

Title	ソフトウェア成果物の設計根拠の抽出法
Author(s)	山内, 崇
Citation	
Issue Date	2005-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1859
Rights	
Description	Supervisor:落水 浩一郎, 情報科学研究科, 修士

修 士 論 文

ソフトウェア成果物の設計根拠の抽出法

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

山内 崇

2005 年 3 月

修 士 論 文

ソフトウェア成果物の設計根拠の抽出法

指導教官 落水浩一郎 教授

審査委員主査 落水浩一郎 教授
審査委員 片山卓也 教授
審査委員 鈴木正人 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

310119 山内 崇

提出年月: 2005 年 2 月

概 要

ソフトウェアは一般に新たな機能の要求や不具合により修正が必要となる。その際、ソフトウェア成果物が設計された根拠が残っていれば修正は容易である。しかし、ソフトウェアが大規模になると膨大な情報の中から必要とする設計根拠を探すのが難しい。本稿では、バージョン管理システム CVS と電子メールによるメーリングリストを用いた開発モデルを対象に、開発者が必要とする設計根拠を自動抽出するシステムを構築する。システムの実現には、ベクトル空間モデルと呼ばれる情報検索モデルを用いる。また、人間が検索するときのヒューリスティック加味して検索対象を絞り込み、精度の向上を図る。

目次

第1章	はじめに	1
1.1	背景	1
1.2	目的	1
1.3	本論文の構成	2
第2章	ベクトル空間モデルによる類似検索の実現	3
2.1	ベクトル空間モデル	3
2.1.1	類似検索の具体例	4
2.2	索引語の抽出	5
2.3	検索処理の流れ	8
2.4	ベクトル空間モデルの評価	9
2.4.1	実験の準備	9
2.4.2	実験結果の評価	9
第3章	ヒューリスティックの適用	13
3.1	ヒューリスティック	13
3.1.1	手作業による探索過程	13
3.2	ヒューリスティックの評価	15
3.2.1	実験の準備	15
3.2.2	コミットの日付とメールスレッドの関係	16
3.2.3	実験結果の評価	17
第4章	システムの実装	20
4.1	システムの構成	20
4.2	システムの使用方法	21
第5章	関連研究	25
5.1	Design Rationale	25
5.2	Hipikat	25
5.3	オープンソース開発支援のためのソースコード及びメールの履歴対応表示システム	26

第 6 章	おわりに	27
6.1	まとめ	27
6.2	今後の課題	27
	謝辞	29

目 次

1.1 システムの概略	2
2.1 文書群	4
2.2 索引語	4
2.3 検索質問	4
2.4 各文書の類似度	5
2.5 元の文章	6
2.6 不要語を除去した後の文章	6
2.7 接尾辞を除去した後の文章	6
2.8 本研究で用いた 571 語の不要語	7
2.9 検索処理の流れ	8
2.10 C9 のコミットログに含まれる索引語	11
2.11 C9 のメールスレッドに含まれる索引語	11
2.12 C4 のコミットログに含まれる索引語	11
2.13 C4 のメールスレッドに含まれる索引語	11
2.14 順位と類似度の関係	12
2.15 順位と類似度の関係を平均化	12
3.1 C9 のコミットログ	14
3.2 コミットログを基に最初にたどったメール	14
3.3 C9 のメールスレッド	15
3.4 コミットの日付とメールスレッドの関係	16
3.5 コミットの日付とメールスレッドの関係を示す図の凡例	17
4.1 システムの構成	20
4.2 起動直後の画面	21
4.3 ファイル・メニュー	21
4.4 mbox ファイルを開く	21
4.5 mbox ファイルを読み込み、メールスレッド化した画面	22
4.6 コミットの日付によるヒューリスティックの選択	22
4.7 コミットの日付と前後 n 日の n を入力	22
4.8 ヒューリスティック適用後の画面	23

4.9 “類似度の計算”を選択	23
4.10 類似度計算ダイアログ	23
4.11 類似度計算の途中経過	24
4.12 類似度計算の終了	24
4.13 類似度の降順に並び替えられたメールスレッド群	24

表 目 次

2.1	類似度の順位	5
2.2	正解の組み合わせと同じメールスレッドの順位	10
3.1	ヒューリスティック不適用と適用の場合の、順位と検索対象のスレッド数 .	18

第1章 はじめに

1.1 背景

多くのオープンソース・ソフトウェア開発では、バージョン管理システム CVS と電子メールによるメーリングリストが用いられる。バージョン管理システムはファイルの更新履歴を管理するシステムであり、ソフトウェア成果物に変更を加える際に、その理由や変更日時、変更者の名前などを付属情報としてリポジトリに保存する。メーリングリストは、複数のユーザが電子メールを用いて情報を交換するためのシステムである。メールの返信情報をもとに生成したメールスレッドから討議スレッドが取得できる。

ソフトウェアを修正する際には、開発者はソフトウェアがどのように設計されているかを把握しなければならない。そのために、過去の変更の付属情報であるコミットログを基に、変更に至るまでの討議を示すメールスレッドを探し出して理解する必要がある。

しかし、ソフトウェアの規模が大きくなると管理される情報も膨大な量となり、開発者は必要なメールスレッドを探すのが困難になる。FreeBSD の開発者向けメーリングリストを例に挙げると、一ヶ月に約 2500 通のメールが投稿されるため、開発者がすべてを逐一読むのは容易ではない。

1.2 目的

本研究では、先に述べた問題を解決するために、メーリングリスト・アーカイブからコミットログに対応するメールスレッドを自動抽出する手法を提案する。開発者はこのメールスレッドを読むことにより、変更に至るまでの討議を容易に理解できる。

コミットログに対応するメールスレッドを抽出するために、ベクトル空間モデルと呼ばれる情報検索モデルを用いる。ベクトル空間モデルでは、検索質問および検索対象となる文書群を多次元ベクトルであらわし、ベクトル間の角度を求めることにより類似度を得る。文書群を類似度の順に並べ替えることにより、大量の文書の中から目的の文書を労力をかけずに探し出せる。本研究では、検索質問にコミットログ、文書群にメールスレッド群を用いて、コミットログに対応するメールスレッドを検索する。

メーリングリスト・アーカイブのすべてのメールスレッド群にベクトル空間モデルを適用すると、必要な情報が含まれるメールスレッドが関係のないメールスレッドに埋もれることが予想される。そこで、開発者が手作業で探すときの手がかりをもとに検索対象を絞り込む方法を考える。開発者は手作業により必要とする情報を探すとき、たとえば変更が

加えられた日付に近いメールだけを読み、見つからなければ次第に範囲を広げるといった探し方をする。これをヒューリスティックとして定義し、検索対象のメールスレッドを絞り込む。ヒューリスティックを用いて絞り込んだメール群に対して、ベクトル空間モデルによる類似度計算を適用することにより検索結果の向上が期待できる。

1.3 本論文の構成

本研究で実現したシステムの概略を図 1.1 に示す。利用者がコミットログを指定すると、対応付け機構はコミットログとメーリングリスト・アーカイブに含まれるメールスレッド群を対応づけ、利用者に提示する。

次章以降の構成を以下に述べる。まず、ベクトル空間モデルの概略と具体例について 2 章で説明し、実験結果の評価を述べる。次に、3 章で手作業で探すときの手がかりであるヒューリスティックの概略と手作業による探索過程について説明し、ヒューリスティックを適用した実験結果の評価を述べる。4 章では、実装したシステムの構成について述べる。また、システム的使用方法について、実際にコミットログに対応するメールスレッドを抽出する手順を示す。

