COMPLEXITY OF GRAPH PROBLEMS

BY

PÉTER HAJNAL

BOLYAI INSTITUTE, SZEGED, HUNGARY

JANUARY, 1992

ACKNOWLEDGEMENT

I am grateful to my advisors, János Simon and László Babai. They have provided encouragement and support, and their insights throughout this research have been invaluable. It has been a real pleasure to work with them.

Two chapters of this thesis would not have been possible without collaboration with Endre Szemerédi. He has been a great and generous collaborator, and deserves very special thanks for sharing his research.

I owe particular gratitude to László Lovász, my former teacher at the University of Szeged, Hungary, who directed my interest to combinatorics and algorithms. His time spent on suggesting and discussing problems with me outside class have helped enormously in starting my own research career.

I should like to thank Howard Karloff and György Turán for listening to my blurbs, giving many suggestions and helping to clarify ideas in various sections of the dissertation.

I am grateful to fellow graduate student Márió Szegedy for his untiring interest in my work and for his selfless help in many details.

Last but not least, I wish to thank the University of Chicago and professors Robert Soare and Mike O'Donnell in particular for the opportunity to work here and the active scholarly atmosphere in the Department under their direction.

This research was supported in part by the National Science Foundation under Grant number NSF 5-27561.

TABLE OF CONTEST

ABSTRACT

We present complexity analysises of graph theoretic problems. We shall seek lower and upper bounds on various complexity measures of classes of graph problems.

We study several models of computation. A decision tree is a scheme of computing a boolean function by asking the values of its variables. The choice of question can only depend on the information gained so far plus, in the randomized model, on the outcome of coin-tosses. A function is said to be 'hard' if in the worst case we are forced to ask at least a positive constant fraction of the variables.

Branching programs are a generalization of decision trees. Roughly speaking we allow two branches in the tree to merge. In the model we consider we make the additional assumption that the program is allowed to query each variable at most once along any computation path. A function is said to be 'hard' in this model if we cannot save considerable amount of work compared to the decision trees.

A parallel random access machine is set of random access machines communicating through a shared memory. A problem is said to be 'easy' in this model if it is solvable on a parallel random access machine in time polynomial in the log of the input size and using a polynomial number of processors.

Lower bounds are the subject of the second and third chapters. In the second chapter we prove that non-trivial, monotone graph properties require many questions in the randomized decision tree model. The third chapter exhibits a polytime computable function that is hard in the read-once-only branching program model. In the next two chapters we present *fast parallel algorithms* for Brooks coloring graphs and for finding a Hamiltonian cycle in graphs that are guaranteed to have one by Dirac's theorem. These algorithms show that these problems are easy on the parallel random access model.

In the last two chapters we consider some related questions in discrete geometry and extremal theory of graphs. We investigate how many vertex pair can be unit distance apart among the vertices of a convex n-gon. Finally we give some algorithms for finding a vertex partition with constrains on the connectivity and minimal degree of a given graphs.

1. INTRODUCTION

1. Tasks, models, complexities

The Theory of Computing investigates the intrinsic complexity of various classes of computational tasks on abstract models of computation.

What is a computational task? It is defined by a function from the set of possible inputs to the set of possible outputs. The inputs and outputs can usually be encoded as $(0,1)$-sequences. Thus, the computational task is a function from $(0,1)$-strings to $(0,1)$-strings. If the range of this function is simply $\{0,1\}$, i.e. the output or answer is 'no' or 'yes', then we have a decision problem.

The other component of the computational problem is the 'machine'. We need an abstract device and a set of instructions we are allowed to use in order to solve the task. Digital computers are physical devices for solving problems. There are several known theoretical models that are idealized versions of digital computers. Some of the most often used ones are: random access machine (RAM), Turing machine, boolean formula, boolean circuit, algebraic circuit, communication protocol, branching program, decision tree, parallel RAM (PRAM), systolic array [AHU74], [Tu37], [Sh49], [Ma74], [Ya79], [Bor77], [Co85], [Go77], [FW78], [HB84], etc.

The definition of machine usually indicates the definition of the 'algorithm'. The algorithm, vaguely, is a 'recipe', a set of instructions. Given an input ($(0,1)$-string) we can execute the instructions and reach the output. The problem is to design an algorithm which computes the given function.

In order to compare different algorithms we need a notion of complexity or a cost measure for algorithms. In order to define the costs we consider the resources available for the computation. Some of the most commonly considered resources are: time, space, number of queries, number of arithmetic operations, number of random bits used, number of parallel processors, chip area, depth and size of boolean circuits. In each model of computation, we specify one or more of those resources which we shall take into account when calculating the cost of a computation. Let, informally, $c(\underline{x})$ denote the complexity of the computation on input $\underline{x}$. In most of this work we shall be concerned with 'worst case' complexity, i.e. for given n, we shall try to estimate the quantity $f(n) = \max\{c(\underline{x}) \mid \underline{x} \in \{0,1\}^n\}$.

We should point out that the same computational task raises different computational problems depending on the model.

Determining the complexity of a problem has two sides.

First, we would like to exhibit an efficient algorithm, i.e. write a fast program, design a small circuit, etc. for the problem. This requires insight into the nature of the specific problem. One can interpret this part as giving an upper bound on the complexity.

Second, we want to prove that there is no algorithm with smaller complexity. Usually the second part is much harder than the first. It requires insight into the nature of computation itself. One has to find obstructions that prevent *any* algorithm from violating the lower bound. While the number of efficient algorithms published during the last few decades is tremendous, the area of lower bounds has produced far fewer impressive results. Lower bounds on general models of computation are virtually non-existent. For example, in the case of circuits the best known lower bound on the number of gates is $3n$ (n is the number of variables) for any explicit function [Bl84], whereas the conjectured lower bound for a large class of practical problems is exponential. It seems that proving any non-linear lower bound, for boolean circuit size is beyond current techniques. In practice what happens is that we restrict our model. In restricted models, like constant depth circuits or monotone circuits, powerful lower bound techniques are available.

The objective of this thesis is to contribute to both areas, the area of proving lower bounds and that of exhibiting efficient algorithms.

2. A brief description of our models

In this section we give a brief description of the models we work with. The exact definitions will be given in the subsequent chapters.

One model we use is the *decision tree* model. A decision tree evaluates a boolean function by asking questions of the form 'What is the value of variable x_i?'. The choice of question may depend only on the information gained so far, and in the randomized model, on the outcome of random coin-tosses. The cost is the number of queries; we ignore computation cost. (Imagine that evaluating a variable requires a costly experiment.) Most previous work in this model was on deterministic computation. While many lower bounds were found for specific problems, like 'to be Hamiltonian', 'having k-coloring', etc. ([Bo77], [BE78]) much effort has been devoted to giving lower bounds not just for specific computational tasks, but to proving a uniform lower bound for a large class of problems. One class of particular interest is graph properties. In this case the variables correspond to pairs of nodes (the value of a variable tells us whether or not that pair is adjacent in the input graph) and the output depends only on the isomorphism class of the graph. For non-trivial, monotone graph properties it is known that the complexity is $\Omega(v^2)$ [RV76].

We shall consider the randomized model. This means that the solver (the algorithm) can use coin flips before deciding the next question. We are looking for uniform lower bounds for all non-trivial, monotone graph properties. In the deterministic case the complexity is known up to a constant factor. Thus one can view our question as investigating how much can be gained by randomization in this particular model, an instance of a more general problem that has attracted considerable interest in a number of models of computation [Ad78], [Ya82], [AW85], [AKSz87], etc.

The second model of computation we examine is branching programs. Branching programs are a generalization of decision trees. Roughly speaking, we allow two branches in the tree to merge. The cost of a branching program is the number of nodes in it. A read-once-only branching program is allowed to test each variable at most once along any computation path.

Lower bounds on the read-once-only branching programs easily imply space lower bounds on a restricted RAM model which we call *eraser RAM*. This is a RAM with a special read-only input tape. Once an input cell has been read, it is erased.

Our algorithmic results concern efficient parallel computation. We use the PRAM model. In this model we have several processors working simultaneously, communicating through a shared memory. The cost of the computation has two components. One is the number of processors, the other is time. A parallel algorithm is considered efficient if it uses polynomially many processors and its time complexity is polylogarithmic in the size of the input; in this case we say that our algorithm is an NC algorithm. In the last few years many authors have considered the additional objective of minimizing the number of processors used.

3. Outline of our results

In the second chapter we give an $\Omega(v^{\frac{4}{3}})$ lower bound on the randomized decision tree complexity for all non-trivial, monotone graph properties. This improves A. Yao's and V. King's previous bounds. The proof follows Yao's approach and improves it in a direction different from King's. At the heart of the proof is a duality argument combined with a new packing lemma for bipartite graphs.

In the third chapter we give a C^n lower bound for read-once-only branching programs computing an explicit Boolean function, for some absolute constant $C > 1$. For $n = \binom{v}{2}$, the function computes the parity of the number of triangles in a graph on v vertices. This improves previous $\exp(c\sqrt{n})$ lower bounds for other graph functions by Wegener and Zák. The result implies a linear lower bound for the space complexity of this Boolean function on eraser machines.

A theorem of Brooks guarantees that one can properly color a graph of maximum degree $D \geq 3$ with D colors if the graph doesn't contain a complete subgraph on $D + 1$ vertices. In the fourth chapter we prove that finding a Brooks coloring is in NC, i.e. we can construct a coloring guaranteed to exist by Brooks' theorem in polylog parallel time, using a polynomial number of processors.

In graph theory several sufficient conditions are known for existence of a Hamiltonian cycle. One of the classical ones is Dirac's condition, which can be stated as follows. If the minimal degree of a simple graph is at least $\frac{v}{2}$ where v is the number of nodes then it has a Hamiltonian cycle. The known proofs provide a polynomial time sequential algorithm. In the fifth chapter we give an NC algorithm for finding a Hamiltonian cycle in such graphs. Our algorithm uses a linear number of processors (therefore, apart from a polylogarithmic factor, is optimal).

In the sixth chapter we investigate how many vertex pair can be unit distance apart among the vertices of a convex n-gon. We give a construction of a convex n-gon determining $\frac{9}{5}n - 13$ unit distances. The upper bound technique of Z. Füredi leads us to consider the following question: We say that a matrix $M = (m_{ij})$ does have the configuration C if one can find u rows $i_1, i_2, \ldots, i_u, i_1 < \cdots < i_u$ and v columns $j_1, j_2, \ldots, j_v, j_1 < \cdots < j_v$ in M such that the corresponding submatrix contains C, i.e. $m_{i_\alpha, j_\beta} = 1$ whenever $c_{\alpha,\beta} = 1$. Let $f(n, m; C)$ denote the maximum number of 1's in an $n \times m$ matrix M not containing C. We determine this threshold function for several configurations.

In the final chapter we prove that there exists a number $f(s, t)$ such that every $f(s, t)$-connected graph G admits a proper partition $\{ S, T \}$ of the vertex-set $V(G)$ so that the induced subgraphs $G(S)$ and $G(T)$ are s-connected and t-connected, respectively.

4. Graph theoretical notation, terminology

We mostly use the standard notation of graph theory (cf. [Lo79], [Be73]).

A *graph* G consists of a finite set of *vertices* or *nodes* $V(G)$ (or simply V) and a finite set of *edges* $E(G)$ (or E). An edge is a pair of nodes. Let $e = (x, y)$ be an edge. We think of e as joining x and y, and we say x and y are *adjacent*. We call x and y the *endvertices* or *endnodes* of e. (Note: it is common to allow E to be a multiset and to have singletons, so called loops. Throughout this thesis we do not use this general notion.) Let us denote the set of all the $2^{\binom{v}{2}}$ possible graphs on the vertex set $V = \{ 1, \ldots, v \}$ by $\mathcal{G}_v$. Two graphs G and $H \in \mathcal{G}_v$ are called *isomorphic* if there exists a permutation σ of $V = \{ 1, 2, \ldots, v \}$ such that $G = H^\sigma$, where $H^\sigma = (V, \{ (x^\sigma, y^\sigma) | (x, y) \in E \})$. Isomorphic graphs look the same but in $\mathcal{G}_v$ one can distinguish them by using the 'names' of the vertices.

We need some other graph universes. A *directed graph* is a set of vertices V and a set of ordered pairs of nodes E, called edges or arcs. For $e = \langle x, y \rangle$ an edge, x is called the tail of e and y is called the head of e. Again we don't allow multiplicities and loops. (In this case the same pair of nodes can be connected by two edges but they must have opposite orientation.) Let $\mathcal{D}_v$ be the set of all the $2^{v(v-1)}$ directed graphs on the vertex set $V = \{ 1, \ldots, v \}$.

We shall also consider the subclass of directed graphs called *oriented graphs*. A directed graph is an oriented graph if no two edges connect the same pair of nodes. These graphs are exactly the directed graphs which can be obtained by orienting the edges of some undirected graph.

An important class of undirected graphs is the class of *bipartite graphs*. A graph G is bipartite with bipartition (U, W) if its vertex set can be partitioned into two sets U, W in such a way that each edge of G joins one vertex from U to a vertex of V. We refer to U and V as *color classes*. Let $\mathcal{B}_{u,w}$ be the set of all the 2^{uw} bipartite graphs on $U \cup W = \{ 1, \ldots, u \} \cup \{ \bar{1}, \ldots, \bar{w} \}$. (Keep in mind that V is the disjoint union of U and W, in notation: $V = U \dot{\cup} W$.)

A *directed bipartite graph* with bipartition (U, W) is a directed graph $(U \dot{\cup} W, E)$ where $E \subset (U \times W) \cup (W \times U)$. Let $\mathcal{DB}_{u,w}$ be the set of all 4^{uw} directed bipartite graphs on the bipartition $(U = \{ 1, \ldots, u \}, W = \{ \bar{1}, \ldots, \bar{w} \})$.

For $\mathcal{G}_v, \mathcal{B}_{u,w}$, etc. our definitions fix the vertex set. Sometimes to avoid confusion, when we work with two graphs from the same universe we distinguish the two vertex sets by using primes in one case.

Let us consider some special graphs. A v-node graph which has all the $\binom{v}{2}$ possible

edges is called *complete graph* and denoted by K_v. If we want to emphasize the vertex set of the complete graph then we write K_V. The empty graph, E_v or E_V is a graph with empty edge set. A *complete bipartite graph* has all the uw edges between U and W. It is denoted by $K_{u,w}$ or $K_{U,W}$. If we take an orientation of the complete bipartite graph such that the tail of each directed edge is in the same color class than we call it a *one-way complete bipartite graph.*

Some more notation.

$G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subset V$, $E' \subset E$ and each element of E' joins two vertices from V'. $G' = (V', E')$ is the subgraph of G *induced* by V' if it is a subgraph and E' has all the edges of G connecting nodes from V'. In this case we use the notation $G' = G|V'$.

If we have a $1-1$ function f defined on the vertex set of G then the image of G will be $G^f = (f(V(G)), \{\, (f(x), f(y)) | (x, y) \in E(G) \,\})$. If f is only a partial function on $V(G)$ then it is defined only on a subset of the nodes. For the other nodes we extend it as the identity map and G^f will be defined according to this.

The *disjoint union* of G and H is $G \dot\cup H = (V(G) \dot\cup V(H), E(G) \dot\cup E(H))$.

Let G be a graph and let H be one of its subgraphs. Then $G - H = (V(G), E(G) - E(H))$.

The *complement* of G is $\overline{G}$, a graph on the same vertex set having all the pairs of nodes as edges which are not in G, i.e. $K_{V(G)} - G$.

Let $d_G(x)$ be the *degree* of a node x, i.e. the number of edges incident on x. If from the context it is clear what the underlying graph G is then we shall omit the subscript. Let $D(G)$ be the maximal degree of G. Let $\delta(G)$ be the minimal degree of G. Let $\bar{d}(G)$ be the average degree of G, i.e. $\bar{d}(G) = \frac{1}{v} \sum_{x \in V} d(x) = \frac{2e}{v}$ (where e is the number of edges in G). $D_S(G), \delta_S(G)$ and $\overline{d}_S(G)$ denote the maximum, minimum and average degrees, resp., over $S \subset V(G)$. (Here degrees are meant relative to G and *not* to the induced subgraph $G|S$.)

The set of nodes adjacent to x is the *neighborhood* of x and it is denoted by $N_G(x)$. If from the context it is clear that we work with G then we omit the subscript.

A subgraph P of G is a *path* if $P = (\{\, x_1, \ldots, x_k \,\}, \{\, (x_1, x_2), \ldots, (x_{k-1}, x_k) \,\})$, for some set $\{\, x_1, \ldots, x_k \,\}$ of k distinct vertices.. A subgraph C of G is a *cycle* if $C = (\{\, x_1, \ldots, x_k \,\}, \{\, (x_1, x_2), \ldots, (x_{k-1}, x_k), (x_k, x_1) \,\})$, for some set $\{\, x_1, \ldots, x_k \,\}$ of k distinct vertices. If a path or cycle goes through all the nodes (i.e. its vertex set is $V(G)$) then it is called a *Hamiltonian path or cycle*, respectively. If a graph has a Hamilton cycle it is called *Hamiltonian.*

By *coloring a graph* G with colors $\{\, 1, 2, \ldots, l \,\}$ we mean a function $c : V \to \{\, 1, \ldots, l \,\}$. A coloring is *proper* if for each edge the colors of the endnodes are different.

A set of edges is *independent* if no two edges from the set share an endpoint. A set of independent edges is called a *matching.* A matching M is a *perfect matching* if the

endvertices of the edges in M cover the whole vertex set. A perfect matching defines a pairing of the vertex set.

Finally we define some specific graph properties.

A vertex v is a *sink* in an oriented graph G if it is connected to all other nodes and each edge incident to it is directed toward v.

A graph G is called a *scorpion graph* if it contains a vertex adjacent to all but one of the vertices, and the one to which it is not adjacent has degree 1 and its single neighbor has degree 2.

The thesis is divided into chapters having sections. If we refer to a theorem then we use the numbering of chapters and theorems (this includes the section number too) to identify it, e.g., theorem II.2.9. If we omit the number of the chapter then the reference in question is in the same chapter.

2. LOWER BOUNDS FOR RANDOMIZED DECISION TREES

1. Decision trees

Decision trees are a computational model for boolean functions.

Definition 1.1. A *decision tree* is a rooted binary tree with labels on each node and edge. Each inner node is labeled by a variable symbol. This label represents a query for the value of the corresponding variable. One of the two edges leaving the node is labeled 0, the other is labeled 1. The two labels represent the two possible answers. The two subtrees at a node describe how the algorithm proceeds after receiving the corresponding answer. Each leaf is labeled 0 or 1. These labels give the output, i.e. the value of the function.

Clearly, each truth-assignment to the variables determines a unique path, the *computation path*, from the root to a leaf of the tree. The boolean function computed by the given decision tree takes the label at this leaf as the value on the given input.

Definition 1.2. Let $\text{cost}(\mathcal{A}, x)$ be the number of queries asked when the decision tree $\mathcal{A}$ is executed on input x. This is the length of the computation path forced by x.
$\max_x \text{cost}(\mathcal{A}, x)$ is the worst case complexity of $\mathcal{A}$, i.e. the depth of the tree.
The *decision tree complexity* of a boolean function f is $\mathcal{C}(f) = \min_{\mathcal{A}} \max_x \text{cost}(\mathcal{A}, x)$, where the first minimum is taken over all decision trees $\mathcal{A}$ computing the function f.

It is obvious that the complexity of any function f is at most the number of its variables.

Definition 1.3. A function f of n variables is *evasive* if $\mathcal{C}(f) = n$.
(Sometimes this property is referred to as elusive or exhaustive.)

The first step toward understanding decision trees might be to consider specific functions and to determine their decision tree complexities. Much effort has been spent in this direction ([Bo77], [BE78], [BBL74], [HR72], [Kir74], [MW74], [MW76]).

It is somewhat surprising that most boolean functions require querying all the variables in the worst case [RV76].

Theorem 1.4. (R.L. Rivest and S. Vuillemin [RV76]) *As $n \to \infty$, almost all boolean functions of n variables are evasive.*

This result gives us hope that one might find uniform lower bounds for a broad class of boolean functions.

Definition 1.5. For a class of boolean functions $\mathcal{F}$ let $\mathcal{C}(\mathcal{F}) = min_{f \in \mathcal{F}} \mathcal{C}(f)$.

Let us mention some classes in the literature.

A boolean function is *monotone* if changing the value of a variable from 0 to 1 cannot change the value of the function from 1 to 0.

A boolean function is *non-trivial* if it is not constant.

We can identify graphs in $\mathcal{G}_v$ with $(0,1)$-strings of length $\binom{v}{2}$. *Graph properties* are boolean functions $f : \mathcal{G}_v \to \{0,1\}$ taking equal values on isomorphic graphs. Let $\mathcal{GP}_v$ denote the set of graph properties over $\mathcal{G}_v$.

Along the same lines one can define digraph properties, bipartite graph properties and directed bipartite graph properties and identify them with classes of boolean functions. Let us denote the corresponding sets of properties (over a given size of universe) by $\mathcal{DP}_v$, $\mathcal{BP}_{u,w}$ and $\mathcal{DBP}_{u,w}$. So we can talk about $\mathcal{C}(\mathcal{GP}_v)$, $\mathcal{C}(\mathcal{DP}_v)$, $\mathcal{C}(\mathcal{BP}_v)$ and $\mathcal{C}(\mathcal{DBP}_v)$.

The case of oriented graphs is somewhat different. The number of oriented graphs on v vertices is $3^{\binom{v}{2}}$. But we can talk about solving oriented graph properties in the decision tree model by slightly modifying the model. Each query $\{x, y\}$ has three possible outcomes (no edge; edge $\langle x, y\rangle$; edge $\langle y, x\rangle$). So in the tree describing the decision process, each inner node has three children. We refer to this as the *ternary decision tree* model.

First let us see some linear lower bounds.

Theorem 1.6. (B. Bollobás and S.E. Eldridge [BE78]) *For any monotone, non-trivial oriented graph property P in the ternary model,*

$$\mathcal{C}(P) \geq 2v - \lfloor \log_2 v \rfloor - 2.$$

Theorem 1.7. (B. Bollobás and S.E. Eldridge [BE78]) *For any non-trivial graph property P*

$$\mathcal{C}(P) = \Omega(v).$$

Both bounds are tight (the second only up to a constant factor). For Theorem 1.6 this is shown by the property 'having a sink' [BE78]. For Theorem 1.7 this is shown by the property 'being a scorpion graph' [BBL74].

We are especially interested in classes of boolean functions where we can't save more than a constant factor compared to $n = \binom{v}{2}$. Aanderaa proposed the class of non-trivial graph properties. Rosenberg was the first one to realize the falsity of this conjecture. He added the condition of monotonicity [Ro73]. That form of the conjecture is referred to as the Aanderaa–Rosenberg conjecture.

The tightness of Theorem 1.6 and Theorem 1.7 shows that omitting the monotonicity requirement or allowing oriented graphs creates totally different situations.

The first step toward confirming the conjecture was made by D. Kirkpatrick who proved the following lower bound.

Theorem 1.8. (D. Kirkpatrick [Kir74]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}(P) = \Omega(v \log v).$$

The Aanderaa–Rosenberg conjecture was subsequently settled by Rivest and Vuillemin.

Theorem 1.9. (R.L. Rivest and S. Vuillemin [RV76]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}(P) \geq \frac{v^2}{16}.$$

Refining their method D.J. Kleitman and D.J. Kwiatkowski improved the constant of the lower bound.

Theorem 1.10. (D.J. Kleitman and D.J. Kwiatkowski [KK80]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}(P) \geq \frac{v^2}{9}.$$

In fact, no non-trivial, monotone graph property is known which would not require $n = \binom{v}{2}$ questions in the worst case. The proof method of Rivest and Vuillemin defines a class of evasive functions. Instead of graph properties they considered boolean functions with 'high symmetry'. Let us consider the group of all permutations of the variables of a given boolean function. Let us consider the subgroup of it which consists of permutations σ of the variables such that $f(\underline{x}) = f(\underline{x}^\sigma)$, where x^σ is the permuted order of the variables. If this permutation group is transitive, i.e. every variable can be carried to every other by some element of this group, than the function is called *transitive*.

Theorem 1.11. (R.L. Rivest and S. Vuillemin [RV76]) *Every boolean function f of n variables such that f is transitive, n is a prime power and $f(\underline{0}) \neq f(\underline{1})$, is evasive.*

In ligth of this theorem there is a natural question: can we get rid of the disturbing assumption that n is a prime power? The answer is no; a counterexample was given by Illies [Il78]. But nobody knows what happens if we have, instead, a monotonicity assumption.

Conjecture 1.12. *Every transitive, non-trivial and monotone boolean function is evasive.*

The special case of this conjecture for graph and digraph properties is also open.

Conjecture 1.13. *Any non-trivial, monotone graph or digraph property is evasive.*

Let us realize that Theorem 1.11 of Rivest and Vuillemine doesn't settle this question even for special values of v. $\binom{v}{2}$ won't be a prime power except for $v = 2$ or 3.

There are strong partial results for graph properties. The following results are based on a topological method of Kahn, Saks and Sturtevant [KSS84].

