University of Zurich[UZH]

Department of Informatics

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

Prof. Dr. Michael Böhlen
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

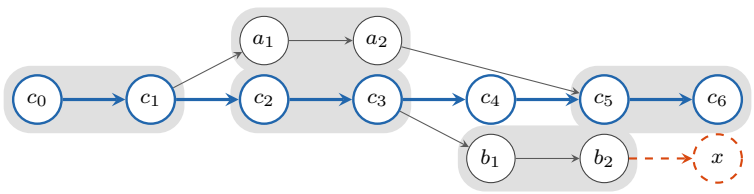
Zürich, 29. September 2026

MSc Thesis (30 points)
Topic: Efficient Partial Sources Queries on Dynamic Commit DAGs through Incremental Index Maintenance

Background and Motivation

Directed acyclic graphs (DAGs) are widely used to model dependencies, for example commit histories in version control systems, where an edge leads from a commit to every commit that was derived from it. A partial sources query asks, for a given subset S of the nodes, for all nodes of S that are not reachable from any other node of S. In a commit history these are the earliest commits of S, and the query is the basic step in determining the provenance of a file [9]. The algorithms of [9] answer such queries in microseconds on histories of more than a million commits. Each algorithm reads, in addition to the graph, structures that are computed once from the complete graph. We call the structures of one algorithm its index. This thesis considers three indexes, which together consist of five structures.

We define the depth of a node as the length of the longest path from any source, i.e. any node without incoming edges, to it, and we call the nodes of one depth a layer. Figure 1 shows a history of eleven commits, on which the five structures are illustrated.



depth	0	1	2	3	4	5	6
nodes in the layer	1	1	2	2	2	2	1

Figure 1: A history of eleven commits. The column of a node is its depth. The thick path is the spine, the shaded paths are contracted in the condensed DAG, and the dashed node x is a commit that is inserted.

The index of the algorithm LPS (Layered Partial Sources) of [9] consists of two structures.

1. The *layering* stores the depth of every node and the number of nodes of every layer. In Figure 1 the depth of a_2 is 3, and layer 3 has two nodes.
2. The *condensed DAG* contracts every maximal path whose edges each lead from a node with a single child to a node with a single parent into the first node of the path, and maps every node to this representative. In Figure 1 six of the eleven nodes remain, and the representative of c_6 is c_5.

The index of the algorithm LPSS (LPS with Skips) of [9] consists of these two structures and two further ones.

3. The *skip table* stores for a node u and every power of two k the first nodes on the paths leaving u that are at least k layers deeper than u. For c_1 these are $\{c_2, a_1\}$ for $k = 1$, $\{c_3, a_2\}$ for $k = 2$ and $\{c_5, b_2\}$ for $k = 4$.
4. The *saturation marks* store for a node the smallest depth at which it reaches all nodes of the layer; it then reaches all nodes of every deeper layer as well. The mark of c_3 is 4, since c_3 does not reach a_2 in layer 3 and reaches both nodes of layer 4.

The index of horizontal saturation, the criterion of our current work that discards elements of S without visiting the graph, consists of the layering and one further structure.

5. The *spine labels* fix one longest path of the graph, the spine, and store for every node the position of the last spine node that reaches it and the position of the first spine node that it reaches, and for every layer the smallest of the former positions among its nodes. In Figure 1 the spine is $c_0, \ldots, c_6$, the two positions of a_1 are 1 and 5, and the value of layer 3 is 1.

All five structures are built for a static graph, whereas a repository changes continuously. The history of the Linux kernel received 88,943 commits in its last recorded year, on average one every six minutes, and the history of Magento received 7,835. A query is answered in microseconds, whereas the construction of the skip table takes ten seconds for Linux and 25 minutes for Magento, since it reads the complete history although a commit adds a single node. If the index is rebuilt after every commit, the constructions add up to about ten days of computation per year for Linux and to more than four months for Magento. While a construction runs, queries that involve the newest commits cannot be answered with the index, and if the index is rebuilt less often this period grows accordingly. An index that is to serve a repository must therefore be maintained under changes rather than rebuilt.

Problem Statement

Every operation on a repository, whether a commit, a merge, a rebase, an amended commit or the deletion of a branch, changes the commit graph through two primitive updates [11, 5]: the insertion of a sink, a new node whose edges come from existing nodes, and the deletion of a set of nodes closed under descendants, since a commit remains in the repository as long as some reference reaches it and every ancestor of a reachable commit is reachable. The frequency of these operations in practice has been studied empirically [2, 21, 14].

