GFT: A tool for data management in the UNIX environment

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Wissenschaftliches Rechnen

Lise M. Pfau

GFT: A Tool for Data Management in the UNIX Environment

May 1990

129

Authors' address:

Wissenschaftliches Rechnen
ETH-Zentrum
CH-8092 Zurich, Switzerland

Abstract

This guide explains the correct application and subsequent usage of GFT, a tool for efficiently managing large amounts of multi-attribute, dynamic data. Based on the Grid File concept, GFT tightly controls the number of disk accesses required for range queries which involve higher dimensional regions. The objectives of GFT are three-fold: to require only a single disk access for a fully-specified query, to furnish a multi-attribute data management system completely integrated into the UNIX environment, and to provide a tool for managing data sets containing an unlimited number of search attributes of arbitrary type.

Table of Contents

1. A Grid File in the UNIX Environment

Many data intensive computer applications become practical only if their underlying data structure efficiently manages multi-attribute, dynamic data. In this context, efficiency implies both quick multi-attribute access to the data and compact data storage on disk. Data structures rarely provide these two features simultaneously, however, since compact structures tend to restrict efficient data access to only one or two attributes. This restriction is unacceptable when the data set contains objects characterized by a value for each of the object's properties, or *attributes* — and searches for particular objects are based on several of these attributes. For example, a data set representing a forest might distinguish tree objects by several attributes such as position, species, height, health, diameter, etc. A statistical analysis of the forest may require the ability to query the data on all the attributes, not just a small subset. In such cases, the selection of an appropriate data structure is critical.

Optimal multi-attribute data structures also provide *symmetric access* to the data, meaning any of an object's attributes (or combinations thereof) may be used for searching without any significant performance variations. The Grid File [5] is one such efficient storage structure which permits very fast access with respect to all attributes, to dynamic amounts of data on disk. The GridFile Tool (GFT) implements the Grid File concept, allowing a user to build or augment data management systems under the UNIX®[1] operating system. Given the UNIX toolbox approach to applications design, GFT would be the tool used to manage the multi-attribute data. For instance, our forestry application might use GFT to manage its multi-attribute tree data while using a chosen graphics tool to display the trees.

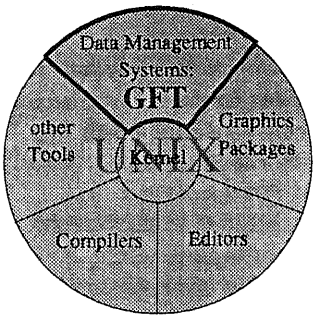

Figure 1. UNIX contains many tools for application development — the GFT data management system could be one such tool.

[1] UNIX is a Trademark of AT&T Bell Laboratories.

This document will guide a novice through the correct application and subsequent usage of the GFT tool. The rest of this section discusses the objectives of the Grid File in general, and GFT in particular. Section 2 describes each of GFT's usage modes activated by a UNIX command, while Section 3 explains three optional command arguments that provide additional features. Finally, Section 4 explains the graphical interface which has been developed for Sun Microsystems hardware, as opposed to the command-level interface described throughout Section 2.3 which is available on any UNIX system. The Appendices provide an abbreviated reference for those concepts discussed in the regular sections. The beginning user will find clarification of *italicized* terms in Appendix A. References for more advanced users such as a command summary and user file structures are outlined in Appendices B and C. The data types which GFT supports and the current GFT functionality are found in Appendices D and E.

1.1 Managing Multi-Attribute Data with a Grid File

Many structures manage multi-attribute data — for example, inverted files, k-d Trees, threaded files, k-d B-trees, and R-trees. However, few other than the Grid File combine high speed, good average space utilization and adaptability for data growth, while maintaining an efficiency proportional to the number of attributes. These advantages along with the design concepts which lie at their foundation will be discussed in the following subsections. Although knowledge about the Grid File design is not required for its usage, understanding how the multi-attribute data will be managed will clarify GFT's benefits to an application.

1.1.1 When and Why is a Grid File practical ?

Applications containing very large, dynamic data sets upon which multi-attribute queries must be made can particularly benefit because a Grid File controls the number of disk accesses. While the number of accesses for retrieving a single record can be restricted to two,[2] the average number of disk accesses per record for a range query is likely to be much less. This economy arises from the fact that a range query involves data records neighboring in attribute space (as opposed to a query involving dependencies in the form of hierarchies, for example). By design, the Grid File stores neighboring records physically close to each other on disk. Thus the Grid File is very efficient for large amounts of data which must remain on disk (at least partially), and whose queries specify ranges forming higher dimensional regions.

Another advantage of the Grid File is that its design enables symmetric access according to all search attributes in the data set. Thus no single attribute will

[2] As will become clearer in the next subsection, GFT guarantees a maximum of one disk access for a single record.

be the principal means for searching through the Grid File data, and no subset of the attributes will be significantly better or worse for searching — meaning that no a priori knowledge about how the data will be searched is required. This implies that the data can be accessed in many different ways; a user need only choose the appropriate one.

Since the Grid File's internal structures readily adapt to any growing and shrinking in the amount of the data, dynamic data sets can be efficiently managed with an average utilization of approximately 70% of the allocated memory. If the data set ever becomes fixed, the amount of secondary memory used could most likely be reduced through optimization techniques (mentioned in Section 2.4). These advantages of fast symmetric accesses, adaptability, and compact secondary storage, make the Grid File a practical solution to many data management needs.

1.1.2 How does a Grid File manage data?

A Grid File organizes data by partitioning the data space into a large-grain irregular multi-dimensional grid, where each dimension corresponds to a search attribute and each hyper-plane partition separates data according to an attribute's value. Each block formed by the grid references a "bucket" of fixed-size containing those objects whose attribute values lie within the gridblock's bounding partitions. Buckets are actually shared amongst neighboring, sparsely occupied gridblocks in order to gain storage and retrieval efficiency.

A Grid File is composed of three basic components: the **scales**, a **directory**, and the **buckets**. The scales are single dimensional arrays (one per search attribute of the data set), whose elements represent the partitions superimposing the grid on the data space. The directory, which directs the search from the scales to the data bucket, dynamically manages the gridblocks, each of which points to a data bucket. The buckets are fixed-size arrays that can be transferred as a single unit to and from disk. The design of these components varies across implementations, and understandably the major differences lie in the directory structure. While the directory could be managed as a multi–dimensional array, GFT's directory, called the region directory, manages only the hyperrectangular regions comprised of all gridblocks that point to the same bucket. Despite these design differences, all Grid Files use the three components in the same manner when searching for data entries, as shown in Figure 2 (note that the bold rectangles in the directory indicate the bucket regions).

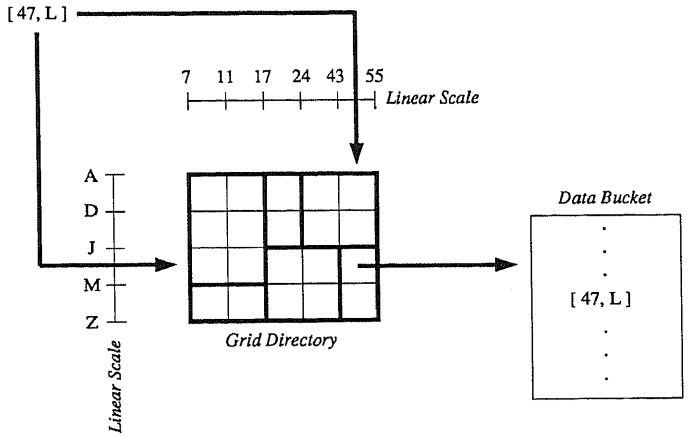

Figure 2. The use of the three Grid File components when searching for the entry [47, L] in a two-attribute data set.

