Symmetric Continuous Subgraph Matching with Bidirectional Dynamic Programming

Seunghwan Min
Seoul National University
shmin@theory.snu.ac.kr

Sung Gwan Park
Seoul National University
sgpark@theory.snu.ac.kr

Kunsoo Park*
Seoul National University
kpark@theory.snu.ac.kr

Dora Giammarresi†
Università Roma "Tor Vergata"
giammarr@mat.uniroma2.it

Giuseppe F. Italiano†
LUISS University
gitaliano@luiss.it

Wook-Shin Han*
Pohang University of Science and Technology (POSTECH)
wshan@dblab.postech.ac.kr

Shortcomings of TurboFlux

1. The state-of-the-art algorithm TurboFlux uses a spanning tree of a query graph for filtering.

   However, using the spanning tree may have a low pruning power because it does not take into account all edges of the query graph.

Shortcomings of TurboFlux

2. TurboFlux proposes an auxiliary data structure called data-centric graph (DCG)

TurboFlux has the disadvantage that processing edge deletions is much slower than edge insertions due to the asymmetric update process of DCG.

- It was shown in previous work that the weak embedding of a directed acyclic graph is more effective in filtering candidates than the embedding of a spanning tree.

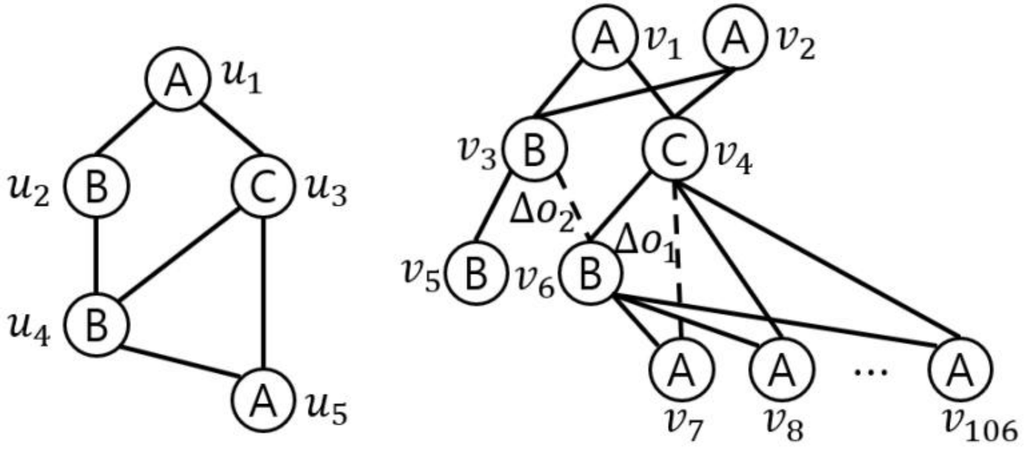
Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In *Proceedings of SIGMOD*. 1429–1446. https://doi.org/10.1145/3299869.3319880

Contributions

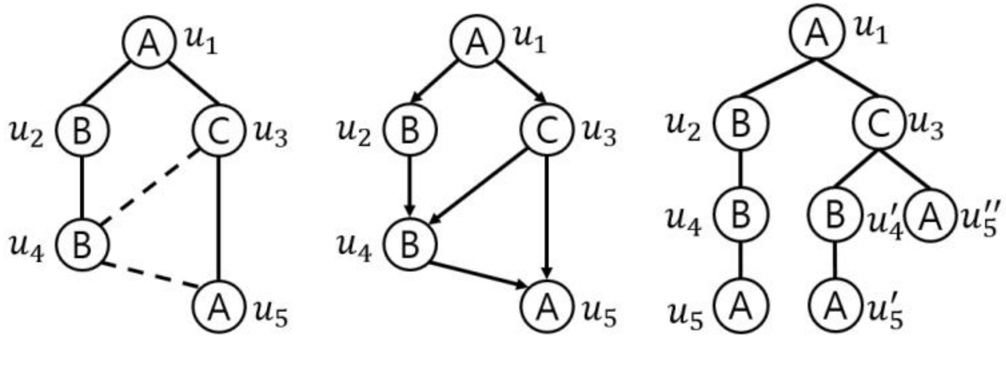
- Propose an auxiliary data structure called dynamic candidate space (DCS)
- Propose a new matching order which is different from the matching orders used in existing subgraph matching algorithms.

