修士学位論文

題目

プログラミングコンテスト 初級者・上級者間における ソースコード 特徴量の比較

指導教員
井上 克郎 教授

報告者
堤 祥吾

平成 30 年 2 月 7 日

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

平成 29 年度 修士学位論文

プログラミングコンテスト 初級者・上級者間におけるソースコード 特徴量の比較

堤 祥吾

内容梗概

現在，プログラミングコンテストはプログラミング初級者から上級者まで世界中から多くの参加者が利用している．プログラミングコンテストでは，参加者にアルゴリズムに関する問題が提示され，参加者は問題に対する回答ソースコードを速く正確に作成して提出する．ソースコードは，コンテストサイトに用意されたオンラインジャッジシステムによって採点され，正答した場合は参加者に得点が与えられる．参加者は得点に応じて順位が決定し，レーティングが変動する．このレーティングが高いほど，コンテストにおける順位の期待値が高い．

本研究では，プログラミングコンテストに公開されている大規模な提出履歴情報やソースコードデータと参加者のレーティングに着目し，プログラミングコンテスト 上級者・初級者間におけるソースコード編集作業の差異について調査を行った．本研究の調査結果を利用することで，素早く適切にアルゴリズムを組むために必要な編集作業の特徴を理解し，プログラミング初級者の教育に役立てることができる．調査にあたって，世界最大規模のプログラミングコンテストシステムである Codeforces より構築した 1,644,636 のソースコードデータとそれに付随する提出履歴情報データベースを用いた．既存の研究では，大規模な人数に対応したソースコードデータの収集や，技術力に対する定量的な評価が難しかったが，本研究ではプログラミングコンテストへの提出データとレーティングを用いることでその問題を解決した．

調査の結果，初級者は上級者と比較して分岐文の割合が増加する傾向にあるという特徴を，ソースコードへの静的解析を行うことにより明らかにした．また，提出履歴への調査を行うことにより，上級者の修正回数や修正量が，初級者と比較して少なくなるということがわかった．

本研究に用いたデータセットは，ソースコードデータと提出履歴情報データに分け，オンラインで公開している．

主な用語

プログラミングコンテスト
ソースコード 修正
レーティングシステム

目 次

1 まえがき

ソースコードの編集は，ソフトウェア開発において必須の作業であり，コンピュータサイエンスではソースコードの編集作業に関する研究は数多く行われている [5,6,14]．本研究におけるソースコードの編集とは，ソースコードの実装・修正等，ソースコードに対して行う改変作業全般を指す．ソフトウェア開発においてはソースコードの実装，テスト，ソースコードの修正を繰り返し，ソースコード編集にかける労力は大きい．特にソフトウェア開発の初級者は上級者と比較して，ソースコードの編集に大きな時間的コストがかかる．そのため，開発者の技術力による編集作業の差異を調査し，初級者が上級者の編集作業を参考にすることでその問題を解決することができる．しかし，開発者の技術力と編集作業との関係についての調査は難しい．理由として，開発者の技術力を定量的に評価することが難しい点や，開発者とソースコードとの結び付けが行われたデータを大量に集めることが難しいことが考えられる．

現在，Topcoder[1]や Codeforces[2]を始めとするプログラミングコンテストサイトが，プログラミング経験者の関心を集めている [11]．典型的なプログラミングコンテストサイトにはレーティングシステムが導入されており，これによって参加者が高いレーティングを取ろうと努力する動機の一つとなっている．また，多くのプログラミングコンテストサイトはオンラインジャッジシステム [12,13] を導入している．オンラインジャッジシステムは参加者に問題を公開し，参加者の回答ソースコードの採点を行う．ソースコードはオンラインジャッジシステムによってコンパイルされ，あらかじめ用意された多くのテストケースを用いて採点する作業が自動化されているため，大規模なコンテストを開催することが可能となっている．

本研究では，世界最大規模のプログラミングコンテストである Codeforces の参加者を対象に，参加者がコンテストの問題へ回答として提出したソースコードの編集作業について調査を行う．特にソースコードにおける特徴量や修正箇所，修正回数がプログラミングコンテスト上級者・初級者間でどのように変化するかについて分析を行い，ソースコード編集者の熟練度と編集作業の特徴との間にどのような関係があるかを明らかにする．本研究における結果である上級者のソースコード編集の特徴を初級者が参考にすることで，初級者のプログラミング技術の向上につながると考えられる．

ソースコード編集作業の特徴調査を行うために，2016/5/19〜2016/11/15 の 6ヶ月間に Codeforces に対して提出された 1,644,636 のソースコードと，それに付随した提出履歴情報を収集した．このうち，収集ソースコードの 9 割を占める C++でのソースコードを本研究での調査対象とした．Codeforces によって提供されている API よりデータを取得し，デー

[1] https://www.topcoder.com/
[2] http://codeforces.com/

タベースに保管した．本研究にて収集したソースコードデータと提出履歴情報はオンラインで公開している[3]．

プログラミングコンテスト上級者･初級者間におけるソースコードの編集作業における特徴調査を行ったところ，次のようなことが明らかになった．ソースコード特徴量について，初級者は上級者と比較して分岐文の割合が増加する傾向にあるということ．ソースコード修正について，上級者は初級者と比較して1度当たりの修正量が少ないということ，また，より局所的な修正が多いということである．

以下，2章では本研究のデータセットとして利用するプログラミングコンテストについて説明する．3章では，プログラミングコンテストサイト Codeforces から構築したデータセットの内容について説明する．4章では，提出されたソースコードの編集作業分析を行い，5章では結果についての考察を行う．6章でまとめと今後の方針について議論する．

[3]https://sites.google.com/site/miningprogcodeforces/

2 背景

この章では，本研究の背景となるプログラミングコンテストと，ソースコード編集作業特徴調査の目的について述べる．プログラミングコンテストには様々な種類があるが，本研究ではアルゴリズムに関する問題を時間内に解く種類のコンテストについて述べる．以下ではまず，プログラミングコンテストに用いられるオンラインジャッジシステムと，プログラミングコンテストの概要について説明する．その後，本研究で行うソースコード編集作業の特徴調査にあたって必要となる用語や本研究の必要性について説明する．

2.1 オンラインジャッジシステム

プログラミングコンテストにおいて，その採点に利用されるオンラインジャッジシステムについて説明する．オンラインジャッジシステムは，利用者に問題を提示し，問題に対する利用者の回答を受け取る．そして，受け取った回答を採点し，結果を利用者に通知する．オンラインジャッジシステムの一例として，国内で代表される AIZU ONLINE JUDGE[4]，アメリカの Topcoder が存在する．

オンラインジャッジシステムが提示する問題には，問題文，サンプルテストケース，実行時間やメモリ制限等の実行上の制約などが含まれる．利用者は制約を満たすプログラムを作成し，そのソースコードをシステムに提出する．システムは提出されたソースコードをコンパイルし，予め用意されている複数のテストケースを実行する．実行後，テストケースを通過すれば正解を，間違った出力や実行制限を違反した場合はその旨を利用者に通知する．システムによってソースコードがコンパイル，実行されるため，システムに環境が用意されている任意の言語で提出することができる．

2.2 プログラミングコンテスト

プログラミングコンテストは，オンラインジャッジシステムを用いて複数の参加者が同じ問題セットを同じ時間に解く方式で行われる．参加者の正解問題数や回答時間に応じて参加者の順位が決定し，コンテスト終了後に順位に応じて参加者のレーティング [3, 10] が変動する．プログラミングコンテストには ACM ICPC[5]のように複数の参加者がチームを組んで回答するものも存在するが，本研究では個人で参加する種類のプログラミングコンテストを対象とする．

[4] http://judge.u-aizu.ac.jp/onlinejudge/
[5] https://icpc.baylor.edu/

2.2.1 ルールと流れ

プログラミングコンテストの実際の流れについて，本研究でデータセットとして用いるCodeforcesを例として説明する．

Codeforcesにおける一般的なコンテストでは，指定の時間になるとコンテスト参加者に複数の問題が公開される．参加者は自由な順序・プログラミング言語で問題を解くことができ，解いた問題の難易度と回答までにかかった時間に応じて参加者に得点が加算される．参加者は各問題に対して何度でも回答を提出することができ，回答ソースコードの正誤やコンパイル可能性については提出の可否に影響しない．回答ソースコードは提出された時点でCodeforcesに用意されたオンラインジャッジシステムによってコンパイル，事前テストの実行が行われる．事前テストにはテストケースの一部が用いられ，事前テストが通過したか否かについては提出後に参加者へ通知される．事前テストの通過は全テストケースの通過を保証しないため，事前テストを通過していても回答が正しくない場合もある．事前テストの結果を確認し，参加者は再提出を行うか別の問題に回答するかの判断を行う．

コンテスト時間終了後に提出ソースコードに対して最終テストが実施され，参加者の最終的な得点が決定する．得点に応じて各参加者の順位が決定し，レーティングが変動する．

2.2.2 開催規模や参加者

本研究で利用するCodeforcesにおいては，

- 開催頻度: 月に2，3回程度のコンテスト開催
- 参加人数: 毎コンテスト7000人程度
- 国籍: 全世界から参加 (言語はロシア，英語)

となっている．また，過去6か月以内に一度でもコンテストに参加したことのあるユーザーをCodeforcesではアクティブユーザーと定義しているが，Codeforcesにおけるアクティブユーザー数は2017年10月10日現在，30796人となっている．

2.2.3 レーティングシステム

Codeforcesは参加者の熟練度を示す指標としてレーティングシステム [10]を用いている．主にチェスやサッカーで用いられるイロレーティング [3]に似た計算方式を採用しており，コンテスト毎に参加者の順位に応じてレーティングを計算する．レーティングが高い参加者ほどコンテストで高い成績を取っているということができる．本研究においても，このレーティングが高い参加者を熟練度が高い参加者とし，アルゴリズムの問題に対して適切にコーディングを行う能力が高いと考える．

