論文 / 著書情報
Article / Book Information

論題(和文)	ユーザ作業を反映する仮想ディレクトリ生成 のためのアクセス履歴解析手法
Title(English)	Access-Log Analysis for Virtual Directory Creation to Restore Files Used in User ' s Works
著者(和文)	小田切健一, 渡辺陽介, 横田治夫
Authors(English)	Kenichi Otagiri, Yousuke Watanabe, Haruo Yokota
出典(和文)	情報処理学会研究報告, Vol. 2009-DBS-148, No. 4, pp. 1-8
Citation(English)	IPSJ SIG Technical Report, Vol. 2009-DBS-148, No. 4, pp. 1-8
発行日 / Pub. date	2009, 7
権利情報 / Copyright	

ユーザ作業を反映する仮想ディレクトリ生成のためのアクセス履歴解析手法

小田切 健一[†1] 渡辺 陽介[†2] 横田 治夫[†1,†2]

近年、個人や企業の扱うファイル数が増加し、ディレクトリ構造だけでは適切に管理しきれなくなっている。そのため、ある作業に関するファイルが複数のディレクトリに分散してしまう事や、他のファイル群に埋もれてしまう事がある。本研究では、そのようなファイル群を発見し、集約して仮想的なディレクトリとして提示することを目的としている。我々の過去の研究ではファイル同士の使用時間の重複回数や時間を数値化し階層的クラスタリングを用いて同一作業のファイル群の発見を行った。しかし、関連ファイルが近い時間に使われていても使用時間の重複がない場合に発見できないという問題があった。そこで、今回の手法では同一期間内のアクセスを同じトランザクションに入れ、アプリオリアルゴリズムによる頻出集合発見を行って目的のファイル集合を得る。使用時間が厳密には重複しないファイルの発見が可能になりリコールの改善が期待される。本稿では手法の解説と過去の提案手法との比較実験を行う。

Access-Log Analysis for Virtual Directory Creation to Restore Files Used in User's Works

Kenichi Otagiri,[†1] Yousuke Watanabe[†2]
and Haruo Yokota[†1,†2]

Due to the increase of the amount of digital data stored in computers, the number of files is increasing. It is difficult for users to maintain appropriate directory structures for large number of files. Some files used in a user's work often disperse and get lost in a file system. Our research group has been developing a system which rediscovers files used in the same work and provides virtual directories including these files. In our previous work, we proposed a file discovery method that computes a score of interfile relationship based on overlaps of file-access time obtained from access logs, then groups files by a hierarchical clustering method. However, the method cannot discover related files that are accessed separately within a short time. This paper proposes another method which divides access logs into transactions by a constant interval, and mines frequent file sets from the transactions. Unlike the previous method, the proposed method can group files used in user's work even if their accesses do not overlap.This paper also includes experimental evaluation comparing the proposed method and the previous method.

1. はじめに

近年のデジタル技術の発展に伴い、個人や企業のコンピュータで扱うデータ量やファイル数が増加している。そのため、非常に注意深く管理しないとある作業に関するファイルが他の作業ファイルに埋もれてしまう。さらに、ファイル数の増加のせいで埋もれたファイルを探すコストも上がっている。そのため、埋もれてしまった同一作業にかかわるファイル群を効率的に発見する仕組みが必要とされている。

本研究グループでは、そのような異なるディレクトリに分散してしまった同一作業に関するファイル群をアクセスログの情報を用いて発見し、仮想的なディレクトリとして集約して提示する手法の提案を行っている。過去の研究[1)2)]ではファイルの使用時間同士の重複時間・重複回数などを用い、ファイル間の関係の強さをスコアで表し階層的クラスタリングを適用することにより同じ作業に属していると思われるファイル群の集合を得た。評価実験を通し、ファイル同士の使用時間の重複時間の長さをファイル間の関係の強さとし、そのデータに対し階層的クラスタリングの一種である単連結法[3)]を用いることが良いという結果が得られた。しかし、同じ作業で用いたファイルであっても他のファイルの前後にアクセスしていて同時にアクセスすることがないファイルを発見できないという問題があった。

本論文では、上記の問題を解決するために同一期間内にアクセスのあったファイルを同じトランザクションに入れ、アプリオリアルゴリズムで全トランザクションを通して頻出するファイル集合を発見する手法を提案する。ファイル間の関係を使用時間の重複ではなく、ある一定期間内で使ったかどうかで判断するため、前後で使用したファイルも発見できると考えられる。本論文では、新しい提案手法が前回の提案手法に対し精度・リコールが総合的に上昇することを示すために、前回の手法と新しい手法の F 値を実験を通して比較する。

†1 東京工業大学院情報理工学研究科計算工学専攻
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 東京工業大学学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Intitute of Technology