Weak Embedding

- Definition: For a rooted DAG $\hat{q}$ with root u, a weak embedding M' of $\hat{q}$ at $v \in V(g)$ is defined as a homomorphism of the path tree of $\hat{q}$ in g such that $M'(u) = v$.



(a) Query graph q (b) Dynamic data graph g with an initial data graph g_0 and two edge insertions



(a) Spanning tree q_T (b) Query DAG $\hat{q}$ (c) Path tree of $\hat{q}$

Weak Embedding

- Every embedding of q in g is a weak embedding of $\hat{q}$ in g, but the converse is not true.

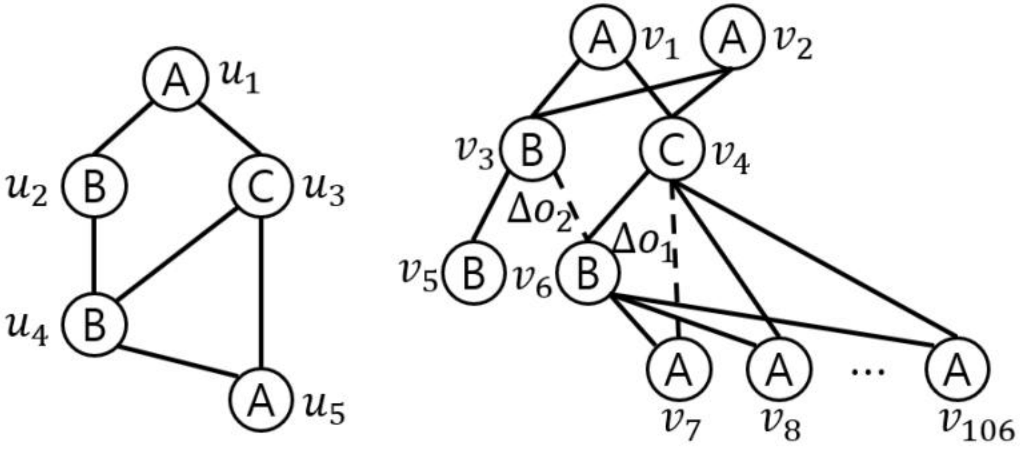
Overview

- BuildDAG(BFS, root vertex makes the DAG highest)
- BuildDCS
- Update DCS and perform continuous subgraph matching.

DCS(dynamic candidate space)

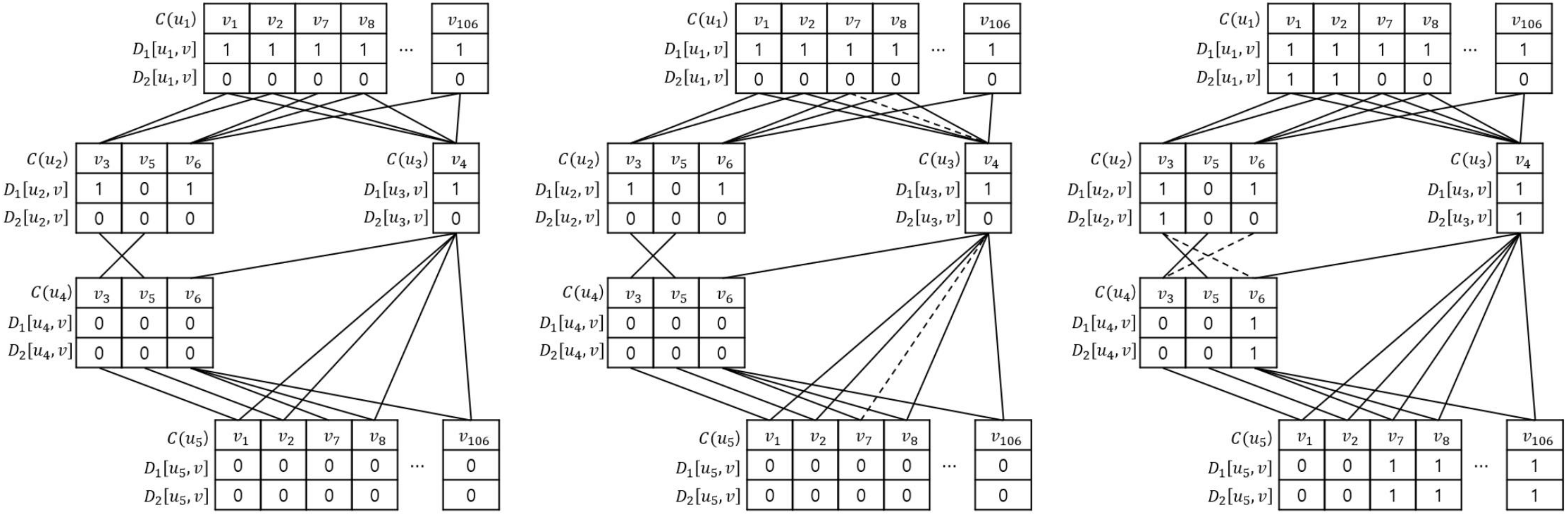
- DCS stores weak embeddings of DAGs as intermediate results to filter candidates

