Vol.2017-SE-197 No.6
2017/11/9

プログラミングコンテスト参加者を対象とした編集作業の特徴調査

堤 祥吾[1]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：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．本研究では，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 の提出履歴データをもとに構築したデータセットを用いて，参加者のソースコード修正箇所や修正回数を調査した．また，コンテストにおけるイロレーティングと修正情報との間にどのような相関があるかを分析した．結果，イロレーティングが上がるほど修正量が減少するという相関があることを確認した．

キーワード：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

Investigating the Modification Features of Programming Competition Archives

Shogo Tsutsumi[1]　Norihiro Yoshida[2]　Eunjong Choi[3]　Katsuro Inoue[1]

1. まえがき

ソースコードの編集/修正作業は，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においてソースコードの修正に関する研究は数多く行われている [3], [4], [10]．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すが，ソースコード修正にかかる労力は大きい．特にソフトウェア開発の初級者は上級者と比較して，ソースコードの修正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/修正作業の差異を調査し，初級者が上級者の編集/修正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/修正作業方法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 [7]．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 [8], [9] 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/修正作業について調査を行う．特にソースコードの修正箇所や修

[1] 大阪大学
Osaka Uniersity
[2] 名古屋大学
Nagoya University
[3] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

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

正回数と参加者のレーティングとの相関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/修正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．分析では，上級者は初級者と比較して編集回数が少ないことがわかった．

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．

2. プログラミングコンテスト

この章では，本研究のデータセット構築のために参照するプログラミングコンテストについて述べる．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，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．

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システムである．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[*3]，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言語であれば自由に提出することができる．

2.2 プログラミングコンテストの概要

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 [1], [6] が変動する．

[*3] http://judge.u-aizu.ac.jp/onlinejudge/

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	△[*4]	○
CodeChef[*5]	12129[*6]	○	×

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時間に応じて参加者に得点が加算される．参加者は何度でも回答を提出することができ，参加者の提出履歴は公開されているため，参加者がソースコードをどのように修正したかを確認できるようになっている．

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．

2.3 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 [6] 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 [1] 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．

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，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．

3.1 データセットの内容

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からなり，提出履歴情報データベースの詳細は表 2 に示される通りである．

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．

3.2 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 1 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．これらの情報を Codeforces が提供する API[*7]から取得することができる．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，公開 HTML ページから直接取得した．

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，参加者の修正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

ソースコードの修正は，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修正に関する研究は数多く行われている [3], [4]．しかし，開発者の技術力と修正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修正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．本研究では，プログラミングコンテスト上級者や初級者と，ソースコード修正方法との間にどのような相関があるかを明らかにする．

上級者と初級者の修正方法の差異を発見することで，初級者が適切な修正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．

4.2 基礎統計

ソースコードの修正情報に関する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の基礎統計を行った．図 2 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．それによって，レーティング 1500 付近に参加者が最も多く集まり，全体としては正規分布に近い分布となっている．図 3 は，正答率ごとの問題の分布を表している．本研究では，正答率 r = 正答数/提出数 と定めて，正答率を問題の難易度として考えている．多くの問題は正答率 0.2 から 0.3 付近に分布している．

図 4 は全回答における，修正提出数の度数分布を表している．ここで修正提出数とは，ある参加者がある問題に正答するまでに何回再提出を行ったか，という値である．例えば修正提出数が 0 とは一度の回答で問題に正答したことを表し，修正提出数が 3 とは 3 回誤った提出を行った後，4 回目の提出で正答したことを表す．ここで，最終的に正答しなかった回答は考慮していない．図 5 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．

4.3 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．図 5 の通り提出ソースコードの 90%が C++で記述されていたため，言語が C++であるソースコードを対象に分析を行った．

分析結果の説明にあたって，以下の用語を定義しておく．

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 [5] 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．

正規化 LD(NLD) $LD_{u,p,i}$ を，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 とする．また，$\overline{LD_p}$ を，問題 p における LD の平均値とする．このとき，$NLD_{u,p,i} = LD_{u,p,i}/\overline{LD_p}$ とする．

