メソッドレベルIR-based Bug Localizationでの模範クエリの発見と性能評価

猪俣 良輔　小林 隆志

IR-based Bug Localization(IRBL) は，バグレポート内の文書を入力としそのバグの原因箇所を情報検索技術を用いて推定する手法である．バグレポートからどのように検索クエリを構成するかが重要となる．特にメソッドレベルでの IRBL は精度が十分でないことが知られているが，検索クエリを改良することでどの程度精度が向上できるかは議論がなされていない．本論文では，これまでファイルレベルの IRBL 向けに調査がなされていた手法をメソッドレベルでの IRBL に適用し模範クエリを構成する．この模範クエリの内容での IRBL の精度を調査することで，メソッドレベルでの IRBL の改善可能性を議論する．32 プロジェクトから取得した合計 2782 バグレポートを対象に実験を行い，メソッドレベルにおいても既存の IRBL 手法の精度を上回る模範クエリが構成でき，検索クエリ改良の潜在的有効性が高いことを示した．

1　はじめに

ソフトウェア開発において, バグを発生させずに開発することは困難である．大きなソフトウェアプロジェクトでは，バグを再現させ，その原因箇所を特定し修正するデバッグ作業に多くの時間を要する [16]．バグ箇所局所化 (Bug Localization) 技術はバグの原因となるソースコード要素を特定するために使用され, バグ修正活動の労力やコストを削減するための重要な支援手法である．

バグ箇所局所化技術には, 動的な手法と静的な手法が存在し, これまで様々な提案がなされてきた [14]．動的な手法にはテストの成否と各テストケースで実行された箇所などの特徴量を Spectrum として記録・解析する Spectrum-based fault localization(SBFL) 手法 [8] などが存在するが，バグの挙動をテストケースとして再現した後でないと適用できない．一方で，静的な手法ではバグレポートをクエリ，ソースコードファイルを文書として情報検索技術を適用する IRBL 手法 [1] が提案されており，デバッグ作業の初期段階でバグ箇所を静的に絞り込むことが出来る利点がある．

IRBL 手法では検索精度向上のために様々な提案がなされてきた [1]．これらの手法は，バグレポートに出現する語彙とソースコード中の関数名や変数名などの識別子の語彙に同じものが含まれるという前提に基づいている．バグレポートが入力として与えられると，そのバグレポートに含まれる語彙を元に検索クエリを構成し，ソースコードとの類似度を計算することで，バグの原因である可能性の高いソースコード要素を出力する．出力結果として表示されるソースコード要素にはファイルやメソッドなどの粒度が存在する．これまでの IRBL 手法の多くはファイル単位での検索を行っているが，バグに関連するファイルが出力された時に修正すべきソースコードをファイル内から特定するには依然として人の労力を必要としている．近年，メソッドレベルでの IRBL 手法が注目されており，より細かな粒度での検索を行うことで人の労力を削減することが期待されている．しかし，メソッド単位での検索精度はファイル単位と比較して低いことが知られている [9]．

本研究の最終的な目的は，バグレポート中の単語を

Ryosuke Inomata, Takashi Kobayashia, 東京工業大学情報理工学院, School of Computing, Tokyo Institute of Technology.

そのまま使用するのではなく，クエリ改良 [2] を適用することによって，メソッド単位の IRBL 手法の検索精度を向上させることである．IRBL 手法におけるクエリ改良の潜在的有効性は， Mills らの報告 [6] で議論されている．彼らは，バグレポート中の単語のみを使用して最適に近い単語組み合わせを探索し，模範クエリを構成し IRBL 手法の有用性の調査を行っている．調査を通して，バグレポートの単語のみで構成される模範クエリが既存手法の精度を大幅に上回ることを示し，クエリを適切に構成することができればIRBL 手法は有効であることを示している．Mills らは，正解が不明な実際の状況下でバグレポートから模範クエリを構成する方法を明らかにはしていないが，適切にクエリ改良 [2] を適用することで精度が改善する可能性を示唆している．

しかしながら，現状で十分な精度が達成できていないメソッドレベルでのバグ箇所局所化は，クエリ改良によりどの程度まで精度が向上する余地があるのかは明らかでない．本稿では，メソッド単位の IRBL 手法においてもクエリ改良の潜在的有効性を明らかにすることを目的とし，Mills らの調査手法をメソッドレベルでのバグ箇所局所化に適用してメソッド単位での IRBL 向けの模範クエリを構成しその精度を調査する．

