

AIを活用したビッグデータ管理の動向と ソフトウェアテストへの期待

Osaka University

Prof. Makoto Onizuka (oni@acm.org)



Onizuka-Lab.

略歴

- 阪大 (2014-現在)
- NTT研究所(1991-2014): 研究11年, 開発10年, 研究管理2年
 - ワシントン大 (2000-2001)
 - 学位取得 (2007 東工大 佐伯先生)
 - 電通大客員(准)教授 (2011-2014)



1. クラウドにおけるデータ管理

- データベースエンジン(DBMS)における試験に関する研究事例
- クラウドスケールのDBMS試験の取り組み

2. グラフ深層学習

- グラフ深層学習の応用事例, ICSE2022 での応用事例
- グラフ深層学習(表現学習)について
- グラフ深層学習技術を評価するベンチマーク

前半: クラウドにおけるデータ管理

4

クラウドにおけるデータ管理とは？

- 本講演では**クラウド環境におけるデータベースサービス**について考える。
 - 例: AWS の分散DB系 (RDS (Relational Database Service), Redshift) やNoSQL系 (DynamoDB), AWS上での分析データサービス事業 (トレジャーデータのCDP)
- 技術動向
 - 分散DBMS, NoSQL から発展してきた流れ.
 - 多くの機能において機械学習を適用する技術が発展してきている.
 - 学習型インデックス, 学習型クエリ最適化, 学習型スキーマ設計 (インデックス設計, 実体化ビュー設計)
- ソフトウェアテストにおける課題
 - OSSが使われることが多く, ソフトウェアスタックの各階層においてバージョンアップが頻繁に行われる. 都度, 回帰試験が必要だが, データもクエリ数も膨大.

クラウドにおけるデータ管理に関する取り組み

7

- クラウドDB管理(トレジャーデータ)
 - DBエンジンの試験(DBTest2022), 性能シミュレータ・クラスタ最適化
- 機械学習によるクエリ最適化(NEDO(2018-2022, 2023-2026), 天文台, 東芝):
 - 実体化ビュー設計, カーディナリティ予測, 強化学習によるデータ分散最適化(aiDM 2023)
- データ統合(基盤S(2017-2021), 挑戦的研究(開拓)(2023-2027))
 - 連合学習(SDM 2022), カーディナリティ予測(SIGMOD 2021, 2022)
- トレーサブルなデータ管理(Crest(2022-2027)):
- グラフクエリ高速化(基盤A(2020-2025)):
 - 探索の枝刈り(SIGMOD2023), インデックス(ICDE2022)
 - グラフ深層学習に関する成果は後半のスライドにて

- Detecting Logic Bugs of Join Optimizations in DBMS, SIGMOD 2023
 - MySQLのような普及しているDBMSでさえも、24時間で31件のバグを発見
- DuckDB Testing Present & Future, DBTest 2022
 - Fuzzing の登場によって多くのバグを見つけられるようになった
- Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift, SIGMOD 2023
 - 予測が外れた際のプロテクションの仕組みが不可欠
- Journey of Migrating Millions of Queries on The Cloud, DBTest 2022
 - 試験が必要な膨大なクエリ群に優先順位を適切につける
 - DBとクエリ結果を秘匿するためクエリ結果はハッシュ化して試験を実施

Detecting Logic Bugs of Join Optimizations in DBMS, SIGMOD2023 (best paper)

9

- SQL join 最適化のバグを見つける研究
- 技術ポイント: テストデータとテストクエリを自動生成する
 - 複数のDBMS(データベースエンジン)に同一のデータ・クエリペアを入力して, 結果が一致しないケースを探索する.
 - 内部的にはクエリからグラフを構築し**グラフ埋め込み**を使って類似判定により, **似たクエリの探索を避ける**ことで, **多様なテストクエリを効率的に生成**している.
- 到達点: “**detected 115 bugs within 24 hours**, including 31 bugs in MySQL, 30 in MariaDB, 31 in TiDB, and 23 in PolarDB respectively.”

検出されたバグの例(その1)

```
mysql> CREATE TABLE t0(c0 INT);
mysql> INSERT INTO t0 VALUES(0);

mysql> CREATE TABLE t1(c0 DOUBLE);
mysql> INSERT INTO t1 VALUES('-0');

mysql> SELECT /*+no_hash_join()*/ * FROM t0, t1 WHERE t0.c0 = t1.c0;
+-----+-----+
| c0    | c0    |
+-----+-----+
|      0 |     -0 |
+-----+-----+
1 row in set (0.00 sec)
```

the underlying hash join algorithm asserts that “0” and “-0” are not equal.

```
mysql> SELECT /*+hash_join()*/ * FROM t0, t1 WHERE t0.c0 = t1.c0;
Empty set (0.00 sec)
```

(a) MySQL's incorrect hash join execution.

検出されたバグの例(その2)

```
mysql> CREATE TABLE ta (`id` bigint, PRIMARY KEY (`id`));
mysql> insert into ta values (1801425248110076165);

mysql> CREATE TABLE tb (value varchar(50));
mysql> insert into tb values ("1801425248110076222");

mysql> select value from tb where value in (select id from ta);
Empty set (0.00 sec)

mysql> select value from tb where value in (select /*+no_index(ta)*/
      id from ta);
+-----+
| value                |
+-----+
| 1801425248110076222 |
+-----+
1 row in set (0.00 sec)
```

varchar is converted to *double*,
resulting in data accuracy loss

(b) MySQL's incorrect semi-join execution.

DuckDB Testing Present & Future, DBTest@SIGMOD 2022

12

- Mark Raasveldt, CTO of DuckDB Labs によるDuckDBの試験に関する講演
 - 補足: CWIは, Python, Pandas DataFrame などを開発(monetDBなども開発)
 - DuckDBは分析系のデータベースエンジン(Python から呼び出し可能)

```
import duckdb
duckdb.read_csv('example.csv')           # read a CSV file into a Relation
duckdb.read_parquet('example.parquet')   # read a Parquet file into a Relation
duckdb.read_json('example.json')         # read a JSON file into a Relation

duckdb.sql('SELECT * FROM "example.csv"') # directly query a CSV file
duckdb.sql('SELECT * FROM "example.parquet"') # directly query a Parquet file
duckdb.sql('SELECT * FROM "example.json"') # directly query a JSON file
```

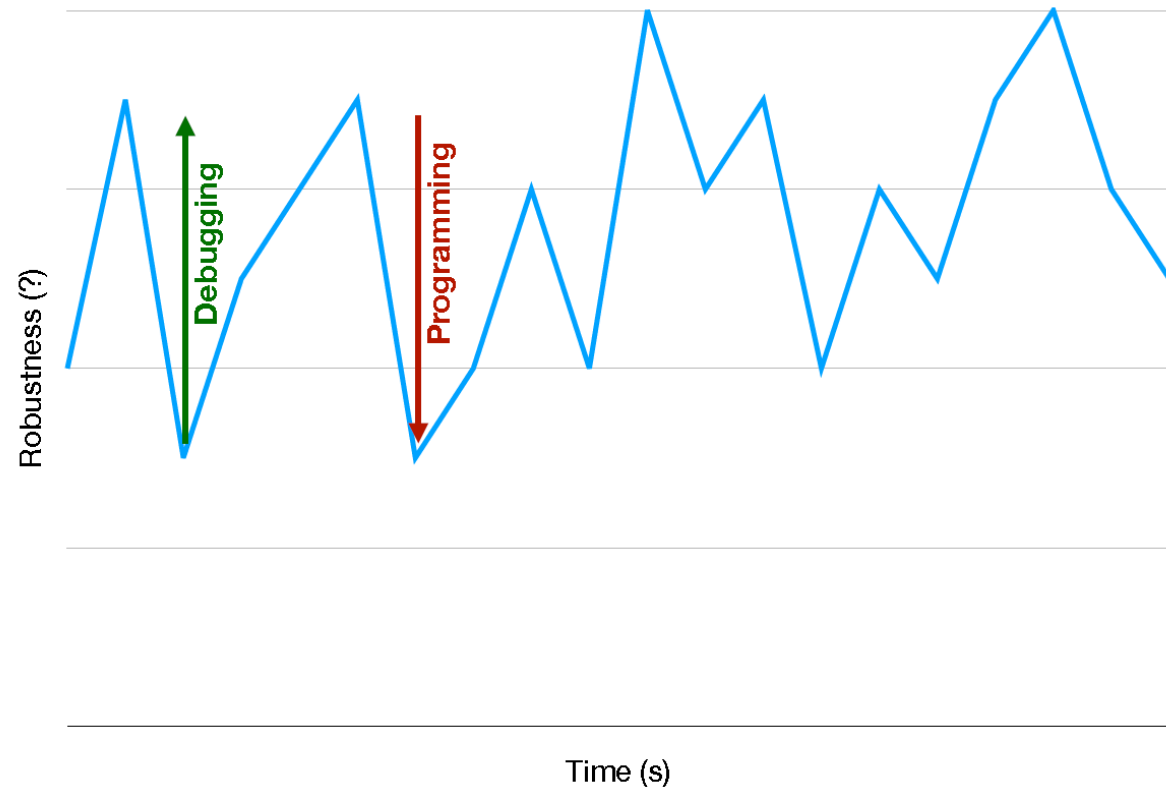
<https://duckdb.org/docs/api/python/overview.html>

DuckDB Testing Present & Future, DBTest@SIGMOD 2022

13

"If debugging is the process of removing software bugs, then programming must be the process of putting them in"

Edsger W. Dijkstra



DuckDB Testing Present & Future, DBTest@SIGMOD 2022

14

- Mark Raasveldt, CTO of DuckDB Labs によるDuckDBの試験に関する講演
 - 補足: CWIは, Python, Pandas DataFrame などを開発(monetDBなども開発)
 - DuckDBは分析系のデータベースエンジン
 - 開発方法: Rely heavily on **test-driven development**
 - Encourage writing many tests
 - Think of as many corner cases as possible
 - **Fuzzing** の登場によって多くのバグを見つけられるようになった
 - ツール: SQLancer, SQLSmith
 - Fuzzers must be run continuously!
 - **データベース特有の問題**
 - クエリ最適化: 最適化する場合/しない場合で同じクエリ結果である必要がある.
 - Performance Regression Testing: クエリ結果だけではなく, 実行性能に関する回帰試験

Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift, SIGMOD2023

15

- Amazon Redshift (分散DBエンジン)における機械学習の導入事例に関する発表. 従来はDB管理者が性能チューニングを行っていた箇所を自動化可能になり, Amazon Redshift に導入されている.
- 要素技術
 - クエリコスト予測: クエリを特徴ベクトルに変換し, XGBoost によりリソース使用量 (実行時間, CPU使用量, メモリ量) 予測
 - クラスタ管理: クエリ実行の並列実行数を制御
- 機械学習技術を実システムに導入する際の知見
 - 予測が外れた際のプロテクションの仕組みが不可欠 (without at least some protection, it is essentially impossible to deploy any ML-enhanced component.)

Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift, SIGMOD2023

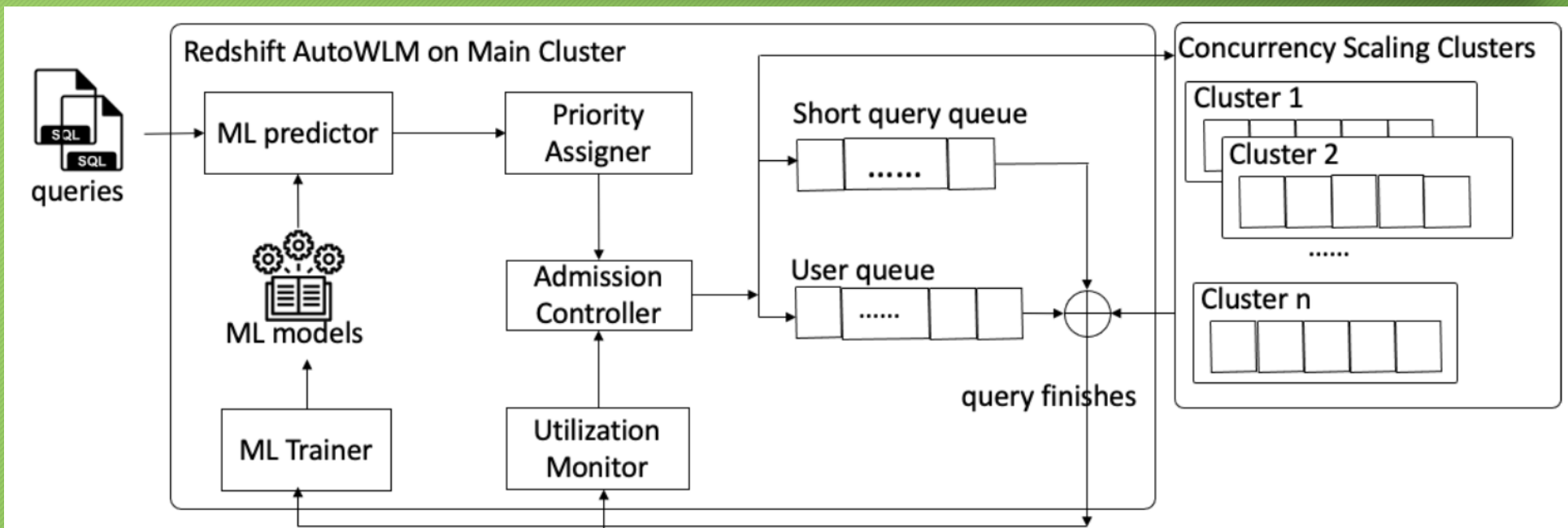


Figure 2: AutoWLM architecture overview. When a query arrives, the ML predictor determines the resource requirements of the query. The priority assigner determines how short the incoming query is to support shortest-job-first scheduling. Given the priority, the admission controller decides if the query should be executed on the main's clusters short query queue, user queue, or sent to a concurrency scaling cluster. Once a query finishes, two things happen: (1) the ML trainer adds the observed query plan and latency to the training set, and may update the ML predictor, and (2) the utilization monitor determines if the cluster's resources are under- or over-subscribed, telling the admission controller to let in more/fewer queries.

Journey of Migrating Millions of Queries on The Cloud, DBTest 2022

17

- トレジャーデータとの共同研究
- 課題感
 - ソフトウェアスタックが複数のOSSで構成され、OSSのバージョンアップの際に回帰試験が必要.
 - チャレンジ: 大規模DBに対して膨大なクエリ群(1.5million/day)の回帰試験の効率的な実施
- 貢献点:
 - クエリの特徴量を設計しクエリ群をクラスタ化. クラスタの代表クエリを優先的に試験(補足: 試験の効率化とカバレッジにはトレードオフがある)
 - DBとクエリ結果を秘匿するためクエリ結果はハッシュ化して試験を実施

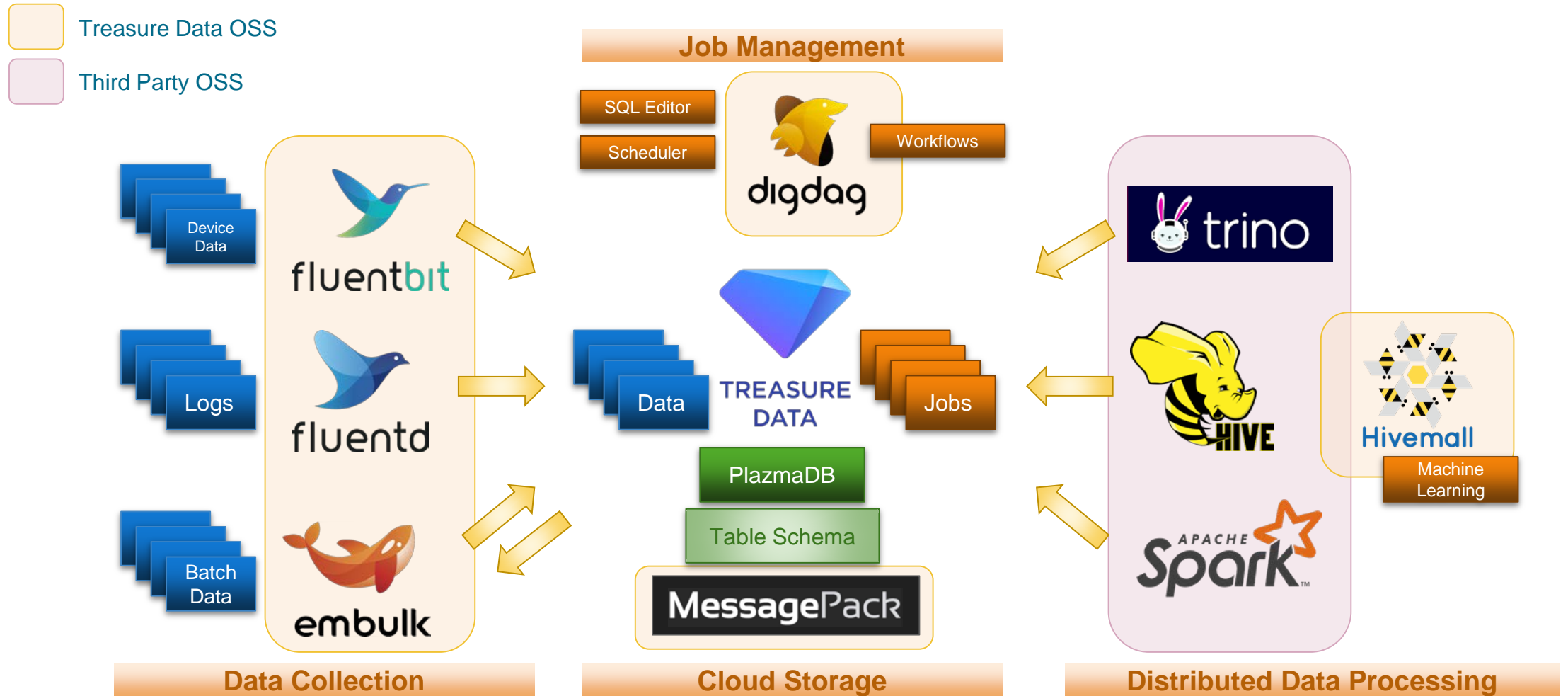
Journey of Migrating Millions of Queries on The Cloud

Taro L. Saito, Naoki Takezoe, Yukihiro Okada, Takako Shimamoto, Dongmin Yu, Suprith Chandrashekharachar, Kai Sasaki, Shohei Okumiya, Yan Wang, Takashi Kurihara, Ryu Kobayashi, Keisuke Suzuki, Zhenghong Yang, Makoto Onizuka

DBTest '22 on June 17th

Treasure Data

is an enterprise customer data platform (CDP) on the cloud



Upgrading query engine is always tough

· Various customer use cases

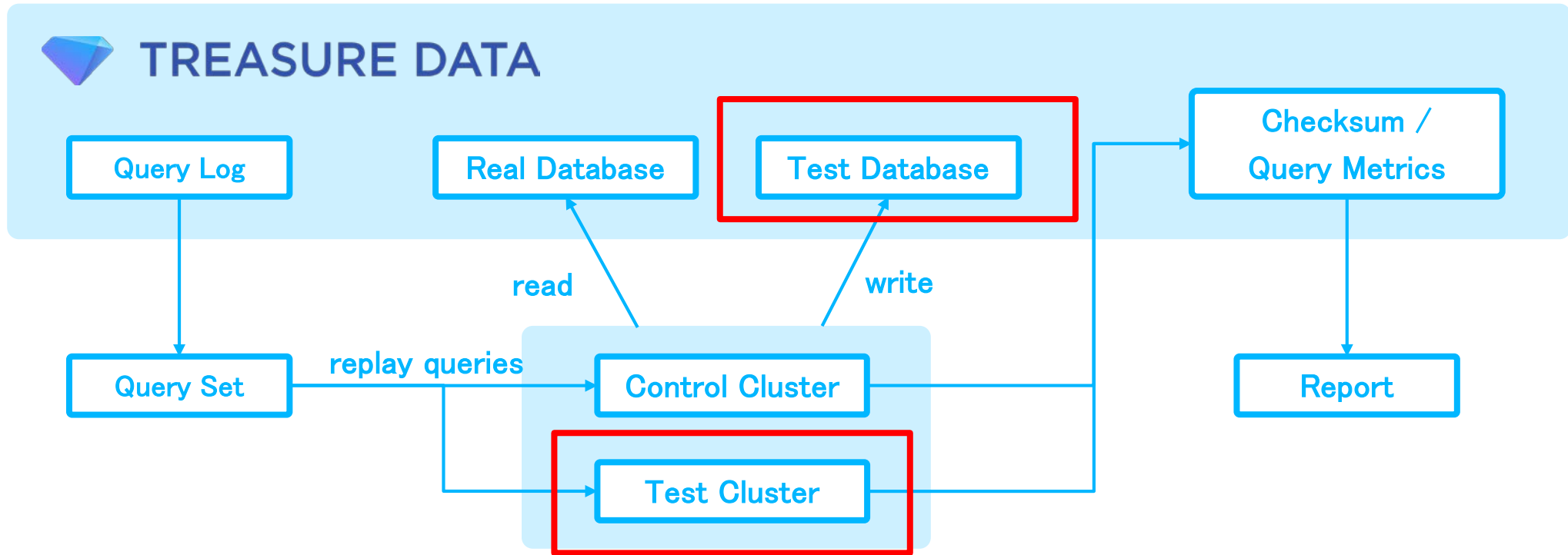
- There are many edge cases in queries, data, and combination of both
 - **General benchmark and test cases are not enough**
- Need to minimize customer frustration caused by upgrading
 - Keep backward compatibility as much as possible
 - **Notify customers of incompatible queries and how to fix them in advance** if we will break compatibility

· Activeness of OSS development

- In particular, **Trino development is super active**
 - Monthly or more frequent release with hundreds of commits
 - No stable versions
- But **staying at the same version so long is also painful**
 - Cannot use new features and optimizations unless backporting
 - Backporting will get harder over time

Query simulator

Test using production data and queries with security and safety



- **Security:** Don't show customer data and query results
- **Safety:** Don't cause any side-effect on customer data

Challenges in query simulation

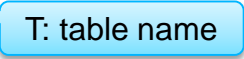
- **Query simulation takes very long time**
 - Very large number of queries need to be tested (1.5 million SQL queries every day)
 - Not only time, but also cost of test clusters
 - **We need to make query simulation faster**
- **Result verification is not straightforward**
 - Many false positives and duplications
 - Result analysis tends to rely on personal knowledge
 - **We need to make result verification easier**

How we can make query simulation faster?

- Reduce the number of queries by **clustering by query signature**
- Reduce the amount of data by **narrowing table scan ranges**
- Test only specific queries (by period, running time, query type, etc)

Clustering queries by query signature

Reduce 90% of queries a day need to be tested

Query signature	Corresponding SQL statements
S(T) 	SELECT ... FROM ...
S[*](T)	SELECT * FROM ... (select all columns)
G(S(T))	SELECT ... FROM ... GROUP BY
S(LJ(T, T))	SELECT ... FROM .. LEFT JOIN ...
WS[A(a,S(T))]	WITH a AS SELECT .. (define aliases to queries)
O(S(T))	SELECT ... ORDER BY
CT(S(T))	CREATE TABLE AS SELECT ...
I(S(T))	INSERT INTO ... SELECT ...
E(S(T))	SELECT distinct ... FROM ... (duplicate elimination)
U(S(T),S(T))	SELECT ... UNION ALL SELECT ...

