OJS内のデータを利用したユーザへの効率的なアルゴリズム要否の提示

柴田 敦也[1,a)]　須藤 克仁[1]　畑 秀明[1]　中村 哲[1]

概要：ソフトウェア開発において効率的に動作するアルゴリズムの選択が重要となる場面が存在するが，全てのソフトウェアに効率的なアルゴリズムが採用されているとは限らない．そこで本稿では，ユーザのプログラム記述に対して，より効率的なアルゴリズムが存在する場合に自動的に実装の推薦を行うために必要な技術/手法について検討する．本研究では効率的なアルゴリズム実装の推薦を，1. ユーザのソースコード中の非効率的なアルゴリズムの有無の検出，2. 検出した非効率的なアルゴリズムに対応する，効率的なアルゴリズムの選択，3. 効率的なアルゴリズムをユーザの元のソースコードに近い形に整形して提示の 3 つの課題に大別した．本稿では 1 の課題をプログラムの非効率的/効率的分類問題として定式化し，オンラインジャッジシステム（OJS）上に提出されたソースコードに基づく機械学習手法について検討した．

キーワード：オンラインジャッジシステム, ソースコード推薦, プログラミング言語処理, 機械学習

Suggesting the Necessity of an Effective Algorithm using Data on Online Judge System

Abstract: The use of efficient algorithms is important in software development. However, it is not necessarily true in practice. In this paper, we pursue a method that automatically suggests an efficient algorithm for user's source code when such an algorithm is available. In this work, we divide the problem into three sub-problems: 1. Detecting inefficient algorithms in user's source code; 2. Choosing an appropriate algorithm for detected inefficient algorithm; 3. Suggesting an efficient source code that can be adapted to user's source code easily. In this paper, we formulate this problem as a binary classification problem of the efficiency and tackle it with a machine learning method based on submitted source code on an online judge system.

Keywords: Online Judge System, Code recommendation, Programming language processing, Machine learning

1. はじめに

1.1 背景

仮想化技術実現のためのサーバ集約によるシステムの大規模化や，ハードウェアの複雑化が進んでおり，高難易度化するソフトウェア開発を支援するシステムの需要が高まっている．また，人工知能・IoT 技術への期待の高まりによって，ソフトウェア開発に複雑なアルゴリズムを要求される場面が増加しており，アルゴリズムに関する知識の重要性が増している．例えば，ネットワークシステムにおけるルーティング処理はアルゴリズムの知識が必要となる身近な例である．しかし，全てのプログラマが多種多様なアルゴリズムに精通し，それぞれの問題に対して適切なアルゴリズムを適用・実装することは決して容易ではない．ツールによりプログラミングを支援する既存技術の一つとしてコード補完があるが，これはすでにユーザに入力された内容の文脈に沿ったコードを補完することで効率化を図るものであり，ユーザの知識以上に高度なプログラムを作成することはできない．そこで本研究では，処理速度が高速である，あるいはメモリをはじめとする計算資源の使用量が少ない処理を効率的な処理と定義する．そして，ユーザの作成したプログラムに対して，より効率的なアルゴリ

[1] 奈良先端科学技術大学院大学 情報科学研究科
Nara Institute of Science and Technology, Information science , Ikoma , Nara, 630-0192, Japan
[a)] shibata.atsuya.ry3@is.naist.jp

© 2018 Information Processing Society of Japan

ズムが存在するかを判定し，存在するならばより効率の良い実装例の推薦を行うことを最終的な目的とし，そのための課題解決を目指す．

ここで，非効率的なアルゴリズムがソフトウェアに採用される原因について考える．プログラマが解決すべき問題に直面した場合，プログラマは書籍や Web 上に存在するリファレンスを参照して最も効率的なアルゴリズムの実装を行えばよい．しかし，実装者の技量の不足や，効率性という観点の欠如，開発期間に制限があり実装するソースコード量の増加を避けたいといった場合等に，非効率的なアルゴリズムが採用されうる．

次に，同じ問題を解決するが処理時間などの観点で効率の異なるアルゴリズムの具体例を挙げる．例として，文字列検索などに応用される Suffix Array（接尾辞配列）を作成するためのアルゴリズムについて考える．この問題を解く最も効率的な方法として，SA-IS と呼ばれる時間計算量が $O(|S|)$ となるアルゴリズム（$|S|$ は対象文字列の文字列長）[1] が知られている．しかし実装が非常に複雑であることを理由に，計算量の観点で非効率的なアルゴリズム（ナイーブな $O(|S|^2 \log |S|)$ アルゴリズムや $O(|S| \log^2 |S|)$ アルゴリズム [2]）が代替として用いられているケースが存在する．このように，ソフトウェアに非効率的なアルゴリズムが含まれる場合，ソフトウェア開発を行うユーザに対して，より効率的なアルゴリズムとその実装を推薦できれば，効率の良いソフトウェアを容易に作成することができる．