Theorem 1.14. (J. Kahn, M. Saks and D. Sturtevant [KSS84]) *Any non-trivial, monotone property of graphs and digraphs with a prime power number of vertices is evasive.*

Theorem 1.15. (A. Yao [Ya88]) *Any non-trivial, monotone bipartite graph property is evasive.*

V. King considered the case of directed bipartite graphs. She noted that we cannot expect evasiveness because we can express properties depending only on edges starting in one fixed color class. She obtained an evasiveness result by adding an extra condition to exclude this possibility.

Theorem 1.16. (V. King [Ki88]) *Any non-trivial, monotone, directed bipartite graph property is evasive, assuming that one color class has prime power size and of the two one-way complete graphs either neither or both have the property.*

Some of the results above give lower bounds only for special sizes of the vertex set. In order to get general lower bounds we need some kind of reductions between different sizes.

Theorem 1.17. (D.J. Kleitman and D.J. Kwiatkowski [KK80])

$$\mathcal{C}(\mathcal{GP}_v) \geq \min\{\, \mathcal{C}(\mathcal{GP}_{v-1}), q(v-q) \,\},$$

where q is the prime power nearest to $\frac{v}{2}$.

Theorem 1.18. (V. King [Ki88])

$$\mathcal{C}(\mathcal{DP}_v) \geq \min\{\, \mathcal{C}(\mathcal{DP}_{v-1}), q(v-q) \,\},$$

where q is the smallest prime power greater than $\frac{v}{2}$.

When we apply these results we lose a constant factor.

Theorem 1.19. (Kahn, Saks and Sturtevant [KSS84]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}(P) \geq \frac{v^2}{4} + o(v^2).$$

Theorem 1.20. (V. King [Ki88]) *For any non-trivial, monotone digraph property P,*

$$\mathcal{C}(P) \geq \frac{v^2}{2} + o(v^2).$$

These are the best known general lower bounds for non-trivial, monotone graph and digraph properties.

* * *

In the manner common in complexity theory one can introduce decision trees using extra power like nondeterminism, randomization, alternation (see [MT85], [MH85b], [Sn85], [Ya77])

Definition 1.21. A *nondeterministic decision tree* is a rooted tree. Each of its inner nodes is labeled by a variable. This label represents a query. Each edge leaving the node is labeled 0 or 1. The subtrees which can be reached from a given node by an edge labeled 0 are the possible continuations of the algorithm after getting answer 0. The role of the edges labeled by 1 is symmetric. During the execution of the algorithm the next step will be chosen nondeterministically.

The definition above describes the notion of a nondeterministic decision tree and its execution on an input. But this execution is nondeterministic. So what function is computed by this tree? We say that an input is accepted if there exists a computation path leading to an accepting leaf.

Definition 1.22. The *nondeterministic decision tree complexity* of a boolean function f is the minimum depth of the nondeterministic decision trees computing f. This complexity is denoted by $\mathcal{C}^{ND}(f)$.

Definition 1.23. A *randomized decision tree* is a rooted tree. Each of its inner nodes is labeled a variable, i.e. by a query. The edges leaving a node are labeled 0 or 1. The subtrees which can be reached from a given node by an edge labeled 0 are the possible continuations of the algorithm after receiving answer 0. The role of the edges labeled 1 is symmetric. During the execution of the algorithm the next step will be chosen randomly.

An alternative definition might be the following. Let us say that the random choice is based on coin tossing. If one fixes the outcome of the coin tosses than we have a deterministic computation. In this way we can describe the probabilistic decision tree as a probability distribution on the set of deterministic trees.

Again we face the question: how to define that a randomized decision tree computes a function?

There are many different ways to answer this questions. We use the simplest convention when we require that the algorithm always give the correct answer. Using the second formalization of the randomized decision tree, it computes a function f iff the distribution is non-zero only on deterministic trees computing f.

Definition 1.24. Let $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ be the set of all the deterministic decision trees computing the function f. Let $\mathcal{R} = \{p_1, \ldots, p_N\}$ be a randomized decision tree computing f, where p_i is the probability of $\mathcal{A}_i$.

The cost of $\mathcal{R}$ on input x is $\text{cost}(\mathcal{R}, x) = \sum_i p_i \text{cost}(\mathcal{A}_i, x)$.

The *randomized decision tree complexity* of a function f is

$$\mathcal{C}^R(f) = \min_{\mathcal{R}} \max_x \text{cost}(\mathcal{R}, x),$$

where the minimum is taken over all randomized decision trees computing the function f.

There are alternative definitions in which we allow errors. We obtain different models, depending on what kind of errors we allow (1-way or 2-way).

Definition 1.25. Let $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ be the set of all the deterministic decision trees (not necessarily computing a given function f). Let $\mathcal{R} = \{p_1, \ldots, p_N\}$ be a probability distribution on deterministic decision trees, where p_i is the probability of $\mathcal{A}_i$.

$\mathcal{R}$ is *λ-tolerant* for f if $\sum_{\mathcal{A}_i \text{ doesn't output } f(x) \text{ on } \underline{x}} p_i \leq \lambda$, for all possible inputs $\underline{x}$.

The cost of $\mathcal{R}$ on input x is $\text{cost}(\mathcal{R}, x) = \sum_i p_i \text{cost}(\mathcal{A}_i, x)$.

The *2-way error randomized decision tree complexity* of a function f with error λ is

$$\mathcal{C}^{R2}_\lambda(f) = \min_{\mathcal{R}} \max_x \text{cost}(\mathcal{R}, x),$$

where the minimum is taken over all λ-tolerant randomized decision trees computing the function f.

Let $\mathcal{C}^{R2}(f) = \mathcal{C}^{R2}_{\frac{1}{3}}(f)$.

The constant $\frac{1}{3}$ doesn't have an important role. If we neglect constants in the complexity than we can substitute it with anything less than $\frac{1}{2}$.

The possible algorithms can output anything. The mistake can be either way. This fact is indicated by the superscript 2. If our randomized algorithm is restricted to produce deterministic trees where the mistake occurs in only one direction (it might output 0 instead of the real value 1 but not the other way around) then it is called *1-way error computation* (the corresponding complexity measure is denoted by $\mathcal{C}^{R1}$). For further information we refer the reader to A. Yao's [Ya77] and Noam Nisan's papers [Ni].

The main question is this: how much can we save by adding the extra power of randomization? We mention some basic inequalities.

Theorem 1.26. (M. Blum [BI87]) *For any boolean function f*

$$\sqrt{\mathcal{C}(f)} \leq \mathcal{C}^{ND}(f) \leq \mathcal{C}^{R}(f) \leq \mathcal{C}(f).$$

Using the $\mathcal{C}^{R1}(f)$, resp. $\mathcal{C}^{R2}(f)$ notation for the randomized complexity of f allowing 1-way and 2-way errors, resp. (see [Ni] for details) Noam Nisan obtained the following results.

Theorem 1.27. (Noam Nisan [Ni]) *For any boolean function f*
(i) $\sqrt{\frac{1}{2}\mathcal{C}(f)} \leq \mathcal{C}^{R1}(f)$,
(ii) $\frac{1}{2}\sqrt[3]{\mathcal{C}(f)} \leq \mathcal{C}^{R2}(f)$.

These theorems give a lower bound for the power of randomization. We refer to them as the basic bounds.

On the other side there are several known examples where randomization does help.

Example 1.28. (M. Saks and A. Wigderson [SW86]) Consider the digraph property 'every vertex has an incoming arc'.

Deterministically, this is an evasive property, so its deterministic complexity is $v(v-1)$.

Let us examine the following randomized algorithm. It considers each vertex one at a time in random order and it scans the possible incoming edges into that vertex until it finds one or realizes that there aren't any. It is easy to see that the complexity of this algorithm is at most $\frac{v(v+1)}{2}$. So randomization can save a constant factor.

Example 1.29. (M. Saks and A. Wigderson [SW86]) Let f be the following boolean function on $n = 2^d$ variables. First let us build a binary tree based on these variables as leaves. Plug a NAND gate into each inner node. The circuit that we get in this way will compute f.

It is not hard to see that the deterministic complexity of this function is n (theorem 1.11).

The following randomized algorithm gives an upper bound on the randomized complexity of f. Choose a child of the root at random and evaluate its subtree recursively. If it evaluates to 0, then the value of f is 1. Otherwise recursively evaluate the other child of the root.

The complexity of this algorithm is $\Theta(n^{.753\cdots})$. As it turns out this is exactly the randomized complexity of f. For more details see [SW86].

R. Boppana exhibited another example of a function where randomized and deterministic complexities differ in the exponent [Bo].

It is conjectured that the examples above are the best possible up to a constant factor.

Conjecture 1.30. (attributed to R.M. Karp by [SW86]) *For any non-trivial, monotone graph property P*

$$\mathcal{C}^R(P) = \Omega(\mathcal{C}(P)) = \Omega(v^2).$$

Conjecture 1.31. (M. Saks and A. Wigderson [SW86]) *For any boolean function f*

$$\mathcal{C}^R(f) = \Omega(\mathcal{C}(f)^{0.753\ldots}).$$

Only in the case of graph properties are there results better than the basic inequalities known (theorem 1.26). (In this case we know that the deterministic complexity is of the order of v^2. Blum's bound shows that the randomized complexity of any graph property is at least linear in v.)

Theorem 1.32. (A. Yao [Ya87]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}^R(P) = \Omega(v \log^{\frac{1}{12}} v).$$

Theorem 1.33. (V. King [Ki88]) *For any non-trivial, monotone graph property P,*

$$\mathcal{C}^R(P) = \Omega(v^{\frac{5}{4}}).$$

These results can be extended to the 2-way error version of randomized computation. (See [Ya87] and [Ki88].)

The main result of this chapter is our improved lower bound for this problem.

Theorem 1.34. *For any non-trivial, monotone graph property P,*

$$\mathcal{C}^R(P) = \Omega(v^{\frac{4}{3}}).$$

Our method can be carried out for the 2-way error model.

Theorem 1.35. *For any non-trivial, monotone graph property P,*

$$\mathcal{C}^{R2}(P) = \Omega(v^{\frac{4}{3}}).$$

* * *

In the following few paragraphs we summarize the different modifications of decision trees and other related models.

Gy. Turán proposed the following generalized decision tree model for deciding graph properties. The model is the same kind of binary tree except that the queries are more general. Each inner node is labeled by a subset of the vertex set and the query asks whether or not there exists at least one edge induced by these vertices. The cost, again, is the number of queries in the worst case.

A. Hajnal, W. Maass and Gy. Turán [HMT88] determined the complexity of a few graph properties. In particular, they proved that the complexity of connectivity and bipartiteness in this model is $\Theta(n \log n)$.

They have results on the oblivious versions of their decision trees.

There are many graph properties with unknown Turán-complexity, including 'having perfect matching' and 'having Hamiltonian cycle'. Nothing is known about the randomized version of their model.

Going back to the computation of boolean functions one might ask the question (following the idea of [HMT88]): what happens if in the tree at a given inner vertex we evaluate functions other than the simplest, one-variable ones?

I don't know any result on this direction. Some natural questions: In the Turán-model our queries are a special kind of disjunctions of variables. What happens if we allow all the disjunctions of variables (or negated variables) as questions? What happens if we allow disjunctions of clauses each of which consist of a conjunction of two variables (or negated variables) as queries.

Other questions arise if we enlarge the domain. So far we have been working with boolean functions, where the variables have $(0, 1)$ values.

If our variables have values from a linearly ordered set we get the well known comparison tree model. Of course changing the domain implies that we have a new type of problem. The most traditional ones are: sorting, selection, and merging.

There are extensions where the domain is a partially ordered set. We mention a few papers discussing these questions: [FT88a],[FT88b],[Pr87],[Sa].

We would like to point out one research project that was mentioned in [FT88a]. Let us consider all possible partial orderings of $\{1, \ldots, v\}$. We can define isomorphism of partially ordered sets. We are interested in deciding properties of partially ordered sets (preserved under isomorphism). The model is a ternary tree where at every node we ask a question about two elements and the answer is their relation (i.e. $<$,$>$ or incomparable). Given a property of partially ordered sets how many queries do we need?

There are some known easy properties, where $o(v^2)$ questions suffice to decide. Such problems include 'having a unique maximal element', 'being a linear order' and 'having bounded width'. Other properties, like 'connectivity', 'bounded height', 'being a lattice', 'being an interval order' are known to be hard in the sense that they require $\Omega(v^2)$ questions. What properties are evasive (i.e., require $\binom{v}{2}$ questions in worst case) ?

If our domain is an algebraic structure like the real numbers, then there aer a lot of possibilities to extend our model. We can query if a linear function of the variables is positive, negative or zero [Gy81],[DM]. The output (the content of the leaves) can be a number instead of a boolean value. Again the new domain puts forward several new problems: knapsack, shortest path, element distinctness, convex hull.

We can further extend the set of possible queries. A. Yao considers quadratic decision trees [Ya81]. In this model at every node we can compute a quadratic function of the

expressions computed already. A query is any expression computed already and the answer is the comparison of its value and 0.

As an extension of A. Yao's model we can allow computing bounded degree polynomials at a given node [BO83], [KS86].

S. Smale allows computing arbitrary rational expressions at a node. He gives [Sm87] a lower bound on the total number of nodes (rather than the depth) on the problem of simultaneous approximation of the roots of the input polynomial.

* * *

Decision trees have relations with other branches of computer science, in particular with PRAM machines.

We can extend our decision model by allowing integer outputs instead of boolean ones. This variation of our model can compute functions $f : \{0,1\}^n \to \mathbf{N}$, like PRAM machines. A PRAM machine is collection of RAM machines (see [AHU74]) communicating through a common memory. There are several versions of this model depending on how we resolve read and write conflicts. If we allow simultaneous read but we don't allow different computers to write into the same cell at the same time then we have the CREW PRAM model. The following theorem shows that decision tree complexity and CREW PRAM complexity are closely related.

Theorem 1.36. (Noam Nisan [Ni]) *(i) Any function that can be computed by a CREW PRAM in time t can be computed by a boolean decision tree of depth $\mathcal{O}(2^t)$.*
(ii) Any function that can be computed by a boolean decision tree of depth d can be computed by a non-uniform CREW PRAM in $\mathcal{O}(\log d)$ steps using $\mathcal{O}(2^d)$ processors.

The connection between another version (namely CROW PRAM's) of PRAM's and decision trees are also discussed in [Ni].

S. Cook, C. Dwork and R. Reischuk [CDR86] pointed out the connection between CREW PRAM's and the dependency of boolean function. $D \subset \{1,\ldots,n\}$ is a *dependency set* for the function $f : \{0,1\}^n \to \mathbf{N}$ on input $x \in \{0,1\}^n$ if $f(x) = f(x')$ for all x' that agree with x on D. The dependency of f on input x is the minimum size of any dependency set for f on x. The *dependency of f* is

$$\max_{x \in \{0,1\}^n} \text{dependency of } f \text{ on input } x.$$

Now it is not very surprising that dependency of a function is related to the decision tree complexity.

Theorem 1.37. (M. Blum) *Any function $f : \{0,1\} \to \mathbf{N}$ can be computed by a boolean decision tree of depth at most $(\text{dependency of } f)^2$.*

Another parameter, block sensitivity, and its connection to decision tree complexity is discussed in [Ni].

Evaluation of game trees is an important problem in artificial intelligence. [SW86] uses the technique developed for analyzing randomized decision trees for (α, β) pruning procedure (see [KM75],[Pe80],[Pe82],[Roi81],[T]).

* * *

The organization of this chapter is as follows.

In the next section we give a detailed description of the previously used methods for lower bounds on randomized decision tree complexity for non-trivial, monotone graph properties.

In the third section we discuss the very important notion of duality. This notion brings forward a purely graph theoretical question called the packing problem. We give a short overview on the known graph theoretical results on this subject.

In section 4 we show how one can use Yao's technique to get non-linear lower bounds. In this section we obtain an $\Omega(v^{\frac{5}{4}})$ lower bound for the bipartite graph properties.

In section 5 we improve the known packing theorems for bipartite graphs. The improved theorem together with the previous proof scheme gives us the $\Omega(v^{\frac{4}{3}})$ lower bound for the bipartite case.

None of the known reductions is able to convert this result for the general case. In section 6 we exhibit an improved reduction which finally gives the main result of the chapter.

Finally, in section 7, we say a few words about the case of 2-way errors: our result also hold in this model.

2. Previous techniques

In the previous section we described several graph properties using well-known notions of graph theory. If we want to say something about all graph properties then it is better to give a uniform way to describe a property. In the case of monotone graph properties there is a convenient way to do that.

Definition 2.1. Let P be any monotone graph property. $min(P)$ denotes the list of *minimal graphs* having property P, i.e. $G \in min(P)$ iff $G \in P$ but for any proper subgraph H of G $\quad H \notin P$.

It is clear that knowing $min(P)$ is equivalent to knowing P.

In previous papers several methods were presented giving lower bounds for the random complexity of properties. The lower bounds on $\mathcal{C}^R(P)$ given by these methods depended explicitly on $min(P)$.

All the known lower bound methods based on a lemma of A. Yao. This lemma transforms the lower bound problem on the randomized decision tree complexity into another problem, where giving lower bounds is much more convenient.

Lemma 2.2. (A. Yao [Ya77]) *Let d be a probability distribution on all the possible inputs and let $d(\underline{x})$ be the probability of input $\underline{x}$. (In the case of graph properties d describes a random graph.) We define the average case performance of a deterministic decision tree $\mathcal{A}$ computing f as $av(\mathcal{A}, d) = \sum_{\underline{x}} d(\underline{x}) cost(\mathcal{A}, \underline{x})$.*

Then for a boolean function f

$$\mathcal{C}^R(f) = \max_d \min_{\mathcal{A}} av(\mathcal{A}, d),$$

where the minimum is taken over all the deterministic decision trees computing f.

$\mathcal{C}^R(P)$ is defined as a minimum. The statement of the lemma is that $\mathcal{C}^R(P)$ is can be written as a maximum. So lemma 2.2 is a so-called minimax theorem. In fact this is basically equivalent to the famous minimax theorem of J. von Neumann [vN28] for game theory.

The meaning of this lemma is that if one wants to give a lower bound for $\mathcal{C}^R(P)$ then there is an easy way to do it. Namely find a distribution which is concentrated on the 'uniformly hard' instances and for every deterministic algorithm find a lower bound on the average cost over this distribution. The bound will be a lower bound on $\mathcal{C}^R(P)$ too.

Now we give a list of the significant results on lower bound techniques for randomized decision trees. We shall give a proof of Yao's method since the result is very important for our proof.

Remember that $\overline{d}(G)$ and $D(G)$ were defined in section I.4 as the average and maximum degree of G, respectively. If the average or maximum is taken over nodes coming from a subset S of the vertex set then we indicate this by a subscript S.

Theorem 2.3. (A. Yao [Ya77]) *(i) Let $P \in \mathcal{GP}_v$ and $G \in min(P)$ be any minimal graph for P. Then*

$$\mathcal{C}^R(P) = \Omega(v\bar{d}(G)).$$

(ii) Let $P \in \mathcal{BP}_{u,w}$ and $G \in min(P)$ be any minimal graph for P. Then

$$\mathcal{C}^R(P) = \Omega(u\overline{d}_U(G)).$$

Definition 2.4. Let $\mathcal{L}$ be a list of graphs from $\mathcal{B}_{u,w}$. For each $G \in \mathcal{L}$ let us consider the sequence of degrees in color class U. Let $\langle d_1, d_2, \ldots, d_u \rangle$ be the sequence of degrees in non-incresing order. If $\langle d_1, d_2, \ldots, d_u \rangle$ is the lexicographically first sequence among the ordered lists from elements of $\mathcal{L}$, then we refer to G as the *U-lexicographically first element* of $\mathcal{L}$.

Theorem 2.5. (A. Yao [Ya87]) *Let $P \in \mathcal{BP}_{u,u}$ and $G \in min(P)$ be the U-lexicographically first graph of $min(P)$. Then*

$$\mathcal{C}^R(P) = \Omega(\frac{D_U(G)}{\overline{d}_U(G)}u).$$

Proof. (Sketch) Without loss of generality we can assume that $D_U(G) > 100\overline{d}_U(G)$ since otherwise theorem 1.26 would immediately give the result. The factor 100 is not crucial, it makes things simpler so we can get a better insight into the proof.

In the first half of the proof we construct a distribution on $\mathcal{B}_{u,u}$.

The construction starts from the U-lexicographically first element G. Let $x_0 \in U$ the node of maximal degree in U. Let $x_1, x_2, \ldots, x_{\frac{u}{4}}$ be the vertices of U with smallest degrees. Let $D = D_U(G) = d(x_0)$, $d_i = d(x_i)$, for $1 \leq i \leq \frac{u}{4}$ and $\overline{d} = \overline{d}_U(G)$. It is easy to see that $d_i \leq 2\overline{d} \ll D$, for all $1 \leq i \leq \frac{u}{4}$. Let N_i be the neighborhood of x_i. We refer to N_0 as the big neighborhood and to N_i ($1 \leq i \leq \frac{u}{4}$) as small neighborhoods.

Let us define G' the following way. We add some new edge to G, namely we connect x_i to all elements of N_0 and N_{i+1} (in the case $i = \frac{u}{4}$ we connect $x_{\frac{u}{4}}$ only to N_0). If we had already an edges between two nodes which we are supposed to join then we do not do anything.

G' has property P since it has G as a subgraph and G is a minimal graph for P. Recognizing G as a subgraph was very easy. An important property of G' is that it has many other subgraph isomorphic to G. To see this, consider the permutation σ_i of the vertex set of G': $(x_0, x_i, x_{i-1}, \ldots, x_1)$, i.e. a cyclic permutation on $\{ x_0, x_i, \ldots, x_i \}$ and all the other vertices are fixed. Then G^{σ_i} is a subgraph of G' for all $1 \leq i \leq \frac{u}{4}$. ($G^{\sigma_i}$ is a twisted copy of G.)

As an easy consequence of this property consider the following truncation of G'. Delete $4\overline{d}$ edges starting at x_i in parallel, for $0 \leq i \leq \frac{u}{4}$. We have $N_{G'}(x_i) = N_0 \cup N_i \cup N_{i+1}$, so $N_{G'}(x_i)$ is a union of two small sets and a big one. The only restriction on which edges to delete is that we can't delete edges connecting x_i to the small neighborhoods. The set of deleted edges starting at a given x_i will be called a *wedge*. The deletion procedure is not yet well-defined, we have many ways to execute it. Let H be any graph which can be constructed as we described. The remark above shows that by putting back only one wedge into H we get a graph that has property P. (If we put back the wedge of x_i then G^{σ_i} proves this fact.)

On the other hand in H the degrees of all the x_i's are smaller than D. The fact that G was the lexicographically first minimal graph shows that H can't have any of the minimal graphs of P as a subgraph. So H doesn't have property P.

If we input H into any deterministic decision tree $\mathcal{A}$ computing P, then it must test at least one edge from each wedge. So H is a 'hard' input. Let d is the distribution on $\mathcal{B}_{u,u}$ that is defined by the random graph space where the deletion procedure described above is a random one (i.e. we have the uniform distribution over all possible H).

In the second half of the proof we show that every deterministic algorithm solving P requires many questions in average, if the input has distribution d. Using lemma 2.2 we shall obtain the statement of the theorem.

Actually we already have all the remarks that we need for the proof. Let us concentrate on the neighborhood of one x_i in input H (let us say that we fix the outside part of the graph). This neighborhood is $N_0 \cup N_i \cup N_{i+1} - R$ where R is defined by the deleted edges. If we consider all possible inputs with the distribution d then R is a random subset of $N_0 - (N_i \cup N_{i+1})$. On the other hand the deterministic algorithm starts to scan the pair of vertices in a deterministic order. The question is: when does this order hit first R? The

size of $N_0 - (N_i \cup N_{i+1})$ is about D, R is of size $4\overline{d}$. It is easy to compute that the expected number of questions asked until we hit the random set R is $\Theta(\frac{D}{\overline{d}})$. Knowing this for $\frac{u}{4}$ nodes we easily get the result. ∎

Theorem 2.5 is very powerful. The only problem with it is that we can apply it only for a very specific graph of the list $min(P)$. We need a slight extension of this method. In our extension we shall have some structural knowledge about the minimal graph which is pointed out by the lemma.

Lemma 2.6. *Let $P \in \mathcal{BP}_{u,u}$ and let us assume that there is a graph in $min(P)$ which has at least $\frac{u}{2}$ isolated nodes in U. Let G be the U-lexicographically first graph among the graphs having at least $\frac{u}{2}$ isolated nodes in U. Then*

$$\mathcal{C}^R(P) = \Omega\big(\frac{D_U(G)}{\overline{d}_U(G)}u\big).$$

Proof. The proof is a simple modification of the proof of theorem 2.5. In that proof we chose the node of maximal degree from U and some other points of degree at most twice the average degree. Now we do the same but we carefully leave $\frac{u}{2}$ isolated nodes out of the consideration. (This way we might loose a constant factor, but it will disappear within Ω.)

The only point where we used that the graph G of theorem 2.5 is the lexicographically first is when we showed that that none of the elements of our random graph space has property P. We concluded this from the fact that these graphs do not have any subgraph from $min(P)$.