Reporting for easier result verification

- List problematic queries

- Differences in query results, errors, performance, resource usage, scan ranges, worker distribution, etc

- Exclude uncheckable queries

- Non-deterministic queries

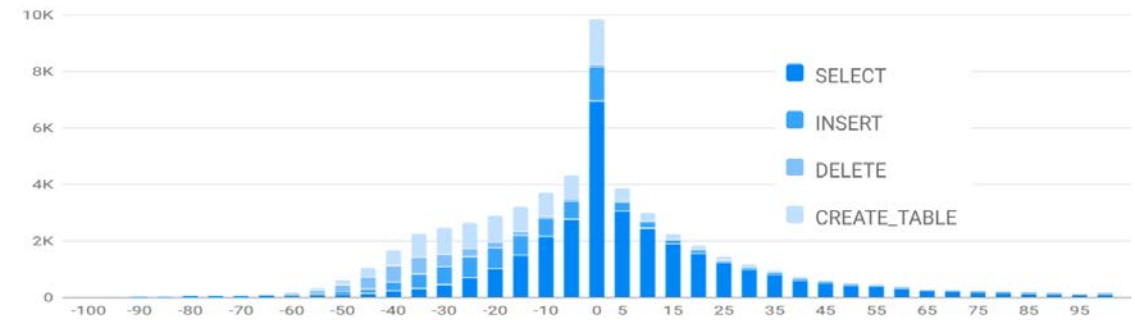
- Failed queries with the same (or similar) error on both versions

- Finally, check remaining queries by human

- Reduced more than 90% of queries need to be checked

- In addition, suggest potential cause of different results

Delay rate -100% ~ +100%

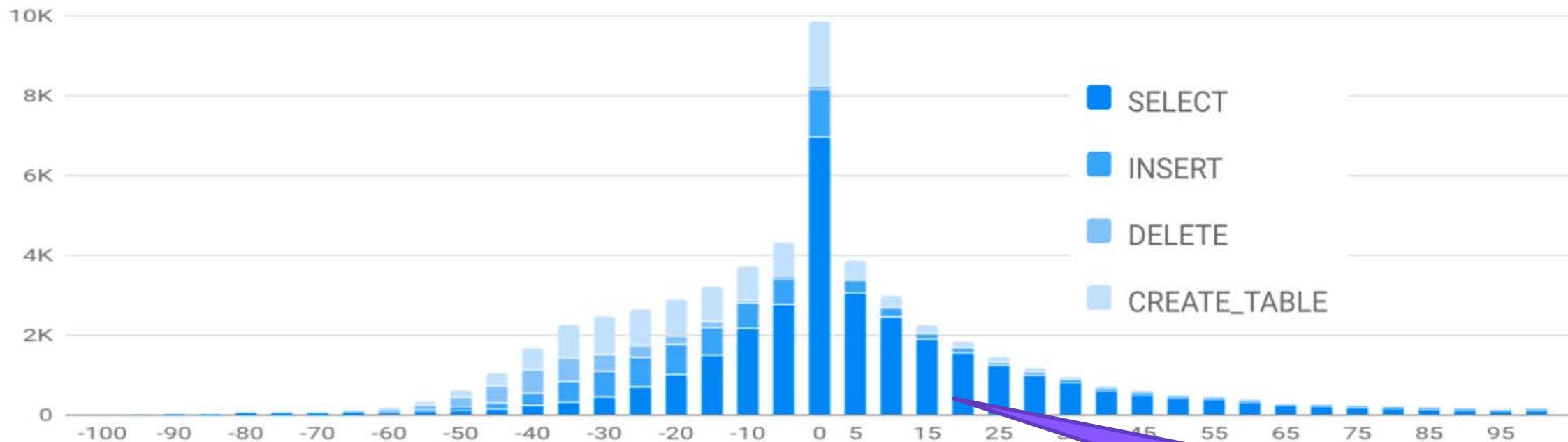


Slow queries

account_id	job_id	query_type	query_id	elapsed_time	query_id	elapsed_time	delay
			control	control	test	test	
		CREATE_TABLE	20211209_173003_24364_hvapa	2.77m	20211209_173004_22041_s7b64	13.11m	10.34m (+372%)
		SELECT	20211210_050226_27983_hvapa	1.10m	20211210_050228_26133_s7b64	4.94m	3.85m (+350%)

Reporting for easier result verification

Delay rate -100% ~ +100%



Slow queries

account_id	job_id	query_type	query_id	elapsed_time	query_id	elapsed_time	delay
			control	control	test	test	
		CREATE_TABLE	20211209_173003_24364_hvapa	2.77m	20211209_173004_22041_s7b64	13.11m	10.34m (+372%)
		SELECT	20211210_050226_27983_hvapa	1.10m	20211210_050228_26133_s7b64	4.94m	3.85m (+350%)

Assistance tools for investigation

trino-compatibility-checker

<https://github.com/takezoe/trino-compatibility-checker>

Run the same query on multiple versions of Trino using docker and compare query results to identify the version that introduced the incompatibility

```
✓ 317: Right(37a6259cc0c1dae299a7866489dff0bd)
✗ 350: Left(java.sql.SQLException: Query failed (#20210526_154140_00004_yzz4q): Multiple entries with
same key: @38f546e: null=expr and @38f546e: null=expr)
✓ 334: Right(37a6259cc0c1dae299a7866489dff0bd)
✗ 342: Left(java.sql.SQLException: Query failed (#20210526_154251_00003_2km75): Multiple entries with
same key: @63f11e0f: null=expr and @63f11e0f: null=expr)
✓ 338: Right(37a6259cc0c1dae299a7866489dff0bd)
✗ 340: Left(java.sql.SQLException: Query failed (#20210526_154338_00002_2xtvz): Multiple entries with
same key: @76615946: null=expr and @76615946: null=expr)
✗ 339: Left(java.sql.SQLException: Query failed (#20210526_154358_00002_jdnni): Multiple entries with
same key: @4aca599d: null=expr and @4aca599d: null=expr)
```


Trino bugs found by query simulation

- [#8027](#) 'Multiple entries with same key' error on duplicated grouping of literal values
- [#19764](#) Missing shallowEquals() implementation for SampledRelation
- [#10861](#) Query fails if IS NOT NULL is used for information_schema
- [#10937](#) Predicate push down doesn't work outside the scope of sub query
- [#11259](#) TRY should handle invalid value error in cast VARCHAR as TIMESTAMP
- [#12199](#) Fix query planning failure on multiple subqueries due to IllegalStateException at ScopeAware.scopeAwareComparison()

Also, we found many bugs in our target version that had been already fixed in the latest version so we could backported them

Future work

- **More support for investigation**
 - After finding bugs, compatibility/performance issues, **the root cause investigation is still tough**
 - Some tools or automations for drill-down investigation will be helpful
- **More efficient query simulation**
 - **Better workload compression** (e.g. exclude more queries that won't improve the test coverage)
 - Isolate simulation traffic from other production components (e.g. pseudo reproduction of real data by synthetic data)

前半:DBMS試験の研究課題

- regression test の効率化: テストケースのコンパクト化, データベースのコンパクト化
 - クエリ量も膨大でDBも規模が膨大. これまでの研究ではDBのコンパクト化とクエリ(テストケース)のコンパクト化が独立に取り組まれているが, これらは双対の関係にあるので, 同時にコンパクト化することが可能と考えられる.
- 機械学習の longtail (corner case)のデバッグ支援および fallback の仕組み
 - 例: 99 percentile の性能劣化の原因の究明が難しい. XGBoost などのホワイトボックスな学習器を使って, 主要な特徴量を解明する等.

後半: グラフ深層学習

35

1. クラウドにおけるデータ管理

- データベースエンジン(DBMS)における試験に関する研究事例
- クラウドスケールのDBMS試験の取り組み

2. グラフ深層学習

- グラフ深層学習の応用事例, ICSE2022 での応用事例
- グラフ深層学習(表現学習)について
- グラフ深層学習技術を評価するベンチマーク

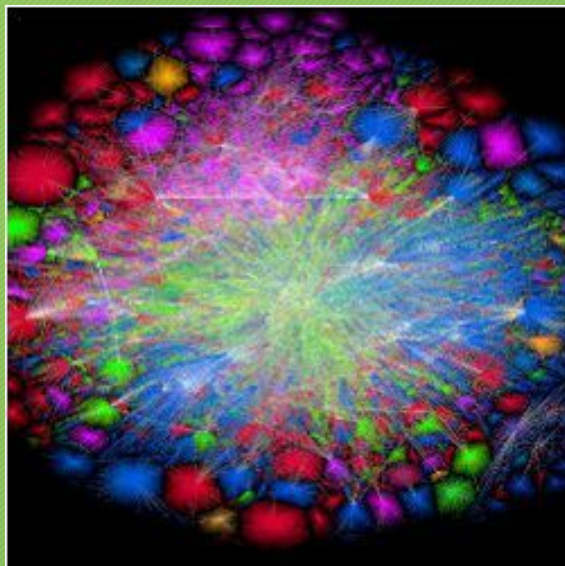
1. グラフ分析の応用例

37

大規模グラフデータの例

38

- Web graph: 100億ページ
- Social graph: 30億ユーザ (Facebook)
- User-item graph: ユーザ数 + 商品数で数億 (Amazon.com)



グラフ深層学習の応用例

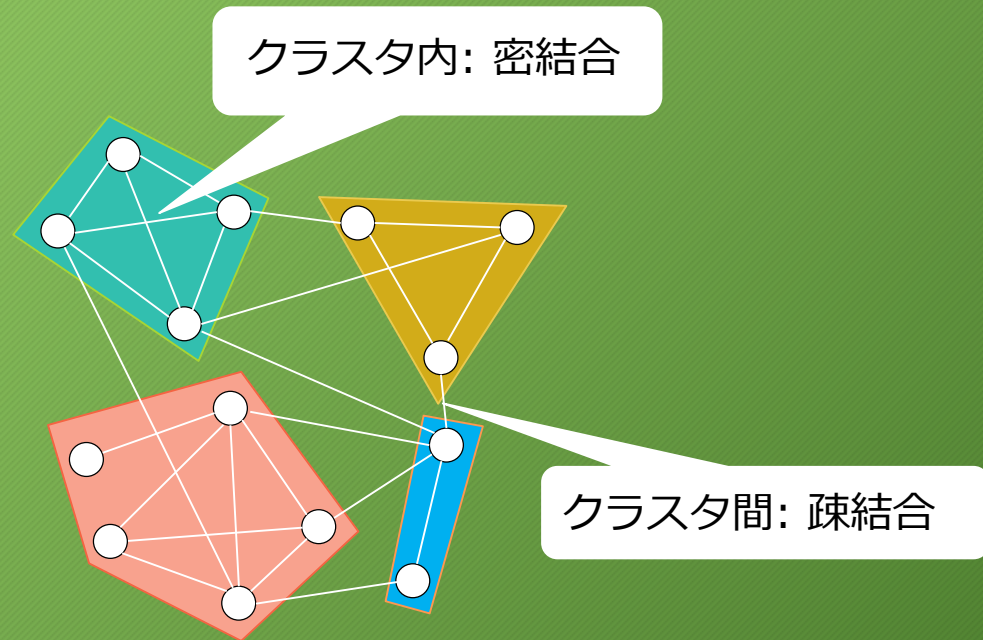
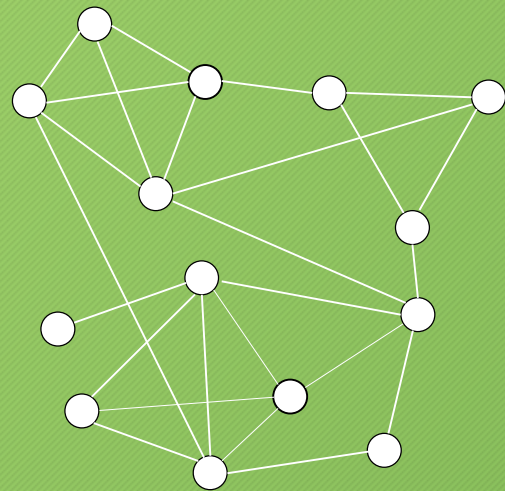
39