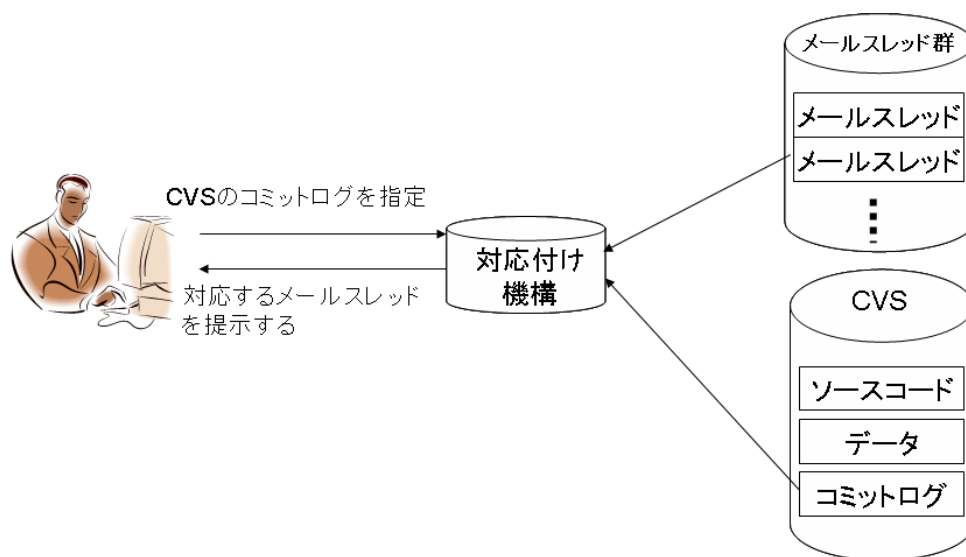


図 1.1: システムの概略

第2章 ベクトル空間モデルによる類似検索の実現

本章では、本研究で用いるベクトル空間モデルの概略と具体例について説明し、実際の検索処理手順と実験結果の評価について述べる。

2.1 ベクトル空間モデル

ベクトル空間モデル [1][2] は、文書をベクトルであらわすことにより類似検索を実現する、情報検索技術のひとつである。

検索対象となる文書を $D_1, D_2 \dots D_m$ としたとき、これらの文書には n 個の索引語 (2.2 節にて後述) が含まれているとする。このとき、文書 D_i を式 2.1 のベクトル $\vec{d}_i$ であらわす。

$$\vec{d}_i = \begin{bmatrix} d_{i1} & d_{i2} & \cdots & d_{in} \end{bmatrix} \quad (2.1)$$

ここで、 d_{ij} は文書 D_i 中の索引語 w_j の出現回数である。 $D_1, D_2 \dots D_m$ は、式 2.2 の $m \times n$ 行列 D によってあらわされる。

$$D = \begin{bmatrix} \vec{d}_1 \\ \vec{d}_2 \\ \vdots \\ \vec{d}_m \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \quad (2.2)$$

検索質問も文書と同じように索引語の出現回数を要素とするベクトルであらわす。検索質問ベクトルは次のようなベクトルとなる。

$$\vec{q} = \begin{bmatrix} d_1 & d_2 & \cdots & d_n \end{bmatrix} \quad (2.3)$$

ベクトル空間モデルでは、先に定義した検索質問ベクトル $\vec{q}$ と各文書ベクトル $\vec{d}_i$ の間の類似度を計算することにより、検索質問に類似する文書を見つけることができる。ベクトル間の類似度は以下の式であらわされる。

$$sim(\vec{d}_i, \vec{q}) = cos\theta = \frac{\vec{d}_i \cdot \vec{q}}{|\vec{d}_i||\vec{q}|} = \frac{\sum_{j=1}^n d_{ij}q_j}{\sqrt{\sum_{j=1}^n d_{ij}^2} \sqrt{\sum_{j=1}^n q_j^2}} \quad (2.4)$$

2.1.1 類似検索の具体例

具体例として、ベクトル空間モデルによる検索の様子を解説する。以下に示すのは、IBM PC/AT 互換機に関する文書群を検索する例である。図 2.1 に検索対象の文書群を示す。この文書群に含まれる単語の中から、索引語として図 2.2 に示す 7 つの単語を用いる。文書全体は、式 2.5 のような 7×5 行列 D によってあらわされる。索引語 ($w_1, w_2 \dots w_7$) の出現回数が、各文書 ($D_1, D_2 \dots D_5$) のベクトル ($\vec{d}_1, \vec{d}_2 \dots \vec{d}_5$) となる。

- D_1 : “Hang during USB device probes on CURRENT”
- D_2 : “ACPI problem: acpi_video on SONY VAIO PCG-Z1”
- D_3 : “USB keyboard problems: unloading USB driver”
- D_4 : “IBM Xseries 345 keyboard & mouse driver freezes”
- D_5 : “PS/2 mouse driver problem with acpi on -CURRENT”

図 2.1: 文書群

w_1 : mouse, w_2 : keyboard, w_3 : problem, w_4 : usb, w_5 : driver, w_6 : current, w_7 : acpi

図 2.2: 索引語

$$D = \begin{bmatrix} \vec{d}_1 \\ \vec{d}_2 \\ \vdots \\ \vec{d}_5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (2.5)$$

検索質問として図 2.3 が与えられたとき、検索質問に含まれる索引語 w_i の出現回数を q_i とすると、そのベクトル $\vec{q}$ は式 2.6 となる。

Q : “USB keyboard driver”

図 2.3: 検索質問

$$\vec{q} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad (2.6)$$

与えられた検索質問と類似する文書を探し出すために、式 2.4 によって検索質問のベクトル $\vec{q}$ と各文書のベクトル $\vec{d}_i$ の間の類似度を求めると図 2.4 のようになる。この計算結果を表 2.4 のように類似度の降順に並び替えると、検索質問 Q に対して各文書は D_3, D_4, D_1, D_5, D_2 の順に類似していることがわかる。

$$\begin{aligned} D_1 : \quad \text{sim}(\vec{d}_1, \vec{q}) &= 0.408 \\ D_2 : \quad \text{sim}(\vec{d}_2, \vec{q}) &= 0 \\ D_3 : \quad \text{sim}(\vec{d}_3, \vec{q}) &= 0.873 \\ D_4 : \quad \text{sim}(\vec{d}_4, \vec{q}) &= 0.667 \\ D_5 : \quad \text{sim}(\vec{d}_5, \vec{q}) &= 0.258 \end{aligned}$$

図 2.4: 各文書の類似度

順位	文書	類似度
1	D_3	0.873
2	D_4	0.667
3	D_1	0.408
4	D_5	0.258
5	D_2	0

表 2.1: 類似度の順位

2.2 索引語の抽出

情報検索では、検索質問から目的の文書を探し出すために、検索質問と検索対象の文書群に含まれる単語による比較がよく行われる。だが、文書群に含まれる単語すべてを対象に比較を行うと、比較回数が増大し応答が遅くなる問題が起こる。

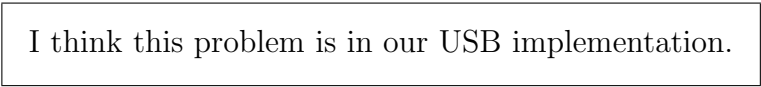
この問題を解決するために、文書中に含まれる“文書の特徴づけない単語”をあらかじめ除去する処理がおこなわれる。この“文書の特徴付けない単語”を不要語という。たとえば、日本語の場合は助詞や助動詞など、英語の場合は冠詞や前置詞などは文書の特徴づけない単語である。不要語を除去して残った単語を索引語という。

索引語を抽出するためには、あらかじめ文書から単語を抜き出さなければならないが、これは対象となる文書がどのような言語で書かれているかにより方法が異なる。英語のように単語と単語の間に空白をいれる言語は、容易に単語を抜き出せるが、日本語のように空白を入れる習慣がない言語は、単語を抜き出すことが難しい¹。本研究は、英語で書かれたメールを検索対象にするので、空白を元に単語を抜き出している。

索引語の抽出では、一般的に事前に決めた不要語のリストをもとに機械的に除去する方法が用いられる。理由は、手作業で索引語を抽出すると作業コストが高く、抽出される単語に偏りが生じる可能性があるからだ。本研究では、SMART システムで採用された 571 語の不要語 [3](図 2.6) を用いて除去を行っている。

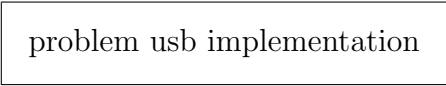
また、英語は単語の語形変化により性、数、格などの文法をあらわす。そのため、単語の語形変化を考慮しないと、本来は同じ意味の単語がそれぞれ別の単語として扱われてしまう問題がある。この問題は、接尾辞と呼ばれる語形変化の部分を除去することで対処できる。本研究では、ポーター・アルゴリズム (Porter algorithm)[4] を用いて接尾辞の除去を行っている。

不要語除去と接尾辞除去の例を以下に示す。図 2.5 が元の文書である。これの不要語を除去すると図 2.6 のようになり、“I”, “think”, “this”, “is”, “in”, “our” といった単語が除去されている。さらに、接尾辞を除去すると図 2.7 のようになり、“implementation” の語形変化部分が削除され “implement” になっている。



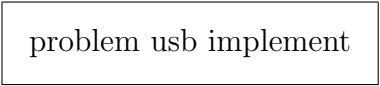
I think this problem is in our USB implementation.

