implemented in the 2002 Census of Agriculture. The results for 2002 were mixed; ultimately the work got done but the system experienced frequent downtime, slow response times for both the edit and imputation routines, and a large volume of records with imputed values that analysts found problematic. A post-census review of the edit and imputation process led NASS to conclude that "Employees must be provided a significantly improved system for the 2007 Census with respect to the speed, stability, and quality of the data generated by the interactive edit/imputation system" (Nealon, 2004).

In the 2002 Census edit, all donors in a recipient's state plus some from adjoining states were considered for imputation. For each request, all records of a state-wide donor pool had to be considered, and there were no explicit strata within the state. After the 2002 edit experience,

[1] Matthew E. Gregg and Michael Hogye are mathematical statisticians in the USDA's National Agricultural Statistics Service (NASS) – Research and Development Division (RDD), located in Room 305, 3251 Old Lee Highway, Fairfax, VA 22030.

there was strong sentiment for reducing the scope of records examined by each donor search, with the hope that imputation could focus on appropriate donors without individually considering all of those available. For the 2007 Census, it proved more practical to create a donor dataset for each module rather than for each state. Only records and data fields relevant to the module were included. For example, only operations with cattle were kept as potential cattle donors, and for those donors, only variables required for cattle imputation were part of the dataset. For each module, the donors were stratified within a state.

Increasing the speed of the response times was one of the main focus areas for improvements to the donor imputation system for 2007. The edit system for the 2007 Census of Agriculture was designed to process records in a batch environment and in an interactive environment. In batch mode, records were processed in groups of 1,000. A batch of records had the potential to produce thousands of imputation calls, so incremental improvements in speed had a large impact on total processing time over the course of editing 1.5 million records. In interactive mode, system responsiveness was important to keep staff working effectively. Having a highly responsive donor imputation routine was an important component of an effective edit and imputation system.

The redesign of the donor imputation program centered around the idea that if the pool of potential donors was made more readily available, the time spent searching for a donor, whenever the edit called the donor imputation routine, could be greatly reduced. To make the donor pools more readily available, the decision was made to store the donor data in random access memory (RAM) instead of on the hard disk. This task was accomplished in SAS using temporary arrays and by running the donor program as a daemon. This idea was a major shift from 2002 where the imputation routine was run anew with each batch of records processed through the edit.

A daemon is a computer program that runs continuously in the background and performs a function in response to certain events. Running the donor program as a daemon consisted of inserting an infinite loop into the SAS code so that the program would never terminate. For the 2007 Census, the event that triggered the function of the donor daemon was a request for imputation. The function of the donor daemon was to find a nearest neighbor among all potential donors for the current request. Within the census of agriculture edit system, 26 modules make requests for donor imputation, and one daemon was used for each of these modules. Thus, during the data collection and editing phase of the census of agriculture, there were 26 donor daemons running concurrently to perform donor delivery to the edit system.

The solution of using daemons and temporary arrays to deliver donors was gradually achieved through experimentation with a variety of SAS programming techniques. The new system had to be completed and thoroughly tested prior to the mail-out of the 2007 Census of Agriculture. This did not allow for an in-depth analysis of how the speed of the new procedure might compare to other SAS programming techniques that could be used to search the donor pools for a suitable donor.

In this study, the speed of the temporary arrays solution used in the 2007 Census of Agriculture was benchmarked against five alternative SAS techniques used to retrieve a donor from a SAS dataset.

2. MEASURING RESPONSE TIMES

2.1 Response Time Data

Measuring the performance times for the various methods broke down into three categories: real time, user central processing unit (CPU) time, and system CPU time. The definitions for these terms are taken from the SAS Online help (SAS Institute Inc., 2002).

Real Time:	the amount of time spent to process the SAS job. Real time is also referred to as elapsed time.
User CPU:	the CPU time spent to execute the SAS code.
System CPU:	the CPU time spent to perform system overhead tasks on behalf of the SAS process.
CPU Time:	the total time spent to execute the SAS code and spent to perform system overhead tasks on behalf of the SAS process. This value is the combination of the user CPU and system CPU statistics from FULLSTIMER.

Measuring user CPU and system CPU times was the focus of the study. Differences in these time measures among the six techniques tested provide the best measure of the speed differences between the methods. In what follows, CPU time will refer to the sum of user CPU time and system CPU time.

2.2 Collecting Response Time Data

The SAS log provides information on response times, however it is limited to reporting times at the conclusion of each DATA or PROC step. This restriction to obtaining response time data using response times from the SAS log did not meet the needs of the study. Instead, a different procedure for tracking response times was needed. Application Response Measurement (ARM) filled this need. ARM was developed by an industry partnership to define a standard for measuring application performance. The result of this collaboration was the ARM application program interface (API) which is a vendor neutral interface for measuring application response time. SAS implements the ARM API through a set of ARM macros. The definition of the ARM macros states that:

> The ARM macros provide a way to measure the performance of applications as they are executing. The macros write transaction records to the ARM log. The ARM log is an external output text file that contains the logged ARM transaction records. You insert the

> ARM macros into your SAS program at strategic points in order to generate calls to the ARM API function calls. The ARM API function calls typically log the time of the call and other related data to the log file. Measuring the time between these function calls yields an approximate response-time measurement (SAS Institute Inc., 2002).

The ability of the programmer to control when and how often time data are logged through the use of ARM macros provided the flexibility needed to track response time data. The use of ARM in this study developed from the need to track response times within a data step. One of the search techniques used temporary arrays and performed repeated searches within a single data step. Using ARM, it was possible to log response time data separately for each of these events. It was then possible to calculate time spent loading data into the arrays and time spent on each search separately. An example program showing the use of ARM logging is included in Appendix A.

3. METHODS

3.1 Search Methods

The records in the census donor pools were extracted from an IBM Red Brick database of clean census records known as the Central Donor Repository or CDR. A donor pool was created for each of the 26 census edit modules that use donor imputation. The data were transformed into a standard format, and each record in a donor pool was matched to its profile (stratum) value. The SAS datasets that resulted from the process were sorted by state, profile, and record number.

Six search techniques were chosen to be included in the testing. Method 1 was based on the methodology used in the 2007 Census of Agriculture. The other search methods were chosen to compare against Method 1. Methods 2-6 follow more traditional SAS programming, where several DATA and PROC steps combine to accomplish the task of finding a donor, and these same tasks are repeated for each request. The majority of the program code for Methods 2-6 was the same, but the descriptions that follow highlight the differences in each.

Method 1 used the technique of temporary arrays, which was the method employed by the 2007 Census of Agriculture donor program. In this method, all donor data for a module and any needed supporting data were loaded and stored in temporary arrays. The program then received a request from a separate SAS program and searched through the temporary arrays for a suitable donor.

Method 2 consisted of searching through the donor pool restricting the search to only those records with the same profile or state as the recipient. The subset of records to search was created using a WHERE clause on the SET statement in the SAS program.

Method 3 was identical to Method 2 except the donor pool dataset was indexed prior to running the program. This method used the index to optimize the search for records that matched the specific profile or state criteria in the WHERE clause used to subset the data.

Method 4 used direct access through the SAS dataset option KEY=. The KEY= <*index name*> option references an index that was created for the dataset being read. In addition to the index being named in the KEY= dataset option, there needs to be a variable in the program data vector (PDV) that matches the index in name and data type. The variable in the PDV supplies the value to search for in the index file when subsetting the data.

Method 5 used direct access through the SAS dataset option POINT=. The POINT= option refers to a temporary variable in the PDV that resolves to the observation number of the record to point to. This method relies on the fact that the donor pools are sorted by profile. A support dataset was created that contained the frequencies for each profile and also the observation number of the first and last record in the profile. Knowing the exact location of the first and last record in the profile allowed the use of a loop to cycle through the observations in the profile and the POINT= option to jump directly to each record in the profile as identified by the incremented value of the loop.

Method 6 used the SAS dataset options FIRSTOBS= and OBS= to limit the records searched in the donor pool. This technique is similar to Method 5 in that it takes advantage of the data being sorted in profile order and knowing the observation number of the first and last record in the profile. Where it differs is that FIRSTOBS= and OBS= are assigned at compile time and records are read sequentially from the SAS dataset. With this technique, there is no need to set up a do loop in the datastep to cycle through the records. The program uses the automatic datastep loop to read each record in the profile.

3.2 Test Methods

For this study, the donor pools used were the final donor pools generated for the 2007 Census of Agriculture. Of the 26 modules that use donor imputation, Modules 110 and 320 were selected for testing, based primarily on the differences in size and complexity of the data. Module 110, the edit module for land use, encompassed a large portion of the land use and crop acreage edits for the census of agriculture. There was a large volume of records (1,435,375) and a large number of variables (247) in the donor pool for this module. Module 320 was the cattle and calf inventory module. The records in the donor pool were limited to those with positive cattle inventory, and the number of variables related to cattle inventory was relatively small. For Module 320, there were 643,769 records and 18 variables in the donor pool.

All programs were written and tested using SAS v9.13 and were run on the same AIX Unix system that ran the 2007 Census of Agriculture edit system. The simulation was run twice and generated two complete sets of data for the study. Programs were run as close together as

possible in order to reduce variation due to system usage by other users or programs. Response time data were downloaded to a PC for analysis and summarization.

A record requiring imputation was randomly selected from each state. Each record was classified into a stratum, called its profile. Alaska, Minnesota, and Texas were chosen for testing based on their respective profile sizes. The search for a donor in these three states represented a search in a small, medium, and large sized profile. In addition, the search programs were run for All States, where the program cycled through an imputation request from each of the 50 states.

Each program conducted 1,000 searches for a donor by looping through the search routine 1,000 times. The same recipient for each state was used throughout. In the case of All States, the same set of 50 recipients was used each time. For All States, the set of 1,000 observations contained 20 observations for each state.

In addition to testing the performance of each search method on four different states, the speed of each search method was tested with the donor pool data stored in two different memory locations. Two copies of the donor pools were created, with one copy stored on the hard disk and the other stored in RAM. Response time data were collected for the six search methods, with donor pools being accessed on hard disk and on RAM.

Finally, the conditions of the search were modified to look at different subsets of potential donors. Response time data were collected by searching through the matching profile of the recipient and also by searching through the matching state of the recipient. The profile search was done for each method with the data resident on disk and on RAM, but the state search was done with the data only resident on disk. It should be noted that the number of records in a profile is always a subset of the number of records in the state.