LD を正規化する理由として，問題によって LD が異なってしまう可能性が挙げられる．相対的に他の参加者よりも

[*7] http://codeforces.com/api/help

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

#	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 1　Codeforces の提出履歴ページのスクリーンショット

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

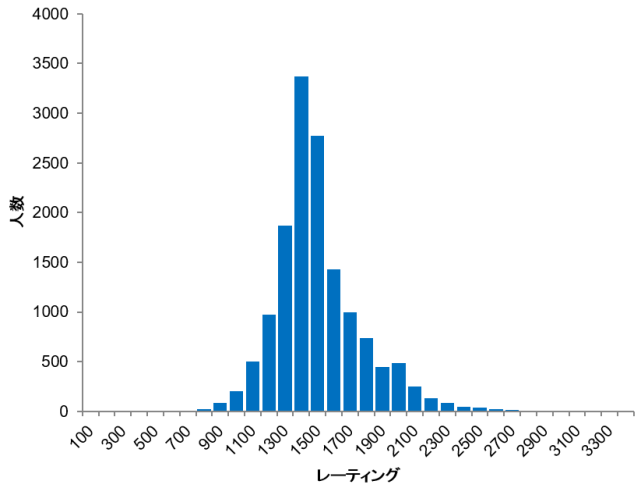

図 2　参加者のレーティング分布

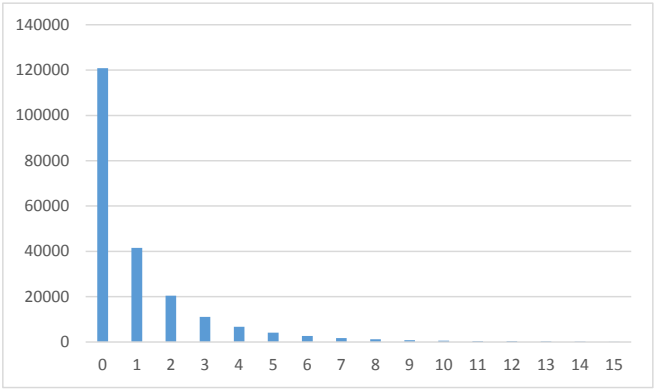

図 4　修正提出数の分布

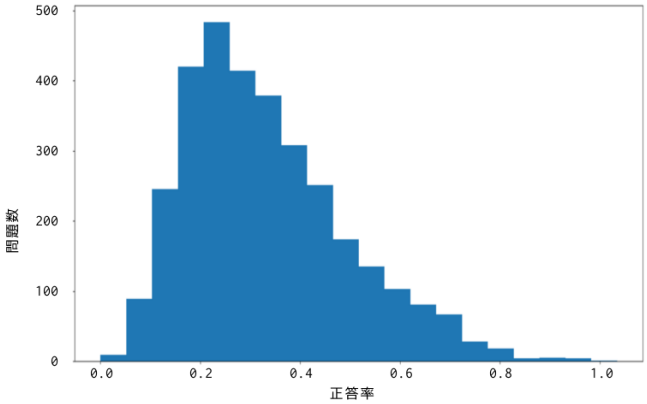

図 3　正答率ごとの問題の度数分布

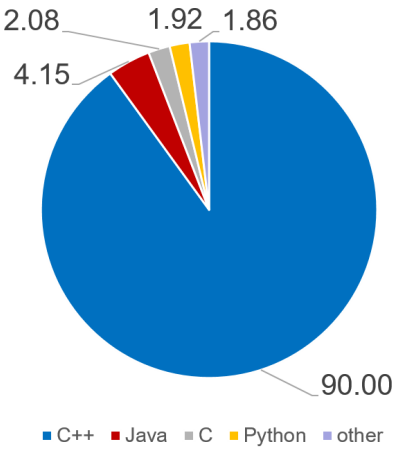

図 5　提出ソースコードの言語分布

提出における LD が小さい参加者であっても，平均 LD が大きい問題に多く提出する参加者は LD が大きいと判定されてしまう可能性がある．それを防ぐために LD を正規化する必要がある．

調査にあたって，我々は 2 つの Research Question(RQ) を設定した．RQ については以下の通りである．

RQ1.　プログラミングコンテストのレーティングとソースコード修正量は相関があるか

RQ2.　修正がソースコードのどの位置に対して行われやすいか

上記 RQ をもとに，ソースコード修正の分析調査を行うこととする．

4.3.1　RQ1: ソースコード修正量の分布

我々は，レーティングとソースコード修正量との間にどのような相関があるかの調査を行った．調査に際しては，初級者は上級者と比較してソースコード修正量が増加するという仮説を立て，相関分析と検定を行った．調査手順は以下の通りである．

