Adjacency Queries in Dynamic Sparse Graphs

Łukasz Kowalik*

Abstract

We deal with the problem of maintaining a dynamic graph so that queries of the form "*is there an edge between u and v?*" are processed fast. We consider graphs of bounded arboricity, i.e., graphs with no dense subgraphs, like for example planar graphs. Brodal and Fagerberg [WADS'99] described a very simple linear-size data structure which processes queries in constant worst-case time and performs insertions and deletions in $O(1)$ and $O(\log n)$ amortized time, respectively. We show a complementary result that their data structure can be used to get $O(\log n)$ worst-case time for query, $O(1)$ amortized time for insertions and $O(1)$ worst-case time for deletions. Moreover, our analysis shows that by combining the data structure of Brodal and Fagerberg with efficient dictionaries one gets $O(\log \log \log n)$ worst-case time bound for queries and deletions and $O(\log \log \log n)$ amortized time for insertions, with size of the data structure still linear. This last result holds even for graphs of arboricity bounded by $O(\log^k n)$, for some constant k.

Keywords: data structures, graph algorithms, adjacency, orientation, dynamic

1 Introduction

In the present paper we study fully dynamic graphs, i.e., graphs which change in time by means of inserting and removing edges (it is straightforward to extend our results for the situation when also vertices may be inserted and removed). Such a setting raises a natural question: how to store the structure of the graph in memory so that some kind of information can be retrieved fast. More specifically, we focus on the most basic sort of information about graph: adjacency. In other words we allow for processing queries of the form *Are vertices u and v adjacent?*. When the graph under consideration is dense, e.g. it has $\Omega(n^2)$ edges[1] there is a trivial and efficient solution: store an adjacency matrix. Then both the updates and the queries take constant time. However, for big sparse graphs, like planar graphs, such the approach may be considered as unacceptable because of huge memory requirements compared to the actual size of the graph stored. Hence we consider only data structures of linear space

*Institute of Informatics, Warsaw University, Banacha 2, 02-097, Warsaw, Poland. This work was partially done when the author was a postdoc in Max-Planck-Institute für Informatik, Saarbrücken, Germany. E-mail: `lkowalik@mimuw.edu.pl`. Supported in part by KBN grant 4T11C04425.

[1] Throughout the paper n and m denote the number of vertices and edges, respectively.

complexity. Then one can use another classic data structure: adjacency lists. Unfortunately, in this case time needed to process a query may be too large, unless there is some bound on the vertices' degrees in the graph. In order to fix that problem in the case of planar graphs, Chrobak and Eppstein [5] used the fact that edges of an undirected planar graph can be oriented so that at most 3 edges leave every vertex. They noticed that it suffices to store the adjacency lists of the resulting directed graph – then u and v are adjacent in the original graph if and only if u is in the adjacency list of v or vice versa. However, Chrobak and Eppstein considered only static graphs.

The dynamic case was studied by Brodal and Fagerberg [4]. They consider more general class than planar graphs – graphs with arboricity bounded by some constant c. *Arboricity* of graph G, denoted as $\mathrm{arb}(G)$, is the smallest number of forests needed to cover all edges of G. A theorem by Nash-Williams [13] says that arboricity is equal to $\max_J \lceil |E(J)|/(|V(J)|-1)\rceil$ where J is any subgraph of G with $|V(J)| \geq 2$ vertices and $|E(J)|$ edges. Intuitively, graphs of bounded arboricity are uniformly sparse, i.e., they do not have dense subgraphs. (In particular, planar graphs have arboricity 3.) Brodal and Fagerberg show how to maintain a bounded outdegree orientation of such graphs. Clearly this allows for processing adjacency queries in constant time. They show that their update algorithm is asymptotically optimal (see Section 2 for details). However, tight time complexity analysis of their approach still remains open. The authors were able to show that when orientation with outdegree Δ is maintained, $\Delta \geq 4c$, the amortized time per operation is constant for insertions and $O(\log n)$ for deletions. Another analysis gives $O(1)$ worst-case deletion time and $O(\log n)$ amortized insertion time. These results have applications in bounded length shortest path oracles [10] and, more surprisingly, graph coloring [9].

Our Results In this paper we extend the work of Brodal and Fagerberg [4] by using a slightly different approach. Instead of maintaining outdegrees in the orientation bounded by a constant and trying to reduce the update time, we ask how one can bound the outdegrees, while the amortized update time is constant. However, since the algorithm of Brodal and Fagerberg is asymptotically optimal, there is no need for designing a new one. We show that when their algorithm is supposed to maintain orientation with outdegree Δ, for $\Delta \geq 4c(\lfloor \log cn \rfloor + 1)$, insertions take $O(1)$ amortized time and deletions $O(1)$ worst-case time (recall that c is the bound on arboricity). Clearly, this allows for processing adjacency queries in $O(\log n)$ worst-case time when the arboricity is bounded. Note that in the applications in which we are interested in the total time of the whole sequence of operations, like for example when the orientation is used as a data structure in some algorithm, this is optimal when the updates are frequent compared to queries, i.e., the ratio of number of updates to the number of queries is $\Omega(\log n)$.

