

SPACE-SAVING TOOLS: COMPRESSION IN SAS® 6.07

Philip Bardsley
Carolina Population Center
The University of North Carolina at Chapel Hill

Abstract

It seems inevitable that no matter how much disk space we start with for our work, we always need more. With disk space at a premium, we need to take advantage of ways to store more data in less room. This paper investigates the pros and cons of space-saving tools that are available on three different computing platforms: an IBM mainframe running MVS, an IBM RISC System/6000 workstation running AIX, and a microcomputer running DOS. The SAS tools investigated are the *compress* option, which is available in release 6.07 running on MVS and AIX, and the SAS *length* command, which is available on all platforms. The tools investigated outside SAS are the UNIX *compress* command and the shareware program PKZIP for DOS machines. The data characteristics investigated are observation length, variable type (character/numeric), and data content (floating point real/integer/missing). The advantage measured is disk-space savings; the disadvantages considered include the added cpu time necessary to compress and uncompress, and ease of use. Interactions among all these factors are complex, often having unexpected effects on disk space and cpu time. These results are summarized, and recommendations are offered for simple techniques that can reduce data storage needs substantially.

Introduction

A number of tools are available for data compression, both in SAS and outside. SAS offers the *compress* option in release 6.07 on the MVS and AIX platforms. This option can be specified either globally (*options compress=yes*) so that it applies to all datasets formed during the SAS session (including temporary files written to libref WORK), or as a data set option so that it applies only to selected data sets. The *compress* option reduces data storage space by replacing repetitive data strings with shorter tokens. These tokens are translated back into their original strings when SAS uncompresses the data set. SAS also offers the *length* statement in all releases within the data step. This statement makes it

possible to assign a minimum length to each variable individually that will handle the largest value found for that variable in any observation in the data set.

On DOS platforms, many data compression tools are available from a variety of software vendors. These can be grouped into those that run automatically in the background, compressing and uncompressing data on the fly, and those that require the user to issue a command to compress or uncompress one or more data sets. On UNIX platforms an operating system *compress* command is available, which can be used on SAS data sets as well as on other binary and raw data forms. Thus it is possible to compress a SAS data set using the SAS *compress* option and/or *length* statement, and then compress it further using the UNIX *compress* command.

Data characteristics affect the operation of all compression tools, both within SAS and outside. The characteristics most important are the length of each observation, the data type (whether character or numeric), and the contents of the variables (whether floating-point real numbers, integers, or SAS missing values). (Graphics catalogues also affect compression differently from data sets; however, this paper focuses on data.)

Data compression methods have both advantages and disadvantages. The chief advantage, of course, is saving disk space. But this comes at some cost, and the costs vary depending on the tool used. The main cost of the SAS *compress* option is cpu time, which can be substantial. The SAS *length* statement requires some programmer time investigating and coding individual lengths for each variable. The tools available outside SAS also require some cpu time, but generally are simple to use and can be incorporated into a SAS job fairly easily.

Methods

To investigate all these aspects of data compression, I set up test SAS data sets, generated values using the *ranuni*

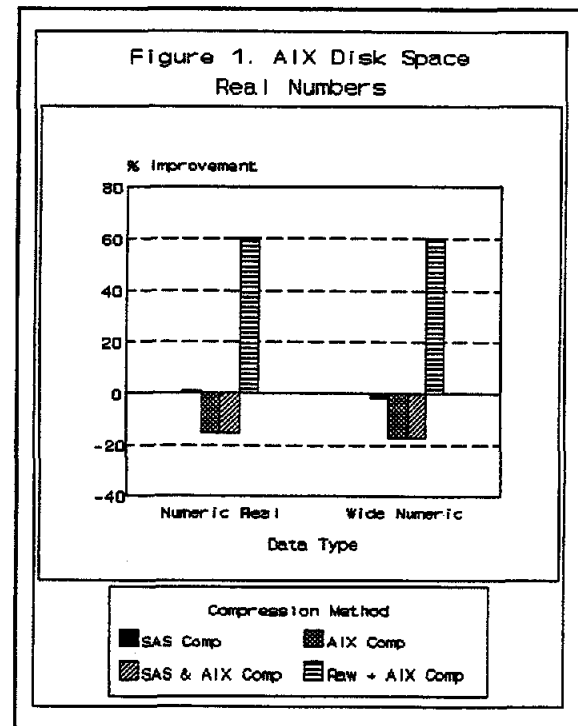
function, and ran benchmarks on three platforms: SAS 6.07 running on an IBM 3090 mainframe under MVS, SAS 6.07 running on an IBM RISC System/6000 workstation under AIX, and SAS 6.04 running on an IBM clone 386-25 under DOS. The test data sets took the following forms: Narrow (20 variables) or wide (120 variables); all character data or all numeric; and all floating-point real numbers, all 1-digit integers, or all SAS missing values. These data sets are not intended to be realistic. Rather, these forms were chosen to focus on the interaction between data characteristics and compression tools on the one hand, and space reduction and cpu usage on the other. By varying one data characteristic at a time it was possible to isolate the effect each has on disk space and cpu time, and to look at its interaction with other data characteristics.

