要求仕様に対するテストカバレッジ分析におけるグラフクエリの適用について

有若 新悟
大阪大学
s-ariwaka@ist.osaka-u.ac.jp

中川 博之
大阪大学
nakagawa@ist.osaka-u.ac.jp

土屋 達弘
大阪大学
t-tutiya@ist.osaka-u.ac.jp

要旨

本研究では，グラフデーターベース，および，グラフクエリのソフトウェア工学上の課題への応用について議論する．具体的には，要求仕様の集合とテストケース記述の集合，および，それらの要素間の類似度がデータとして与えられていることを前提に，どの要求仕様がどの程度テストされているかをグラフデータベースを用いて分析する方法を提案する．典型的な分析項目に対するグラフクエリと *SQL* クエリを示し，記述の簡潔さについてグラフクエリが優れていることを示す．また，実用規模のデータを用いて，クエリの処理時間についても比較を行う．

1. はじめに

グラフデータベースは，グラフ構造を有するデータを管理するデータベースであり，関係データベースにおける表形式での扱いが難しいデータの扱いに優れる [1]．グラフクエリは，グラフデータベース上のデータへのアクセスを行うクエリであり，探索したいデータのグラフ上の特徴を直観的に指定することができる．様々なグラフ構造を有するデータを取り扱うために，グラフデータベースが普及しつつある一方で，ソフトウェア工学分野における課題への応用例は多くない．

本研究では，ソフトウェアシステムの要求仕様をテストケースがどの程度網羅しているかというカバレッジの分析において，グラフデータベース，および，データクエリが有効に利用できるかどうかについて議論する．前提として，要求仕様の集合とテストケース記述の集合，および，それらの要素間の類似度が得られていることを仮定する．その上で，このデータをネットワーク構造として表現し，データクエリを用いて，カバレッジ分析に必要な情報を得ることが可能なことを示す．また，関係データベースで用いられる SQL クエリと，クエリの理解しやすさと簡潔さの点で比較を行う．更に，実際のプロジェクトから得られたデータを用いて，クエリの処理時間についても議論する．

2. データと抽出する情報

本論文では，既存研究 [2, 3, 4, 5] で扱っている形式のトレーサビリティデータが得られることを想定する．これらの研究では，自然言語で記述されたテストケースと要求仕様について，それらの類似度を自然言語処理を用いて推定している．このトレーサビリティデータは，テストケースの集合と要求仕様の集合，および，それらの要素間の類似度から構成される．類似度は，1) 異なる二つの要求仕様間，および，2) テストケースと要求仕様間，に定義される．また，類似度は，0 以上 1 以下の実数とする．表 1 にデータの例を示す．ここでは，要求仕様の集合は $\{r0, r1, \ldots, r6\}$, テストケースの集合は $\{t0, t1, \ldots, t9\}$, 類似度は表 1 の通りである．

次に，このデータを用いて，テストケースによってどの要求仕様がどの程度テストされているのかを調べることを考える．このカバレッジ分析において，データから得られる情報で有用と考えられる例を検討した [6]．そ

表 1. トレーサビリティデータの例

(a) 要求仕様間

$r0$	$r1$	0.14
$r0$	$r5$	0.08
$r0$	$r6$	0.59
$r1$	$r4$	0.97
$r1$	$r5$	0.84
$r2$	$r3$	0.2
$r2$	$r4$	0.94
$r2$	$r6$	0.34
$r4$	$r5$	0.93

(b) テストケースと要求仕様間

$t0$	$r0$	0.13
$t1$	$r0$	0.57
$t1$	$r1$	0.82
$t2$	$r2$	0.35
$t3$	$r2$	0.16
$t4$	$r2$	0.59
$t4$	$r4$	0.21
$t5$	$r4$	0.01
$t5$	$r5$	0.03
$t7$	$r1$	0.91
$t7$	$r2$	0.41
$t7$	$r4$	0.09
$t8$	$r0$	0.06
$t8$	$r1$	0.28
$t8$	$r2$	0.48
$t8$	$r5$	0.5
$t9$	$r1$	0.21
$t9$	$r2$	0.66
$t9$	$r5$	0.54

れらの例のいくつかを以下に挙げる．

1. 仕様 R を直接・間接にテストしているテストケースの一覧
2. テストケース T が直接・間接にテストしている仕様の一覧
3. 一つ以上のテストケースで直接・間接にテストされている仕様の一覧