イロレーティングは 2 者間における強さの指標を表す．A と B のレーティングをそれぞれ r_A, r_B と表すとき，A が B に勝利する確率が，

$P(r_A, r_B) = \frac{1}{1+10^{\frac{r_B - r_A}{400}}}$

となるようにレーティングが調整される．プログラミングコンテストでは，A が B の得点を上回る確率といえる．Codeforces においてはコンテスト終了後の順位を用いてレーティングの変更計算を行う．Codeforces でのレーティング変更計算のアルゴリズムを Algorithm 1 に示した．(1) ではおおよそのレーティング変動を計算している．$seed_i$ は，レーティング r_i の参加者の順位期待値を表し，$seed_i$ と実際の順位 $rank_i$ との幾何平均を m_i とする．ここで，$getSeed(R_i) = m_i$ となるようなレーティング R_i の探索を行い，この R_i と r_i の平均が参加者 i の暫定的なレーティングとなる．また，上位参加者のレーティングがインフレしてしまうことを防ぐために (2) の調整を行っている．(1) と (2) によって計算されたレーティングの変動を r_i に加え，最終的な参加者のレーティングを得られる．

2.3 ソースコード編集作業特徴収集の必要性

ソースコード編集作業はソフトウェア開発において必須作業であり，コンピュータサイエンスではソースコード編集作業について多方面から研究が行われている [5,6,14]．本研究における編集作業とは，実装や修正等のソースコードに対して行われる作業全般を指す．ソースコード編集作業はソフトウェア開発を行う上で大きな労力を必要とするが，特にソフトウェア開発の初級者は上級者と比較して編集作業に大きな時間的コストを必要とする．そこで，開発者の技術力による編集作業の差異を明らかにすれば，初級者の編集作業上の改善点が判明し，ソフトウェア開発教育の効率を上げることができる．しかし，初級者と上級者を比較した大規模な実証研究は難しい [1,7]．主要な理由として，ソフトウェア開発者の技術力を定量的に評価することが難しい点，多くの開発者から大規模なデータを集めることが難しい点が挙げられる．そのため，ソースコードの編集作業に関して定量的な評価観点を有する大規模なデータを集め，実証研究を行うことは有用である．

ソースコードの編集技術に関して定量的な評価を行うため，本研究では大手プログラミングコンテストサイトである Codeforces より大規模なソースコードデータとそれに付随する提出履歴情報を収集し，Codeforces でのレーティングと結び付けて分析を行った．調査の結果として，上級者と初級者における初回提出ソースコードの特徴や，ソースコードへの修正作業にどのような差異が存在するかを明らかにした．

Algorithm 1: Codeforces におけるレーティング変更計算

Data: U : コンテスト の全参加者

Data: r_i : 参加者 i のコンテスト 前のレーティング

Data: $rank_i$: 参加者 i のコンテスト での順位

Result: r'_i : 参加者 i の変化後のレーティング

Function *getSeed(r)*

  return $\sum_{i \in U} P(r_i, r) + 1$;

Function *getRatingToRank(r)*

  $left \leftarrow 1$;

  $right \leftarrow 8000$;

  while $right - left > 1$ **do**

    $mid \leftarrow \frac{left+right}{2}$;

    if $getSeed(mid) < r$ **then**

      $right \leftarrow mid$;

    else

      $left \leftarrow mid$;

  return *left*;

$sum \leftarrow 0$;

for $i \in U$ **do** // (1)

  $seed_i \leftarrow getSeed(r_i)$;

  $m_i \leftarrow \sqrt{seed_i * rank_i}$;

  $R_i \leftarrow getRatingToRank(m_i)$;

  $d_i \leftarrow \frac{R_i - r_i}{2}$;

  $sum \leftarrow sum + d_i$;

for $i \in U$ **do** // レーティング変動の合計を 0 にする

  $d_i \leftarrow d_i - (\frac{sum}{|U|} + 1)$;

// (2) 上位のレーティングのインフレを抑える処理

$topU \leftarrow U$ の上位 $min(|U|, 4\sqrt{|U|})$ 人;

$sum \leftarrow \sum_{i \in topU} d_i$;

$inc \leftarrow min(max(-\frac{sum}{|topU|}, -10), 0)$;

for $i \in U$ **do** // 最終的なレーティングの決定

  $d_i \leftarrow d_i + inc$;

  $r'_i \leftarrow r_i + d_i$;

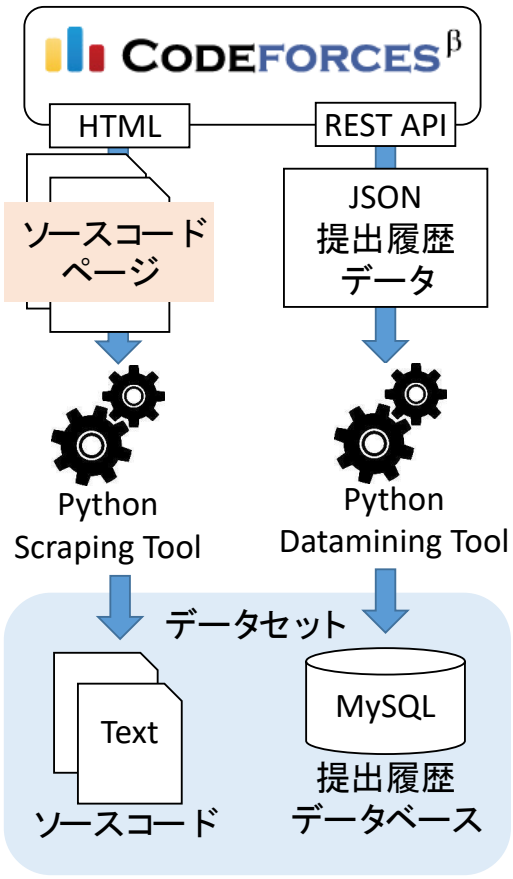

図 1: データセット 構築手段

3 データセット 構築

本章では，本研究で用いるデータセットの内容と収集方法について述べる．本研究では世界最大規模のプログラミングコンテストサイトである Codeforces から提出履歴を収集している．Codeforces の選定理由としては，他のサービスと比較してアクティブユーザーが多いこと，提出履歴が公開されておりアクセスしやすいこと，提出履歴取得用の API が整備されていることが挙げられる．大手プログラミングコンテストサイトの比較を表 1 にまとめている．

[6]各問題に対する最終提出のみ閲覧可能
[7]https://www.codechef.com/
[8]過去 6ヶ月以内に 1 度でも提出したことがあるユーザー数

#	When	Who	Problem	Lang	Verdict	Time	Memory
24802974	2017-02-19 16:28:54	wucebrayne	E - Change-free	GNU C++14	Wrong answer on test 1	0 ms	3100 KB
24802963	2017-02-19 16:28:25	Vlad3d	D - Cartons of milk	Java 8	Accepted	1762 ms	96400 KB
24802962	2017-02-19 16:28:22	delete1	C - Garland	GNU C++11	Wrong answer on test 8	30 ms	17700 KB

図 2: Codeforces の提出履歴ページのスクリーンショット

表 1: 各種プログラミングコンテスト サービスの特徴

名前	AU	提出履歴	API
Codeforces	30796	○	○
Topcoder	3675	△[6]	○
CodeChef[7]	12129[8]	○	×

3.1 データセット の収集方法

本研究で用いるデータセット 構築にあたって，Codeforces に公開されている提出履歴情報と提出ソースコードを取得した．図 2 は Codeforces のコンテスト 767 における提出履歴ページのスクリーンショット であり，左から順に提出 ID，提出時刻，提出者，対象問題，言語，正誤，実行時間，実行時使用メモリを表す．これらの情報は図 1 の右に表される通り，Codeforces が提供する API[9]から取得することができる．Codeforces の API は JSON 形式でデータを返すようになっているため，これを取得してリレーショナルデータベースに保存した．本研究では，マイニングスクリプトとして Python を，リレーショナルデータベースとして MySQL を利用した．構築されたデータセットは Codeforces で公開されているものの一部であり，得られるデータの詳細や API の利用方法については Codeforces 公式 API ページ[10]から得ることができる．

ソースコード 収集については図 1 に表されている．参加者が提出したソースコードについては API が提供されていないため，ソースコード 収集用のマイニングスクリプトを作成し，それを用いてソースコード データを集めた．ソースコード データは Codeforces 上で HTML ページとして公開されており，ソースコード ページの URL は提出履歴情報に含まれる提出 ID とコンテスト ID から構築される．そのため，提出履歴情報を収集することにより，ソースコード データも集めることができる．

3.2 データセット の内容

[9]http://codeforces.com/api/help
[10]http://codeforces.com/api/help/methods

表 2: データセットの内訳

テーブル名	カラム名	キー	説明	データ型
Participant	user_name	PK	Codeforces の参加者名	String
	rating		参加者の現在のレーティング	Integer
	max_rating		2016/11/15までの最大到達レーティング	Integer
Participant-Submission	user_name	PK FK	*Participant* テーブルにおける *user_name*	Integer
	files		データセット内における *user_name* の提出数	Integer
File	file_name	PK	データセット上でのソースファイル名	String
	submission_id		Codeforces における提出 ID	Integer
	lang		ソースファイルのプログラミング言語	String
	user_name	FK	ソースファイルの提出者	String
	verdict		提出の正誤	String
	timestamp		提出時刻	Date/Time
	competition_id	FK	*Competition* テーブルにおける *competition_id*	Integer
	problem_id		この提出に対応する *problem_id*	String
	url		Codeforces におけるこのソースファイルの URL	String
	during_competition		この提出がコンテスト時間内に提出されたかどうか	Boolean
Competition	competition_id	PK	コンテスト ID	Integer
	name		コンテスト名	String
	start_time		コンテスト開始時刻	Date/Time
	duration_time		コンテスト時間	Integer
	participants		コンテスト参加者数	Integer

