平成 24 年度卒業論文

論文題目

キーワードの含有率を用いたクラウドファイルシステム

神奈川大学 工学部 電子情報フロンティア学科
学籍番号　200902763
竹村　孝太

指導担当者　木下宏揚　教授

目 次

図 目 次

第1章　序論

インターネットが登場したことにより，人間の生活は大きな変化を迎えた。具体的な例としては，誰もが簡単に特に移動することもなく瞬時に情報を入手することが可能となったことや，Facebook などの SNS（Social Networking Service）により世界規模でコミュニケーションを行えるようになったことが挙げられる．利便性が高い環境故に，利用者は増加し，そうした人々を対象にインターネット上でサービスを提供する企業も増えていった．[1]

インターネットが普及し，近年の社会ではクラウド・コンピューティングが注目されている．従来のコンピュータの利用方法では，企業や個人がコンピュータのハードウェア，ソフトウェアやデータなどを自分自身で保有・管理していたため，管理運用のコストが高く，データ管理の手間もかかった．それに対してクラウドコンピューティングは，サービス利用料金を支払うことでインターネット上でそれらをサービスとして利用するため，従来に比べてコストが削減される．[2][3]

こうして利便性が高まるのは望ましいことではあるが，大規模で利用者が多い故に多くの問題も出てくる．

インターネットの利用が増えるとともに，インターネット上に多種多様な情報（ファイル，ページ）が散在し，現在も増加の一途を辿っている．それにより，情報漏洩や必要な情報のみを取り出すことが困難であるという問題が出てきた。それを解決するためのシステムとしてクラウドファイルシステムという構想がある。これは，インターネット上の情報をある規則の元に扱いやすいように再構成した上で，ネットワーク透過的に利用できるシステムである．

クラウドファイルシステムの構成要素の１つとして視覚的に判りやすいインターフェースを作ることがあげられる. これにより様々な情報の中から必要な情報のみを取り出すことが容易となってくる．

そこで，関係性のあるファイルを集めるシステムを考えたいと思う. これについて群知能を用いた研究がなされており，以下にその説明を記す．

その研究ではまず，PC 上のファイルをインターネット上の情報に見立て，研究を行っている．

従来の木構造のファイルシステムでは自らファイルをフォルダーに分けて保管していたため，ファイルの量が増えるにつれてファイルの所在が把握しきれなくなるなどの問題が出てきた．そこでファイルに対して群知能である ACO や，Boid の Separation，Alingment，Cohesion を適用することにした．それにより，大量の

ファイルが類似したもの同士で自動的に群れを作成することとなる．さらにファイルに重要度を設けることで，群れの中でも重要であるものとそうでないものを判別できるようにした．つまりファイルに群知能を適用することで，人間がファイルを管理する手間を軽減させると共に，ファイルの関係性と重要性を視覚的に判りやすくしようと試みたのである．ファイルが自動的に群れを作成するため，予想外なファイル同士の関係性が見えてくる可能性があることも恩恵の一つである．

ファイルの関係性と重要度を定める指標として以下のようなものが提案されていた．

『重要度について』

- 使用頻度

使用頻度が高いファイルを重要度が高いとする．

- 時系列

仕事や学校での課題，研究などの締切を設定する．締切が近いファイルは重要度が高いとする．また締切が近づいてくると自動的に重要度が高くなる．

- 依存関係

ファイル同士の重要度の依存関係のこと．A の仕事の前に B の仕事をやらなくてはならない場合，重要度は A¡B となる．

- ノルマ

ノルマが設定されている仕事などに対して，ノルマが達成されていない場合は重要度が高いとする．ノルマが達成された時点で重要度は低くなる．

『関係性について』

- タグ

ファイルに対してあらかじめタグというものを付けておく．例えば色である．ユーザーがそのファイルに対して色を指定する．卒業研究に必要なファイルであれば青色，趣味に関係するファイルであれば赤色といったように色を付ける．または，ファイルに対して自らキーワードを付ける．ファイルに付けた色やキーワードが関係性になる．

こうした仕組みを開発・実装するために Android を使用している．ファイルが群れを成すようなアプリケーションを開発するに当たり，画面上のボールが動き回り，画面の境界をバウンドしたり，ボール同士が衝突するとバウンドするサンプルアプリケーションが利用できるということもあり，それを元に開発が行われた．現在も開発途中である．[4]

この研究の中で私自身が着目した点は，ファイル同士の関係性や重要度についてである．

上記の研究における情報同士の関係性は全て，ユーザーが自ら付与したものである．全てのファイルに対してタグをつけて関係性を持たせるため，ユーザー自身の手間が掛かる．さらに，色分けやキーワードの付与により関係性を持たせるため，木構造のファイルシステムにおけるフォルダーを用いた管理に近い点がみられ，思いもよらないファイル同士の関係性を見つけられるかもしれないという恩恵が受けられないように思える．

ファイルの重要度に関しては，使用頻度や時系列が視覚的に明らかになるという点で，実用化された際には非常に役立つように思われる．

ファイルの整理を行う上ではこれらの関係性と重要度でも十分と思われるが，予想外なファイルの関係性を見つける可能性を含め，よりユーザーにとって恩恵をもたらすシステムとなるよう，ファイル同士の関係性と重要度について考察したい．

ここで解決したい問題点を二つにまとめる．

１つ目は，情報同士の関係性の持たせ方である．なるべくユーザーの手間が掛からず，ユーザーの予期しないファイル同士の関係性が見つかるような方法であることが望ましい．

２つ目は，インターネット上の情報に対しても有効であるような重要度が足りない点である．先に挙げた時系列やノルマといった重要度は，PC 上のファイルに対しては有効であるが，インターネット上の情報に対しては扱いづらい．使用頻度に関してはどちらにも有効であるが，これだけで情報の重要度を決定するのも些か心もとないように感じる．

そこで，ファイルの内容に着目した関係性と重要度を考える．

まずは関係性についてである．ファイル同士の内容が類似していることを，ファイル同士の関係性として捉える．ファイルにタグをつけることによる関係性にはユーザーの主観が含まれるが，ファイルの内容の類似はユーザーの主観を除いて判断するため，ユーザーの手間を削減し，ファイル同士の予想外な関係性が見つかる可能性が大きくなる．

次に重要度についてである．ファイルの中身に着目した重要性ということで，ファイルの内容が濃いなら重要度が高く，内容が薄いなら重要度が低いとする．この重要度の決定方法も使用頻度による重要度と同じく PC 上のファイルにもインターネット上の情報にも有効である．

上記のように，ファイルの中身に着目することでファイルの関係性と重要度を説明することができる．しかし，問題なのは何を持ってファイルの内容の類似を判断し，ファイルの内容の濃さを表すかである．

今回の研究では，ファイルの内容に着目した関係性と重要度の決定について,『キーワードの含有率』を提案する．これはファイル内にどの程度提示したキーワードが含まれているかを示す指標である．ファイル内に提示したキーワードが含まれているかどうかを調べることで，そのキーワードについて関係性があるファイルを検出する．そして，関係性があるファイルの中でも，含むキーワードの量に差が出るため，これにより重要度も定義することができる．

つまり，キーワードの含有率を算出することで，関係性と重要性の双方を一括して定義することができる．

クラウドファイルシステムの目的の１つとして，様々な情報をインターフェース上で視覚的に判りやすく，かつ扱いやすくするため，何かしらの関係性の下で情報の群れを作成し，何かしらの重要度により群れの中での情報の位置を決定する．先行研究ではこの関係性と重要度を分けて考えていたため，労力がかかり困難であった．しかしキーワードの含有率は双方を一括して定義できるため，より目的を実現しやすいと言える．

第2章　基礎知識

2.1　クラウド・コンピューティング

2.1.1　定義

アメリカ国立標準技術研究所（NIST）により，クラウドコンピューティングは以下のように定義されている．

クラウドコンピューティングとは，ネットワーク，サーバー，ストレージ，アプリケーション，サービスなどの構成可能なコンピューティングリソースの共用プールに対して，便利かつオンデマンドにアクセスでき，最小の管理労力またはサービスプロバイダ間の相互作用によって迅速に提供され利用できるという，モデルの一つである．このクラウドモデルは可用性を促進し，５つの基本特性と，３つのサービスモデルと，４つの配置モデルによって構成される．