2 関連研究

2.1 IRBL 手法の有用性

情報検索を使用したバグ箇所局所化 (IRBL) 技術に関して多くの研究がなされている中で，IRBL 手法の有用性を疑問視する研究も存在する [3,11]．Kochhar ら [3] は，バグ箇所局所化の研究はバイアスの制御が不十分であると指摘した．バグレポートに記述される情報には観測された事象や期待される動作などの他にコードの一部であるクラス名やメソッド名が含まれる場合がある．クラス名やメソッド名などはバグの原因箇所を明示的に示す可能性が高く，これらの検索ヒントが存在する場合，バグ箇所局所化の精度が本来の性能よりも過大評価されると示した．Wang ら [11] は，検索に必要な情報がバグレポートに記述されることが少ないことを指摘し，IRBL 手法の適用に制限がかかっていることを示した．

2.2 模範クエリ

前述の IRBL 手法の有用性に関する懸念を受け，Mills ら [6] は，バグレポート中の単語からバグ箇所局所化に有効な単語を抽出し，ヒント情報との関係を調査している．彼らは，バグレポートの単語のみで構成される模範クエリが既存手法の精度を大幅に上回ることを示し，クエリを適切に構成することができれば IRBL 手法は有効であることを示した．さらに Mills らは検索ヒントを取り除いた模範クエリも構成し，検索ヒントが無い場合でも精度は低下するが模範クエリが有効であることも示した．

彼らの模範クエリ構成手法は，遺伝的アルゴリズムを使用している．遺伝的アルゴリズムは進化計算の一種であり，生物の進化を模倣した最適化手法である．クエリを個体として表現するために，バグレポートの単語数 n に対して n ビットのバイナリベクトルを使用する．ビットが 1 の場合，その単語がクエリに含まれ，ビットが 0 の場合は含まれないことを示す．そして，これらのクエリ（個体）を検索精度（適応度）に基づいて選択，交叉，突然変異を繰り返すことで最適解を探索する．

Mills らは jmetal framework[†1] を使用して遺伝的アルゴリズムを実装した．彼らは Lucene [5] を用いて 3 種類の検索精度指標（*Effectiveness*，逆 *Average Precision*，逆 *Recall*）を計算し，個体を評価した。交叉には SinglePointCrossover，突然変異には BitFlipMutation，選択オペレータにはルーレットホイールを使用し，各パラメータは， 集団サイズ：100，最大世代数：30,000，交叉確率：0.9，各単語の突然変異確率：1/n (n はバグレポートの単語数) を使用したと報告されている．

模範クエリは検索の正解情報を用いて探索的に構成されるため，実際にバグ箇所局所化を行う支援には利用できない．この問題に対し，我々は模範クエリに近いクエリを構成するために，ファイルレベルの IRBL を対象に模範クエリに選ばれた単語の特徴分析 [20]

†1 https://jmetal.sourceforge.net

とそれらの特徴に基づくクエリ改良方法 [19] を提案している．

2.3 FinerBench4BL

これまでの IRBL 手法研究の多くはファイル単位で行われており，メソッド単位での手法やデータセットは多くない．積田ら [9] はソースコード，検索の正解セット，バグレポートを含むメソッド単位のデータセットと 5 つの代表的な検索手法を提供する評価フレームワーク FinerBench4BL [10] を作成し，ファイル単位とメソッド単位の IRBL 手法を検索精度とデバッグ効率の観点で比較した．積田らの分析によれば，メソッド単位の IRBL 手法はファイル単位のものよりもデバッグ効率を向上させる一方で，検索精度は低下しており，メソッド単位の IRBL 手法には改善が必要であることが示されている．

FinerBench4BL は，ファイル単位の IRBL 手法評価フレームワークである Bench4BL [4] をメソッド単位に変換することで作成される．FinerBench4BL は Bench4BL で提供されている 6 つの代表的な検索手法のうち，5 つ（BugLocator [17]，BLUiR [7]，BRTracer [13]，AmaLgam [12]，BLIA [15]）を提供し，メソッド単位のデータセットを使用して検索精度を評価することができる．BugLocator [17] は過去の類似バグレポートやファイルサイズを使用する．BLUiR [7] はソースコードの構文情報を使用する．BRTracer [13] はバグレポートに含まれるスタックトレース情報を使用する．Amalgam [12] は履歴データを使用する．BLIA [15] はこれらの副次情報を統合する手法である．

FinerBench4BL のデータセットを構築する手順は以下の通りである．まず Bench4BL が提供する Git リポジトリに対してリポジトリ変換ツール Historinc [18] を適用し，メソッド単位のリポジトリに変換する．その後，検索の正解として扱われるバグに関連するソースコード要素をメソッド単位に変換し，バグレポートと修正ファイルの対応関係を取得する．そして最後に，ファイル単位の検索手法をメソッド単位に修正することで評価フレームワークを構築した．