ここでは，ある要求仕様に対して，類似度が一定のしきい値（X）以上のテストケースが存在している場合，この要求仕様はこのテストケースによって直接テストされていると定義する．また，ある要求仕様に対して，関連度が一定のしきい値（Y）以上である要求仕様が存在し，後者の要求仕様を直接テストしているテストケースが α 個以上存在する場合，前者の要求仕様は後者の要求仕様を直接テストしているテストケースによって間接テストされていると定義する．論文記述の簡素化のため，以降，三つのパラメータには X=0.5，Y=0.7，α=2 を割り当てる．

3. グラフデータとグラフクエリ

3.1. グラフデータ

本論文では，グラフ構造の表現形式として，グラフデータベース Neo4j のものを用いる．グラフ構造は，ノード・リレーションシップ・プロパティの 3 要素から成るデータ構造である．通常のグラフ理論の用語では，ノードは頂点，リレーションシップはノード同士を繋ぐ有向辺を表す．また，プロパティはノードとリレーションシップに付与される属性を意味する．

2 節で述べたトレーサビリティデータをグラフデータとして以下のように表現する．テストケースと要求仕様はノードに対応する．以降，テストケースに対応するノードをテストケースノード，要求仕様に対応するノードを要求ノードと呼ぶ．そして，要求仕様間，および，テストケースと要求仕様間に定義される類似度は，リレーションシップと，それに付与されるプロパティに対応する．ただし，類似度が 0 の場合は，リレーションシップ (辺) を設定しない．また，リレーションシップの向きは，ここで扱うデータにとっては意味を持たないので，要求ノード間，および，要求ノードとテストケースノード間

には，いずれかの向きのリレーションシップを一方のみを設定する．

3.2. グラフクエリ

2 節で挙げたカバレッジ分析に関する項目それぞれについて，対応する情報を抽出するためのグラフクエリを作成した．グラフクエリの記述には，Neo4j で用いられているクエリ言語である Cypher を用いた．Cypher はグラフ構造上の検索パターンを短いクエリで表現できることが特徴である．以降ではこれらのグラフクエリと，2 節の例に適用して得られる結果を示す．

(1) 仕様 R を直接・間接にテストしているテストケースの一覧

```
1  CALL {
2      MATCH (r:Requirement)-[s:
           Similarity]-(t:Testcase)
3      WHERE r.name = $s AND s.value >=
            0.5
4      RETURN t.name AS test_name
5
6      UNION
7
8      MATCH (r1:Requirement)-[s1:
           Similarity]-(r2:Requirement)
9      WHERE r1.name = $s AND s1.value
           >= 0.7
10     OPTIONAL MATCH (r2)-[s2:
           Similarity]-(t:Testcase)
           WHERE s2.value >= 0.5
11     WITH r2, count(t) AS count WHERE
            count >= 2
12     OPTIONAL MATCH (r2)-[s3:
           Similarity]-(t2:Testcase)
           WHERE s3.value >= 0.5
13     RETURN t2.name AS test_name
14 }
15 RETURN test_name
16 ORDER BY test_name
```

このクエリは，要求ノードの名前を指定し，その要求仕様を直接・間接にテストしているテストケースの一覧を取得する．3 行目，9 行目の WHERE 句で，要求ノードの名前を指定している．直接テストしているテストケースを求める部分（2 行目〜4 行目）と，間接にテストしているテストケースを求める部分（8 行目〜13 行目）を UNION 句によって合成している．

2 節の例に対して，クエリを実行した場合，要求ノード r1 を直接または間接にテストしているテストケースとして，t1, t7, t8, t9 が得られる．

(2) テストケース T が直接・間接にテストしている仕様の一覧

```
1  CALL {
2      MATCH (t:Testcase)-[s:Similarity
           ]-(r:Requirement)
3      WHERE t.name = $s AND s.value >=
            0.5
4      RETURN r.name AS req_name
5
6      UNION
7
8      MATCH (t1:Testcase)-[s2:
           Similarity]-(r1:Requirement)
9      WHERE t1.name = $s AND s2.value
           >= 0.5
10     OPTIONAL MATCH (r1)-[s3:
           Similarity]-(t2:Testcase)
           WHERE s3.value >= 0.5
11     WITH r1, count(t2) AS count
           WHERE count >= 2
12     OPTIONAL MATCH (r1)-[s1:
           Similarity]-(r2:Requirement)
            WHERE s1.value >= 0.7
13     RETURN r2.name AS req_name
14
15 }
16 RETURN req_name
17 ORDER BY req_name
```