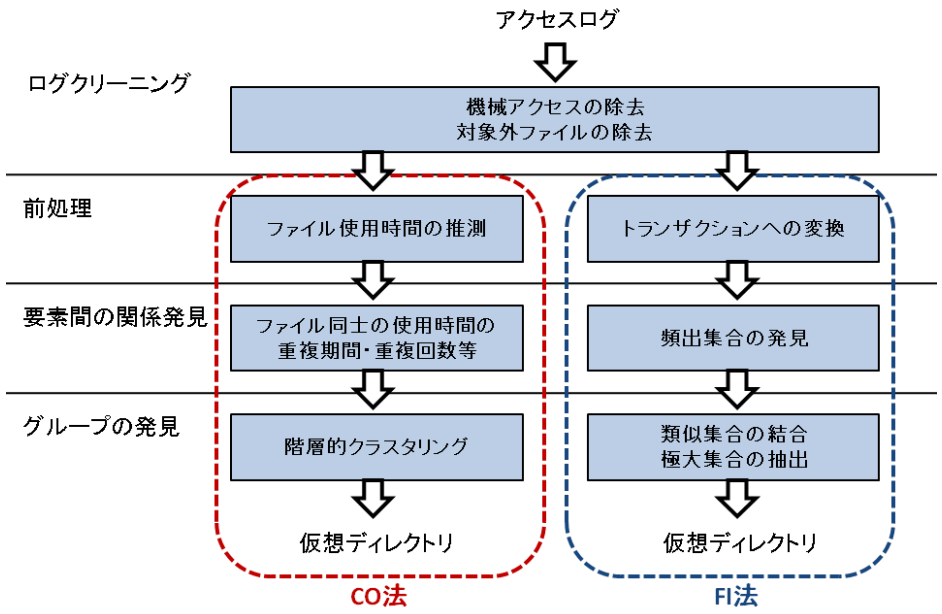

図 1 仮想ディレクトリ生成のフレームワーク

本論文の構成は以下の通りである。2 節では仮想ディレクトリ生成のフレームワークについて解説する。3 節では過去の提案手法について解説する。4 節では本論文で提案する手法について解説する。5 節では提案手法の最適パラメータの探索と過去の提案手法との比較を行う。6 節で関連研究を紹介する。7 節でまとめと今後の課題を述べる。

2. 仮想ディレクトリ生成のフレームワーク

本研究で述べる仮想ディレクトリとは一つの作業に属するファイル群を仮想的なディレクトリとして表してユーザに提示するものである。例としてユーザが、画像ディレクトリ・論文ディレクトリ・実験データディレクトリ・関連論文ディレクトリ等にファイルを配置して論文執筆を行ったとする。数ヶ月後、ユーザがその論文に使ったファイル群を再利用しようと思っても、配置していたディレクトリを忘れていることがあり、その場合はファイルシステムを全探索して人手で探す必要がある。しかし、本研究のフレームワークは、画像ディレクトリ・論文ディレクトリ・実験データディレクトリ・関連論文ディレクトリにおかれた論文執筆に使用したファイル群を発見し一つの仮想ディレクトリにその全てのファイル群へのリンクを集約する。これにより、ユーザは過去の作業の配置を忘れても、仮想ディレクトリを見ることで各ファイルの発見やファイル配置の確認ができる。

実際のシステムの動作としては、ファイルサーバのアクセスログとパラメータの入力に対し、システムは同じ作業に属すると考えられるファイル群を複数発見しそれぞれを仮想ディレクトリとして出力する。

仮想ディレクトリ生成の処理手順は図 1 の通りである。図の左側は過去に提案した、ファイル使用時間の重複に基づくクラスタリング法（CO 法：Clustering using Overlap of file-use）の処理手順を表している。図の右側は今回提案する、アプリオリアルゴリズムによる頻出ファイル集合発見法（FI 法：FrequentItemsets discovery）を表している。各ステップは以下のような処理を行う。

(1) ログクリーニング：仮想ディレクトリ生成にとって無意味なアクセス（find, DLL やプラグインへのアクセス等）や提示すべきでないファイル（中間ファイル等）をログファイルから削除する。
(2) 前処理：ログファイルに記述されたプリミティブな情報（OPEN,CLOSE）を、その後の処理に適切なデータ型に変換する。
(3) 要素間の関係発見：前処理で用意されたデータを元にファイル間の関係を発見する。
(4) グループの発見:発見された要素間の関係を元に関係する大きな集合を発見する。

