Constructing relationships in a hierarchical file system

Young Woo Yoon
Information and Communications University
zeroyy@icu.ac.kr

Abstract

We propose a scheme for more efficient navigation in a hierarchical file system. In the proposed scheme, a program running in the background computes the degree of relationship between a current file and others, and builds a list of the most related files. The current relationship metric being used by the program is a linear combination of five parameters: the name, the directory path, the type, the created time, and the last accessed time of a file. A simulated annealing algorithm is used in order to determine the weighting factors of the parameters. A set of experiments were conducted in order to access the effectiveness of the proposed scheme.

Keyword : graphical user interface, hierarchical file system, more related files, relationship metric, simulated annealing

1. INTRODUCTION

Most operating systems including Windows, UNIX, and Linux use hierarchical file systems. In a hierarchical file system, users usually arrange related files in the same directory and classify files in a tree according to the contents of files. However, a file in some directory may also have a relationship with files in other distant directories. For example, users may reuse a source code file that was written for another programming project for the current project. In this case, users have to travel several directories to reach the related files. Using a shortcut to a file in MS Windows or a soft link in UNIX systems, users can specify explicitly remote files that are related to files in the current directory. However, it is a troublesome work to specify relationships between files manually when there are a lot of files in a file system. It is too demanding, though not impossible, to specify relationships between all the files.

We propose in this paper a scheme that provides links to related files automatically for more efficient navigation in a hierarchical file system. The central part of the scheme is a computational method of constructing relationships between files and directories automatically. In the proposed scheme a navigation helper program suggests a list of related files with the current document on request. The ideal goal of the proposed scheme is to enable users to access related files mostly from the list of suggested files, thereby obviating the need to traverse up and down a complex directory hierarchy.

The overview of the whole system is described in Section 3, the implementation and design details are given in Section 4, and the results of some experiments with the proposed scheme is given in Section 5.

2. RELATED WORKS

Several researchers have tried to improve the usability of the hierarchical file system. One of the works is ‘Sifting through Hierarchical Information’ by Doug Schaffer and Saul Greenberg [1]. They tried to sift the files based on their attributes like a name, a type, a created time, and so on. Similar approaches can be seen in ‘Semantic File System’ [2] and ‘Improving the

Usability of the Hierarchical File System' [3]. These works consider the attributes of the files to classify the files. But the outcome is not beyond simple hierarchical structures. To overcome the shortcomings of these hierarchical structures, this paper proposes the web structure by assigning the relationships between the files.

3. SYSTEM DESIGN

The system that we implemented shows the related files for a file. Based on the file information, the system compares the source file with the others (target files) then shows ten files in order of the similarity. At here, the similarity means a kind of semantic similarity of the files. In other words, if some files are related to a project, the files have a high degree of the similarity.

3-1. Related Attributes

To construct the relationship automatically, we need some attributes that determines the degree of the similarity between the files. We just consider the fundamental properties of the files, not user-specified properties. Users have to write the descriptions for a few thousand of files. They dislike this bothering work [1] [4]. Therefore, we consider name, path, type (extension), created time, and last accessed time of the files. These attributes are primitive properties that the operating system supports. Moreover, users tend to make the meaningful name of the file according to the contents of the files. In addition, users classify the similar files into the same directories or the same hierarchies. Therefore, the name and path of the files can be good relevant attributes.

When the users are in a work, they may create the related files one after the other, and also users access the files sequentially. Accordingly, these two factors of created and last accessed time may be used to compare the similarity. We also consider the type of the files because users would access the files of a type to reference.

From these factors, the similarity between two files can be calculated through the relationship expression that gives a numeric value of the similarity between two files. Here, each property should be represented in numeric value to construct the relationship expression. The relationship expression is discussed in Section 2.2 and 4.3 in details.