Problem	problem_id	PK	問題 ID	String
	competition_id	FK	この問題が掲載されたコンテストの *competition_id*	Integer
	prob_index		Index of the problem in the competition	String
	points		コンテストにおけるこの問題の得点	Integer
Acceptance	problem_id	PK FK	対応する問題の *problem_id*	String
	submission_in_sample		データセット上におけるこの問題に対する提出数	Integer
	solved_in_sample		データセット上におけるこの問題に対する正答数	Integer
	submission		この問題に対する全提出数	Integer
	solved		この問題に対する全正答数	Integer
	acceptance_rate		*solved/submissions*	Double
	lastmodified		データの最終更新日	Date/Time
Problem-Submission-Statistics	problem_id	PK	対応する *problem_id*	String
	filesize_max		この問題に対する提出ファイルサイズの最大値	Integer
	filesize_min		この問題に対する提出ファイルサイズの最小値	Integer
	filesize_mean		この問題に対する提出ファイルサイズの平均値	Integer
	filesize_median		この問題に対する提出ファイルサイズの中央値	Integer
	filesize_variance		この問題に対する提出ファイルサイズの分散	Integer
	filesize_max_competition		*filesize_max* のうちコンテスト中に提出されたもの	Integer
	filesize_min_competition		*filesize_min* のうちコンテスト中に提出されたもの	Integer

	filesize_mean_competition		*filesize_mean* のうちコンテスト中に提出されたもの	Integer
	filesize_median_competition		*filesize_median* のうちコンテスト中に提出されたもの	Integer
	filesize_variance_competition		*filesize_variance* のうちコンテスト中に提出されたもの	Integer
Submission-Distance	file_name	PK	対応するソースファイル名	String
	problem_id		この提出に対応する *problem_id*	String
	next_file		*file_name* の次の提出	String
	submission_index		同じ問題に対してこの提出が何番目の提出か	Integer
	levenshtein_distance		*file_name* と *next_file* とのトークンベースの編集距離	Integer
	add_node		*file_name* から *next_file* にかけての追加ノード数	Integer
	delete_node		*file_name* から *next_file* にかけての削除ノード数	Integer
	update_node		*file_name* から *next_file* にかけての更新ノード数	Integer
	move_node		*file_name* から *next_file* にかけての移動ノード数	Integer
	node_sum		追加，削除，更新，移動ノード数の合計	Integer

本研究で用いるデータセットについて説明する．データセットは，参加者が問題への回答として提出したソースコードと，言語やタイムスタンプ等の提出履歴情報データベースの 2 種類からなり，提出履歴情報データベースの内容は表 2 に示される通りである．データセットの統計情報は表 3 にて示しており，本データは 2016/5/19~2016/11/15 の期間に収集されている．ソースコードのファイル数は 1,644,636 で，プログラミングコンテストにおける提

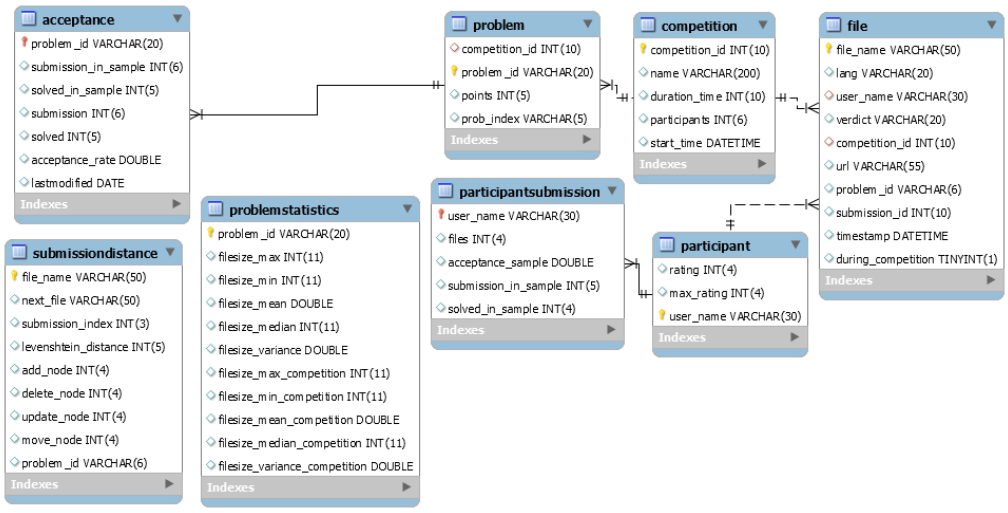

図 3: 提出履歴情報データベースの ER 図

表 3: データセットの統計情報

収集期間	ファイル数	参加者数	コンテスト数	問題数	DB サイズ
2016/5/19~2016/11/15	1,644,636	14,520	739	3,218	357MB

出は 1 つのソースファイルにまとめられるため，これは提出数と一致している．参加者数は，2016/5/19~2016/11/15 の期間に 1 度以上 Codeforces のコンテストに参加したユーザーの総数である．また，各コンテストには複数の問題が含まれるため，コンテストの数と比較して問題数が多くなっている．

以下では各データセットについて説明する．

3.2.1 ソースコードデータ

本研究において収集されたソースコードは，Codeforces のプログラミングコンテスト参加者が，コンテストの問題に対する回答として提出したものである．ソースコードは，Codeforces がコンパイル環境を用意している任意の言語で記述することができる．また，回答ソースコードが正答であるか，コンパイル可能かどうかは提出の可否に影響しないため，文法的に不完全なソースコードが含まれる場合もある．本研究では，言語やコンパイル可能性に依存せず，収集期間内のすべての提出ソースコードをテキストファイルとして収集した．

3.2.2 提出履歴データ

提出履歴情報には，参加者のレーティング情報や，どの参加者がどの問題にいつ提出したか，提出が正解であったか等の情報が含まれている．このため，提出履歴情報を用いて参加者のソースコード修正履歴を追うことができるようになっている．提出履歴情報はCodeforcesのAPIを用いて取得できるようになっており，我々はJSONで返されるデータをパースしてMySQL上に保管した．データベースの構成は表2と図3に示す通りであり，以下では各テーブルの詳細について述べる．

- **Participant** このテーブルには，2016/5/19~2016/11/15の期間中1度以上Codeforcesで開催されたプログラミングコンテストに参加したユーザーの情報を含む．各参加者情報には表2に示す通り3種類の項目が含まれる．Codeforcesは一意のユーザーIDを提供していないため，本データセットにおいてユーザー名をIDとしている．また，レーティングはコンテストごと変化するが，本データに含まれるレーティングは2016/11/15時点のものである．

- **ParticipantSubmission** 各参加者が提出したソースコードのうち，本データセットに含まれるものの数を保管しているテーブルである．

- **File** 2016/5/19~2016/11/15の期間中にCodeforcesに対して提出されたソースコードの情報を収集したテーブルである．このテーブルには，ソースコードデータの総数と同じ1,644,636のデータを含む．各提出履歴に与えられた一意の提出IDと提出対象である問題のIDをもとに対応するソースコードのURLを構築することができ，urlカラムに格納されている．

- **Competition** Codeforces上で開催されたコンテストのうち，Problemset[11]で公開されているコンテストの情報を含む．多くのコンテストは5~6程度の問題を含んでいる．コンテストには月に数回行われる定期的なものや，年に一度開催されるコンテスト等複数の種類があり，コンテストに含まれる問題の難易度も様々である．

- **Problem** このテーブルには3,218問のプログラミングコンテストの問題に関する情報が含まれている．問題IDは，問題が掲載されたコンテストのIDと，コンテスト内で問題を識別するためのアルファベットからなる．例えば問題IDが601Bである場合，IDが601であるコンテストでB問題として出題されたということである．また，問題には得点が設定されており，この得点は多くの場合に問題の難易度が高くなるほど

[11] http://codeforces.com/problemset/

大きくなるようになっている．ただし，問題の得点は問題作成者によって設定された目安である．そのため本研究では，問題の難易度として正答率を用いている．

- **Acceptance** このテーブルには，Problem テーブルに含まれる問題への提出数に関する統計情報を格納している．接尾辞が sample であるカラムは，File テーブルに情報が格納されている提出に限定した値である．そうでないカラムの値は，2016/11/15 までに行われた前提出に対する提出数や正解数を表している．また，acceptance_rate は問題に対する正答率を表しており，本研究ではこの値を問題の難易度として扱うこととしている．

- **ProblemSubmissionStatistics** このテーブルには，Problem テーブルに格納されている問題に対する提出における，提出ソースコードのサイズを格納している．サイズの単位は Byte である．接尾辞が competition であるカラムについては，コンテスト中に提出されたソースコードに限定して統計をとった．

- **SubmissionDistance** このテーブルには，連続した提出である file_name と next_file 間における編集距離に関する情報が格納されている．levenshtein_distance はソースコードにおけるトークンベースのレーベンシュタイン距離 [8] を表している．接尾辞が node であるカラムの値は，抽象構文木に対する編集距離である．抽象構文木上の編集距離は GumtreeDiff [4] を用いて計算した値である．

3.3 基礎統計

本項ではデータセットの基礎統計を示す．図 4 は参加者のレーティングの分布である．Codeforces では参加者の初期レーティングを 1500 としている．そのため，レーティング 1500 付近に参加者が最も多く集まり，それを中心として左右に広がる分布となっている．ここで，本項以降で用いられる用語についての定義を行う．

正答率 プログラミングコンテストにおける問題の公式的な難易度は得点として表されているが，設定は問題作成者の主観に依存する．本研究では，より客観的なデータに近づけるために，2017/11/15 時点の回答状況をもとに 正答率 r = 正答数/提出数 と定めた．回答者のレーティングは考慮しないこととする．

正答率グループ 問題には難易度の分散が存在するため，調査によっては問題を正答率でグループ分けした．本研究では，正答率グループ $a = \lfloor 5r + 1 \rfloor / 5$ とした．これにより，問題を正答率 0.2 刻みの 5 群に分割することができる．