- $D1\ [u, v]$ & $D2\ [u, v]$



(a) Query graph q

(b) Dynamic data graph g with an initial data graph g_0 and two edge insertions



(a) Initial DCS_0

(b) DCS_1 after $\Delta o_1 = (+, v_4, v_7)$ occurs

(c) DCS_2 after $\Delta o_2 = (+, v_3, v_6)$ occurs

DCS Update(edge insertion)

We can see that there are two cases that $\langle up, vp \rangle$ affects $D1$ [u, v]:

(i) If $D1$ [up, vp] = 1 and an edge between $\langle up, vp \rangle$ and $\langle u, v\rangle$ is inserted.

(ii) If $D1$ [up, vp] changes from 0 to 1 and there is an edge between $\langle up, vp \rangle$ and $\langle u, v\rangle$.

Redundant Computations

First, if $\langle u, v\rangle$ has n updated parents then the above method computes $D1\,[u, v]$ n times in the worst case.

Second, to compute $D1\,[u, v]$, we need to reference the non-updated parents of $\langle u, v\rangle$ even if they do not change during the update.

Solution

$N^1_{u,v}[up]$ stores the number of candidates vp of up such that there exists an edge $(\langle up, vp\rangle, \langle u, v\rangle)$ and $D1[up, vp] = 1$,

$N^1_P[u, v]$ stores the number of parents up of u such that $N^1_{u,v}[up] \neq 0$.

$D1[u, v] = 1$ if and only if $N^1_P[u, v] = |\mathrm{Parent}(u)|$.

Use the same methon can update D2[u,v].

Edge deletion

The first case of the updated parent (or updated child) is changed to when an edge is deleted

The second case is changed to when $D1$ [up, vp] (or $D2$ [uc, vc]) changes from 1 to 0.

Next, if $D2$ [u, v] = 1 and $D1$ [u, v] becomes 0 during the $D1$ update, then $D2$ [u, v] also changes to 0.

Backtracking

First, we find all extendable vertices, and choose one vertex among them according to the matching order.

Once we decide an extendable vertex u to match, we compute its extendable candidates, which are the vertices in the data graph that can be matched to u.

Finally, we extend the partial embedding by matching u to one of its extendable candidates and continue the process.

Computing Extendable Candidates

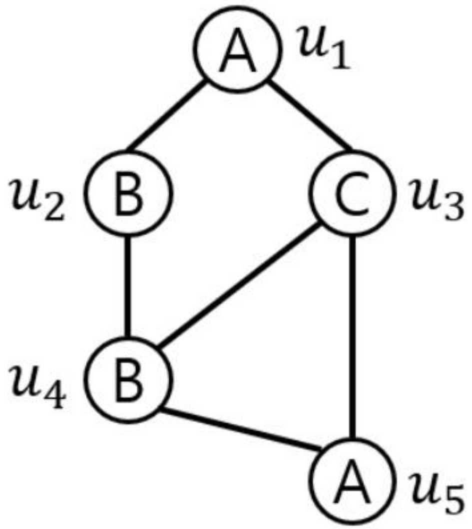
extendable candidates $C_M(u)$:

$$C_M(u) = \{v : D_2[u, v] = 1, \forall u' \in \mathrm{Nbr}_M(u), (v, M(u')) \in E(g)\},$$

$$C_M(u) = \{v \in S_{u_{min}} : \forall u' \in \mathrm{Nbr}_M(u) \backslash \{u_{min}\}, (v, M(u')) \in E(g)\}.$$

Isolated Vertices:

For a query graph q, a data graph g, and a partial embedding M, an isolated vertex is an extendable vertex in q, where all of its neighbors are mapped in M.