For the name of files, the system uses the length of LCS (longest common subsequence) that is a well-known measure at comparing two strings. At here, the system divides the length of target file name to prevent taking advantage of the length of name from the length of LCS.

$$V_{name} = (\text{length of LCS}) \div (\text{length of target file name})$$

For the path of files, the system counts the number of same parent directories for the source and target file from the root directory. For example, suppose there are two files - 'c:\foo\bar\paper.doc' and 'c:\foo\present.ppt'. From root directory, they have two same parent directories 'c:' and 'foo'. Moreover, we assumed that a deep directory – a far from root directory - is more classified, so the system doesn't divide the value of the target file like above.

$$V_{path} = (\text{number of same parent directory})$$

For the type (extension) of file, the system marks the value one or zero. If two files have same type, the system marks one, otherwise zero.

$$V_{type} = (\text{same type ? 1 : 0})$$

For the time factors including the created time and the last accessed time, the system uses below expression basically.

$$(T_{base} - T_{diff})^2$$

For two files, the system calculates T_{diff} as the

difference of the time in second then subtract it from T_{base} that is 14400 (3 hours in second), and square finally. If the difference is smaller than T_{base}, this factor goes to zero because we assumed that if the time difference is big enough, the contribution of the time factor is definitely small. Similarly, we thought that the relationship between the time difference and its actual contribution in the expression is not in linear, so we put the square in the expression. If the time difference is small, the contribution may be relatively high in this expression.

$$V_{create} = (T_{base} - \text{CreateTimeDiff})^2$$
$$V_{access} = (T_{base} - \text{AccessTimeDiff})^2$$

If CreateTimeDiff is less than T_{base}, $V_{create} = 0$. In case of AccessTimeDiff < T_{base}, $V_{access} = 0$.

3-2. Relationship Expression

The factors discussed in previous section have numeric values but the relationship expression that describes the portions of the each factor is necessary to compare the similarity between two files. The system just uses the summation of each factor with coefficients as the relationship expression.

$$Similarity(\text{source file}, \text{target file}) = V_{name} \times C_{name} + V_{path} \times C_{path} + V_{type} \times C_{type} + V_{create} \times C_{create} + V_{access} \times C_{access}$$

In this expression, the coefficient C_x ($x = name, path, type, create, access$) should be decided properly to get good results. To discover proper coefficients, SA (Simulated Annealing) algorithm (variation of local search) is used [5]. The details are in Section 4.3.

4. IMPLEMENTATION

The system consists of three main functions; initial files and directories indexing, database maintaining, and retrieving related files. As progress of this study, we also implemented SA method to calculate the coefficients in relationship expression.

Basically, each part in the system uses a database that uses MySQL. The database contains the information of all document files and directories. Table 1 shows the schema of the database.

Column Name	Type	Comment
Index	Integer	Primary Key
Name	Varchar	
Path	Varchar	
Type	Varchar	
Create Time	Timestamp	
Access Time	Timestamp	

Table 1 Database schema of the system

4-1. Initial Indexing

At first, the system collects the information of the files and directories in the local disks. The indexing program searches all local disk drives then traverses all directories recursively as insert files and directories information in the database. At here, the system just concerns the document files including image, video, music, and source code because users spend their times with these files. Moreover, if there are many files in the database, the system takes a long time to calculates all similarities when users retrieves related files.

4-2. Database Maintaining

After initial indexing, the system should maintain the database up-to-date. The background program captures the changes in local drives then updates the database. The system concerns the changes of file name, directory name, and last accessed time.

4-3. Relationship Expression Constructing (Simulated Annealing)

Each term in the expression has the difference ranges.

Therefore, we added some adjusting multiplications that are independent to the final solution of the SA. This is final form of the expression.

$$Similarity(\text{source file}, \text{target file}) = V_{name} \times 100 \times S_{name} + V_{path} \times 10 \times S_{path} + V_{type} \times 100 \times S_{type} + V_{create} \div 10^6 \times S_{create} + V_{access} \div 10^6 \times S_{access}$$

For this expression, we designed the SA to find five unknowns S_x.

- Initial Solution (S_{name}, S_{path}, S_{type}, S_{create}, S_{access}): (3, 3, 3, 3, 3), The absolute value of the solution has no meaning because the relative values of each factor decide the similarity. Initially, we supposed the significances of all factors are same.

- Initial Temperature: 1.0, the system takes a long time to process, so we set the initial temperature to 1.0 that is a small value.

- Update Temperature: Multiply 0.95 to the temperature. According to the initial temperature, we made the multiplication. At this environment, the system reaches to the lowest bound of the temperature at least once in most case.

- For Statement Repetition: 5, same reasoning with above.