This study's focus does not involve a comparison *across* platforms, such as might be found in a conventional benchmark study of similar competing hardware or software products. In such studies it is important to hold both the number of observations and the observation length constant across competing environments. Instead, the focus here is, given a platform, what compression tools are available, and which ones work best with which types of data. Thus the number of observations was *not* held constant across platforms. Instead, the number of observations used on each platform was chosen for convenience - enough to give a measurable variation in cpu usage and disk space, but not so many that I would become frustrated waiting for a test to complete.

The tools chosen were: on MVS the SAS compress option and SAS length statement, on AIX the UNIX compress command and the SAS compress option and length statement, and on DOS the shareware package PKZIP from PKWare and the SAS length statement. (The SAS compress option was not available on version 6.04 under DOS. It is available on later versions running under OS/2 and Windows. I hope to test it and present the results in this tutorial at the SESUG meeting). PKZIP was selected on DOS because it is heavily used, it received a favorable review in a recent *PC Magazine* issue, and it is relatively inexpensive. PKZIP is one of many programs that compresses only at the user's command. Another group of DOS programs, including products such as Stacker, work automatically in the background on all data sets, compressing and uncompressing "on the fly." While these products are much more convenient, they are relatively more costly than PKZIP, and I did not have access to them for testing.

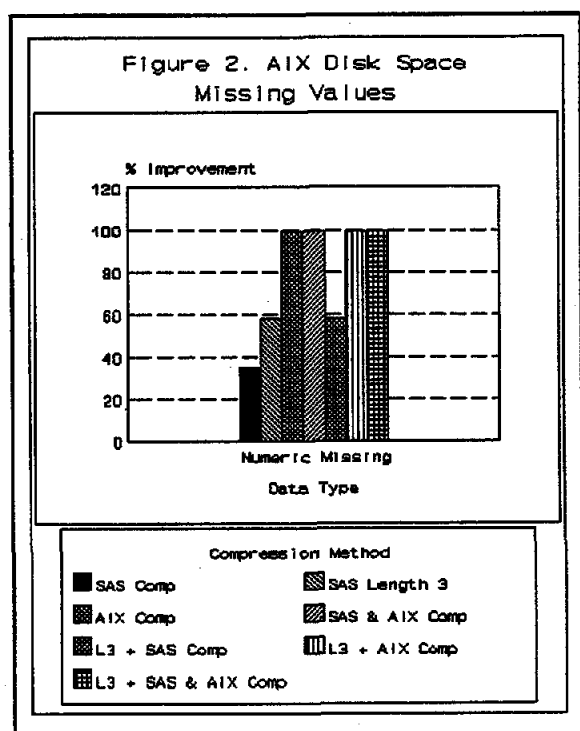
Results - Disk Space Reduction

AIX. The SAS compress option failed to squeeze any space out of a data set full of randomly generated floating-point real numbers (see Figure 1). This was



true of both narrow and wide data sets. This is because randomly generated data have few repeating strings, and when stored in binary form, these values leave no space unused. Because real numbers occupy all significant digits available in the SAS maximum length for numeric data (length 8 gives 16 significant digits), using the SAS length statement to reduce storage would result in a loss of precision and was not tested. The UNIX compress command also would not compress the real numbers. In fact, it refused to try until forced with the -f switch, and the resulting data set was actually larger than its uncompressed original. Real numbers stored at full precision, then, appear to be uncompressible.