図 2.5: 元の文章



problem usb implementation

図 2.6: 不要語を除去した後の文章



problem usb implement

図 2.7: 接尾辞を除去した後の文章

¹自然言語処理の分野で形態素解析という技術が研究されている。日本語を対象とした形態素解析プログラムのひとつに「茶筌」がある。

corresponding unfortunately consequently nevertheless particularly respectively accordingly appropriate considering furthermore
 afterwards appreciate associated beforehand concerning containing definitely especially everything everywhere particular presumably
 reasonably regardless relatively specifying themselves thereafter thoroughly throughout whereafter yourselves according available
 certainly currently described different downwards elsewhere everybody following greetings hereafter hopefully immediate
 indicated indicates meanwhile necessary obviously otherwise ourselves regarding seriously shouldn't something sometimes
 somewhere specified therefore thereupon whereupon actually although anything anywhere becoming consider contains couldn't entirely
 everyone followed formerly hereupon inasmuch indicate latterly moreover normally possible probably provides secondly sensible
 somebody sometime somewhat thorough together unlikely whatever whenever wherever wouldn't yourself against already amongst
 another anybody anyways awfully because becomes believe besides between certain changes clearly contain despite doesn't exactly
 example follows further getting happens haven't herself himself howbeit however ignored insofar instead looking neither nothing
 nowhere outside overall perhaps regards seeming serious several somehow someone specify there's thereby therein they'll they're
 they've through towards usually various welcome weren't where's whereas whereby wherein whether whither whoever willing without
 across allows almost always anyhow anyone anyway appear aren't around asking became become before behind beside better beyond
 cannot causes course didn't during either enough except former gotten hadn't hardly hasn't having here's hereby herein hither
 indeed inward itself lately latter likely little mainly merely mostly myself namely nearly nobody others placed please rather
 really saying second seeing seemed selves should thanks that's theirs thence theres they'd though toward trying unless useful
 wasn't what's whence within wonder you'll you're you've about above after again ain't allow alone along among apart aside
 being below brief c'mon can't cause comes could doing don't eight every fifth first forth given gives going hello hence
 inner isn't it'll keeps knows known later least let's liked looks maybe might needs never noone novel often other ought
 quite right seems seven shall since sorry still taken tends thank thanx thats their there these think third those three
 tried tries truly twice under until using value wants we'll we're we've where which while who's whole whose won't would
 would you'd yours able also away been best both came cant come does done down each else even ever five four from gets goes
 gone have he's help here hers i'll i've into it'd it's just keep kept know last less lest like look many mean more most
 much must name near need next nine none okay once ones only onto ours over plus said same says seem seen self sent some
 soon such sure take tell than that them then they this thru thus took unto upon used uses uucp very want we'd well went
 were what when whom will wish with your zero a's all and any are ask but c's can com did edu etc far few for get got had
 has her him his how i'd i'm inc its let ltd may new non nor not now off old one our out own per que saw say see she six
 sub sup t's the too try two use via viz was way who why yes yet you am an as at be by co do eg et ex go he hi ie if in is
 it me my nd no of oh ok on or qv rd re so th to un up us vs we a b c d e f g h i j k l m n o p q r s t u v w x y z

図 2.8: 本研究で用いた 571 語の不要語

2.3 検索処理の流れ

本研究では、検索質問としてコミットログを、検索対象として開発者向けメーリングリストのメールを用いて、図 2.9 の流れにより検索処理を行う。

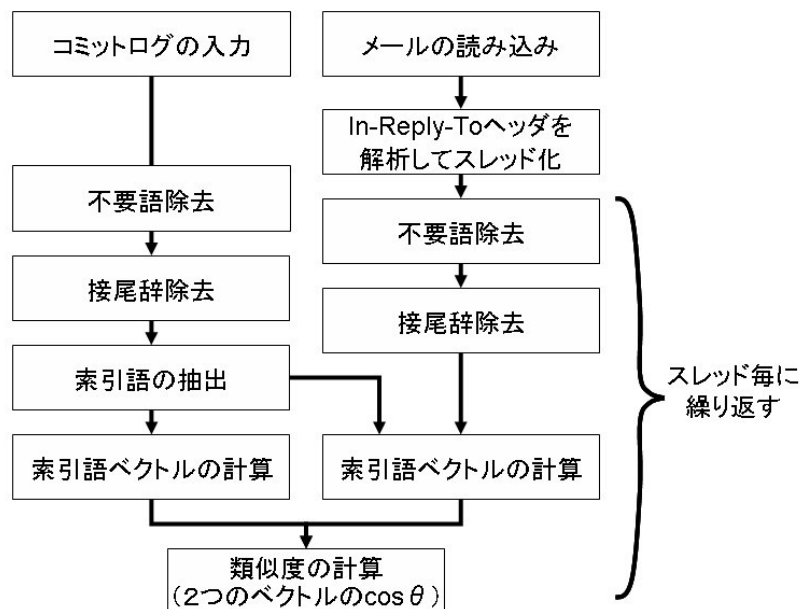


図 2.9: 検索処理の流れ

はじめに、検索対象になるメールスレッド群に対して前処理を行う。まず、開発者向けメーリングリストのメールを読み込み、In-Reply-To ヘッダを解析してスレッドにする。検索対象をメールスレッド単位にするために、ひとつのメールスレッドに含まれる本文をすべて結合する。最後に各メールスレッドの本文から不要語と接尾辞を除去する。以上で前処理は終わる。

次に、検索質問になるコミットログに対して処理を行う。まず、利用者によって入力されたコミットログから不要語と接尾辞を除去する。ここで残った単語を索引語にする。この索引語を用いて、コミットログの索引語ベクトルと各メールスレッドの索引語ベクトルを求める。

最後に、コミットログの索引語ベクトルと各メールスレッドの索引語ベクトルに対し式 2.4 を適用すると、ベクトル間の類似度が求められる。この類似度の降順に各メールスレッドを並び替えることにより、コミットログの内容に類似するメールスレッドを得ることができる。

2.4 ベクトル空間モデルの評価

前節で説明した検索モデルを、実際の開発プロジェクトに適用する評価実験を行った。

2.4.1 実験の準備

実験対象として、2003 年 12 月から 2004 年 11 月までの 1 年間に FreeBSD CURRENT のメーリングリストへ投稿されたメール (メール総数は 28273 通、スレッド数は 8480 スレッド) を用いた。

この実験対象から、事前に手作業によって“任意のコミットログに対応するメールスレッド”の組み合わせを探し出し、これを“正しい組み合わせ”と決める。検索モデルは、この正しい組み合わせが検索結果の上から何番目にあるかで評価する。正しい組み合わせは、以下の手順で探し出す。

1. 実験対象から無作為にコミットログを選ぶ
2. コミットログに対応するメールスレッドを実験対象のメーリングリストの中から探し出す
3. コミットログに対応するメールスレッドが見つからない場合は 1. に戻り、見つかるまで繰り返す

2.4.2 実験結果の評価

実験結果の評価は以下の手順で行った。

1. コミットログと各メールスレッドの類似度を計算する
2. メールスレッドを類似度順に並べ替える
3. 正しい組み合わせが上から何位にあるかを調べ、1 位に近いほど良い評価とする

実験の結果、順位は表 2.2 のようになった。正しい組み合わせを C1, C2 ... C12 であらわす。

この表の索引語数は、各コミットログに含まれる索引語数である。順位は、正解の組み合わせと同じメールスレッドが何番目にあるかをあらわす。類似度は各コミットログに対応するメールスレッドとの類似度である。

表の順位を見ると、C4(1 位)、C8(14 位)、C9(1 位) のように順位が高いものと、C6(2400 位)、C10(1916 位)、C12(1285 位) のように順位が低いものがあり、順位にばらつきがあることがわかる。そこで、順位が高いものと低いものについて、コミットログとメールスレッドの内容について具体例を挙げ、順位にばらつきが起こった原因を考察する。

正しい組合せ の ID	索引語数	順位	類似度
C1	69	71	0.59
C2	35	845	0.52
C3	26	83	0.57
C4	40	1	0.84
C5	56	105	0.83
C6	20	2400	0.29
C7	126	252	0.59
C8	51	14	0.57
C9	31	1	0.86
C10	62	1916	0.29
C11	40	389	0.73
C12	67	1285	0.54