このクエリでは，テストケースノードの名前を指定することで，そのテストケースが直接・間接にテストしている要求仕様の一覧を得る．3 行目，9 行目の WHERE 句で，テストケースノードの名前を指定している．このクエリを実行すると，テストケース t1 が直接または間接にテストしている要求仕様として，r0, r1, r4, r5 が得られる．

(3) 一つ以上のテストケースで直接・間接にテストされている仕様の一覧

```
1  CALL {
2      MATCH (r:Requirement)-[s:
           Similarity]-(t:Testcase)
           WHERE s.value >= 0.5
3      RETURN r.name AS req_name
4
5      UNION
6
7      MATCH (r1:Requirement)-[s1:
           Similarity]-(t:Testcase)
           WHERE s1.value >= 0.5
8      WITH r1, count(t) AS count WHERE
            count >= 2
9      MATCH (r1:Requirement)-[s2:
           Similarity]-(r2:Requirement)
            WHERE s2.value >= 0.7
10     RETURN r2.name AS req_name
11 }
12 RETURN req_name
13 ORDER BY req_name
```

このクエリでは，一つ以上のテストケースで直接・間接にテストされている仕様の一覧を得るクエリである．2

節の例に対してこのクエリを実行した場合，要求仕様 r0, r1, r2, r4, r5 が得られる．

3.3 SQL クエリとの比較

比較のため関係データベースを用いて，同様の処理を実現する方法を検討した．まず，2 節のトレーサビリティデータを管理するためのテーブルを設計した．具体的には，要求仕様の集合，テストケースの集合，類似度の集合を管理するため，それぞれに対して独立したテーブルを設定した．グラフデータベースでは一つのグラフ構造によってデータが表現できたのに対し，このように関連データベースでは複数の表を操作する必要があった．

上記のようにテーブルを設定した上で，クエリ言語である SQL を用いて，同様の目的を果たすクエリを記述した．表 2 に，対応するクエリにおける文字数の比較結果を示す．Cypher を用いた場合，記述量を半分近く削減できていることが分かる．

表 2. クエリの文字数

クエリ	Cypher	SQL
クエリ 1	415	724
クエリ 2	413	795
クエリ 3	325	599

例として，クエリ 2 に対する SQL クエリを以下に示す．SQL クエリは，データ同士の関係性を処理するために，JOIN キーワードを用いて何度もテーブルを結合する必要がある．また，グラフ構造に対して，グラフクエリのような，パターンマッチを用いた検索を行うこともできない．そのため，直接グラフ構造を扱うことができるグラフクエリに対して，クエリが冗長になっている．

(2) テストケース T が直接・間接にテストしている仕様の一覧 (SQL)

```
(SELECT req.name AS req_name
FROM edge_req_to_test edge
JOIN node_req req
ON edge.from_id = req.id
JOIN node_test test
ON edge.to_id = test.id
WHERE test.name = %s AND edge.
    similarity >= 0.5

UNION

(WITH req AS (
    SELECT id1 FROM (
        SELECT req.id AS id1
        FROM edge_req_to_test edge
        JOIN node_req req
        ON edge.from_id = req.id
        JOIN node_test test
        ON edge.to_id = test.id
        WHERE test.name = %s AND
            edge.similarity >= 0.5
    ) AS r
    JOIN edge_req_to_test edge
    ON edge.from_id = r.id1
    JOIN node_test test
    ON edge.to_id = test.id
    WHERE edge.similarity >= 0.5
    GROUP BY id1
    HAVING count(*) >= 2
)

SELECT req2.name AS req_name
FROM edge_req_to_req edge
JOIN node_req req2
ON edge.to_id = req2.id
WHERE edge.from_id IN (SELECT id1
    FROM req) AND edge.similarity >=
     0.7

UNION

SELECT req2.name AS req_name
FROM edge_req_to_req edge
JOIN node_req req2
ON edge.from_id = req2.id
WHERE edge.to_id IN (SELECT id1 FROM
     req) AND edge.similarity >= 0.7
))

ORDER BY req_name;
```