Given the three indexes and a sequence of primitive updates, we seek for each of the five

structures update procedures that (i) are exact, in the sense that a query answered with the maintained index returns the partial sources of S in the current graph, i.e. the result of an evaluation without any index, (ii) modify only the entries that the update affects, and (iii) have a cost per update that is measured on real histories. For some of the structures the affected entries are known to be local. A sink insertion changes the depth of no existing node, increments one layer count, and changes the condensed DAG only in the neighbourhood of the parents of the new node. For the saturation marks and the skip table the question is open. A mark at depth d becomes invalid as soon as a node at depth d is inserted that the marked node does not reach, and an invalid mark is not merely slow but incorrect, since the sweep terminates early and omits a partial source. In Figure 1 the nodes a_1, a_2, c_4, c_5 and c_6 have the mark 6 and do not reach x, so that the insertion of x at depth 6 invalidates their marks. The new node further has to enter the skip table rows of every node from which it is the first node beyond a stored depth, in Figure 1 the rows of b_2, b_1 and c_2. Which entries these are, how many a single update affects on a real history, and how they are repaired is the core of the thesis. For the spine labels a first update procedure exists in our current work.

Related Work

Reachability indexes have been studied for four decades and are surveyed in [19]. The fastest are built for static graphs and are rebuilt after a change. The tree cover of Agrawal, Borgida and Jagadish [1] and the interval labels of GRAIL [17] assign to every node one or a few intervals, which a merge commit changes for all of its ancestors, and the hub labels of DAG-HL [16] form a global structure with the same difficulty. The maintenance of the transitive closure itself under edge insertions was solved by Italiano [12] in amortized constant time per new pair, Demetrescu and Italiano [8] established the trade-offs for fully dynamic transitive closure on DAGs, and Dadush et al. [7] recently revisited the insertion-only case by maintaining a bounded superset of the closure. These results maintain the closure itself, whose size is quadratic, rather than an index of bounded size.

Indexes that survive updates fall into three groups. Two-hop labelings have been maintained incrementally since HOPI [15] and the work of Bramandia, Choi and Ng [3], and the total order approach of Zhu et al. [20] and DLCR [6] maintain them under insertions and deletions; all of them rewrite the labels of existing nodes. DAGGER [18] maintains interval labels on dynamic graphs in the same manner. The third group is specific to append-only DAGs and forbids such rewriting. The incremental reachability index presented at SEA 2025 by Bulteau et al. [4], developed for the Mercurial version control system, combines the chain labeling of Jagadish [13] with the on-line chain partition of Felsner [10]: the entry of a new commit is written once and never modified, at the price of an index whose size grows with the number of chains through the history. Git itself keeps its commit-graph file in layers that are merged periodically [11]. The three indexes of this thesis are designed for such append-only DAGs. The thesis compares them, under the same update workload, with the chain index and DLCR, for which implementations are available in our group.

Proposed Approach

The goal of this thesis is to design, prove correct, implement and evaluate incremental update procedures for the five structures of the three indexes. We propose the following main steps:

1. **Update Model:** Formalize the two primitive updates, show how the repository operations decompose into them, and establish for every structure which entries a primitive update

can affect.

2. **Layering and Condensed DAG:** Derive the update procedures for the depths, the layer counts, the condensed DAG and the representative map from the locality established in the first step, with correctness arguments, and implement them within the existing C++ code base.
3. **Saturation Marks and Skip Table:** Measure on real histories how many marks a sink insertion invalidates and how many skip table rows it enters. Design the repair procedures, for instance a list per depth of the nodes whose mark ends at that depth, which is checked when the layer grows, and a bounded backward search from the parents of the new node for the affected rows. Prove that the maintained structures coincide with the rebuilt ones or, where the definition admits several valid structures, that the query results coincide with those of an evaluation without an index.
4. **Deletions and Batches:** Derive the procedures for the deletion of a descendant-closed set for all five structures, using that reachability and depth among the surviving nodes are unchanged, and compare the application of a batch of updates, as produced by one push, with their application one at a time.
5. **Evaluation on Real Histories:** Our clones record the complete dated history of every repository. For every repository the state one year before its newest commit is taken as the initial state, all structures are built for it, the commits of that year are applied in the order of their arrival, and the maintained structures are compared with structures rebuilt on the final state. Correctness is verified with ten thousand partial sources queries per repository: the results obtained with the maintained indexes have to agree with the results of a breadth-first search that uses no index. Deletions are generated as descendant closures of recent commits, since deleted commits are not visible in a clone. The measurements cover the five hundred repositories of the corpus with more than ten thousand commits and include, on the repositories where they complete within a day, the chain index [4] and DLCR [6].