A maximum of two disk accesses are needed for most gridfiles to retrieve a fully-specified record, whenever the scales are kept in central memory. Assuming the other two components remain on disk, one disk access is needed to retrieve the correct directory entry, and a second is needed to retrieve the bucket which actually contains the record. GFT, however, guarantees a maximum of **one** disk access — its region directory grows linearly with the data set, thus is compact enough to be stored in central memory for the majority of applications and hardware configurations.

More information can be obtained about the Grid File from the computer science literature. The gridfile concept in general and the region directory in particular can be found in [5] and [3]. [7] and [6] explain GFT in more detail. Other implementations of the gridfile are explained in [2], [4]and [1].

1.2 The GFT Philosophy

The primary motive for building GFT was to provide a flexible data management tool especially suited for the UNIX/C environment. According to the UNIX style, the UNIX command line serves as a user's major interface thus permitting GFT to be combined with many other commands in a UNIX shell script. GFT also makes extensive use of the standard I/O files, enabling data to be piped into or out of other tools. The choice of implementation language, C, allows integration with a wide variety of existing software (e.g., graphics packages) and virtually eliminates problems when porting the software to different platforms. This flexibility and portability should permit GFT to be used in a wide variety of applications.

Several different data management facilities are already available under UNIX, however GFT's functionality is quite unique. The UNIX operating system itself includes a tool, dbm/ndbm which provides minimal functionality for single key[3] access to data. Other databases such as INGRES, are available commercially; however these are huge systems which provide maximal functionality. In these cases, a user must first understand many detailed aspects of the database management system (e.g., the data model, the command language) in addition to incurring severe overhead costs for features not even utilized.

GFT is a very small, straightforward, and easy-to-use tool focusing on access to multi-attribute data stored in secondary memory. In central memory, the data are organized as arrays, although an application is not prevented from imposing additional structure. Being a tool, GFT provides a general, powerful service but might not be ideal for some very specialized applications. Several features one might expect of a full-blown database (e.g., query optimization, access controls, consistency checking) have been excluded, although the GFT design permits a programmer to easily supplement the system with further functionality. This flexibility and exactly how it can be utilized are discussed in the following subsections.

1.2.1 The GFT-UNIX relationship

GFT provides an operating system interface for the casual user, along with a programmatic interface for larger programs in which a data management system such as the Grid File is just one of many modules. The exact relationship between UNIX, GFT, the user and the programmer is depicted in Figure 3. The GFT programmer may develop an application (with an editor and C compiler) using functions in GFT and perhaps those of other UNIX tools. One such application, sungrid (explained in Section 4), has been developed in order to provide a graphical interface to the tool. End-users will want to use GFT in conjunction with some editor, and may use sungrid in addition. Users will primarily interface directly with GFT through UNIX commands (as depicted in bold in Figure 3) — exactly the interface this document focuses on.

Also depicted in Figure 3 is the well-defined separation between the core of GFT and those internal parts that relate to the outside world. GFT has only a very small, segregated interface with UNIX, allowing the tool to be ported very easily to a different operating system. The user interface is also cleanly separated from the heart of GFT, thus it can easily be discarded when programmers wish to imbed GFT into their application. Hence the core functionality of GFT is strictly isolated from possible effects of external alterations.

[3] Actually this key is not even an attribute of the data, it is a relatively meaningless piece of information used simply as a pointer to the data.

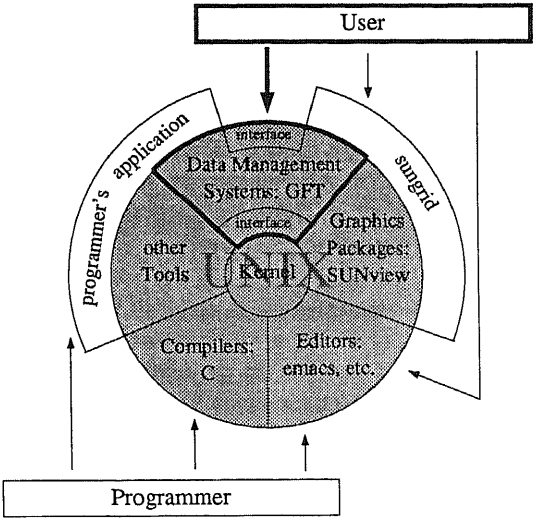

Figure 3. The relationships of the GFT user and programmer with a Unix toolbox.

1.2.2 The GFT Lifecycle

GFT may be used to create Grid Files for data sets which differ in the number and type of attributes — although the entries of a single *gridfile* (instance of a GFT Grid File) must be identical in structure. Each gridfile contains a set of system files recording the current state of the scales, directory, and buckets. As shown in Figure 4, a gridfile utilizes common GFT Software along with its individual set of system files so that the same GFT Executable can operate on different gridfiles. In fact, several GFT gridfiles may be used simultaneously, however, this is an advanced functionality explained in [7]. The set of system files, which distinguish a gridfile from all others, will be used only during the lifespan of that particular gridfile.

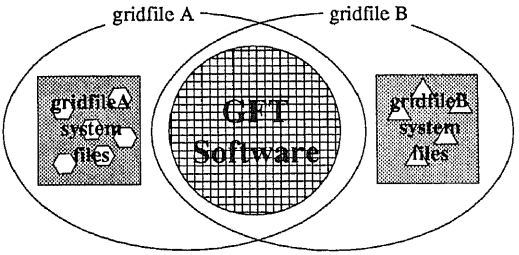

Figure 4. GFT gridfiles are composed of common GFT software and individual sets of system files.

The typical lifecycle of a data management application involves creation, followed by adding/querying data, and ending with deletion (or off-line archival). The GFT lifecycle is somewhat expanded to contain the following modes: creating the gridfile, adding data to the gridfile, querying/updating the data, testing the gridfile, optimizing the gridfile, extracting all the data, and finally deleting the gridfile. Although the list of modes might imply a chronological usage, after creation and before deletion, any mode may be used to operate upon the gridfile data. Each of these modes actually corresponds to a UNIX command line argument to GFT.

Typically, GFT will be used non-interactively, with a user communicating only through UNIX command lines. A command line will specify `gridfile`, followed by the particular gridfile name along with additional arguments specifying the usage mode. In some cases, GFT will expect a user's own input or output files to be referenced on the command line. Input files, which contain information such as the entries to add, should be created outside of GFT in any text editor. During execution, GFT will transfer the information from the input files to the gridfile's system files; afterwards the input files may be deleted. Output files, into which GFT will put information about the command processing, will be overwritten if they already exist (the user may delete these files at any time). Unless names of the files are specified, several GFT commands will assume that information is coming from standard input or that output should be sent to standard output. This feature enables the commands to be used in conjunction with UNIX pipes. Note that any errors while processing GFT commands will be sent to the standard error file.

2. Basic GFT Commands

This section explains the basic usage modes of GFT: creation, addition of data, performing transactions, optimization, testing, extracting all the data, and deletion. Each mode operates on a single gridfile, and is described in its own subsection. Although the descriptions should initially be read in order, any mode may be used after creation and before deletion. In order to use a mode, the user simply executes the UNIX command enclosed in a box at the beginning of the subsection. Each command, whose format is given in Backus-Naur Form, includes one or more parameters (given in italics). Some parameters refer to the user's input or ouput files — the input files should be created by the user before executing the UNIX command, although GFT will create (or overwrite) the output files. The user's files may be deleted any time after the command has been processed.

Within each subsection, an example will be given of the GFT functionalities using a small forestry application, appropriately named `sample`. Although an application's real gain is felt when GFT manages data on a much larger scale (e.g., a great number of attributes, very large amounts of data), the example gridfile is kept small and simple. Each record (or *entry*) of the `sample` gridfile will refer to a single botanical tree, and each tree is defined by four *attributes* which are the x- and y-coordinates of the tree's position (integers), the tree's diameter (a real number), and a the tree type (a short string). The sample sessions are shown using this gridfile in each of the different modes according to the ordering of the subsections, thus the resulting state of the gridfile from one mode will be the starting state for the mode described in the next subsection.