4. ケーススタディ

本節では，実際のシステム開発において得られたデータをグラフとしてグラフデータベースに格納した上で，前節で示したクエリを実行した際の結果について説明する．用いたデータの規模は，テストケース 3,855 個，要求仕様 260 個であった．

グラフデータベースとして Neo4j を，関係データベースとして postgreSQL を用いた．実験を行った計算機は，CPU AMD Ryzen 5 3600, メモリ 16GB，OS は Windows 10 である．

実験の前処理として，自然言語処理を通して得られた要求仕様間，および，テストケースと要求仕様との間の関連度のデータを，Cypher クエリを繰り返し実行することで，グラフ構造としてデータベース上に保存した．この処理は Python プログラムからデータベースの API を呼び出す形で実行した．この処理にかかった時間は，Neo4j では約 47 分，PostgreSQL では約 4 分であった．

各クエリを実行した結果，完了までにかかった時間を表 3 に示す．なお，クエリ 1 と 2 については，検索の起点として，要求仕様，テストケースをすべてを選んで実行し，実行時間の平均を取った．また，Neo4j では初回のアクセスには時間がかかるため，その影響を受けないように，事前に適当なクエリを実行した後に各クエリの実行時間の計測を行うようにした．PostgreSQL ではどのクエリも短時間で実行することができた．Neo4j では，一つ以上のテストケースで直接・間接にテストされている仕様の一覧を得るクエリ 3 で，比較的長い実行時間が必要であった．260 個の要求ノードすべてに対して網羅的に計算を行い，140 個もの仕様が抽出されたが，その場合でも実行時間は 5 秒程度であった．したがって，関係データベースよりは時間が必要ではあるが，十分実用的と考えられる．

表 3. クエリの実行時間 (秒)

クエリ	Neo4j	PostgreSQL
クエリ 1	0.054	0.075
クエリ 2	0.047	0.125
クエリ 3	5.086	0.098

5. 関連研究

本研究は，著者の以前の研究である文献 [6] を発展させたものである．具体的には，グラフクエリの記述を見直し，より簡素な記述を得た．また，その結果，処理時間についても改善が見られ，たとえばクエリ 3 では 50% 以上低減された．さらに，SQL クエリとの比較を行うため，関係データベースでのテーブルの設計と，その設計に基づく SQL クエリを記述し，記述の簡潔さだけでなく，クエリの処理時間についても評価を行った．

ソフトウェアアーティファクトのトレーサビリティに関して，グラフデータベースを応用した研究が幾つか知られている．文献 [7] では，テストケースのトレーサビリティに関するクエリについて，SQL と Cypher を含む 4 種類のクエリ言語を比較し，Cypher が表現力，理解容易性などの点で優れていることを示している．また，文献 [8] では，ネットワークによって要求，コード，テストケース間のトレーサビリティリンクを表現し，2 種類の単純なクエリについて，SQL と Cypher とでの表現の簡潔さについて比較している．これらの研究と本研究とでは，前提としているトレーサビリティデータや，クエリ実行の目的が大幅に異なっている．例えば，本研究ではこれらの先行研究で扱っているようなアーティファクト間の明示的なリンクではなく，類似度という数量を前提としており，また，要求仕様に対するテストカバレッジの分析という目的も，先行研究とは異なっている．文献 [9] では，データが大規模な場合におけるクエリの処理性能について議論されており，クラスタコンピューティングフレームワークである Spark 上の関係データベースによって SQL クエリを処理する方が，Neo4j と Cypher を用いるよりも処理性能が高いことを示している．単一の計算機を用いた本研究でも SQL クエリの方が処理時間の点で比較的有利であったが，前節の通り，ケーススタディで用いたデータセットに対してはグラフクエリでも十分に実用的な時間で処理ができた．トレーサビリティ以外のソフトウェア工学分野におけるグラフクエリの応用としては，文献 [10] において，グラフデータベースを用いて，ソフトウェアプロダクトラインにおける部品関係の可視化を行った事例について報告されている．