2.1 ログデータ

本研究ではアクセスログとして Samba[4] が出力するアクセスログを使用する。Samba のログ出力設定にはログレベル 2 を用いる。レベル 2 の Samba のアクセスログにはユーザ名・時刻・ファイル名・アクション（Open/Close）が記録されており、CO 法・FI 法は共にこのログを利用する。

2.2 出力について

両手法ともに発見した作業数の仮想ディレクトリを出力する。出力する際のインターフェイスについては本論文では論じない。CO 法と FI 法では出力される仮想ディレクトリが若干異なり、CO 法は出力される仮想ディレクトリ間に重複がないのに対し、FI 法では出力される仮想ディレクトリ間に重複を含むことがある。

3. ファイル使用時間の重複に基づくクラスタリング法（CO 法）

過去の研究において提案した、ファイル使用時間の重複に基づくクラスタリング法（CO 法：Clustering using Overlap of file-use) について解説する。この手法は要素間の関係発見にファイル使用時間の重複を利用している。これは「図表を見ながら論文を書く」といった作業を想定し、関連するファイルであれば同時に使うはずだという考えに基づいている。その後発見した重複を数値化し、使用時間の重複関係が密なファイル集合を一つの作業としてクラスタリングにより発見する。

3.1 各処理手順

(1) ログクリーニング：FI 法と共通なため詳細は 4.1 節で解説する。

(2) 前処理：この手法ではユーザがファイルを使用していた正確な時間が必要だが、ファイルを開く際メモリに読んですぐに閉じてしまうアプリケーションではその時間がわからない。そのため、ユーザが PC を操作していた期間のファイルの最初のアクセスから最後のアクセスまでをユーザがそのファイルを触っていた時間として推測する。なお、ユーザ PC を操作していた期間もなんらかのファイルアクセスがあるかどうかで推測する。

(3) 要素間の関係発見：ファイル同士の使用時間の重複に着目し、重複時間・重複回数・重複開始のずれ・重複が起こる間隔を得る。

(4) グループの発見:前述の情報をもとにノードをファイルとした重み付きネットワークを作成する。そのネットワークに対する階層的クラスタリングにより関係が密な集合を得る。

3.2 問　題　点

1 節で述べたように、CO 法には使用時間が厳密に重複しているファイルしか発見できないという問題があった。同じ作業に関係するファイルを同時に複数開いて編集閲覧することは確かにあり、このアプローチは間違っていない。しかし、関連するファイルであっても前後で使用することもあり、この場合は使用時間が厳密には重複しない。特にファイル数が多い仕事や作業時間が長い仕事では関係あるファイル同士でも触った時間は数分〜数日離れているということがあり得る。CO 法ではそのようなファイルに対する配慮が欠けていたため、発見することができなかった。

4. アプリオリアルゴリズムによる頻出ファイル集合発見法（FI 法）

本論文で提案する、アプリオリアルゴリズムによる頻出ファイル集合発見法（FI 法：Frequent Itemsets discovery）について述べる。CO 法の問題であった厳密に使用時間が重複していないが関連するファイルを発見するため、使用時間の重複ではなく一定期間内にそれらのファイルのアクセスがあるかどうかという情報を用いる。これにより、CO 法では発見できなかった前後で使用したファイルも発見できると考えられる。しかし、この手法では関係のないファイルを含みやすい。そこで、アプリオリアルゴリズムによる頻出集合発見を行い、繰り返し出現するファイルの組み合わせを発見する。繰り返して近い時刻に使われるファイルであれば、無関係なファイルは含まれないと考えられる。よって、この処理で関連するファイル同士を発見できると考える。しかし、実際には作業には様々なファイルが含まれている。例として、プログラム・出力データ・データをまとめた文書を使う作業があるとする。これまでの処理で「プログラム、出力データ」「出力データ、データをまとめた文書」が発見できたとしてもまだ 1 つの作業が 2 つに分割されたままである。そこで、一部重複がある集合同士を結合する事を考える。この例の場合は出力データが共通要素としてあるので、結合処理が行われると「プログラム・出力データ・データをまとめた文書」という集合が生成され、作業全てを包含した集合を発見できる。

以上のアイディアを具体的な処理として表すと、「一定期間にアクセスされたファイル群を一つのトランザクションに入れる」「全トランザクションを通して頻出する集合を頻出集合発見アルゴリズムでマイニングする」「一定以上重複のある集合同士を結合する」の 3 ステップからなり、フレームワークの各ステップに対応する。以降、処理の詳細な手順について述べる。

4.1 ログクリーニング

ログクリーニングについて解説する。この処理手順は CO 法でも共通である。

4.1.1 機械アクセスの除去

アクセスログにはユーザが意図的にファイルに触ったアクセス以外にもユーザの意図と関係なく起こったファイルアクセスも記録されている。例として、バックアップ作業、スライドショー、ファイル検索、アプリケーションが自動で読み込む設定やプラグイン、等がある。これらはユーザが意図的に触ったものではないので結果として出力されてもメリットはない。

