FERRARI: Flexible and Efficient Reachability Range Assignment for Graph Indexing

Stephan Seufert[1], Avishek Anand[1], Srikanta Bedathur[2], Gerhard Weikum[1]

[1]*Max Planck Institute for Informatics, Germany*
{sseufert,aanand,weikum}@mpi-inf.mpg.de

[2]*IIIT Delhi, India*
bedathur@iiitd.ac.in

***Abstract*—In this paper, we propose a scalable and highly efficient index structure for the reachability problem over graphs. We build on the well-known node interval labeling scheme where the set of vertices reachable from a particular node is compactly encoded as a collection of node identifier ranges. We impose an explicit bound on the size of the index and flexibly assign approximate reachability ranges to nodes of the graph such that the number of index probes to answer a query is minimized. The resulting tunable index structure generates a better range labeling if the space budget is increased, thus providing a direct control over the trade off between index size and the query processing performance. By using a fast recursive querying method in conjunction with our index structure, we show that, in practice, reachability queries can be answered in the order of microseconds on an off-the-shelf computer – even for the case of massive-scale real world graphs. Our claims are supported by an extensive set of experimental results using a multitude of benchmark and real-world web-scale graph datasets.**

I. Introduction

Reachability queries are a fundamental operation in graph mining and algorithmics and ample work exists on index support for reachability problems. In this setting, given a directed graph and a designated source and target node, the task of the index is to determine whether the graph contains a path from the source to the target.

Computing reachability between nodes is a building block in many kinds of graph analytics, for example biological and social network analysis, traffic routing, software analysis, and linked data on the web, to name a few. In addition, a fast reachability index can prove useful for speeding up the execution of general graph algorithms – such as shortest path and Steiner tree computations – via search-space pruning. As an example, Dijkstra's algorithm can be greatly sped up by avoiding the expansion of vertices that cannot reach the target node.

While the reachability problem is a light-weight task in terms of its asymptotic complexity, the advent of massive graph structures comprising hundreds of millions of nodes and billions of edges can render even simple graph operations computationally challenging. It is thus crucial for reachability indices to provide answers in sublinear or ideally near-constant time. Further complicating matters, the index structures, which generally reside in main-memory, are expected to satisfy an upper-bound on the size. In most scenarios, the available space is scarce, ranging from little more than enough to store the graph itself to a small multiple of its size.

Given their wide applicability, reachability problems have been one of the research foci in graph processing over recent years. While many proposed index structures can easily handle small to medium-size graphs comprising hundreds of thousands of nodes (e.g., [1], [4], [6], [11], [12], [13], [16], [18], [19], [20]), massive problem instances still remain a challenge to most of them. The only technique that can cope with web-scale graphs while satisfying the requirements of restricted index size and fast query processing time, employs guided online search [22], [23], leading to an index structure that is competitive in terms of its construction time and storage space consumption, yet speeds up reachability query answering significantly when compared to a simple DFS/BFS traversal of the graph. However, it suffers from two major drawbacks. Firstly, given the demanding constraints on precomputation time, only *basic heuristics* are used during index construction, which in many cases leads to a suboptimal use of the available space. Secondly and more importantly, while the majority of reachability queries involving pairs of nodes that are not reachable can be efficiently answered, the important class of *positive queries* (i. e. the cases in which the graph actually contains a path from the source to target) has to be regarded as a worst-case scenario due to the need of recursive querying. This can severely hurt the performance of many practical applications where positive queries occur frequently.

The reachability index structure we propose in this paper – coined FERRARI (for Flexible and Efficient Reachability Range Assignment for gRaph Indexing) – overcomes the limitations of existing approaches by *adaptively compressing the transitive closure during its construction*. This technique enables the efficient computation of an index geared towards minimizing the expected query processing time given a user-specified constraint on the resulting index size. Our proposed index supports positive queries efficently and outperforms GRAIL, the best prior method, on this class of queries by a large margin, while in the vast majority of our experiments also being faster on randomly generated queries. To achieve these performance gains, we adopt the idea of representing the transitive closure of the graph by assigning identifiers to individual nodes and encoding sets of reachable vertices by intervals, first introduced by Agrawal et al. [1]. Instead of materializing the full set of identifier ranges at every node, we adaptively merge adjacent intervals into fewer yet coarser representations at construction time, whenever a certain space budget is exceeded. The result is a collection of *exact and approximate* intervals that are assigned as labels of the nodes in the graph. These labels allow for a guided online search procedure that can process positive as well as negative reachability queries significantly faster than previously proposed size-constrained index structures.

The interval assignment underlying our approach is based on the solution of an associated interval cover problem. Efficient algorithms for computing such a covering structure together with an optimized guided online search facilitate an efficient and flexible reachability index structure.

In summary, this paper makes the following technical contributions:

- a space-adaptive index structure for reachability queries based on selective compression of the transitive closure using exact and approximate reachability intervals,
- efficient algorithms for index construction and querying that allow extremely fast query processing on web-scale real world graphs, and
- extensive experiments that demonstrate the superiority of our approach in comparison to the best prior method that satisfies index size constraints, GRAIL.

The remainder of the paper is organized as follows: In Section II we introduce necessary notation and the basic idea of reachability interval labeling. Afterwards, we give a short introduction to approximate interval indexing in Section III, followed by an in-depth treatment of our proposed index (Section IV). An overview over our query processing algorithm is given in Section V, followed by the experimental evaluation and concluding remarks.

II. Preliminaries

In the following, let $G = (V, E)$ denote a directed graph with $n := |V|$ nodes and $m := |E|$ edges. For a node v, let

$$\mathcal{N}^+(v) := \{w \in V \mid (v, w) \in E\} \tag{1}$$

$$\text{and} \quad \mathcal{N}^-(v) := \{u \in V \mid (u, v) \in E\} \tag{2}$$

denote the sets of nodes with an edge coming from and leading to v, respectively.

Given nodes $u, v \in V$, we call v *reachable* from u, written as $u \sim v$, if E contains a directed path from u to v. Further, we denote the set of vertices reachable from $v \in V$ in G by $\mathcal{R}_G(v) := \{w \in V \mid v \sim w\}$. We call $\mathcal{R}_G(v)$ the *reachable set* of v (we drop the subscript whenever the graph under consideration is clear from the context). A *reachability query* with source node u and destination node v of a graph G is expressed as a triple (G, u, v) and answered with a boolean value by a reachability index.

A pair (u, v) of nodes exhibits *strong reachability* if $u \sim v$ and $v \sim u$, that is, u and v are mutually reachable in G. Note that strong reachability induces an equivalence relation on the set of nodes. The equivalence classes of this relation, that is, the maximal subsets $V' \subseteq V$ with $u \sim v$ for all $u, v \in V'$, are called *strongly connected components* of G. For a node $v \in V$ let $[v]$ denote the strongly connected component that contains v.

Condensed Graph. We define the *condensed graph* of G, denoted as $G_C = (V_C, E_C)$, as the graph obtained after collapsing the maximal strongly connected components into "supernodes", i. e.

$$V_C := \big\{[v] \bigm| v \in V\big\} \tag{3}$$

$$\text{and} \quad E_C := \big\{\big([u], [v]\big) \bigm| (u, v) \in E, [u] \neq [v]\big\}. \tag{4}$$

By definition, G_C is a directed acyclic graph (DAG).

It is important to note that the reachability queries (G, u, v) and $(G_C, [u], [v])$ are equivalent. Thus, the existing index structures (including ours) consider only directed acyclic graphs, that is, create an index over the condensed graph G_C. At query time, the input nodes u, v are then mapped to their respective strongly connected components, allowing early termination whenever $[u] = [v]$.

Tree Cover and Graph Augmentation. For a graph G, we define a *tree cover* of G, denoted as $T(G)$, as a directed spanning tree of G. If G contains more than one node with no incoming edges, instead of a tree only a spanning forest can be obtained.

In this case, we augment the graph G by introducing an artificial root node r that is connected to every node with no incoming edge:

$$G' := \big(V \cup \{r\}, E \cup \{(r, v) \mid v \in V, \mathcal{N}^-(v) = \emptyset\}\big). \tag{5}$$

Note that this modification has no effect on the reachability relation among the existing nodes of G.

Integer Intervals. For integers $x, y \in \mathbb{N}, x \leq y$, we use the interval $[x, y]$ to represent the set $\{x, x+1, \ldots, y\}$. Let $I = [a, b]$ and $J = [p, q]$ denote integer intervals. We define $|I| := b - a + 1$ to denote the number of elements contained in I. Further, we call J *subsumed by* I, written $J \sqsubseteq I$, if J corresponds to a subinterval of I. Further, J is called an *extension* of I, denoted $I \hookleftarrow J$, if the start-point but not the end-point of J is contained in I.