The nearest neighbor calculation used in the test programs was a reduced form of the calculation used in the operational program for the 2007 Census of Agriculture. The distance between a recipient and a potential donor was calculated as a standardized Euclidean distance. For the census there was a geographical distance component and a dissimilarity penalty added to the distance calculation that contributed to the total difference between the recipient and the potential donor. For this study, the geographical and dissimilarity components were not included in the difference calculation.

4. RESULTS

A partial summary of response times as measured by total CPU time is shown for Minnesota in Table 1. The results in the table show the mean and median CPU time as well as the standard deviation of the CPU time for Module 110. Table 1 shows that Method 1 produced the fastest results. The results shown for Minnesota were consistent across the other states, with temporary arrays outperforming the other search methods. In general, Method 2 produced the slowest

results. In Module 320 for Texas, Method 2 was faster than Methods 3 and 4. Methods 3-6 performed at about the same level throughout testing.

Table 1 also shows that restricting the search to the recipient's profile produced faster response times than searching through the entire state. Response times at the state level took roughly twice as long as searches at the profile level. In general, state searches were slower than searches at the profile level. The exception was Alaska in Module 320, where state level searches were as fast as or faster than searches at the profile level. Minnesota's results for Module 110, where there were approximately six times as many records in the state as in the recipient's profile, were the most extreme case of size disparity between the profile and the state.

Another result that was consistent across states was that response times when the SAS dataset resided on disk vs. RAM were virtually indistinguishable. In Table 1, the largest difference in average response times between the profile search on disk and the profile search on RAM is 0.01459 seconds. Also, the small differences don't point in a single direction. Full summary tables are available in Appendix B.

Table 1. Summary of Search Times (in seconds) for Minnesota for Module 110

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Minnesota	1	1	Temporary arrays	Profile	9,394	Disk	0.0190	0.02	0.0048
				Profile	9,394	RAM	0.0196	0.02	0.0028
				State	56,187	Disk	0.1099	0.11	0.0163
		2	WHERE clause non-indexed dataset	Profile	9,394	Disk	0.6203	0.68	0.1168
				Profile	9,394	RAM	0.6349	0.69	0.1148
				State	56,187	Disk	1.4641	1.50	0.3713
		3	WHERE clause indexed dataset	Profile	9,394	Disk	0.1729	0.18	0.0194
				Profile	9,394	RAM	0.1712	0.18	0.0191
				State	56,187	Disk	0.3557	0.37	0.0421
		4	Direct access with KEY= option	Profile	9,394	Disk	0.1747	0.18	0.0206
				Profile	9,394	RAM	0.1814	0.19	0.0204
				State	56,187	Disk	0.3752	0.40	0.0480
		5	Direct access with POINT= option	Profile	9,394	Disk	0.1680	0.18	0.0224
				Profile	9,394	RAM	0.1664	0.17	0.0215
				State	56,187	Disk	0.3035	0.32	0.0330
		6	Subsetting with FIRSTOBS= OBS= options	Profile	9,394	Disk	0.1648	0.17	0.0206
				Profile	9,394	RAM	0.1676	0.17	0.0201
				State	56,187	Disk	0.3078	0.33	0.0454

Figure 1 shows a set of box plots for Minnesota. The graph clearly shows that temporary arrays have the fastest response times among the methods tested. In addition, the spread in the data is far less for temporary arrays than for the other methods. The graph also shows that the Method 2, labeled with a D, produced the slowest and least consistent response times, and that Methods 3-6 had very similar results. See Appendix B for a full set of box plots.

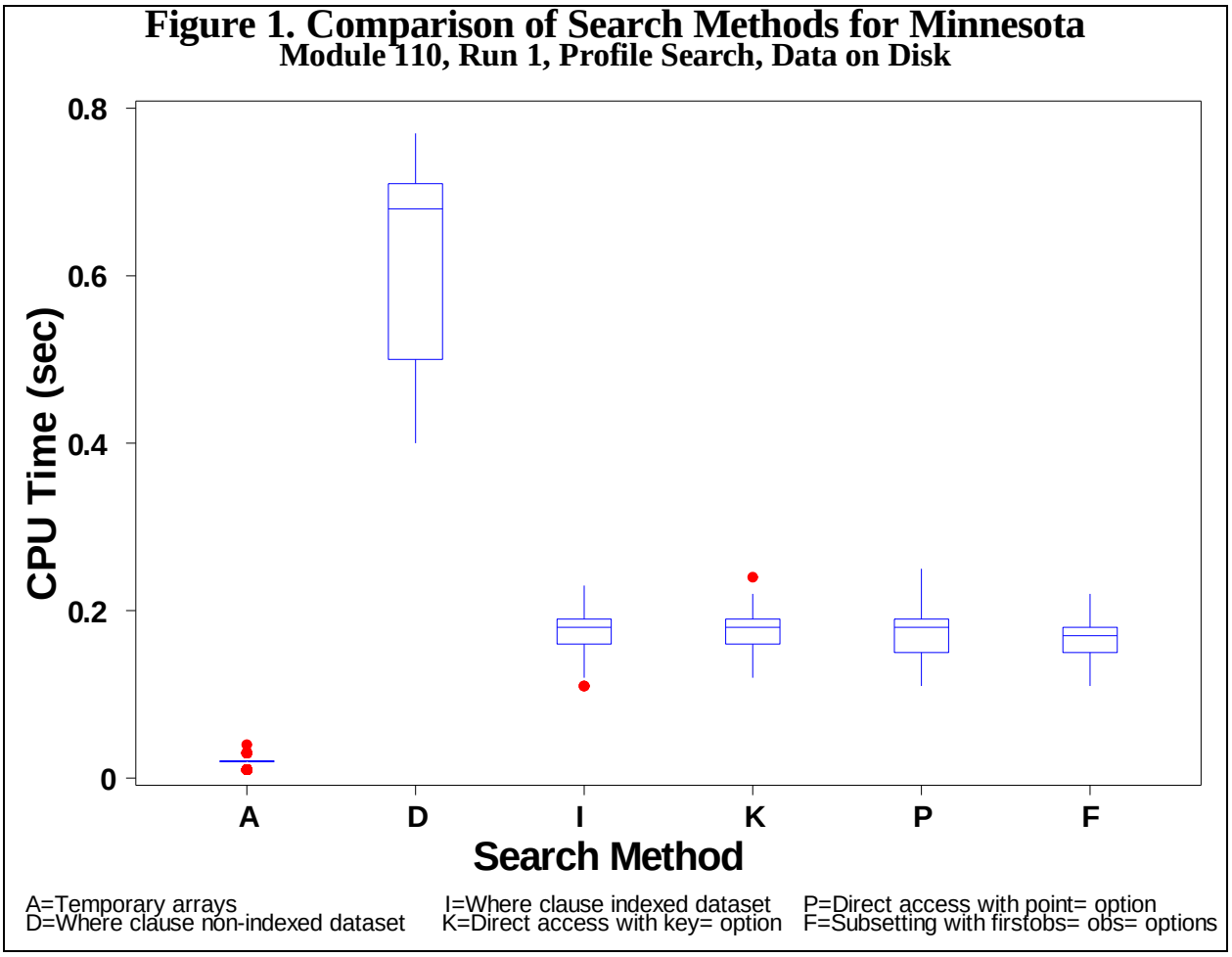
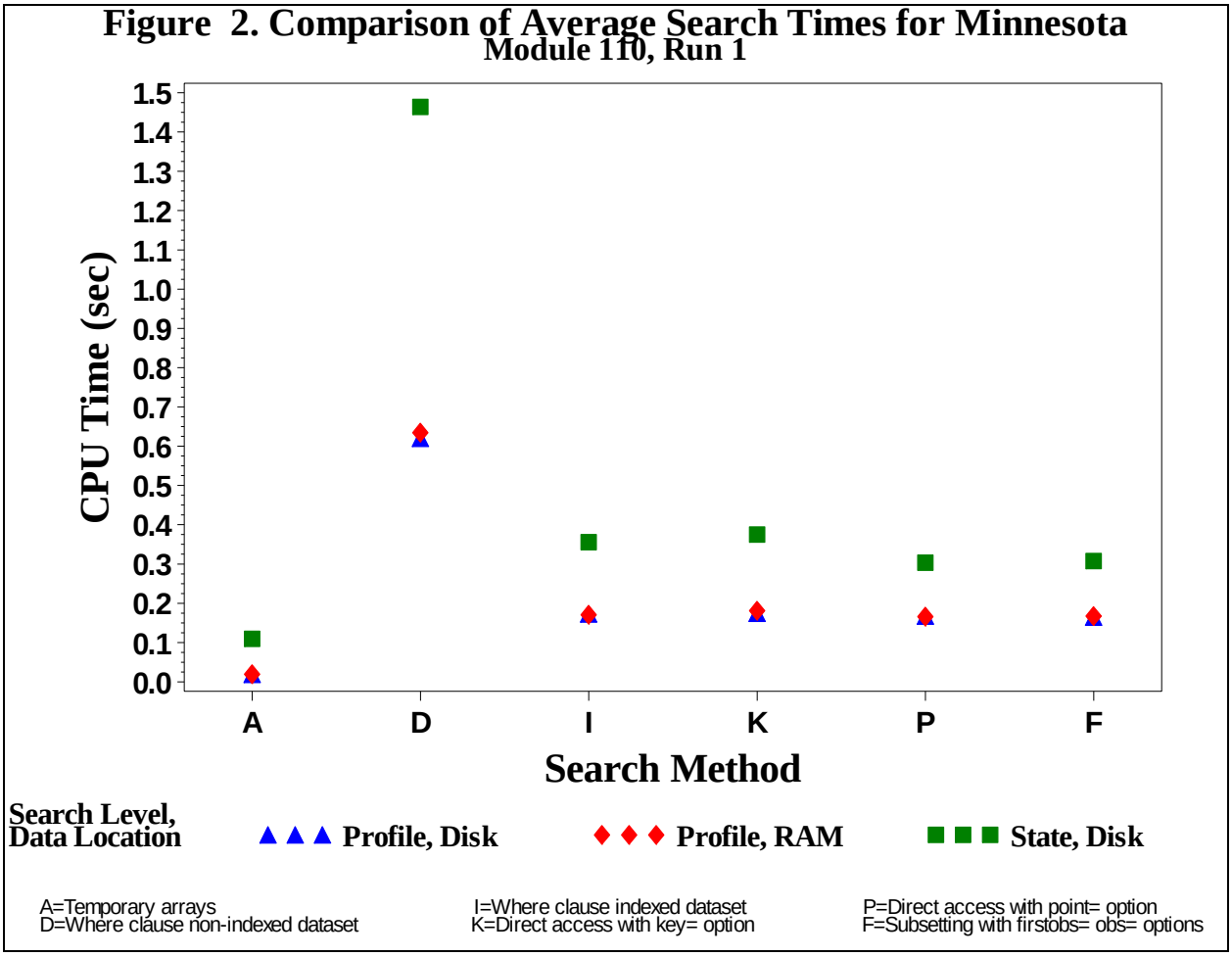


Figure 2 shows a comparison of the average response times for the six methods for Minnesota for the different combinations of data location and state (vs. profile) level. As the graph shows, there is little difference between data residing on disk or on RAM. However, there does tend to be a difference between searching through the profile and searching through the state. Searching through the state was slower for all methods and all states except for Alaska. For Module 110 and Module 320, the average response times for Alaska at the profile and state level were the same or nearly the same. In Alaska this held true for all search methods except Method 2, where it was still slower to search through the entire state. Refer to Appendix C for a full set of plots.