(a) Query graph q

Matching order

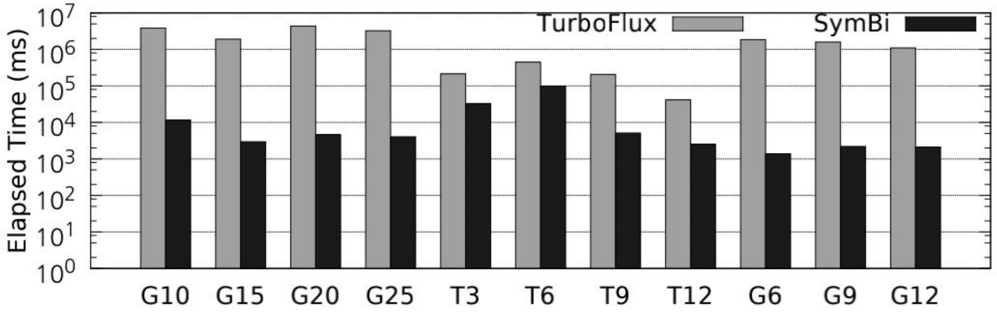
1. Backtrack if there exists an isolated vertex u such that all data vertices in $CM(u)$ have already matched.
2. If there exists at least one non-isolated extendable vertex in q, we choose the non-isolated extendable vertex u with smallest $E(u)$.
3. If every extendable vertex is isolated, we choose the extendable vertex u with smallest $E(u)$.

Experiment

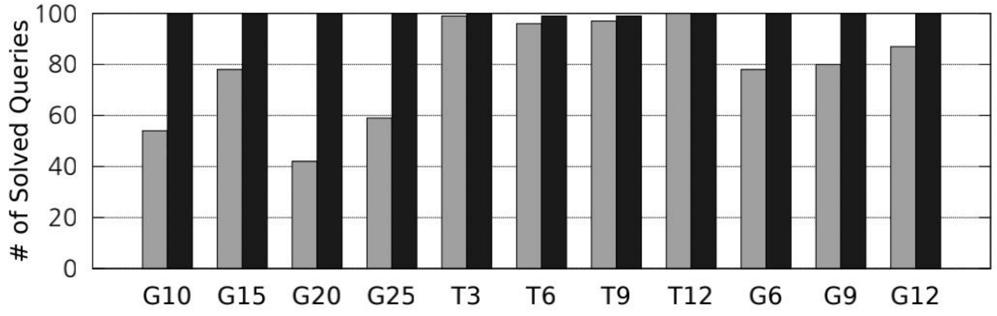
Table 2: Experiment settings

Parameter	Value Used
Datasets	LSBench, Netflow
Query size	**G10**, G15, G20, G25,
	T3, T6, T9, T12, G6, G9, G12
Insertion rate	2, 4, 6, 8, **10**
Deletion rate	**0**, 2, 4, 6, 8, 10
Dataset size	**0.1**, 0.5, 2.5 million users (LSBench)

Varying query size for Netflow

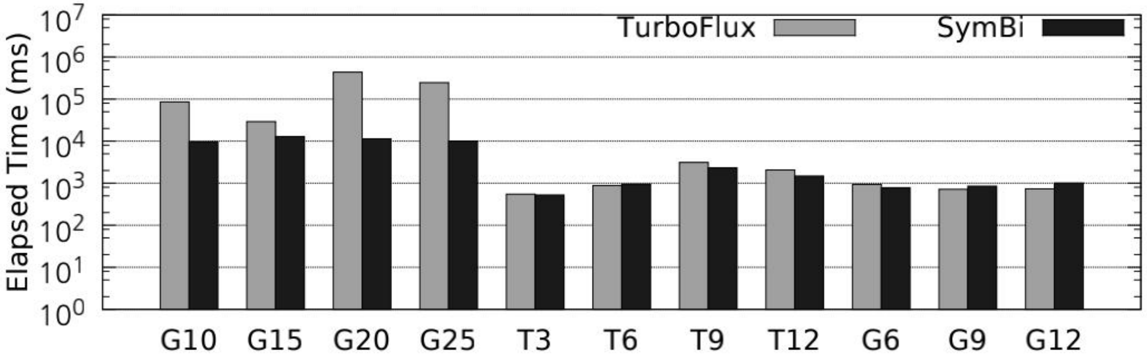


(a) Average elapsed time (in milliseconds)