しかし，世間一般ではバズワードといわれるほどにその定義はあいまいであるため，インターネット経由によるサービス利用というくらいに捉えておくのが無難であるように思える．[5]

2.1.2　サービス形態

クラウドファイルシステムにおけるサービス形態は以下の３つに分類される．

- SaaS

SaaS（Software as a service）とはソフトウェアをインターネット経由のサービスとして利用できるようにしたサービス形態である．

従来のソフトウェアの販売は，ソフトウェアをパッケージ製品としてユーザーに販売する形態であり，ユーザーは自分の持つPC上でソフトウェアを稼働して利用していた．

それに対してSaaSでは，ソフトウェアを提供者側で稼働させ，それをユーザーがネットワークを通して利用し，サービス利用料金を支払うという形態である．

利点としては，必要とする機能を必要に応じて利用でき，使用した期間と量の分だけサービス料を支払えばよい点と，ユーザーによるソフトウェアの管理やバージョンアップなどの手間が掛からないことなどが挙げられる．

欠点としては，サービスを提供する側のサーバーに異常が生じたときや，ネットワークが繋がらない場所では利用できないことが挙げられる．

サービスの具体例

- PaaS

PaaS（Platform as a Service）とはソフトウェアを開発・稼働するためのプラットフォームを，インターネット経由のサービスとして利用できるようにしたサービス形態である．

従来，企業が業務用のシステムなどを構築する際には，開発あるいは運用のためのハードウェアやOS，開発環境，ミドルウェアなどを購入し組み合わせることでプラットフォームを構築・維持する必要があった．

それに対してPaaSでは，プラットフォームをインターネット上のサービスとして利用できるため，運用・管理に必要なコストの大幅な削減を実現した．

- IaaS

IaaS（Infrastructure as a Service）とは情報システムに必要な機材や回線などの基盤（インフラ）を，インターネット経由のサービスとして利用できるようにしたサービス形態である．

従来，企業が業務用のシステムなどを構築する際には，開発あるいは運用のための事業所や機材，回線，OSやミドルウェアなどのソフトウェア環境，開発環境などを購入し，これらを組み合わせてシステムが稼働するためのインフラを構築・維持する必要があった．

それに対してIaaSは情報システムの稼働に必要な基盤をインターネット上のサービスとして利用できるため，大幅なコスト削減を実現した．

IaaSとPaasの違いを挙げると，Paasがアプリケーションやソフトウェア環境であるのに対し，IaaSは機械や回線などのハードウェア環境に重きを置いた用語である．

2.2 群知能

生物の個々の動きだけを見ても，特に秩序だった行動は見られないが，群れとして集団で行動する際には秩序が見て取れる．そのような生物の群れとしての行動を群知能と呼ぶ．[6]

2.2.1 Boid

Boid は，１９８７年に Craig Raynolds により発表された理論で，３つのルール（Separation，Alingment，Cohesion）を規定することで鳥の群れのをシミュレーションすることができるというものである．Boid というモデル名は，鳥もどきという意味の言葉である「bird - android（バード・アンドロイド）」が短くなったことに由来している．
群れの一連の動きは，自分自身と仲間との間の距離を最適に保とうとするルールを重要視している．このような３つの単純な行動規範をそれぞれの個体が持ち，全体として複雑な群れの行動が創発する．[7]
Boid における３つのルールは以下の通りである．

- Separation（衝突を回避するルール）

Separation は図 2.1 のように，それぞれの鳥が群れの仲間の鳥や障害物などにぶつからないように，接近しすぎたら離れるというルールである．

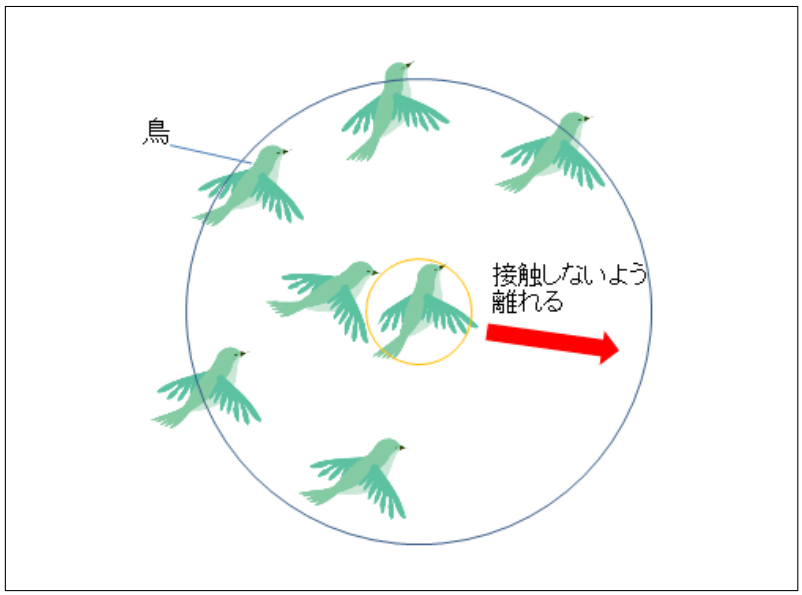

図 2.1: Separation

- Alignment（周りと速度と向きを合わせるルール）

Alingment は図 2.2 のように，周りの鳥たちと速度と向きを合わせることで，距離を空けすぎないようにするルールである．これにより，個々が散らばることなく群れを形成することができる．

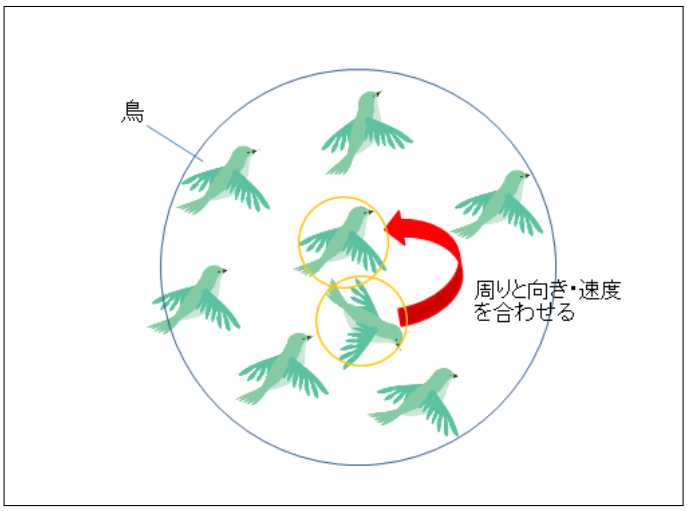

図 2.2: Alingment

- Cohesion（群れの中心に向かって飛ぶルール）

Cohesion は図 2.3 のように，仲間の鳥が多くいる群れの中心に向かって飛ぶことで群からはぐれないようにするルールである．

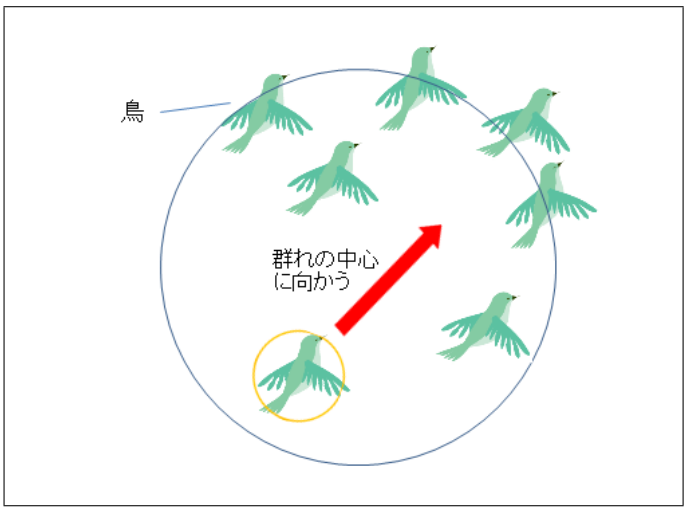

図 2.3: Cohesion

2.2.2 ACO（蟻コロニー最適化）

ACO（Ant Colony Optimization）とはアリの摂食行動から着想を得たアルゴリズムである。

実世界では，アリは始めランダムにうろつき，食物を見つけるとフェロモンの跡を付けながらコロニーへ戻る．他のアリがその経路を見つけると，アリはランダムな彷徨を止めてその跡を辿り始め，食物を見つけると経路にフェロモンを付けて補強しながら戻る。しかし，時間とともにフェロモンの痕跡は蒸発しはじめ，その吸引力がなくなっていく．その経路が長いほどフェロモンは蒸発しやすい．それに対して、経路が短ければ行進にも時間がかからず，フェロモンが蒸発するよりも早く補強されるため，フェロモン濃度は高いまま保たれる．