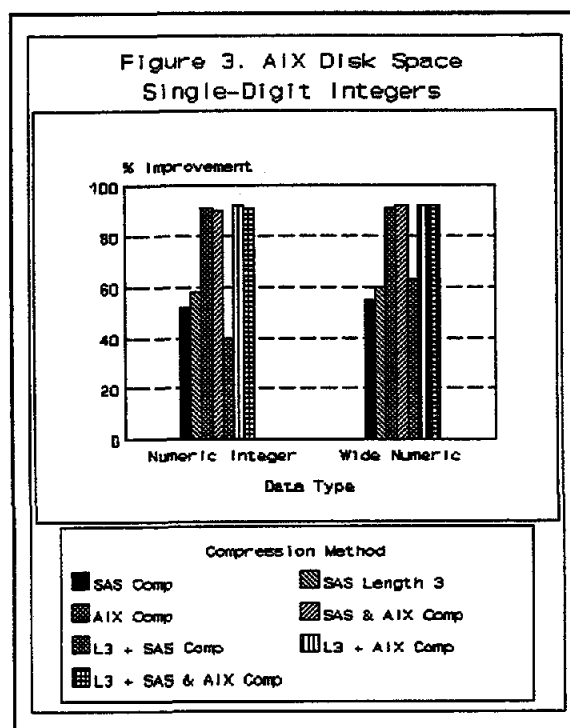
SAS numeric missing values, on the other hand, are astonishingly compressible. The AIX compress command yielded 99% compression (i.e., the compressed data set was only 1% the size of its uncompressed form), whether used alone or with the SAS compress option or length statement, or both (see Figure 2). The SAS length statement by itself was the next most effective, yielding 58% compression. The



SAS compress option was the least useful, yielding only 35% compression of variables with length 8, and no additional compression of variables with length 3. From this it seems that data sets containing a large percentage of missing values should be highly compressible, but the AIX compress command is much more effective than either the SAS compress command or length statement.

Single-digit integer values stored in numeric variables also are highly compressible, about 90%, using the AIX compress command (see Figure 3). Using the SAS length of 3 compressed 58% relative to length 8. The SAS compress option, however, gave only 50% compression on variables stored at length 8, and 6% on length 3. The same results occurred when the observation length was increased from 20 variables to 120 variables, with one exception: on longer observations, the SAS compress option and length statements combined to achieve 63% compression (compared with 40% compression of short observations). This difference may be due to the 12-byte overhead per observation that SAS adds to compressed files to manage the compression. The overhead constituted 20% of the short observations, but only 3% of the long observations, so the corresponding compression ratios were substantially different.

Storing small values in character, rather than numeric

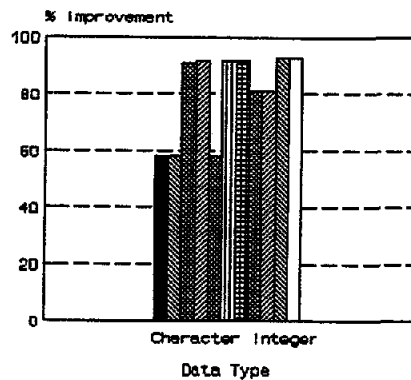


variables offers more space-saving flexibility. This is because the minimum length for character variables is 1, not 3. In the present case, using the narrow test file, length 3 compressed the data by 58% compared with the default length of 8, while length 1 compressed the data by 81% compared with length 8 (see Figure 4). The SAS compress option was unable to compress the file at either length 1 or 3, but it compressed the length 8 file by 58%. The AIX compress command achieved a consistent 91-93% compression in all tests. Combining the SAS compress option with the AIX compress command provided no additional space savings.

The SAS compress option managed a substantial 90% compression of missing character values, much better than it did with missing numeric data (see Figure 5). This is because character missing values are stored as blanks, which are more compressible than the SAS system missing value used for numeric variables. The AIX compress command, at 99%, did better with missing character data than it did with missing numeric. The lesson here is that, whenever possible, 1- or 2-digit values should be stored as character data using the minimum length necessary, and the data set should be compressed using the AIX compress command.