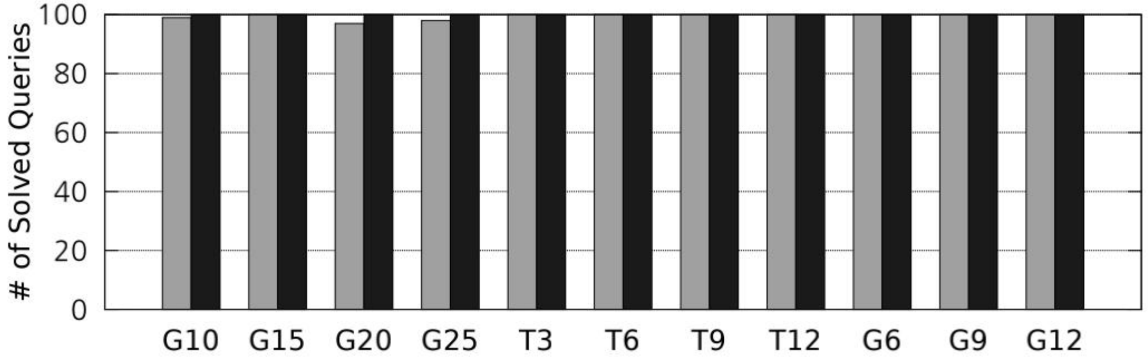


(b) Solved queries

Varying query size for LSBench

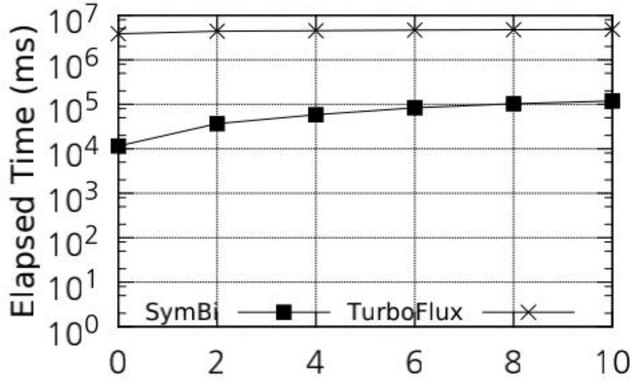


(a) Average elapsed time (in milliseconds)

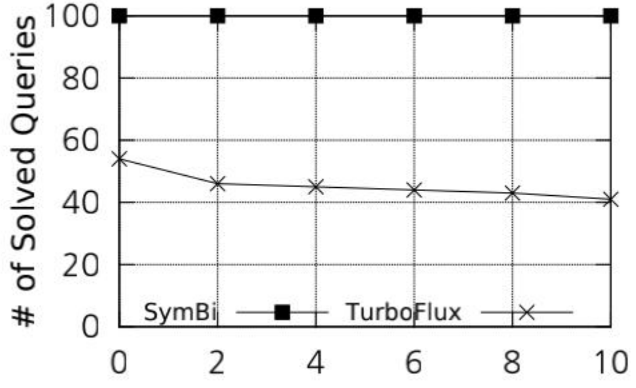


(b) Solved queries

Varying deletion rate for Netflow

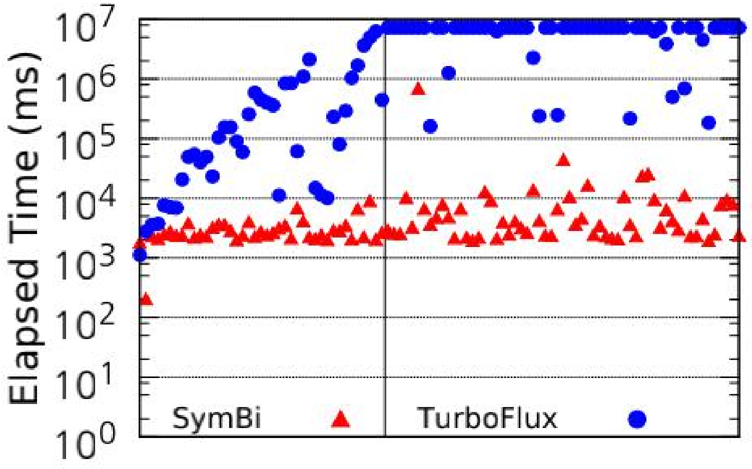


(a) Average elapsed time (in milliseconds)