表 1　対象プロジェクト

Group	Project	min #methods	max #methods	#versions	#BR
Commons	CODEC	439	1,310	6	27
	COLLECTIONS	5,967	6,997	5	59
	COMPRESS	788	2,591	15	105
	CONFIGURATION	1,449	6,073	11	107
	CRYPTO	488	488	1	4
	CSV	360	405	3	6
	IO	452	2,608	12	70
	LANG	2,661	6,336	15	158
	MATH	1,906	15,695	15	175
	WEAVER	469	469	1	1
JBoss	ENTESB	3,210	3,210	1	4
	JBMETA	4,477	4,813	3	15
Spring	AMQP	1,476	3,729	32	86
	ANDROID	1,901	1,926	2	8
	DATACMNS	835	4,043	30	104
	DATAJPA	559	1,970	32	107
	DATAMONGO	1,992	5,125	40	209
	DATAREDIS	2,525	8,490	15	44
	DATAREST	574	2,108	23	89
	MOBILE	182	780	3	8
	ROO	1,967	7,803	15	568
	SEC	4,359	8,500	41	422
	SECOAUTH	2,253	2326	6	61
	SGF	368	4,874	19	83
	SHDP	1,259	6,331	8	37
	SHL	409	588	2	6
	SOCIAL	804	1,267	4	10
	SOCIALFB	962	1,760	4	11
	SOCIALLI	830	830	1	2
	SOCIALTW	890	1,191	5	6
	SPR	34,788	55,691	10	89
	SWF	3,663	6,837	19	101

3 手法: メソッド単位模範クエリの作成

3.1 概要

本研究の目的は，メソッド単位の IRBL 手法においてクエリ改良が検索精度の向上にどの程度寄与しうるかを調査することである．そのために，我々は 2.2 で紹介した Mills らの手法と，2.3 で紹介したメソッド単位の IRBL 評価フレームワークを組み合わせて，メソッド単位の模範クエリを作成する．そして，その模範クエリの検索精度をベースラインと比較して評価し，メソッド単位でのクエリ改良による改善可能性を議論する．

3.2 データセット

我々は模範クエリの作成に必要なソースコード，バグレポートテキスト，およびバグレポートに紐づくバグ関連メソッドを取得するために FinerBench4BL のデータセット構築を再現した．FinerBench4BL の再現によって 46 プロジェクトのデータセットを取得した．

積田ら [9] は 46 プロジェクトのうち 37 プロジェ

クトを使用してメソッド単位の IRBL 手法の評価を行っており，本研究でもその 37 プロジェクトに含まれない 9 プロジェクトは対象外とした．また，37 プロジェクトのうちメソッド単位への変換が不十分であった LDAP，DATAGRAPH，SWS の 3 プロジェクトも本研究では対象外とした．さらに，BATCH は再現したデータセットでバグレポートの数が異なるため除外し，BATCHADM はバグレポートに紐づく修正ファイルの数が少ないため本研究の対象から除外した．最終的に本研究では 32 プロジェクトを対象として模範クエリの作成を行った．

表 1 に本研究で使用するバグレポート数を含むプロジェクトの情報を示す．メソッド数は検索で対象となるソースコード要素の数であり，数が多いほど検索で正解を見つける難易度が高くなる．データセットにはバグレポートの報告されたバージョン情報が記述されており，対応するバージョンのソースコードを使用して検索を行う．バージョン毎にメソッド数は異なり，各プロジェクトにおけるバージョン数および最小メソッド数と最大メソッド数を表 1 に示す．

3.3 実装

本研究では，FinerBench4BL を使用してメソッド単位で模範クエリを生成するために，遺伝的アルゴリズムを含む模範クエリ生成ツールを実装した．我々は Mills らの実装を参考にし，検索には Lucene 2.9.1 を使用し，*Effectiveness* を適合度として用いた．*Effectiveness* は，正解メソッドのうち検索結果の最上位にランク付けされたもののランクをスコアとし，値が小さいほど検索精度が高く，最も精度が高い場合は 1 である．

開発者はバグに関連するソースコードを修正するために，検索結果の上位にランク付けされたソースコード要素を参照する．そのため，*Effectiveness* は本研究において最も重要な検索精度指標であり，模範クエリを生成する遺伝的アルゴリズムの適合度として使用した．

遺伝的アルゴリズムの選択，交叉，突然変異のオペレーターや世代間の個体更新方法は Mills らの実装を参考にしている．次世代の子個体を作成するために親個体 2 つを選択する方法として BinaryTournament2 を使用し，交叉には SinglePointCrossover，突然変異には BitFlipMutation を採用した．世代交代による個体更新は親個体のうち適合度の高い個体をエリート個体として次世代に残すエリート保存戦略を採用した．具体的には，親個体から 2 個体をエリート個体として選び，それ以外を子個体の生成によって置き換える．遺伝的アルゴリズムのパラメータは，Mills らの実装を参考に，個体数 500，世代数 10，交叉確率 0.9，各単語ビットの突然変異確率 1/n (n は単語数) として設定した．