2.1 Creating a Gridfile

UNIX Command: gridfile *gridfile_name* -n [*profile_filename*]

In order to create a GFT gridfile, the user must only provide a *profile* file describing the data set. This profile contains all the meta-information generally explaining the attributes and values of the data to be managed. Its most important information, and indeed the only information the profile is required to contain, is the number of attributes and the type of each attribute. However, several profile options are available which will further tailor the gridfile to the application's needs. As shown in Figure 5, upon successful creation of a gridfile (according to the profile), three GFT system files with the extensions `.gfb`, `.gfd`, and `.gfs` will appear in the current directory. If any errors occur during creation, the user should simply correct the profile and create the gridfile again.

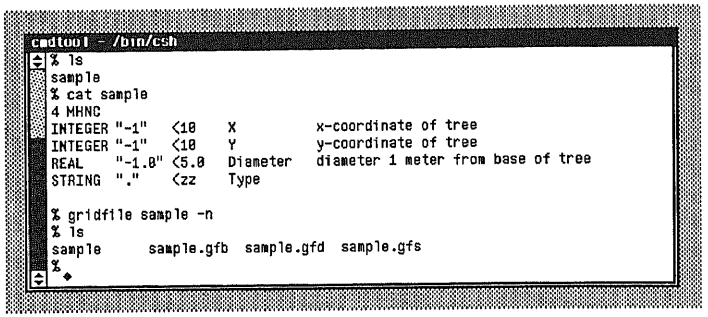

Figure 5. Creating the `sample` gridfile

The profile options permit the user to reduce the number of search attributes, associate a name and/or comment with the attribute, provide a precision for the attribute value, allow entries to contain missing values for the attribute, or provide the bounds for the attribute data. As a rule, upon indicating a specific option on the first line of the profile, the user will need to specify the option's value for each attribute to which it applies — for instance when indicating that the gridfile should handle missing attribute values, the value `-1` might be specified to denote that a `-1` for that attribute's value in any entry actually means the value is missing. The only option which does not follow this rule is the number of search attributes since this option applies to the entire set of attributes. For flexibility, the user is not required to specify an indicated option for each attribute, only for those on which the option should be exercised. However, this feature requires that the attribute option value be formatted in a particular manner: the value should be either preceded by or enclosed between specific character(s). See the Table in Figure 6 for details about each option, including a specification of which options actually make the gridfile more efficient, and which are provided only for the user's convenience. In this table, a `value` in the Attribute Specification Format should be of the same type as the attribute.

In order to increase the gridfile performance, the user could reduce the number of search attributes for each entry. Since non-search attributes are ignored when organizing the data in a gridfile, only the search attributes are used by GFT when accessing entry. As previously mentioned, the complexity of the gridfile grows linearly with the number of search attributes, thus fewer such attributes lead to greater efficiency. By default all attributes are search attributes, however, in some applications queries on the entries will involve only a subset of the attributes. In this case, an integer indicating the number of search attributes must appear on the first line of the profile. When the number of search attributes is *n*, the first *n* attribute specifications will be those for the

search attributes, whereas the rest will be non-search attributes. Remember, this is the only option which need not be further specified for each attribute. Any of the other options may be specified for either the search or non-search attributes, though for the non-search attributes only the missing option (to be discussed) will be be utilized by GFT.

Profile Option	*Usage Justification*	*Profile Indicator*	*Attribute Specification Format*
Search Attributes	Efficiency	*integer*	*Not Applicable*
Maximum Value	Efficiency	H	<value
Minimum Value	Efficiency	L	>value
Missing Value	Convenience	M	"value"
Value Precision	Convenience	P	#integer
Attribute Name	Convenience	N	string
Attribute Comment	Convenience	C	string

Figure 6. Profile options and their attribute specification

A gridfile can be made slightly more efficient by providing an open upper bound for each dimension, indicating that all input data are strictly less than this upper bound[4]. If not specified, a default upper bound of the maximum representable value for the given type will be assigned. A lower bound may also be provided which would reduce the space requirements for the given data type. For example, if one attribute's values are real numbers whose integer part is always 400, then a short floating point number might be used in order to represent just the fractional part whereas double precision would usually be needed to represent the total number. This lower bound would be added to the stored data item for output, and subtracted from the input item for storage. Note that in the case of a lower bound, although storage space is saved, computational time is increased.

Several options have no effect upon the efficiency of the data management, but are provided simply for the user's convenience. Data often contain missing values; thus the user may indicate this fact, and provide a value which may be used as an internal representation for an attribute's missing value. This value must be of the same type as any regular value for that attribute, though it should not conflict with any data which is not intended to be missing (it should also be less than the upper bound). Another option associates a precision with an

[4] Actually, the specified upper bound need only be greater than the *first* entry added.

attribute's value, thus an integer corresponding to the number of decimal places which are significant would be specified (2 means that the hundredths place is the last significant digit; by default, all places are significant. The user may also provide a name and comment for each attribute (defaults are assigned if no values are provided).

Once the user has created the gridfile with the desired profile, the next logical action is to insert data as described in Section 2.2.

2.1.1 Sample Profile File

A profile and the process of creating the `sample` gridfile using this profile, are shown in Figure 5. This profile specifies that the gridfile should manage entries with four attributes, all of which are search attributes. The first line in the file indicates the options to be exercised over the entire set of attributes, whereas each of the following lines specifies the details associated with each attribute. Notice that the `sample` entries might contain missing values (indicated by the `M` on the first line), and indeed the values are specified for each attribute (those values within double quotes). The maximum value and name of each attribute (indicated by the `H` and `N`, respectively) are given, and a comment (indicated by the `C`) is given for the first three attributes. Although the text in this profile is aligned in columns, this format was followed simply to enhance readability.

When creating the gridfile, since the name of the gridfile and the profile were both `sample`, no profile name was necessary. During creation, `GFT` created three `GFT` system files for the `sample` gridfile: `sample.gfd`, `sample.gfb`, and `sample.gfs`. These files should not be deleted by the user; `GFT` will delete them when the `sample` gridfile is deleted using the command described in Section 2.7. After creation, the name `sample` is used to refer to this gridfile when executing further manipulation commands on the `sample` data set.

For a complete specification of a Profile File, refer to Appendix C.2.

2.2 Adding a Data Set to a Gridfile

UNIX Command: gridfile *gridfile_name* -a [*data_filename*]

A user can add data to a gridfile as an entire set by providing GFT with an ASCII file in which each line specifies all the attributes of a single data entry (i.e., entries are separated by newline characters). The attribute values should be of the type specified in the profile, separated by whitespace, and ordered according to the original ordering of attributes in the profile. When specifying the UNIX command to add the data, the name of the user's data file should be included, otherwise GFT will assume the data is coming from standard input. If GFT detects an incorrectly formatted entry during the addition process, it will try to recover from the error and continue to add those entries which are correctly formatted. For a demonstration of the process, along with an example Data File, see Figure 7. Note that entries may also be added individually, using the Transaction mode described in Section 2.3, however that mode is much less efficient for adding large sets of data.

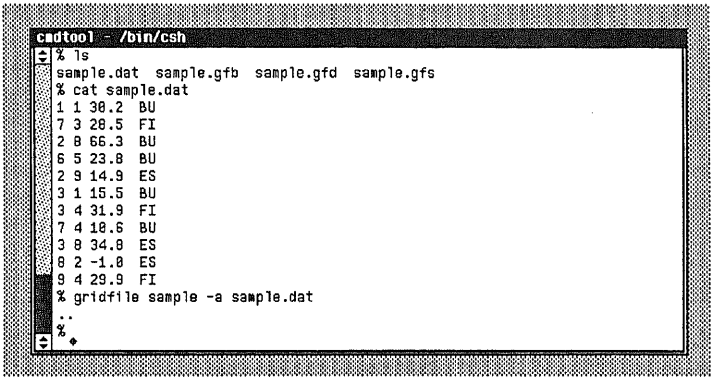

Figure 7. Adding Data to the `sample` gridfile

2.2.1 Sample Addition File

Figure 7 shows the data contained in the `sample.dat` file, and the process of adding it to the `sample` gridfile. Eleven entries are specified in the file. Notice that the tenth entry's third attribute (corresponding to the diameter of the tree) has a value of `-1.0`, which denotes a missing value as specified for that attribute in the profile. While adding the data, two periods (..) were output; GFT sends one period to standard output for every five entries added, in order to convey the processing state to the user.