But, what about high-precision numeric data? Neither

Figure 4. AIX Disk Space
Character Integers



substantially more room to store than did the SAS data set, but the AIX compress command reduced it dramatically, to 58% relative to the SAS data set. The main disadvantage of this approach, of course, is that you lose the convenience and processing efficiency of SAS data libraries. However, the raw file does not need to be uncompressed completely before SAS can read it. Instead, you can use the following technique that was circulated on SAS-L recently by Stephen Cha (attributed to Don Olivier):

filename in pipe 'zcat rawfile.Z';

The zcat command uncompresses the raw data file a little at a time, and the pipe option routes it to filename in. Thus, the file need never reside in its complete, uncompressed form on disk. (You can do the opposite when writing the raw data file as well.) Note that this scheme does not work with SAS data libraries, only with raw data.

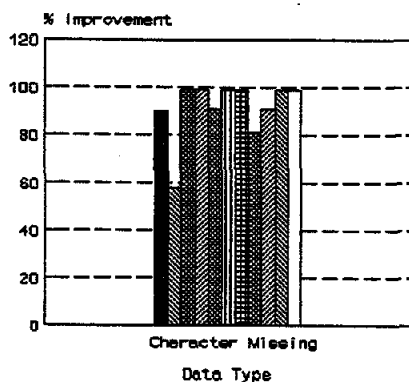
DOS. PKZIP has two modes: fast, which it also calls "shrinking," and maximum, which it also calls "imploding." In all tests I tried both modes. Almost without exception, the fast method resulted in at least as much if not more compression than the maximum method. Fast always was much faster than maximum as well. Therefore, only the results of fast are reported here.

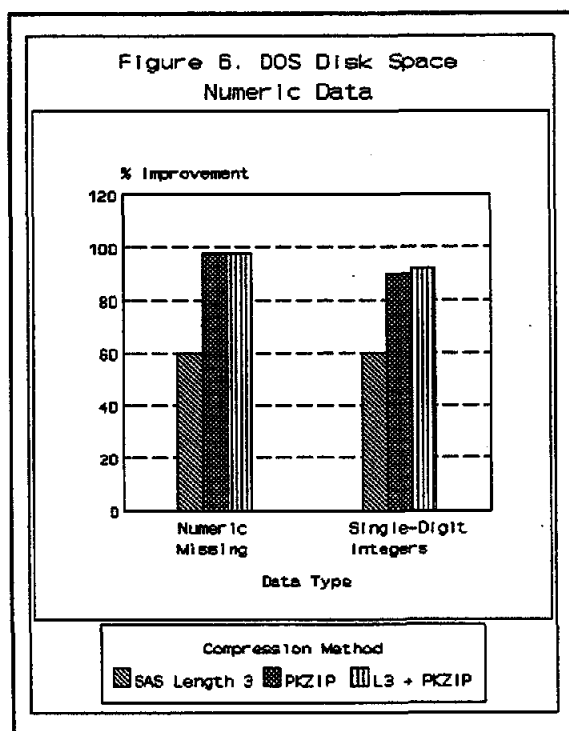
As on AIX, PKZIP was unable to compress a SAS data set containing high-precision real numbers. Again, raw data was explored as an alternative. With both narrow and wide files the raw data set was substantially larger than its SAS counterpart. It was compressible, but the resulting file was about the same size as the SAS file, so no space was saved. It appears, then, that nothing can be done to reduce the storage requirements of high-precision data.

Missing values stored in length 8, though, were highly compressible with the file dropping in size by 98% (see Figure 6). The SAS length statement saved 60% by storing data at length 3 rather than length 8. Combining the length statement with PKZIP resulted in about the same savings as PKZIP alone. The lesson here, again, is that sparse numeric data are highly compressible. If you have much missing data you can expect substantial space savings using software like PKZIP.

