

Department of Computer Science
Faculty of Mathematics and Physics
Comenius University, Bratislava

Ivona Bezáková

**Compact Representations of Graphs
and Adjacency Testing**

Master Thesis

Advisor:
Branislav Rován

April 2000

Contents

1	Introduction	2
2	Basic Definitions	4
2.1	Notation	4
2.2	Models of Representation	5
2.3	Measures of Complexity	6
2.4	Straightforward Representations	6
3	Related Work	8
4	Hashing Algorithms	9
4.1	Data Structures	9
4.1.1	Static Dictionary	9
4.1.2	Dynamic Dictionary	12
4.2	Applications to Graphs	14
5	Representation by Distance	18
5.1	Labeling by Distinguishing Set	18
5.2	Recursive Distance-Labeling	19
5.3	Properties of RDL and LDS	21
5.4	A Note on Digraphs	28
6	Pseudoarboricity and Density	29
6.1	More Definitions and Basic Theorems	29
6.2	Computing the Pseudoarboricity	30
6.3	Approximation Algorithm	37
7	Summary of Open Problems	39

1 Introduction

A succinct representation of a graph is a widely studied problem. A number of criteria can be used to determine the succinctness of the representation. We examined the representation of a graph in these two aspects:

1. The space complexity of the representation.
2. The time complexity of the basic “graph-operations” (usually the adjacency test of two given vertices, eventually the update operations, for example the adding or removing of an edge).

On the one hand, one needs to represent a graph in as few bits as possible, on the other hand, fast manipulation with the compressed structure is required. The two most common representations of a graph – an adjacency matrix and an adjacency list show the contrast between these two aspects: The adjacency matrix is an optimal representation with respect to the time complexity of an adjacency test of two given vertices; however, if n is the number of the vertices in the graph to be represented, the space complexity of this representation is always $O(n^2)$ bits, regardless of the number of edges m . On the other hand, the adjacency list representation is space-optimal, but the adjacency test takes more time than it is necessarily needed.

The aim of this thesis was to find a representation as close as possible to the optimal space complexity $O(m \log n)$ bits, and to the optimal time complexity of the adjacency test (and the update operations) $O(\log n)$ bit-operations.

The structure of this work is as follows: The Chapter 2 introduces the basic definitions, notions and models of computation used in the thesis.

Several authors studied the problem of representation of graphs from the same or related point of view. Their results are summarized in Chapter 3.

In Chapter 4 we present a representation using the technique of hashing: We can store a graph of n vertices and m edges in $O(m \log n)$ bits and the adjacency test can be performed in $O(\log n \log \log n)$ bit-operations. Further, the “list-of-all-neighbors” procedure takes $O((k + \log \log n) \log n)$ time (bit-operations), where k is the number of the neighbors to be listed. This is the only representation presented here which works also for the dynamic graphs, i.e. graphs which are changing in time. In this case the space complexity remains $O(m \log n)$ and the time complexities of the adjacency test and the list-of-all-neighbors are $O(\log n \log \log n)$ and $O((k + \log \log n) \log n)$, respectively, and the expected amortized time complexity of the update operations is $O(\log n \log \log n)$.

A new representation based on computing the distance from all vertices to some given vertex is suggested in Chapter 5. We show that this representation is optimal (in both time and space complexities) for the family of planar graphs and we conjecture that it is optimal for the family of graphs of bounded arboricity too.

The Chapter 6 deals with the problem of computing the arboricity of a graph. This problem is motivated by difficulties encountered when we wanted to find a suitable representation of dynamic graphs of bounded arboricity. We developed a new technique based on orienting the edges of an undirected graph. This technique can be easily understood and it resulted in an algorithm running in the same time complexity as the best known algorithm for this problem (which uses non-trivial algebraic structures — graphic matroids).

2 Basic Definitions

In this chapter we present basic terminology used in this work. We give terminology of graph theory and corresponding notation, especially of terms not so widely used or not standardized in computer science. We extend the definitions of a connected component, a tree, etc. to directed graphs as well.

2.1 Notation

Let $G = (V, E)$ be a graph. If G is directed (called a *digraph*), E consists of ordered pairs (u, v) , $u, v \in V$, such that there is an *arc* from u to v in G . Otherwise, if G is undirected and there is an *edge* between u and v , both pairs (u, v) and (v, u) are in E . We denote an edge of an undirected graph by (u, v) .

Let $v \in V$ be an arbitrary vertex. The most important notations and corresponding terms (the emphasized words) are presented in the following table:

Denotation	Definition	Comment (Term)
O_v	$= \{u \mid (v, u) \in E\}$	
I_v	$= \{u \mid (u, v) \in E\}$	for undirected graphs $O_v = I_v$
N_v	$= O_v \cup I_v$	a set of all neighbors of v
$\deg(v)$	$= N_v $	<i>degree</i> of vertex v
$\text{indeg}(v)$	$= I_v $	<i>indegree</i> of vertex v
$\text{outdeg}(v)$	$= O_v $	<i>outdegree</i> of vertex v

By $V(H)$, resp. $E(H)$ we denote the set of vertices, resp. edges of a graph H . We use V and E instead of $V(H)$ and $E(H)$, when the graph H is understood from the context. In this case, n stands for the number of vertices and m for the number of edges in H ($n = |V|$, $m = |E|$).

A subgraph H of a given graph $G = (V, E)$ is *induced* by a set of vertices $S \subseteq V$, if $H = (S, E \cap S \times S)$. In other words, H is a graph consisting of all vertices in S and all edges of G joining two vertices from S .

A *path* P is a sequence of vertices $P = u_1, \dots, u_k$, $u_i \in V$ for $i = 1, \dots, k$, such that $(u_i, u_{i+1}) \in E$ for $i = 1, \dots, k - 1$ and $u_i \neq u_j$ for $i \neq j$. Sometimes we say that P is a path from u_1 to u_k , or, if G is undirected, we say that P is a path between u_1 and u_k . A *length* of the path P is the number $k - 1$. A *cycle* is a path $P = u_1, \dots, u_k$, $k > 2$, such that $(u_k, u_1) \in E$.

We say that a graph G is *connected*, if there is a path between any two vertices in G . If G is directed, we ignore the directions of the edges. A *connected component* of a graph G is the connected subgraph H of G such that H is induced by the set $V(H)$ and there is no edge in G connecting a vertex from $V(H)$ to a vertex outside $V(H)$. Again, if G is directed, the directions of the edges are ignored.

In some chapters we study special kinds of well-known graphs. Here is a summary of definitions: A graph G without a cycle is called a *forest*. If a forest G is connected, it is a *tree*. A special type of a tree having one vertex joined by an edge to all other vertices, is called a *stargraph*. A graph $G = (V, E)$ such that $V = \{v_1, \dots, v_n\}$ and $E = V \times V - \{(v_i, v_i) \mid 1 \leq i \leq n\}$ is called *complete* and denoted K_n . A *hypercube* $H_p = (V, E)$ of dimension p is a graph of 2^p vertices such that $V = \{0, 1\}^p$ (all binary numbers of at most p digits) and $E = \{(x, y) \mid x \text{ and } y \text{ differ in exactly 1 digit}\}$. A graph G is *planar* if it can be embedded into a plane so that no two edges cross. Let G' be such embedding of G . A *face* F is a topological term for a connected open set of points in plane such that its boundary is formed by some edges and vertices in G' and there is no vertex from G' in F . The *outer* face is the face containing the point in the infinity. All these notions shall be used for digraphs as well (directions of the edges are ignored).

The last definitions from the graph terminology introduced are the closely related terms of arboricity and density of a graph. Several authors (see [12]) have studied the family of graphs of bounded arboricity. In this work we deal with this family of graphs in Chapters 5 and 6.

Definition 2.1 An arboricity $\gamma(G)$ of a graph G is the minimal number of edge-disjoint forests into which G can be decomposed.

Definition 2.2 Let $G = (V, E)$ be a graph. The density $\delta(G)$ of the graph G is $\delta(G) = \max_{H \subseteq G} \frac{E(H)}{V(H)}$.

The correlation between arboricity and density is explained in Chapter 6, where we propose an algorithm for finding a density (more precisely $\lceil \delta(G) \rceil$) of a given graph G . Deeper insight into this relation gives also the well-known theorem of Nash-Williams (see [16]):

Theorem 2.3 (Nash-Williams) Let $G = (V, E)$ be a graph. Then:

$$\gamma(G) = \lceil \max_{H \subseteq G} \frac{E(H)}{V(H) - 1} \rceil$$

From the well-known fact that a planar graph of n vertices has at most $3n - 6$ edges one can now easily conclude that the family of planar graphs is a family of graphs of arboricity upper-bounded by 3.

2.2 Models of Representation

We study two basic types of representations of graphs. The term *classic representation* means that the structure is stored in a global memory and all algorithms can access the information stored there. The representation must provide an algorithm for adjacency test of two vertices.

The following type of representation of graphs is inspired by distributed networks. It assigns to each vertex w a *label* $\text{lab}(w)$. If there exists an algorithm for testing adjacency of two vertices u and v such that the only information about the graph needed to run the algorithm is stored in $\text{lab}(u)$ and $\text{lab}(v)$, the representation is called a *labeling schema* (or just *labeling*).

Both of these models of representation of graphs can be used for both directed and undirected graphs.

We compare the representations of a *static* graph (i.e. not changing after being represented) in two attributes: the amount of memory used and the time needed for adjacency test. When representing *dynamic* graphs, the representation must provide algorithms for *update* operations (addition or deletion of an edge or a vertex) too. In this case another attribute for comparison of representations is added — the time complexity of the update operations.

2.3 Measures of Complexity

In this work we shall use several measures of complexity. The most important here is the one called *bit-model*: the space used for storing a structure is counted in bits, and the time complexity is determined by the number of bits visited during the run of an algorithm.

Another widely used model in computer science, especially in the theory of algorithms, is the *cell-model*: Information is stored in a cell (or in several cells), each of which can accommodate element from a set $U = \{0, 1, \dots, m_U\}$, where m_U is fixed. The set U is called a *universe*. The space complexity in this model is counted in the number of cells used, and the time complexity corresponds to the number of basic operations performed by the algorithm. Thus, basic arithmetic operations such as adding, subtracting, multiplying and dividing are considered to be performed in constant time.

In both models we assume that jumping to a given address takes constant time. Therefore in the cell-model the jump-operation is performed in time $O(1)$, while in the bit-model it is proportional to the length of the address. In fact, we shall bound the address also in the cell-model (usually by m_U).

The main difference between these two models lies in the time complexity of arithmetic operations. While in the cell-model adding (or subtracting) two numbers of size $O(m_U)$ takes constant time, in the bit-model it takes time $O(\log m_U)$. However, multiplying (or dividing) these two numbers takes in the bit-model time $O(\log m_U \log \log m_U)$, while in the-cell model its time complexity remains constant.

2.4 Straightforward Representations

An *adjacency list* is a representation of a graph $G = (V, E)$ which gives for every vertex a list of its neighbors. The list of neighbors can be ordered, or stored

in a weighted binary search tree. Thus, the space complexity is $O(m)$ in the cell-model, resp. $O(m \log n)$ in the bit-model, and the time complexity of the adjacency test and the update operation is $O(\log n)$, resp. $O(\log^2 n)$.

Let $V = \{1, \dots, n\}$. The *adjacency matrix* is a matrix of type $n \times n$ having 1 in the position $[i, j]$ iff $(i, j) \in E$. The space complexity is $O(n^2)$ in both models, the time complexity is constant, resp. $O(\log n)$. The main advantage of this representation is that the worst-case time complexity of the update operations is constant, resp. $O(\log n)$ too.

We consider a representation to be optimal, if its space complexity is $O(m)$, resp. $O(m \log n)$ and the time complexity of all operations is $O(1)$, resp. $O(\log n)$, depending on the model used.