トレーサビリティリンクの自動構築については，いくつか研究が知られている．テストケースと要求仕様との類似度を用いて要求仕様のテストカバレッジを分析するアプローチについては，本研究の著者等による先行研究で，類似度の算出とトレーサビリティリンクの自動構築を行っている [2, 3, 4, 5]．本研究のケーススタディでは，これらの研究の方法で類似度を計算した．その他のトレーサビリティリンクの自動構築については，文献 [11, 12, 13] 等の研究がある．

6. おわりに

本研究では，要求仕様とテストケースに対して類似度がデータとして与えられていることを前提に，カバレッジ分析にグラフクエリを用いることが可能なことを示した．具体的には，重要な情報がグラフクエリによって記述できることを示し，SQL クエリより記述量を減らせることを示した．また，実用規模のデータを用いたケーススタディにより，クエリの処理時間について SQL 程ではないが，十分に実用的であることを示した．

今後の課題としては，より定量的な評価がある．たとえば，今回は二つの種類のクエリを文字数で比較したが，トークン数など，クエリの構造をより反映した尺度での比較評価の方が，より適切である可能性がある．また，

他のデータ，特に，規模の大きいデータへの適用や，ソフトウェア工学上の他の課題への応用などを検討したい．

謝辞 本研究は JSPS 科研費 JP20K11747 の助成を受けたものである．

参考文献

[1] I. Robinson, J. Webber, and E. Eifrem, Graph Databases: New Opportunities for Connected Data, 2nd edition, O'Reilly Media, Inc., 2015.

[2] H. Nakagawa, T. Hasegawa, S. Matsui, and T. Tsuchiya, "Visualization of specification coverage: A case study of a web application development in industry," 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp.77–80, 2017.

[3] H. Nakagawa, S. Matsui, and T. Tsuchiya, "A visualization of specification coverage based on document similarity," Proceedings of the 39th International Conference on Software Engineering Companion, p.136–138, ICSE-C '17, IEEE Press, 2017. https://doi.org/10.1109/ICSE-C.2017.117

[4] 東和幸，中川博之，土屋達弘，"文書間の類似度に基づいたトレーサビリティリンクの精度向上手法の検討," 電子情報通信学会 知能ソフトウェア工学研究会 (SIG-KBSE) 信学技報，pp.25–30，Nov. 2019．

[5] 松井勝利，中川博之，土屋達弘，"文書間の類似度に基づいた要求カバレッジ可視化手法," 日本ソフトウェア科学会 学会誌『コンピュータソフトウェア』，vol.35，no.1，pp.67–75，Feb. 2018．

[6] 有若新悟，中川博之，土屋達弘，"要求-テストケース間のカバレッジ分析におけるグラフクエリの応用可能性の検討," 電子情報通信学会 知能ソフトウェア工学研究会 (SIG-KBSE) 信学技報，pp.53–58，Nov. 2020．

[7] M. Rath, D. Akehurst, C. Borowski, and P. Mäder, "Are graph query languages applicable for requirements traceability analysis?," Joint Proceedings of REFSQ-2017 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017), vol.1796, p.(unassigned), CEUR Workshop Proceedings, 2017.

[8] R. Elamin and R. Osman, "Implementing traceability repositories as graph databases for software quality improvement," 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp.269–276, 2018.

[9] J. Lin, Y. Liu, and J. Cleland-Huang, "Supporting program comprehension through fast query response in large-scale systems," Proceedings of the 28th International Conference on Program Comprehension, p.285–295, ICPC '20, Association for Computing Machinery, New York, NY, USA, 2020.

[10] 川井隆之，小川雄太，水藤倫彰，"文書間の類似度に基づいたトレーサビリティリンクの精度向上手法の検討," ソフトウェア・シンポジウム 2020，p.84，2020．

[11] F. Erata, M. Challenger, B. Tekinerdogan, A. Monceaux, E. Tüzün, and G. Kardas, "Tarski: A platform for automated analysis of dynamically configurable traceability semantics," Proceedings of the Symposium on Applied Computing, p.1607–1614, SAC '17, Association for Computing Machinery, New York, NY, USA, 2017. https://doi.org/10.1145/3019612.3019747

[12] A. Goknil, I. Kurtev, and K. Van Den Berg, "Generation and validation of traces between requirements and architecture based on formal trace semantics," Journal of Systems and Software, vol.88, pp.112–137, 2014.

[13] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," Computer, vol.40, no.6, pp.27–35, 2007.