A. Interval Indexing

In this section, we introduce the concept of node identifier intervals for reachability processing, first proposed by Agrawal et al. [1], which provided the basis of many subsequent indexing approaches, including our own. The key idea is to assign numeric identifiers to the nodes in the graph and represent the reachable sets of vertices in a compressed form by means of interval representations. This technique is based on the construction of a tree cover of the graph followed by post-order labeling of the vertices.

Let G' denote the augmented input graph as defined above. Further, let $T = (V_T, E_T)$ denote a tree cover of G'. In order to assign node identifiers, the tree is traversed in depth-first manner. In this setting, a node v is visited after all its children have been visited. The post-order number $\pi(v)$ corresponds to the order of v in the sequence of visited nodes.

Tree Indexing. The enabling feature, which makes post-order labeling a common ingredient in reachability indices, is the resulting *identifier locality*: For every (complete) subtree of T, the ordered identifiers of the included nodes form a contiguous sequence of integers. The vertex set of any such subtree can thus be compactly expressed as an integer interval. Let $T_v = (V_{T_v}, E_{T_v})$ denote the subtree of T rooted at node v. We have

$$\begin{aligned}\{\pi(w) \mid w \in V_{T_v}\} &= \left[\min_{w\in V_{T_v}} \pi(w),\ \max_{w\in V_{T_v}} \pi(w)\right] \\ &= \left[\min_{w\in V_{T_v}} \pi(w),\ \pi(v)\right].\end{aligned} \tag{6}$$

Above interval is called *tree interval* of v and will be denoted by $I_T(v)$ in the remainder of the text.

The complete reachability information of the spanning tree T is encoded in the collection of tree intervals. For a pair of nodes $u, v \in V$, there exists a path from u to v in T iff the post-order number of the target is contained in the tree interval of the source, that is,

$$u \sim_T v \quad \iff \quad \pi(v) \in I_T(u). \tag{7}$$

This reachability index for trees allows for $O(1)$ query processing at a space consumption of $O(n)$.

Extension to DAGs. While above technique can be used to easily answer reachability queries on trees, the case of general DAGs is much more challenging. The reason is that, in general, the reachable set $\mathcal{R}(v)$ of a vertex v in the DAG is only partly represented by the interval $I_T(v)$, as the tree interval only accounts for reachability relationships that are preserved in T. Vertices that can only be reached from a node v by traversing one or more non-tree edges have to be handled seperately: instead of merely storing the tree intervals $I_T(v)$, every node v is now assigned a *set of intervals*, denoted by $\mathcal{I}(v)$. The purpose of this so-called *reachable interval set* is to capture the complete reachability information of a node. The sets $\mathcal{I}(v), v \in V$ are initialized to contain only the tree interval $I_T(v)$. Then, the vertices are visited in reverse topological order. For the current vertex v and every outgoing edge $(v, w) \in E$, the reachable interval set $\mathcal{I}(w)$ is merged into the set $\mathcal{I}(v)$. The merge operation on the intervals resolves all cases of interval subsumption and extension exhaustively, eventually ensuring interval disjointness. Due to the fact that the vertices are visited in reverse topological order, it is ensured that for every non-tree edge $(s, t) \in E \setminus E_T$, the reachability intervals in $\mathcal{I}(t)$ will be propagated and merged into the reachable interval sets of *s and all its predecessors*. As a result, all reachability relationships are covered by the resulting intervals.

Query Processing. Using the reachable interval sets $\mathcal{I}(v)$, queries on DAGs can be answered by checking whether the post-order number of the target is contained in *one of the intervals* associated with the source:

$$u \sim v \quad \iff \quad \exists\big([\alpha, \beta] \in \mathcal{I}(u)\big) : \alpha \leq \pi(v) \leq \beta. \tag{8}$$

By ordering the intervals contained in a set, reachability queries can now be answered efficiently in $O(\log n)$ time on DAGs. The resulting index (collection of reachable interval sets) can be regarded as a materialization of the transitive closure of the graph, rendering this approach potentially infeasible for large graphs, both in terms of space consumption as well as computational complexity.

III. Approximate Intervals

For massive problem instances, indexing approaches that materialize the transitive closure (or compute a compressed variant without an a priori size restriction), suffer from limited applicability. For this reason, recent work on reachability query processing over massive graphs includes a shift towards guided online search procedures. In this setting, every node is assigned a concise label which – in contrast to the interval sets described in Section II-A – is restricted by a predefined size constraint. These labels in general do not allow answering the query after inspection of just the source node, yet can be used to prune portions of the graph in an online search.

As a basic example, consider a reachability index that labels every node $v \in V$ with its topological order number $\tau(v)$. While this simple variant of node labeling is obviously not sufficient to answer a reachability query by means of a single-lookup, a graph search procedure can greatly benefit from the node labels: For a given query (s, t), the online search rooted at s can terminate the expansion of a branch of the graph whenever for the currently considered node v it holds

$$\tau(v) \geq \tau(t). \tag{9}$$

This follows from the properties of a topological ordering. The recently proposed GRAIL reachability index [22], [23] further extends this idea by labeling the vertices with approximate intervals:

Suppose that for every node v we replace the set $\mathcal{I}(v)$ by a single interval

$$I'(v) := \left[\min_{w\in\mathcal{R}(v)} \pi(w), \max_{w\in\mathcal{R}(v)} \pi(w)\right], \tag{10}$$

spanning from the lowest to the highest reachable id. This interval is approximate in the sense that all reachable ids are covered whereas false positive entries are possible:

Definition 1 (False Positive). *Let $v \in V$ denote a node with the approximate interval $I'(v) = [\alpha, \beta]$. A vertex $w \in V$ is called* false positive *with respect to $I'(v)$ if*

$$\alpha \leq \pi(w) \leq \beta \quad \textit{and} \quad v \not\sim w. \tag{11}$$

Obviously, the single interval $I'(v)$ is not sufficient to establish a definite answer to a reachability query of the form (G, v, w). However, all queries involving a target id $\pi(w)$ that lies outside the interval, i. e.

$$\pi(w) < \alpha \quad \text{or} \quad \pi(w) > \beta, \tag{12}$$

can be answered instantly with a negative answer, similar to the basic approach based on Equation (9). In the opposite case, that is, $\alpha \leq \pi(w) \leq \beta$, no definite answer to the reachability query can be given and the online search procedure continues with an expansion of the child vertices, terminating as soon as the target node is encountered or all branches have been expanded or pruned, respectively.

In practical applications the GRAIL index assigns a number of $k \geq 1$ such approximate intervals to every vertex, each based on a different (random) spanning tree of the graph. The intuition behind this labeling is that an ensemble of independently generated intervals improves the effectiveness of the node labels since each additional interval potentially reduces the remaining false positive entries.
The advantage of this indexing approach over a materialization of the transitive closure is obvious: the size of the resulting labels can be determined a priori by an appropriate selection of the number (k) of intervals assigned to each node. In addition, the node labels are easily computed by means of k DFS traversals of the graph.

Empirically, GRAIL has been shown to greatly improve the query processing time over online DFS search in many cases. However, especially in the case of positive queries, a large portion of the graph still has to be expanded. While extensions have been proposed to GRAIL to improve performance on positive queries [23], the processing time in these cases remains high. Furthermore, while an increase of the number of intervals assigned to the nodes potentially reduces false positive elements, no guarantee can be made due to the heuristic nature of the underlying algorithm. As a result, in many cases superfluous intervals are stored, in some cases *negatively* impacting query processing time.

IV. The FERRARI Reachability Index

In this section, we present the FERRARI reachability index which enables fast query processing performance over massive graphs by a more involved node labeling approach. The main goal of our index is the assignment of a *mixture of exact and approximate* reachability intervals to the vertices with the goal of minimizing the expected query processing time, given *a user-specified size constraint on the index*. Contrasting previously proposed approaches, we show both theoretically and empirically that the interval assignment of the FERRARI index utilizes the available space for maximum effectiveness of the node labels.

Similar to previously proposed index structures [1], [19], [22], [23], we use intervals to encode reachability relationships of the vertices. However, in contrast to existing approaches, FERRARI can be regarded as an *adaptive transitive closure compression* algorithm. More precisely, FERRARI uses *selective interval set compression*, where a subset of adjacent intervals in an interval set is merged into a smaller number of approximate intervals. The resulting node label then retains a high pruning effectiveness under a given size-restriction.
Before we delve into the details of our algorithms and the according query processing procedure, we first introduce the basic concepts that facilitate our interval assignment approach.