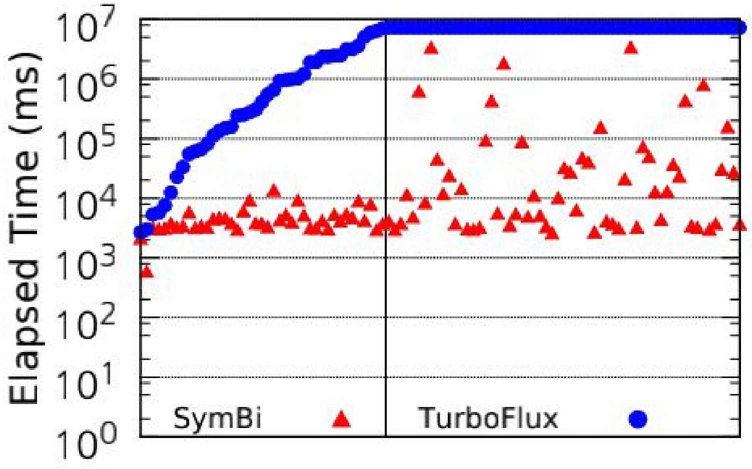


(b) Solved queries

Elapsed time of all queries for each algorithm with deletion rate 0% and 10%

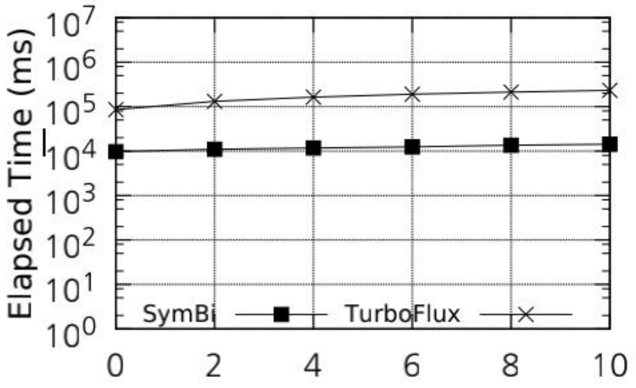


(a) Deletion rate 0%

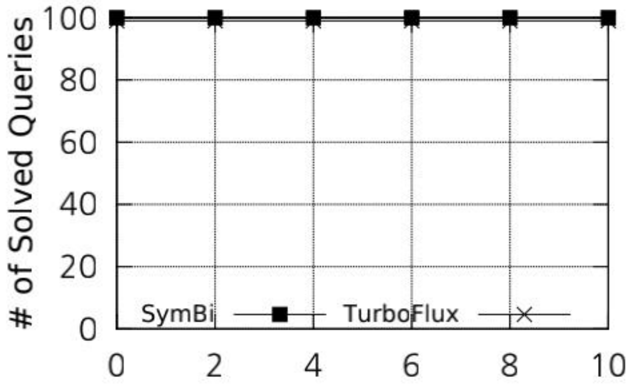


(b) Deletion rate 10%

Varying deletion rate for LSBench

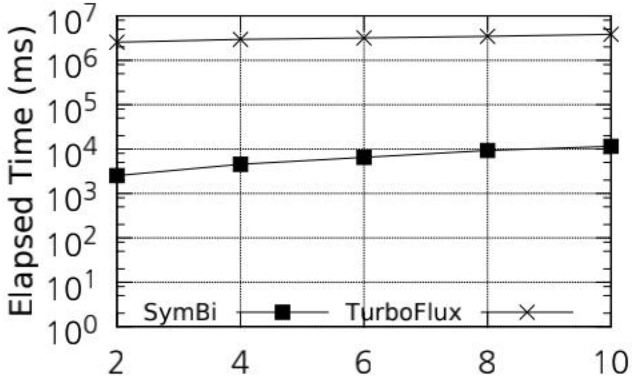


(a) Average elapsed time (in milliseconds)

