大規模ディレクトリ空間視覚化のための論理構造抽出

望月　祥司† 　児玉麻莉子†† 　森嶋　厚行††† 　石川　憲一† 　田島　敬史††††

† 筑波大学大学院 図書館情報メディア研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学 図書館情報専門学群　〒 305–8550 茨城県つくば市春日 1–2
††† 筑波大学大学院 図書館情報メディア研究科/知的コミュニティ基盤研究センター
〒 305–8550 茨城県つくば市春日 1–2
†††† 京都大学大学院 情報学研究科　〒 606-8501 京都市左京区吉田本町
E-mail: †{mshoji,mori,ishiken}@slis.tsukuba.ac.jp, ††s0312151@ipe.tsukuba.ac.jp, †††tajima@i.kyoto-u.ac.jp

あらまし　現在，計算機のファイルシステムに格納されているファイル数が増大し，人手での管理が困難になっている．これに対するアプローチとして，デスクトップサーチなどの研究が行われているが，本論文ではこれとは異なるアプローチとして，大規模ディレクトリ構造の視覚化に取り組む．ディレクトリ構造をディスプレイ上に配置する手法は数多く提案されているが，内容更新が行われると，一般にその配置構造が大幅に影響を受ける．そのためブラウジングする際に，以前の記憶を頼りにすることが困難となる．本論文では，ディレクトリ構造をそのまま視覚化するのではなく，ディレクトリ構造に内在する安定した論理構造を抽出する手法を提案し，それをディレクトリ構造の視覚化に応用する．

キーワード　情報視覚化，ディレクトリ構造，情報抽出

Extraction of Logical Structures for the Visualization of Large Directory Spaces

Shoji MOCHIZUKI†, Mariko KODAMA††, Atsuyuki MORISHIMA†††, Kenichi ISHIKAWA†, and Keishi TAJIMA††††

† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba,　1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
†† Sch. of Library and Information Science, Univ. of Tsukuba,　1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
††† Grad. Sch. of Library, Information and Media Studies/ Research Center for Knowledge Communities, Univ. of Tsukuba.　1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
†††† Grad. Sch. of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606–8501, Japan
E-mail: †{mshoji,mori,ishiken}@slis.tsukuba.ac.jp, ††s0312151@ipe.tsukuba.ac.jp, †††tajima@i.kyoto-u.ac.jp

Abstract　The current rapid increase of the files stored in computer file systems makes it difficult for us to manage the files in a manual fashion. To approach the problem, researches on effective solutions, including desktop search technologies, have been conducted. This paper approaches the problem by developing a method to visualize large directory structures. So far, many methods for visualizing directory structures have been proposed, but their file arrangements on the screen suffer sharp fluctuations if the underlying directory structure is changed. Therefore, it is difficult for us to rely on the past experiences in their browsing. This paper proposes a method that extracts stable structures from the raw directory structures and applies the method to the construction of stable visualization of directory structures.

Key words　Information Visualization, Directory Structure, Information Extraction

1. はじめに

現在，計算機のファイルシステムに格納されているファイル数が増大し，人手での管理が困難になっている．これに対するアプローチとして，デスクトップサーチなどの研究が行われているが，本論文ではこれとは異なるアプローチとして，大規模ディレクトリ構造の視覚化に取り組む．ディレクトリ構造をディスプレイ上に配置する手法は数多く提案されているが，ディレクトリ構造をそのまま視覚化すると，ディレクトリの移動などの内容更新が行われた場合に，一般にその配置構造が大幅に影響を受ける．そのためブラウジングする際に，以前の記憶を頼りにすることが困難となる．本論文では，ディレクトリ構造をそのまま視覚化するのではなく，ディレクトリ構造に内在する安定した論理構造を抽出する手法を提案し，それをディレクトリ構造の視覚化に応用する．

本研究は，我々のコミュニティ情報空間ガバナンスプロジェクトの一環で行われるものである．このプロジェクトについては本研究の動機を説明するために後述するが，本論文の内容は一般性があり，このプロジェクトに特化したものではない．

本論文の構成は次の通りである．まず，2 章で我々が取り組んでいるコミュニティ情報空間ガバナンスプロジェクトについて説明する．3 章で，大規模ディレクトリ構造視覚化のための論理構造抽出手法について提案する．4 章で予備実験の結果を示す．5 章で関連研究を説明する．6 章はまとめと今後の課題である．

2. コミュニティ情報空間ガバナンスプロジェクト

現在，我々が進めているコミュニティ情報空間ガバナンス (Governance of Community Information Spaces) プロジェクトでは，各クライアント PC やファイルサーバに格納されている多量のファイル群の管理を行うシステム (InfoSpace Governor) を開発している．

