人とコンピュータによる「協調型情報検索」のための特徴抽出手法に関する研究

大野, 麻子

(Degree)
博士（学術）

(Date of Degree)
2009-03-25

(Date of Publication)
2011-11-11

(Resource Type)
doctoral thesis

(Report Number)
甲4619

(URL)
https://hdl.handle.net/20.500.14094/D1004619

博　士　論　文

人とコンピュータによる「協調型情報検索」のための特徴抽出手法に関する研究

平成 21 年 3 月

神戸大学大学院総合人間科学研究科

大 野　麻 子

目次

図目次

表目次

第 1 章

緒論

1.1 本研究の概要

本研究の背景

「情報化社会」と言われて久しい現代社会において，コンピュータの果たす役割は大きく，もはや日常生活から切り離すことは出来ない程にまで人々の生活に浸透している．コンピュータに依存した生活形態は既に人間の社会活動のあらゆる場面における前提条件となりつつあると言っても過言ではない．

しかしコンピュータが人の意図する通りのサービスを常に提供しているかといえば，実際には人の要求をそのまま実現することは困難である場合が多い．これは，コンピュータを用いて行う全てのタスクにおいて，程度の差こそあれ人がその思考や処理手順をコンピュータに合わせる必要があり，それが，人が自らの意図をコンピュータに的確に伝え，思い通りの処理結果を得ることを困難にしているためである．

一般的でないタスク，つまり，予め用意された一般化された形をとらないタスクを実行しようとした途端に，ユーザの目の前に必要とされる前提知識が山積されるという現実が，実際にはコンピュータを活用することで容易に実行できるはずの多くのタスクを放棄させる要因となっている．これが人とコンピュータの間の大きな隔たりとして存在しており，個々の意図に沿った思い通りの処理を実現できないという状況を生み出している．結果として我々の多くは知らず知らずのうちにコンピュータによって提供される「まあまあの結果」を受け入れているのである．それどころか，はじめから本来の意図を放棄し，「まあまあの要求」をコンピュータに投げかけ続けているのである．

「ユビキタス・コンピューティング」や「バーチャル・リアリティ」など，人とコン

ピュータの距離を縮め，隔たりをなくすというようなニーズは以前から存在し，これを実現するような技術が次々と生み出されている．しかし，もっと根本的で潜在的に存在し続ける 2 つのニーズ，すなわち，「人がコンピュータに自分の意図を的確に伝える」こと，そして「コンピュータが人の意図を適切に定量化し演算に用いる」ことの 2 つが実現されない限り，人がコンピュータに従属する形でコンピュータを活用しているという現状が変わることはなく，我々はこれからも「まあまあ」な現実を甘受し続ける他ないであろう．

本研究の目的と内容

本研究では，人が自分の意図する通りの要求をコンピュータに対し専門知識無く伝えられるようなしくみをソフトウェア的に実現することを目指している．そのインタフェースとして，多くの人々が接点を持ち，用途も多岐に渡る情報検索を研究対象とした．

現代社会において，必要な情報を早く効率的に検索する技術が高い経済的価値を持つことは明らかである．個人レベルで見ても，我々は日々の生活の中で膨大な情報に自由にアクセスし様々な用途に活用することで，情報化社会の恩恵を大いに享受している．しかしここでも，人の意図を的確にコンピュータに伝え，適切に定量化し演算に用いるしくみがなければ，人はしばしば必要以上の時間を費やしたり，処理結果について妥協を余儀なくされることとなる．

このような問題に対して，本研究において追求するものは，人とコンピュータによる「協調型情報検索」である．本研究では情報検索を「人とコンピュータのコミュニケーション・インタフェース」と位置づけ，人の意図や好みに合った情報検索を難解な操作や専門知識を要求することなく実現することを目指している．

「協調型情報検索」とは，「人がコンピュータを使って便利に行う情報検索」の先にある，「人とコンピュータの協調により生み出される新たな知によって，より豊かに行われる情報検索」を意味する．ユーザがその意図を正しくシステムに伝え，これをもとにシステムが新たな類似性尺度を生成し，その処理結果をユーザが評価する．ユーザの意図をシステムに学習させるだけではなく，ユーザもまたシステムの出力をもとに学習し，その意図を再形成する．このような人とコンピュータの双方向のコミュニケーションにより，人に新たな知の発見をもたらし，これをもとに人にも学習を促すという新しい情報検索の形を「協調型情報検索」と定義する．

このような情報検索の実現のためには，次の 2 つが必要であると考えられる．

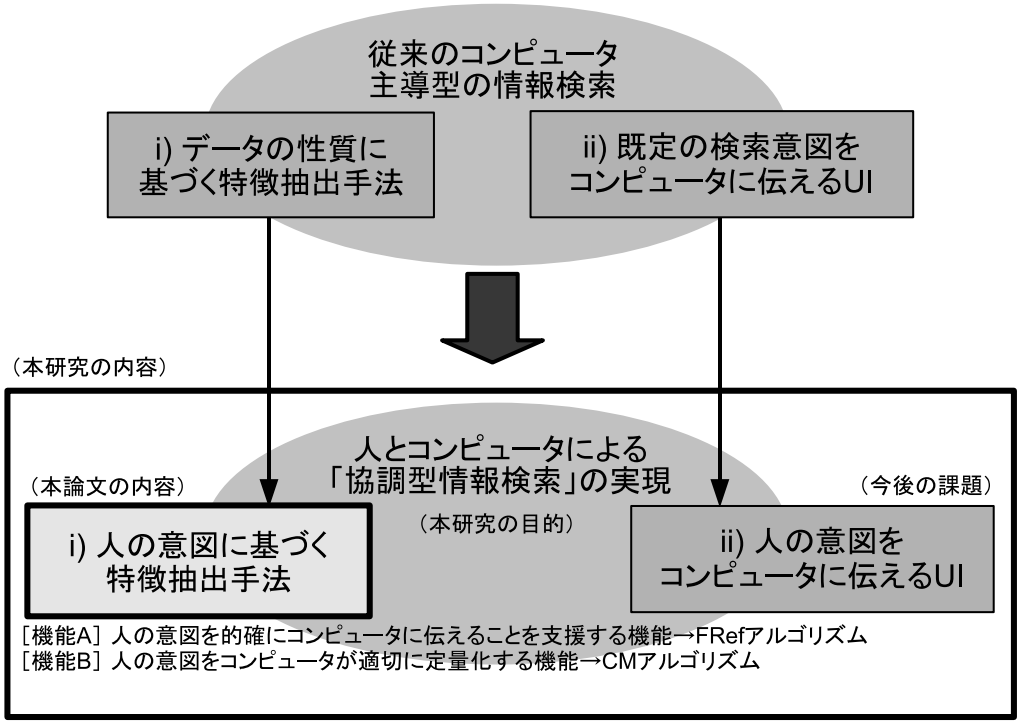


図 1.1　人とコンピュータによる「協調型情報検索」の実現に必要な 2 つの要素

i) **人の意図に基づく特徴抽出手法**

ii) **人の意図を簡易な操作でシステムに伝えるための UI（ユーザ・インタフェース）**

本研究ではこのうち i) にあげられる「人の意図に基づく特徴抽出手法」の開発を行ってきた．これは，対象となるデータそのものではなく特徴抽出を行う人の意図に注目した特徴抽出手法であり，次のいずれかもしくは両方の機能を持つ．

［機能 A］**人の意図を的確にコンピュータに伝えることを支援する機能**

［機能 B］**人の意図をコンピュータが適切に定量化する機能**

図 1.1 は本研究において目指される，人とコンピュータによる「協調型情報検索」の実現に必要な要素についてまとめたものである．

既存研究では，データの性質や構成を詳細に解析し，これをもとに抽出した特徴を用いて既定の検索意図に基づき情報検索を行うようなシステムが数多く開発されてきた．しかし，いかに優れたシステムが開発されたとしても，これを使用するユーザが個々の検索意図に応じて適切なシステムを選択する必要があることに変わりはない．そのためにはユー

ザが検索対象となるデータやシステムの特性について熟知していることが要求されるが，これは一般に困難である．一方で，2.1 にその例を挙げるように，潜在的意味などを用いてユーザの意図に基づく特徴抽出を行う手法も提案されている．これらは本研究と同じく［機能 B］の実現を目指すものであると考えられる．しかし［機能 A］をシステム側に固定で行っている以上，先に述べたユーザに前知識を要求するという問題は解決されない．

本研究では，［機能 A］をシステム側で完結させるのではなく，その一部をユーザ側に委ねることで，個々のユーザが自らその検索意図に基づき，個別の尺度により特徴抽出を行うことを可能とするような特徴抽出手法の開発を行っている．これが 3.1 で説明する「参照を用いた一次元データの特徴抽出手法（**FRef アルゴリズム**）」である．FRef アルゴリズムは，人がデータを比較する際の特徴表現の近似を別の手続きによって得ようとする手法である．特徴抽出の基準となる類似性尺度をユーザ側で変更可能とすることで［機能 A］の実現を目指す．また，本研究では［機能 B］により人の意図とシステムの算出した特徴量との間のセマンティック・ギャップを解消するような特徴表現の開発を行っている．これが 3.2 で説明する「記述スタイルに基づく一次元データの特徴抽出手法（**CM アルゴリズム**）」である．CM アルゴリズムは，人がデータから特徴を抽出する手続きそのものに倣った手法である．これにより，人の意図する特徴を抽出し定量化すること，すなわち［機能 B］の実現を目指す．

図 1.2 はデータから特徴を抽出し，検索を行うまでのプロセスを簡略化して表したものである．この流れを追いながら先に述べた 2 つの手法の位置づけについて説明を行う．まず，図中 (1) に示すように，対象となるデータから何らかの類似性尺度に基づき特徴を抽出する．これを人が自ら行う場合，常に既定のデータ分析を行って特徴を抽出するのではなく，毎回の検索意図に基づきその都度規定した類似性尺度に基づき特徴抽出を行う．例えば，画像データが対象である場合，「似たような色使いの画像を探したい」という意図（意図 A）に基づく類似性尺度 A により画像の持つ膨大な特徴の中から特定の特徴を抽出する．このように，データそのものの成分に注目するのではなく，人がその意図に基づいて行う特徴抽出の手続きに倣った手法が CM アルゴリズムである．

次に，図中 (2) に示すように，抽出した特徴をもとにデータ間の類似性を検出し，最終的な検索順位を決定する．この図のようにクエリとしてデータが与えられた場合，そのデータに対し最も高い類似度を示したデータが最も類似したデータであるといえる．既に述べたように，(1) で示した特徴抽出における類似性尺度は，検索意図や好みなどにより

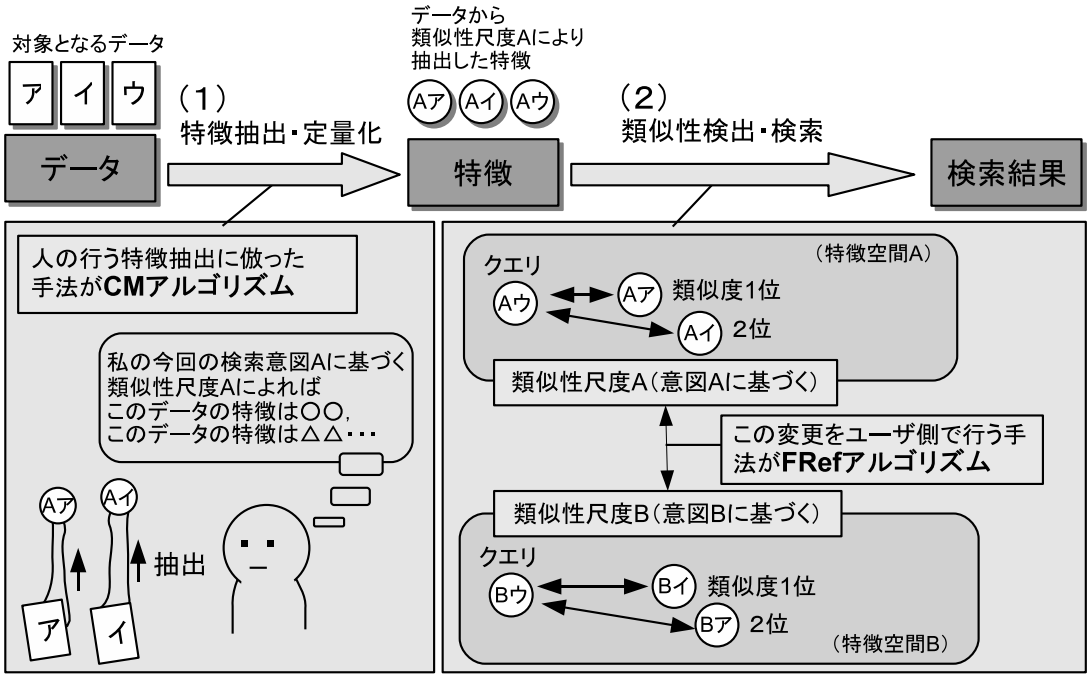


図 1.2　FRef アルゴリズムと CM アルゴリズムの位置づけ

異なりうる．例えば同じ画像データが対象であっても，「写実的な画風の画像を探したい」という別の意図（意図 B）に基づき行われる検索であれば，類似性尺度 A とは別の類似性尺度 B により行われねばならない．このような検索意図の変更に対応するため，本研究では，特徴抽出における類似性尺度をユーザ側で変更可能とする手法を開発した．これが FRef アルゴリズムである．

本研究ではこれらの特徴抽出手法を，社会で解決が求められている問題に対し適用した．具体的には，長期的なニーズが期待でき，かつ多様な検索手法が求められているソースコードと画像データに適用し，それぞれ独立した類似性検出手法としてまとめた．ソースコードはそれ自体が人間の意図をコンピュータに伝えるためのコミュニケーション・ツールであり，自然言語文書と異なり，その記述そのものよりもコンピュータ上で実行される処理内容や手順が大きな意味を持つため，一般的なテキスト検索ではなくソースコードの特性に合った検索手法が求められる．画像検索の主流は内容検索であるが，近年は特にユーザの意図に沿った類似画像を検索する手法へのニーズが高まっている．本論文では，2.2 にてソースコードおよび画像間の類似性検索の背景について述べる．

1.2 本論文の構成

本論文では，人とコンピュータによる「協調型情報検索」の実現を目指し開発した 2 つの特徴抽出手法とその適用について論じる．

本論文の構成は，以下の通りである．まず，第 2 章において本研究の背景である情報検索の現状と課題について，そして社会で解決が望まれているソースコード間類似性検出および画像間類似性検出という 2 つの問題に対し，本研究がどのようなアプローチで解決を試みるのかについて述べる．次に，第 3 章では本研究において提案する 2 つの特徴抽出手法，すなわち，「参照を用いた一次元データの特徴抽出手法 (FRef アルゴリズム)」および「記述スタイルに基づく一次元データの特徴抽出手法（CM アルゴリズム)」について説明する．第 4 章では FRef アルゴリズムの適用例として，ソースコード間類似性検出手法および画像間類似性検出手法を提案し，それぞれについてアルゴリズムを実装したツールによる実験を行い，その有効性を評価する．続く第 5 章では CM アルゴリズムの適用例として，授業課題ソースコードにおける盗用発見手法を提案し，アルゴリズムを実装したツールによる評価実験を行う．最後に，第 6 章において，本研究全体の総括を行い，今後の課題について述べる．

第 2 章

研究の背景

2.1 情報検索と特徴抽出

ここでは，情報検索の概要と，既存の代表的な特徴抽出手法について述べる．

情報検索とは，膨大なデータの中からユーザの所望する情報を何らかの基準で検出することである．ここでデータとは様々な形態をなす情報の電子的な媒体を指す．Google などのオンライン検索サービスに代表されるように，情報検索はもはや我々の生活においてなくてはならない存在となっている．Google で検索することを英語で “google（動詞）” と言ったり日本語で “ググる” と表現したりすることからも，その浸透の程が伺える．情報検索の歴史について，その起源を特定することは難しいが，少なくとも “Information Retrieval（情報検索）” という言葉の起源は 1948 年に Calvin Mooers が自身の修士論文の中で定義し，1950 年に発表した論文で用いたことに端を発するといわれている [1].

データはただ蓄積されているだけでは意味がなく，必要な時に必要なデータが得られてこそ，その価値があると言っても過言ではない．情報検索は，大まかに言うと次のようなステップで行われる．

1) 特徴抽出
2) 分類（索引付け）
3) 類似性検出
4) 結果の提示
5) （必要に応じ）ユーザの評価，再分類

つまり，データの持つ情報をユーザの意図に基づく何らかの基準により 1) 抽出，2) 分類

し，3) ユーザの検索意図をその基準に対応する表現に変換して照合し，4) 結果を提示するというシステムが必要となる．検索システムによっては，5) 検索結果をユーザが評価し，システムがユーザの求める基準を学習するまで 1)〜4) の操作を繰り返す場合もある．この各ステップにおいて，それぞれ独立した研究課題として古くから様々な試みが行われてきた．1970 年に笹森 [2] によって概観された情報検索研究の動向では，文書内における単語の共起などの統計データをもとに作成したシソーラスを活用した索引付けなど，内容分析の自動化に注目が集まっており，情報検索のオンライン化に伴う UI の操作性向上や構文解析による対話形式の UI の実現など，現在でも研究の対象となる内容が散見される．また，ユーザによる検索結果の評価とシステム側の評価にはズレがあることについても，既にこの時点で指摘されている．

文書データの検索には，キーワード検索が用いられるのが一般的である．単純なキーワード検索では，データベースに含まれる全ての文書データを文書－キーワード行列で表現し，ユーザがクエリとして与えたキーワードの含まれる文書が提示される．ここで特徴ベクトル空間上の軸は単語であり，文書は単語の集合として捉えられる．特徴ベクトル空間上の座標として表すことにより，内積やコサイン尺度，ユークリッド距離など，簡易な計算で類似度を求めることが可能となる．しかしこのような単純なシステムには次のような問題がある．まず，(1) 文書に含まれる単語のバリエーションが増えるに従い，それらの特徴を表すベクトルの次元が増大し，計算量が爆発してしまうという問題，そして (2) 単純なキーワード同士のマッチングのみではユーザの意図する検索結果を得ることは難しいという問題である．また，(3) ユーザが検索対象を端的に表すキーワードを生成できるとは限らない．つまり，ユーザの知識や表現力によって検索結果が大きく変わってしまうという問題もある．

(1) の問題の解決方策として，潜在的意味解析（Latent Semantic Analysis：LSA）[3] による次元削減が挙げられる．これは，文書－キーワード行列で表現される文書ベクトル $\boldsymbol{D}$ を特異値分解し，得られた低次元の直交ベクトルを文書ベクトルとして利用するというものである．文書は単語を軸とする特徴ベクトル空間ではなく，LSA により実現される潜在的意味空間によって表される．単語の共起に基づき軸を合成することにより，潜在的意味空間を構成する．LSA は目的ベースで見れば主成分分析の一つともいえるが，この他に共分散行列に主成分分析を行うことで，多次元データのばらつきを保持しながら低次元空間へ射影するという方法もある．具体的には，$\boldsymbol{D}$ のキーワード間の共起情報を保

持する共分散行列を固有値分解する．共分散行列の主成分分析を用いることで，類似したデータ同士は近くに，類似していないデータ同士は遠くに配置するような次元削減が行えるという点では，後者の方が情報検索に向いているといわれている [4].

(2) に対する解法は，同時に (3) についての解法でもある場合が多い．文書中に出現する単語と，ユーザが提示した検索キーワードとの単純なマッチングを行った場合の問題点は次の 2 つである．それは，検索キーワードとして使用された単語が全く異なる意図で使用されている文書も適合文書に含められるため，検索結果に膨大なノイズが含まれるおそれがあること，そしてユーザが提示したキーワードが適切でない場合，意図する検索結果が得られないということである．しかしここでは，ユーザにより提示されたキーワードとデータベースに貯蔵された文書に対し何らかの意味的なつながりによる分類を行うことで，この問題を解消する方策について挙げる．まずユーザから提示されたキーワードに対し，シソーラスを用いて揺らぎを持たせたクエリを自動生成する方法が考えられる．この他に，概念検索という，文書の表面に現れる単語ではなくその背後に存在する概念を抽出し，その類似をもとに検索を行う方法がある．つまり，特徴ベクトル空間の軸が単語から概念に変わるということである．ここで，先に挙げた潜在的意味による表現との違いは，前者は単語の共起に基づく意味抽出であり，概念そのものを考慮するものではないが，後者は文書からその背後に存在する概念を推測している点である．川前ら [5] は，文書に含まれる単語から概念を推測することにより，単語空間から概念空間への射影を試みた．同じ概念を背後にもつ文書には同じ単語が含まれる可能性が高いという仮説から，単語からの概念抽出を行い，高精度の検索結果を得ている．しかし著者も述べているように，この手法は対象となる文書の性質に大きく影響されると考えられる．つまり，技術書や論文など，ある程度用語が統一されるような文書においては単語ゆらぎによるノイズは少ないと考えられるが，独特な比喩表現を多用する文書や個人のブログなど，著しく著者の個性が前面に出るような文書の場合，例え同じ概念を背後に持つ文書であっても前者に比べ単語の共起は少なく，また，同じ概念を持つとされた単語が必ずしもその目的で使用されるとも限らない．しかし，概念検索は自由記述文を用いた検索を容易にすることから，適切なキーワードを想起することが難しいビジネスモデルの検索や，特許検索 [6]，漠然とした知識から行う図書検索などに活用されている．

(3) は特徴抽出の問題というより，ユーザ・インタフェースの問題といえるかもしれない．大橋ら [7] は，ユーザの提示したクエリ画像からシステムが画像解析により抽出した

特徴では，ユーザがそのクエリ画像を通してシステムに伝えたかった意図を十分に表現できないとし，ユーザが作成したスケッチをクエリとして使用する手法を提案した．この手法では，予め画像データベースに含まれる画像に対し複数の人間が作成したスケッチを関連付けておく．これにより，検索はユーザが作成したスケッチとデータベースに保存されていたスケッチの間の類似度比較により行われ，検索結果に対しユーザが与えた評価をもとに関連付けの更新が行われる．この場合少なくとも，データベースに保存するスケッチを作成する人間と検索を行うユーザとの間に共通の認識があること，そしてそれを類似した形状のスケッチとして表すことが求められる．中島ら [8] は，ユーザにクエリの提示を求めるのではなく，システムから提示されたサンプル集合の中からユーザの求めるデータに最も近いものを選ばせ，このデータとサンプル集合の相対的関係を用いて新たなサンプル集合を提示することを繰り返す，という作業によりクエリを自動生成し，ユーザの意図に沿った検索を実現するという手法を提案した．ユーザ自らクエリを作成する必要がないため，未知のデータに対しても検索を行うことができるが，クエリの生成の精度はデータベース内のデータのばらつきに依存しており，また，ユーザは毎回の検索においてクエリ生成のために何度もデータ選択を行う必要があるという問題点がある．

2.2 本研究が対象とする問題

ここでは，本研究で提案する特徴抽出手法の適用により解決を目指す諸問題と，それらに対し本研究のとるアプローチについて概説する．個々の問題の詳細については，第 4 章および第 5 章においてそれぞれ説明を行う．

現在，本研究において実現を目指している「協調型情報検索」の主たる検索対象はソースコードである．ソースコード間の類似性検出は，産業面ではプログラム開発，管理，保守において有用であり，教育面ではプログラミング授業における採点支援や盗用発見，学習用開発環境のプログラム作成支援機能などに活用されている．ソースコード間の類似性検出アルゴリズムの多くはゲノム分野における遺伝子構造の推定とも深く関連しており，テキスト解析にとどまらない幅広い応用が期待できる分野であるといえる．これまでに多くの研究成果が発表されているが，いくつかの解決すべき問題が残されている．

まず，既存研究ではそれぞれの研究において個別に用意されたソースコード集合に対し，独自の類似性尺度に基づいた特徴抽出を行っているため，単純にそれらの精度を比較

することは難しいという問題が挙げられる．また，ユーザはソースコードの持つどのような特徴に着目した類似性検出を行いたいのかによってツールを選択する必要があるが，これはソースコードに関する知識の十分でないユーザにとっては困難であると考えられる．さらに，既存手法に多く見られるソースコードの構造や意味に基づく特徴を抽出し比較する手法は，プログラミング授業における課題ソースコードの様な，同じ意図を持って作成されるソースコード間で類似性検出を行った場合，偶然の類似を盗用と誤判定する恐れがあることが指摘されている [9]．

本研究では，このような諸問題を解決する 2 つの類似性検出手法を提案する．これらはそれぞれ異なるアプローチによりソースコードから特徴を抽出し，類似性検出を行う．

1 つ目のアプローチは，類似性を検出したい対象ソースコード以外に参照ソースコードと呼ばれる第三者のソースコードを用意し，これを用いて対象ソースコードの特徴を表現するというものである．対象ソースコードの特徴は参照ソースコードとの類似に基づいた相対的な尺度により表されるため，異なる参照ソースコードを用いることにより，ユーザは恣意的に類似性尺度を変更することができる．何をもってソースコードの「類似」と成すかは，類似性検出を行うユーザの意図により様々であると考えられる．本手法では，ユーザに高度な知識を要求することなく，適切な参照ソースコードの選択を行うという簡易な操作により，ユーザが意図する様々な特徴に着目した類似性検出を単一のシステムで実現することが期待できる．また，ソースコード長によらず高速な類似性検出が行えるため，個人レベルの小規模なリポジトリから，大規模なソフトウェアリポジトリまで，幅広い適用が考えられる．

また，このような特徴をもつ本手法の適用は，ソースコードにとどまらない．現在，情報の電子化が進み，様々な情報リソースがデジタルデータとしてデータベースに蓄積され，活用されている．この様な情報資源を効率的に管理・利用するため，様々な検索・分類手法が提案されているが，そこで用いられている特徴量抽出手法の多くは，前途のソースコードの例と同じく，既定の類似性尺度に基づき特徴抽出を行うため，予め想定されたデータ集合に対してのみ高い有効性を示すものであったり，ユーザに対し高度な知識を要求するものである場合が多い．本手法は，現在主流となっているデータ解析に基づく特徴抽出手法とは異なり，参照データが適切に選択されればどのようなデータに対しても適用が可能であるという汎用性の高さと，特徴ベクトル空間の座標軸をユーザ側で変更可能であるという柔軟性の高さを最大の特徴としている．

この，ユーザ側で特徴抽出における類似性尺度を変更可能であるという特性により，近年問題となっている画像解析に基づく手法における類似とユーザの意図する類似との間のセマンティック・ギャップの解消も期待できる．本手法は，対象データの特徴を，参照データと対象データの類似度を表す参照ベクトルにより表す．これにより複雑な計算を行うことなく，対象データの多面的な特徴を低次元の特徴ベクトルで表現することが可能となっている．また，対象となるデータの種類により個別の特徴抽出法を選択する必要がないため，専門的知識のないユーザにも容易に利用可能であるという利点がある．

2 つ目のアプローチは，ソースコードそのものの特徴ではなく，ソースコードを記述する際に現れる，作成者の記述スタイルを特徴として抽出し，モデルに学習させるというものである．この手法では，ソースコードの構造や意味のようなソースコードのアルゴリズム的な特徴ではなく，作成者の記述スタイルという表面的な特徴を抽出するため，プログラミング授業における課題ソースコードのような，ソースコード長が短くアルゴリズム的に類似した複数個のソースコードにおける盗用発見に適している．

既存研究においてクリアすべき課題の 1 つである誤判定問題については，現在主流となっているアルゴリズム的な特徴抽出ではなく，むしろこれまでノイズとして活用されなかったソースコードの表面的な特徴に着目すべきであると考えられる．表面的な特徴といっても，単に字句解析によるマッチングを行うのではなく，人が記述した複数のソースコードにわずかに見られる共通の傾向，つまり表記上の癖をモデルに学習させ，確率モデルの構造とパラメータにより表現するという立場をとっている．具体的には，作成者の記述スタイルを隠れマルコフモデルをベースとした記述スタイルモデルにより表現することにより，ソースコードにおける，ソースコードの内容には直接関与しない表面的な特徴を定量化する手法を開発した．

本特徴抽出手法はソースコード間の類似性検出手法としての活用も考慮されてはいるものの，その主たる適用として想定されるのは作成者認識機能を有するシステムへの実装であり，ソースコード盗用問題の解決を目指すものである．ソースコード盗用発見という目的においては，ソースコードそのものの類似を検出するよりも，同一の作成者により作成されたことが明示できるような特徴を抽出することが有効であると考えられる．