Figure 2. Comparison of Average Search Times for Minnesota Module 110, Run 1



5. DISCUSSION

The results of the investigation show that the use of temporary arrays was the fastest search method of the six chosen for review. The response times for temporary arrays were faster and more consistent across all methods, modules, runs, data locations and search levels. Using temporary arrays is an effective technique for performing fast searches through a SAS dataset when a large number of searches are performed during a single SAS session, or if the program is set up to run continuously.

The determination as to whether to conduct a large number of searches or set up a continuously running program is an important consideration for temporary arrays because there is a fixed time cost associated with loading data into the arrays. The average response times reported in Table 1 do not include the time spent loading data into the temporary arrays. This overhead time was not included because the main focus of the study was testing the time spent searching for a donor and because the impact of the overhead on individual search times decreases as the number of searches increases. However, the overhead is a consideration that should not be ignored. For

Module 110, loading data into the temporary arrays took an average of 78.78 seconds. This would have added an average of 0.078 seconds to the response times for each of the 1,000 searches. Adding this amount to the average response times reported in the study does not change the status of temporary arrays as the fastest method. However, consider a program that performs 100 searches in Module 110. This would translate into 0.788 seconds, which when added to the reported average response times, would make temporary arrays the slowest method.

With the exceptions for Texas detailed in the results section, Method 2, subsetting with a where clause on a non-indexed dataset, was the slowest method with the largest variation in response times. This makes sense because this technique was the most basic of the techniques tested, and nothing was done to try and improve the efficiency of finding a record with this technique.

The results for the other four search methods were close enough that it was not possible to say one was better than the other. However, there is a difference in how Methods 3-6 are implemented. Methods 3 and 4 rely on a SAS index when reducing the full donor pool to a subset of potential donors. A SAS index file stores information about each key variable and each observation with that key variable value. When the program is run, it refers to the index file to find the location of the records in the dataset that match the search criteria.

Methods 5 and 6 use what Keintz (2009) calls a compressed index that identifies the first and last record in the profile or state. The compressed index is a SAS dataset that functions in a similar fashion to a SAS index by identifying where in the main dataset a subset of records is located. Methods 5 and 6 rely on the data being sorted by the key variable. If the data are not sorted by the key variable, these methods do not work.

A clear advantage to Methods 3 and 4 is that the index is automatically updated whenever data are added, changed or deleted. A potential disadvantage to Methods 3 and 4 is that a SAS index can be a large file. The compressed index dataset by comparison is a small file. Table 2 shows the difference in size between the SAS index file and the compressed index dataset. Considerable space savings can be achieved by using the compressed index.

Table 2. Index File Sizes

	Profile		**State**	
Module	SAS Index	Compressed Index	SAS Index	Compressed Index
110	18,244 KB	17 KB	17,605 KB	17 KB
320	8,553 KB	17 KB	7,988 KB	17 KB

Methods 3-6 improved search speeds over using a where clause on a non-indexed dataset. SAS indexes worked well in this test because the data were sorted by the key variable and the key variable is used in subsetting the data. Further, the number of records to be searched was a small subset of the total number of records in the donor pool. If data in a SAS dataset are used for many different analyses where the data are subset on different variables depending on the analysis, an index may not improve performance.

6. RECOMMENDATIONS

1. NASS should continue to use donor daemons and temporary arrays as the means of donor delivery for the 2012 Census of Agriculture.

2. In cases where large datasets are repeatedly subset on the same variable, SAS indexes or compressed indexes should be considered to improve the speed of data retrieval.

7. REFERENCES

Keintz, M. (2009). *A Faster Index for Sorted SAS® Datasets*. SAS Global Forum 2009, Washington, D.C.

Nealon, J. (2004), "*PRISM Interactive Edit/Imputation System*," Decision Memorandum for internal NASS use, November 2004

SAS Institute Inc., SAS 9.1.3 Help and Documentation, Cary, NC: SAS Institute Inc., 2002-2004.

APPENDIX A. AN EXAMPLE OF THE USE OF ARM MACROS AND TEMPORARY ARRAYS IN SAS.

```
/* Turn on the ARM Macros. */
%let _armexec=1;

/* Specify the external file where ARM data will be logged. */
options armloc='C:\SAS-stuff\SAS_tempdata\armlog.txt';

/* Create a test dataset with 1,000,000 observations and 20 variables all
generated from a uniform distribution. */

data test;
      array rv{20};
      do _n_ = 1 to 1000000;
            do i = 1 to 20;
                  rv{i}=ranuni(0);
            end;
            output;
      end;
      drop i;
run;

/* Initialize the ARM log with the %arminit macro. */

%arminit(APPNAME='ARM Example', MACONLY=YES, APPIDVAR=app1);
%armgtid(TXNNAME="Datastep Transactions", MACONLY=YES, APPIDVAR=app1,
TXNIDVAR=txn1);

data test2;

/* Write the start of array loading transaction to the ARM log */
%armstrt(TXNIDVAR=txn1,txndet='Load Data into Temporary Arrays',SHDLVAR=sh1);

/* Create a two dimensional temporary array with 1,000,000 rows and 20
   columns to store the data from the test dataset. Each row will be
   treated as a unique record and each column as a unique variable.    */

      array rand_num{1000000,20} _temporary_;
      array new_rand{20} _temporary_; *Create an array with 20 columns;
      array rv{20};

      /* Populate the rand_num array from the test dataset created above. */
      do i = 1 to 1000000;
            set test;
            do j = 1 to 20;
                  rand_num[i,j]=rv{j};
            end;
      end;

/* Write a stop transaction to the ARM log to indicate loading of the array
is complete. */

%armstop(STATUS=0,SHDLVAR=sh1);
```

```
/*Write the start of the search transaction to the ARM log. */

%armstrt(TXNIDVAR=txn1, txndet='Begin Search Routine',SHDLVAR=sh2);

/* Conduct 25 searches in the rand_num array. */
      do repeat = 1 to 25;

      /* Populate the new_rand array with random numbers from a uniform
      distribution. The new_rand array simulates the record we want to
      compare against the records in the rand_num array. */

            do i = 1 to 20;
                  new_rand[i] = ranuni(0);
            end;

            nn_diff = 1E50; *Reset the nearest neighbor distance for each
                                                                  search.;

      /* Find the record in the rand_num array that is closet to the record
      represented by the values in the new_rand array by computing the sum of
      the squared distances between the variables. */

            do i = 1 to 1000000;
                  diff = 0;
                  do j = 1 to 20;
                        diff = sum(diff,(new_rand[j]-rand_num[i,j])**2);
                  end;

                  if diff < nn_diff then do;
                        nn_id=i;
                        nn_diff=diff;
                  end;

            end;
            output test2;

      /* *Update the time spent on each individual search to the ARM log. */

      %armupdt(data="Search Complete", SHDLVAR=sh2);

      end; *End repeat loop;

      keep nn_id nn_diff diff;

      /* *Write the stop time of the total search time to the ARM log. */

      %armstop(STATUS=0, SHDLVAR=sh2);
run;

%armend(MACONLY=YES,APPIDVAR=app1); *Mark the end of logging transactions.;


/* Process the arm log using the SAS supplied ARM Macros %armproc and
%armjoin */

%armproc(log=C:\SAS-stuff\SAS_tempdata\armlog.txt);
%armjoin;
```

APPENDIX B. SUMMARY TABLES

Table 3. Summary of Search Times (in seconds) for All States for Module 110

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
All	1	1	Temporary arrays	Profile	757,346	Disk	0.0320	0.02	0.0412
				Profile	757,346	RAM	0.0320	0.02	0.0412
				State	1,435,375	Disk	0.0615	0.05	0.0574
		2	WHERE clause non-indexed dataset	Profile	757,346	Disk	0.7179	0.68	0.3695
				Profile	757,346	RAM	0.7123	0.69	0.3669
				State	1,435,375	Disk	1.3402	1.40	0.2246
		3	WHERE clause indexed Dataset	Profile	757,346	Disk	0.1963	0.18	0.0802
				Profile	757,346	RAM	0.1926	0.17	0.0742
				State	1,435,375	Disk	0.2557	0.23	0.1155
		4	Direct access with KEY= option	Profile	757,346	Disk	0.2057	0.19	0.0868
				Profile	757,346	RAM	0.2029	0.18	0.0847
				State	1,435,375	Disk	0.2697	0.24	0.1286
		5	Direct access with POINT= option	Profile	757,346	Disk	0.1853	0.17	0.0660
				Profile	757,346	RAM	0.1848	0.17	0.0657
				State	1,435,375	Disk	0.2247	0.21	0.0821
		6	Subsetting with FIRSTOBS= OBS= options	Profile	757,346	Disk	0.1909	0.17	0.0687
				Profile	757,346	RAM	0.1860	0.17	0.0650
				State	1,435,375	Disk	0.2363	0.22	0.0989
	2	1	Temporary arrays	Profile	757,346	Disk	0.0283	0.02	0.0374
				Profile	757,346	RAM	0.0329	0.02	0.0420
				State	1,435,375	Disk	0.0581	0.05	0.0551
		2	WHERE clause non-indexed dataset	Profile	757,346	Disk	0.7018	0.64	0.3732
				Profile	757,346	RAM	0.6727	0.62	0.3544
				State	1,435,375	Disk	1.2590	1.31	0.2413
		3	WHERE clause indexed dataset	Profile	757,346	Disk	0.1876	0.17	0.0728
				Profile	757,346	RAM	0.1930	0.17	0.0744
				State	1,435,375	Disk	0.2450	0.22	0.1107
		4	Direct access with KEY= option	Profile	757,346	Disk	0.1932	0.17	0.0819
				Profile	757,346	RAM	0.1918	0.17	0.0784
				State	1,435,375	Disk	0.2622	0.24	0.1245
		5	Direct access with POINT= option	Profile	757,346	Disk	0.1791	0.16	0.0612
				Profile	757,346	RAM	0.1889	0.17	0.0698
				State	1,435,375	Disk	0.2170	0.20	0.0850
		6	Subsetting with FIRSTOBS= OBS= options	Profile	757,346	Disk	0.1786	0.16	0.0605
				Profile	757,346	RAM	0.1855	0.17	0.0629
				State	1,435,375	Disk	0.2242	0.21	0.0910

