修士学位論文

題目

LSA 手法を用いたソフトウェア変更情報のクラスタリング手法

指導教員
井上 克郎 教授

報告者
今枝 誉明

平成 19 年 2 月 13 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

平成 18 年度 修士学位論文

LSA 手法を用いたソフトウェア変更情報のクラスタリング手法

今枝 誉明

内容梗概

近年，複雑化しているソフトウェアを効率よく開発・管理するために，ソフトウェアに対して行われた変更を記録するシステムである，版管理システムが広く使われている．また，版管理システムに蓄積された過去の開発情報を用いることで，新たな開発や保守作業を効率的に行うことができると言われている．しかし，蓄積された膨大な情報の中から開発者が必要とする情報を的確に取得するのは容易ではない．版管理システムに蓄積された変更の記録は，互いに関連を持つことがある．そのため，開発者が過去に行われた変更を参考にする際には，1 つの変更だけでなく，関連する変更を同時に参照しなければならない．しかし，現在用いられている変更記録の閲覧システムは，個々の変更を表示するのみであるため，関連する変更は人手で探さなければならず，開発者の負担となっている．

本研究では，版管理システムに蓄積された変更間の関連を抽出することを目的として，変更をクラスタリングする手法を提案する．本手法は，変更箇所に出現する単語は，その変更を特徴付けているという仮定に基づいている．まず，ソフトウェアの変更記録から同時に行われた変更群を抽出する．次に，抽出した変更群から単語を取り出し，潜在的意味解析手法 LSA を用いて変更群の間の類似度を算出する．最後に，得られた類似度を基に変更群をクラスタリングすることで，関連する変更群を抽出する．また，提案手法を実現するシステムを試作し，実際のソフトウェアに対して適用した．その結果，関連する変更群が抽出でき，本手法の有効性を確認できた一方，本手法が単語に基づいているための問題点も確認できた．

主な用語

ソフトウェア理解 (Software Comprehension)
クラスタリング (Clustering)
版管理システム (Version Control System)

目 次

1　はじめに

近年のソフトウェアの開発規模の増大や，インターネットに代表されるネットワーク環境の発展に伴い，ソフトウェアの開発形態は多人数化，分散化している．また，ソフトウェアの大規模化や複雑化，その使用環境の変化等に伴い，ソフトウェア保守はますます困難になり，長期間に渡って使用されるソフトウェアでは，保守コストがソフトウェア全体のコストの大部分を占めると言われている [1].

複雑化しているソフトウェアを効率よく開発・管理するため，近年のソフトウェア開発では，版管理システム [2] に代表される開発支援システムが一般的に用いられている．版管理システムとは，ソフトウェアを構成するファイル (ソースコード，ドキュメント等) に対する変更を記録するシステムであり，開発者は版管理システムを用いることで，誤った変更を行っても容易に元に戻したり，他の開発者の行った変更を参照することが可能となる．

開発支援システムは，開発プロセスを円滑に進めるための基盤としての価値に加え，ソフトウェアの開発情報が蓄積されることで，他の開発に再利用されたり，参考にされたりするといった，ソフトウェア資産の貯蔵庫としての価値も有する．例えば，版管理システムに蓄積された過去の変更を理解することで，開発および保守作業を効率的に行うことができると言われている [3].

しかし, 蓄積された膨大な情報の中から，開発者が必要とする情報を的確に取得することは容易ではない．既存のシステムにより，過去の変更を個別に閲覧することは可能である．しかし，変更は互いに関連を持つことがあるため，開発者は 1 つの変更だけでなく，関連する変更を同時に参照しなければならない．例えば，ある関数定義の変更と対応する関数呼び出しの変更や，欠陥を埋め込んだ変更とその欠陥を修正した変更が挙げられる．しかし，既存の版管理システムでは，関連する変更を人手で探さなければならず，開発者の負担となっている．

そこで本研究では，版管理システムに蓄積されたファイルに対する変更の間の関連を抽出することを目指す．これにより，関連した変更を知ることができ，変更記録を参照する際の開発者の負担が軽減されることが期待できる．

そのために本稿では，版管理システムに蓄積された変更記録の間の類似度を計算し，クラスタリングする手法を提案する．本手法は，変更箇所に含まれる単語が，その変更を特徴付けているという仮定に基づいており，自然言語で記述された文書を分類するための手法である潜在的意味解析手法 LSA [4] を応用して，変更間の類似度を算出する．そして，算出した類似度を基に，変更をクラスタリングする．また，本手法は出現する単語のみを利用しており，適用対象のプログラミング言語に依存しないため，構文解析が不要であり，複数言語が混じっている場合や，ドキュメント等の，ソースコード以外のものが含まれている場合でも

適用可能であるという利点がある．

そして，提案手法を実現するシステムを試作し，本手法により関連のある変更の集合が抽出できるかを確認するために，実際のソフトウェアに対して適用する．

以降，2 節では，ソフトウェア開発で用いられる版管理システムについて説明し，版管理システムに蓄積された変更の記録を参考にする際の問題点を指摘する．3 節では，問題解決のために提案する，変更情報のクラスタリング手法について説明し，4 節では，手法の実現のために試作したシステムについて説明する．5 節で本手法の適用事例を紹介し，考察を与える．6 節で関連研究について述べ，最後に 7 節でまとめと今後の課題について述べる．

2　版管理システムとソフトウェア開発

本節では，版管理システムについて説明した後，そこに蓄積された変更の記録を利用する際の問題点を指摘する．

2.1　版管理システム

版管理システムとは，ファイルに対して行われた追加・削除・修正等の変更を記録するシステムである．更に，開発者間の変更の競合を検知して，並行開発をサポートするシステムも存在する．

版管理システムでは，ソースコードやドキュメントといった各ファイルの変更記録は，**リポジトリ** と呼ばれるデータベースに蓄積される．その内部では，ファイルのある時点における状態である **リビジョン** を単位として管理する．1 つのリビジョンには，その時点でのファイル内容である実データと，リビジョン作成日時や作成時のログメッセージ等の属性データが格納されている．また，リポジトリとのデータ授受をするために，開発者はシステムに依存した操作を利用する必要がある．

版管理手法の基礎となるモデルは数多く存在する [5, 6]．本節では，多くの版管理システムが採用しているチェックアウト・チェックインモデルについて概要を述べ，既存の版管理システムを紹介する．

2.1.1　チェックアウト・チェックインモデル

チェックアウト・チェックインモデルは，ファイルを単位としたリビジョンの制御に関して定義されている．版管理下にあるファイル群はシステムに依存した形式のファイルとしてリポジトリに格納されている．開発者はそれらのファイルを直接操作するのでなく，各システムに実装されている操作を介して，リポジトリとのデータ授受を行う．リポジトリから特定のリビジョンのファイルを取得し，開発者の手元にコピーする操作を**チェックアウト** という．逆に，ファイルに対して行った変更内容をリポジトリに送り，新たなリビジョンを作成する操作を **チェックイン**，あるいは **コミット** と呼ぶ．

チェックアウト・チェックインモデルに基づく版管理プロセスを，図 1 に示す．このモデルでは，各開発者が以下の 3 つの作業を繰り返すことでソフトウェアの開発を行なっていく．

1. リポジトリにアクセスし，必要なファイル群をチェックアウトする
2. チェックアウトしたファイル群に対し，手元で変更を行なう
3. リポジトリにアクセスし，変更を行なったファイル群をチェックインする．この作業により，リポジトリ内に新たなリビジョンが作成される

2.1.2 既存の版管理システムの紹介

版管理システムは多数存在する．UNIX 系 OS では多くの場合，RCS [7] や CVS [8] が標準で利用可能となっている．また近年開発された Subversion [9] は，CVS の後継として注目されている．ClearCase [10] のように商用のものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，Visual SourceSafe [11] や PVCS [12] をはじめ，数多く存在する．さらに，ローカルネットワーク内のみではなく，よりグローバルなネットワークを介したシステム [13] も存在する．

以下，RCS および CVS について詳しく説明する．

RCS ：

RCS (Revision Control System) [7] は UNIX 上で動作するツールとして作成された版管理システムであり，1985 年に発表されて以来，現在でも使用されているシステムの 1 つである．単体で使用される他，システム内部に組み込み，版管理機構を持たせる等の用途もある．RCS では管理対象のファイルをそれぞれ UNIX 上のファイルとして扱い，1 ファイルに対する記録は 1 つのファイルに行われる．

RCS におけるリビジョンは，管理対象となるファイルの中身がそれ自身によって定義され，リビジョン間の差分は diff コマンドの出力として定義される．各リビジョンに対する識別子は数字で表記され，数え上げ可能な識別子である．この数字を **リビジョン番号**と呼ぶ．新規リビジョンの登録や，任意のリビジョンの取り出しは，RCS の持つツールを利用する．

CVS ：

CVS (Concurrent Versions System) [8] は RCS 同様，UNIX 上で動作するシステムとして構築された版管理システムであり，近年最も広く使われるシステムの 1 つである．RCS と同様に，リビジョン間の差分は diff コマンドの出力として記録される一方，RCS と大きく異なるのは，複数のファイルを処理する点である．変更した複数のファイルを一度のコミット作業でリポジトリに登録でき，この時の一群のファイルに対する変更をまとめて **コミットトランザクション** (以降単に **トランザクション**) と呼ぶ．

また，リポジトリを複数の開発者で利用することも考慮し，開発者間の競合にも対処可能となっている．さらに，ネットワーク環境 (ssh, rsh 等) を利用することも可能である為，FreeBSD [14] や OpenBSD [15] 等のオープンソースによるソフトウェア開発に用いられることが多い．