- クラスタリング
- 分類
- link予測
- ノードの中心性の計算: 次数中心性, PageRank, 媒介中心性

グラフクラスタリング

40

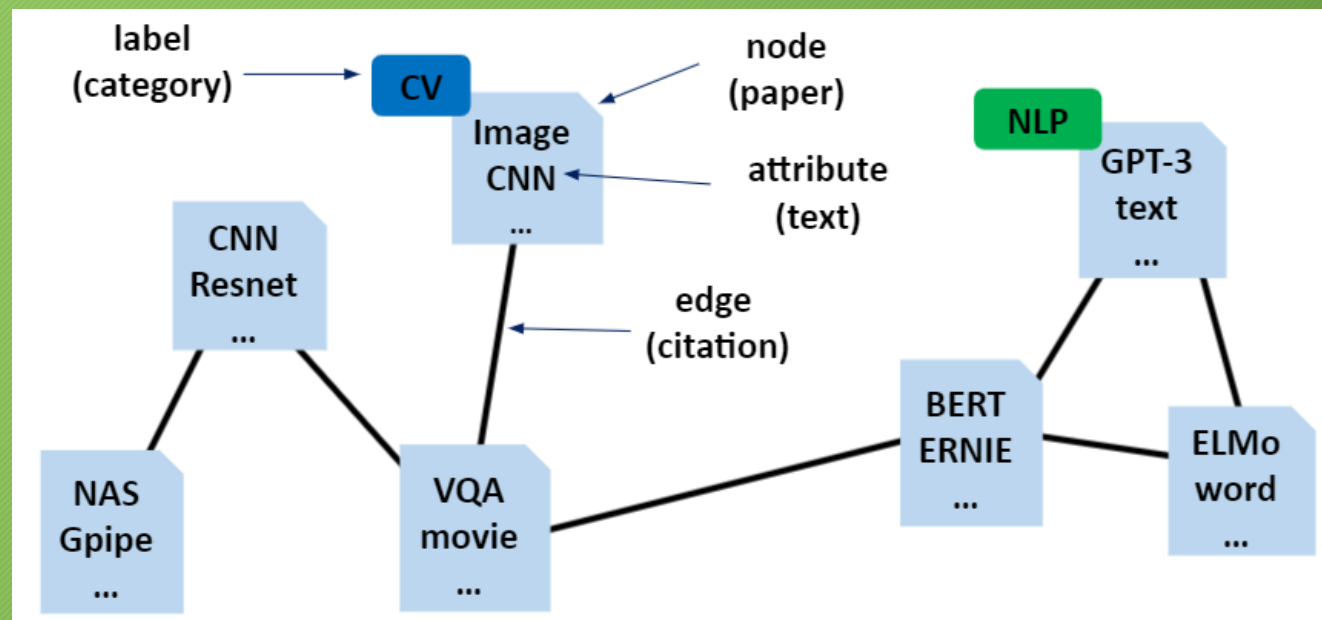
- 構造的に近隣 and/or 属性的に類似の観点から、ノード群をクラスタ化
- 構造: コミュニティ抽出



ノード分類

41

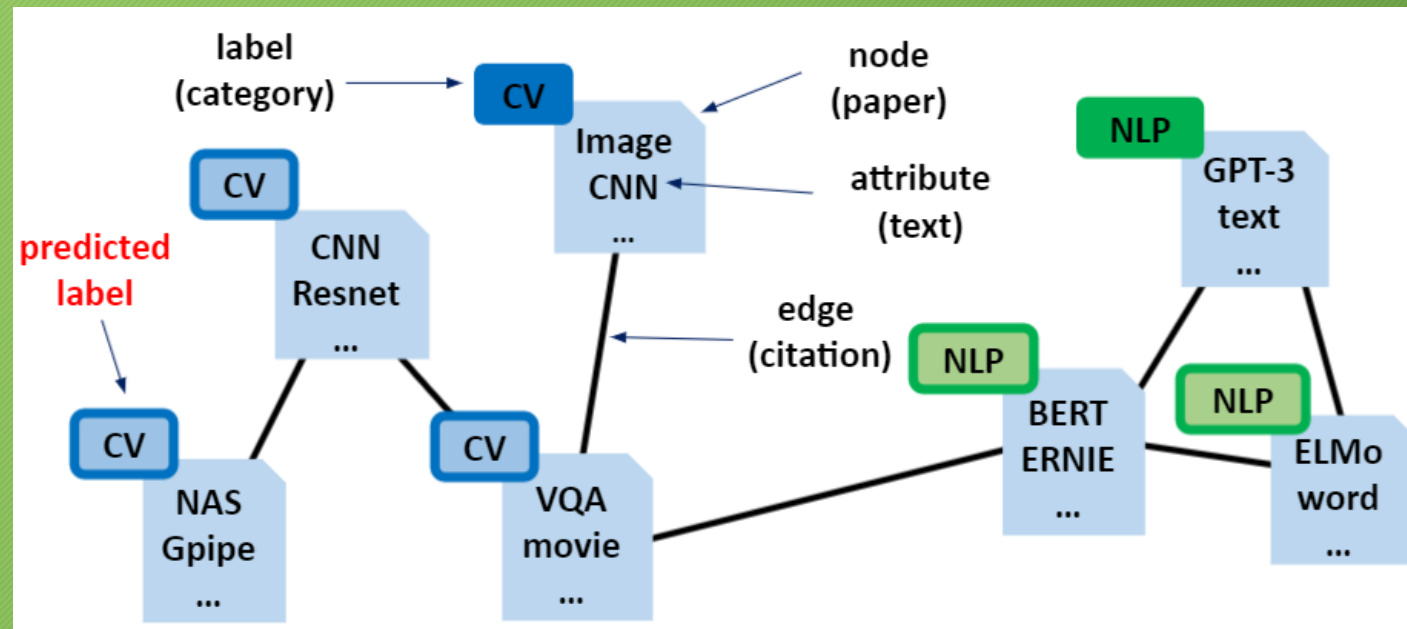
- グラフの構造・属性と一部の正解ノードラベルを入力として、未知のノードラベルを推定するタスク
 - 隣接ノードの情報を活用することで、属性だけの分類よりも精度を高められる



ノード分類

42

- グラフの構造・属性と一部の正解ノードラベルを入力として、未知のノードラベルを推定するタスク
 - 隣接ノードの情報を活用することで、属性だけの分類よりも精度を高められる



リンク予測

43

<https://lab.pasona.co.jp/data-operation/skill/788/>

- リンクが生じる可能性高いノードペアを見つける
- 応用例
 - ソーシャルグラフでの友人推薦
 - タンパク質相互作用



グラフ分析に関する取り組み

44

- クラスタリング: Modularity [AAAI'13], SCAN++ [VLDB'15], PPNMF [GEM'19]
- 分類: ANEPN [ECLM/PKDD'21], compiler [ECML/PKDD'22]
- 中心性計算: top-k PPR [AAAI'13, SIGMOD'13, VLDB'12, KDD'12], SimRank [ICDE'13]
- 評価ベンチマーク [NuerIPS'22, ECML/PKDD'22], グラフ生成器 [Information Systems'23, ASONAM'21]
- 時系列グラフ分析 [ACL'23]
- 分散クエリ最適化 [VLDB'14]
- グラフ処理高速化: Graph ordering [IPDPS'16], Graph パーティション分割 [DSE'17]

ICSE 2022 (technical track)でグラフ学習に関連する研究

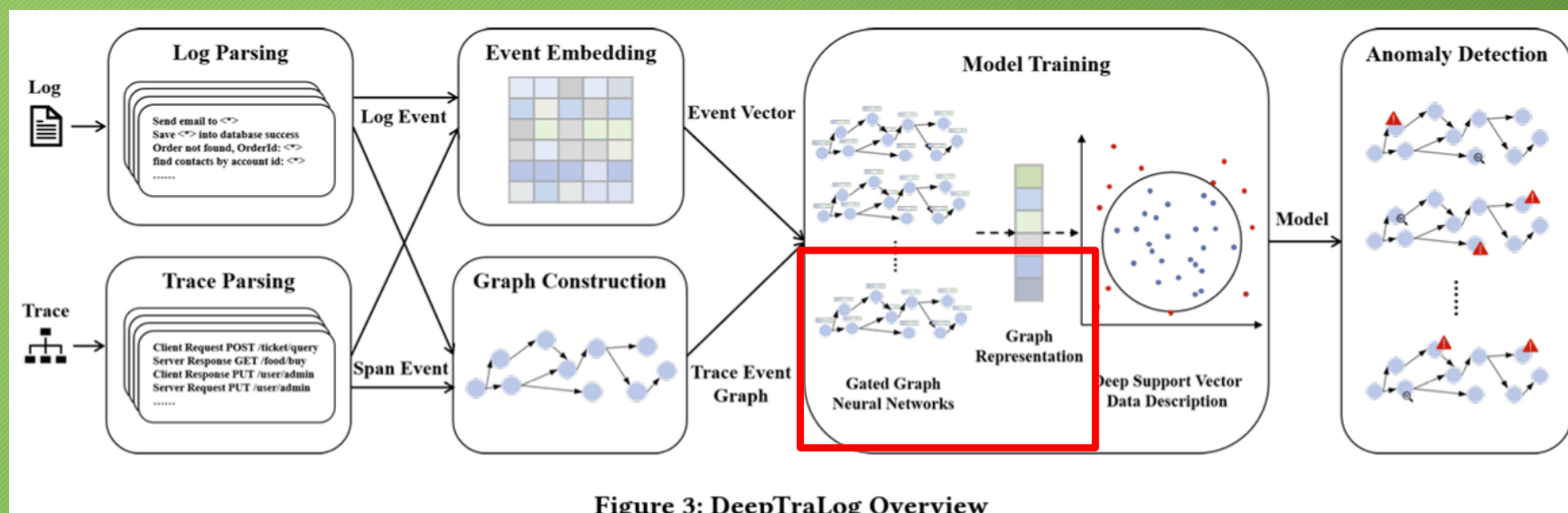
45

- “graph” の文字をタイトルに含む論文10件
- その中でもグラフ深層学習を利用する研究は3件(+1件は適用可能そう)
 1. [DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning](#), Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, Dongmei Zhang
 2. [FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation](#), Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, Dan Hao
 3. [MVD: Memory-related Vulnerability Detection Based on Flow-Sensitive Graph Neural Networks](#), Sicong Cao, Xiaobing Sun, Lili Bo, Rongxin Wu, Bin Li, Chuanqi Tao
 4. [CodeFill: Multi-token Code Completion by Jointly Learning from Structure and Naming Sequences](#), Maliheh Izadi, Roberta Gismondi, Georgios Gousios
- AST (abstract syntax tree) などのグラフ構造を用いて特徴量を表現し、下流工程のタスク(不正検知, 文書生成, コード補完)を学習

DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning, ICSE 2022

46

- 課題: microservice anomaly detection
- アイデア: 従来は簡易な service invocation sequence を利用していたが, invocation hierarchy and parallel/asynchronous invocationsを扱うよう拡張した.
- F1-scoreで0.37の改善を達成 (precision (0.93) and recall (0.97)).



FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation, ICSE 2022

- 課題: git commit 時に commit message を自動作成する研究
- アイデア: コードの変更差分をグラフにより表現することを特徴としていて, GNNエンコーダー+transformerデコーダーによりcommit message を生成
- BLEU, ROUGE-L, and METEOR指標で従来技術より高性能

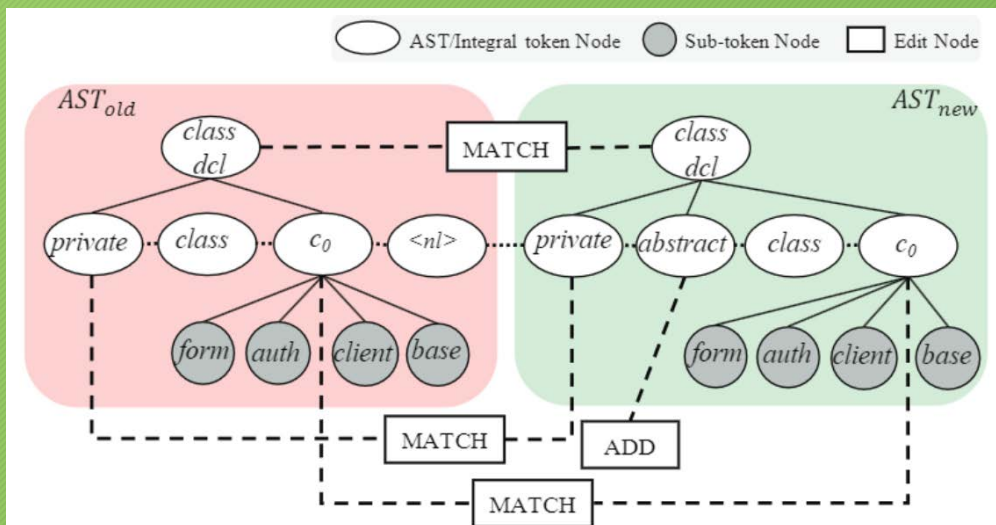


Figure 8: Graph_{final}: fine-grained representation for code changes

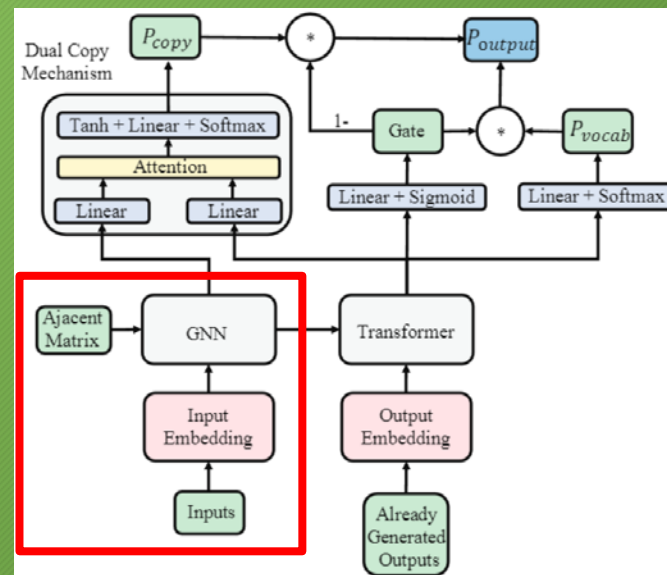


Figure 9: Architecture of the proposed model

MVD: Memory-related Vulnerability Detection Based on Flow-Sensitive Graph Neural Networks, ICSE 2022