表 2.2: 正解の組み合わせと同じメーリスレッドの順位

まず、この表で最も順位が高い C9 について、コミットログに対応するメーリスレッドの内容を示し、類似度が高いことを述べる。

C9 のコミットログでは USB に関する修正を行っており、メーリスレッドでは USB ドライバがロードできないという問題の投稿から解決に至るまでの討議が行われている。コミットログに含まれる索引語を見ると、図 2.10 に示すとおり `usb` や `hub`, `ehci` といった USB ドライバに関係する単語が数多く出現している。メーリスレッドに含まれる索引語では、図 2.11 に示すとおり `usb` や `hub`, `ehci` といった単語が多く出現していることがわかる。このように、同じ単語が数多く出現しているため類似度が高くなったといえる。

反対に、最も順位が低い C6 について、コミットログとメーリスレッドに含まれる索引語を示し、類似度が低いことを述べる。C6 のコミットログでは、BDF(バイナリファイル記述子) ライブラリに関する修正を行っており、メーリスレッドでは GDB が動かなくなったという質問に対して、参照すべきコミットが示され解決している。図 2.12 のコミットログに含まれる索引語と、図 2.13 のメーリスレッドに含まれる索引語を比較すると、明らかに索引語の数が少ないことから類似度が低くなる要因と考えられる。

この実験では、先に述べたように 2003 年 12 月から 2004 年 11 月までの 1 年間に FreeBSD CURRENT のメーリングリストへ投稿されたメール (メール総数は 28273 通、スレッド数は 8480 スレッド) を対象としているため、本来対応するメーリスレッドが埋もれる問題がある。この問題の原因を以下に説明する。

図 2.14 は、任意のコミットログと 8480 のメーリスレッド群との類似度を計算したときの、上位 500 位までの類似度の変化を示す。C1 から C12 まで、いずれも似たような曲線を描いている。問題を簡略化するため、この 12 本のグラフの平均を取ったものを図 2.15

```
usb hub ehci src dev sy ehcivar implement connect support log  
pipe file modifi peripher optim attach correct path revis  
interrupt basic stack devic freebsd split iedows transact utc  
work repositori
```

図 2.10: C9 のコミットログに含まれる索引語

```
usb hub ehci dev implement connect support pipe attach correct  
interrupt stack devic freebsd split iedows transact work
```

図 2.11: C9 のメールスレッドに含まれる索引語

```
bfd libbfd bin usr binutil gnu src powerpc  
log obrien file enabl modifi freebsd configur  
repositori path revis bit pst
```

図 2.12: C4 のコミットログに含まれる索引語

```
usr gnu src file freebsd configur revis
```

図 2.13: C4 のメールスレッドに含まれる索引語

に示す。この図では1位から50位の間の類似度の差は約0.1であるが、50位から類似度が0.1下がると順位は300位くらいまで下がってしまう。この原因として、検索対象の

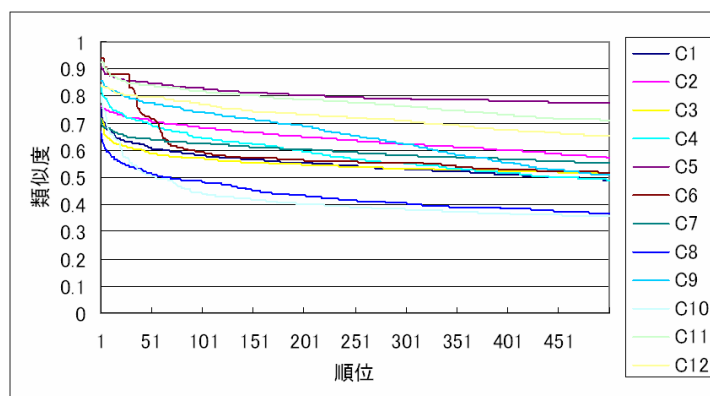


図 2.14: 順位と類似度の関係

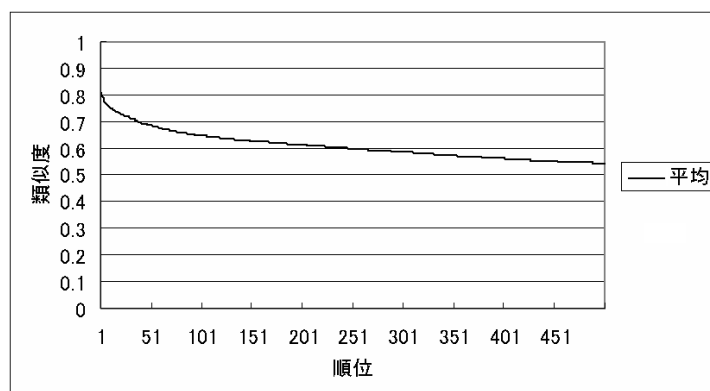


図 2.15: 順位と類似度の関係を平均化

数が多いためコミットログに関係がない同じような文脈のメールスレッドが存在することがあげられる。

以上の結果から、ベクトル空間モデルはコミットログに対応するメールスレッドを探す方法としてある程度は有効といえる。しかし、順位の低いものは、関係のないメールスレッドに埋もれることにより順位が下がっていると考えられ、それらの順位を上げる対策が必要である。

第3章 ヒューリスティックの適用

本章では、ヒューリスティックを用いる理由やその定義、ヒューリスティックを適用した実験結果の評価について述べる。

3.1 ヒューリスティック

すべてのメールスレッドを検索対象に類似度計算を適用すると、関係のないメールスレッドが大量にヒットしてしまい、本来対応するメールスレッドが埋もれる問題がある。原因は、コミットログに関係がない同じような文脈のメールスレッドが存在するからである。

人間が手作業で探し出すときは、すべてのメールを逐一読むわけではない。経験的に、あるコミットに対応する討議が行われたメールスレッドは、コミットされた日付に近いはずだと考え、例えばコミットの日付と同じ月に範囲を絞って探す。もし、見つからなければさらに範囲を広げて、これを見つけるまで繰り返す。

そこで、コンピュータによる検索にも、人間が手作業で探し出す時の手がかりを用いて検索対象を絞り込むことが有効と考えた。本研究では、コンピュータに与える手作業で探す時の手がかりをヒューリスティックと定義する。ヒューリスティックを検索対象を絞り込むことにより、コミットログに関係がない同じような文脈のメールスレッドを省くことができ、順位が上がることを期待できる。

3.1.1 手作業による探索過程

2.4 節において、ベクトル空間モデルの評価を行う際に用いた 12 組のサンプルは手作業で探し出したと述べた。この手作業で探し出す過程では、次の 3 つの手がかりを用いた。

1. 重要な人物が投稿しているメールスレッド
2. コミットの日付、コミットした人の名前
3. 内容を特徴付ける単語

ここでは具体例として、ベクトル空間モデルの評価で順位と類似度が最も高かった C9 を探し出す過程を説明する。

cvs commit: src/sys/dev/usb ehci.c ehciivar.h

コミットした人の名前

Ian Dowse iedowse@FreeBSD.org

Sun Aug 1 11:47:42 PDT 2004

- Previous message: [cvs commit: src/sys/dev/null null.c](#)
- Next message: [cvs commit: src/sys/alpha/include memdev.h src/sys/ia64/include memdev.h](#)
- Messages sorted by: [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#)

コミットした日付

iedowse 2004-08-01 18:47:42 UTC

FreeBSD src repository

Modified files:

sys/dev/usb ehci.c ehciivar.h

Log:

Implement basic support for **EHCI** interrupt pipes. This is unlikely to be particularly correct or optimal, but it seems to be enough to allow the attachment of USB2 hubs and USB2 devices connected via USB2 hubs. None of the split transaction support is implemented in our **USB** stack, so USB1 peripherals will definitely not work when connected via USB2 hubs.

特徴づける単語

特徴づける単語

Revision	Changes	Path	ファイルパスとリビジョン番号
1.12	+266 -11	src/sys/dev/usb/ehci.c	
1.2	+17 -0	src/sys/dev/usb/ehciivar.h	

図 3.1: C9 のコミットログ

Unloading **USB** driver while device is attached.

コミットした人と一致 コミットログの単語と一致

Ian Dowse iedowse@maths.tcd.ie

Mon Jul 19 16:45:11 PDT 2004

- Previous message: [Unloading USB driver while device is attached](#)
- Next message: [Unloading USB driver while device is attached](#)
- Messages sorted by: [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#)

In message <20040719.170132.52458790.imp_at_bsdimp.com>, "M. Warner Losh" writes:

>In message: <200407191950.aa12733_at_salmon.maths.tcd.ie>

> Ian Dowse <iedowse@maths.tcd.ie> writes:

>: <http://people.freebsd.org/~iedowse/usb.diff>

>:

>: but unfortunately I haven't had time to work on this lately (and
>: there are all the problems of divergence from NetBSD etc. if bits
>: of this get committed). The (small) uhub.c changes look like they
>: might possibly help in your case. With the full patch I was able
>: to 'kidunload usb' without crashes a while ago, but there were still
>: memory leaks.