4 評価

4.1 実験方法・手順

FinerBench4BL を使用して作成した模範クエリの検索精度を評価するために，以下の手順で実験を行った．

1. FinerBench4BL で提供される 32 プロジェクトのデータセットを使用して，メソッド単位の模範クエリを生成する．
2. 生成した模範クエリを使用して Lucene で検索を行い，検索精度 (*MAP*, *HITS@K*) を計算する．
3. バグレポートの単語全てを使用する初期クエリを Lucene での検索に使用し，模範クエリと *MAP*, *HITS@K* を比較する．
4. FinerBench4BL で提供される 5 つの検索手法をベースラインとして模範クエリと *MAP* を比較する．

初期クエリを使用した Lucene と FinerBench4BL で提供される検索手法はバグレポートの単語全てを使用して検索を行う手法である．Lucene ではバグレポートとソースコードの類似度のみを使用してバグに関連するソースコード要素を検索する．それに対して，FinerBench4BL で提供される 5 つの検索手法は 2.3 で紹介した通り，バグレポートとソースコードの類似度だけでなく，副次情報を使用することで検索精度の向上を図っている．

表 2　初期クエリと模範クエリの検索精度

		平均正解	初期クエリ				模範クエリ			
project	#BR	含有率	*HITS@1*	*HITS@5*	*HITS@10*	*MAP*	*HITS@1*	*HITS@5*	*HITS@10*	*MAP*
CODEC	27	0.64 %	0.2963	0.3704	0.4815	0.2500	0.7407	0.7778	0.8148	0.4621
COLLECTIONS	59	0.14 %	0.4068	0.5424	0.6271	0.3597	0.7627	0.8814	0.8814	0.5180
COMPRESS	105	0.19 %	0.3333	0.4571	0.5048	0.3000	0.8095	0.9238	0.9238	0.6284
CONFIGURATION	107	0.11 %	0.2336	0.4393	0.5421	0.2598	0.8505	0.8879	0.9346	0.6168
CRYPTO	4	4.71 %	0.0000	0.5000	0.5000	0.1602	0.5000	0.5000	0.7500	0.2668
CSV	6	0.93 %	0.1667	0.5000	0.6667	0.2801	0.8333	1.0000	1.0000	0.6987
IO	70	0.2 %	0.4571	0.6429	0.7857	0.4320	0.9000	0.9286	0.9286	0.6808
LANG	158	0.08 %	0.4114	0.6266	0.6962	0.4326	0.7532	0.8671	0.8987	0.6122
MATH	175	0.12 %	0.3600	0.5600	0.6229	0.3194	0.8286	0.9086	0.9257	0.5565
WEAVER	1	1.71 %	0.0000	0.0000	1.0000	0.0641	1.0000	1.0000	1.0000	0.2825
ENTESB	4	0.32 %	0.0000	0.0000	0.0000	0.0105	0.2500	0.5000	0.5000	0.2148
JBMETA	15	0.09 %	0.1333	0.4667	0.5333	0.1748	0.5333	0.6000	0.7333	0.3995
AMQP	86	0.31 %	0.2442	0.4884	0.5698	0.1655	0.8023	0.8721	0.9186	0.3514
ANDROID	8	0.94 %	0.5000	0.5000	0.6250	**0.4412**	0.5000	0.7500	0.8750	**0.4378**
DATACMNS	104	0.25 %	0.2212	0.3462	0.4423	0.1285	0.6538	0.7596	0.7981	0.2828
DATAJPA	107	0.41 %	0.2804	0.4112	0.4953	0.1806	0.6636	0.8224	0.8505	0.3483
DATAMONGO	209	0.21 %	0.1531	0.3445	0.4163	0.1009	0.6220	0.7751	0.8278	0.2566
DATAREDIS	44	0.16 %	0.2727	0.4773	0.5227	0.1610	0.7955	0.8636	0.8636	0.3109
DATAREST	89	0.63 %	0.1124	0.3371	0.4157	0.0864	0.6067	0.7416	0.8202	0.2259
MOBILE	8	0.64 %	0.2500	0.2500	0.6250	0.3032	0.6250	0.8750	1.0000	0.5960
ROO	568	0.28 %	0.1849	0.3468	0.4454	0.1561	0.6532	0.7799	0.8451	0.3667
SEC	422	0.07 %	0.3555	0.5569	0.6327	0.3126	0.7891	0.8768	0.9005	0.5180
SECOAUTH	61	0.19 %	0.2623	0.5082	0.6557	0.2568	0.7213	0.8852	0.9180	0.4373
SGF	83	0.6 %	0.3373	0.4940	0.5904	0.1885	0.8554	0.9518	0.9639	0.3458
SHDP	37	0.14 %	0.2432	0.3514	0.4324	0.2025	0.7838	0.8649	0.9189	0.3727
SHL	6	0.84 %	0.3333	0.8333	1.0000	0.2648	1.0000	1.0000	1.0000	0.5745
SOCIAL	10	0.34 %	0.6000	0.7000	0.7000	0.4585	1.0000	1.0000	1.0000	0.6563
SOCIALFB	11	0.26 %	0.2727	0.5455	0.5455	0.2480	0.6364	0.8182	0.8182	0.4309
SOCIALLI	2	0.18 %	0.5000	0.5000	0.5000	0.5082	1.0000	1.0000	1.0000	0.7500
SOCIALTW	6	0.13 %	0.3333	0.6667	1.0000	0.5000	0.8333	1.0000	1.0000	0.8458
SPR	89	0.02 %	0.1573	0.2809	0.3933	0.1416	0.6067	0.7978	0.8315	0.3534
SWF	101	0.12 %	0.2970	0.4752	0.5545	0.2444	0.6733	0.8119	0.8812	0.4330