- Perturb Solution: Choose one unknown at random then add a random number x (-0.5 < x < 0.5)

- Boltzmann Constant: 3, SA may goes to worse solution based on the Boltzmann constant and the temperature. When the constant is 3, the solution goes worse at 20%. Surely, the solution tends to goes worse at an early stage according to the temperature.

- Terminate Condition: Time duration in sec, we put 28800 (8 hours) at main experiments.

- Cost Function: Number of matches, when compare two solutions, our SA compares the number of matches between user-chose related files and computer-chose related files that uses the relationship expression.

4-4. Retrieving Related Files

If user gives a source file, the program grades all the files and directories based on the relationship expression. After that, the program shows a few files in top place.

5. EXPERIMENTS & RESULTS

We built up four databases for five people. For acquaintances, we asked to participate in the experiment and they accepted. The participants are male, good hand at computing and in 22-33. They chose three source file and ten related files for each source file. Choosing some files in same directory is permitted and they chose the source files they like.

For each user, we had the experiments including the 8 hours – terminate condition in SA – solution searching work. The experiments find final solution, coefficients, and a number of matches between user-chose related files and computer-chose relate files among 30 target files when the expression uses final solution.

Participant	Final Solution
	Number of matches
User 1	(0.1556, 3.9458, 0.2148, 0.3033, 0.5681)
	12 (40.00%)
User 2	(0.5195, 3.1078, 4.1068, 1.9653, 0)
	13 (43.34%)
User 3	(0.0195, 1.8966, 0.1678, 1.2202 0.7789)
	4 (13.34%)
User 4	(0, 0.5018, 0.2899, 3.9875, 1.2179)
	9 (30.00%)

User 5	(3.9848, 0, 0.9751, 3.9131, 5.0168)
	2 (6.67%)

Table 2 Experiment results after solution searching (SA). The solution consists of five values that represent the coefficient of each term in the relationship expression. This table also shows the performance in number of matches for each solution.

There are few participants so we couldn't establish one integrated solution. However, we need one integrated solution or some automatic construction without experiments because this experiment, searching final solution, is very bothersome works for users. In this experiment, users have to choose 33 files including 3 source files and 30 target files. In addition, they have to do a long time simulated annealing works. It will be discussed in future works.

5-1. Experiment for the factor contributions

In the previous experiment, the relationship expression was discovered. However, the coefficient of the term of path is relatively high and the term of last accessed time is low. This is an obvious result because the adjusting multiplication is not accurate. Anyway, we were anxious about the contribution of the factors. Therefore, we had additional experiment to verify the relationship expression.

Coefficients in the expression	User 1	User 2	Average
(1, 0, 0, 0, 0)	3	6	4.5
(0, 1, 0, 0, 0)	3	11	7
(0, 0, 1, 0, 0)	0	0	0.0
(0, 0, 0, 1, 0)	1	8	4.5
(0, 0, 0, 0, 1)	9	7	8
Average	3.2	6.4	4.8
(1, 1, 1, 1, 1)	10	9	9.5
(1000, 1, 1, 1, 1)	4	6	5
(1, 1000, 1, 1, 1)	12	9	10.5
(1, 1, 1000, 1, 1)	10	10	10
(1, 1, 1, 1000, 1)	4	8	6
(1, 1, 1, 1, 1000)	9	7	8
Average	8.17	8.17	8.17
(1, 1, 1, 1, 0)	7	9	8
(1, 1, 1, 0, 1)	10	8	9
(1, 1, 0, 1, 1)	5	8	6.5
(1, 0, 1, 1, 1)	6	9	7.5
(0, 1, 1, 1, 1)	10	8	9
Average	7.6	8.4	8

Table 3 Results of the experiment for the factor contribution: The table shows number of matches for several solutions.

Table 3 shows the results of the experiments. According to the experiment design, absolute value of the coefficient is not important but the ratio between the coefficients as described before. When the expression just concerns one factor, the average performance is 4.8 in number of matches. However, there is 8.17 when the expression concerns all factors. It's larger than the case of (0, 1, 0, 0, 0) that concerns path term. Therefore, we can say not only directory factor but also the others contribute to the results. In addition, user 2 has zero at last accessed time term in final solution. However, (0, 0, 0, 0, 1) give a proper result, so the term will not have zero contribution. In addition, (0, 0, 1, 0, 0) gives 0 matches, however, (1, 1, 1, 1, 1) is much better than (1, 1, 0, 1, 1). In conclusion, all factors in the expression contribute to determine the similarity.