This is still true for the random graph space constructed from the new graph defined in our lemma. For any element of $min(P)$ having $\frac{u}{2}$ isolated nodes this follows from the fact that G was the U-lexicographically first among these kind of graphs. For the other graphs it is true because they have less than $\frac{u}{2}$ isolated nodes in U. ∎

V. King gave the following lower bound method.

Theorem 2.7. (V. King [Ki88]) *(i) Let $P \in \mathcal{GP}_v$ and $G \in min(P)$ be any minimal graph for P. Then*

$$\mathcal{C}^R(P) = \Omega\big((\frac{|V(G')|}{\overline{d}(G')D(G)})^2\big),$$

where G' is the subgraph of G induced by the non-isolated nodes.
(ii) Let $P \in \mathcal{BP}_{u,w}$ and $G \in min(P)$ be any minimal graph for P. Then

$$\mathcal{C}^R(P) = \Omega\big((\frac{|U'|}{\overline{d}_{U'}(G)D_U(G)})^2\big),$$

where U' is the subset of U of nodes with positive degree.

We note that in the previous claims about bipartite universe the roles of U and W are exchangeable.

None of these methods works for every graph property, but for any property one of them gives a good lower bound. Combining all of them one get a superlinear lower bound for bipartite graphs. Unfortunately Yao's method doesn't seem to work on general graph properties. This is the reason that the known lower bounds handle the bipartite properties first.

Theorem 2.8. (V. King [Ki88]) *The randomized decision tree complexity of any non-trivial, monotone bipartite graph property $P \in \mathcal{BP}_{v,v}$ is $\Omega(v^{\frac{5}{4}})$, i.e.,*

$$\mathcal{C}^R(\mathcal{BP}_{v,v}) = \Omega(v^{\frac{5}{4}}).$$

* * *

In order to get a lower bound for general graph properties we need a reduction from general graphs to bipartite graphs. The two strongest known reductions are due to V. King.

Theorem 2.9. (V. King [Ki88]) *For any q such that $1 \leq q \leq \frac{v}{2}$ and for any non-trivial, monotone graph property $P \in \mathcal{P}_v$*

$$\mathcal{C}^R(P) = \Omega\big(\min\Big\{\frac{v^2}{q}, \min_{q \leq r \leq \frac{v}{2}} \mathcal{C}^R(\mathcal{BP}_{v-r,r})\Big\}\big).$$

Theorem 2.10. (V. King [Ki88]) *The randomized decision tree complexity of any non-trivial, monotone graph property $P \in \mathcal{P}_v$ is*

$$\mathcal{C}^R(P) = \Omega\big(\min\Big\{v^{\frac{6}{5}}, \mathcal{C}^R(\mathcal{BP}_{\frac{v}{2},\frac{v}{2}})\Big\}\big).$$

3. Using duality: packing

A. Yao's original paper [Ya87] had to treat specially the case when all the elements of $min(P)$ have small number of edges. V. King had problems with graphs with many isolated nodes (because of lemma 2.7). In both cases the problem was solved using the notion of duality.

Definition 3.1. $G \in P^*$ iff $\bar{G} \notin P$. P^* is the *dual* of the property P.

It is easy to check that if P is non-trivial, monotone graph property then so is P^*.

Let $\bar{P}$ be the complement property of P, i.e. the property such that $G \in \overline{P}$ iff $\overline{G} \in P$. If we have a deterministic decision tree $\mathcal{A}$ for P then let $\overline{\mathcal{A}}$ be the decision tree where at each inner node the labels of the two edges leaving that node are exchanged. It is clear that $\overline{\mathcal{A}}$ computes $\overline{P}$. If in addition we complement the label of each leaf then we get a decision tree $\mathcal{A}^*$ computing P^*. This argument shows that there is an 'nice' $1-1$ map

between decision trees computing P and P^*. In particular we get that the deterministic and randomized decision complexities of P and P^* are the same.

In the proof we shall utilize our ability to choose between the properties P and P^*, each having the same complexity. In this section we investigate the dependence between $min(P)$ and $min(P^*)$. The dependence will be very useful for us because it gives guidance in choosing the right list to work with.

Definition 3.2. (a) Let $G, H \in \mathcal{G}_v$. Let us assume that G has vertex set V and H has the vertex set V'. A *packing* is a bijection $f : V' \to V$ such that the edge set of G and H^f are disjoint.
(b) Let $G, H \in \mathcal{B}_{u,w}$. Let us assume that G has color classes U and W and H has color classes U' and W'. A *bipartite packing* is a bijection f that maps U' to U and W' to W such that the edge set of G and H^f are disjoint.

The following lemma shows the importance of packing.

Lemma 3.3. ([Ya2]) *(i) If $P \in \mathcal{P}_v$, $G \in min(P)$ and $H \in min(P^*)$ then G and H cannot be packed.*
(ii) If $P \in \mathcal{BP}_{u,w}$, $G \in min(P)$ and $H \in min(P^*)$ then G and H cannot be packed as a bipartite graphs.

Packing graphs is a heavily studied subject in graph theory. A good survey of this research can be found in [Bo78]. Bellow, we summarize the known results on packing.

Much effort has been spent for packing sparse graphs [SS78], [TY87], [BS77], [BS78], [HHS81], [STY85], [FRSS81]. An typical theorem from this area is:

Theorem 3.4. (N. Sauer and J. Spencer [SS78]) *If $|E(G)|, |E(H)| \leq v-2$ (where $|V(G)| = |V(H)| = v$) then G and H can be packed.*

One can extend packing to packing several graphs. We just refer the reader to [GyL]. We mention a nice conjecture from this paper.

Conjecture 3.5. (A. Gyárfás and J. Lehel [GyL]) *Let T_k be any tree with vertex set of size k ($k = 1, \ldots, n$). Then there is a packing of $T_1, T_2, \ldots, T_n$ into the complete graph on n vertices.*

The following few theorems give sufficient conditions on the number of edges for the existence of a packing.

Theorem 3.6. (B. Bollobás and S.E. Eldridge [BE78]) *If $|E(G)| + |E(H)| \leq \lfloor \frac{3}{2}(v-1) \rfloor$ (where $|V(G)| = |V(H)| = v$) then there is a packing of G and H.*

For improvements (but still with a linear upper bound in the condition on the sum of the number of edges) see [BE78].

Theorem 3.7. (B. Bollobás and S.E. Eldridge [BE78])
(i) If $G, H \in \mathcal{G}_v$ and $|E(G)||E(H)| < \binom{v}{2}$ then G and H can be packed.

(ii) If $G, H \in \mathcal{B}_{u,w}$ and $|E(G)||E(H)| < uw$, then G and H can be packed as bipartite graphs.

Theorem 3.8. (B. Bollobás and S.E. Eldridge [BE78]) *(i) If $G, H \in \mathcal{G}_v$, $|E(H)| < \frac{v}{3}$ and $|E(G)| < \frac{1}{15}v^{\frac{3}{2}}$ then G and H can be packed.*
(ii)If $G, H \in \mathcal{B}_{u,u}$, $|E(H)| < \frac{u}{3}$ and $|E(G)| < \frac{1}{15}u^{\frac{3}{2}}$ then G and H can be packed as bipartite graphs.

For us the most important sufficient conditions will be the following ones on the maximal degrees.

Theorem 3.9. (Conditions on the maximal degree [SS78],[Ca74]) *(i) If $G, H \in \mathcal{G}_v$ and $D(G)D(H) < \frac{v}{2}$ then G and H can be packed.*
(ii) If $A, B \in \mathcal{B}_{u,w}$ and $D_U(A)D_W(B) + D_W(A)D_U(B) < u$, then A and B can be packed as bipartite graphs.

In the last two statements the bounds in the conditions are tight (up to negligible factors). For theorem 3.9.(ii) (the bipartite case) this can be easily shown using the probabilistic method in [SS78]. For theorem 3.9.(i) there is an easy construction [BE78] showing that one cannot improve the condition with more than a factor 2. This example suggests the following conjecture.

Conjecture 3.10. (B. Bollobás and S.E. Eldridge [BE78]) *Let G and H be two graph on a vertex set of size v. If $(D(G)+1)(D(H)+1) \leq v+1$ then there is a packing of G and H.*

Our proof will heavily use the conditions of theorem 3.9. So any strengthening of the results might be valuable for us. Unfortunately, even if conjecture 3.10 were to be proven, as we will see the constant improvement would disappear in the Ω. Instead, we extend these results, proving the following improved packing theorem for bipartite graphs, which is very helpful.

Theorem 3.11. *Let $G, H \in \mathcal{G}_{u,w}$. Assume that*
(a) $u \leq w \leq 2u$,
(b) $\overline{d}_{U'}(G)D_W(H) \leq \frac{u}{100}$,
(c) $\overline{d}_U(H)D_{W'}(G) \leq \frac{u}{100}$,
(d) $D_U(G), D_{U'}(H) \leq \frac{u}{1000 \log u}$.
Then G and H can be packed.

We will prove this theorem in section 5.

The following remark is an example of the power of these theorems.

If our goal is to prove a lower bound not better than $v^{\frac{3}{2}}$ than we can assume that $min(P)$ contains only graphs with at least $\frac{v}{3}$ edges. This follows from theorem 3.8 and theorem 2.3.

¿From the packing property one can get the following useful information for bipartite property $P \in \mathcal{BP}_{u,w}$: at least one of $min(P)$ and $min(P^*)$ contains a graph where the class U has at least $\frac{u}{2}$ nodes of positive degree.

The packing property does not give all the dependences between the two minimal lists. Actually $min(P^*)$ is the set of minimal graphs which can't be packed with any element of $min(P)$. The maximality gives us further information about the lists.

Lemma 3.6. *Let $P \in \mathcal{P}_{u,w}$. Then $min(P)$ or $min(P^*)$ has a graph G such that it has at least $\frac{u}{2}$ isolated nodes in U.*

Proof. $G = K_{\frac{u}{2},w} \dot{\cup} E_{\frac{u}{2}} \in \mathcal{B}_{u,w}$ is a graph with $\frac{u}{2}$ isolated nodes in U and with all the possible edges among the other nodes. If $G \in P$ then the statement is clearly true. Otherwise $\bar{G} = G \in P^*$ and again the statement is clearly true. ∎

4. Surgery on the maximal degree

For simplicity, in this section we restrict ourselves to the bipartite universe where the two color classes have the same cardinality. (Only this case is needed for the proof of theorem 1.33.) In this case $\overline{d}(G) = \overline{d}_U(G) = \overline{d}_W(G)$.

The basic idea of this section is the following. Let us fix a bipartite property P. Let us consider $min(P)$. If we have a graph of high average degree in $min(P)$ then we get a good lower bound by theorem 2.3. If theorem 2.5 points out a graph from $min(P)$ such that its average degree is low and its maximal degree in the corresponding color class is high then Yao's technique gives us a good bound. In some sense we can interpret these statements as follows: if we cannot get a good lower bound on $\mathcal{C}(P)$ by these two techniques then we have an upper bound on the maximal degree in one of the color classes of a special graph from $min(P)$. Let us assume that this happens to both $min(P)$ and $min(P^*)$. Now we are left with two graphs. We have bounds on the maximal degree in some color classes. That gives us the possibility to force a contradiction using packing theorems. The first step in this program is to choose graphs to which we can apply Yao's method.

Definition 4.1. Let $P \in \mathcal{P}_{u,u}$ and let, as before, U and W be the two color classes in this universe. Assume that $min(P)$ has some graphs with at least $\frac{u}{2}$ isolated nodes in U.
(a) Let G be the U-lexicographically first among the graphs from $min(P)$ having at least $\frac{u}{2}$ isolated nodes in U.
(b) Let H be the W'-lexicographically first element of $min(P^*)$. To avoid confusion, we

shall think of the vertex set of H as being disjoint from $U \cup W$ and denote the color classes of H by U' and W'.

We would like to use G and H in the strategy described above. We have an upper bound on the maximal degrees in their color classes. In order to apply Catlin's packing theorem (theorem 3.9.(ii)), we need bounds on the maximal degree in all other color classes. We proceed as follows. We start to build a packing between G and H. This partial packing leaves us with some leftover, unpacked nodes. These nodes define a new packing problem which, in some sense, is independent from the original one. At the same time we will have a more complete knowledge about the maximal degrees in the color classes. Eventually this will yield the desired contradiction.

Definition 4.2. (Prepacking) We are going to define sets $U_0 \subset U$, $W_0 \subset W$, $U_0' \subset U'$ and $W_0' \subset W'$ ($|U_0| = |U_0'|, |W_0| = |W_0'|$) and a packing between the corresponding induced subgraphs of G and H.
(a) Let U_0 be the set of $\frac{u}{2}$ isolated nodes in G.
(b) Let W_0' be the set of $\min\left\{ \frac{u}{8\overline{d}(H)}, \frac{u}{2} \right\}$ nodes of lowest degree in H.
(c) Let U_0' be the neighborhood of W_0' and plus as many of the highest degree nodes from U' as needed in order to get a set of size $\frac{u}{2}$. (We will see that the size of the neighborhood of W_0' is at most $\frac{u}{4}$.)
(d) Let W_0 be the set of $\min\left\{ \frac{u}{8\overline{d}(H)}, \frac{u}{2} \right\}$ nodes in W of highest degree in G.
Let G_0 and H_0 be the subgraphs of G and H induced by the sets defined above. Note that G_0 is empty. Let U_1, W_1, U_1' and W_1' the leftover parts of the corresponding vertex sets. Let G_1 and H_1 be the subgraphs induced by these leftover sets.
Any pair of bijections $U_0' \to U_0$ and $W_0' \to W_0$ gives a packing of G_0 and H_0. Choose arbitrarily one of these, and call it prepacking.

Lemma 4.3. *(i) If G_1 and H_1 can be packed then the packing and the prepacking together yield a total packing.*
(ii) $D_{U_1}(G_1) \leq D_V(G)$.
(iii) $D_{W_1'}(H_1) \leq D_{W'}(H)$.
(iv) $D_{U_1'}(H_1) \leq 4\overline{d}(H)$.
(v) $D_{W_1}(G_1) \leq 8\overline{d}(G)\overline{d}(H)$.

Proof (i) There are no edges in G between U_0 and W_1 and in H between W_0' and U_1'. Thus we won't have any conflict putting together the two packings.

(ii) and (iii) are obvious because G_1 is a subgraph of G and H_1 is a subgraph of H.

(iv) $U_1' = U' - U_0'$. We are going to show that the neighborhood of W_0' has at most $\frac{u}{4}$ nodes. This implies that U_0' has the $\frac{u}{4}$ highest degrees in U'. The claim about the size of the neighborhood of W_0' is clear because all degrees in W_0' are not greater then $2\overline{d}(H)$.

(v) If the statement were not true then the contribution of the edges having an endpoint in W_0 to the total number of edges in G would be greater than $|W|\overline{d}(G)$. ∎

Theorem 4.4. *The randomized decision tree complexity of any non-trivial monotone bipartite graph property $P \in \mathcal{BP}_{u,u}$ is $\Omega(u^{\frac{5}{4}})$, i.e.,*

$$\mathcal{C}^R(\mathcal{BP}_{u,u}) = \Omega(u^{\frac{5}{4}}).$$

Proof Let us fix an arbitrary $P \in \mathcal{BP}_{u,u}$. Let $G \in min(P)$ and $H \in min(P^*)$ be the two graphs defined in definition 4.1.

Case 1. $\overline{d}(G)$ or $\overline{d}(H)$ is at least $\frac{1}{10}u^{\frac{1}{4}}$.

In this case theorem 2.3 gives the lower bound.

Case 2. $D_U(G)$ or $D_{W'}(H)$ is at least $\frac{1}{10}u^{\frac{1}{2}}$ and case 1 does not hold.

Without loss of generality we can assume that $D_U(G)$ is at least $\frac{1}{10}u^{\frac{1}{4}}$. Because of the choice of G we can apply Yao's method and we get the lower bound

$$\Omega(\frac{D_U(G)}{\overline{d}(G)}u).$$

We know that $\overline{d}(G)$ is at most $\frac{1}{10}u^{\frac{1}{4}}$. Thus we get the desired lower bound.

Case 3. None of the previous cases holds.

Let G_1 and H_1 be the graphs defined in definition 4.2. It is easy to check that the condition of theorem 3.9.(ii) is satisfied. So G_1 and H_1 can be packed. Using lemma 4.3.(i) we get that G and H can also be packed, which is a contradiction. ∎

5. The improved packing theorem for bipartite graphs

In the previous section we used Catlin's simple packing theorem for bipartite graphs. In this section we improve this result and derive an improved lower bound on our problem, too.

First, we review Catlin's idea. Given $G, H \in \mathcal{G}_{u,w}$ with color classes U, W and U', W', resp. We want to find a sufficient condition for existence a packing. We take an arbitrary bijection $f : U' \to U$. Define a bipartite graph between W and W' based on whether two nodes can be identified or not. Now the problem is simply finding a matching in this auxiliary graph.

Definition 5.1. Let $G, H \in \mathcal{G}_{u,w}$. Let U, W, U' and W' be the corresponding color classes. Given f, a bijection $U' \to U$, we define a bipartite graph B_f with color classes W and W'. We make $x \in W$ and $y \in W'$ adjacent iff x and y can be identified, i.e., the neighborhoods of x in G and of y in H^f are disjoint subsets of V.

Now it is easy to show that if G and H satisfy the condition of Theorem 3.9.(ii) then for any bijection f B_f satisfies the condition of König's theorem (see e.g. [Lo79], Chap. 7, prob. 4.) and therefore possesses a perfect matching along which we can map W' to W to obtain a packing. It is worth to state this fact as a separate lemma. Remember that $\delta(G)$ is the minimal degree of G and $\delta_S(G)$ is the minimal degree among nodes from S (see section I.4.).

Lemma 5.2. *(i) Let $G \in \mathcal{B}_{u,u}$. If $\delta_U(G), \delta_W(G) \geq \frac{u}{2}$ then G has a perfect matching.*
(ii) Let $G \in \mathcal{B}_{u,u}$. If $\delta_U(G) + \delta_W(G) \geq u$ then G has a perfect matching.

The condition in theorem 3.9.(ii) restricts the product of the maximum degrees of G and H. Our improvement comes from relaxing one of the terms to average degree.

Theorem 5.3. *Let $G, H \in \mathcal{G}_{u,w}$. Assume that*
(a) $u \leq w \leq 2u$,
(b) $\overline{d}_{U'}(G) D_W(H) \leq \frac{u}{100}$,
(c) $\overline{d}_U(H) D_{W'}(G) \leq \frac{u}{100}$,
(c) $D_U(G), D_{U'}(H) \leq \frac{u}{1000 \log u}$.
Then G and H can be packed.

The proof of this result is probabilistic. The probabilistic method for proving existence of a combinatorial object was introduced by P. Erdős [Er47]. Since then it has been a very useful tool in combinatorics [ES74], [Sp87], [Bo85], etc. The basic idea of this method that if a property has positive probability then at least one element of the probabilistic space must have that property. In our case the goal is to show that there exists a bijection $f : U \to U'$ such that B_f has a perfect matching. We are going to show that for a random bijection this is true.

Let $\{ d_1, \ldots, d_u \}$ be all the degrees in U, and let $\{ e_1, \ldots, e_u \}$ be all the degrees in U'.

Lemma 5.4. *Let $f : U \to U'$ be a random bijection, all bijections being equally likely.*

$$Prob(B_f \text{ has perfect matching}) \geq 1 - w \; Prob(\sum_{i \in R} d_i \geq \frac{w}{2}) - w \; Prob(\sum_{i \in S} e_i \geq \frac{w}{2}),$$

where S is a random subset of U' of size $D_W(G)$, all such subsets being equally likely, and R is a random subset of U of size $D_{W'}(G)$, all such subsets being equally likely.

Proof. We are interested in the event

$$E = B_f \text{ has a perfect matching .}$$

By lemma 5.2 the following event is a subset of E.

$$F = \text{ Each node of } B_f \text{ has degree at least } \frac{w}{2}.$$

One elementary bad event is

$$F_x = x \text{ has degree in } B_f \text{ less then } \frac{w}{2} \text{ (for } x \in W \cup W').$$

Using this notation

$$E \supseteq F = \Omega - \cup_{x \in W \cup W'} F_x.$$

Thus

$$Prob(E) \geq 1 - \sum_{x \in W \cup W'} Prob(F_x).$$

For $x \in W$, F_x is exactly the event that the image $f(N(x))$ of $N(x)$ ($f(N(x)) \subset U'$) has a neighborhood in W' of size more than $\frac{w}{2}$. The event that the sum of the degrees in $f(N(x))$ is at least $\frac{w}{2}$ is a superset of F_x. If $x \in W$ then $f(N(x))$ is a random set of size $|N(x)|$ and its size is at most $D_W(G)$. This completes the proof. ∎

Our conditions on G and H are symmetric. So it is enough to show that

$$Prob(\sum_{i \in R} d_i \geq \frac{w}{2}) < \frac{1}{2w}.$$

R is a random subset of U'. There are different models for random sets. In our case R is a random set of a given size. Another model is that each element of our universe will be in the set with a given probability. This model is more convenient. It is well-known in the theory of random graphs [Bo85] that by choosing the right parameters the two models yield basically the same theorems. So our next step is to change to the second model. For this we need some inequality for Bernoulli random variables.

Lemma 5.5. (Chernoff [Ch52]) *Let $X_1, X_2, ..., X_N$ be independent 0-1 random variables such that $Prob(X_i = 1) = p$. If $m \geq Np$ is an integer then*

$$Prob(\sum_{i=1}^{N} X_i \geq m) \leq \left(\frac{Np}{m}\right)^m \exp(m - Np).$$

An easy consequence of this is the following.

Lemma 5.6. ([ES74], [AV79]) *Let $X_1, X_2, ..., X_N$ be independent 0-1 random variables such that $Prob(X_i = 1) = p$. Then for every $0 < \beta < 1$,*
(i) $Prob(\sum_{i=1}^{N} X_i \leq \lfloor (1-\beta)Np \rfloor) \leq \exp\left(-\frac{\beta^2 Np}{2}\right)$.
(ii) $Prob(\sum_{i=1}^{N} X_i \leq \lfloor (1+\beta)Np \rfloor) \leq \exp\left(-\frac{\beta^2 Np}{3}\right)$.

And now let us see the reduction.

Lemma 5.7. *Let $X_1, \ldots, X_u$ be independent random variables such that $Prob(X_i = d_i) = p > 2\frac{D_W(G)}{u}$ and $Prob(X_i = 0) = 1 - p$. Let Δ be a random subset of $\{1, 2, \ldots, u\}$ of size $D_W(G)$. Then*

$$Prob(\sum_{i \in \Delta} d_i \geq \frac{w}{2}) < 2\ Prob(\sum_{i=1}^{u} X_i \geq \frac{w}{2}).$$

Proof. Let Δ_i be a random subset of $\{1, 2, \ldots, u\}$ of size i, all i-subsets of $\{1, 2, \ldots, u\}$ being equally likely. Let $P_i = Prob(\sum_{j \in \Delta_i} d_j > \frac{w}{2})$. Then $P_0 \leq P_1 \leq ... \leq P_u$.

Then

$$\begin{aligned}
Prob(\sum_{i=0}^{u} X_i d_i > \frac{w}{2}) &= \sum_{k=1}^{u} \binom{u}{k} p^k (1-p)^{u-k} P_k \\
&\geq P_{\lfloor \frac{1}{2}up \rfloor} \sum_{\lfloor \frac{1}{2}np \rfloor \leq k \leq \lfloor \frac{3}{2}up \rfloor} \binom{u}{k} p^k (1-p)^{u-k} \\
&\geq \frac{1}{2} P_{\lfloor \frac{1}{2}up \rfloor} \geq \frac{1}{2} P_{D_W(G)} \\
&= \frac{1}{2} Prob(\sum_{i \in \Delta} d_i > \frac{w}{2}).
\end{aligned}$$

∎

So at this point using the notation of the previous lemma we should give an upper bound on $Prob(\sum_{i=1}^{u} X_i \geq \frac{w}{2})$.

Let us fix the value of p to be $10\frac{D_W(G)}{u}$. Notice that the conditions of theorem 5.3 imply $\frac{w}{2} \gg E(\sum_{i=1}^{u} X_i) = \sum_i 10\frac{D_W(G)}{u} d_i = 10 D_W(G) \overline{d}_{U'}$. So we need an upper bound on the probability of that a sum of independent random variables is much greater than their expected sum. Chernoff bound is that kind of result but it is about Bernoulli random variables. We use the method of the proof of Chernoff's theorem to get the desired upper bound. For that we need the notion of characteristic function.

Definition 5.8. Let X be a random variable. Its characteristic function is e^{tX}, a random variable depending on the real parameter t.

The following lemma shows an important property of the characteristic function.

Lemma 5.9. *Let $X_1, \ldots, X_N$ be independent random variables. Then*

$$E(\prod_{i=1}^{N} e^{tX_i}) = \prod_{i=1}^{N} E(e^{tX_i}).$$

Now we have everything in order to prove the last lemma that we need.

Lemma 5.10. *Let $0 \leq d_1, d_2, ..., d_u \leq L = \frac{u}{1000 \log u}$ be integers and define $\overline{d}$ by $\sum_{i=1}^{u} d_i = \overline{d}u$. Let $X_1, X_2, ..., X_u$ be independent random variables such that $Prob(X_i = d_i) = p$ and $Prob(X_i = 0) = 1 - p$. Then $Prob(\sum_{i=1}^{u} X_i > 10p\overline{d}u) \leq \frac{1}{u^2}$.*

Proof. For all positive t

$$Prob(\sum_i X_i > 10p\overline{d}u) = Prob(e^{(\sum_i X_i)t} > e^{10p\overline{d}ut}).$$

Let us compute $E(e^{\sum_i X_i t})$.

$$E(e^{\sum_i X_i t}) = E(\prod_i e^{X_i t}) = \prod_i E(e^{X_i t})$$
$$= \prod_i (1 - p + pe^{d_i t}) = \prod_i (1 - p(1 - e^{d_i t})).$$

An easy calculation shows that this product is maximal if all d_i's are 0 or L, the maximal value of them. So

$$E(e^{\sum_i X_i t}) \leq (1 - p(1 - e^{Lt}))^{\frac{\overline{d}u}{L}} < (1 - p(1 - (1 + 2Lt)))^{\frac{\overline{d}u}{L}} < (1 + 2pLt)^{\frac{\overline{d}u}{L}} < e^{2p\overline{d}ut},$$

assuming that $Lt < 1$.