4.2　評価指標

本研究ではプロジェクト毎の検索精度の指標として *Mean Average Precision* (*MAP*) と *HITS@K* を使用する．*MAP* はバグレポート毎の検索精度として計算される *Average Precision* (*AP*) の平均であり，検索の正解セットを全体としてどれだけ高精度に検索できるかを示す．具体的には，*MAP* は次のように定義される．

$$MAP = \frac{1}{|BR|} \sum_{b=1}^{|BR|} AP(b)$$

ここで，$|BR|$ はプロジェクトに含まれるバグレポートの総数，$AP(b)$ はバグレポート b に対する AP を表し，以下のように定義される．

$$AP(b) = \frac{1}{R_b} \sum_{k=1}^{n} P(k) \cdot \mathrm{rel}(k)$$

ここで，R_b はバグレポート b に対する検索の正解メソッド総数，n は検索結果の総数，$\mathrm{rel}(k)$ はランク k に位置したファイルが正解セットに含まれる場合は 1，それ以外は 0 を表す．$P(k)$ は，検索結果の上位 k 件までの中で正解メソッドが占める割合を示し，次のように定義される．

$$P(k) = \frac{\text{ランク } k \text{ までの正解数}}{k}$$

例えば，検索結果の上位 10 件のうち正解メソッドが 7 件含まれている場合，$P(10) = \frac{7}{10}$ となる．

HITS@K は，プロジェクト毎の検索精度指標であり，検索結果の上位 K 件までの中で正解メソッドが含まれているバグレポートの割合を示す．*HITS@K* は次のように定義される．

$$HITS@K = \frac{1}{|BR|} \sum_{b=1}^{|BR|} hits(b, K)$$

ここで，$|BR|$ はバグレポートの総数を示す．$hits(b, K)$ はバグレポート b に対して検索結果の

上位 K 件の中に正解メソッドが含まれている場合は 1，含まれていない場合は 0 を示す．例えば，あるプロジェクトに対して 10 件のバグレポートがあり，*HITS@10* を計算する場合，全バグレポートのうち 7 件が上位 10 件の検索結果に正解メソッドを含めることができると，*HITS@K* は 0.7 となる．本研究では，$K = 1, 5, 10$ について *HITS@K* を計算した．

4.3 実験結果と考察

4.3.1 初期クエリと模範クエリの精度比較

表 2 に初期クエリと模範クエリの検索精度 (*MAP*, *HITS@K*) を示す．初期クエリの検索精度が模範クエリと同等の場合，両者に対応する箇所に下線が引かれている．また，初期クエリが模範クエリよりも高精度である場合は，対応する箇所が太字で示されている．

HITS@K を比較すると，すべてのプロジェクトで模範クエリの方が初期クエリと同等またはそれより高精度である．初期クエリが模範クエリと同等の精度を示したのは *HITS*@10 における WEAVER, SHL，SOCIALTW の 3 プロジェクトである．これらのプロジェクトはいずれも初期クエリと模範クエリの *HITS*@10 が最大値の 1 であり，バグレポート数が 10 以下である．バグレポート数が少ないため，*HITS*@10 の精度を比較するのは難しいが，*HITS*@1 や *HITS*@5 では模範クエリがより高精度であるため，模範クエリは初期クエリの検索精度を改善している．