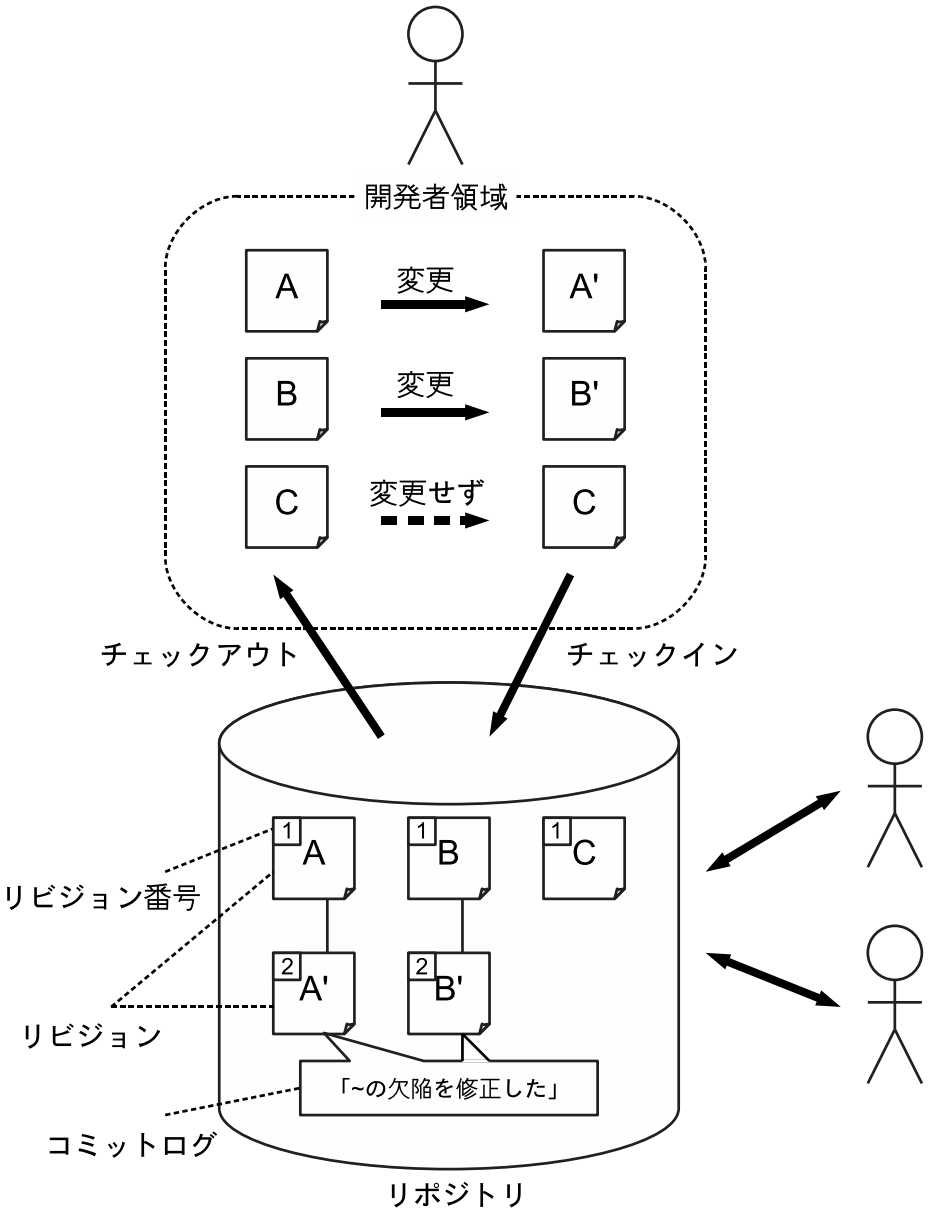

図 1: チェックアウト・チェックインモデルに基づく版管理プロセス

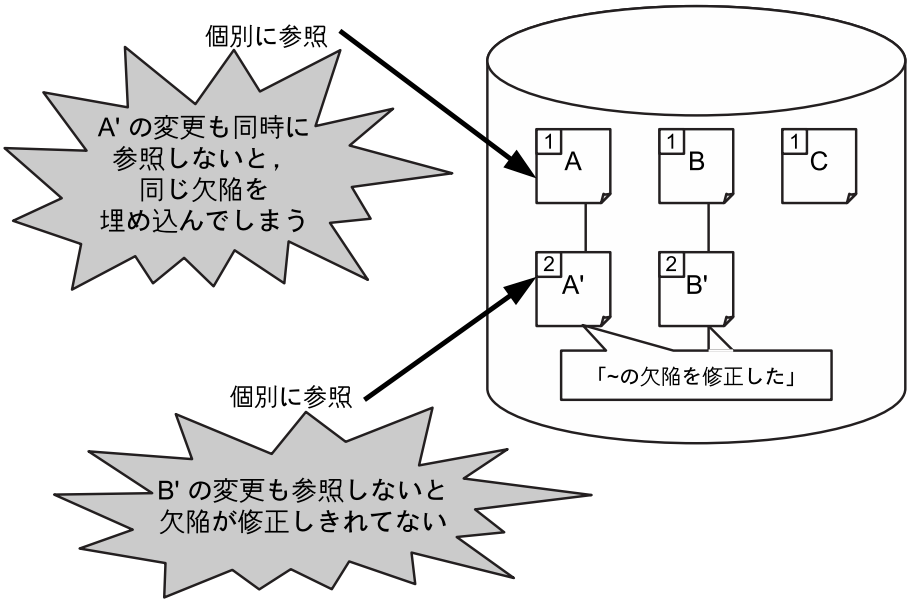

図 2: 変更を個別に参照する際の問題

2.2 ソフトウェア開発時の変更記録の利用

開発者は，開発・保守工程において版管理システムに蓄積された変更記録を参考にすることで，効率的に作業することができる．例えば，機能を追加する時や欠陥を修正する時に，過去の類似した変更を真似ることで簡単に作業を終えることができる．また，版管理システムのリポジトリを閲覧するためのシステム [16, 17] 等が開発されており，開発者は，過去に行われた変更を視覚的に確認することが可能となっている．

しかし，過去に行われた変更を個別に調べるのみでは，関連する変更を見落としてしまい，誤解が生じる可能性がある．例えば，ある箇所の変更の影響により，他の箇所が変更されていた場合や，ある変更により欠陥を混入させ，後に行われた変更でその欠陥が修正されていた場合に，前者の変更のみを真似ると欠陥を埋め込んでしまう，といったケースが考えられる (図 2).

そのため，開発者は過去の変更を参考にする際には，全ての関連する変更を把握しなければならない．しかし，蓄積された膨大な変更の中から関連する変更を人手で探すのは困難であり，開発者の負担となっている．

3　変更情報のクラスタリング手法

2.2 節で述べた問題点を解決するために，開発者に対して，ある 1 つの変更だけでなく，関連する変更を自動的にまとめて提示することを考える．

本研究ではここで，コミットログ，および変更時に記録される差分の中に出現する単語に着目した．一般的に，コミットログには変更の理由が記述されている．また，差分は何が削除され，何が追加されたかという変更の内容である．そして，ソースコード中に出現する関数名や変数名等の識別子は，一般的にそれぞれの関数や変数の意味や役割を表すように命名される．そのため，これらに出現する単語は変更を特徴付けていると考えられる．

そこで，出現する単語を基に変更をクラスタリングすることで，関連する変更の集合が得られることが期待できる．本手法では，単語を利用して変更間の関連を抽出するために，情報検索 (Information Retrieval) の分野で用いられる，ベクトル空間モデルに基づいた手法の一種である LSA (Latent Semantic Analysis) を用いる．LSA の詳細は付録 A を参照のこと．

本手法の概要を図 3 に示す．本手法は版管理システムのリポジトリに蓄積された変更の記録をクラスタリングするもので，大きく 3 つのステップからなる．

1. リポジトリからトランザクション情報を抽出する
2. LSA を用いて，トランザクション間の類似度を計算する
3. 計算した類似度を基に，トランザクション情報をクラスタリングする

以降の節で，各ステップの詳細を説明する．

3.1　トランザクション情報の抽出

手法の最初のステップでは，版管理システムのリポジトリから **トランザクション情報** を取り出してくる．トランザクション情報は **変更情報** の集合からなり，変更情報は以下の情報から構成される．

- 変更したファイルのパス名
- 変更後のリビジョン番号
- コミットした開発者
- コミットされた日時
- コミットログ

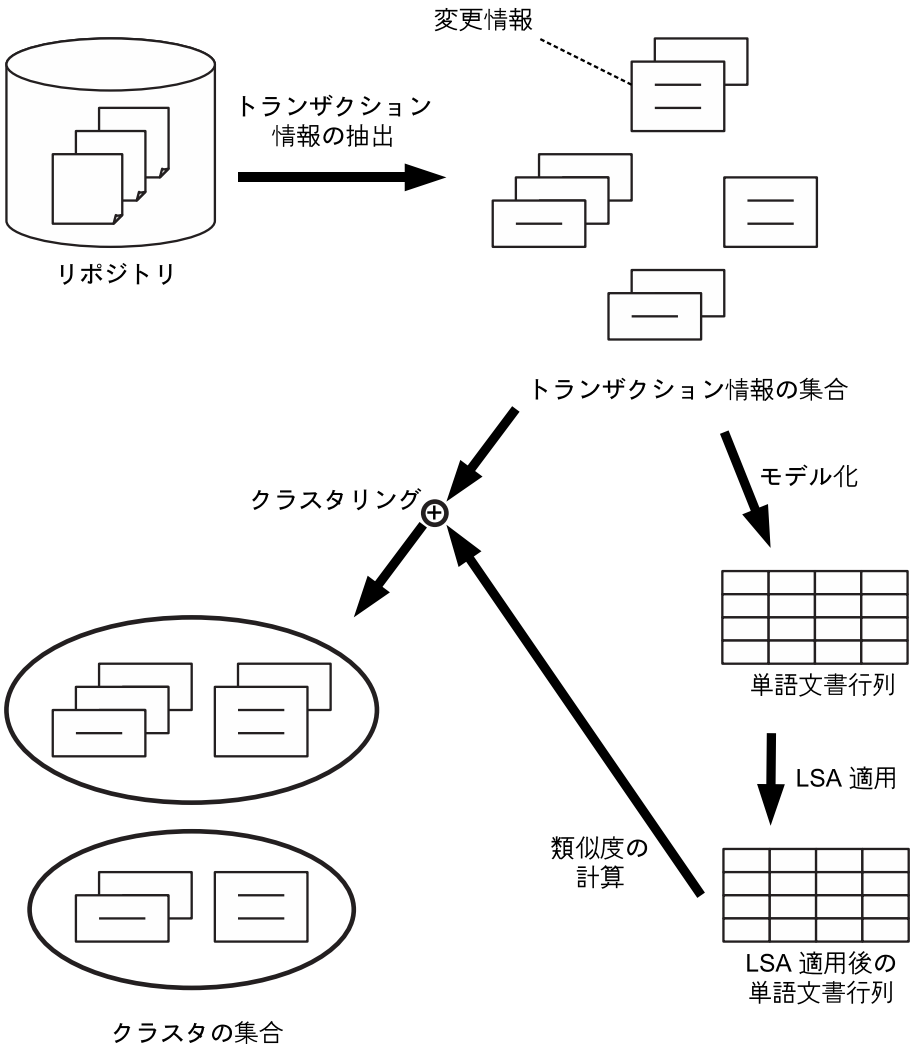
変更情報
トランザクション
情報の抽出
リポジトリ
トランザクション情報の集合
クラスタリング
モデル化
単語文書行列
LSA 適用
類似度の
計算
LSA 適用後の
単語文書行列
クラスタの集合

図 3: 手法の概要

- 変更差分

また，変更差分は，2.1.2 節で述べた，ファイルに対する変更を表す差分である diff の出力，すなわち変更開始行番号と変更範囲，追加行の内容と削除行の内容からなる．

図 4 に変更情報の例を示す．図の変更差分中，行頭に '>' が付いている行は，その時の変更で追加された行を表し，'<' が付いている行は削除された行を表している．また，数字で始まる行は変更開始行番号や変更範囲を表している．

3.2 LSA を用いたトランザクション間の類似度計算

このステップでは，A 節で述べた LSA を用いてトランザクション間の類似度を求める．このステップは，以下の 3 つのサブステップから構成される．

1. トランザクション情報からの単語抽出
2. 単語文書行列作成
3. LSA 適用

本節ではこの 3 つのサブステップを順に説明する．

3.2.1 トランザクション情報からの単語抽出

まず，3.1 節で取り出した各トランザクション情報から，単語とその出現回数を抽出する．トランザクション情報からの単語抽出は，トランザクション情報に含まれる全ての変更情報に共通するコミットログと，個々の変更差分を対象に行う．まず全てのトランザクション情報から単語を抽出し，出現数を数え上げる．トランザクション情報 T 中の単語 w の出現数は，T に含まれる全ての変更差分中の単語 w の出現数を足し合わせたものに，コミットログ中に w が出現した回数を加えたものである．

変更差分から単語とその出現数を抽出するために，本手法では以下で説明する WCA 法，WCD 法の 2 種類の方法を用いる．ただし説明中の，$C(D, w)$ は変更差分 D 中に単語 w が出現した回数, D_{add}は D の内の追加行の集合, D_{del}は D の内の削除行の集合を表すものとする．つまり，$D = D_{\mathrm{add}} \cup D_{\mathrm{del}}$ である．