3 Related Work

Several authors worked on the problem of succinct encoding of a graph, regardless of the time complexity of the basic “graph-operations”. Sampath Kannan, Moni Naor and Steven Rudich (see [12]) encode a graph of arboricity α by a labeling representation with labels of size $O(\alpha \log n)$. György Turán showed that unlabeled planar graphs can be encoded using $12n$ bits, and he gave an asymptotically optimal representation for labeled planar graphs. Alon Itai and Michael Rodeh (see [10]) showed that the adjacency list method is optimal in many instances — for planar graphs it requires $3n \log n + O(n)$ bits. They present also a linear algorithm, which obtains a $3/2n \log n + O(n)$ representation (in the case of planar graphs). Moni Naor (see [15]) showed that general unlabeled graphs can be represented by $\binom{n}{2} - n \log n + O(n)$ bits which is optimal up to the $O(n)$ term. Both encoding and decoding algorithms that he suggested are linear.

Guy Jacobson (see [11]) was interested also in the second aspect – the time complexity of the operations. He developed a data structure that represent static unlabeled trees, planar graphs and graphs of bounded genus in asymptotically optimal space, and is time-efficient for traversal operations. Tomás Feder and Rajeev Motwani (see [5]) proposed a technique of partitioning the edges of a graph into bipartite cliques. The technique works for sufficiently dense graphs and can be used to speed-up some algorithms, for example matching or shortest paths. Maurizio Talamo and Paola Vocca (see [19]) represent a general graph by labeling with labels of size $O(\deg(v) \log^3 n)$ bits. They are able to perform the adjacency test in $O(\log n)$ bit-operations. Their work is the closest to the problem studied here.

4 Hashing Algorithms

Hashing is a very useful technique for storing any data structure. A *static dictionary* is a data structure consisting of a set of elements picked from a given universe and supporting a test of membership of an element from the universe in this set (a FIND operation). If the dictionary supports also the update operations (an insertion and a deletion of an element) it is called a *dynamic dictionary*. Several optimal or near optimal algorithms have been developed for maintaining a dictionary.

In this chapter we recall a result of M.L. Fredman, J. Komlós, and E. Szemerédi (see [6]) who developed an optimal algorithm for storing a static dictionary. Based on their work M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R.E. Tarjan (see [3]) proposed an algorithm for maintaining a dynamic dictionary running in expected optimal time. We modified this algorithm for storing a dynamic graph. Our algorithm not only tests the adjacency of two vertices and performs the update operations on the graph, but it also lists all current neighbors of a given vertex. The time and space complexity of all supported operations are optimal, using the cell-model as a measure of complexity. Throughout this chapter we shall use only the cell-model.

4.1 Data Structures

First, we sketch the important ideas of Fredman, Komlós and Szemerédi (FKS) and Dietzfelbinger et. al., the stepping stones for our algorithm presented at the end of the chapter. We start by describing the double hashing scheme of FKS which stores the static dictionary in optimal space and supports FIND operation in optimal time (using cell-model as a measure of complexity). Then, we outline the technique of Dietzfelbinger et. al. which uses FKS scheme for a dynamic dictionary.

4.1.1 Static Dictionary

Let M be a universe of elements and let $S \subseteq M$ be the set to be stored. The following theorem is the basis of all algorithms described in this chapter.

Theorem 4.1 (Fredman, Komlós, Szemerédi)

For any $S \subseteq M = \{0, 1, \dots, m - 1\}$ with $|S| = s$ there exists a hash table representation of S that uses space $O(s)$ and permits the processing of a FIND operation in constant time. Such representation can be found in expected $O(s)$ time.

The proof of this theorem is very important for understanding our algorithm. The proof presented here is only a slight modification of the original proof of FKS.

We need to define the basic hashing notation:

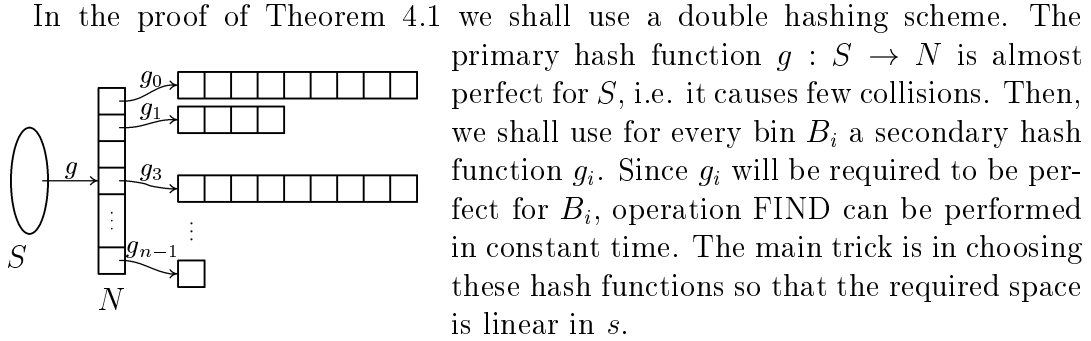
Definition 4.2 Let $S \subseteq M$ and let $h : M \rightarrow N$ be a hash function, $N = \{0, 1, \dots, n-1\}$. For each table location $0 \leq i \leq n-1$, we define the bin

$$B_i(h, S) = \{x \in S \mid h(x) = i\}.$$

The size of a bin is denoted by $b_i(h, S) = |B_i(h, S)|$.

Collision caused by h on S is any set $\{x, y\} \subseteq S$, $x \neq y$, such that $h(x) = h(y)$.

Hash function $h : M \rightarrow N$ is called perfect for S , if it causes no collisions on S , i.e. $b_i(h, S) \leq 1$ for every i .



In the proof of Theorem 4.1 we shall use a double hashing scheme. The primary hash function $g : S \rightarrow N$ is almost perfect for S , i.e. it causes few collisions. Then, we shall use for every bin B_i a secondary hash function g_i . Since g_i will be required to be perfect for B_i , operation FIND can be performed in constant time. The main trick is in choosing these hash functions so that the required space is linear in s .

Let $p_{[m]} \geq m$ be a prime such that $p_{[m]} = O(m)$. By Bertrand's Postulate such $p_{[m]} \leq 2m$ always exists. For each prime $p_{[m]}$ and a hash table $R = \{0, 1, \dots, r-1\}$ we define a family \mathcal{H}_r of hash functions $h_k : M \rightarrow R$:

$$h_k(x) = (kx \bmod p_{[m]}) \bmod r, \quad \text{where } 1 \leq k < p_{[m]}.$$

The most important property of the family \mathcal{H}_r is:

Lemma 4.3 For all $V \subseteq M$ of size v , and all $r \geq v$,

$$\sum_{i=0}^{r-1} \binom{b_i(h_k, V)}{2} < 2 \frac{v^2}{r}$$

for at least one-half of the choices of $k \in \{1, \dots, p_{[m]}-1\}$, i.e. for at least one-half of the functions in \mathcal{H}_r .

Proof : First, for fixed V and r , we show:

$$\sum_{k=1}^{p_{[m]}-1} \sum_{i=0}^{r-1} \binom{b_i(h_k, V)}{2} < \frac{(p_{[m]}-1)v^2}{r}. \quad (1)$$

The left-hand side counts all the pairs $(k, \{x, y\})$ such that $x \neq y$ and $h_k(x) = h_k(y)$. For fixed $\{x, y\}$, the number of pairs $(k, \{x, y\})$ is: $|\{k \mid k(x-y) \bmod p_{[m]} = zr, z \in \mathbf{Z} \setminus \{0\}\}|$. Since $\mathbf{Z}_{p_{[m]}}$ is a field, there is a unique solution for k satisfying the equation $kc \bmod p_{[m]} = zr$ for $1 \leq |z| \leq \lfloor (p_{[m]}-1)/r \rfloor$ and a constant c . (For other z there is no solution.) Therefore for a fixed $\{x, y\}$ there are at most

$2(p_{[m]} - 1)/r$ tuples $(k, \{x, y\})$. The number of sets $\{x, y\}$ is $\binom{v}{2}$. Multiplying it by $2(p_{[m]} - 1)/r$ implies the right-hand side of the inequality above.

We shall finish the proof by contradiction. Let us suppose that

$$\sum_{i=0}^{r-1} \binom{b_i(h_k, V)}{2} \geq 2 \frac{v^2}{r},$$

holds for more than one-half of the choices of k . Then

$$\sum_{k=1}^{p_{[m]}-1} \sum_{i=0}^{r-1} \binom{b_i(h_k, V)}{2} \geq \frac{(p_{[m]} - 1)}{2} \cdot \frac{2v^2}{r},$$

a contradiction with (1). \square

Corollary 4.4 *Let h_k be a hash function satisfying inequality from the previous lemma. Then:*

$$\sum_{i=0}^{r-1} b_i(h_k, V)^2 < 4 \frac{v^2}{r} + v$$

The proof follows directly from the lemma and from the fact, that $\sum_{i=0}^{r-1} b_i(h_k, V) = v$.

Now we can prove the theorem:

Proof of Theorem 4.1: We show that there exists a primary hash function $g = h_k \in \mathcal{H}_s$ such that the sum of squares of the sizes of the bins is linear in s . For every bin $B_i(h_k, S)$ (denoted S_i) we find a perfect hash function $g_i = h_{k_i} \in \mathcal{H}_{b_i(h_k, S)^2}$ such that $h_{k_i} : B_i(h_k, S) \rightarrow \{0, 1, \dots, b_i(h_k, S)^2\}$. For any x the cell $h_k(x)$ will store k_i of corresponding secondary hash function, i.e. $k_{h_k(x)}$. Since k and each k_i is $O(m)$, the required space is linear in s .

The existence of such primary hash function follows immediately from the Corollary 4.4, if $V = S$ and $v = r = s$.

Let S_i be a bin of size s_i . By Lemma 4.3 for $V = S_i$, $v = s_i$, and $r = 2s_i^2$, there exists $h_{k_i} \in \mathcal{H}_{s_i^2}$ such that $\binom{b_j(h_{k_i}, S_i)}{2} = 0$ for every $j \in \{0, \dots, s_i^2 - 1\}$. Therefore h_{k_i} is perfect for S_i .

The total space T used by this scheme is bounded by:

$$T \leq s + 1 + \sum_{i=0}^{s-1} 2b_i(h_k, S)^2 < s + 1 + 8 \frac{s^2}{s} + 2s = 11s + 1 = O(s).$$

By Lemma 4.3, hash functions h_k and h_{k_i} can be found by random sampling in expected $O(s)$ and $O(b_i(h_k, S))$ time, respectively, i.e. in time linear in s . \square

4.1.2 Dynamic Dictionary

The following randomized algorithm for dynamic dictionary is due to Dietzfelbinger et al. (see [3]). The algorithm uses FKS scheme for a static dictionary and, if necessary, rehashes part or all of the structure.

Let $M = \{0, \dots, m - 1\}$ be the universe and let $S \subseteq M$ be the set to be represented. Let $s = |S|$. Using the double-hashing scheme, the primary hash function hashes the set S into bins B_i of size b_i , where i stands for each table location, $i = 0, \dots, n - 1$. The content of the bin B_i is then hashed into a block of memory T_i . The currently used double-hashing scheme should accommodate up to t elements.

During the process (except while processing the rehashing phase), the following invariants hold:

$$(1 + c)s \leq t \leq (1 + 2c)s, \quad \text{for a suitable constant } c \quad (2)$$

$$|T_i| = 2m_i^2, \quad \text{where } m_i \text{ is such that } b_i \leq m_i \leq 2b_i \quad (3)$$

$$\sum_{i=0}^{n-1} b_i^2 < 4\frac{t^2}{n} + t \quad (4)$$

The values of t and each m_i shall be determined (and if necessary changed) so that the total space used is linear in s .

Clearly, the FIND operation can be performed in constant time in the same manner as in the FKS scheme. The update operations (INSERT, DELETE) are slightly more complicated.

The algorithm consists of four main procedures: Insert(x), Delete(x), Find(x), and RehashAll. The procedure Insert(x) first checks the violation of invariant (2). If the invariant does not hold, complete rehashing is performed. Otherwise, the procedure detects the position where x should be stored if the current hashing functions were used. If the position is empty, x is stored there unless the invariant (3) is violated. If the invariant does not hold anymore, or the position is occupied, rehashing is necessary. Either complete rehashing, or rehashing of the corresponding table T_j is performed (depends on the invariant (4)). The procedure Delete(x) deletes x from its position in the corresponding table. If the invariant (2) is violated, the whole structure is rehashed. The procedure Find(x) is identical to its counterpart for static dictionary and the procedure RehashAll is similar to the initial phase for static dictionary. While rehashing, the new hashing functions and the values of t and each m_i are determined. Here is the sketch of the algorithm.

