A FILING SYSTEM WITH VARIABLE SIZE ATTRIBUTES FOR THE GASP SIMULATION LANGUAGES

Robert E. Young

Terrence E. Wilson

ABSTRACT

This paper presents a new filing system which dynamically allocates storage based upon the attribute requirements for each file. Thus, we can eliminate the storage overhead associated with the current method which requires using the same number of attributes for all files based upon the maximum number of attributes required for any one file. The new filing system is transparent to the user and its computational overhead is minimal.

INTRODUCTION

Simulation techniques are being utilized by management personnel and design engineers at an ever-increasing rate. Simulation models provide insight into the future with much smaller costs than alternative means, such as physical models or prototypes.

GASP-IV and GASP-PL/1 are simulation languages which are gaining in popularity due to their ability to process combined (continuous and discrete) models. These languages utilize a filing system which facilitates dynamic entries and exits into and out of the file. Each item in the filing system has a specific number of attributes. Entities defined by these attributes are grouped into files based upon user defined relationships. Files allow the user to group entities together to facilitate the movement of items within the simulated model.

GASP currently uses the same number of attributes for each file, based upon the largest number of attributes required for any one file. In general, this procedure is inefficient in its allocation of storage. This paper involves the development of a new technique for allocating storage to a file according to the number of attributes required for that file.

Initially, the current GASP filing system is discussed. A presentation of the new system follows detailing both the storage allocation and recovery methods.

CURRENT GASP FILING SYSTEM*

The current GASP filing system will be presented through the insertion and deletion of entries which each have four attributes. In the GASP input

*This discussion of the GASP filing system is abstracted from Pritsker [PRIT74] and from Pritsker and Young [PRIT75].