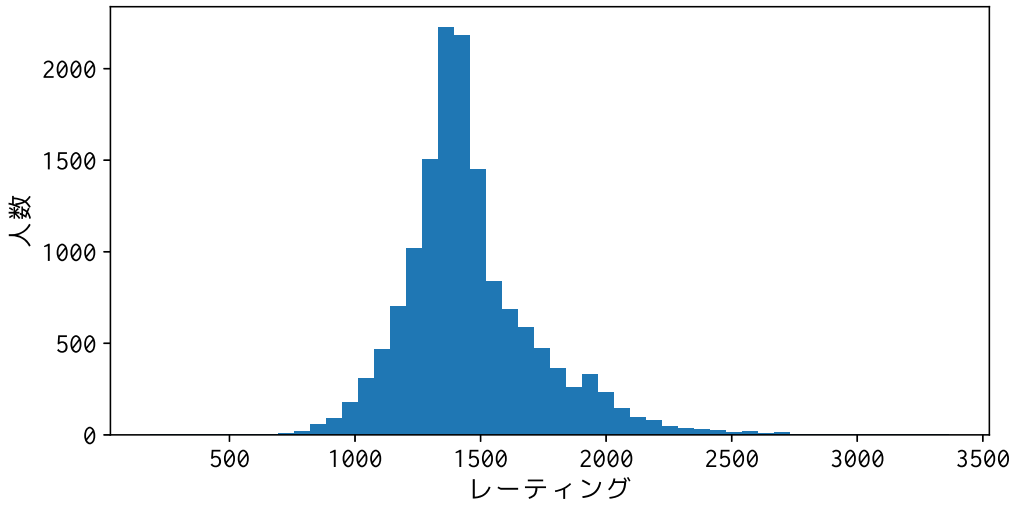

図 4: 参加者のレーティング分布

修正提出 ある参加者が問題に対して誤答を出し，同一の問題に再提出を行うことを指す．一度正答した問題に対する提出は，本研究では修正提出としない．

修正提出回数 ある参加者が問題に対して初めて正答するまでに行った修正提出の回数．一度の回答で正答した場合の修正提出回数は 0 である．最終的に一度も正答していない場合は，提出していない状態と同一視される．

図 5 は，正答率ごとの問題の分布を表している．多くの問題は正答率 0.2 から 0.3 付近に分布しているため，3~5 程度の提出を行って正解する問題が多いと考えられる．図 6 は全回答における，修正提出数の度数分布を表している．0 が最も多く，修正提出回数の増加に従って頻度は単調減少している．多くの問題の正答率は 0.2~0.3 であったことを考えると，多くの提出が正答率の低い問題に集まる傾向があるといえる．図 7 は本データセットのソースコードの，言語別提出数の割合を表している．提出されたソースコードのうち，90%は C++によって記述されており，その次は Java の 4%となっている．また，ソースコード平均サイズは 1.2KB であった．本研究では，提出ソースコードにおいて “GNU C++”，“GNU C++11”，“GNU C++14” のいずれかで記述されたものを対象として実験を行った．対象となったソースコードの数は 1405734 で，本データセットの 85%を占める．

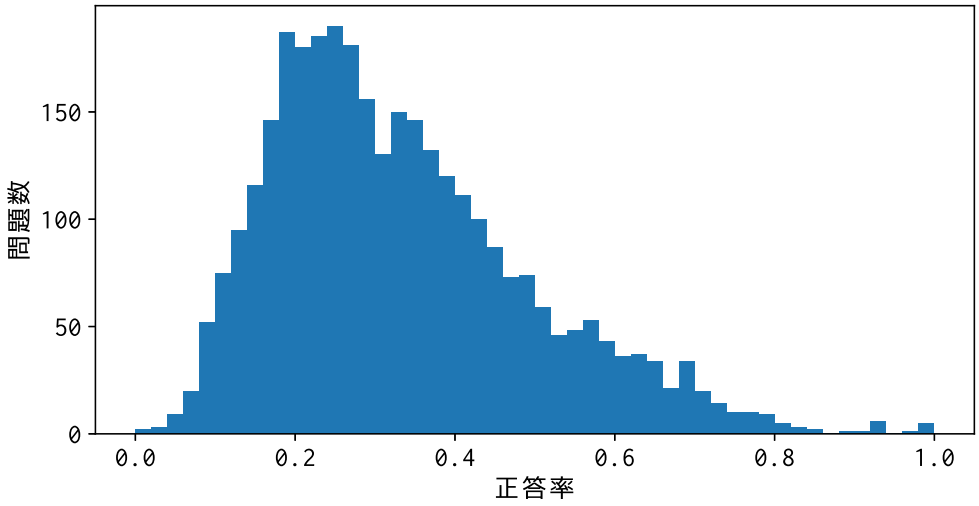

図 5: 正答率ごとの問題の度数分布

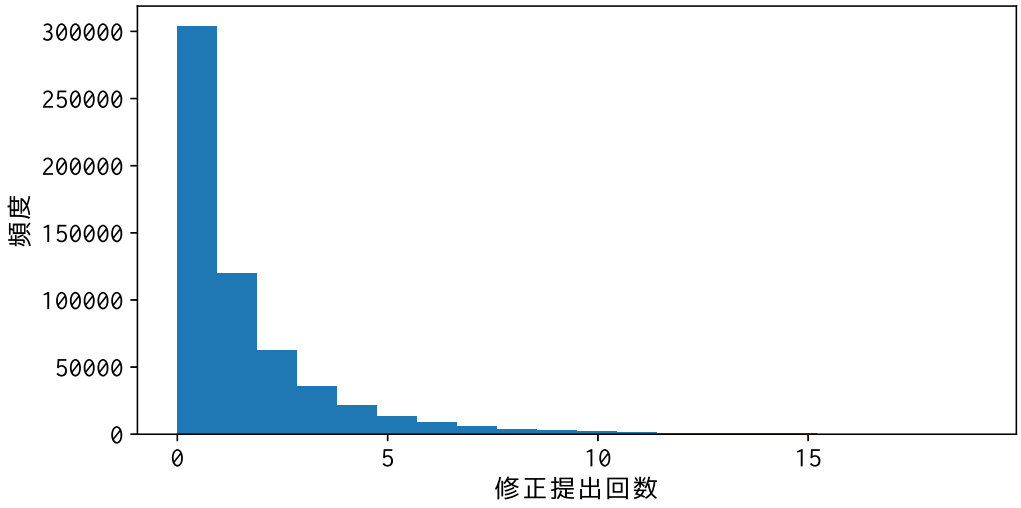

図 6: 修正提出数の分布

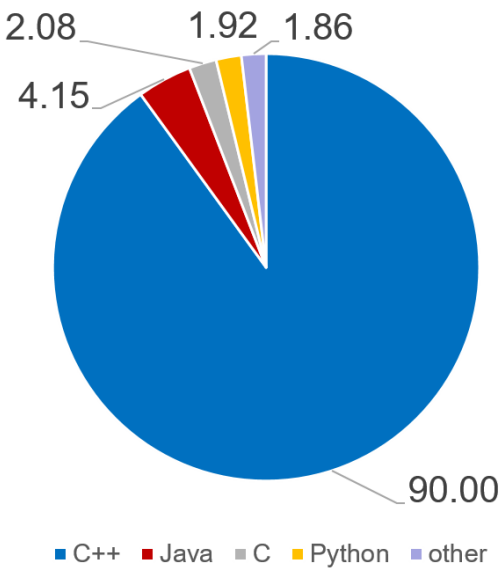

図 7: 提出ソースコードの言語分布

4　提出履歴分析

本研究では，先述のデータセットから参加者の提出を追い，参加者の提出履歴について分析を行った．また分析を行うにあたり，データセットの特徴を調査するために基礎統計を行った．以下ではその詳細について述べる．

4.1　分析の目的

ソースコード編集作業は，開発に際して必然的に行われる作業である．そのため，コンピュータサイエンスにおいてソースコードの編集に関する研究は数多く行われている [5,6]．しかし，開発者の技術力と編集作業の特徴との関連性に関する大規模な調査は現在行われていない．それは，技術力の指標を定義する難しさや，同様の目的に対する開発の編集情報を大規模に収集することが難しいことに起因する．

本研究では，プログラミングコンテストへの回答という同一の目的で記述された大量のソースコードに対する調査および，レーティングという指標との関連性を調査することにより，上記の問題に対して解決を図る．本章では，プログラミングコンテストへの提出ソースコードに対する予約語利用頻度分析，メトリクス調査，修正作業の特徴調査を行った結果を示す．それによってプログラミングコンテスト上級者や初級者において，ソースコード編集にどのような差異があるかを明らかにする．上級者と初級者の編集作業の差異を発見することで，初級者が適切な編集作業を行うための参考にすることができる．また，一般的にどういった箇所に修正が施されるかを知ることで，ツールが修正位置の推薦を行う際に参考にすることができる．

本研究では，プログラミングコンテスト参加者のソースコード編集作業が初級者・上級者間でどのような差異があるかを調査するにあたって，以下の 3 つの Research Question(RQ) を設定した．

RQ1　上級者は，初級者と比較して初回提出ソースコードにどのような違いがあるか

RQ2　上級者は，初級者と比較して修正量が少なくなるか

RQ3　上級者は，初級者と比較して修正箇所に違いはあるか

各 RQ について簡単に概要を説明する．RQ1 は，初級者と上級者のプログラミングコンテストの各問題に対する初回提出ソースコードについて，ソースコード内で使われている予約語やソースコードメトリクスといった特徴を比較する．ソースコードの特徴を明らかにすることで，初級者の改善すべきソースコードの書き方を提示することができる．RQ2 は，初級者と上級者が問題に対して修正提出を行った際に，前回の提出ソースコードとの差分がど

表 4: 上級者・初級者の 2 群におけるレーティングの統計情報

		初級者	上級者
レーティング	平均	1171.12	1824.824
	分散	113.62	226.54
	最小値	-39	1573
	中央値	1202	1764
	最大値	1299	3367
人数		3634	3622

の程度異なるかを比較する．この比較により，初級者と上級者間のソースコード編集能力の違いを明らかにすることができる．例えば上級者の差分が初級者と比較して少ない場合，上級者の提出ソースコードや修正箇所が適切である可能性が高い．RQ3 は，初級者と上級者の修正提出において，ソースコードのどの箇所が修正されているかに注目して比較を行う．ソースコードの修正箇所を明らかにすることにより，初級者が上級者の修正箇所を参考にして，修正能力の向上に繋げることができる．

4.2 本研究における上級者・初級者の定義

ソースコード編集作業について分析を行う上で，プログラミングコンテスト参加者を上級者と初級者の 2 群に分割して比較調査を行う．本研究では，以下のように参加者を分割した．

1. 参加者をレーティングでソート
2. ソートした参加者を人数が等しくなるように 4 分割
3. 4 分割された参加者のうち，最もレーティングが高いグループを上級者，最もレーティングが低いグループを初級者とする