本研究では上記のような効率的なアルゴリズムの推薦を，1. ユーザのソースコード中の非効率的なアルゴリズムの有無の検出，2. 検出した非効率的なアルゴリズムに対応する，効率的なアルゴリズムの選択，3. 効率的なアルゴリズムをユーザの元のソースコードに近い形に整形して提示の 3 つの課題に大別した．本稿では，ソースコード中のアルゴリズムの効率性の評価は困難であるが，ユーザへの効率的なアルゴリズムの推薦には欠かせない課題であることから，1 の課題について検討を行った．

1.2 本研究で対象とする問題

本研究では，ソフトウェア内に含まれるソースコード中の非効率な処理を，動的なソフトウェアの実行・ビルドなしに検出し，同様の処理を実現する効率的なソースコード提示によるユーザ支援を目的とする．この問題は，大きく分けて以下の 3 つの課題からなる．

(1) ソフトウェア内から非効率的な処理を行うソースコードの検出
ソフトウェア内で非効率的な処理が利用されているかを判定することで，効率的なアルゴリズムの推薦が必要であるか不要であるかを確認（**図 1** の 1）．

(2) 非効率的なアルゴリズムに対応する，効率的なアルゴリズムの選択
ソフトウェア内に非効率的な処理がある場合，具体的にどういったアルゴリズムであるかを判定（図 1 の 2）．

(3) 効率的なアルゴリズムをユーザの元のソースコードに近い形に整形して提示
効率的なアルゴリズムに該当するオンラインジャッジシステム内のソースコードに対して，ソフトウェア内のソースコードと対応する処理単位の切り出しや，ソフトウェア内のコードとの変数名の統一（図 1 の 3）．

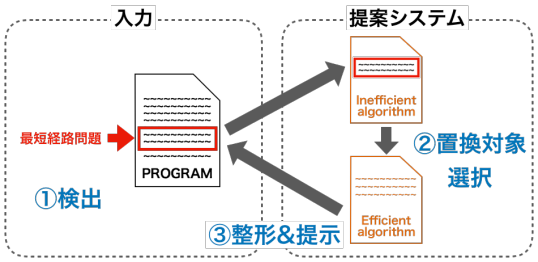

図 1 本研究で対象とする問題

効率性の評価とプログラムの等価性の評価を行うことができればこれらの問題は比較的容易に解決可能だが，これらは非常に困難な問題である．例えば，2 つのプログラムが等価であるためには，任意の入力において両プログラムの出力が同じでなければならない．またプログラムの効率性を比較するためには，一部の値だけでなく多種多様な値を入力とし，実行速度などの評価を行う必要がある．しかし，プログラムへの入力の組み合わせは膨大であるため，ソースコードの情報のみからプログラムの等価性や効率性を厳密に評価することは困難である．また，一般にソフトウェアは複数のプログラムからなり，ソフトウェアごとにビルド・実行を適切な手順で行う必要がある．このため，様々なソフトウェアを機械的にビルド・実行し，動作を動的に検証することは非常に困難である．

この技術の実現のため，本稿では 1 の課題に注目し，ユーザのソースコード中に非効率的な処理が含まれているかを判定し，含まれていれば効率性改善の余地があることを通知可能とする．これにより，実際のソフトウェア開発においてプロダクト内のソースコード中から，非効率な処理を行っている可能性のあるソースコードを発見することができる．本稿ではこれらの問題を解決するにあたり，オンラインジャッジシステムに提出されたソースコード群を利用した．オンラインジャッジシステムを利用するメリットとして，汎用的なアルゴリズムを実装したコードが多数存在すること，提出された全てのソースコードが動的なソフトウェアテストにより機能・性能を検証済みであることなどが挙げられる．このメリットは，一般にソフトウェアの動的な検証は困難であるという問題を解決可能であり，本稿でオンラインジャッジシステム上のソースコードをデータ

© 2018 Information Processing Society of Japan

として利用する最大の理由である．

2. 関連研究

2.1 コード補完

ユーザの入力したソースコードに対してソースコードを提示する試みの一つとして，コード補完が挙げられる．これは，同じコーパス内ですでに宣言されている変数名をタイプする際に，その一部をタイプするだけで該当変数名の提示を行ったり，入力したクラスのもつメンバ関数の一覧の提示を行う技術である．この技術は，ユーザのミスタイプの発見・訂正や，タイピング数の削減を支援することでプログラミングに必要な時間を短縮することができる．実現のためには，N-gram に基づいた言語モデル [3], [4] が多く利用されている．また，近年では深層学習を用いて言語モデルを作成する研究 [5] も存在する．コード補完は，ユーザに入力された文脈に沿ったコードを提示することで効率化を図るものであるが，ユーザの知識以上に高度なプログラムを作成することはできない．一方，本研究はユーザーの知識以上の効率的なアルゴリズムを提示する点でコード補完とは異なる．