>

>I'm going to start merging in some of these changes, if you don't
>mind.

If you have the time that would be great! FYI, here is a rough
summary of the changes in no particular order:

コミットログの単語と一致

- o Beginnings of interrupt pipe support for **EHCI** (very little done)
- o Support for unloading the usb driver (leaks some memory)
- o Support for removing cardbus USB controllers (also leaks memory)
- o Avoid most (but not all) uses of contigmalloc and data buffer

図 3.2: コミットログを基に最初にたどったメール

- スレッドの始まり
- Unloading USB driver while device is attached. Pawel Jakub Dawidek
 - Unloading USB driver while device is attached. Doug White
 - Unloading USB driver while device is attached. Pawel Jakub Dawidek
 - Unloading USB driver while device is attached. Scott Long
 - Unloading USB driver while device is attached. Pawel Jakub Dawidek
 - Unloading USB driver while device is attached. Ian Dowse
 - Unloading USB driver while device is attached. M. Warner Losh
 - Unloading USB driver while device is attached. Scott Long
 - Unloading USB driver while device is attached. M. Warner Losh
 - Unloading USB driver while device is attached. Ian Dowse
 - Unloading USB driver while device is attached. M. Warner Losh
 - Unloading USB driver while device is attached. Ian Dowse
 - Unloading USB driver while device is attached. Craig Rodrigues
 - Unloading USB driver while device is attached. Ian Dowse
 - Unloading USB driver while device is attached. M. Warner Losh
 - Unloading USB driver while device is attached. M. Warner Losh
- 最初にたどったメール
- スレッドの終わり
- Unloading USB driver while device is attached. M. Warner Losh

図 3.3: C9 のメールスレッド

図 3.1 に示す C9 のコミットログについて、手がかりとしてコミットした人の名前、コミットした日付、内容を特徴付ける単語 (EHCI, USB)、ファイルパスとリビジョン番号に注目した。このコミットログを基に、コミットした日付に近いメール群の中から最初にたどったメールが図 3.2 である。これをみるとコミットした人の名前 (Ian Dowse) や内容を特徴付ける単語 (EHCI, USB) が出現していることがわかる。このメールのスレッドは図 3.3 のようになっている。このスレッドの一連の討議を読むと C9 のコミットログに示される変更箇所について、変更に至るまでの討議であることが確認できる。

以上のように、人間は手がかりをもとに検索範囲を絞り込んで探し出している。

3.2 ヒューリスティックの評価

ヒューリスティックの有効性を確かめるための実験と結果について述べる。

3.2.1 実験の準備

検索対象の絞込みに用いるヒューリスティックを以下に示す。

1. コミットした人の名前

- コミットした人が投稿しているメールスレッドのみを残す

2. コミットの日付

- コミットの日付から前後 n 日以内に投稿されたメールスレッドのみを残す
- 実験では n を 31, 15, 7 に変えて試行する (数値の根拠は 3.2.2 節にて説明)

“コミットした人の名前”をヒューリスティックに用いる根拠は、コミットに至るまでの討議がメールスレッドに残っているならば、コミットした人もそのメールスレッドに投稿していると考えからである。

“コミットの日付”をヒューリスティックに用いる根拠は、コミットに至るまでの討議が行われたメールスレッドへの最後の投稿と、コミットの日付は近いと考えられるからである。

3.2.2 コミットの日付とメールスレッドの関係

“コミットの日付から前後 n 日以内に投稿されたメールスレッドのみを残す”というヒューリスティックにおいて、 n を 31, 15, 7 に変えて試行する根拠を説明する。

C1 から C12 までの 12 組のサンプルについて、コミットの日付から前後何日以内にメールスレッドが存在するのか調べた結果を図 3.4 に示す。

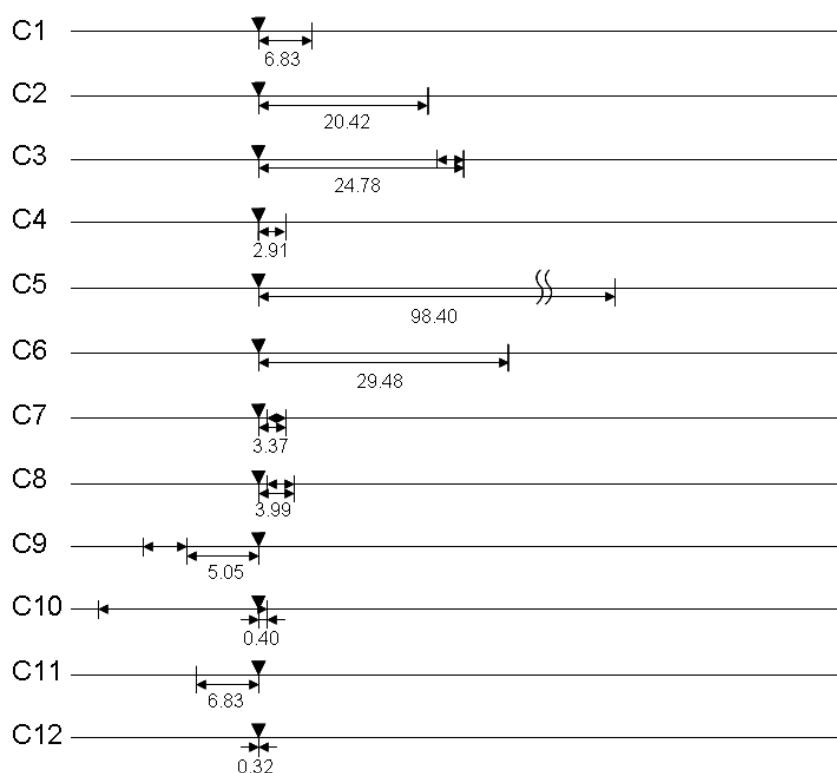


図 3.4: コミットの日付とメールスレッドの関係

図 3.4 について述べる前に、図の見方を図 3.5 の凡例によって説明する。この凡例は、コミットの日付とメールの始まりと終わりの相対関係をあらわしている。横軸は時間軸で、時間は右方向へ進む。コミットの日付は、▼（黒い下向きの三角）であらわす。|←→|（2

本の縦線の間)に双方向の矢印)は、コミットの日付とメーリスレッドの最後の投稿の日付をの期間をあらわす。また、矢印の下の数値は期間の長さ(単位は日)をあらわし、この例の 6.83 は“6 日と 19 時間 55 分”を意味する。

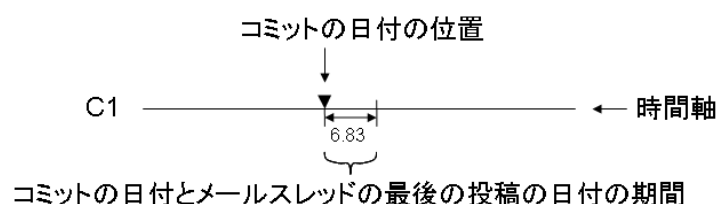


図 3.5: コミットの日付とメーリスレッドの関係を示す図の凡例

以上を踏まえて、図 3.4 を見ると、コミットの日付とメーリスレッドに最後に投稿されたメールの日付の期間(| $\longleftrightarrow$ |の下の数値)が C5 以外はすべて前後 31 日以内であることがわかる。C5 はコミットから約 98 日後にメールが投稿されているが、メーリスレッドを読むと「すでに直っているバグについて質問が投稿され、過去のコミットを参照するように促している」という内容であった。よって、このような投稿は滅多に起こらない例外とみなし、31 という数値に決めた。

なお、15, 7 という数値は特に根拠はないが、31 の半分と、さらにその半分に期間を減らしたときに、どのような結果になるか比較するために決めた。

3.2.3 実験結果の評価

実験結果の評価は以下の手順で行った。

1. ヒューリスティックを適用して検索対象のメーリスレッドを絞り込む
2. コミットログと各メーリスレッドの類似度を計算する
3. メーリスレッドを類似度順に並べ替える
4. 正しい組み合わせが上から何位にあるかを調べ、1 位に近いほど良い評価とする