Using Markov's inequality

$$Prob(\sum_i X_i > 10p\overline{d}u) = Prob(e^{\sum_i X_i t} > e^{10p\overline{d}ut}) \leq \frac{e^{2p\overline{d}ut}}{e^{10p\overline{d}ut}} = e^{-8p\overline{d}ut}.$$

Fixing the value of t to be $\frac{1}{L}$ our bounds are still true and we obtain the desired upper bound. ∎

We obtain the promised packing theorem (theorem 5.3) as corollary.

Proof of Theorem 5.3. Applying lemma 5.10, lemma 5.7 and lemma 5.4 we obtain that for a random f with positive probability B_f has a perfect matching. This proves that there exists a concrete bijection f such that for the corresponding B_f has a perfect matching. This perfect matching is an identification of W and W', which together with f gives us a packing. ∎

The improved packing theorem yields the following improved lower bound on the randomized complexity of bipartite graph properties.

Corollary 5.11. *The randomized decision tree complexity of any non-trivial monotone bipartite graph property $P \in \mathcal{BP}_{u,u}$, is $\Omega(u^{\frac{4}{3}})$, i.e.,*

$$\mathcal{C}^R(\mathcal{BP}_{u,u}) = \Omega(u^{\frac{4}{3}}).$$

Proof. Let $P \in \mathcal{BP}_{u,u}$ be an arbitrary graph property. Let G and H be the graphs defined in definition 4.1. We are going to consider three cases.

Case 1. $\overline{d}(G)$ or $\overline{d}(H)$ is at least $\frac{1}{100}u^{\frac{1}{3}}$.

Applying theorem 2.3 we get the lower bound.

Case 2. $D_U(G)$ or $D_{W'}(H)$ is at least $\frac{1}{100}u^{\frac{2}{3}}$ and case 1 does not hold.

Because of the choice of G we can apply Yao's method and we get the lower bound $\Omega(\frac{D_U(G)}{\overline{d}(G)}u)$. In this case we know that $\overline{d}(G)$ is at most $\frac{1}{100}u^{\frac{1}{3}}$. These imply the lower bound.

Case 3. None of the previous cases holds.

Let us consider G_1 and H_1, the graphs defined in definition 4.2. It is easy to check that the conditions of the new packing theorem (theorem 5.3) are satisfied. So G_1 and H_1 can be packed. This leads to a contradiction that proves our theorem. ∎

6. The improved reduction from general to bipartite graphs

Given a graph property P one can construct other graph properties, that are useful for proving lower bounds on the complexity of P.

Let $P \in \mathcal{P}_v$ be a non-trivial, monotone graph property, viewed as a set of graphs with vertex set V. Divide V into equal parts $V = U \dot{\cup} W$, where $|U| = |W| = \frac{v}{2}$

Definition 6.1. Assume that $K_U \dot{\cup} E_W \notin P$ and $K_V - K_U \in P$. Let $\tilde{P} \in \mathcal{BP}_{\frac{v}{2},\frac{v}{2}}$ be the following property. $G \in \tilde{P}$ iff adding all the possible edges between the nodes of U to G gives us a graph having property P.

Definition 6.2. Let $P \in \mathcal{P}_v$. Let us assume that $K_V - K_U \notin P$. Let $\hat{P} \in \mathcal{P}_{\frac{v}{2}}$ be the following property. $G \in \hat{P}$ iff adding $\frac{v}{2}$ new nodes and all the possible edges incident to a new node to G gives us a graph having property P.

Considering these problems helps us because of lemma 6.3 below. Basically it says that it is enough to give a lower bound on the constructed property. In the case when the first definition might be applied the advantage is obvious, since we get a reduction to the bipartite case.

Lemma 6.3. *Let $P \in \mathcal{P}_v$ be an arbitrary non-trivial graph property. Let us assume that $\tilde{P}$ and $\hat{P}$ are the properties defined in 6.1 and 6.2. Then the following are true.*
(i) $\tilde{P}$ is a non-trivial, monotone bipartite graph property.
(ii) $\mathcal{C}^R(P) \geq \mathcal{C}^R(\tilde{P})$.
(iii) $\hat{P}$ is a non-trivial, monotone graph property.
(iv) $\mathcal{C}^R(P) \geq \mathcal{C}^R(\hat{P})$. ∎

Another advantage is that we might have "nice" critical graphs for the constructed property. This way it is easier to handle a lower bound on that property.

Lemma 6.4. *Let $P \in \mathcal{P}_v$ be a graph property and $\hat{P}$ be the property defined in 6.2. Let $G \in min(P)$. Then there is an $H \in min(\hat{P})$ such that the following are true.*
(i) $D(H) \leq 4\overline{d}(G)$.
(ii) H has at least $\frac{v}{10\overline{d}(G)}$ isolated nodes.

Proof. Let V be the vertex set of G, ($|V| = v$). Let us take any subset V_0 of V of size $\frac{v}{2}$. Then the subgraph of G induced by V_0, $G|V_0$ has the property $\hat{P}$. Thus $min(\hat{P})$ has

an element that is a subgraph of $G|V_0$. So it is enough to show that for an appropriate set V_0, $G|V_0$ has the properties (i) and (ii).

Choose $\min\left\{ \frac{v}{2}, \frac{v}{10\overline{d}(G)} \right\}$ nodes by the following greedy algorithm. Choose the node of minimum degree in G. Throw away that point and its neighborhood. Choose the node of minimum degree in the remaining graph and continue this procedure. The set that we shall get will be an independent set and its neighborhood will have size less than $\frac{v}{4}$. Let us refer to this independent set as A. Let us extend $N(A)$ to a set of size $\frac{v}{2}$ by adding some nodes of largest degree. Notice that we add at least $\frac{v}{4}$ new nodes. Let B be the set obtained after this extension of $N(A)$. Let V_0 be the complement of B. Let us remark that $A \subset V_0$.

It is easy to see that V_0 defined above is a good set. (i) follows from the fact that B has the set of nodes of the greatest $\frac{v}{4}$ degrees. (ii) is true because $A \subset V_0$. ∎

Now we are ready to prove the improved reduction to the bipartite case.

Theorem 6.5. *The randomized decision tree complexity of any non-trivial, monotone graph property $P \in \mathcal{P}_v$ is*

$$\mathcal{C}^R(P) = \Omega\big(\min\left\{ v^{\frac{4}{3}}, \mathcal{C}^R(\frac{v}{2}, \frac{v}{2}) \right\}\big).$$

Proof Let $P \in \mathcal{P}_n$ be arbitrary graph property. We consider two cases.

Case 1. $K_{\frac{v}{2}} \dot{\cup} E_{\frac{v}{2}} \notin P$ and $K_v - K_{\frac{v}{2}} \in P$.

Then $\mathcal{C}^R(P) \geq \mathcal{C}^R(\tilde{P}) \geq (\mathcal{C}^R(\frac{v}{2}, \frac{v}{2}))$.

Case 2. Case 1. does not hold.

Without loss of generality we assume that $K_v - K_{\frac{v}{2}} \notin P$. (This must hold for P or P^*.)

Let us consider $\hat{P}$. For any $G \in min(P)$ construct an $H \in min(\hat{P})$ guaranteed to exist by lemma 6.4. Choose any $F \in min(\hat{P}^*)$. We know that F and H have no packing. Start a prepacking the following way. Pack all the nodes of the top $\frac{v}{10\overline{d}(G)}$ degrees of F into isolated nodes of H. Let the unpacked nodes span the graphs F_1 and H_1.

It is easy to see that F_1 and H_1 can't have a packing. F_1 has maximal degree at most $10\overline{d}(G)\overline{d}(F)$. From 6.4 H_1 has maximal degree at most $4\overline{d}(G)$.

We finish the proof by considering the following two subcases.

Subcase 1. $\overline{d}(F) \geq \frac{1}{10} v^{\frac{1}{3}}$ or $\overline{d}(G) \geq \frac{1}{10} v^{\frac{1}{3}}$.

Then lemma 2.3 gives us that $\mathcal{C}^R(P) \geq \mathcal{C}^R(\hat{P}) = \Omega(v^{\frac{4}{3}})$.

Subcase 2. The hypothesis of subcase 1 is not satisfied.

Then Catlin's theorem (theorem 3.9.(ii)) gives us a contradiction. ∎

Combining Corollary 5.10. and Theorem 6.5 we get the following improved lower bound on general graph properties.

Theorem 6.6. *The randomized decision tree complexity of any non-trivial, monotone graph property $P \in \mathcal{P}_v$ is $\Omega(v^{\frac{4}{3}})$, i.e.,*

$$\mathcal{C}^R(\mathcal{P}_v) = \Omega(v^{\frac{4}{3}}). \quad ∎$$

7. Allowing two-sided error

In this section we consider how one can extend our result to the case when we allow two sided errors. Recall that $C_\lambda^{R2}(P)$ denotes the randomized decision tree complexity of property P when two-sided error is allowed (see definition 1.25). Remember that the coin tossing can result in an arbitrary decision tree. (Each tree has an associated probability.) Computing a function means that for each input the probability of error is small.

Again the basic questions and methods were described in [Ya77]. In the errorless case the proof started by a saddle-point transformation (see lemma 2.2). The important observation is that that transformation can be carried out in the 2-sided error case too.

Lemma 7.1. *Let d be a probability distribution on all the possible inputs and let $d(\underline{x})$ be the probability of input $\underline{x}$. (In the case of graph properties d describes a random graph.)*

We say that a deterministic decision tree $\mathcal{A}$ computes f with error λ over the input distribution d if $\sum_{\mathcal{A}\ \text{outputs}\ f(\underline{x})\ \text{on}\ \underline{x}} d(\underline{x}) \leq \lambda$.

We define the average case performance of a deterministic decision tree $\mathcal{A}$ over an input distribution d as $av(\mathcal{A}, d) = \sum_{\underline{x}} d(\underline{x}) cost(\mathcal{A}, \underline{x})$.

Then for a boolean function f

$$\mathcal{C}_\lambda^{R2}(f) \geq \frac{1}{2} \max_d \min_{\mathcal{A}} av(\mathcal{A}, d),$$

where the minimum is taken over all the deterministic decision trees computing f with error 2λ.

Again the lemma suggests a proof scheme for giving lower bound on $\mathcal{C}^{R2}(f)$ as follows. We define a 'hard' input distribution and prove that any algorithm that computes f for the majority of the inputs must ask many queries.

It is clear that if we have theorem 2.5 and lemma 2.6 for the two sided case that by plugging them into the rest of the proof we obtain the same lower bound. So the crucial question is what can we substitute for these theorems in the more general model. In this section we sketch the proof of the fact that theorem 2.5 and lemma 2.6 remain true even in the 2-sided error model. We follow the proof given in section 2 and explain what modification we need. We assume familiarity with the proof of theorem 2.5.

Let $\mathcal{A}$ be any 2-sided error algorithm computing a monotone, non-trivial bipartite graph property P. Repeating the algorithm several times and finally outputting the majority of the outputs we obtain a $\frac{1}{10}$-tolerant algorithm. The price of this is a constant factor in the complexity of the tree.

As lemma 7.1 suggest, first we define a distribution d on the bipartite graph universe. The definition of the distribution of the possible inputs is based on the same observation as in the original proof. Take the lexicographically first element from the list of minimal graphs for P. By adding edges to it we obtain G'. By deleting enough edges we obtain a graph not having property P (this is the point where we use the fact that G is the lexicographically first graph). Let $\mathcal{G}_{del}$ be the set of graphs which can be constructed this way. By putting back one 'wedge' we obtain a graph having the property P. Let $\mathcal{G}_{add}$ be the set of graphs which can be obtained this way. Our distribution is zero outside $\mathcal{G}_{del} \cup \mathcal{G}_{add}$, $prob(G \in \mathcal{G}_{del}) = prob(G \in \mathcal{G}_{add}) = \frac{1}{2}$ and the elements in $\mathcal{G}_{del}$ are equally probable and so are the elements of $\mathcal{G}_{add}$.

The final step in the proof is to prove that any algorithm $\mathcal{A}$ which is correct most of the time when the inputs have the distribution just described must ask many queries. In order to see this we concentrate on the neighborhood of a given x_i node. For this reason we fix which edge set to delete from the neighborhood of x_j, where $i \neq j$. Restricting the edges not adjacent to x_i defines a unique element of $\mathcal{G}_{add}$. For most of the possible restrictions $\mathcal{A}$ works correctly on this graph. We consider only these kinds of restrictions. After fixing the outside part let us take a look at the neighborhood of x_i. This neighborhood is (using the notation of the proof for theorem 2.5) $N_0 \cup N_i \cup N_{i+1} - R$, where R is a set of size $4\overline{d}$ or R is an empty set. To show that algorithm $\mathcal{A}$ asks many questions we must see that for most of the possible neighborhoods it outputs the right answer on the corresponding input. This can be easily shown by throwing away a constant fraction of restrictions. For the remaining ones now an easy calculation gives us the analysis that we need: even if R doesn't run over all possible subsets but only over a constant fraction then the average time that a fixed order of $N_0 - (N_i \cup N_{i+1})$ hits R is still $\Theta(\frac{D}{\overline{d}})$. That completes the proof of the 2-sided version of theorem 2.5.

With the same technique we obtain the 2-sided error version of lemma 2.6. That gives that our lower bound stands even when we allow 2-sided errors.

7.2 Theorem. *For any non-trivial, monotone graph property P*

$$\mathcal{C}^R(P) \geq \mathcal{C}^{R1}(P) \geq \mathcal{C}^{R2} = \Omega(v^{\frac{4}{3}}).$$

3. A LOWER BOUND FOR READ-ONCE-ONLY BRANCHING PROGRAMS

1. Branching programs

Branching programs are a model generalizing decision trees.

Definition 1.1. A branching program is a directed acyclic graph. To avoid confusion we will use the terms *nodes* and *arcs* to refer to the elements of this digraph. (We will use branching programs to do computation on graphs; these graphs (input objects) will have *vertices* and *edges*.)

One of the nodes of the branching program is a source (has fan-in zero) and is called START; other nodes are sinks (fan-out zero) and are called *terminal* nodes. All non-terminal nodes have fan-out two. The two arcs leaving a non-terminal node are labeled 0 and 1. Each non-terminal node is labeled by an input variable and each terminal node is labeled 0 or 1.

Each input string $\alpha = \alpha_1 ... \alpha_n \in \{0,1\}^n$ defines a unique path from START to a terminal node: the *computation path* determined by α. This path, after entering a nonterminal node labeled x_i, proceeds along the arc labeled α_i. The path ends at a terminal node. The function f computed by this branching program is defined by setting $f(\alpha)$ equal to the label of this terminal node.

The *size* of a branching program is the number of nodes. The *length* is the maximum length of the computation paths. The *multiplicity of reading* is the maximum number of times any particular variable is encountered as a node label along any computation path. In the case when the program is *leveled*, i.e. START is on level one and arcs only go from each level to the next level, we can introduce another complexity measure: the *width* of the program is the maximum number of nodes on any level.

* * *

An easy counting argument shows that most Boolean functions require exponential size branching programs. It is desirable to find nontrivial lower bounds for explicit Boolean functions (functions that belong to P or at least to NP).

The only known lower bound for the size of an unrestricted branching program computing an explicit Boolean function is due to Nečiporuk [Ne66], [Sav76] and is $\Omega(n^2/\log^2 n)$.

P. Beame and S. Cook observed [BC85] that Nečiporuk's technique actually applies to the "element distinctness" problem in the following sense. Let $x_1, \ldots, x_m$ be m integers between 1 and m^2. Written in binary, they form the input string of length $n = 2m \log m$. Any branching program deciding whether or not all the x_i are distinct must have size $\Omega(m^2) = \Omega(n^2/\log^2 n)$.

* * *

Another approach that has recently gained popularity is proving lower bounds for branching programs with bounds on various "resources" (width, multiplicity of reading). A similar approach to Boolean circuits has been quite successful recently [Ya85], [An85], [Ra85a], [Ra85b], [Ha86], [AB87], [Be86], [Ra87], [Sm87].

Our aim is to present a result of this kind.

Bounded width branching programs were introduced by Borodin, Dolev, Fich and Paul [BDFP83]. Their main result, completed by Yao [Ya83] , is a superpolynomial lower bound for width-2 branching programs computing the majority function. Shearer [Sh] recently proved an exponential lower bound for width-2 branching programs computing the "0 mod 3" function. These functions are *symmetric* (invariant under permutations of the variables). Interest in such functions was in part motivated by the conjecture stated in [BDFP83] that any bounded width branching program computing the majority function would require exponential size. This conjecture has been proved false by David Barrington's surprising result [Ba86] that the class of Boolean functions computed by polynomial size, width-5 branching programs coincides with nonuniform NC^1 (log-depth, fan-in 2 Boolean circuits) and thus contains all symmetric functions. This may be part of the reason why it is so difficult to find even *nonlinear* lower bounds for bounded width branching programs for symmetric functions.

The first such lower bound was derived using a beautiful Ramsey argument by Chandra, Furst, and Lipton [CFL83] for the function $\sum_{i=1}^{n} x_i = n/2$. Unfortunately, as it is often the case with Ramsey arguments, the bound is barely nonlinear: it is $\Omega(nw(n))$ where $w(n)$ is the inverse function of the van der Waerden numbers (see [GRS80]).

A more effective lower bound was obtained by P. Pudlák [Pu84]. Using a different Ramsey argument, he proves $\Omega(n \log\log n/\log\log\log n)$ lower bounds for threshold functions and separates (by the same amount) the power of width k and width $k+1$ branching programs for each k. He also proves a nonlinear lower bound *under no width constraint* for the majority function as well as an $\Omega(n \log\log n/\log\log\log n)$ lower bound for bounded width branching programs for all but a bounded number of symmetric Boolean functions.

The result of Babai, Pudlák, Rödl and Szemerédi [BPRSz] gives more effective, $\Omega(n \log n/\log\log n)$ and $\Omega(n \log n)$ lower bound for bounded and unbounded width resp. branching programs computing any member of a large class of symmetric Boolean functions.

N. Alon and W. Maass [AM86] using similar techniques obtain similar results.

A *read-k-times-only* branching program is allowed to encounter each variable at most k times along any computation path. This hierarchy of classes of branching programs was introduced by Masek [Mas76]. Wegener [We84] conjectures an exponential gap between the levels of this hierarchy and gives candidate Boolean functions computable with polynomial size read-k-times-only programs but conjectured to require exponential size read $(k-1)$-times-only programs.

No superpolynomial lower bounds are known, however, even for read-twice-only branching programs computing an explicit Boolean function, and no such bound will appear in this paper.

In connection with the history of read-once-only branching programs, we should mention a paper by Fortune, Hopcroft and Schmidt [FHS77]. In the context of program schemes, they gave an $\exp(c\sqrt{n})$ lower bound for computing an explicit function by read-once-only branching programs satisfying the additional restriction that the variables have to be examined in precisely the same order along each computation path. Without this restriction, however, their function is computable by a read-once-only branching program of polynomial size and is indeed defined by such a program.

Wegener [We84], Zák [Zá84] and Dunne [Du85] independently prove an $\exp(c\sqrt{n})$ lower bound for read-once-only branching programs computing certain, clique related graph properties. Wegener's property is NP-complete (presence of a clique of size $v/2$ where v is the number of vertices), Zák's is polynomial time decidable (recognizing the graphs that consist of a clique of size $v/2$ and $v/2$ isolated vertices.) We shall improve the lower bound to C^n (for a different function, also a polynomial time decidable graph property) (section 2 and 3).

Let $n = \binom{v}{2}$ and $x \in \{0,1\}^n$ representing a graph $G(x)$. Let $f_n(x)$ denote the number of triangles in $G(x)$ modulo 2.

Theorem 1.1. *There exists a positive constant α such that every read-once-only branching program computing f_n has size at least $2^{\alpha n}$.*

This result was obtained with co-authors and it appeared in [ABHKPRSzT86] and [BHST87].

Since then K. Kriegel and S. Waack [KW86] and M. Krause [Kr86] have obtained exponential lower bounds for different functions. For M. Kriegel and S. Waack's function the lower bound remains true even in a more general model. Their restriction on the branching program is that each computation path has length at most n. Krause [Kr88] and S.P. Jukna [Ju87] also relax the restriction on the branching program and obtain an exponential lower bound. Still, no nontrivial lower bound is known in the case of read-twice-only branching programs

* * *

The organization of the chapter is as follows.

The proof our main theorem will be presented in the next section. In the third section we discuss the relation between branching programs and space complexity. Our lower bound on read-once-only branching programs implies a lower bound on the space

complexity of the same function on a restricted Turing machine.

2. Read-once-only branching programs: the result

First we fix some notation.

We shall use the term "edge" to mean any of the $\binom{v}{2}$ pairs of vertices. (These are the edges of the complete graph K_v.) Let P be a path in a branching program. We shall say that an arc of P labeled 1 from a node labeled x_e has the effect of *accepting* the edge e; the arc labeled 0 from the same node *rejects* e. The edges accepted by P form the graph $A(P)$, the rejected edges form the graph $R(P)$. The union of these two edge sets constitutes the set $D(P)$ of edges *determined* by P. The *depth* of a node is its distance from START.

The strategy of our proof is the following.

Assume f_n is computed by a read-once-only branching program of size less than $2^{\varepsilon n}$ for some appropriately selected small positive constant ε. From this assumption we shall derive

(1) *the existence of a node w in the program, two paths P_0 and P_1 both leading from START to w, and an edge e not determined by either P_i, such that the parity of the number of triangles containing e in the graph $A(P_i) \cup e$ is i.*

The read-once-only property implies that after w, the program follows the same path of computation for input graphs $A(P_0) \cup e$ and $A(P_1) \cup e$ and thus leads to the same terminal node. This means these two graphs have the same number of triangles mod 2; the same observations hold for $A(P_0)$ and $A(P_1)$. This contradicts the choice of the P_i and e.

We proceed to showing how w, e, P_0, and P_1 satisfying (1) are found.

Proposition 2.1. *Let P be a path from START to a terminal node. If three edges are undetermined by this path, they cannot form a triangle.*

Proof. Suppose, to the contrary, that the edges e_1, e_2, e_3 of a triangle are left undetermined by P. Then the parity of the number of triangles in each graph $A(P) \cup e_i$ must agree with the parity of the number of triangles in $A(P)$. But then adding all the three edges at once will change the parity, a contradiction. ∎

Corollary 2.2. *The depth of each terminal node is at least* $v(v-2)/4$.

Proof: by Turán's Theorem in graph theory (cf. [Lo79, Probl.10.30,34]). Any path of length less than $v(v-2)/4$ leaves more than $v^2/4$ edges undetermined, forcing the graph of undetermined edges to contain a triangle. ∎

It follows that for any constant $c < 1/4$, there are precisely 2^{cn} computation paths of length cn beginning at START. Since the branching program has size less than $2^{\varepsilon n}$ there exists a node w such that at least $2^{(c-\varepsilon)n}$ paths of length cn connect START to w.

Let us fix c at a quite small value; any $c \leq 10^{-5}$ will be safe. Then, ε must be even smaller; let us set $\varepsilon = c^{3/2}$. At the same time we assume that v is sufficiently large.

Using w as a "checkpoint", we shall classify the edges according to their status at the time various computation paths pass through w. We shall see that these classes exhibit a strong structure.

Let D denote the set of edges determined by at least one path from START to w. Let U denote the set of the remaining (undetermined) edges; $|D| + |U| = n$.

Proposition 2.3. *Let P be any path from START to w. It is impossible that three edges e_1, e_2, e_3 form a triangle, where $e_1 \in D - D(P)$, $e_2, e_3 \notin D(P)$.*

Proof. The proof is similar to that of proposition 2.1. Suppose the contrary. The read-once-only property implies that e_1 is not tested along any path starting at w and therefore the parity of the number of triangles in $A(P)$ and $A(P) \cup \{ e_1 \}$ is the same. In other words, e_1 is contained in an even number of triangles in $A(P) \cup \{ e_1 \}$. Similarly we infer that the number of triangles containing e_1 in the graph $A(P) \cup \{ e_1, e_2, e_3 \}$ is even. But this number is precisely one greater than the number just shown to be even, a contradiction. ∎

Let AR denote the set of those edges which are accepted along some path from START to w and are rejected along some other. Clearly, $AR \subseteq D$.