For a complete specification of a Data File, see Appendix C.3.

2.3 Performing Transactions on the Gridfile

UNIX Command: gridfile *gridfile_name* -t [*transactions_filename*]

Alterations or queries to the data in a gridfile, *transactions*, are perhaps the most important mode of gridfile usage. Thus GFT provides two different methods through which transactions can be processed — a graphical interactive method as described in Section 4, and a terminal mode as explained here. Using the terminal method, the user provides a list of transactions which will be executed in order. A single character specifies the type of operation while the entries to be operated upon are specified by one or two points of a specific type (explained below). The user may enter the desired transactions into a file and specify the filename on the UNIX command line, or an interactive mode can be simulated by omitting a filename thereby signaling GFT to read the commands from standard input. The output from the commands will be sent to standard output, separated by blank lines.

The transaction operations are separated here into three groups simply for ease of explanation: additions, queries, and deletions. In general, each operation is specified by either a *point* or a *range* of points which identify the entries upon which operations should be performed. A point refers to entries in the gridfile whose attributes have specific values. If all attributes are specified, the point refers to a single entry or several identical entries. A range of points is specified by a lower bound point and an upper bound point. It refers to all entries in the gridfile whose attribute values are within (and including) this range. All points (single, upper bound, and lower bound) are either *definite* or *indefinite*, the latter meaning that one or more attribute values may be left unspecified by using an asterisk (the * character). More specifically, a query or deletion is specified either by point or range, indefinite or definite, although each addition must be made on a definite point. The format of the transaction commands is summarized in the Table in Figure 8. Note that point values (whether definite or indefinite) must be given for each attribute of the point regardless of whether it is a non-search attribute.

Transaction Type	*Command character*	*Number of points*	*Type of Point(s)*
Add Point	a	1	definite
Definite point query	p	1	definite
Indefinite point query	q	1	indefinite
Definite range query	r	2	definite
Indefinite range query	s	2	indefinite
Definite point delete	d	1	definite
Indefinite point delete	e	1	indefinite
Definite range delete	f	2	definite
Indefinite range delete	g	2	indefinite

Figure 8. The command specifications for each transaction type

2.3.1 Adding Data

A datum is added simply by specifying all the attributes of the entry — obviously! In order to add several identical entries, each must be added separately since cardinality of an entry cannot be specified when adding the single datum. Note that no output results from adding an entry, unless an error occurs. Here, no periods are output as occurs when taking data from an Addition File as explained in the previous section.

2.3.2 Querying for Data

The user formulates a query by specifying one or two points, which may be definite or indefinite although multiple points must be of the same type. By satisfying a query, GFT displays all appropriate entries with all their attributes even if the query is indefinite. Multiple identical entries will be sent a multiple number of times to standard output.[5]

Definite Point Query: This point query is, in effect, a degenerate query since the only information which the user could possibly gain is the cardinality of the point.

Indefinite Point Query: This query is identical to the definite point query, except one or more attributes may be unspecified. Although an indefinite point query may be used to identify all the entries in the gridfile, this "extraction" can be done more formally using the command described in Section 2.6.

[5] Performing operations on data other than just printing/displaying, are described in the *Programmer's Guide* [7].

Definite Range Query: A range query is made by specifying any two points — the first being the lower bound, and the second the upper bound (these points are permitted to be outside the attribute space, though the value of each attribute in the lower bound should be less than the value of that attribute in the upper bound). The attribute values of all entries which satisfy the range query will lie between (or exactly match) the corresponding values of the two points.

Indefinite Range Query: This query is identical to the definite range query, except one or more attributes may be unspecified. Those attributes which are unspecified, must be unspecified in **both** points.

2.3.3 Deleting Data

Deleting data is handled very similarly to querying. One or two, indefinite or definite points must be provided by the user. Just as each entry satisfying a query will be printed, each entry identified with this command is deleted (i.e., its cardinality will be zero regardless of its cardinality prior to the delete). No output results from deleting entries however, unless there is a warning that no entries were identified by the point(s).

Definite Point Delete: A fully specified entry is deleted simply by providing all the attributes of the point.

Indefinite Point Delete: This deletion is identical to the definite point delete, except that one or more attributes may be unspecified.

Definite Range Delete: A range delete is made by specifying any two points — the first being the lower bound, the second being the upper bound. These points are permitted to go outside the bounds of the attribute space though the value of each attribute in the lower bound should be less than the value of that attribute in the upper bound. All entries which lie within (and including) the range will be deleted.

Indefinite Range Delete: This deletion is identical to the definite range delete, except that one or more attributes may be unspecified. Those attributes which are unspecified must be unspecified in **both** points.

2.3.4 Sample Transactions File

The process shown in Figure 9 interactively executes transactions from standard input. If these interactively typed transactions had instead been entered into a file and the filename specified on the command line, those same transactions could have been performed on the `sample` gridfile non-interactively. In the process shown, five different transactions have been specified by the user,

and the resulting output of each is printed immediately after the transaction has been fully specified. These specifications were given on either one or two lines, according to the number of points required for that type of transaction (as given by the initial single letter). The transactions types shown above are: add (a), definite point delete (d), indefinite point query (q), indefinite range delete (g), indefinite range query (s).

For a complete specification of the Transactions File, see Appendix C.4.

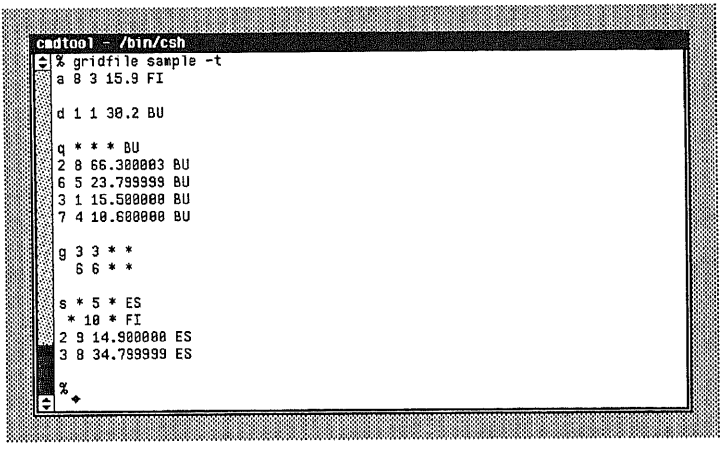

Figure 9. Performing transactions with the `sample` gridfile

2.4 Optimizing the Gridfile

UNIX Command: gridfile *gridfile_name* -o

This command will optimize the gridfile directory structure and bucket allocation. The primary gain will be in space (as opposed to time), since all the empty memory cells will be eliminated and the buckets compressed. This command should be used only after the data set in the gridfile remains static, in other words, no more entries will be added or deleted. Note that this command does not produce any output.