2.2 プログラミング学習者向けソースコード提示

オンラインジャッジシステムでプログラミング学習を行うユーザ（以下学習者）の不正解であるソースコードに対して，正解したソースコードのうち類似のアルゴリズムを用いたものを提示することで，学習を支援する研究が存在する [6]．オンラインジャッジシステムには過去に提出されたソースコードが保管されており，学習者は他の学習者の正解したソースコードを閲覧できる．しかし，上級者による理解の困難なソースコードを閲覧しても，自力で理解し解答までたどり着くことは困難である．また，同じ問題を解くソースコードであっても実装方法は多彩であり，学習者と同じアルゴリズムを用いて正答しているソースコードを参考にすることが重要である．そこで，不正解となった他の学習者が正解となるまでに辿った道筋を分析する．そして，学習者のソースコードと同様のアルゴリズムが用いられたソースコードを，正解への道筋に従って漸進的に提示する．

学習者と同じアルゴリズムを用いているソースコード判別のため，この研究では N-gram IDF[7] を指標とした階層クラスタリングによる分類を行っている．N-gram IDF では任意の長さの単語列に重み付けすることができ，テキスト内の特徴的な単語 N-gram を抽出できる．正解への道筋は，オンラインジャッジシステムに保管されている提出履歴を用いて抽出する．この研究の適用対象はオンラインジャッジシステムを利用する学習者であり，一般のソフトウェアへの適用でない点が本研究とは異なる．

3. 非効率的なアルゴリズムと効率的なアルゴリズム

3.1 アルゴリズムの効率性の評価

アルゴリズムの汎用的な評価方法の代表的なものとして，オーダー記法による時間計算量および空間計算量を用いた評価が挙げられる．本研究ではこの時間計算量および空間計算量を軸として，ソースコードおよびアルゴリズムの効率性を評価する．

3.2 同一問題を解く効率の異なる複数のアルゴリズム

同一，あるいは類似の問題を解くアルゴリズムが複数存在し，かつ時間計算量や空間計算量が異なる場合が多数存在する．ここでは例として 最短経路問題を解くアルゴリズムを考える．最短経路問題を解くアルゴリズムにも様々な種類がある．dijkstra 法，bellman-ford 法， floyd-warshall 法などがその例である．頂点数を $|V|$，辺の数を $|E|$ としたときの各アルゴリズムの時間計算量は，dijkstra 法は $O(|V|^2)$ の計算量，dijkstra 法に二分ヒープを用いることで $O(|E|\cdot\log|V|)$ の計算量，bellman-ford 法は $O(|E|\cdot|V|)$ の計算量，floyd-warshall 法は $O(|V|^3)$ の計算量となる．計算量の観点では頂点数および辺の数の条件によらず，常に二分ヒープを用いた dijkstra 法を利用すべきである．しかし，必ずしも bellman-ford 法や floyd-warshall 法が劣っているとは言えない．何故ならば表 1 に示すように，dijkstra 法はコストが負の辺があるグラフに対して正しく最短経路を求めることができないからである．つまり，負の辺を含むグラフでの最短経路問題に対して bellman-ford 法を使用しているソースコードを検出しても，時間計算量の優れる dijkstra 法への置き換えは不可である．なお，負の辺を含まないグラフに対して bellman-ford 法を使用している場合は，もちろん dijkstra 法へ置き換え可能である．このように，時間計算量が優れていても置き換え不可である場合が存在するが，本研究では適用不可能である場合は提示後にユーザが棄却することを期待し，考慮から外すものとする．

4. オンラインジャッジシステムの性質

本稿では非効率的なソースコードであるかそうでないかを判定するため，オンラインジャッジシステム上のソースコードを利用する．本節では，オンラインジャッジシステムの特徴と利用する場合のメリットについて述べる．

4.1 オンラインジャッジシステムとは

プログラミング教育の一環として，プログラミングに競争の要素を導入することで意欲の向上を目的とした，プログラミングコンテストが多く開かれている．プログラミングコンテストには種類があり，自然言語によって記述され

© 2018 Information Processing Society of Japan

表 1　最短経路問題を解くアルゴリズムの比較

アルゴリズム	時間計算量	コストが負の辺
dijkstra 法（二分ヒープ不使用）	$O(\|V\|^2)$	非対応
dijkstra 法（二分ヒープ使用）	$O(\|E\| \cdot \log \|V\|)$	非対応
bellman-ford 法	$O(\|E\| \cdot \|V\|)$	対応
floyd-warshall 法	$O(\|V\|^3)$	対応