従って，あるアリがコロニーから食料源までの良い（すなわち短い）経路を見つけると，他のアリもその経路を辿る可能性が高くなり，正のフィードバック効果によって結局すべてのアリが 1 つの経路を辿ることになる．図 2.4 を参照されたい．

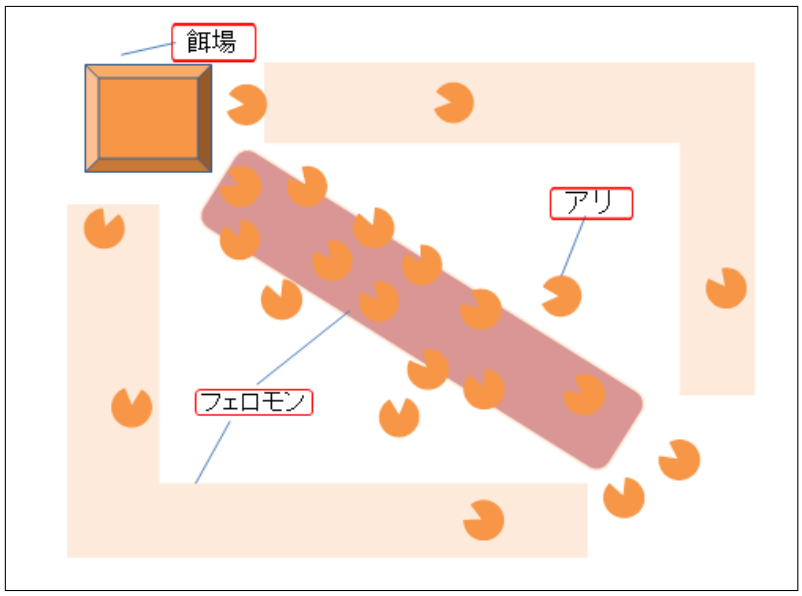

図 2.4: ACO

2.3 Android

Android とは，スマートフォン・タブレット PC などのモバイルデバイス向けのオープンでフリーなソフトウェアプラットフォームである．ここで言うソフトウェアプラットフォームとは，アプリケーションやアプリケーションフレームワークのみではなく OS やミドルウェアなども含んだ広範にわたるソフトウェアの集合体を指す．このことから、Android ではソフトウェア開発は大きく 2 つの視点に分けて考えることができる．[8][9]

- ある端末の上で動作する Android プラットフォームを作る

Android はソフトウェアの集合体であるため，最終的にはあるハードウェアの上に載せて動かす必要がある．Android とハードウェアを結び付けていく作業のことをポーティング（porting）と呼ぶ．この領域の開発を行うには，特に Linux カーネルやデバイスドライバに関する知識が必要不可欠．アプリケーションの開発者は，Android プラットフォームを直接操作することはなく，アプリケーションフレームワークの機能を通じて利用する形になる．

- Android プラットフォームの上で動作するアプリケーションを作る

開発者は「Android SDK」として提供されているアプリケーションフレームワークの機能や開発ツールを用いてアプリケーションを開発する．Android でアプリケーションを開発するためには，Java やアプリケーションフレームワークに関する知識が必要になる．

2.3.1 Android のアーキテクチャ

アーキテクチャは大きく 4 つの層，5 つの領域に分かれている．

- Linux カーネル層

最下層となる「Linux カーネル」は、バージョン 2.6 を元にモバイルデバイス向けに変更が加えられている．システムサービスをはじめとしたアプリケーションを実行するために必要な基本的な機能を提供する，いわば基礎となる層．他の 3 つの層で動作する機能は，この Linux カーネル上で動作する．

- ライブラリ層

「Linux カーネル」層の上位層は、「ライブラリ」層となる．Android は、さまざまなコンポーネントから使用されるライブラリを提供する．ライブラリは，C や C++言語で作成されたライブラリを含む．これらの機能は，基本的にアプリケーション開発者が直接使用することはなく，上位層であるアプリケーションフレームワーク層の機能を通じて使用されることになる．

- Android ランタイム

「ライブラリ」層と同レベルの機能として「Android ランタイム」がある．Android ランタイムは Java 言語に準拠するコアライブラリと Android アプリケーションを実行する Dalvik 仮想マシンにより構成されている．

- アプリケーションフレームワーク層

「ライブラリ」層の上位層は、「アプリケーションフレームワーク」層になる．アプリケーション開発者は，SDK にあらかじめ含まれているアプリケーションで使用されているものと同じクラスを使用することができる．アプリケーションを構築する際に使用するアーキテクチャは，コンポーネントの再利用が簡単にできるように設計されている．

- アプリケーション層

最上位の層が「アプリケーション」層。SDK には，電話や Web ブラウザなどの実行可能なアプリケーションがあらかじめ含まれている．アプリケーション開発者が作成したさまざまなアプリケーションもこの層に位置づけられる．

2.4 家族的類似性

家族的類似性とは，オーストリアの哲学者であるルートヴィヒ・ウィトゲンシュタインの著書「哲学探究」で語られている言語哲学・認知言語学上の概念である．

これは語の意味に必ずしも共通の本質が存在するわけではなく，様々な類似性が隙間なく重なり合うことで，全体が直接的にあるいは間接的にゆるい関係で結ばれているという概念である．これに関してウィトゲンシュタインは「ゲーム」という言葉を取り上げている．そして,「ゲーム」と呼ばれている全ての対象を特徴づけるような共通の意義は存在せず，実際には「勝敗が定まること」や「娯楽性」など部分的に共通する特徴によって全体が緩くつながっているいるに過ぎないと指摘している．実際，ゲームと呼ばれるものには「ゲーム理論」,「テレビゲーム」や「スポーツの試合」などがあり，全てに共通する本質は見受けられない．しかし，全てにおいてゲームという言葉が当てはまることから見ても，何かしらの共通性のようなもので直接的，ないしは間接的に繋がっているように思われる．[10][11]

2.5 形態素解析

文章を構成する要素のうち，意味を持つ最小の単位を形態素という．その形態素ごとに文章を分解し，辞書を利用して品詞を調べたり，内容を判別したりすることを形態素解析という．

英語では「Thank you.」の「Thank」,「you」が形態素に当たり，文章が単語ごとに区切られているため，解析が比較的容易であるとされている．

しかし，日本語の文章は単語ごとに区切られているわけではなく，すべての文字が続けて書かれているため，形態素ごとの分割が困難であるとされている．

形態素解析の技術はPCや携帯電話におけるかな漢字変換などに応用されている．

2.5.1 形態素解析における問題点

- 単語の境界を判別する上での問題

日本語の文章は全て文字が連なっているため，文章によっては様々な単語の区切り方が出てくる．例えば,「うらにわにはにわにわにはにわにわとりがいる.」という文章について考えてみると，