こういったアクセスは人手によるアクセスに比べて、短時間に大量のファイルに触る傾向

がある。そのため１秒間にスレッショルド T_{MA} 以上のファイルを触っている瞬間のログを削除することで機械的アクセスを除く。このスレッショルドの適切な値は 5.3 節で実験を通して決定する。

4.1.2 対象外ファイルの除去

今回の評価実験で対象とならないアプリケーションプログラムや中間ファイル等をログファイルより除去する。人手で不要なディレクトリや不要なファイル名、不要な拡張子を指定する。例としては、.class .exe .aux desktop.ini 等である。（特に desktop.ini は Windows のフォルダ表示設定ファイルであり常にアクセスされているため、本手法に悪影響を及ぼしやすい）

4.2 前処理：トランザクションへの変換

ファイルアクセスログを $TransactionTime$[sec] ごとに分割し、その中でアクセスがあったファイルを各トランザクションへ入れる。なお、ログに記録されているアクセスは Open/Close のみであり、Read/Write は記録されていない。そのため実際には TransactionTime で区切った期間内で Open/Close のあったファイルをトランザクションに入れている。TransactionTime の適切な値は 5.4.1 節で実験を通して決定する。

例として、「12:00:01 a.txt」「12:05:54 b.txt」「12:08:16 c.txt」「12:11:25 d.txt」「12:15:26 c.txt」のようなアクセスがあり、$TransactionTime = 600[sec]$ とすると、トランザクション 1=「a.txt, b.txt c.txt」・トランザクション 2=「d.txt, c.txt」となる。

4.3 要素間の関係発見：頻出集合の発見

上記過程で作ったトランザクション群から頻出集合を発見する。今回はアプリオリアルゴリズム[5] を用いた。アプリオリアルゴリズムで指定された $MinSupport$ 以上の頻出集合を発見する。特に MinSupport を大きく設定した場合に偶然同時に利用されたケースが排除されて高い精度の集合が得られると期待する。この MinSupport の適切な値は 5.4.2 節で実験を通して決定する。

例として、トランザクション「Task1-1.txt, Task1-2.txt, Task2-6.txt」「Task1-1.txt, Task1-2.txt, Task4-9.txt, Task8-2.txt」「Task1-1.txt, Task1-2.txt, Task5-2.txt」を考える。TaskX-Y は「仕事 X に使った Y 番目のファイル」という意味である。前述のトランザクションに着目すると、Task1 と Task2,5,8 のファイルが混在して出現していることがわかる。しかし同時に出現する回数に着目すると「Task1-1.txt,Task1-2.txt」は 3 回出ているが他の組み合わせは 1 回以下しか出ていない。よってサポート 2 回以上で頻出集合の発見行えば「Task1-1.txt,Task1-2.txt」以外の組み合わせは発見されず、関係ない作業のファイルは発見されない。

4.4 グループの発見

4.4.1 類似集合の結合

「ファイル A,B,C,D」と「ファイル A,B,C,E」が頻出するとわかった場合、「D,E」は組み合わせとしては頻出しないが D も E も「A,B,C」と同じ作業に属していると考えられる。そこで、このように重複がある集合同士を結合する。これにより、同じトランザクションには入らなかったが同じ作業に属するファイルを発見する。重複の度合いは Dice 係数を用いて測る。集合 A,B があるとき、結合の基準は以下のとおりである。

$$\frac{2|A \cap B|}{|A| + |B|} \geq T_{Comb}$$

この T_{Comb} の適切な値は 5.4.3 節で実験を通して決定する。

例として、「G1:A,B,C,D」「G2:A,B,C,E」「G3:A,F,G,H」という集合を考える。Dice 係数のスレッショルドを 0.5 とする。各集合間の Dice 係数を計算すると (G1,G2)=0.75, (G1,G3)=0.25, (G2,G3)=0.25 となり、G1 と G2 のみが統合される。よって、統合処理後には「G1+G2:A,B,C,D,E」「G3:A,F,G,H」という集合が残る。

4.4.2 極大集合の抽出

この時点である集合が他の集合の部分集合になっている場合がある。例として「A,B,C,D」,「A,B」,「B,C」,「C,D」といった集合が存在する場合、ユーザには「A,B,C,D」が提示されれば他の集合は必要ない。そのため、ここで他の集合の部分集合になっている不要な集合を削除する。