Dictionary Approach Another natural approach to our problem is storing the information about the edges of the dynamic graph in a dictionary, i.e., a data structure which enables adding, removing and finding keys (elements). In our case these elements are edges of the graph. For convenience, we will assume that vertices of the graph are enumerated from 1 to n and that a pair of vertices describing an edge can be stored in one word of memory (it is common in analysis of graph algorithms to assume that each vertex can be stored in $O(1)$ words of memory).

Dietzfelbinger et al. [6] show a linear-size randomized dictionary based on hashing with $O(1)$ worst-case time lookups and $O(1)$ amortized expected time updates.

Without randomization the dynamic dictionary problem seems to be harder: Mehlhorn, Näher and Rauch [11] show that $\Omega(m \log\log m)$ time is needed for m insertions in some deterministic model of linear-space dictionary that encompasses both hashing strategies and search trees, which are the two most efficient solutions to the dictionary problem. However, in our case, when the size of the universe (number of possible edges) is rather small, there is a solution very close to this lower bound. Namely, the dynamization technique by Andersson and Thorup [2] applied to the exponential search trees by Beame and Fich [3], achieves the $O(\log\log m \cdot \frac{\log\log U}{\log\log\log U})$ worst-case time bound for both lookups and updates, where m is the number of keys stored and U is the maximal key stored in dictionary (it is assumed that the dictionary stores integers). Note that in the case of storing edges $U \leq n^2$, which gives us a bound of $O\left(\frac{(\log\log n)^2}{\log\log\log n}\right)$ on the worst-time complexity of each operation.

Our Results Combined With Dictionary Approach The main asset of our result is simplicity of the algorithm with its asymptotic optimality in situations when the updates are very frequent. However, by combining our approach with deterministic dictionaries one obtains theoretically extremely efficient solution: $O(\log\log\log n)$ worst-case time for query and edge deletion and $O(\log\log\log n)$ amortized time for insertion, when the dynamic graph under consideration has arboricity bounded by $O(\log^k n)$, for some constant k. Hence we get the best known deterministic method for storing adjacency of sparse graphs in the situation when queries and updates appear similarly often. This should be compared with the already mentioned $\Omega(\log\log n)$ lower bound [11] for amortized insertion time in the dynamic deterministic dictionary which stores all the edges of the graph.

Adjacency Labeling Schemes Kannan et al. [8] introduced the idea of a labeling scheme, where each vertex of a graph is assigned a label so that adjacency of two vertices can be decided based only on their labels. We note that having an orientation of a graph G one gets a labeling scheme for G in which the label of a vertex v is the number of v together with the numbers of endvertices of the edges leaving v. It follows that the performance bounds from both the paper of Brodal and Fagerberg and the present paper may be reformulated as the relevant space and time bounds for dynamic labeling scheme in graphs of bounded arboricity. The problem of maintaining dynamic adjacency labels was also considered recently by Morgan [12], who focused on the case of line graphs.

Comparison The above discussion shows that there are two leading approaches for the problem of maintaining adjacency of a dynamic graph of bounded arboricity: randomized dynamic hashing and bounded outdegree orientations. We point out the following assets of the orientation approach:

- deterministic algorithm,
- the information is distributed evenly over the nodes of the graph (labelling scheme).

2 The Algorithm of Brodal and Fagerberg

In this section we sketch the approach from the paper [4]. We will use the following notions. *Orientation* of an undirected graph G is a directed graph $\vec{G}$ obtained from G by replacing each edge, say uv, either by arc (u, v) or by arc (v, u). We will also say that $\vec{G}$ is a d-*orientation* when the outdegree of every vertex does not exceed d. Let $\vec{G}_1, \vec{G}_2, \ldots, \vec{G}_t$ be a sequence of orientations. We say that edge uv is *reoriented* in graph G_i when uv has different orientations in G_{i-1} and G_i. Each such pair (uv, i) is called a *reorientation*. However, the term reorientation with respect to an algorithm will mean simply an operation of reversing the orientation of an edge.

The algorithm of Brodal and Fagerberg works as follows. Let Δ be the bound on vertices' outdegrees that has to be maintained. Then when an edge is removed from the graph the algorithm simply removes its oriented counterpart. After adding an edge the algorithm orients it arbitrarily. Next, as long as the orientation contains a vertex of outdegree larger than Δ such a vertex x is picked and the orientation of all the edges leaving x is reversed.