Algorithm 4.5 (Dynamic Dictionary)

Description: *Represent a set S in $O(s)$ space. Provide FIND, INSERT and DELETE operations in constant time.*

1. **Procedure** RehashAll;
2. *put all elements in S into a list L ;*
3. **for** all $i = 0, \dots, n - 1$ **do** delete T_i ;
4. $t \leftarrow (1 + c)s$;
5. *by random sampling find $g = h_k \in \mathcal{H}_n$ such that condition (4) is satisfied;*
6. **for** $i = 0, \dots, n - 1$ **do**
7. $m_i \leftarrow 2b_i$;
8. $B_i \leftarrow \emptyset$; $b_i \leftarrow 0$;
9. allocate T_i such that $|T_i| = 2m_i^2$;
10. randomly choose $h_{k_i} : M \rightarrow T_i$ from $\mathcal{H}_{2m_i^2}$;
11. $S \leftarrow \emptyset$; $s \leftarrow 0$;
12. **for** all $y \in L$ **do** Insert(y);

13. **Procedure** Insert(x);
14. $s \leftarrow s + 1$; $S \leftarrow S \cup \{x\}$;
15. **if** $s \leq t$ **then**
16. $j \leftarrow h_k(x)$;
17. $B_j \leftarrow B_j \cup \{x\}$; $b_j \leftarrow b_j + 1$;
18. **if** $b_j \leq m_j$ **and** position $h_{k_j}(x)$ of subtable T_j is empty **then**
19. store x in this position
20. **else**
21. **if** $b_j \leq m_j$ **then**
22. put all elements in T_j into a list L_j ;
23. by random sampling from $\mathcal{H}_{2m_j^2}$ find $h_{k_j} : M \rightarrow T_j$ perfect for B_j ;
24. **for** all elements $y \in L_j$ **do**
25. store y in position $h_{k_j}(y)$ of subtable T_j ;
26. **else**
27. $m_j \leftarrow 2m_j$;
28. **if** there is not enough space for the new subtable T_j ($|T_j| = 2m_j^2$)
or (4) is no longer satisfied **then** RehashAll
29. **else**
30. put all elements in T_j into a list L_j ;
31. delete T_j ;
32. allocate new subtable T_j ;
33. by random sampling find h_{k_j} in $\mathcal{H}_{2m_j^2}$ perfect for B_j ;
34. use h_{k_j} for hashing elements in L_j into T_j ;

35. **else** RehashAll;

36. **Procedure** Delete(x);

37. $j \leftarrow h_k(x)$;

38. $s \leftarrow s - 1$;

39. $B_i \leftarrow B_i - \{x\}$; $b_i \leftarrow b_i - 1$;

40. delete x from position $h_{k_j}(x)$ of subtable T_j ;

41. **if** $(1 + 2c)s < t$ **then** RehashAll;

42. **Function** Find(x);

43. $j \leftarrow h_k(x)$;

44. **if** position $h_{k_j}(x)$ of subtable T_j is empty **then return** $x \notin S$

45. **else return** $x \in S$;

The space complexity of the algorithm follows from the conditions (2), (3) and (4). The space used by subtables T_i is:

$$\sum_{i=0}^{n-1} |T_i| \stackrel{(3)}{=} \sum_{i=0}^{n-1} 2m_i^2 \stackrel{(3)}{\leq} 8 \sum_{i=0}^{n-1} b_i^2 \stackrel{(4)}{<} 32 \frac{t^2}{n} + 8t$$

By (2), $t = \Theta(s)$. If we choose $n = \Omega(s)$, for example $n = t$, by the condition (4), the space complexity of the algorithm is $\Theta(s)$.

The time complexity of the FIND operation is clearly constant. We show that the amortized expected time complexity of the update operations is constant too. The main trick of this algorithm lies in the fact, that the scheme should accommodate additional cs elements after expected 2 calls of *RehashAll* (see Lemma 4.3). The time complexity of this procedure is $O(s)$. Therefore amortized expected time complexity of the update operations is constant.

4.2 Applications to Graphs

A graph of n vertices can be represented as a set of its edges. A suitable coding $f : E \rightarrow \mathbf{N}$ of an edge $(u, v) = e \in E$, $u, v \in V$, $V = \{0, \dots, n - 1\}$, is a number $f(e) = un + v$, since it fits into a cell of $O(\log n)$ size. Using this method a graph can be represented in $O(m)$ cells and the adjacency testing can be performed in constant time, regardless of whether the graph is directed or undirected. Update operations can be performed in constant amortized expected time. (Note: In the bit-model the total space used is $O(m \log n)$ bits, the adjacency testing takes $O(\log n \log \log n)$ bit-operations and the update operations take amortized expected $O(\log n \log \log n)$ bit-operations.)

A labeling schema can be developed similarly. A label of vertex v corresponds to the representation of set N_v . (Recall that N_v is the set of all neighbors of v .)

Both of these representations allow efficient adjacency testing.

Another useful operation on graphs is ALL-NEIGHBORS-LIST for a given vertex v . The algorithm for ALL-NEIGHBORS-LIST operation is efficient, if it can be performed in $O(d_v)$ steps ($d_v = \deg(v) = |N_v|$) and the whole representation uses storage of no more than $O(m + n)$ cells. In the static case it suffices to add an adjacency list to the structure to solve the problem. It is more interesting to consider the dynamic case. In the rest of the chapter we shall modify the Algorithm 4.5 so that it can be used efficiently for both adjacency testing and listing all neighbors of a given vertex.

Let L_G be a labeling schema (described above) for a dynamic graph G . The main idea of the modification of this schema lies in hashing a double linked list of all neighbors of every vertex v into L_G .

For each vertex v we modify its label in the following way: In every non-empty cell of every table $T_{v,i}$ we store a triple $(x, prev_x, next_x)$, where x is the original content of the cell (i.e. the vertex x is adjacent to v), and $prev_x$ (resp. $next_x$) is a pair of integers determining the hash-position of the previous (resp. next) member in a double linked list of all neighbors of v . Let y be the previous (resp. next) vertex in the linked list of vertices adjacent to v . The hash-position of y is a pair of numbers j and l such that y is stored in table the $T_{v,j}$ in the position l . Thanks to the double linking, the linked list can be updated in constant time.

More precisely, here is a sketch of the algorithm. (Deviations from Algorithm 4.5 are denoted by \Leftarrow .)

Algorithm 4.6 (Dynamic Digraph – Hashing)

Description: *Represent a dynamic graph $G = (E, V)$ of m edges and n vertices in $O(m + n)$ space. Provide $\text{AddEdge}(v, w)$, $\text{DeleteEdge}(v, w)$, $\text{AddVertex}(v)$ and $\text{AdjacencyTest}(v, w)$ operations in constant time and operations $\text{DeleteVertex}(v)$ and $\text{ListAllNeighbors}(v)$ in $O(d_v)$ time.*

1. **Procedure** $\text{RehashAll}(v)$;
2. put all elements in N_v into a list L_v ;
3. $z_v \leftarrow \text{undefined}$; \Leftarrow
4. **for** all $i = 0, \dots, t_v - 1$ **do** delete $T_{v,i}$;
5. $t_v \leftarrow (1 + c)d_v$;
6. by random sampling find $g_v = h_{v,k} \in \mathcal{H}_{t_v}$ such that $\sum_{i=0}^{t_v} b_{v,i}^2 < 5t_v$;
7. **for** $i = 0, \dots, t_v - 1$ **do**
8. $m_{v,i} \leftarrow 2b_{v,i}$;
9. $B_{v,i} \leftarrow \emptyset$; $b_{v,i} \leftarrow 0$;
10. allocate $T_{v,i}$ such that $T_{v,i}$ contains $2m_{v,i}^2$ cells, each cell is a 5-tuple of integers (over the universe);
11. randomly choose h_{v,k_i} from $\mathcal{H}_{2m_{v,i}^2}$;

12. $N_v \leftarrow \emptyset; d_v \leftarrow 0;$
13. **for** all $u \in L_v$ **do** AddEdge(v, u);

14. **Procedure** StoreNeighbor(v, w, j, p); ←
15. **if** z_v is undefined **then**
16. $z_v \leftarrow (j, p);$
17. store (w, z_v, z_v) in position p of subtable $T_{v,j};$
18. **else**
19. $(r, l) \leftarrow z_v;$
20. $(q, p_q, n_q) \leftarrow$ triple from position r of subtable $T_{v,l};$
21. store (w, z_v, n_q) in position p of subtable $T_{v,j};$
22. store ($q, p_q, (j, p)$) in position r of subtable $T_{v,l};$
23. $(r, l) \leftarrow n_q;$
24. $(q, p_q, n_q) \leftarrow$ triple from position r of subtable $T_{v,l};$
25. store ($q, (j, p), n_q$) in position r of subtable $T_{v,l};$

26. **Procedure** AddEdge(v, w);
27. $d_v \leftarrow d_v + 1; N_v \leftarrow N_v \cup \{w\};$
28. **if** $d_v \leq t_v$ **then**
29. $j \leftarrow h_{v,k}(w);$
30. $B_{v,j} \leftarrow B_{v,j} \cup \{w\}; b_{v,j} \leftarrow b_{v,j} + 1;$
31. **if** $b_{v,j} \leq m_{v,j}$ **and** position $h_{v,k_j}(w)$ of subtable $T_{v,j}$ is empty **then**
32. StoreNeighbor($v, w, j, h_{v,k_j}(w)$);
33. **else**
34. **if** $b_{v,j} \leq m_{v,j}$ **then**
35. put all elements in $T_{v,j}$ into a list $L_{v,j};$
36. by random sampling in $\mathcal{H}_{2m_{v,j}^2}$ find h_{v,k_j} perfect for $B_{v,j};$
37. **for** all elements $u \in L_{v,j}$ **do**
38. StoreNeighbor($v, u, j, h_{v,k_j}(u)$);
39. **else**
40. $m_{v,j} \leftarrow 2m_{v,j};$
41. **if** there is not enough space for the new subtable $T_{v,j}$
 ($|T_{v,j}| = 2m_{v,j}^2$) **or** $\sum_{i=0}^{t_v-1} b_{v,i}^2 \geq 5t_v$ **then** RehashAll(v)
42. **else**
43. put all elements in $T_{v,j}$ into a list $L_{v,j};$
44. delete $T_{v,j};$
45. allocate new subtable $T_{v,j};$

5 Representation by Distance

In this chapter we propose two new labeling schemes, both based on computing the distances between some chosen vertices and all other vertices. The main reason why we have chosen the representation based on distance is that the distance from all vertices to some fixed vertex has a nice property: in relatively small number of bits ($O(\log n)$ per vertex) one can store useful information for adjacency testing. Throughout this chapter we shall use the bit-model for the measure of complexity.

A *distance* between two vertices u and v in a graph $G = (V, E)$, denoted by $\text{dist}(u, v)$, is the length of the shortest path between u and v . The proposed representations are based on the symmetry of distance, i.e. $\text{dist}(u, v) = \text{dist}(v, u)$, therefore they can be used directly for undirected graphs only. A slight modification for directed graphs is mentioned at the end of the chapter.

5.1 Labeling by Distinguishing Set

Definition 5.1 Let $G = (V, E)$ be a graph. A set $S = \{w_1, w_2, \dots, w_s\} \subseteq V$ is called a *distinguishing set*, if: for any pair $(u, v) \notin E$ there exists $w \in S$ such that $|\text{dist}(w, u) - \text{dist}(w, v)| \geq 2$. We say that the vertex w distinguishes the anti-edge (u, v) .

Labeling by distinguishing set (*LDS*) assigns to each vertex $v \in V$ a label corresponding to a sequence of integers $\{\text{dist}(v, w_i)\}_{i=1}^s$.

Lemma 5.2 Definition 5.1 is correct, i.e.:

1. There always exists a distinguishing set S of a graph $G = (V, E)$.
2. For every $(u, v) \in E$ and each $w \in S$ is $|\text{dist}(w, u) - \text{dist}(w, v)| < 2$.