SAS nor UNIX can compress it. One possibility is to write the data in raw form, using a format that will preserve the level of precision kept by SAS. Using the narrow test file as an example, the raw data required

Figure 5. AIX Disk Space
Character Missing





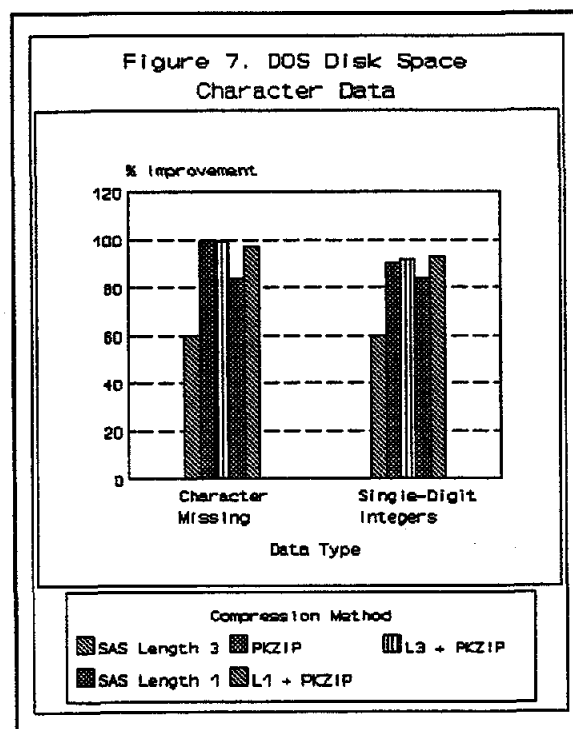
values (see Figure 6). PKZIP reduced storage by about 90%, while the SAS length statement offered savings of 60%. Together they yielded 92% savings. I tested both the narrow and wide data set shapes, but the results differed by at most 1%.

The results for single-digit integers stored in character variables were identical to the numeric variable results (see Figure 7). PKZIP achieved 90% reduction in size, and length 3 gave 60% savings compared with length 8. Storage at length 1 gave savings of 84% compared with length 8, and combining PKZIP with length 1 gave 93% savings.

It is apparent from Figure 7 that short integer values are highly compressible, more so using the PKZIP software than using the length statement, but both yield substantial savings.

Character missing values are even more compressible, showing 99% compression for all variable lengths (see Figure 7). The length statement again gave 60% savings at length 3 and 84% at length 1, compared with length 8.

MVS. As expected from the other operating systems, the SAS compress option was unable to compress high-precision real numbers on MVS (see Figure 8). Leaving the data in raw form again resulted in data sets much

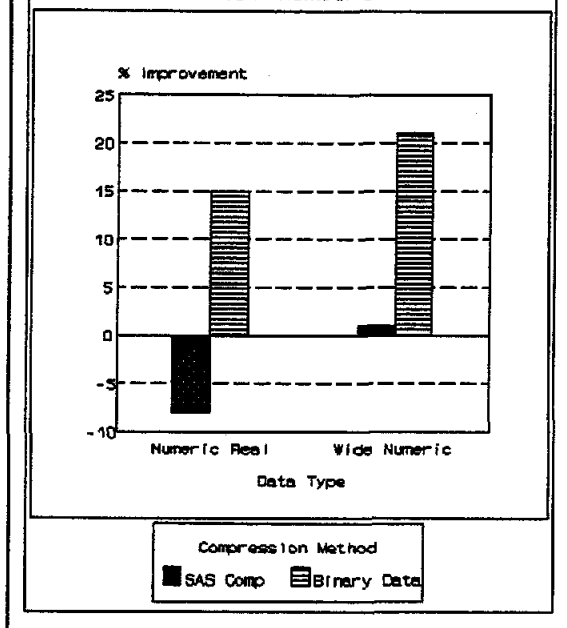


larger than their SAS counterparts, and MVS does not offer any system compression software. So, I examined the option of storing the data in binary form. Writing the file using the SAS format rb8. uses the maximum precision required (16 digits), and results in some disk space savings, 15% for the narrow file, and 21% for the wide file. Presumably, the percent savings increase as the observation length and/or data set size increases.

Compared with raw data, binary data has the advantage of being a format that SAS can read about three times more efficiently. Thus conversion from binary data to a SAS data set requires very little additional cpu time. For example, adding a DATA step onto a PROC MEANS added about 75% to the processing time (one second for the narrow file, 5 seconds for the wide file). Binary data storage, then, seems to be a viable space-savings method.

The SAS compress option reduced numeric missing values by 55% (see Figure 9). Using the minimum length value (2 under MVS) resulted in 68% savings. Compressing the data set with length 2 resulted in only 61% savings, less than the length statement alone achieved. This is again because of the overhead that SAS adds to a data set when it compresses.