第 3 章

提案手法

3.1 参照を用いた一次元データの特徴抽出手法：FRef アルゴリズム

概要

本章では，本研究において提案する特徴抽出手法の一つである **FRef アルゴリズム**[10]について説明する．既存の多くの手法では対象となるデータから直接特徴抽出を行うのに対し，本手法では複数の “参照（Reference）” と呼ばれる第三者のデータを用いて対象となるデータの特徴を表現する．

本手法を用いて 2 つの対象データにおける類似性検出を行う手順は次の通りである．

1) **［参照データの選出］**類似性検出を行う対象となるデータの中から複数のデータを参照データとして選出する．
2) **［データの一次元配列化と標準化］**参照データを含む全ての対象データをそれぞれ一次元配列に変換し，その要素を標準化する．
3) **［マトリクスの生成］**参照データから変換した一次元配列と対象データから変換した一次元配列からマトリクスを生成する．
4) **［テクスチャ特徴量の計算］**マトリクスの特徴をテクスチャ特徴量により定量化し，対象データの特徴を表す参照ベクトルを作成する．
5) **［類似度の計算］**このようにして作成された 2 つの参照ベクトル間のユークリッド距離を計算することにより，対応する 2 つの対象データの類似性検出を行う．

本手法では，対象データに対し詳細な解析に基づく特徴抽出を行う必要がなく，単に参照

データとの相対的な関係をもとに特徴表現を行うため，一次元データとみなせるどのようなデータにも容易に実装可能である．

参照データの選出

本手法を用いて類似性検出を行う場合，その目的に応じて参照データの選出法を選択する必要がある．

i) 未知のデータに対し類似性検出を行う場合，または単に高速性を優先する場合：
対象データの中からランダムに選出する．このようにして選出された参照データは母集団（対象データ）の性質を表す代表値と考えることができる．

ii) ユーザの意図に基づく類似性検出を行う場合：
この様な場合，ユーザの意図する類似性尺度を参照データにより表現する必要がある．現段階では，選出した参照データによりどの特徴が定量化されるかについての詳細は解明されていないが，本アルゴリズムを適用する前処理として何らかの方法でデータ集合をクラスタリングしておき，そこから代表として 1 つずつ選出したデータを参照データとすることなどが考えられる．例えば，画像データを対象とするのであれば，画像集合を “風景”，“人物”，“図形”，· · · などのようにクラスタリングしておき，各クラスタより代表データを選出し，参照データとして用いる．

データの一次元配列化と標準化

全ての対象データを一次元配列に変換する．例えばテキストデータを対象とする場合は，テキストを字句解析によりトークン（字句）単位で区切り，一次元配列であるトークン列へと変換する．この場合，次に行うマトリクス生成はトークンの共起に基づき行われるため，共通の意味を持つトークンを同一の表記に変換するなどの標準化を行う．

画像データや音声データについては，ピクセル値や量子化された周波数成分を一次元配列の要素とする．このように，配列の要素が二値ではなく多段階の値をとるときは，次のマトリクス生成のステップで要素間の差分を計算するため標準化は行わなくともよい．

マトリクスの生成

参照データから変換した一次元配列と対象データから変換した一次元配列を縦横に並べ，マトリクスを生成する．マトリクスの要素はそれぞれの配列の要素の共起，もしくは

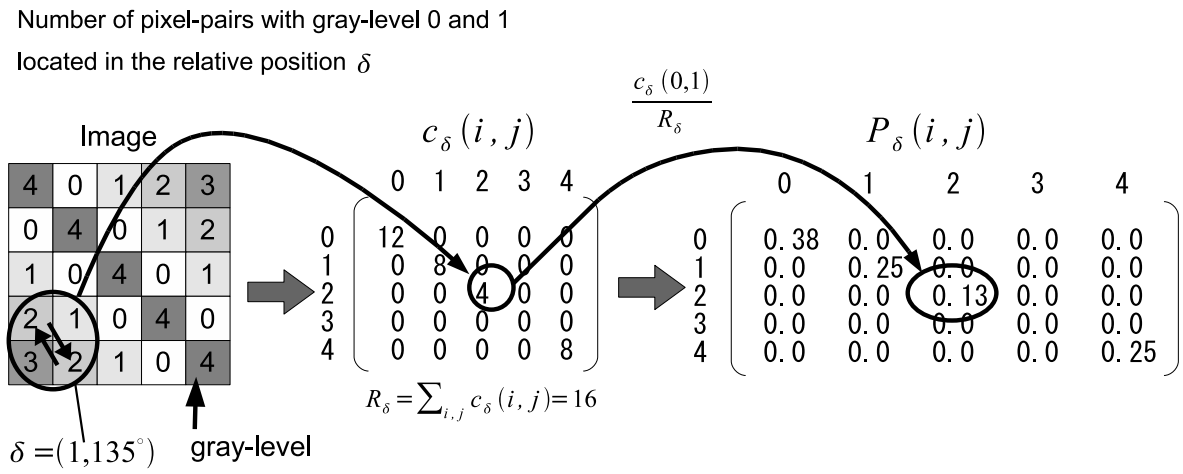


図 3.1　変位 $\delta = (d, \theta)$ と同時生起行列

要素間の差分である．マトリクスは 1 つの対象データにつき，参照データの数と同数生成する．例えば，参照データが 3 つであれば，1 つの対象データにつき 3 つのマトリクスを生成する．このようにして作られたマトリクスは，特徴ベクトル空間上の軸として用いられる．

テクスチャ特徴量の計算

マトリクスにより表された特徴を定量化するため，テクスチャ特徴量の計算を行う．ここで，テクスチャ特徴量は対象データの特徴を参照データを軸とする特徴ベクトル空間に写像する関数の役割を果たす．

画像の特徴を定量化したものをテクスチャ特徴量という．本手法では，先に生成したマトリクスを画像とみなして同時生起行列を生成し，テクスチャ特徴量を抽出する．同時生起行列は，画像領域上で δ だけ離れたピクセル間の階調の変化の生起確率を成分とした行列である．δ は図 3.1 に示すように距離と角度を用いて $\delta = (d, \theta)$，もしくは x 軸方向と y 軸方向の変位を用いて $\delta = (d_x, d_y)$ のように表す．本手法では，対象データの種類に応じて適切に設定された距離 d におけるテクスチャ特徴量を変位 $(d, 135°)$ において抽出する．尚，$\delta = (d, 0°)$ と $(d, 180°)$，$(d, 45°)$ と $(d, 225°)$，$(d, 90°)$ と $(d, 270°)$，そして $(d, 135°)$ と $(d, 315°)$ は明らかに同じ同時生起行列となる．

今，横方向 N_x 画素，縦方向 N_y 画素の画像を考える．画像に含まれる画素の階調が 0 から $N_g - 1$ までの離散的な値をとるとき，同時生起行列 P_δ は画像の大きさによらず N_g

階の正方行列となる．その要素 $P_\delta(i,j)$ $(i,j=0,1,2,\cdots,N_g-1)$ は階調 i の点から一定の変位 $\delta=(d,\theta)$ だけ離れた点の階調が j である確率であり，式 (3.1) の様に表される．

$$P_\delta(i,j)=\frac{c_\delta(i,j)}{R_\delta} \tag{3.1}$$

ここで $c_\delta(i,j)$ は階調 i の点から一定の変位 δ だけ離れた点の階調が j である組の数である．また，R_δ は δ だけ離れた点の組の総数であり，$\delta=(d_x,d_y)$ のとき，

$$R_\delta=(N_x-|d_x|)\times(N_y-|d_y|) \tag{3.2}$$

となる．

Haralick[11] らは同時生起行列を用いて 14 種類のテクスチャ特徴量を定義している．本手法ではこのうち，マトリクスを構成する 2 つのデータの要素間の関係をよく表すと考えられる以下の 5 つの特徴量を用いる．

i) ASM（Angular Second Moment）：

$$\mathrm{ASM}=\sum_{i=0}^{N_g-1}\sum_{j=0}^{N_g-1}P_\delta(i,j)^2 \tag{3.3}$$

テクスチャの一様性を示す．$P_\delta(i,j)$ が大きな値を持つと ASM は大きくなる．つまり一様性が高いと判断できる．

ii) CON（Contrast）：

$$\mathrm{CON}=\sum_{i=0}^{N_g-1}\sum_{j=0}^{N_g-1}(i-j)^2P_\delta(i,j) \tag{3.4}$$

濃度変化の強さを示す．画素対の濃度差 $|i-j|$ の画素全体についての平均であり，濃度差の高い画素対が多いほど値が大きくなる．

iii) COR（Correlation）：

$$\mathrm{COR}=\frac{1}{\sigma_x\sigma_y}\sum_{i=0}^{N_g-1}\sum_{j=0}^{N_g-1}(ij\cdot P_\delta(i,j)-\mu_x\mu_y) \tag{3.5}$$

相関の強さを示す．画像内に特定のパターンが現れる場合にこの値が大きくなる．尚，μ_x, μ_y, および σ_x, σ_y はそれぞれ x, y 方向に関する階調値

$$\begin{cases} P_x(j) = \sum_{i=0}^{N_g-1} P_\delta(i,j) \\ P_y(i) = \sum_{j=0}^{N_g-1} P_\delta(i,j) \end{cases} \tag{3.6}$$

の平均と標準偏差である．

iv) IDM（Inverse Difference Moment)：

$$\mathrm{IDM} = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} \frac{1}{1+(i-j)^2} P_\delta(i,j) \tag{3.7}$$

濃度変化の一様性を示す．画像が局所的な変化に乏しい場合大きな値を持つ．

v) ENT（Entropy)：

$$\mathrm{ENT} = -\sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} P_\delta(i,j) \log(P_\delta(i,j)) \tag{3.8}$$

画像の複雑さの尺度であると同時に情報量も表す．$P_\delta(i,j)$ の値が均等に割り当てられている程大きくなる．つまり，ENT が大きい場合には画像中に多くの階調が満遍なく現れていることを示す．

類似度の計算

本手法では，データの特徴量を “参照ベクトル (Reference Vector)” で表す．参照ベクトルはテクスチャ特徴量を要素とする特徴ベクトルであり，1 つの対象データにつき，1 つの参照ベクトルが作成れる．参照ベクトルの次元数は以下のように定まる．

$$\text{参照ベクトルの次元数 } (mk) = \text{参照データの数 } (m) \times \text{テクスチャ特徴量の種類 } (k) \tag{3.9}$$

14 個のテクスチャ特徴量のうち，k 個の特徴量を用いるとすると，参照ベクトルの次元数はいずれも mk となる．例えば，3 個の参照ソースコードを用いて 5 種類のテクスチャ特徴量を算出した場合，15 次元の参照ベクトルにより 1 つの対象ソースコードの特徴を表すこととなる．i 番目の対象データの参照ベクトル $\boldsymbol{r}_i$ は以下のように定義される．

$$\boldsymbol{r}_i = \left\{ r^i_{11}, \cdots, r^i_{1k}, \cdots, r^i_{m1}, \cdots, r^i_{mk} \right\} \tag{3.10}$$

ここで r^i_{uv} は，u 番目の参照データ $(u = 1, 2, \cdots, m)$ との間で，v 番目 $(v = 1, 2, \cdots, k)$ の特徴量を用いて算出した参照データの要素である．

ここまでの処理は，参照データが変更されない限り，類似度の計算に先駆けて一度だけ行われる前処理である．この参照ベクトルを用いて一対の対象データ間の類似度を算出し，類似性検出を行う．

参照ベクトルを用いた類似度の算出

類似度はそれぞれの対象データの持つ全ての参照ベクトル間のユークリッド距離により求められる．

2 つのデータ d_1, d_2 間の類似度 R_{d_1,d_2} は対象データ d_1 の参照ベクトル $\boldsymbol{r}_1$ および対象データ d_2 の参照ベクトル $\boldsymbol{r}_2$ を用いて以下のように定義する．

$$R_{d_1,d_2} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{k} w_j (r^1_{ij} - r^2_{ij})^2} \tag{3.11}$$

ここで w_j $(0 \leq w_j \leq 1.0,\ j = 1, \cdots, k)$ は j 番目のテクスチャ特徴量に対する重みである．

3.2 記述スタイルに基づく一次元データの特徴抽出手法：CM アルゴリズム

マルコフモデル

マルコフモデル（Markov Model）とは，不確定な事象をモデル化する確率過程の 1 つである．マルコフモデルは，マルコフ連鎖（Markov Chain）に従い離散時刻 $t(t = 1, \cdots, t, \cdots, T)$ ごとに遷移し，状態空間 $\boldsymbol{\mathcal{L}} = \{s_i | 1 \leq i \leq L\}$ 上の状態 s_i を値にとる有限個の確率変数 S_t と，遷移した状態により一意に決まる記号の観測確率により定義される．ある確率変数 S_t から次の確率変数 S_{t+1} への遷移は遷移確率に基づいており，各々の確率変数のとる状態にセットされた観測確率に基づき，決まった記号が観測される．

マルコフ連鎖とは，マルコフ過程（Markov Process）のうち，とりうる値が離散的であるものを指す．マルコフ過程とは，マルコフ性をもつ確率過程であり，一般に，マルコフ

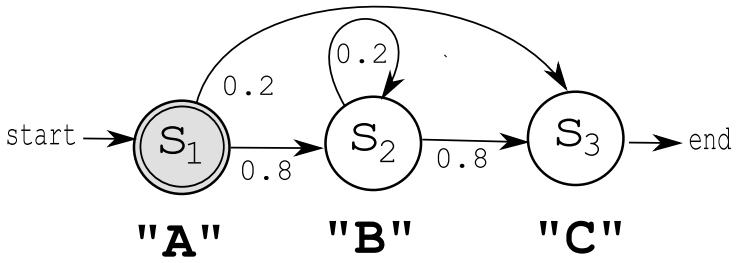


図 3.2　マルコフモデルの例

過程とは単純マルコフ過程を指す．単純マルコフ過程においては，時刻 t において S_t がとる状態は直前の S_{t-1} のとる状態の影響のみを受け，それ以前の状態とは無関係に決まるため，次状態に遷移する確率は条件つき確率を用いて $P(S_t|S_{t-1})$ と書くことができる．これにより，マルコフモデルをベイジアンネットワーク（Bayesian Network）に書き換えることも可能である．

図 3.2 はマルコフモデルの例である．図中 “A”，“B”，“C” は記号であり，各状態では対応する記号のみが観測される．例えば，記号 “A” は s_1 に遷移したときのみに観測される記号であり，s_2，s_3 ではそれぞれ “B”，“C” のみが観測される．

テキストデータに対して用いる場合，モデル上の各状態がトークンを表し，その変遷経路によりトークン列が決定されることになる．例えば，図 3.2 において出力可能な系列データの上位 3 パターンは “ABC”（64%），“AC”（20%），“ABBC”（12.8%）となる．

隠れマルコフモデル

隠れマルコフモデル（Hidden Markov Model: HMM）[12, 13] の例を図 3.3 に示す．HMM では，各状態において得られる出力記号は一意に決まっておらず，その状態において観測され得る複数の記号についてそれぞれ確率が決められている．

HMM では，時刻 t において，マルコフ連鎖に従う確率変数 S_t の実現値が状態 s_i であるとき，記号 o_k が観測される．ただし，どの状態に遷移するかは時刻 t に非依存である．

記述スタイルモデル

“記述スタイルモデル（Coding Model）” は HMM をベースとしたモデルである．

テキストデータを一次元の時系列データとみなし，その要素である記号（トークン）の出現頻度や共起に基づく特徴を記述スタイルとしてモデル化する．

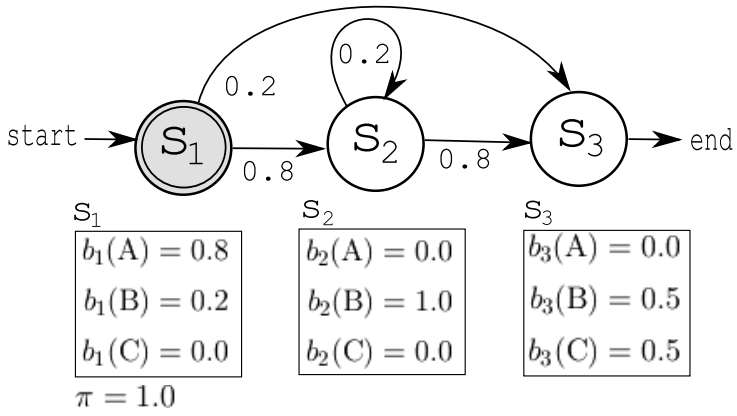


図 3.3　隠れマルコフモデルの例

記述スタイルモデルのパラメータ

記述スタイルモデル M はマルコフ連鎖に従い遷移する有限個の状態系列として表され，長さ T の記号系列をモデル化する．HMM と同様に，状態集合 $\boldsymbol{\mathcal{L}}$，記号集合 $\boldsymbol{\Sigma}$，そしてパラメータ集合 $\boldsymbol{\lambda}$ により構成される．

$$M = (\boldsymbol{\mathcal{L}}, \boldsymbol{\Sigma}, \boldsymbol{\lambda}) \tag{3.12}$$

$\boldsymbol{\mathcal{L}}$ は L 個の状態からなり，その遷移は時刻 t における確率変数 S_t のなす状態遷移列 $\boldsymbol{S}_{1:T}$ により表される．

$$\boldsymbol{\mathcal{L}} = \{s_i | 1 \leq i \leq L\} \tag{3.13}$$

$$\boldsymbol{S}_{1:T} = \{S_1, S_2, \cdots, S_T\} \tag{3.14}$$

$S_t (S_t \in \boldsymbol{S}_{1:T})$ は時刻 t において記述スタイルモデルがとる状態の確率 $P_t(s_i)$ を保持する．

$$P_t(s_i) \in [0, 1], \quad \sum_{s_i \in \boldsymbol{\mathcal{L}}} P_t(s_i) = 1 \tag{3.15}$$

S_t は例えば，$\{P_t(s_1) = 0, \cdots, P_t(s_i) = 1, \cdots, P_t(s_L) = 0\}$ となる．以降ではこれを $\{s_i\}$ と書く．すなわち

$$S_t = \{s_i\}. \tag{3.16}$$

$\boldsymbol{\Sigma}$ は N 個の記号からなる記号集合である．

$$\boldsymbol{\Sigma} = \{\sigma_k | 1 \leq k \leq N\} \tag{3.17}$$

S_t において観測される記号 σ_k は確率変数 O_t により決定される．すなわち，モデルが状態遷移列 $\boldsymbol{S}_{1:T}$ をとるとき，各時刻において観測される記号 σ_k のなす記号系列は観測確率変数列 $\boldsymbol{O}_{1:T}$ により決定される．

$$\boldsymbol{O}_{1:T} = \{O_1, O_2, \cdots, O_T\} \tag{3.18}$$

$O_t (O_t \in \boldsymbol{O}_{1:T})$ は S_t における記号の観測確率を表す．

$$O_t = P(\sigma_k | S_t = \{s_i\}) \tag{3.19}$$

但し，$\sigma_k \in \boldsymbol{\Sigma}$, $s_i \in \boldsymbol{\mathcal{L}}$, $0 \leq P(\sigma_k | S_t = \{s_i\}) \leq 1$ である．

隠れマルコフモデル M では，マルコフ連鎖に従う状態遷移列 $\boldsymbol{S}_{1:T}$ が各状態において実現値 s_i をとるとき，観測確率変数列 $\boldsymbol{O}_{1:T}$ に従い記号系列が観測される．現在の状態は，直前の状態にのみ依存し，時刻 t における S_t が s_i であるとき観測される記号 σ_k は，$P(\sigma_k | S_t = \{s_i\})$ により確率的に決定される．

M のパラメータ集合 $\boldsymbol{\lambda}$ は初期確率 $\boldsymbol{\pi}$, 遷移確率 $\boldsymbol{A}$，観測確率 $\boldsymbol{B}$ により構成される．

$$\boldsymbol{\lambda} = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B}) \tag{3.20}$$

(1) 初期確率

状態 s_i が開始状態である確率を π_i とする．

$$\pi_i = P(S_1 = \{s_i\}) \tag{3.21}$$

ここで，

$$\sum_{i=1}^{L} \pi_i = 1 \tag{3.22}$$

である．状態空間 $\boldsymbol{\mathcal{L}}$ についてその確率集合を

$$\boldsymbol{\pi} = \{\pi_i | 1 \leq i \leq L\} \tag{3.23}$$

で表す.

(2) 遷移確率

時刻 $t-1$ において記述スタイルモデルが状態 s_i をとるとき，次時刻 t において状態 s_j に遷移する確率は,

$$a_{i,j} = P(S_t = \{s_j\} | S_{t-1} = \{s_i\}) = a(S_{t-1}, S_t) \tag{3.24}$$

である．ここで，すべての $s_i \in \boldsymbol{\mathcal{L}}$ について,

$$\sum_{j=1}^{L} a_{i,j} = 1 \tag{3.25}$$

であり，モデル全体では,

$$\boldsymbol{A} = \begin{pmatrix} a_{1,1} & \cdots & a_{L,1} \\ \vdots & \ddots & \vdots \\ a_{1,L} & \cdots & a_{L,L} \end{pmatrix} \tag{3.26}$$

となる.

(3) 観測確率

時刻 t において記述スタイルモデルが状態 s_i をとるとき，記号 σ_k が観測される確率は

$$b_i(\sigma_k) = P(\sigma_k | S_t = \{s_i\}) = b_{S_t}(O_t) \tag{3.27}$$

である．ここで，全ての $s_i \in \boldsymbol{\mathcal{L}}$ について,

$$\sum_{k=1}^{N} b_i(\sigma_k) = 1 \tag{3.28}$$

であり，モデル全体では,

$$\boldsymbol{B} = \begin{pmatrix} b_1(\sigma_1) & \cdots & b_L(\sigma_1) \\ \vdots & \ddots & \vdots \\ b_1(\sigma_N) & \cdots & b_L(\sigma_N) \end{pmatrix} \tag{3.29}$$

とまとめる.

記述スタイルモデルによる作成者認識

本手法では，作成者 A の作成したテキストデータを N_A 個のトークン部分列集合 $\boldsymbol{X}_A = \{x_n | 1 \leq n \leq N_A\}$ とみなし，作成者 A が作成したことが確かであるテキストデータを用いて予め作成しておいた作成者 A の記述スタイルモデル M_A と，トークン部分列 x_n の尤度 $P(M_A, x_n)$ の平均値 $M_A(X_A)$ を用いてテキストデータの作成者認識を行う.

ここで，長さ T_n の $x_n (x_n = \{\sigma_{k1}, \cdots, \sigma_{kT_n} | 1 \leq t \leq T_n,\ 1 \leq k \leq N\})$ は以下の定義に基づき，観測確率変数列 $O_{1:T_n}$ として表される.

$$O_{1:T_n} = \{O_1, \cdots, O_{T_n} | 1 \leq t \leq T_n,\ O_t = P(\sigma_{kt} | S_t = \{s_i\})\} \tag{3.30}$$

以上により，作成者認識に用いられるモデルの平均出力確率は，以下の 2 式により求められる.

$$P(M_A, x_n) = \sum_{\mathrm{all} S_{1:T_n}} \prod_{i=1}^{L} \pi_i b_{S_1}(O_1) \prod_{t=2}^{T_n} a(S_{t-1}, S_t) b_{S_t}(O_t) \tag{3.31}$$

$$M_A(X_A) = \frac{\sum_{n=1}^{N_A} P(M_A, x_n)}{N_A} \tag{3.32}$$

学習アルゴリズム

HMM の推定, 学習に用いられるアルゴリズムのうち, 本手法で用いるものは, Forward アルゴリズムおよび Baum-Welch アルゴリズムである．このうち，Forward アルゴリズムはモデルの状態系列がとりうる最適経路のみではなく，全ての経路に関する確率の和を求めるものであり，ある観測系列 O，つまりテキストデータがある作成者によって作成されたものであるかを計測するために用いる.

本手法においてモデルの学習に用いられている Baum-Welch アルゴリズムは，確率最大の経路だけでなく，確率の低い経路についても，経路に確率重みをかけて，データから

モデル（パラメータや遷移確率，出力確率）を学習する計算方法である．これを一般化したものがベイジアンネットでも用いられる EM アルゴリズムである．但しこの学習方法を用いるには，ある程度のデータ量が必要であるとされている．このため，本手法ではその対象となるデータに応じ，より多くの学習データを得られるような工夫が必要となる．

現行のアルゴリズムでは，1 つの記述スタイルを単純な構造を持つ複数の記述スタイルモデルにより表現している．これにより，同一の対象データから異なる種類の学習データを重複して抽出することができるため，学習効率の向上が期待できる．

第 4 章

FRef アルゴリズムの適用例

4.1 参照を用いたソースコード間類似性検出

4.1.1 背景

概要

文書間の類似性検出については，これまで言語学・工学両領域において盛んに研究が行われてきた。その適用範囲の広さと適用によるメリットの大きさから，当該技術に対する社会的需要と期待は益々高まっている。例えば教育現場においては，教育支援を目的としたレポートの類似性検出ツールなどの開発が行われ，活用されている。他にも，機械翻訳や文書の自動要約，文書からの情報抽出，盗用の発見など，多くの適用例がある。ここで使用されている類似度検出アルゴリズムの多くは，自然言語を対象として広く用いられてきた，tf/idf 法により特徴ベクトルを抽出する方法や，n-gram により語の共起頻度を測る方法などである [14]．しかし，それらの知見の全てがソースコード間の類似性検出技術として活用されるわけではない．つまり，プログラムソースコードの様な，人工言語を対象とした類似性検出を行う場合は，人工言語の特性に沿ったメトリクスを設定する必要があるため，自然言語処理とは異なった，独自の手法の開発が必要となってくる．この様なニーズを受けて，ソフトウェア開発現場におけるリファクタリングや，ソースコード盗用問題解決などを目的としたプログラムソースコード間の類似性を検出する技術が注目を集め，多くの研究成果が報告されている．これらの既存研究には，文字列やトークンベースのメトリクスを用いるものや，文書構造をグラフ表現し，その形状の比較を行うもの，ソースコード中の識別子数や関数の依存関係に着目したものなど様々な手法がある．ソー

スコード類似性検出に関連する既存の研究については次章で概説する．このうち文字列やトークンベース，またグラフなどの比較的複雑な構造を用いて複数個のソースコードのクラスタリングを行う場合には，コストのかかる類似度計算を繰り返す必要がある．意味情報を用いる手法の場合は比較的少ない計算量で類似度計算を行うことが可能であるが，ソースコード中の重要な特徴を全てメトリクスとして抽出するのは困難である．また，いずれの手法もソースコードそのものを類似度比較に用いる必要があるため，実際のアプリケーションにおいてはその管理におけるセキュリティ上の危険性も無視できない．

本稿で提案する手法は，文字列・トークンベースの手法に分類されるが，参照を用いてソースコードの特徴を間接的に抽出するという点が既存手法との大きな相違点である．すなわち，本手法ではまず任意個の参照ソースコードを予め選出しておき，これを用いて対象ソースコードの参照ベクトルを算出する．そしてこの参照ベクトル間のユークリッド距離を 2 つのソースコード間の類似度として定義する．このように，計算量の多い参照ベクトル算出までの過程を類似度比較処理と分離したことにより，参照ソースコードを変更しない限り，一度算出した参照ベクトルを用いて高速な類似度比較を行うことができる．さらに，ソースコード自体は前処理後には不要となるため，実データを常に保管することによる情報漏えいのリスクの回避や管理コストの削減が期待できる．これは例えばソフトウェア・リポジトリの Web 検索システムなどへの適用を考える際，非常に大きなメリットであるといえる．

以降，4.1.1 で先行研究について触れた後，4.1.2 で提案手法の詳細について述べ，4.1.3 では Java プログラムソースコードを用いて行った評価実験について報告する．

既存研究

関連する既存研究にはコードクローンの検出を目的とするものが多い．コードクローンとは，ソースコード中の同一または極めて類似した部分列のことを指す [15]．これらは主にコーディング中のコピー&ペーストによって生じる．コードクローンはバグの温床になりやすく，ソフトウェアの修正を困難にするため，ソフトウェア管理・保守における大きな問題となっている．既に 1995 年の時点で，ある 1.1MLOC（Line of Codes）を有する大規模ソフトウェアのうち約 20 %が何らかのコードクローンを含んでいたという報告がある [16]．このため，新たにコードクローンを作らない工夫だけではなく，既存のコードクローンを除去する取り組みが重要視されている．ソースコードを再利用する際，その

ソースコードをそのまま使用することは少なく，通常は変数名・関数名の変更や構造の変更などの改変が行われる．したがって，単純な文字列の比較のみではコードクローンの発見は難しい．

本章では，既存の手法をそのアルゴリズムにより i)「文字列ベースの類似性検出」，ii)「構造の比較による類似性検出」，そして iii)「意味情報を用いた類似性検出」の 3 つに大別し，それぞれについて概説する．

i) の「文字列ベースの類似性検出手法」の多くは，前処理としてソースコードのトークン化や，変数名の置き換えなどによる抽象化処理を行う．この代表的な例として，上記の抽象化処理により得られた 2 つのトークン列を表形式で表現し，類似度比較に用いる手法がある [17]．このアプローチは本手法のアルゴリズムの一部としても取り入れられている．他に，比較対象となる 2 つのソースコードのファイル名やソースコード内の最長共通部分列から算出した類似度を用いてソースコードの分類を行う手法 [18] や，k-gram を用いて算出された Fingerprint という情報の共有数により文書間の類似度を計測している手法 [19]，Parameterized Suffix Tree というデータ構造に基づいたアルゴリズムを用いる手法 [16] などが提案されている．

ii) の「構造の比較による類似性検出手法」とは，ソースコードを主にグラフ形式で表現し，その形状を比較することにより類似性を検出する手法である．この例としては，ソースコードを概念グラフとして表現し，構造情報と内容から類似性を検出する手法 [20] や，プログラムにおける制御とデータの流れや手続き間の呼び出し関係をグラフで表現し，その最大共通部分グラフを近似する類似グラフの形状により類似性を判断する手法 [21] などがあげられる．

さらに別の角度からのアプローチとして，ソースコードの記述内容に着目した，iii) の「意味情報を用いた類似性検出手法」がある．カテゴリ検索のためのソースコードの分類法として，潜在的意味解析によりこれを行った例や [22]，トークンのクラスタからソフトウェアのクラスタを算出するという例などがこれにあたる [23]．他にも，自己組織化マップを用いて入力データ全体の相関関係を表す手法や [24]，重要なメトリクス値の組からなるハッシュ値が閾値以内にある 2 つのソースコードの差分を用いて算出した非類似度を比較する手法 [25]，コメントや識別子などから抽出した意味的・構造的情報を利用したソースコードのクラスタリング手法や [26]，ソースコード中の全ての語の中から選出した複数の特徴語をそれぞれ 1 つのカテゴリとみなしてソースコードを分類する手法などがある

[27].