た問題文が仕様として与えられ，その問題の解答プログラムをいかに正確かつ短時間で作成することができるかを競うコンテスト [8] や，いかに実行速度の高速な解答プログラムを作成することができるかを競うコンテスト [9]，いかに少ない文字数の解答プログラムを作成することができるかを競うコンテスト [10] 等がある．有名なプログラミングコンテストとして，計算機科学分野の学会である ACM の主催する大学対抗コンテストである ACM-ICPC（1979〜）や，国際科学オリンピックの一つである情報オリンピック（1989〜）等がある．

プログラミングコンテスト環境のうち重要な要素の一つとして，参加者によって提出されたソースコードの正確さを自動的に検証するシステムが挙げられる．このシステムは，事前に決められた入力を用いて実行を行い，その結果に基づいて提出された解答の正確性を評価する．また，提出されたソースコードが規定リソース使用量（実行時間や使用メモリなど）を超えていないことも確認する．行われた評価に基づいて，全参加者のランキングが計算され，コンテスト実施中にリアルタイムで表示される．この，ユーザのソースコードをコンパイル・実行することで，プログラムの正確性の評価を自動的かつリアルタイムに行うシステムがオンラインジャッジシステムである．

4.2　オンラインジャッジシステムの仕組み

オンラインジャッジシステムは，ネットワーク上のサーバプログラムとして用意される図 2 のようなシステムで，コンテスト参加者から提出されたプログラムに対して正誤判定を自動で行う．システムは正誤判定のために入力データとその入力データに対する正解出力を入出力ケースとして用意し，システム上に保管しておく．コンテスト参加者はプログラムを完成させると，ソースコードをシステムに提出する．システムはユーザのソースコードをコンパイル・実行し，用意された入力データを入力として与える．このときの出力を正解出力と比較することで，システムは正誤を判定する．正誤判定では，出力結果の正しさの評価のみならず，コンパイルエラー，実行時エラー，規定リソース使用量超過の有無（非効率的なアルゴリズム使用による実行時間超過・メモリ使用量超過など）についても評価し，全ての項目において問題がない場合にのみ正解と評価する．なお，オンラインジャッジシステムは主にコンテストでの使用を目的としているため，誤りの詳細や不正解となった具体的な入出力については報告されない．

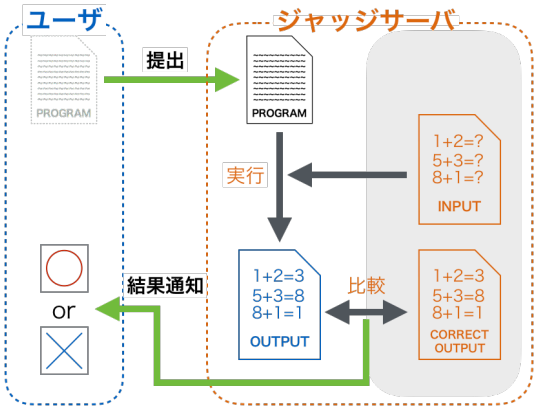

図 2　オンラインジャッジシステムの仕組み

例として，提出したソースコードが正誤判定によって実行時間超過と判定されるまでの流れを述べる．ここでは，4 つの入出力ケースに対して正しい答えを 5,000ms（5 秒）以内の実行時間で解答することが求められる場合として，図 3 の状況を考える．オンラインジャッジシステムは 1

	入力	出力	実行時間	結果
Case: 1	1+2=?	3	300[ms]	OK
Case: 2	5+3=?	8	400[ms]	OK
Case: 3	8+1=?	9	10,000[ms]	TLE
Case: 4	9+9=?			

図 3　正誤判定の流れ

つ目の入出力ケースから順に正誤判定を行う．その入出力ケースにおいて出力および規定リソース使用量に問題がなければ OK と判定し，同様に 2 つ目，3 つ目の入出力ケースへと進む．3 つ目の入出力ケースのように出力は正しいが規定実行時間を超過している場合や，そもそも出力が正しくない場合などはその場で不正解となり，それ以降の入出力ケースは実行されない．今回の例では規定実行時間超過により TLE（Time Limit Exceeded）と判定されている．

全ての入出力ケースにおいて OK と判定された場合は Accepted に，図 3 のように判定中に不正解となった場合は不正解と判定される．不正解にも種類が複数あり，出力が正しくないことを示す Wrong Answer，出力は正しいがメ