Table 4. Summary of Search Times (in seconds) for Alaska for Module 110

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Alaska	1	1	Temporary arrays	Profile	434	Disk	0.0008	0.00	0.0027
				Profile	434	RAM	0.0011	0.00	0.0032
				State	578	Disk	0.0013	0.00	0.0034
		2	WHERE clause non-indexed dataset	Profile	434	Disk	0.1631	0.17	0.0198
				Profile	434	RAM	0.1594	0.16	0.0203
				State	578	Disk	1.2995	1.38	0.2560
		3	WHERE clause indexed dataset	Profile	434	Disk	0.1395	0.14	0.0150
				Profile	434	RAM	0.1399	0.14	0.0152
				State	578	Disk	0.1360	0.14	0.0149
		4	Direct access with KEY= option	Profile	434	Disk	0.1389	0.14	0.0154
				Profile	434	RAM	0.1393	0.14	0.0179
				State	578	Disk	0.1372	0.14	0.0155
		5	Direct access with POINT= option	Profile	434	Disk	0.1394	0.14	0.0155
				Profile	434	RAM	0.1427	0.14	0.0173
				State	578	Disk	0.1409	0.15	0.0159
		6	Subsetting with FIRSTOBS= OBS= options	Profile	434	Disk	0.1415	0.14	0.0155
				Profile	434	RAM	0.1416	0.15	0.0174
				State	578	Disk	0.1380	0.14	0.0155
	2	1	Temporary arrays	Profile	434	Disk	0.0012	0.00	0.0032
				Profile	434	RAM	0.0011	0.00	0.0032
				State	578	Disk	0.0015	0.00	0.0036
		2	WHERE clause non-indexed dataset	Profile	434	Disk	0.1633	0.17	0.0186
				Profile	434	RAM	0.1638	0.17	0.0189
				State	578	Disk	1.2870	1.36	0.2062
		3	WHERE clause indexed dataset	Profile	434	Disk	0.1392	0.14	0.0152
				Profile	434	RAM	0.1416	0.15	0.0151
				State	578	Disk	0.1375	0.14	0.0150
		4	Direct access with KEY= option	Profile	434	Disk	0.1403	0.14	0.0138
				Profile	434	RAM	0.1403	0.14	0.0163
				State	578	Disk	0.1369	0.14	0.0149
		5	Direct access with POINT= option	Profile	434	Disk	0.1435	0.15	0.0147
				Profile	434	RAM	0.1449	0.15	0.0161
				State	578	Disk	0.1410	0.15	0.0154
		6	Subsetting with FIRSTOBS= OBS= options	Profile	434	Disk	0.1412	0.15	0.0157
				Profile	434	RAM	0.1405	0.14	0.0154
				State	578	Disk	0.1400	0.14	0.0157

Table 5. Summary of Search Times (in seconds) for Minnesota for Module 110

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Minnesota	1	1	Temporary arrays	Profile	9,394	Disk	0.0190	0.02	0.0048
				Profile	9,394	RAM	0.0196	0.02	0.0028
				State	56,187	Disk	0.1099	0.11	0.0163
		2	WHERE clause non-indexed Dataset	Profile	9,394	Disk	0.6203	0.68	0.1168
				Profile	9,394	RAM	0.6349	0.69	0.1148
				State	56,187	Disk	1.4641	1.50	0.3713
		3	WHERE clause indexed dataset	Profile	9,394	Disk	0.1729	0.18	0.0194
				Profile	9,394	RAM	0.1712	0.18	0.0191
				State	56,187	Disk	0.3557	0.37	0.0421
		4	Direct access with KEY= option	Profile	9,394	Disk	0.1747	0.18	0.0206
				Profile	9,394	RAM	0.1814	0.19	0.0204
				State	56,187	Disk	0.3752	0.40	0.0480
		5	Direct access with POINT= option	Profile	9,394	Disk	0.1680	0.18	0.0224
				Profile	9,394	RAM	0.1664	0.17	0.0215
				State	56,187	Disk	0.3035	0.32	0.0330
		6	Subsetting with FIRSTOBS= OBS= options	Profile	9,394	Disk	0.1648	0.17	0.0206
				Profile	9,394	RAM	0.1676	0.17	0.0201
				State	56,187	Disk	0.3078	0.33	0.0454
	2	1	Temporary arrays	Profile	9,394	Disk	0.0179	0.02	0.0056
				Profile	9,394	RAM	0.0198	0.02	0.0047
				State	56,187	Disk	0.1056	0.11	0.0204
		2	WHERE clause non-indexed dataset	Profile	9,394	Disk	0.6304	0.69	0.1096
				Profile	9,394	RAM	0.6051	0.66	0.1214
				State	56,187	Disk	1.4015	1.50	0.2361
		3	WHERE clause indexed dataset	Profile	9,394	Disk	0.1682	0.17	0.0214
				Profile	9,394	RAM	0.1718	0.18	0.0201
				State	56,187	Disk	0.3428	0.37	0.0491
		4	Direct access with KEY= option	Profile	9,394	Disk	0.1710	0.17	0.0218
				Profile	9,394	RAM	0.1738	0.17	0.0218
				State	56,187	Disk	0.3669	0.39	0.0493
		5	Direct access with POINT= option	Profile	9,394	Disk	0.1674	0.17	0.0212
				Profile	9,394	RAM	0.1649	0.16	0.0209
				State	56,187	Disk	0.2918	0.31	0.0388
		6	Subsetting with FIRSTOBS= OBS= options	Profile	9,394	Disk	0.1634	0.17	0.0217
				Profile	9,394	RAM	0.1683	0.17	0.0204
				State	56,187	Disk	0.3015	0.32	0.0469

Table 6. Summary of Search Times (in seconds) for Texas for Module 110

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Texas	1	1	Temporary arrays	Profile	122,625	Disk	0.2450	0.25	0.0349
				Profile	122,625	RAM	0.2487	0.25	0.0312
				State	158,341	Disk	0.3197	0.33	0.0359
		2	WHERE clause non-indexed dataset	Profile	122,625	Disk	1.4672	1.53	0.2887
				Profile	122,625	RAM	1.4220	1.53	0.2459
				State	158,341	Disk	1.6078	1.72	0.3269
		3	WHERE clause indexed dataset	Profile	122,625	Disk	0.6072	0.65	0.0872
				Profile	122,625	RAM	0.6183	0.66	0.0806
				State	158,341	Disk	0.7478	0.80	0.1017
		4	Direct access with KEY= option	Profile	122,625	Disk	0.6697	0.71	0.0887
				Profile	122,625	RAM	0.6760	0.72	0.0874
				State	158,341	Disk	0.8165	0.87	0.1046
		5	Direct access with POINT= option	Profile	122,625	Disk	0.5465	0.58	0.0800
				Profile	122,625	RAM	0.5084	0.55	0.0962
				State	158,341	Disk	0.5797	0.63	0.0819
		6	Subsetting with FIRSTOBS= OBS= options	Profile	122,625	Disk	0.5131	0.56	0.0857
				Profile	122,625	RAM	0.5022	0.55	0.0898
				State	158,341	Disk	0.6320	0.68	0.1015
	2	1	Temporary arrays	Profile	122,625	Disk	0.2441	0.25	0.0270
				Profile	122,625	RAM	0.2501	0.25	0.0334
				State	158,341	Disk	0.3124	0.32	0.0408
		2	WHERE clause non-indexed dataset	Profile	122,625	Disk	1.4912	1.54	0.2451
				Profile	122,625	RAM	1.4483	1.52	0.3108
				State	158,341	Disk	1.6691	1.75	0.2344
		3	WHERE clause indexed dataset	Profile	122,625	Disk	0.6003	0.64	0.0848
				Profile	122,625	RAM	0.6094	0.65	0.0840
				State	158,341	Disk	0.7620	0.81	0.0960
		4	Direct access with KEY= option	Profile	122,625	Disk	0.6638	0.70	0.0837
				Profile	122,625	RAM	0.6759	0.72	0.0853
				State	158,341	Disk	0.8256	0.87	0.0959
		5	Direct access with POINT= option	Profile	122,625	Disk	0.5246	0.56	0.0867
				Profile	122,625	RAM	0.5248	0.57	0.0890
				State	158,341	Disk	0.5902	0.63	0.0743
		6	Subsetting with FIRSTOBS= OBS= options	Profile	122,625	Disk	0.5047	0.55	0.0859
				Profile	122,625	RAM	0.5148	0.56	0.0847
				State	158,341	Disk	0.6454	0.69	0.1001

Table 7. Summary of Search Times (in seconds) for All States for Module 320

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
All	1	1	Temporary arrays	Profile	308,783	Disk	0.0083	0.00	0.0140
				Profile	308,783	RAM	0.0087	0.00	0.0139
				State	643,769	Disk	0.0158	0.01	0.0184
		2	WHERE clause non-indexed dataset	Profile	308,783	Disk	0.0705	0.07	0.0307
				Profile	308,783	RAM	0.0687	0.06	0.0298
				State	643,769	Disk	0.1173	0.11	0.0237
		3	WHERE clause indexed dataset	Profile	308,783	Disk	0.0411	0.04	0.0246
				Profile	308,783	RAM	0.0416	0.04	0.0255
				State	643,769	Disk	0.0525	0.05	0.0331
		4	Direct access with KEY= option	Profile	308,783	Disk	0.0430	0.04	0.0278
				Profile	308,783	RAM	0.0427	0.04	0.0277
				State	643,769	Disk	0.0575	0.05	0.0385
		5	Direct access with POINT= Option	Profile	308,783	Disk	0.0395	0.04	0.0175
				Profile	308,783	RAM	0.0371	0.03	0.0162
				State	643,769	Disk	0.0537	0.05	0.0297
		6	Subsetting with FIRSTOBS= OBS= options	Profile	308,783	Disk	0.0374	0.04	0.0164
				Profile	308,783	RAM	0.0386	0.04	0.0165
				State	643,769	Disk	0.0447	0.04	0.0212
	2	1	Temporary arrays	Profile	308,783	Disk	0.0121	0.00	0.0190
				Profile	308,783	RAM	0.0120	0.00	0.0200
				State	643,769	Disk	0.0227	0.02	0.0262
		2	WHERE clause non-indexed dataset	Profile	308,783	Disk	0.0823	0.08	0.0358
				Profile	308,783	RAM	0.0829	0.08	0.0357
				State	643,769	Disk	0.1391	0.14	0.0313
		3	WHERE clause indexed dataset	Profile	308,783	Disk	0.0481	0.04	0.0286
				Profile	308,783	RAM	0.0499	0.04	0.0315
				State	643,769	Disk	0.0631	0.05	0.0406
		4	Direct access with KEY= option	Profile	308,783	Disk	0.0511	0.04	0.0344
				Profile	308,783	RAM	0.0527	0.04	0.0352
				State	643,769	Disk	0.0675	0.06	0.0468
		5	Direct access with POINT= option	Profile	308,783	Disk	0.0489	0.05	0.0231
				Profile	308,783	RAM	0.0472	0.04	0.0221
				State	643,769	Disk	0.0547	0.05	0.0294
		6	Subsetting with FIRSTOBS= OBS= options	Profile	308,783	Disk	0.0451	0.04	0.0202
				Profile	308,783	RAM	0.0441	0.04	0.0199
				State	643,769	Disk	0.0515	0.05	0.0255