4 分割された参加者のうち，レーティングが中央値付近の 2 群については本研究では考慮しない．このとき，上級者・初級者の分布は表 4 のようになった．

人数が 2 群で異なるのは，同じレーティングを持つ参加者が境界に複数分布するためである．また，初級者のレーティング最小値が-39 であるが，レーティングの増減に制限はないため，レーティングが負になる場合もある．ただし，レーティングが負である参加者は 1 名のみで，その次に小さい値は 185 であった．初級者のレーティング範囲は-39~1299，上級者は 1573~3367 となる．

4.3 初回提出ソースコード 特徴量の分析

本項では，初級者と 上級者の初回提出ソースコード における特徴量の比較調査を 行う．本項を 調査する 目的として，上級者と 初級者間のソースコード の特徴の差異を 明らかにすることで，初級者がその差異を 参考にソースコード の改善に 繋げることができる．RQ1 を 初回提出ソースコード に限定した 理由として，2 度目以降の提出である 修正提出は 平均的に 初回提出ソースコード との差異が小さい 点，また 修正提出との 差異は RQ2，RQ3 にて 調査を 行う 点が 挙げられる．

以下では，

- ソースコード における 予約語の利用頻度
- ソースコード メトリクス

の 2 つの点について，初級者と 上級者との 間でどの程度異なるか 調査した 結果を 示す．

4.3.1 RQ1-a: ソースコード における 予約語利用頻度

RQ1 を 明らかにするにあたって，初回提出ソースコード で用いられている 予約語[12]の利用頻度を 調査した．本研究では，3.3 で述べたように 提出ソースコード のうち C++を 対象に調査を 行う．調査にて 検出対象とする 予約語を 表 5 に示した．

本項では，初級者と 上級者の予約語の利用頻度を 比較する．調査内容の説明のため，以下でいくつかの用語を 定義する．

U_p := 問題 p へ提出を 行った 参加者の集合

$k_{u,p,t}$:= 参加者 u，問題 p における 初回提出ソースコード 内の予約語 t の利用数

$\overline{k_{p,t}}$:= $\frac{1}{|U_p|}\sum_{x\in U_p} k_{x,p,t}$ (問題 p における 全初回提出ソースコード 内の予約語 t の利用数の平均)

$\sigma_{p,t}$:= $\sqrt{\frac{1}{|U_p|}\sum_{x\in U_p}(k_{x,p,t}-\overline{k_{p,t}})^2}$ (問題 p における 全初回提出ソースコード 内の予約語 t の利用数の標準偏差)

$k'_{u,p,t}$:= $\frac{k_{u,p,t}-\overline{k_{p,t}}}{\sigma_{p,t}}$ ($k_{u,p,t}$ を 各問題について平均 0，標準偏差 1 となるよう 標準化したもの)

予約語利用頻度調査を 行うにあたり，問題 p ごとに $k_{u,p,t}$ の標準化を 行った．理由として，プログラミングコンテスト の問題によって用いられるアルゴリズムが異なるため，ソースコード に使われる 予約語の種類や量を 異なる 問題で直接比較することは 危険であると 判断し

[12]http://en.cppreference.com/w/cpp/keyword

たためである．標準化を行うことで，同一問題内での相対的な利用頻度を得ることができるため，回答する問題に偏りが存在する場合でも比較を行うことができる．

上記の用語を用いて，以下で調査の流れについて説明する．

1. 各参加者，各問題の初回提出についてソースコードの字句解析を行い，$k_{u,p,t}$ を求める
2. $k_{u,p,t}$ をもとに $k'_{u,p,t}$ を計算
3. $k'_{u,p,t}$ を予約語 t ごとに初級者を起源とする集合である $K'_{t,low}$ と上級者を起源とする集合である $K'_{t,high}$ に分割
4. "$K'_{t,low}$ と $K'_{t,high}$ の母平均が等しい" という帰無仮説を立てて Welch の t 検定を実施する
5. $K'_{t,low}$ と $K'_{t,high}$ の差を明らかにするため Cohen's d [2] を用いて効果量を測定する

Cohen's d

2 群の平均値の差を表す値として，Cohen's d [2] を利用する．Cohen's d は x_1 と x_2 の 2 群において，各群の標準偏差に対してどの程度平均が異なるかを表す．Cohen's d は以下の式によって定義される．

$$d = \frac{\overline{x_1} - \overline{x_2}}{s}$$

ただし，

$$s := \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}}$$

$$n_i := |x_i|$$

$$s_i^2 := x_i \text{ の分散}$$

表 6 と表 7 は初級者と上級者の予約語利用頻度を比較した結果の抜粋である．"$K'_{t,low}$ と $K'_{t,high}$ の母平均が等しい" という帰無仮説を有意水準 5%で棄却した予約語 t について表に掲載した．表 6 は $\overline{K'_{t,high}} < \overline{K'_{t,low}}$，つまり，上級者の利用頻度が高かった予約語を Cohen's d の降順に並べている．表の上部ほど，初級者と比較して上級者の利用頻度が高いといえる．また表 7 は $\overline{K'_{t,high}} > \overline{K'_{t,low}}$，つまり，初級者の利用頻度が高かった予約語を Cohen's d の昇順に並べている．こちらの表では，表の上部ほど上級者と比較して初級者の利用頻度が高い予約語となる．

表 6 の上位予約語に着目すると，typedef や template，class 等のプログラムの構造化を促進する予約語を多く用いている傾向が確認された．上級者は初級者と比較して，ソースコー

表 5: 調査対象の予約語一覧 [12]

alignas	decltype	namespace	static_cast
alignof	default	new	struct
and	delete	noexcept	switch
and_eq	do	not	template
asm	double	not_eq	this
auto	dynamic_cast	nullptr	thread_local
bitand	else	operator	throw
bitor	enum	override	true
bool	explicit	or	try
break	export	or_eq	typedef
case	extern	private	typeid
catch	false	protected	typename
char	final	public	union
char16_t	float	register	unsigned
char32_t	for	reinterpret_cast	using
class	friend	requires	virtual
compl	goto	return	void
concept	if	short	volatile
const	inline	signed	wchar_t
constexpr	int	sizeof	while
const_cast	long	static	xor
continue	mutable	static_assert	xor_eq

ドの抽象度が高いのではないかと考えられる．また，表 7 の上位予約語に着目すると，else, break，if 等の分岐に関する 予約語を 多く 用いる傾向にあることが確認された．この表から，初級者は分岐文を 多用し，ソースコード が複雑化する 傾向があるのではないかと 考えられる．

4.3.2 RQ1-b: ソースコードにおけるメトリクス分析

本項では，ソースコード のメトリクス分析を 用いて RQ1 へのアプローチを 行う．ソースコード におけるメトリクスには様々なものがあるが，本研究では，Web 上に 公開されてい

表 6: 上級者の利用頻度が高い予約語

t	$\overline{K'_{t,low}}$	$\overline{K'_{t,high}}$	初級者分散	上級者分散	p 値	Cohen's d
typedef	-0.111	0.118	0.896	1.082	0.000e+00	0.228
template	-0.071	0.070	0.778	1.162	0.000e+00	0.140
return	-0.083	0.047	0.881	1.042	0.000e+00	0.131
typename	-0.063	0.066	0.602	1.225	4.822e-291	0.130
while	-0.049	0.029	0.863	1.088	4.466e-115	0.078
operator	-0.033	0.045	0.793	1.156	1.042e-107	0.077
class	-0.037	0.038	0.883	1.100	1.740e-101	0.074
using	-0.038	0.030	0.720	1.270	7.801e-75	0.066
continue	-0.029	0.034	0.914	1.043	8.446e-76	0.063
struct	-0.036	0.017	0.800	1.061	7.348e-59	0.055
do	-0.029	0.018	0.564	0.925	5.805e-65	0.053
public	-0.018	0.012	0.633	1.016	2.597e-22	0.032
namespace	-0.025	0.002	0.885	1.106	4.304e-15	0.027
enum	-0.008	0.012	0.179	1.012	8.389e-14	0.025
asm	-0.005	0.008	0.049	0.772	3.056e-09	0.017
private	-0.009	0.003	0.390	0.814	7.058e-08	0.016
typeid	-0.003	0.004	0.010	0.489	1.143e-08	0.014
friend	-0.007	0.003	0.650	0.844	8.956e-05	0.012
decltype	-0.003	0.004	0.180	0.657	3.321e-04	0.009
try	-0.001	0.002	0.011	0.410	1.004e-02	0.006
catch	-0.001	0.002	0.011	0.408	1.050e-02	0.006

るソースコードメトリクス計測ツールである SourceMonitor[13]を用いて計測できる 11 種類のメトリクスについて調査した．調査対象となったメトリクスの一覧と説明は表 8 に掲載した．また，メトリクスの一部で用いた循環的複雑度 [9] について以下で補足する．

[13]http://www.campwoodsw.com/sourcemonitor.html

表 7: 初級者の利用頻度が高い予約語

t	$\overline{K'_{t,low}}$	$\overline{K'_{t,high}}$	初級者分散	上級者分散	p 値	Cohen's d
else	8.760e-02	-9.499e-02	1.061e+00	0.915	0.000e+00	-0.185
break	5.797e-02	-4.005e-02	1.074e+00	0.942	1.018e-183	-0.098
if	2.283e-02	-3.016e-02	9.862e-01	1.003	8.989e-55	-0.053
for	3.069e-02	-1.241e-02	1.005e+00	1.023	1.707e-35	-0.043
goto	9.375e-03	-4.003e-03	8.934e-01	0.657	1.993e-07	-0.015
case	-3.696e-04	-6.822e-03	6.454e-01	0.436	3.069e-04	-0.009
switch	-8.421e-04	-6.476e-03	6.406e-01	0.463	2.068e-03	-0.008
extern	-3.413e-05	-1.058e-04	8.354e-04	0.001	1.259e-61	-0.002