© 2018 Information Processing Society of Japan

モリの規定使用量を超過したことを示すは Memory Limit Exceeded，出力は正しいが規定実行時間を超過したことを示す Time Limit Exceeded などがある．

オンラインジャッジシステムにおいてどのような入出力ケースを用いてソースコードの正誤判定を行うかというのも非常に重要な問題である．例えば，足し算の繰り上がりにバグのあるプログラムを誤答と見抜くためには、繰り上がりの発生する入出力ケースが含まれている必要がある．また，非効率的で規定実行時間を超過しうるプログラムを誤答と見抜くためには、効率性によって大きな差が生まれる巨大な入出力ケースが含まれている必要がある．この問題は処理可能な値の全ての組み合わせを入力として試すことで回避可能だが，この組み合わせは膨大である．しかし、オンラインジャッジシステム内の各問題には問題作成者によって用意された誤りのあるプログラムを正解としないための多様な入出力が用意されており，信頼性の高い正誤判定が行われている．

4.3 オンラインジャッジシステムに保管されるソースコードの特徴

本稿では，ソフトウェア内から非効率的な処理を行うソースコードの検出を行うためにオンラインジャッジシステム内のデータを利用する．オンラインジャッジシステムに提出されている全てのソースコードは，検証用入出力ケースや検証器を用いて正確さと性能がすでに検証されている．今回利用する AOJ（Aizu Online Judge）[11] には 1000 問以上の問題が収録されており，300 万以上のソースコードが提出されている．これだけの量の動作や性能が検証されたソースコードが存在することは，オンラインジャッジシステム特有の大きなメリットである．また，多くのオンラインジャッジシステムはアルゴリズムの習得を目的としたものであり，さまざまなアルゴリズムを要求する問題が揃っている [12], [13]．そのため，本研究の目的である，非効率的なアルゴリズムによる処理の検出を行うためのデータとして利用する上で適切であると考えられる．

5. 機械学習による非効率的な処理の検出

本節では，5.1 節で分類問題の定義について，5.2 節で各特徴量の詳細とソースコードからの抽出方法について述べる．

5.1 非効率的/効率的なアルゴリズム分類問題の定義

1.2 節で述べたように，本稿ではユーザのソースコード中の非効率的なアルゴリズムの有無を検出する課題を解決する．そのためのアプローチとしてこの課題を，特定のアルゴリズムを含むソースコードに対して「非効率的なアルゴリズム」を含むか「効率的なアルゴリズム」を含むかを判定する 2 値分類問題と定義する．入力はプログラミング言語の構文を満たす一つのソースコードであり，出力は「非効率的なアルゴリズム」あるいは「効率的なアルゴリズム」であるという判定結果である．分類にはロジスティック回帰を用い，教師データとしてオンラインジャッジシステム内のソースコードを用いる（詳細は 6.1.1 項に後述）．本稿では 5.2 節に示すように，ソースコードを構成する文字列から得られる様々な特徴量を用いた分類を行う．分類問題の評価には分類精度（classification accuracy），再現率（recall），適合率（precision）などの指標があるが，本研究では分類精度と適合率に注目して評価を行う．本研究では特に，非効率的なソースコードが正しく非効率的だと判別される（True Positive の値が大きい）こと，また効率的なソースコードが誤って非効率的だと判別されない（False Positive の値が小さい）ことが重要であるため，適合率が高いことがより重要である．表 2 のように分類結果を定義すると，分類精度は式 1 で，適合率は式 2 で表される．

$$classification\ accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

$$precision = \frac{TP}{TP + FP} \quad (2)$$

5.2 特徴量

ソースコードが効率的か否かを判断するため，各ソースコードの文字列的特徴および構造的特徴を特徴量として利用する．表 3 に，利用する特徴量の一覧を示す．なお，分類の際に各特徴量の値は 0〜1 の値に正規化して利用している．

5.2.1 N-gram IDF 特徴量

2.2 節に示した関連研究でも使用されていた，N-gram IDF を特徴量として用いる．この特徴量により，ソースコード中のトークン N-gram による表層的な特徴を分類に利用可能とする．ソースコード全体から N-gram IDF 値の高い，特徴的な単語 N-gram（N $\leq$ 5）のリストを作成し，各ソースコードでの各単語 N-gram の出現回数を特徴量として用いる．N-gram IDF 値の計算には，N-gram IDF に関する論文 [7] の著者である Shirakawa らが開発しているツール，N-Gram Weighting Scheme[14] を用いる．なお，前処理として各ソースコード中の大文字を全て小文字に変換する．