本システムの第一の特徴は，個々のファイルシステムではなく，**コミュニティ情報空間** (Community Information Spaces, CIS) と呼ぶ複数のファイルシステム群を含む空間を管理の範囲としていることである．図 1 はある大学の研究室における CIS の例である．一般に，CIS には複数の計算機が含まれており，それぞれが情報を分散して管理している．例えば，この研究室で行われているプロジェクトに関連する情報は，ファイルサーバと参加メンバの PC に分散されて管理されている．それらに管理されている情報は，お互い関連しているものの，それらはシステムレベルでは明示的には関連づけられていない．例えば，そのプロジェクトに関連するあるファイルの最新バージョンはどれか，といった情報や，そのファイルが参照するファイルはどれか，といった情報をシステムはもっていない．したがって，これらの問にシステムが答えることは出来ない．ネットワークを通じてこれらの情報源が物理的には接続されているにもかかわらず，先に述べたような関連のレベルでは実は接続されていないのである．

本システムの第二の特徴は，これらの関連を明示的に表すメタデータを保持することにより，情報空間の統治を行う事である．具体的には，今説明したような情報資源間の関連を明示的に保持するための InfoSpace Server と呼ぶメタデータデータベースを利用し，コミュニティ情報空間の管理を行う (図 2)．また，InfoSpace Maps と呼ぶ視覚化ツールを用意し，管理している情報空間の情報を表示する．直観的には，これは管理下にあるファイルシステムが持つディレクトリ構造と，先に説明したようなファイル間の関連を視覚化したものである (図 3)．利用者は，InfoSpace Maps のブラウジング機能を用いて，コミュニティ情報空間をブラウジングすることが出来る．InfoSpace Maps は，Google Maps [1] 等の地図ブラウザのアナロジで設計されたもので，ファイルやディレクトリを表すノードが配置された空間をマウスで移動する．そこでは，上下左右のスクロールやマウスホイールによるズーミング，ズームレベルに応じたラベルの表示などが実現されている．

図 1　インフォメーションスペースの例
Fig. 1　Example of Community Information Space

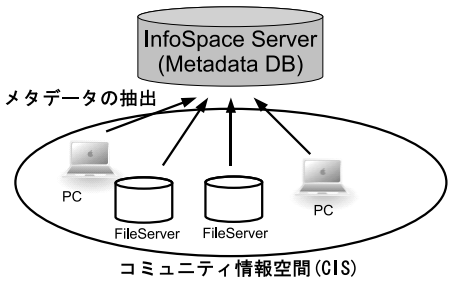

図 2　InfoSpace Governor のアーキテクチャ
Fig. 2　Architecture of InfoSpace Governor

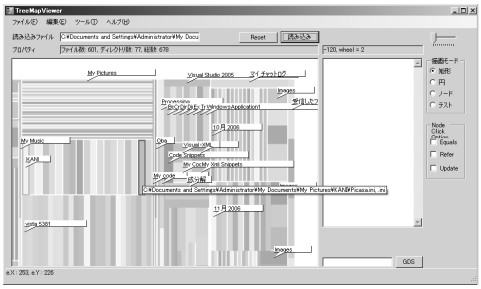

図 3　InfoSpace Maps のプロトタイプのスナップショット
Fig. 3　A screen snapshot of InfoSpace Maps

2.1 問　　題

コミュニティ情報空間に含まれるディレクトリ構造を視覚化

する際，最も単純な方法は，ディレクトリ構造にしたがってファイル群の配置を決定することである．しかし，このアプローチでは，ディレクトリのコピーや新たなディレクトリの追加の際に大幅な再配置が必要になる事がある．これは，ブラウジングのための視覚化ツールという観点からは望ましくない．なぜなら，以前にブラウジングを行った経験を生かすことが出来ず，記憶を手がかりに出来ないからである．そこで，より安定した構造をベースにファイル群を配置することが望まれる．そうすれば，以前利用したファイルの発見や，内容の変化に気付きやすくなると考えられる．

3. 提 案 手 法

3.1 概　　要

我々は，ディレクトリ空間を表現する安定した構造を発見するため，ディレクトリ構造とその構造に含まれている論理構造の関係に着目した．一般に，ディレクトリ構造 (図 5) はそこに含まれるファイル間の論理構造 (図 6) をある程度反映している．例えば，Projects の下に WISH と SMART があることから，WISH と SMART というプロジェクトが存在することが分かる．

現実のディレクトリ構造の特徴は，これらの論理構造を素直に反映していないことである．その理由は様々なものが考えられる．まず，ディスク容量を節約したり，コピーファイルの管理の手間をかけないようにするため，同じファイルのコピーを出来るだけ持たないようにしている，ということが考えられる．例えば，図 6 の論理構造における Projects, Member, Conf, Year などは論理的には任意の組合せが可能であるため，全ての組合せでディレクトリ構造を作ると図 4 のようになってしまうが，そのためには同じファイルのコピーを多数作る必要がある．したがって，現実には図 5 のように全ての組合せを持つようなディレクトリ構造は作成されない．他にも，よく使うファイルはルートからの距離ができるだけ浅いディレクトリに保存したいため，本来あるべき場所から外れたところにディレクトリやファイルを作成することや，単にメンテナンスが不行届きであるという理由等によって，ディレクトリ構造は変更されやすいアドホックな構造になる傾向にある．