Proposition 2.4. *There is no triangle e_1, e_2, e_3 with $e_1 \in AR$, $e_2, e_3 \in U$.*

Proof: The proof is a parity argument similar to the proofs of propositions 2.1 and 2.3. Suppose the contrary. Let be P be any path from START to w, accepting e_1 and let Q be some other path from START to w, rejecting e_1. Then the parity of the number of triangles in $A(P)$ must agree with the parity of the number of triangles in $A(Q)$. Similarly the parity of the number of triangles in $A(P) \cup \{ e_2 \}$ and in $A(Q) \cup \{ e_2 \}$ is the same. In other words the number of triangles containing e_2 in $A(P) \cup \{ e_2 \}$ and the number of triangles containing e_2 in $A(Q) \cup \{ e_2 \}$ have the same parity. The same holds for e_3. Clearly the number of triangles in $A(P) \cup \{ e_2, e_3 \}$ and the number of triangles in $A(Q) \cup \{ e_2, e_3 \}$ are the same mod 2. One can divide the triangles in $A(Q) \cup \{ e_2, e_3 \}$ into three classes, namely the triangles in $A(Q)$, the triangles containing e_2 in $A(Q) \cup \{ e_2 \}$ and the triangles containing e_3 in $A(Q) \cup \{ e_3 \}$. In the case of the triangles in $A(P) \cup \{ e_2, e_3 \}$ one must add the triangle $\{ e_1, e_2, e_3 \}$ to the corresponding classes. This contradicts the parity arguments above. ∎

One can deduce from proposition 2.3 that most edges determined along any path between START and w are actually determined along P, i.e. the set $D - D(P)$ is small.

Moreover, most edges determined by some path to w are both accepted and rejected along paths to w, i.e. $D - AR$ is a small set. More specifically:

Lemma 2.5. *(a)* $|D - D(P)| \leq 3c^{3/2}n$.
(b) $|U| > (1 - c - 3c^{3/2})n$.
(c) $|AR| \geq (c - \varepsilon)n$.
(d) $|D - AR| \leq 4c^{3/2}n$.

Proof. For a set A of edges, let $\deg_A(p)$ denote the degree of p with respect to the graph formed by A.

(a) Let $e = pq$ be any edge in $D - D(P)$. By proposition 2.3, every vertex is adjacent in $D(P)$ to at least one end of e. Therefore,

$$\deg_{D(P)}(p) + \quad \deg_{D(P)}(q) \geq (v - 2).$$

Adding up these inequalities for all $pq \in D - D(P)$ we obtain

$$\sum_p \deg_{D-D(P)}(p)\deg_{D(P)}(p) \geq (v-2)|D - D(P)|. \tag{2}$$

On the other hand, also by proposition 2.3, the neighborhood in $D - D(P)$ of any vertex p induces a clique in $D(P)$. Therefore

$$\binom{\deg_{D-D(P)}(p)}{2} \leq |D(P)| = cn = c\binom{v}{2}.$$

Consequently,

$$\deg_{D-D(P)}(p) \leq 1 + c^{1/2}v. \tag{3}$$

Combining (2) and (3),

$$|D - D(P)| \leq \frac{1 + c^{1/2}v}{v-2} \sum_p \deg_{D(P)}(p) = \frac{2 + 2c^{1/2}v}{v-2}|D(P)| \leq 3c^{3/2}n.$$

(b) follows immediately from (a) since $|U| = n - |D|$.

(c) Clearly, the logarithm of the number of START-to-w paths is a lower bound for $|AR|$.

(d) By (a), $|D| = |D - D(P)| + |D(P)| \leq 3c^{3/2}n + cn$. Combining this inequality with (c) we obtain $|D - AR| \leq (\varepsilon + 3c^{3/2})n = 4c^{3/2}n$. ∎

Lemma 2.5(b) implies that the graph U has a vertex p_0 of degree greater than $d = (1 - c - 4c^{3/2})v$. Let S be a set of precisely d neighbors of p_0 in U and let T be the complement of S ($|T| + |S| = v$).

Proposition 2.4 implies that no edge in AR has both of its endpoints in S. From this, it follows that AR is "mostly" bipartite, with bipartition (S, T). We can actually deduce even more structure: most vertices in T are adjacent in AR to either almost all or to almost no vertices in S (about half of the vertices will satisfy each alternative). More precisely, let us divide T into three classes, T_0, T_1, T_2. We shall refer to a moderately large constant K, $20 \leq K \leq 1/(8c^{1/2})$.

Let T_0 consist of those $p \in T$ which have more than $Kc^{1/2}v$ neighbors in S in the graph $D - AR$. We put $p \in T - T_0$ into T_1 or T_2 according to whether p has more AR-neighbors in S than U-neighbors or not. Let $\deg^S_{AR}(p)$ denote the number of AR-neighbors of p in S and analogously for other classes.

Lemma 2.6. *(a)* $|T_0| < 2cv/K$.
(b) For each $p \in T_1$, $\deg_U^S(p) \le 5c^{3/2}v$.
(c) For each $p \in T_2$, $\deg_{AR}^S(p) \le 5c^{3/2}v$.

Proof. By lemma 2.5(d),

$$|T_0|Kc^{1/2}v \le |D - AR| \le 4c^{3/2}n.$$

Claim (a) is now immediate.

To prove (b) and (c), let $p \in T - T_0$. Let N_1 and N_2 denote the sets of U-neighbors and AR-neighbors of p in S, respectively; let $n_i = |N_i|$. Since $p \notin T_0$, we have

$$n_1 + n_2 \ge |S| - Kc^{1/2}v > 6v/7. \tag{4}$$

On the other hand, by proposition 2.4, all edges between N_1 and N_2 belong to $D-AR$. By lemma 2.5(d) it follows that $n_1 n_2 \le 4c^{3/2}n < 2c^{3/2}v^2$. Consequently,

$$\min\{n_1, n_2\} \le \frac{2n_1n_2}{n_1+n_2} < 5c^{3/2}v. \quad \spadesuit$$

Let X denote the set of AR-edges between T_1 and S.

Corollary 2.7. *(a)* $(1 - \frac{8}{K})cv/2 \le |T_1| \le (1 + 4c^{1/2})cv/2$.
(b) $|AR - X| < \frac{3c}{K}v^2$.

Proof. We begin with (b). Clearly,

$$|AR - X| < |T|^2 + |T_2|\max_{p\in T_2}\deg_{AR}^S(p) + |T_0||S|.$$

By definition, $|T_2| \le |T| \le (c + 4c^{3/2})v$. We use lemma 2.6(c) to estimate the second term and lemma 2.6(a) and the fact $|S| < v$ for the last term.

For the upper bound in (a), we obtain from lemma 2.6(b) that

$$|T_1| \le \frac{|D|}{\min_{p\in T_1}\deg_D^S(p)} \le \frac{|D|}{|S| - 5c^{3/2}v}.$$

Lemma 2.5(a) provides the bound $|D| \le (c + 3c^{3/2})n$. By the definition of S (after the proof of lemma 2.5), $|S| = \lfloor(1 - c - 4c^{3/2})v\rfloor$. A combination of these estimates yields the desired upper bound.

For the lower bound we first observe that $|X| > (c - \varepsilon - 7c/K)v^2/2 > (1 - 8/K)cv^2/2$. This follows from lemma 2.5(c) and part (b) of this corollary. On the other hand, trivially, $|T_1| \ge |X|/v$. ∎

The structural consequence of lemma 2.6 and corollary 2.7 for the AR graph is that the subgraph X induced between T_1 and S is a nearly complete bipartite graph, and X contains almost all edges of AR.

In order to focus on X, let us make a decision on the value of each input variable (edge) in $AR - X$. There are $2^{|AR-X|} < 2^{(3/K)cv^2}$ possible outcomes (by corollary 2.7(b)). Let us choose the one that is the most frequent among the START to w computation paths. Having fixed these values,we still have at least

$$2^{(c-\varepsilon)n-(3/K)cv^2} > 2^{\frac{c}{2}v^2(1-8/K)} \tag{5}$$

computation paths left. Let Π denote the set of these paths:

$$\log|\Pi| \geq \frac{c}{2}v^2(1-8/K). \tag{6}$$

(The base of the log is 2.)

Let $t = |T_1|$ and $s = |S|$. We see, that $\log|\Pi|$ is nearly ts. In order to complete the proof, we show, that, unless situation (1) arises, the number of subgraphs of X arising from paths $P \in \Pi$ must be substantially less than 2^{ts}: only about $2^{ts/2}$. This is impossible because different paths define different subgraphs of X. (This in turn is true since the possible branchings on variables in $AR - X$ have been eliminated.)

The proof is based on a linear algebra counting lemma for $GF(2)$.

Let A, B, C be $(0,1)$-matrices of the same dimensions.

We shall say that $A \equiv C \bmod B$ if for every i, j, $B[i,j] = 0$ implies $A[i,j] = C[i,j]$.

Lemma 2.8. *Let $A_1, \ldots, A_N$ be different $t \times s$ matrices over the two-element field $GF(2)$. Furthermore, let B and C be $s \times s$ matrices over $GF(2)$. Let β be the number of 1's in B. Assume that $A_i^T A_i \equiv C \bmod B$ for every i. Then*

$$\log N < \beta + \frac{t}{2}(s + t + \log s). \tag{7}$$

Proof. First we estimate the number of $t \times s$ matrices of rank $\leq t/2$ over $GF(2)$. There are less than $2^{t^2/2}$ possible choices of the column space. Given the column space of dimension $\leq t/2$, there are $\leq 2^{t/2}$ choices for each column, giving a total of $\leq 2^{t(s+t)/2}$ matrices.

Next, we estimate the number of those A_i having rank $> t/2$. Such a matrix has a set of $t/2$ linearly independent columns; they are positioned in any of $\binom{s}{t/2} < s^{t/2}$ ways. Let us fix their positions, say columns $1, \ldots, t/2$, and decide their entries. Let us estimate, how many ways the remaining columns can be filled. For each pair (i,j) where $1 \leq j \leq t/2 < i \leq s$ and $B[i,j] = 0$, we have a linear condition $\sum_{k=1}^{t} x_{ik}A[k,j] = C[i,j]$ for the prospective entries x_{ik}. All these equations are linearly independent and their number is $\geq t(2s-t)/4 - \beta$. This reduces the number of candidates (2^{ts}) by a factor of $2^{-t(2s-t)/4+\beta}$. The number of those A_i of rank $> t/2$ is thus

$$< s^{t/2}2^{ts-t(2s-t)/4+\beta} = 2^{\beta+\frac{t}{2}(s+\frac{t}{2}+\log s)}. \tag{8}$$

Add the bound $2^{t(s+t)/2}$ on the number of low rank matrices to this; the figure in (7) is a generous overestimate of logarithm of the sum. ∎

Now we can complete the proof of our theorem.

Proof of theorem 1.1. Let now $s = |S|$, $t = |T_1|$ and for each $P \in \Pi$ let A_P be the $t \times s$ adjacency matrix of the bipartite subgraph of X defined by P. (This graph is the restriction to $T_1 \times S$ of $A(P)$.) Let B be the $s \times s$ adjacency matrix of the induced subgraph of $D - AR$ on S. (Recall that the complement, relative to S, of this graph belongs entirely to U by proposition 2.4.) Observe that the entries of $A_P^T A_P$ count modulo 2 the number of common neighbors of each pair of vertices in S. The falsity of (1) implies the statement that all the $A_P^T A_P \equiv C \bmod B \bmod 2$ for some fixed $s \times s$ matrix C. The number of 1's in B is $\beta = 2|D - AR| \leq 8c^{3/2}n < 4c^{3/2}v^2$ by Lemma 5(b). Using the upper bound of Lemma 7(a) for t we now infer from Lemma 8 that

$$(9) \qquad \log|\Pi| < \beta + \frac{t}{2}(s + t + \log s) < \beta + \frac{t}{2}(v + \log v) < \frac{c}{4}v^2(1 + 20c^{1/2} + \frac{2\log v}{v}).$$

This contradicts (6) for large v, completing the proof of the Theorem. ∎

3. Space-complexity: the eraser RAM

It has been noted ([Mas76], [BFKLT81], [Pu84]) that a lower bound $S(n)$ on the size of the smallest branching program computing a Boolean function f_n of n variables implies an $\Omega(\log S(n))$ lower bound on the space complexity of the family $\{ f_n : n = 1, 2, \dots \}$ on any reasonable model of computation.

Theorem 3.1. W. Masek [Mas76] *For $S(n) > \log n$, if $A \in SPACE(S(n))$ then A has branching program complexity $c^{S(n)}$ for some constant c.*

The converse of this theorem is not true. This has several resons. First, branching programs are a non-uniform model of computation as opposed to Turing machines. Second branching program complexity is bounded by 2^n while the space complexity on Turing machines can be arbitrary high. But for the non-uniform $SPACE(\log n)$ class the converse remains true. For definitions and details see [BoS88].

We might try to convert lower bound results on restricted branching programs for lower bound on the space complexity of restricted class of Turing machines or even RAM's.

The Fortune-Hopcroft-Schmidt result mentioned above corresponds to *on-line space complexity:* the input bits are read once and in a given order only. The [FHS77] result provides an $\Omega(\sqrt{n})$ space lower bound for such computation (independently of the given order of input bits).

General read-once-only branching programs suggest the following machine model which we call *eraser RAM*. This is a RAM with a special read-only input tape. The

machine decides in the course of the computation in what order to read the input but once an input cell has been read, it is erased. Let us measure the space required by a computation by the number of *bits* stored at any given time on the worktape.

The following is immediate.

Proposition 3.2. *If a language L can be recognized by an eraser RAM in space $S(n)$ then the set $L_n = L \cap \{0,1\}^n$ can be recognized by a read-once-only branching program of size $c^{\mathcal{O}(S(n))}$ for some constant c.*

The results of Wegener and Zák thus imply an $\Omega(\sqrt{n})$ lower bound for the eraser RAM space complexity of their respective Boolean functions. Our result implies a linear lower bound on the same model.

Theorem 3.3. *Let L be the language containing all graphs of even number of triangles. Then the eraser RAM space complexity of L is $\mathcal{O}(n)$.*

4. BROOKS COLORING IN PARALLEL

1. Models for parallel computation

In the last few years parallel computation has attracted a great deal of attention in the theory of algorithms. In this section we describe the two models most often used by theoretical computer scientists investigating parallel computation.

The first is parallel random access machines [Go77], [FW78]. A random access machine (RAM) (see [AHU74]) is more similar to a high level computer than Turing machines or circuits are. A RAM has its own local random access memory, each cell of which can store an arbitrarily large integer. The instructions for RAM's are multiplication, division, addition, substraction, conditional branches based on predicates '=','<','and','or' and 'not' and reading and writing into its memory. A parallel random access machines is a collection of RAM's $R_1, R_2, R_3 \ldots$ operating synchronously in parallel. The RAM's have access to an infinite common memory. All of the processors execute the same program in lock-step fashion, except that each processor R_i knows its unique processor number i, and this can be used in the instructions.

We need to specify what happens when two processors want to write into the same memory cell at once. There are several possible conventions for resolving this problem, defining different parallel RAM's. We list a few of these models: if concurrent writing is forbidden, we refer to the model as the CREW PRAM (concurrent read, exclusive write). In the CRCW PRAM model two processors can write into the same cell. Again, there are different possible conventions for what should happen when two processors want to write into the same location. One possibility is to allow concurrent write only when the processors want to write the same data. Another possibility is to let the the lowest numbered processor succeed in writing. These distinction are mostly technical. One can simulate a program following a conventions by another one following different convention at a cost which is $\mathcal{O}(\log n)$ parallel steps, where n is the size of the input. This factor is negligible for our purposes.

The input, an n bit number, is placed in the memory before the execution of an algorithm begins (the i-th bit is in the i-th cell). The output will be the contents of the first cells, when the execution halts.

We charge for several computational resources. One is the number of processors used. The other one is the time. The time of the computation is the total cost of instructions

executed by the processors. There are several conventions for measuring the time of an operation. The problem is that if we charge 'one' for every operation then we can produce large numbers very cheaply, such large numbers that to output them on a Turing machine takes a long time. But we don't want our time complexity to be far from the Turing machine complexity. What we can do is to charge to an arithmetic operation an amount proportional to the number of bits in the operands. Another solution is to require that any cell can hold only a number whose length is bounded by a polynomial in the input size. If we have programs satisfying this condition than the 'unit-cost' criterion gives us a complexity measure compatible with the one for Turing machines. In this case we define the computation time of a single run as the number of steps T executed during the run. The time complexity of an algorithm is a function $T(n)$, the maximum value of T over all 2^n possible size n inputs.

A PRAM algorithm is said to be efficient if it runs in time polynomial in the log of the input size and uses polynomially many processors. A problem solvable by such a PRAM algorithm is said to be NC. We refer to the algorithm as an NC algorithm.

A major goal in parallel computation is to prove that a given problem belongs to NC. An additional objective might be minimizing the number of processors used. In some situations we may also want to look at the precise parallel time bounds.

Our notion of uniform circuits is the one used in Cook [Co85].

A *circuit* is a directed acyclic graph with nodes (called *gates*) labeled as follows. A circuits has nodes distinguished as the inputs. They are of indegree 0 and are labeled $x_1, x_2, \ldots$. Other nodes of indegree 0 are labeled 1 or 0, representing boolean values. Nodes of indegree 1 are negation gates. All other nodes have indegree two and have a label, either AND or OR. Some distinguished nodes are output nodes and have labels $y_1, y_2, \ldots$.

If the circuit has n inputs and m outputs then it computes a function $f : \{0,1\}^n \to \{0,1\}^m$ in the obvious way.

A circuit family is a set $\{C_i\}_{i \in \mathbf{N}}$ of circuits, where C_i has i inputs. This computes a function $f : \{0,1\}^* \to \{0,1\}^*$.

A circuit family is logspace uniform if there exists a Turing machine which can compute the circuit C_i, given i in binary, in logarithmic space.

Theorem 1.1. *$f \in NC$ if and only if there exists a logspace uniform family of circuits computing f.*

Actually Cook's work [Co85] provided uniform circuits (with a different uniformity condition) as the definition of NC. The more convenient PRAM model was introduced later. We use PRAM's which are more convenient to program.

2. Parallel coloring algorithms

In I.4. we defined a proper coloring of a graph. Now we extend that definition to the case when some of the nodes are uncolored. A partial k-coloring is a function $c : V(G) \to \{ 1, \ldots, k \} \cup \{ * \}$ for which no two adjacent nodes have the same integer label. The set of nodes with image $*$ is the set of uncolored nodes.

If we have an unbounded number of colors then we can use different colors for different nodes and obtain a proper coloring. The problem arises if we want to use fewer colors. Let G be a given graph. The problem of determining the minimal number of colors for which a proper coloring exists (the *chromatic number*) is NP-complete [GJ79]. It is widely accepted that there is no hope of finding a fast sequential algorithm to solve this problem.

A possible plan of attack might be to relax our objective. We can look for an approximation for the optimum instead of the real value. For close approximation (better than a factor 2) the problem remains NP-complete. There are no known good approximate coloring algorithms. If we restrict ourselves to 3-regular graphs, then the best coloring algorithm uses $\mathcal{O}(\sqrt{v})$ colors [Wi].

Thus, we are forced to relax our objective again, to seek an upper bound for the chromatic number, and to find a corresponding coloring. Several theorems of this kind are known. It is very easy to show that any graph can be vertex colored with a number of colors at most one greater than its maximum degree. The proof gives a very simple sequential algorithm too. Usually though, colorings with fewer colors exist.

Theorem 2.1. (Brooks, see [Lo79]) *A simple graph G of maximum degree D can be colored with D colors iff either $D \geq 3$ and G contains no clique on $D + 1$ nodes, or $D <= 2$ and G has no odd cycle.*

The proofs in [Lo79], and [Be73], although not trivial, are algorithmic. Converting the standard algorithms to parallel algorithms seems difficult. M. Luby remarked that from the maximal independent set algorithm ([KW84],[Lu86]), it is easy to get an NC algorithm which colors a graph with $D + 1$ colors where D is the maximum degree of the graph. He defined the following 'maximal coloring' problem and stated the theorem below.The input is a graph, and associated with each vertex, a set of allowable colors. The answer is a partial coloring in which every point is colored with one color from its given color set in such a way that the output coloring can't be extended to additional nodes (while still insisting that the color of each node is chosen from its color set). By reducing the maximal coloring problem to the maximal independent set problem, M. Luby (in [Lu86]) exhibited an NC algorithm to find a maximal coloring. ¿From this algorithm one can easily conclude that we can find a maximal extension of a given coloring in NC, where a *maximal extension* of a coloring c is a coloring c' which uses the same number of colors and for which each uncolored node is adjacent to a point of each color.

(To do so, we apply M. Luby's maximal coloring algorithm to the graph induced by points not colored by c, and the allowable color set at point v is the set of colors not represented among v's neighbors.) We will apply this form of Luby's algorithm, so we state it in a separate lemma.

Theorem 2.2. (M. Luby [Lu86]) *Given a graph with a partial coloring, we can find a maximal extension of it in NC.*

Another expression of this result is that the following NC procedure exists.

Procedure **Extend** (S)
Given: A graph G, a partial coloring c with exactly t colors and a subset S of the points. Let C be the set of colored vertices.
Compute: A partial coloring c', using exactly t colors, that is an extension of c. Nodes not in $C \cup S$ are still uncolored and every uncolored point in S is adjacent to a point of each of the t colors.

Luby's maximal coloring algorithm did not settle the question of finding a Brooks coloring. In his thesis [Ka85], H. Karloff gave an algorithm for the case when the maximum degree is three. Using this result and the fact that the case $D \leq 2$ is easy we can assume that $D \geq 4$.

The main result of this chapter is to establish the parallel complexity of finding a Brooks coloring.

Theorem 2.3. *Given a graph G of maximum degree D, with no clique on $D+1$ nodes ($D \geq 4$), we can find a D -coloring of G in NC.*

This result was obtained with E. Szemerédi and it is published in [HSz88]. Recently H. Karloff [Ka88] and M. Karchmer and J. Naor [KN88] have announced alternative NC algorithms for Brooks' theorem. The proof will be given in the next few sections.

In the rest of this section we summarize what is known about parallel algorithms for other coloring problems.

Another way to relax a very hard problem is to restrict the inputs. There are several classes of graphs where better bounds are known on the chromatic number than the one in Brooks' theorem.

The best known class is the planar graphs. The famous four color conjecture says that every planar graph can be properly colored with 4 colors. The known proof of this theorem (and thus the known algorithm for finding a 4-coloring) is quite complicated. If we want only a 5-coloring, then simple proofs and algorithms are available. The parallel complexity of this problem is discussed in [BK]. They give an NC algorithm for finding a 5-coloring of planar graphs. There are other faster algorithms for this now [GPS87].

A theorem of Vizing shows that line graphs can be colored with very few colors. An obvious lower bound on the colors needed is the maximal degree of the original graph. Vizing's theorem says that one more color suffices. The parallel complexity of this problem is open.

A partition theorem of Lovász [Lo66] implies that graphs G without triangles can be colored with $\lfloor \frac{3D(G)}{4} \rfloor$ colors. The parallel complexity of finding such a coloring is unknown.

3. Outline of the algorithm

Using the procedure *Extend*, one can find a (partial) $D-3$-coloring such that every uncolored vertex is adjacent to a node of each color. Since we seek a D-coloring, we can assume that our coloring uses exactly $D-3$ colors; from this point until section 5 we assume that our coloring uses exactly $D-3$ colors. From the maximality it follows that the uncolored graph's maximum degree is at most 3. This is not enough for us - if this part contains a K_4 as a subgraph then we cannot extend our coloring to a total D-coloring of the whole graph. In this section we resolve this problem by modifying the original coloring.

Let denote the color classes by C_i $(i = 1, \ldots, D-3)$ and the set of uncolored points by U. As stated above, it is easy to see that in $G|U$ every vertex has degree at most 3. The definition of maximal coloring gives a stronger result, namely that every point u of degree 3 in $G|U$ has exactly one neighbor in every C_i. For every point of degree two in $G|U$, there are two possibilities: it has exactly one neighbor in every C_i or there is exactly one exception where it has two. These remarks are true not only for the initial coloring but for every maximal coloring. CN_u will denote the colored neighborhood of the uncolored point u : $CN_u = \{$ colored nodes adjacent to $u\}$. (The definitions of this paragraph depend on the current coloring. During the algorithm we will be changing it many times.)

To color U with three colors we need to get rid of K_4's in $G|U$. We can get rid of an uncolored K_4 by coloring one of its vertices. Using this method we will get an edge inside a color class so we have to remove the color from its other endpoint. If we are unlucky then we can create a new K_4 in $G|U$. In this case we must continue this procedure until we reach another configuration where we have the hope of using the color extension theorem. So the modification goes along a path which alternates between colored points and uncolored triangles. The procedure that finds this path, and the method that cleans up after every exchange phase are described in section 4.

In section 5 we finish the coloring and mention some open questions.

Before we start the procedure outlined we must overcome some problems. If we have a K_4 in $G|U$ all of whose vertices have the same colored neighborhood then our method will get stuck in an infinite loop.

This suggests the following definition of 'bad' K_4.

Definition 3.1. A K_4 A in $G|U$ is a *bad* K_4 iff for all $v, u \in A$, $CN_u = CN_v$. If a K_4 is not bad then we say that it is *good*.

If A is a bad K_4 then its common colored neighborhood contains two non-adjacent vertices, as otherwise G would have K_{D+1} as a subgraph. Using these two points and two points from the K_4 we can perform an exchange procedure which will improve our situation. The parallel implementation is the following procedure.

Procedure **Eliminate**
Given: A graph G with maximum degree $D \geq 4$, not having K_{D+1} as a subgraph, and a maximal $D-3$-coloring of G.
Compute: A new coloring of G, not having any bad K_4's, for which every uncolored K_4 was also uncolored when the procedure was called.