5.2.2 コード中で宣言された変数の型特徴量

ソースコード中で宣言された変数，およびデータ構造を特徴量として用いる．この特徴量により，アルゴリズムの効率性を分ける要因の一つである，データ構造の情報を分類に利用可能とする．この特徴量は，ANTLR（ANother Tool for Language Recognition）[15] を用いて作成した C++パーサによって変数の宣言部を検出することで抽出する．

5.2.3 制御構文によるネスト特徴量

ソースコード中の最も深いネストの深さ，および制御構文によるネストからなるスコープの個数を特徴量として用

© 2018 Information Processing Society of Japan

表 2　混同行列の例

		Label	
		非効率的	効率的
Predict	非効率的	True Positive(TP)	False Positive(FP)
	効率的	False Negative(FN)	True Negative(TN)

表 3　使用する特徴量

特徴量	概要
Ngram_IDF	各単語 N-gram-IDF の出現回数
variable_type	コード中で宣言された変数の型
depth_of_max_nest	制御構文による最も深いネストの深さ
#_of_nest	深さ 1 以上の独立なネストの数
#_of_for	コード中の for ループの数
#_of_while	コード中の while ループの数
#_of_if	コード中の if 文の数

いる．この特徴量により，計算量のボトルネックとなることの多い，処理の中心となるループを検出し，分類に利用可能とする．この特徴量は，自作の簡易的な構文解析器により for 文・while 文・if 文 などの制御構文を検出することで抽出する．

5.2.4　制御構文特徴量

ソースコード中の for 文・while 文・if 文といった制御構文の出現頻度を，特徴量として用いる．この特徴量により，アルゴリズムの概形を表現する制御構文が，それぞれどれほど使用されているかを検出し，分類に利用可能とする．この特徴量は，N-Gram Weighting Scheme[14] の機能を用いてソースコードをトークナイズし，各制御構文の出現回数をカウントすることで抽出する．

6.　評価実験

本節では，オンラインジャッジシステムのデータを用いてソースコードの非効率的/効率的を判定する手法の評価を行う．評価には，オンラインジャッジシステム上のソースコードを対象とした実験を用いた．

6.1　オンラインジャッジシステム内ソースコードからの非効率的な処理の検出

6.1.1　実験設定

ソースコード中の非効率的なアルゴリズムを検出する手法の評価のため，オンラインジャッジシステム上のソースコードを対象とした実験を行った．オンラインジャッジシステム上のソースコードは，実行時間や正確さが評価されている．本実験では対象とする問題を最短経路問題に限定し，非効率的/効率的 のラベルのついたソースコードを教師データとした，効率的/非効率的 をロジスティック回帰により判定する 2 値分類器を設計した．

次に，教師データの詳細について述べる．オンラインジャッジシステム上の最短経路問題を解くソースコードのうち，少なくとも前半 50%のケースで OK と判定されており（正誤判定については 4.2 節参照），なおかつ規定実行時間を超過しているソースコードを非効率的なアルゴリズム，正解率が 100%であるようなソースコードを効率的なアルゴリズムとし，作成した．非効率的なアルゴリズムとして採用する際に正解率の条件を考慮するのは，単に無限ループなどを含みそもそも最短経路問題を解けていないソースコードを教師データに含めないためである．なお，ここでは提出数の多さを理由として C++で記述されているソースコードのみを対象とした．結果として得られた教師データの情報を表 4 に示す．

本節では，2.2 節の関連研究を参考に N-gram IDF のみを特徴量としてロジスティック回帰で分類を行ったものをベースライン手法，表 3 に示す特徴量を用いてロジスティック回帰で分類を行ったものを提案手法とし，それぞれ 5-fold の交差検証で性能を評価した．また，ロジスティック回帰の正則化には L2 正則化を用い，正則化パラメータ $\lambda = 0.5$ とした．利用するソースコードに対し前処理として，コメントアウト部分の除去・#include 文の除去・#define マクロの置換 を行った．

6.1.2　実験結果と考察

本項では，6.1.1 項で述べた設定で実験を行った結果を示す．結果は 30 回試行の平均の形で，表 5 に示す．ベースライン手法に対して提案手法がわずかに高い平均適合率と平均分類精度を示しているが，有意差は認められなかった．いずれの実験も学習データにおける分類精度は 100%であったことから，学習データへのオーバーフィッティングとみなせる．

ここで，実験プログラムにおける乱数のシード値を固定したときの，それぞれの手法において重みの絶対値の大きい素性を表 6，表 7 に示す． ベースライン手法においても提案手法においても，分類において N-gram IDF 特徴量が非常に有効に作用していることがわかる．特筆すべきは，ベースライン手法では「dijkstra r」という 2-gram が分類に有効であるが，一方提案手法では上位 10 件に登場していない点である．この 2-gram は，ソースコード中の「dijkstra(r)」という，問題文中で始点となる頂点を表す変数 r を引数とした dijkstra 関数の呼び出しに相当する．一見これは効率的なアルゴリズムを判別する有効な特徴量であるが，表 1 に示すように dijkstra 法には 2 つの種類が存在し，二分ヒープを使用しない dijkstra 法の場合は効率的なアルゴリズムと言うことはできない．その点二分ヒープ（C++では priority_queue コンテナ変数）の存在を考慮