Table 8. Summary of Search Times (in seconds) for Alaska for Module 320

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Alaska	1	1	Temporary arrays	Profile	62	Disk	0.0014	0.00	0.0009
				Profile	62	RAM	0.0019	0.00	0.0008
				State	99	Disk	0.0014	0.00	0.0008
		2	WHERE clause non-indexed dataset	Profile	62	Disk	0.0314	0.03	0.0077
				Profile	62	RAM	0.0301	0.03	0.0075
				State	99	Disk	0.1014	0.10	0.0123
		3	WHERE clause indexed dataset	Profile	62	Disk	0.0295	0.03	0.0075
				Profile	62	RAM	0.0291	0.03	0.0076
				State	99	Disk	0.0271	0.03	0.0073
		4	Direct access with KEY= option	Profile	62	Disk	0.0290	0.03	0.0075
				Profile	62	RAM	0.0289	0.03	0.0075
				State	99	Disk	0.0265	0.03	0.0073
		5	Direct access with point= option	Profile	62	Disk	0.0316	0.03	0.0077
				Profile	62	RAM	0.0317	0.03	0.0077
				State	99	Disk	0.0321	0.03	0.0093
		6	Subsetting with FIRSTOBS= OBS= options	Profile	62	Disk	0.0308	0.03	0.0076
				Profile	62	RAM	0.0307	0.03	0.0076
				State	99	Disk	0.0261	0.03	0.0069
	2	1	Temporary arrays	Profile	62	Disk	0.0021	0.00	0.0010
				Profile	62	RAM	0.0021	0.00	0.0010
				State	99	Disk	0.0020	0.00	0.0011
		2	WHERE clause non-indexed dataset	Profile	62	Disk	0.0359	0.04	0.0089
				Profile	62	RAM	0.0371	0.04	0.0089
				State	99	Disk	0.1191	0.12	0.0166
		3	WHERE clause indexed dataset	Profile	62	Disk	0.0353	0.04	0.0089
				Profile	62	RAM	0.0340	0.03	0.0089
				State	99	Disk	0.0319	0.03	0.0084
		4	Direct access with KEY= option	Profile	62	Disk	0.0330	0.03	0.0083
				Profile	62	RAM	0.0349	0.04	0.0085
				State	99	Disk	0.0327	0.03	0.0082
		5	Direct access with POINT= option	Profile	62	Disk	0.0386	0.04	0.0089
				Profile	62	RAM	0.0379	0.04	0.0088
				State	99	Disk	0.0331	0.03	0.0086
		6	Subsetting with FIRSTOBS= OBS= options	Profile	62	Disk	0.0364	0.04	0.0090
				Profile	62	RAM	0.0360	0.04	0.0088
				State	99	Disk	0.0350	0.04	0.0081

Table 9. Summary of Search Times (in seconds) for Minnesota for Module 320

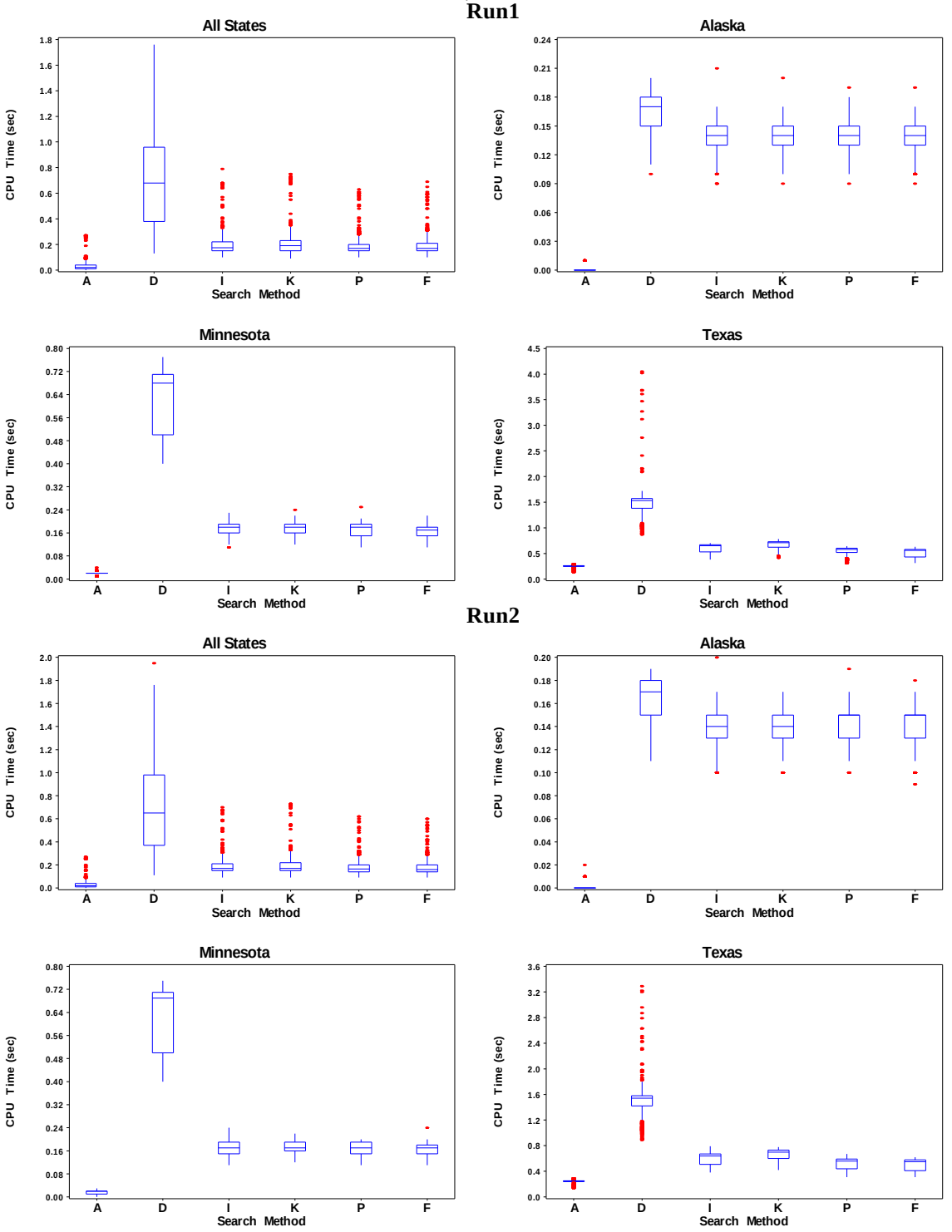
State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Minnesota	1	1	Temporary arrays	Profile	3,256	Disk	0.0056	0.00	0.0049
				Profile	3,256	RAM	0.0057	0.00	0.0049
				State	17,075	Disk	0.0219	0.02	0.0059
		2	WHERE clause non-indexed dataset	Profile	3,256	Disk	0.0607	0.06	0.0092
				Profile	3,256	RAM	0.0592	0.06	0.0090
				State	17,075	Disk	0.1203	0.12	0.0142
		3	WHERE clause indexed dataset	Profile	3,256	Disk	0.0367	0.04	0.0081
				Profile	3,256	RAM	0.0362	0.04	0.0080
				State	17,075	Disk	0.0619	0.06	0.0101
		4	Direct access with KEY= option	Profile	3,256	Disk	0.0370	0.04	0.0080
				Profile	3,256	RAM	0.0368	0.04	0.0079
				State	17,075	Disk	0.0655	0.06	0.0111
		5	Direct access with POINT= option	Profile	3,256	Disk	0.0379	0.04	0.0080
				Profile	3,256	RAM	0.0377	0.04	0.0080
				State	17,075	Disk	0.0590	0.06	0.0140
		6	Subsetting with FIRSTOBS= OBS= options	Profile	3,256	Disk	0.0373	0.04	0.0078
				Profile	3,256	RAM	0.0363	0.04	0.0081
				State	17,075	Disk	0.0496	0.05	0.0090
	2	1	Temporary arrays	Profile	3,256	Disk	0.0075	0.01	0.0050
				Profile	3,256	RAM	0.0071	0.01	0.0050
				State	17,075	Disk	0.0305	0.03	0.0058
		2	WHERE clause non-indexed dataset	Profile	3,256	Disk	0.0711	0.07	0.0111
				Profile	3,256	RAM	0.0720	0.07	0.0100
				State	17,075	Disk	0.1420	0.15	0.0207
		3	WHERE clause indexed dataset	Profile	3,256	Disk	0.0417	0.04	0.0093
				Profile	3,256	RAM	0.0422	0.04	0.0099
				State	17,075	Disk	0.0759	0.08	0.0141
		4	Direct access with KEY= option	Profile	3,256	Disk	0.0435	0.04	0.0098
				Profile	3,256	RAM	0.0440	0.04	0.0098
				State	17,075	Disk	0.0800	0.08	0.0148
		5	Direct access with POINT= option	Profile	3,256	Disk	0.0454	0.05	0.0095
				Profile	3,256	RAM	0.0443	0.05	0.0098
				State	17,075	Disk	0.0638	0.07	0.0133
		6	Subsetting with FIRSTOBS= OBS= options	Profile	3,256	Disk	0.0423	0.04	0.0090
				Profile	3,256	RAM	0.0419	0.04	0.0094
				State	17,075	Disk	0.0598	0.06	0.0129