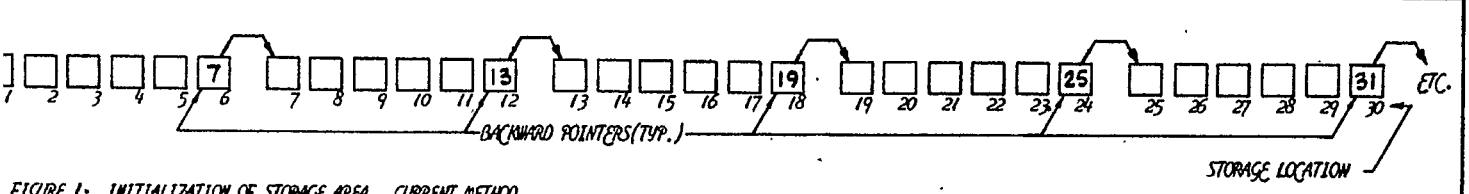

FIGURE 1: INITIALIZATION OF STORAGE AREA - CURRENT METHOD

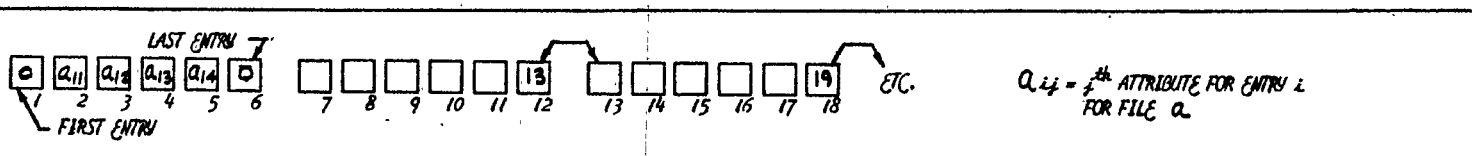

FIGURE 2: FILING OF FIRST ENTRY - CURRENT METHOD

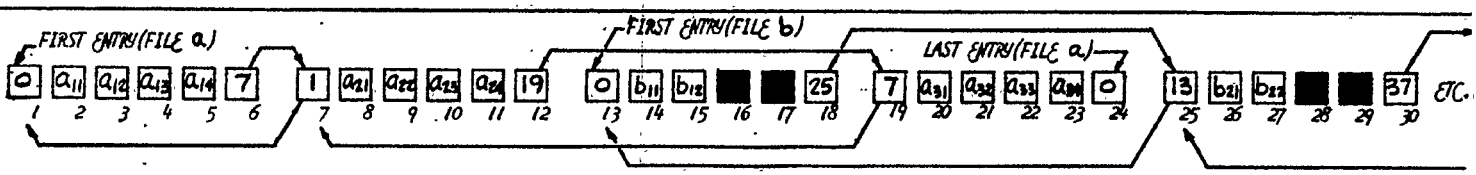

FIGURE 3: FILING OF MULTIPLE ENTRIES FOR TWO FILES - CURRENT METHOD

data, the maximum number of entries and maximum number of attributes required for any one file must be provided by the user. GASP then sets aside an amount of storage equal to the maximum number of attributes plus two (these two additional places are occupied by file pointers) times the maximum number of entries for all files. The result is a storage area divided into subareas with each subarea a potential storage location for a file entry. The subareas are linked together by backward chaining. (see Figure 1).

Entries within the filing system are linked together with both predecessor and successor pointers producing both backward and forward chaining. This allows insertion and removal of entries with minimal effort. If a predecessor/successor pointer has a zero value it signifies the associated entry is the first/last entry in the file. Thus, if a file has only one entry, both the entry's predecessor and successor pointers will be zero. Figure 2 illustrates a file with only one entry.

As successive entries are filed, the forward pointer of the previous entry for this file is set to the first location of the entry (see Figure 3). When an entry is removed from a file the forward pointer of the previous entry is set to the first location of the next entry. The backward pointer of the next entry is set to the first location of the previous entry, and the vacated space becomes available to file the next entry. (see Figure 4).

Each time an entry is filed or removed, file statistics are updated. The statistics include mean, standard deviation and maximum number in the file. Entries to multiple files are interspersed among each other and connected to previous and subsequent entries by backward and forward pointers.

The priority of an entry is determined at the time it is placed in the file. Thus, for example, if a file used a FIFO priority, an inserted item would be linked with the current last entry at the time it was inserted into the file. The ranking priorities available are first-in-first-out (FIFO), last-in-first-out (LIFO), low-value-first (LVF) and high-value-first (HVF). The event file (file number one) is always ranked LVF based upon time (attribute number one).

Recently, a variable number of attributes capability was added to GASP by Pritsker & Associates (see Grant and Sabuda [GRAN77]) based upon the work by Gehringer [GEHR74]. The system is similar to the buddy system described in Knuth [KNUT75], p. 442-445. In this system the GASP file structure row maintains up to four different block sizes of 4, 8, 16 and 32 cells. Files are categorized into one of the block sizes by using the closest block size with a value greater than or equal to the number of attributes plus two. Thus, entries with one or two attributes would be placed in blocks of size 4; entries with between three and six attributes would be placed in blocks of size 8, etc. A simulation program is restructed to a maximum of 30 attributes in any one file.