The FERRARI index distinguishes between two types of intervals: approximate (similar to the intervals in Section III) and exact (as in Section II-A), depending on whether they contain false positive elements or not.

Let I denote an interval. To easily distinguish between interval types, we introduce an indicator variable η_I such that

$$\eta_I := \begin{cases} 0 & \text{if } I \text{ approximate,} \\ 1 & \text{if } I \text{ exact.} \end{cases} \tag{13}$$

As outlined above, a main characteristic of FERRARI is the assignment of size-restricted interval sets comprising approximate and exact intervals as node labels. Before we introduce the algorithmic steps that facilitate the index construction, it is important to explain how reachability queries can be answered using the proposed interval sets. Let (G, s, t) denote a reachability query and $\mathcal{I}(s) = \{I_1, I_2, \ldots, I_N\}$ the set of intervals associated with node s. In order to determine whether t is reachable from node s, we have to check whether the post-order identifier $\pi(t)$ of t is included in one of the intervals in the set $\mathcal{I}(s)$. If $\pi(t)$ lies outside of all intervals $I_1, \ldots, I_N$, the query terminates with a negative answer. If however it holds that $\pi(t) \in I_i$ for one $I_i \in \mathcal{I}(s)$, we have to distinguish two cases: (i) if I_i is exact then s is guaranteed to reach node t and (ii) if I_i is approximate, the neighbors of node s have to be queried recursively until a definite answer can be given. Obviously, recursive expansions are costly and it is thus desirable to minimize the number of cases that require lookups beyond the source node.
To formally introduce the according optimization problem, we define the notion of *interval covers*:

Definition 2 (k-Interval Cover). *Let $k \geq 1$ denote an integer and $\mathcal{I} = \{[\alpha_1, \beta_1], \ldots, [\alpha_N, \beta_N]\}$ a set of intervals. A set $\mathcal{C} = \{[\alpha'_1, \beta'_1], \ldots, [\alpha'_l, \beta'_l]\}$ is called* k-interval cover *of $\mathcal{I}$, written as $\mathcal{C} \sqsupseteq_k \mathcal{I}$, if $\mathcal{C}$ covers all elements from $\mathcal{I}$ using no more than k intervals, i. e.*

$$\bigcup_{i=1}^{N} \{\, j \mid \alpha_i \leq j \leq \beta_i \,\} \subseteq \bigcup_{i=1}^{l} \{\, j' \mid \alpha'_i \leq j' \leq \beta'_i \,\} \tag{14}$$

$$\text{with} \quad l \leq k. \tag{15}$$

Note that an interval cover of a set of intervals is easily obtained by merging an arbitrary number of adjacent intervals in the input set. Next, we address the problem of choosing a k-interval cover that maximizes the pruning effectiveness.

Definition 3 (Optimal k-Interval Cover). *Let $k \geq 1$ denote an integer and $\mathcal{I} = \{I_1, I_2, \ldots, I_N\}$ an interval set of size N. We define the* optimal k-interval cover *of $\mathcal{I}$ by*

$$\mathcal{I}_k^* := \operatorname*{arg\,min}_{\mathcal{C}\,:\,\mathcal{I} \sqsubseteq_k \mathcal{C}} \sum_{I \in \mathcal{C}} (1 - \eta_I)\, |I|, \tag{16}$$

that is, the cover of $\mathcal{I}$ with no more than k intervals and the minimum number of elements in approximate intervals.

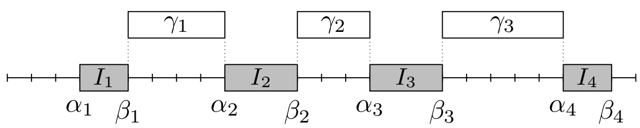
Note that by replacing the set of exact reachability intervals $\mathcal{I}(v)$ by its optimal k-interval cover $\mathcal{I}_k^*(v)$ – which is then used as the node label in our index – we retain maximal effectiveness for terminating a query. The reason is that the number of cases that require recursive querying directly corresponds to the number of elements contained in approximate intervals.

A. Computing the Optimal Interval Cover

While the special cases $k = N$ (optimal k-interval cover of $\mathcal{I}$ is the set $\mathcal{I}$ itself) and $k = 1$ (optimal solution corresponds to the single approximate interval assigned by GRAIL, see Equation 10) are easily solved, we next outline an algorithm that solves the problem for general values of k:

As hinted above, an interval cover can be computed by selectively merging adjacent intervals from the original assignment made to the node v. In order to derive an algorithm for computing $\mathcal{I}_k^*(v)$, we first transform the interval set at the node v into its dual representation, where the *gaps between intervals* are specified. As before, let $\mathcal{I} = \{I_1, I_2, \ldots, I_N\}$ with $I_i = [\alpha_i, \beta_i]$.

The set $\Gamma := \{\gamma_1, \gamma_2, \ldots, \gamma_{N-1}\}$, $\gamma_i = [\beta_i + 1, \alpha_{i+1} - 1]$ denotes the *gaps* between the intervals contained in $\mathcal{I}$:



Note that the gap set Γ together with the boundary elements α_1, β_N is an equivalent representation of $\mathcal{I}$. For a subset $G \subseteq \Gamma$ we denote by $\zeta(G)$ the induced interval set obtained by merging adjacent intervals I_i, I_{i+1} if for their mutually adjacent gap γ_i it holds $\gamma_i \notin G$. As an illustrative example, for the interval set depicted above we have $\zeta\big(\{\gamma_2\}\big) := \big\{[\alpha_1, \beta_2], [\alpha_3, \beta_4]\big\}$.

Every induced interval set $\zeta(G)$ actually corresponds to a $|G| + 1$-interval cover of the original set $\mathcal{I}$. It is easy to see that the *optimal* k-interval cover can equivalently be specified by a subset of gaps.

In order to compute the optimal k-interval cover, we thus transform the problem defined in Equation (16) into the equivalent problem of selecting the "best" $k-1$ gaps from the original gap set Γ (or, equivalently, determining the $|\mathcal{I}| - k - 1$ that are not included in the solution). For a potential solution $G \subseteq \Gamma$ of at most $k-1$ gaps to preserve, we can assess its *cost*, measured by the number of elements in the induced interval cover that are contained in approximate intervals:

$$c(G) := \sum_{I \in \zeta(G)} (1 - \eta'_I)\, |I|, \tag{17}$$

where for $I \in \zeta(G)$ it holds

$$\eta'_I := \begin{cases} 1 & \text{if } I \in \mathcal{I} \ \wedge \ \eta_I = 1, \\ 0 & \text{else.} \end{cases} \tag{18}$$

Clearly, our goal is to determine the set $\Gamma_{k-1}^* \subseteq \Gamma$ such that

$$\Gamma_{k-1}^* := \operatorname*{arg\,min}_{G \subseteq \Gamma,\ |G| \leq k-1} c(G). \tag{19}$$

The gap selection problem exhibits an optimal substructure property: The optimal k-interval cover of the first i intervals is represented by either (i) a $k-1$-interval cover of the first $i-1$ intervals combined with the last gap γ_{i-1} or (ii) a k-interval cover of the first $i-1$ intervals extended to the last interval. The respective dynamic programming procedure permits the computation of the optimal k-interval cover in time $O(kN)$. For further details, we refer the reader to the technical report version of this paper [17].

In some practical applications the amount of computation required by the dynamic programming approach can become prohibitive, as one instance of the problem has to be solved for every node in the graph. Thus, in our implementation, we use a simple and fast greedy algorithm that, starting from the empty set iteratively adds the gap $\gamma \in \Gamma$ that leads to the greatest reduction in cost given the current selection G, until at most $k-1$ gaps have been selected, then compute the interval cover from $\zeta(G)$. While the gain in speed comes at the cost of a potentially suboptimal cover, our experimental evaluation demonstrates that this approach works well in practice.

In the next section, we explain how above node labeling technique is eventually used as a building block during the reachability index computation.