HITS@K は検索結果上位 K 件までに正解を 1 つでも出力できるかどうかを評価するため，検索問題の難しさはソースコードのメソッド数に対する正解メソッド数に依存する．各プロジェクトの正解メソッド含有率の平均値を表 2 の 3 列目に示す．CRYPTO と WEAVER を除くすべてのプロジェクトで平均正解含有率は 1% 未満である．正解含有率 1% の場合，ランダムに 10 個のメソッドを選んで正解メソッドが 1 つでも含まれる確率は 0.096 となる．ランダムに選ぶ時の確率 0.096 と比較して，模範クエリの *HITS*@10 は ENTESB を除く全てのプロジェクトで 0.7 を超えており，ENTESB でも 0.5 である．これにより，模範クエリは検索精度の向上に貢献していることがわかる．平均正解含有率が 1% を超えた CRYPTO と WEAVER でも *HITS*@10 の値はそれぞれ 0.75 と 1.0 と高く，CRYPTO では *HITS*@1, 10 で，WEAVER では *HITS*@1, 5 で精度が改善していることから，模範クエリの貢献が確認できる．

表 3　初期クエリと模範クエリの AP の中央値と Wilcoxon の符号付き順位検定の p 値

project	初期クエリ 中央値	模範クエリ 中央値	p 値
CODEC	0.0228	0.1961	**0.0004**
COLLECTIONS	0.0455	0.3500	**0.0000**
COMPRESS	0.0455	0.5000	**0.0000**
CONFIGURATION	0.0645	0.5015	**0.0000**
CRYPTO	0.1301	0.1402	0.2500
CSV	0.1000	0.5000	0.0625
IO	0.1884	0.5917	**0.0000**
LANG	0.1274	0.4711	**0.0000**
MATH	0.0656	0.4136	**0.0000**
WEAVER	0.0641	0.2825	1.0000
ENTESB	0.0187	0.3313	0.2500
JBMETA	0.1000	0.5000	**0.0039**
AMQP	0.0495	0.2208	**0.0000**
ANDROID	0.1556	0.2262	0.2969
DATACMNS	0.0171	0.2426	**0.0000**
DATAJPA	0.0194	0.2001	**0.0000**
DATAMONGO	0.0135	0.1667	**0.0000**
DATAREDIS	0.0286	0.2160	**0.0000**
DATAREST	0.0234	0.1301	**0.0000**
MOBILE	0.0732	0.2799	**0.0312**
ROO	0.0276	0.1837	**0.0000**
SEC	0.0500	0.3333	**0.0000**
SECOAUTH	0.0655	0.2791	**0.0000**
SGF	0.0491	0.1747	**0.0000**
SHDP	0.0246	0.2440	**0.0000**
SHL	0.2216	0.5082	0.0938
SOCIAL	0.2030	0.5071	0.3125
SOCIALFB	0.2072	0.3562	0.0840
SOCIALLI	0.0165	0.5000	1.0000
SOCIALTW	0.1667	0.8500	0.1250
SPR	0.0079	0.1578	**0.0000**
SWF	0.0417	0.3239	**0.0000**

表 2 で初期クエリと模範クエリの *MAP* を比較すると ANDROID を除くすべてのプロジェクトで模範クエリが高精度を示した．表 3 に各プロジェクトの *AP* について Wilcoxon の符号付き順位検定の p 値を示す．p 値 0.05 を基準とすると，初期クエリに対して統計的に有意に精度が改善したのは CRYPTO，CSV，WEAVER，ENTESB，ANDROID，SHL，SOCIAL，SOCIALFB，SOCIALLI，SOCIALTW の 10 プロジェクトを除く 22 プロジェクトであった．*MAP* が悪化した ANDROID と統計的な有意差を確認できなかった 9 プ

ロジェクトは，すべてバグレポート数が 11 以下と少ないプロジェクトである．逆に，バグレポート数が多いプロジェクトでは模範クエリの精度が初期クエリよりも有意に高いことが確認された．

4.3.2 既存 IRBL 手法と模範クエリの精度比較

表 4 に初期クエリ，FinerBench4BL が提供する 5 つの検索手法，および模範クエリの *MAP* を示す．FinerBench4BL の 5 つの手法の *MAP* は，論文中 [9] に記載されたものである．7 つの *MAP* のうち最も高精度なものが太字で示され，次に高精度なものが下線で示されている．模範クエリは ENTESB, ANDROID，MOBILE を除く 29 プロジェクトで最も高精度を示した．残りの 3 プロジェクトでは 2 番目に高い精度を示し，これらはバグレポート数が 10 以下のプロジェクトであった．

模範クエリは *MAP* において初期クエリと既存の 5 つの IRBL 手法よりも高い精度を示し，*HITS@K* では問題の難しさに対して高い数値を示した．これはバグレポートの単語のみという限定的な情報でも，IRBL 手法を改善できることを示す．また，クエリを適切に構成することで精度が改善することが示されたことは，メソッド単位の IRBL 手法においてクエリ作成戦略が精度向上に寄与することを示している．