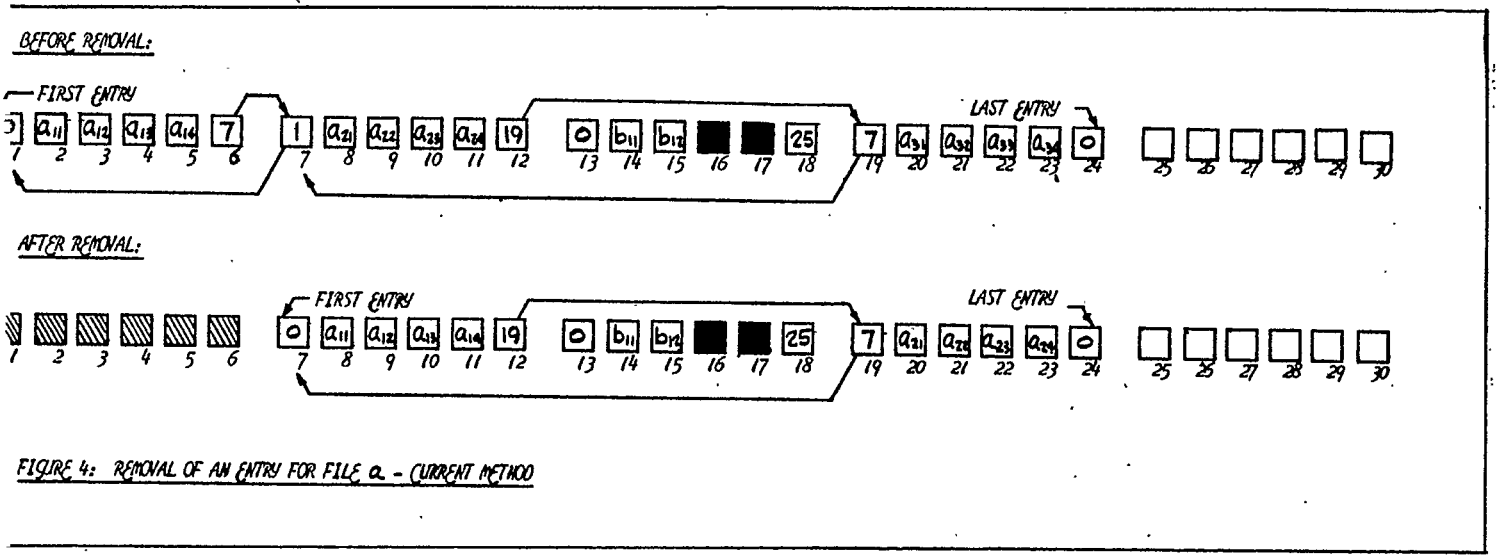

FIGURE 4: REMOVAL OF AN ENTRY FOR FILE a - CURRENT METHOD

CURRENT GASP FILING SYSTEM INEFFICIENCIES

The number of attributes for an entry, regardless of the file, is equal to the maximum number of attributes required for any file. When the actual number of attributes needed for a file is less than this maximum value, the additional attributes are included with the entry. Figure 3 and 4 illustrate this storage inefficiency. The additional attributes filed per entry are the difference between the maximum required for any file and the actual number of attributes needed for the file. As this difference increases and the number of file entries becomes large, the amount of unused space can become significant. The new filing system eliminates this storage overhead by efficiently allocating storage to a file based upon the file's specified number of attributes.

The variable number of attributes scheme recently implemented by Pritsker & Associates only partially solves the storage inefficiency problem. If we have a file with 17 attributes an entry for it will be assigned to a 32 cell block incurring a 15 storage location overhead. This can lead to situations where Gehringer's scheme actually requires more storage overhead than the old GASP filing system.

THE NEW SYSTEM

In the new system, an area is set aside which contains available storage, designated the "available storage area." When a file requires storage to file an entry it seizes only that amount from the available storage area consistent with its number of attributes. The seized storage remains with the file, being allocated and released within the file as needed. When all available storage has been seized by requesting files, all files are examined for free storage. Any free storage which is located is returned to the available storage area to be redistributed to requesting files as the simulation continues.

The maximum number of entries for any file is not required in the GASP input data; however, the number of attributes per file must be specified. A storage space is initially set aside. However, pointers are not initialized as in the current GASP method. When the first entry is filed, the forward and backward pointers of this entry are set at zero and the attribute values are placed in storage. Figure 5 illustrates a single entry placed in the storage system.

As successive entries are filed, additional space is allocated to files as needed. Refer to Figure 6. Note that only sufficient space is allocated to satisfy the exact number of attributes specified for each file (i.e., the storage area contains no unused space). The storage remains with a given file until the compression routine is called to recover empty storage (the compression routine will be discussed later). Thus, storage is dynamically allocated during execution based upon each files storage requirements.

Since allocated storage remains with a file until the compression routine is called, we must keep track of spaces within each file which become empty due to the removal of entries. This is accomplished by backward chaining empty spaces for each file but not between files. Thus, pointers are created which locate the first available storage location for each file. Let us denote these pointers as $\rho_k, k = 1,\ldots,n$ and n is the maximum number of files.