表 3.1 はヒューリスティックの評価実験を行った結果である。“正しい組み合わせ”の列は、事前に手作業によって探し出したコミットログに対応するメーリスレッドの組み合わせの ID で、C1, C2 ... C12 であらわす。“ヒューリスティック不適用”の列は、ヒューリスティックを適用せず一年間すべてのメーリスレッドを対象にしたときの順位と、そのときの検索対象のスレッド数をあらわす。“コミットした人の名前”の列は、コミットした人が投稿しているメーリスレッドのみを残すというヒューリスティックを適用したときの順位と、そのときの検索対象のスレッド数をあらわす。“コミットの日付の列は、コミットの日付から前後 n 日 (n は 31, 15, 7 に変えて試行) 以内に投稿されたメーリスレッドを残す

というヒューリスティックを適用したときの順位と、そのときの検索対象のスレッド数をあわす。

正しい組み合わせ の ID	ヒューリスティック		コミットした		コミットの日付					
	不適用		人の名前		前後 31 日		前後 15 日		前後 7 日	
	順位	検索対象の スレッド数	順位	検索対象の スレッド数	順位	検索対象の スレッド数	順位	検索対象の スレッド数	順位	検索対象の スレッド数
C1	71	8480	0	429	15	1486	10	807	0	209
C2	845	8480	0	141	190	1543	0	770	0	388
C3	83	8480	0	88	16	1520	0	739	0	353
C4	1	8480	0	29	1	1052	1	525	1	227
C5	105	8480	0	379	0	1405	0	585	0	326
C6	2400	8480	0	141	440	1343	0	638	0	298
C7	252	8480	49	365	37	1406	19	760	11	354
C8	14	8480	4	53	2	1505	1	713	1	310
C9	1	8480	1	18	1	1596	1	706	0	321
C10	1916	8480	113	168	375	1731	204	882	93	419
C11	389	8480	0	189	73	1785	41	905	21	429
C12	1285	8480	11	53	200	1743	87	878	36	416

表 3.1: ヒューリスティック不適用と適用の場合の、順位と検索対象のスレッド数

まず、ヒューリスティックを適用したときに検索対象を絞り込まれているか確認する。

表の左から三列目を見ると、ヒューリスティックを適用しない場合は、一年間に投稿されたメールスレッドすべて (8480 スレッド) が検索対象となっていることがわかる。

これに対し、コミットした人の名前による絞り込みを適用した場合は、検索対象のスレッド数が減っている。また、コミットした日付による絞り込みを適用した場合も、前後 31 日、前後 15 日、前後 7 日のいずれも、検索対象のスレッド数が減っている。

次に、絞り込んだ検索対象に対しベクトル空間モデルを適用したとき、順位がどれだけ上がっているか確認する。

コミットした人の名前で絞り込むヒューリスティックを適用した場合の順位を見ると 12 のうち 7 つが 0 になっている。これは、手作業で探した正しいメールスレッドにコミットした人が参加していないことを意味し、検索結果として悪い。しかし、コミットした人が参加している場合はいずれも順位が上がっている。特に、C10 は 1916 位から 113 位に、C12 は 1285 位から 11 位に順位が上がっていることから、このヒューリスティックはある程度有効と考える。

コミットの日付で絞り込むヒューリスティックを適用した場合も、前後 31 日、前後 15 日、前後 7 日のいずれも順位が上がっていることが確認できる。特に、C7 は 252 位から、前後 31 日で絞り込んだ場合は 37 位、前後 15 日だと 19 位、前後 7 日だと 11 位と確実に

上がっている。また、C8 も 14 位から、前後 31 日で絞り込んだ場合は 2 位、前後 15 日だと 1 位、前後 7 日でも 1 位になっており、ヒューリスティックが有効に働いていると確認できる。なお、前後 n 日の n が、コミットの日付とメールスレッドに最後に投稿されたメールの日付の期間より短い場合は見つからなくなるという問題がある。

第4章 システムの実装

本研究では、コミットログに対応するメールスレッドを自動抽出するシステムを実現した。システムは、CVSのコミットログを入力とし、メーリングリスト・アーカイブの中から対応するメールスレッドを抽出する。本章では、システム的使用方法について具体例を示して説明する。

4.1 システムの構成

これまでに述べた手法に基づく、コミットログに対応するメールスレッドを自動抽出するシステムの実装について述べる。システムの構成を4.1に示す。

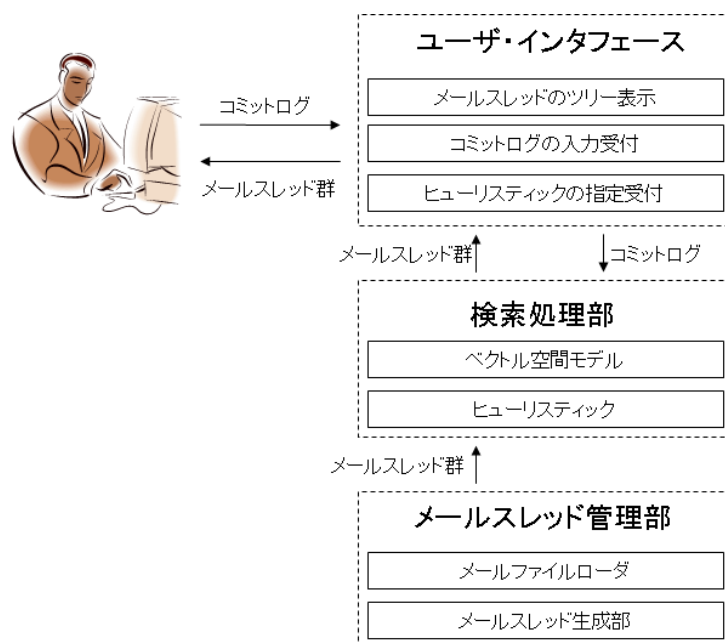


図 4.1: システムの構成

開発者はユーザ・インタフェースでヒューリスティックとコミットログを指定する。検索処理部は、ユーザ・インタフェースから受け取ったヒューリスティックで検索対象の絞込み

を行い、コミットログとメールスレッド群との類似度を計算した結果を表示する。メールスレッド管理部は、メーリングリスト・アーカイブの mbox ファイルを読み込んでスレッド化したメールスレッド群を保持する。

4.2 システムの使用方法

このシステムで実際にコミットログに対応するメールスレッドを抽出する手順について述べる。

起動すると図 4.2 の画面になる。まず、メーリングリスト・アーカイブの mbox ファイルをプログラムに読み込むために、図 4.3 のようにメニューを開いて “Open” を選択する。すると、図 4.4 のようにファイル・オープン・ダイアログが開くので、読み込みたい mbox ファイルを選択する。この例では、2003 年 11 月から 2004 年 10 月まで 1 年間にメーリングリストに流れたメールの mbox ファイルを選択している。