Proof :

1. A set $S = V$ fulfills the requirements of distinguishing set. Let $(u, v) \notin E$. Then there exists $w = u \in S$ such that $\text{dist}(w, v) = \text{dist}(u, v) \geq 2$ (otherwise $(u, v) \in E$).
2. Let S be any distinguishing set of G . Let us suppose that there is $(u, v) \in E$ and $w \in S$ such that $|\text{dist}(w, u) - \text{dist}(w, v)| \geq 2$. W.l.o.g. let $\text{dist}(w, u) < \text{dist}(w, v)$. There exists a path from w to u of length $\text{dist}(w, u)$. We can prolong that path to v by the edge (u, v) , thus constructing a path from w to v of length $\text{dist}(w, u) + 1 < \text{dist}(w, v)$, a contradiction.

□

The length of the label $\text{lab}(v)$ assigned by LDS to a vertex v is $O(\sum_{i=1}^s \log(\text{dist}(v, w_i)))$. If $\text{dist}(v, w_i) = \infty$ (i.e. v and w_i belong to different

connected components of G), we can define $\text{dist}(v, w_i) = -2$. Therefore we can assume that for v and w_i from different connected components $\text{dist}(v, w_i) = O(1)$. The time complexity of the adjacency test of two vertices u and v is proportional to the sum of lengths of labels $\text{lab}(u)$ and $\text{lab}(v)$. This time is upper-bounded by $s(\log n + c)$ for a suitable constant c .

5.2 Recursive Distance-Labeling

The second representation is based on a similar principle. The intuition behind is as follows: Let us number the connected components of a graph G . We choose a representative vertex in every component and we compute distance from this representative to every other vertex in the component. The label of every vertex starts with the number of the component to which the vertex belongs and continues with the distance to the representative. We know that two vertices from various components or two vertices with the difference of distances greater than 1 cannot be adjacent. We decompose the vertices in each component C into layers: Each layer $L_{C,d}$ consists of vertices with the same distance d to the representative. Each layer $L_{C,d}$ induces new graph $H_{C,d}$ and each pair of adjacent layers $L_{C,d}$ and $L_{C,d+1}$ “induce” the new bipartite graph $K_{C,d}$. We use the same method for representing $H_{C,d}$ and $K_{C,d}$ (recursion). More precisely, here is the definition:

Definition 5.3 *Let $G = (V, E)$ be a graph. A recursive distance-labeling (RDL) assigns to each vertex $v \in V$ a label corresponding to a sequence of 5-tuples $\{(c_{v,i}, d_{v,i}, a_{v,i,1}, a_{v,i,0}, a_{v,i,-1})\}_{i=1}^{n_v}$, ($n_v > 0$), each of which corresponds to some subgraph $G_{v,i}$ of G , such that:*

- $G_{v,1} = G$,
- *the connected components in every $G_{v,i}$ are numbered by some fixed unique numbers from $\{1, 2, \dots, |V|\}$, and every connected component C of $G_{v,i}$ is represented by a fixed vertex $w_C \in V(C)$ (called a representative), i.e. if two tuples $(c_{v,i}, d_{v,i}, a_{v,i,1}, a_{v,i,0}, a_{v,i,-1})$ and $(c_{u,j}, d_{u,j}, a_{u,j,1}, a_{u,j,0}, a_{u,j,-1})$ correspond to the same subgraph $G_{v,i} = G_{u,j}$, the numbers of the connected components of $G_{v,i}$ and $G_{u,j}$ and their representatives are identical,*
- *the distance of a vertex to the representative w_C of a connected component C decomposes the vertices of C into layers:*

$$L_{C,d} = \{u \in V(C) \mid \text{dist}(u, w_C) = d\},$$

we refer to $L_{C,d}$ as the d -th layer of C ,

- $c_{v,i}$ *is the number of a connected component $C_{v,i}$ of $G_{v,i}$ containing v ,*
- $d_{v,i} = \text{dist}(w_{C_{v,i}}, v)$ *for each v in $V(C_{v,i})$,*

- if $d_{v,i} = 0$, then $a_{v,i,1} = a_{v,i,0} = a_{v,i,-1} = 0$,
otherwise $a_{v,i,1} = k_1$ (resp. $a_{v,i,0} = k_0$, resp. $a_{v,i,-1} = k_{-1}$) stands for the k_1 -th (resp. k_0 -th), resp. k_{-1} -th) 5-tuple such that
 - $V_1 = \{u \mid u \in V(C_{v,i}), \text{dist}(w_{C_{v,i}}, u) = d_{v,i} + 1\}$,
i.e. V_1 is the $(d_{v,i} + 1)$ -th layer of the graph $C_{v,i}$,
 - $V_0 = \{u \mid u \in V(C_{v,i}), \text{dist}(w_{C_{v,i}}, u) = d_{v,i}\}$,
i.e. V_0 is the $d_{v,i}$ -th layer of the graph $C_{v,i}$,
 - $V_{-1} = \{u \mid u \in V(C_{v,i}), \text{dist}(w_{C_{v,i}}, u) = d_{v,i} - 1, d_{v,i} \geq 2\}$,
i.e. if $d_{v,i} \geq 2$, V_{-1} is the $(d_{v,i} - 1)$ -th layer of the graph $C_{v,i}$, otherwise V_{-1} is the empty set,
 - if $V_1 = \emptyset$, let $k_1 = 0$, otherwise:
 $G_{v,k_1} = (V_0 \cup V_1, E(C_{v,i}) \cap (V_0 \times V_1 \cup V_1 \times V_0))$,
 - $G_{v,k_0} = (V_0, E(C_{v,i}) \cap (V_0 \times V_0))$,
 - if $d_{v,i} = 1$, let $k_{-1} = 0$, otherwise:
 $G_{v,k_{-1}} = (V_0 \cup V_{-1}, E(C_{v,i}) \cap (V_0 \times V_{-1} \cup V_{-1} \times V_0))$.

i.e. G_{v,k_0} is the graph induced by the set of vertices V_0 and G_{v,k_1} (resp. $G_{v,k_{-1}}$) is the bipartite graph “induced” by partitions V_0 and V_1 (resp. V_0 and V_{-1}),
- every $(u, v) \notin E$ can be detected by the following algorithm:

Algorithm 5.4 (Adjacency Test)

Description: Test adjacency of two distinct vertices u and v .

1. $i \leftarrow 1; j \leftarrow 1;$
2. **if** $c_{v,i} \neq c_{u,j}$ **then report** $(u, v) \notin E$;
3. **if** $|d_{v,i} - d_{u,j}| \geq 2$ **then report** $(u, v) \notin E$
4. **else**
5. *w.l.o.g.* let $d_{v,i} \geq d_{u,j}$;
6. **if** $d_{u,j} = 0$ **then report** $(u, v) \in E$
7. **else**
8. **if** $d_{v,i} > d_{u,j}$ **then** $i \leftarrow a_{v,i,-1}; j \leftarrow a_{u,j,1}$
9. **else** $i \leftarrow a_{v,i,0}; j \leftarrow a_{u,j,0}$;
10. **goto** 3;

The maximum number of calls, over all pairs (u, v) , of **goto** in the previous algorithm is called a depth of the RDL.

First, we notice that RDL-scheme can be applied to an arbitrary undirected graph G , i.e. that the recursive process does not get into a loop. In the previous definition, each G_{v,k_j} has less vertices than $G_{v,i}$ (the representative $w_{C_{v,i}}$ is

contained in none of the G_{v,k_j}), thus the number of vertices in $G_{v,i}$ is the measure of recursion. After at most $n - 1$ “recursive calls” v must be chosen as the representative and no additional “recursive call” is made. We have proven this lemma:

Lemma 5.5 *For any graph $G = (V, E)$ there exists a representation by RDL-scheme.*

Now we shall discuss the correctness of Algorithm 5.4: Let $e = (u, v)$. Clearly, if u and v belong to different connected components, then $e \notin E(G)$. Further, if there exists a vertex w such that $|\text{dist}(w, u) - \text{dist}(w, v)| \geq 2$, then $e \notin E(G)$. (The argument is very similar to the proof of Lemma 5.2.) Otherwise, e may or may not belong to $E(G)$. We recursively test if $e \in E(H)$, where H is the graph induced by the corresponding layers of G . This recurrent process ends if either $e \notin E(G)$ was detected or if u or v is the representative of their common connected component of the corresponding subgraph of G . Therefore:

Lemma 5.6 *The Algorithm 5.4 is correct.*

In the bit-model, the length of a label $\text{lab}(v)$ is counted in the number of bits. We need to represent a sequence of tuples by a string of bits so that the address of the i -th tuple can be easily detected. Hence, we add a table of these addresses to the sequence of tuples. By the length of a label we understand the number of bits needed to represent the sequence of tuples plus the size of the table, i.e. the sequence of tuples is $q_v = O(\sum_{i=1}^{n_v} (\log c_{v,i} + \log d_{v,i} + \log n_v))$ bits long and the table takes additional $n_v \log q_v$ bits.

We shall try to minimize the length of the labels assigned by the RDL-scheme, with respect to its depth. The sum of the lengths of the labels corresponds to the space complexity, while the depth multiplied by $\log n + \log l$, where l stands for the maximum of the lengths of labels (in bits), upper bounds the time complexity of the adjacency test.

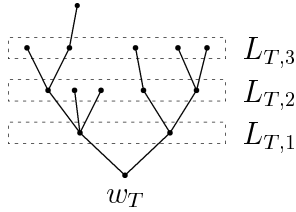
If a graph can be represented by RDL of depth p , the number of tuples n_v in a label $\text{lab}(v)$ can be upper bounded by $O(3^p)$, because every tuple induces the existence of up to 3 more tuples and $\sum_{i=0}^p 3^i = O(3^p)$. Hence, if for some family of graphs the depth is constant, the length of labels is $O(\log n)$ and also the adjacency test is $O(\log n)$. We shall study several graph families with constant depth.

5.3 Properties of RDL and LDS

Lemma 5.7 *Any tree T of n vertices can be represented by an RDL-scheme of constant depth.*

Proof : First we prove that any stargraph S can be represented by RDL of constant depth. Recall that a stargraph S is a tree such that each of its vertices is joined by an edge to one fixed vertex v_S . There are no other edges in S . If we choose v_S as the representative w_S of S , we obtain an RDL of constant depth, namely 2.

Now we can prove the lemma. Let us choose an arbitrary vertex w_T as the representative. The proof is based on simple observations: a) There are no cross-



edges within a layer. (Otherwise T would contain a cycle.) b) Graph “induced” by layers $L_{T,d}$ and $L_{T,d+1}$ consists of $|L_{T,d}|$ connected components, each of which is a stargraph. (If there were two distinct vertices $u_1, u_2 \in L_{T,d}$ both adjacent to the same $v \in L_{T,d+1}$, T would contain a cycle $u_1, \dots, w_T, \dots, u_2, v$.) From a) and b) one can conclude that there exists RDL for T of depth 3.

□

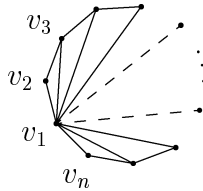
At first sight one would say that the RDL is at least as efficient as the LDS. This conjecture is disproved in the following lemma:

Lemma 5.8 *Labeling by distinguishing set and recursive distance-labeling are incomparable, i.e.:*

1. *There exists a family of graphs \mathcal{G} which needs asymptotically more space for labels in LDS than in RDL. Time spent for adjacency testing is asymptotically smaller for the latter labeling.*
2. *There exists a family of graphs \mathcal{G} which needs asymptotically more space for labels in RDL than in LDS. Time spent for adjacency testing is asymptotically smaller for the latter labeling.*

Proof :

1. Let \mathcal{G} be the family of graphs $G_n = (V_n, E_n)$ such that $V_n = \{v_1, \dots, v_n\}$, $E_n = \{(v_1, v_i) \mid i = 3, \dots, n\} \cup \{(v_i, v_{i+1}) \mid i = 1, \dots, n-1\}$.



First we assign an RDL for $G_n \in \mathcal{G}$: Let $w_g = v_1$ be the representative of G_n (G_n is connected). Therefore all vertices (except v_1) belong to the first layer, i.e. $L_{G_n,1} = V_n - \{v_1\}$. Since the graph H_d induced by $L_{G_n,1}$ is a path and thus a tree, by Lemma 5.7 there exists an RDL of constant depth for H_d . Therefore, the assigned G_n 's RDL has constant depth too.