4.5 出　力　例

本手法で出力された結果の一部を図 2 に示す。１つ目の集合は論文作成に関連したものである。複数のディレクトリに分散して置かれているファイル群を発見できており、ユーザにとって有用な集合である。ただし 1 つ無関係なファイルも含まれてしまっている。２つ目の集合には講義のレポートとその際に利用したプログラムの一部が発見されている。このような集合が複数出力される。

5. 実　　験

CO 法と FI 法の比較を行うために、両手法の最適パラメータを決定した。具体的に 6 つの正解セットに対し平均 F 値が最大になるパラメータを探索した。FI 法はパラメータが 3 つあるので、1 パラメータごとに決定を行った。

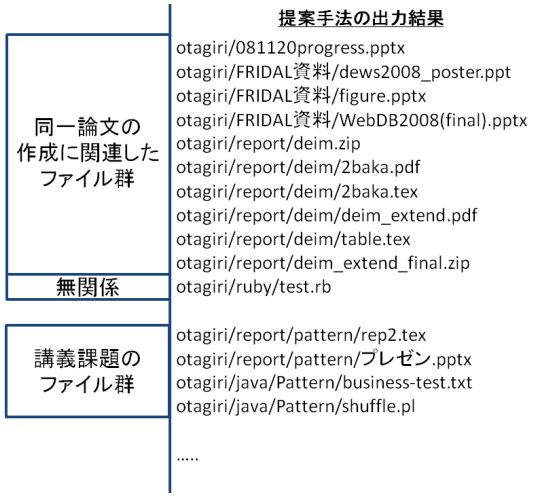

図 2 出力例

表 1 正解セット

	ユーザ	内容	ファイル数
セット S_1	ユーザ 1	実験データ	9
セット S_2	ユーザ 2	論文・発表資料	57
セット S_3	ユーザ 2	講義のレポート・プログラム・発表資料	84
セット S_4	ユーザ 2	講義のレポート・プログラム	32
セット S_5	ユーザ 3	論文・発表資料	33
セット S_6	ユーザ 3	CGI 作成	34

5.1 実験データ

本実験では 3 ユーザのアクセスログを用いた。それぞれの期間は、ユーザ 1:2008/06/10〜2009/06/11, ユーザ 2:2008/04/08〜2009/05/26, ユーザ 3:2008/08/07〜2009/05/29 である。

正解セットは 3 ユーザ合計 6 セットを用いた（表 1）。それぞれの正解セットは一つの作業単位を示しており、それらのファイル群は 2 ディレクトリ以上に分散している。

5.2 評価方法

まず、生成された仮想ディレクトリの中に正解セットが含まれる集合があるかどうかチェックする。なければ発見ファイル数は 0 である。正解セットが含まれる集合があっても、その集合に正解セット中の 1 ディレクトリ中のファイルしか入っていなければカウントしない。例として、正解セットが「A/a.txt, A/b.txt, B/c,txt, C/d.txt」であった場合、生成された集合内に「A/a.txt, B/c.txt」が含まれていれば正解数 2 と数えるが、「A/a.txt, A/b.txt」しか含まれない場合は複数のディレクトリを含んでいないので正解数は 0 となる。正解セットが複数の集合に含まれていた場合は、F 値が最大になるものを選択する。

正解数と不正解ファイル数が分かったら精度・リコール・F 値を計算する。その後全ての集合の F 値を平均し、その大小で良否を判断する。

5.3 ログクリーニングのパラメータ調整

機械アクセス除去のスレッショルド T_{MA} を決定するための実験を行った。この実験には 3 ユーザの合計 6 ファイル・26 サンプルの人手によるアクセスを利用した。このログに対し T_{MA} を変化させて 4.1.1 節の機械アクセス除去を適用し、人手のアクセス 26 サンプルが失われないスレッショルドを探索した。その結果、T_{MA} として 4 を採用することで 26 サンプル全てを残してログファイルを小さくできることがわかった。以降の実験では $T_{MA} = 4$ を用いることとする。

$T_{MA} = 4$ を用いたところ、例として以下のようなファイルアクセスが除去された事を確認した。

(1) アプリケーションの設定ファイルの読み書き
(2) メーラーのファイル読み書き
(3) ディレクトリにある全ファイルに対する同時アクセス
(4) アーカイブの展開
(5) ソースコードのコンパイル

5.4 FI 法の最適パラメータの探索

5.4.1 *TransactionTime*

$MinSupport = 3, T_{Comb} = 0.2$ に固定し、最適な $TransactionTime$ を探索した（図 3）。図中の精度・リコール・F 値は全ての正解セットの平均である。$TransactionTime = 900[sec], 1800[sec], 3600[sec], 7200[sec], 14400[sec]$ を調べた結果、$TransactionTime = 7200$ が最良という結果が得られた。900 から 7200 までは精度が下がりつつもリコールが上がっている。これは TransactionTime を大きくすることで離れた時間に触ったファイル同士が同じトランザクションに入るようになり、関係しているが近い時間には触らなかったファイルを発見できるようになったためと考えられる。しかし、14400 では精度もリコールも下がっている。これは TransactionTime を長くしすぎたため、長い間使ったファイルし