「裏庭には二羽庭には二羽鳥がいる」

「裏庭に埴輪庭には二羽鳥がいる」

「裏庭に埴輪庭には鶏がいる」

のように，様々な解釈が出てくる．この中から正しい区切り方を判別するためには，文脈，筆者の意図や背景をくみ取る必要があるため非常に困難である．

- 品詞を判別する上での問題

この問題は日本語よりも英語において顕著である．例えば「front」という単語には「正面」,「正面の」,「正面に」,「面する」のように様々な意味があり，それぞれの意味は別の品詞である．品詞は文章の構造と深く結びついているため，判別が困難である．

- 未知語の問題

未知語とは辞書に存在しない単語のことで，固有名詞（人名，地名，商品名など），擬音語，擬態語や顔文字などがそうとみなされやすい．形態素解析では辞書によって品詞分類を行うためこのような文字を扱うのが困難である．

- 文法が誤っている文章の解析における問題

話し言葉やメールで使われる言葉は，扱う人間にとっては意味が通じても，文法的には誤りがある場合がある．また正しく文法を使ったつもりでも書き誤っている場合もある．そういった文章に対しては正しい解析が成されない．特に話し言葉やメールの文章は時代の流れとともにルーズになる傾向があるため，それに合わせた解析を行うのは困難である．

2.5.2 ChaSen（茶筌）

茶筌は，奈良先端科学技術大学院大学松本研究室で開発された形態素解析のツールであり、自然言語学処理研究に資するため無償のソフトウェアとして開発されたものである．図 2.5 は解析時の画面である．

茶筌という名前は，開発拠点である奈良先端科学技術大学院大学のある奈良県生駒市高山町が，日本有数の茶筌の産地であることからつけられた．

茶筌では，入力した文章を形態素ごとに分解し品詞分類をするだけでなく，読み，発音，活用なども分析できる．解析する際，文章を直接入力するほかに，文章ファイルを読み込ませて別のファイルにその結果を出力することも可能である．[12]

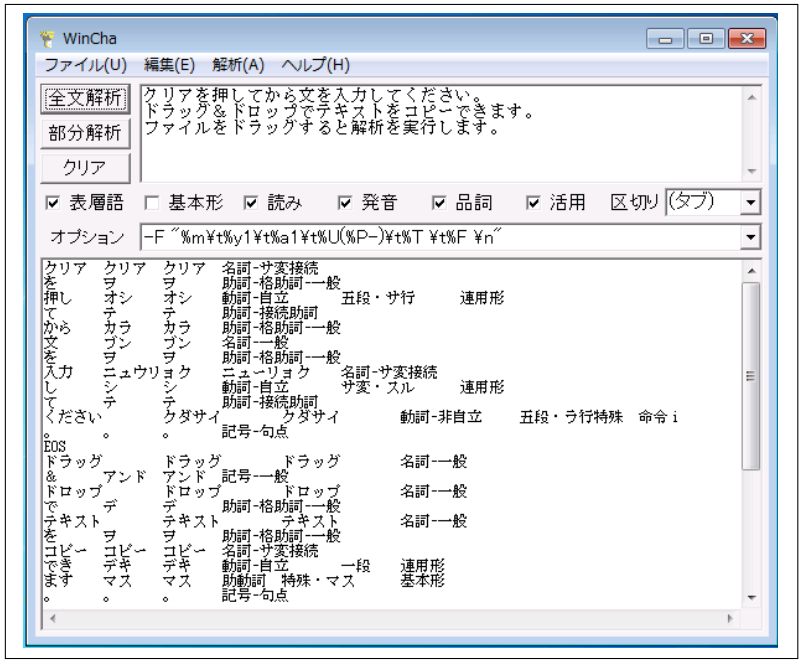

図 2.5: 茶筌による解析のサンプル

2.6 Perl

2.6.1 概要

Perlとは１９８７年にラリー・ウォールによって開発されたプログラミング言語で，UNIXのテキスト処理のための言語である．Perlは Practical Extraction and Report Language（実用的情報抽出レポート言語）の略である．

これは実用性と多様性を重視した言語であり，webアプリケーション，システム管理，テキスト処理などのプログラムを書くのに広く用いられている．現在では電子掲示板などに用いられるCGIプログラムの多くがPerlによって記述されている．

Perlはインタープリタ型の言語であり，コンパイラーを必要とせず，テキストエディタによって記述したプログラムに拡張子「.pl」を付与して保存し，コマンドプロンプトなどから実行することができる．

2.6.2 利点と欠点

Perlはインタープリタ型言語であるため高速計算にはあまり向いていないという欠点がある．しかし，動作を確認しながら順次プログラムを記述できるため開発効率がよく扱いやすい言語であるともいえる．Perlの正規表現は非常に高性能なため言語処理を行う上では最適である．[13][14][15][16][17][18]

第3章　群知能による群れ作成に向けたパラメータの定義

クラウドファイルシステムという構想の中で，インターネット上の多種多様な情報の中から必要な情報のみを取り出し情報の群れを作成することで，インターフェース上で視覚的に判りやすくするシステムを開発することが最終的な目的である．その結果として，情報同士の予想外な関係性が見つかり，人間の創造性や思考を刺激・支援することに繋がることも意図するところである．その際にBoidなどの群知能を用いて群れを作成することとなる．

今回の研究では，Boidによってファイルの群れを作成する際のパラメータを，群れが集まる力（「情報同士の関係性」と「情報の重要度」）に着目して提案し算出することが目的である．

ただし，今回も先行研究に習いPC上のファイルをインターネット上の情報に見立て行う．

3.1 キーワードの含有率

3.1.1 関係性

ファイルが集まり群れを作る際にまず問題となるのは，なぜ集まるのかである．そこで生物の群れに着目してみる．

生物の群れを見たとき，その群れに属した個々には何かしらの共通点がみられる．その共通点とは種であったり環境への対応力であったりと様々である．鳥と昆虫が群れを作ったり，鳥と魚が群れを作ったりするのを見たことがないのは，まさにこの共通点がないことによるものと考えられる．

共通点を持っていること，つまり関係性があることが群れを作る際の集まる力になるといえる．ただし，共通点があるから全てのものが群れを成すわけではなく，共通点が少ないため集まる力と成りえない場合や，共通点があっても群れを作る必要がない場合には群れを成さない．

故に，ファイル同士の関係性に成りうるものを考えれば，必要なファイルを集めるという必要性があるため，ファイルを集め群れを形成することが可能になるといえる．

その関係性として『キーワードの含有率』を用いることとする．これはファイル内にどの程度キーワードが含まれるかを表すパラメータである．

共通点と関係性については図 3.1 参照のこと．

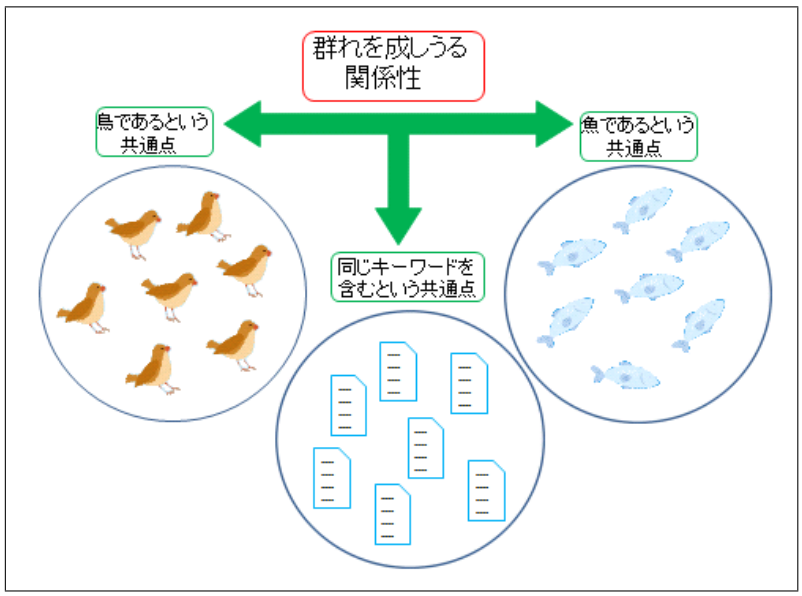

図 3.1: 共通点と関係性

3.1.2 重要度

ある関係性の下にファイルを集め群れを作ったとして，その中でどのファイルが特に必要なのかはわからない．そこで，その判断材料となるパラメータとして重要度を定義する．

群れの中で特に重要度が高いファイルが群れの中心に位置し，低いものが中心から外れるようにすることで，特に必要なファイルが視覚的に判断しやすくなる．

重要度は序論で前述した通り，ファイルの内容の濃さで表したい．そこで『キーワードの含有率』を用いることとする．

ファイルの重要度はファイルの内容の濃さにより決定され，ファイルの内容の濃さはキーワードの含有率により決定する．それぞれの関係性はシンプルで，図 3.2 の通りである．