2.5 Testing the Gridfile

UNIX Command: gridfile *gridfile_name* -z

This mode is targetted to very cautious, serious users of GFT — it will provide detailed information about the gridfile and test the gridfile's internal structures. The detailed information consists of the number and types of the attributes, along with the positioning and occupancy of each bucket. Testing confirms the consistency of the gridfile's internal structures (e.g., that there are no empty buckets, that the buckets contain the correct data entries). Any errors are probably due to empty buckets which sometimes result from many entries being deleted (i.e., a bucket was emptied but could not be merged with a neighboring bucket). In this case, storage efficiency will be gained by extracting all the entries from the gridfile, and creating a new gridfile with those entries added. If this is not the case, the fidelity of the GFT software might have been compromised, thus the person(s) responsible for the software should be contacted. The screen in Figure 10 shows the result of testing the `sample` gridfile. The information below the line reading `test SUCCESSFUL` applies only to users with detailed knowledge of the Grid File's internal structures. Note that all output from this command is sent to standard error.

```
cmdtool - /bin/csh
% gridfile sample -z

   * ATTRIBUTE INFORMATION *
0. X (INTEGER : KEY) x-coordinate of tree
   Missing_value : -1
1. Y (INTEGER : KEY) y-coordinate of tree
   Missing_value : -1
2. Diameter (REAL : KEY) diameter 1 meter from base of tree
   Missing_value : -1.000000
3. Type (STRING : KEY)
   Missing_value : .

Page 0  Percentage Occupancy (average) 3.002192
test SUCCESSFUL: 9 entries

   * THE CURRENT SCALES *
Attribute 0    number of scales 2
1 10
Attribute 1    number of scales 2
1 10
Attribute 2    number of scales 2
-1.000000 74.500000
Attribute 3    number of scales 2
BU zz

   * THE CURRENT DIRECTORY *

ENTRY 0 ADDRESS: 0  OCCUPANCY: 9

 NEAR VERTEX: 0 0 0 0
 FAR  VERTEX: 1 1 1 1
%
```

Figure 10. Testing the `sample` gridfile

2.6 Extracting all the Data

UNIX Command: gridfile *gridfile_name* -e [*data_filename*]

At some point throughout the life of an application, other tools providing functionalities different from those of GFT are likely to operate upon the data. In anticipation of such a situation, this mode provides an ASCII interface which will extract all the data currently stored in a gridfile. All entries will be printed to a file when a filename is specified on the UNIX command line, otherwise they will be sent to standard output. Since this command is intended to be the dual of adding data to GFT, the structure of the output data is identical to the structure of the Addition data found in Section 2.2. Refer to Figure 11 for an example of the extraction process. Note that this mode does not alter the state of the gridfile in any way.

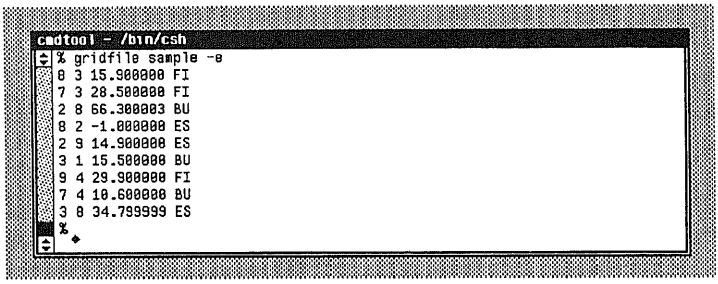

Figure 11. Extracting the `sample` gridfile entries

2.7 Deleting a Gridfile

UNIX Command: gridfile *gridfile_name* -d

After the utility of managing data in a gridfile has expired, the gridfile should be deleted. This command will delete a gridfile's three GFT system files resulting in the end of the gridfile's existence. As a precautionary measure, if the gridfile is not empty, the user will be asked if the gridfile should really be deleted. See Figure 12 for a sample session.

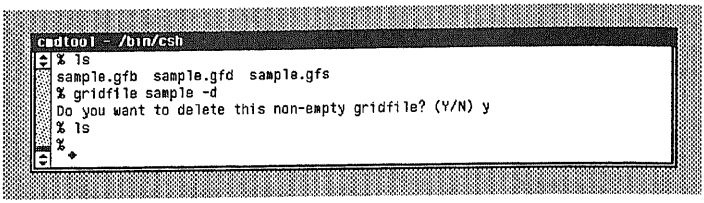

Figure 12. Deleting the `sample` gridfile

3. Options to GFT commands

This section explains three options which may be used in conjunction with the modes explained in the previous section: formatting the data, logging a session, and taking extra safety precautions. Although these options may be exercised with any mode, they might not always influence processing — for instance using the format option would not alter the results of the optimize command at all since no input or output data needs to be formatted.

3.1 Input and Output Format

Addition to UNIX Command: -f *format_filename*

Throughout a gridfile's life, it is often useful to input and output the data in a format different from the internal representation (as specified by the Profile outlined in Section 2.1). The Format File provides exactly this functionality by allowing the user to specify for each attribute a string for missing values, a field length, and a different ordering. Although these specifications apply to both input and output for a single GFT session, different specifications can be achieved by reading in one session and writing in another. Note that unlike the Profile specification, if an option is exercised it must be specified for *each* attribute — whether search or non-search (the exception being the missing string option which need only be specified for those attributes exercising the missing option in the profile). Figure 13 gives an overview of the options, including their affects (i.e. whether they merely provide a convenience for the user, or whether they also result in a loss of efficiency).

Formatting Option	*Effect of Usage*	*Option Indicator*	*Attribute Specification Format*
Reoredering of Attributes	Efficiency Loss	P	integer
Field Width I/O	Convenience	F	integer
Missing String	Efficiency Loss	M	"string"

Figure 13. I/O Format File options and Attribute Specifications

The simplest formatting options are reordering and field width. The reordering option simply indicates that an attribute should be printed out (or read in) in a different order from the regular ordering as specified in the Profile. The field width option is essential for fixed field width input or output formatting. The

field width option mainly offers convenience and might be more or less efficient (depending on the platform upon which GFT is run), although the reordering option is always slightly less efficient.

The missing string option is slightly more complex, and can be used only in conjunction with the missing value option of the profile file. As opposed to the profile missing value option, the missing string option denotes that a missing value should be represented externally to GFT differently from its internal representation. This flexability is especially convenient when the missing value is of a type different from its associated attribute. The exact interpretation of the missing string specification depends on the previous field width option. If the I/O is to be performed by field width, then the missing string specification must be of the same field width. In this case, however, the string need not represent a value of the same type as the attribute would normally be. On the other hand, if the field width is not specified, the missing string must represent a value of the same type as the attribute but may be of any length.

3.1.1 Sample Format File

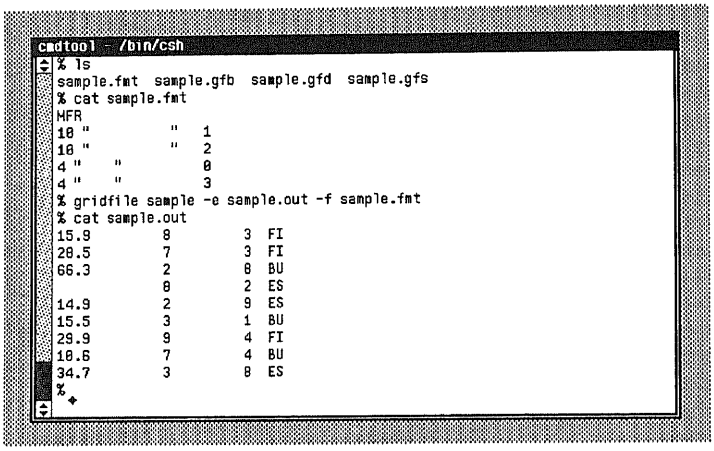

Figure 14. Extracting Data from the `sample` gridfile using a Format file

The session shown in Figure 14 uses a Format File to extract the data in the same manner as Section 2.6 except with a different format. The first line of the file indicates that the missing string (M), field width (F), and re-ordering (R) options will be exercised. The succeeding lines specify the options for each of the four attributes. This format dictates that the entries which are input/output should have the value of the third attribute first, the first attribute second, the

second attribute third, and the fourth attribute last. A field width for each attribute and the exact string to print for missing values are also given. When the data are extracted, the fourth entry contains a missing value in the first attribute which was output as a string of blanks.