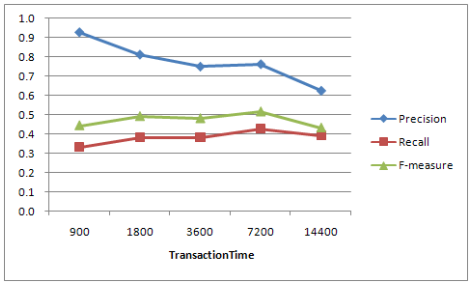

図 3 $TransactionTime$ の比較

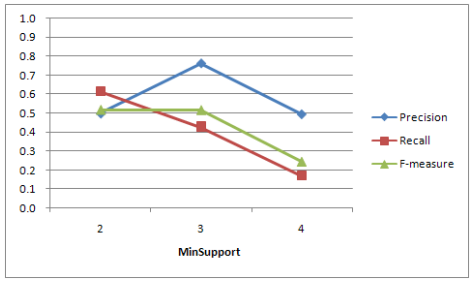

図 4 $MinSupport$ の比較

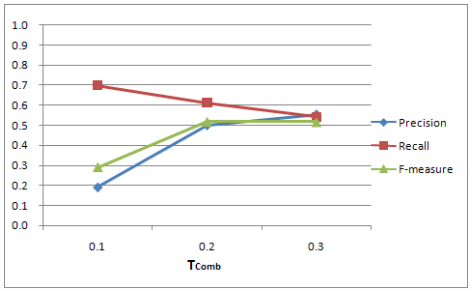

図 5 T_{Comb} の比較

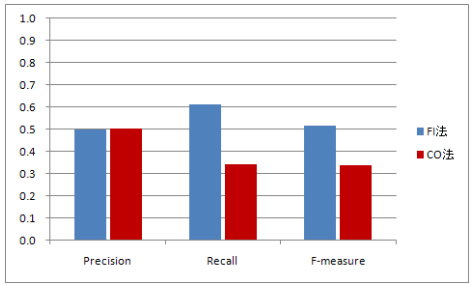

図 6 CO 法と FI 法の比較

かトランザクションに 3 回以上出現しなくなったためと考えられる。

5.4.2 $MinSupport$

前の実験で得られた $TransactionTime = 7200$ を使用し $T_{Comb} = 0.2$ に固定し、最適な $MinSupport$ を探索した（図 4）。$MinSupport = 2, 3, 4$ を比較したところ、$MinSupport = 2$ が最良という結果が得られた。MinSupport を下げることでリコールが向上し、MinSupport を上げることで精度が向上する。$MinSupport = 4$ で精度が下がっているのは不正解が増えたためではなく、条件が厳しすぎて正解セットを含む結果が得られなくなってしまったためである。

5.4.3 T_{Comb}

前の実験で得られた $TransactionTime = 7200, MinSupport = 2$ を使用し、最適な T_{Comb} を探索した（図 5）。$T_{Comb} = 0.1, 0.2, 0.3$ を比較したところ、$T_{Comb} = 0.2$ が最良

表 2 FI 法最適パラメータの結果
Table 2 Experiment Result of FI-method

正解セット	正解数	不正解数	Precision	Recall	F 値
S_1	6	30	0.167	0.667	0.267
S_2	26	107	0.195	0.419	0.267
S_3	42	15	0.737	0.500	0.596
S_4	25	17	0.595	0.781	0.676
S_5	24	6	0.800	0.727	0.762
S_6	20	20	0.500	0.588	0.541
				平均 F 値	0.518

という結果が得られた。スレッショルド T_{Comb} を上げることで重複の大きい集合のみ結合するようになり精度が向上し、下げることで少しでも共通要素のある集合同士が結合しリコールが上がる。

5.4.4 パラメータに関する考察

今回の実験では最適パラメータは $TransactionTime = 7200, MinSupport = 2, T_{comb} = 0.2$ となり、当初考えていたより TransactionTime は長時間、MinSupport は極端に低くなった。2 時間以内に全ての要素にアクセスすることが 2 回以上あるファイル集合が正解とされる。これは非常に緩い条件だと考えられるが、これは時間のかかる作業を正解セットとして用いたためだと考えられる。数十分で終わるような作業を正解セットとして用いていれば、これらの値は異なったと考えられる。