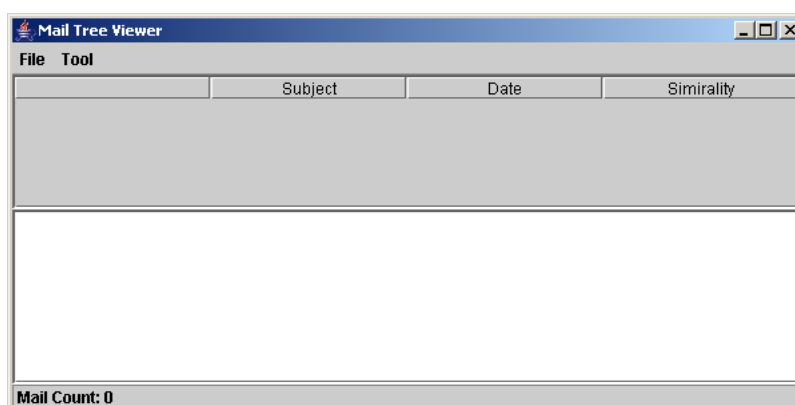


図 4.2: 起動直後の画面

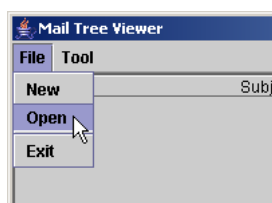


図 4.3: ファイル・メニュー

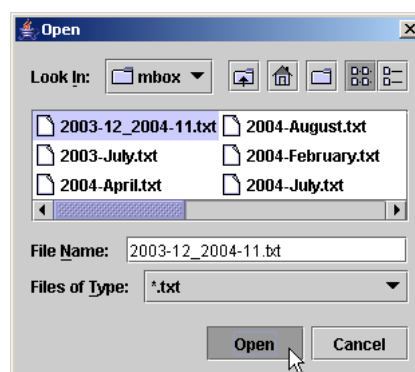


図 4.4: mbox ファイルを開く

mbox ファイルが読み込まれると、図 4.5 のようにスレッド化されたメールスレッド群

が表示される。このメールスレッド群に含まれるメールの数はウィンドウの左下に “Mail Count: 28273” のように表示される。

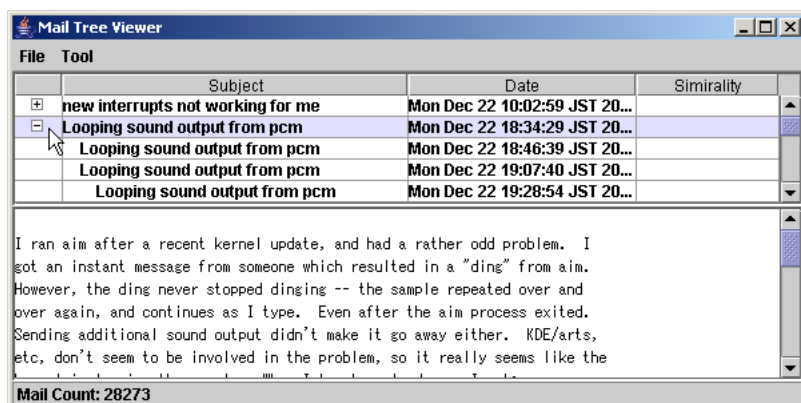


図 4.5: mbox ファイルを読み込み、メールスレッド化した画面

いま読み込んだメールは 1 年間分であり、28273 通という膨大な数のメールが検索対象になっている。そこで、コミットの日付から前後 31 日に最後の投稿がされたメールスレッド群だけを残すヒューリスティックを用いて検索対象を絞り込む。図 4.6 のように “Heuristics2” を選ぶと、図 4.7 のダイアログが表示される。

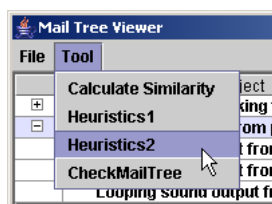


図 4.6: コミットの日付によるヒューリスティックの選択

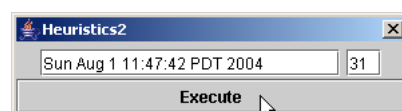


図 4.7: コミットの日付と前後 n 日の n を入力

この例では、コミットの日付に “Sun Aug 1 11:47:42 PDT 2004” を指定し、隣の欄は前後 31 日で絞りこむため “31” を指定した。“Execute” ボタンを押すと、図 4.8 のように検索対象が絞り込まれたメールスレッド群が表示される。図 4.8 のウィンドウの左下を見ると “Mail Count: 5662” となっており、28273 通から 5662 通に検索対象が絞り込まれていることが確認できる。

次に、ベクトル空間モデルによってコミットログとメールスレッド群の間の類似度計算を行う。図 4.9 のように、メニューから “Calculate Similarity” を選択すると、図 4.10 のようなダイアログがあらわれる。このダイアログの上側のテキストフィールドにコミットログを指定し、“Calc Similarity” ボタンを押すと、図 4.11 のようにコミットログとメールスレッド群との類似度計算が開始される。なお、途中経過はプログレスバーによってパーセント表示される。図 4.12 のようにプログレスバーが 100% になったら類似度計算は終了で

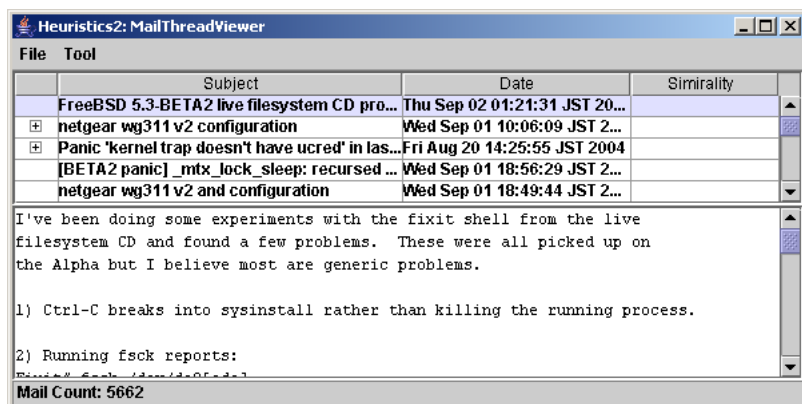


図 4.8: ヒューリスティック適用後の画面

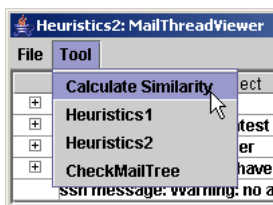


図 4.9: “類似度の計算”を選択

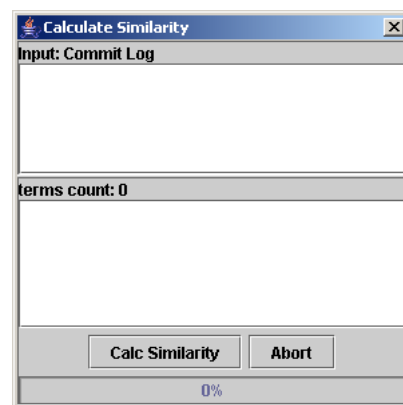


図 4.10: 類似度計算ダイアログ

ある。なお、下側のテキストフィールドにはコミットログから抽出された索引語が表示される。ウィンドウの中央の行に“terms count: 31”とあるが、これは抽出された索引語数をあらわす。ここで、メールスレッドのビューアに視点を戻すと、図 4.13 のように、メールスレッド群が類似度順に並び替えられている。あとは、この結果を上から順に調べることで、目的のメールスレッドを探すことができる。

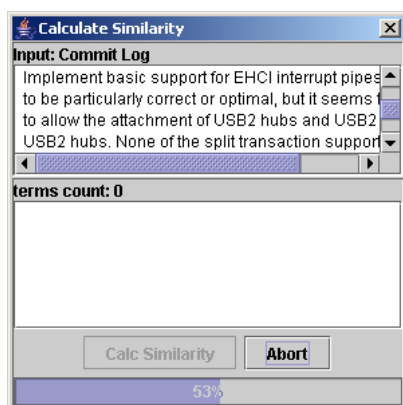


図 4.11: 類似度計算の途中経過

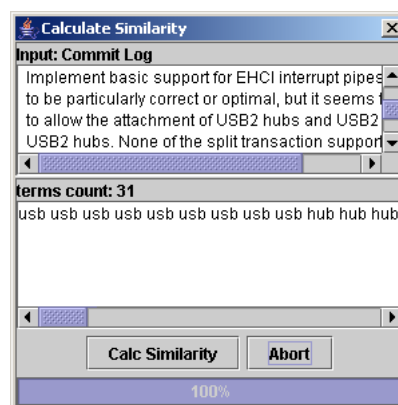


図 4.12: 類似度計算の終了

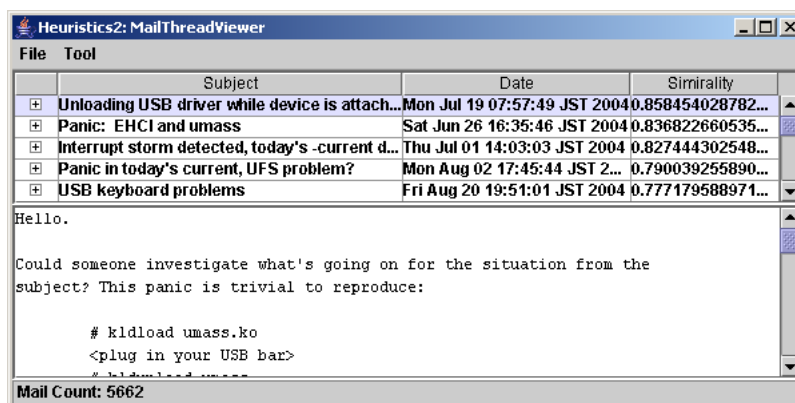


図 4.13: 類似度の降順に並び替えられたメールスレッド群

第5章 関連研究

5.1 Design Rationale

Design Rationale は、大規模で複雑なシステムを設計するためのドキュメンテーションの方法論であり、1980 年代から 1990 年代にかけて DR をソフトウェア工学に応用する研究が行われた。Design Rationale は以下に示す二つの仮定に基づいた方法論である。