The exact specification of the Format File is found in Appendix C.5.

3.2 Log File

Addition to UNIX Command: -l

This optional command line argument will cause all information about a single GFT session's processing to be written in a log file named *gridfile_name*.log. This option is especially useful when performing transactions since both the specification of each transaction and its output are recorded. Errors are also recorded to this file; thus if an error might occur during a batch process, the log file would be a good option to use.

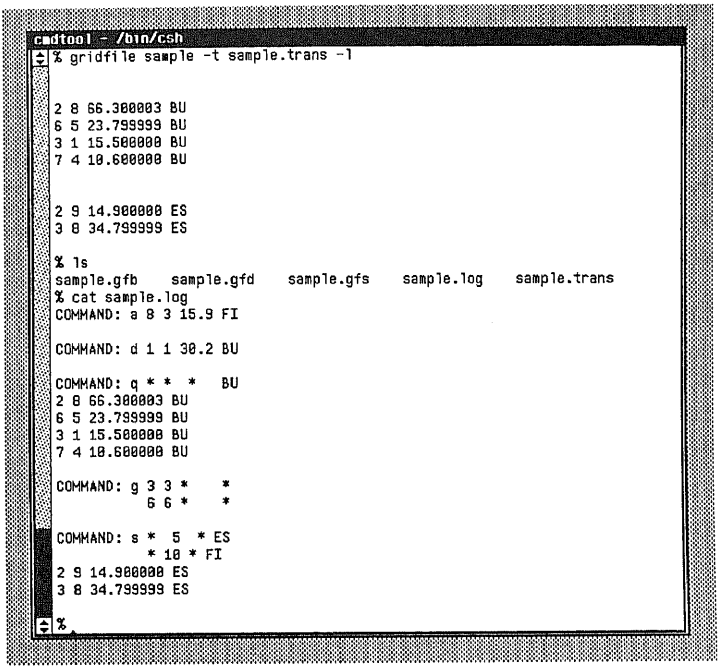

Figure 15. Performing transactions on the `sample` gridfile while keeping a Log file

The session shown in Figure 15 shows the process of performing transactions (using a transactions file containing the same transactions given in Figure 9),

while keeping a log file. Note that during the transactions processing, the results of each transaction are still sent to standard output (separated by blank lines), while both the input and output has been printed to the log file.

3.3 Safety Precautions

Addition to UNIX Command: -s

This option adds a certain amount of fault tolerance to GFT operations that change the gridfile contents. If the hardware system crashes or the UNIX process is unexpectedly terminated, this command line argument will greatly increase the likelihood that the gridfile contents are left in a consistent state. It does not, however, provide a 100% guarantee that consistency will be maintained. Exercising this option does slow down processing considerably, since it causes buckets that are normally cached to be written to disk after every atomic operation.

4. Interactive, Graphical Transactions

UNIX Command: sungrid *gridfile_name*

Due to its importance and frequency, transaction processing may be performed in GFT using two different interfaces: the non-graphical method (explained in Section 2.3), which is useful for input and output on any "dumb terminal;" and a more sophisticated, graphical method in which GFT takes advantage of the graphics capabilities of engineering workstations.

Graphical input and output for transactions has been provided on Sun workstations using the graphics packages SUNview and SUNcore. With this interface, the results of queries are shown graphically on a black/white monitor (instead of being written to the screen), while the transactions are input via both mouse and keyboard. In order to utilize this interface, GFT should be run from within SUNview (SUNtools will also suffice). Note, however, that sungrid must be called from the UNIX command line instead of gridfile since this interface uses a separate executable.

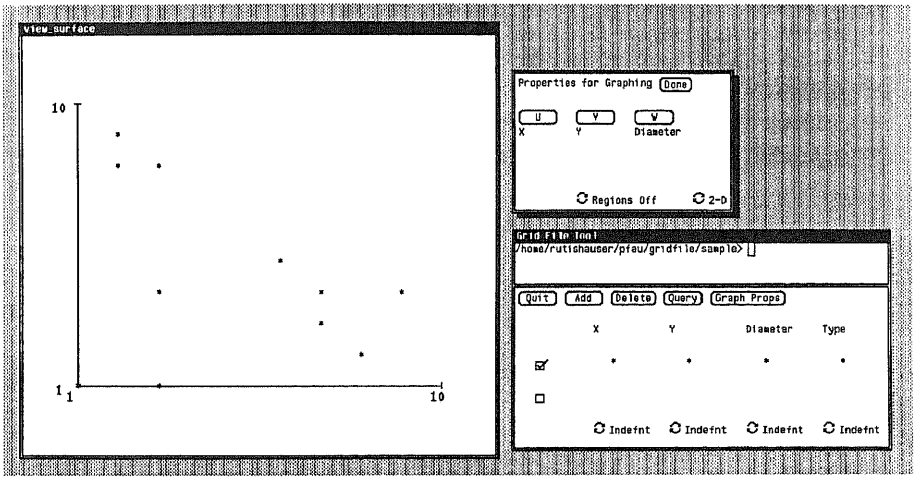

Figure 16. Graphical Interactive Transactions with the sample gridfile

The graphical interface is fairly self-explanatory (see an example of a screen in Figure 16), therefore only a short introduction is given here. The screen is organized to include a large view surface for plotting the results of queries and another window for specifying transactions. A third, smaller window will pop up in order to alter the graphing properties. The top of the transactions window

contains the transaction command buttons (along with two other buttons, to be explained); below these buttons lies a table containing the point specifications. The first column in the table contains a choice of one or two point specifications and the rest of the columns correspond to attributes. In the first row, the name of the attribute is displayed, and in the last row is the type of the attribute. The fourth row allows the user to cycle between definite and indefinite values for the attribute — a definite specification allows the user to type in the particular value, whereas indefinite values are automatically "*"s. The second and third rows correspond to the one or two points associated with the particular transaction. At the beginning of these rows boxes may be clicked to toggle between whether or not the point should be used in the transaction.

The three transaction commands are executed in post-fix order, thus a command button should be clicked on only after all the appropriate points are specified. The commands available are ADD, DELETE, and QUERY; for each command there is space for either one or two entries. Thus the exact command, in terms of range/point and definite/indefinite, is determined by attributes and entries desired (if two entries are given for the ADD command, then both will be added).

Besides the actual transaction commands ADD, DELETE, and QUERY, there are two other buttons QUIT and GRAPH PROPS. The first button allows the user to quit the session, and the second pops up a panel to control the graphing properties (i.e., which attributes are graphed, whether graphing is done in two or three dimensions, and whether the bucket regions of the gridfile should be superimposed on the query results).

Appendix A: Glossary

attribute: a property, quality, or characteristic of an entry to which a value can be assigned. An attribute's values must be of a specific type (e.g., integer).

definite point: a reference to an entry (or several identical entries) in which each of the attributes are quantified or specified. Typically used to specify a query.

entry: the object or record which is to be managed. Each entry can be described by a set of attributes, and is specified by giving values for each of its attributes. An entry is of cardinality one, thus several entries may be identical.

GFT (GridFile Tool): a particular implementation of a Grid File which runs under UNIX.

Grid File: a type of secondary-memory data management system which superimposes a large-grain grid over a data space, and stores all the data lying in one gridblock together.

gridfile: a specific instance of a GFT Grid File whose entries all have the same attributes.

indefinite point: a reference to a set of entries in which a proper subset of the attributes have been specified, while any values are acceptable for the rest of the attributes (since the user "does not care" about the value for these attributes, they are left unspecified).

indefinite range: a reference to a set of entries in hyperspace where two indefinite points specify the lower and upper bounds of the attributes. The same attributes in the two points must be unspecified.

point: a reference to an entry with a value for each attribute (these values may be unspecified in the case of an indefinite point, or must be completely specified in the case of a definite point).

profile: the meta-information, or schema, describing the nature of the entries in the gridfile (e.g., type).

range: a reference to a set of entries in a hyper-rectangle where two points specify the lower and upper bounds of the attribute dimensions.

symmetric access: searching the data structure without biasing any particular attribute, e.g. without restrictions on, or prejudice towards a particular subset of the attributes.