WCA 法 – 変更差分中に出現する単語を全て数える方法

この方法は，変更差分中に出現する単語が全て，変更の特徴付けに寄与するとする算出方法である．WCA 法では，変更差分中の単語の出現数は次のように定義する．

$$D \text{ 中の } w \text{ の出現数} = C(D, w)$$

ファイルのパス名

cscope/src/mypopen.c

リビジョン番号

1.8

コミットした開発者

broeker

コミットされた日時

2002 年 1 月 14 日 21 時 11 分 50 秒

コミットログ

Fix against Cygwin binmode mounts of source files

変更差分

```
44a45,48
> #if !defined(HAVE_SETMODE) && defined(HAVE__SETMODE)
> # define setmode _setmode
> #endif
>
62a67,73
>       /* 20020103: if file is not explicitly in Binary mode, make
>        * sure we override silly Cygwin behaviour of automatic binary
>        * mode for files in "binary mounted" paths */
> #if O_BINARY != O_TEXT
>       if (! (flag | O_BINARY))
>               flag |= O_TEXT;
> #endif
98a110,115
> #if HAVE_SETMODE
>       if (! strchr(mode, 'b')) {
>               setmode(fileno(fp), O_TEXT);
>       }
> #endif /* HAVE_SETMODE */
>
108c125,126
<       else return(NULL);
---
>       else
>               return(NULL);
```

図 4: 変更情報の例

例えば，ある変更により削除された行が

```
foo bar baz
```

追加された行が

```
foo bar qux
```

であった場合，単語 foo，bar はそれぞれ 2 回，単語 baz，qux はそれぞれ 1 回出現したと数える．

WCD 法 – 変更差分中の削除行・追加行における各単語の出現数の差の絶対値を単語の出現数とする方法

この方法は，変更差分中の削除行または追加行のどちらか一方にのみ多く表れる単語が，変更の特徴付けに寄与するという考えに基づく算出方法である．WCD 法では，変更差分中の単語の出現数は次のように定義する．

$$D \text{ 中の } w \text{ の出現数} = |C(D_{\text{add}}, w) - C(D_{\text{del}}, w)|$$

WCA 法と同じ例の場合，単語 foo，bar は削除・追加両方に 1 回ずつ出現しているため 0 回，単語 baz，qux は 1 回出現したと数える．

3.2.2 単語文書行列作成

続いて，抽出した単語とそれらの出現回数を基に単語文書行列を作成する．単語文書行列については，付録 A.1 節で詳しく説明する．本手法では，単語文書行列を作成する際に，1 つのトランザクション情報を 1 つの文書とみなして行列を作成する．この行列は，トランザクション数を m，単語数を n とした時，$m \times n$ 行列となる．

この時あらかじめ，抽出した単語のうち，少数のトランザクション情報にのみ出現する単語，および多数のトランザクション情報に出現する単語を取り除いておく．これは，少数の文書にのみ出現する単語は文書間を関連付けるのに役に立たず，多数の文書に出現する単語は文書を特徴付けるのに不適当とされるためである [18]．本手法では，単一トランザクション情報にのみ出現する単語，および過半数のトランザクション情報に出現する単語を除去するものとした．

3.2.3 LSA の適用

トランザクション情報間の類似度を算出するために，不要語を除去した単語文書行列に対して付録 A 節で述べる LSA を適用する．

そして，LSA を適用した行列から，各トランザクションに対応する文書ベクトルを取り出し，トランザクション間の類似度を求める．文書ベクトル間の類似度算出には，付録 A.1 節でも述べる通り，ベクトル間の cosine 尺度を用いる．

3.3 トランザクション情報のクラスタリング

3.2 節の方法により算出されるトランザクション情報間の類似度に基づいて，トランザクション情報をクラスタリングする．クラスタリングアルゴリズムには，階層的クラスタリング手法 [19] を採用した．階層的クラスタリング手法とは，入力となる集合に含まれる要素それぞれを独立したクラスタであるとみなし，そして最も類似度の高いクラスタから順次結合する手法である．

クラスタ間の類似度算出には，最も類似度の低い文書間の類似度をクラスタ間の類似度とする完全リンク法 [19] を採用する．クラスタ C, D 間の類似度 $\mathrm{sim}(C, D)$ は，以下の式により求められる．

$$\mathrm{sim}(C, D) = \min_{v_C \in C, v_D \in D} \cos(v_C, v_D)$$

具体的なクラスタリングのアルゴリズムは，以下の通りである．ただし，s_{th} は閾値を表す．

N : 総トランザクション数
$T_1, T_2, \ldots, T_N$: トランザクション情報
$S \leftarrow \{\{T_i\} \mid 1 \leq i \leq N\}$
while $|S| \neq 1$ **do**
  以下の条件を満たすクラスタの組 $(a, b)(a, b \in S)$ を求める :
    $a \neq b \ \wedge \ \forall c, d \in S, \ c \neq d \ \wedge \ \mathrm{sim}(a, b) \geq \mathrm{sim}(c, d)$
  if $\mathrm{sim}(a, b) \leq s_{\mathrm{th}}$ **then**
    ループ終了
  end if
  $S \leftarrow S \cup \{a \cup b\} - \{a, b\}$
end while
S がトランザクション情報のクラスタリング結果となる

図 5 に，上記アルゴリズムに基づき，閾値 0.7 としてトランザクション情報をクラスタリングする経過を示す．

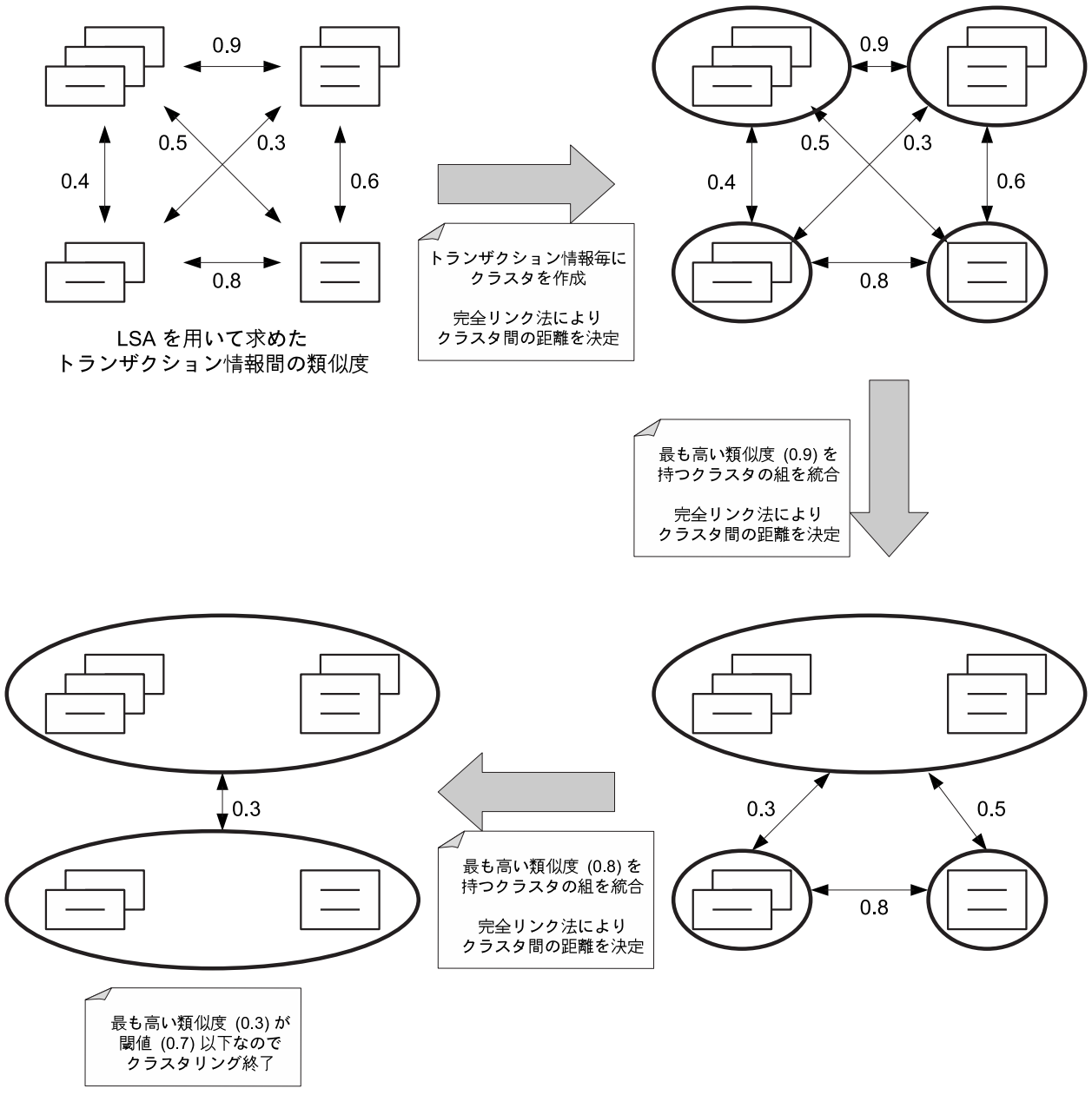

図 5: 閾値 0.7 におけるクラスタリングの経過

表 1: システムの実装環境

プログラミング言語	Ruby 1.8.5
データベース	Berkeley DB 4.2
対象とする版管理システム	CVS

4 システムの実装

本節では，3 節で述べた変更情報のクラスタリング手法を実現するために試作したシステムの詳細について述べる．

まず表 1 に，システムの実装環境を示す．版管理システムとして CVS を対象としたのは，現在までに広く活用されており，多くのソフトウェア・リポジトリが入手可能となっているためである．

続いて図 6 に，実装したシステムの概要を示す．本システムは，大きく 4 つのサブシステムから構成されている．変更情報抽出部は， CVS リポジトリから変更情報を取り出す部分である．トランザクション復元部は，変更情報からトランザクションを復元する部分である．LSA 適用部は，トランザクション情報を基に LSA を適用する部分である．そしてクラスタリング実行部が，トランザクションをクラスタリングする部分である．さらに，これらのサブシステムがデータを保存・取得するための，データベース部が存在する．

以降の節で，本システムを構成するデータベース部と 4 つのサブシステムについて詳しく説明する．

4.1 データベース部

データベース部は，以下の 3 つのデータベースからなっている．

- 変更情報データベース
- トランザクションデータベース
- クラスタデータベース

本節では，各データベースの役割と保持する情報を説明する．なお，データベースには Berkeley DB を用いているため，各データベースの内容は，キーと値の組を要素とした集合となる．

変更情報データベース :