Table 10. Summary of Search Times (in seconds) for Texas for Module 320

State	Run	Method Number	Method Name	Search Type	Records	Data Location	Mean CPU Time	Median CPU Time	Standard Deviation CPU Time
Texas	1	1	Temporary arrays	Profile	77,471	Disk	0.0950	0.09	0.0122
				Profile	77,471	RAM	0.0844	0.08	0.0118
				State	99,971	Disk	0.1221	0.12	0.0175
		2	WHERE clause non-indexed Dataset	Profile	77,471	Disk	0.1936	0.19	0.0193
				Profile	77,471	RAM	0.1961	0.20	0.0193
				State	99,971	Disk	0.2159	0.21	0.0232
		3	WHERE clause indexed dataset	Profile	77,471	Disk	0.1950	0.20	0.0214
				Profile	77,471	RAM	0.1946	0.20	0.0211
				State	99,971	Disk	0.2310	0.23	0.0273
		4	Direct access with KEY= option	Profile	77,471	Disk	0.2218	0.23	0.0240
				Profile	77,471	RAM	0.2193	0.22	0.0243
				State	99,971	Disk	0.2657	0.27	0.0332
		5	Direct access with POINT= option	Profile	77,471	Disk	0.1387	0.14	0.0177
				Profile	77,471	RAM	0.1408	0.14	0.0182
				State	99,971	Disk	0.1985	0.22	0.0452
		6	Subsetting with FIRSTOBS= OBS= options	Profile	77,471	Disk	0.1364	0.14	0.0162
				Profile	77,471	RAM	0.1336	0.13	0.0164
				State	99,971	Disk	0.1579	0.16	0.0180
	2	1	Temporary arrays	Profile	77,471	Disk	0.1290	0.13	0.0153
				Profile	77,471	RAM	0.1289	0.13	0.0146
				State	99,971	Disk	0.1626	0.17	0.0248
		2	WHERE clause non-indexed dataset	Profile	77,471	Disk	0.2255	0.24	0.0333
				Profile	77,471	RAM	0.2236	0.24	0.0327
				State	99,971	Disk	0.2716	0.29	0.0389
		3	WHERE clause indexed dataset	Profile	77,471	Disk	0.2299	0.25	0.0378
				Profile	77,471	RAM	0.2266	0.25	0.0374
				State	99,971	Disk	0.2871	0.31	0.0431
		4	Direct access with KEY= option	Profile	77,471	Disk	0.2597	0.28	0.0381
				Profile	77,471	RAM	0.2623	0.28	0.0388
				State	99,971	Disk	0.3232	0.35	0.0497
		5	Direct access with POINT= option	Profile	77,471	Disk	0.1624	0.18	0.0341
				Profile	77,471	RAM	0.1683	0.19	0.0339
				State	99,971	Disk	0.2078	0.23	0.0446
		6	Subsetting with FIRSTOBS= OBS= options	Profile	77,471	Disk	0.1574	0.17	0.0277
				Profile	77,471	RAM	0.1570	0.17	0.0297
				State	99,971	Disk	0.1962	0.21	0.0318

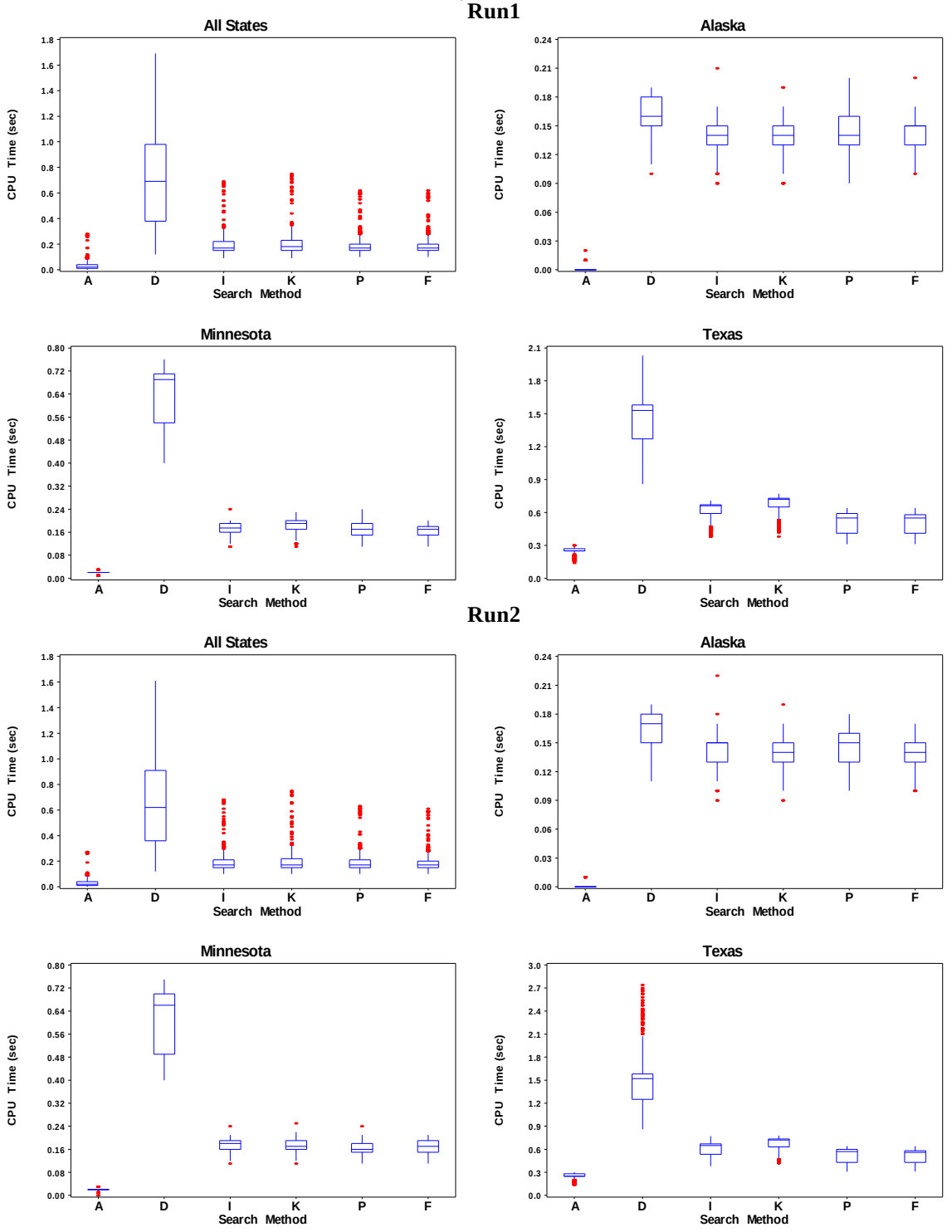
APPENDIX C. BOX PLOTS OF SEARCH METHODS

Figure 3. Comparison of Search Methods for Module 110
Profile Search, Data on Disk



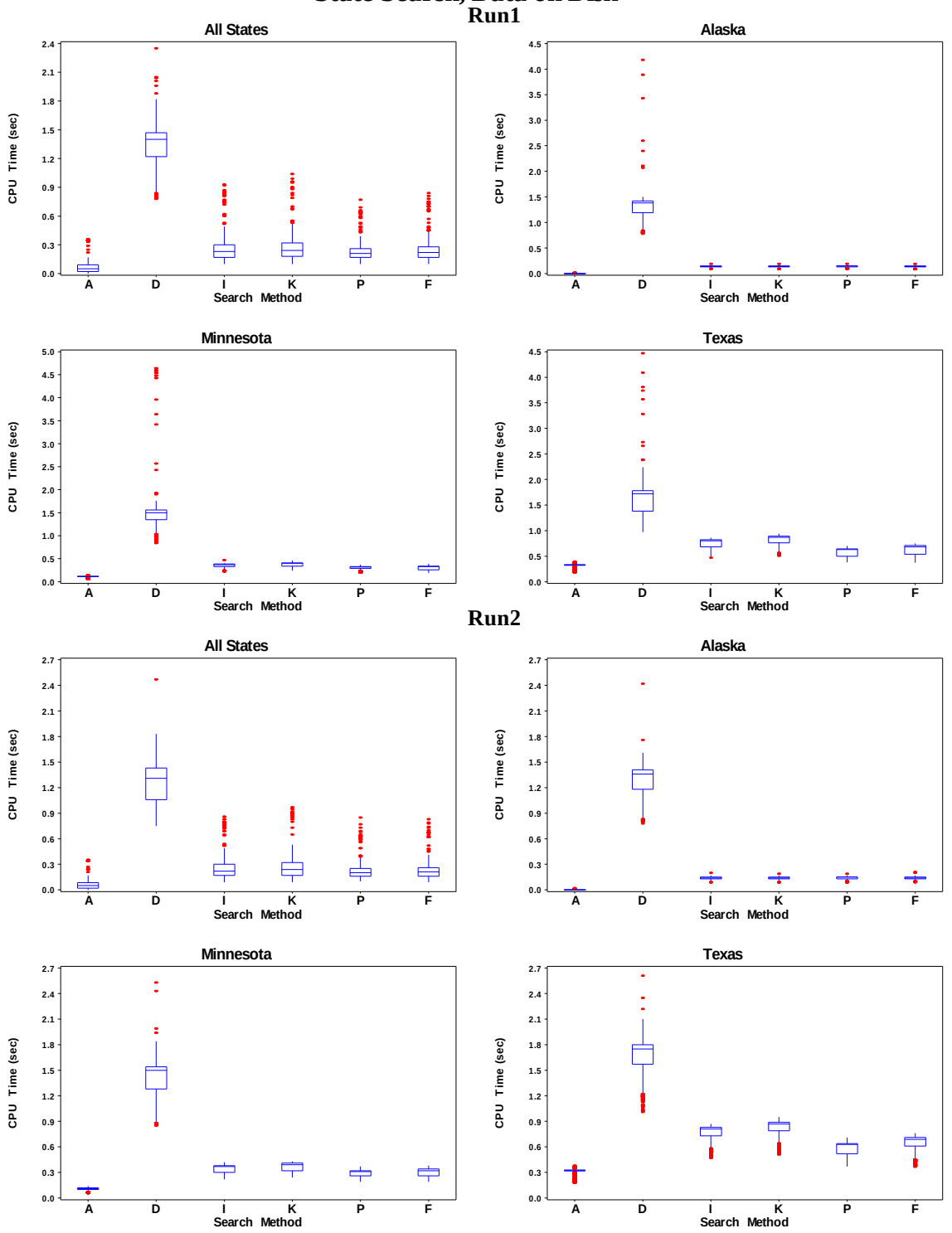
A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