Burd[28] は，代表的なコードクローン発見ツール CCFinder[17]，CloneDR，Covet および盗用発見ツール JPlag[29]，MOSS[30] について，検出されたコードクローンの数や再現率，適合率などをもとにその性能を比較している．実験では，どのツールにもそれぞれ長所があり，突出して優れているツールはないということが確認されている．また，類似性の定義自体も各手法により異なるため，現状では用途や対象に応じて最適なツールを選択する必要がある．但し，この比較実験ではツールの性能が十分に活かされていないことや，コードクローンの定義が筆者らの主観により行われたことが指摘されている [31].

4.1.2 参照を用いたソースコードの特徴表現

概要

ここでは，3.1 において提案した特徴抽出アルゴリズムをもとに構築したソースコード間類似性検出手法について論じる．本手法では既存の手法とは異なり，類似性検出を行いたいソースコードである “対象ソースコード（Target Source Code）” を直接比較するのではなく，予め選出しておいた “参照ソースコード（Reference Source Code）” という複数のソースコードを用いてソースコードの特徴を表す “参照ベクトル（Reference Vector）” を算出し，これを用いた間接的な比較を行う．計算量の多い参照ベクトル算出過程は類似性検出と分離されているため，類似性検出処理自体の計算量は大幅に削減される．本手法のアルゴリズムを図 4.1 に示す．図中の各手順の概要は以下の通りである．

1) **［参照ソースコードの選出と前処理］** まずはじめに，複数の参照ソースコードを選出し，トークン化・抽象化の前処理を行い，トークン列を生成する．これを “参照抽象トークン列（Reference Abstract Token Sequence）” と定義する．
2) **［対象ソースコード前処理］** 対象ソースコードについても同様の前処理を行い，トークン列を得る．これを “対象抽象トークン列（Target Abstract Token Sequence）” と定義する．
3) **［トークン共起マトリクスの生成］** この対象抽象トークン列と先に生成した参照抽象トークン列を表形式で表現し，2 つの抽象トークン列の対応する要素間における同一トークンの共起分布を表す共起マトリクスを形成する．これを “トークン共起マトリクス（Token-co-occurrence Matrix）” と定義する．

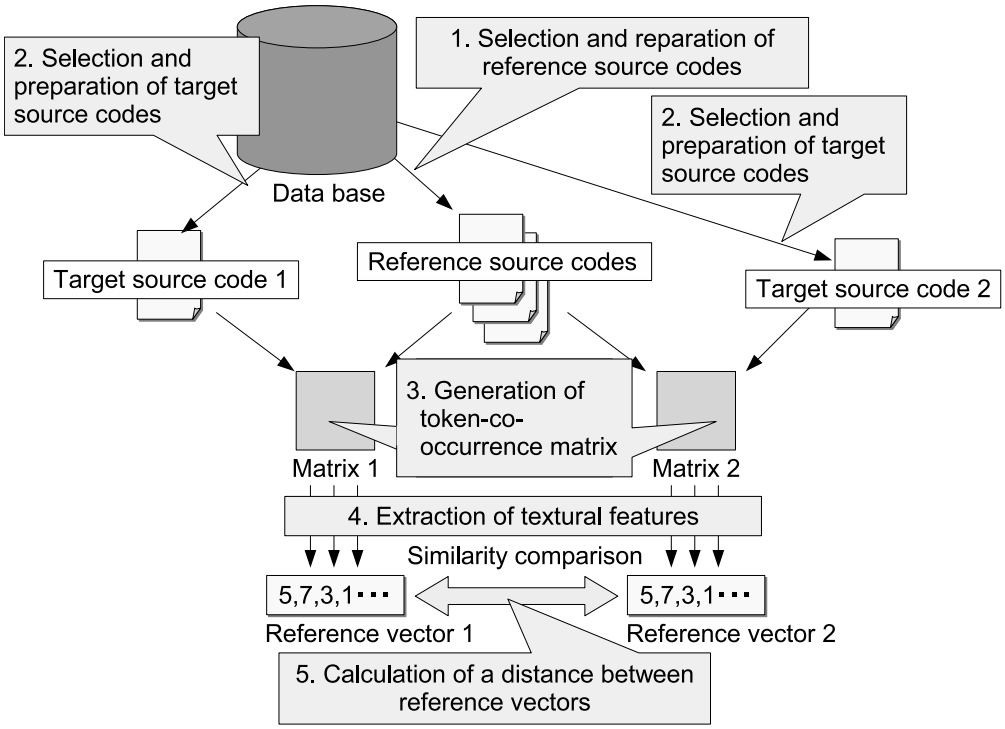


図 4.1　参照を用いたソースコード間類似性検出手法の概要

4) **［テクスチャ特徴量の抽出］**トークン共起マトリクスより 5 種類のテクスチャ特徴量を抽出し，これを要素とする特徴ベクトルを参照ベクトルと定義する．ここまでの処理は，参照ソースコードが変更されない限り，類似性検出に先駆けて一度だけ行われる前処理である．

5) **[類似度の算出]** 算出した参照ベクトル間のユークリッド距離をソースコード間の類似度とする．

次小節より各手順の詳細を示す．

準備

類似性検出を行いたい N 個の対象ソースコードからなるソースコード集合を S とする．

$$S = \{s_1, \cdots, s_i, \cdots, s_N\} \tag{4.1}$$

まず，ソースコードの類似性検出に先駆けて，S より任意の基準で m 個の参照ソースコード $\hat{s}_1, \hat{s}_2, ..., \hat{s}_m$ を選出する．

$$\{\hat{s}_1, \hat{s}_2, ..., \hat{s}_m\} \subset S \tag{4.2}$$

$\hat{s}_1, \hat{s}_2, ..., \hat{s}_m$ のそれぞれのソースコードから，類似度計算に不要な空白やコメントを除去し，トークン単位で分割する．定数および変数の置換により抽象化し，抽象トークン列 $\hat{t}_1, \hat{t}_2, ..., \hat{t}_m$ を得る．

$$\hat{t}_i = \left\{\hat{t}_1^i, \hat{t}_2^i, \cdots, \hat{t}_{\hat{n}_i}^i\right\} \tag{4.3}$$

但し，$\hat{t}_j^i$ はトークン列 $\hat{t}_i$ を構成する各トークン，$\hat{n}_i$ はトークン列 $\hat{t}_i$ の総トークン数である．

一般に，ソースコードの類似を考えるとき，その表記上の差異ではなく，処理内容であるデータや制御の流れに着目する．ソースコードはコンピュータへの命令書であり，自然言語文書と異なり人間が見て感じる表記上の類似が必ずしもその意味上の類似とは一致しないという性質を持つ．そのため，本手法では一般的なソースコードの類似性検出としての適用を考え，標準化により類似性検出の障害となるノイズを削減する．具体的には，空白やコメントなどの不要なトークンを除去し，表記は異なるが共通の意味を持つトークンを共通の表現に変換する．図 4.2 に 2 つの異なるソースコードに前処理を施した例を示す．

対象ソースコードの前処理

類似度計算を行う 2 つの対象ソースコード s_1, s_2 に対して，参照ソースコード同様の前処理を行う．これにより得られたトークン列を t_1, t_2 とする．

$$t_j = \left\{t_1^j, t_2^j, \cdots, t_{n_j}^j\right\} \tag{4.4}$$

トークン共起マトリクスの生成

1 つの対象抽象トークン列と 1 つの参照抽象トークン列を縦横に並べ，トークン共起マトリクスを生成する．これは 2 つの抽象トークン列における同一トークンの共起分布を表すマトリクスである．マトリクスの各要素は縦と横に配置された 2 つのトークン列の対応する要素が一致すれば “1” とし，一致しなければ “0” とする．マトリクス上の “1” と “0” の分布により 1 つの参照抽象トークン列と 1 つの対象抽象トークン列における類似性

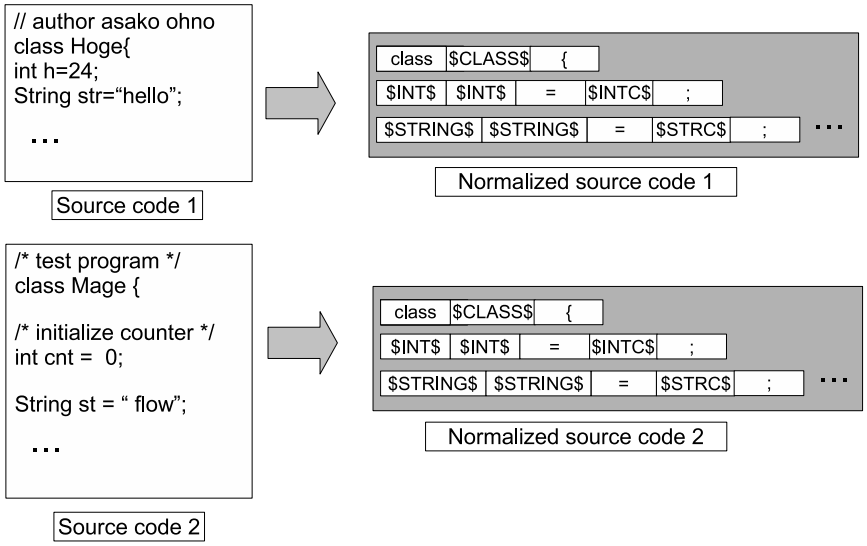


図 4.2　ソースコードの抽象化

を表現できる．ここで用いられた参照ソースコードと対象ソースコードが同一である場合は，マトリクスの左上から右下にかけての対角線上に “1” が表示され，またこの直線の左右の領域の “1” の分布は線対称となる．図 4.3 にトークン共起マトリクスの例を示す．例えば，縦に配置された対称抽象トークン列の一番目の要素と横に配置された参照抽象トークン列の一番目の要素は同一の “class” というトークンであるため，トークン共起マトリクスにおける $(1,1)$ セルの要素 $e^M_{1,1}$ の値は “1” となる．

この様なマトリクスを t_1 と $\hat{t}_1, \hat{t}_2, ..., \hat{t}_m$ のそれぞれ，また同様に t_2 と $\hat{t}_1, \hat{t}_2, ..., \hat{t}_m$ のそれぞれについて生成する．生成したマトリクス $M(t_1, \hat{t}_1)$, $M(t_1, \hat{t}_2), \cdots$, $M(t_1, \hat{t}_m)$, および $M(t_2, \hat{t}_1)$, $M(t_2, \hat{t}_2), \cdots$, $M(t_2, \hat{t}_m)$ は以下のように定義される．

$$M(t_i, t_j) = \begin{pmatrix} e^M_{1,1} & \cdots & e^M_{n_i,1} \\ \vdots & \ddots & \vdots \\ e^M_{1,n_j} & \cdots & e^M_{n_i,n_j} \end{pmatrix} \tag{4.5}$$

$$e^M_{k,l} = \begin{cases} 1 \ \text{ if } \ t^i_k = t^j_l, \\ 0 \ \text{ otherwise,} \end{cases} \quad (k = 1, \cdots, n_i, l = 1, \cdots, n_j) \tag{4.6}$$

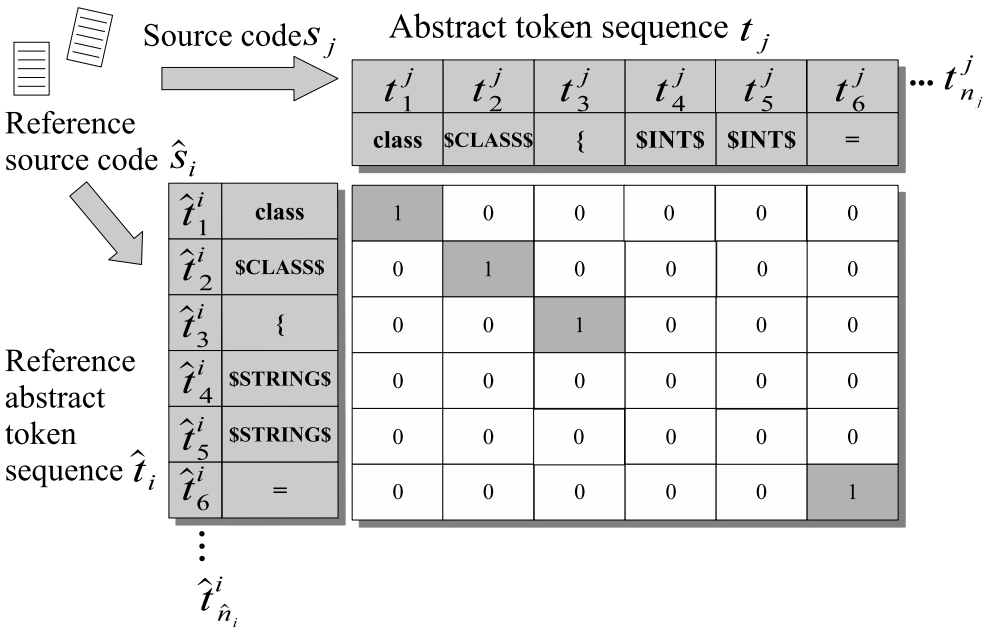


図 4.3　トークン共起マトリクス

テクスチャ特徴量の計算

これまでの手順により得られた複数のトークン共起マトリクスにより，対象ソースコードと参照ソースコード間のトークン共起に基づく類似がそれぞれ表現されている．つまり，1 つの対象ソースコードの特徴は，参照ソースコードを用いて生成した複数のトークン共起マトリクスにより表現されている．

これを特徴ベクトルとして定量化する．トークン共起マトリクス上で “0” を有する要素を白画素，“1” を有する要素を黒画素とみなすと，マトリクスにより表されるトークンの共起分布は二値画像における白黒画素の分布と考えることができる．本手法では，二値画像とみなしたトークン共起マトリクスから同時生起行列を生成し，テクスチャ特徴量を抽出し，これを特徴ベクトルの要素とする．

本手法では Haralick[11] によって定義された 14 種類のテクスチャ特徴量のうち，2 つのソースコードにおける同一トークンの分布をよく表すと考えられる，ASM（Angular Second Moment），CON（Contrast），COR（Correlation），IDM（Inverse Difference Moment），ENT（Entropy），の 5 つの特徴量を用いる．

本手法では，4.1.2 で求めたトークン共起マトリクスを二次元二値画像とみなし，テクスチャ特徴量を抽出する．ここでは特に対角方向の要素の相関が重要であるので，斜め

135 度，すなわち $\delta = (d, 135°)$ におけるテクスチャ特徴量を，各ソースコード s_1, s_2 それぞれについて，m 個の参照ソースコードに対して求める．これらより，ソースコード s_1 の参照ベクトル $\boldsymbol{r}_1$ およびソースコード s_2 の参照ベクトル $\boldsymbol{r}_2$ が形成される．14 個のテクスチャ特徴量のうち，k 個の特徴量を用いるとすると，$\boldsymbol{r}_1$ と $\boldsymbol{r}_2$ の大きさはいずれも mk となる．例えば，3 個の参照ソースコードを用いて 5 種類のテクスチャ特徴量を算出した場合，15 次元の参照ベクトルにより 1 つの対象ソースコードの特徴を表すこととなる．

$$\boldsymbol{r}_i = \left\{ r^i_{11}, \cdots, r^i_{1k}, \cdots, r^i_{m1}, \cdots, r^i_{mk} \right\} \tag{4.7}$$

ここで r^i_{uv} は，u 番目の参照ソースコード $(u = 1, 2, \cdots, m)$ との間で，v 番目 $(v = 1, 2, \cdots, k)$ の特徴量を用いて算出した参照ベクトルの要素である．

ここまでの処理は，参照ソースコードが変更されない限り，類似度の計算に先駆けて一度だけ行われる前処理である．

参照ベクトルを用いた類似度の算出

2 つのソースコード s_1, s_2 間の類似度 R_{s_1,s_2} をソースコード s_1 の参照ベクトル $\boldsymbol{r}_1$ およびソースコード s_2 の参照ベクトル $\boldsymbol{r}_2$ を用いて以下のように定義する．

$$R_{s_1,s_2} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{k} w_j (r^1_{ij} - r^2_{ij})^2} \tag{4.8}$$

ここで w_j $(0 \leq w_j \leq 1.0,\ j = 1, \cdots, k)$ は j 番目のテクスチャ特徴量に対する重みである．

4.1.3 実験

設定

4.1.2 で説明したアルゴリズムを実装した類似ソースコード検出ツールを開発し，本手法の評価実験を行った．以下は実験を行った計算機環境である．

[計算機環境]

- CPU：AMD Athlon64 X2 4400+ (2.22GHz)
- 主記憶容量：2GB RAM

[実験対象データ]

・神戸大学工学部情報知能工学科
「情報知能工学演習 IV」 Java 課題ソースコード（ファイル総数：434）

実験対象となる課題ソースコードは，Infoseek のテレビ番組表で公開されている iEPG ファイルの情報を保存・出力するというもので，`EPGCollectorFromInfoseek`, `EPGManager`，`EPGItem`，`PerformerItem` の 4 つのクラスにより構成される．このうち，`EPGCollectorFromInfoseek`，`PerformerItem` の 2 つのクラスは担当教員によりほぼ完成されており，受講者は `EPGManager` クラスの 4 つのメソッドと `EPGItem` クラス 1 つのメソッドを各自記述する．授業用 Web ページ上で各メソッド記述のヒントや推奨 API の提示が行われており，また，いくつかの課題メソッドは部分的に記述がなされた状態の穴埋め形式となっているため，同名のソースコード間の類似度は全体的に高いことが予想される．また，本アルゴリズムで用いられる以下の 3 つのパラメータについても実験により検証を行う．

(1) テクスチャ特徴量について：
5 種類のテクスチャ特徴量それぞれ，あるいは全てを用いて実験を行い，類似性検出に最も有効な特徴量を特定する．
(2) 参照ソースコード数について:
1，3，5，7，14 個の参照ソースコードを用いて実験を行い，類似性検出に最も有効な参照ソースコードの数を特定する．
(3) テクスチャ特徴量算出時に用いる距離 d の値について：
テクスチャ特徴量算出時に用いる変位 δ の要素である距離 d を 1〜10 まで値を変えて実験を行い，出力結果の変化を分析する．

実験方法

次の手順により実験項目【E1】〜【E4】について実験を行う．

[**手順**]

1) ソースコード集合 S から任意個の参照ソースコードを選出する．
2) 参照ソースコードを含む全ての対象ソースコード，つまり S の全てのソースコードに対し，参照ベクトルの計算を行う．
3) S から単一のソースコードを選び，これを検索ソースコード（クエリ）とする．検

索ソースコードと S の全ての対象ソースコード間で (4.8) 式に従って類似度を計算する．

[実験項目]

【E1】テクスチャ特徴量の種類による評価の差:

テクスチャ特徴量による性能の違いを検証するため，参照ベクトルの計算に用いるテクスチャ特徴量を ASM，CON，COR，ENT，IDM のそれぞれを用いる場合と，5 つの特徴量全てを用いる場合それぞれについて実験を行う．このときの参照ソースコードの数は一律 14 個とする．

【E2】参照ソースコード数の違いによる評価の差:

参照ソースコードの数による性能の違いを検証するため，参照ソースコードの個数を 1，3，5，7，14 と変えて実験を行う．このとき用いるテクスチャ特徴量は【E1】の結果に基づき決定する．

【E3】参照ソースコード数の違いによる前処理および検索時間の差:

【E2】のそれぞれの場合において前処理および検索処理に要する時間を計測する．

【E4】距離 d の違いによる評価の差:

テクスチャ特徴量算出時に用いる変位 δ の要素である距離 d を 1 から 10 まで値を変えて実験を行い，出力結果の変化を分析する．

これらの設定による性能の違いを評価するため，以下では，同一のソースコード `x-A0-EPGManager.java` を検索ソースコードとして類似度計算に用いることとする．この課題では作成するプログラムに命名規則が与えられているため，同じファイル名を持つソースコードは同じ意図を持って作成された，つまり類似したソースコードであるといえる．ファイル名に “EPGManager” という文字列を含み，その内容が検索対象ソースコードに極めて近いと考えられるソースコードは，S の全ソースコード中に 73 個含まれている．これらのソースコードを適合ソースコードと定義する．

評価方法

検索されたソースコードはツールにより類似度の高い順に提示される．これを平均適合率 [32] を用いて評価する．平均適合率の計算は次のように行われる．今，n 個の対象ソースコードが類似度に関して降順にソートされるとする．この時上から i 番

目 ($i = 1, 2, \cdots, n$) のソースコード s_i について，適合ソースコードであるか否かを $\delta_i = \{1, 0\}$ で示すとすると，上位 i 個のソースコードによる適合率 P_i は以下の式で表される．

$$P_i = \frac{\sum_{j=1}^{i} \delta_j}{i} \tag{4.9}$$

このとき，平均適合率 P_{ave} は以下の様に定義される．

$$P_{\mathrm{ave}} = \frac{1}{\sum_{i=1}^{n} \delta_i} \sum_{i=1}^{n} \delta_i P_i \tag{4.10}$$

実験結果

前小節の【E1】〜【E2】の項目について (1)〜(3) の手順でそれぞれ実験を行い，得られた評価の平均と標準偏差を図にまとめた．

【E1】ではテクスチャ特徴量の種類による評価の差を見るため，参照ソースコード数を 14 個に固定し，テクスチャ特徴量を ASM，CON，COR，ENT，IDM のそれぞれを用いる場合と，複数のテクスチャ特徴量を併用する場合，5 つの特徴量全てを用いる場合それぞれについて 100 回ずつ実験を行った．このときの評価の平均と標準偏差について，その一部を図 4.4 に示す．図 4.4 に示されるように 5 種類全てのテクスチャ特徴量を用いた場合には及ばないものの，COR のみを用いた場合にも高い評価が得られた．また，複数のテクスチャ特徴量を併用する場合も，COR が含まれる場合に特に高い評価が得られた．この理由として，2 つのソースコード間にトークン共起が連続して見られた場合，マトリクス上の左上から右下にかけて斜線が表示されるが，斜め方向の相関を表す特徴量である COR はその分布傾向を特徴量として良く表すことが考えられる．以降の実験は全て 5 種類の特徴量を用いた場合と COR のみの場合の 2 条件下で行うこととする．

【E2】では参照ソースコード数の違いによる評価の差を見るため，5 種類全ての特徴量を用いた場合と COR のみを用いた場合の 2 つの条件下において，参照ソースコード数を 1，3，5，7，14 と変えてそれぞれ 100 回ずつ実験を行った．このときの評価の平均と標準偏差を図 4.5 に示す．図に示されるように，参照ソースコード数が 1 の場合に平均評価が 90% と最も低く，は単一では評価が低く標準偏差も大きい．参照ソースコード数 3 以上では評価は 98% 以上となる．この結果より，参照ソースコード数は多ければ多いほど良好な性能が得られることが分かる．

【E3】では参照ソースコード数の違いによる前処理および検索時間の差を見るため，先

に行った【E2】の条件において，その前処理および検索時間の差を計測した．このときの経過時間の平均と標準偏差を図 4.6 および図 4.7 に示す．図 4.6 は 5 種類の特徴量と COR のみを用いた場合の前処理時間の比較を示し，図 4.7 は同条件下における検索時間の比較を示している．前処理時間についてもこれと同様の傾向が見られた．この結果より，5 種類の特徴量を用いた場合のほうが若干長い時間を必要とするものの，その差は僅差であることから，テクスチャ特徴量の数による処理時間への影響は少ないことが分かる．また，参照ソースコード数の増加と処理時間はほぼ比例しているといえる．処理時間そのものについては，参照ソースコードが 1 個の場合は前処理に 25 秒，検索処理に 0.2 秒，3 個では前処理 50 秒，検索 0.3 秒，14 個では前処理 270 秒，検索に 1.4 秒かかっている．図 4.7 に示されるように，処理時間は線形に増加しているが，図 4.5 では，参照ソースコード数が 3 以上で急激に評価が上昇している．これらのことから，時間との兼ね合いを考慮し参照ソースコード数は 3 個あれば良いと考えられる．さらに【E2】の結果より，参照ソースコード 3 個以上で十分な評価が得られていることから，本手法では 434 個のソースコード集合中の，ある 1 つのソースコードに対する全ソースコードの類似度を前処理を含めても 1 分以内に算出し，0.3 秒で検索を行うことが可能であることが分かる．

【E4】では d の値による評価の差を見るため，テクスチャ特徴量算出時に用いる変位 δ の要素である距離 d を 1 から 10 まで値を変えてそれぞれ 100 回ずつ実験を行い，出力結果の変化を分析した．図 4.8 に示す様に，5 種類の特徴量を用いた場合と COR のみを用いた場合の傾向に大差はない．また，d の値によらず 99% 以上という良い評価を得られたことから，d の値の検索結果への影響は少ないといえる．

まとめ

今回の実験で以下のことが明らかになった．

(1) 実験に用いた 5 種類のテクスチャ特徴量のうち，COR を単体で用いた場合と 5 種類全てを用いた場合最も高い評価を得ることができた．テクスチャ特徴量の種類や数による前処理および検索処理の時間への影響は殆どないため，現時点ではわずかな差ではあるが 5 種類全ての特徴量を用いることが最適であると考えられる．

(2) 参照ソースコードは単数より複数で用いることで評価が高まるが，3 個以上の検索結果に大差はなく，また，前処理・検索処理ともに計算時間は参照ソースコードの数に比例するため，現時点では 3 個が最適であると考えられる．参照ソースコードを 3 個用いた