- 課題: Memory-related vulnerabilities
- アイデア: statement-level memory-related vulnerability detection approach based on flow-sensitive graph neural networks (FS-GNN)

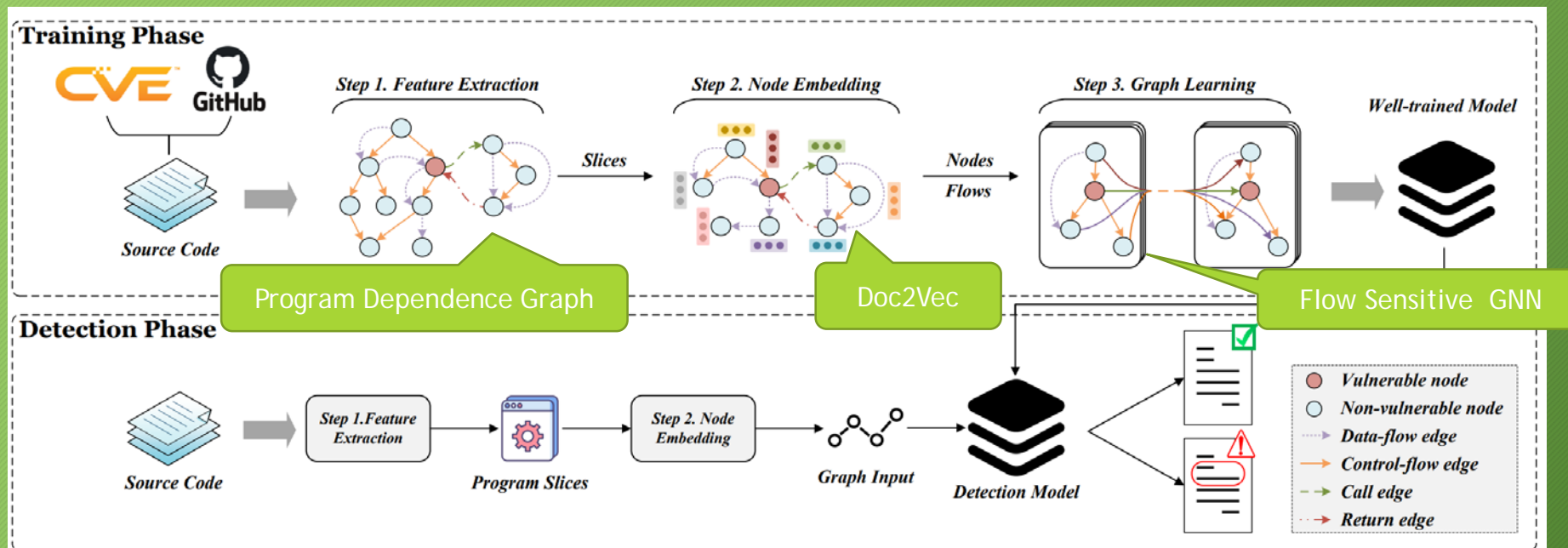


Figure 2: Overview of MVD

2. グラフ深層学習(表現学習)

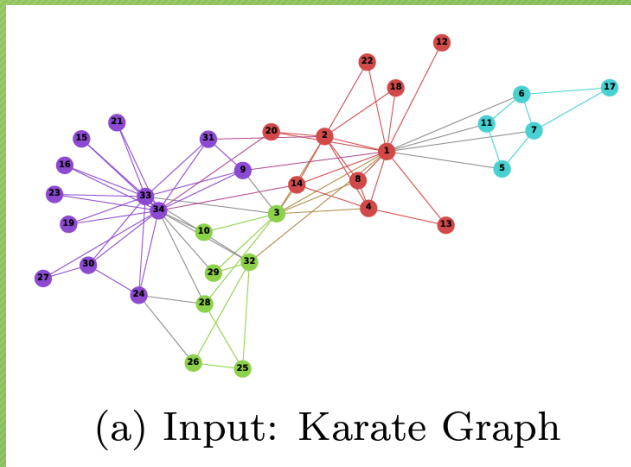
49

グラフ深層学習(GNN, 表現学習)

- グラフマイニング応用: クラスタリング, 分類, link予測
- 基本となる考え方:
 - 2つの**ノード同士の類似度**が高い場合に, 同一クラスタに所属, link発生を予測
 - ノード同士の類似性は**ノードの特徴量**に基づいて計算
 - **構造: 局所構造(microscopic), 大域構造(macroscopic)**を活用して特徴量を計算
- 技術の変遷
 - 多様な分析タスク向け: ノードの特徴量を計算し後工程の分析で利用:
 - 分類タスク向け特化してノードの特徴量を計算: GCN, GAT, GCNの拡張(GIN, H2GCN, GPRGNN, LinkX, ...)
- グラフ深層学習のベンチマークフレームワーク

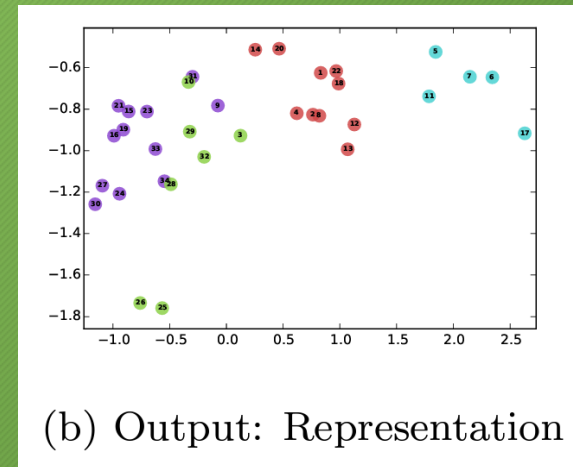
表現学習/node embedding

- 表現学習: ノードの特徴量を得る方法
 - 構造情報 and/or 属性情報を利用して(多次元の)特徴量を得る
 - メリット: 多次元データ化すれば, クラスタリング・分類など既存の技術が適用可能
 - 特徴: 隣接ノード同士は, 多次元空間でも近傍に埋め込み(以下, DeepWalk の例)



グラフ空間

表現学習/埋め込み

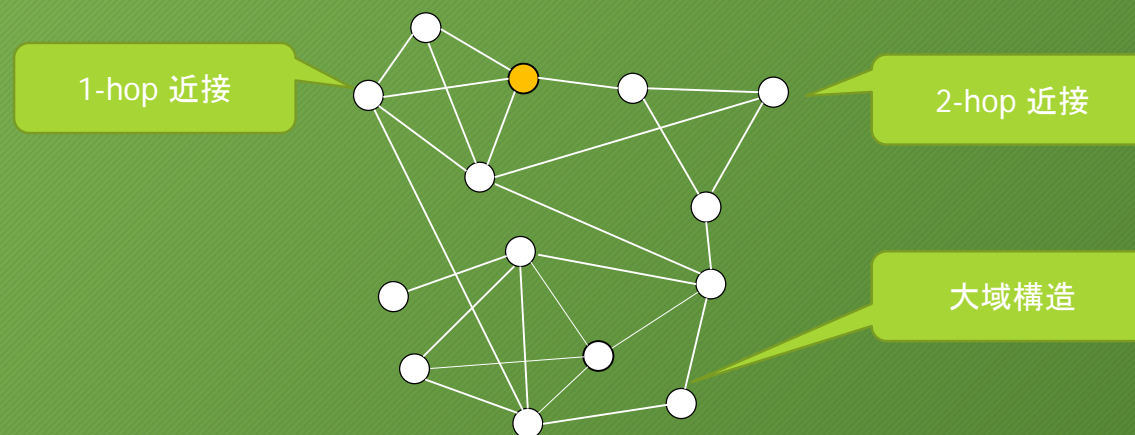


多次元空間

局所構造(microscopic), 大域構造(macroscopic)

52

- 各ノードの特徴量を計算する際, 局所構造(1-hop, 2-hop以内の近傍ノード)と大域構造(多hop先のノード)の特徴を反映することが有効
 - **局所構造**の例: ソーシャルネットワークでは, 友人の特徴(1-hop), 更には友人の友人の特徴(2-hop)を活用して, ノードの特徴量を計算することが有用.
 - **大域構造**の例: 同一クラスには多くのノードが含まれるため, これらの離れたノードの特性を活用して, ノードの特徴量を計算することが有用(特に教師ラベルが少ない場合).



局所構造(microscopic), 大域構造(macroscopic)

53

- 各ノードの特徴量を計算する際, 局所構造(1-hop, 2-hop以内の近傍ノード)と大域構造(多hop先のノード)の特徴を反映することが有効
 - **局所構造**の例: ソーシャルネットワークでは, 友人の特徴(1-hop), 更には友人の友人の特徴(2-hop)を活用して, ノードの特徴量を計算することが有用.
 - **大域構造**の例: 同一クラスタには多くのノードが含まれるため, これらの離れたノードの特性を活用して, ノードの特徴量を計算することが有用(特に教師ラベルが少ない場合).
- 既存技術の例
 - 1-hop (1st order) 近接性を活用: **Spectral clustering** 等
 - 2-hop (2nd order) 近接性を活用: SCAN (構造類似度) 等
 - 1-hop+2hop 近接性: SDNE, **GCN**, SEAL 等
 - 局所構造+クラスタ構造: M-NMF (クラスタ指標を正則化としてロスに導入), ANEPN
 - 局所構造+大域構造を活用: **node2vec**, ALaGCN, ANEPN 他

Spectral clustering, NIPS2001

54

- 考え方:

- ノード毎に多次元の特徴量(x_i, x_j, \dots)を計算し, 多次元空間においてノード集合をクラスタリングして結果を得る.
- ノードの特徴量の計算: **1-hop 近接性**を制約(ロス)としてを用いる

$$L(x) = \frac{1}{2} \sum_{i,j} A_{ij} (x_i - x_j)^2$$

隣接行列

次数行列

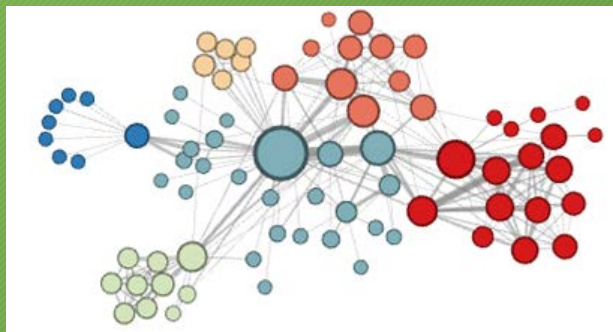
この式は後述するGCNでも出てくる

- 計算方法

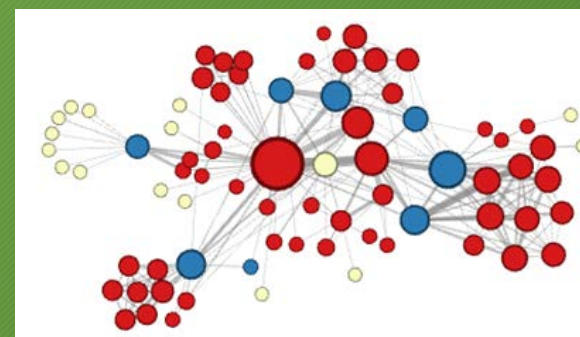
- $L(x)$ の最小化問題はグラフラプラシアン $\mathcal{L}(= D - A)$ を使うと $L(x) = x^T \mathcal{L} x$ と書き換えられ, 最終的に \mathcal{L} の固有値問題 $\mathcal{L}x = \lambda x$ に帰着できる
- 補足: 入力データが多次元の場合, k近傍グラフ等の方法で事前にグラフ化する