変更情報データベースは，CVS リポジトリから抽出してきた変更情報を保持するデー

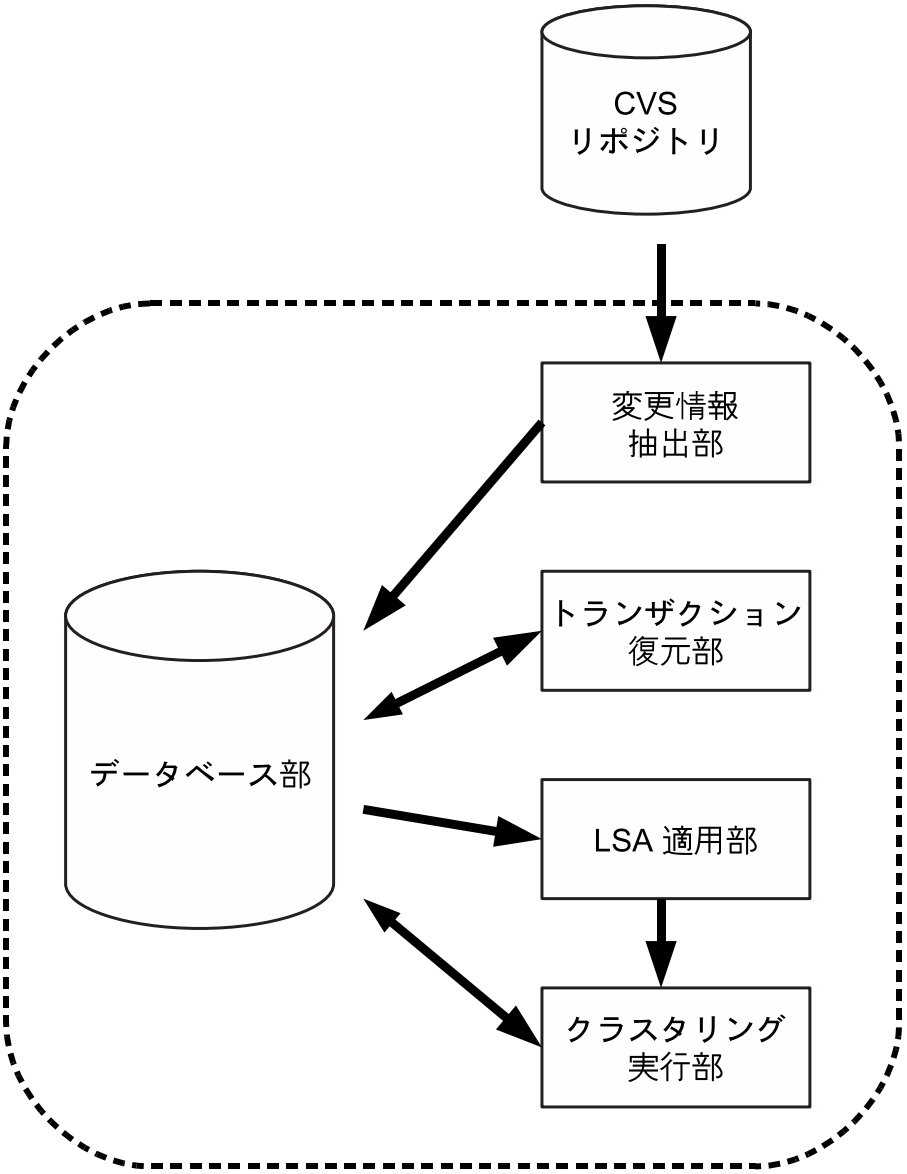

図 6: システムの概要

タベースである．ファイルのパス名をキーとし，そのファイルの変更情報の配列を値にもつ．

トランザクションデータベース ：
トランザクションデータベースは，変更情報から復元されたコミットトランザクション情報を保持するデータベースである．各トランザクション情報には，それらを特定するためのトランザクション ID を割り振る．そして本データベースでは，トランザクション ID をキーとし，対応するトランザクションに含まれる変更情報の配列を値にもつ．

クラスタデータベース ：
クラスタデータベースは，トランザクションをクラスタリングした結果を保持するデータベースである．各クラスタには，それらを特定するためのクラスタ ID を割り振る．そしてクラスタデータベースでは，クラスタ ID をキーとし，クラスタ中の全トランザクションに含まれている変更情報の配列を値にもつ．

4.2 変更情報抽出部

変更情報抽出部は，3.1 節で述べた処理の内，CVS リポジトリから変更情報を抽出する部分を実現している．

CVS リポジトリでは，変更の記録はファイル単位で管理されているため，まず始めに管理対象のファイルを全て取得してくる．そして，取得したファイルそれぞれに対して，記録されている全ての変更を，変更情報として抽出する．

抽出した変更情報は，変更情報データベースに書き出す．

4.3 トランザクション復元部

トランザクション復元部は，3.1 節で述べた処理の内，変更情報からトランザクションを復元する部分を実現している．

変更情報データベースより変更情報を取得し，トランザクションを復元する．CVS リポジトリでは，トランザクションを構成するファイルやリビジョン情報が記録されていない．そのため本システムでは，文献 [20] で提案されている次の定義を用いて，トランザクションを特定した．

> 変更の集合が以下の 3 つの条件を全て満たす時，その変更の集合をトランザクションとする:
>
> - コミットした開発者が同じである

- コミットログが同じである
- コミットされた日時の差が 200 秒以内である

そして，復元したトランザクション情報を，トランザクションデータベースに書き出す.

4.4 LSA 適用部

LSA 適用部は，3.2 節で述べた処理を実現している.

トランザクションデータベースより，トランザクション情報を取り出し LSA を適用する．LSA で用いる特異値分解の実行には，線形代数ライブラリ LAPACK [21]，および，LAPACK を Ruby から扱うためのライブラリ Linalg [22] を用いている.

また，LSA 適用後の行列を，以下の書式に従ってファイルに書き出す.

1 行目: (行数) (列数)

2 行目以降: (数字の並び．1 行に，行列 1 行分を，空白区切りで書き出す)

例:

```
4 4
3.2 2.1 0.4 -0.3
2.2 1.6 0.6 0.2
0.7 0.5 0.7 0.7
-0.1 0.2 2.5 2.8
```

4.5 クラスタリング実行部

クラスタリング実行部は，3.3 節で述べた処理を実現している.

トランザクションデータベース中のトランザクション情報と，LSA 適用後の行列を入力とし，トランザクション情報をクラスタリングする.

この工程は，以下の 3 つのステップからなっている.

1. 全てのトランザクション情報の組み合わせに対する，トランザクション情報間の類似度を計算する
2. トランザクション情報の組を，類似度の降順に整列する
3. 階層的クラスタリング手法により，類似度の高いトランザクション情報から順次結合する

そして，生成されたクラスタを，クラスタデータベースに書き出す.

表 2: 適用対象ソフトウェアの概要

	Cscope	SPARS-J
開発期間	2000 年 4 月 – 2007 年 1 月	2003 年 3 月 – 2006 年 1 月
総リビジョン数	451	2795
総トランザクション数	160	834
総ファイル数	44	345
最新リビジョンでの総行数	14386	95479
開発言語	C	C/C++

表 3: 実行環境

CPU	Intel Xeon 2.8 GHz
RAM	4 GB
OS	FreeBSD 6.2-STABLE

5 適用事例

本手法により関連のある変更情報が抽出できるかを確認するため，2 つのソフトウェアに対して本システムを適用した．適用対象には，オープンソースソフトウェアの C/C++/Java 用クロスリファレンサ・ソースコードビューア Cscope [23] と，我々の研究室で開発されたソフトウェア部品検索システム SPARS-J (Software Product Archive, analysis and Retrieve System for Java) [24] を用いた．それぞれのソフトウェアの概要は表 2 の通りである．また，適用に用いた実行環境は表 3 の通りである．

本節では，まず適用の際の方針について述べ，その後それぞれの適用結果について説明する．

5.1 適用方針

WCA 法と WCD 法の両方で単語を抽出し，それぞれでクラスタリングする．クラスタリングの際の類似度閾値 s_{th} は共に 0.8 とした．

そして，生成されたクラスタの内，複数トランザクション情報を含むものを，そこに含まれるトランザクション間の関連により，以下の 7 つに分類する．

欠陥修正 :

クラスタ内の，あるトランザクションに含まれる変更が，別のトランザクションに含まれる変更で混入させてしまった欠陥の修正である

リファクタリング :
クラスタ内の，あるトランザクションに含まれる変更が，別のトランザクションに含まれる変更箇所のリファクタリングである

共通の目的 :
クラスタ内の，あるトランザクションに含まれる変更と，別のトランザクションに含まれる変更が，共通の目的を持つ

影響波及 :
クラスタ内の，あるトランザクションに含まれる変更が，別のトランザクションに含まれる変更の影響を受けて行われている

変更取り消し :
クラスタ内の，あるトランザクションに含まれる変更が，別のトランザクションに含まれる変更の取り消しである

その他 :
仕様の変更，マージ，変更量が大きく理解に不適，等

無関係 :
クラスタ内のトランザクション間に関連がない

欠陥修正・共通の目的・リファクタリング・影響波及・変更取り消しに分類されたクラスタについては，これに含まれる複数のトランザクションを同時に参照することが望ましいと考え，本研究ではこれらのクラスタを「有用である」と判断する．

以降，上記の適用方針に基づいた適用結果を示す．

5.2 Cscope に対する適用結果

本節では，Cscope に対する適用結果を述べる．

クラスタリングの結果は表 4 の通りである．実行時間は WCA 法・WDC 法ともに約 30 秒で，WCA 法により 71.4%，WCD 法により 57.1%，有用であると思われるクラスタを抽出することができた (表中網掛けの分類)．また，総クラスタ数に占める複数トランザクション情報を含むクラスタ数の割合は WCA 法で 29.2%，WCD 法で 23.7% であった．

表 4: Cscope を対象としたクラスタリング結果

		WCA 法	WCD 法
総クラスタ数		96	118
複数トランザクション情報を含むクラスタ数		28 (100%)	28 (100%)
	欠陥修正	9 (32.1%)	5 (17.9%)
	リファクタリング	2 (7.1%)	1 (3.6%)
	共通の目的	7 (25.0%)	6 (21.4%)
	影響波及	2 (7.1%)	4 (14.3%)
	変更取り消し	0 (0%)	0 (0%)
	その他	4 (14.3%)	5 (17.9%)
	無関係	4 (14.3%)	7 (25.0%)
クラスタ内の平均トランザクション数		3.29	2.50

ここでは，WCA 法により単語を抽出したクラスタリング結果から「欠陥修正」分類のクラスタ，WCD 法により単語を抽出したクラスタリング結果から「影響波及」分類のクラスタを例として示す．但し，説明の都合上一部のみを抜粋している．

- WCA 法により単語抽出したクラスタリング結果

  図 7 に示したのは,「欠陥修正」に分類されたクラスタの例である．

  このクラスタは 3 つのトランザクション情報からなっており，含まれる一連のトランザクションでは，ファイルのオープン処理に関する変更が行われている．1 つ目に示したトランザクションで，文字列変数 `mode` に文字 'b' が含まれていなかったら，変数 `fp` の指し示すファイルのモードを `O_TEXT` (テキストモード) にセットするようにコードが追加されている．2 つ目のトランザクションは，説明上不要なためここでは割愛する．そして 3 つ目のトランザクションにおいて，この条件文では `fp` で指し示されるファイルが存在しない場合にシステムがクラッシュしてしまうために，`fp` が `NULL` でないかを確認するための条件が追加されている．

- WCD 法により単語抽出したクラスタリング結果

  図 8 に示すのは,「影響波及」に分類されたクラスタの例である．

  このクラスタは 2 つのトランザクション情報からなっていた．1 つ目に示したトランザクションで，グローバル変数 `pattern` が `Pattern` に書き換えられており，その影

クラスタ

トランザクション

変更

```
パス名: cscope/src/mypopen.c
リビジョン番号: 1.8
コミットされた日時: 2002-01-04 21:11:50
98a110,115
> #if HAVE_SETMODE
>       if (! strchr(mode, 'b')) {
>               setmode(fileno(fp), O_TEXT);
>       }
> #endif /* HAVE_SETMODE */
```

... (省略) ...

トランザクション

... (省略) ...