Primary Goals and Tasks

The project will focus on the following core objectives:

1. **Formalization:** Define the primitive updates, the decomposition of the repository operations into them, and for every structure the set of entries a primitive update affects.
2. **Algorithm Design:** Develop the update procedures for insertion and deletion for all five structures, with correctness arguments.
3. **Implementation:** Implement the procedures, the replay of dated histories and the verification against an evaluation without an index and against a full rebuild within the existing C++ code base.
4. **Evaluation:** Measure the update cost per commit, the entries written, the index size and the query performance of the maintained index against a rebuilt one on five hundred real histories, and compare with the maintainable indexes from the literature.

Project Schedule (22 Weeks)

The thesis is planned to be completed over a 22-week period with the following tentative schedule:

- **Weeks 1–3:** Literature review, familiarization with the supervisor's previous work and the code base, and implementation of the replay of dated histories.
- **Weeks 4–6:** Formalization of the update model; update procedures for the layering, the condensed DAG and the representative map; verification against an evaluation without an index and against a full rebuild.
- **Weeks 7–11:** Measurement of the entries affected by an insertion; design, correctness arguments and implementation of the update procedures for the saturation marks and the skip table.
- **Weeks 12–14:** Integration of the spine labels; replay of all five structures over one year of history on the corpus; comparison with the chain index and DLCR.
- **Weeks 15–17:** Deletion procedures and batch updates; analysis of the conditions under which maintenance is preferable to a rebuild.
- **Weeks 18–22:** Finalizing the analysis, writing the thesis, and preparing the final presentation.

University of Zurich
Department of Informatics

Prof. Dr. Michael Böhlen
Professor

Supervisor: Mateo Dujić (dujic@ifi.uzh.ch)

Start date: October 1, 2026

End date: March 1, 2027

References

[1] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, page 253–262, New York, NY, USA, 1989. Association for Computing Machinery.

[2] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. The promises and perils of mining git. In *Proceedings of the 6th International Working Conference on Mining Software Repositories (MSR)*, pages 1–10, 2009.

[3] Ramadhana Bramandia, Byron Choi, and Wee Keong Ng. Incremental maintenance of 2-hop labeling of large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):682–698, 2010.

[4] Laurent Bulteau, Pierre-Yves David, Florian Horn, and Euxane Tran-Girard. Incremental reachability index. In *23rd International Symposium on Experimental Algorithms (SEA 2025)*, volume 338 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

[5] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2nd edition, 2014. `https://git-scm.com/book`.

[6] Xin Chen, You Peng, Sibo Wang, and Jeffrey Xu Yu. DLCR: Efficient indexing for label-constrained reachability queries on large dynamic graphs. *Proceedings of the VLDB Endowment*, 15(8):1645–1657, 2022.

[7] Daniel Dadush, James B. Orlin, Aaron Sidford, and László A. Végh. From incremental transitive cover to strongly polynomial maximum flow. arXiv:2510.20368, 2025.

[8] Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on DAGs: Breaking through the $O(n^2)$ barrier. *Journal of the ACM*, 52(2):147–156, 2005.

[9] Mateo Dujic, Michael Böhlen, Sven Helmer, and Oded Lachish. Determining file provenance in directed acyclic graphs using partial sources, 2026. Under submission.

[10] Stefan Felsner. On-line chain partitions of orders. *Theoretical Computer Science*, 175(2):283–292, 1997.

[11] Git project. Git reference manual. `https://git-scm.com/docs`. Accessed August 2026.

[12] Giuseppe F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273–281, 1986.

[13] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Trans. Database Syst.*, 15(4):558–598, December 1990.

[14] Matheus Paixão and Paulo Henrique Maia. Rebasing in code review considered harmful: A large-scale empirical investigation. In *Proceedings of the 19th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 45–55, 2019.

[15] R. Schenkel, A. Theobald, and G. Weikum. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In *21st International Conference on Data Engineering (ICDE'05)*, pages 360–371, 2005.

[16] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1601–1606. ACM, 2013.

[17] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. Grail: scalable reachability index for large graphs. *Proc. VLDB Endow.*, 3(1–2):276–284, September 2010.

[18] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. DAGGER: A scalable index for reachability queries in large dynamic graphs. arXiv:1301.0977, 2013.

[19] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. Indexing techniques for graph reachability queries. arXiv:2311.03542, 2023.

[20] Andy Diwen Zhu, Wenqing Lin, Sibo Wang, and Xiaokui Xiao. Reachability queries on large dynamic graphs: a total order approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 1323–1334, New York, NY, USA, 2014. Association for Computing Machinery.

[21] Weiqin Zou, Juan Chen, Yang Yang, Yu Huang, Bin Xu, and Wei Hu. Branch use in practice: A large-scale empirical study of 2,923 projects on github. In *Proceedings of the 2019 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 92–103, 2019.