- 評価タスク: クラスタリング

- 特徴: 各ノードを起点として, 幅優先と深さ優先を組み合わせた random walk (長さ l のノード系列)を得て, **word2vec (skip-gram)** に入力することで, 該当ノードの特徴量を計算する方法.
 - Random walk パラメータ (p, q) で BFS, DFS の度合いを制御
 - 分析例:



$q=0.5$; DFS 優先設定
コミュニティ構造が得られている



$q=2$; BFS 優先設定
媒介性が高いノード(青)が得られている

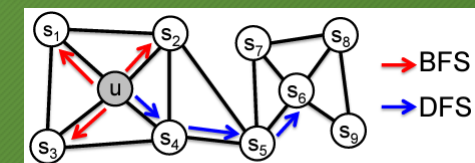


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

- 評価タスク: ノード分類, link 予測

GCN (Graph Convolutional Networks), ICLR 2017

被引用数25,200件

57

- 特徴: 分類タスクに特化

- ロス: 分類ロス(予測ラベルと正解ラベルの誤差) + 1-hop 近接性(但し, f がニューラルネットワークによる関数)

隣接行列

Spectral clustering のロスに f を適用したもの

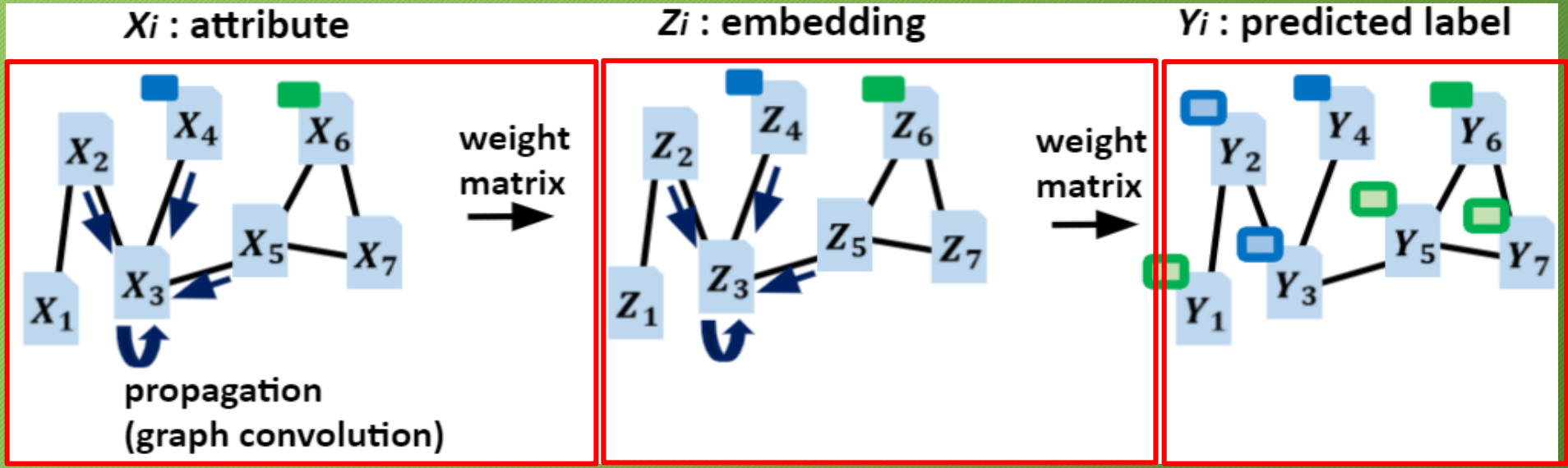
$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(X)^\top \Delta f(X).$$

- 設計: 分類ロス \mathcal{L}_0 (クロスエントロピーロス)を最小化するよう $f(X, A)$ をニューラルネットワークで学習し, ノードの特徴量 X を得る
 - f : 隣接ノードと自ノードの特徴量を集約(convolution)して, 自ノードの特徴量を計算する(この計算を複数回実行する)
 - 一般的には多層レイアから構成されるが, 分類タスクでは2層(つまり 2-hop 近接性までを考慮)が最も高精度

2層レイアのGCNの例

- 隠れ層: $Z = f(X, A) = \text{softmax}(\hat{A} \text{ReLU}(\hat{A}XW^{(0)}) W^{(1)})$.
- 出力層: $\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^F Y_{lf} \ln Z_{lf}$,

$\hat{A} = \tilde{D}^{-1/2} A \tilde{D}^{-1/2}$
 \tilde{D} : 自己ループを追加した隣接行列における次数行列



- 静的グラフ
 - GAT [ICLR 2018]: 注意(attention)機構の導入
 - 何hop先が重要かを判定する技術の導入: H2GCN [NeurIPS 2020], JK-Nets [ICML 2018]
 - 構造情報を活用する技術
 - GIN [ICLR 2019]: グラフ同型構造を識別する能力を有する拡張
 - LINKX [NeurIPS 2021]: 通常のGNNに隣接行列を説明変数に追加
 - SEAL [NeurIPS 2018]: link 発生予測向け
 - CG³ [AAAI 2021]: 対照学習の導入
- 時間グラフ対応: JODIE [KDD 2019], TGAT [ICLR 2020], NAT [NeurIPS 2022]

3. グラフ深層学習の評価ベンチマーク

60

Beyond Real-world Benchmark Datasets: An Empirical Study of Node Classification with GNNs, NeurIPS 2022

61

- 課題感: 膨大なグラフ深層学習の技術が提案されているが, それぞれがチャンピオンデータで評価しているため, 公平な評価が不十分. またデータセットのバリエーションが乏しい問題がある.
- アイデア: グラフの特徴を連続的に変化できる人工グラフ生成器 [Information Systems 2023]を用いて, 最先端のグラフ深層学習の技術の特性と問題点を明らかにした.

Comprehensive Evaluation of Graph Neural Networks

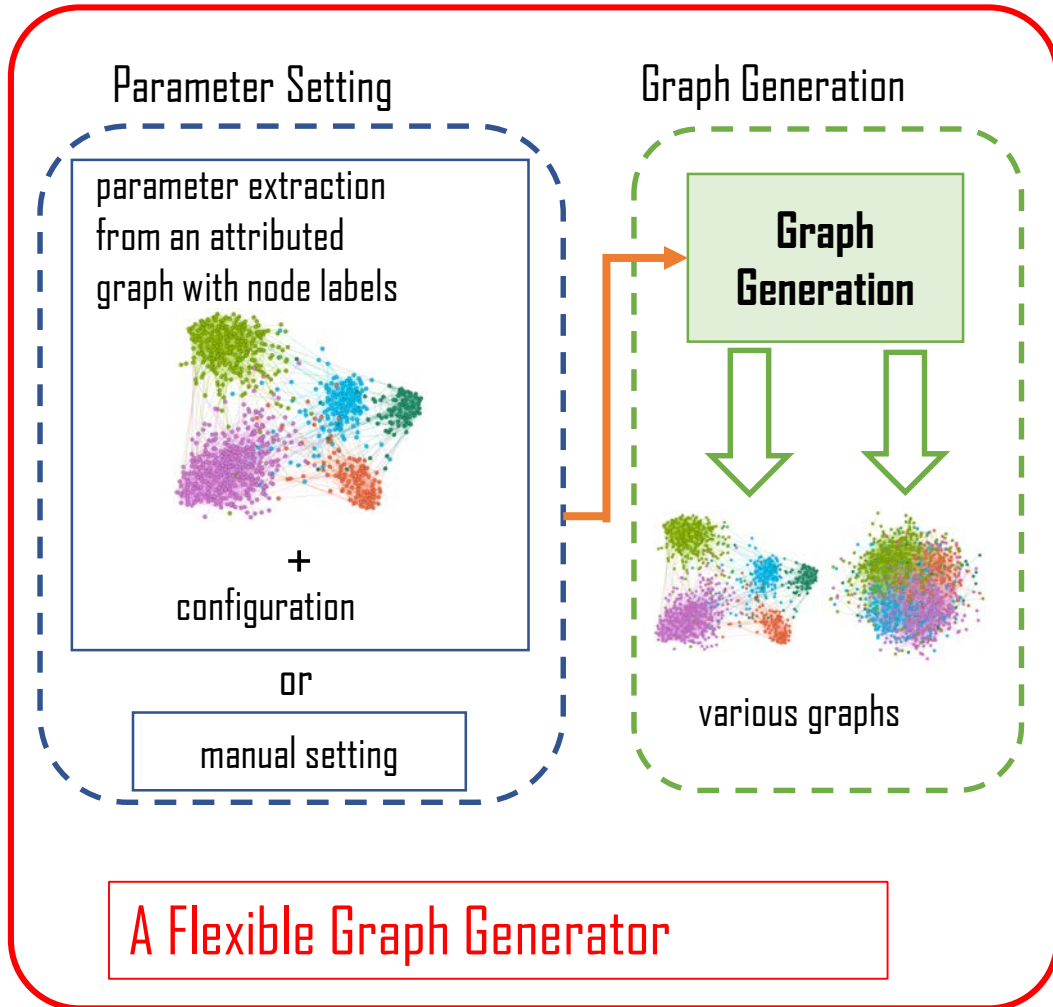
Graph neural networks have attracted broad attention.

However, the **comprehensive evaluation** of the methods is challenging because we have only limited **real-world graphs with node labels**.

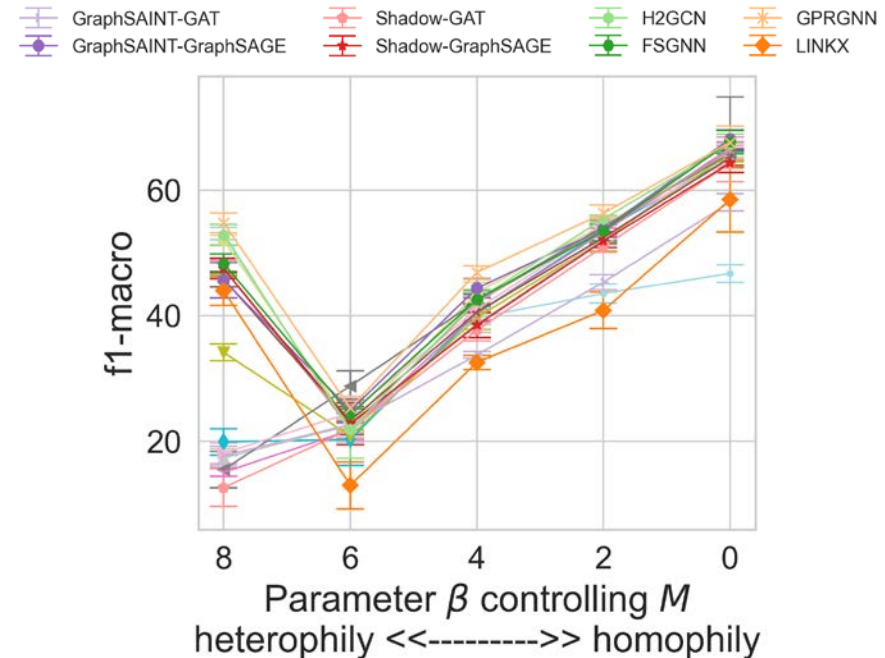


If we had a variety of graphs,
we could assess the methods from many different aspects...