As an entry is removed a backward pointer is placed in the first position of the entry (see Figure 7). It points to the previous blank entry for the file. The second position of the removed entry is filled with a "-1", this is a blank space indicator. The third position contains the number of blank spaces for the file (i.e., the number of attributes plus two).

As successive entries are filed in file k, ρ_k is checked for a zero value. If ρ_k is zero, indicating that there are no blank entries available within this file, new storage space equal to the number of attributes for the file plus two locations for the two pointers is acquired from the available storage area. The forward pointer of the preceding entry and the backward pointer of the successive entry are updated as before, ρ_k is also updated. If no blank space is available for the file a com-

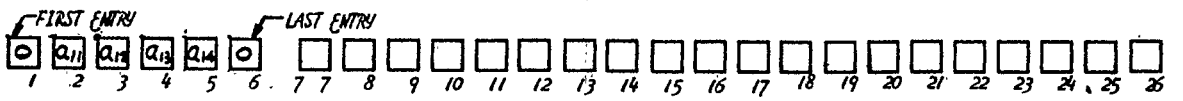

FIGURE 5: FILING OF FIRST ENTRY - NEW METHOD

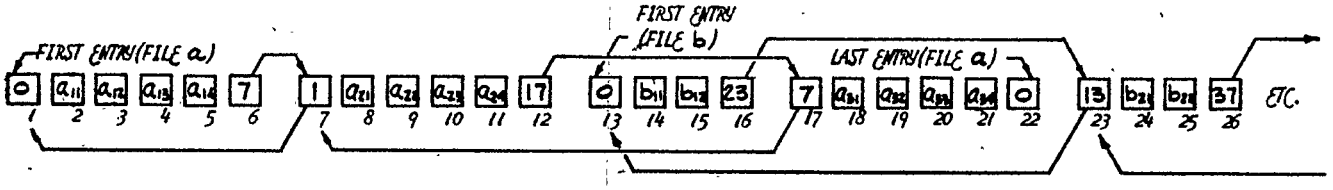

FIGURE 6: FILING OF MULTIPLE ENTRIES - NEW METHOD

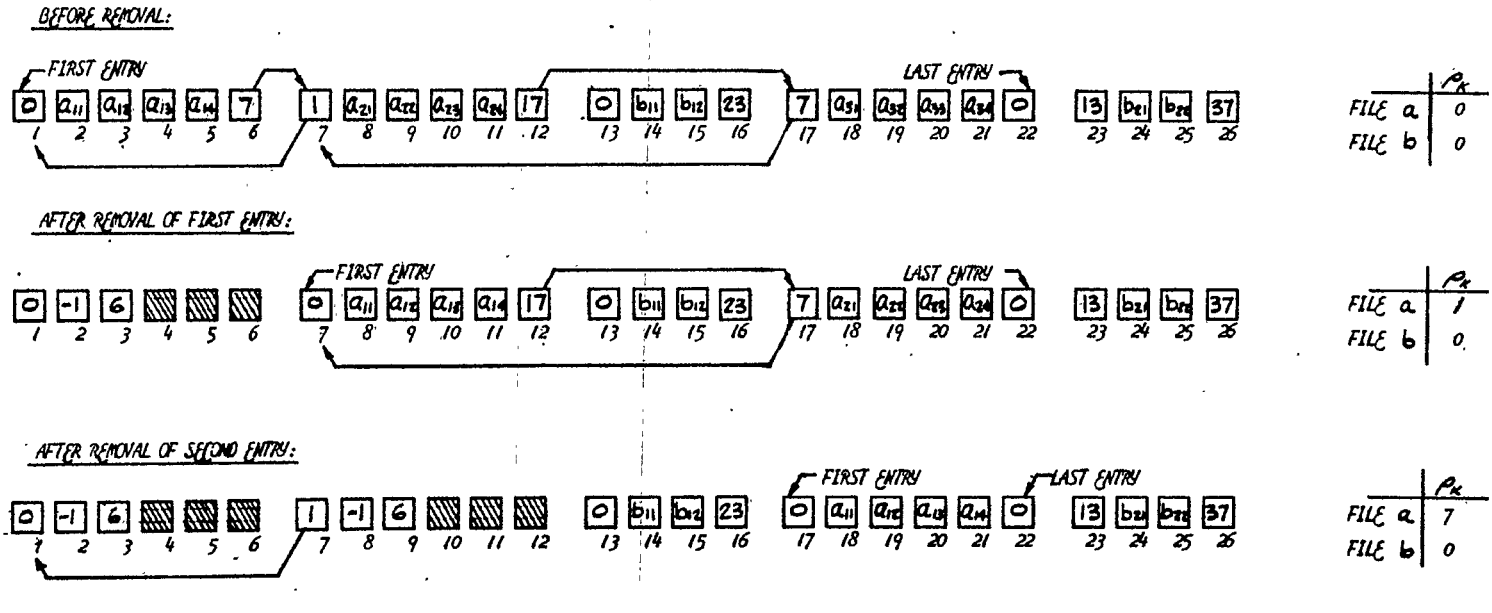

FIGURE 7: REMOVAL OF ENTRIES FROM FILE a - NEW METHOD

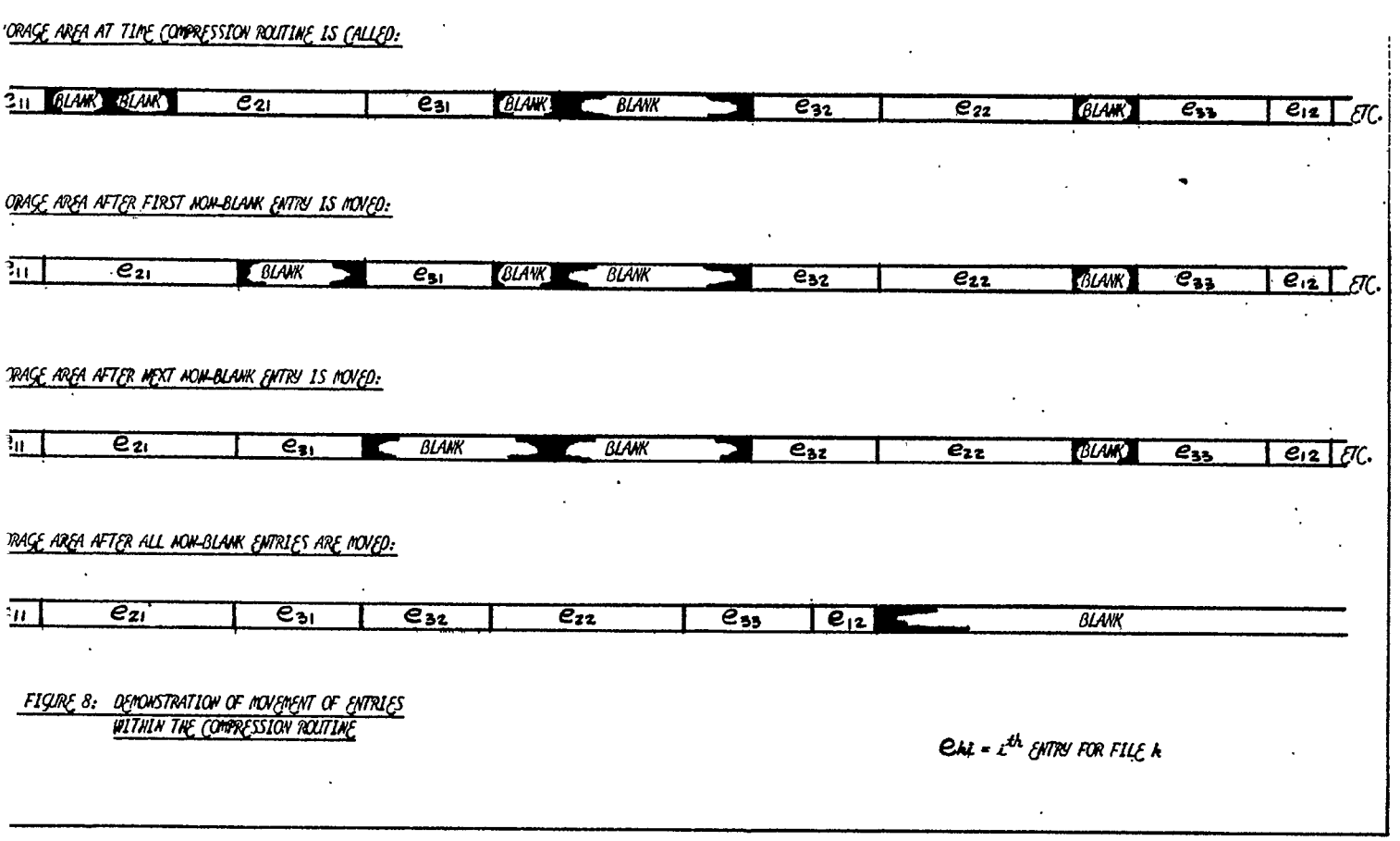

FIGURE 8: DEMONSTRATION OF MOVEMENT OF ENTRIES WITHIN THE COMPRESSION ROUTINE

e_{hi} = i^{th} ENTRY FOR FILE h

pression routine is called to recover blank space from all the files.

THE COMPRESSION ROUTINE