Clearly, the total time used by the above algorithm to maintain Δ-orientation during a sequence of updates is linear in the length of the sequence added to the number of reorientations performed. The following lemma states that the above algorithm is asymptotically optimal with respect to the number of reorientations performed. It follows that it is also optimal in running time since its time complexity is linear in the number of reorientations and any algorithm which maintains orientation has to make reorientations.

Lemma 2.1 (Brodal and Fagerberg [4]). *Let σ be a sequence of insertions and deletions on an initially empty graph. Let G_i be the graph after i-th operation and let k denote the number of edge insertions.*

If there exists a sequence $\vec{G}_0, \vec{G}_1, \ldots, \vec{G}_{|\sigma|}$ of δ-orientations with at most r edge reorientations in total, then the algorithm performs at most

$$(k + r)\frac{\Delta + 1}{\Delta + 1 - 2\delta}$$

edge reorientations in total on the sequence σ, provided $\Delta \geq 2\delta$. □

3 Analysis for Logarithmic Outdegrees

Lemma 2.1 implies that in order to bound the amortized time of insert operations in Brodal-Fagerberg algorithm it suffices to construct for an arbitrary sequence of edge deletions and insertions, a sequence of orientations of the relevant graphs with a small number of edge reorientations. However, in what follows we show that when the bound on outdegree is logarithmic in the length of the sequence, then there exists a sequence of orientations with no single reorientation.

Lemma 3.1. *Any graph with arboricity c can be c-oriented.*

Proof. The orientation can be found by decomposing the graph into c forests, choosing a root in each tree and orienting edges of each tree towards its root. □

Lemma 3.2. *Let $G_1, \ldots, G_t$ be any sequence of graphs with arboricity bounded by c. Then there exists a sequence $\vec{G}_1, \ldots, \vec{G}_t$ of $c(\lfloor \log t \rfloor + 1)$-orientations with no edge reorientations.*

Proof. The proof is by the induction on t. For $t = 1$ the lemma is equivalent to Lemma 3.1. Now assume $t > 1$ and let $k = \lfloor t/2 \rfloor$. Let $\vec{G}'_1, \ldots, \vec{G}'_k$ be a sequence of $c(\lfloor \log k \rfloor + 1)$-orientations of graphs $G_1, \ldots, G_k$ with no reorientations, which exists by the induction hypothesis. Similarly, when $k + 2 \leq t$, from the induction hypothesis we get $\vec{G}'_{k+2}, \ldots, \vec{G}'_t$ — a sequence of $c(\lfloor \log k \rfloor + 1)$-orientations of graphs $G_{k+2}, \ldots, G_t$ with no reorientations.

We set $\vec{G}_{k+1}$ to be a c-orientation of graph G_{k+1} obtained by Lemma 3.1. Now consider any $i \neq k + 1$ and an edge $uv \in G_i$. If $uv \in G_{k+1}$, we orient uv in $\vec{G}_i$ the same as in $\vec{G}_{k+1}$. Otherwise we orient uv in $\vec{G}_i$ the same as in $\vec{G}'_i$. Clearly, for any vertex $v \in \vec{G}_i$ we have $\mathrm{outdeg}_{\vec{G}_i}(v) \leq \mathrm{outdeg}_{\vec{G}'_i}(v) + \mathrm{outdeg}_{\vec{G}_{k+1}}(v) \leq c(\lfloor \log k \rfloor + 1) + c \leq c(\lfloor \log t \rfloor + 1)$. Finally, we consider any edge uv which is present in two successive graphs G_i, G_{i+1} and we will show that its orientation is the same. If $uv \in G_{k+1}$ the orientation of uv in both $\vec{G}_i$ and $\vec{G}_{i+1}$ is the same as in $\vec{G}_{k+1}$. Otherwise the orientations of uv in $\vec{G}_i$ and $\vec{G}_{i+1}$ are the same as in $\vec{G}'_i$ and $\vec{G}'_{i+1}$, hence they are the same. □

In the following lemma we show that when one allows reorientations, the bound on outdegrees becomes independent from the length of the sequence.

Lemma 3.3. *Let $G_1, \ldots, G_t$ be any sequence of graphs with arboricity bounded by c and let α be any integer. Then there exists a sequence $\vec{G}_1, \ldots, \vec{G}_t$ of $c(\lfloor \log \alpha n \rfloor + 1)$-orientations with at most ct/α reorientations.*

Proof. We partition the sequence $G_1, \ldots, G_t$ into blocks of length αn. For each $i = 0, \ldots, \lfloor t/(\alpha n) \rfloor$ graphs in block $G_{i\alpha n+1}, G_{i\alpha n+2}, \ldots, G_{(i+1)\alpha n}$ are $c(\lfloor \log \alpha n \rfloor + 1)$-oriented using Lemma 3.2.