5-2. Experiment for time factor model

The system uses special expression for the time factors.

Model 1:

If (TimeDiff < T_{base} *)*

Return ($T_{base} - T_{diff}$*)*$^2 \div 10^6 \times S$*;*

Else

Return 0;

However, is this expression really good? To verify this approach, we made another model that describes the time factors.

Model 2:

If (TimeDiff ≠ 0)

Return $1 \div T_{diff} \times 300 \times S;$

Else

Return $300 \times S;$

For each model, we had the experiments for two people. Based on two models, the system found the solution using simulated annealing algorithm then shows maximum matches.

	User 1	User 2
Model 1	12	13
Model 2	11	12

Table 4 Results of the experiment for the time factor model in number of matches.

The results show Model 1 is better than Model 2 even it is a small difference. We couldn't say Model 1 is best but our model (Model 1) for the time factors is good enough to use.

6. CONCLUSION

This study shows simple method to construct the relationships in the hierarchical file system. The system that classify based on one attribute is not enough. By considering multiple factors, we made the relationship expression that marks similarity between two files. Through this expression, our system finds the related files. Users can open the files what they wants through the relationship expression that combines multiple attributes of the files. Through the experiments, we knew that the expression after solution searching gives acceptable and good results. We think, in real application, this system can be implemented like recent files list.

7. FUTURE WORKS

7-1. User Interface

This paper didn't discuss about the user interface but it's very important part. Users tend to avoid executing additional program. They want to do inside a current running program. Therefore, our system should be integrated to the current programs. At first, we can think a combining with the file explorer. However, users really find the related files in the file explorer? We didn't traced user's activity but we think users find the related files when they use word processor, document viewer, editor, and so on. Therefore, our system should be integrated into the actual editing and viewing program not the file manager. Many editing programs support recent files tab when users try to open a document. The related files may be displayed in the same interface with the recent files. Likewise, the application may have a section of the related files in the menu.

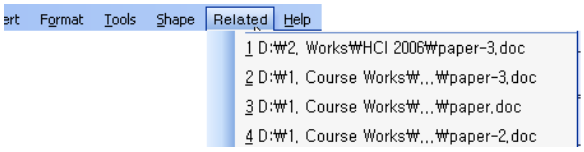

Figure 1 Proposed design of the user interface in the menu.

7-2. Environment Adaptive System

Each user has different ways to organize the files. Therefore, the relationship expression in the system should be different according to user. As you see in the experiments, two users have the different coefficients in the relationship expression. Most simplicity way is a user-specified expression. Users may modify the coefficient or expression model by themselves. More convenient way is re-doing our experiment. Users can

search the good coefficients according to our experiment procedure – simulated annealing method. However, these ways requires user's efforts. It's not our wish. If possible to analysis the user's style and construct the personalized expression automatically, it will be great.

REFERENCES

[1] Doug Schaffer and Saul Greenberg, "Sifting through Hierarchical Information", INTERACT '93 and CHI '93 conference companion on Human factors in computing systems, p.173-174, April 1993

[2] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, James W. O'Toole, Jr, "Semantic File Systems", Proceedings of the 13th ACM Symposium on Operating Systems Principles, p. 16-25, 1991

[3] Gary Marsden and David E. Cairns, "Improving the Usability of the Hierarchical File System", Proceedings of SAICSIT 2003, p. 122-129, September 2003

[4] Kerry Rodden and Kennth R. Wood, "How Do People Manage Their Digital Photographs?", CHI Letters Volume No. 5, Issue No. 1, April 2003

[5] Stuart Russell and Peter Norvig, "Artificial Intelligence – A Modern Approach", 2nd edition, p. 115-116

[6] Deborah Barreau and Bonnie A. Nardi, "Finding and Reminding", SIGCHI Bulletin, Volume 27, Number 3, p. 39-43, July 1995

[7] Paul Dourish, W. Keith Edwards, Anthony Lamarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas B. Terry, and James Thornton, "Extending Document Management Systems with User-Specific Active Properties", ACM Transactions on Information Systems, Vol. 18, No.2, p. 140-170, April 2000