トランザクション

変更

```
パス名: cscope/src/mypopen.c
リビジョン番号: 1.11
コミットされた日時: 2002-08-16 01:07:07
106c106
<       if (! strchr(mode, 'b')) {
---
>       if (fp && ! strchr(mode, 'b')) {
```

図 7: Cscope を対象とした，WCA 法でのクラスタの例 (抜粋)

表 5: SPARS-J を対象としたクラスタリング結果

		WCA 法	WCD 法
総クラスタ数		644	666
複数トランザクション情報を含むクラスタ数		116 (100%)	113 (100%)
	欠陥修正	33 (28.4%)	23 (20.4%)
	リファクタリング	10 (8.6%)	11 (9.7%)
	共通の目的	14 (12.1%)	19 (16.8%)
	影響波及	2 (1.7%)	8 (7.1%)
	変更取り消し	5 (4.3%)	2 (1.8%)
	その他	28 (24.1%)	20 (17.7%)
	無関係	24 (20.7%)	30 (26.5%)
クラスタ内の平均トランザクション数		2.26	2.47

響を受け 2 つ目のトランザクションでも，`pattern` を `Pattern` にするための変更が行われている．

上記で挙げているようなクラスタは，クラスタに含まれるトランザクション間に明確な関連があり，有用だと思われるクラスタの例であったが，そうでないクラスタも見られた．有用でないと思われるクラスタでは，そこに含まれるトランザクション群に files や avoid 等の単語が共通して出現していたが，変更内容に関連は見受けられなかった．

5.3 SPARS-J に対する適用結果

本節では，SPARS-J に対する適用結果を述べる．

クラスタリングの結果は表 5 の通りである．実行時間は WCA 法・WCD 法ともに約 11 分で，WCA 法により 55.2%，WCD 法により 55.8%，有用だと思われるクラスタが抽出できた．また，総クラスタ数に占める複数トランザクション情報を含むクラスタ数の割合は WCA 法で 18.0%，WCD 法で 17.0 % であった．

ここでは，WCA 法により単語を抽出したクラスタリング結果から「欠陥修正」分類のクラスタ，WCD 法により単語を抽出したクラスタリング結果から「共通の目的」分類のクラスタを例として示す．

- WCA 法により単語抽出したクラスタリング結果

  図 9 に示したのは,「欠陥修正」に分類されたクラスタである．

クラスタ

トランザクション

変更

```
パス名: cscope/src/command.c
リビジョン番号: 1.23
コミットされた日時: 2004-05-01 00:24:18
58c58,60
< char  pattern[PATLEN + 1];    /* symbol or text pattern */
---
> /* HBB 20040430: renamed to avoid lots of clashes with function
arguments
>  * also named 'pattern' */
> char  Pattern[PATLEN + 1];    /* symbol or text pattern */
... (中略) ...
457c463
<                       (void) strcpy(pattern, item->text);
---
>                       (void) strcpy(Pattern, item->text);
```

トランザクション

変更

```
パス名: cscope/src/display.c
リビジョン番号: 1.25
コミットされた日時: 2004-05-01 00:29:14
178c178
<    printw("Change \"%s\" to \"%s\"", pattern, newpat);
---
>    printw("Change \"%s\" to \"%s\"", Pattern, newpat);
```

図 8: Cscope を対象とした，WCD 法でのクラスタの例 (抜粋)

このクラスタは 3 つのトランザクション情報からなっており，SPARS-J で利用しているデータベースの操作時に必要な，比較関数に関連のある変更集合が抽出できた．1 つ目のトランザクションでは，ファイル SPARS/src/DB/db_common.c で比較関数を定義しており，その中で変数 ai，bi が u_int8_t * 型と宣言されている．2 つ目のトランザクションでその型に合わせるために，変数に代入している箇所でキャストを行う修正がなされたが，そのキャストも間違っていたため，3 つ目のトランザクションで同一箇所が更に修正されている．

- WCD 法により単語抽出したクラスタリング結果

  図 10 に示したのは,「共通の目的」に分類されたクラスタである．

  図 10 の例では，データベース項目に関連する変更群が抽出できた．まず最初のトランザクションでデータベースの項目にファイルの mtime (更新時間) を追加する変更が行われ，続くトランザクションでデータベースへアクセスする関数の引数に対し，その項目が追加されている．

また，SPARS-J の場合も Cscope 同様，有用でないと思われるクラスタが見られた．そのようなクラスタの例として，コマンドラインオプションを追加する変更からなるクラスタや，CGI プログラムに対する変更からなるクラスタ等があった．前者では，あるコマンドラインツールに対して，別々のオプションを追加するための変更が存在し，それぞれで usage, getopt (C 言語のコマンドラインオプション解析ライブラリ) 等の単語が共通して出現していた．また後者では，HTML タグ文字列が共通して多く出現していた．これらのクラスタに含まれるトランザクション間には，機能的な関連がない等の理由で，同時に参照する必要がないと判断し，有用でないクラスタに分類した．

5.4 考察

本稿の適用事例では，有用と思われるクラスタが生成できた一方，含まれるトランザクション間に関連がないクラスタも多数見られた．これは，出現する単語を基にする手法の性質によるもので，類似した単語が出現すればクラスタとしてまとめられるのが原因であると考えられる．有用であると思われる例でもそうでない例でも，クラスタを構成する各トランザクション情報が単語を共有していたからである．有用なクラスタには，ソースコード中の特定の変数・関数名等の識別子が多く見られ，有用でないクラスタには，コミットログやコメント文中の一般的な単語や，上記 HTML タグ等が多く見られた．

また，WCA 法と WCD 法の比較においても，生成されるクラスタの傾向に違いが見られた．

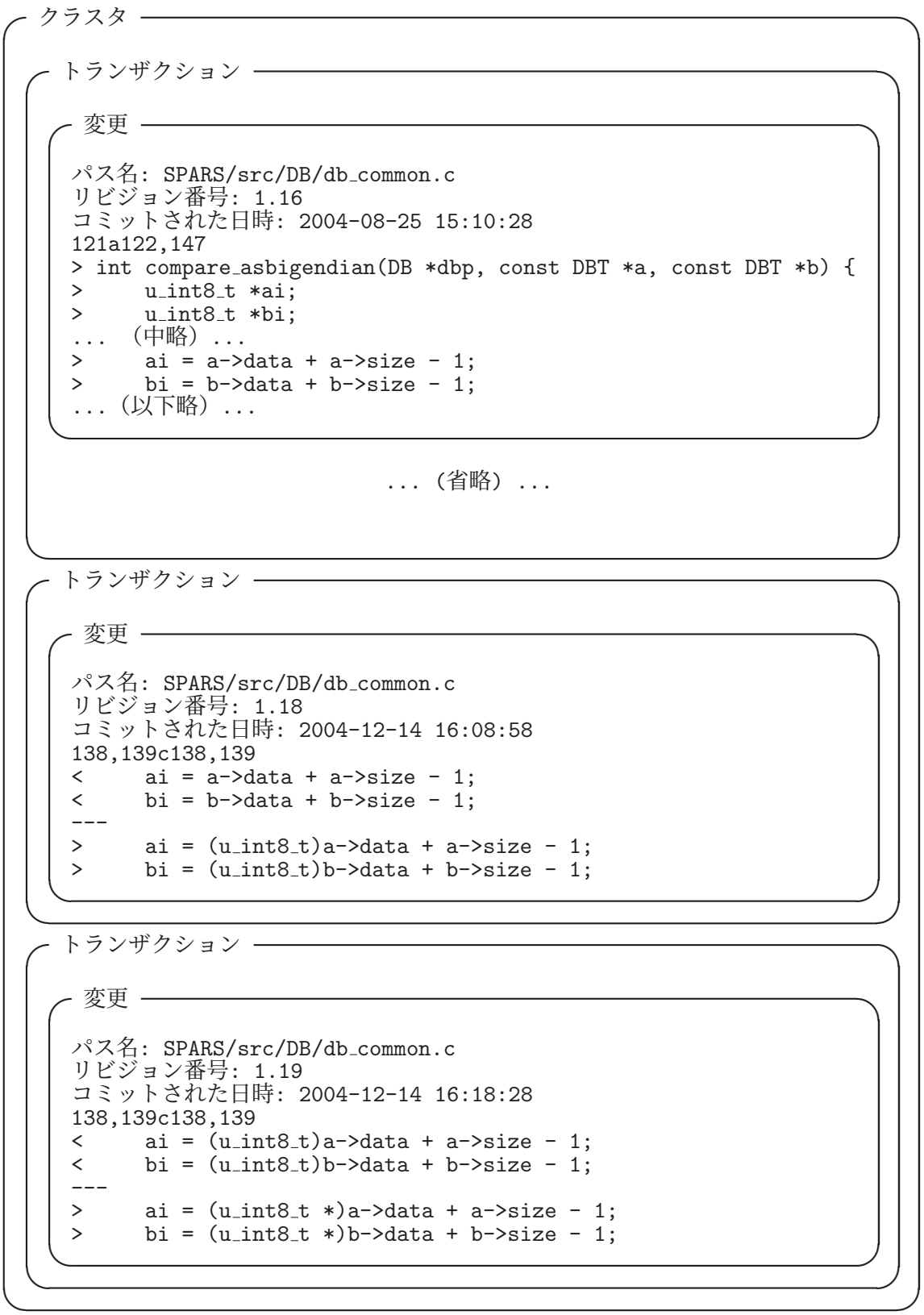

図 9: SPARS-J を対象とした，WCA 法でのクラスタの例 (抜粋)

クラスタ

トランザクション

変更

```
パス名: SPARS/src/DB/spars_db.h
リビジョン番号: 1.5
コミットされた日時: 2003-04-15 14:11:35
122c122
<  * ファイル ID -> (コメント)
---
>  * ファイル ID -> (更新時間, コメント)
126a127
>     time_t mtime;
```

変更

```
パス名: SPARS/src/Register/regist.c
リビジョン番号: 1.4
コミットされた日時: 2003-04-15 14:11:35
22c22
< void regist(char *path) {
---
> void regist(char *path, time_t mtime) {
50c50
<     put_fileinforepository(id, description);
---
>     put_fileinforepository(id, mtime, description);
```

... (省略) ...

トランザクション

変更

```
パス名: SPARS/src/Search/perform_search.c
リビジョン番号: 1.5
コミットされた日時: 2003-04-15 16:09:05
265c265
<     get_fileinforepository(ci->file_id, &(ci->fileinfo));
---
>     get_fileinforepository(ci->file_id, &(ci->mtime), &(ci->
fileinfo));
```

... (省略) ...

図 10: SPARS-J を対象とした，WCD 法でのクラスタの例 (抜粋)

WCA 法では，変更差分中に出現する全単語を数え上げるため，変更箇所が重複しており変更内容が類似している，すなわち同一単語を多くもつトランザクション群がクラスタを形成するパターンが大半を占めていた．特に変更箇所が重複する「欠陥修正」分類や「リファクタリング」分類のクラスタが多く見つかった．

他方 WCD 法では，変更差分中の追加箇所・削除箇所に出現する単語が相殺される形になるため，変更内容が類似していなくても，特定の変数・関数名等の識別子のような，変更の特徴を表していると思われる単語を共通して持つトランザクション群がクラスタを形成するパターンが多く見られた．その結果,「共通の目的」や「影響波及」に分類されたクラスタが WCA 法より多く生成される傾向にあった．しかし，トランザクション間に関連のないクラスタが WCA 法より多く生成されていた．これは WCA 法に比べ，抽出できる単語が少なくなり，変更行数が少ない場合等に特徴が得られなくなるためであると考えられる．

そのため，両者の方法を組み合わせることで，各変更内容に関してより特徴的な単語を抽出する，あるいは出現する単語によってはクラスタを除去するといったことを行う必要がある．

Cscope と SPARS-J の比較としては，Cscope の方が，総クラスタ数に占める複数トランザクション情報を含むクラスタ数の割合が多いという結果が出た．Cscope では，1 つのトランザクションが複数の意味的な変更を含んでいることがあり，一方 SPARS-J では，コミットの粒度が Cscope に比べ小さかった．そのため，Cscope の方がトランザクション間に共通する単語が多く，結果トランザクション間の類似度が高くなり，複数トランザクション情報を含むクラスタが多く生成されたのではないかと考えている．また，分類結果としては，Cscope の方が有用だと思われるクラスタが若干多く生成されており,「共通の目的」・「影響波及」の分類にその傾向が主に現れていた．しかし，全体としての傾向に大きな差は見られなかった．本稿の事例では，比較的小規模なソフトウェアを対象とした適用にとどまったため，今後はより大規模なソフトウェアを対象にした適用を試みる必要があるだろう．

6 関連研究

6.1 変更履歴閲覧

変更履歴の閲覧を支援するシステムとして，CVSweb [16] や ViewVC [17] 等が挙げられる．これらのシステムは，版管理システムのリポジトリ内に保存されている変更履歴情報を Web ブラウザ経由で視覚的に閲覧できるシステムであり，ファイルの各リビジョンの内容を表示する機能やファイルの任意のリビジョン間の差分を色付きで表示する機能，正規表現を用いたリビジョン検索機能等を備える．

これらのシステムは，時間によるリビジョン間の関連を認識することが困難であるため，中山らはリビジョン関係を考慮した関数クロスリファレンス機能を備えた，変更履歴閲覧システムを開発した [25].