On the other hand, let S be one of the distinguishing sets for G_n of minimal size. All anti-edges join vertices from the set $\{v_2, \dots, v_n\}$. If $v_1 \in S$, v_1 would distinguish no anti-edge, since $\text{dist}(v_1, v_i) = 1$ for every $i =$

$2, \dots, n$. Thus, the set $S' = S - \{v_1\}$ would be distinguishing too, and $|S'| < |S|$, a contradiction. Let us suppose that there are three distinct vertices $v_{i_1}, v_{i_2}, v_{i_3}$, $2 \leq i_1 < i_2 < i_3 \leq n$, such that $v_{i_1}, v_{i_2}, v_{i_3} \notin S$. The distance of any two vertices in G_n is at most two and it is 1 iff one of the vertices is v_1 or if the indices of the two vertices differ by one. Therefore the anti-edge (v_{i_1}, v_{i_3}) is distinguished by none of the vertices from S , a contradiction. We have proven that $|S| \geq n - 3$.

The proposed RDL assigns to each vertex a label of $O(\log n)$ bits and adjacency test can be performed in $O(\log n)$ bit operations. Every distinguishing set has size $\Theta(n)$, therefore the label of each vertex has length $\Omega(n)$ bits and the adjacency test takes time $\Omega(n)$ too.

2. Let $\mathcal{G} = \{K_n \mid n > 0\}$. Since K_n has no anti-edges, the size of the labels assigned by LDS and the time complexity of the adjacency test are constant. On the other hand, one can easily prove by induction on n that every RDL for K_n has depth $\Theta(n)$.

□

Although it seems that the size of a distinguishing set depends on the number of anti-edges in the graph (“the fewer anti-edges, the smaller the set”), the following lemma contradicts this hypothesis.

Lemma 5.9 *There exists a family of graphs \mathcal{G} such that for any $n > 0$ there exists a graph $G \in \mathcal{G}$ of n vertices and $\Theta(n^2)$ edges such that every distinguishing set S of G has size $\Theta(n)$.*

Proof : Let \mathcal{G} consist of graphs $G_n = (V_n, E_n)$, where $V_n = \{v_1, \dots, v_n\}$ and $E = V_n \times V_n - \{(v_{2i-1}, v_{2i}) \mid 1 \leq i \leq \lfloor \frac{n}{2} \rfloor\}$. Let G_n be a graph from \mathcal{G} . Let S be any distinguishing set of G_n . The distance from v_{2i-1} (or v_{2i}) to any other vertex (different from v_{2i-1} and v_{2i}) is 1. Thus, no vertex different from v_{2i-1} and v_{2i} can distinguish the anti-edge (v_{2i-1}, v_{2i}) , and therefore $|S| \geq \lfloor \frac{n}{2} \rfloor$. □

We shall try to specify those families of graphs which are effectively representable by either LDS or RDL.

We studied the representation by RDL or LDS of several well-known and in computer science widely studied families of graphs, such as trees, planar graphs, p -dimensional meshes and hypercubes. Although all of these families are representable efficiently by another representation, the proposed distance representations could serve as a generalization of various representations of the graph families. The most challenging problem was to prove that the family of graphs of arboricity upper bounded by some α is of depth proportional to α . Unfortunately, this problem remains open. We solved it only for a restricted class of planar graphs.

Lemma 5.10 *There exists a distinguishing set of size $p + 1$ for p -dimensional hypercube.*

Proof : Let H_p be the p -dimensional hypercube. We choose the distinguishing set S as follows: S contains all vertices with at most one digit 1. We show that every anti-edge (u, v) is distinguished by some vertex from S . We denote the number of 1s in u by $\|u\|$ — the *weight* of u . If $|\|u\| - \|v\|| \geq 2$, the anti-edge (u, v) is distinguished by the vertex $\underbrace{0 \dots 0}_p \in S$. Otherwise, w.l.o.g. let $\|u\| \leq \|v\|$. Clearly, $\|u\| > 0$ (if not, $\underbrace{0 \dots 0}_p$ is joined to every vertex of weight 1 and therefore (u, v) would not be an anti-edge). Since (u, v) is an anti-edge, at least one 1 in u is in a position where v has 0. As a distinguishing vertex for (u, v) we can choose any vertex $w \in S$ such that the digit 1 in w is in a position in which u has 1 and v has 0. Then $\text{dist}(u, w) = \|u\| - 1$, and $\text{dist}(v, w) = \|v\| + 1$. Thus, the anti-edge (u, v) is distinguished by w . \square

Corollary 5.11 *A hypercube can be represented by LDS as efficiently as by hashing.*

The best lower bound for the size of a distinguishing set for a hypercube is an open problem. An answer to the question what is the complexity of the best RDL representation of a hypercube remains open too.

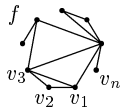
We conjecture that the RDL representation of graphs of bounded arboricity gives the same time and space complexity as the representation proposed by Kannan et. al. (see [12]). We prove this claim for the family of planar graphs (which have arboricity upper bounded by 3). First, we prove it for the family of boundary planar graphs, and then we generalize it for any planar graph.

Definition 5.12 *Let G be a planar graph. We call G boundary planar if it can be embedded into a plane so that all its vertices are on a boundary of the same face.*

Let us note that a boundary planar graph is not necessarily connected. It is a union of its connected boundary planar components.

Theorem 5.13 *For any boundary planar graph G there exists a recursive distance-labeling of constant depth.*

Proof : Without loss of generality we can assume that G is connected. Let us embed G into a plane as described in Definition 5.12. We denote the face common to all vertices by f . Let us denote the vertices on f clockwise by v_1, v_2, \dots, v_n and let us choose $w_G = v_1$ to be a represen-



tative of G . The distance of a vertex to the representative of G decomposes the vertices of G into layers $L_{G,d}$. First, we show the following claim:

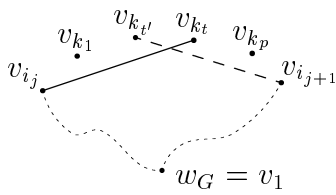
Let $v_{i_1}, v_{i_2}, \dots, v_{i_q}$, where $1 \leq i_1 < i_2 < \dots < i_q \leq n$, $q = |L_{G,d}|$, be the vertices in the $L_{G,d}$, d -th layer of G . If $(v_{i_j}, v_{i_k}) \in E(G)$, then $|j - k| = 1$.

We show, that $e = (v_{i_j}, v_{i_k}) \in E(G)$, for some j, k such that $|j - k| \geq 2$, leads to a contradiction. We know that the shortest path from v_1 to every v_{i_l} for $l = 1, \dots, q$ has length d . If $|j - k| \geq 2$, w.l.o.g. let $j < k$, any path from v_1 to $v_{i_{j+1}}$ must pass through either v_{i_j} or v_{i_k} (because of the topology). Therefore the shortest path from v_1 to $v_{i_{j+1}}$ has length at least $d + 1$, a contradiction.

Let $H_{G,d} = (L_{G,d}, E(G) \cap L_{G,d} \times L_{G,d})$. Such a graph corresponds to G_{v, k_0} , where $\text{dist}(w_G, v) = d$ and k_0 is from the first 5-tuple in $\text{lab}(v)$. As shown in the above claim, $H_{G,d}$ is a union of vertex-disjoint paths. By Lemma 5.7, each path can be represented by RDL-scheme of a constant depth.

We need to prove that also $K_{G,d} = (L_{G,d} \cup L_{G,d+1}, E(G) \cap (L_{G,d} \times L_{G,d+1} \cup L_{G,d+1} \times L_{G,d}))$ corresponding to G_{v, k_1} can be represented by RDL-scheme of a constant depth.

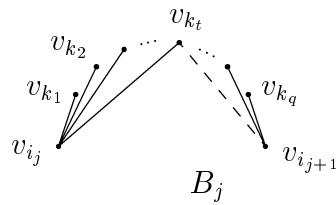
Let us recall, that v_{i_1}, \dots, v_{i_q} denote the vertices in $L_{G,d}$ and let $j < q$. We denote by v_{k_1}, \dots, v_{k_r} all the vertices from $L_{G,d+1}$ such that $i_j < k_l < i_{j+1}$ for $l = 1, \dots, r$ (i.e. all the vertices from $L_{G,d+1}$ lying between v_{i_j} and $v_{i_{j+1}}$). W.l.o.g.



we can assume that $k_1 < \dots < k_r$. Let t be the longest number such that $(v_{k_t}, v_{i_j}) \in E(G)$. Because of the embedding into the plane, no $v_{k_{t'}}$, $t' < t$ can be adjacent to $v_{i_{j+1}}$. However, every v_{k_l} must be adjacent to at least one of the vertices v_{i_j} and $v_{i_{j+1}}$. Therefore $(v_{k_l}, v_{i_j}) \in E(G)$ for $l = 1, \dots, t$ and $(v_{k_l}, v_{i_{j+1}}) \in E(G)$

for $l = t + 1, \dots, r$. The “edge” $(v_{k_t}, v_{i_{j+1}})$ is optional. We call a bipartite graph “induced” by vertices $v_{i_j}, v_{i_{j+1}}$ and all v_{k_l} a *block* B_j , i.e. $B_j = (V'_1 \cup v'_2, E(G) \cap (V'_1 \times V'_2 \cup V'_2 \times V'_1))$, where $V'_1 = \{v_{i_j}, v_{i_{j+1}}\}$ and $V'_2 = \{v_{k_l} \mid l=1, \dots, r\}$.

We have shown that every block B_j ($j = 1, \dots, q - 1$) is either a tree or a forest consisting of 2 stargraphs. The graph $K_{G,d}$ is a union of all blocks B_j . By the above claim, two distinct blocks B_j and B_k share a vertex iff $|j - k| = 1$. If $j + 1 = k$, the only shared vertex is v_{i_k} . Hence $K_{G,d}$ is a forest and by Lemma 5.7 it can be represented by an RDL of constant depth.



□

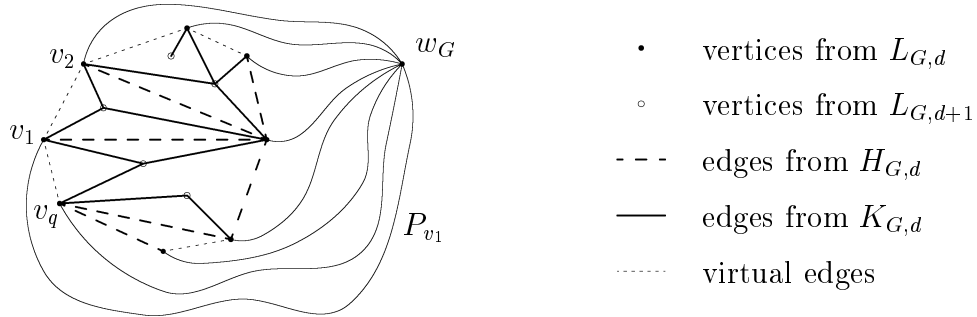
If G is “boundary”-embedded into a plane, the RDL of constant depth can be created in $O(n)$ time. Later we shall discuss the case when G is not embedded into a plane.

Let us mention that there is a subfamily of the family of boundary planar graphs such that for any n there exist a boundary planar graph of n vertices which requires the distinguishing set of size $|S| = \Theta(n)$. (An example of such a family of graphs is given in the proof of Lemma 5.8, item 1.)