本論文で提案する手法のアイデアは，図 5 のようなディレクトリ構造から図 6 のような論理構造を抽出することである．抽出された論理構造は，実際のディレクトリ構造に比べて安定していると考えられるため，ディレクトリ構造そのままをベースにファイルの配置を決めるのではなく，抽出された論理構造を主な手がかりとして，ディレクトリやファイルを表すノードの配置を決定する．それにより，安定度の高い視覚化が実現出来ると考えられる．

望ましい性質のひとつは，抽出された論理構造が，ディレクトリ構造に含まれるファイル群を全てカバーすることである．しかし，本提案手法ではその性質は保証せず，通常のディレクトリ構造によるアクセス手段も同時に提供することによって，全ファイルへのアクセス手段を提供することにする．したがって，本論文で提案した手法で抽出された論理構造による地図は，ファイル群に対する一種の (必ずしも全てをカバーしていない) インデックスを提供するものであると言える．

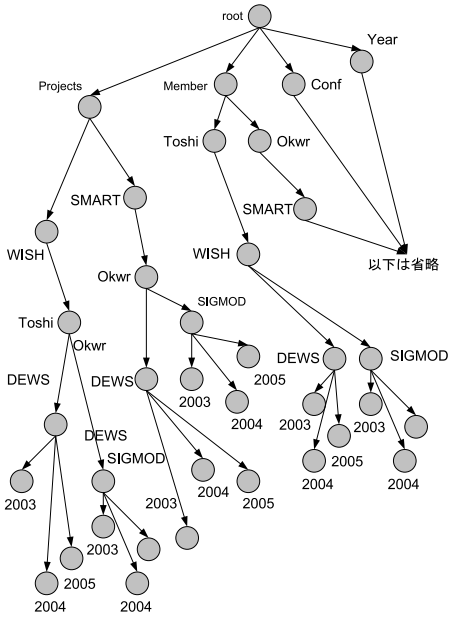

図 4　任意の組み合わせによるディレクトリ構造
Fig. 4　Directory Structure by every combination

3.2 提案アルゴリズム

ディレクトリ構造に内在する安定した論理構造を抽出する手法について説明する．ここでは例として，図 5 に示すディレクトリ構造からの論理構造抽出をとりあげる．本アルゴリズムで扱う，ディレクトリ構造 D の**論理構造** $L(D)$ は下記で定義される．

$$L(D) = \{(p, C) \mid p \in dirs(D) \wedge \forall c \in C(c \in dirs(D) \wedge isSuperordinateTo(p, c))\}$$

ここで，$dirs(D)$ はディレクトリ構造 D に含まれるディレクトリ名の集合，$isSuperordinateTo(p, c)$ は，p が c の上位概念を表すディレクトリ名であることを示す．ここでいう上位概念とは，isa 関係および partof 関係のことである．一つの (p, C) を**論理構造ユニット**とよび，p を $c \in C$ の**親**と呼ぶ (C を子集合，c を**子**と呼ぶ)．例えば，図 6 は 4 つの論理構造ユニットを表している．一般には，ディレクトリ構造に内在する論理構造では，必ずしも親子関係が木にならないものも存在すると考えられるが，そのような場合への対応の検討は今後の課題である．

本アルゴリズムは下記の 3 ステップから構成される．

ステップ 1: $L(D)$ に含まれる子集合 C となる可能性がある候補 C'_j を見つけ，それらの集合 C' を作成する．

ステップ 2: 全ての $C'_j \in C'$ に対して，C'_j の親候補の集合 $ParentCandidates_j$ を作成する．

ステップ 3: 各 $ParentCandidates_j$ に含まれる親候補から不適切な候補を削除する．最も適切な親が最終的に 1 つに選ばれることが理想であるが，親候補が複数残った場合にはその中からランダムに 1 つを選ぶ．親候補は全く残らない場合もある．その場合は，無名の親が存在すると仮定する．