6.2 リポジトリ解析

版管理システムを用いて，ソースコードを検索する研究も行われている．Chen らは，版管理システム中のコミットログを基にソースコードを検索するシステムを作成した [26]．この文献では，コミットログは変更の目的を記述するだけではなく，変更箇所の機能も間接的に説明しているとみなして，コミットログと変更された行を対応付けする．田原らは，版管理システムに蓄積された変更記録から，類似したコード片を検索するシステムを開発した [27]．このシステムでは，変更記録のコード片をトークン列としてデータベース化しておき，入力としてコード片を与えると，トークン列に対するアラインメント操作を行い，類似したコード片を提示する．

また，ソフトウェアの変更履歴を用いて，ファイルや変数，関数等のプログラム要素間の関連を抽出するための研究も行われている．Gall らは，ソフトウェアの変更履歴から，システムの論理的依存関係 (*Logical Dependency*) やモジュール間の変更パターンを抽出するための研究を行った [28]．このように，プログラム要素間が論理的依存関係をもっていることを *Logical Coupling* と呼び，ソースコード構造の依存関係 (*Syntactic Dependency*) の解析のみでは抽出できない関係として，それ以降の研究でも用いられている．Ying らは，変更履歴からファイル間の相関ルールを抽出し，あるファイル集合が変更された時に，抽出したルールを基に変更すべきファイル集合を推薦する手法を提案した [29]．Zimmermann らも文献 [29] と同様の手法を提案しているが，ファイル以外にも，変数や関数等のプログラム要素も対象として，変更すべき要素を提示する研究を行った [30]．Hassan らは，ソースコードの同時更新情報以外に，プログラム要素間の参照関係やファイルやディレクトリ構造等も利用したヒューリスティック手法により，あるプログラム要素が変更された時に，他に変更しなければならないプログラム要素を予測する手法を提案した [31]．Kagdi らは，ト

ランザクション (文献中では change-set) 間の時間間隔やコミットした開発者等の情報に基づいたヒューリスティック手法を用いて，ファイルの変更順序を調べるための研究を行った [32]．楠田は，オブジェクト指向ソフトウェアを対象にして，メソッドの同時更新履歴を用いてクラスを機能別に分類する手法を提案している [33]．

版管理システムのみではなく，他のシステムに蓄積された情報も用いて，開発・保守を支援する研究も行われている．Canfora らは，版管理システム中のコミットログと，欠陥追跡システム中の変更要求記述を関連付け，新たな変更要求記述が入力されると過去の類似した変更要求記述を抽出し，変更される可能性のあるファイルを提示するシステムを作成した [34]．佐々木らは，版管理システム，メーリングリストアーカイブ，欠陥追跡システムに蓄積された情報を相互に結び付け，関連する一連の開発履歴情報として開発者に提示するシステムを作成している [35]．

7　まとめ

本研究では，版管理システムに蓄積された変更情報をクラスタリングし，関連ある変更群を抽出する手法を提案した．本手法では，コミットログと変更差分中に出現する単語を基にトランザクション情報間の類似度を計測し，クラスタリングする．

また，提案した手法に基づいてシステムを試作し，実際のソフトウェアに対し適用を試みた．その結果，関連のある変更を抽出できる場合があることが分かったが，手法が有効に働かない場合があることも分かった．

今後の課題としては，以下のものが挙げられる．

- 単語の抽出手法の改善

  本手法では，変更差分からの単語抽出の際，2 通りの方法を試みたが，いずれの方法でもトランザクション間に関連のないクラスタを生成する場合があった．そこでこれらを組み合わせて，より特徴的な単語を抽出する，あるいは単語を除去するといった，手法の改善が必要である．また，単語に対する重み付けも検討事項として挙げられる．

- クラスタリングの際の閾値決定方法の確立

  本稿では，閾値を恣意的に 0.8 と決定し手法を適用したが，対象とするソフトウェア・リポジトリの規模や，抽出された単語の総数等に基づき，適切な閾値を決定できることが望ましいと考える．

- 静的解析手法と組み合わせた手法の検討

  適用対象をソースコードに限定し，本手法と共にソースコードの静的解析結果を用いることで，類似した単語が多く出現しない場合でも，関連する変更群が抽出されることが期待できる．

- 手法の定量的評価

  本稿の適用では，生成されたクラスタを著者の判断に基づき分類し，その結果を紹介するのみにとどまったが，生成されたクラスタに対し明確な基準を設けることで，手法を定量的に評価し，改善していきたいと考えている．

謝辞

本研究の全過程を通して，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本研究の過程を通して，適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 助教授に深く感謝いたします.

本論文の作成にあたり，御指導や御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 早瀬 康裕 氏に深く感謝いたします.

本論文の作成にあたり，御指導や御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 市井 誠 氏に深く感謝いたします.

最後に，その他様々な御指導，ご助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] Neville J. Ford and Mark Woodroffe: "Introducing software engineering", *Prentice-Hall*, 2004.

[2] Jacky Estublier: "Software Configuration Management: A Roadmap", In *Proceedings of the Conference on The Future of Software Engineering in 22nd ICSE*, pp.279–289, 2000.

[3] Peter H. Feiler: "Configuration Management Models in Commercial Environments", *CMU/SEI-91-TR-7 ESD-9-TR-7*, 1991.

[4] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman: "Indexing by Latent Semantic Analysis", *Journal of the Society for Information Science*, Vol.41, No.6, pp.391–407, 1990.

[5] Reidar Conradi and Bernhard Westfechtel: "Version models for software configuration management", *ACM Computing Surveys*, Vol.30, No.2, pp.232–282, 1998.

[6] Ulf Asklund, Lars Bendix, Henrik B Christensen, and Boris Magnusson: "The Unified Extensional Versioning Model", In *Proceedings of the 9th International Symposium on System Configuration Management (SCM-9)*, pp.100–122, 1999.

[7] Walter F. Tichy: "RCS – A System for Version Control", *Software – Practice and Experience*, Vol.15, No.7, pp.637–654, 1985.

[8] Brian Berliner: "CVS II: Parallelizing Software Development", In *Proceedings of the USENIX Winter 1990 Technical Conference*, pp. 341–352, 1990.

[9] "Subversion", `http://subversion.tigris.org/`

[10] "Rational clearcase", `http://www-306.ibm.com/software/awdtools/clearcase/`

[11] "Microsoft Visual SourceSafe", `http://msdn2.microsoft.com/en-us/vstudio/aa718670.aspx`

[12] "Serena Software PVCS Professional Suite", `http://www.serena.com/US/products/pvcs/index.aspx`

[13] Peter Fröhlich and Wolfgang Nejdl: "WebRC: Configuration Management for a Co-operation Tool", In *Proceedings of the SCM-7 Workshop on System Configuration Management*, pp.175–185, 1997.

[14] “The FreeBSD Project”, http://www.freebsd.org/

[15] “OpenBSD”, http://www.openbsd.org/

[16] “FreeBSD CVSweb Project”, http://www.freebsd.org/projects/cvsweb.html

[17] “ViewVC: Repository Browsing”, http://www.viewvc.org/

[18] 北 研二, 津田 和彦, 獅子堀 正幹: “情報検索アルゴリズム”, 共立出版, 2002.

[19] 徳永 健伸: “情報検索と言語処理”, 東京大学出版会, 1999.

[20] Thomas Zimmermann, and Peter Weissgerber: “Preprocessing CVS Data for Fine-Grained Analysis”, In *Proceedings of 1st International Workshop on Mining Software Repositories (MSR)*, 2004.

[21] “LAPACK – Linear Algebra PACKage”, http://www.netlib.org/lapack/

[22] “Linalg – Ruby Linear Algebra Library”, http://linalg.rubyforge.org/

[23] “Cscope Home Page”, http://cscope.sourceforge.net/

[24] 横森 励士, 梅森 文彰, 西 秀雄, 山本 哲男, 松下 誠, 楠本 真二, 井上 克郎: “Java ソフトウェア部品検索システム SPARS-J”, 電子情報通信学会論文誌 D-I, Vol.J87-D-I, No.12, pp.1060-1068, 2004.

[25] 中山 崇, 松下 誠, 井上 克郎: “関数の変更履歴と呼出し関係に基づいた開発履歴理解支援システム”, 電子情報通信学会技術研究報告 SS2004-2, Vol.104, No.47, pp.7–12, 2004.

[26] Annie Chen, Eric Chou, Joshua Wong, Andrew Y. Yao, Qing Zhang, Shao Zhang, and Amir Michai: “CVSSearch: Searching through Source Code using CVS Comments”, In *Proceedings of the 17th International Conference on Software Maintenance (ICSM'01)*, pp.364–374, 2001.

[27] 田原 靖太, 松下 誠, 井上 克郎: “既存ソフトウェアの変更履歴を利用したソースコード修正支援手法の提案”, 情報処理学会研究報告, 2002-SE-136, Vol.2002, No.23, pp.57–64, 2002.

[28] Harald Gall, Karin Hajek, and Mehdi Jazayeri: “Detection of Logical Coupling Based on Product Release History”, In *Proceedings of the International Conference on Software Maintenance (ICSM'98)*, pp.190–198, 1998.

[29] Annie T.T. Ying, Gail C. Murphy, Raymond Ng, and Mark C. Chu-Carroll: “Predicting Source Code Changes by Mining Change History”, *IEEE Transactions on Software Engineering*, vol.30, No.9, pp.574–586, 2004.

[30] Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller: “Mining Version Histories to Guide Software Changes”, In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pp. 563–572, 2004.

[31] Ahmed E. Hassan and Richard C. Holt: “Predicting Change Propagation in Software Systems”, In *Proceedings of the 20th International Conference on Software Maintenance (ICSM'04)*, pp.284–293, 2004.

[32] Huzefa Kagdi, Shehnaaz Yusuf, and Jonathan I. Maletic: “Mining Sequences of Changed-files from Version Histories”, In *Proceedings of International Workshop on Mining Software Repositories (MSR'06)*, 2006.

[33] 楠田 泰三: “メソッドの同時更新履歴を用いたクラスの機能別分類法”, 大阪大学大学院情報科学研究科 修士論文報告会, 2006.

[34] Gerardo Canfora and Luigi Cerulo: “Impact Analysis by Mining Software and Change Request Repositories”, In *Proceedings of the 11th International Software Metrics Symposium (METRICS'05)*, 2005.

[35] 佐々木 啓，松下 誠，井上 克郎: “開発履歴情報に基づいたダイナミックコミュニティ選定支援手法”, 電子情報通信学会技術研究報告, SS2004-50, Vol.104, No.571, pp.1–6, 2005.