Theorem 5.14 *For any planar graph G there exists a recursive distance-labeling of constant depth.*

Proof : Let $G = (V, E)$ be a planar graph embedded into the plane. W.l.o.g. let G be connected. Let us choose some vertex w_G lying on the boundary of the outer face of this embedding of G as the representative of G . First we prove that each graph $H_{G,d}$, $d > 0$, induced by the layer $L_{G,d}$ is a boundary planar graph (not necessary connected) and thus, by Theorem 5.13 it is representable by RDL of constant depth. Let us suppose that $H_{G,d}$ is not boundary planar, i.e. there is no embedding of $H_{G,d}$ into a plane such that all its vertices share a common face. Therefore the embedding of $H_{G,d}$ corresponding to the embedding of G has this property too. Let $V'_{G,d}$ be the set of vertices in $H_{G,d}$ lying on the boundary of the outer face. By the above assumption, there is a vertex $v \in V(H_{G,d}) - V'_{G,d}$ lying “inside” this embedding of $H_{G,d}$ (i.e. not on the boundary of the outer face). Hence, all paths in G from v to w_G have to pass through at least one vertex in $V'_{G,d}$ (because w_G lies on the boundary of the outer face of G), a contradiction with the assumption that v and all vertices in $V'_{G,d}$ are equally distant from w_G in G .

We need to prove that also the graph $K_{G,d}$ “induced” by the layers $L_{G,d}$ and $L_{G,d+1}$ has a suitable RDL-representation. There exists a set of paths $\{P_v \mid v \in L_{G,d}\}$ such that each path P_v is of length d , and joins v with w_G . We can rearrange the paths in this set so that no two path P_u, P_v cross, i.e. there exists l such that paths $P_u(w_G = u_1, u_2, \dots, u_{k_u} = u)$ and $P_v(w_G = v_1, \dots, v_{k_v} = v)$ share exactly the first l vertices: $u_i = v_i$ for $i \leq l$, and $u_i \neq v_j$ for $i, j > l$. Let P_u, P_v be two paths crossing each other, i.e. $u_i = v_j$ for some $i, j \geq 1$ and there exists $k < \min\{i, j\}$ such that $u_k \neq v_k$. If $i \neq j$, either P_u or P_v can be shortened, a contradiction. Therefore $i = j$ and P_u, P_v can be rearranged so that they share the first i vertices. Continuing with this process we obtain the desired set of paths.



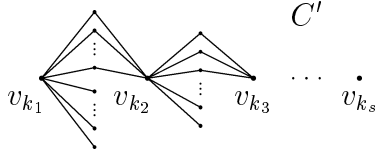
We can denote the vertices in $L_{G,d}$ by v_1, \dots, v_q ($q = |L_{G,d}|$) in clockwise order following the paths from w_G (i.e. there is no other path P_{v_j} “between” paths P_{v_i} and $P_{v_{i+1}}$). We can add a virtual edge between every pair of vertices v_i, v_{i+1} (and v_1, v_q) without disturbing the topology. Thus, we can suppose that $H_{G,d}$ is a connected boundary planar graph such that joining any two non-adjacent vertices by an edge embedded in the outer face of the actual embedding implies that there is a vertex not lying on the boundary of the outer face. Further, we can suppose that all vertices from $L_{G,d+1}$ are located in non-boundary faces of $H_{G,d}$.

Let C be a connected component of $K_{G,d}$ and let any vertex from $L_{G,d}$ be its representative, w.l.o.g. let it be $v_1 = w_C$. Since $K_{G,d}$ is a bipartite graph, each $H_{C,d}$ consists of isolated vertices and it is of constant depth. We show that all $K_{C,d}$ are of constant depth too.

Let d' be odd. Then $L_{C,d'} \subseteq L_{G,d+1}$. Further, there exists a set of paths $\{R_u \mid u \in L_{C,d'}\}$ such that each R_u (length of R_u is d') connects v_1 with u in C , and no two path R_u, R_v ($u \neq v$) cross. We denote the vertices in $L_{C,d'}$ by u_1, \dots, u_r in clockwise order following the paths from v_1 . Let us suppose that u_i and u_j ($i < j$) share a neighboring vertex v' . Then, due to the topology, for every k ($i < k < j$), u_k is either isolated in C , or it is joined by an edge to the only vertex v' . Moreover, distinct u_i and u_j share at most one vertex. Therefore, $K_{C,d'}$ is a forest, and by Lemma 5.7 it is representable by an RDL of constant depth.

Let d' be even and let $L_{C,d'} = \{v_{k_1}, \dots, v_{k_s} \mid k_1 < \dots < k_s\}$. Let u be the vertex from the $(d' + 1)$ -th layer of C . We show that if $(u, v_{k_i}), (u, v_{k_j}) \in E(C)$ for some $i \neq j$, then $|i - j| = 1$. Let us suppose that $(u, v_{k_i}), (u, v_{k_j}) \in E(C)$ and $j > i + 1$. Then any path from $v_{k_{i+1}}$ to w_C in C has to pass through either v_{k_i}, v_{k_j} , or u . Hence, $d' = \text{dist}(v_{k_{i+1}}, w_C) > \min\{\text{dist}(v_{k_i}, w_C), \text{dist}(v_{k_j}, w_C), \text{dist}(u, w_C)\} = d'$, a contradiction.

We can choose v_{k_1} to be the representative of the connected graph $K_{C,d'}$ (if $K_{C,d'}$ is not connected, we choose the “leftmost” v_{k_i} as the representative of a connected component C' , i.e. if C' consists of vertices $v_{k_i}, v_{k_{i+1}}, \dots, v_{k_j}$, the representative of C' is v_{k_i}). Both $H_{C',d'}$ and $K_{C',d'}$ are representable by a constant depth RDL, because $H_{C',d'}$ does not contain any edge and $K_{C',d'}$ is a stargraph.



□

There is an $O(n)$ algorithm embedding a planar graph into a plane (N. Chiba, T. Nishizeki, and S. Abe, see [1]). Therefore we can represent a planar graph by RDL in $O(n)$ time because every boundary planar subgraph that needs to be rep-

resented is already “boundary”-embedded into the plane. Since every boundary planar graph is planar we can represent a not embedded boundary planar graph in $O(n)$ time as well.

5.4 A Note on Digraphs

Digraphs can be represented by RDL with this slight modification: Let $(c_{v,i}, d_{v,i}, a_{v,i,1}, a_{v,i,0}, a_{v,i,-1})$ be the i -th tuple in the label $\text{lab}(v)$ and let $d_{v,i} = 1$. Let us denote by C the connected component of $G_{v,i}$ containing v . In this case the number $a_{v,i,-1} = 0$. We can modify the RDL and assign to $a_{v,i,-1}$ either 0, -1 , or -2 according to the direction of the arc between v and w_C (for example, 0 stands for (w_C, v) , -1 stands for (v, w_C) , and -2 stands for both of these arcs).

LDS can be modified similarly.

6 Pseudoarboricity and Density

As we have already mentioned, several authors studied representations of static graphs from the family of graphs of bounded arboricity. We want to represent dynamic graphs from this family. There are two basic approaches: Either we can assume that the graph to be represented always belongs to the family of graphs of arboricity bounded by some given α (“promise”), or we can check during the process whether the arboricity is less than α and if not, we can report an error. We have developed an algorithm which can check if some added edge results in arboricity greater than α in $O(m)$ time. The space complexity of this algorithm is optimal and the graph is always represented in the optimal space too. Since an $O(m)$ time for an update operation is too long, we wanted to find some lower bound of this time complexity. Although we were not successful in finding the lower bound, we have developed an algorithm for finding a density of a graph (see Theorems 6.6 and 6.7 for the relation to the arboricity) which gives an intuition what can or what cannot speed up the algorithm.

6.1 More Definitions and Basic Theorems

In the proofs of correctness of the algorithms we shall need several well-known theorems of the graph theory. First we introduce some basic notions.

Let $G = (V, E)$ be a graph and let $A, B \subseteq V$ be two subsets of its vertices. Any path from a vertex in A to a vertex in B is called an A - B path. We say that a set of vertices S (resp. a set of edges H) separates A from B if every A - B path contains a vertex from S (resp. an edge from H).

We can now formulate the Menger’s Theorem and its corollary:

Theorem 6.1 (Menger) *Let $G = (V, E)$ be a graph and $A, B \subseteq V$. Then the minimum number of vertices separating A from B in G is equal to the maximum number of disjoint A - B paths in G .*

Corollary 6.2 *Let a and b be two distinct vertices of G . The minimum number of edges separating a from b in G is equal to the maximum number of edge-disjoint a - b paths in G .*

In our algorithms we shall find the maximal flow in a proper network. A *network* is a directed graph which has a nonnegative capacity assigned to each of its arcs. In every network one vertex is determined as a *source* and one vertex as a *sink*. A *flow* is an assignment of nonnegative values to the arcs satisfying:

1. the value of an arc is less than or equal to the capacity of this arc,
2. let the “invalue” of a vertex be the sum of the values of arcs incoming to this vertex, and the “outvalue” be the sum of the values of arcs outgoing from this vertex,

3. for any vertex different from the source and the sink the “invalue” equals the “outvalue”,
4. the “outvalue” of the source is greater or equal to its “invalue”.

The difference of the “outvalue” and the “invalue” of the source is called a *flow-value*.

A pair of disjoint sets of vertices A and B such that $A \cup B = V$, A contains the source and B contains the sink, is called a *cut*. A *capacity of a cut* is the difference between the sum of the capacities of the arcs from A to B and the sum of the capacities of the arcs from B to A .

Theorem 6.3 (Ford & Fulkerson) *In every network, the maximum flow-value equals the minimum capacity of a cut.*

In our algorithms we shall use the decomposition of a graph into the “pseudoforests” instead of finding the “continuous” density (recall Definition 2.2). A pseudoarboricity is defined similarly as the arboricity.

Definition 6.4 *A pseudotree is a connected graph which contains exactly one cycle. A pseudoforest is a graph in which each connected component is either a pseudotree or a tree.*

Definition 6.5 *A pseudoarboricity $\vartheta(G)$ of a graph G is the minimal number of edge-disjoint pseudoforests into which G can be decomposed.*

The following results are due to Pickard and Queyranne (see [17]). They stated the relation between arboricity, density and pseudoarboricity. In the next section we give another proof of the following theorem. Our algorithms are based on this proof.

Theorem 6.6 *Let $G = (V, E)$ be a graph of density $\delta(G)$. Then $\vartheta(G) = \lceil \delta(G) \rceil$.*

Theorem 6.7 *Let G be a graph. Then either $\gamma(G) = \vartheta(G)$ or $\gamma(G) = \vartheta(G) - 1$.*

6.2 Computing the Pseudoarboricity

In this section we present an algorithm which decides whether a given number k upper-bounds the pseudoarboricity of a given graph G . (We refer to this problem as the *pseudoarboricity decision problem* (PDP).) Although its time and space complexity are the same as these complexities of the best known algorithm by Gabow and Westermann (see [7]), we believe that the straightforward graph-theoretic approach which we have developed is easier to understand than the approach of Gabow and Westermann using matroids. Our algorithm has one more

advantage — its complexity depends on a complexity of a max-flow algorithm. Thus, speed up of a max-flow algorithm results in a speed up of our algorithm.

Step by step we shall develop several algorithms leading to the desired algorithm for PDP. The main idea of the algorithms presented here consists in a proper “orientation” of the edges of the undirected graph G .

Definition 6.8 *Let $G = (V, E)$ be an undirected graph. By replacing each edge $e = (u, v) \in E$ by a directed arc (either (u, v) or (v, u)) we get an orientation $\mathcal{O}(G)$ of G .*

The following theorem states a relation between the orientations of a graph and its decompositions into the pseudoforests. The proposed algorithms are based on this relation.

Theorem 6.9 *The pseudoarboricity of a graph G equals the minimum, over all orientations of G , of the maximal outdegree of its vertices; i.e.*

$$\vartheta(G) = \min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G)),$$

where $\max_{\text{out}}(\mathcal{O}(G))$ is the maximal outdegree of vertices in the orientation $\mathcal{O}(G)$.

Proof : To prove $\vartheta(G) \geq \min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G))$, we construct an orientation $\mathcal{O}'(G)$ of G with $\max_{\text{out}}(\mathcal{O}'(G)) \leq \vartheta(G)$.

Let us decompose G into $\vartheta(G)$ edge-disjoint pseudoforests $F_1, \dots, F_{\vartheta(G)}$. For every F_i we orient its edges as follows: Let us root every tree in F_i and orient every edge of the tree from a child to the parent. In all remaining pseudotrees in F_i let us orient the edges on the cycle so that the result contains a directed cycle. Let us remove one arc, say (u, v) from the cycle and let v be the root of the resulting tree. Let us orient all remaining undirected edges from a child to the parent. Clearly, the outdegree of every vertex in F_i with respect to this orientation is at most 1. Therefore the maximal outdegree in G is at most $\vartheta(G)$.