1) Let U be the set of uncolored nodes.
   Let $A_1, \ldots, A_r$ be the bad K_4's in $G|U$.
   Let $A_{r+1}, ..., A_t$ be the good K_4's in $G|U$.
2) Let U' be the empty set.
   For all i, $1 \leq i \leq r$, do in parallel:
   Let CN_i = the common colored neighborhood of nodes in A_i.
   CN_i does not induce a clique, so there exist $u_i \neq v_i \in CN_i$ that are nonadjacent. Add u_i, v_i to U'.
   Let $x_i \neq y_i$ be in A_i (arbitrarily). Color x_i with $c(u_i)$ and y_i with $c(v_i)$. Uncolor u_i and v_i.
   (It is worth noting that several processors may simultaneously be uncoloring a node.)
   U' consists of the newly uncolored nodes. The new coloring is proper because every new node with the same color came from a different component of $G|U$ and its neighbor in this color class lost its color.
3) Call procedure *Extend*$(A_{r+1} \cup \ldots \cup A_t)$.
4) Call procedure *Extend*(U').
5) Call procedure *Extend*$(V(G))$.

End Eliminate

The following claim shows that we achieved our goal.

Claim 3.1. *When Eliminate terminates, the $D-3$-coloring of G is maximal, every uncolored K_4 was also uncolored when procedure Eliminate was called and there are no bad K_4's.*

Proof. That the coloring of G is maximal is obvious.

Assume K is a currently-uncolored K_4 that was not completely uncolored when procedure *Eliminate* was called. Hence $z \in K$ lost its color during execution of step 2 of procedure *Eliminate.*

Case a) $z \in CN_i \cap CN_j$ for $1 \leq i < j \leq r$.

So z is adjacent to every node in A_i and every node in A_j. So after step 2, z has at least 4 uncolored neighbors, none of which is colored in step 3. Hence step 4 colors z, a contradiction.

Case b) $z \in CN_i$ for exactly one i, $1 \leq i \leq r$.

The same argument shows that after step 4, z has exactly 2 uncolored neighbors in A_i. Because no node in a bad K_4 had an uncolored neighbor outside of that K_4, the 4th node w in K must have been colored before procedure *Eliminate* was called. So $w \in CN_j$ for some $1 \leq j \leq r$. If $j \neq i$, w is adjacent to two uncolored nodes in A_j just before step 4

is run. So step 4 will color w. If $i = j$, $\{ w, z \} = \{ u_i, v_i \}$, yet u_i and v_i are nonadjacent, a contradiction.

The only thing that remains to be shown is that there are no bad K_4's. Because all the currently-uncolored K_4's were good when the procedure was called, it is enough to show that the corresponding colored neighborhoods didn't change. This is clear from the fact that otherwise step 3 would have colored the node. ∎

4. The alternating paths

After executing *Eliminate* we can execute our procedure which decreases the number of uncolored K_4's by a constant fraction.

Procedure **Modify_Coloring**
Given: A graph G of maximum degree D, and a maximal $D-3$-coloring without bad K_4's. Let U be the set of uncolored nodes.
Compute: A maximal coloring of G for which if U' is the set of uncolored nodes, and such that

$$\sharp \text{ of } K_4\text{'s in } G|U' \leq \frac{3}{4} \sharp \text{ of } K_4\text{'s in } G|U.$$

1) Let $\overline{Q} = \{ K_4$'s in $G|U \}$.
   Let Q be the set of nodes in $\overline{Q}$. (Q stands for "quadrilaterals".)
   Let $\overline{T} = \{ K_3$'s that are components in $G|U \}$. (T stands for "triangles".)
   Let T be the set of nodes in $\overline{T}$.
   Let $\overline{T}_i = \{ K_3$'s in $\overline{T}$ such that each node in the K_3 has two neighbors colored $i \}$, for $1 \leq i \leq D-3$. (Note that each node in such a K_3 has two neighbors colored i and exactly one colored j, for all $j \neq i$.)
   Let $\overline{T}_0 = \overline{T} - (\overline{T}_1 \cup \ldots \cup \overline{T}_{D-3})$.
   For $0 \leq i \leq D-3$, let T_i be the set of nodes in $\overline{T}_i$.
   Let $R = U - (Q \cup T)$. (R - which stands for "remainder" - is the part of the set of uncolored nodes lying in components that are neither K_4's nor K_3's.)
   Let C_i be the set of nodes colored i.
2) For $1 \leq i \leq D-3$ build an auxiliary digraph H_i on $C_i \cup \overline{T}_i$ as follows:
   Start with no arcs.
   For each triangle $L \in \overline{T}_i$, choose a representative node.
   For all $u \in C_i$ and $t \in \overline{T}_i$, add arc (u, t) iff u is adjacent to every node in t, and there is no $t' \in \overline{T}$, $t' \neq t$, every node of which is adjacent to u.

Then, for each arc $(u,t) \in E(H_i)$, add to $E(H_i)$ an arc from t to the node $v \in C_i$, that is different from u and adjacent to t's representative. The outdegrees of nodes in C_i are at most one. A node t in $\overline{T}^i$ either has indegree = outdegree ≤ 1, or there are two nodes $a \neq b \in C_i$, and the only arcs in H_i incident with t are $(a,t),(t,b),(b,t),(t,a)$.
Motivation: If one removes the color of a node colored by i then this node may create a new uncolored K_4 with a previously uncolored triangle. If this triangle doesn't belong to $\overline{T}_i$ then we can handle the problem by using the coloring extension procedure. Otherwise we must continue the exchange method. We color the representative of this triangle with color i and uncolor its neighbor in C_i. The edges of H_i tell the exchange procedure what to do next after we remove a color or color a representative.

2a) We need the following notation. Say $v \in C_i$ is an *endpoint* iff v's outdegree in H_i is 0, and an *inner point* otherwise.

3) For each $K \in \overline{Q}$ (K is a K_4), choose a colored vertex v such that v has 1,2 or 3 neighbors in K - such a vertex always exists, because K is a good K_4 - and if there are several such points choose one among them which has the most neighbors in K. (Why we choose the vertex with the most neighbors in K will become clear only at the end of the proof of lemma 4.1.) Node v is an *initial point with respect to* K. It is the *initial point* iff it is an initial point with respect to some K. If v is the initial point with respect to K, choose one of v's neighbors in K. We will refer to this chosen point as K's *representative*.
Let $\overline{Q}_i = \{ K \in \overline{Q} |$ the initial point with respect to K lies in $C_i \}$.
Label K with 'color the representative with color i' where $K \in \overline{Q}_i$.

4) For each i, label $x \in C_i$ with 'remove color' iff there exists a path in H_i from an initial point in C_i to x, and label $y \in \overline{T}_i$ with 'color the representative with color i' iff there exists a path in H_i from an initial point in C_i to y.

5) For each i in parallel do:
Uncolor each node labeled 'remove color'.
Next, color with color i the representative of any K_3 or K_4 labeled with 'color the representative with color i'.
Note that the coloring remains proper.

6) For each i, $1 \leq i \leq D-3$, do:
Let $\overline{E}_i = \{ T - (\text{representative of } T) \mid T \text{ was a labeled } K_3 \text{ in } \overline{T}_i \}$. ($E_i$ consists of the uncolored K_2's that remain after each labeled K_3's representative is colored.)
Let $\overline{T}_i{}' = \{$ unlabeled $T \in \overline{T}_i \}$.
Let $\overline{Q}_i{}' = \{ K -$ representative of $K) \mid K \in \overline{Q}_i \}$.
Let $E_i, T_i{}', Q_i{}'$ be the respective sets of points.
Let $N_i = \{$ labeled vertices in $C_i \}$. (The N stands for 'newly uncolored'.) The nodes in N_i can be either inner points or endpoints. It is easy to see that $\sharp$ of endpoints in $N_i \leq |\overline{Q}_i|$.
Note that the uncolored vertices of G are exactly $R \cup T_0 \cup \{ \cup_i [E_i \cup T_i{}' \cup Q_i{}' \cup N_i] \}$.

7) For each $L \in \cup_i (\overline{T}_i{}' \cup \overline{Q}_i{}' \cup \overline{E}_i)$, find a vertex in L, if it is exists, not having a neighbor of every color (find only one, even if 2 or 3 exist). Color that vertex with some color not occurring on its neighbors. Note that the coloring remains proper.

8) Apply procedure *Extend* ($\cup_i N_i$).
Now every completely uncolored quadrilateral K either contains 2 endpoints, or contains one endpoint v and $K - \{ v \} \subset R$. We prove this fact below. (If the first case held for every completely uncolored K_4, we would have succeeded in halving the number of uncolored K_4' s.)
9) For each completely uncolored quadrilateral K containing exactly one endpoint v (so that $K - \{ v \} \subset R$), choose one node $y \in K - \{ v \}$ with 4 or more uncolored neighbors. (This y must exist.) Let $Y \subset R$ be the set of chosen y's. $G|R$ has maximum degree at most 3. Find a maximal independent set of $G|Y$. (Because of the degree bound on $G|R$ the cardinality of this independent set must be at least $\frac{|Y|}{4}$.) Color each node in the independent set with some color not occurring on its neighborhood.
10) Apply procedure *Extend*(V).

End Modify_Coloring

In order to prove the correctness of the algorithm we must first prove the lemma mentioned under step 8.

Lemma 4.1. *Just after step 8 of Modify_Coloring, every completely uncolored quadrilateral K either contains 2 endpoints, or contains 1 endpoint v and $K - \{ v \} \subset R$.*

Proof. Each uncolored quadrilateral K, just after step 8, must fall into (at least) one of the following classes.

a) K contains exactly one newly uncolored node z, and z is an inner point.
b) K contains at least two inner points.
c) K contains exactly one endpoint and exactly one inner point.
d) K contains exactly one endpoint, and the remaining nodes are in T.
e) K contains exactly one endpoint, and the remaining nodes are in Q.
f) K contains exactly one endpoint, and the remaining nodes are in R.
g) K contains at least 2 endpoints.

We now prove cases a)-e) cannot occur. We will be making heavy use of the following two simple facts.

Remark 1. Just after step 5, every inner point originally colored with color i has, in its uncolored neighborhood, an edge in E_i. At least one of these points remains uncolored after steps 7 and 8 are executed.

Remark 2. Immediately after step 8 every newly uncolored node in an uncolored K_4 has exactly $D - 3$ colored neighbors, and its 3 uncolored neighbors are exactly the remaining nodes of the K_4.

Case a) cannot occur, because if z is the only newly-uncolored node in an uncolored K_4 K, then z's uncolored neighbors span a K_3, yet by remarks 1 and 2, this can't occur.

If, instead, any uncolored quadrilateral K contained distinct inner points u, v just after step 8, u and v must have originally had colors i and j, respectively, $i \neq j$. As for case a), by remarks 1 and 2, after step 8 u has one uncolored neighbor from E_i in K. The same goes for v, with 'E_i' replaced by "E_j". A contradiction, because no node in E_i is adjacent to a node in E_j. So case b) is impossible.

Now we discuss case c). Let x be K's lone endpoint, and y, its lone inner point. Node x originally has color i, and y, color j. Again, $i \neq j$. Let z and t be the remaining two vertices of K. By remarks 1 and 2, z or t came from E_j. In fact, both must be in E_j. By the definition of T_j, z and t have exactly one neighbor colored i, namely x. Step 5 uncolored x so that after step 5, z and t had no neighbor colored i. This is a contradiction, since step 7 will color either z or t.

Let K be a case d) K_4; z, its lone endpoint; and i, its original color. If $K - \{ z \} \subset T_j$, $j \neq i$ (possibly $j = 0$) , for one of those three nodes w, z was w's only neighbor colored i (otherwise K would be in T_i). After color i is removed from z, step 7 will color at least one of those three nodes. Hence the other three nodes are in T_i. Since z is an endpoint, z had to have three pairwise adjacent neighbors, none of which is in K, all of which are in T. At least one of these, s, will remain uncolored after step 7. Hence, after step 7, s and all of K are still uncolored, and step 8 will color z.

Let K be a case e) K_4; z, its lone endpoint; and i, its original color. If the other three nodes are in Q_j, $j \neq i$, then the argument in case d) produces a contradiction. So $K - \{ z \} \subset A$, where A is a K_4 in $\overline{Q} - i$. By the method used to choose an initial point in step 3, z must be the initial point with respect to A, and therefore some node in $K - \{ z \}$ is A's representative and is colored in step 5. ∎

Using this lemma the proof of correctness is easy.

Lemma 4.2. *The procedure Modify_Coloring decreases the number of uncolored K_4's to at most $\frac{3}{4}$ of its previous value.*

Proof. All the uncolored K_4's existing when the algorithm terminated were uncolored after step 8. Using lemma 4.1 we divide the uncolored K_4's into two classes. This implies a classification of endpoints: in an uncolored K_4 an endpoint is *ugly* if immediately after step 8 it is the lone endpoint of its uncolored K_4, and *nice* otherwise. Using this notation it is easy to see the following inequalities.

$$\sharp \text{ of old uncolored } K_4\text{'s } \geq \sharp \text{ of endpoints } = \sharp \text{ of ugly endpoints } + \sharp \text{ of nice endpoints.}$$

But we know that

$$\frac{3}{4}\sharp \text{ of ugly endpoints} \geq \sharp \text{ of uncolored } K_4 \text{ having exactly one endpoint,}$$

and

$$\frac{1}{2}\sharp \text{ of nice endpoints} \geq \sharp \text{ of uncolored } K_4 \text{ having at least two endpoints.}$$

So

$$\frac{3}{4}\sharp \text{ of old uncolored } K_4\text{'s } \geq \sharp \text{ of uncolored } K_4\text{'s.}$$

∎

5. Conclusion

Using the previous result we can find in NC a partial coloring with $D-3$ colors such that the uncolored part has maximum degree 3 and doesn't contain any K_4's. So completing the coloring involves solving the original question on maximum-degree-3 graphs, which is an easier problem. We use H.Karloff's NC algorithm [Ka85] for 3-coloring a maximum-degree-3 graph without K_4's. So the final algorithm is the following:

Program **Brooks_Coloring**
Given: A graph G with maximum degree $D \geq 4$, not having K_{D+1} as a subgraph.
Compute: A D-coloring of G.

1) Using procedure *Extend* get an initial maximal $(D-3)$-coloring.
2) Call procedure *Eliminate*.
3) Call procedure *Modify_Coloring*.
4) If there are four pairwise-adjacent uncolored nodes then go to 2.
5) Color the set of uncolored nodes with 3 new colors.

End Brooks_Coloring

The correctness of the algorithm easily follows from the lemma 4.2.

The main result of this chapter is an application of the maximal coloring algorithm. The essence of our Brooks coloring algorithm is a procedure designed to get rid of the troublesome K_4's, and we perform a natural exchange procedure along alternating paths to do so. The extension lemma takes care of the messy uncolored part after the exchange. The natural question is whether there are further applications of the maximal coloring algorithm.

5. A FAST PARALLEL ALGORITHM ON DENSE GRAPHS

1. The Hamiltonian cycle problem

In this chapter we consider the problem of finding a Hamiltonian cycle. Recall that a Hamiltonian cycle of a graph G is a cycle going through all nodes of G. Deciding whether a graph has a Hamiltonian cycle is a classical NP-complete problem [GJ79]. As we saw in the previous chapter it is often very fruitful to relax the requirements of a problem. First let us consider some related problems which will help us later.

The maximum independent set problem is NP-complete. On the other hand, if we simply want to find a non-extendible independent set (a maximal independent set) then it becomes an NC problem ([KaW85],[Lu86],[ABI86],[GS87]).

The deterministic parallel complexity of other very important problems like matching is still not known. There is a method ([Lo79b], [KUW86],[Ka86]) which shows that matching is in RNC (random parallel polylog time), but it is still unknown whether matching is in NC. One relaxation of the matching problem is maximal matching. This obviously is in NC, as implied by the result for maximal independent set. An easier and more efficient algorithm can be found in [II86].

The Hamiltonian path problem can be relaxed to the maximal path problem. R. Anderson [And87] presented a reduction of this problem to matching. The reduction implies that the maximal path problem is in RNC. His algorithm starts out from a system of paths and glues them together. He and A. Aggarwal [AA88] extended this method to the problem of finding a depth first search tree. With M.Y. Kao they applied the same method to find a depth first search tree in directed graphs [AAK].

Now let us return to the Hamiltonian cycle problem. There are known classes of graphs for which all the members of the class are Hamiltonian. One sufficient condition for being Hamiltonian is Dirac's condition ([Di52], [Be73],[Lo79]): if a graph G has minimal degree at least $\frac{n}{2}$ where n is the number of vertices then G has a Hamiltonian circuit. At STOC'87 M.Goldberg proposed the problem: Is there any NC algorithm which finds a Hamiltonian cycle for graphs lying in the class defined by Dirac's theorem? In this paper we present an NC algorithm which uses methods similar to Anderson's. In our case we have the advantage that the input graph has many edges. Thus it is relatively easy to merge different paths.

The algorithm as described here was published as a technical report. Meanwhile I learned that M. Karpinski and E. Delhaus obtained the same result. A joint version will be submitted for publication.

2. The outline of the algorithm

First, our algorithm finds a Hamiltonian path in the given graph. Having the Hamiltonian path it will be very easy to finish the algorithm, i.e., to find a Hamiltonian cycle.

The main idea of finding a Hamiltonian path is to maintain a path system that covers the graph, i.e., a set of paths such that the vertex sets of the paths are pairwise disjoint and their union is the whole vertex set. The algorithm consists of phases. In each phase we try to merge different paths: this way we can reduce the number of paths by a constant factor.

In order to merge paths we use a special operation. The operation merges two paths P, and Q. (In our case every path contains at least one edge.) Let u, v be the two endpoints of P. Let us assume that u and v have distinct neighbors in Q and these neighbors are consecutive nodes in Q. In this case one can easily merge P into Q. The endpoints of the new path will be the same as Q's endpoints.

We want to merge several paths into others in parallel. We might have conflicts during the parallel merging. To overcome this problem we want to find for each path several ways to merge it into another path. The main observation is the following. Let us assume that we have a path system and a fixed path on it. If the endpoints of the given path have high degree toward the outside of the path, then we can find many ways to merge it into other paths.

As the idea above suggests, we will handle the paths differently depending on whether the endpoints have high degree toward the outside. We will refer to these paths as "social". We say that a path is "introverted" if it is not social. The exact definitions will be given in the next section.

We use standard graph theoretical notation. We refer the reader to [Lo79]. G is the input of the algorithm, i.e., it is a simple graph, with minimal degree at least $\frac{n}{2}$ where n is the number of nodes. $d(u)$ is the degree of the node u. $d_S(u)$ is the number of edges going from node u to the set of vertices S. $\{ P_i \}_{i=0}^{k}$ will denote a set of paths in G such that $\cup_i V(P_i) = V(G)$ and the $V(P_i)$'s are disjoint. We will refer to this as a *path-cover* of G.

In this section we give the formal definition of our elementary merging operation and introduce the notion of different types of paths. The basic idea of these types comes from Lemma 3.3, which says that if the endpoints of a path are connected with many edges to another path then there are several possible ways to merge that path into the other one.

Definition 3.1. Let P and Q be two paths whose vertex sets are disjoint. Let their vertex sets (corresponding to the order on the paths) be $\{ u_1, ..., u_l \}$ and $\{ v_1, ..., v_m \}$. If u_1v_i and u_lv_{i+1} are edges of the graph then $v_1...v_iu_1...u_lv_{i+1}...v_m$ is a path. If u_1v_{i+1} and u_lv_i are edges of the graph then $v_1...v_iu_l...u_1v_{i+1}...v_m$ is a path. If in a path system we transform two paths into one using one of the remarks above then we'll say that we performed an *elementary merging operation.* We'll say that we *merged P into Q along the edge v_iv_{i+1}.*

Definition 3.2. Let P be a path in G. Let u, v be the two endpoints of P. We call P *social* if

$$d_{V(P)}(u) + d_{V(P)}(v) + 1 \leq |V(P)|.$$

We say a path P is *introverted* if it is not social.

We need the following lemma.

Lemma 3.3. *Let P be a path with endpoints u, v. Let Q be a path with vertex set disjoint from P's. If there are no edges from any endpoint of P to an endpoint of Q then there are at least*

$$\max \{ d_{V(Q)}(u) + d_{V(Q)}(v) + 1 - |V(Q)|, 0 \}$$

edges on Q along which we can merge P into Q via an elementary merging operation.

Proof. Let $\{ q_1, q_2, ..., q_l \}$ be the vertex set of Q (q_1 and q_l are the endnodes and the indices follow the order on the path) . So $l = |V(Q)|$. Let $d = d_{V(Q)}(u)$ and $e = d_{V(Q)}(v)$. Let $\{ q_{i_1}, ..., q_{i_d} \}$ be the neighborhood of u on Q. Because of our assumption $1 < i_1$ and $i_d < l$. Let $F = \{ q_{i_1-1}, q_{i_1+1}, q_{i_2+1}, ..., q_{i_d+1} \}$. F contains $d + 1$ nodes not in Q. If v is adjacent to one of them then one can easily find an edge where our elementary operation can be performed. Actually we can assign different edges of Q to different elements of F in such a way that an edge between v and an element of F gives a possible elementary merging operation along the corresponding edge. If the degree of v toward Q is greater than $|Q-F|$, then one can find a way to merge. Actually there will be at least $e - (l - (d + 1))$ edges going from v to F. This implies the statement of the lemma. ∎

The lemma above has the following important consequence for the path covering.

Consequence 3.4. *Let $\{ P_i \}_{i=0}^{k-1}$ be a path-cover of G. We assume that P_0 is social and there are no edges going from its endpoint to the endpoints of other covering paths. Then there are at least k edges on one of the paths, i.e. on $\cup_{i=1}^{k-1} P_i$ along which one can merge P_0 into another path.*

Proof. Let u, v be the endpoints of P_0. Let $d_i (i = 0, \ldots, k-1)$ be the degree of u toward P_i and let $e_i (i = 0, \ldots, k-1)$ be the corresponding degrees of v. Let n_i be the number of nodes on P_i. Using this notation we have

$$\sum_{i=0}^{k-1} d_i + \sum_{i=0}^{k-1} e_i = d(u) + d(v) \geq \frac{n}{2} + \frac{n}{2} = n = \sum_{i=0}^{k-1} n_i.$$

After rearrangment we get

$$\sum_{i=0}^{k-1} (d_i + e_i + 1 - n_i) \geq k.$$

We can delete the nonpositive terms in the sum and the inequality remains valid. We assumed that P_0 is social so during the simplification above the term $d_0 + e_0 + 1 - n$ is at most 0 and will be deleted. After the deletion we have

$$\sum_{i=1}^{k-1} max\{ d_i + e_i + 1 - n_i, 0 \} \geq k.$$

By lemma 3.3 we get the result. ∎

In the case of introverted paths we need a little trick. The truth of the generalization of consequence 3.4 stated below, easily follows from the proof we just presented. It will be used in the next section.

Lemma 3.5. *Let S be a subset of $V(G)$ and $\{ P_i \}_{i=1}^{k-1}$ be a path-cover of $V(G) - S$. Let P be a path in S with endpoints u, v. Let us assume that $d_S(u) + d_S(v) + 1 - |S| \leq 0$ and that there are no edges going from u or from v to any endpoints of the path-cover. Then there are at least k edges on $\cup_{i=1}^{k-1} P_i$ along which P can be merged into a covering path.* ∎

The lemmas above show that if we have a path-cover such that there are no edges between endpoints of different paths and all paths are social then we have many options for performing the merging operation. An easy application of matching theory (to be shown later) shows that in this case we can assign a merging operation to each path in such a way that the corresponding edges are different for different paths. We will see that these operations can be performed in parallel and in this way the number of paths can be reduced by a factor of 2. In order to apply this idea for the case of introverted paths we need a little trick.

4. Introverted paths

First we prove that a introverted path can be closed to a cycle, i.e., the graph induced by the vertex set of an introverted path has a Hamiltonian cycle.

Lemma 4.1. *Let P be an introverted path between endpoints u and v. Then one of the following three statements is true:*
(a) The number of nodes on P is at most 2, i.e., every node on P is an endpoint.
(b) There exists an edge between the two endpoints of P.
(c) There exists a neighbor u' of u on P and a neighbor v' of v on P such that u' and v' are adjacent via an edge of P and u' is between v' and v on P.

Proof. Let us assume that P has more than 2 nodes. Let $\{ w_1, ..., w_l \}$ be the vertex set of P ($w_1 = u$ and $w_l = v$). The order on the path is the same as the order of the indices. Let $d = d_{V(P)}(u)$ and $e = d_{V(P)}(v)$. We can assume that there is no edge between u and v. Let $\{ w_2, w_{i_2}, ..., w_{i_d} \}$ be the neighborhood of u. If (c) does not hold then $\{ w_1, w_{i_2-1}, ..., w_{i_d-1}, w_l \}$ cannot be adjacent to v. This means that $l - (d+1) \geq e$. So P is social. This contradicts our assumption. ∎

In the first case of the lemma above the path has only endvertices, in the other two cases one can easily find a Hamiltonian cycle in the graph induced by the introverted path. This allows the possibility of doing another kind of merging. What we are going to do is the following. We make pairs of the introverted paths. If there is an edge between two matched paths then we can merge them along this edge. After repeating this step we are left only with pairs of independent paths. If we have an independent pair of introverted paths then the shorter path will behave like a social path. What we mean is that if we consider the union of the two underlying vertex sets and the shorter path in it then we can apply lemma 3.5. In this way we can use the method developed in section 3.