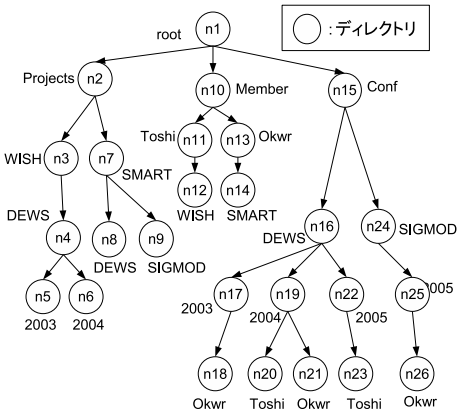

図 5　ディレクトリ構造

Fig. 5　Directory Structure

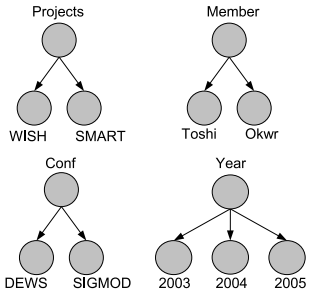

図 6　論 理 構 造

Fig. 6　Logical Structure

次に，これら 3 ステップの詳細を説明する．以下では，D に含まれるディレクトリノードの集合を N で表す．また，ディレクトリの各兄弟のノードの集合を N_i で表す (図 5 の場合を図 7 に示す)．すなわち，N は N_i の直和集合 $N = N_1 + N_2 + \ldots$ である．

N_1	$\{n_2, n_{10}, n_{15}\}$
N_2	$\{n_3, n_7\}$
N_3	$\{n_4\}$
N_4	$\{n_5, n_6\}$
N_5	$\{n_8, n_9\}$
N_6	$\{n_{11}, n_{13}\}$
N_7	$\{n_{12}\}$
N_8	$\{n_{14}\}$
N_9	$\{n_{16}, n_{24}\}$
N_{10}	$\{n_{17}, n_{19}, n_{22}\}$
N_{11}	$\{n_{18}\}$
N_{12}	$\{n_{20}, n_{21}\}$
N_{13}	$\{n_{23}\}$
N_{14}	$\{n_{25}\}$
N_{15}	$\{n_{26}\}$

図 7　各兄弟ノードの集合一覧

Fig. 7　List of sibling sets

(1) ステップ 1 の詳細

本アルゴリズムのステップ 1 では，子集合の候補 C'_j として，ディレクトリの木構造における兄弟ノード N_i のうちお互いにディレクトリ名がオーバラップするものを集めた閉包を，和集合演算によって計算する．そして，発見した子集合候補 C'_j の全てを集合 C' に格納する．例として，図 7 から求められる全ての $C'_j \in C'$ を図 8 に示す．

C'_1	$\{n_2, n_{10}, n_{15}\}$
C'_2	$\{n_3, n_7, n_{12}, n_{14}\}$
C'_3	$\{n_4, n_8, n_9, n_{16}, n_{24}\}$
C'_4	$\{n_5, n_6, n_{17}, n_{19}, n_{22}, n_{25}\}$
C'_5	$\{n_{11}, n_{13}, n_{18}, n_{20}, n_{21}, n_{23}, n_{26}\}$

図 8　子集合候補の集合

Fig. 8　Set of ChildSet Candidates

入力：ノード集合 D(ただし，ノード集合 $N = N_1 + N_2 + \ldots$)
出力：論理構造 $L(D)$

```
1   // ステップ 1: C' を求める
2   C' ← ∪_i{N_i}
3   for each C'_i ∈ C' {
4       if (∃C'_j ∈ C'(C'_i ∩_dname C'_j ≠ φ)) {
5           C'_j ← C'_j ∪ C'_i;
6           C' ← C' − C'_i;
7       }
8   }
9
10  // ステップ 2: 親候補の初期値を求める
11  for each C'_j ∈ C' {
12      ParentCandidats_j ← φ;
13      if (∃N_k ∈ N(N_k ∩_dname C'_j ≠ φ)) {
14          ParentCandidates_j ←
15              ParentCandidates_j ∪ {parent(N_k)};
16      }
17  }
18
19  // ステップ 3:
20  // 親がコンフリクトするものを削除する
21  for each ParentCandidates_j {
22      if (∃N_k ∈ N(|N_k ∩_dname ParentCandidates_j| ≧ 2)) {
23          ParentCandidates_j ←
24              ParentCandidates_j − N_k ∩ ParentCandidates_j;
25      }
26  }
27
28  // 極大の部分集合以外のものを削除する
29  for each C'_j ∈ C' {
30      for each N_k ∈ N(N_k.isNotMaximal(C'_j)) {
31          ParentCadidates_j ←
32              ParentCandidates_j − {parent(N_k)};
33      }
34  }
35
36  // 子供集合の子供であるものを削除する
37  for each ParentCandidates_j {
38      for each p ∈ ParentCandidates_j {
39          for each c ∈ C'_j {
40              if ({child(c)} ∩_dname {p} ≠ φ) {
41                  ParentCandidates_j ←
42                      ParentCandidates_j − {p};
43          }
44      }
45  }
46
47  // ルートに一番近いものを選択する
48  for each ParentCandidates_j {
49      ParentCandidates2_j ← {p | p ∈ ParentCandidates_j;
50          (p has the shortest path from the root.) }
51      parent_j is an elemnt taken from ParentCandidates2_j;
52      output(parent_j, C'_j);
53  }
```