SAS compressed single-digit numeric integers by 47%,

Figure 8. MVS Disk Space
Real Numbers

slightly less than it compressed numeric missing values (see Figure 9). Otherwise, numeric integer data behaved the same - 68% savings with the length statement, and 61% savings with both length and compression.

Single digits stored as character data at the default length 8 were compressible to 49%, similar to numeric data (see Figure 10). Shortening the variable length to 2 had the expected result as well: 68% savings alone, 61% when combined with the compress option. Shortening the length further to 1 resulted in 78% savings alone, 73% in conjunction with compression.

Single digit missing stored as character was more highly compressible, by 83% (see Figure 10). Using length 2 gave less attractive results - only 68% relative to length 8, and 83% in addition to compression. Length 1 missing values were 78% smaller than length 8, but compression again yielded 83% savings.

In sum, sparse numeric data and small numeric values appear to benefit most from the judicious use of the length statement. However, the SAS compress option also reduces their storage needs substantially. Using character variables only proves beneficial if the data are single digits (present or missing) so that you can use the minimum variable length of 1. Otherwise, numeric and character data behave the same with both the SAS compression option and the length statement.

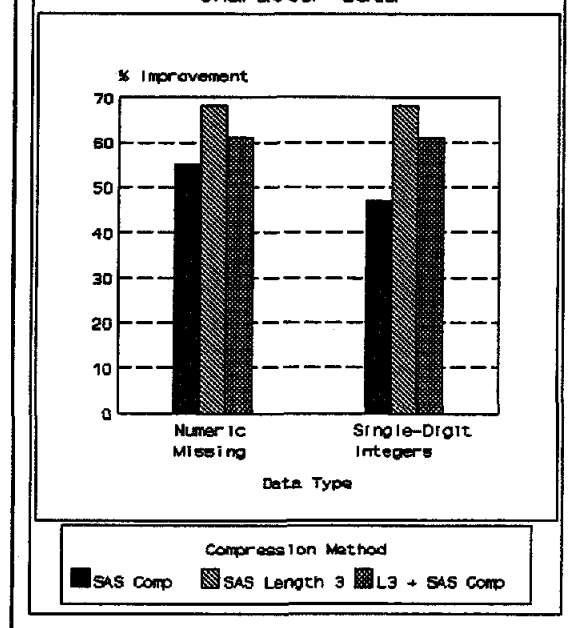
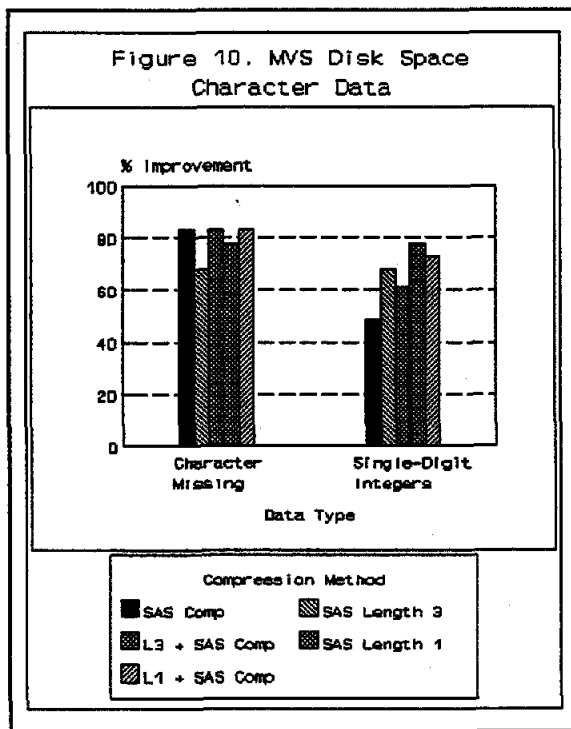
Figure 9. MVS Disk Space
Character Data

Table 1 offers recommendations based on the above findings for all three platforms tested. To use it, decide first which one or more data types best describe your data. Your data set need not be composed exclusively of these data types, but if it has a substantial proportion of one, it is probably worth testing whether any of the recommended space-saving techniques work in your case. The methods are listed in order by the amount of space they save, the best method first. If the best method is not practical in your case, consider the next best.