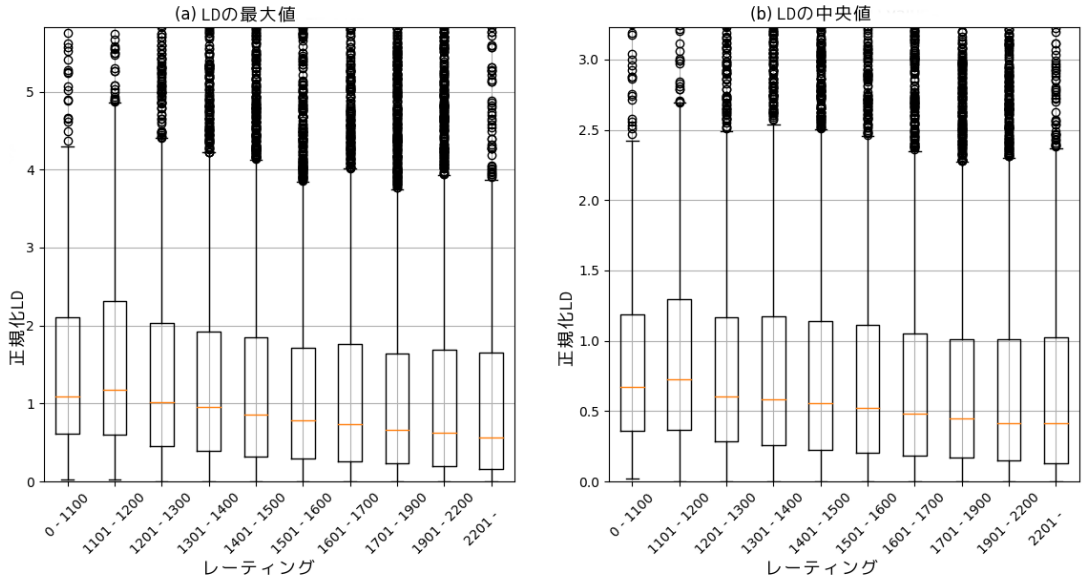

図 6 レーティング範囲ごとの NLD 最大値/中央値の箱ひげ図 ($0.2 \leq r < 0.4$)

表 4 LD 最大値とレーティングの相関

正答率 r	相関係数	有意差	延べ人数
$0 \leq r < 0.2$	-0.069	✓	3,638
$0.2 \leq r < 0.4$	-0.033	✓	25,682
$0.4 \leq r < 0.6$	-0.035	✓	14,368
$0.6 \leq r < 0.8$	-0.051	✓	5,360
$0.8 \leq r < 1.0$	0.010		220

表 5 LD 中央値とレーティングの相関

正答率 r	相関係数	有意差	延べ人数
$0 \leq r < 0.2$	-0.029		3,638
$0.2 \leq r < 0.4$	-0.019	✓	25,682
$0.4 \leq r < 0.6$	-0.028	✓	14,368
$0.6 \leq r < 0.8$	-0.039	✓	5,360
$0.8 \leq r < 1.0$	0.037		220

(1) 各問題で各参加者について NLD の最大値/中央値を計算しておく
(2) 手順 1 で計算した NLD 最大値/中央値と参加者のレーティングとのピアソン相関係数を計算する
(3) “NLD 最大値/中央値とレーティングとの間には有意な相関はない” という帰無仮説を立てて t 検定を実施する

表 4 と表 5 は NLD の最大値/中央値とレーティングとの相関分析結果である．表の最も左の列は問題の正答率を表す．各行は，問題を正答率ごとに分類し，分類されたグループにおける相関分析の結果を表している．我々は今回の分析で，正答率 0.2 区切りで 5 つのグループに問題を分類した．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，正答率ごとに相関分析を行った．表 4 と表 5 の 2 列目は，NLD の最大値/中央値とレーティングとの相関係数を表している．また，これらの表の 3 列目は相関分析における t 検定の結果を表しており，チェックがついているものは有意差が認められるということを表している．表 4 と表 5 の最後の列はそのグループに含まれる問題に提出を行った延べ人数である．図 6 は正答率 0.2 から 0.4 の範囲の問題に対する，各レーティング範囲における NLD 最大値/中央値の箱ひげ図である．我々は，レーティングごとに参加者を 10 グループに区切って，それぞれの NLD の分布を図示した．

表 4 と表 5 のどちらとも，相関係数は小さいものの有意に相関があることを表している．この結果から，上級者は初級者と比較して，有意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 6 は表 4，5 の 2 行目の分布を図示しており，実際にレーティングが高いほど NLD が小さいことが確認できる．

表 4 と表 5 において，$0.8 \leq r < 1.0$ の行は他と異なっている．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，図 3 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．

4.3.2 RQ2: ソースコード修正箇所の分布調査

4.3.1 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[2] 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 7 と 8 は，1100 < レーティング ≤ 1200 と 1700 <