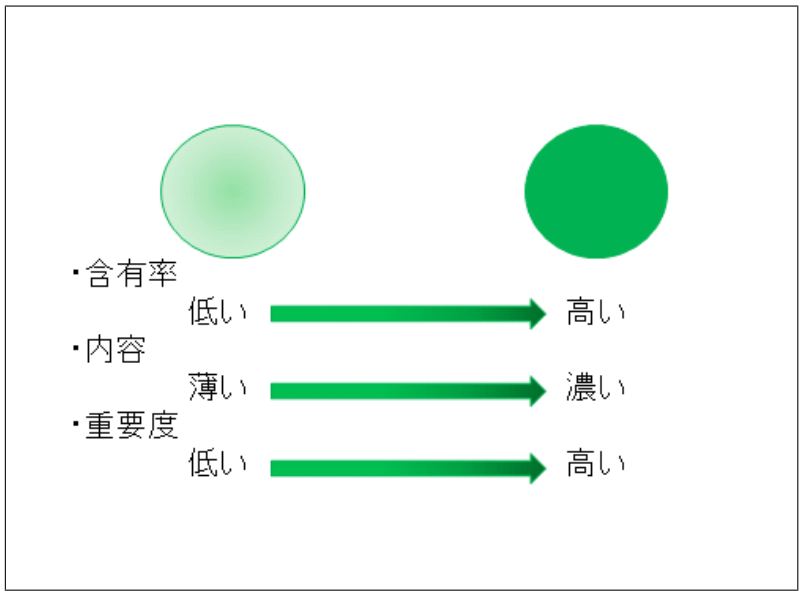

図 3.2: 重要度と含有率

3.1.3 キーワードの含有率について

キーワードの含有率とは，ファイル内にキーワードがどの程度含まれているかを示すパラメータである．前述した通り，これによりファイルの関係性と重要度を決定する．これによる利点を以下に挙げる．

- 関係性と重要度の一括決定

序論で紹介した中では，関係性と重要度が別々に決定されていたのに対して，この方法であればキーワードを１つ提示するだけで，双方をまとめて決定することができる．
ユーザーがファイルごとに関係性と重要度を付与する必要がなくなるため，労力を削減できる．

- ファイルの予想外な関係性の取得

ユーザーがファイル内に入っているキーワードの数や種類を全て把握しているということはまずありえない．そして，ファイル内のキーワードの含有率はプログラムにより算出する．つまり，ユーザーの思いがけないファイルにキーワードが入っていた場合，それは新たな関係性の発見となりユーザーの創造性や思考への刺激・支援となりうる．

キーワードの含有率は単にファイル内のキーワードの個数を示すものではなく，以下のようなものを要素として考える．

- キーワードの個数

ファイル内に含まれているキーワードの個数のこと．ファイル内にキーワードが含まれない場合は，そのキーワードによる関係性を持たないため，群れのを構成する個としてみなされない．もしあるキーワードを含んでいた場合，その個数が多い程重要度が高くなり，少ない場合は低くなる．

- キーワードの色・大きさ

文章ファイル内でキーワードが色付けされていたり，大きさが変更されていたりした場合，そのキーワードの含有率が高くなる．主に文章中でキーワードにその様なしょちが成されていた場合，それは特別な意味を持つ場合が多い．つまり，重要度が高くなるということである．

- キーワードの位置

キーワードのファイル内での位置も重要度に関わる．例えばファイルのタイトルにキーワードが含まれていた場合と文章中に含まれていた場合とでは，タイトルに含まれていた場合のほうが重要度が高いと考えられる．それ以外にも，カギ括弧に含まれていた場合などに重要度が高いとして含有率を高くする．

含有率の要素は図 3.3 を参照のこと．

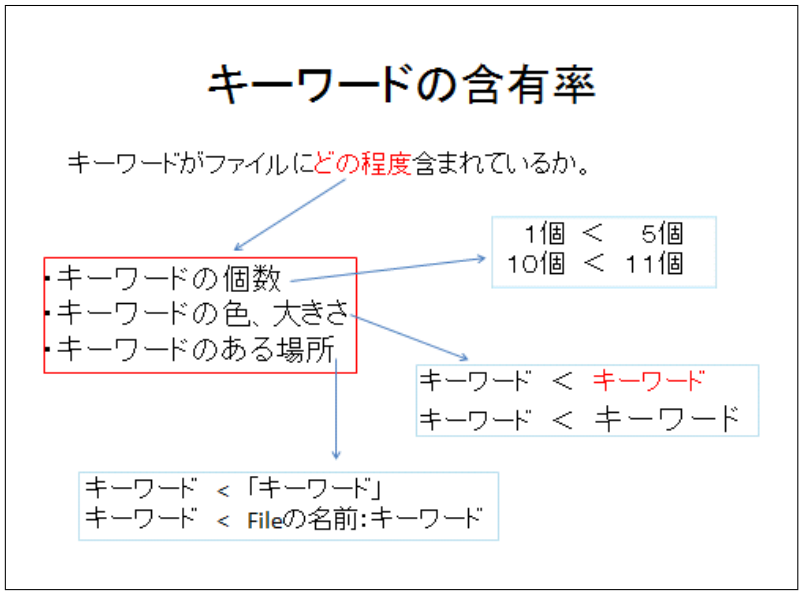

図 3.3: 含有率の要素

3.2 キーワードの提示方法

キーワードにより関係性と重要度を決定する際に，キーワードの提示方法を工夫することでより必要に応じた群れを作成することが可能となる．

3.2.1 単体で意味を成す語の提示

キーワードは単体で意味を成す語であるとする．単体で意味を成さない語ではファイルに関係性を持たせられないからである．あるいは，全てのファイルが関係していることになり，目的に応じた群れを作成できなくなる．つまり助詞や助動詞はキーワードとして望ましくない．

何かしらの意味を成すように思われる語でも，それによってファイルに関係性を持たせることが困難であると思われる語も望ましくない．代名詞や感嘆詞などがそれにあたる．代名詞は使用される文脈により成す意味が変わり，感嘆詞は感

動していることが伝わるものの，何に対しての感動なのかは文脈により変わるため，ファイルに対して関係性を持たせるのは困難である．

最も望ましいのは，主語と成りうる名詞である．名詞はその語自体が指し示す対象が比較的明確であり，ファイルに関係性を持たせる上では最適である．

3.2.2 キーワードの個数と組み合わせ

一つの語が持つ意味は家族的類似性による緩い繋がりにより幅広い．つまり意味集合とでも呼べるような範囲が存在する．その範囲に引っかかるファイルは全て関係性があると言える．

あるキーワードの意味集合の範囲が狭い場合には，群れの規模はそれほど大きくなく必要なファイルをのみを取り出す点では望ましいが，思いがけないファイルの関係性を見つける点では望ましくない．キーワードの意味集合の範囲が広い場合には，思いがけないファイルの関係性を見つけられる可能性は高いが，群れの規模が大きく必要なファイルのみを取り出すという点で望ましくない．

そこで複数のキーワードを組み合わせて提示することで，欠点を補うのである．キーワード A，B の持つ意味の集合を Am，Bm とした場合の組み合わせの一例を挙げる．