Evaluation Framework with Synthetic Graphs

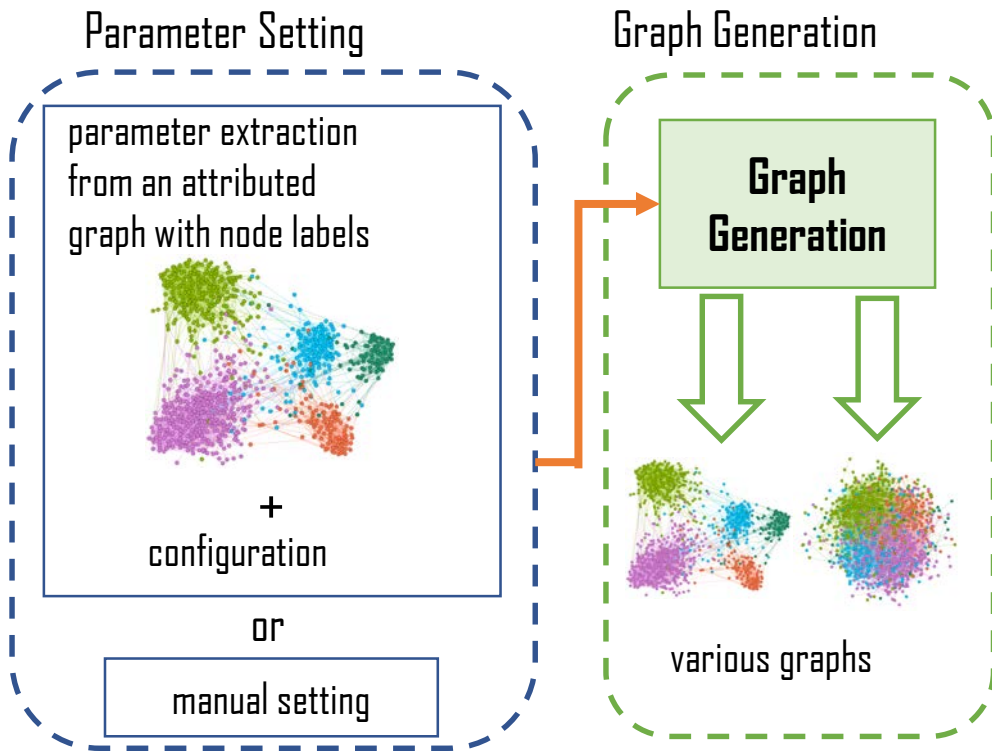


Use case; Evaluating graph neural network models with generated graphs having various characteristics.



An Empirical Study of GNNs

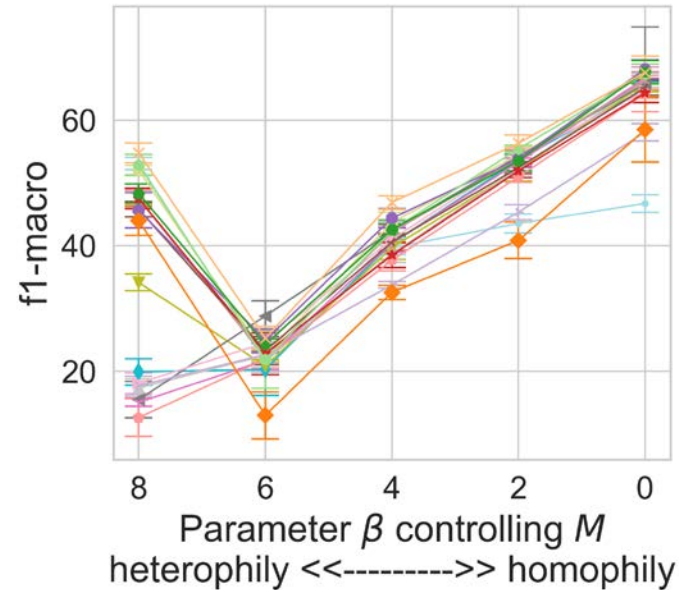
Evaluation Framework with Synthetic Graphs



A Flexible Graph Generator

Use case; Evaluating graph neural network models with generated graphs having various characteristics.

GraphSAINT-GAT, Shadow-GAT, H2GCN, GPRGNN, GraphSAINT-GraphSAGE, Shadow-GraphSAGE, FSGNN, LINKX



An Empirical Study of GNNs

Beyond Real-world Benchmark Datasets: An Empirical Study of Node Classification with GNNs

Seiji Maekawa¹, Koki Noda², Yuya Sasaki¹, Makoto Onizuka¹

¹Osaka University, ²TDAI Lab

[code]



[paper]



Background of Empirical Studies of GNNs

Graph machine learning methods, e.g., GNNs,
have been assessed with **limited benchmark datasets**.

Towards practical use cases of GNNs, researchers and developers need
to deeply understand the **strengths/weaknesses of GNNs from various aspects**.

For extensive experiments, **can we utilize
various synthetic graphs with different characteristics?**

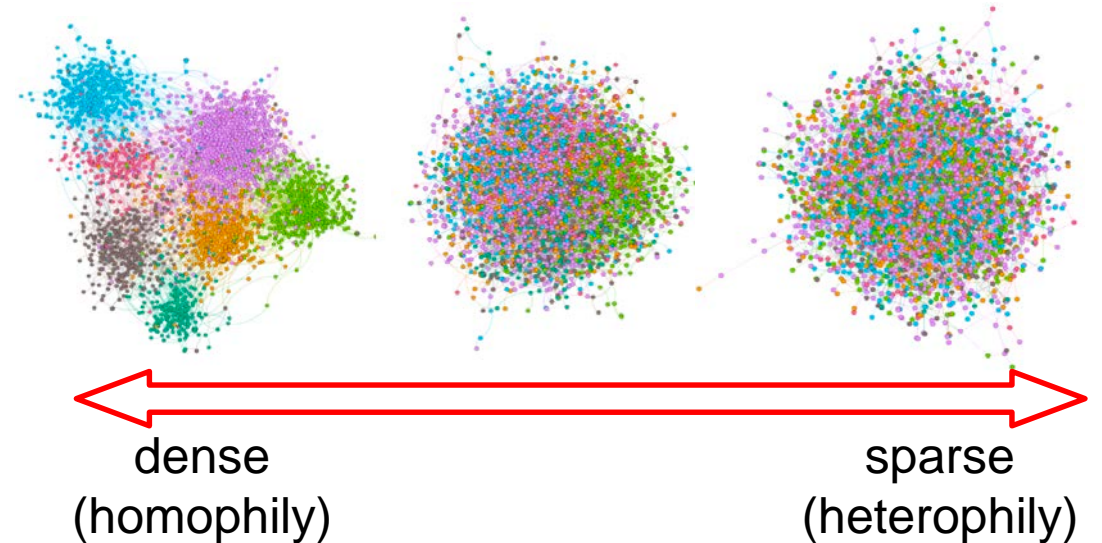
Approach to Comprehensive Evaluations of GNNs

We empirically study the performance of GNNs by using various graphs by synthetically **changing one or a few target characteristic(s)** of graphs,

Ex.1: Class size distributions



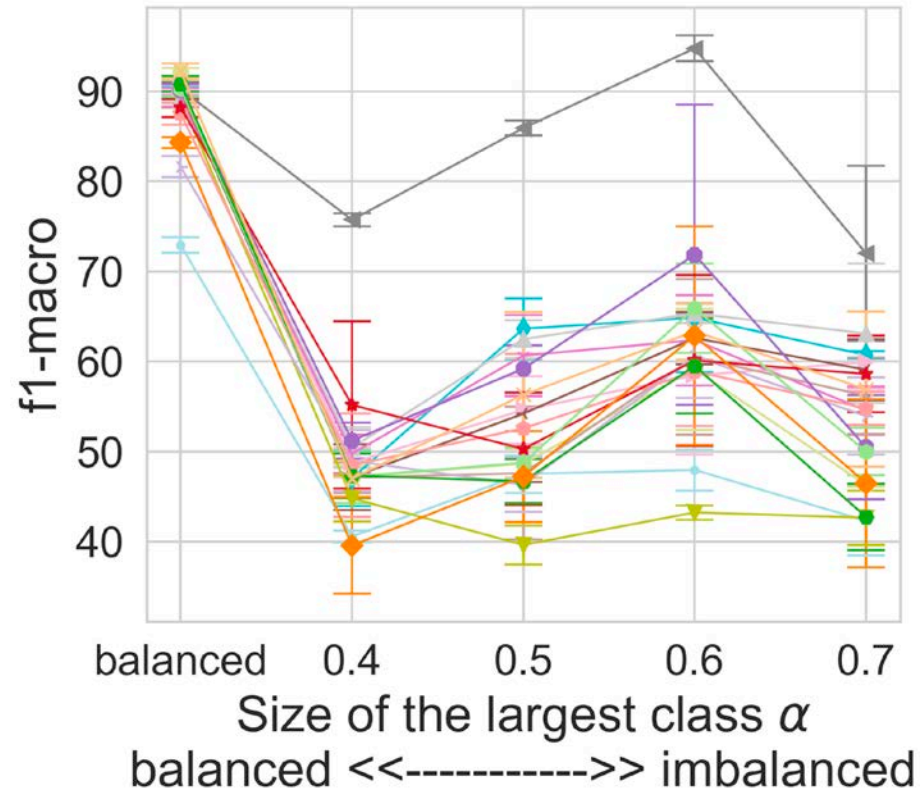
Ex.2: Class connection proportions



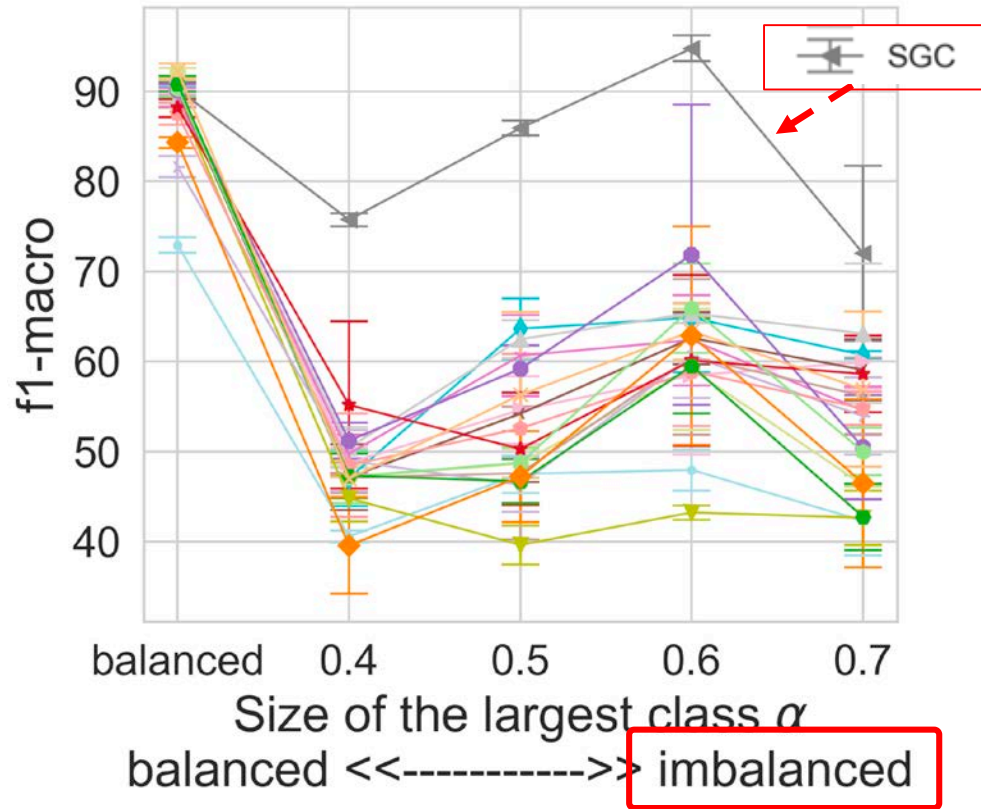
Empirical Study 1: Class Size Distributions



While most existing works use accuracy, we use f1-macro that is more suitable for the imbalanced settings.