© 2018 Information Processing Society of Japan

表 4　教師データの情報

	件数	割合	備考
非効率的なアルゴリズム	166 件	24.9%	TLE かつ正解率 50%以上
効率的なアルゴリズム	501 件	75.1%	Accepted

表 5　実験結果

手法	平均適合率	平均分類精度	最低 - 最高分類精度	標準偏差
ベースライン	69.75%	81.72%	80.30% - 83.46%	0.0073
提案手法	69.80%	82.01%	80.45% - 83.31%	0.0078

表 6　ベースライン手法において分類に有効な特徴量上位 10 件

特徴量のタイプ	値	係数
N-gram	if update	-0.936
N-gram	1 break	0.631
N-gram	min c	0.609
N-gram	i push back	0.516
N-gram	c s	-0.512
N-gram	r scanf d d d	0.483
N-gram	dijkstra r	-0.433
N-gram	que empty	-0.430
N-gram	edge e g v	-0.411
N-gram	push r	-0.407

表 7　提案手法において分類に有効な特徴量上位 10 件

特徴量のタイプ	値	係数
N-gram	if update	-0.900
N-gram	1 break	0.618
N-gram	min c	0.606
N-gram	c s	-0.515
N-gram	i push back	0.501
variable	priority queue p	0.494
N-gram	r scanf d d d	0.463
N-gram	que empty	-0.408
N-gram	top pq	-0.393
N-gram	push r	-0.390

表 8　ベースライン手法における混同行列

		Label	
		非効率的	効率的
Predict	非効率的	19	5
	効率的	16	93

表 9　提案手法における混同行列

		Label	
		非効率的	効率的
Predict	非効率的	19	6
	効率的	16	92

可能な提案手法においては「dijkstra r」という 2-gram が上位 10 件に登場しないことは妥当な結果と言える．

次に，実験プログラムにおける乱数のシード値を固定したときの，混同行列を表 8，表 9 に示す．

これらの混同行列は概ね同じ結果となっているが，ベースライン手法では正しく効率的であると分類できていた 1 ケースを，提案手法では非効率的であると分類していることがわかる．以下に，該当するソースコードの一部を添付する．

```
1  using namespace std;
2  typedef long long int ll;
3  typedef pair<int,int> P;
4  typedef vector<int> vi;
5  inline int in() { int x; scanf("%d",&x);
       return x;}
6  inline void priv(vi& a) { for(int i = 0; i < (
       int)(a).size(); ++i) printf("%d%c",a[i],i
       ==(int)(a).size()-1?'\n':' ');}
7  const int MX = 100005, INF = 1000010000;
8  const ll LINF = 1000000000000000000ll;
9  const double eps = 1e-10;
10 const int di[] = {-1,0,1,0}, dj[] =
       {0,-1,0,1};
11 struct Dijk {
12   typedef int TT;
13   struct dat {
14     TT d; int v;
15     bool operator<(const dat& a)const{return d
       >a.d;}
16   };
17   int n;
18   vector< vector<dat> > to;
19   Dijk(int n):n(n),to(n){}
20   void add(int a, int b, TT c) {
21     to[a].push_back((dat){c,b});
22   }
23   vector<TT> dist; vi pre;
24   void solve(int sv) {
25     dist = vector<TT>(n,INF); pre = vi(n,-1);
26     priority_queue<dat> q;
27     dist[sv] = 0;
28     q.push((dat){0,sv});
```

注目すべきは 2〜13 行目に見られる，型名指定子（typedef マクロ）や，関数指定子（inline マクロ）や，const 修飾子である．この指定子の役割は，例えば 2 行目の「typedef ling long int ll;」であれば long long int 型の変数を宣言する際に long long int と入力する代わりに ll と入力することで同様の機能を実現することが可能となる．この実質的な置換機能により，特徴量として本来 long long int という型

© 2018 Information Processing Society of Japan

が情報として取得できるはずが ll という未知の型が得られてしまう．このことから，本ソースコードのような置換機能が多く用いられたものが入力として与えられた場合，提案手法において追加された特徴量が正しく働いていない可能性が考えられる．本実験では，define マクロについては対応済みであるが，typedef などの置換については未対応である．これは，置換の実現のためには C++ソースコードに対する複雑な構文解析処理が必要となるためである．