Appendix B: Summary of UNIX Commands

```
NAME
     gridfile - operate upon a multi-attribute dataset

SYNOPSIS
     gridfile gridfile_name [-n [profile_name]] [-l]
     gridfile gridfile_name [-a [data_filename] ] [-f format_filename] [-l] [-s]
     gridfile gridfile_name [-t [transactions_fname]] [-f format_fname] [-l] [-s]
     gridfile gridfile_name [-o] [-l] [-s]
     gridfile gridfile_name [-z] [-l]
     gridfile gridfile_name [-e [data_filename]] [-f format_filename] [-l]

DESCRIPTION
     gridfile manages a dataset on disk, where the dataset is composed of
     records described by a single set of attributes.  The first argument,
     which is required, identifies the particular dataset while the second
     specifies the type of operation(s) desired.

OPTIONS
     -a add data to gridfile_name.  If a data_filename is not specified,
        data will be assumed to be coming from stdin.
     -d delete an empty gridfile.
     -e output all the data of gridfile_name.  If a data_filename is
        specified, then the data will be put into this ASCII file,
        otherwise it will be sent to stdout.
     -f format the input and output data of gridfile_name according to
        the specifications found in format_filename.
     -l keep a log of all the transactions performed graphically.  The
        log is found in the file named gridfile_name.log
     -n create a new gridfile (to be referenced by gridfile_name)
        according to the specified profile_file. If no profile_file is
        specified, it is assumed to have the same name as gridfile_name.
     -o optimize the gridfile_name.
     -s take extra safety precautions when altering gridfile contents.
     -t perform transactions on gridfile_name.  If transactions_filename
        is not specified then the transactions are assumed to be coming
        from stdin.
     -z perform a consistency check on gridfile_name.

SEE ALSO
     sungrid

BUGS
     There are no interlocks and no reliable cache flushing; thus concurrent
     updating/reading and asynchronous operations are very risky.

FILES
     gridfile_name.gfb
     gridfile_name.gfd
     gridfile_name.gfs

Sun Release 4.0                    Last Change: 09 Feb 1990
```

NAME

sungrid - perform graphical transactions upon a multi-attribute dataset

SYNOPSIS

sungrid gridfile_name [-l][-s]

DESCRIPTION

sungrid performs graphical transactions on a dataset managed by gridfile. All transactions are input via mouse and keyboard, and the output from query transactions are graphically displayed as a scatterplot.

OPTIONS

-l keeps a log of all the transactions performed graphically. The log is found in the file named gridfile_name.log

-s takes extra safety precautions when altering gridfile contents.

SEE ALSO

gridfile

BUGS

There are no interlocks and no reliable cache flushing; thus concurrent updating/reading and asynchronous operations are risky.

FILES

gridfile_name.gfb
gridfile_name.gfd
gridfile_name.gfs

Sun Release 4.0 Last Change: 09 Feb 1990

Appendix C: User File Specifications

The following subsections fully specify the structure of the GFT input files provided by the user. The specifications are made in Backus-Naur Form as described below.

C.1 Backus-Naur Form (BNF)

The format of each file described in the rest of Section C is specified in Backus-Naur Form. The symbols in the specification lead to the generation of tokens which are either constant (e.g., EOLN) or specified by the user (e.g., 4). Constants will be enclosed in single quotes. Although the BNF usage is fairly standard, it will be summarized here. Each of these rules are to be followed, unless noted otherwise within the specification.

Each of the symbols requires a single token to exist, unless the symbol is contained within braces ({ }) or brackets ([ ]), or the symbol expands into several symbols (denoted by = followed by one or more symbols). A symbol (or sequence of symbols) within braces dictates that the symbol can be repeated any number of times, including none. Within brackets, a symbol or sequence of symbols is optional thus may be omitted.

All tokens forming a sequence (generated either by a sequence of symbols or by a symbol within braces), should be separated by whitespace. Whitespace characters include blanks, tabs, or newlines although in general, newlines (EOLN) should not be used unless specified in the BNF.

The type specifications, following the colon (:) are fairly standard. Perhaps STRING should be clarified — it refers to a set of characters, at least one character must exist, and although there may be any number of characters, none of them should be whitespace characters.

C.2 Profile File Specification

The structure of the Profile File is shown below. The first line of the file will contain overall information about the data, and on each following line, a different attribute will be described.

```
profile_file =
    total_number_of_attributes [ number_of_search_attributes ]
    [ number_of_bucket_entries ] option_indicators EOLN
    { attribute_type attribute_options EOLN }          EOF

  total_number_of_attributes : INTEGER
  number_of_search_attributes : INTEGER ( < total_number_of_attributes)
  option_indicators = { [ missing_value_indicator ] |
      [ precision_indicator ] | [ maximum_value_indicator ] |
      [ minimum_value_indicator ] | [ name_indicator ] |
      [ comment_indicator ] } : STRING (including the empty string)
  attribute_type = 'INTEGER' | 'REAL' | 'STRING' | 'SYMBOL' : STRING
  attribute_options = [ '"' missing_value '"' ] [ '#' precision ]
       [ '<' maximum_value ] [ '>' minimum_value ] [ name ] [ comment ]

    missing_value_indicator = 'M' : CHARACTER
    precision_indicator =     'P' : CHARACTER
    maximum_value_indicator = 'H' : CHARACTER
    minimum_value_indicator = 'L' : CHARACTER
    name_indicator =          'N' : CHARACTER
    comment_indicator =       'C' : CHARACTER
    missing_value :  INTEGER | REAL | STRING | SYMBOL
    precision :      INTEGER
    maximum_value :  INTEGER | REAL | STRING | SYMBOL
    minimum_value :  INTEGER | REAL | STRING | SYMBOL
    name :           STRING
    comment :        STRING (may contain whitespace, except EOLN)
```

Note that:

- The `total_number_of_attributes` should be greater than `0`. The `number_of_search_attributes` should be strictly less than the `total_number_of_attributes` but greater than `0`.
- The `number_of_bucket_entries` option should only be used in very rare cases where the application desires to control the maximum number of entries in each bucket. If this is specified, then `number_of_search_attributes` must also be specified (although it may be equal to `total_number_of_attributes`).
- The number of lines in the file should be one more than the `total_number_of_attributes` since each attribute specification (a `type`, and possibly `options`) should be contained on a single line.

- A particular `option_indicator` dictates whether the particular option might appear as one of the `attribute_options`. Although the indicators may appear in any order on the first line of the profile, the actual option specification must be in the specific order given in the BNF above.
- For each attribute, the type of `missing_value`, `maximum_value`, and `minimum_value` must be identical to that specified by its `attribute_type`.

C.3 Data File Specification

A Data File, which is input when adding data and output when extracting or performing transactions on data, is described below. In this file, each entry is completely contained on a single line, although the output when performing transactions might include blank lines. The attributes of entries are delimited with whitespace.

```
data_file = { [ definite_data_entry ] EOLN } EOF

  definite_data_entry = { definite_data_point }

    definite_data_point : INTEGER | REAL | STRING | SYMBOL
```

Note that:

- The number of `definite_data_point`s should be exactly equal to the `total_number_of_attributes` given in the profile. The types of these `definite_data_point`s should also correspond to the profile specification.
- Any missing datum should be specified by the value identical to the `missing_value` for that attribute given in the profile.