To prove $\vartheta(G) \leq \min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G))$, we show that given an orientation $\mathcal{O}'(G)$ of G one can decompose G into $\max_{\text{out}}(\mathcal{O}'(G))$ edge-disjoint pseudoforests.

Let $\mathcal{O}'(G)$ be an orientation of G . Let us color the arcs of $\mathcal{O}'(G)$ by colors numbered $1, \dots, \max_{\text{out}}(\mathcal{O}'(G))$ so that no two arcs (u, v_1) and (u, v_2) have the same color. We call such coloring an *edge-coloring of a digraph*. Let $F_i = (V, E_i)$ be the digraph consisting of the arcs of color i and let C be one of its connected components. We need to show that C is either a tree or a pseudotree. This property follows directly from the following claim (for $H = C$ and $F = F_i$):

Let F be a digraph edge-colored by a single color. Any connected subgraph H of F is either a tree or a pseudotree.

We prove the claim by induction on $|E(H)|$. If $|E(H)| = 0$, H consists of one vertex and it is a tree. Let $|E(H)| > 0$. It is simple to observe that

$\text{outdeg}_H(v) \leq 1$ for every $v \in V$ (because of the coloring). Let u be a vertex in H such that u has no incoming arcs in H . If there is no such u , from the previous observation it follows that the component H is a cycle and thus a pseudotree. Otherwise, let (u, v) be the arc in H . By removing this arc from H we obtain connected subgraph H' which, by the induction hypothesis, is a (pseudo)tree. Since $\text{deg}_H(u) = 1$, H is a (pseudo)tree too. \square

The following min-max theorem can be proved using the previous theorem and Theorem 6.6. We give another proof of it which will result in the first proposal of an algorithm for pseudoarboricity (or ceiling of the density, by the min-max theorem).

Theorem 6.10 *Let G be a graph. Then the minimum, over all orientations of G , of maximal outdegree of its vertices equals the ceiling of the density, i.e.*

$$\min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G)) = \lceil \delta(G) \rceil$$

Proof : To prove $\min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G)) \geq \lceil \delta(G) \rceil$, let us fix some orientation $\mathcal{O}'(G)$ and suppose that H is a subgraph of G . Then $|V(H)| \cdot \max_{\text{out}}(\mathcal{O}'(G))$ edges cover all edges in H and therefore $|V(H)| \cdot \max_{\text{out}}(\mathcal{O}'(G)) \geq E(H)$. This inequality holds for every subgraph H , thus: $\max_{\text{out}}(\mathcal{O}'(G)) \geq \lceil \delta(G) \rceil$.

To prove $\min_{\mathcal{O}} \max_{\text{out}}(\mathcal{O}(G)) \leq \lceil \delta(G) \rceil$, we show the following claim:

For any graph G with n vertices, m edges and density upper bounded by some fixed $\alpha \in \mathbf{N}$ there exists an orientation $\mathcal{O}'(G)$ such that $\max_{\text{out}}(\mathcal{O}'(G)) \leq \alpha$.

We prove it by induction on m . For $m = 0$ the claim holds. Let G_1 be a graph with n vertices, $m > 0$ edges and the density upper bounded by α . Let us suppose that the claim holds for any graph of n vertices, $m - 1$ edges and density $\delta \leq \alpha$. Removing one edge (u, v) from G_1 results in a graph G_2 which fulfills the induction hypothesis. Therefore there exists an orientation $\mathcal{O}'(G_2)$ such that $\max_{\text{out}}(\mathcal{O}'(G_2)) \leq \alpha$. Now we add the removed edge to the oriented G_2 and we shall orient it. If any of its end vertices u, v has outdegree less than α , we can orient the edge out from this vertex. If not, we shall follow the outgoing arcs from both of these vertices (we can use, for example, the breadth-first search). We shall stop either if there is a vertex of outdegree less than α or if there are no more vertices to discover.

In the first case, let w be the vertex of $\text{outdeg}(w) < \alpha$ and let, e.g., u be the vertex from which there exists a directed path to w . We switch the directions of all arcs on this path and orient the edge (u, v) from u to v . The outdegrees of the interior vertices on the path and of the vertex u do not change and the outdegree of w increases by 1. Therefore every vertex in this orientation has outdegree less than or equal to α .

In the second case, the density of G_1 is greater than α , a contradiction. \square

One can easily prove Theorem 6.6 by combining Theorems 6.9 and 6.10.

We are now ready to present the first algorithm for pseudoarboricity. We shall later show that a slight modification of this algorithm solves the PDP with a better complexity.

Algorithm 6.11 (Pseudoarboricity)

Description: *Given a graph $G = (V, E)$, compute its pseudoarboricity $\vartheta(G)$ and decompose it into corresponding pseudoforests.*

1. Set $\vartheta = 0$.
2. Choose some undirected edge (u, v) . If there is none, go to step 7.
3. Perform the breadth-first search from u , following directed arcs only. If any w such that $\text{outdeg}(w) < \vartheta$ is found, reverse all arcs on the path from u to w and orient (u, v) from u to v .
(Note: Arcs counted in outdegree must be directed.)
4. If there is no such w , perform the step 3 with vertex v and continue with step 5. Otherwise go to 2.
5. If there is no such w , increase ϑ by 1. Orient (u, v) arbitrarily.
6. Go to step 2.
7. Report pseudoarboricity ϑ .
8. Color each arc by some color from $\{1, \dots, \vartheta\}$ so that for each vertex u it holds that no two distinct arcs (u, v_1) and (u, v_2) are of the same color. Report pseudoforest $F_i = (V, E_i)$ to be the graph consisting of exactly those edges colored by the color i .

Correctness of the algorithm follows from the proofs of Theorems 6.9 and 6.10. Its time complexity is $O(m^2)$, where $m = |E|$ (since the breadth-first search is $O(m)$ and it is performed for all edges), and the space complexity is $O(m)$.

Algorithm 6.12 (Pseudoarboricity Decision Problem)

Description: *Given a graph $G = (V, E)$ and a number k , decide whether $\vartheta(G) \leq k$. If yes, decompose G into at most k pseudoforests.*

1. Choose some undirected edge (u, v) . If there is none, go to step 6.
2. Perform the breadth-first search from u , following directed arcs only. If any w such that $\text{outdeg}(w) < k$ is found, reverse all arcs on the path from u to w and orient (u, v) from u to v .
(Note: Arcs counted in the outdegree must be directed.)
3. If there is no such w , process the step 2 with vertex v and continue with step 4. Otherwise go to 1.
4. If there is no such w , return $\vartheta(G) > k$.
5. Go to step 1.

6. Report that $\vartheta(G) \leq k$.
7. Color each arc by some color from $\{1, \dots, k\}$ so that for each vertex u it holds that no two distinct arcs (u, v_1) and (u, v_2) are of the same color. Report pseudoforest $F_i = (V, E_i)$ to be the graph consisting of exactly those edges colored by the color i .

The time complexity of this algorithm is $O(k^2n^2)$, while the space complexity is $O(m)$. In the rest of the chapter we shall optimize the time complexity of these algorithms. The idea of the optimization is simple — in the previous algorithms we have repeatedly performed some set of steps for each edge, and now we shall try to perform these steps for several edges at the same time. We shall reduce the pseudoarboricity problem to the max-flow problem:

Algorithm 6.13 (Pseudoarboricity Decision Problem)

Description: *The same as for the Algorithm 6.12*

1. Arbitrarily orient all edges in G .
2. Add two new vertices to G , one source s and one sink t .
3. Let S be the set of all vertices of outdegree greater than k . Add new directed arcs from s to every vertex in S .
4. Let T be the set of all vertices of outdegree less than k . Add new directed arcs from every vertex in T to t .
5. Assign a capacity $\text{outdeg}(u) - k$ to every arc (s, u) and a capacity $\min\{\text{indeg}(u), k - \text{outdeg}(u)\}$ to every arc (u, t) . Let any other arc (not incident to s or t) be unit-cost (i.e. set its capacity to 1).
6. Find a max-flow on the directed G .
7. If the value of the flow is less than $\sum_{u \in S} (\text{outdeg}(u) - k)$, return $\vartheta(G) > k$. Otherwise report that $\vartheta(G) \leq k$, switch the directions of all the “flow” edges, delete s and t with all their incident edges, and report pseudoforests (see Algorithm 6.12, step 7).

Theorem 6.14 *Algorithm 6.13 is correct.*

Proof : We prove the following claim:

Let $\mathcal{O}_1(G)$ be some orientation of G , $k \in \mathbf{N}$ be an integer, $S = \{u \in V \mid \text{outdeg}(u) > k\}$ and $T = \{u \in V \mid \text{outdeg}(u) < k\}$. Let $M = \sum_{u \in S} (\text{outdeg}(u) - k)$. Then the pseudoarboricity is less than or equal to k iff there exist M edge-disjoint directed S - T paths such that every $v \in S$ is used as a start-vertex of an S - T path exactly $\text{outdeg}(v) - k$ times and no end-vertex $u \in T$ is used as an end-vertex more than $k - \text{outdeg}(u)$ times.

Let us suppose that there exist M edge-disjoint S - T paths claimed. We can reverse all arcs on each of the paths and obtain an orientation $\mathcal{O}_2(G)$. (Notice

that outdegrees have changed only for the start and end-vertices.) Since every vertex $u \in S$ is a start-vertex of exactly $\text{outdeg}(u) - k$ paths, by reversing the arcs every $v \in S$ decreases its outdegree by $\text{outdeg}(u) - k$. Since every vertex $u \in T$ is an end-vertex of no more than $k - \text{outdeg}(u)$ paths, its outdegree increases by at most $k - \text{outdeg}(u)$. Therefore the maximal outdegree in orientation $\mathcal{O}_2(G)$ is upper-bounded by k and by the Theorem 6.9 the pseudoarboricity of G is less or equal to k .

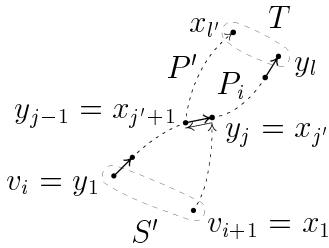
Conversely, if $\vartheta(G) \leq k$, we shall inductively construct M required edge-disjoint S - T paths. Let S' be the multiset consisting of all vertices from S such that every vertex $v \in S$ is in S' with a multiplicity $\text{outdeg}(u) - k$ times. (Notice that a set of edge-disjoint S - T paths is also a set of edge-disjoint S' - T paths, and vice-versa.) Let us denote the vertices in S' by v_1, \dots, v_q (not necessarily distinct) and for $i \in \{1, \dots, q\}$ let us suppose that edge-disjoint paths P_1, \dots, P_i have been found, such that P_j starts in v_j and no vertex $u \in T$ is an end-vertex of more than $k - \text{outdeg}(u)$ paths. Let $\mathcal{O}_{2,i}(G)$ denote the orientation of G obtained from $\mathcal{O}_1(G)$ by reversing all arcs on the paths P_1, \dots, P_i . By performing the breadth-first search from v_{i+1} on $\mathcal{O}_{2,i}(G)$, one can find a vertex $w \in T$ such that $\text{outdeg}_{\mathcal{O}_{2,i}(G)}(w) < k$. If no such w exists, the subgraph of G containing all the vertices inspected by the search has density greater than k and therefore, by Theorem 6.6, $\vartheta(G) > k$, a contradiction.

Let $P' = x_1, \dots, x_{\nu}$ be a directed path from $x_1 = v_{i+1}$ to $x_{\nu} = w$. We shall inductively rearrange the paths P_1, \dots, P_i and P' so that the new set of paths $P_{1,\text{new}}, P_{2,\text{new}}, \dots, P_{i+1,\text{new}} = P'_{\text{new}}$ satisfies:

1. Any two different paths from the set are edge-disjoint,
2. the set of edges in G covered by paths P_1, \dots, P_i, P' is identical with the set of edges covered by $P_{1,\text{new}}, \dots, P_{i+1,\text{new}}$,
3. P_j starts in v_j for each j ,
4. no vertex $u \in T$ is an end-vertex of more than $k - \text{outdeg}(u)$ paths.