B. Index Construction

At precomputation time, the user specifies a certain budget $B = kn, k \geq 1$ of intervals that can be assigned to the nodes, thus directly controlling the tradeoff between index size/precomputation time and pruning effectiveness of the respective nodes labels. The subsequent index construction procedure can be broken down into the following main stages:

1) Tree Cover Construction: Agrawal et al. [1] propose an algorithm for computing a tree cover that leads to the minimum number of exact intervals to store. This tree can be computed in $O(mn)$ time, rendering the approach infeasible for the case of massive graphs. While, in principle, heuristics could be used that are based on centrality measures or estimates of the sizes of reachable sets [5], [15], we settle for a simpler solution that does not yield a certain approximation guarantee yet performs well in practice. We argue that a good tree cover should cover as many reachability relationships as possible in the initial tree intervals (see Equation 6). Therefore, every edge included in the tree should provide a connection between as many pairs of nodes as possible. To this end, we propose the following procedure to heuristically construct such a tree cover T:

Let $\tau : V \to \{1, 2, \ldots, n\}$ denote a topological ordering of the vertices, i. e. for all $(u, v) \in E$ it holds $\tau(u) < \tau(v)$. Such a topological ordering is easily obtained by the classical textbook algorithm [7] in time $O(m + n)$. We interpret the topological order number of a vertex as the number of *potential predecessors* in the graph, because the number of predecessors of a given node is upper bounded by its position in the topological ordering. For a vertex v with set of predecessors $\mathcal{N}^-(v)$, we select the edge from node $p \in \mathcal{N}^-(v)$ with highest topological order number for inclusion in the tree, that is, $p := \arg\max_{u \in \mathcal{N}^-(v)} \tau(u)$. The intuition is that node p has the highest number of potential predecessors and thus the selected edge (p, v) has the potential of providing a connection from a large number of nodes to v, eventually leading to a large number of reachability relationships encoded in the resulting tree intervals.

2) Interval Set Assignment: As the next step, given the tree cover T, indexing proceeds by assigning the exact tree interval

$I_T(v)$, encoding the reachability relationships within the tree at each node v. This interval assignment can be obtained using a single depth-first traversal of T.

In order to label every node v with a k-interval cover $\mathcal{I}'(v)$ of its true reachable interval set, we visit the vertices of the graph in reverse topological order, that is, starting from the leaf with highest topological order, proceeding iteratively backwards to the root node. We initialize for node v the reachable interval interval set as $\mathcal{I}'(v) := \{I_T(v)\}$. For the currently visited node v and every edge $(v, w) \in E$, we merge $\mathcal{I}'(w)$ into $\mathcal{I}'(v)$, such that the resulting set of intervals is closed under subsumption and extension.[1]

Then, in order to satisfy the size restriction of at most k intervals associated with a node, we replace $\mathcal{I}'(v)$ by its k-interval cover which is then stored as the node label of v in our index. The complete procedure is shown in detail in Algorithm 1. It is easy to see that the resulting index consisting of the sets of approximate and exact intervals $\mathcal{I}'(v), v \in V$ comprises *at most* $nk = B$ intervals. The upper bound $\sum_{v \in V} |\mathcal{I}'(v)| \leq B$ is usually not tight, i. e. in practice, much less than B intervals are assigned. As an example, in the case of leaf nodes or the root vertex, a single interval suffices. The name of the algorithm – FERRARI-L – thus reflects the fact that a *local* size restriction, $|\mathcal{I}'(v)| \leq k$, is satisfied by every interval set $\mathcal{I}'(v)$.

Note that even though an optimal algorithm can be used to compute the k-interval covers, the optimality of the local result does in general not extend to the global solution, i. e. the full set of node labels. The reason for this is the fact that adjacent intervals that are merged during the interval cover computation are propagated to the parent nodes. As a result, at the point during the execution of the algorithm where the interval set of the parent p has to be covered, the k-interval cover is computed without knowledge of the true (exact) reachability intervals of p. More precisely, the input to the covering algorithm is a combination of approximate (thus previously merged) and exact intervals. Nevertheless, the resulting node labels prove very effective for early termination of reachability queries, as our experimental evaluation indicates.

To further improve our reachability index, in the next section we propose a variant of the labeling algorithm that leads to an even better utilization of the available space.

Algorithm 1: FERRARI-L(G, B)

Input: directed, acyclic graph G, interval budget $B = kn$
Result: set of at most k approximate and exact reachability intervals $\mathcal{I}'(v)$ for every node $v \in V$

begin
  $T \leftarrow$ TREECOVER(G)
  $I_T \leftarrow$ ASSIGNTREEINTERVALS(T)
  $k \leftarrow \frac{B}{n}$
  for $i = n$ **to** 1 **do** ▷ visit nodes in reverse topological order
    $\{v\} \leftarrow \{v \in V \mid \tau(v) = i\}$
    $\mathcal{I}'(v) \leftarrow \{I_T(v)\}$
    foreach $w \in \mathcal{N}^+(v)$ **do**
      $\mathcal{I}'(v) \leftarrow \mathcal{I}'(v) \oplus \mathcal{I}'(w)$ ▷ merge interval sets
    ▷ replace intervals by k-interval cover
    $\mathcal{I}'(v) \leftarrow k$-INTERVALCOVER$(\mathcal{I}'(v))$
  return $\{\mathcal{I}'(v) \mid v \in V\}$

C. Dynamic Budget Allocation

As mentioned above, the interval assignment as described in Algorithm 1 usually leads to a total number of far less than B intervals stored at the nodes. In order to better exploit the available space, we extend our algorithm by introducing the concept of *deferred interval merging* where we can assign more than k intervals to the nodes on the first visit, potentially requiring to revisit a node at a later stage of the algorithm.

The indexing algorithm for this interval assignment variant works as follows: Similar to FERRARI-L, nodes are visited in reverse topological order and the interval sets of the neighboring nodes are merged into the interval set $\mathcal{I}'(v)$ for the current vertex v. However, in this new variant, subsequent to merging the interval sets we compute the interval cover comprising at most ck intervals, given a constant $c \geq 1$. This way, more intervals can be stored in the node labels. After the ck-interval cover has been computed, the vertex v is added to a min-heap structure where the nodes are maintained in ascending order of their degree. This procedure continues until the already assigned interval sets sum up to a size of more than B intervals. In this case, the algorithm repeatedly pops the minimum element from the heap and restricts its respective interval set by computing the k-interval cover. This deferred interval set restriction is repeated until the number of assigned intervals again satisfies the size constraint B.

Above procedure leads to a much better utilization of the available space and thus a better quality of the resulting reachability index. The improvement comes at the cost of increased index computation time, in practice the increase is two-fold in the worst-case, negligible in others. Our experimental evaluation suggests that a value of $c = 4$ provides a reasonable tradeoff between efficiency of construction and resulting index quality. This second indexing variant is shown in detail in Algorithm 2. We refer to the algorithm as the *global* variant (FERRARI-G) as in this case the size constraint is satisfied over all vertices – in contrast to the local size constraint of FERRARI-L.

In the next section, we provide more details about our query answering algorithm and additional heuristics that further speed up query processing over the FERRARI index.

V. QUERY PROCESSING AND ADDITIONAL HEURISTICS

The basic query processing over FERRARI's reachability intervals is straightforward and resembles the basic approach of Agrawal et al. [1]: For every node v, the intervals in the set $\mathcal{I}'(v)$ are maintained in sorted order. Then, given a reachability query (G, s, t), it can be determined in $O(\log |\mathcal{I}'(v)|)$ time whether the target id $\pi(t)$ is contained in one of the intervals of the source. The query returns a negative answer ($s \not\sim t$) if the target id lies outside all of the intervals and a positive

[1] In our implementation we require non-adjacent intervals in the set, that is, for $[\alpha_1, \beta_1], [\alpha_2, \beta_2] \in \mathcal{I}$ it must hold $\beta_1 < \alpha_2$. When sets of approximate and exact intervals are merged, the type of the resulting interval is based on several factors. For example, when an exact interval is extended by an approximate interval, the result will be one long approximate range.