付録

A Latent Semantic Analysis

ここでは，提案手法が利用している LSA の概要として，その前提となるベクトル空間モデルと，LSA で用いられる特異値分解の概要を述べる．

A.1 ベクトル空間モデル

一般に文書に含まれる単語間の関連や類似度を計算し，統計学的にある種の傾向を見出すためには，それぞれの単語を何らかの数学的モデル上に変換することが必要となる．ベクトル空間モデルでは，文書内に含まれる単語とその頻度を基にして行列を作成し，その行列から文書や単語に対応するベクトルを定義する．

対象となる文書の集合を $D = \{d_1, \ldots, d_m\}$ として，d_i に含まれる単語の集合を $W(d_i)$ とする．この時全体の語彙 W は全ての $W(d_i)$ の和集合となる．

そして，c_{ij} を文書 d_i に単語 $w_j \in W$ が出現した回数として，文書 d_i に対応する文書ベクトル $\overrightarrow{d_i}$ を次の式で定義する (ただし $|W| = n$)．

$$\overrightarrow{d_i} = (c_{i1}, \ldots, c_{in})$$

このベクトルを全ての文書について計算すると，以下のような $m \times n$ の行列 A が得られる．

$$A = \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mn} \end{bmatrix}$$

ここで得られた行列 A を単語文書行列と呼ぶ．

このような方法で文書や単語をモデル化し，上記で定義した文書ベクトル $\overrightarrow{d}$ を用いて文書同士の類似度を計算する．ベクトルからの類似度の計算方法としては，一般に $\cos\theta$ の値が用いられる．2 つのベクトル $\overrightarrow{a} = (a_1, \ldots, a_n)$ と $\overrightarrow{b} = (b_1, \ldots, b_n)$ の間の $\cos\theta_{\overrightarrow{a},\overrightarrow{b}}$ は以下の式によって求められる．

$$\cos\theta_{\overrightarrow{a},\overrightarrow{b}} = \frac{\overrightarrow{a} \cdot \overrightarrow{b}}{|\overrightarrow{a}| \left|\overrightarrow{b}\right|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}}$$

角度が 0 に近い程，$\cos\theta$，すなわち類似度が高くなり，類似度の最大値は 1 となる．

A.2 特異値分解

A.1 節で述べたベクトル空間モデルでは，文書間に全く同じ単語が含まれていない限り，類似度は 0 である．そのため同一の単語がなくても関連した単語を多く含んだ文書間の類似度が適正なものとならない．

LSA では，単語文書行列 A に対して特異値分解 (Singular Value Decomposition; SVD) を利用することにより，この問題の解決を図っている．特異値分解を行うと $m \times n$ 行列 A を次のような U, S, V という 3 つの行列に一意に分解することができる．

$$A = USV^T,$$
$$l = min(m, n), U : m \times l, S : l \times l, V : n \times l$$

ここで，S は特異値行列 (singular value matrix) と呼ばれる対角行列で，その対角要素に並ぶ値を特異値と呼ぶ．特異値分解を行うと，特異値は行列 S 中で降順に並ぶ．

こうして特異値分解によって分解された行列には,「上位 r 個の特異値のみを使って USV^T を掛け合わせた結果は，元の行列 A の rank r における最小二乗誤差になる」という性質がある．すなわち，$\hat{U} : m \times r, \hat{S} : r \times r, \hat{V} : n \times r$ とした時，$\hat{A} = \hat{U}\hat{S}\hat{V}^T$ は A の rank r における最小二乗誤差である．

LSA ではこの性質を利用しており，誤差を最小に保ったまま行列の rank をあえて削減することによって，より関連の強い文書ベクトルが同一次元に縮退され，類似した値に近似されることが期待できる．そのため，従来のベクトル空間モデルでは適正な類似度が得られなかった，間接的に関連のある文書間についても高い類似度を得ることができる．

B　本文中で例として取り上げたクラスタの全内容

本節では，5 節の適用事例において，例として取り上げ，一部抜粋していたクラスタの内容の全てを掲載する．

- 5.2 節，図 7 のクラスタ

```
==== トランザクション区切り ================================================
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/mypopen.c
リビジョン: 1.8
コミット日時: 2002-01-04 21:11:50

44a45,48
> #if !defined(HAVE_SETMODE) && defined(HAVE__SETMODE)
> # define setmode _setmode
> #endif
>
52c56
< static char const rcsid[] = "$Id: mypopen.c,v 1.7 2001/07/05 15:08:21 broeker Exp $";
---
> static char const rcsid[] = "$Id: mypopen.c,v 1.8 2002/01/04 12:11:50 broeker Exp $";
62a67,73
>  /* 20020103: if file is not explicitly in Binary mode, make
>   * sure we override silly Cygwin behaviour of automatic binary
>   * mode for files in "binary mounted" paths */
> #if O_BINARY != O_TEXT
>  if (! (flag | O_BINARY))
>  flag |= O_TEXT;
> #endif
98a110,115
> #if HAVE_SETMODE
>  if (! strchr(mode, 'b')) {
>  setmode(fileno(fp), O_TEXT);
>  }
> #endif /* HAVE_SETMODE */
>
108c125,126
<  else return(NULL);
---
>  else
>  return(NULL);
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/vp.h
リビジョン: 1.4
コミット日時: 2002-01-04 21:11:50

33c33
< /* $Id: vp.h,v 1.3 2001/07/05 14:31:00 broeker Exp $ */
---
> /* $Id: vp.h,v 1.4 2002/01/04 12:11:50 broeker Exp $ */
50,60d49
< /* In view of DOS portability, we may need the vale of the O_BINARY
<  * bit mask. On Unix platforms, it's not defined, nor is it needed -->
<  * set it to a no-op value */
< #ifndef O_BINARY
< # ifdef _O_BINARY
< #  define O_BINARY _O_BINARY
< # else
< #  define O_BINARY 0x00
< # endif
< #endif
<
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/global.h
リビジョン: 1.20
コミット日時: 2002-01-04 21:11:50

33c33
< /* $Id: global.h,v 1.19 2001/10/10 16:49:22 broeker Exp $ */
---
> /* $Id: global.h,v 1.20 2002/01/04 12:11:50 broeker Exp $ */
108a109,127
> /* HBB 20020103: Need to force text or binary mode opens on Cygwins,
>  * because of their "binary/text mode mount" silliness :-( */
```

```
> #ifndef O_TEXT
> # ifdef _O_TEXT
> #  define O_TEXT _O_TEXT
> # else
> #  define O_TEXT 0x00
> # endif
> #endif
> /* Same for binary mode --- moved here from vp.h */
> #ifndef O_BINARY
> # ifdef _O_BINARY
> #  define O_BINARY _O_BINARY
> # else
> #  define O_BINARY 0x00
> # endif
> #endif
>
>
---- 変更区切り ------------------------------------------------------------
==== トランザクション区切り ==========================================
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/constants.h
リビジョン: 1.12
コミット日時: 2002-07-11 23:23:45

33c33
< /* $Id: constants.h,v 1.11 2001/08/13 15:31:22 broeker Exp $ */
---
> /* $Id: constants.h,v 1.12 2002/07/11 14:23:45 broeker Exp $ */
79,80c79
< #define READ 4 /* access(2) parameter */
< #define WRITE 2 /* access(2) parameter */
---
>
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/mypopen.c
リビジョン: 1.9
コミット日時: 2002-07-11 23:23:45

45,48d44
< #if !defined(HAVE_SETMODE) && defined(HAVE__SETMODE)
< # define setmode _setmode
< #endif
<
56c52
< static char const rcsid[] = "$Id: mypopen.c,v 1.8 2002/01/04 12:11:50 broeker Exp $";
---
> static char const rcsid[] = "$Id: mypopen.c,v 1.9 2002/07/11 14:23:45 broeker Exp $";
110c106
< #if HAVE_SETMODE
---
> #ifdef SETMODE
112c108
<  setmode(fileno(fp), O_TEXT);
---
>  SETMODE(fileno(fp), O_TEXT);
114c110
< #endif /* HAVE_SETMODE */
---
> #endif /* SETMODE */
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/global.h
リビジョン: 1.21
コミット日時: 2002-07-11 23:23:45

33c33
< /* $Id: global.h,v 1.20 2002/01/04 12:11:50 broeker Exp $ */
---
> /* $Id: global.h,v 1.21 2002/07/11 14:23:45 broeker Exp $ */
126a127,156
> #undef SETMODE
> #if O_BINARY || O_TEXT
> /* OK, looks like we are on an MSDOS-ish platform ---> define SETMODE
>  * to actually do something */
> # ifdef HAVE_SETMODE
> #  define SETMODE(fildes, mode) setmode(fildes,mode)
> # else
> #  ifdef HAVE__SETMODE
> #   define SETMODE(fildes, mode) _setmode(fildes,mode)
> #  endif
> # endif
> #endif
```

```
>
> /* access(2) parameters. Only make assumptions about their values if
>  * <unistd.h> fails to define them. */
> #ifdef R_OK
> # define READ R_OK
> #else
> # define READ 4
> #endif
> #ifdef W_OK
> # define WRITE W_OK
> #else
> # define WRITE 2
> #endif
>
> /* This can happen on only vaguely Unix-ish platforms... */
> #ifndef HAVE_LSTAT
> # define lstat(file,buf) stat(file,buf)
> #endif
---- 変更区切り ------------------------------------------------------
==== トランザクション区切り ==============================================
---- 変更区切り ------------------------------------------------------
パス名: cscope/src/mypopen.c
リビジョン: 1.11
コミット日時: 2002-08-16 01:07:07

51c51
< static char const rcsid[] = "$Id: mypopen.c,v 1.10 2002/07/28 15:40:07 broeker Exp $";
---
> static char const rcsid[] = "$Id: mypopen.c,v 1.11 2002/08/15 16:07:07 broeker Exp $";
106c106
<  if (! strchr(mode, 'b')) {
---
>  if (fp && ! strchr(mode, 'b')) {
---- 変更区切り ------------------------------------------------------
==== トランザクション区切り ==============================================
```

- 5.2 節，図 8 のクラスタ

```
==== トランザクション区切り ==============================================
---- 変更区切り ------------------------------------------------------
パス名: cscope/src/command.c
リビジョン: 1.23
コミット日時: 2004-05-01 00:24:18

48c48
< static char const rcsid[] = "$Id: command.c,v 1.22 2004/04/30 12:09:14 broeker Exp $";
---
> static char const rcsid[] = "$Id: command.c,v 1.23 2004/04/30 15:24:18 broeker Exp $";
58c58,60
< char pattern[PATLEN + 1]; /* symbol or text pattern */
---
> /* HBB 20040430: renamed to avoid lots of clashes with function arguments
>  * also named 'pattern' */
> char Pattern[PATLEN + 1]; /* symbol or text pattern */
177c179
< #if TERMINFO
---
> #ifdef KEY_ENTER
190c192
< #if TERMINFO
---
> #ifdef KEY_DOWN
191a194,195
> #endif
> #ifdef KEY_RIGHT
212c216
< #if TERMINFO
---
> #ifdef KEY_UP
213a218,219
> #endif
> #ifdef KEY_LEFT
232c238
< #if TERMINFO
---
> #ifdef KEY_HOME
263c269
< #endif
---
```

```
> #endif /* def(KEY_HOME) */
267c273
< #if TERMINFO
---
> #ifdef KEY_NPAGE
283c289
< #if TERMINFO
---
> #ifdef KEY_PPAGE
409c415
< #if TERMINFO
---
> #ifdef KEY_CLEAR
435,436c441,442
<  if (*pattern != '\0') {
<  (void) addstr(pattern);
---
>  if (*Pattern != '\0') {
>  (void) addstr(Pattern);
457c463
<  (void) strcpy(pattern, item->text);
---
>  (void) strcpy(Pattern, item->text);
470,471c476,477
<  if (mygetline(pattern, newpat, COLS - fldcolumn - 1, '\0', caseless )) {
<  strcpy (pattern, newpat);
---
>  if (mygetline(Pattern, newpat, COLS - fldcolumn - 1, '\0', caseless )) {
>  strcpy (Pattern, newpat);
508c514
<  (void) strcpy(pattern, newpat);
---
>  (void) strcpy(Pattern, newpat);
511c517
<  addcmd(field, pattern); /* add to command history */
---
>  addcmd(field, Pattern); /* add to command history */
630c636,638
<  if ((c = mygetch()) == EOF || c == ctrl('D') || c == ctrl('Z')) {
---
>  if ((c = mygetch()) == EOF
>       || c == ctrl('D')
>       || c == ctrl('Z')) {
638c646
< #if TERMINFO
---
> #ifdef KEY_NPAGE
642c650
< #if TERMINFO
---
> #ifdef KEY_PPAGE
651c659
< #if TERMINFO
---
> #ifdef KEY_CLEAR
745c753,754
<  for (s = pattern; *s != '\0'; ++s) { /* old text */
---
>  for (s = Pattern; *s != '\0'; ++s) {
>  /* old text */
---- 変更区切り ------------------------------------------------------------
==== トランザクション区切り ========================================
---- 変更区切り ------------------------------------------------------------
パス名: cscope/src/display.c
リビジョン: 1.25
コミット日時: 2004-05-01 00:29:04

60c60
< # typedef jmp_buf sigjmp_buf;
---
> typedef jmp_buf sigjmp_buf;
63c63
< static char const rcsid[] = "$Id: display.c,v 1.24 2004/02/12 18:13:14 broeker Exp $";
---
> static char const rcsid[] = "$Id: display.c,v 1.25 2004/04/30 15:29:04 broeker Exp $";
178c178
<  printw("Change \"%s\" to \"%s\"", pattern, newpat);
---
>  printw("Change \"%s\" to \"%s\"", Pattern, newpat);
181c181
<  fields[field].text2 + 1, pattern);
---
```

```
>  fields[field].text2 + 1, Pattern);
439c439
<  findresult = (*f)(pattern);
---
>  findresult = (*f)(Pattern);
445c445
<  if ((rc = findinit(pattern)) == NOERROR) {
---
>  if ((rc = findinit(Pattern)) == NOERROR) {
447c447
<  findresult = (*f)(pattern);
---
>  findresult = (*f)(Pattern);
485c485
<          findresult, pattern);
---
>          findresult, Pattern);
488c488
<          pattern);
---
>          Pattern);
491c491
<          pattern);
---
>          Pattern);
495c495
<          pattern);
---
>          Pattern);
498c498
<          fields[field].text2, pattern);
---
>          fields[field].text2, Pattern);
---- 変更区切り ------------------------------------------------------------
==== トランザクション区切り ========================================
```