- 設計を対象とした討議の内容をある規則に基づいて整理し、設計理由の理解を支援する
- 設計理由の把握を支援することは設計の効率を向上させることになる

Design Rationale に関する研究として、Colin Potts らや、Jeff Conklin らの研究がある。

Colin Potts らは、設計を決定するまでに至る討議の記録方法について論じている [5]。論文の中では、ハイパーテキストを用いて成果物の設計過程を視覚的に表現することにより、どういう経緯で成果物が作られたかを判断できると述べている。しかし、設計過程を意識的に記録する手間がかかるため現実的ではないことや、既存の開発事例を元に実験をしているので、実際に効果があるのか疑問である。

Jeff Conklin らは、gIBIS[6] という討議検索のためのハイパーテキストツールの有効性について論じている。しかし、討議の枝分かれが多数できて理解できなくなる、投稿を分類するとき判断が難しいなどの問題がある。

Design Rationale について総論すると、よく整理された設計根拠が得られるが、分類・整理・探索に手間がかかり現実的ではない。この問題については、Conklin 自らが gIBIS のアプローチは現実的でなかったと述べている [7]。

しかし、設計・変更した理由を容易に把握できれば、同じ失敗を繰り返すなどの無駄な作業が削減されたり、間違いを修正しやすくなるなど作業効率の向上が期待できるため、本研究では Design Rationale とは異なるアプローチでこれを実現する。

5.2 Hipikat

本研究とは目的は異なるが、Hipikat で用いられているベクトル空間モデルと呼ばれる情報検索モデルが、本研究に応用できると考えた。Hipikat は、オープンソース・ソフトウェア開発の支援ツールの実装である [8]。

Hipikat は次のような背景から開発された。開発プロジェクトは、CVS リポジトリ、バグデータベースおよびニュースグループのような膨大な量のアーカイブに保管された情報によって維持されている。新規参加者が開発プロジェクトに参加したくても、経験が浅いため何万通ものメッセージから適切なものを見つけるのは非常に困難であるという問題がある。

そこで、すでに開発された成果物の中から参加者が行おうとしている作業に適切な成果物を推奨するシステムが必要と考えた。暗黙グループメモリとして、生産された成果物（ソースのバージョン、バグ、メールアーカイブ、Web ドキュメント）を全て考慮する。ベクトル空間モデルで各々の成果物間の類似度を計算し、類似度が高い順にリストアップして提示する。

開発プロジェクトを構成するアーカイブの中から暗黙グループメモリを形成することにより、新規参加者が開発プロジェクトに貢献しようとしたときに、適切な成果物を提示できるようにする。このことにより、オープンソース・ソフトウェアの新規参加者がより早く生産的になると論文では述べられている。

5.3 オープンソース開発支援のためのソースコード及びメールの履歴対応表示システム

石川氏らの研究 [9] では、オープンソース・ソフトウェア開発を対象に、CVS リポジトリから効率的に情報を取得する手法や、CVS リポジトリとメーリングリストの関連付けの手法について述べられている。

CVS リポジトリから情報を取得するとき、ファイル単位での履歴は CVS コマンドの `rlog` を利用すればよい。しかし、ある開発者がこれまでに行った開発作業の履歴や、特定の日付に行われた更新作業の履歴を、容易に取得できないという問題がある。

また、開発プロジェクトのソフトウェア成果物の管理を効率的におこなうために、CVS やメーリングリストが用いられるが、これらは互いに独立に利用されている。開発者は、CVS リポジトリで管理されるソースコードの修正から、その修正に至るまでにメーリングリストで行われた討議を取得したい場合がある。そのとき、開発者は CVS リポジトリとメールスレッドの両方から個々に情報を取得して、それらを関連付ける必要があるが、その関連付けのコストが非常に高いという問題がある。

前者の CVS リポジトリから効率的に情報を取得する手法の評価では、従来は直接取得できなかった情報が取得できている。

後者の CVS リポジトリとメーリングリストを関連付ける手法の評価では、抽出精度は低く有用な情報が得られていない。その理由は、CVS リポジトリとメーリングリストの関連付けに用いているキーワードの取得方法に問題があるためと論文では述べられている。

第6章 おわりに

6.1 まとめ

本研究では、ベクトル空間モデルを利用し、CVS のコミットログから関連するメールスレッドを抽出し提示するシステムの開発を行なった。

すべてのメールを対象に類似度を計算すると、1 位や 14 位のように順位が高いものと、1916 位や 2400 位のように順位が低いものがあり、順位に大きな差が見られた。原因は、検索対象の数が多いためコミットログに関係がない同じような文脈のメールスレッドに本来対応するメールスレッドが埋もれるためである。そこで、検索対象をヒューリスティックによって絞り込むという手法を考えた。ヒューリスティックで検索対象を絞り込んだ後、メールスレッド群との類似度を計算することにより順位が上がると期待できる。ヒューリスティックを適用した場合の実験結果を以下に述べる。

コミットした人が投稿しているメールスレッドを対象にするというヒューリスティックを適用した場合は、12 のうち 7 は対応するメールスレッドが見つからなかった。理由として、サンプルのメールスレッドのほかにも関連するメールスレッドが存在するか、本当に存在しないことが考えられる。しかし、コミットした人が投稿しているメールスレッドでは、いずれも順位が上がっており、特に 1916 位から 113 位、1285 位から 11 位と大きく順位が上がっている例もある。

コミットされた日付の前後 n 日に最後の投稿がされたメールスレッドを対象にするというヒューリスティックを適用した場合は、前後 31 日、前後 15 日、前後 7 日のいずれも順位が上がっていることが確認できた。ただし、前後 n 日の n が、コミットの日付とメールスレッドに最後に投稿されたメールの日付の期間より短い場合は見つからなくなるという問題がある。

これらの実験結果から、ヒューリスティックで検索対象を絞り込み、ベクトル空間モデルで類似度を計算するという方法はある程度有効といえる。

6.2 今後の課題

今後の課題を以下に示す。

- ベクトル空間モデル

本研究で用いたベクトル空間モデルの欠点として、意味が類似する単語を判別でき

ないことがあげられる。このことが、オープンソース・ソフトウェア開発のメーリングリストに投稿されるメールを検索する上で問題になるか調査が必要である。

- ヒューリスティック

本研究ではふたつのヒューリスティックについて実験を行い、ヒューリスティックがある程度有効と確認できた。このふたつ以外にも、ヒューリスティックがあるか調査が必要である。また、実験ではふたつのヒューリスティックを独立に適用していたが、例えば日付で絞った結果をさらに名前で絞れば、より順位が上がることを期待できる。

- 実験対象を増やす

コミットした人の名前で絞るというヒューリスティックの実験の結果、見つからない場合が多々認められた。これは、あらかじめ手作業による関連付けで正解と仮定したメールスレッドとは別に正解のスレッドが存在する可能性が高い。よって、実験対象を増やして再評価が必要と考える。

謝辞

本研究を行なうにあたり、終始変わらぬ御指導を賜りました落水浩一郎教授に心から深く感謝申し上げます。

本研究を進めるにあたり、随所で貴重なご意見を賜りました藤枝和宏助手に深く感謝致します。

研究の節目節目において、適切な助言を下さいました鈴木正人助教授に深く感謝致します。

服部哲助手には多大な御助言をいただき深く感謝致します。

最後に、落水研究室の皆様の日頃の討論と助言、励ましに深く感謝致します。

参考文献

- [1] 長尾 真, 黒橋 禎夫, 佐藤 理史, 池原 悟, 中野 洋. 言語情報処理, 岩波書店, 1998.
- [2] 北 研二, 津田 和彦, 獅々堀 正幹. 情報検索アルゴリズム, 共立出版, 2002.
- [3] Salton, G. SMART system: stop word list.
<ftp://ftp.cs.cornell.edu/pub/smart/english.stop>
- [4] Fotis Lazarinis. Porter stemming algorithm.
http://www.dcs.gla.ac.uk/ir_resources/linguistic_utils/
- [5] Colin Potts and Glenn Bruns. Recording the Reasons for Design Decisions, IEEE Computer Society Press, 1988.
- [6] Jeff Conklin and Michael L. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion, ACM, 1988.
- [7] Jeff Conklin, Albert Selvin, Simon Buckingham Shum and Maarten Sierhuis. Facilitated Hypertext for Collective Sensemaking: 15 Years on from gIBIS, ACM, 2001.
- [8] Davor Cebraic and Gail C. Murphy. Hipikat: Recommending Pertinent Software Development Artifacts, IEEE Computer Society Press, 2003.
- [9] 石川 武志. オープンソース開発支援のためのソースコード及びメールの履歴対応表示システム, 大阪大学 修士学位論文, 2001.