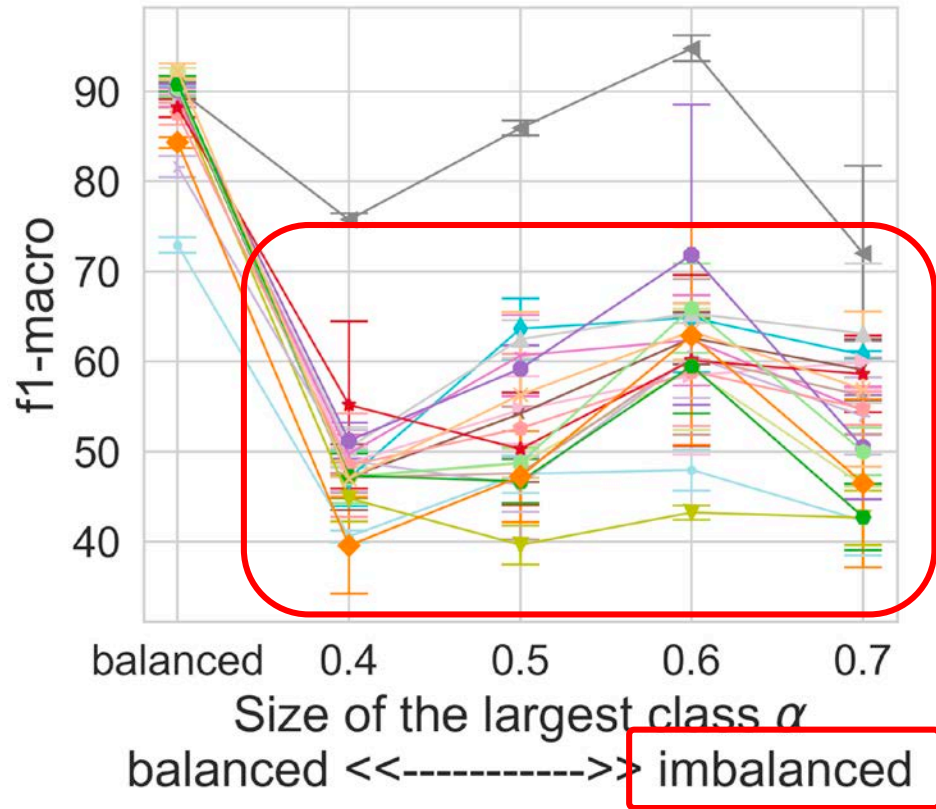
Empirical Study 1: Class Size Distributions



While most existing works use accuracy, we use f1-macro that is more suitable for the imbalanced settings.

Interestingly, a **linear** model (SGC) achieves the **best scores** in the imbalanced settings.

Empirical Study 1: Class Size Distributions

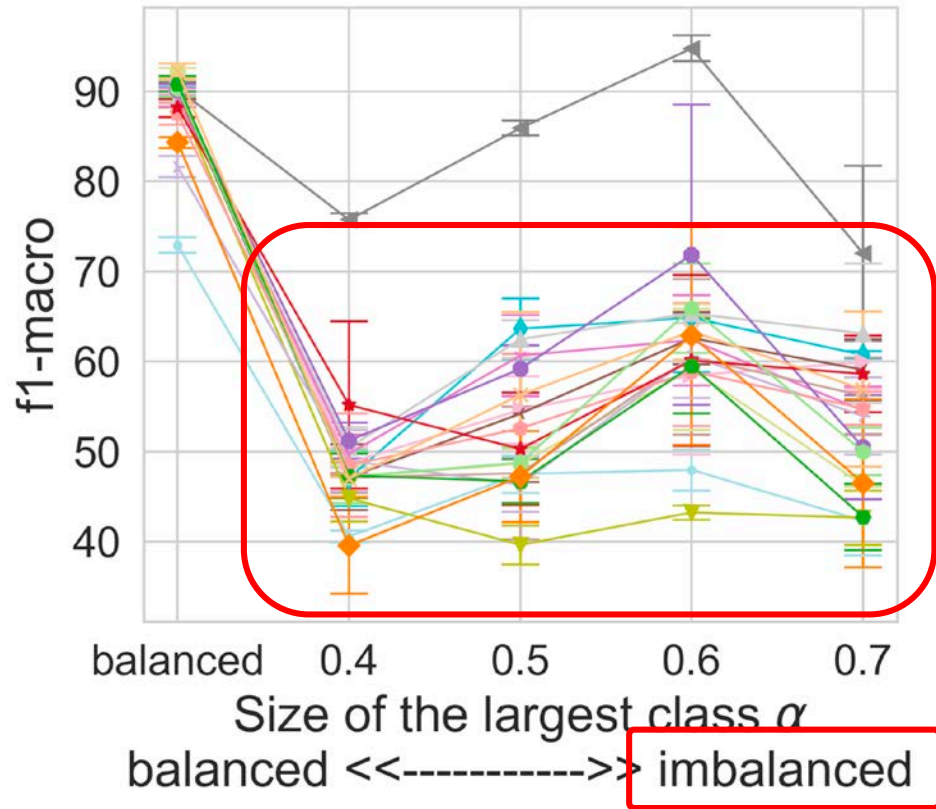


While most existing works use accuracy, we use f1-macro that is more suitable for the imbalanced settings.

Interestingly, a **linear** model (SGC) achieves the **best scores** in the imbalanced settings.

Other complicated GNNs tend to **overfit major classes**.

Empirical Study 1: Class Size Distributions



While most existing works use accuracy, we use f1-macro that is more suitable for the imbalanced settings.

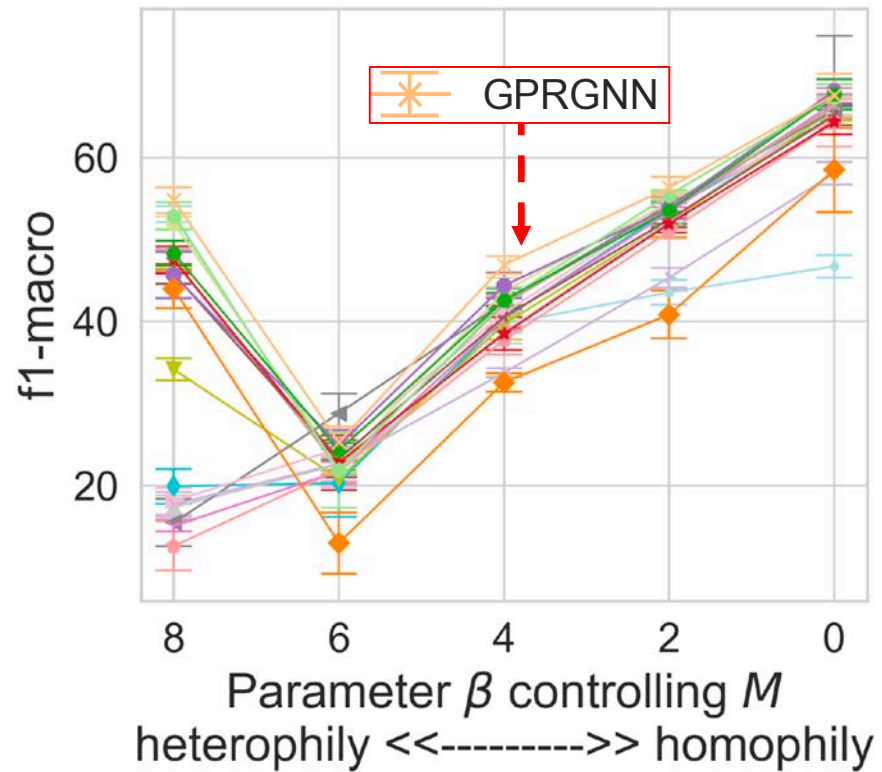
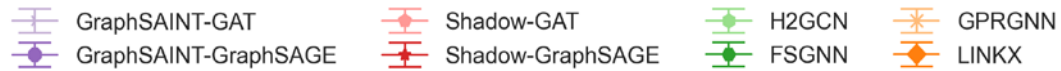
Interestingly, a **linear** model (SGC) achieves the **best scores** in the imbalanced settings.

Other complicated GNNs tend to **overfit major classes**.

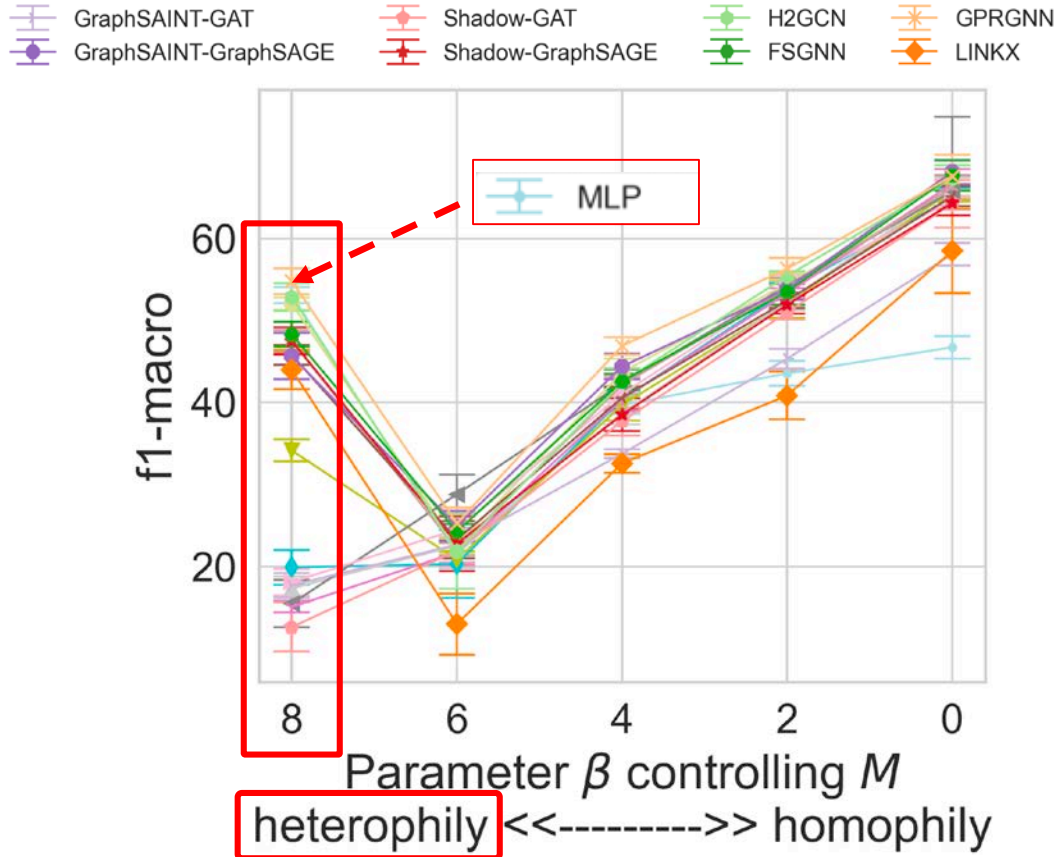
In the imbalanced settings,
a simple method outperforms others!

Empirical Study 2: Class Connection Proportions

A very recent GNN (GPRGNN) achieves the best scores in most cases.



Empirical Study 2: Class Connection Proportions



(classes are sparsely connected)

A very recent GNN (GPRGNN) achieves the best scores in most cases.

In the heterophily setting, a graph-agnostic model (MLP) achieves comparable results to SOTA GNNs.

There is room for performance improvement.

Short summary

We proposed a flexible graph generator, GenCAT,
which can **control the class structure** and **scale well**.

We conducted empirical studies of GNN for node classification and
clarified the **limitations and opportunities** of the SOTA GNNs.

We provide an evaluation framework for future research.

まとめ

76

1. クラウドにおけるデータ管理
 - データベースエンジン(DBMS)における試験に関する研究事例
 - クラウドスケールのDBMS試験の取り組み
2. グラフ深層学習
 - グラフ深層学習の応用事例, ICSE2022 での応用事例
 - グラフ深層学習(表現学習)について
 - グラフ深層学習技術を評価するベンチマーク

前半:DBMS試験の研究課題

78

- regression test の効率化: テストケースのコンパクト化, データベースのコンパクト化
 - クエリ量も膨大でDBも規模が膨大. これまでの研究ではDBのコンパクト化とクエリ(テストケース)のコンパクト化が独立に取り組まれているが, これらは双対の関係にあるので, 同時にコンパクト化することが可能と考えられる.
- 機械学習の longtail (corner case)のデバッグ支援および fallback の仕組み
 - 例: 99 percentile の性能劣化の原因の究明が難しい. XGBoost などのホワイトボックスな学習器を使って, 主要な特徴量を解明する等.

後半:グラフ深層学習のソフトウェア工学への適用

79

- ICSE 2022 論文の傾向: AST (abstract syntax tree) などのグラフ構造を用いて特徴量を表現し, 下流工程のタスク(不正検知, 文書生成, コード補完)を学習
- **注意点:** 単にGNNを適用するだけでうまくいかないことがある. 通常の機械学習と同様に「特徴量設計」が重要.
 - アルゴリズムによって利用する特徴量が異なっている
 - GCN: クラスラベルを正確に推定する+隣接ノードの特徴は似せる. ノードの次数や, 隣接ノードID情報は利用しない. エッジの向きも使わない. -> LINKX や A2DUG (<https://github.com/seijimaekawa/A2DUG>)が良い.
 - グラフの構築方法
 - FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation, ICSE 2022では, git リポジトリにおける変更前後のAST構造を明示的に入力している.