Results - CPU Efficiency

Compression was introduced in SAS release 6.06. At that time, many people voiced concern about the amount of cpu time the compression option added to their jobs. For them the tradeoff between space savings and added processing time made the compress option less attractive. The software developers at SAS made some efficiency improvements, which were present in release 6.07.

This paper tested only 6.07 and did not uncover unduly large cpu times for SAS or system compression software. Granted, the test files were relatively small (2 MB on AIX and MVS, and 200 KB on DOS). But the additional processing costs to compress these test files was never more than 4 seconds, and usually was much



less. Furthermore, processing a SAS compressed data set directly with a procedure required only a few percent more cpu time than processing its uncompressed counterpart.

Two points need to be considered when working with data sets large enough to require compression. First, even when working with very large files, you can start with small files containing representative samples of the data while developing a program. After debugging the program, you only need to compress the full data set once, then uncompress it during production runs.

Second, you probably should resist the temptation to set the global option `compress=yes`. This compresses, or at least attempts to compress, all data sets, including temporary data sets on libref WORK, as well as permanent ones. As shown earlier, compression often is not appropriate for certain types of data. Even failed attempts to compress cost some cpu time, and may in fact result in a larger file. Unless disk space is at an absolute premium, both cpu and real time are virtually free, and you are certain that your data sets are in fact compressible, compressing all SAS data sets is probably not a good choice.

Because the current tests did not uncover substantial differences in cpu time between the compression methods

or among the data types studied, this paper will only present a summary of the findings.

AIX. The longest time either compression task required on a 2 MB test file was 4 seconds. The AIX command usually required more cpu time than the SAS compress option, for both compression and uncompression. For example, on the file containing single-digit integers stored at length 8, AIX compression required 4 seconds (and achieved 91% space savings) while SAS compression required about 1 second (but achieved only 52% space savings). Uncompression required 3 seconds for AIX and 2 seconds for SAS. (Simply accessing the compressed file with a SAS procedure required less than 1 second additional time.)

Presumably, larger files would magnify the differences in processing time between the two methods. However, it would require a file several orders of magnitude larger than this to see truly substantial differences between the AIX compress command and the SAS compress option. Even then, I doubt that any of us would choose, for example, 50% faster processing over higher compression when the compression ratios are 60% for SAS and 90% for AIX. It is much cheaper to wait an extra minute or two for a task to finish than to triple or quadruple one's disk space.

DOS. Using the fast (-es) option, PKZIP required 1-3 seconds to compress the narrow (200 KB) test files, and never more than one second to uncompress them. With the wide file (1 MB) it required 7 seconds to compress and 6 seconds to uncompress. Note that the maximum (-ex) option almost always created a larger file than the fast option, and required up to 10 times longer to run. These findings suggest the fast option lives up to its name, and may be better in all respects than the maximum option. These processing times are very attractive, given how effective PKZIP is, so it is easy to see why that software has become so popular.

MVS. The SAS compression option under MVS behaved about as efficiently as under AIX. For example, a 2 MB data set of single-digit integers stored in numeric length 8 variables took about 60% (1.3 cpu seconds) longer to create in compressed than noncompressed form, with about 50% space savings. Reading the compressed form in PROC MEANS required about 30% longer. Generally, the more compressible a data set, the more cpu time SAS required to write it and read it in compressed form. This suggests that even though a data set is minimally

compressible, the cpu costs of compressing it may be so low that it is worth doing.

Results - Ease of Use

All Platforms - SAS Compress Option. The SAS compress option is very convenient: simply add

```
options compress=yes;
```

to a command file and all SAS data sets will be compressed. For reasons discussed above, you probably should not use compression so indiscriminantly, but rely instead on the data set option:

```
data out.one(compress=yes);
```

In either case, uncompression is automatic and transparent. Compression sets a flag in the SAS data set, which PROC CONTENTS reads, so you can easily determine whether a data set has been compressed. You cannot use random-access methods (e.g., POINT= on the SET statement), and the OBS= option behaves a little differently with a compressed data set. Otherwise, you can use a compressed data set just as you would a noncompressed one.