図 9　論理構造抽出アルゴリズム

Fig. 9　Algorithm for Extracting Logical Structure

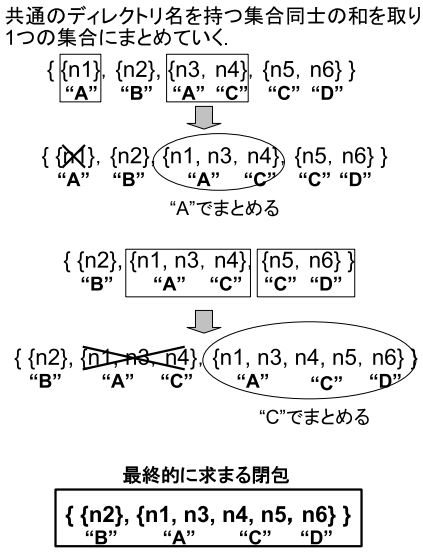

図 10　子集合候補の計算

Fig. 10　How to compute child sets

ステップ 1 は，図 9 の擬似コードにおける 1-8 行目に対応する．擬似コード中における演算 $M \cap_{dname} N$ は，M 中のノードが持つディレクトリ名の集合と N 中のノードが持つディレクトリ名の集合の積集合を求める演算である．まず，C' の初期値として兄弟ノードの集合 N_i を全て含むような集合を代入する (2 行目)．次に，C' の要素である集合 C'_i について，ディレクトリ名がオーバラップする C'_j が存在する場合にはそれらをまとめていく (3-8 行目)．図 10 は C' の初期値が $\{\{n1\},\{n2\},\{n3,n4\},\{n5,n6\}\}$ (これらのディレクトリ名の集合は順に"A", "B", "A", "C", "C", "D") であるときの C' の計算過程である．

(2) ステップ 2 の詳細

ステップ 2 では，ステップ 1 で求めた全ての子集合候補 $C'_j \in C'$ に対して，親となりうる候補 $ParentCandidates_j$ を求める．擬似コードでは 10-17 行目になる．14 行目にある $parent(N_k)$ は，兄弟集合 N_k の親であるノードを返す関数である．

ある C'_j の親候補集合 $ParentCandidates_j$ は，その C'_j に含まれるノードを持つ兄弟集合 N_i の全ての親の集合である．例を図 11 に示す．子集合候補 $C'_3 = \{n_4, n_8, n_9, n_{16}, n_{24}\}$ (これらのディレクトリ名の集合は { "DEWS", "SIGMOD" }) とオーバーラップする兄弟集合は，$N_3 = \{n_4\}$，$N_5 = \{n_8, n_9\}$，$N_9 = \{n_{16}, n_{24}\}$ である．兄弟集合 N_3，N_5，N_9 の各親ノードは n_3，n_7，n_{15} である．したがって，$ParentCandidates_3 = \{n_3, n_7, n_{15}\}$ となる．

(3) ステップ 3 の詳細

ステップ 3 では，ステップ 2 で求めた親候補の中から，適切でないと考えられるものを除去し，1 個ないしは (適切なものが存在しない場合は)0 個の親を選択する．　具体的には下記の場合に，該当する親候補を除去する．

- 複数の親候補が，ディレクトリ構造中でお互いに兄弟であるような場合．例えば，図 11 において，親候補 $ParentCandidate_3 = n_3, n_7, n_{15}$ の内，n_3 と n_7 は共通の親を持つ兄弟同士である．よって，親候補 $ParentCandidate_3$ から n_3 と n_7 を削除する．
- その親候補がもつ子集合のディレクトリ名集合を完全に含むような子集合を別の親候補が持つ場合．
- その親候補のディレクトリ名と同じ名前のディレクトリを，C'_j 中のいずれかのノードの子ノードが持つ場合．例えば，{ "DEWS", "SIGMOD" } の親候補に"WISH"がある場合に，"DEWS"もしくは"SIGMOD"が別の"WISH"ディレクトリを子ノードとして持つ時である．この場合，"WISH"はこれらと関連はあっても上位下位の関係ではないと考えられるため，親候補から削除する．

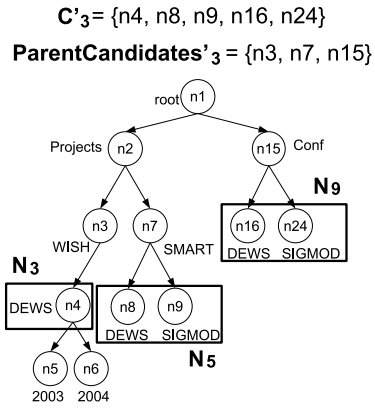

図 11　$ParentCandidates_j$ の計算例

Fig. 11　Example of Computation of $ParentCandiddates_j$

これらを削除しても親候補が 1 つに絞れない場合，残った親候補の中で，ディレクトリのルートノードからの距離が一番短いものを選ぶ．さらに 1 つに決まらない場合は，本アルゴリズムではランダムに選択を行う．