- キーワードの意味の範囲が狭い場合（図 3.4 参照）

キーワード単体：必要なファイルのみを取り出す
複数のキーワード：多くのファイルの関係性が見られる

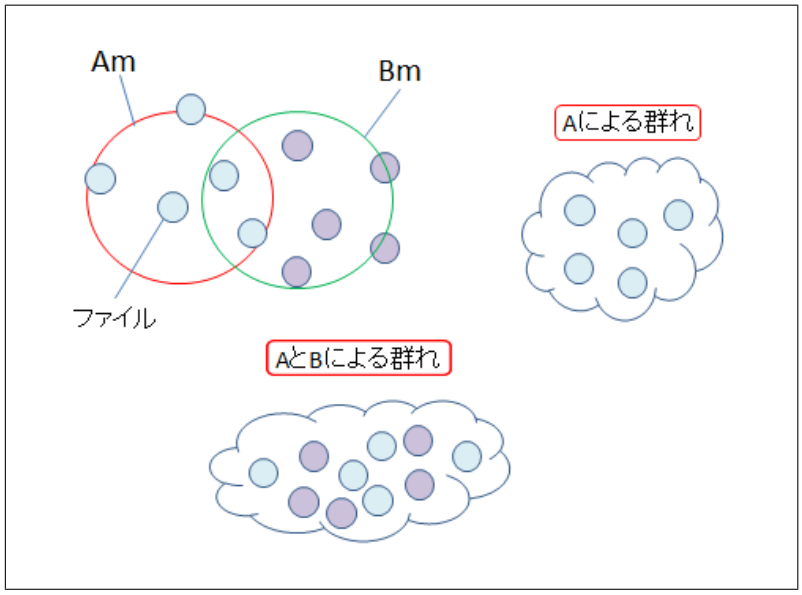

図 3.4: 意味集合が狭い場合の組み合わせ

- キーワードの意味の範囲が広い場合（図 3.5 参照）

キーワード単体：多くのファイルの関係性が見られる
複数のキーワード：必要なファイルのみを取り出す

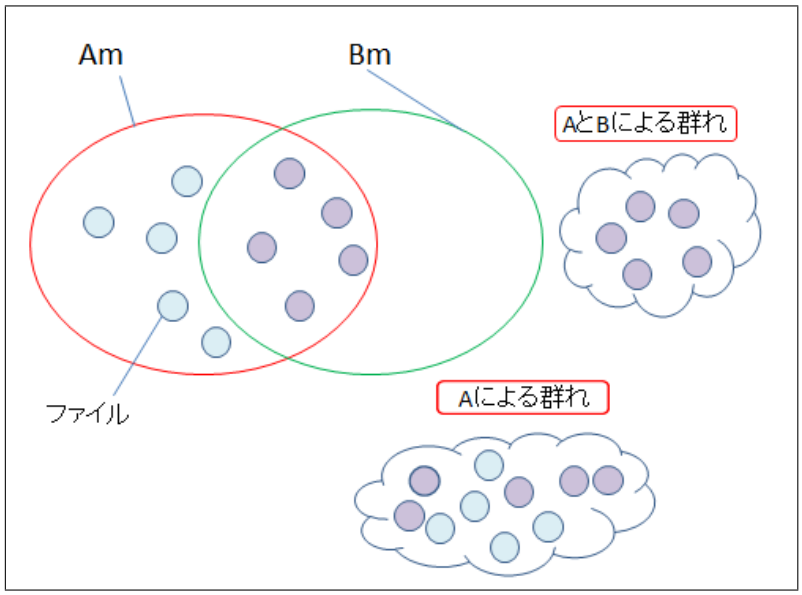

図 3.5: 意味集合が広い場合の組み合わせ

第4章　含有率の算出

4.1　含有率の式

キーワードの含有率の要素について「キーワードの個数」,「キーワードの色・大きさ」,「キーワードの位置」の３つを挙げたが，今回はその中の「キーワードの個数」のみを用いて算出する．キーワードの個数は含有率の要素の中でも，最も比重の大きい要素と思われる．他の２つの要素はこれの補助要素と言っても差し支えないように思われる．ゆえに，今回はキーワードの個数のみに着目した含有率を求める．その際の式は以下の通りである．

- キーワードの含有率の式

$$CBP = C\frac{K}{A} \times 100$$

（CBP:含有率（Content by percentage）C:個人による補正　K:キーワードの個数　A:ファイル内の総単語数）

まずファイル内に含まれるキーワードの個数を調べる．次にファイル内に含まれる単語の総数を調べる．そしてキーワードの個数をファイル内の単語の総数で割ることで含有率を算出する．

キーワードの個数のみでファイルを比較した場合，キーワードの個数が同数であった場合に含有率が等しくなる．そこで，ファイル内の単語の総数で割ることとする．これによりキーワードの個数が同じ場合であっても，ファイル内でのキーワードの密度を比較することで含有率の差を生み出すことができる．しかし，これにより算出された含有率がユーザーにとって最適な関係性と重要度をもたらすとは限らない．

そこで個人による補正を加える余地を与える．キーワードの含有率は高いがユーザーにとっては重要度が低い，またはその逆の場合に，任意でそのファイルに対して係数を加えることで，含有率による重要度とユーザーにとっての重要度の溝を埋める．

4.2 プログラムによる算出

茶筌と Perl により含有率を算出する.

4.2.1 環境

- Perl

  言語処理に適したプログラミング言語.

- 茶筌

  形態素解析ツール.

- EmEditor

  Microsoft Windows 向けのテキストエディタ. [19]

4.2.2 手順

一連の流れは，形態素解析したファイルを Perl のプログラムに通してキーワードの数と単語総数を集計し，含有率の式に代入するというものである．ここでは，個人の補正を含めないものとする．さらに対象とするファイルは文章ファイルのみとする．詳しい手順は以下を参照されたい.

1. 環境のセットアップ

   算出に当たり必要な環境を整える．とはいえ，ダウンロードするのは茶筌と EmEditor のみであり，どちらもフリーのソフトウェアであるため入手は容易である.

2. ファイルの形態素解析

   ファイル内のキーワードを集計する際に Perl のプログラムでは，ファイル内の文章を位置ずつ確認していく．その際に対象となるキーワードを発見した時点で次の行へと進行してしまう．つまり，１行の中に２つキーワードが含まれていたとしても１つしかカウントされない．茶筌では形態素を１行１行改行して出力することができるため，その出力結果をプログラムに通すことで問題点を解決することができる.

   今回，含有率を求めるための文章ファイルのサンプルとして「haiku.txt」（図 4.1）を用いる．これを茶筌に通して文章ファイルにしたものを「haiku.cha.txt」（図 4.2）として保存する.