The compression routine returns unused storage space to the available storage area. The returned space is recovered from space previously allocated to a file but currently not in use. The recovery technique relies upon a modified bubble sort algorithm. Occupied storage is pushed to the top and unoccupied storage is pushed to the bottom. The boundary represents the new location for the beginning of the available storage area.

To recover unoccupied storage it first must be located. This is accomplished by sequentially searching the filing system looking for a blank storage indicator. The indicator was placed in the blank storage at the time an entry was removed from a file. When blank storage is found the next occupied entry is located. The blank storage and the entry are "swapped." Pointers for the entry, and its predecessor and successor entry, are adjusted to reflect its new location. The next entry is then located and "swapped." Continuing in this manner the blank storage is pushed to the bottom. The procedure is illustrated in Figure 8.

When a occupied entry is located, its number of attributes must be determined. Since we are sequentially searching the file storage array, we have no prior knowledge about the number of attributes from either the file or the entry itself. As a consequence, the number of attributes is determined by testing among the possibilities represented by the various files. The test is provided by examining the successor and predecessor entries. If their respective predecessor and successor pointers contain the location of the current entry being tested then we have correctly identified the proper number of attributes. If not, then we choose a new candidate.

To improve the efficiency of searching for the correct number of attributes, the sequence employed in the search was based upon the attribute number to which the largest amount of available storage had been allocated. For example, if 500 allocations had been made to files with ten attributes and only 150 allocations to all others, we would expect more unoccupied storage with a length of ten attributes.

Thus, we rank the number of attributes by their allocation count and test sequentially based upon the greatest probability of occurance for a given number of attributes.

CONCLUSION

The new filing system has been successfully run with sets of test data. Future testing will include the processing of some classic simulation examples, and comparison of results with the current GASP filing system.

REFERENCES

GER74 Gehringer, Edward F. "GASPIV with Variable Number of Attributes" IE 680 project report submitted to Allan Pritsker School of Industrial Engineering, Purdue University, June, 1974.

GRAN77 Grant, Floyd H. and Sabuda, Jerome P. "New GASPIV Capabilities" Pritsker & Associates, Inc. W. Lafayette, Indiana October, 1977

KNUT75 Knuth, Donald The Art of Computer Programming, Vol. 1, Fundamental Algorithms, 2nd Edition Addison-Wesley Inc., Reading, Mass., 1973, p. 442-445.

PRIT74 Pritsker, A. Alan B. The GASP IV Simulation Language John Wiley & Sons, New York, 1974

PRIT75 Pritsker, A. Alan B., and Young, Robert E., Simulation with GASP-PL/1 John Wiley & Sons, New York, 1975