疑似コードの 19-26 行目で，親候補の中からお互い兄弟同士であるものを削除している．28-34 行目では，$ParentCandidates_j$ に含まれる全ての親候補の子集合について，それらの C' がもつディレクトリ集合間の包含関係を調べ，極大であるような集合を持つ親候補以外は削除する．30 行目にある $N_k.isNotMaximal(C'_j)$ は，兄弟集合 N_k が子集合候補 C'_j に含まれる兄弟集合のうち，極大であるような集合でないときに true を返す関数である．36-45 行目では，親候補のうち，C'_j 中の要素の子が同じディレクトリ名をもつものを削除する．

49-50 行目では，ここまでで残った親候補の中で，ルートからの距離が一番短いものを親にする．それでも候補が 1 つに絞れない場合は，残ったものの中から任意に一つを選ぶ (51 行目)．

4. 予 備 実 験

本章では，3.2 節の論理構造抽出アルゴリズムを用いて 2 つの予備実験を行う．実験対象のデータとしては，筑波大学森嶋研究室のファイルサーバに存在する実ディレクトリ構造と，同研究室のあるノート PC X に存在する実ディレクトリ構造を利用する．ファイルサーバとノート PC X に含まれる全ディレク

ディレクトリ総数 $\|N\|$	1224
兄弟集合 N_i の総数	145
出力した論理構造ユニットの総数 $\|L\|$	68

図 12　データと出力のサイズ (森嶋研究室ファイルサーバ)

Fig. 12　Sizes of the data and outputs (mlab file server)

ディレクトリ総数 $\|N\|$	772
兄弟集合 N_i の総数	117
出力した論理構造ユニットの総数 $\|L\|$	58

図 13　データと出力のサイズ (ノート PC X)

Fig. 13　Sizes of the data and outputs (note PC X)

トリ数は，それぞれ 27428 と 8410 であるが，後述するように評価に人手がかかるため，対象とするディレクトリをルートからの距離が 3 までのものに限定した．その結果，今回の実験で扱うディレクトリ数はそれぞれ 1224，772 となった．

まず，実験 1 では，提案アルゴリズムによってディレクトリ構造から抽出した論理構造ユニット (p, C) の適切さについて評価を行う．

次に，実験 2 では，実験 1 で抽出した論理構造ユニットのうち正しいものだけを選んで木を構築し，それを，木の視覚化手法の一つである Tree Map [2] で視覚化する．次に，ディレクトリの操作を行ったときのノード配置の安定度への影響を，ディレクトリ構造をそのまま視覚化した場合と比較しながら検証する．具体的には，これら 2 つの場合についてディレクトリ・ファイルの同じ操作を行い，操作前後での表示画面を比較し，変化の割合を面積のパーセンテージで表す．この値が小さいほど，ディレクトリ・ファイル操作に対する画面の安定度が高いと判断する．

4.1　実験 1: 抽出した論理構造ユニットの評価

本実験の目的は，本論文のアルゴリズムを用いて抽出した論理構造ユニットの適切さを評価することである．具体的には，抽出した各 $(p, C) \in L(D)$ を，下記の 4 種類に分類する．

(A)　p, C 共に適切であり，p が各 $c \in C$ の上位概念になっている．

(B)　C は適切な要素の集合を含んでいるが，p がそれらの上位概念になっていない．

(C)　p に対して下位概念となりうるようなものが C に含まれているが，全てではない．

(D)　p が，いずれの $c \in C$ の上位概念にもなっていない．

さらに，分類 A については，その関係が isa 関係か partof 関係かによって分類する．

結果. 対象としたディレクトリの総数と，ディレクトリの兄弟集合の総数，本アルゴリズムが出力した論理構造ユニットの総数を図 12，図 13 に示す．また，出力された論理構造ユニットの分類結果を図 14，図 15 に示す．

評価の結果，抽出した論理構造ユニットのうち，森嶋研究室のファイルサーバでは約 25% が，ノート PC X では約 43% が適切な論理構造ユニットであることが分かった．それ以外のユニットは適切とは考えられなかったが，これらが結果として出

	分類					
	A					
	isa	$partof$	B	C	D	合計
数	10	7	10	8	33	68
割合	14.7%	10.2%	14.7%	11.7%	48.5%	100%

図 14　出力の分類結果 (森嶋研究室ファイルサーバ)

Fig. 14　Classification of Outputs (mlab file server)

	分類					
	A					
	isa	$partof$	B	C	D	合計
数	12	13	10	8	15	58
割合	20.6%	22.4%	17.2%	13.7%	25.8%	100%

図 15　出力の分類結果 (ノート PC X)

Fig. 15　Classification of Outputs (note PC X)

力された原因としては次のものがあげられる．

- p が C の上位概念になっていないこと (項目 B) について:

現在の手法では，親の候補を子集合 C'_j とオーバーラップする兄弟集合 N_k の親としているが，それ以外の親ノードの方が望ましい場合も考えられる．親候補の対象として，祖先ノードなどを候補として考えてみるアプローチも試してみる価値があると考える．

- 適切な C が抽出できていないこと (項目 C) について:

現在の子集合はオーバーラップする兄弟集合の閉包として計算するが，単純に閉包を求めるだけでは，子集合の中に望ましくないものが含まれてしまう可能性がある．子集合をより適切な集合とするためには，何らかの工夫により，子集合をフィルタリングする必要があると考える．

実際に項目 B に分類したユニットには，p = ”address”, C = {”2002”,”2003”,”2004”,”2005”,”2006”,”2007”} や，p = ”ariyama”, C = {”最終”,”中間”} などがある．また，項目 C に分類したユニットには，p = ”$Photo\_Video$”, C = {”20051112 中間発表”,”iTunesLibraryUpdater”, . . .} などがある．

4.2　実験 2: ノード配置への影響に関する評価

この実験の目的は，本手法によって抽出した論理構造を利用したファイル群配置を，ディレクトリ構造そのままを反映させたファイル群配置と比較することにより，提案手法の安定度への影響を検討することである．具体的には，これら 2 つの手法で視覚化された画面を対象として，同一のディレクトリ・ファイル操作を行い，操作前後での画面を比較して変化の割合をパーセンテージで表す．この値が小さいほど，ディレクトリ・ファイル操作に対する画面の安定度が高いと判断する．

実験内容は次の通りである．まず，2 つの手法によるファイル配置を決定するための木をそれぞれ下記のように作成した．

(1) 抽出した論理構造を用いたファイル配置: 実験 1 で得られた論理構造ユニット $(p, C) \in L(D)$ のうち，項目 A に分類したユニットだけを用いて木を構築した．この木は，安定していると考えられるファイル配置を決定するために利用する．具体的には，あるユニットの子と別のユニットの親が一致する場合に

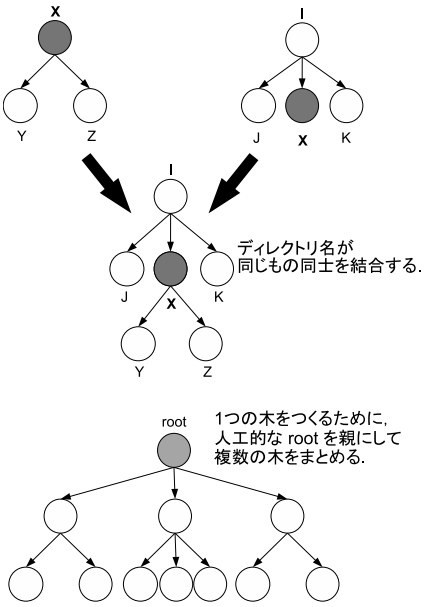

図 16　T_1 の 作 成

Fig. 16　How to construct T_1

はそれらを連結し，その結果複数の木が出来た場合には，人工的にルートノードを用意してそれらを一つの木にまとめた（図 16)．さらに，元のディレクトリ構造 D において，各ノードに対応するそれぞれのディレクトリに含まれていたディレクトリとファイルを，各ノードの下に連結した．この結果作成された木を T_1 と呼ぶ．ルートを含めたディレクトリ数は 4762, ファイル総数は 62066 となった．ここで，T_1 に含まれるファイルの集合を F と呼ぶ．F は元のディレクトリ構造に含まれていたファイルの集合のサブセットである．T_1 は実験 1 で分類 A と評価した論理構造ユニットだけで構築するため，他の論理構造ユニットが含んでいると考えられるファイルは F には含まれない．

(2) ディレクトリ構造をそのまま反映したファイル配置: 次に，元のディレクトリ構造 D のうち，F に含まれる全てのファイルを含むような最小の部分木を作成した．これを T_2 と呼ぶ．T_2 に含まれるディレクトリ総数は 4847 である．この木はディレクトリ構造をそのまま反映したファイル配置を決定するために利用される．したがって，特別な木の再構成は行わない．

以上の T_1 と T_2 において TreeMap アルゴリズムを用いてディスプレイ上にノードを配置した．TreeMap は，木構造を，領域の入れ子構造としてディスプレイに配置するアルゴリズムである．

以上 2 つの手法によるディスプレイ配置について，次の操作を行った後の画面変化を調べた．具体的には，ファイルサーバ中のディレクトリ/Shibaura/ を /Share/ に移動した．/Shibaura/ のディレクトリ数は 650，ファイル数は 5822 である．また，/Shibaura/ の下には 2 つの子供ディレクトリがある．ルート下にあるディレクトリを他のディレクトリに移動することは，研究室のファイルサーバで実行する典型的な操作例である