Algorithm 2: FERRARI-G(G, B)

Input: directed, acyclic graph G, interval budget $B = kn$, constant $c \geq 1$
Result: set of approximate and exact reachability intervals $\mathcal{I}'(v)$ for every node $v \in V$ s.t. the total number of intervals is upper-bounded by B

begin
  $T \leftarrow$ TREECOVER(G)
  $I_T \leftarrow$ ASSIGNTREEINTERVALS(T)
  $H \leftarrow$ INITIALIZEMINHEAP$()$
  $s \leftarrow 0$ ▷ number of currently assigned intervals
  for $i = n$ **to** 1 **do** ▷ visit nodes in reverse topological order
    $\{v\} \leftarrow \{v \in V \mid \tau(v) = i\}$
    $\mathcal{I}'(v) \leftarrow \{I_T(v)\}$
    foreach $w \in \mathcal{N}^+(v)$ **do**
      $\mathcal{I}'(v) \leftarrow \mathcal{I}'(v) \oplus \mathcal{I}'(w)$ ▷ merge interval sets
    ▷ replace intervals by ck-interval cover
    $\mathcal{I}'(v) \leftarrow ck$-INTERVALCOVER$(\mathcal{I}'(v))$
    $s \leftarrow s + |\mathcal{I}'(v)|$
    if $|\mathcal{I}'(v)| > k$ **then**
      HEAP-PUSH$\big(H, v, |\mathcal{N}^+(v)|\big)$
    while $s > B$ **do**
      $w \leftarrow$ HEAP-POP(H)
      $\mathcal{I}'(w) \leftarrow k$-INTERVALCOVER$(\mathcal{I}'(w))$
      $s \leftarrow s - |\mathcal{I}'(w)| + k$
  return $\big\{\mathcal{I}'(v) \mid v \in V\big\}$

answer if it is contained in one of the exact intervals. Finally, if $\pi(t)$ falls into one of the approximate intervals, the neighbors of s are expanded recursively using a DFS search algorithm. Next, we introduce some heuristics that can further speed up query processing.

A. Seed Based Pruning

It is evident that, in the case of recursive querying, the performance of the algorithm depends on the number of vertices that have to be expanded during the online search. Nodes with a very high outdegree are especially costly as they might lead to a large number of recursive queries. In practice, such high degree nodes are to be expected due to the fact that (i) most of the real-world graphs in our target applications will follow a power-law degree distribution and (ii) the condensation graph obtained from the input graph produces high-degree nodes in many cases because the large strongly connected components usually exhibit a large number of outgoing edges.

To overcome this problem, we propose to determine a set of *seed vertices* $S \subseteq V$ and assign an additional label to every node v in the graph, indicating for every $\sigma \in S$ whether G contains a forward (backward) directed path from v to s.

This labeling scheme works as follows: Every node will be associated with two sets, $S^-(v)$ and $S^+(v)$, such that $S^-(v) := \big\{\sigma \in S \mid \sigma \sim v\big\}$, and $S^+(v) := \big\{\sigma \in S \mid v \sim \sigma\big\}$.

Next, we describe the procedure for assigning the sets S^+: For every node v, we initialize $S^+(v) = \{v\}$ if $v \in S$ and $S^+(v) = \emptyset$ otherwise. We then maintain a FIFO-queue of all vertices, initialized to contain all leaves of the graph. At each step of the algorithm, the first vertex v is removed from the queue. Then, for every predecessor $u, (u, v) \in E$ we set $S^+(u) \leftarrow S^+(u) \cup S^+(v)$. If all successsors of u have been processed, u itself is added to the end. The algorithm continues until all nodes of the graph have been labeled. It is easy to see that above procedure can efficiently be implemented. The approach for assignment of the sets S^- is similar (starting from the root nodes).

Once assigned, the sets can be used by the query processing algorithm in the following way: For a query (G, s, t),

1) if $S^+(s) \cap S^-(t) \neq \emptyset$, then $s \sim t$.
2) if there exists a seed node σ s.t. $\sigma \in S^-(s)$ and $\sigma \notin S^-(t)$, that is, the seed σ can reach s but not t, the query can be terminated with a negative answer ($s \not\sim t$).

In our implementation we choose to elect the s nodes with maximum degree as seed nodes (requiring a minimum degree of 1). The choice of s can be specified prior to index construction, in our experiments we set $s = 32$.

B. Pruning Based on Topological Properties

We enhance the FERRARI index with two additional powerful criteria that allow additional pruning of certain queries. First, we adopt the effective topological level filter that was proposed by Yıldırım et al. for the GRAIL index (see [23] for details). Second, we maintain the topological order $\tau(v)$ of each vertex v for pruning as shown in Equation (9).

Before we proceed to the experimental evaluation of our index, we first give an overview over previously proposed reachability indexing approaches.

VI. RELATED WORK

Due to the crucial role played by reachability queries in innumerable applications, indices to speed them up have been subject of active research. Instead of exhaustively surveying previous results, we briefly describe some of the key proposals here. For a detailed survey, we direct the reader to [21]. In this section, we distinguish between reachability query processing techniques that are able to answer queries using only the label information on nodes specified in the query, and those which use the index to speed up guided online search over the graph.

Before we proceed, it is worth noting that there are two recent proposals that aim to speed up the reachability queries from a different direction compared to the standard graph indexing approaches. First, in [10], authors propose a novel way to compact the graph before applying any reachability index. Naturally, this technique can be used in conjunction with FERRARI, hence we consider it orthogonal to the focus of the present paper. The other proposal is to compress the transitive closure through a carefully optimized word-aligned bitmap encoding of the intervals [19]. The resulting encoding, called PWAH-8, is shown to make the interval labeling technique of Nuutila [14] scale to larger graph datasets. In our experiments, we compare our performance with both Nuutila's Intervals assignment technique as well as the PWAH-8 variant.

A. Direct Reachability Indices

Indices in this category answer a reachability query (G, s, t) using just the labels assigned to s and t. Apart from the classical algorithm [1] described in Section II, another approach based on tree covering [20] focused on sparse graphs (targeting near-tree structures in XML databases), labeling each node with its tree interval and computing the transitive closure of non-tree edges separately.

Apart from trees to cover the graph being indexed, alternative simple structures such as chains and paths have also been used. In a chain covering of the graph, a node u can reach v if both belong to the same chain and u precedes v. In [9] an optimal way to cover the graph with chains in $O(n^3)$ was proposed, later reduced to $O(n^2 + dn\sqrt{d})$, where d denotes the diameter of the graph [4]. Although chain covers typically generate smaller index sizes than the interval labeling and can answer queries efficiently, they are very expensive to build for large graphs. The PathTree index proposed recently [11] combines tree covering and path covering effectively to build an index that allows for extremely fast reachability query processing. Unfortunately, the index size can be extremely large, consuming upto $O(np)$ space, where p denotes the number of paths in the decomposition.

Instead of indexing using covering structures, Cohen et al. [6] introduced 2-Hop labeling which, at each node u, maintains a subset of the node's ancestors and descendants. Using this, reachability queries between s and t can be answered by intersecting the descendant set of s with the ancestors of t. This technique was particularly attractive for query processing within a database system since it can be implemented efficiently using SQL-statements performing set intersections [16]. The main hurdle in using it for large graphs turns out to be its construction – optimally selecting the subsets to label nodes with is an NP-hard problem, and no bounds on the index size can be specified. HOPI indexing [16] tried to overcome these issues by clever engineering, using a divide-and-conquer approach for computing the covering. 3-Hop labeling [12] combines the idea of chain-covering with the 2-Hop strategy to reduce the index size.

B. Accelerating Online Search

From the discussion above, it is evident that accurately capturing the entire transitive closure in a manner that scales to massive size graphs remains a major challenge. Some of the recent approaches have taken a different path to utilize scalable indices that can be used to speed up traditional *online search* to answer reachability queries. In GRIPP [18], the index maintains only one interval per node on the tree cover of the graph, but some nodes reachable through non-tree edges are replicated to improve the coverage.

The recently proposed GRAIL index [22], [23] uses k random trees to cover the condensed graph, generating as many intervals to label each node with. As we already described in Section III, the query processing proceeds by using the labels to quickly determine non-reachability, otherwise recursively querying the nodes underneath in the DAG, resulting in a worst-case query processing performance of $O(k(m + n))$. Although GRAIL was shown to be able to build indices over massive scale graphs quite efficiently, it suffers from the previously discussed drawbacks. In our experiments, we compare various aspects of our FERRARI index against GRAIL which is, until now, the only technique that deals effectively with massive graphs while satisfying a user-specified size-constraint.

VII. Experimental Evaluation

We conducted an extensive set of experiments in order to evaluate the performance of FERRARI in comparison with the state of the art reachability indexing approaches, selected based on recent results. In this paper, we present the results of our comparison with: GRAIL [23], PathTree [11], Nuutila's Intervals [19], and PWAH-8 [19]. For all the competing methods, we obtained original source code from the authors, and set the parameters as suggested in the corresponding publications.

A. Setup

Fortunately, all indexing methods are implemented using C++, making the comparisons fairly accurate without any platform-specific artifacts. All experiments were conducted using a Lenovo ThinkPad W520 notebook computer equipped with 8 Intel Core i7 CPUs at 2.30 GHz, and 16 gigabyte of main memory. The operating system in use was a 64-bit installation of Linux Mint 12 using kernel 3.0.0.22.