All Platforms - SAS Length Statement. The difficulty of writing a SAS length statement depends entirely on the complexity of your data. If all your variables are single-digit integers, you can write a simple statement such as

```
length default=2;
```

and save yourself 75% storage space. On the other hand, if your data vary between 1 and 16 digits and contain both character and numeric variable types, you may have to write a very complex length statement in which you name every variable and assign each a different length. I have often done this simply by editing a SAS input statement. It may require half an hour to assign optimum lengths to 500 variables, but the result is the most efficient storage I can achieve. As added incentive, the efficiency will be carried forward through successive generations of data sets containing these variables.

AIX Compress Command. You must apply the AIX compress command to each file or group of files you want to compress. The command is:

```
compress myfile.ssd01
```

You must also uncompress each file or group of files with the uncompress command. Compressed files are renamed with the suffix ".Z", so they are easily identified with standard UNIX commands (such as ls).

If you choose AIX compression, the SAS x command is available to make the task easier. You can add the uncompress command to a SAS command file for batch or production type jobs as follows:

```
x 'uncompress /path/myfile.ssd01.Z';
libname in '/path/myfile';
```

Uncompression then becomes virtually transparent. However, if you are running SAS from the Display Manager, the x command will leave DM, execute the uncompress command, and require that you press enter to return to SAS. If the SAS file is not compressed, AIX will issue an error message and SAS will continue normally.

Similarly, the compress command can be added to a command file after SAS writes a permanent data set as follows:

```
x 'compress /path/myfile.ssd01';
```

Compression then becomes transparent.

DOS PKZIP Command. Like its AIX counterpart, you must issue the PKZIP command for every data set. One difference is that PKZIP leaves the noncompressed file intact, while creating a compressed copy of the file with the extension .zip. You need to delete the noncompressed copy in order to realize the space savings.

You can build PKZIP into production jobs using the SAS x command:

```
x 'pkzip -es myfile.zip myfile.ssd';
```

You can unzip just as easily:

```
x 'pkunzip myfile.zip myfile.ssd';
```

Discussion

The SAS compression algorithm in the current release is relatively simple. It looks for repetitive strings and replaces them with shorter strings. More complex algorithms, based on the Lempel-Ziv system, are used in

the AIX compress command and in PKZIP and similar software. These algorithms are called "adaptive" because they learn from patterns in the data. The more uniform the data remain throughout a data set, the more effective these algorithms are at compressing it. Yet, as we have seen, the more effective algorithms are not necessarily more efficient. In fact, the time required to compress a data set seems roughly to be a function of how well the algorithm can compress it - the more it compresses the longer it takes. The developers at SAS expect to introduce an adaptive compression algorithm in a later release. If the results presented here hold, we can expect the new algorithm to be more effective, probably on a par with AIX or PKZIP, but also to consume somewhat more cpu time. Since SAS compression will be easier to use, I expect it will become the space-savings option of choice.

References

Beatrous, S.M. and Stokes, J.T. "I/O Performance Improvements in Release 6.07 of the SAS System under MVS, CMS, and VMS." In *Proceedings of the*

Seventeenth Annual SAS Users Group International Conference, 1992, 960-964. Cary, NC: SAS Institute Inc.

Muller, S.M., Hardy, K.A., and Brown, K.J. "Getting it and Keeping it with SAS Software -- Advanced Strategies." In *Proceedings of the Seventeenth Annual SAS Users Group International Conference*, 1992, 221-227. Cary, NC: SAS Institute Inc.

Pastrick, G. "Double - No - Triple Your Hard Disk Space with On-the-Fly Data Compression Utilities." *PC Magazine*, January 28, 1992, 261-284.

SAS Institute Inc. *SAS Language: Reference, Version 6, First Edition*. Cary, NC: SAS Institute Inc., 1990.

Simon, B. "Squeeze Play." *PC Magazine*, October 15, 1991, 291-320.

SAS" is a registered trademark of SAS Institute Inc., Cary, NC.

Table 1: Recommended Space-Saving Methods

Data Type	AIX	DOS	MVS
Numeric, Real (High Precision)	Raw data with AIX compression	Nothing helps	Binary data (esp for wide files)
Numeric, Missing (Sparse)	AIX compression Length SAS compression	PKZIP Length	Length SAS compression
Numeric, Short Integer	AIX compression Length or SAS compression	PKZIP Length	Length SAS compression
Character, Short Integer	AIX compression Length or SAS compression	PKZIP Length	Length SAS compression
Character, Missing	AIX compression SAS compression Length	PKZIP Length	SAS compression Length