4.4 妥当性への脅威

本研究では遺伝的アルゴリズムの適合度として *Effectiveness* を使用したが，逆 *AP* や 逆 *Recall* を適合度として使用することを試していない．*MAP* という検索精度指標において，逆 *AP* を使用した方が高精度となる可能性があるためこの点についての検証が必要である．

我々は初期クエリと模範クエリの *AP* について Wilcoxon の符号付き順位検定を行っているが，データ入手の問題から FinerBench4BL の手法と模範クエリに対する統計検定を実施できなかった．*MAP* は平均値であるため，バグレポートごとに精度が実際に向上しているかを確認するには FinerBench4BL の手法に関するデータを入手し，統計検定を行う必要がある．

FinerBench4BL に含まれる一部のプロジェクトでは，バグレポート数が少ないため，統計的有意性を確保するのが難しい場合がある．そのため，これらの結果を一般化する際には慎重さが求められ，妥当性に対する懸念が残る．今後，FinerBench4BL のデータ構築手法を他のファイル単位データセットに適用し，バグレポート数が多いプロジェクトのメソッド単位データセットを構築することで，それらに対して本研究を適用したい．

5 おわりに

本研究では，Mills らが提案した模範クエリの作成方法をメソッド単位のデータセットを含む FinerBench4BL に適応することでメソッド単位の模範クエリを作成した．作成された模範クエリは多くのプロジェクトで既存の手法の精度を上回る高い精度を示した．特に，バグレポート中の単語のみを使用した模範クエリの精度が高いことは，メソッド単位の IRBL 手法において，クエリ作成戦略が精度改善に重要な役割を果たすことを示している．

今後の研究では，ファイル単位とメソッド単位の模範クエリの違いを分析し，どのようにして効果的な模範クエリを構築できるかを調査する．この分析により，模範クエリに近いクエリを構成するクエリ改良手法を検討し，メソッド単位のバグ箇所局所化においてバグレポートの情報を効率的に活用する IRBL 手法の提案を目指す．

謝辞： 本研究の一部は科研費（#22H03567）の助成を受けた．

参 考 文 献

[1] Akbar, S. and Kak, A.: A Large-Scale Comparative Evaluation of IR-Based Tools for Bug Localization, *Proc. MSR*, 2020, pp. 21–31.

[2] Huang, J. and Efthimiadis, E. N.: Analyzing and evaluating query reformulation strategies in web search logs, *Proc. CIKM*, 2009, pp. 77–86.

[3] Kochhar, P. S., Tian, Y., and Lo, D.: Potential Biases in Bug Localization: Do They Matter?, *Proc. ASE*, 2014, pp. 803–814.

[4] Lee, J., Kim, D., Bissyandé, T. F., Jung, W., and Le Traon, Y.: Bench4BL: reproducibility study on the performance of ir-based bug localization, *Proc. ISSTA*, 2018, pp. 61–72.