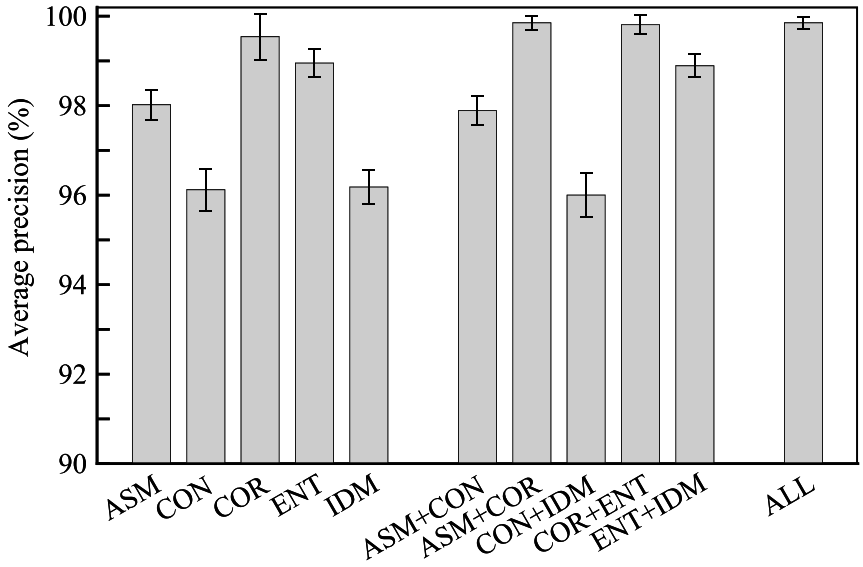


図 4.4　テクスチャ特徴量の種類と平均適合率

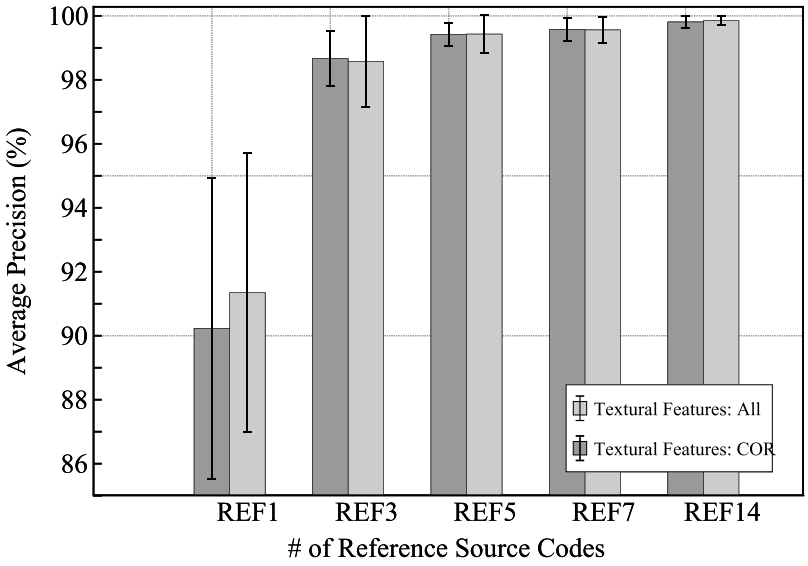


図 4.5　参照ソースコードの数と平均適合率

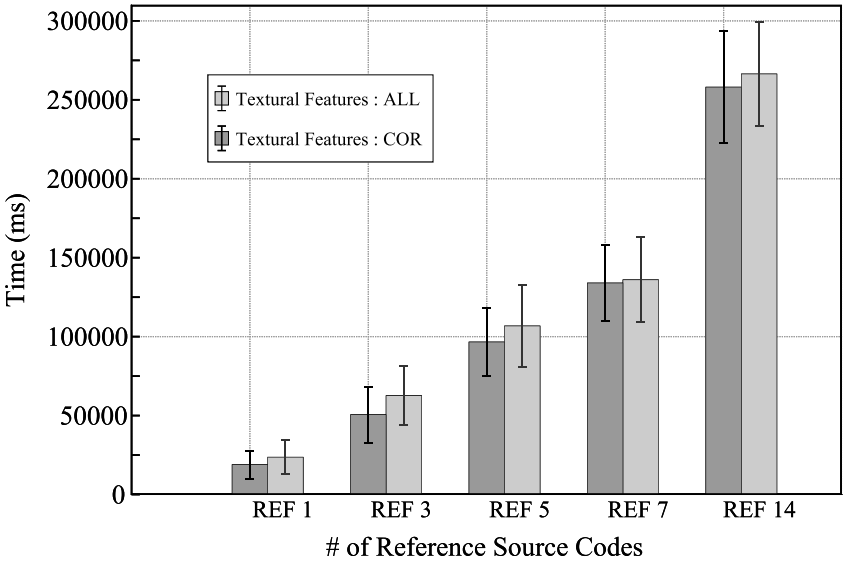


図 4.6　参照ベクトルの計算時間

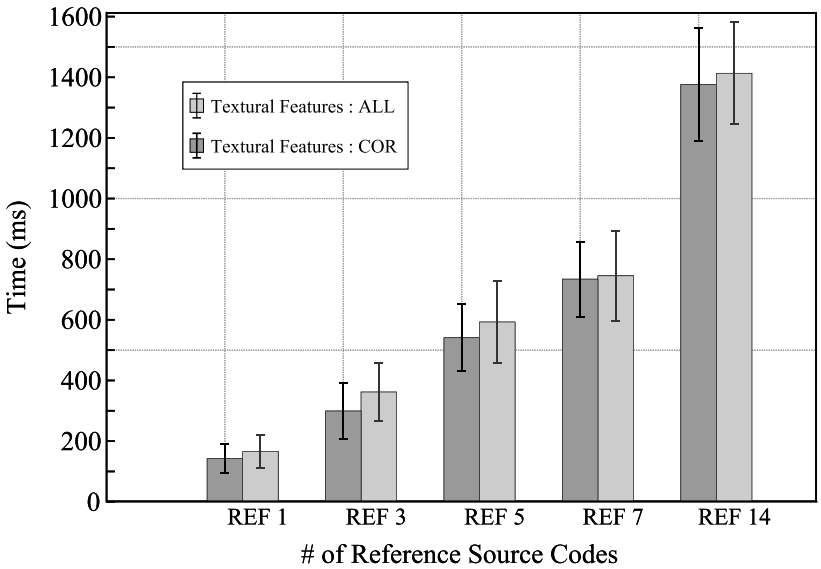


図 4.7　類似度の計算に要する時間

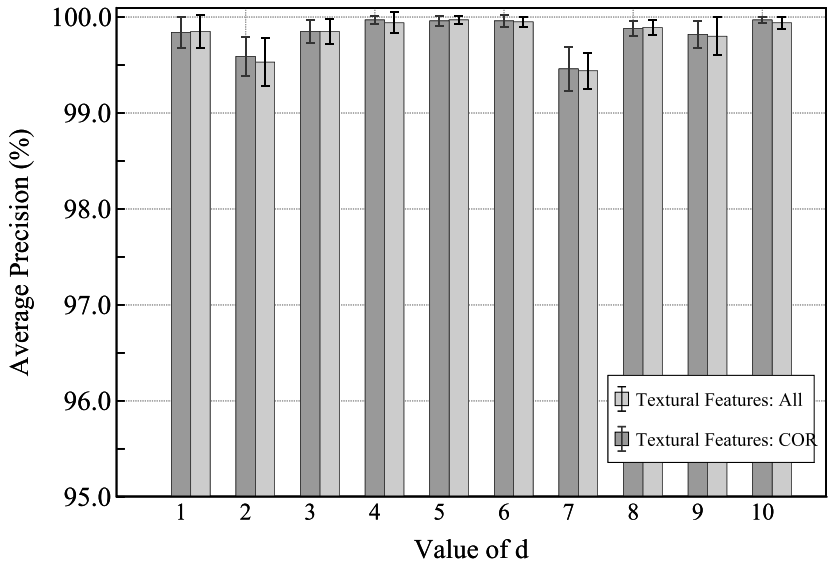


図 4.8　d ($\delta = (d, \theta)$) の値と平均適合率

場合の検索時間は 0.3 秒であり，前処理時間は 50 秒であった．しかしこの前処理は一度行えば参照ソースコードを変更しない限り必要ないため，検索そのものについては高速に行えることが確認できた．

(3) 距離 d の値については，これによる評価に顕著な差は見られなかったが，他のソースコード集合においても同様の傾向が見られるかどうか確認する必要がある．

4.2　参照を用いた画像間類似性検出

4.2.1　背景

概要

本章では，3.1 で述べた FRef アルゴリズムの画像データへの適用について論じる．

画像の類似度比較手法の用途は，指紋や顔の認証，画像データベースにおけるインデキシング，筆跡鑑定，盗作の発見など多岐に渡る．これに加え，動画の中から希望する静止画像を検出する技術や，監視カメラで捉えた映像を定期的に切り出し，直前の静止画と比較する自動監視システムなど，動画を対象としたものも数多く考案されている．このうち

本研究で提案するアルゴリズムの適用例としてまず考えられるのは，画像データベースにおける画像データのインデキシング及び検索である．

これまでに様々な画像間の類似性検出手法が提案されているが，これらの手法で用いられる特徴量の多くは，特定の属性をもつ画像群に対してのみ，その有効性が認められたに過ぎない．どのような属性の画像が検索対象となるかが予め明らかである場合は，その画像の属性を最も良く表現できる特徴量を適用することにより十分な結果が得られるであろうが，一般に，画像データベースに格納されている個々の画像の属性は必ずしも類似していないため，検索対象となる画像により有効な特徴量は異なる場合が多い．たとえ同じ意図をもって作成された画像同士であっても，画像の作成者や作成環境が異なる場合，同一の特徴量でその属性を一様に表現できるとは限らない．つまり多くの場合，属性の異なる複数の画像に対し同一の方法で特徴抽出を試みたとしても，それらの特徴を同程度の忠実さで定量的に表現することは難しい．しかし，一つ一つの画像に対し個別に最適と思われる特徴抽出を行うことや，前処理として簡単なクラスタリングを行った後に複数種類の特徴抽出を適用するというような方法では，あまりにも膨大な計算量が必要とされるため現実的とはいえない．

メタデータを用いた検索

これまでの画像検索は，画像に付与されたメタデータというテキストデータに対しキーワード検索を行うことで実現されてきた．しかし，既存の全ての画像にメタデータが付与されているというわけではない．また，これらに対し新たにメタデータを作成する際，作成者それぞれの基準により異なるキーワード付けが行われ，本来検索されるべき画像のうちいくつかが検索されない可能性があるという問題も考えられる．

統一された基準で新たにメタデータを作成するには膨大な労力が必要である．自動でメタデータ付けを行う場合は作業コストが削減できる反面，手作業のような自然なキーワード付けは難しい．いずれの場合も以後にどのような属性を持つ画像がデータベースに追加されるかという展望がない場合，検索対象となり得る全ての画像に対して最適なメタデータを付与することは不可能である．仮にあらゆる条件下において統一された基準でメタデータを作成できるような環境が整ったとしても，多様な属性をもつ画像群に対し，その特徴を文字列で的確に表現することは困難である．さらにキーワード検索では，同じ意味をもつ別の表現が検索キーワードとして用いられた場合，適合する画像全てを検索できな

い，もしくは実際は適合しない画像を検索結果に含めてしまうというデメリットもある．

しかし一般にメタデータによる検索は高速であるため，自動で的確なメタデータ付けを行うための研究が進められている．例えば，メタデータを用いた検索に画像の特徴量抽出を取り入れた手法が提案されている．Amagasa ら [33] は，複数の部分画像からウェーブレット特徴量を抽出し，検索対象画像とクエリ画像における特徴量の共起からキーワードと内容の相関を定量化し，画像の内容に相当すると思われるキーワード候補の提示を行い，ユーザはこれを基にメタデータを付与するという手法を提案している．これにより検索時のユーザの負担が軽減され，メタデータ付与作業が簡略化されることが期待できるが，最初に基準となるメタデータ付画像を手動で選出しておく必要があり，前途の問題の多くが解決されるものではないといえる．

内容に基づく検索

現在，データそのものの特徴に基づく検索，すなわち内容に基づく検索 (Content-Based Image Retrieval：CBIR) が注目されている．内容に基づく検索を行う際，最も重要なのが画像データの有するどの特徴を抽出し，どのように表現するかという点である．類似画像検索手法として一般的なのが画像を特徴ベクトルにより表現する手法である．これは画像から抽出した色，テクスチャ，形状などの情報を特徴量として定量化して特徴ベクトル空間上の座標として表現し，座標間の距離を計測することにより画像間の類似度を求めるという手法である．画像検索システムへの適用を考えた場合，まずデータベースとして蓄積された画像データ全てについて特徴ベクトルを算出し，検索画像が与えられた時，その画像について算出した特徴量とデータベース内の画像の特徴量との距離計算を行い，その距離の近いものを類似画像として検索する．

横山ら [34] はフラクタル画像圧縮の写像情報を画像の空間領域における相似領域の関係に基づいた構造表現とみなし，この相似関係を写像元と先の座標により構成される写像ベクトルという 4 次元のベクトルで表現し，二つの画像における写像ベクトル集合の類似度を用いて圧縮されたままの画像に対し検索および分類を行う手法を提案した．実験により，一定範囲内の画像の変動に対する堅牢性と，ウェーブレット変換や輝度ヒストグラムによる検索に対する優位性が実証された．しかしレンジサイズが最も小さい場合検索性能が高くなるということから，計算量と検索性能のトレードオフを解消する必要性が残されている．

呉ら [35] はウェーブレット変換により画像データを階層化された分解画像に分解し，これにより得られた色とテクスチャの情報からなる特徴ベクトルを自己組織化マップで分類し，検索に要する記憶領域の削減，処理の高速化を目指した．しかし類似していると思われるノードのみが検索されるため，算出した特徴量が検索対象画像の属性を十分に表せない場合はマッピングが適切に行われず，検索結果にノイズが多く含まれたり，逆に多くの検索もれが出るおそれがある．

このように，現在数多くの手法が提案されているが，総じて特徴の定量化に計算コストがかかり，これを軽減するために特徴量ベクトルの低次元化を行うことにより，検索精度が低下するというトレードオフを抱えている．また，画像解析に基づく多くの手法は，ある程度共通した属性を持つ画像群に対しては高い有効性を示しているが，多様な属性を持つ画像群に対しては，統一された基準で全ての画像の特徴を表現しようとしても，採用した特徴抽出法によってはその検索精度に大きく差が出てしまうという問題もある．

これらの問題を回避するためには，少なくも次の 2 点に留意する必要がある．1 点目は，個々の画像の属性を最もよく表す特徴量を用いること．2 点目は，類似性検出に影響の少ない特徴量だけを選んで使用しないことにより適合率の低下を極力抑えながら計算量を減少させることである．しかし現実的にはこれらの条件を満たすことは難しい．例えば複数の種類の特徴量を採用し，クエリ画像の属性により算出した特徴量ベクトルの一部または全部を用いるなどの方法が考えられるが，膨大な数の画像データを有するデータベースや，画像データの入れ替わりの激しいデータベースへの適用が困難であり，また，特徴ベクトルの次元数が増加するにつれてより膨大な記憶領域が必要とされるという問題もある．

もし類似性検出に用いる特徴をユーザによる簡易な操作で柔軟に変更することができれば，前途の問題の解決策となることが期待できる．また，画像解析に基づく手法における類似とユーザの意図する類似との間のセマンティック・ギャップを埋める必要があると主張する研究も増えている．ユーザ側で特徴ベクトル空間の基底を変更可能なアルゴリズムが考案されれば，ユーザの意図に基づく特徴表現をユーザ自らが構成することが可能となる．今後は画像解析に基づく絶対的な指標に変わり，ユーザの検索意図や画像データの多様な属性に即した類似性検出を実現するような柔軟な特徴抽出手法の開発が期待される．

但し，用途によってはユーザの意図に従った特徴抽出を行わず，画像解析に基づく絶対的な指標を用いた方が良い場合もある．小原ら [36] は高度道路交通システム（Intelligent

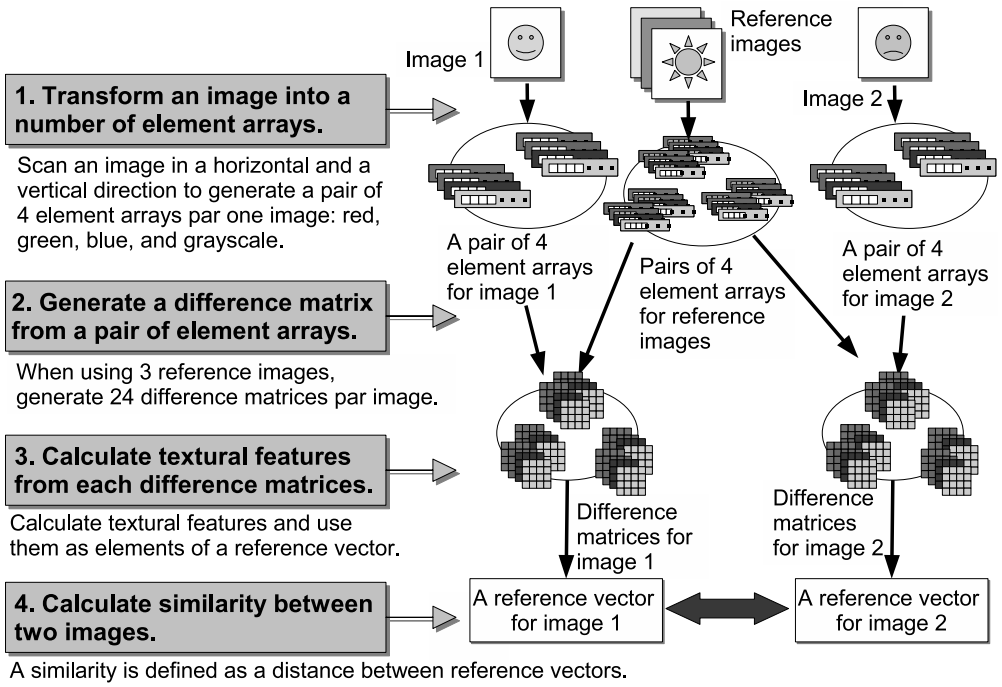


図 4.9　参照を用いた画像間類似性検出手法の概要

Transport Systems：ITS）の機能の一つである車両前方の環境情報の認識に注目し，階層型ニューラルネットワークによる道路標識の検出と自己組織化マップによる低次元化により，様々な形状や色をもつ道路標識の高速で高精度の認識の実現を目指した．このように高速かつ正確な処理が求められ，かつ検索意図が固定されているような場合には，画像の特徴を表現する特徴ベクトル空間の基底は固定である方が良い．

4.2.2　参照を用いた画像の特徴表現

概要

ここでは，3.1 において提案した特徴抽出アルゴリズムをもとに構築した画像間類似性検出手法について論じる．本手法では，予め選出しておいた “参照画像（Reference Image）” という複数の画像を用いて，特徴抽出を行いたい対象である “対象画像（Target Image）” の特徴を定量的に表現する “参照ベクトル（Reference Vector）” を算出する．参照ベクトルは，対象画像と参照画像から生成した複数の “差分マトリクス（Difference Matrix）” から抽出したテクスチャ特徴量を要素としている．

対象画像が持つ全ての特徴を，F_{all} とする．

$$F_{\mathrm{all}} = \begin{pmatrix} 1,1 & \cdots & m.1 \\ \vdots & \ddots & \vdots \\ 1,n & \cdots & m,n \end{pmatrix} \tag{4.11}$$

人が目視で画像の類似度を判定するとき，無意識のうちにその検索の目的や嗜好などの何らかの指標に基づき F_{all} から低次元の特徴ベクトル F_{sel} を抽出している．このとき，人の脳内には全ての画像が絶対的な尺度によりマッピングされているような特徴ベクトル空間が構築されているのではなく，既知の対象画像から抽出した特徴と新しく目にした対象画像の特徴を比較し，検索意図により適合する，もしくはより適合しないような小数の画像の特徴を代表値とし，これとの相対的な類似関係をもとに全体の類似度比較を行っていると考えられる．つまり，人は現時点での検索意図を関数として F_{all} を F_{sel} への写像を行っており，検索意図が変わるごとに関数は自動生成されていると考えることができる．

本手法では，複数個の参照画像を用いて間接的に画像の特徴を表現することにより，人が自然に行っている類似性検出プロセスに倣った類似性検出を行う．参照画像は，先に述べた人の類似性検出プロセスにおいて，代表値の役割を果たす．参照画像を軸とする特徴ベクトル空間上に F_{all} を写像することで，低次元の特徴ベクトルで人の検索意図に基づく特徴表現を行うことを目指す．多くの既存手法では，画像解析により対象画像から直接詳細な特徴を抽出している．しかし，このようにして得られた特徴量から全ての対象画像の特徴を統一された基準で表現するような低次元の射影を得るためには，高度な専門的な知識や膨大な試行錯誤が必要とされる．このため，人の意図に沿った類似性検出の実現は難しいと考えられる．

本手法では画像データベース内の全ての対象画像を参照ベクトル空間上の座標として表現することで，全ての対象画像の特徴を統一された基準で低次元の特徴ベクトルにより表現することができる．本手法における統一された基準，つまり類似性尺度は，対象画像の参照画像に対する類似度（あるいは非類似度）である．参照ベクトルを算出するということは，対象画像がどの参照画像にどの程度類似しているかという情報を定量化するということである．

このように，本手法では参照画像を用いた相対的な類似性尺度により類似性検出を行うため，対象画像から直接抽出した特徴量という絶対的な尺度を用いる既存の多くの手法に

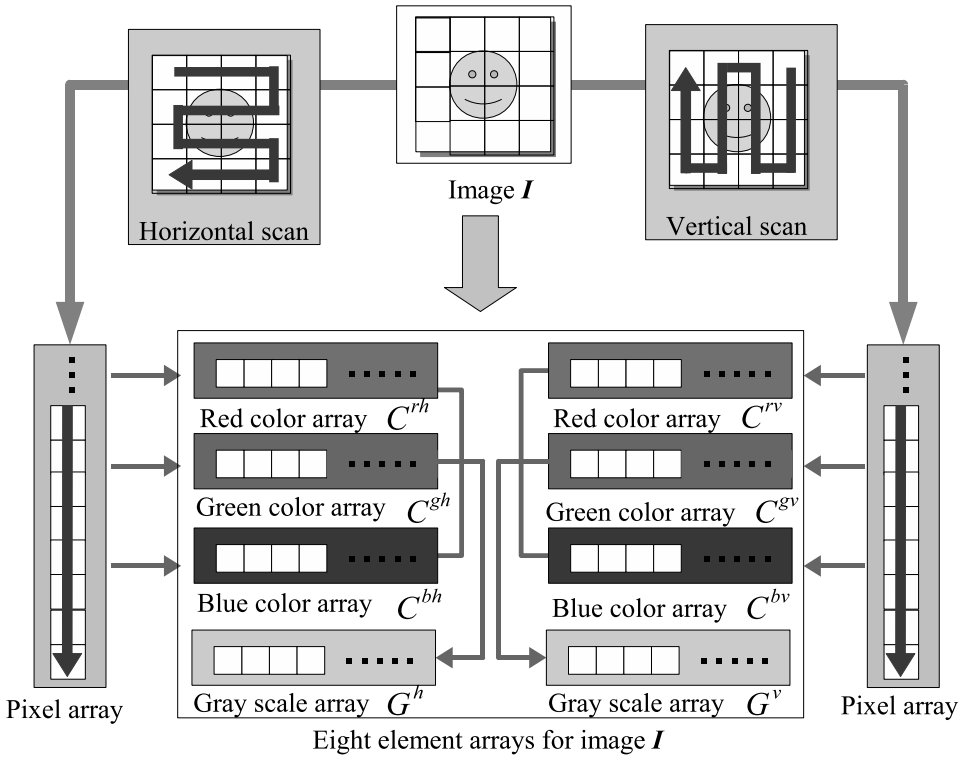


図 4.10　色成分の一次元配列化

比べ，より対象画像の多様な属性に対応し，人の意図に沿った類似性検出を行えることが期待できる．

前処理

本手法における前処理とは，参照画像と検索対象画像から参照ベクトルを算出するまでの処理のことを指す．前処理には一般に膨大な計算時間が必要とされるが，適切な参照画像が選出されていれば，以後既存の画像と大きく異なる属性を持つ画像が追加された場合や一部の画像が削除・修正された場合も，画像データベース内の画像全体の特徴量ベクトルを再計算する必要はない．

差分マトリクスの生成

本手法では，全ての画像のピクセル値を赤，緑，青の各色成分に分解し，それぞれを別個の配列に格納する．１つの参照画像の色成分配列と１つの検索対象画像の色成分配列との間で差分マトリクスを生成する．参照画像は複数であるため，各参照画像と検索対象画像との間には，色成分配列の数と同数の差分マトリクスが作られる．これら全てのマトリ

クスより抽出した特徴量を 1 つの検索対象画像の特徴ベクトルとして用いる．

Jacobs ら [37] にも指摘されているように，一般に RGB 表色系では輝度や各色要素を独立した値として扱わない．このため，本手法ではまず画像 I の画素からピクセル値を抽出する．この時，隣接する画素には類似した色が配置される場合が多いことから，4.10 に示すように，画像を縦方向と横方向にジグザグにスキャンし，それぞれ一次元配列 P_h^i，P_v^i に格納する．これらのピクセル値配列をそれぞれ “水平ピクセル配列（Horizontal Pixel Array）” および “垂直ピクセル配列（Vertical Pixel Array）” と呼ぶ．

続いて P_h^i，P_v^i より，シフト演算により赤，緑，青の色成分を抽出し，それぞれを一次元配列 C^{rh}, C^{gh}，C^{bh}，C^{rv}, C^{gv}, C^{bv} に格納する．これと同時に，この 3 つの色成分から輝度を表すグレースケールを算出し，これも同じく一次元配列 G^h, G^v に格納する．このような一次元配列を “要素配列（Element Array）” とよぶ．1 つの対象画像から得られたこれらの一次元配列を “対象要素配列（Target Element Array）” とよぶ．このうち横方向のスキャンで得られた対象要素配列を “水平対象要素配列（Horizontal Target Element Array）” とよび，縦方向のスキャンで得られ対象要素配列を “垂直対象要素配列（Vertical Target Element Array）” とよぶ．

この時点で 1 つの対象画像は 4 つの水平対象要素配列と 4 つの垂直対象要素配列，あわせて 8 つの対象要素配列により表現される．

次に，予め選出しておいた複数の参照画像のうちの 1 つから同様の手順で生成した 8 つの “参照要素配列（Reference Element Array）” $\hat{C}^{rh}$, $\hat{C}^{gh}$, $\hat{C}^{bh}$, $\hat{G}^h$, $\hat{C}^{rv}$, $\hat{C}^{gv}$, $\hat{C}^{bv}$, $\hat{G}^v$ と，先程検索対象画像から生成した 8 つの対象要素配列を，それぞれ赤，緑，青，グレースケールに分けて縦横に配置し，2 次元の差分マトリクス $D^{rh}(I_t, \hat{I}_u)$, $D^{gh}(I_t, \hat{I}_u)$, $D^{bh}(I_t, \hat{I}_u)$, $D^{gsh}(I_t, \hat{I}_u)$, $D^{rv}(I_t, \hat{I}_u)$, $D^{gv}(I_t, \hat{I}_u)$, $D^{bv}(I_t, \hat{I}_u)$, $D^{gsv}(I_t, \hat{I}_u)$ を生成する．ここで，I_t は対象画像を，$\hat{I}_u$ は参照画像を示す．1 つの対象画像の特徴を表すこれらのマトリクスは，参照画像の数と同じ数だけ生成される．図 4.11 に示すように，対象画像と参照画像がそれぞれ有する 8 つの要素配列から，1 種類の要素配列につき参照画像の数と同数，つまり 3 つずつの差分マトリクスが生成される．例えば参照画像の数が 3 つであればそれぞれの対象画像につき 24 ずつの差分マトリクスが生成される．このマトリクスの要素は，図 4.12 に示すように，それぞれの要素配列の対応する要素の差分である．図中では，差分マトリクス $D^{rv}(I_t, \hat{I}_u)$ が画像 I_t における赤色の情報を保持する対象垂直要素配列 C_t^{rv} と参照画像 $\hat{I}_u$ の赤色の情報を保持する参照垂直要素配列 $\hat{C}_u^{rv}$ によって生