5.5 提案手法と CO 法との比較

5.5.1 CO 法の最適パラメータの探索

CO 法についても FI 法と同じく最適なパラメータを探索した。スコアの計算とクラスタリング方法の選択については、前回の報告[2] で最適と分かった重複時間と単連結法を用いた。FI 法と同様に平均 F 値が最大になる階層的クラスタリングのスレッショルドを探索した。その結果スレッショルド＝ 0.6 が最適であった。以降の比較にはこの設定を用いる。

5.5.2 比　　較

FI 法の最適パラメータを用いた結果を表 2、CO 法の最適パラメータを用いた結果を表 3 に示す。

正解セット S_1 に着目すると、FI 法では発見できているのに対し CO 法では全く発見できていない。CO 法でスレッショルドを緩くしリコールを極限にしても S_1 は発見できなかった。これは、S_1 のファイル群は厳密に使用時間が重複していないためだと考えられる。FI

表 3 CO 法最適パラメータの結果
Table 3 Experiment Result of CO-method

正解セット	正解数	不正解数	Precision	Recall	F 値
S_1	0	0	0.000	0.000	0.000
S_2	27	34	0.443	0.435	0.439
S_3	46	22	0.676	0.548	0.605
S_4	5	3	0.625	0.156	0.250
S_5	24	64	0.273	0.727	0.397
S_6	7	0	1.000	0.206	0.341
				平均 F 値	0.339

法ではこのような集合も発見でき、FI 法のほうが発見できる物が多様であるとわかった。正解セット S_4, S_6 でも FI 法と CO 法で発見ファイル数に大きな差がある。これらも厳密には使用時間が重複していないため CO 法では発見できないのだと考えられる。

正解セット S_2 に着目すると、FI 法の精度が低くなっているのが確認できる。これは、TransactionTime に 2 時間という長い期間が選択されたため、正解セットと近い時間に使われた無関係なファイル群が正解セットとトランザクションに入ってしまったためだと考えられる。実際に正解セット S_2 は 2，3ヶ月ほど作業を行っており、他の作業と同時に行われることもあった。このような正解セットでは厳密な使用時間の重複を見る CO 法の方が不正解ファイルを含みにくいと判断できる。

平均精度・平均リコール・平均 F 値の比較図を図 6 に示す。F 値に着目すると今回の正解セットでは FI 法のほうがよい結果を出している。FI 法で厳密に使用時間が重複しないものを発見可能になったために狙い通りリコールが向上したためだと考えられる。

6. 関 連 研 究

ファイルを検索するシステムとしてはデスクトップ検索[6][7] が有名である。しかし、デスクトップ検索システムはファイルに含まれるテキストにするキーワード検索を用いるため、テキストが含まれないファイルは探すことができないという問題がある。また、同じ作業に属するファイルであってもソースコード・レポート・実験データ等で含まれるテキストは異なるため共通のキーワードで全種類のファイルを探し出すことは難しい。

デスクトップ検索システムの改良として提案された手法として渡部らの FRIDAL[8] がある。この手法ではキーワードでテキストを含むファイルを発見した後、テキスト検索で発見したファイル群に更にそれらと関連する別のファイル群も検索結果に含めて結果出力する。この関連ファイルの発見は、ファイルを同時に使用したという情報を使っている。これによりキーワードを含まないファイルであっても、キーワードで間接的に検索可能になる。本研究の CO 法はこのシステムの同時に使用したファイル群を発見する仕組みを利用している。

Soules らによる Connections[9] も、FRIDAL と同様のデスクトップ検索の拡張である。FRIDAL と同様にテキスト検索の結果ファイル群と関連するファイル群も出力に含めることでキーワードを含まないファイルを検索可能にする。しかし、ファイル同士の関連を発見する方法は FRIDAL とは異なり、使用時間の厳密な重複ではなく「それらのファイルが N 秒以内にアクセスされたか」といった尺度を使用している。これは FI 法のトランザクションの考え方に近い。ただし、ある 2 ファイルが N 秒以内にアクセスされたか見ているだけで、FI 法のように 3 ファイル以上の組み合わせを発見しているわけではない。

また、井ノ口らの研究[10] では、ファイルアクセスの順序やアクセスの密度に着目した尺度を提案している。FRIDAL の尺度が仮想ディレクトリのために提案された訳ではないが CO 法の元になったように、これらの方式も仮想ディレクトリ生成に応用可能であると考えられる。

大澤らによる俺デスク[11] では、ブラウザや Word の動作を記録し、その操作履歴をタイムライン表示することでユーザが「Word で企画書を作成した時に参照していた Web ページ」といった物を検索できるようにしている。イベントの共起をタイムラインで表示するが、本研究のように自動で同時に使ったファイルを集約する機能がない点が異なる。