5. The general case

We need only a few additional tricks to handle the general case. So now we are able to present the complete algorithm.

The algorithm starts with an initial path-covering. Because of our technique each path in the cover must have two different endnodes. Because of this, the initialization step is not totally obvious. After initialization, using the results of the previous sections, we reduce the number of paths by a factor $\frac{3}{4}$. Having these two procedures we can easily get the complete algorithm.

Procedure **Initialization**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G.
Compute: A path-cover of G such that each path in the cover has at least two points.

1) Find in parallel a maximal mathing M in G.

2) Find in parallel a maximal matching N between the nodes not covered by M and the nodes covered by M.

3) The edge set $M \cup N$ will be $\{ P_i \}_i$, the initial path-covering.
{ Lemma 2.5.2. proves that $M \cup N$ is really a path-cover of G.}

End Initialization

Procedure **Reduce_path_cover**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G, and a path-cover of G with at least 2 paths.
Compute: A new path-cover with at most $\frac{3}{4}$ as many paths.

1) Classify each path as social or as introverted in parallel.
2) Pair up the introverted paths with at most one leftover.
3) Repeat until there are only pairs of independent introverted paths.
   For all connected introverted path pairs do a)-d):
   a) Find all the pairs where there is an edge between the two paths (we will refer to these two paths as *connected paths*).
   b) Find a Hamiltonian cycle in each introverted path having at least 3 nodes.
   c) Merge the pair of cycles and possible edges by a connecting edge.
   d) Pair up the unmatched introverted paths.
   { Exiting this loop we have a path-cover of G. The cover consists of social paths and pairs of independent introverted paths. We will refer to the social paths and to the pairs of introverted paths as *generalized components* of the cover. Sometimes we'll omit the word generalized.}
4) Find a maximal matching in parallel between the edges connecting endpoints of different paths. Connect these paths by this edges. Kill all the cycles by deleting one new edge of each.
   {At this point we have a path-cover such that there are no edges between endpoints of different paths. Let us consider the path-cover after step 3. We refer to a generalized component that was not changed in step 4 as an *untuoched component*. The idea is that the part of the old cover which was changed by step 5 got the advantage that we seek. So we need to work with the untouched part, where we can use our technique.}
5) For every untouched component do the following: If it is a pair of introverted paths then take the smaller one, otherwise take the corresponding social path. Now we have a path and we want to merge it into another one. Find all the edges on the path system such that an elementary merging can be performed along it.
6) For each untouched component we have many possible merging operations. For different untouched components find a merging from step 5 which uses different edges. We will see that this can be solved in the following way. We construct an auxiliary bipartite graph H_1 between the untouched components and the edges along the paths. A component will be connected to an edge iff along it there is a possible merging (found in step 6) into the corresponding path. A maximal matching of H_1 will give the 1-1 map from untouched components into edges.

7) Perform as many elementary operations as possible. This can be done as follows. Make the following auxiliary directed graph H_2. The nodes will correspond to paths in the cover. There is an edge going from a path P to a path Q iff one of the merging operations, found in step 6 merges P into Q. In this digraph every node has outdegree 1 or 0. So it will contain rooted trees, edges directed toward the root and directed cycles with trees connected to it, the edges of the trees directed toward the cycle. If we delete exactly one edge from each directed cycle then we get a system of rooted directed trees. The corresponding merging operations can be performed in parallel.

End Reduce_path_cover

Program **Find_Hamiltonian_circuit**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of nodes in G.
Compute: A Hamiltonian circuit of G.

1) Initialization
2) Repeat while there are at least two paths
   a) Reduce_path_cover
   {At this point of the algorithm we have a Hamiltonian path $\{ v_1, ..., v_n \}$ of G.}
3) If v_1v_n is not an edge of G then find a pair of edges of the form v_1v_{i+1} and v_nv_i.
   {We will see that in this specific case this kind of edge pair exists in the graph.}
4) Output the cycle $v_1...v_nv_1$ or $v_{i-1}...v_1v_{i+1}...v_nv_iv_{i-1}$ according to the case in step 3.

End Find_Hamiltonian_circuit

At several points of the proof of correctness we will need the following crucial idea. Let us assume that we have a bipartite graph between the sets A and B such that all the nodes in A have degree at least $|A|$. Then each maximal matching covers the whole set A. It is worthwhile to state this statement as a separate lemma.

Lemma 5.1. *Let H be is a bipartite graph between sets A and B. Assume that every node in A has degree at least $|A|$. Then every maximal matching of H covers the whole set A.* ∎

Lemma 5.2. *The procedure Initialize constructs a path-covering of the graph.*

Proof. Let A be the set of nodes not covered by M. We can assume that A is not empty. Let B be the complement of A (i.e., the set of nodes covered by M). M is a maximal matching so A is an independent set. Every point in A has degree at least $\frac{n}{2}$, so the size of B must be at least $\frac{n}{2}$. This implies that the size of A is at most $\frac{n}{2}$. So the bipartite graph between A and B satisfies the condition of lemma 5.1. Therefore the matching constructed in step 2 of procedure Initialization covers the whole set A. This proves the result. ∎

The following lemma shows how the number of paths in the path-cover changes during the procedure Reduce_path_cover.

Lemma 5.3. *Let us run the procedure Reduce_path_cover once. Let p be the number of paths at the beginning. Let $p\prime$ be the number of paths and c be the number of generalized components after executing step 3. Let t be the number of components touched in step 4. Then*
(i) In step 4 after $\log p = \mathcal{O}(\log n)$ iterations we stop with $c \geq \frac{p'}{2}$ generalized components.
(ii) The procedure outputs a path-cover with at most $\frac{3}{4}p$ paths.

Proof. (i) Let a be the number of paths, coming from a connected pair of introverted paths. Then after executing one round of the loop in step 3 these paths will be merged into $\frac{a+1}{2}$ paths (counting a possible unmatched introverted path). If a generalized component is a social path or a pair of independent introverted paths then it remains that way. This proves (i).

(ii) After executing step 4 we have at least $\frac{t}{2}$ fewer paths then $p\prime$ (i.e., the number of paths before step 4). In step 6 for all untouched components we found an edge along which one can merge it into another path. This is true because the auxiliary bipartite graph H_1 we built up has the property of lemma 4.1. (See lemma 3.5). So the number of edges in H_2 is exactly $c - t$. We delete some of them. But this deletion kills only one edge from each cycle. It is easy to see that we still have at least $\frac{c-t}{2}$ edges remaining. The corresponding merging operations will reduce the number of paths by at least $\frac{c-t}{2}$. So the final number of paths is at most

$$p\prime - \frac{t}{2} - \frac{c-t}{2} = p\prime - \frac{c}{2} \leq p\prime - \frac{p\prime}{4} = \frac{3}{4}p\prime \leq \frac{3}{4}p.$$

∎

The previous lemma easily implies the correctness of the algorithm.

Lemma 5.4. *The program Find_Hamiltonian_cycle terminates in polylog steps, uses polynomial number of processors and outputs a Hamiltonian cycle of G.*

Proof. It follows from that step 2 we have a Hamiltonian path of G. The fact that we can also find a Hamiltonian cycle follows from an argument similar to lemma 4.1. The analysis of the number of processors and time is easy for any straightforward implementation of the algorithm. ∎

6. Conclusion and open problems

If the number of nodes is even, by constructing a Hamiltonian path we obtain a perfect matching too. So we have an NC algorithm to find a perfect matching in dense graphs.

The same problem on general graphs is not known to be in NC. As we saw there is a striking difference between the general case and the case of dense graphs in the case of Hamiltonian cycle problem.

In this section we consider the question what happens if we enlarge the class of possible inputs. In both case (matching and Hamiltonian cycle) we obtain 'hardness' results.

Let G be an α-dense graph if the minimal degree of G is at least $\alpha|G|$, where $\alpha < \frac{1}{2}$.

Theorem 6.1. *For $\alpha < \frac{1}{2}$ the existence problem for a perfect matching restricted to α-dense graphs $G = (V, E)$ is NC-hard for the general matching problem. This means that an NC-algorithm for the matching problem restricted to α-dense graphs would imply an algorithm for the general perfect matching problem.*

Proof. Let $G = (V, E)$ be any graph. We construct a graph $G' = (X \dot\cup Y \dot\cup V, E')$ as follows: X and Y are sets of equal size, Y is an independent set in G' and G is an induced subgraph of G'. Every node in X is adjacent to all nodes in G'. We choose the size of X to be $|X| = \lceil \frac{\alpha}{1-2\alpha}|V| \rceil$. ∎

A similar proof technique was used by A. Broder [Br86], when he showed that determining the permanent on "dense" bipartite graphs is $\#P$-hard.

The same reduction works for the Hamiltonian cycle problem. But in this case the difference (NP-completeness and NC) is more striking.

Theorem 6.2. *For every $\alpha < \frac{1}{2}$, the Hamiltonian cycle problem restricted to α-dense graphs is NP-complete.*

Proof. We reduce the existence problem of a Hamiltonian path in a graph G to the existence problem of a Hamiltonian cycle in an α-dense graph G'. We construct $G' = (V', E')$ from a given $G + (V, E)$ as follows:

- the vertex set of G' is a disjoint union of the vertex set of G and two other sets, $V' = V \dot\cup X \dot\cup Y$, where $|X| = |Y| + 1 = k = \lceil \frac{\alpha}{1-2\alpha}(|V| + 1) \rceil$.
- the edge set of G' consists of: (i) the edge set of G, (ii) the complete bipartite graph between V and X, (iii) the complete bipartite graph between X and Y

Each Hamiltonian cycle in G' must have a subpath $x_1 y_1 x_2 y_2 \dots x_{k-1} y_{k-1} x_k$, where $X = \{ x_1, \dots, x_k \}$ and $Y = \{ y_1, \dots, y_{k-1} \}$. The rest of the Hamiltonian path passes through G. Therefore G has a Hamiltonian path if and only if G' has a Hamiltonian cycle. Easy to check that G' is α-dense by setting the size of X and Y as we did. ∎

Finally we mention some open problems.

There are more sufficient conditions for having Hamiltonian cycles ([Be73],[Lo79]). Some of them are weaker than Dirac's condition, in the sense that they define a wider class of graphs. I don't know anything about NC algorithms for finding a Hamiltonian cycle in these classes.

Another relaxation similar to the Hamiltonian path problem is the maximal cycle problem. One can fix some elementary operations for enlarging a cycle. One example is the following. Let us assume that we have a situation in which a node not on the cycle has two neighbors that are consecutive on the cycle. One can easily merge this node into the

cycle. In this case we can define the notion of maximal cycle, i.e., a cycle for which the operation above can not be applied. Is there any NC algorithm which finds a maximal cycle in a graph?

6. GEOMETRY, GRAPHS AND COMPLEXITY

1. Unit-distances between vertices of a convex polygon

This chapter is motivated by the following question of Erdős: What is the maximal multiplicity of a distance in all distances between vertices of a convex n-gon. We can assume that the most common distance among the vertices is the unit-distance.

Let $\mathcal{S}$ be any n element point set on the plane.

$$d(\mathcal{S}) = \max_{d \in R^+} \Big| \{ (P,Q) : P,Q \in \mathcal{S} \text{ and } d(P,Q) = 1 \} \Big|.$$

We call $\mathcal{S}$ a *convex set* if S is the vertex set of a convex polygon.

$$d(n) = \max_{\mathcal{S} \text{ is an } n \text{ element convex set}} d(\mathcal{S}).$$

Our goal is to consider the $d(n)$ function. It is known that

$$cn \log n \geq d(n) \geq \frac{5}{3}n - 3.$$

In this section we are going to give a construction what improves the previous lower bound. To do so we will define a convex point set which contains many unit-distances. First we describe a configuration of eight points with many unit-distances. Finally we blow up our set to get the improved lower bound.

We start with the construction of the eight element set. We take a $PQR\triangle$ from the plane such that $d(P,Q) = d(P,R) = 1 - \epsilon$ and $d(Q,R) = 1 - \delta$ where $0 < \epsilon < \delta$ small parameters. We remark that our assumptions on the order implies that $PQR\triangle$ close to an equilateral triangle and its two angles on the side QR are a little bit greater then $\frac{\pi}{3}$ (Figure 1.1). To make easier the references we introduce some notations. We will refer to the line RQ as horizontal line. This determines the vertical direction too. The RQ line defines two halfplanes. The one which doesn't contain the point P will be called the bottom half plane. This allows us to use the expressions under and above. Let PT be a unit-interval, with T as a lower endpoint. We draw a TRU triangle to the top of the interval TR. Its two unknown sides have length 1.

We do the same with interval TQ. The new point what we get is denoted as V. Finally we rotate the point U around R as a center with angle $\frac{\pi}{3}$ to get U'. The same kind of rotation (around Q with angle $-\frac{\pi}{3}$) maps V to V'.

Figure 1.1.

Now we have our full configuration which is totally determined by the two parameters. We call it $\mathcal{CONF}(\epsilon, \delta) = \{ P, Q, R, T, U, V, U', V' \}$.

Lemma 1.2. *If ϵ and δ are small enough then $\{ P, Q, R, T, U, V, U', V' \}$ spans a convex octagon.*

Proof. If one varies the parameters then all the distances and angles change a continuous way. We are going to study the $\epsilon = 0$ situation (for small ϵ our picture is "close" to this one). This configuration is on the Figure 1.3 (the corresponding points have a 0 index).

Figure 1.3.

$U_0', Q_0, T_0, R_0, V_0'$ are all on the unit-circle drawn around P_0 as a center. Since $U_0'RP\angle = \frac{\pi}{3} < QRP\angle$ the order of the points on the arc is the same as we wrote. So the consecutive triples on this arc determine a convex angle. $U_0U_0'Q_0\angle = V_0V_0'R_0\angle$ and both of them are an angle of a triangle. This implies that they are convex. From this follows that the corresponding angles are convex too in the original configuration if ϵ is too small. When we vary ϵ P_0 splits into three points. It is easy to see that this doesn't cause a problem. The PT vertical line is symmetric axe of the whole picture. If we start from $\mathcal{CONF}(0, \delta)$ and increase ϵ a little bit then UV becomes a short horizontal interval symmetric to the PT line. P will a little bit above this interval. U, P, V are on an arc drawn around T as center. So $UPV\angle$ is a convex angle close to π. The angle between $U_0'U$ and $V_0'V$ lines is about $\frac{\pi}{3}$. This facts prove our lemma. ∎

This configuration contains eight points and determines nine unit-distances. To get our final construction we substitute some points of this set with n-tuples of points. During this substitution we multiply the number of unit-distances and preserve convexity.

Let $\mathcal{C} = \mathcal{CONF}(\epsilon, \delta)$, where ϵ and δ small numbers, such that the conclusion of Lemma 1.2 is true. Now we draw a unit-circle around P. This circle goes through T. We take a length η arc starting at T to both direction. Let $\mathcal{A}_T$ be the union of these two arcs (the length of $\mathcal{A}_Y$ is 2η). We do the same with the unit-circle around R and points U, U_0' and finally with the unit-circle around Q and points V, V_0'. Let the corresponding arcs are denoted as $\mathcal{A}_U, \mathcal{A}_{U'}, \mathcal{A}_V$ and $\mathcal{A}_{V'}$. Let $\mathcal{C}(\eta)$ be the point set that we obtain when we add all these arcs to $\mathcal{C}$ (Figure 1.4.).

Figure 1.4.

Lemma 1.5. *If η is small enough then any finite subset of $\mathcal{C}(\eta)$ is a vertex set of a convex polygon.*

Proof Easy consequence of our previous remarks ∎

We consider $\mathcal{C}(\eta)$ with a small η. Let take an n-element subset $\mathcal{T} = \{ T_1, T_2, \ldots, T_n \}$ of $\mathcal{A}_T$. We choose the T_i's to be very close to T. We draw a unit-circle with center T_i. This circle has a unique intersections with $\mathcal{A}_U$ and $\mathcal{A}_V$. We denote these intersections as U_i and V_i. For different points from $\mathcal{T}$ these intersections are different. This is true because the unit-circles around T_i and T_j have only two common points. One of them is P and the other is on the other side of T_iT_j. So we have $\mathcal{U} = \{ U_1, \ldots, U_n \}$ and $\mathcal{V} = \{ V_1, \ldots, V_n \}$ subsets of $\mathcal{A}_U$ and $\mathcal{A}_V$. Finally we rotate these set with the same rotation which maped U to U' and V to V'. So we get $\mathcal{U}'$ and $\mathcal{V}'$. (Figure 1.6.)

Figure 1.6.

Our set is

$$\mathcal{U} \cup \mathcal{U}' \cup \mathcal{V} \cup \mathcal{V}' \cup \mathcal{T} \cup \{ P, Q, R \}.$$

This set contains $5n+3$ points and the number of unit-distances are $9n$. This shows that $d(5n+3) \geq 9n$. One can easily extend this result to every possible set size. So we have the following theorem.

Theorem 1.7.

$$d(n) \geq \frac{9}{5}n - 13. \qquad \blacksquare$$

This construction was improved with H. Edelsbrunner. The improved construction is published in [EH91].

2. Unit-distances and excluded configurations in matrices

A *configuration*, $C = (c_{ij})$ $(1 \leq i \leq u,\ 1 \leq j \leq v)$, is a partial matrix with 1's and blanks at the entries. All the matrices we are going to work with will be $0-1$ matrices. We say that a matrix $M = (m_{ij})$ does have the configuration C if one can find u rows $i_1, i_2, \ldots, i_u, i_1 < \cdots < i_u$ and v columns $j_1, j_2, \ldots, j_v, j_1 < \cdots < j_v$ in M such that the corresponding submatrix contains C, i.e. $m_{i_\alpha, j_\beta} = 1$ whenever $c_{\alpha,\beta} = 1$. Let $f(n, m; C)$ denote the maximum number of 1's in an $n \times m$ matrix M not containing C. In the case of $n = m$ we write $f(n; C)$. One can allow several forbidden configurations, the corresponding threshold function is $f(n, m; \{C^1, \ldots, C^n\})$ or $f(n; \{C^1, \ldots, C^n\})$.

Our research on these threshold functions motivated by giving upper bound on g(n).

Let $\mathcal{C}$ be a closed convex curve. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two disjoint arcs of $\mathcal{C}$. Let $P_1, P_2, \ldots, P_k$ be a sequence of points of $\mathcal{A}_1$ (in the same order as they occur on $\mathcal{A}_1$). Similarly $Q_1, \ldots, Q_l$ are points from $\mathcal{A}_2$ Let $M(\{ P_1, \ldots, P_k, Q_1, \ldots, Q_l \}) = M = (m_{i,j})$ be the following matrix of size $k \times l$. $m_{i,j} = 1$ if $d(P_i, Q_j) = 1$, otherwise $m_{i,j} = 0$. Hence the unit-distances of the form $d(P_i, Q_j)$ is the number of 1's in M.

The following Lemma defines a forbidden configuration for the distance matrix defined above.

Lemma 2.1. *Let $\mathcal{A}_1, \mathcal{A}_2, P_1, P_2, P_3, Q_1, Q_2, Q_3$ as above (see figure 2.2). It is impossible, that $d(P_1, Q_1) = d(P_1, Q_3) = d(P_2, Q_3) = d(P_3, Q_2) = d(P_3, Q_3) = 1$.*

Proof. Let us assume that all the distances mentioned in the lemma are unit-distances.

Figure 2.2.

We are going to prove that in this case all angles of the quadrangle $P_1P_3Q_3Q_1$ are acute. $P_1P_3Q_3\angle$ is an angle in an isosceles triangle, hence it is acute. $Q_1P_1Q_3\angle < Q_1P_1P_2\angle$. $Q_1P_1P_2\angle$ is an angle in an isosceles triangle, hence $Q_1P_1Q_3$ is acute. The other two angles are acute by the same argument. ∎

Using these ideas Z. Fu;redi ([Fü]) could prove a $c \cdot n \log n$ upper bound an the $g(n)$ function. The main step in the proof is determining the order of magnitude of $f(n, m; \begin{pmatrix} 1 & 1 & \\ 1 & & 1 \end{pmatrix})$ threshold function.

Other configuration was considered in [BGy]. The analysis of a computational geometrical algorithm led to their problem.

Our research is closely related to previous works in combinatorics.

Let us mention Turán's theory in extremal graph theory. There the question is: Given a graph G, what is $T(n; G)$, the maximum number of edges of a graph with n vertices and not containing G as a subgraph? A special case is when we work in the universe of bipartite graphs. Our matrices can be considered as bipartite graphs. The important difference between Turán's theory and our question that in our case the vertices (the rows and columns) are ordered. This is a very important difference but in some special case the restriction on the order is insignificant. An example is the four cycle (complete bipartite graph between two color classes of size 2 each). Classical results in graph theory [KST54], [ERS66], [Bo78] immediately give us the following theorem.

Theorem 2.3.

$$f(n; \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}) = \Theta(n^{\frac{3}{2}}). \quad \blacksquare$$

We do not know exactly how these two problems are related, but the following facts are known. The Erdős-Stone-Simonovits theorem ([ESi66], [ESt46], for a survey see Bollobás' book [Bo78]) says that the order of magnitude of $T(n; G)$ depends on the chromatic number of G, namely $\lim_{n\to\infty} \frac{T(n;G)}{\binom{n}{2}} = 1 - (\chi(G) - 1)^{-1}$. This theorem gives sharp estimate on $T(n; G)$, except for bipartite G. For every bipartite graph B which is not a tree there are positive constants c_1 and c_2 (not depending on n) such that

$$\Omega(n^{1+c_1}) \leq T(n; B) \leq O(n^{2-c_2})$$

holds. If the graph is a tree F, then it is straightforward that $T(n;F) = \Theta(n)$. However we will see that our problem has completely different threshold functions. For a special matrix (such that the corresponding graph is a tree, hence it has linear Turán function) our threshold function turns out to be $\Theta(n \log n)$.

An other related question is raised by Davenport and Schinzel. A sequence $s = x_1 x_2 \dots x_l$ is called a *Davenport-Schinzel sequence*, $s \in DS_k(n)$, if $x_i \neq x_{i+1}$, $x_i \in \{1, 2, \dots, n\}$ and s does not contain a subsequence $x_{i_1} x_{i_2} \dots x_{i_k}$ such that

$$x_{i_1} = x_{i_3} = \dots = x_{i_{2t-1}} = \dots \neq x_{i_2} = x_{i_4} = \dots = x_{i_{2t}} = \dots$$

$(i_1 < i_2 < \dots < i_k)$. Let $ds_k(n)$ denote the maximum length of an $s \in DS_k(n)$. It is obvious that

$$ds_3(n) = n, \qquad ds_4(n) = 2n - 1.$$

Szemerédi [Sz74] proved that $ds_k(n) = O(n \log^*(n))$ for all fixed k while n tends to infinity. (Here, as usual, $\log^* n$ denotes the inverse of the function $p: \mathbf{N} \to \mathbf{N}$ with $p(1) = 2$, $p(n+1) = 2^{p(n)}$.) Recently, mainly due to the works of M. Sharir ([Sh87], [HS86], [Ko88]) it is known that the true order of the magnitude of $ds_k(n)$ for $k \geq 5$ is really superlinear, e.g. (Hart and Sharir [HS86])

$$ds_5(n) = \Theta(n\alpha(n)),$$

where $\alpha(n)$ is the inverse Ackermann function, a very slowly growing function. More on this see in Section 8 and 9.

For a matrix M (or vector as a special case) $||M||$ denotes the number of its entries equal to 1, M^T is its transpose. $[n]$ is the set of the first n positive integers, and $[a, b] =: \{a, a+1, \dots, b\}$.

3. A reduction between matrices

Let C be a configuration of 1's. We are going to define two operations on C. The first one is simply deleting an entry. The second one is attaching a new column or row to the boundary of C and placing an entry 1 in the new column or row, next to an existing one in C.

Definition 3.1. *If D can be constructed from C using one of these operations we say that D is obtained by an elementary operation from C. We use the notation $C \stackrel{e}{\longrightarrow} D$. Let $\longrightarrow$ be the transitive closure of $\stackrel{e}{\longrightarrow}$, i.e. $C \longrightarrow D$ if D can be constructed from C using a sequence of elemantary operations.*

Note that the size of the matrix can decrease by the first type of elementary operation if the deletion of the given entry creates an empty row or column.

Figure 3.2 shows several configurations and their relations.

$$C_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\downarrow$$

$$C_2 = \begin{pmatrix} 1 & 1 & \\ 1 & & 1 \end{pmatrix}$$

$$\swarrow \qquad \downarrow \qquad \searrow$$

$$C_3 = \begin{pmatrix} & 1 & \\ 1 & & \\ 1 & & 1 \end{pmatrix} \quad C_4 = \begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix} \quad C_5 = \begin{pmatrix} 1 & & \\ & 1 & \\ 1 & & 1 \end{pmatrix}$$

$$C_6 = \begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}$$

$$\swarrow \searrow$$

$$C_7 = \begin{pmatrix} & & 1 & \\ 1 & & & \\ & 1 & & 1 \end{pmatrix} \quad C_8 = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & 1 \end{pmatrix}$$

$$\downarrow \qquad \downarrow$$

$$C_9 = \begin{pmatrix} & & 1 & \\ 1 & & & \\ & & & 1 \\ & 1 & & \end{pmatrix} \quad C_{10} = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{pmatrix}$$

Figure 3.2.