Let us switch back to the orientation $\mathcal{O}_1(G)$. Let j' be the longest index such that $(x_{j'+1}, x_{j'})$ belongs to some of the P_1, \dots, P_i paths, w.l.o.g. let it be P_i . If there is no such j' , the path P' shares no edge with any other path and therefore

it could be chosen for P_{i+1} . Otherwise, $P_i = y_1, \dots, y_l$ (in orientation $\mathcal{O}_1(G)$), where $y_1 = v_i$ and $y_j = x_{j'}$ for some $j \geq 2$. We rearrange the edges on the paths P_i and P' . Let the new P_i be $y_1, \dots, y_{j-1} = x_{j'+1}, x_{j'+2}, \dots, x_{\nu}$ and let the new P' be $x_1, \dots, x_{j'} = y_j, y_{j+1}, \dots, y_l$. We can continue with the process described in this paragraph with the new paths P_1, \dots, P_i and the new P' . Since each step decreases the number of edges of P' shared with some other path, the process is finite and results in finding a suitable P_{i+1} . This completes the proof of the claim.



In order to complete the proof of the theorem, let $\mathcal{O}_1(G)$ be the orientation of G in step 1 and let G' be the weighted digraph with added s and t on which max-flow is to be performed in step 6. Notice, that S - T paths specified in the claim extended to the start-vertex s and the end-vertex t form a max-flow on G' . Conversely, if max-flow on G' of value M is found, by Ford & Fulkerson and Menger's Theorems (see 6.3 and 6.2) one can find such S - T paths. \square

The time complexity of the Algorithm 6.13 depends on the time complexity of the max-flow algorithm. According to the survey by Andrew V. Goldberg (see [8]), the integer capacity max-flow can be computed in $O(\min(n^{2/3}, m^{1/2})m \log(n^2/m) \log U)$, where $U > 1$ is the upper bound of the capacity of an edge and n and m are the number of vertices and edges, respectively. This is the result of Andrew V. Goldberg and Satish Rao, see [9]. In our case, the graph G' on which the max-flow is going to be performed has capacities less than $n' = O(n)$, and $m' = O(m)$, because $\sum_{v \in V} (\text{indeg}(v) + \text{outdeg}(v)) = O(m)$. Therefore finding the max-flow on G' has time complexity $O(\min(n^{2/3}, m^{1/2})m \log(n^2/m) \log n)$. If we determined $\vartheta(G)$ by the binary search using Algorithm 6.13, the overall time complexity of the algorithm for pseudoarboricity would be $O(\log(\delta(G)) \min(n^{2/3}, m^{1/2})m \log(n^2/m) \log n)$.

We can improve this time complexity by using a max-flow algorithm on a graph with all edges of unit-cost. We shall slightly modify the previous algorithm by omitting the step 5 and modifying the steps 3, 4, 6, and 7:

Algorithm 6.15 (Pseudoarboricity Decision Problem)

Description: *The same as for the Algorithm 6.12*

1. *Arbitrarily orient all edges in G .*
2. *Add two new vertices to G , one source s and one sink t .*
- 3'. *Let $p_u = \text{outdeg}(u) - k$. For every vertex $u \in S$ add p_u new vertices v_1, \dots, v_{p_u} and new directed arcs (s, v_i) and (v_i, u) for all $i \in \{1, \dots, p_u\}$.*
- 4'. *Let $q_u = \min\{\text{indeg}(u), k - \text{outdeg}(u)\}$. For every vertex $u \in T$ add new vertices v_1, \dots, v_{q_u} and new directed arcs (u, v_i) and (v_i, t) for all $i \in \{1, \dots, q_u\}$.*
- 5'. *Skip.*
- 6'. *Find a max-flow on G with all unit-cost edges.*
- 7'. *If the value of the flow is less than $\sum_{u \in S} (\text{outdeg}(u) - k)$, return $\vartheta(G) > k$. Otherwise report that $\vartheta(G) \leq k$, switch the directions of all the "flow" edges, delete every new vertex with all its incident edges, and report pseudoforests (see Algorithm 6.12, step 7).*

The problem of finding the max-flow on a directed graph with unit-cost edges was solved independently by Alexander V. Karzanov (see [13]) and by Shimon

Even and R. Endre Tarjan (see [4]). The time complexity of the algorithm is $O(\min(n^{2/3}, m^{1/2})m)$.

In the Algorithm 6.15, the graph G' on which the max-flow algorithm is going to be performed (in step 6') has $O(n+m)$ vertices and $O(m)$ edges. Therefore the max-flow algorithm takes $O(m^{3/2})$ time (we can assume that $n = O(m)$). Using the idea of the binary search then completes the proof of the following theorem:

Theorem 6.16 *For a given graph G and an integer k the question “Is the density of G less than or equal to k ?” can be answered in $O(m^{3/2})$ time and $O(m)$ space. The pseudoarboricity $\vartheta(G)$ of a graph G and a decomposition of G into $\vartheta(G)$ pseudoforests can be found in time $O(\log(\vartheta(G))m^{3/2})$.*

Several authors studied the k -pseudoforest problem (here pseudoarboricity decision problem, see Algorithm 6.12): Jean-Claude Picard and Maurice Queyranne ($O(m^4)$, see [17]), James Roskind and Robert E. Tarjan ($O(m^2)$, see [18]) and Harold N. Gabow and Herbert H. Westermann ($O(m^{3/2})$, see [7]). In their paper, Gabow and Westermann developed some algorithms for matroids and sums of matroids and by running the algorithms on a suitable graphic matroid they obtained the same time complexity as the algorithm presented here. The main contribution of our algorithm is in the reduction to a max-flow algorithm. The coherence between the lower-bounds for the time complexity of the max-flow problem and for the decision problem for the pseudoarboricity remains an open problem.

6.3 Approximation Algorithm

At the end of the chapter we propose an approximation algorithm for the pseudoarboricity problem.

Algorithm 6.17 (Approximation of Pseudoarboricity)

Description: *Given a graph $G = (V, E)$, compute a number ϑ' not greater than $2\vartheta(G)$ and decompose G into ϑ' pseudoforests.*

1. Initially, let $G_1 = (V_1, E_1)$ be a copy of G and let $\vartheta' = \min_{v \in V} \deg(v)$.
2. If there is no remaining vertex in G_1 , go to step 6. Otherwise choose a vertex $u_1 \in V_1$ having minimal degree. If $\deg(u_1) > \vartheta'$, let $\vartheta' = \deg(u_1)$.
3. Orient each undirected edge $(u, v) \in E$ from u to v , i.e. change it into an arc (u, v) .
4. Delete vertex u_1 and all its incident edges from G_1 .
5. Go to step 2.
6. Report ϑ' and construct pseudoforests by coloring the outgoing arcs by colors $1, \dots, \vartheta'$. (See Algorithm 6.11, step 8.)

Proof of correctness: Let us suppose that G is a graph of density δ . By the pigeon-hole principle in any subgraph H of G there exists a vertex u such that $\deg_H u \leq 2\delta$. (Otherwise $E(H) > \delta V(H)$, a contradiction with the definition of the density.) Therefore the vertex u_1 chosen in step 2 has $\deg_{G_1} u_1 \leq 2\delta$, thus $\vartheta' \leq 2\delta \leq 2\vartheta$. \square

It often suffices to have some reasonable approximation of the pseudoarboricity. The Algorithm 6.17 may be helpful in this respect. Its main advantage is its time and space complexity — both are optimal: $O(m)$. Finding a better approximation may further contribute as follows: If there was an approximation algorithm reporting ϑ' such that $\vartheta' \leq f(G)\vartheta(G)$, where $f(G) = 1 + o(1)$, the time complexity of the algorithm for pseudoarboricity using binary search could be improved (the searched interval would be smaller). Finding such approximation algorithm remains an open problem too.

7 Summary of Open Problems

We propose several open problems and suggestions for further work:

- Is it possible to improve the hashing-based representation of dynamic graphs?
- Does there exist an optimal recursive distance-representation for the graphs of bounded arboricity?
- What is the lower bound for the size of a distinguishing set of a p -dimensional hypercube? (We conjecture that it is $p + 1$.)
- Is it possible to represent a p -dimensional hypercube by a recursive distance-labeling efficiently?
- Can the knowledge of the arboricity of a dynamic graph be helpful in maintaining its representation?
- What is the relation between the time complexity of the max-flow algorithm and the algorithm for pseudoarboricity decision problem? We have shown that the PDP can be solved in the same time as the max-flow on directed graph with all unit-cost edges. Does it hold conversely?
- Does there exist a better approximation algorithm for pseudoarboricity problem running in reasonable time? Such an algorithm could be used to find a faster algorithm for computing the pseudoarboricity.

References

- [1] N. Chiba, T. Nishizeki, and S. Abe, *A Linear Algorithm For Embedding Planar Graphs Using Pq-Trees*, J. COMPUTER SYSTEM SCIENCES(30), 1985, pages 54-76.
- [2] R. Diestel, *Graph Theory*, Springer-Verlag, 1997.
- [3] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R.E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds*, Proc. 29th IEEE Annual Symposium on Foundations of Computer Science, 1988, pages 524-531.
- [4] S. Even and R.E.Tarjan, *Network Flow and Testing Graph Connectivity*, SIAM J. Comput., Vol. 4., No. 4, 1975, pages 507-518.
- [5] T. Feder, and R. Motwani, *Clique Partitions, Graph Compression and Speeding-up Algorithms*, Proc. 23rd Annual ACM Symposium on the Theory of Computing, 1991, pages 123-133.
- [6] M.L. Fredman, J. Komlós, and E. Szemerédi, *Storing a Sparse Table with $O(1)$ Worst Case Access Time*, Journal of the ACM, 31:538-544, July 1984.
- [7] H.N. Gabow and H.H. Westermann, *Forests, frames and games: Algorithms for matroid sums and applications*, Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pages 407-421.
- [8] A.V. Goldberg, *Recent Development in Maximum Flow Algorithms*, Technical Report #98-045, NEC Research Institute, Inc., 1998.
- [9] A.V. Goldberg and S. Rao, *Beyond the Flow Decomposition Barrier*, Proc. 38th IEEE Annual Symposium on Foundations of Computer Science, 1997, pages 2-11.
- [10] A. Itai, and M. Rodeh, *Representation of Graphs*, Acta Informatica 17, 1982, pages 215-219.
- [11] G. Jacobson, *Space-efficient Static Trees and Graphs*, Proc. 30th IEEE Annual Symposium on Foundations of Computer Science, 1989, pages 549-554.
- [12] S. Kannan, M. Naor, and S. Rudich, *Implicit Representation of Graphs*, Proc. 20th Annual ACM Symposium on the Theory of Computing, 1988, pages 334-343.

- [13] A.V. Karzanov, O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh (in Russian, title translation: *On Finding Maximum Flows in Networks with Special Structure and Some Applications*), Matematicheskie Voprosy Upravleniya Proizvodstvom, Vol. 5, Moskow State University Press, Moskow, 1973.
- [14] R. Motwani, and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- [15] M. Naor, *Succinct Representation of General Unlabeled Graphs*, Discrete Applied Mathematics 28, 1990, pages 303-307.
- [16] C.St.J.A. Nash-Williams, *Edge-disjoint spanning trees of finite graphs*, J. London Math. Soc. 36, 1961, pages 445-450.
- [17] J.-C. Picard and M. Queyranne, *A Network Flow Solution to Some Nonlinear 0-1 Programming Problems, with Applications to Graph Theory*, Networks, Vol. 12, 1982, pages 141-159.
- [18] J. Roskind and R.E. Tarjan, *A Note on Finding Minimum-Cost Edge-disjoint Spanning Trees*, Mathematics of Operations Research, Vol. 10, No. 4, 1985, pages 701-708.
- [19] M. Talamo, and P. Vocca *Compact Implicit Representation of Graphs*, Proc. 24th International Workshop, Graph-Theoretic Concepts in Computer Science, WG, Lecture Notes in Computer Science, Vol. 1517, Springer, 1998, pages 164-176.
- [20] G. Turán, *On the Succinct Representation of Graphs*, Discrete Applied Mathematics 8, 1984, pages 289-294.