循環的複雑度

McCabe によって提唱された，ソフトウェアの複雑度を示す指標の一種である．循環的複雑度は以下の式によって定義される．

$$M = E - N + 2P$$

ただし，

M := 循環的複雑度

E := 制御フローグラフ (Control Flow Graph: CFG) における辺の数

N := CFG における頂点数

P := CFG におけるコンポーネント (結合しているグラフ) の数

循環的複雑度はソースコード中の分岐の数や組み合わせに依存して増加するため，循環的複雑度が高いほどプログラムが複雑であるといえる．

SourceMonitor で計測した結果を上級者と初級者間で比較する上で，4.3.1 と同様に用語の定義を行う．標準化の計算方法は 4.3.1 と同様であるため省略する．

U_p := 問題 p へ提出を行った参加者の集合

$m_{u,p,c}$:= 参加者 u，問題 p における初回提出ソースコードのメトリクス c の計測値

$m'_{u,p,c}$:= $m_{u,p,c}$ を各問題について平均 0，標準偏差 1 となるよう標準化したもの

$M'_{c,low}$:= $m'_{u,p,c}$ のうち，u が初級者に属する集合

$M'_{c,high}$:= $m'_{u,p,c}$ のうち，u が上級者に属する集合

調査の流れは 4.3.1 と同様であるため省略する．各 c ごとに，$M'_{c,low}$，$M'_{c,high}$ の 2 群について t 検定の実施と Cohen's d の算出を行った．結果を表 9 と表 10 に示す．また，図 8 と図 9 は各メトリクス c について，$M'_{c,low}$ と $M'_{c,high}$ の分布を箱ひげ図にプロットしたものである (外れ値は除外)．調査の結果，いずれのメトリクスについても t 検定において有意水準 5%で有意差が認められた．

表 9 には初級者と比較して上級者の計測値が大きかったメトリクスを Cohen's d の降順に掲載した．表の上部ほど初級者との差が大きかったメトリクスで，上級者はソースコードの行数が増える傾向が伺える．関数の数やクラス数も初級者と比較して増加する傾向にあり，処理を関数に分離することで行数の増加につながっている可能性がある．また，プログラミングコンテストにおいては予め汎用的な処理をテンプレートとしてソースコード上に準備しておく場合があり，上級者ほどその傾向が高いということが考えられる．表 10 には上級者と比較して初級者の計測値が大きかったメトリクスを Cohen's d の昇順に掲載した．メトリクスの傾向として，ネストの深さや複雑度の高さが確認された．これは 4.3.1 における分岐文利用頻度の高さと一致する．

4.4 ソースコード修正情報の分析

本項では，プログラミングコンテストの提出履歴情報とソースコードをもとに，参加者のソースコード修正について分析を行った結果を示す．3.3 の通り言語が C++であるソースコードを対象に分析を行った．以下では，4.4.1 と 4.4.2 にて RQ2 についての調査であるソースコード修正量の分析を行い，4.4.3 にて RQ3 のソースコード修正箇所についての分析を行う．

4.4.1 RQ2-a: ソースコード修正提出回数の分布

本項では，ソースコード修正量を提出回数の観点から調査を行った結果を示す．3.3 で説明した通り，修正提出とは正答までに要した再提出の回数である．RQ2 で議論するソースコード修正量とは，修正提出回数と一提出当たりの修正量の積である．つまり，修正提出回数が大きいということは修正量が大きいことに繋がるため，本項についての調査を行った．説明にあたって用語の定義を行う．標準化の計算方法は 4.3.1 と同様であるため省略する．

U_p := 問題 p へ提出を行った参加者の集合

$s_{u,p}$:= 参加者 u，問題 p における修正提出回数

表 8: 調査対象メトリクス

メトリクス	説明
avg_complexity	各関数の循環的複雑度の平均値
max_complexity	各関数の循環的複雑度の最大値
avg_depth	各関数のネスト 深さの平均値
max_depth	各関数のネスト 深さの最大値
methods_per_class	クラス当たり のメソッド 数
n_classes	クラス数
n_func	関数の数
n_lines	物理行数
n_statements	セミコロンで区切られた論理行数
percent_branch_statements	全体の論理行数に占める 分岐文 (if, else, for, while, goto, break, continue, switch, case, default, return を 用いた文) の割合
percent_comments	全体の物理行数に 占めるコメント の割合

表 9: 上級者において値が大きいメトリクス

c	$\overline{M'_{c,low}}$	$\overline{M'_{c,high}}$	初級者分散	上級者分散	p 値	Cohen's d
n_statements	-0.139	0.108	0.894	1.078	0.000e+00	0.245
n_func	-0.105	0.089	0.843	1.118	0.000e+00	0.191
n_lines	-0.065	0.043	0.931	1.022	3.037e-220	0.109
methods_per _class	-0.045	0.035	0.567	1.134	1.660e-165	0.084
n_classes	-0.044	0.023	0.768	1.086	8.370e-98	0.069

$s'_{u,p}$:= $s_{u,p}$ を 各問題について平均 0，標準偏差 1 となるよう 標準化したもの

$S'_{a,low}$:= $s'_{u,p}$ のうち，u が初級者に属し，p が正答率グループ a に属する 集合

$S'_{a,high}$:= $s'_{u,p}$ のうち，u が上級者に属し，p が正答率グループ a に属する 集合

修正量に関しては特に問題の正答率による影響が大きいと 考えられるため，問題を 正答率グ

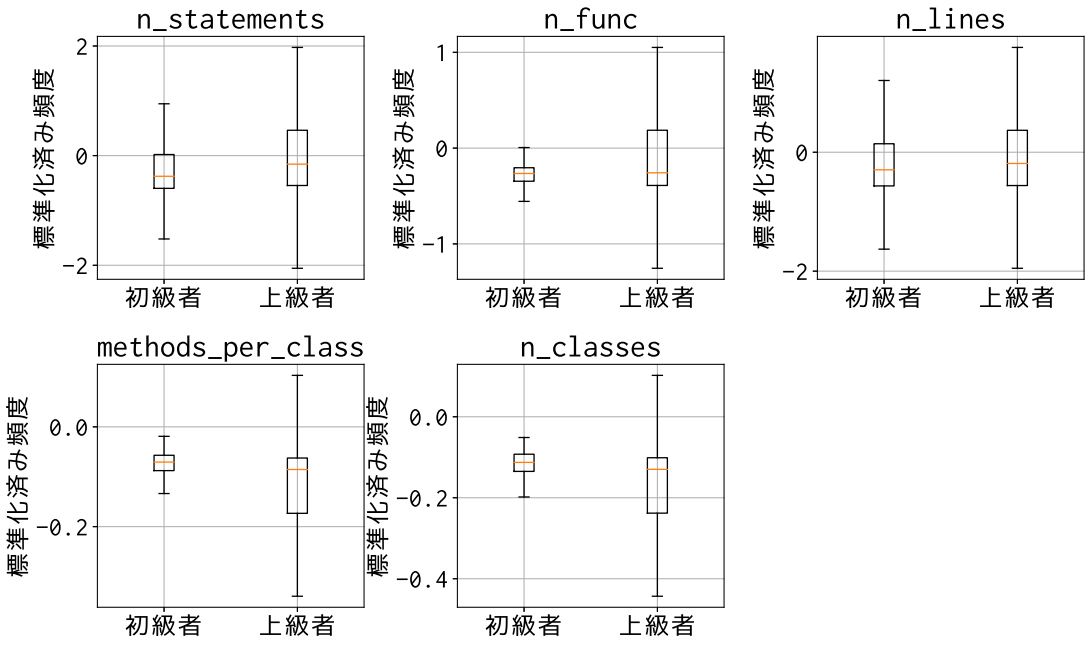

図 8: SourceMonitor 計測メトリクスの 2 群間比較の箱ひげ図 ($M'_{c,low} < M'_{c,high}$)

表 10: 初級者において値が大きいメトリクス

c	$\overline{M'_{c,low}}$	$\overline{M'_{c,high}}$	初級者分散	上級者分散	p 値	Cohen's d
avg_depth	0.246	-0.202	1.034	0.944	0.000e+00	-0.457
percent_branch	0.195	-0.154	0.993	0.995	0.000e+00	-0.351
_statements						
avg_complexity	0.162	-0.137	1.073	0.926	0.000e+00	-0.303
max_complexity	0.113	-0.100	1.060	0.953	0.000e+00	-0.214
max_depth	0.096	-0.081	1.028	0.971	0.000e+00	-0.179
percent_comments	0.022	-0.035	1.031	0.916	8.580e-61	-0.059

ループ (3.3 参照) で分割して比較を行った．調査の流れは 4.3.1 と同様であるため省略する．各 a ごとに，$S'_{a,low}$，$S'_{a,high}$ の 2 群について t 検定の実施と Cohen's d の算出を行った．結果を表 11 に示す．t 検定の結果，すべての正答率グループで有意差があることが示された．

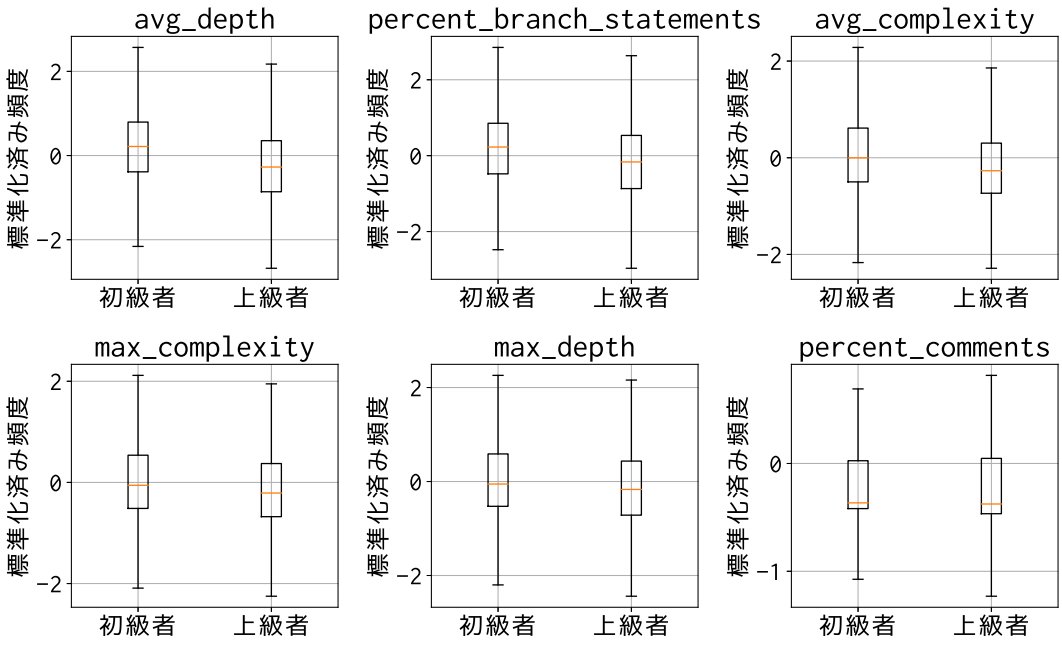

図 9: SourceMonitor 計測メトリクスの 2 群間比較の箱ひげ図 ($M'_{c,low} > M'_{c,high}$)