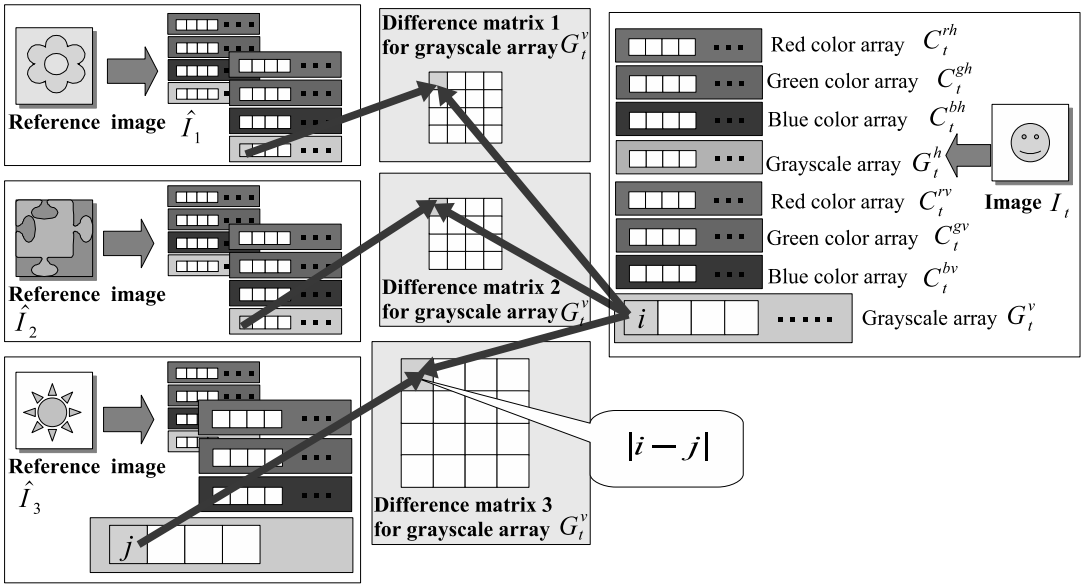


図 4.11　参照画像が 3 つの場合の差分マトリクス生成の例

成されている．

差分マトリクスは，1 つの検索対象画像の特徴を，複数の参照画像との要素の差分という形で相対的に表現している．この差分マトリクスの性質を定量化し，対象画像の特徴ベクトルとして用いる．

テクスチャ特徴量の抽出

本手法では差分マトリクスから特徴を抽出する手法として，Haralick[11] のテクスチャ特徴量を用いる．差分マトリクスをグレースケール画像とみなし，ASM，CON，COR，ENT，IDM の 5 種類のテクスチャ特徴量を算出する．このテクスチャ特徴量の詳細については 4.1.2 を参照されたい．

参照ベクトルを用いた類似度の算出

それぞれの差分マトリクスより算出したテクスチャ特徴量を用いて参照ベクトルを作成する．各画像データ間の類似度は，それぞれの画像に対し算出された参照ベクトル間のユークリッド距離とする．

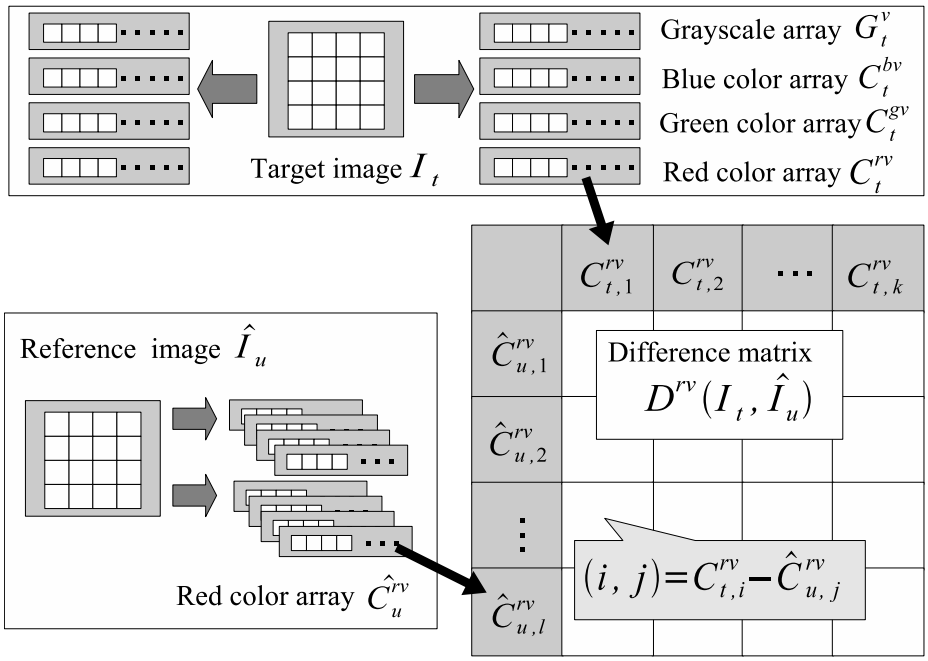


図 4.12　差分マトリクスの生成方法

類似度の定義

2 つの対象画像 I_1, I_2 間の類似度 S_{I_1,I_2} を画像 I_1 の参照ベクトル $\boldsymbol{r}_1$ および画像 I_2 の参照ベクトル $\boldsymbol{r}_2$ を用いて以下のように定義する。

対象画像 I_t の特徴を表す参照ベクトル $\boldsymbol{r}_t$ は次のように定義する．

$$\boldsymbol{r}_t = \left\{\boldsymbol{e}_1^t, \cdots, \boldsymbol{e}_{N_r}^t\right\} \tag{4.12}$$

ここで,

$$\boldsymbol{e}_n^t = \left\{e_n^t(1,1), \cdots, e_n^t(N_e,1), \cdots, e_n^t(1,m), \cdots, e_n^t(l,m), \cdots, e_n^t(1,N_t)\cdots, e_n^t(N_e,N_t)\right\} \tag{4.13}$$

であり，N_r は参照画像の数，N_e は要素配列の数，そして N_t はテクスチャ特徴量の数を表す．また，$\boldsymbol{e}_n^t$ は参照ベクトル $\boldsymbol{r}_t$ の要素のうち，n 番目の参照画像から得られた要素を表す．

画像 I_1 と I_2 の間の類似度 S_{I_1,I_2} は (4.14) 式に示すようにそれぞれの画像の参照ベクトル $\boldsymbol{r}_1$，$\boldsymbol{r}_2$ 間のユークリッド距離により求められる．

$$S_{I_1,I_2} = \|\boldsymbol{r}_1 - \boldsymbol{r}_2\| = \sqrt{\sum_{k=1}^{N_r}\sum_{l=1}^{N_e}\sum_{m=1}^{N_t} w_m \left(e_k^1(l,m) - e_k^2(l,m)\right)^2} \tag{4.14}$$

ここで，w_m（$0 \leq w_m \leq 1.0$）は m 番目のテクスチャ特徴量の重みである．

4.2.3 実験

概要

4.2.2 で述べた手法を実装した簡易画像検索ツールを作成し，評価実験によりその効果を検証した．

実験には携帯電話の写真撮影機能を用いて撮影された次のような画像集合を用いた．

[**実験データ**]

- 24 ビットカラー
- 120 × 160 ピクセル
- JPEG 形式

この画像集合は上記の規格に従った合計 24 の画像により構成される．これらは図 4.13 に示されるように，それぞれ 4 つの関連する画像により構成される 6 つのグループに分けられている．これらの画像は著者によりグループ分けされたものであり，本研究では同一グループに所属する画像はユーザの意図に基づき類似と判断されている画像として取り扱う．つまり，この画像集合の中から選出したある 1 つの画像に対し類似画像を求める場合，その画像と同一グループに属する 3 つの画像を正解画像とする．この方法により，我々はシステムにより検索結果として提示された画像がユーザの求める類似画像であるかどうかを判定する．

以降，画像集合に属する 24 の画像を “対象画像” とよび，この中からランダムに選出した 3 つの異なる画像を “参照画像” とし，別に選出した 1 つの “クエリ画像” の類似画像検索を行う．なお，参照画像はランダムに選出されるため，クエリ画像と同一のものが含まれる場合がある．

実験を行った計算機環境は以下の通りである．

[**計算機環境**]

- CPU: AMD Athlon(TM) 64 X2 Dual Core Processor 4400+ (2.22GHz)
- メモリ: 2GB RAM
- OS: Microsoft(R) windows XP Professional Version 2002 Service Pack 2
- Java version: JDK 5.0

パラメータ

今回の実験では，以下の w_m により参照ベクトルの要素全てを 0〜1 の間に正規化する．これは，事前に行った実験結果から，幾つかの要素が出力に与える影響が極めて小さいということが明らかになったためである．

$$w_m = \left(\frac{1}{max_m - min_m} \right)^2 \tag{4.15}$$

ここで，min_m，max_m はそれぞれ参照ベクトルにおける m 番目のテクスチャ特徴量の最小値，最大値である．

なお，w_m は (4.14) 式より以下の様に求められる．

$$\begin{aligned} S_{I_1,I_2} &= \sqrt{\sum_{k=1}^{N_r}\sum_{l=1}^{N_e}\sum_{m=1}^{N_t}\left(\frac{e_k^1(l,m) - min_m}{max_m - min_m} - \frac{e_k^2(l,m) - min_m}{max_m - min_m} \right)^2} \\ &= \sqrt{\sum_{k=1}^{N_r}\sum_{l=1}^{N_e}\sum_{m=1}^{N_t}\left(\frac{1}{max_m - min_m} \right)^2 \left(e_k^1(l,m) - e_k^2(l,m)\right)^2} \end{aligned} \tag{4.16}$$

手順

実験は次の手順により評価項目【E1】〜【E4】について行う．

[**手順**]

1) 画像集合からクエリ画像として 1 つの画像を選出する．
2) 画像集合から 3 つの画像をランダムに選出し，参照画像とする．
3) クエリ画像と全ての対象画像との間で類似度計算を行い，類似画像を検索する．
4) 2〜3 を 10 回繰り返し，その平均を求める．
5) 全ての対象画像について上記を繰り返す．

[**評価項目**]

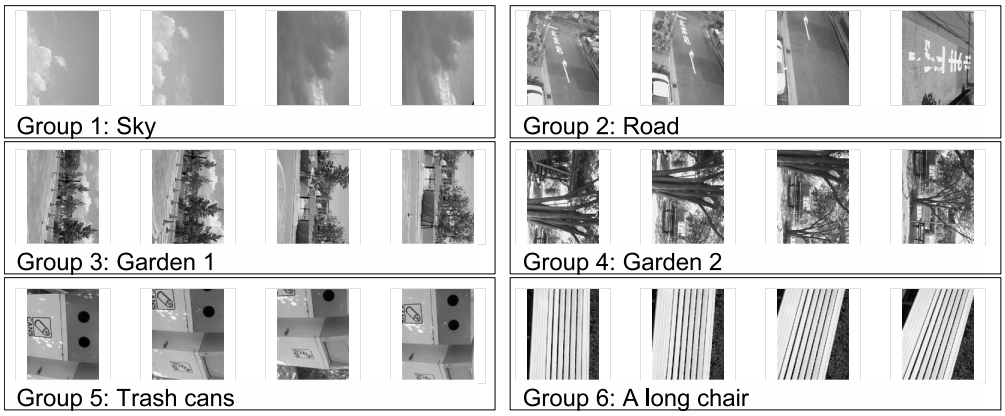


図 4.13　ユーザの意図に基づき 6 つのグループに分類された画像集合

【E1】参照ベクトル生成の平均計算時間

【E2】類似画像検索の平均計算時間

【E3】平均適合率

【E4】参照画像の種類と結果の関係

実験結果

1 つのクエリ画像につき毎回ランダムに選出した参照画像を使って 10 回ずつの検索を行った結果，全ての参照ベクトルの平均計算時間は 882 秒（14.7 分），1 回の検索の平均計算時間は 37 秒であった．次に，システムの性能を評価するため，平均適合率の計算を行った．上記手順により，1 つの画像につき 10 個の検索結果を得ている．まず，全ての結果について平均適合率を計算し，その平均を求め，グループの平均適合率として扱う．

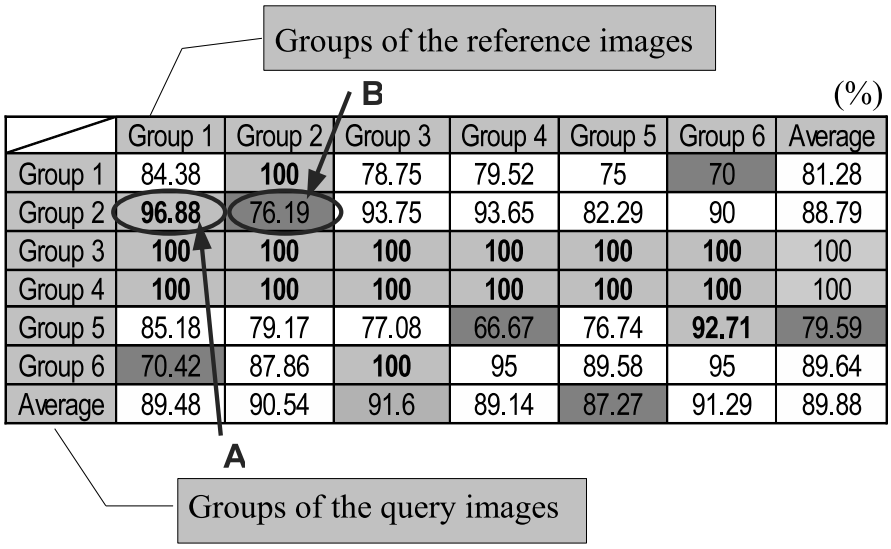
全体の平均適合率の平均は 94.63 % であった．このうち，グループ 1 の画像 1 と画像 2，グループ 3，グループ 4 の全ての画像，およびグループ 6 の画像 1 と画像 2 の検索においては常に最高の値である 100% という結果が得られた．グループ 1 の画像 3 を検索した場合のみ，最低の値である 57.43 % が得られた．この数少ない例外を除き，全体的な平均適合率は高く，本システムが画像検索に高い性能を示すことが確認された．

また，参照画像が結果に与える影響についても評価した．この評価指標として，次に定義する “Hit Ratio（HR）” を用いた．

$$\text{HR for a query image}(\%) = \frac{\text{Top 4 of retrieved images} \cap \text{Relevant images}}{\text{Top 4 of retrieved images}} \times 100 \tag{4.17}$$

HR は平均適合率に近いが, 検索順位を考慮しない点が異なる．平均適合率ではなくこの指標を用いた理由は，異なる参照画像を用いたときシステムが画像群をどのように分類するかを検証することを目的としているためである．もしシステムがあるクエリ画像に対し，類似画像全てを上位に提示していれば，検索結果上位 4 つは関連する画像，つまりクエリ画像そのものを含め，同じグループに属する 4 つの画像であるはずである．そのため，検索結果上位 4 位以内にいくつの関連する画像が含まれているかをパーセンテージで表すこの指標を用いて評価を行った．本実験では 1 つのクエリ画像につき 10 回の検索を行っている．この 10 回の検索結果から得られた HR の平均を単に HR と呼ぶこととする．

次に，全てのグループにおいて，この HR の平均値を算出した．図 4.14 は参照画像と HR の関係を示している．図中の各行はクエリ画像がどのグループから選出されたかを表し，各列はどのグループから少なくとも 1 つの参照画像が選出されたかを示す．セル内の数値はそれぞれ，グループごとに算出された HR の平均値である．例えば，**A** と示された “グループ 2” 行，“グループ 1” 列の交点に位置するセルはグループ 1 から選ばれた少なくとも 1 つの画像を含む参照画像を用いてグループ 2 の画像を検索した際の HR の平均を表している．この値は 96.88% であり, グループ 2 の画像をクエリ画像としたときの検索結果としては最高の値である．一方，**B** で示されるように，参照画像として 1 つ以上のグループ 2 の画像が用いられた場合の値は最低の 76.19% となっている．しかしながら，グループ 1 に属する画像をクエリ画像として用いた場合，グループ 2 の画像が参照画像として用いられた場合の値は最も高い．一方，グループ 3 と 4 の画像をクエリ画像として用いた場合の検索結果は参照画像を用いた場合でも同じであった．平均としては，平均 HR が 91.6% であるグループ 3 の画像が参照画像として最も高い性能を示し，グループ 5 の画像は 87.27% と，参照画像としての性能が最も低いと考えられる．クエリ画像としては，グループ 3 およびグループ 4 の画像はどの参照画像を用いても 100% という高い平均 HR を示し，グループ 5 の画像は 79.59% と最も低い平均 HR を示している．



	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Average
Group 1	84.38	**100**	78.75	79.52	75	70	81.28
Group 2	**96.88**	76.19	93.75	93.65	82.29	90	88.79
Group 3	**100**	**100**	**100**	**100**	**100**	**100**	100
Group 4	**100**	**100**	**100**	**100**	**100**	**100**	100
Group 5	85.18	79.17	77.08	66.67	76.74	**92.71**	79.59
Group 6	70.42	87.86	**100**	95	89.58	95	89.64
Average	89.48	90.54	91.6	89.14	87.27	91.29	89.88

図 4.14　HR の平均

まとめ

実験結果から，以下のことが明らかになった．

(1)24 ビットカラー，120 × 160 ピクセル，JPEG 形式，4 つの類似画像を有する 6 つのグループにより構成される画像データ集合を対象とし，参照画像 3 枚を用いて検索を行った結果，参照ベクトルの平均計算時間は 882 秒（14.7 分），1 回の検索の平均計算時間は 37 秒であった．

(2)1 つの画像につき 10 回の検索を行い，全ての結果について算出した平均適合率を平均した値は 94.63% であった．このうちグループ 1 の画像 3 を検索した場合のみ，最低の値である 57.43% という結果が得られたが，これ以外ではグループ 1 の画像 1 と 2，グループ 3 と 4 の全ての画像，そしてグループ 6 の画像 1 と 2 で 100% という結果が得られた．このことから，本システムが画像検索に高い性能を示すことが確認された．

(3) 参照画像の種類と検索結果の関係については，多くの場合について参照画像は検索結果に影響を与えていると考えられるが，いくつかの例外も確認された．これについてはさらに実験を行い，参照画像におけるいかなる特徴が検索結果に影響を与えているのかを突き止め，それに基づきどのようにしてユーザの意図に沿った検索を行うために適切な参照画像を選択すればよいかについて定義する必要がある．

4.3 考察

本章では，3.1 において提案した FRef アルゴリズムのソースコードおよび画像データへの適用を行った．FRef アルゴリズムの最も独創的な点は，参照データを用いた間接的な特徴表現である．対象データの特徴を抽出するために，データの構造や成分の解析により直接対象データから特徴抽出を行うのではなく，対象データと複数の参照データとの類似度により間接的に対象データの特徴を表現する．これは，データの種類によらず適用可能であるという本手法の汎用性の高さにも結びついている．

本章で提案したソースコード間類似性検出手法および画像間類似性検出手法は，人がデータを比較する際の特徴表現の近似を参照データを用いた特徴抽出アルゴリズムにより得ることで，1.1 にて挙げた「[機能 A] 人の意図を的確にコンピュータに伝えることを支援する機能」の実現を目指している．本章ではそれぞれの手法について実験を行い，その有効性を評価した．

ソースコード間類似性検出手法については，4.1.3 において，同じ意図に基づき作成された複数のソースコードを含む 434 個のソースコード集合に対し，FRef アルゴリズムに基づくソースコード間類似性検出手法を実装したツールを用いて実験を行った．実験結果から，98% 以上という高い平均適合率でクエリソースコードと類似する全てのソースコードを検索可能であることが明らかとなった．また，最適な参照データ数とテクスチャ特徴量の種類が確認された．本章で対象としているソースコード間類似性検出においては，個々のユーザ間の検索意図の違いはあまりないと考えられるため，現在用いられている，参照ソースコードをランダムに選出するという方法でも十分に実用的な検索を行うことが可能である．しかし，図 4.5 で示されるように，検索結果に含まれる誤差は主に参照ソースコードの違いによるものであると考えられるため，参照ソースコードを 3 個用いた場合に見られる，最大 ±1.5% の誤差も許容できないような用途への適用も可能とするためには，参照ソースコードの有する特徴とこれを用いて抽出される特徴について更なる分析を行う必要があると考えられる．

画像間類似性検出手法では，4.2.3 にて，あらかじめ人の手により類似画像ごとにクラスタリングされた小規模の画像集合を用いて実験を行った．FRef アルゴリズムに基づく画像間類似性検出手法を実装したツールにより検索を行った結果，人が類似画像とみなし

た画像に対し，ツールが約 95% と全体的に高い類似度を算出したことから，ユーザの意図に沿った類似画像検索を高い精度で行えたといえる．また，参照画像を特定の組み合わせで用いることにより，検索結果が異なるということが確認された．しかし現時点では参照画像の有するどの特徴が，ユーザの意図するどのような特徴に影響を与えるかということまでは解明されていない．ユーザが参照画像の選出により異なる意図に基づく検索を自由に行うことを可能とするため，今後アルゴリズムの再検討を行っていく必要がある．

本章における，FRef アルゴリズムのソースコードおよび画像に対する適用を通して，FRef アルゴリズムのもつ，データの特徴を間接的に表現するという特徴，そしてユーザ側で特徴抽出における類似性尺度が可変であるという特徴が，実社会におけるソースコード間類似性検出および画像間類似性検出という問題の解決にとって有益であるということ，そして人の意図に基づく特徴抽出の実現も期待できることが確認された．しかし現時点ではいずれの類似性検出手法においても人の意図を的確にシステムに伝えるには至っていないため，参照データと抽出される特徴の関係について詳細な分析を行う必要がある．

第 5 章

CM アルゴリズムの適用例

5.1 記述スタイルモデルによる授業課題ソースコードの盗用発見

5.1.1 背景

概要

一般に,「ソースコードの類似」とは，計算機に命令する処理内容や手順，つまり，アルゴリズムの類似を指す場合が多い．大規模ソフトウェア開発現場において，リポジトリからアルゴリズムが類似したソースコードやその断片を効率よく検出し，再利用や冗長な箇所の削除を行うことにより開発効率が向上することなどから，近年多くの類似性検出手法が提案されている．授業課題ソースコードにおける類似性検出もこのような研究の対象とする分野の一つである．授業課題ソースコードは比較的小規模なデータ集合ではあるが，担当教員が１つ１つ手作業で採点や盗用発見を行うことは困難であり，多大な労力を要することから，これらの作業を支援する手法の研究が数多く進められている．多くの既存手法では，授業課題ソースコードを対象とする場合も，ソースコードの構造や意味に着目した類似性検出を行っている．

しかし，授業課題ソースコードにおける盗用発見を対象とする場合，そこで着目すべき「ソースコードの類似」はこれまでに挙げたものと必ずしも一致しないということはあまり議論されていない．熟練者の作成したソースコードであれば，構造や制御の流れのようなアルゴリズム的な特徴の類似が盗用と判定される根拠となることが考えられる．しかし，広く行われている初学者を対象とした授業においては，このような特徴は出題時に教

員から条件や指示が与えられたりすることにより画一化され，実際に提出された課題ソースコードはアルゴリズム的に極めて類似したものとなる場合が多い．そもそも，短いソースコードから類似性検出に必要な特徴が十分に抽出できるかどうかという問題もある．また，インターネット上のコンテンツからの盗用や，第三者に代行させるといった盗用の場合，提出されたソースコード群に盗用元と盗用先のペアが存在しないため，従来の方法では発見することが難しい [38].

そこで，本手法では 3.2 の特徴抽出手法を用いて，ソースコードのアルゴリズム的な特徴ではなく表記上の特徴を作成者の記述スタイル特徴として表現し，作成者認証を行うというアプローチを提案する．このような特徴表現は，既存手法において問題視されている，偶然の一致による誤判定を防ぐための作成者認証システムとしての機能も期待できる．

本研究と類似したアプローチとしては，ソースコードの表面的な特徴を特徴量ベクトルとして定量化し，ニューラルネットワークにより重み付けを学習させるというものがある [39]．Engels は本研究と同じくソースコードの表面的な特徴が既存手法では失われている点について指摘しているが，ソースコードの特徴を定量化し，比較しているという点では他の既存手法と同様のアプローチであるといえる．

授業課題ソースコード盗用問題

近年，プログラミング授業の普及に伴い，授業課題ソースコードにおける盗用問題が深刻化している [40]．例えば，MIT の初級プログラミング授業において，30% の学生が盗用を認めたという報告がある [41]．また，Bull ら [42] によりインターネット上で行われた授業課題ソースコード盗用に関する 321 件のアンケートでは，50% が「近年盗用が増加している」と回答しており，「増加傾向は見られない」と回答しているのは 15% に留まっている．さらに，Dubin City University の Java 初級プログラミング授業では，約 300 名の受講者の半数が盗用に関わっていたことが確認されている [9]．このような現状において，教師がクラス全員の学生の答案を採点し，さらにその中に存在する可能性のある盗用を目視により発見することは難しく，多大な時間と労力を要する．

このようなことから，プログラミング授業課題における盗用発見の自動化は急務であるといえる．

盗用の定義

授業課題ソースコードにおける盗用の定義については，各研究において微妙に差異があり，グレーゾーンも存在するため，何をもって盗用と見なすかの線引きが難しいのが現状である．例えば，ある論文ではアンケート回答の分析結果から「学生が過去の課題で自分が作成したソースコードをコピーして新たな課題の回答を作成する行為」すら Self-Plagiarism として盗用の範疇に入れている [40]．しかし一般的には「学生が他の学生のソースコードを自作のものとして（改変などを加えて）提出する行為 [9]」を盗用の定義とする場合が多い．

盗用元，つまり盗用される側のソースコードについては，(1) 教本や論文 (2) インターネット上の Web ページ (3) 他の学生の答案の順に数が多いと報告されている [42]．また，電子掲示板での投稿による第三者への作成依頼や，業者への依頼も盗用の範疇に入れられる場合がある [9]．

本研究では，授業課題ソースコード盗用を「一部または全部を自らが作成していないソースコードを自作のものとして提出すること」と定義し，上に挙げたいずれのケースについても，授業課題ソースコード盗用問題とみなす．

盗用の偽装

ソースコード盗用が行われる際，単にコピーするのではなく何らかの偽装が行われる．既存研究により以下のような偽装が報告されている [43, 44, 45, 41]．

1. コメントの内容や体裁の変更．
2. 空白やインデントの入れ方の変更．
3. 式の等価な式への変更．
4. 識別子名の変更．
5. 被演算子の順序の変更．
6. 類似したデータ構造への置換（ex. integer を real に拡張する）．
7. 命令文の書き換え（ex. ”while not found do”を”while found = false do”に書き換える ）．
8. 冗長なステートメントや変数の挿入（ex. 不要な初期化や出力命令の追加）．
9. 独立したステートメントやコードブロックの順序入れ替え．

10. イテレーションの構造の変更（ex. while の変わりに repeat を使う．または for のかわりに while を使う）.
11. 選択文の構造変更（ex. ネストされた if 文を線形化．case の代わりに if を使用）.
12. 手続き呼び出しを手続き本文に書き換える.
13. 構造化されていないステートメントを取り込む.
14. 元のソースコードにコピーしたソースコード断片を結合する.
15. 異なる言語間で盗用を行う.