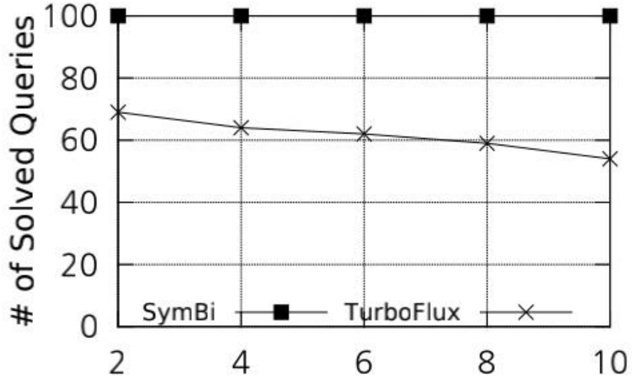


(b) Solved queries

Varying insertion rate for Netflow

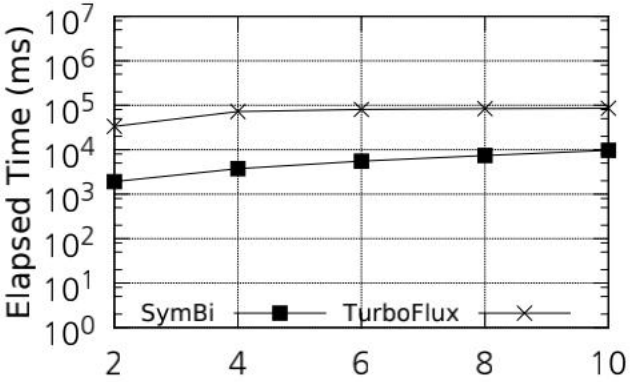


(a) Average elapsed time (in milliseconds)

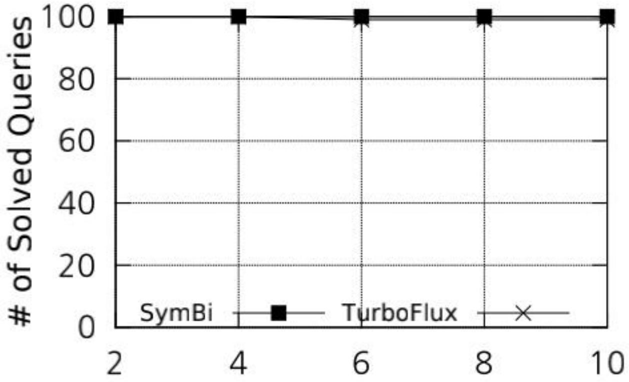


(b) Solved queries

Varying insertion rate for LSBench

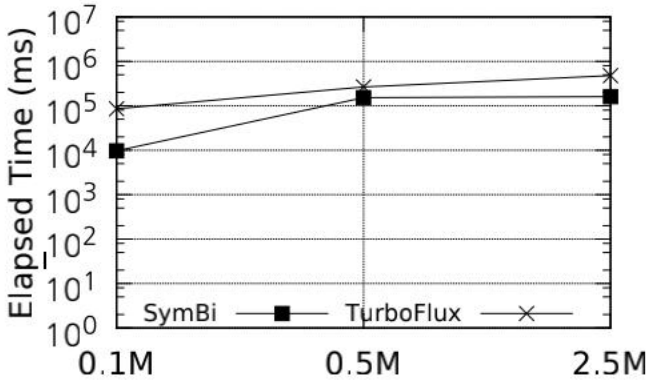


(a) Average elapsed time (in milliseconds)

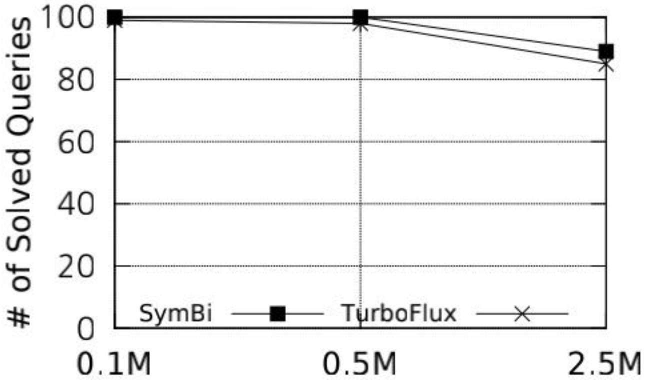


(b) Solved queries

Varying dataset size



(a) Average elapsed time (in milliseconds)



(b) Solved queries

Average peak memory (in MB)