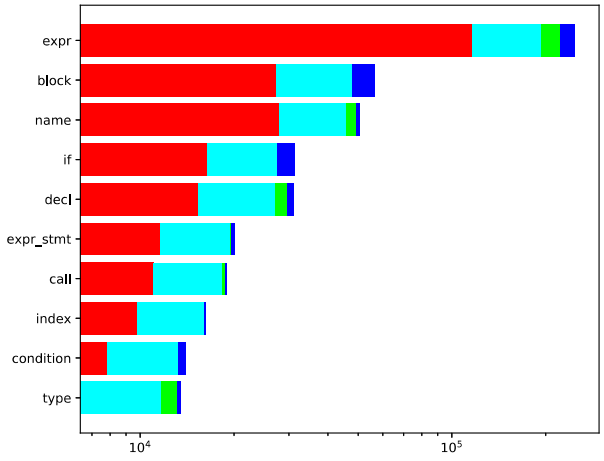

図 7　1100 < レーティング ≤ 1200 かつ $0.2 \leq r < 0.4$ における修正箇所の度数分布の上位 10 件 (対数軸)

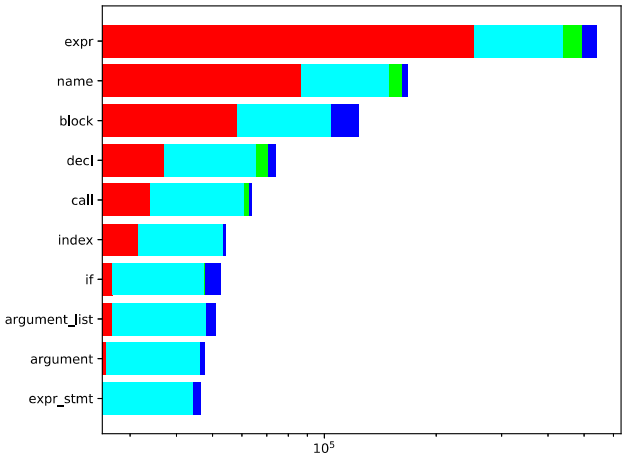

図 8　1700 < レーティング ≤ 1900 かつ $0.2 \leq r < 0.4$ における修正箇所の度数分布の上位 10 件 (対数軸)

レーティング ≤ 1900 の参加者が正答率 $0.2 \leq r < 0.4$ 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抽象構文木のノードを表し，横軸に変更回数を表す．グラフの色は以下の通り変更の種類を表す．

赤　ノードの追加
水色　ノードの削除
緑　ノードの置換
青　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．

4.4　考察

4.3.1 節では，プログラミングコンテスト参加者における修正量がどの程度であるかを調査した．RQ1 に対する回答として，相関係数は少ないものの，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，開発技術の向上につながる可能性がある．このため，4.3.2 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．4.3.2 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．RQ2 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．

4.5　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 4.3.1 節ではレーベンシュタイン距離を，4.3.2 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，修正/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修正作業量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．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 本研究は JSPS 科研費 25220003，16K16034，15H06344 の助成を受けた．

参考文献

[1] Elo, A. E.: *The rating of chessplayers, past and present*, Arco Pub. (1978).
[2] Falleri, J., Morandat, F., Blanc, X., Martinez, M. and Monperrus, M.: Fine-grained and accurate source code differencing, *Proc. of ASE 2014*, pp. 313–324 (2014).
[3] Goues, C. L., Dewey-Vogt, M., Forrest, S. and Weimer, W.: A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each, *Proc. of ICSE 2012*, pp. 3–13 (2012).
[4] Hanam, Q., de M. Brito, F. S. and Mesbah, A.: Discovering Bug Patterns in JavaScript, *ACM SIGCSE Bulletin*, pp. 144–156 (2016).
[5] Levenshtein, V. I.: Binary codes capable of correcting deletions, insertions, and reversals, *Soviet Physics Doklady 10*, pp. 707–710 (1966).
[6] Mirzayanov, M.: Codeforces Rating System, `http://codeforces.com/blog/entry/102` (2010).
[7] Mirzayanov, M.: Codeforces: Statistics 2015, `http://codeforces.com/blog/entry/22618` (2016).
[8] Petit, J., Giménez, O. and Roura, S.: Jutge.Org: An Educational Programming Judge, *Proc. of SIGCSE*, pp. 445–450 (2012).
[9] Revilla, M. A., Manzoor, S. and Liu, R.: Competitive learning in informatics: The UVa online judge experience, *Olympiads in Informatics*, Vol. 2, pp. 131–148 (2008).
[10] Zhong, H. and Su, Z.: An Empirical Study on Real Bug Fixes, *Proc. of ICSE 2015*, pp. 913–923 (2015).