既存研究

授業課題ソースコードにおける盗用発見を目的として提案された既存手法は，i) 属性値に基づく手法，ii) 構造情報に基づく手法，iii) 意味・情報量に基づく手法，iv) その他の手法の 4 つに大別できる.

i) 属性値に基づく手法では，ユニークな演算子・被演算子の数やその出現頻度を元にプログラムのプロファイルを作成する．ソースコード中の演算子・被演算子の数や出現頻度をメトリクス値として用いる手法 [46]，ソースコード中の空白・非空白文字，空白行・非空白行，ネストの深さ，総文字数，大文字・小文字の数，ブロック数・メソッド数などをカウントし，それらの統計からメトリクス値として用いる手法などがある.

ii) 構造情報に基づく手法としては，ソースコードを抽象構文木で表現し 2 つの抽象構文木における共通の部分木を探す手法や，ソースコードをトークン列化し，2 つのトークン列における共通部分列を探す手法など，構造の比較を行う手法が提案されている [47, 48, 29]. また，抽象的にアルゴリズムを表現するチャートでソースコードを表現し，共通部分木内のトークンをさらに比較する手法がある [49]．一般にこのような手法ではまず構文解析によりソースコードをトークン化し，次のこのトークン列同士を比較する.

ii) 意味・情報量に基づく手法には次の様なものがある．鷹岡ら [21] は，プログラムの字句列の類似では上記の偽装のような偽装が見破れない場合が多く，誤検出の可能性が高いことを指摘し，プログラムの制御とデータの流れや手続き間の呼び出し関係をグラフ表現し，比較することによりプログラムの意味レベルにおける類似性の検出を行っている．また，ソースコードをデータや制御の流れを表すプログラム依存グラフにおける共通部分グラフを探す手法や [50]，ソースコードをトークン列化し，Kolmogorov Complexity を用いて 2 つのシーケンス間で共有された情報量を測る手法が提案されている [51].

iv) ソースコードの特徴を抽出する様な手法ではないが，盗用発見手法としては他に以下のようなものもある．

Daly らは課題提出用の環境を構築し，提出されたソースコードに擬似的な電子透かしを挿入することにより作成者識別を行っている [9]．擬似電子透かしはソースコードに直接コメント文字列として挿入されるため，脆弱ではあるが，他の手法の様に偶然の類似を盗用と誤判定するおそれがない．また，完成したソースコードの提出以外にも，提出されたソースコードの内容についてのインタビューを行うという方法や，ソースコード作成中に定期的にログを出力するような開発環境を導入し，盗用しにくいような環境を整えるというアプローチも提案されている [44]．

なお，ソースコード盗用問題解決手法を提案している論文の多くでは，既存手法を属性情報による類似性検出手法と構造情報による類似性検出手法の 2 つに分類している．産業用の大規模ソースコードに対する類似性検出手法で見られる，識別子名やコメントの内容をメトリクスとして用いる様な手法は盗用の偽装を考慮してか，一般に授業課題ソースコードの盗用発見手法としては用いられていない．

また，既にいくつかの商用のレポート盗用発見ソフト販売されているが，代表的なツール（*Findsame*，*Eve2*，*Turnitin*，*CopyCatch*，*WordCHECK*）について調査した文献 [42] によると，実際にツールを使用して検証した結果，いくつかのツールの出力結果に異常が見られ，また，全てのタスクについて許容できるレベルのパフォーマンスが得られるツールは存在しないと報告されている．

既存手法の問題点

一般に，プログラミング授業において作成されるソースコードは，商用ソフトウェアなどと比べて行数が少ない場合が多いため，得られる特徴量は必然的に少なくなる．また，ある出題意図に基づき，教員の指導の下で作成されるソースコードは，必然的に類似してしまう場合が多い．

既存手法では，ソースコードの意味的・構造的特徴を i) メトリクス値により表現，ii) グラフまたはトークン列により表現している．i) の場合は特徴量ベクトル空間上の距離を求め，ii) の場合は共通する部分木や部分列を検出してソースコード間の類似度比較を行っている．

いずれの方法においても，まずソースコードに対し何らかの標準化処理を行って表面的

な差異を減少させ，次にソースコードのアルゴリズム的な特徴を比較している．このようなアプローチは，大規模ソースコードにおけるコードクローン検出や，異なる意図を持って作成されたソースコードが混在する集合のクラスタリングなどにおいて有効である．

しかし，授業課題ソースコードの様に，ある共通の意図を持って作成された小規模のソースコード集合を対象とする場合，標準化の過程でノイズとして無視される微小な差異にこそ重要な特徴が含まれている．つまり，表面的な差異を除外したソースコードの比較を行ったとしても，それは出題意図であるアルゴリズムの比較であり，類似してしかるべき部分を比較していることに他ならない．特に ii) の各手法においては，同じ意図を持って作成された短いソースコードを対象とした場合，偶然の一致を盗用と誤判定する可能性が危惧されるが，これを防ぐにはソースコード自体の特徴よりも作成者のコーディング特徴に着目すべきである．

また，ソースコードの特徴を比較する手法の問題点として，提出されたソースコード集合の中に，盗用元と盗用先のソースコードが含まれている必要があり，5.1.1 で触れたインターネットや教本などからの盗用やソースコード作成を第三者に代行させるような盗用を発見できないということが挙げられる．

そこで本研究では，アルゴリズム的な特徴ではなく，ソースコードの表記上に見られる表面的な特徴，すなわち作成者の記述特徴に着目した手法を考案した．本手法では従来の様に提出されたソースコード間で類似性検出を行うのではなく，作成者の記述特徴をモデルで表現し，モデルとソースコードの間で作成者認証を行うことで盗用を発見する．

5.1.2 記述スタイルモデルによるソースコード記述特徴の定量化

概要

ここでは，3.2 において提案した **CM アルゴリズム**[38] をもとに構築した，授業課題ソースコードにおける盗用発見手法について論じる．本手法では，1 つ以上のソースコードから抽出された作成者の記述特徴を記述スタイルモデルにより表現する．新たに提出されたソースコードを入力記号系列としてモデルに与えて出力確率を計算し，作成者認証を行う．本手法の概要は図 5.1 の通りである．以下にその手順を示す．

1) **［記述規則の規定］**学生のソースコードから等しく記述スタイルを抽出するため，基準となる記述規則を規定する．記述規則には例えばコメントの体裁など，ソース

コードの内容に関わらない表記上のもので，かつ多くのソースコードに共通して現れるものを選ぶ.

2) **［記述スタイルモデルの定義］**記述規則に基づき，単数または複数のモデルを定義する．これを記述スタイルモデルとよび，学生の記述特徴はこのモデルの構造やパラメータとして表現される.

3) **［記述スタイルの抽出］**記述規則に基づき，既提出ソースコードを字句解析によりトークン列化する．このとき，必要であればトークンの標準化を行い，記述スタイルを保持する記号系列に変換する．記述規則が局所的な記述スタイルを表すものであれば，この処理により 1 つのソースコードから複数の記号系列が生成される．これらをモデルの学習データとして用いることにより，記述スタイルを抽出する．表記ゆらぎによるノイズを最小化するため，学習に用いるソースコードは複数であることが望ましいが，それらは全て本人によって作成したものであることが確証されねばならない.

4) **［モデルの学習］**ソースコードから得られた記号系列を入力データとして記述スタイルモデルに学習させる．これにより，作成者固有の記述特徴はモデルのパラメータとして表現される.

5) **［作成者認識］**新たに提出されたソースコードが提出した学生本人により作成されたものであるかどうかは，ソースコードを入力記号系列として記述スタイルモデルに与えたときの記述スタイルモデルの出力確率により判断する．入力記号系列やモデルが複数である場合は，全ての出力確率の平均を判定基準として用いる．現在，盗用か否かを判定する閾値は設定されていないため，モデルが他の学生ソースコードに対し出力した確率よりも高ければ本人のものによる可能性が高く，逆に極端に低い値であれば本人の作成したソースコードでない可能性が高いと判断する.

記述規則と記述スタイルモデル

記述規則は，ある作成者がソースコードを記述する際に意識的に，または無意識のうちに行っているトークンの選択・配置であり，本手法における類似性尺度である．これをトークンの出現確率や共起確率として表したものが記述スタイルであり，本手法ではこの記述スタイルにより作成者の記述特徴を表現する．記述規則により決定された記述スタイルは，具体的には記述スタイルモデルの構造やパラメータとして表現される．本手法では

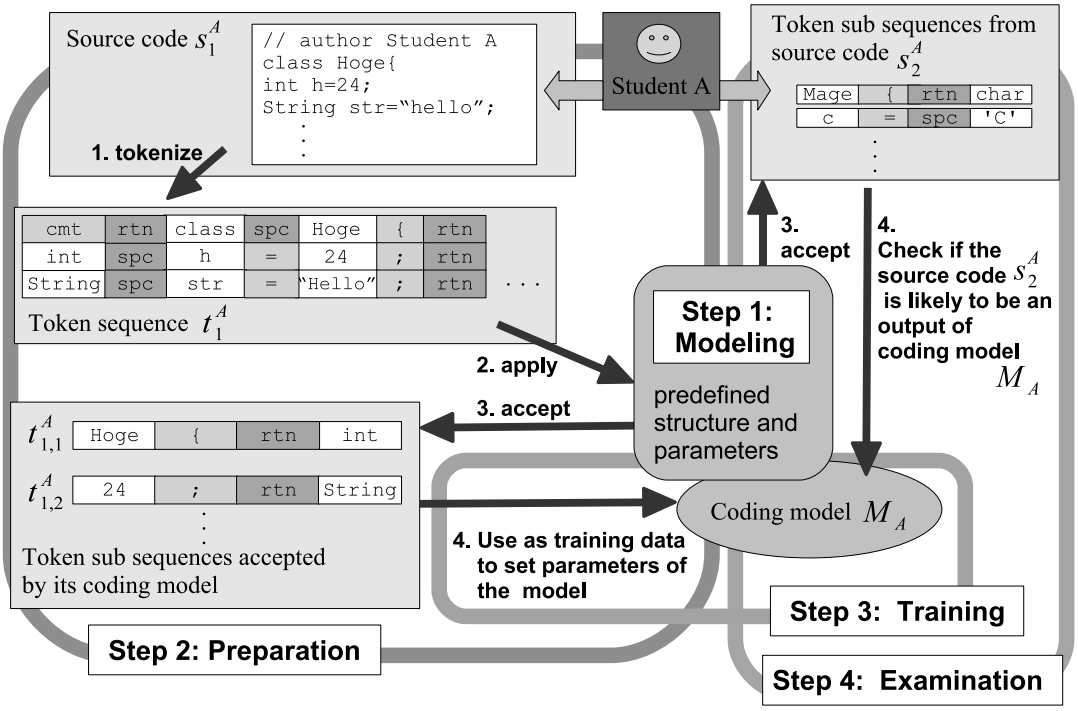


図 5.1　記述スタイルモデルによる授業課題ソースコード間盗用発見手法の概要

記述規則に従いソースコードをモデルの入力データである複数の記号系列に変換し，これをモデルに学習させることにより記述スタイルとして定量化する．作成者のソースコード中で，記述ルールに従った記述を記述スタイルと呼び，これを抽出し，記述スタイルモデルと呼ぶ．記述スタイルは記述スタイルモデルの上では構造やパラメータ上の特徴として表される．

作成者の記述特徴を区別する尺度としては様々なものが考えられるが，本論文では，Java のソースコードを対象とした次の様な記述規則を規定する．

- Java のソースコードにおいて頻出する記号を用途や形の類似したものごとにまとめたものを，表 5.1 中 (1) の，14 種類の “基準トークン（Basing Point Token）” とする．
- (2) の 1 文字から 4 文字までの空白，タブ，そして改行の計 6 種類のトークンを “着目トークン（Identification Token）” とする．
- 上記以外のトークンは全て (3) の “その他のトークン（Other Tokens）” に含める．
- 提出されたソースコードを上記 3 グループのいずれかのトークンからなるトークン列に変換し，基準トークンの前後の着目トークンの出現傾向を記述特徴として記述

スタイルモデルにより表現する．

前処理として行われるソースコードのトークン部分列集合への変換は，表 5.1 の分類に従い以下の様に行われる．

例えば，学生 A の作成したソースコード A は前処理により “トークン列（Token Sequence）” に変換される．さらにトークン列中に基準トークンが出現するたびにその前後のトークンは記述スタイルを表す最小単位ごとに “トークン部分列（Token Sub Sequence）” として切り出される．こうして得られたトークン部分列集合は学生 A の記述スタイルモデルから観測される系列データである “出力系列（Output Sequence）” の集合とみなされる．ソースコード A が学生 A によって作成された確率は，記述スタイルモデルが出力系列を出力する確率により示される．

記述スタイルモデルは，基準トークンを表す状態 s_3 を中心として，その前後に位置し着目トークンを表す s_2 および s_4 と，さらにその前後に位置しその他のトークンを表す s_1 により構成される．s_1 は開始状態・終了状態を兼ねている．状態間は記述規則に基づき有向リンクにより接続されており，それぞれに遷移確率が割り当てられる．また，各状態は単一または複数の記号とこれが観測される確率を保持する．

本稿では 14 種類の基準トークンを規定しているため，1 種類の基準トークンを s_3 により表す同様のモデルが全部で 14 個作られる．つまり，1 つのソースコードから 14 種類の基準トークンそれぞれを中心とした前処理を行って 14 セットのトークン部分列集合を生成し，それぞれ対応する記述スタイルモデルの学習データとして用いる．こうして得られた 14 種類の記述スタイルモデルにより，1 つの記述スタイルを表現する．

図 5.2 はこの記述規則により作成し，学習によりパラメータを設定したモデルの例である．これは 14 種類の基準トークンのうち，obr に関わる記述特徴を表現するモデルである．他の 13 種類のモデルも s_3 で表す基準トークンの種類が異なる以外は同一の構造を持ち，それぞれの学習により得られた観測確率と出力確率がセットされている．

記述スタイルモデルによる作成者認識

本手法では，学生 A のソースコードを N_A 個のトークン部分列集合 $\boldsymbol{X}_A = \{x_n | 1 \leq n \leq N_A\}$ とみなし，既提出ソースコードにより予め作成しておいた学生 A の記述スタイルモデル M_A と，トークン部分列 x_n の尤度 $P(M_A, x_n)$ の平均値 $M_A(X_A)$ を用いてソースコードの作成者認識を行う．

表 5.1 “基準トークン”, “着目トークン”, “その他のトークン” の具体例

(1) 基準トークン	
obr	opening brace (ex. “{”)
cbr	closing brace(ex. “}”)
orb	opening round bracket(ex. “(”)
crb	closing round bracket(ex. “)”)
osb	opening squared bracket(ex. “[”)
csb	closing squared bracket(ex. “]”)
aop	arithmetic operators(ex. “+”, “-”, “*”, “/”, “%”, “++”, “--”)
aso	assignment operators(ex. “=”, “+=”, “-=”, “*=”, “/=”, “%=”,)
cop	comparative operators(ex. “<”, “>”, “<=”, “>=”, “==”, “!=”) and conditional operators(ex. “?”)
lob	logial operators (ex. “&”,“|”, “&&”, “||”) and bit wise operators(ex. “^”, “!”, “<<”, “>>”, “|=”, “^=”, “&=”)
cmt	comments (ex. “// *bra bra*”, “/* *bra bra* */”)
cln	colon(ex. “:”)
cma	comma(ex. “,”)
scl	semi colon(ex. “;”)

(2) 着目トークン	
sp1	single space
sp2	2-lettered spaces
sp3	3-lettered spaces
sp4	4-lettered spaces
tab	tabs
rtn	linefeeds

(3) その他のトークン	
els	reserved words, identifiers, etc. (ex. ““”, “””, “.”)

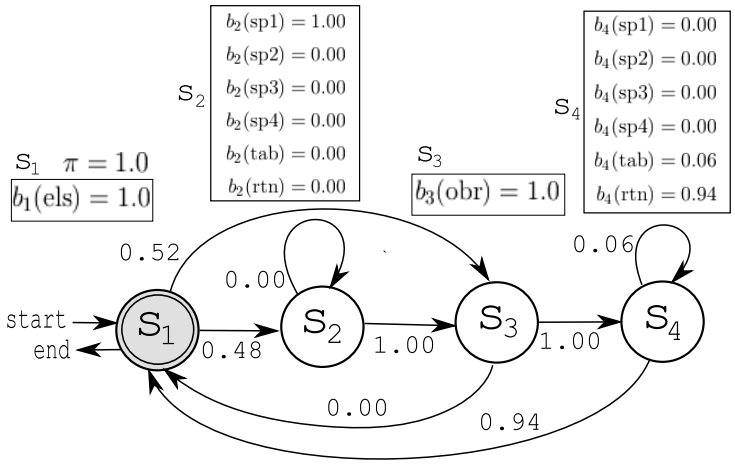


図 5.2　記述スタイルモデルの一例

ここで，長さ T_n の $x_n(x_n = \{\sigma_{k1}, \cdots, \sigma_{kT_n} | 1 \leq t \leq T_n,\ \ 1 \leq k \leq N\})$ は以下の定義に基づき，観測確率変数列 $O_{1:T_n}$ として表される．

$$O_{1:T_n} = \{O_1, \cdots, O_{T_n} | 1 \leq t \leq T_n,\ \ O_t = P(\sigma_{kt} | S_t = \{s_i\})\} \tag{5.1}$$

以上により，作成者認識に用いられるモデルの平均出力確率は，以下の 2 式により求められる．

$$P(M_A, x_n) = \sum_{\mathrm{all} S_{1:T_n}} \prod_{i=1}^{L} \pi_i b_{S_1}(O_1) \prod_{t=2}^{T_n} a(S_{t-1}, S_t) b_{S_t}(O_t) \tag{5.2}$$

$$M_A(X_A) = \frac{\sum_{n=1}^{N_A} P(M_A, x_n)}{N_A} \tag{5.3}$$

5.1.3　実験 1

概要

前章で説明した手法を実装したシステムを構築し，実際にソースコードを使ってモデルの作成と作成者認識を行い，評価を行った．今回の実験では，5.1.2 において規定された記述規則に基づいて抽出された記述スタイルを保持する記述スタイルモデルを用いて以下の各項目について評価を行った．

[**評価項目**]

【E1】記述スタイルモデルがソースコード作成者の記述スタイルを表現することができているか.

【E2】複数のソースコードをある記述スタイルモデルに入力した場合，その出力確率によってモデルの保持する記述スタイルと同じ記述スタイルモデルで作成されたソースコードを判別することができるか.

【E3】ある作成者未知のソースコードが与えられたとき，複数の記述スタイルモデルにこれを入力し，その出力確率によって正しい作成者を見つけることができるか.

実験内容

実験用のソースコードとして，5 名の被験者に図 5.3 に示す 5 種類の課題を与え，それぞれについて Java でソースコードを作成させた．つまり，同じ意図を持って作成された 5 つのソースコード 5 種類，計 25 個のソースコードを用いて実験を行う．5 名の被験者については，以降 A，B，C，D，E，とアルファベットで区別する．作成された 5 種類のソースコードはそれぞれ 1，2，3，4，5，と番号で区別する．例えば，被験者 A が作成した課題 1 のソースコードは A1 と呼ぶこととする．図 5.4 および図 5.5 はそれぞれ A，B，C，D，E によって作成された課題 1 のソースコードである．実験は次の手順により行った.

まず，全ての被験者によって作成された課題 1〜課題 4 のソースコードをモデルに学習させ，A，B，C，D，E の記述スタイルを表す記述スタイルモデル群を作成する．例えば，A の記述スタイルを表現する記述スタイルモデル群を M_A とする.

$$M_A = \left\{ M_A^{\mathrm{obr}}, M_A^{\mathrm{cbr}}, \cdots, M_A^{\mathrm{cma}}, M_A^{\mathrm{scl}} \right\} \tag{5.4}$$

つまり，M_A はソースコード 1〜4 から抽出された A の記述特徴を保持する 14 種類の記述スタイルモデルにより構成されるモデル群である．次に，M_A，M_B，M_C，M_D，M_E に A5，B5，C5，D5，E5 を入力し，それぞれ平均出力確率を計算する．ここで，例えば M_A の平均出力確率とは 14 種類の記述スタイルモデルから出力された 14 個の出力確率の平均を指す．具体的には，次のような手順で求められる.

1) 各記述スタイルモデルはソースコードから生成された複数個のトークン部分列が入力される度に出力確率を出力する．これら全ての出力確率の平均を，記述スタイルモデルの出力確率とする.

以下の5つの問題の指示に従い，Javaで5つのクラスを作成して下さい．
全てのクラス内に1つ以上のコメント入れて下さい．

■問題１■以下の様なかけ算九九の表を表示するプログラムを作成して下さい．

1×1=1 1×2=2 1×3=3 1×4=4 1×5=5 1×6=6 1×7=7 1×8=8 1×9=9
2×1=2 2×2=4 2×3=6 2×4=8 2×5=10 2×6=12 2×7=14 2×8=16 2×9=18
3×1=3 3×2=6 3×3=9 3×4=12 3×5=15 3×6=18 3×7=21 3×8=24 3×9=27
4×1=4 4×2=8 4×3=12 4×4=16 4×5=20 4×6=24 4×7=28 4×8=32 4×9=36
5×1=5 5×2=10 5×3=15 5×4=20 5×5=25 5×6=30 5×7=35 5×8=40 5×9=45
6×1=6 6×2=12 6×3=18 6×4=24 6×5=30 6×6=36 6×7=42 6×8=48 6×9=54
7×1=7 7×2=14 7×3=21 7×4=28 7×5=35 7×6=42 7×7=49 7×8=56 7×9=63
8×1=8 8×2=16 8×3=24 8×4=32 8×5=40 8×6=48 8×7=56 8×8=64 8×9=72
9×1=9 9×2=18 9×3=27 9×4=36 9×5=45 9×6=54 9×7=63 9×8=72 9×9=81

■問題２■簡単な入出力プログラムを作成して下さい．

・プログラムを起動すると「名前を入力して下さい」と表示
・名前を入力してEnterを押すと「こんにちは，○○（入力した名前）さん」と
表示する．

■問題３■金額と税率を入力すると合計金額を表示するプログラムを作成して下さい．
（※金額の計算用のメソッドを作ってください）

・プログラムを起動すると「金額を入力して下さい」と表示．
・金額を整数で入力してEnterを押すと「税率を入力して下さい」と表示．
・税率を小数で入力すると，合計金額を「○○円です」と表示する．

■問題４■ユークリッドの互除法を使って最大公約数を求めるプログラムを作成して下さい．

・「１つ目の自然数を入力して下さい」と表示．→ int num1に代入．
・自然数を入力してEnterを押すと「2つ目の自然数を入力して下さい」と表示．
→int num2に代入．
（※ここで，自然数以外の数が入力されていれば（num1<=0 || num2<=0)エラー
メッセージを表示．）
・num2>0ならば（while文）
　・まず，現時点でのnum1とnum2を表示．
　・num1をnum2で割った余りをint tmpに代入．
　・num1にnum2を代入．
　・num2にtmpを代入．
・計算が終わったら，「最大公約数は○○（num1）です」と表示する．

■問題５■問題４のプログラムを，再帰を使ったプログラムに書き換えて下さい．

※処理内容は基本的に問題４と同じ．ただし，以下の変更を行う．
　・while文で行っていた処理を再帰関数のメソッドとして独立させる．
　・入力された2つの数が自然数であればメソッドを呼び出し，最大公約数を求める．

図 5.3　課題の内容

```
1 ↵
2 public class Class1 {↵
3 ↵
4 ^    public static void main(String[] args) {↵
5 ^    ^    for (int i = 1 ; i <10 ; i = i +1){↵
6 ^    ^    ^    for (int j = 1 ; j <10 ; j = j +1){↵
7 ^    ^    ^    ^    String s = i + "×" + j + "=" + i*j + " ";↵
8 ^    ^    ^    ^    System.out.print(s);//出力↵
9 ^    ^    ^    }↵
10^    ^    ^    System.out.println();//改行↵
11^    ^    }↵
12^    }↵
13↵
14}↵
```

A1

```
1 package javasrc2;↵
2 ↵
3 public class Class1{↵
4 ↵
5 ^    public static void main (String[] args){↵
6 ^    ^    //九九の表の計算↵
7 ^    ^    for(int i=1; i<10; i++){↵
8 ^    ^    ^    for(int j=1; j<10; j++){↵
9 ^    ^    ^    ^    System.out.print(i+"×"+j+"="+i*j+" ");↵
10^    ^    ^    //改行↵
11^    ^    ^    ]System.out.println();↵
12^    ^    }↵
13^    }↵
14}↵
```

B1

```
1 public class MultiTab {←
2     public void show() {←
3         for( int i = 1 ; i <= 9 ; i++ ) {←
4             for( int j = 1 ; j <= 9 ; j++ ) {←
5                 System.out.print(i + "テ " + j + "=" + (i * j) + " ");←
6             }←
7             System.out.println();←
8         }←
9     }←
10←
11    public static void main(String[] args) {←
12        MultiTab mt = new MultiTab();←
13        mt.show();←
14    }←
15}←
```

C1

図 5.4　被験者 A，B，C により作成されたソースコード 1

```
 1 /*問１　九九算*/↵
 2 ↵
 3 public class qqsann {↵
 4 ↵
 5 ^    public static void main(String[] args) {↵
 6 ^    ^    int i, j;↵
 7 ^    ^    for (i = 1; i <= 9; i++){↵
 8 ^    ^    ^    for (j = 1; j <= 9; j++){↵
 9 ^    ^    ^    ^    System.out.print(i+"*"+j+"="+ i*j + "¥t");↵
10 ^    ^    ^    }↵
11 ^    ^    System.out.println();↵
12 ^    ^    }↵
13 ^    }↵
14 }↵
```

D1

```
 1 ↵
 2 public class kakesann {↵
 3 ↵
 4 ^    /**↵
 5 ^     * @param args↵
 6 ^     */↵
 7 ^    public static void main(String[] args) {↵
 8 ^    ^    // TODO Auto-generated method stub↵
 9 ^    ^    for(int i = 1;i < 10;i++){    //tate↵
10 ^    ^    ^    for(int j = 1;j < 10;j++){ //yoko↵
11 ^    ^    ^    ^    System.out.print(i + "x" + j + "=" + i*j + " ");↵
12 ^    ^    ^    }↵
13 ^    ^    ^    System.out.print('¥n');↵
14 ^    ^    }↵
15 ^    }↵
16 }↵
```

E1

図 5.5　被験者 D，E により作成されたソースコード 1

2) 1) で求めた 14 個の記述スタイルモデルの出力確率の平均を，記述スタイルモデル群 M_A の平均出力確率とする．

実験結果

前小節に示された評価項目【E1】～【E4】について実験を行った結果は以下の通りである．

【E1】図 5.6 および図 5.7 は 5 名の被験者の記述スタイルを表す記述スタイルモデル群，M_A，M_B，M_C，M_D，M_E のうちの 1 つであり，基準トークン “obr” に関わる記述スタイルを表すモデルである．図 5.4 および図 5.5 において示されたソースコードの記述スタイルのうち，“obr” に関わるスタイルの違いがそれぞれのモデルのパラメータの違い

として表現されていることが確認できる．

【E2】および【E3】については，表 5.2 に示される平均出力確率をもとに評価する．表の各行は入力されたソースコードを，各列はモデルを表す．なお，括弧内の数値は分散である．

【E2】については，例えば，表中 (1) で示される M_A 列に注目すると，A5 が入力された場合の平均出力確率は 74.79% であり，B5（71.99%），C5（64.11%），D5（64.21%），E5（73.61%）と比べ最も高い値である．これは，A の記述スタイルを表す M_A が，A によって作成されたソースコードに対し他の作成者よりも高い値を出力した，つまり，A の記述スタイルを M_A が判別したと考えることができる．この他，M_B，M_C，M_D，についても，同様の結果が得られた．しかし M_E については E によって作成された E5 よりも C によって作成された C5 に対し高い値が出力され，E5 には 2 番目に高い平均出力確率が出力された．

【E3】については，例えば，表中 (2) で示される A5 行に注目すると，ソースコード A5 が与えられた場合の各記述スタイルモデル群の平均出力確率はそれぞれ，M_A が 74.49%，M_B が 71.10%，M_C が 67.63%，M_D が 71.31%，M_E が 64.34% と，M_A の出力した値が最も高かった．つまり，作成者未知のソースコードとして A5 が与えられた場合，最も高い値を出力したモデル群が M_A であることから，未知のソースコードの作成者は A であると推定され，実際にこのソースコードは A5 であることから的中していることが確認できる．B5，C5，D5，についても同様の結果が得られたが，E5 については，最も高い値を出力したモデル群は M_A，最も低い値を出力したモデル群は M_E であり，これはつまり，最も可能性の高い作成者は A，最も可能性の低い作成者は E いうことであり，全く見当違いな結果であることが確認された．

考察

評価項目【E1】～【E3】より，本手法で提案する記述スタイルモデルが実際にソースコードの表面的な特徴を表現可能であることが確認された．また，被験者 E を除く 4 名の被験者については，記述スタイルモデルを用いた作成者認証と未知のソースコードの作成者推定に全て成功した．これらの結果から，本手法は授業課題ソースコードにおける盗用発見に有効であると考えられる．しかし，被験者 E のソースコードについて，誤判定があったことも事実である．この原因としては，次のようなことが考えられる．