また，Cohen's d の算出により，上級者ほど修正提出回数が減少する傾向が確認された．図 10 は，$S'_{0.4,low}$ と $S'_{0.4,high}$ の分布を箱ひげ図にプロットしたものである．実際に上級者の修正提出回数が減少していることが確認できた．

4.4.2 RQ2-b: 修正提出あたりの修正量の分布

我々は，初級者と上級者間におけるソースコード修正量がどの程度異なるかについて調査を行った．説明にあたって，以下の用語を定義しておく．標準化の方法は 4.3.1 と同様であるため省略する．

LD ソースコード間におけるトークンベースのレーベンシュタイン距離 [8] を表す．特に本研究においては，ある参加者が同一の問題に対して行った提出のうち，連続した提出のレーベンシュタイン距離を表す．

$l_{u,p,i}$:= 参加者 u の問題 p に対する提出のうち，i 番目と $i+1$ 番目の提出間の LD

$l'_{u,p,i}$:= $l_{u,p,i}$ を各問題について平均 0，標準偏差 1 となるよう標準化したもの

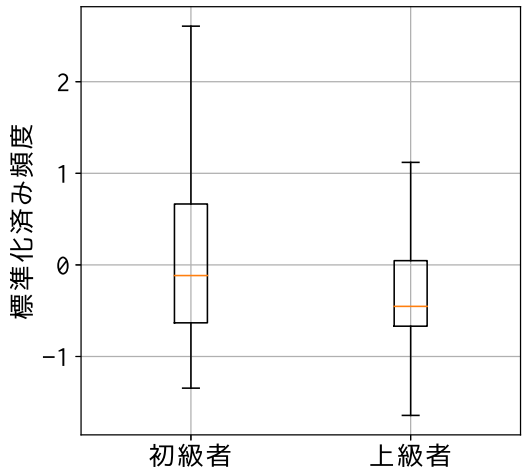

図 10: 正答率グループ 0.4 における初級者・上級者の修正提出回数の分布

$$l'_{max,u,p} := max_{j=0}^{j<s_{u,p}} l'_{u,p,j} \text{ (参加者 } u\text{，問題 } p \text{ における } l_{u,p,i} \text{ の最大値)}$$

$$l'_{med,u,p} := median_{j=0}^{j<s_{u,p}} l'_{u,p,j} \text{ (参加者 } u\text{，問題 } p \text{ における } l_{u,p,i} \text{ の中央値)}$$

$$L'_{max,a,low} := l'_{max,u,p} \text{ のうち，} u \text{ が初級者に属し，} p \text{ が正答率グループ } a \text{ に属する集合}$$

$$L'_{med,a,low} := l'_{med,u,p} \text{ のうち，} u \text{ が初級者に属し，} p \text{ が正答率グループ } a \text{ に属する集合}$$

$$L'_{max,a,high} := l'_{max,u,p} \text{ のうち，} u \text{ が上級者に属し，} p \text{ が正答率グループ } a \text{ に属する集合}$$

$$L'_{med,a,high} := l'_{med,u,p} \text{ のうち，} u \text{ が上級者に属し，} p \text{ が正答率グループ } a \text{ に属する集合}$$

調査手順としては 4.3.1 と同様に，$L'_{func,a,low}$ と $L'_{func,a,high}(func \in \{max, med\})$ に対して，“両集合の母平均に差はない” という帰無仮説を立てて t 検定を実施し，Cohen's d の算出を行った．表 12 と表 13 は初級者・上級者間における標準化 LD の最大値/中央値の比較結果である．表の最も左の列は正答率グループを表す．各行は，正答率グループにおける初級者・上級者の標準化 LD 統計結果を表している．正答率は問題の複雑度を表すため，正答率が LD に影響を与える可能性を考慮し，正答率グループごとに比較を行った．図 11 は正答率グループ 0.4 の問題に対する，上級者・初級者間における標準化 LD 最大値/中央値の箱ひげ図である．

表 12 と表 13 の p 値を確認すると，正答率グループ 1.0 以外の行はいずれも有意水準 5%で有意差があると言えることが確認された．この結果から，上級者は初級者と比較して，有意

表 11: 初級者・上級者間の修正提出回数比較

a		サイズ	平均値	分散	p 値	Cohen's d
0.2	初級者	6892	0.132	1.126	1.904e-44	-0.212
	上級者	28528	-0.074	0.932		
0.4	初級者	48515	0.222	1.186	0.000e+00	-0.400
	上級者	105007	-0.162	0.832		
0.6	初級者	49788	0.216	1.213	0.000e+00	-0.421
	上級者	49741	-0.204	0.717		
0.8	初級者	26983	0.148	1.220	8.359e-297	-0.319
	上級者	22126	-0.173	0.658		
1.0	初級者	1070	0.098	1.136	5.943e-05	-0.168
	上級者	1800	-0.063	0.831		

に一度のソースコード修正量が少ないということがいえる．また図 11 は，表 12 と 13 の 2 行目の分布を図示しており，実際に上級者の LD が小さいことが確認できる．表 12 と表 13 において，正答率グループ 1.0 の行のみ有意差があるとは言えないという結果になった．この行に関しては，正答率が高く簡単な問題であるため，あまり差が出なかったのではないかと考えられる．また，表 12 や表 13，図 5 に見られるようにこの行に含まれる問題自体が少ないため，十分な結果が得られていない可能性がある．

4.4.3 RQ3: ソースコード修正箇所の分布調査

4.4.2 節ではレーティングごとにソースコード修正量がどの程度であるかを調査した．本項では，修正がソースコードにおけるどの位置に施されるかを調査する．ソースコードの修正を調査する上では，GumTreeDiff [4] という抽象構文木ベースのソースコード差分取得ツールを用いた．GumTreeDiff を用いた理由として，抽象構文木ベースなので実際のソースコード修正に近い差分が得られる，他のツールと比較して差分サイズが小さい，高速に動作するということが挙げられる．

図 12 は，初級者と上級者が正答率グループ 0.4 の問題に対して提出したソースコードにおいて，変更が行われた箇所の度数分布の上位 10 件を表している．表の縦軸のラベルは変更が行われたソースコードの抽象構文木における頂点を表し，横軸に変更回数の合計を表す．グラフの色は以下の通り変更の種類を表す．

青 ノードの追加

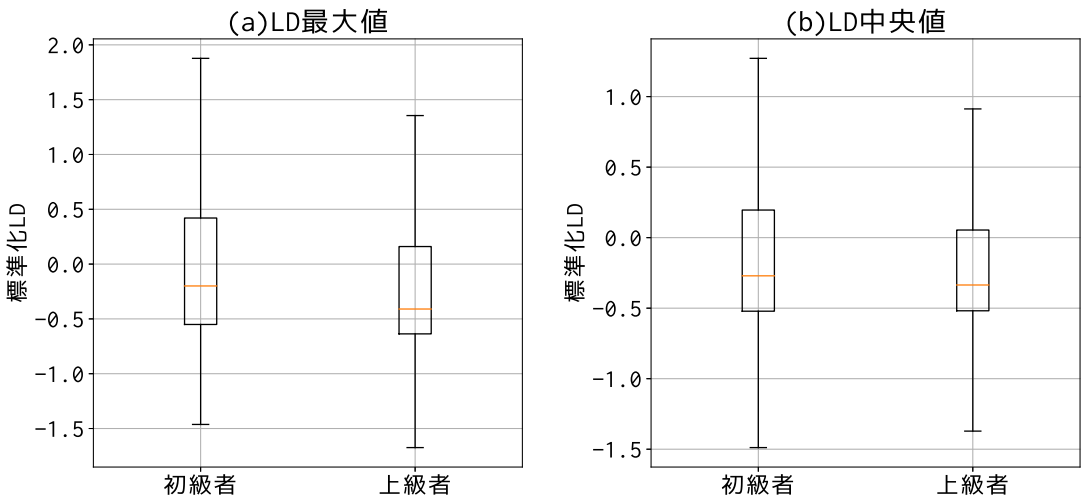

図 11: $L'_{max,0.4,low}$ と $L'_{max,0.4,high}$(a)，$L'_{med,0.4,low}$ と $L'_{med,0.4,high}$(b) の分布比較

橙 ノードの削除

緑 ノードの置換

赤 ノードの移動

修正の多くは不足箇所の追加であり，移動や置換は *expr* 以外ではほとんど行われていなかった．また，上級者と初級者とでは，上級者ほど *name* に対する変更が多く *block* に対する変更が少なくなる傾向が見られた．これは，上級者ほど変更の粒度が小さいのではないかと考えられる．

4.5 考察

4.3.1 節と 4.3.2 節では，問題への初回提出ソースコードの特徴がどのようであるかを調査した．RQ1 に対する回答として，上級者のソースコードは初級者と比較して複雑度が小さくなる傾向にあることがわかった．4.3.1 節を考慮すると，初級者は分岐に関する予約語を多用するため，また上級者はプログラムの構造化を行う予約語を多用するためであるといえる．初級者はこれらの傾向を考慮してコードを書くことで，上級者に近づくことができる．