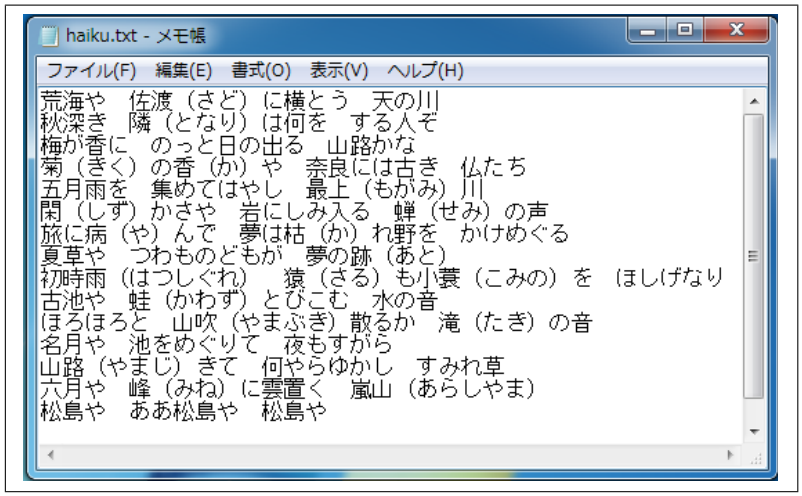

図 4.1: haiku.txt

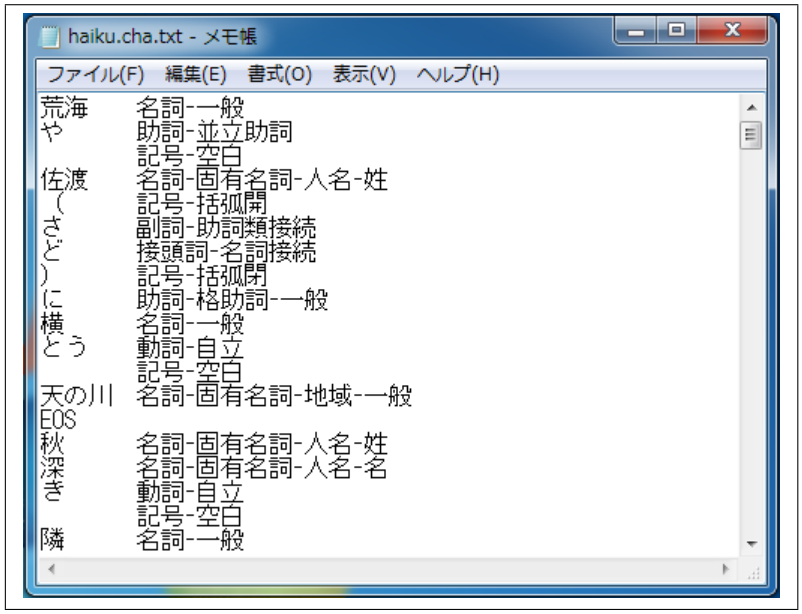

図 4.2: haiku.cha.txt

3. Perl による含有率の算出形態素解析により形態素ごとに改行されたファイルを Perl によるプログラムにかける．プログラムの実行後の含有率算出までのプロセスは以下の通りであり，結果は図 4.3 を参照されたい．

   - キーワードの数を入力する
   - キーワードを入力する
   - それぞれのキーワードの個数を集計する
   - ファイル内の総単語数を集計する
   - 含有率の式にこれらを代入し算出する
   - 結果を出力する

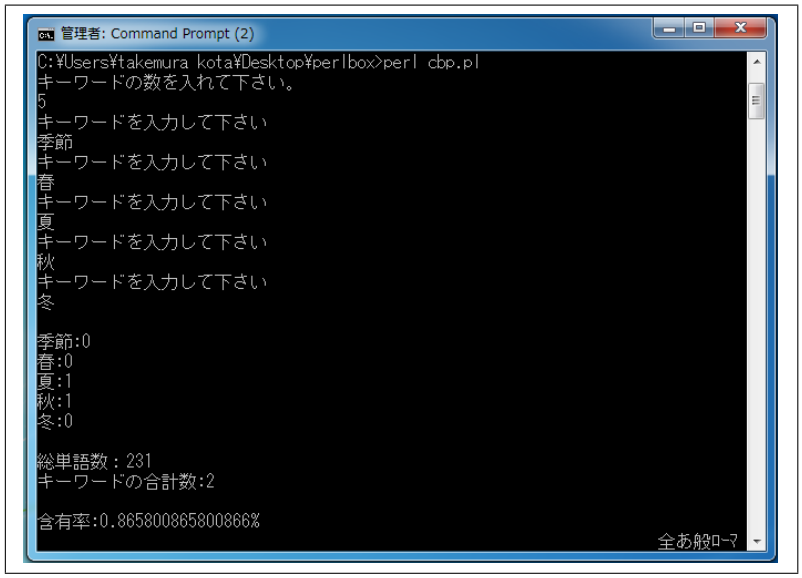

図 4.3: プログラムの実行結果

4.3 位置を考慮したキーワードの検出

今回，含有率の算出には用いなかったが，括弧などの中にあるキーワードの個数を検出するプログラムの実行結果を載せておく．

ただし，この検出にはいくつか問題点が存在する．

1. 茶筌を使用できない点

   キーワードが括弧内に存在するかどうかを判断するためには，括弧開きと括弧閉じが同じ行になくてはならない．しかし，茶筌を使用した場合ファイル内の括弧は未知語として分解，改行されてしまうため，キーワードの位置を

検出できなくなる（図 4.4 参照）．そのため，ユーザーがあらかじめ検出しやすいよう改行しておく必要がでてくる．

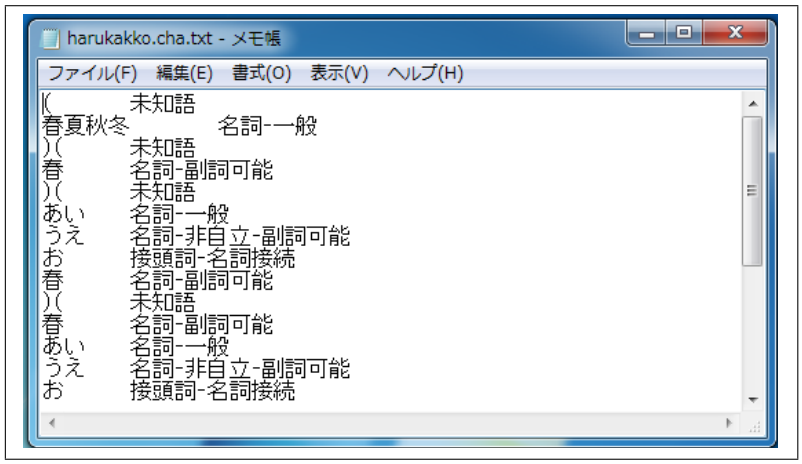

図 4.4: 茶筅による括弧の分解

2. 括弧内にキーワードが複数ある場合の問題

   perl では，ある行でキーワードをカウントすると，すぐに次の行に移ってしまう．このとき，括弧の中に複数キーワードが存在していたとしても，先に検出されたキーワードしかカウントされない．

以下は，あらかじめ括弧ごとに改行した文章ファイルをプログラムにかけた実行結果である．「春」という漢字が１０個あり，括弧で囲われたもののみカウントしている（図 4.5 参照）．

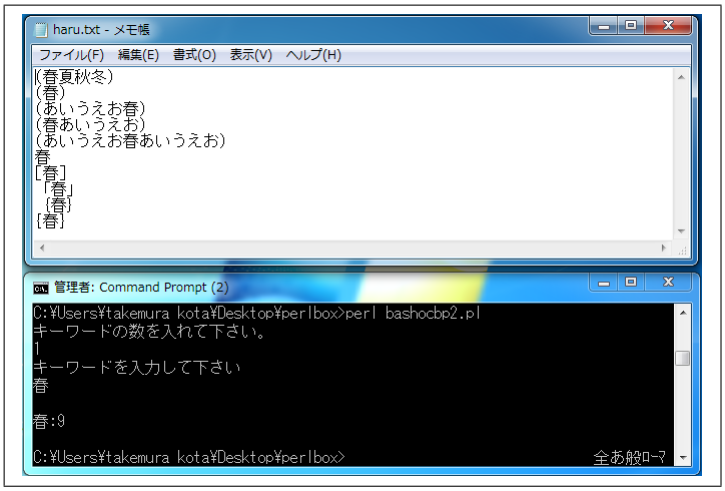

図 4.5: 位置を考慮したキーワードの個数の検出

第5章　結論

先行研究において，ファイル同士の関係性と重要度は別々に定義されていた．さらに，どちらもユーザー自身が全てのファイルに対してこれらを付与する形であったため，実際にクラウドファイルシステムに組み込む場合にユーザーの労力を考えると，やや難儀に思われた．

これに対し，今回算出したキーワードの含有率は，元々ファイル内に含まれている情報を元にファイル同士の関係性と重要度を一括して定義するため，ユーザーの労力を大幅に削減することが可能となる．これは，ファイルの内容に着目したものであるため，実際にファイルの群れを作成した際にも実用的であると言える．

さらにユーザーの手により関係性と重要度を決めるわけではないため，ユーザーの意図しない予想外なファイル同士の関係性を見つける可能性を広げ，それによるユーザーの創造性や思考を刺激・支援することにも繋がる．