Clearly, reorientations may appear only immediately after the end of a block, i.e., in graphs $\vec{G}_{i\alpha n+1}$ for $i > 0$. Since there are $\lfloor t/(\alpha n) \rfloor$ such graphs and each of them contains at most $c(n-1)$ edges, hence the total number of reorientations does not exceed ct/α. $\square$

Corollary 3.4. *Consider a sequence of edge insertions and deletions performed on an initially empty graph such that after each operation the resulting graph has arboricity bounded by* c*. Let* k *be the number of insertions and let* α *be an integer. When the algorithm of Brodal and Fagerberg is set to maintain orientation with outdegree at most* $\Delta = 4c(\lfloor \log \alpha n \rfloor + 1)$ *then it performs at most* $2(k + 2kc/\alpha)$ *edge reorientations.*

Proof. Since the number of deletions does not exceed the number of insertions, the sequence of operations has length at most $2k$. Then the corollary follows immediately from lemmas 2.1 and 3.3. $\square$

By setting α equal to the bound on arboricity c we get the following theorem.

Theorem 3.5. *The algorithm of Brodal and Fagerberg can maintain* $O(c \log n)$*-orientation of an initially empty dynamic graph with arboricity bounded by* c *with constant amortized insertion time and constant worst-case deletion time.* $\square$

4 Applying Deterministic Dictionaries

In the previous section we analyzed the time complexity of the algorithm of Brodal and Fagerberg maintaining $O(c \log n)$-orientation of a dynamic graph with arboricity bounded by c. Now consider an implementation of this algorithm, in which for each vertex v there is a separate dictionary storing the ends of the edges leaving v. Moreover, let the bound on arboricity be $c = O(\log^k n)$, for some constant k. Theorem 3.5 implies that each dictionary stores $O(\log^{k+1} n)$ keys and each edge insertion causes amortized constant number of dictionary insertions and worst-case constant number of dictionary deletions (namely 2).

In order to get the best bounds, we will use the dictionary obtained by applying the dynamization technique by Andersson and Thorup [2] to fusion trees by Andersson [1] and Fredman and Willard [7]. Let ω denote the memory word length (in bits) and $\#_{\text{keys}}$ be the number of keys stored. Then, as stated in [2], this dictionary performs both the lookups and updates in worst-case time $O(\log \log \#_{\text{keys}} + \frac{\log \#_{\text{keys}}}{\log \omega})$. Since in our case the word length is $\omega = O(\log n)$ and $\#_{\text{keys}} = O(\log^{k+1} n)$, we get the bound of $O(\log \log \log n)$ for all the three operations performed on a single dictionary. This gives us $O(\log \log \log n)$ worst-case time for query and edge deletion and $O(\log \log \log n)$ amortized time for insertion.

Finally, we note that in practical situations it may be sufficient to use dictionaries which are simpler and easier in implementation, like splay trees for which we get $O(\log \log n)$ amortized time bounds. Similarly, when one considers weaker model than the word RAM, redblack trees can be used as dictionaries, giving $O(\log \log n)$ worst-case time for query and edge deletion and $O(\log \log n)$ amortized time for insertion.

References

[1] A. Andersson. Faster deterministic sorting and searching in linear space. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science (FOCS '96)*, pages 135–141, 1996.

[2] A. Andersson and M. Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 335–342. ACM Press, 2000.

[3] P. Beame and F. F. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. System Sci.*, 65:38–72, 2002.

[4] G. S. Brodal and R. Fagerberg. Dynamic representations of sparse graphs. In *Proc. 6th Int. Workshop on Algorithms and Data Structures (WADS'99)*, volume 1663 of *LNCS*, pages 342–351, 1999.

[5] M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, 1991.

[6] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, and F. Meyer auf der Heide. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.

[7] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. System Sci.*, 47:424–436, 1993.

[8] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 334–343, New York, NY, USA, 1988. ACM Press.

[9] Ł. Kowalik. Fast 3-coloring triangle-free planar graphs. In S. Albers and T. Radzik, editors, *Proc. 12th Annual European Symposium on Algorithms (ESA 2004)*, volume 3221 of *Lecture Notes in Computer Science*, pages 436–447. Springer-Verlag, 2004.

[10] Ł. Kowalik and M. Kurowski. Oracles for bounded length shortest paths in planar graphs. *ACM Trans. Algorithms*, 2(3):335–363, 2006.

[11] K. Mehlhorn, S. Näher, and M. Rauch. On the complexity of a game related to the dictionary problem. *SIAM J. Comput.*, 19(5):902–906, 1990.

[12] D. Morgan. A dynamic implicit adjacency labelling scheme for line graphs. In *Proc. 9th Int. Workshop on Algorithms and Data Structures (WADS'05)*, volume 3608 of *LNCS*, pages 294–305, 2005.

[13] C. S. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39:12, 1964.