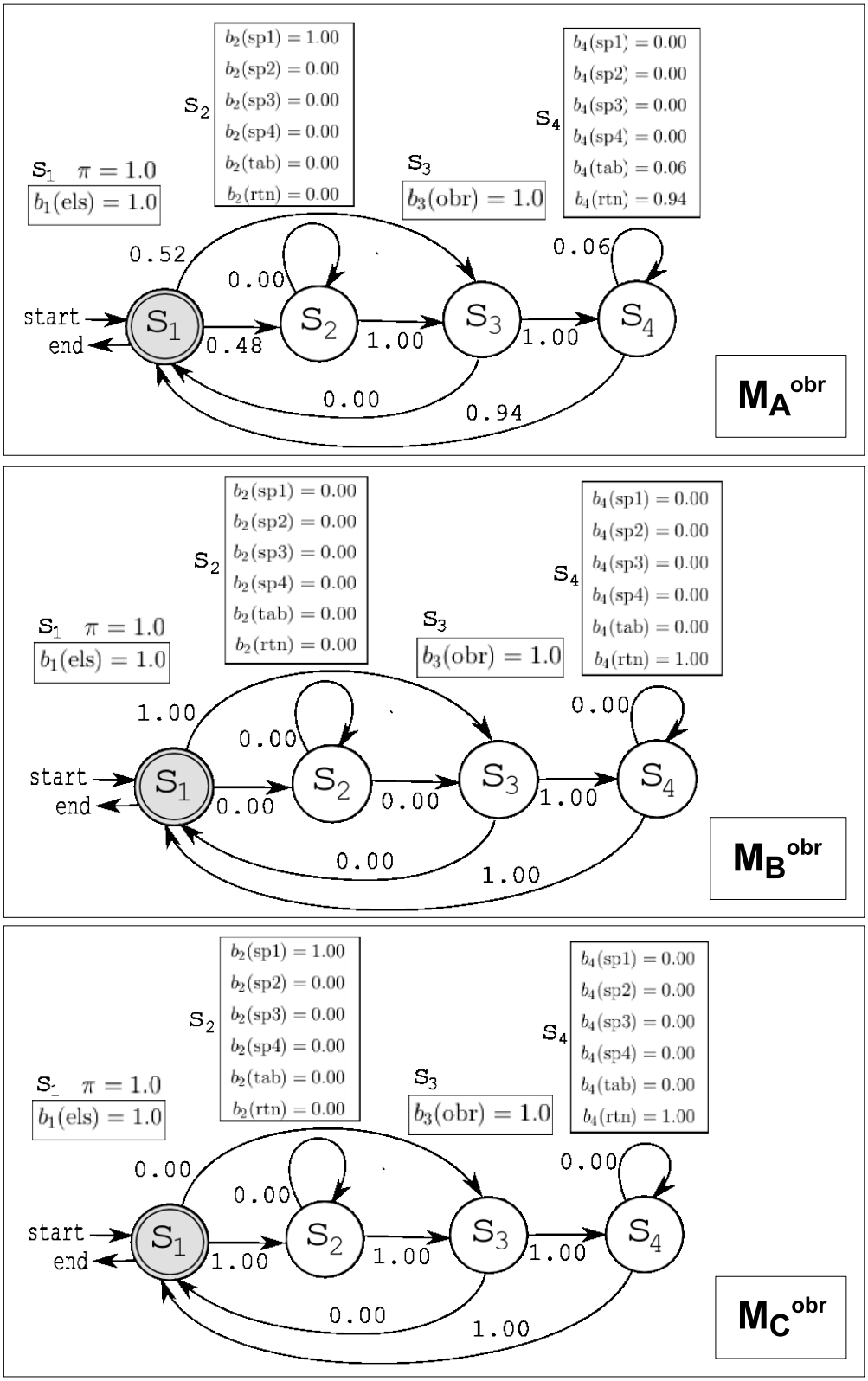


図 5.6　被験者 A，B，C の記述スタイルを表す記述スタイルモデルの一例

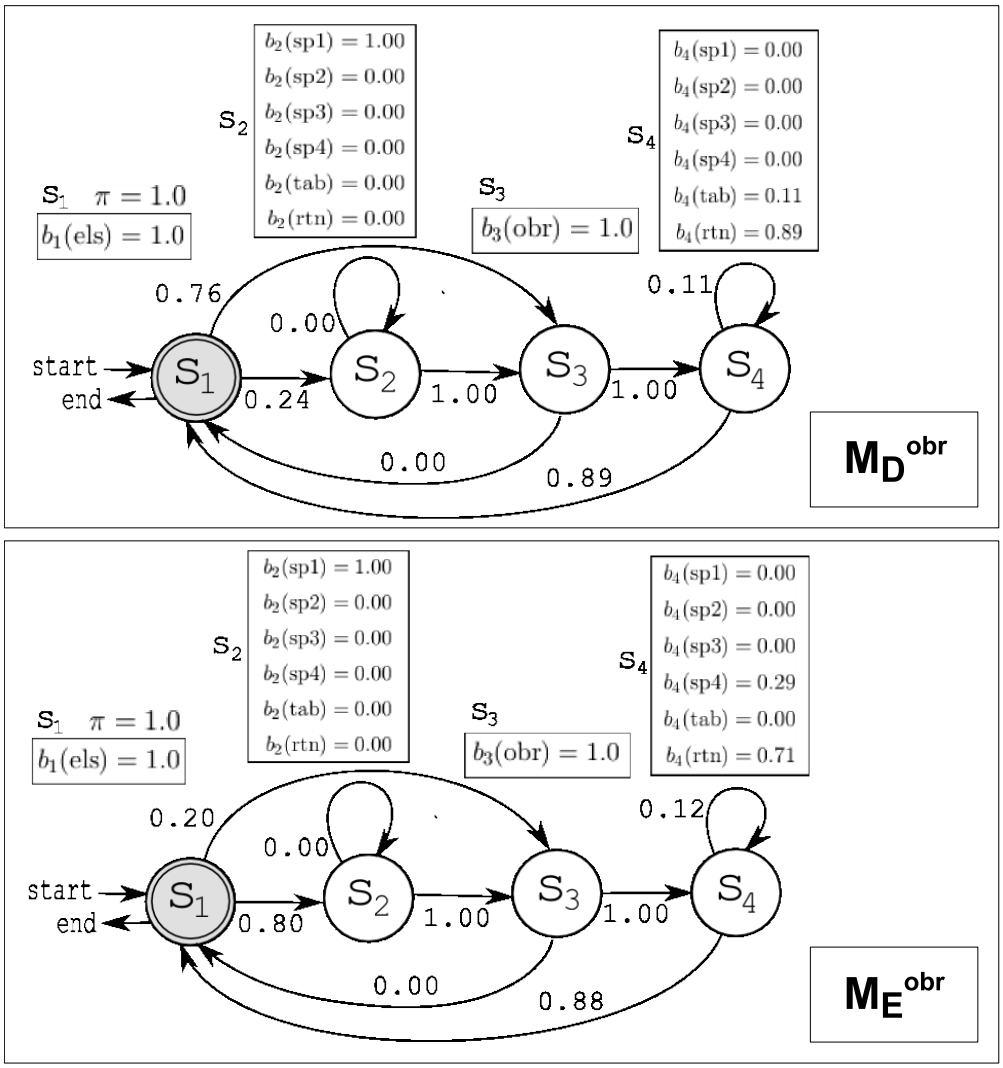


図 5.7　被験者 D，E の記述スタイルを表す記述スタイルモデルの一例

A5，B5，C5，D5，E5 が与えられたときのそれぞれのモデルの分散を見てみると，A5，B5，C5，D5 についてはその作成者である A，B，C，D の記述スタイルモデル M_A，M_B，M_C，M_D の分散が他の 4 つのモデルと比較して最も低い値であるのに対し，E5 についてはその作成者である E の記述スタイルモデル M_E の分散が最も高い値を示している．また，列ごとに分散を見てみると，M_A は例外となってしまうが，M_B，M_C，M_D については，5 つのソースコードが与えられたとき，自身が表現する記述スタイルを持つ作成者により作成されたソースコード，すなわち B5，C5，D5 について，他の 4 つのソースコードよりも低い分散を示している．これに対し，M_E は E5 よりも C5 に対し低い分散を示し，E5 に対しては 2 番目に低い分散を示している．

表 5.2　A5，B5，C5，D5，E5 が与えられたときの M_A，M_B，M_C，M_D，M_E の平均出力確率と分散

(%)

	M_A	M_B	M_C	M_D	M_E
A5	74.79(8.38)	71.10(11.99)	67.63(11.33)	71.31(10.21)	64.34(13.31)
B5	71.99(8.30)	73.04(7.94)	69.75(13.79)	70.63(9.22)	68.65(11.21)
C5	64.11(9.94)	69.58(10.89)	85.09(3.47)	47.52(15.34)	70.63(10.10)
D5	64.21(11.55)	69.55(14.08)	59.70(15.33)	83.13(5.23)	53.62(14.12)
E5	73.61(7.85)	71.87(9.64)	72.92(9.09)	67.20(9.16)	66.82(11.50)

(1)

(2)

一般に，モデルの平均出力確率の分散が高いということは，入力されたソースコード 5 から生成されたトークン部分列集合の中に，モデルが表す記述スタイルと異なるスタイルを持つ部分列が多く含まれているということ，つまりソースコード 5 はモデルの表す記述スタイルとは異なる特徴を持った作成者により作成された可能性が高いということであるが，もう一つの可能性として，モデルの学習に使用されたソースコード 1〜4 の間に，もしくはソースコード 1〜4 とソースコード 5 の間に，表記のゆらぎが多く含まれていることが考えられる．本手法では，少なくともモデルの学習に用いるソースコードは必ずその作成者によって作成されたことが保障されているという前提条件があるため，学習に用いたソースコードの間に存在する表記ゆらぎについても作成者の記述特徴として扱う必要がある．そのためには，現在採用している平均出力確率による判定を見直し，新しい指標を定義するべきであると考えられる．

また，ソースコード 1〜4 とソースコード 5 の間にたまたま表記ゆらぎが存在する場合，現行のアルゴリズムでは，盗用と区別することは難しい．対処法としては，今回の考察のように，分散から得られる情報を見て総合的に判断するということも考えられるが，まずはクロスバリデーションにより全てソースコードの組み合わせについてテストする必要があると考えられる．

5.1.4 実験 2

概要

実験 1 では，記述スタイルモデル群の平均出力確率を元に記述スタイルの判定を行った．これはモデルの学習に用いる 4 つのソースコードの記述スタイルが全て同じパターンをとるという前提をもとにしている．しかし同一の作成者が常に同じパターンで記述を行うとは限らない．そこで，複数のソースコード間にあらわれる表記のぶれもその作成者の記述特徴とみなすこととする．具体的には，14 個の記述スタイルモデルの出力確率とその分散を用いて記述スタイルの比較を行う（【E1】）．

また，モデルの性能を正確に判定するため，モデルの学習に使用する 4 つのソースコードの全ての組み合わせについてクロスバリデーションを行い，全ての結果を平均した総合的な指標による作成者認証および記述スタイル判別を行う（【E2】）．

実験内容

実験 1 と同じソースコードを用いて次の実験項目【E1】,【E2】について実験を行い，実験 1 の結果と比較する．

【E1】では，以下の手順により個々の記述スタイルモデルより得られた出力確率および分散を用いた作成者認識を行う．

1) 被験者 A の作成したソースコード A1，A2，A3，A4 を学習させた記述スタイルモデル群 M_A に，同じく A により作成された A1，A2，A3，A4 を入力する．このとき 14 個のモデルから出力された出力確率を教師データ，すなわち正解とする．個々のモデルの出力確率は，入力された複数のトークン部分列に対し出力された出力確率の平均であるが，この分散についても作成者認証に用いる．ここで得られた 14 個の出力確率と分散をそれぞれ “正解出力確率”，“正解分散” とする．
2) 次に，1) と同じモデル群 M_A にソースコード A5，B5，C5，D5，E5 を入力し，得られた 14 個の出力確率および分散と 1) で求めた正解出力確率および正解分散をそれぞれ比較する．このとき，A5 を入力したときの出力確率および分散が正解出力確率および正解分散に最も近ければ，M_A が記述スタイル A を判別可能であるとする．

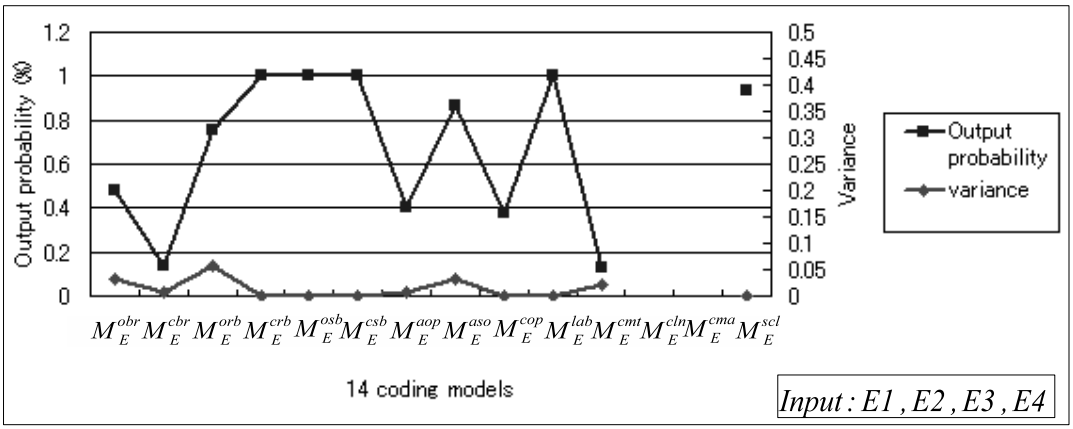


図 5.8　記述スタイルモデル群 M_E にモデル学習と同じソースコード E1, E2, E3, E4 を入力したときの 14 個の記述スタイルモデルの出力確率と分散

3) 1)〜2) を M_B，M_C，M_D，M_E についても同様に行う．

【E2】では，前途の【E1】の 2) で得られた 14 個の出力確率を 14 次元のベクトルとみなし，正解出力確率との距離を計算する．この距離の逆数を類似度とし，この値が高い程記述スタイル A に近い，すなわち，被験者 A によって作成された可能性が高いとみなす．

ここで，複数のソースコードから得られる平均的な特徴を抽出するため，全てのソースコードの組み合わせについてクロスバリデーションを行う．具体的には，以下を全ての被験者について行う．

1) ソースコードの総数を N 個とする．
2) n 回目に n 番目 $(1 \leq n \leq N)$ のソースコードを取り出す．
3) 残りの $N-1$ 個のソースコードを記述スタイルモデル群に学習させる．
4) 2) のソースコードを 3) のモデルに入力し，得られた出力確率に基づき類似度を算出する．
5) 1)〜4) を $n=1$ から $n=N$ まで N 回繰り返し，得られた N 個の類似度の平均値を最終的な類似度とする．

実験結果

前小節に示された実験項目【E1】,【E2】について実験を行った結果は以下の通りである．

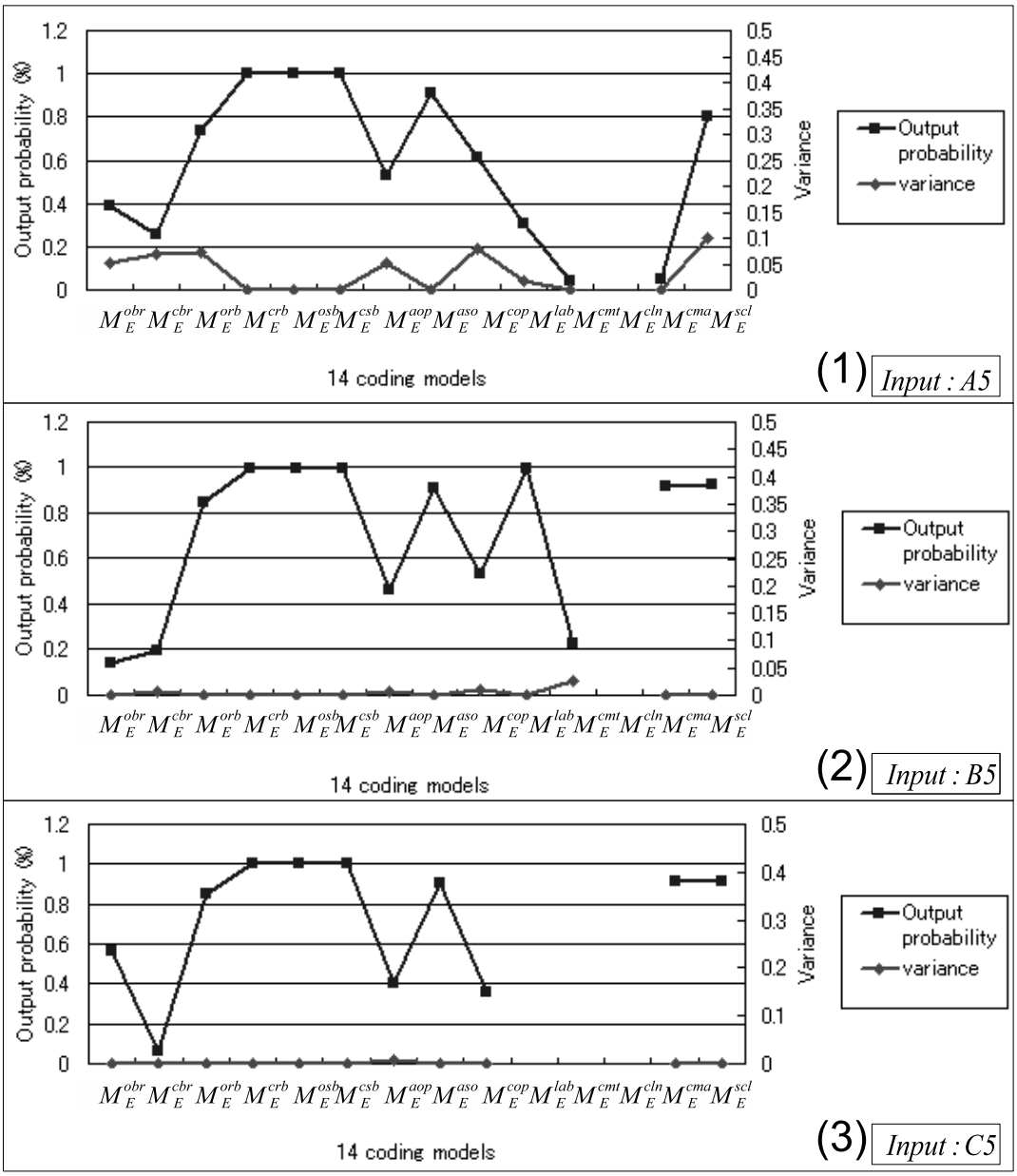


図 5.9　記述スタイルモデル群 M_E に (1) ソースコード A5, (2) ソースコード B5, (3) ソースコード C5 を入力したときの 14 個の記述スタイルモデルの出力確率と分散

【E1】実験 1 では，5.1.3 の表 5.2 に示されるように M_E が自らの記述スタイルを判別できなかったという結果が出ていた．しかし図 5.8 と図 5.9 の (1)〜(3)，図 5.10 の (1)〜(2) を比較すると，明らかに M_E により表される正解記述スタイルと E5 の記述スタイルが類似していることが確認できる．また，他の記述スタイルについても，平均出力確率としてではなくこのように個々の記述スタイルモデルごとの出力確率に注目すると，いくつ

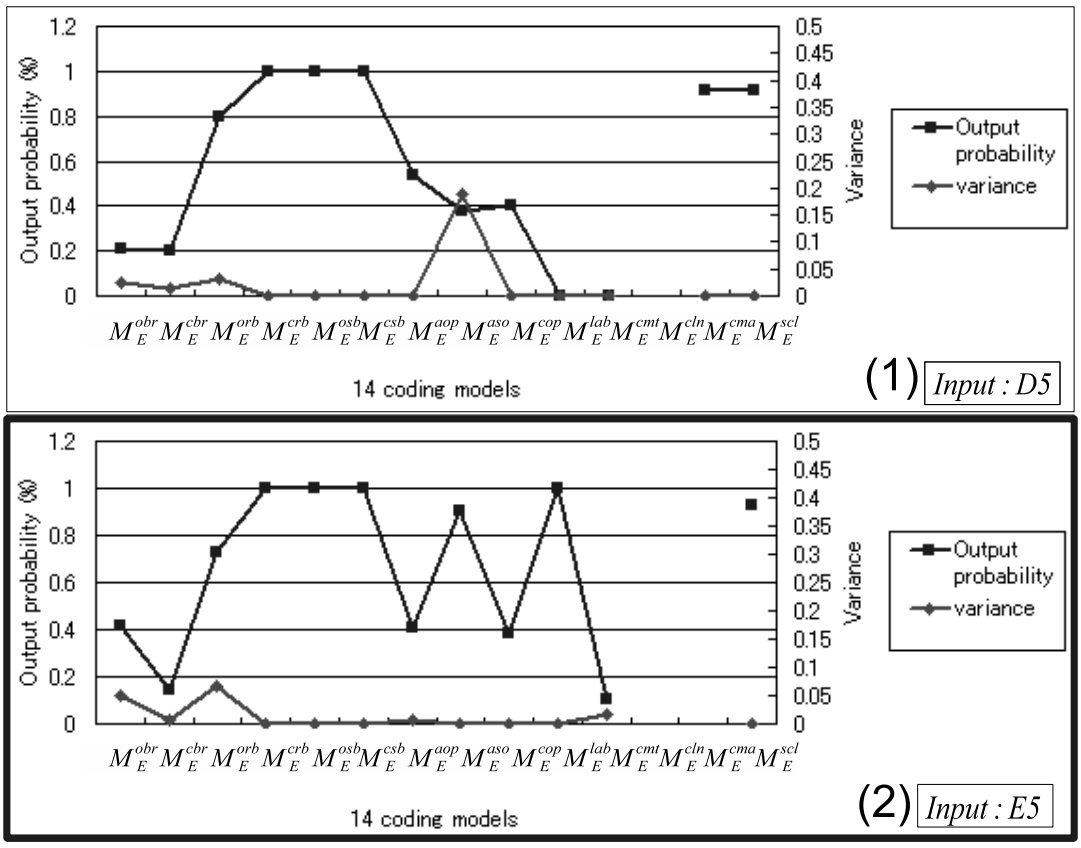
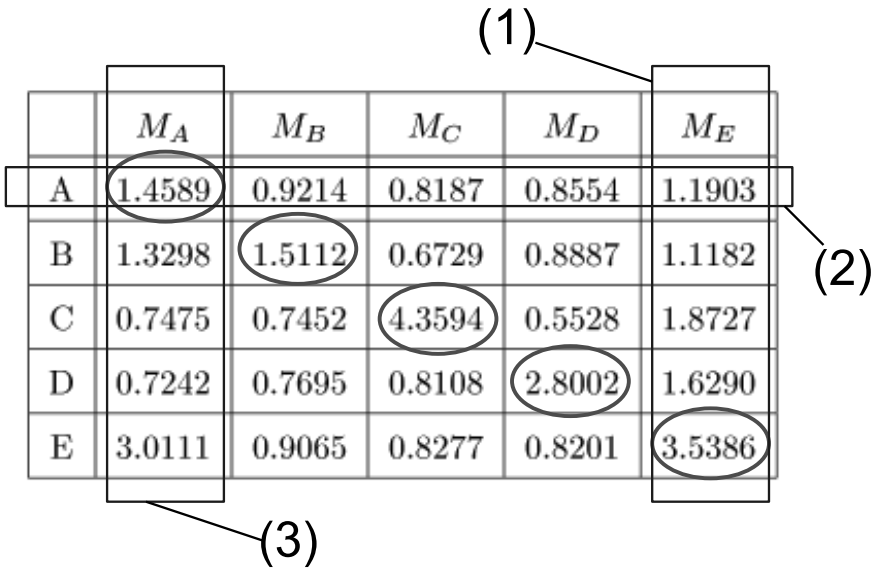


図 5.10　記述スタイルモデル群 M_E に (1) ソースコード D5，(2) ソースコード E5 を入力したときの 14 個の記述スタイルモデルの出力確率と分散

かのモデルの出力確率において類似した傾向が見られた．これらのことから，記述スタイルモデル群全体ではなく，個々の記述スタイルモデルの出力確率に注目することで，実験 1 では見落とされた記述スタイルの類似を検出可能であると考えられる．

【E2】表 5.3 は，5 人の被験者の記述スタイルを表す記述スタイルモデル群 M_A，M_B，M_C，M_D，M_E に，A，B，C，D，E によって記述されたソースコードを入力したときの類似度の平均を一覧表示したものである．縦方向の 5 つの数値は，例えば表中 (1) では E の記述スタイルを表す M_E に A，B，C，D，E によって作成されたソースコードが入力された場合の類似度であり，M_E が E の記述スタイルにより作成されたソースコード E に対し最も高い値を示したとき，M_E が自らの記述スタイルを判別したと評価することができる．これをグラフ表示したものが図 5.11 である．横方向の 5 つの数値は，例えば表中 (2) では E の記述スタイルにより作成されたソースコード E が入力されたとき，M_A，M_B，M_C，M_D，M_E からそれぞれ得られた類似度であり，M_E が他のモデルに比べ最も

表 5.3　記述スタイルモデル群 M_A，M_B，M_C，M_D，M_E より得られたソースコード A，B，C，D，E の類似度



	M_A	M_B	M_C	M_D	M_E
A	1.4589	0.9214	0.8187	0.8554	1.1903
B	1.3298	1.5112	0.6729	0.8887	1.1182
C	0.7475	0.7452	4.3594	0.5528	1.8727
D	0.7242	0.7695	0.8108	2.8002	1.6290
E	3.0111	0.9065	0.8277	0.8201	3.5386

高い値を示したとき，ソースコード E の記述スタイルを M_A，M_B，M_C，M_D，M_E の出力により判別できたと評価することができる．これをグラフ表示したものが図 5.12 である．表中 (1)，(2) に示されるように，実験 1 において誤判定の見られた M_E による記述スタイル E の判別およびソースコード E の作成者認識が正しく行われたことが確認された．しかし表中 (3) に示されるように，実験 1 では正確に行われた M_A による記述スタイル判別において，本来の記述スタイルである A でなく E に最も高い類似度を示すという誤判定が見られた．

考察

実験 2 では，実験 1 の結果を踏まえ，14 個の記述スタイルモデルの出力確率を活かす形で作成者認識を行うという新しい評価指標に基づきモデルの性能評価を行った．

【E1】では，14 個の出力確率とその分散を視覚化し，その形状を比較することで，全体で見ると見落としてしまうような局所的な類似点を発見可能なことが確認された．【E2】では，14 個の出力確率を 14 次元のベクトルとし，正解ベクトルとの作成者未知のソースコードを入力としたときに得られたベクトルとの距離を記述スタイルの類似度とし，これを用いて作成者認証と記述スタイルの判定を行った．【E1】，【E2】ともに，実験 1 におい

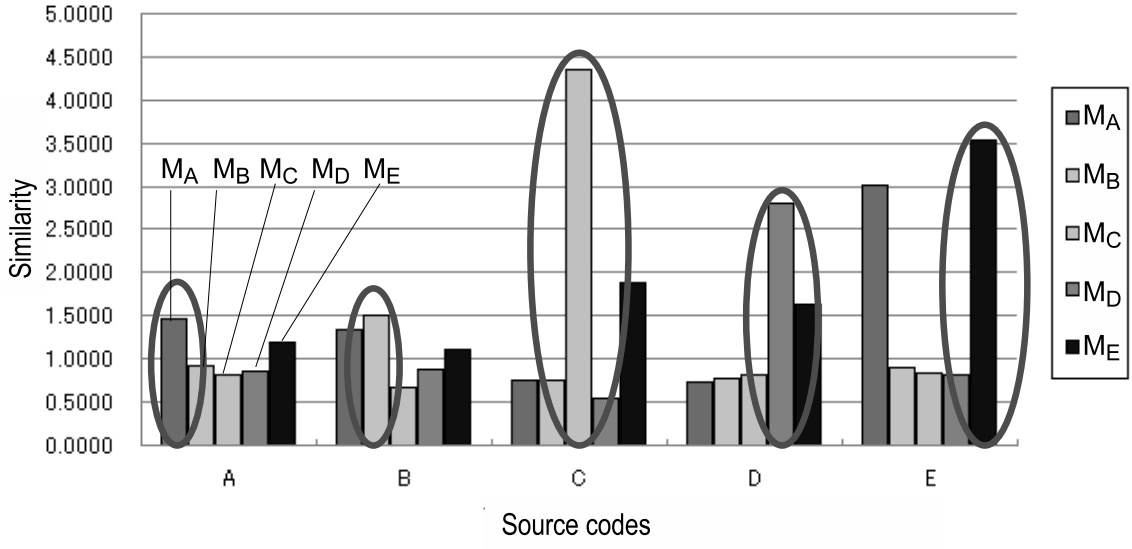


図 5.11　ソースコード A，B，C，D，E が入力されたとき記述スタイルモデル群 M_A，M_B，M_C，M_D，M_E より得られた類似度

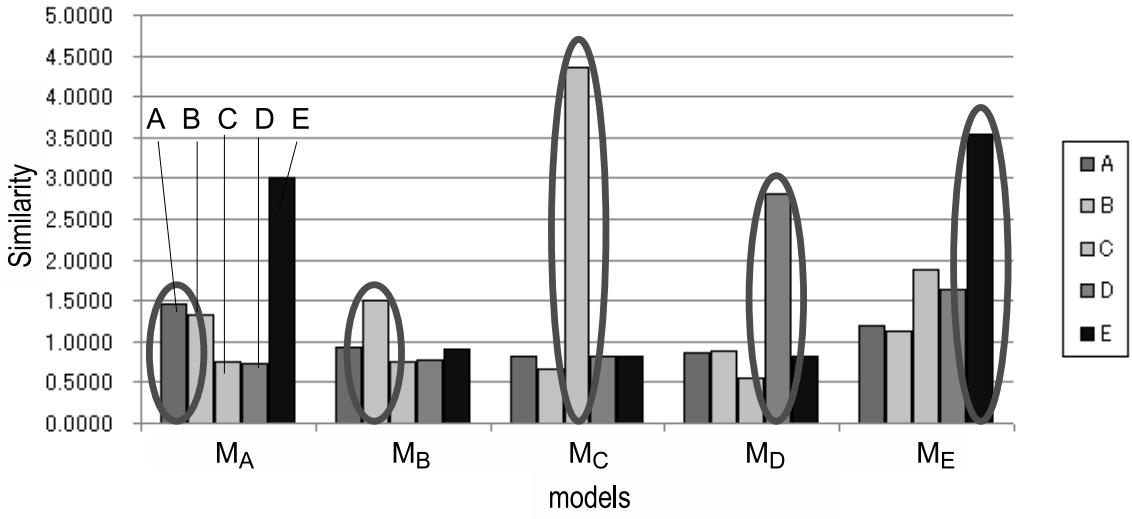


図 5.12　記述スタイルモデル群 M_A，M_B，M_C，M_D，M_E より得られたソースコード A，B，C，D，E の類似度

て誤判定の見られた M_E による記述スタイル E の判別およびソースコード E の作成者認識について良好な結果が得られたが，【E2】において，実験 1 では誤判定のなかった M_A による記述スタイル A の判別において誤判定が見られた．A が入力されたとき，表 5.3 の (2) をみると，5 つの記述スタイルモデル群中 M_A と M_E がひときわ高い値を示している．同じく，E が入力されたときも，M_A と M_E が飛び抜けて高い値を示しており，いずれの場合も，M_B，M_C，M_D はほぼ同程度の値を示している．また，B，C，D が入力されたとき，このような傾向は見られない．これらのことから，現行の記述スタイル抽出では本手法の目的である作成者認証については正確に行われているが，記述スタイルの判別については，M_A と M_E の記述スタイルの違いを正確に判別できていないことがわかった．