[5] McCandless, M., Hatcher, E., and Gospodnetić,

表 4　初期クエリと 5 つのベースラインと模範クエリの MAP

project	#BR	初期クエリ	BugLocator	BLUiR	BRTracer	AmaLgam	BLIA	模範クエリ
CODEC	27	0.2500	0.1920	0.3520	0.0940	0.3450	0.3810	**0.4621**
COLLECTIONS	59	0.3597	0.3660	0.3690	0.0940	0.3770	0.3940	**0.5180**
COMPRESS	105	0.3000	0.2810	0.3180	0.1700	0.3020	0.3490	**0.6284**
CONFIGURATION	107	0.2598	0.2100	0.3290	0.1380	0.3130	0.3630	**0.6168**
CRYPTO	4	0.1602	0.1060	0.0600	0.2640	0.0600	0.0580	**0.2668**
CSV	6	0.2801	0.1550	0.4230	0.1540	0.4370	0.4530	**0.6987**
IO	70	0.4320	0.4000	0.5060	0.2270	0.4990	0.4950	**0.6808**
LANG	158	0.4326	0.3740	0.5030	0.1670	0.5250	0.5360	**0.6122**
MATH	175	0.3194	0.2630	0.3550	0.1220	0.3550	0.3750	**0.5565**
WEAVER	1	0.0641	0.0680	0.0830	0.0150	0.0830	0.0380	**0.2825**
ENTESB	4	0.0105	0.0310	0.1670	**0.3500**	0.1660	0.2140	0.2148
JBMETA	15	0.1748	0.1460	0.2260	0.0350	0.2260	0.2140	**0.3995**
AMQP	86	0.1655	0.1550	0.1670	0.1060	0.1650	0.1870	**0.3514**
ANDROID	8	**0.4412**	0.2980	0.3430	0.2370	0.3530	0.3670	0.4378
DATACMNS	104	0.1285	0.1250	0.2040	0.1240	0.2010	0.2140	**0.2828**
DATAJPA	107	0.1806	0.1690	0.1730	0.1150	0.1750	0.1830	**0.3483**
DATAMONGO	209	0.1009	0.1140	0.1440	0.0960	0.1420	0.1650	**0.2566**
DATAREDIS	44	0.1610	0.1630	0.1600	0.1230	0.1520	0.1900	**0.3109**
DATAREST	89	0.0864	0.1000	0.1270	0.0770	0.1280	0.1280	**0.2259**
MOBILE	8	0.3032	0.3720	**0.6270**	0.3760	**0.6270**	0.5950	0.5960
ROO	568	0.1561	0.2000	0.1930	0.0870	0.2000	0.2280	**0.3667**
SEC	422	0.3126	0.2990	0.3340	0.1860	0.3390	0.3620	**0.5180**
SECOAUTH	61	0.2568	0.3010	0.2750	0.1760	0.2850	0.3180	**0.4373**
SGF	83	0.1885	0.1820	0.1590	0.1230	0.1680	0.1690	**0.3458**
SHDP	37	0.2025	0.2000	0.1610	0.1660	0.1600	0.2110	**0.3727**
SHL	6	0.2648	0.2270	0.3270	0.2260	0.3270	0.3140	**0.5745**
SOCIAL	10	0.4585	0.3160	0.4990	0.4060	0.4990	0.5010	**0.6563**
SOCIALFB	11	0.2480	0.1840	0.3000	0.3280	0.3180	0.3180	**0.4309**
SOCIALLI	2	0.5082	0.5110	0.5140	0.5050	0.5140	0.5150	**0.7500**
SOCIALTW	6	0.5000	0.4880	0.3200	0.1930	0.3200	0.2820	**0.8458**
SPR	89	0.1416	0.2030	0.1670	0.1410	0.2270	0.2540	**0.3534**
SWF	101	0.2444	0.2120	0.2430	0.1910	0.2290	0.2710	**0.4330**

O.: *Lucene in action*, Vol. 2, Manning Greenwich, 2010.

[6] Mills, C. et al.: On the relationship between bug reports and queries for text retrieval-based bug localization, *Empir. Software Eng.*, Vol. 25, No. 5(2020).

[7] Saha, R. K., Lease, M., Khurshid, S., and Perry, D. E.: Improving bug localization using structured information retrieval, *Proc. ASE*, 2013, pp. 345–355.

[8] Sarhan, Q. I. and Beszédes, A.: A Survey of Challenges in Spectrum-Based Software Fault Localization, *IEEE Access*, Vol. 10(2022), pp. 10618–10639.

[9] Tsumita, S., Hayashi, S., and Amasaki, S.: Large-Scale Evaluation of Method-Level Bug Localization with FinerBench4BL, *Proc. SANER*, 2023, pp. 815–824.

[10] Tsumita, S., Hayashi, S., and Amasaki, S.: Replication package of FinerBench4BL, `https://github.com/salab/FinerBench4BL`, 2023.

[11] Wang, Q. et al.: Evaluating the usefulness of IR-based fault localization techniques, *Proc. ISSTA*, 2015, pp. 1–11.

[12] Wang, S. and Lo, D.: Version history, similar report, and structure: putting them together for improved bug localization, *Proc. ICPC*, 2014, pp. 53–63.

[13] Wong, C.-P., Xiong, Y., Zhang, H., Hao, D., Zhang, L., and Mei, H.: Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis, *Proc. ICSME*, 2014, pp. 181–190.

[14] Wong, W. E. et al.: A Survey on Software Fault Localization, *IEEE Trans. Soft. Eng.*, Vol. 42, No. 8(2016), pp. 707–740.

[15] Youm, K. C., Ahn, J., Kim, J., and Lee, E.: Bug Localization Based on Code Change Histories and Bug Reports, *Proc. APSEC*, 2015, pp. 190–197.

[16] Zeller, A.: *Why Programs Fail, 2nd ed.*, Morgan Kaufmann, 2009.

[17] Zhou, J., Zhang, H., and Lo, D.: Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports, *Proc. ICSE*, 2012, pp. 14–24.

[18] 柴駿太, 林晋平: Historinc: 細粒度履歴追跡のための増分的なリポジトリ変換ツール, コンピュータソフトウェア, Vol. 39, No. 4(2022), pp. 75–85.

[19] 猪俣良輔, 小林隆志: 情報検索に基づくバグ箇所局所化でのクエリ改良の効果, 電子情報通信学会技術研究報告 Vol.123, No.123, pp.37–42, 2023.

[20] 林和輝, 小林隆志: 欠陥箇所検索に有効なバグレポート中のキーワード抽出, 電子情報通信学会技術研究報告 Vol.121, No.416, pp.84–89, 2022.