- 5.3 節，図 9 のクラスタ

```
==== トランザクション区切り ========================================
---- 変更区切り ------------------------------------------------------------
パス名: SPARS/src/DB/db_common.c
リビジョン: 1.16
コミット日時: 2004-08-25 15:10:28

6c6
<  * $Id: db_common.c,v 1.15 2004/08/09 09:11:19 t-yamamt Exp $
---
>  * $Id: db_common.c,v 1.16 2004/08/25 06:10:28 t-yamamt Exp $
121a122,147
> int compare_asbigendian(DB *dbp, const DBT *a, const DBT *b) {
>     u_int8_t *ai;
>     u_int8_t *bi;
>     size_t len;
>
>     /* * Returns:
>      * < 0 if a < b
>      * = 0 if a = b
>      * > 0 if a > b
>      */
>
>     if (a->size < b->size)
>  return -1;
>     if (a->size > b->size)
>  return 1;
>
>     ai = a->data + a->size - 1;
>     bi = b->data + b->size - 1;
>     for (len = a->size; len--; --ai, --bi) {
>  if (*ai != *bi) {
>      return *ai - *bi;
>  }
>     }
>     return 0;
> }
>
144a171,177
>     if (db->dbtype == DB_BTREE) {
>  if ((ret = (dbp->set_bt_compare)(dbp, compare_asbigendian)) != 0) {
>      dbenv->err(dbenv, ret, "set_flags(%s)", db->dbname);
```

```
>       close_exit(ret);
>   }
>     }
>
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/version.c
リビジョン: 1.6
コミット日時: 2004-08-25 15:10:28

6c6
<   * $Id: version.c,v 1.5 2004/08/10 01:59:07 t-yamamt Exp $
---
>   * $Id: version.c,v 1.6 2004/08/25 06:10:28 t-yamamt Exp $
16c16
< static const int version = 4;
---
> static const int version = 5;
---- 変更区切り ------------------------------------------------------
==== トランザクション区切り ============================================
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/db_common.c
リビジョン: 1.18
コミット日時: 2004-12-14 16:08:58

6c6
<   * $Id: db_common.c,v 1.17 2004/09/01 10:41:13 m-itii Exp $
---
>   * $Id: db_common.c,v 1.18 2004/12/14 07:08:58 t-yamamt Exp $
138,139c138,139
<     ai = a->data + a->size - 1;
<     bi = b->data + b->size - 1;
---
>     ai = (u_int8_t)a->data + a->size - 1;
>     bi = (u_int8_t)b->data + b->size - 1;
---- 変更区切り ------------------------------------------------------
==== トランザクション区切り ============================================
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/db_common.c
リビジョン: 1.19
コミット日時: 2004-12-14 16:18:28

6c6
<   * $Id: db_common.c,v 1.18 2004/12/14 07:08:58 t-yamamt Exp $
---
>   * $Id: db_common.c,v 1.19 2004/12/14 07:18:28 t-yamamt Exp $
138,139c138,139
<     ai = (u_int8_t)a->data + a->size - 1;
<     bi = (u_int8_t)b->data + b->size - 1;
---
>     ai = (u_int8_t *)a->data + a->size - 1;
>     bi = (u_int8_t *)b->data + b->size - 1;
---- 変更区切り ------------------------------------------------------
==== トランザクション区切り ============================================
```

- 5.3 節, 図 10 のクラスタ

```
==== トランザクション区切り ============================================
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/db_fileinfo.c
リビジョン: 1.2
コミット日時: 2003-04-15 14:11:35

6c6
<   * $Id: db_fileinfo.c,v 1.1 2003/04/10 04:35:24 t-yamamt Exp $
---
>   * $Id: db_fileinfo.c,v 1.2 2003/04/15 05:11:35 t-yamamt Exp $
16a17
> #include <time.h>
34c35
< int put_fileinforepository(FID_t id, char *info) {
---
> int put_fileinforepository(FID_t id, time_t mtime, char *info) {
42a44
>     filerepo->mtime = mtime;
55c57
< int get_fileinforepository(FID_t id, char **info) {
---
> int get_fileinforepository(FID_t id, time_t *mtime, char **info) {
66a69
```

```
>     *mtime = data->mtime;
93c96
<  printf("Key "__FID_FORMAT__"\n\tFile_ID "__FID_FORMAT__", info %s\n",
---
>  printf("Key "__FID_FORMAT__"\n\tFile_ID "__FID_FORMAT__", mtime %s, info %s\n",
95a99
>         ctime(&((FileInfoRepository *)data.data)->mtime),
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/main.c
リビジョン: 1.3
コミット日時: 2003-04-15 14:11:35

6c6
<  * $Id: main.c,v 1.2 2003/03/11 10:34:31 t-yamamt Exp $
---
>  * $Id: main.c,v 1.3 2003/04/15 05:11:35 t-yamamt Exp $
27a28
>     int fileinfo;
57a59
>     (void)fprintf(stderr, "  -J  File Information\n");
97c99
<     while ((ch = getopt(argc, argv, "ihrf:ACDEFGIMOPRSTUWXZ:")) != -1) {
---
>     while ((ch = getopt(argc, argv, "ihrf:ACDEFGIJMOPRSTUWXZ:")) != -1) {
119a122,124
>  case 'J':
>      repo.fileinfo = 1;
>      break;
261a267,274
>     if (repo->fileinfo || repo->all) {
>  printf("File Information Repository\n");
>  open_readfileinforepository();
>  print_fileinforepository();
>  close_fileinforepository();
>  printf("\n");
>     }
>
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/spars_db.h
リビジョン: 1.5
コミット日時: 2003-04-15 14:11:35

6c6
<  * $Id: spars_db.h,v 1.4 2003/04/10 04:35:24 t-yamamt Exp $
---
>  * $Id: spars_db.h,v 1.5 2003/04/15 05:11:35 t-yamamt Exp $
122c122
<  * ファイル ID -> (コメント)
---
>  * ファイル ID -> (更新時間, コメント)
126a127
>     time_t mtime;
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/DB/spars_global.h
リビジョン: 1.7
コミット日時: 2003-04-15 14:11:35

6c6
<  * $Id: spars_global.h,v 1.6 2003/04/10 04:35:24 t-yamamt Exp $
---
>  * $Id: spars_global.h,v 1.7 2003/04/15 05:11:35 t-yamamt Exp $
335,336c335,336
< int put_fileinforepository(FID_t id, char *info);
< int get_fileinforepository(FID_t id, char **info);
---
> int put_fileinforepository(FID_t id, time_t mitme, char *info);
> int get_fileinforepository(FID_t id, time_t *mtime, char **info);
---- 変更区切り ------------------------------------------------------
パス名: SPARS/src/Register/regist.c
リビジョン: 1.4
コミット日時: 2003-04-15 14:11:39

6c6
<  * $Id: regist.c,v 1.3 2003/04/12 08:05:04 t-yamamt Exp $
---
>  * $Id: regist.c,v 1.4 2003/04/15 05:11:39 t-yamamt Exp $
22c22
< void regist(char *path) {
---
> void regist(char *path, time_t mtime) {
50c50
```

```
<     put_fileinforepository(id, description);
---
>     put_fileinforepository(id, mtime, description);
---- 変更区切り ------------------------------------------------------------
パス名: SPARS/src/Register/regist.h
リビジョン: 1.4
コミット日時: 2003-04-15 14:11:39

6c6
<  * $Id: regist.h,v 1.3 2003/04/12 08:05:04 t-yamamt Exp $
---
>  * $Id: regist.h,v 1.4 2003/04/15 05:11:39 t-yamamt Exp $
23c23
< extern void regist(char *path);
---
> extern void regist(char *path, time_t mtime);
---- 変更区切り ------------------------------------------------------------
パス名: SPARS/src/Register/traverse.c
リビジョン: 1.2
コミット日時: 2003-04-15 14:11:39

6c6
<  * $Id: traverse.c,v 1.1 2003/03/03 08:22:35 t-yamamt Exp $
---
>  * $Id: traverse.c,v 1.2 2003/04/15 05:11:39 t-yamamt Exp $
91c91,92
<  regist(name);
---
>  lstat(name, &statbuf);
>  regist(name, statbuf.st_mtime);
117c118
<      regist(pathname);
---
>      regist(pathname, statbuf.st_mtime);
151a153
>     struct stat statbuf;
171c173,174
<      regist(p->fts_path);
---
>      lstat(p->fts_path, &statbuf);
>      regist(p->fts_path, statbuf.st_mtime);
---- 変更区切り ------------------------------------------------------------
==== トランザクション区切り ============================================
---- 変更区切り ------------------------------------------------------------
パス名: SPARS/src/Search/perform_search.c
リビジョン: 1.5
コミット日時: 2003-04-15 16:09:05

1c1
< /* $Id: perform_search.c,v 1.4 2003/04/10 07:49:37 y-takao Exp $ */
---
> /* $Id: perform_search.c,v 1.5 2003/04/15 07:09:05 y-takao Exp $ */
265c265
<     get_fileinforepository(ci->file_id, &(ci->fileinfo));
---
>     get_fileinforepository(ci->file_id, &(ci->mtime), &(ci->fileinfo));
---- 変更区切り ------------------------------------------------------------
パス名: SPARS/src/Search/spars_search.h
リビジョン: 1.4
コミット日時: 2003-04-15 16:09:06

1c1
< /* $Id: spars_search.h,v 1.3 2003/04/10 07:49:37 y-takao Exp $ */
---
> /* $Id: spars_search.h,v 1.4 2003/04/15 07:09:06 y-takao Exp $ */
31a32
>     time_t mtime;     // 最終更新時間
---- 変更区切り ------------------------------------------------------------
==== トランザクション区切り ============================================
```