個々の記述スタイルモデルに注目してみると，例えば 5.1.3 の図 5.4，図 5.5 に示される A と E のトークン obr に関わる記述スタイルモデルのパラメータは大きく異なっている．しかし，本小節の図 5.9 の (1) および図 5.10 の (2) に示される，M_E^{obr} に A5 および E5 を入力としたときの出力確率はいずれも約 0.4 と近い値となっている．

実験 2 では，類似度という 14 個の記述スタイルモデルの出力確率を活かす新しい評価指標を定義し，モデルの評価を行った．全体としては，本手法で提案した記述スタイルに基づく作成者認証と確率モデルを用いた記述スタイルの判別という 2 つの機能が記述スタイルモデル群により実現可能であることが期待できるような良好な結果が得られたが，先に挙げた問題点については改善する必要がある．

5.2 考察

本章では，3.2 で提案した CM アルゴリズムをもとに開発した，ソースコード間類似性検出手法について述べた．本手法は主に，プログラミング授業課題ソースコードにおける盗用発見手法としての適用を想定している．本手法では，ソースコードに対し字句解析や構文解析を行うことにより得られる特徴量ではなく，人が自らの手でソースコードの比較を行う際の処理に倣った表面的な特徴抽出により得られた特徴量を用いて類似性検出を行う．これにより，人のそれと近い類似性尺度に基づく特徴抽出の実現を目指す．これは 1.1 にて挙げた，「[機能 B] 人の意図をコンピュータが適切に定量化する機能」にあたる．このようにして得られた特徴を作成者の記述スタイルとして確率モデルにより表現

し，ソースコードの作成者認証を行うというプロセスを通じて人の意図に基づく特徴を定量化し，類似性検出に用いる．

実験により，本手法で提案するモデルが複数の人物の記述スタイル特徴を内部パラメータにより表現し，これをもとに異なる記述スタイルを判別可能であることが確認された．これにより，授業課題ソースコードのような行数が短くアルゴリズム的に類似したソースコードから作成者の特徴を抽出するという，盗用発見を行いたい人の意図に基づく特徴抽出を行い，的確に定量化できたことが確認された．さらに，14 種類の記述スタイルモデルの出力と分散をグラフ表示したものを用いて，記述スタイルを識別可能であることが確認された．しかし，現在用いている評価指標では抽出された記述スタイルの違いを十分に活かすことができず，結果としてわずかではあるが最終的な判定結果において誤判定を出してしまうケースもあることが確認された．この原因の一つとして，個々の記述スタイルモデルの出力確率が全ての入力データに対する出力確率の平均であることが推測される．これについては，個々のモデルごとの出力の算出方法を再検討する必要があると考えられる．例えば，モデルごとに得られた全ての出力確率を 1 つの離散時間信号とみなし，離散フーリエ変換により周波数成分に変換した値を個々のモデルの出力として用いることなどが考えられる．

また，本手法では，14 種類の記述スタイルモデルを用いて 1 つの記述スタイルを表現することにより，1 つのソースコードから重複して多数の学習データを得られるよう工夫されている．しかし，授業課題ソースコードにおける盗用発見手法としての適用を考えると，モデルの学習に用いることのできるソースコードの数は 10〜15 個程度であることが予想される．1 つのソースコードから得られる学習データの数は，14 種類のトークンの出現頻度に依存している．例えば，本章の実験で用いたソースコード E5 の行数は空行も含め 50 行であり，得られた学習データの総数は 113 個であった．1 つのソースコードからより詳細な記述スタイルを抽出するためには，例えば識別子名の命名規則など，これまでと異なる観点から新たな記述規則を定義し，記述スタイルとして抽出する必要があると考えられる．

第 6 章

結論

現代社会において，我々はコンピュータを様々な用途に活用している．しかし，常にその意図する通りの処理を実現できているとは言いがたい．これは，あらゆる処理において人がその思考や操作をコンピュータに合わせる必要があり，それが人の意図の実現を制限しているためであると考えられる．本研究では，情報検索を「人とコンピュータのコミュニケーション・インタフェース」ととらえ，人とコンピュータの相互作用により生みだされる新たな知の発見により，人の意図を更新するような情報検索の形を，人とコンピュータによる「協調型情報検索」と定義した．本論文では，これを実現するために必要な 2 つの要素のうちの 1 つである「人の意図に基づく特徴抽出手法」について，次の 2 つの機能を有するものであるとし，そのような機能をもつ特徴抽出手法の開発を行ってきた．

［機能 A］**人の意図を的確にコンピュータに伝えることを支援する機能：**人は類似性検出を行うとき，対象や意図に応じて柔軟にその類似性尺度を変更している．これをコンピュータ上でシミュレートするには，高い専門知識を要求せずにユーザ自らの手で特徴抽出の類似性尺度を変更できるような機能が必要である．

［機能 B］**人の意図をコンピュータが適切に定量化する機能：**人の意図する類似性を検出するためには，その意図に基づく特徴を抽出し適切に定量化する機能が必要である．

本論文では第 3 章において，「参照を用いた特徴抽出手法（FRef アルゴリズム）」および「記述スタイルモデルに基づく特徴抽出手法（CM アルゴリズム）」という 2 つの特徴抽出手法を提案した．

FRef アルゴリズムではデータの多面的な特徴を参照ベクトルという低次元の特徴ベクトルで表現する．FRef アルゴリズムを実装した類似性検出手法は既存の手法とは異なり，類似性検出対象となるデジタルデータ同士を直接比較するのではなく，予め選出しておいた参照データとの間でそれぞれ算出した参照ベクトルを相対的類似性尺度として用いる手法であり，参照ベクトル間のユークリッド距離をデータ間の類似度とする．FRef アルゴリズムでは，対象データに対し詳細な解析に基づく特徴抽出を行う必要がなく，単に参照データとの相対的な関係をもとに特徴表現を行うため，一次元データとみなせるどのようなデータにも容易に実装可能である．また，ユーザ側で用いる参照データを変更することにより，特徴抽出における類似性尺度を容易に変更することが可能であるため，対象となるデータについての専門知識を要せずに，異なる特徴ベクトル空間上の写像を得ることができる．これは先に挙げた［機能 A］を実現する手法であるといえる．

第 4 章では，FRef アルゴリズムの適用例について述べた．まず，4.1 では，FRef アルゴリズムを基にしたソースコード間類似性検出手法を提案し，これを実装したシステムを用いて実験を行った結果，高速で精度の高い検索が行えることを確認した．一般的な類似ソースコード検索のような用途においては，類似性尺度を自由に変更する必要性は低いかもしれない．参照ソースコードの数が 3 つ以上であれば，どのソースコードを用いても検索結果の誤差は ±1.5% 以内に収まることが実験により確認されている．このことから，毎回同じ意図に基づき検索が行われるような場合には，単に参照ソースコードを母集団から 3 つランダムに選出して検索に用いれば良いと考えられる．

4.2 では，FRef アルゴリズムを基にした画像間類似性検出手法の提案を行った．人が自らの手で画像間の類似性検出を行う際，画像の持つ高次元の特徴を瞬時にその検索意図に応じ構築された低次元特徴ベクトル空間へと写像する．この処理を複数の参照画像を用いた間接的な特徴表現によりシミュレートすることにより，人の感覚に近い類似性検出を目指した．人の意図する類似性を検出するために，人の行う類似性検出処理と同等の結果を別の処理により得ようとするという点では，本手法は［機能 B］を実現する手法でもあるといえる．実験により，参照画像をランダムで選出し検索を行った結果は，人が予め行った分類結果と高い確率で一致することが確認された．また，今回の実験では，参照画像のどのような特徴が検索結果にどのように影響を及ぼすかについて解明するには至らなかったが，少なくとも，恣意的に参照画像を選出した場合，その検索結果は大きく異なることが確認された．

4.1，4.2 の結果を通して，FRef アルゴリズムを基にした類似性検出手法が実際に社会で解決の求められているソースコード間類似性検出や画像間類似性検出に対し有効であることが確認された．今後も参照データが検索結果に及ぼす影響について更に検討を行い，FRef アルゴリズムの特性を活かした新しい類似性検出手法として改良を続けていくことが目指される．

本論文で提案するもう一つの特徴抽出手法が CM アルゴリズムである．CM アルゴリズムは，対象となるデータの性質に特化した特徴抽出ではなく，人の行う特徴抽出の手続きそのものに倣った手法である．第 5 章では，CM アルゴリズムの適用例として，一次元データから特定のパターンを抽出し，確率モデルにより数理的に表現することを試みた．具体的には，複数のソースコードから作成者の記述特徴を抽出し，記述スタイルモデルによって定量化する手法を提案した．これは，一般的なソースコード間類似性検出手法ではノイズとして除去されてきたソースコードの表面的な特徴に着目し，作成者の記述スタイルを確率モデルにより表現するという手法である．

本手法の用途としては，例えば大規模ソースコード開発現場における記述スタイルの統一や授業における可読性の高い記述スタイルの習得の支援，著作権保護など多岐に渡るが，特に近年世界中で注目を集め，多くの研究成果が報告されている授業課題ソースコード盗用問題の解決に高い効果が期待できる．授業課題ソースコード盗用問題とは，授業課題として提出された，ソースコード長が短く内容が互いに類似したソースコード間において行われる盗用を発見するという問題であり，対象となるソースコードの性質上，従来の類似性検出手法では解決が難しい．本手法は人が手作業で授業課題ソースコード間の盗用を発見しようとするときに行う特徴抽出処理に倣っていることから，［機能 B］を実現する手法であるといえる．実験により，本手法のモデルが異なる記述スタイルを識別可能であることが確認された．しかしモデルのパラメータとして表現された特徴の一部が後処理において失われているため，最終的な出力において誤判定が出てしまうという問題が残されている．情報の損失の少ない評価指標の算出方法について検討が必要である．

本論文では，FRef アルゴリズムおよび CM アルゴリズムという 2 つの特徴抽出手法を提案し，それぞれ，現実社会において解決が望まれている問題に対し適用を行った．前者は純粋にソースコード間類似性検出手法に，ユーザ側で類似性尺度を変更可能という特性を付加したものであり，4.2 にて行った画像データに対する適用からもわかるように，データに非依存な，汎用性の高い特徴抽出手法としての性質も併せ持つ．後者はソース

コードそのものではなく，ソースコードを記述する際にみられる作成者の癖を定量化することにより，これまでの手法では区別が困難であった「類似」と「盗用」を見分けるという，人の行う高次の処理をコンピュータ上でシミュレートすることを目指すものである．このように，本研究で提案した 2 つの手法はそれぞれ，少なくとも現実社会において実際に解決が望まれている 2 つのソースコードに関する問題を解決する手段としてその有効性が期待されるものである．

本論文で提案した 2 つの特徴抽出手法はそれぞれ独立した手法として開発したものであるが，例えば，大学などの教育機関におけるプログラミング授業の採点・盗用発見支援システムとしての実装という形で，組み合わせて用いることも考えられる．また，これまでに提案した手法のアルゴリズムの細部について各章で挙げた課題について更なる検討を行うと共に，新たな特徴抽出手法の開発や，これを実装した実用的なシステムの開発，そして実際にシステムを使用し，その効果を検証することなどを今後の課題として挙げておきたい．

謝辞

本研究の全般に渡り懇切丁寧なご指導・ご助言を頂き，ご高配を賜りました，神戸大学大学院国際文化学研究科 村尾 元 准教授に深く感謝致します．先生のご指導なくしては，私は本研究をこのような形でまとめることはおろか，スタートラインからいくばくも離れることができないままであったと思います．

常に愛情と忍耐をもってご指導頂き，ご高配を賜りました，神戸大学大学院国際文化学研究科 鏑木 誠 教授，大月 一弘 教授，森下 淳也 教授，康 敏 准教授，清光 英成 准教授，神戸大学国際コミュニケーションセンター 柏木 治美 准教授に深く感謝致します．研究室の垣根を越えた強い結束と温かで自由な雰囲気の中で熱心なご指導を頂き，楽しく充実した 5 年間を送ることが出来ました．私にとって大変恵まれた環境であったと思います．

定例ミーティングへの参加をご快諾頂き，本研究の遂行にあたり貴重なご助言を頂きました神戸大学大学院工学研究科 玉置 久 教授，太田 能 准教授，稲元 勉 博士，松本 卓也 博士，杉川 智 氏，Augusto Foronda 氏，大原 誠 氏に感謝致します．理系院生のいろはをご教示頂き，自分の目指すべき姿を示して頂いた諸氏に多くを学ばせて頂きました．

本研究の実験にあたりご助力を頂きました，神戸大学大学院工学研究科 鎌田 十三郎 助教に深く感謝致します．

大学院入学当初より様々なご助言を頂き，ご高配を賜りました，神戸大学大学院経営学研究科 小川 進 教授に深く感謝致します．

本研究に関しご協力を頂きました加治屋 紘子 氏，王 暁磊 氏，孫 一 氏，朱 孫賢 氏，孫 荻 氏，他，神戸大学大学院国際文化学研究科の大学院生並びに研究生の皆様，神戸大学大学院国際文化学部の学部生の皆様，そして苦楽を共にし，現在もお世話になっております OB・OG の皆様に感謝致します．

また，約 4 年半の間共に心身の鍛錬に励み，沢山の楽しい思い出を頂いた神戸大学極真空手部 OB・OG 並びに在籍部員の皆様に感謝致します．

そして最後に，入籍とほぼ同時にスタートした長い学生生活を支え続けてくれた最愛の夫 尚之と，私の研究活動を応援してくれた家族にも深謝の意を表したいと思います.

参考文献

[1] Eugene Garfield. A Tribute to Calvin N. Mooers, A Pioneer of Information Retrieval. *The Scientist*, Vol. 40, No. 1, pp. 9–11, 1997.

[2] 笹森勝之助.「情報検索の現状と動向」.『情報処理』, Vol. 11, No. 12, pp. 721–730, 1970.

[3] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, Vol. 41, pp. 391–407, 1990.

[4] 竹内広宜, 小林メイ, 青野雅樹, 寒川光.「多変量解析に基づいた情報検索手法の比較検討」.『情報処理学会研究報告：情報学基礎研究会報告』, Vol. 2002, No. 28, pp. 87–93, 2002.

[5] 川前徳章, 青木輝勝, 安田浩.「情報理論的モデルを用いた情報検索」.『電子情報通信学会技術研究報告：DE, データ工学』, Vol. 101, No. 192, pp. 159–166, 2001.

[6] 六車正道.「特許情報検索の課題と概念検索システムの役割」.『知財管理』, Vol. 51, No. 12, pp. 1891–1990, 2001.

[7] 大橋剛介, 久森隆史, 望月圭太.「適合性フィードバックを用いたスケッチ画像検索システム」.『知能と情報』, Vol. 19, No. 5, pp. 537–545, 2007.

[8] 中島伸介, 田中克己.「相対的マッピング処理に基づく相対的情報検索手法」.『情報処理学会論文誌：データベース』, Vol. 11, No. 6, pp. 63–75, 1997.

[9] Charlie Daly and Jane Horgan. Patterns of Plagiarism. *SIGCSE Bull.*, Vol. 37, No. 1, pp. 383–387, 2005.

[10] Asako Ohno and Hajime Murao. A Similarity Measuring Method for Images Based on the Feature Extraction Algorithm Using Reference Vectors. *Proceedings of the 2nd International Conference on Innovative Computing, Information and*

Control, pp. 454–457, 2007.

[11] Robert M. Haralick, K. Shanmugam, and Its'hak Dinstein. Textural Features for Image Classfication. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, No. 6, pp. 610–621, 1973.

[12] Rakesh Dugad and U. B. Desai. A Tutorial on Hidden Markov Models. `http://vision.ai.uiuc.edu/dugad/`, pp. 1–16, 1996.

[13] Anders Krogh. An Introduction to Hidden Markov Models for Biological Sequences. *In S. L. Salzberg, et al., eds., Computational Methods in Molecular Biology*, Chapter 4, pp. 45–63, 1998.

[14] 小高知宏, 村田哲也, 高建斌, 諏訪いずみ, 白井治彦, 高橋勇, 黒岩丈介, 小倉久和. 「n-gram を用いた学生レポート評価手法の提案」. 『電子情報通信学会論文誌 D-I』, Vol. J86-D-I, No. 9, pp. 702–705, 2003.

[15] 井上克郎, 神谷年洋, 楠本真二. 「チュートリアル　コードクローン検出法」. 『コンピュータソフトウェア』, Vol. 18, No. 5, pp. 47–54, 2001.

[16] Brenda S. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. *IEEE Second Working Conference on Reverse Engineering*, pp. 86–95, 1995.

[17] Toshihiro Kamiya, Shinji Kusumoto, and Katsuhiro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for language scale source code. *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670, 2002.

[18] 山本哲男, 松下誠, 神谷年洋, 井上克郎. 「ソフトウェアシステムの類似度とその計測ツール SMMT」. 『電子情報通信学会論文誌 D-I』, Vol. J85-D-I, No. 6, pp. 503–511, 2002.

[19] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: Local Algorithms for Document Fingerprinting. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 76–85, 2003.

[20] Marius Marin, Arie van Deursen, and Leon Moonen. Identifying Aspects Using Fan-In Analysis. *Proceedings of the 11th Working Conference on Reverse Engineering*, pp. 132–141, 2004.

[21] 鷹岡良治, 脇田健. 「静的解析のプログラムの盗用発見への応用」. 『日本ソフトウェア科学会第 21 回大会論文集』, pp. 406–410, 2004.

[22] 川口真司. 「潜在的意味解析法 LSA に基づくソフトウェアシステム分類法の提案」. 大阪大学大学院基礎工学研究科修士学位論文, 2003.

[23] 川口真司, パンカジガーグ, 松下誠, 井上克郎. 「MUDABlue: ソフトウェアリポジトリ自動分類システム」. 『電子情報通信学会論文誌 D-I』, Vol. J88-D-I, No. 8, pp. 1217–1225, 2005.

[24] 北川俊広, 杉山安洋. 「自己組織化マップを用いた Java ソースコード間類似度測定ライブラリの試作」. 『電子情報通信学会技術研究報告』, Vol. 104, No. 723, pp. 19–24, 2005.

[25] 小堀一雄. 「類似度メトリクスを用いた Java ソースコード間類似度測定ツールの試作」. 『電子情報通信学会技術研究報告』, Vol. 103, No. 102, pp. 7–12, 2003.

[26] Andrian Marcus and Jonathan I. Maletic. Identification of High-Level Concept Clones in Source Code. *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pp. 107–115, 2001.

[27] 仁井谷竜介, 松下誠, 井上克郎. 「ソースコードの特徴語を用いた Java ソフトウェア部品の自動分類手法の提案」. 『ソフトウェア工学』, Vol. 2005, No. 75, pp. 49–56, 2005.

[28] Elizabeth Burd and John Bailey. Evaluating Clone Detection Tools for Use During Preventative Maintenance. *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation*, pp. 36–43, 2002.

[29] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. JPlag: Finding Plagiarisms Among a Set of Programs. *Technical Report, Universitaet Karlsruhe*, No. 2000-1, 2000.

[30] Alex Aiken. Moss: A Measure of Software Similarity. `http://www.cs.berkeley.edu/~aiken/moss.html`.

[31] 肥後芳樹, 楠本真二, 井上克郎. 「コードクローン検出とその関連技術」. 『電子情報通信学会論文誌 A』, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[32] 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.

[33] Toshiyuki Amagasa, Minoru Nakai, Kenji Hatano, Masatoshi Yoshikawa, and

Shunsuke Uemura. Keyword Assignment to Images Using Sliding Windows. *Proceedings of 2000 ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, pp. 1–10, 2000.

[34] 横山貴紀, 菅原研, 渡辺俊典. 「フラクタル符号に基づく圧縮領域における類似画像検索手法」. 『情報処理学会論文誌：データベース』, Vol. 45, No. SIG4(TOD 21), 2004.

[35] 呉君錫, 金子邦彦, 牧之内顕文, Sang-Hyun Bae. 「Wavelet-SOM に基づいた類似画像検索システムの設計・実装と性能評価」. 『情報処理学会論文誌：データベース』, Vol. 41, No. SIG1(TOD 8), pp. 1–11, 2001.

[36] 小原拓文, 金川明弘. 「SOM による色・形状・テクスチャ特徴量を用いた道路標識の分類」. 『知能と情報』, Vol. 19, No. 5, pp. 466–475, 2007.

[37] C.E. Jacobs, A. Finkelstein, and D.H Salesin. Fast Multiresolution Image Querying. *Proceedings of SIGGRAPH95*, pp. 6–11, 1995.

[38] Asako Ohno and Hajime Murao. A Quantification of Students' Coding Style Utilizing HMM-Based Coding Models for In-Class Source Code Plagiarism Detection. *Proceedings of the 3rd International Conference on Innovative Computing, Information and Control*, pp. 553–556, 2008.

[39] Steve Engels, Vivek Lakshmanan, and Michelle Craig. Plagiarism Detection Using Feature-Based Neural Networks. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pp. 34–38, 2007.

[40] Georgina Cosma and Mile Joy. Source-Code Plagiarism: a UK Academic Perspective. *In Proceedings of the 7th Annual Conference of the HEA Network for Information and Computer Sciences*, pp. 116–120, 2006.

[41] Neal R. Wagner. Plagiarism by Student Programmers. `http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html`, 2000.

[42] Joanna Bull, Carol Collins, Elisabeth Coughlin, and Dale Sharp. Technical Report of Plagiarism Detection Software. *University of Luton and JISC Plagiarism Advisory Service*, 2001.

[43] Christian Arwin and S. M. M. Tahaghoghi. Plagiarism Detection Across Programming Languages. *Proceedings of the 29th Australasian Computer Science*

Conference, pp. 277–286, 2006.

[44] Peter Vamplew and Julian Dermoudy. An Anti-Plagiarism Editor for Software Development Courses. *Proceedings of the 7th Australasian Conference on Computing Education*, pp. 83–90, 2005.

[45] G. Whale. Identification of Program Similarity in Large Populations. *The Computer Journal*, Vol. 33, No. 2, pp. 140–146, 1990.

[46] Shauna D. Stephens. Using Metrics to Detect Plagiarism. *Proceedings of the 7th Annual Consortium for Computing in Small Colleges Central Plains Conference on The Journal of Computing in Small Colleges*, pp. 191–196, 2001.

[47] Aleksi Ahtiainen, Sami Surakka, and Mikko Rahikainen. Plaggie: GNU-Licensed Source Code Plagiarism Detection Engine for Java Exercises. *Proceedings of the 6th Baltic Sea Conference on Computing Education Research, Koli Calling*, pp. 141–142, 2006.

[48] David Gitchell and Nicholas Tran. Sim: An Utility for Detecting Similarity in Computer Programs. *SIGCSE Bull.*, Vol. 31, No. 1, pp. 266–270, 1999.

[49] B. Belkhouche, Anastasia Nix, and Johnette Hassell. Plagiarism Detection in Software Designs. *Proceedings of the 42nd Annual Southeast Regional Conference*, pp. 207–211, 2004.

[50] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. GPlag: Detection of Software Plagiarism by Program Dependence Graph Analysis. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 872–881, 2006.

[51] B. Mckinnon A. Seker X. Chen, M. Li. A Theory of Uncheatable Program Plagiarism Detection and Its Practical Implementation. *IEEE Trans. Information Theory*, 2002.

本論文に関連する研究発表

■学術雑誌 論文発表（査読あり）

1. Asako Ohno and Hajime Murao, Measuring Source Code Similarity Using Reference Vectors, *International Journal of Innovative Computing, Information and Control*, Vol.3, No.3, pp.525–538, 2007.
2. 大野麻子, 村尾元, 「参照ベクトルを用いたソースコード間の類似性検出」, 『電気学会論文誌 C』, Vol.128, No.1, pp.133–142, 2008.
3. Asako Ohno and Hajime Murao, A Similarity Measuring Method for Images Based on the Feature Extraction Algorithm using Reference Vectors, *International Journal of Innovative Computing, Information and Control*, Vol.5, No.3, pp.763–771, 2009.

■国際会議 口頭発表（査読あり）

1. Asako Ohno and Hajime Murao, Measuring Source Code Similarity Using Reference Vectors, *Proceedings of the 1st International Conference on Innovative Computing, Information and Control*, Vol.2, pp.19–24, 2006.
2. Asako Ohno and Hajime Murao, A Multi-Reference Pronunciation Scoring For A Computer-Assisted Pronunciation Learning Tool, *Proceedings of the 8th International Conference on Information Technology Based Higher Education and Training*（CD-ROM）, pp.686-689, 2007.
3. Asako Ohno and Hajime Murao, A Similarity Measuring Method for Images Based on the Feature Extraction Algorithm using Reference Vectors, *Proceedings of the 2nd International Conference on Innovative Computing, Information*

and Control（CD-ROM）, pp.454-457, 2007.

4. Asako Ohno and Hajime Murao, A Quantification of Students' Coding Style Utilizing HMM-Based Coding Models for In-Class Source Code Plagiarism Detection, *Proceedings of the 3rd International Conference on Innovative Computing, Information and Control*（CD-ROM）, pp.553-556, 2008.

■国内会議 口頭発表（査読なし）

1. 大野 麻子, 村尾 元, 「隠れマルコフモデルを用いたソースコード作成者の記述スタイルモデルの獲得」, 『平成 20 年電気学会 電子・情報・システム部門大会』（CD-ROM）, pp.1043-1048, 2008.
2. 大野 麻子, 村尾 元, 「記述特徴に着目した授業課題ソースコードにおける盗用の発見」, 『平成 20 年度情報処理学会関西支部大会講演論文集』, pp.303-306, 2008.

KOBE UNIVERSITY

A Study on Development of Feature Extraction Algorithms for a New Information Retrieval Method Achieved through Interaction between Human and Computer

DISSERTATION

Submitted in partial satisfaction of the degree of

DOCTOR OF PHILOSOPHY

by

Asako Ohno

Dissertation Supervisors:

Associate Professor Hajime Murao

Professor Makoto Kaburagi

Professor Jun-ya Morishita

2009

Graduate School of Cultural Studies and Human Science, Kobe University