4.4.1 節と 4.4.2 節では，プログラミングコンテスト参加者の修正量がどの程度であるかを調査した．RQ2 に対する回答として，効果量は少ないものの，有意に上級者は初級者と比較して修正量が少ないことがわかった．理由として，上級者は初級者よりも修正箇所の特定が適切であること，上級者のソースコード記述方法は修正の必要性が少ないこと等が考えられる．前者の場合は，上級者による修正箇所の特定方法を学習者が理解することにより，開

表 12: $L'_{max,0.4,low}$ と $L'_{max,0.4,high}$ の比較結果

a		サイズ	平均値	分散	p 値	Cohen's d
0.2	初級者	5329	0.086	1.039	5.718e-17	-0.133
	上級者	21235	-0.046	0.966		
0.4	初級者	32378	0.073	1.007	2.023e-93	-0.144
	上級者	54324	-0.071	0.978		
0.6	初級者	22442	0.070	0.974	6.436e-42	-0.152
	上級者	13713	-0.082	1.071		
0.8	初級者	6336	0.037	0.957	1.442e-08	-0.132
	上級者	2997	-0.094	1.085		
1.0	初級者	135	-0.094	0.890	9.531e-02	0.165
	上級者	289	0.075	1.120		

発技術の向上につながる可能性がある．このため，4.4.3 節における修正箇所の調査に加えて，どのような場合にどういった修正が行われるかを調査することで，プログラミング教育に生かせる有用な結果が得られるのではないかと考えられる．

4.4.3 節では，参加者の修正が抽象構文木上でどのノードに位置するかを調査した．RQ3 に対する回答として，

1. 上級者，初級者ともに式 (*expr*) への修正が多くを占める
2. 修正種別としては要素の追加が半数近く最も多い
3. 上級者になるにつれて *name* への修正が多くなり *block* への修正が少なくなる

という傾向が見られた．上記の (1) については，アルゴリズムの記述を必要とするプログラミングコンテストの特性による傾向だと考えられる．(2) については，考慮漏れテストケースに対応するために，修正者が新たな機能を追加する場合が多いのではないかと考えられる．また，(3) については，初級者と比較して上級者は修正位置の特定が適切となるため，修正の粒度が細かくなるのではないかということが考えられる．今後は調査結果と実際のソースコードを目視で確認し，どのような場合が上記結果の理由となっているかを確認する必要がある．

表 13: $L'_{med,0.4,low}$ と $L'_{med,0.4,high}$ の比較結果

a		サイズ	平均値	分散	p 値	Cohen's d
0.2	初級者	5329	0.046	1.010	9.647e-06	-0.069
	上級者	21235	-0.023	0.982		
0.4	初級者	32378	0.030	0.971	1.504e-18	-0.061
	上級者	54324	-0.031	1.019		
0.6	初級者	22442	0.037	0.933	4.009e-12	-0.079
	上級者	13713	-0.042	1.128		
0.8	初級者	6336	0.013	0.926	1.104e-02	-0.061
	上級者	2997	-0.048	1.162		
1.0	初級者	135	-0.092	0.935	9.723e-02	0.168
	上級者	289	0.082	1.145		

4.6 妥当性への脅威

レーティング 本研究では，参加者の熟練度にレーティングを指標として用いている．プログラミングコンテストにおけるレーティングの高さは，アルゴリズムを適切に記述する能力が高いことを示すが，プログラミング能力全般に通用する指標ではない．このため，参加者の熟練度を表す他の指標も試すことで，適切な熟練度の表現を求めることが必要である．

正答率 問題の難易度として，提出に対する正答の割合である正答率を利用している．しかし問題の難易度が同じでも，解法に用いるアルゴリズムや，提出する参加者のレーティングの偏りによっては正答率が異なる可能性がある．そのため，問題に対する解法や，提出している参加者のレーティングを考慮し，より適切な問題分類方法を探る必要がある．

他のコンテスト 本研究において，プログラミングコンテストサイト Codeforces への提出のみをデータセットとして用いた．しかし，一つのコンテストサイトを用いるのみでは問題の性質や参加者に偏りが生じてしまう可能性がある．この問題に対しては，今後他のコンテストサイトもデータセットとして利用することで解決することができる．

差分検出アルゴリズム 4.4.2 節ではレーベンシュタイン距離を，4.4.3 節では GumTreeDiff によるソースコード差分を差分検出アルゴリズムとして利用した．差分検出方法によっ

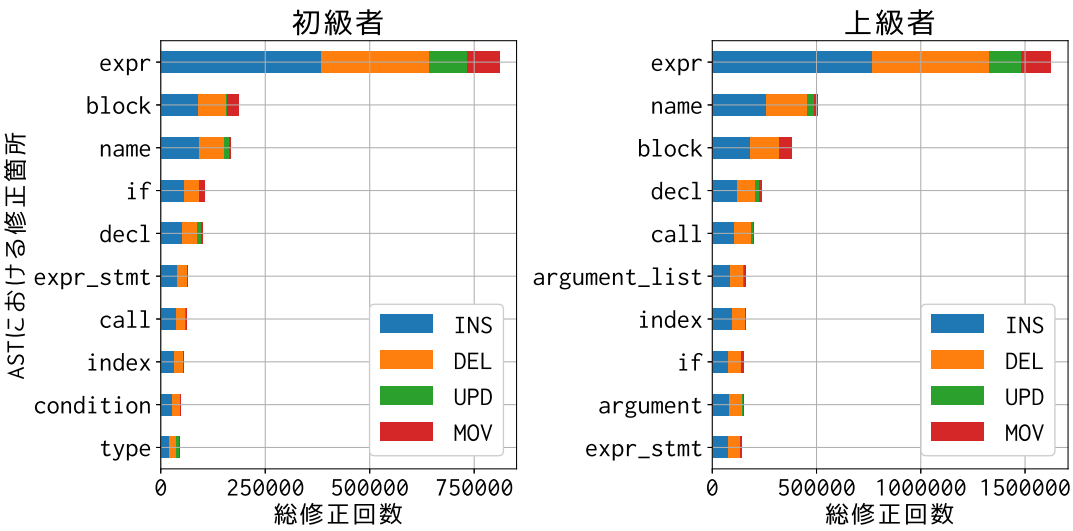

図 12: 正答率グループ 0.4 の問題における初級者・上級者の修正箇所度数分布の上位 10 件

てはソースコード差分の結果が異なる場合があり，より実際の編集に近い差分を取得できる必要がある．そのため，他の差分検出手法を試しつつ，目視でソースコード差分を確認することで，適切な編集作業の特徴抽出を行えると考えられる．

5 まとめ

本研究では，プログラミングコンテストの提出ソースコードを対象として，編集作業の特徴調査を行った．世界最大規模のコンテストサイト Codeforces からデータを収集することによって調査規模を拡大することができた．また，参加者のレーティングを考慮することにより，上級者と初級者との間にある編集作業の差異を見つけることができた．

今後の研究課題として，さらに詳細な修正箇所の調査が挙げられる．本研究では修正された抽象構文木上のノード名のみを抽出していたが，修正されたノードの構文木における親をたどることで，修正がソースコード上のどのブロックで行われたかを知ることができる．修正位置に関する具体的な情報を得ることで，学習者への修正に関する教育や，ツールによる修正推薦等に役立てることができる．本研究では，レーティングが時系列情報であることを利用しなかった．レーティングが時系列とともに変化することを利用して，ソースコードの特徴とレーティングの変化との関係を調査することができる．また，本研究では修正/編集作業の調査のみを行ったが，本研究に用いたデータセットを利用することで，主にアルゴリズムに関するソースコードの書き方とレーティングとの関係を調査し，学習者へのプログラミング指導に役立てることについても検討していきたい．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上克郎 教授には研究について適切な御指導及び御助言を賜りました．本論文を井上教授のもとで完成させることができ，心より深く感謝しております．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下誠 准教授には研究について様々なご意見を賜りました．いただいた御意見を参考に，論文を適切に修正することができました．松下准教授に深く感謝しております．

名古屋大学大学院情報科学研究科附属組込みシステム研究センター 吉田則裕 准教授には，本研究の方針から論文の構成に至るまで直接の御指導及び御助言をして頂きました．本論文の完成は吉田准教授のおかげであると，深く感謝しております．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学研究室 崔恩瀞 助教には，研究及び研究生活について様々な御助言を賜りました．崔助教の御助言により，滞りなく研究を行うことができ，深く感謝しております．

最後に，大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様には，本論文の執筆にあたって様々な場面で支えて頂きました．皆様のおかげで本論文を完成させることができました．心より深く感謝しております．

参考文献

[1] E. T. Chen. Program complexity and programmer productivity. *IEEE Trans. Softw. Eng.*, Vol. SE-4, No. 3, pp. 187–194, 1978.

[2] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences, 2nd edition.* Lawrence Erlbaum Associates, 1988.

[3] Arpad E Elo. *The rating of chessplayers, past and present.* Arco Pub., 1978.

[4] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proc. of ASE 2014*, pp. 313–324, 2014.

[5] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Proc. of ICSE 2012*, pp. 3–13, 2012.

[6] Quinn Hanam, Fernando S. de M. Brito, and Ali Mesbah. Discovering bug patterns in javascript. *ACM SIGCSE Bulletin*, pp. 144–156, 2016.

[7] T. C. Jones. Measuring programming quality and productivity. *IBM Systems Journal*, Vol. 17, No. 1, pp. 39–63, 1978.

[8] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10*, pp. 707–710, 1966.

[9] Thomas J McCabe. A complexity measure. *IEEE Trans. Softw. Eng.*, Vol. 2, No. 4, pp. 308–320, 1976.

[10] Mikhail Mirzayanov. Codeforces rating system. `http://codeforces.com/blog/entry/102`, 2010.

[11] Mikhail Mirzayanov. Codeforces: Statistics 2015. `http://codeforces.com/blog/entry/22618`, 2016.

[12] Jordi Petit, Omer Giménez, and Salvador Roura. Jutge.org: An educational programming judge. In *Proc. of SIGCSE*, pp. 445–450, 2012.

[13] Miguel A Revilla, Shahriar Manzoor, and Rujia Liu. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, Vol. 2, pp. 131–148, 2008.

[14] Hao Zhong and Zhendong Su. An empirical study on real bug fixes. In *Proc. of ICSE 2015*, pp. 913–923, 2015.