ファイル検索ではなくファイル整理という観点に立った研究として、Gifford らによる Semantic File Systems[12] がある。このシステムは各ファイルから属性を抽出し、その属性を用いて管理をするファイルシステムを提案している。しかし、属性を抽出するためにファイルタイプごとに属性抽出プログラムを書く必要があり、ファイル形式に依存しない本研究と異なる。

暦本は Time Machine Computing[13] を提案している。ファイルは全てデスクトップに置かれ時間とともに消えていくシステムを提案している。ファイルを探したい場合は、過去のデスクトップに遡ってそのときデスクトップに置かれていたファイル群と一緒に発見できる。時間の概念を利用している点では本研究と近いが、ユーザが明示的にファイルをデスクトップに置く必要がある点と、本研究のディレクトリのような明確な分類を行わない点が異なっている。

また、アクセスログのマイニングとしては、Web アクセスログを利用したリコメンデーション[14] などが関連する。これらの成果を利用しファイルのリコメンデーションができれ

ば有用である。しかし、Web アクセスログでは性質も異なり、そのまま応用するのは難しいと考えられる。

7. まとめと今後の課題

これまでの研究で提案した CO 法の問題を解決するために、新たに FI 法という手法を提案した。CO 法がファイル同士の使用時間の厳密な重複でファイル間の関係を発見するのに対し、FI 法は同一期間内にアクセスがあるかで関係を発見することで発見できるファイルの範囲を広げた。最適パラメータを探索したところ当初の予想とは異なるパラメータが選択された。しかし、そのパラメータにおいて FI 法は狙い通り CO 法よりも高いリコールを出すことを確認した。その結果 F 値が向上することを実験を通して確認した。

また、同じ作業に属するファイル群を発見するにはファイルの使用期間が厳密に重複しない物にも着目すべきという知見や、関連するファイル同士でも同一期間内に使われる回数は想像より少ないという考察が得られた。

今後の課題として、短い TransactionTime と高い MinSupport を用いた場合の FI 法の評価、FI 法の結合処理の改良、CO 法と FI 法の手法の統合、両手法のパラメータの自動決定方法、より多様な正解セットによる評価、等が挙げられる。

参 考 文 献

1) 小田切健一，渡辺陽介，横田治夫：アクセス履歴に基づくファイル間関連度を用いたデスクトップ情報管理ツールの開発 (Poster Presentation)，信学技報，DE2008-72, Vol.108, No.329, p.49 (2008).
2) 小田切健一，渡辺陽介，横田治夫：アクセス履歴を用いたユーザの作業に対応する仮想ディレクトリの生成 (2009). 第 1 回データ工学と情報マネジメントに関するフォーラム（DEIM2009）.
3) 新納浩幸：R で学ぶクラスタ解析，オーム社 (2007).
4) *Samba*. http://us3.samba.org/samba/.
5) Agrawal, R., Imieliński, T. and Swami, A.: Mining association rules between sets of items in large databases, *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, New York, NY, USA, ACM, pp. 207–216 (1993).
6) Windows デスクトップサーチ. http://www.microsoft.com/japan/windows/desktopsearch/default.mspx.
7) Google デスクトップ. http://desktop.google.co.jp/.
8) 渡部徹太郎，小林隆志，横田治夫：キーワード非含有ファイルを検索可能とするファイル間関連度を用いた検索手法の評価 (2008). 第 19 回データ工学ワークショップ (DEWS2008).
9) Soules, C. A.N. and Ganger, G.R.: Connections: using context to enhance file search, *SIGOPS Oper. Syst. Rev.*, Vol.39, No.5, pp.119–132 (2005).
10) 井ノ口伸人，吉川正俊：アクセス履歴を考慮したファイル間の関連度を用いたデスクトップ検索 (履歴応用, 夏のデータベースワークショップ DBWS 2006)，情報処理学会研究報告. データベース・システム研究会報告， Vol.2006, No.77, pp.141–146 (20060712).
11) 大澤　亮，高汐一紀，徳田英幸：俺デスク:ユーザ操作履歴に基づく情報想起支援ツール (2006). 第 47 回プログラミング・シンポジウム報告集，pp.15-21.
12) Gifford, D.K.， Jouvelot, P.， Sheldon, M.A.， O'Toole, J.W.：*Semantic File Systems* (1991). 13th ACM Symposium on Operating Systems Principles.
13) Rekimoto, J.: Time-machine computing: a time-centric approach for the information environment, *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, New York, NY, USA, ACM, pp.45–54 (1999).
14) Mobasher, B., Dai, H., Luo, T. and Nakagawa, M.: Effective perssonalization based on association rule discovery from web usage data, *Proc. 3rd Intl. Workshop on Web information and data management* (2001).