最終的な目的であるクラウドファイルシステムという構想における，視覚的に判りやすいインターフェースの構築においても，キーワードの含有率およびキーワードの提示方法により目的に応じた情報の群れの作成が可能と思われる．

今後の課題としては,「キーワードの色・大きさ」と「キーワードの位置」の要素を含めた含有率を求めることである．そして，Android 上で Boid などの群知能のパラメータとして適用し，シミュレーションを行いたい．実際に群知能のパラメータとしてファイルに適用することで，現時点では想定していなかった問題点や利点を発見し，より利便性のあるシステムとなるよう努めたい．

ただし，毎回全てのファイルに対して含有率の算出を行うのは現実的ではない．そこで文法圧縮を用いたい．これにより，頻出単語などのデータがメタデータとして保存されるため，提示するキーワードがファイルの頻出単語とマッチした際に含有率を算出するようにすれば現実的となってくる．

付録

今回，研究で作成したソースコードを以下に添付する．ソースコードは Perl により記述している．

- 含有率の算出プログラム

```
#!/usr/bin/perl
use strict;
use warnings;

                                        #～キーワードの個数を求める～

print "キーワードの数を入れて下さい。\n";
my $kazu = <STDIN>;

my $i = 0;
my @keyword;
while($i < $kazu){
    print "キーワードを入力して下さい\n";
    $keyword[$i] = <STDIN>;
    $i++;
}
foreach(@keyword){
    chomp($_);
}



my @count;
for(my $y = 0;$y < $kazu;$y++){
    $count[$y] = 0;
}
```

```
my $num = 0;
foreach my $word(@keyword){
    open (IN, "haiku.cha.txt") or print STDERR "ファイルを開けません";
    while(<IN>){
        if(/$word/){
            $count[$num]++;
        }
    }
    $num++;
}

my $z = 0;
while($z < $kazu){
    print "\n$keyword[$z]:$count[$z]";
    $z++;
}
```

#～単語の総数を求める～

```
open (IN, "haiku.cha.txt") or print STDERR "ファイルを開けません";

my $kosuu = 0;
while(<IN>){
    if(/.+/){
        $kosuu++;
    }
}

open (IN, "haiku.cha.txt") or print STDERR "ファイルを開けません";

my $kaigyou = 0;
while(<IN>){
    if(/EOS/){
        $kaigyou++;
    }
}
my $sousuu = $kosuu - $kaigyou;
print "\n\n 総単語数 : $sousuu\n";
```

```
                                        #～含有率を求める～

my $goukei = 0;
foreach (@count){
    $goukei = $goukei + $_;
}
print "キーワードの合計数:$goukei\n";

my $cbp = ($goukei/$sousuu)*100;
chomp $cbp;
print "\n含有率:$cbp%\n";
```

- キーワードの位置を考慮した個数検出のプログラム

```
#!/usr/bin/perl
use strict;
use warnings;

print "キーワードの数を入れて下さい。\n";
my $kazu = <STDIN>;

my $i = 0;
my @keyword;
while($i < $kazu){
    print "キーワードを入力して下さい\n";
    $keyword[$i] = <STDIN>;
    $i++;
}
foreach(@keyword){
    chomp($_);
}

my @kakko;
my $h;
for($h = 0;$h < 9;$h++){
    for($i = 0;$i < $kazu;$i++){
        $kakko[$h][$i] = 0;
    }
}

my $j = 0;
my $word;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/\(.*$word.*\)/){
        $kakko[0][$j]++;
        }
    }
    $j++;
```

```
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/\[.*$word.*\]/){
        $kakko[1][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/\{.*$word.*\}/){
        $kakko[2][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/「.*$word.*」/){
        $kakko[3][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
```

```
    while(<IN>){
        if(/ {.*$word.*} /){
        $kakko[4][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/\".*$word.*\"/){
        $kakko[5][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/”.*$word.*”/){
        $kakko[6][$j]++;
        }
    }
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/\’.*$word.*\’/){
        $kakko[7][$j]++;
        }
    }
```

```
    $j++;
}

$j = 0;
foreach $word (@keyword){
    open (IN, "haru.txt") or print STDERR "ファイルが開けません";
    while(<IN>){
        if(/'.*$word.*'/){
        $kakko[8][$j]++;
        }
    }
    $j++;
}

my @kakkokei;
for($i = 0;$i < $kazu;$i++){
    $kakkokei[$i] = 0;
}



for ($i = 0;$i < $kazu;$i++){
    for($h = 0;$h < 9;$h++){
        $kakkokei[$i] = $kakkokei[$i] + $kakko[$h][$i]
}
}

print "\n";
$i = 0;
while($i < $kazu){
    print "$keyword[$i]:$kakkokei[$i]\n";
    $i++;
}
```

謝辞

本研究を行うにあたり，御指導頂いた木下宏揚教授，宮田純子氏に感謝いたします．また，ご多忙な中数多くの助言及び御指導を頂いた森住哲也氏に感謝いたします．

最後に，木下研究室の先輩方，並びに学友の面々にも感謝いたします．

2013 年　2 月
竹村　孝太

参考文献

[1] 横藤雅人，インターネット活用原論，2000
http://members.jcom.home.ne.jp/yosi009/katuhou-1.htm

[2] 水野信也，永田正樹，坂田智之，長谷川孝博，井上春樹，クラウド VPS 入門 運用・構築から高度な利用まで，静岡学術出版，2010

[3] 渡辺達也，中村有一，クラウドコンピューティングによるビジネスの変化，2009

[4] 磯村淳，木下宏揚，クラウドファイルシステム，2012

[5] 森 正弥，日経 BP 社出版局編『クラウド大全 The Complete Cloud Computing サービス詳細から基盤技術まで』，日経 BP 社，2009

[6] 「群知能」
http://www.sist.ac.jp/~kanakubo/research/swarm_intelligence.html

[7] 「Boid とは」
http://members.jcom.home.ne.jp/ibot/boid.html

[8] 「JavaDrive」
http://www.javadrive.jp/android/

[9] 「初心者のための Android アプリ開発」
http://www.hp3200.com/android-app-development/

[10] 「ウィトゲンシュタイン・セレクション」
http://www.geocities.jp/mickindex/wittgenstein/witt_other_quote.html

[11] 「IDEA-MOO ＜知の体験（家族的類似性）＞」
http://idea-moo.net/chi-taiken/ono08100301.html

[12] 「ChaSen – 形態素解析器」
http://chasen-legacy.sourceforge.jp/

[13] Jeffrey E.F.Friedl，詳説正規表現，歌代和正監訳，春遍雀來，鈴木武生共訳，オライリー・ジャパン，1999

[14] ラリー・ウォール，ジョン・オーワント，トム・クリスチャンセン著，近藤嘉雪訳,『プログラミング Perl』VOLUME 1，オライリー・ジャパン，2002

[15] 斉藤靖，小山裕司，前田薫，布施有人，新 Perl の国へようこそ，サイエンス社，１９９６

[16] 「perl とは？」
http://wisdom.sakura.ne.jp/programming/perl/perl1.html

[17] 「Perl とは」
http://php-web.net/introduction/whatis-perl.html

[18] 「Perl 講座 — Smart -Web Magazine」
http://rfs.jp/sb/perl

[19] 「EmEditor - 窓の杜ライブラリ」
http://www.forest.impress.co.jp/library/software/emeditor/

第6章　質疑応答

Q1:クラウドファイルシステムでファイルの群れを作成する際に，各々の見られたくないファイルなどは群れの対象外になるのか.（平岡隆晴助教授）

A1:なります．クラウドファイルシステムをインターネット上で実装した場合，オープンになっているファイルやウェブページを対象に群れの作成を行うため，見られたくないようなファイルなどは対象外となります.

Q2:キーワードの含有率はそれぞれのキーワードについて求めるものなのか.そして，求めた数値をどうするのか.（平岡隆晴助教授）

A2:キーワードの含有率はそれぞれのファイルに対して求めます．任意のキーワードを指定し，それについての含有率を各ファイルに対して求めます.各ファイルで求めた含有率を比較することで，群れの中での位置を定めます.