Figure 4. Comparison of Search Methods for Module 110
Profile Search, Data on RAM



A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

Figure 5. Comparison of Search Methods for Module 110
State Search, Data on Disk

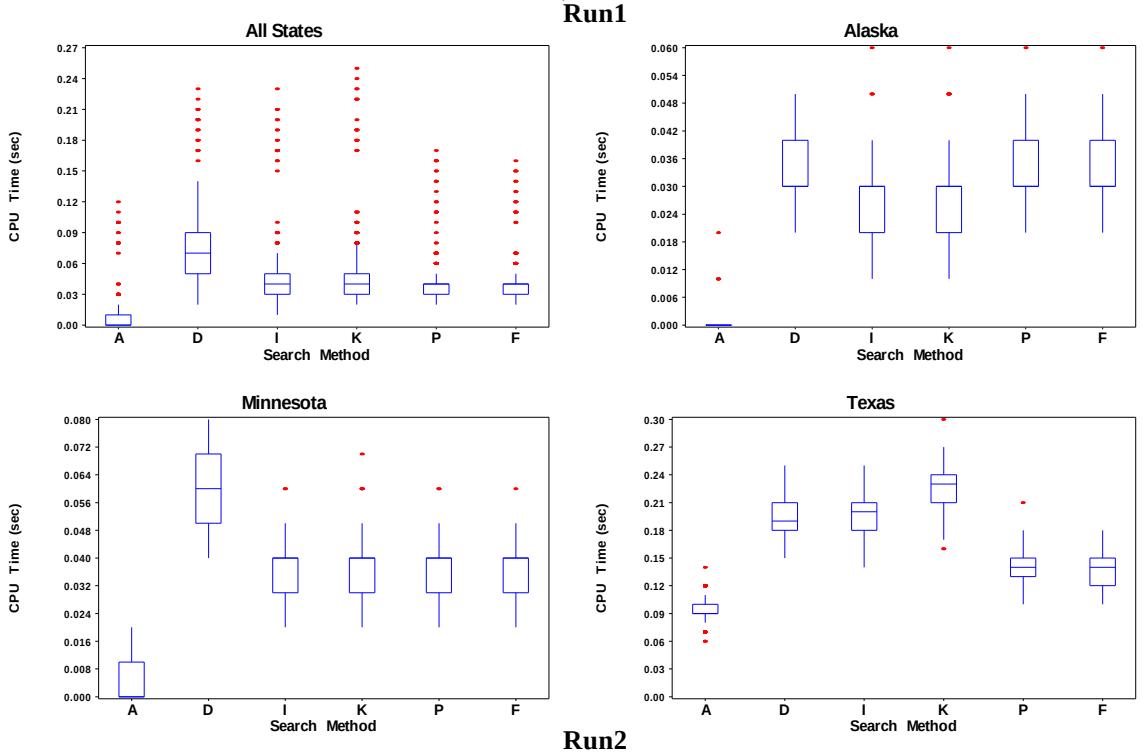


A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

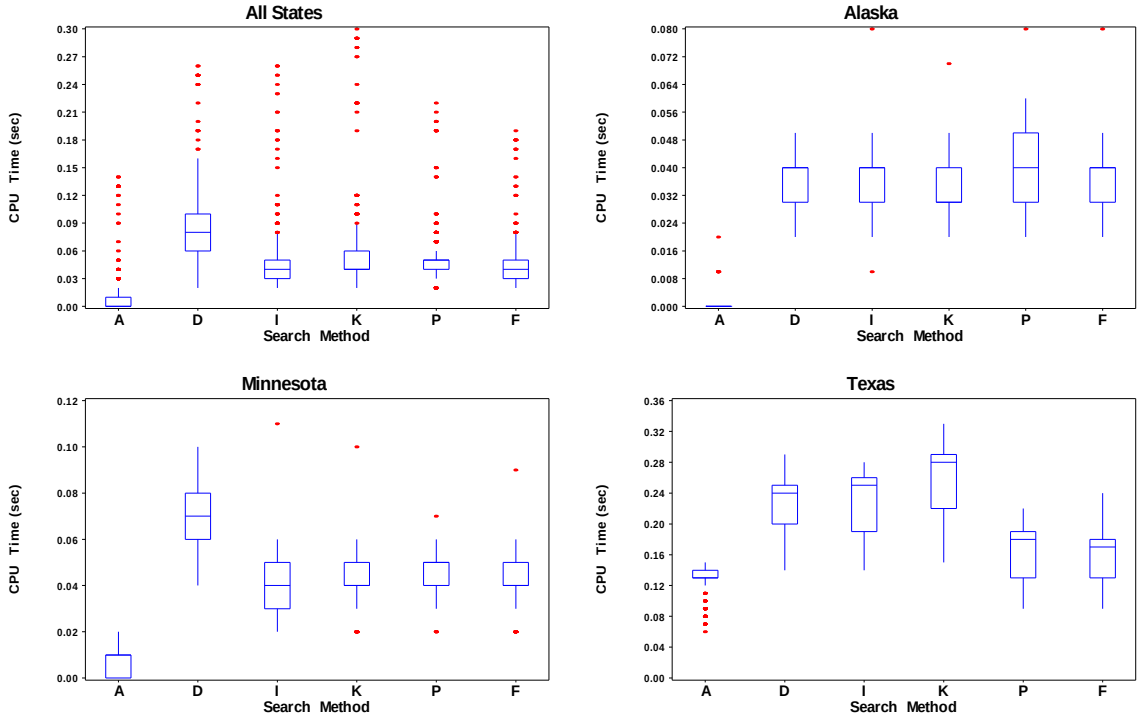
Figure 6. Comparison of Search Methods for Module 320

Profile Search, Data on Disk

Run1



Run2

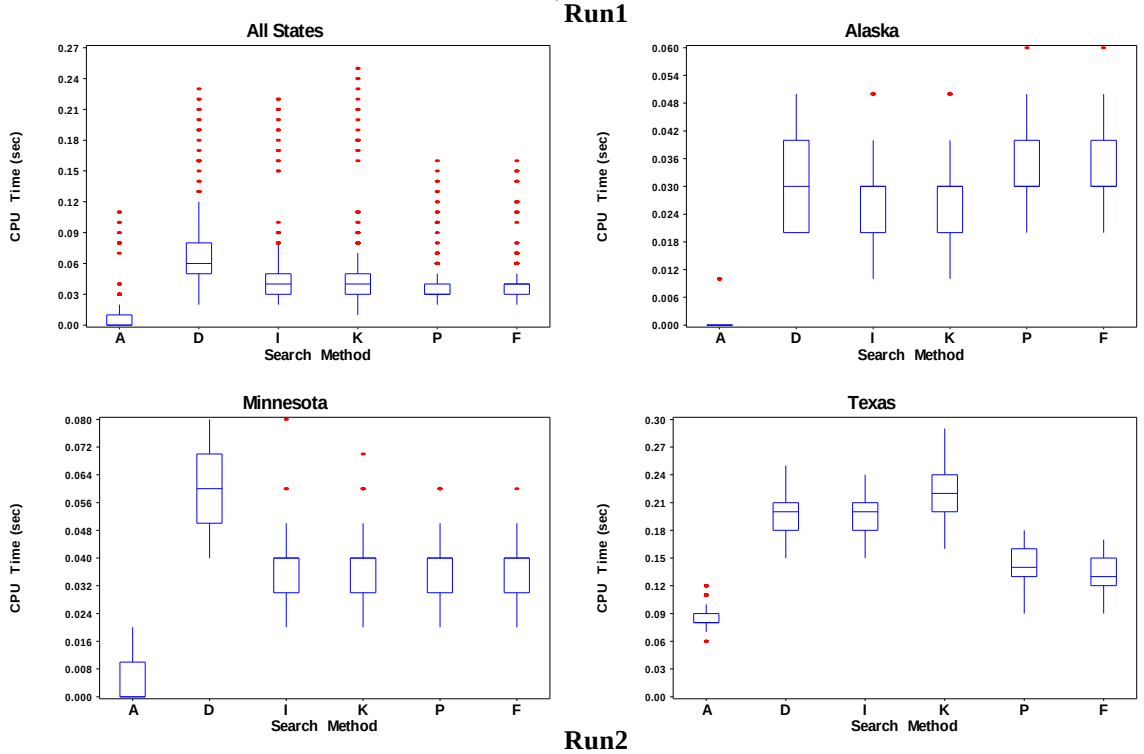


A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

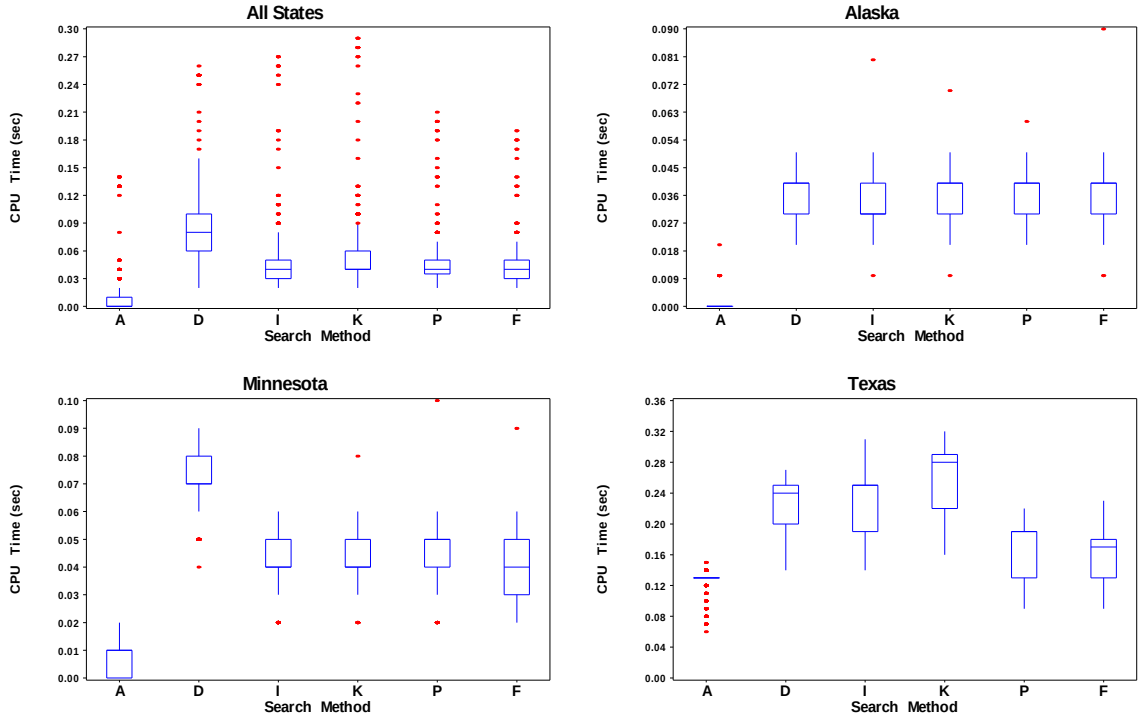
Figure 7. Comparison of Search Methods for Module 320