B. Methodology

The metrics we compare on are:

1) **Construction time** for each indexing strategy over each dataset. Since the input to all considered algorithms is always a DAG, we do not include the time for computing the condensation graph into our measurements.
2) **Query processing time** for executing 100,000 reachability queries. We consider *random* and *positive* sets of queries and report numbers for both workloads separately.
3) **Index size** in memory that each index consumes. It should be noted that although both FERRARI and GRAIL take as input a size restriction parameter, the resulting size of the index can be quite different. PathTree, (Nuutila's) Intervals and PWAH-8 have no parameterized size, and depend entirely on the dataset characteristics.

C. Datasets

We used the selection of graph datasets (Table Ia) that, over the recent years, has become the benchmark set for reachability indexing work. These graphs are classified based on whether they are small (with 10-100s of thousands of nodes and edges) or large (with millions of nodes and edges), and dense or sparse. Due to lack of space, we refer to the detailed description of these datasets in [23] and [11] and, for the same reason, report results only for a salient subset (the full set of results can be found in [17]). We term these datasets as *benchmark datasets* and present results accordingly in Section VII-D.

In order to evaluate the performance of the algorithms under real-world settings, where massive-scale graphs are

TABLE I
DATASETS USED

Dataset	Type	$\|V\|$	$\|E\|$	Source
ArXiV	small, dense	6,000	66,707	[12]
GO	small, dense	6,793	13,361	[12]
Pubmed	small, dense	9,000	40,028	[12]
Human	small, sparse	38,811	39,816	[13]
CiteSeer	large	693,947	312,282	[23][6]
Cit-Patents	large	3,774,768	16,518,947	[23]
CiteSeerX	large	6,540,401	15,011,260	[23]
GO-Uniprot	large	6,967,956	34,770,235	[23]

(a) Benchmark Datasets

Dataset	$\|V\|$	$\|E\|$	$\|V_C\|$	$\|E_C\|$
GovWild	8,027,601	26,072,221	8,022,880	23,652,610
YAGO2	16,375,503	32,962,379	16,375,503	25,908,132
Twitter	54,981,152	1,963,263,821	18,121,168	18,359,487
Web-UK	133,633,040	5,507,679,822	22,753,644	38,184,039

(b) Web Datasets

encountered, we use additional datasets derived from publicly available sources. These include RDF data, an online social network, and a World Wide Web crawl. To the best of our knowledge, these constitute some of the largest graphs used in evaluating the effectiveness of reachability indices to this date. In the following, we briefly describe each of them, and summarize the key characteristics of these datasets in Table Ib.

- **GovWild** is a large RDF data collection consisting of about 26 million triples representing relations between more than 8 million entities.[2]
- **Yago2** is another large-scale RDF dataset representing an automatically constructed knowledge graph [8]. The version we used contained close to 33 million edges (facts) between 16.3 million nodes (entities).[3]
- The **Twitter** graph [3] is a representative of a large-scale social network. This graph, obtained from a crawl of `twitter.com`, represents the follower relationship between about 50 million users.[4]
- **Web-UK** is an example of a web graph dataset [2]. This graph contains about 133 million nodes (hosts) and 5.5 billion edges (hyperlinks).[5]

We present the results of our evaluation over these web-scale graphs in Section VII-E.

D. Results over Benchmark Graphs

Tables IIa-d and the charts in Figures 1–4 summarize the results for the selected set of benchmark graphs. In the tables, we provide the absolute values – time in milliseconds and index size in KBytes, while the figures help to visualize the relative performance of algorithms over different datasets. In all the tables missing values are marked as "–" whenever a dataset could not be indexed by the corresponding strategy – either due to memory exhaustion or for taking too long to index (timeout set to 1M milliseconds). The best performing strategy for each dataset is shown in bold. For GRAIL we set the number of dimensions as suggested in [23], that is, to 2 for small sparse graphs (Human), 3 for small dense graphs (ArXiV, GO, PubMed) and to 5 for the remaining large graphs. The input parameter value for FERRARI was also set correspondingly for a fair comparison.

1) Index Construction: Table IIa and Figure 1 present the construction time for the various algorithms. The results show that the GRAIL index can be constructed very efficiently on small graphs, irrespective of the density of the graph. On the other hand, the performance of PathTree is highly sensitive to the density of the graph as well as the size. While GRAIL and FERRARI's indexing time increases corresponding to the size of the graphs, PathTree simply failed to complete building for 3 of the larger graphs – Cit-Patents and CiteSeerX due to memory exhaustion, GO-Uniprot due to timeout.

The transitive closure compression algorithms Interval and PWAH-8 can index quite efficiently even the large graphs and their index is also surprisingly compact. A remarkable exception to this is the behavior on the Cit-Patents dataset, which seems to be by far the most difficult graph for reachability indexing. The Interval index failed to process the graph within the given time limit. The related PWAH-8 algorithm finished the labeling only after around 12 minutes and ended up generating the *largest index* in all our experiments (including the indices for the Web graphs). This is rather surprising, as both algorithms were able to index the larger and denser GO-Uniprot.

When compared to other algorithms, the construction times of FERRARI-L and FERRARI-G are highly scalable and are not affected much by the variations in the density of the graph. On all graphs, FERRARI constructs the index quickly while maintaining a competitive index size. For the challenging Cit-Patents dataset, it generates the most compact index among all the techniques considered, and very fast – amounting to a 23x-36x speedup over PWAH-8. Further, FERRARI consistently generates smaller indices than GRAIL, and exhibits comparable indexing time. With a few more clever engineering tricks (e. g., including the PWAH-8-style interval encoding), it should be possible to further reduce the size of FERRARI.

2) Query Processing: Moving on to query processing, we consider random and positive query workloads, with results depicted in Figures 3 and 4 (Tables IIc and IId), respectively. These results help to highlight the consistency of FERRARI in being able to efficiently process both types of queries over all varieties of graphs very efficiently. Although, for really small graphs, PathTree is the fastest as we explained above, it cannot be applied on larger datasets. As graphs get larger, the Interval indexing turns out to be the fastest. This is not very surprising, since Interval materializes the exact transitive closure of the graph. FERRARI-G consistently provides competitive query processing times for both positive as well as random queries over all datasets. As a remarkable result of our experimental evaluation consider the CiteSeerX dataset. In this setting, the Interval index consumes almost twice as much space as the corresponding FERRARI-G index, yet is only faster by 0.04

[2]http://govwild.hpi-web.de/project/govwild-project.html
[3]http://www.mpi-inf.mpg.de/yago-naga/yago/
[4]http://twitter.mpi-sws.org/
[5]http://law.di.unimi.it/webdata/uk-union-2006-06-2007-05/
[6]We use the version provided at http://code.google.com/p/grail/