結果. この移動後のディレクトリ構造は移動前に比べて，T_2 に関してはディスプレイの面積にして 10.6% の変化があった．一方，T_1 の変化は全くなかった．その理由は下記の通りである．T_1 では，T_2 が持つ /Shibaura/ の子供ディレクトリに対応するディレクトリが，それぞれ直接ルートの下に置かれている．したがって，/Shibaura/ が移動を行っても，論理構造では /Shibaura/ の子供ディレクトリの移動は行われない．すなわち，このディレクトリの移動は，抽出された論理構造に矛盾しない移動であったということである．

以上のように，論理構造に影響を与えない範囲でのディレクトリの移動は，提案方式でのノード位置に大幅な影響を与えない．一方，この例とは異なり，抽出された論理構造と矛盾するような移動に関しては，提案手法でも大幅なノードの再配置が行われることがあることは明らかである．したがって，安定した論理構造を抽出できるかどうかが鍵になることがわかる．

5. 関 連 研 究

ディレクトリ構造を視覚化するための既存の手法として，Cone Trees [3] や Hyperbolic Tree [4] がある．Cone Trees はディレクトリ構造などの階層的なデータを 3 次元の木として視覚化する．各木は円錐として表示され，親ノードが円錐の頂点に，子ノードが円錐の底面の円周上に配置される．奥行きを上手く利用することによって，Cone Trees は効率よく情報を表示することができる．Hyperbolic Tree は Hyperbolic Space(非ユークリッド空間) 上にノードを配置する．これにより，近くにあるノードほど大きく，遠くにあるノードほど小さく表示することができる．これらの視覚化手法では，効率のよい情報の見せ方が可能であるが，視覚化の対象とするのはディレクトリ構造そのものである．そのため，ディレクトリの移動や削除といったディレクトリ構造の更新が行われると，視覚化されたビューもその影響を受けてしまう．したがって，もしルートに近い位置にあるディレクトリの移動を行った場合，ビューの構造が大きく変わってしまう．そして，更新の度に構造が大きく変化してしまうことは，人間にとって覚えづらく，使いにくいと考えられる．本論文の手法は，これらの視覚化手法と組み合わせて利用できる．

情報検索や Web の分野では，コンテンツそのものやリンク等のメタデータを用いてクラスタリングを行う手法が数多く研究されてきた．本研究は，既存のディレクトリ構造をメタデータと考え，ファイルのクラスタリングを行う手法の一つと考えることが出来る．

また，ファイルを階層的に分類するのではなく，タグを付けることによってファイルの整理を行うタギングの研究も近年盛んに行われている．ファイルの管理にタグを利用する試みも行われてきた [5]. タギングによって，階層的な分類では難しかった多面的な検索が容易になるが，ただタグを付けるだけでは，複数の分類軸を表現することはできない．例えば，同じタグでも”VLDB”と”SIGMOD”というタグ，”WISH”と”SMART”というタグは，それぞれ別の分類軸に属していると考えられる．そして，単純にタグを付けるだけでは，このような分類軸を表すことはできない．本論文で提案するアプローチは，そのよう

なタグ間の兄弟関係や親子関係等の抽出を試みていると考えることもできる．

本研究の動機となったディレクトリ空間の視覚化システムは，Google Maps [1] をはじめとする地図ブラウザのように情報空間をブラウズできないかとの発想で開発しており，それらの影響を受けている．

6. まとめと今後の課題

本論文では，ディレクトリ構造の視覚化においてノード配置を安定化するために，ディレクトリ構造に含まれている論理構造に着目して，抽出した論理構造を基にノード配置を決定することを提案した．具体的には，論理構造の抽出アルゴリズムを提案し，それを用いたノード配置の安定度に関する検証を行うための簡単な予備実験を行った．今後の課題としては，論理構造抽出アルゴリズムの精度向上，およびディレクトリ構造ブラウジングのユーザインターフェースにおける，安定したノード配置の利用者への影響の調査，などを行う予定である．

謝　　辞

ゼミなどでご議論いただきました筑波大学図書館情報メディア研究科杉本重雄教授，阪口哲男助教授，永森光晴講師に感謝いたします．

文　　献

[1] Google Maps, http://maps.google.com/

[2] Brian Johnson and Ben Shneiderman : Treemaps: A space-filling approach to the visualization of hierarchical information structures, Proc. of the 2nd International IEEE Visualization Conference, pp. 284–291, San Diego, October 1991.

[3] G. G. Robertson, J. D. Mackinlay, and S. K. Card : Cone Trees: Animated 3D visualizations of hierarchical information, In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI’91), pp. 189–194. ACM Press, 1991.

[4] John Lamping, Ramana Rao, and Peter Pirolli : A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, Computing Systems (CHI’95). Addison-Wesley, May 1995.

[5] D.K. Gifford, P. Jouvelot, M.A. Sheldon, and J. O’Toole, ”Semantic File Systems,” Proc. ACM Symp. Operating Systems Principles, pp. 16-25, Oct. 1991.