Profile Search, Data on RAM

Run1



Run2

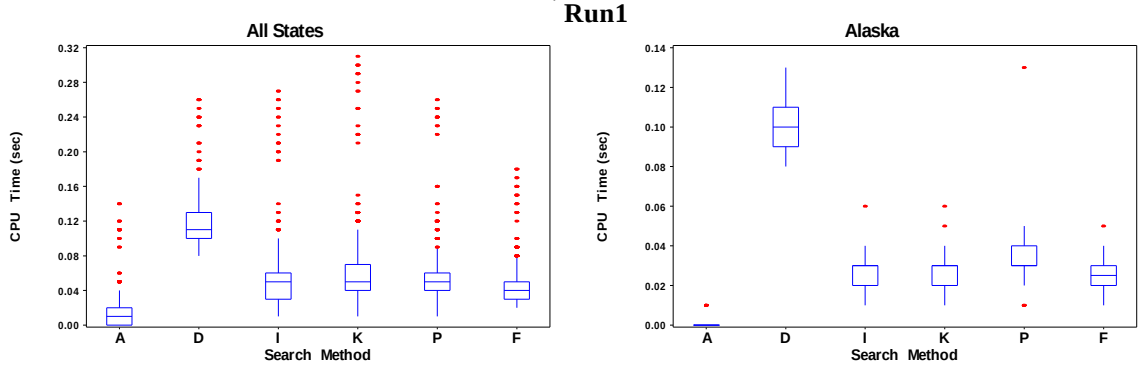


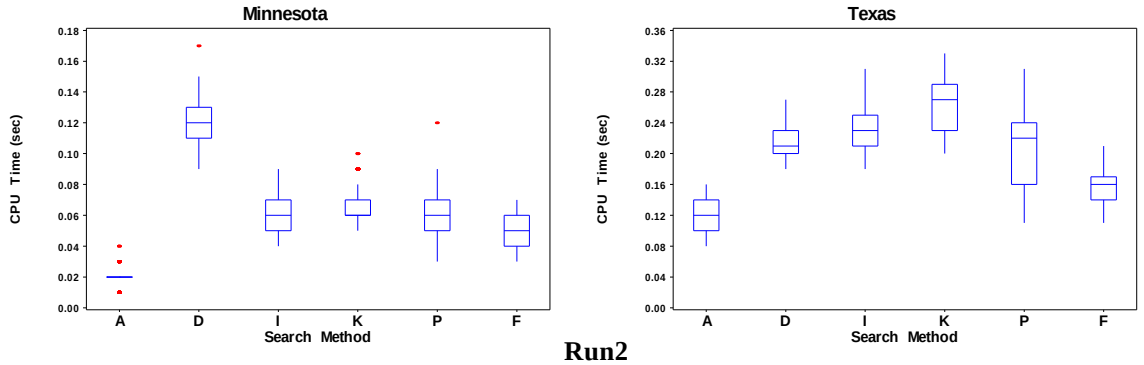
A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

Figure 8. Comparison of Search Methods for Module 320

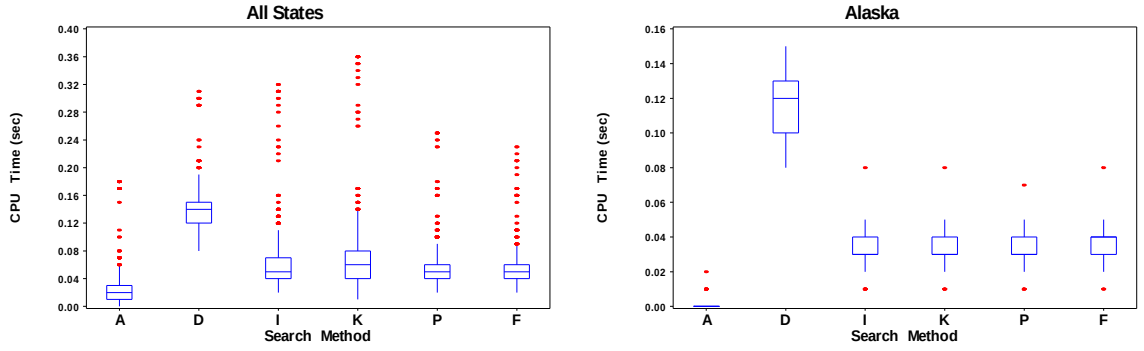
State Search, Data on Disk

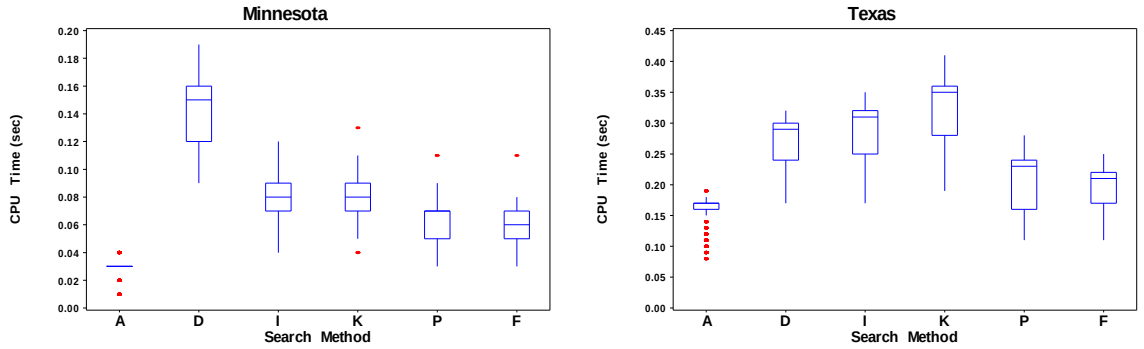
Run1





Run2





A=Temporary arrays
D=Where clause non-indexed dataset
I=Where clause indexed dataset
K=Direct access with key= option
P=Direct access with point= option
F=Subsetting with firstobs= obs= options

APPENDIX D. PLOTS OF SEARCH METHODS

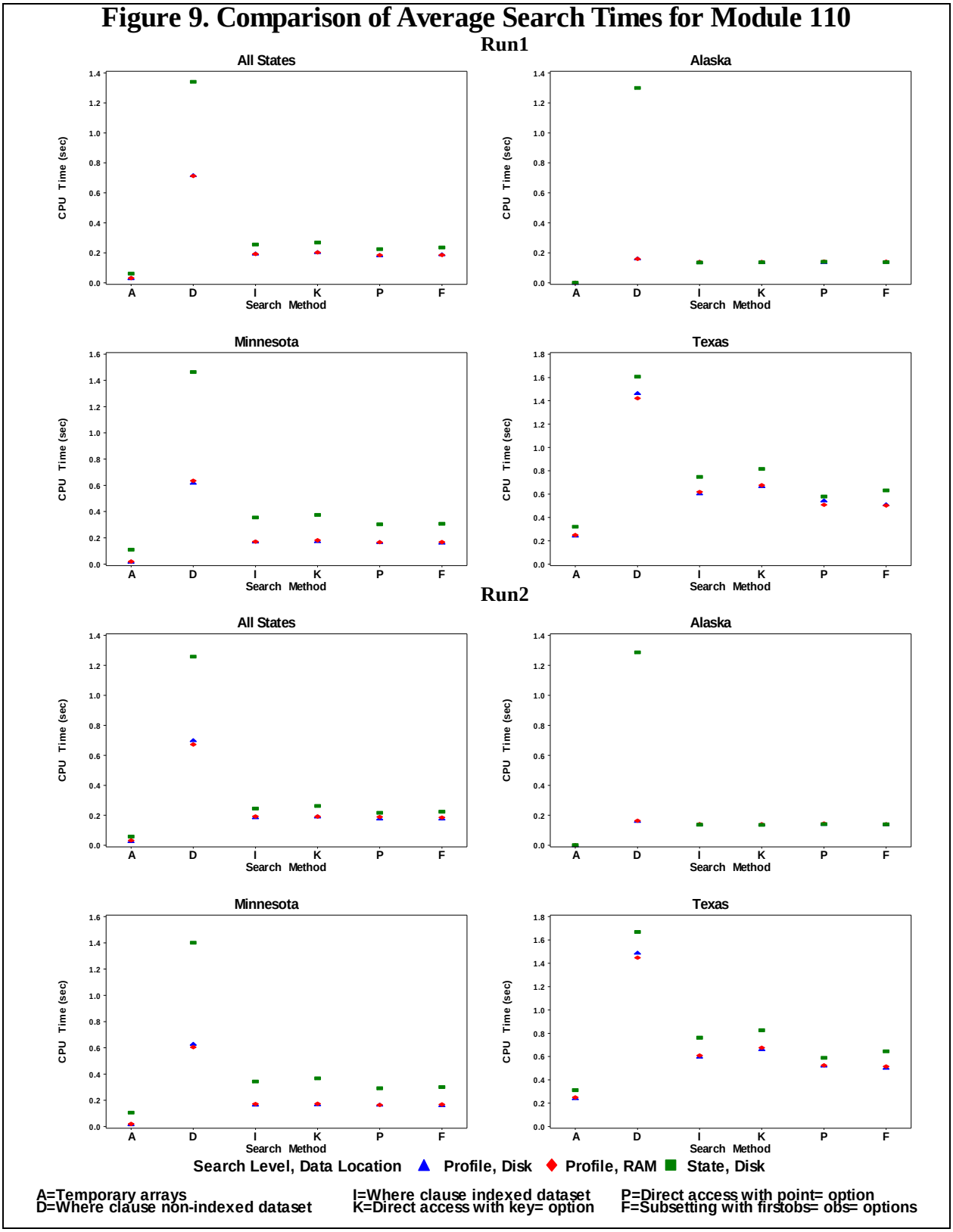


Figure 10. Comparison of Average Search Times for Module 320