C.4 Transaction File Specification

The structure of a Transactions File, used as input to perform addition, querying and deletion operations, is as follows:

```
transaction_file = { add_command definite_point EOLN |
                     single_command point_entry EOLN |
                     double_command point_entry EOLN point_entry EOLN }
                   EOF

  single_command = delete_point_command |
                   delete_indefinite_point_command |
                   query_point_command  |
                   query_indefinite_point_command
  double_command = delete_range_command |
                   delete_indefinite_range_command |
```

```
                query_range_command  |
                query_indefinite_range_command
point_entry = definite_point | indefinite_point


  add_command =                          'a' : CHARACTER
  delete_point_command =                 'd' : CHARACTER
  delete_indefinite_point_command =      'e' : CHARACTER
  delete_range_command =                 'f' : CHARACTER
  delete_indefinite_range_command =      'g' : CHARACTER
  point_query_command =                  'p' : CHARACTER
  indefinite_point_query_command =       'q' : CHARACTER
  range_query_command =                  'r' : CHARACTER
  indefinite_range_query_command =       's' : CHARACTER
  definite_point =      { definite_value }
  indefinite_point =    { definite_value | indefinite_value }

    definite_value :    INTEGER | REAL | STRING | SYMBOL
    indefinite_value :  '*'
```

Note that:

- The number of `values` for each `point` should be equal to the `total_number_of_attributes` specified in the Profile.
- The type of the `definite_values` must be the same type as was specified by `attribute_type` in the Profile.

C.5 Format File Specification

The specification of the Format File, used to format the input from the user and output to the user, is similar in structure to that of the Profile File. The first line of the file contains a string representing the desired options. Each succeeding line, one for each attribute, contains the option specifications for each attribute. Unlike the Profile, however, if an option is indicated, it should be specified for each attribute.

```
format_file = format_option_indicators EOLN
              { attribute_format_option EOLN }

  format_option_indicators = { [ field_length_indicator ] |
      [ missing_string_indicator ] | [ reordering_indicator ] } : STRING
  attribute_format_option = [ field_length ] [ '"' format_missing_value '"'
                            [ reordered_number ]
    missing_string_indicator =    'M' : CHARACTER
    field_length_indicator =      'F' : CHARACTER
    reordered_indicator =         'R' : CHARACTER
```

```
field_length :          INTEGER
format_missing_value : STRING
reordered_number :      INTEGER
```

Note that:

- At least one `format_option_indicator` should be given otherwise this file serves no purpose. Each `attribute_format_option` should contain the corresponding option for each `format_option_indicator` that was specified. The ordering of the option specifications in the `attribute_format_option` is very important thus if a field length is specified, it should come before a missing value, etc.
- The number of lines in the file should be one more than the `total_number_of_attributes` since each attribute should have one or more `attribute_format_options` defined. The *nth* `attribute_format_option` pertains to the *nth* attribute specified in the profile file.
- If the reordering option is used, the reordering numbers specified for each dimension should entirely cover the range `[0 .. (total_number_of_attributes - 1)]`. These numbers indicate the placement of the attribute in the external entry specification.
- When the `field_length_indicator` is used in conjunction with the `missing_format_indicator`, the `field_length` dictates the length of the `format_missing_value` string. Otherwise, the `format_missing_value` should be of the same type as the attribute normally would be.
- The `format_missing_value` will only be recorded for those attributes which have missing values as specified in the profile. Although the format file will not be in error if values are specified in the format meaninglessly, they may be left out of the `attribute_format_option` — if the `missing_string_indicator` is the only option exercised, some lines might be blank.

Appendix D: Data Types and Representations

Currently three types of values can be stored and retrieved by the GridFile-Tool: string, integer, and real. The actual representation of these types, for example the number of bytes which an integer occupies, depends on the hardware in which GFT is running. These representations restrict the range and in the case of reals, accuracy, of possible values to be stored. The exact representations of the data types on two different hardware platforms are given below.

D.1 On the SUN-3 ...

String: A string is currently limited to three characters. These characters can, in theory, be any in the ASCII character set (0-127), although only the printable characters (i.e., 32 (' ') - 126 ('~')) provide guaranteed results.

Integer: Integers are represented in four bytes with no fractional part — their exact representation follows that of SUN. In affect, the range of integer numbers is `[ -2'147'483'648 .. 2'147'483'648)`.

Real: Real numbers are represented in four bytes and their exact representation follows that of the ANSI IEEE 754–1985 standard. Note that precision problems do arise - for instance, `685163.26` will be represented internally (and from thereon externally) as `685163.25`. This problem may be ameliorated by using a minimum option, if appropriate.

D.2 On the Cray X-MP ...

String: A string is currently limited to seven characters. These characters can, in theory, be any in the ASCII character set (0-127), although only the printable characters (i.e. 32 (' ') - 126 ('~')) provide guaranteed results.

Integer: Integers are represented in eight bytes but with a precision of only 46 bits — their exact representation follows that of Cray. The precision can be easily expanded to 64 bits by recompiling GFT with the faster integer arithmetic mode disabled (see the Cray Standard C Programmer's Reference Manual, SR-2074).

Real: Real numbers are represented in eight bytes — their exact representation follows that of Cray.

Appendix E: Status of GFT Functionality

Although this document describes the intended functionality of the software, some functions are either not currently implemented or do not achieve the expected results. Although all will be rectified in the future, currently the user is advised not to exercise the functions (or ignore its results).

- Minimum Values: these have not been implemented.
- Optimization Mode: this has not been implemented.
- Bucket Occupancy: this is terribly low, expect it to be around 50 percent.
- Types Represented: only integers, reals, and short strings have been implemented. Symbols, long integers, and double precision numbers will be implemented in the future.
- Empty Buckets: A drastically shrinking data set could cause some buckets to be empty. This will produce a warning message which may be ignored. If many buckets are empty, however, the data should be extracted, the gridfile deleted, the gridfile created anew, and the old extracted data should be added.

GFT's software and additional documentation can be acquired by contacting :

Dr. Hans Hinterberger
Department Informatik, Institute for Scientific Computing
ETH - Zentrum
8092 Zurich Switzerland
Tel: (01) 254-7436
Email: hinterberger@inf.ethz.ch

Bibliography

[1] Christos Faloutsos. A grid file for spatial objects. Technical Report UMIACS-TR-87-15 CS-TR-1829, University of Maryland, April 1987.

[2] Klaus Hinrichs. Implementation of the grid file: Design concepts and experience. *BIT*, 25:569–592, 1985.

[3] Hans Hinterberger. *Data Density: A Powerful Abstraction to Manage and Analyze Multivariate Data.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 1987.

[4] Andreas Hutflesz, Hans-Werner Six, and Peter Widmayer. Twin grid files: A performance evaluation. In Hartmut Noltemeier, editor, *Proceedings of the International Workshop on Computational Geometry*, pages 15–24, Würzburg, FRG, March 1988. Springer.

[5] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[6] Lise Pfau. File structure in the GridFile Tool. Draft, April 1989.

[7] Lise Pfau. GridFile Tool : Programmer's guide. Draft, April 1989.

Gelbe Berichte des Departements Informatik

117	N. Wirth	Modula-2 and Object-Oriented Programming. Drawing Lines, Circles, and Ellipsis in a Raster. Flintstone.
118	H.-J. Schek H.-B Paul M.H. Scholl G. Weikum	The DASDBS Project: Objectives, Experiences, and Future Prospects
119	J. Gutknecht	The Oberon Guide
120	D. Mey	A Predicate Calculus with Control of Derivations
121	H.P. Frei P. Schäuble M.F. Wyle	The Assessment of Information Retrieval Algorithms
122	P. Läuchli	An Elementary Theory for Planar Graphs
123	B. Wüthrich	Detecting Inconsistencies in Deductive Databases
124	C. Pfister	The Graphics Editor Condor The Layout System Pedro
125	R. Crelier	OP2: A Portable Oberon Compiler
126	A. Szyperski	Network Communication in the Oberon Environment
127	H. Mössenböck	Coco/R: A Generator for Fast Compiler Front-Ends
28	B. Sanders	Eliminating the Substitution Axiom from UNITY Logic
29	L. Pfau	GFT: A Tool for Data Management in the UNIX Environment