7. おわりに

本稿では，効率的なアルゴリズム実装の推薦という問題を，達成に必要な課題に分割し，その第一歩である入力ソースコード中の非効率的なアルゴリズムの有無の検出を行った．オンラインジャッジシステム内のソースコードを教師データ，入力ソースコードの文字列的特徴および構造的特徴を特徴量としたロジスティック回帰により分類を行った結果，約 82%の分類精度で非効率的なアルゴリズムの有無を判別できた．また，非効率的であると判定されたソースコードのうち約 70%は，実際に非効率的なアルゴリズムを用いたソースコードであると正しく判定された．一方残り約 30%のソースコードでは，効率的なアルゴリズムを用いているにも関わらず非効率的であると誤って判定された．表 6，表 7 に示した有効な特徴量一覧の比較により，提案手法で採用した，宣言された変数の型に関する特徴量が有効であることが確認できた．さらなる精度向上には，型名指定子（typedef マクロ）や関数指定子（inline マクロ）などによる置換に対応すべく，特徴量の抽出方法のさらなる工夫が必要であると考えられる．

一方，本研究の最終的な目的である，ユーザへの効率的なアルゴリズム実装推薦のためには，検出した非効率的なアルゴリズムに対応する，効率的なアルゴリズムの選択，および効率的なアルゴリズムをユーザの元のソースコードに近い形に整形して提示という 2 つの課題が残されており，本稿を端緒として引き続き課題に取り組む必要がある．より効率的なアルゴリズムの選択のためには，非効率的であると判定されたソースコードがどのようなアルゴリズムを使用しているのかを判別するという点が重要であると考えられる．また，効率的なアルゴリズムを整形して提示するためには，非効率的であると判定されたソースコード中の変数名や構文的構成など，より複雑な情報について分析を行う必要があると考えられる．加えて，本稿で取り組んだ入力ソースコード中の非効率的なアルゴリズムの有無の検出に関しては，dijkstra 法以外のアルゴリズムへの適用対象拡張が今後の課題となる．また，さらなる特徴量の追加や，入力ソースコード中の typedef マクロなどによる置換処理の適用による分類精度・適合率の向上が課題である．

謝辞

オンラインジャッジシステムのデータを提供して下さった，会津大学 渡部 有隆 上級准教授に感謝致します．

参考文献

[1] Nong, Ge, Sen Zhang, and Wai Hong Chan. 2009.“Linear Suffix Array Construction by Almost Pure Induced-Sorting.”In 2009 Data Compression Conference, IEEE, 193202. http://ieeexplore.ieee.org/document/4976463/.
[2] Manber, Udi, and Gene Myers. 1993. “Suffix Arrays: A New Method for On-Line String Searches.” SIAM Journal on Computing 22(5): 93548. http://epubs.siam.org/doi/10.1137/0222058.
[3] Hindle, Abram et al. 2012.“On the Naturalness of Software.” 2012 34th International Conference on Software Engineering.
[4] Nguyen, Tung Thanh, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2013. “A Statistical Semantic Language Model for Source Code.” In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, New York, New York, USA: ACM Press, 532. http://dl.acm.org/citation.cfm?doid=2491411.2491458.
[5] Nguyen, Anh Tuan, Trong Duc Nguyen, Hung Dang Phan, and Tien N. Nguyen. 2018.“A Deep Neural Network Language Model with Contexts for Source Code.” In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 32334. http://ieeexplore.ieee.org/document/8330220/.
[6] 中川 尊雄, 藤原 新, 畑 秀明, and 松本 健一. 2016.“プログラミング学習者向けソースコード提示システム TAMBA.” ソフトウェアエンジニアリングシンポジウム 2016 論文集: 34–41.
[7] 白川 真澄, 原 隆浩, 西尾 章治郎. 2015.“コルモゴロフ複雑性に基づく IDF の単語 N-Gram への適用.” データ工学と情報マネジメントに関するフォーラム (DEIM 2015).
[8] ACM: The ACM-ICPC International Collegiate Programming Contest Web Site sponsored by IBM.
[9] http://sortbenchmark.org/
[10] http://codegolf.com/
[11] http://judge.u-aizu.ac.jp/onlinejudge/
[12] 渡部 有隆. 2015. “オンラインジャッジの開発と運用 — Aizu Online Judge —.” 56(10): 9981005.
[13] WASIK, SZYMON et al. 2018. “A Survey on Online Judge Systems and Their Applications.” Lakartidningen 108(3): 9091.
[14] “N-Gram Weighting Scheme.” https://github.com/iwnsew/ngweight.
[15] “ANTLR.” https://www.antlr.org/.

© 2018 Information Processing Society of Japan