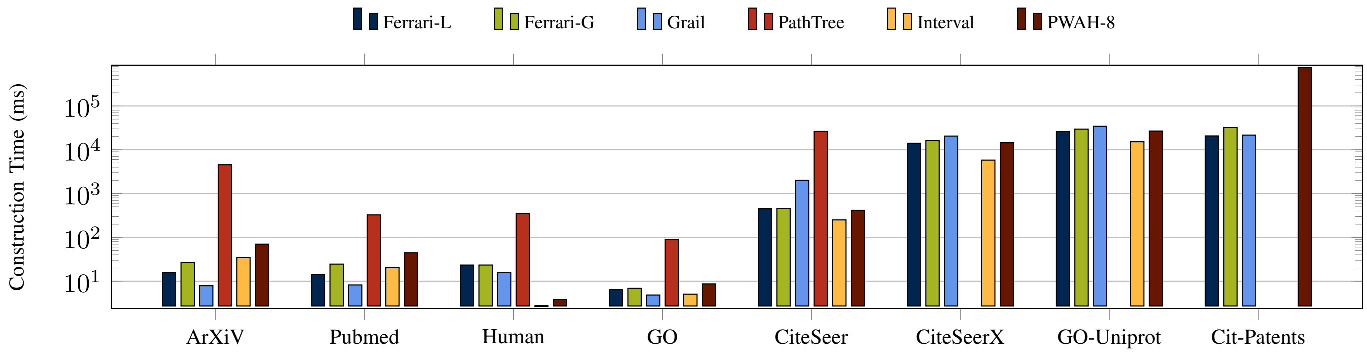


Fig. 1. Index Construction Time

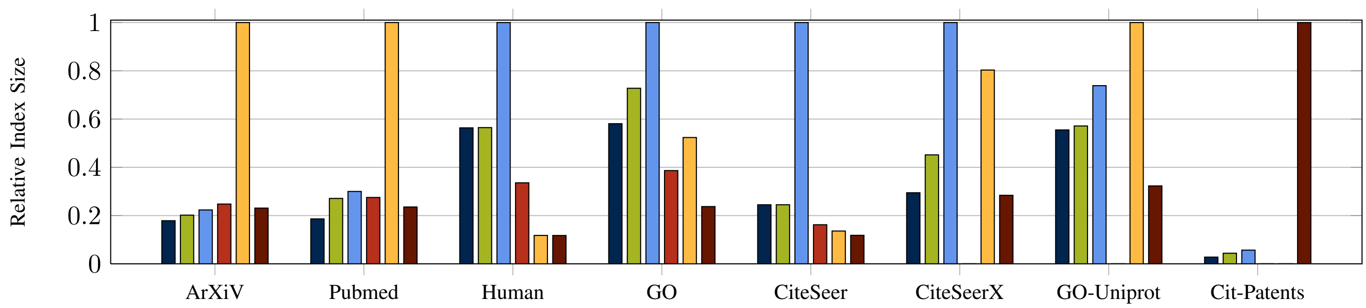


Fig. 2. Index Space Consumption

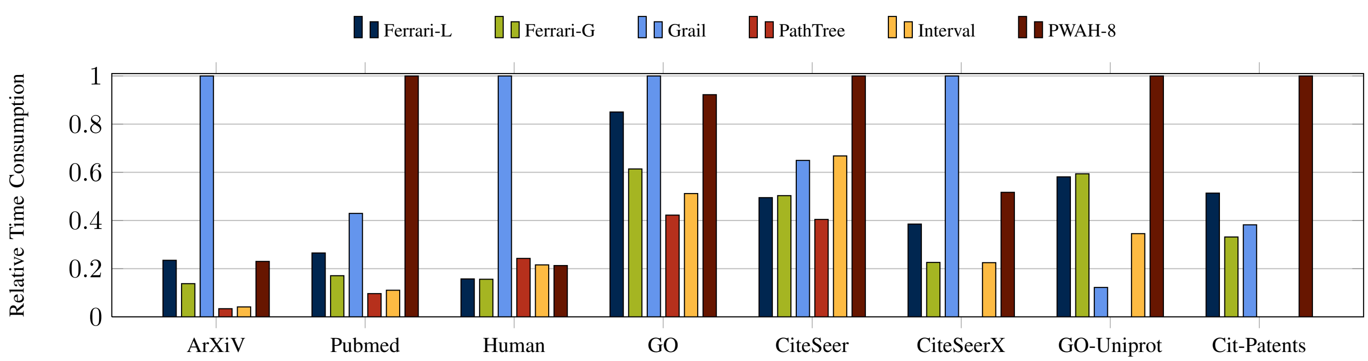


Fig. 3. Query Processing Times for 100k Random Queries

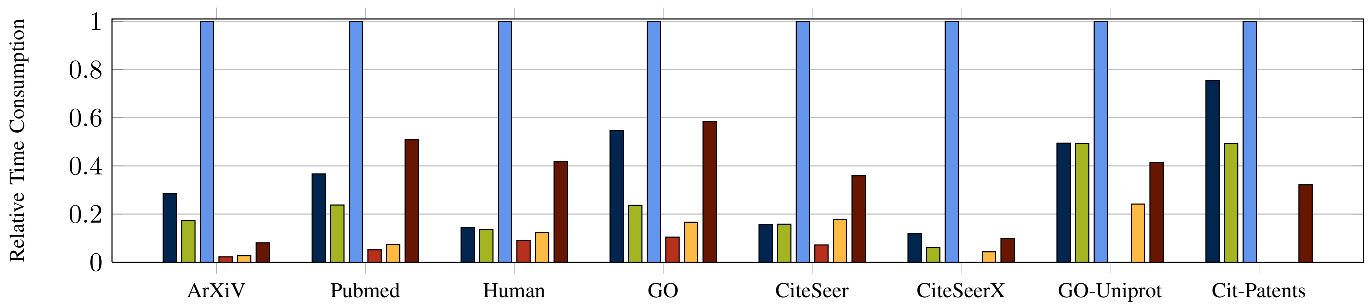


Fig. 4. Query Processing Times for 100k Positive Queries

TABLE II
EXPERIMENTAL EVALUATION ON BENCHMARK DATASETS

Dataset	Ferrari-L	Ferrari-G	Grail	PathTree	Interval	PWAH-8
ArXiV	15.84	26.62	**7.86**	4,537.39	34.54	70.10
Pubmed	14.28	24.54	**8.21**	326.54	20.35	44.41
Human	23.36	23.37	15.93	348.48	**2.70**	3.82
GO	6.48	6.91	**4.83**	89.83	5.06	8.67
CiteSeer	450.12	459.90	2,015.90	26,479.70	**251.10**	416.41
CiteSeerX	14,110.20	16,233.40	20,528.40	–	**5,808.79**	14,444.09
GO-Uniprot	26,105.90	29,611.90	34,518.40	–	**15,213.55**	26,745.61
Cit-Patents	**20,665.50**	32,366.20	21,621.70	–	–	751,984.08

(a) Construction Time (ms)

Dataset	Ferrari-L	Ferrari-G	Grail	PathTree	Interval	PWAH-8
ArXiV	**243.86**	275.33	304.69	338.07	1,364.99	315.24
Pubmed	**283.68**	413.06	457.03	419.03	1,523.83	358.96
Human	768.88	770.30	1,364.45	458.01	160.56	**160.22**
GO	200.37	251.01	344.96	133.30	180.58	**81.86**
CiteSeer	13,933.90	13,934.29	56,925.34	9,221.61	7,733.94	**6,723.36**
CiteSeerX	158,046.72	242,236.08	536,517.27	–	430,913.36	**152,354.44**
GO-Uniprot	429,564.04	442,301.79	571,590.14	–	774,081.33	**249,883.80**
Cit-Patents	**151,631.73**	239,609.23	309,648.94	–	–	5,462,135.76

(b) Index Size (Kb)

Dataset	Ferrari-L	Ferrari-G	Grail	PathTree	Interval	PWAH-8
ArXiV	23.69	13.91	100.92	**3.41**	4.17	23.22
Pubmed	7.58	4.88	12.27	**2.76**	3.16	28.58
Human	**0.78**	**0.78**	4.98	1.21	1.07	1.06
GO	4.10	2.96	4.83	**2.04**	2.47	4.45
CiteSeer	6.13	6.24	8.05	**5.01**	8.28	12.39
CiteSeerX	15.88	9.31	41.23	–	**9.27**	21.32
GO-Uniprot	28.30	28.92	**5.94**	–	16.82	48.70
Cit-Patents	778.09	**502.20**	578.83	–	–	1,514.91

(c) Query Processing Performance (ms), 100k random queries

Dataset	Ferrari-L	Ferrari-G	Grail	PathTree	Interval	PWAH-8
ArXiV	62.64	37.98	220.31	**4.94**	5.95	17.74
Pubmed	31.31	20.28	85.38	**4.42**	6.21	43.58
Human	2.08	1.96	14.48	**1.30**	1.79	6.07
GO	10.72	4.64	19.59	**2.04**	3.26	11.43
CiteSeer	13.37	13.47	85.22	**6.12**	15.17	30.60
CiteSeerX	82.76	43.06	700.49	–	**30.38**	69.21
GO-Uniprot	65.00	64.72	131.46	–	**31.76**	54.55
Cit-Patents	4,086.21	2,667.38	5,409.82	–	–	**1,739.30**

(d) Query Processing Performance (ms), 100k positive queries

TABLE III
EXPERIMENTAL EVALUATION ON WEB DATASETS

Dataset	Ferrari-L	Ferrari-G	Grail	Interval	PWAH-8
YAGO2	27,713.50	26,865.30	17,163.00	**5,844.87**	9,236.71
GovWild	12,998.80	18,045.30	**6,756.67**	15,060.55	20,703.06
Twitter	13,065.40	13,897.20	9,717.39	36,480.57	**8,219.09**
Web-UK	17,604.90	18,754.40	**12,275.90**	–	166,531.10

(a) Construction Time (ms)

Dataset	Ferrari-L	Ferrari-G	Grail	Interval	PWAH-8
YAGO2	372,150.70	448,139.06	575,701.28	182,962.96	**137,878.21**
GovWild	**206,475.06**	297,724.03	282,054.38	921,605.13	311,359.35
Twitter	384,049.21	384,368.44	637,072.31	**85,648.12**	97,859.81
Web-UK	616,486.63	647,050.45	799,932.80	–	**266,342.83**

(b) Index Size (Kb)

Dataset	Ferrari-L	Ferrari-G	Grail	Interval	PWAH-8
YAGO2	12.00	10.95	16.56	**10.45**	12.62
GovWild	60.27	31.77	42.62	**13.33**	33.30
Twitter	**5.55**	5.65	19.27	8.66	10.32
Web-UK	**19.11**	19.29	39.21	–	20.45

(c) Query Processing Performance (ms), 100k random queries

Dataset	Ferrari-L	Ferrari-G	Grail	Interval	PWAH-8
YAGO2	59.39	38.43	97.99	**21.7**	44.19
GovWild	171.46	85.12	228.98	**29.84**	126.96
Twitter	10.24	**10.18**	76.07	18.21	36.01
Web-UK	25.54	**18.01**	95.25	–	43.73

(d) Query Processing Performance (ms), 100k positive queries

milliseconds for random and 12.68 milliseconds for positive queries.

E. Evaluation over Web Datasets

As we already pointed out in the introduction, our goal was to develop an index that is both compact and efficient for use in many analytics tasks when the graphs are of web-scale. For this reason, we have collected graphs that amount to up to 5 billions of edges before computing the condensation graph. These graphs are of utmost importance because the resulting DAG exhibits special properties absent from previously considered benchmark datasets. In this section, we present the results of this evaluation. Due to its limited scalability, we do not use PathTree index in these experiments. Also, through initial trial experiments, we found that for GRAIL the suggested parameter value of $k = 5$ does not appear to be the optimal choice, so instead we report the results with the setting $k = 2$, which is also used for FERRARI. The summary of results is provided in Table III.

1) Index Construction: When we consider the index construction statistics in Tables IIIa and IIIb, it seems that there is no single strategy that is superior across the board. However, a careful look into these charts further emphasizes the superiority of FERRARI in terms of its consistent performance. While GRAIL can be constructed fast, its size can be quite large (e. g., in the case of Twitter). On the other hand, PWAH-8 can take an order of magnitude more time to construct than FERRARI as well as GRAIL as we notice for Web-UK. In fact, the Interval index which is much smaller than FERRARI for Twitter and Yago2, fails to complete within the time allotted for the Web-UK dataset. In contrast, FERRARI and GRAIL are able to handle any form of graph easily in a scalable manner.

As an additional note, the index size of Interval is sometimes smaller than FERRARI which seems to be counterintuitive at first glance. The reason for this lies in the additional information maintained at every node by FERRARI, for use in early pruning heuristics. In relatively sparse datasets like Twitter and Yago2 the overhead of this extra information tends to outweigh the gains made by interval merging. If needed, it is possible to turn off these heuristics easily to get around this problem. However, we retain them across the board to avoid dataset-specific tuning of the index.

2) Query Processing: Finally, we turn our attention to the query processing performance over Web-scale datasets. As the results summarized in Tables IIIc and IIId demonstrate, the FERRARI variants and the Interval index provide the fastest index structures. For Web-UK and Twitter, the FERRARI variants outperform all other approaches. The performance of GRAIL, as predicted, and PWAH-8 are inferior in comparison to Interval and FERRARI-G when dealing with both random and positive query loads.

In summary, our experimental results indicate that FERRARI, in particular the global budgeted variant, is highly scalable and consistent in being able to index a wide variety of graphs, and can answer queries extremely fast[7]. This, we believe, provides a compelling reason to use FERRARI-G on a wide spectrum of graph analytics applications involving large to massive-scale graphs.

VIII. Conclusions & Outlook

In this paper, we presented an efficient and scalable reachability index structure, FERRARI, that allows to directly control the query processing/space consumption tradeoff via a user-specified restriction on the resulting index size. The two different variants of our index allow to either specify the maximum size of the resulting node labels, or to impose a global size constraint which allows the dynamic allocation of budgets based on the importance of individual nodes. Using a theoretically sound technique, FERRARI assigns a mixture of exact and approximate identifier ranges to nodes so as to speed up both random as well as positive reachability queries. Using an extensive array of experiments, we demonstrated that the resulting index can scale to massive-size graphs quite easily, even when some of the state of the art indices fail to complete the construction. FERRARI provides very fast query execution, demonstrating substantial gains in processing time of both random and positive queries when compared to the previous state-of-the-art method, GRAIL.

Results presented in this paper open up a range of possible future directions that we plan to pursue. First of all, we would like to integrate FERRARI in a number of graph analytics algorithms, starting from shortest path computation to more complex graph mining techniques (e. g. Steiner trees) in order to study its impact on the performance of these expensive operations. We also would like to pursue further optimizations of the FERRARI index by, for instance, the use of an improved tree covering algorithm designed to work synergistically with FERRARI to generate high quality intervals to begin with, and the use of compression techniques for interval encoding afterwards.

Acknowledgements

The work of Srikanta Bedathur is supported by the Max Planck Society (MPG) and the Dept. of Science and Technology (DST), India, under the cooperation scheme of the Max Planck Partner Group for Large-scale Graph Mining.

References

[1] Rakesh Agrawal, Alexander 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*, pages 253–262. ACM, 1989.

[2] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. A Large Time-Aware Graph. *SIGIR Forum*, 42(2):33–38, 2008.

[3] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *In Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[4] Yangjun Chen and Yibin Chen. An Efficient Algorithm for Answering Graph Reachability Queries. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 893–902. IEEE Computer Society, 2008.

[7] In a light-hearted view, our index shares the characteristics of its namesake F-1 racing car, in the sense that although not being the most powerful in the fray, it consistently tends to win on all circuits under diverse input conditions.

[5] Edith Cohen. Size-Estimation Framework with Applications to Transitive Closure and Reachability. In *Journal of Computer and System Sciences*, volume 55, pages 441–453, Orlando, FL, USA, December 1997. Academic Press, Inc.

[6] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and Distance Queries via 2-Hop Labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM/SIAM, 2002.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[8] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 2013.

[9] H. V. Jagadish. A Compression Technique to Materialize Transitive Closure. *ACM Transactions on Database Systems (TODS)*, 15(4):558–598, December 1990.

[10] Ruoming Jin, Ning Ruan, Saikat Dey, and Jeffrey Xu Yu. SCARAB: Scaling Reachability Computation on Large Graphs. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[11] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. Path-Tree: An Efficient Reachability Indexing Scheme for Large Directed Graphs. *ACM Transactions on Database Systems (TODS)*, 36(1):7, 2011.

[12] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 3-HOP: A High-Compression Indexing Scheme for Reachability Query. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 813–826. ACM, 2009.

[13] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently Answering Reachability Queries on Very Large Directed Graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 595–608. ACM, 2008.

[14] Esko Nuutila. An Experimental Study on Transitive Closure Representations. Technical Report TKO-B134, Helsinki University of Technology, 1996.

[15] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'02, pages 81–90, New York, NY, USA, 2002. ACM.

[16] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. HOPI: An Efficient Connection Index for Complex XML Document Collections. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004)*, LNCS 2992, pages 237–255. Springer, 2004.

[17] Stephan Seufert, Avishek Anand, Srikanta Bedathur, and Gerhard Weikum. High-Performance Reachability Query Processing under Index Size Restrictions. *arXiv:1211.3375 [cs.DB]*, 2012.

[18] Silke Trißl and Ulf Leser. Fast and Practical Indexing and Querying of Large Graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 845–856. ACM, 2007.

[19] Sebastiaan J. van Schaik and Oege de Moor. A Memory Efficient Reachability Data Structure Through Bit Vector Compression. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 913–924. ACM, 2011.

[20] Haixun Wang, Hao He, Jun Yang, Philip S. Yu, and Jeffrey Xu Yu. Dual Labeling: Answering Graph Reachability Queries in Constant Time. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, page 75. IEEE Computer Society, 2006.

[21] Jeffrey Xu Yu and Jiefeng Cheng. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, chapter 6: Graph Reachability Queries: A Survey, pages 181–215. Springer, 2010.

[22] Hilmi Yıldırım, Vineet Chaoji, and Mohammed J. Zaki. GRAIL: Scalable Reachability Index for Large Graphs. In *Proceedings of the VLDB Endowment (PVLDB)*, volume 3, pages 276–284. VLDB Endowment, 2010.

[23] Hilmi Yıldırım, Vineet Chaoji, and Mohammed J. Zaki. GRAIL: A Scalable Index for Reachability Queries in Very Large Graphs. *The VLDB Journal*, 21:509–534, 2012.