Theorem 3.3. *Let C, D be configurations such that $C \longrightarrow D$ by t elementary steps. Then $f(n, m; D) \leq f(n, m; C) + t \cdot \max(n, m)$.*

Proof. It is sufficient to prove the case $t = 1$, we can assume that $C \xrightarrow{e} D$. If D is constructed by deleting an entry then the claim is obvious. So we can assume that D is constructed by adding an extra column to the end of C with an extra 1 (the other cases are very similar). Let M be a matrix of size $n \times m$ with $f(n, m; D)$ many 1's such that it doesn't have D as a subconfiguration. Let M' be the matrix that we get if we delete the last 1 in each row (assuming that there is any). Easy to realize that M' doesn't have C as a subconfiguration. So the number of remainder 1's in M' is at most $f(n, m; C)$. ∎

The natural way to apply Theorem 3.3 is that in the case of $C \longrightarrow D$ an upper bound on $f(n;C)$ gives an upper bound on $f(n;D)$ and a construction for a matrix not having D as a submatrix gives a good construction for C.

Figure 3.4 contains some additional matrices with four 1's and some of their $\longrightarrow$ relations.

$$C_{11} = \begin{pmatrix} 1 & & & 1 \\ & 1 & 1 & \end{pmatrix}$$

$$\swarrow \quad \searrow$$

$$C_{12} = \begin{pmatrix} 1 & & & 1 \\ & 1 & & \\ & & 1 & \end{pmatrix} \quad C_{13} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & 1 & 1 & \end{pmatrix}$$

$$\downarrow \qquad\qquad \downarrow$$

$$C_{14} = \begin{pmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \end{pmatrix} \quad C_{15} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & 1 & & \\ & & 1 & \end{pmatrix}$$

Figure 3.4.

Let B_2 be $(1,1)$, a 1×2 configuration.

Proposition 3.5. *If $B_2 \to C$ and C has at least 2 entries in it then*

$$\min(n,m) \leq f(n,m;C) \leq c_C(n+m).$$

Proof. Trivial. The lower bound comes considering a matrix M with 1's only in one row or in one column.

The upper bound is immediate from Theorem 3.3. ∎

$$C_{16} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}, C_{17} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}, C_{18} = \begin{pmatrix} & 1 & & \\ 1 & & & \\ & & & 1 \\ & & 1 & \end{pmatrix},$$

$$C_{19} = \begin{pmatrix} & 1 & & \\ 1 & & & 1 \\ & & 1 & \\ & & & \end{pmatrix}, C_{20} = \begin{pmatrix} 1 & 1 & & \\ & & 1 & \\ & & & 1 \\ & & & \end{pmatrix}, C_{21} = \begin{pmatrix} 1 & 1 & & \\ & & & 1 \\ & & 1 & \\ & & & \end{pmatrix},$$

$$C_{22} = \begin{pmatrix} 1 & & & \\ & 1 & 1 & \\ & & & 1 \\ & & & \end{pmatrix}, C_{23} = \begin{pmatrix} 1 & & & \\ & & 1 & 1 \\ & 1 & & \\ & & & \end{pmatrix}, C_{24} = \begin{pmatrix} 1 & & & \\ & 1 & & 1 \\ & & 1 & \\ & & & \end{pmatrix},$$

$$C_{25} = \begin{pmatrix} 1 & 1 & \\ & 1 & \\ & & 1 \end{pmatrix}, C_{26} = \begin{pmatrix} 1 & 1 & \\ & & 1 \\ & 1 & \end{pmatrix}, C_{27} = \begin{pmatrix} 1 & 1 & \\ & & 1 \\ & & 1 \end{pmatrix},$$

$$C_{28} = \begin{pmatrix} 1 & & 1 \\ & 1 & \\ & 1 & \end{pmatrix}, C_{29} = \begin{pmatrix} 1 & & \\ & 1 & 1 \\ & 1 & \end{pmatrix}, C_{30} = \begin{pmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{pmatrix},$$

$$C_{31} = \begin{pmatrix} 1 & 1 & 1 & \\ & & & 1 \end{pmatrix}, C_{32} = \begin{pmatrix} 1 & 1 & & \\ & & 1 & 1 \end{pmatrix}, C_{33} = \begin{pmatrix} 1 & & 1 & 1 \\ & 1 & & \end{pmatrix},$$

$$C_{34} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & & \end{pmatrix}, C_{35} = \begin{pmatrix} 1 & 1 & 1 \\ & 1 & \end{pmatrix}, C_{36} = \begin{pmatrix} 1 & 1 & \\ & 1 & 1 \end{pmatrix},$$

$$C_{37} = (1 \quad 1 \quad 1 \quad 1).$$

Figure 3.6.

We remark that Figure 3.2, Figure 3.4 and Figure 3.6 contain all the 37 configurations with four 1's (not distinguising two if they are the same upto rotations and reflections). The simple reduction principle yields that 22 of them have linear complexity.

Corollary 3.7.
(a) If M has at most 3 non-zero entries then $f(n,m;M) \leq 2(n+m)$.
(b) The 22 matrices on Figure 3.6 have linear complexity, $f(n,m;C_i) \leq 3(n+m)$ for $16 \leq i \leq 37$. ∎

One can extend the $\longrightarrow$ relations to sets of configurations. This will be proven very useful.

Definition 3.8. *Let $C^1, \ldots, C^k$ be a set of configurations. We are going to define two operations. One is simply adding a new configuration to our set. The second is substitute a C^i with D if $C^i \longrightarrow D$. The transitive closure of these relations is $\longrightarrow$.*

The notation is not in conflict with Definition 3.1, which is a special case of this. Note that $\{C^1, \ldots, C^k\} \longrightarrow \{D^1, \ldots, D^l\}$ iff for every i there is a j such that $C^i \longrightarrow D^j$ according to Definition 3.3.

The analog of Theorem 3.3 is the following.

Theorem 3.9. *If $\{C^1, \ldots, C^k\} \longrightarrow \{D^1, \ldots, D^l\}$ then*

$$f(n,m;\{D^1, \ldots, D^l\}) \leq f(n,m;\{C^1, \ldots, C^k\}) + const(n+m),$$

where the constant depends only on the two systems, and not on n and m. ∎

A few examples:

$$\{\begin{pmatrix} 1 & 1 & \\ 1 & & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ & 1 \\ 1 & \end{pmatrix}, \begin{pmatrix} 1 & & 1 \\ & 1 & 1 \end{pmatrix}, \begin{pmatrix} & 1 \\ 1 & \\ 1 & 1 \end{pmatrix}\} \longrightarrow \begin{pmatrix} & 1 & & \\ & & & 1 \\ 1 & & & \\ & & 1 & \end{pmatrix},$$

$$\{\begin{pmatrix} 1 & 1 & \\ 1 & & 1 \\ & & \end{pmatrix}, \begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}\} \longrightarrow \begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}.$$

4. Matrices with $n \log n$ complexity

Theorem 4.1. ([Fü]) $f(n, \begin{pmatrix} 1 & 1 & \\ 1 & & 1 \end{pmatrix}) < 6n \log n.$ ∎

The construction in [Fü] shows that this upper bound is the best up to a constant factor. Below we give another, a simpler recursive construction.

Construction 4.2. *Let*

$$A_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix},$$

and

$$A_{n+1} = \begin{pmatrix} E_{2^n} & A_n \\ A_n & 0_{2^n} \end{pmatrix},$$

where E_n is an $n \times n$ matrix with 1's only in the diagonal connecting the upper right to the lower left corner, and 0_n the $n \times n$ zero matrix.

Claim 4.3.
(1) A_n is a $2^n \times 2^n$ matrix with $(n+2)2^{n-1}$ many 1's.
(2) A_n does not have $C_4 = \begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}$ as a subconfiguration.

Proof. (1) Easy induction.

(2) Using induction. The initial case is obvious. Let us assume that the claim is verified for A_k, when $k < n$.

Suppose on the contrary that A_n has the forbidden configuration. A_n is, by definition, divided into 4 submatrices. We distinguish different cases depending on which submatrix has the upper left corner of the forbidden configuration. If one of the A_{n-1}'s is the one, then our inductional hypothesis gives the contradiction. If $E_{2^{n-1}}$ has that entry then easy to verify that the bottom right corner of the configuration must be in $0_{2^{n-1}}$. This contradicts the fact that $0_{2^{n-1}}$ has no 1 entry at all. ∎

Corollary 4.4.
(1) $f(n;C_2), \quad f(n;\{C_2,C_2^T\}), \quad f(n;C_4)=\Theta(n\log n)$.
(2) $f(n;C_i)<10n\log n$, *for* $4\le i\le 15$.

Proof. (1) Both the lower and upper bound comes from the following relations. $C_2 \longrightarrow \{C_2,C_2^T\} \longrightarrow C_4$.
(2) See Figure 2.2 and Figure 2.4. ∎

5. A construction with $\frac{n\log n}{\log\log n}$ 1's

In the previous section we saw an $n\log n$ upper bound on $f(C_5)$. Now we construct a matrix with $\Theta(\frac{n\log n}{\log\log n})$ 1's and not having C_5 as a subconfiguration. This section is a slightly simplified version of [BGy]. Our construction will be recursive and it defines $N(s,t)$, a matrix of size $st\times st$, where $s,t\ge 1$.

First we discuss a few properties of $N(s,t)$ what we need for the formal definition of the matrix. The st rows are divided into s blocks, each having t consecutive rows. In each block we have a column such that each of its entries are 1's and these are the first 1's in the corresponding rows. This column is the *leading column* of that block.

Let $N(s)$ be a $s\times s$ matrix without the configurations:

$$C_5=\begin{pmatrix}1 & & 1\\ & 1 & \\ 1 & & \end{pmatrix},\begin{pmatrix}1 & & 1\\ 1 & 1 & \end{pmatrix},\begin{pmatrix}1 & 1\\ & 1\\ 1 & \end{pmatrix},\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}. \tag{5.1}$$

Definition 5.2. *$N(1,t)$ is a $t\times t$ matrix with t 1's in the first column and 0's everywhere else.*

$N(s,1)$ is the $s\times s$ identity matrix.

The construction of $N(s,t+1)$ is the following (we assume that $N(s,t')$ and $N(s',t'')$ are already constructed for $t'\le t$, $s'<s$ and t'' arbitrary). Take a copy of $N(s,t)$ and insert an extra row after each block. In each extra row put a 1 at the leading column of the block just above it. Add s new columns at the end of the already constructed part. At the intersection of extra rows and new columns we have an $s\times s$ space. Put a copy of N(s) with maximum number of ones.

The promised propeties are maintained so our recursion is correct.

Theorem 5.3. *$N(s,t)$ doesn't have the configurations given in (5.1).*

Proof. An easy induction by case by case check. ∎

The previous theorem gives lower bounds on the complexity of several configurations and sets of configurations.

Corollary 5.4. *([BGy])* $f(n;C_5)=\Omega(\frac{n\log n}{\log\log n})$.

Proof. Let $f(s,t)=||N(s,t)||$ and $f(s)=\max||N(s)||$. We have

$$f(s,t+1)\geq f(s,t)+f(s)+s,$$

and for $s\geq ab$

$$f(s)\geq f(a,b).$$

These inequalities imply that

$$f(l^a,t)\geq(t-1)l(l^a+l^{a-1}-(l-1)^a)+l^a,$$

especially

$$f(l^{a+1})\geq f(l^a,l)\geq l^{a+2}-l(l-1)^{a+1}.$$

Letting $a=l-1$, $n=l^l$ we obtain the desired bound. ∎

6. More matrices with linear complexity

Recall that $C_{11}=:\begin{pmatrix}1&&&1\\&1&1&\end{pmatrix}$. In this section we prove, that the complexities of $C_{11},\ldots,C_{15}$ are all linear, at most $9n$. As one can see from Fig. 2.4, and Theorem 2.3 the above result implied by the following theorem.

Theorem 6.1. $f(n,C_{11})\leq 7n$.

Proof. Let $A'=(a'_{ij})$ be an $n\times m$ 0–1 matrix without C_{11}. Delete the first and the last entry in each row, and delete all entries in that row if $||(a'_{ij})_{1\leq j\leq m}||\leq 3$. For the obtained matrix $A=(a_{ij})$ we have

$$||A'||\leq||A||+3n. \tag{6.2}$$

A does not contain the following configurations either:

$$\begin{pmatrix}1&&1\\&1&1\end{pmatrix},\quad\begin{pmatrix}1&1\\1&1\end{pmatrix},\quad\begin{pmatrix}1&&1\\1&1&\end{pmatrix}.$$

For the i'th row (if it is non-empty) let $m(i)$ ($M(i)$) denote the minimum (maximum, resp.) index in that row, i.e. $m(i):=\min\{j:a_{ij}=1\}$. Then $[m(i),M(i)]\subset[m(i'),M(i')]$ implies $i\leq i'$.

The element a_{ij} is called *type* α if $a_{ij} = 1$, it is not the first neither the last one in its row ($m(i) < j < M(i)$), $\alpha < i$, $j \in [m(\alpha), M(\alpha)]$, and i is minimal with respect to these constraints. By definition, there are no two entries of type α in distinct rows. But there are no two 1's of type α in the i'th row either, otherwise together with $a_{\alpha,m(\alpha)}$ and $a_{\alpha,M(\alpha)}$ they form a forbidden subconfiguration. So the number of entries in A which are

(1) first or last in their row,
(2) on the top of their column, or
(3) have a type α is at most $4n$. We claim that all the entries of A fall one of the above 3 categories, implying $||A|| \leq 3n + m$. Then (6.2) finishes the proof of the Theorem.

Suppose that the entry $a_{ij} = 1$ is not the first or the last one in the i'th row, and that there exists a $t \leq i$ with $a_{tj} = 1$. Then $j \in [m(t), M(t)]$. Let α be the maximum index, such that $\alpha < j$, and $j \in [m(\alpha), M(\alpha)]$. Then a_{ij} has type α.

Indeed, suppose on the contrary, that some entry $a_{i'j'}$ has type α, with $\alpha < i' < i$. Then, $j \in [m(\alpha), M(\alpha)] \subset [m(i'), M(i')]$, so the existence of i' contradicts the definition of α. ∎

Let C^t be a $2 \times (t+2)$ configuration with 1's in the positions $(1,1), (1,t+2)$ and $(2,2), \ldots, (2,t+1)$. $C_{11} = C^2$. Deleting from every row the *middle* $t-2$ entries, Theorem 6.1 implies

Corollary 6.3. $f(n; \begin{pmatrix} 1 & & & \ldots & & & 1 \\ & 1 & 1 & \ldots & 1 & 1 & \end{pmatrix}) = f(n; C^t) \leq (t+5)n$. ∎

Finally we mention a generalization of this idea in the direction of sequences with forbidden subsequences. The following corollary is a special case of the result in [AKV].

Corollary 6.4. *Suppose that the seqence $s = x_1, x_2, \ldots, x_l$ with $x_i \in [n]$, does not have two identical consecutive members, and does not contain the subsequence abba, where $a < b$, then $l \leq 100n$.*

Proof. (*sketch*) Split s into n equal parts $s = s_1 s_2 \ldots s_n$, $||s_i|| = 100$. Then there is a subset $s_i' \subset s_i$ containing only distinct elements with $|s_i'| \geq 9$. Put 1's into the i'th column of an $n \times n$ matrix A according to s_i'. Finally, apply 6.1 to A to get a C_{11}, and then to get an *abba* in s. ∎

7. A covering lemma

In this section we prove a covering lemma about $0-1$ matrices. As an easy application of our lemma we get several new matrices with linear complexity.

We start with a definition. An intersection of s consecutive rows and t consecutive columns is called a *rectangle*. The horizontal size of R is t and it is denoted by $h(R)$, the vertical size of R is s and it is denoted by $v(R)$. M, itself is an example for a rectangle.

Lemma 7.1. *Let M be arbitrary $0-1$ matrix. Then there is a system of rectangles $\{R_i\}$ such that*

(1) R_i's cover all the 1's,

(2) $\sum_i h(R_i) \leq 4h(M)$ and $\sum_i v(R_i) \leq 4v(M)$,

(3) each R_i has a 1 in the upper left or bottom right corner.

Proof. Let us define a partial order between the positions in a given matrix. We say that $a \leq b$, if the row of a is not later than b's one and a's column is not later than b's one. $a \nwarrow b$ if $a \leq b$ and $a \neq b$.

There are incomparable positions. For two incomparable positions c and d we say that $c \nearrow d$ if c's row is earlier than d's.

Take M and consider only the positions where we have a 1. Let $m_1 \nearrow m_2 \nearrow \ldots \nearrow m_k$ be the set of minimal 1's for the partial order $\nwarrow$. Let $M_1 \nearrow M_2 \nearrow \ldots \nearrow M_l$ be the set of maximal 1's for the partial order $\nwarrow$. We can assume that m_1 is in the first column, m_k is in the first row, M_1 is in the last row and M_l is in the last column of M.

Let $m_{i+\frac{1}{2}}$ (for $i = 1, \ldots, k-1$) be the position in the intersection of the row of m_i and the column of m_{i+1}. Let $m_{\frac{1}{2}}$ be the lower left corner of M. Let $m_{k+\frac{1}{2}}$ be the upper right corner of M. Let $M_{j+\frac{1}{2}}$ (for $j = 1, \ldots, l-1$) be the position in the intersection of the column of M_i and the row of M_{i+1}. Let $M_{\frac{1}{2}} = m_{\frac{1}{2}}$ and $M_{l+\frac{1}{2}} = m_{k+\frac{1}{2}}$. Let $h_i = [m_i, m_{i+\frac{1}{2}}]$ be a horizontal interval of positions in the row of m_i, with endpoints at m_i and $m_{i+\frac{1}{2}}$. Let v_i be the vertical interval $[m_{i-\frac{1}{2}}, m_i]$. We define the corresponding intervals for maximal 1's. Let $V_i = [M_i, M_{i+\frac{1}{2}}]$ and $H_i = [M_{i-\frac{1}{2}}, M_i]$. It is clear that $v_1, h_1, v_2, ..., v_k, h_k$ and $H_1, V_1, H_2, V_2, ..., H_l, V_l$ defines two stair shaped curves. Let us denote them by s and S. By definition it is straightforward that there are no 1 above s and below S.

Now we are starting to construct our covering system of rectangles. This system is containing two sequences of rectangeles: $\{Q_i\}$ and $\{P_i\}$. The Q_i's are going to have an entry 1 at the bottom left corner, the P_i's are going to have a 1 at the upper left corner. We define them recursively.

Let Q_1 be a rectangle with lower right corner at M_1, with lower left corner at $m_{\frac{1}{2}}$. So its right vertical side is on the vertical half line starting at M_1, going up. The missing corner of Q_1 on this line is where it first hits s.

Q_1 might cover several h_i intervals. Let h_i the first one which is not covered by Q_1. Let P_1 be a rectangle with upper left corner at m_i. This fact gives us two half lines starting at m_i and going down and to right. They hit S at two positions. They will be two other corners of Q_1.

Next, we will explain the general step in the definition.

Let us assume that we already defined $Q_1, P_1, \ldots, Q_i, P_i$. Let V_j be the first vertical interval of S which is not covered by $Q_1 \cup \ldots \cup P_i$. Let M_j be bottom right corner of Q_{i+1}. That defines two half lines starting at M_j, one going up (let us say e_{i+1}) and one going to the left. They hit s at two positions. They give us two other corner of Q_{i+1}. This completes the definition of Q_{i+1}.

Let us assume that we already defined $Q_1, P_1, \ldots, Q_i, P_i, Q_{i+1}$. Let h_j be the first horizontal interval of s which is not covered by $Q_1 \cup \ldots \cup P_i \cup Q_{i+1}$. Let m_j be upper left corner of P_{i+1}. That defines two half lines starting at m_j, one going down and one going

to the right (f_{i+1}). They hit S at two positions. They give us two other corner of P_{i+1}. This completes the definition of P_{i+1}.

The procedure stops when the already constructed rectangles cover all the V_j's (or all the h_j's).

Now we prove that the constructed system of rectangles satisfy (1)-(3).

(3) is immediate.

In order to prove (1) we need a few remarks.

It is immediate from the definition that as i is increasing the lines, e_i's are moving to the left and the lines f_i's are moving up.

The definition also implies that the upper left corner of P_i is on e_i or is left from e_i. Similarly the lower right corner of Q_{i+1} is on f_i or is below f_i. This guarantes that $Q_1 \cup \ldots \cup P_i \cup Q_{i+1}$ covers everything left from e_{i+1} in the region between s and S. Similarly $Q_1 \cup \ldots \cup P_i \cup Q_{i+1} \cup P_{i+1}$ covers everything below f_{i+1} in the region s and S. This proves (1).

For (2): From the definition the top side of Q_i (and this way the whole rectangle) is not above f_i. The lower right corner of Q_{i+1} (let us say M_j) is not above f_i, but it is the last maximal 1 with this property. This guarantees that Q_{i+2}'s lower right corner (and this way the whole rectangle) is above f_i. So the rows of Q_i and Q_{i+2} are completely disjoint. One gets the corresponding statements for the columns and for the P_i's similarly. (2) is an easy consequence of this.

This completes the proof. ∎

Corollary 7.2.

(1) $f(n,m;C_{10})$ *is linear.*

(2) $f(n,m;\begin{pmatrix} 1 & & & & \\ & & & 1 & \\ & & 1 & & \\ & 1 & & & \\ & & & & 1 \end{pmatrix})$ *is linear.*

(3) $f(n,m;\{\begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}, \begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & 1 \end{pmatrix}\})$ *is linear.*

Proof. (1) Take the cover guaranted by Theorem 7.1. Count the 1's separately in different covering rectangles. We know that in the upper leftcorner or in the lower right corner there is a 1. So we can bound the number of 1's using that $f(n,m;\begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & \end{pmatrix})$ and $f(n,m;\begin{pmatrix} & 1 & \\ 1 & & \\ & & 1 \end{pmatrix})$ are linear. If we add up these bounds we obtain the claim in (1).

The same proof works for (2), but there we use Theorem 5.1.

(3) follows the same way. ∎

8. Davenport-Schinzel matrices

In this and the next section we consider the complexity of $C_6 = \begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}$.

Definition 8.1. *A matrix M is called Davenport-Schinzel matrix if it does not have C_6 as a subconfiguration.*

The naming is based on the analogy between this kind of matrices and Davenport-Schinzel sequences (see [DS65]).

The main result in this section is to construct a Davenport-Schinzel matrix with $\Omega(n\alpha(n))$ many 1's. Finally we discuss other configurations, missing from our matrix.

Our construction is very similar to known constructions of Davenport-Schinzel sequences (see [HS86],[Wi86]). We use the same double induction. But instead of sequences we work with matrices.

The matrices we are constructing have two parameters s and t. We refer to them as $M(s,t)$. First we describe a few properties of $M(s,t)$. The recursive definition of these matrices is assuming these properties so we need to maintain them.

(a) The size of the matrix is $tC(s,t) \times tC(s,t)$, where $C(s,t)$ is defined as follows. $C(s,t) = C(s,t-1)C(s-1,C(s,t-1))$ and $C(1,s) = 1$ and $C(s,1) = 2$, for $s > 1$.

(b) The $tC(s,t)$ many rows are divided into blocks. We will refer to them as horizontal blocks. One block contains t rows (hence we have $C(s,t)$ many blocks). Let H_i be the set of the $((i-1)t+1)^{\text{st}}$,...,$(it)^{\text{th}}$ rows, i.e. the i^{th} horizontal block.

(c) Inside H_i the appearance of the first 1 happens in the same column (considering different rows). Let us say this is the $(c_i)^{\text{th}}$ column. The 1's in these columns are called *leading 1's.*

(d) $1 = c_1 < c_2 < c_3 < \ldots < c_{C(s,t)}$. These columns divide the matrix into vertical blocks. Let V_i be the set of columns from the $(c_i)^{\text{th}}$, through $(c_{i+1}-1)^{\text{st}}$, i.e. the i^{th} vertical block.

The definition of $M(s,t)$ is going to use the matrices $S = M(s,t-1)$ and $B = M(s-1,C(s,t-1))$. (Think about S as a small matrix and about B as a big matrix.) B has $C(s-1,C(s,t-1))$ many horizontal blocks of size $C(s,t-1)$. B has $C(s-1,C(s,t-1))$ many vertical blocks too. Let v_i be the number of columns contained in the i^{th} one. S has $C(s,t-1)$ many blocks (one for each row in a block of B).

The following definition assumes properties (a)-(d). (So one must check that these properties are maintained.)

Definition 8.2. *$M(1,s)$ is an identity matrix of size $s \times s$. $M(s,1)$ is $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ (for $s > 1$).*

In order to define $M(s,t)$ take $C(s-1,C(s,t-1))$ many copies of S (one for each horizontal block of B). The construction of M will be completed in $C(s-1,C(s,t-1))$ many stages. In the i^{th} stage we add $(t-1)C(s,t-1) + C(s,t-1)$ many new rows and

$(t-1)C(s,t-1)+v_i$ *many new columns to the part already built. The construction starts with the empty matrix. The general* (i^{th}) *stage is the following.*

(1) We put $(t-1)C(s,t-1)$ *many new rows and new columns after the already existing ones. In the intersection of the new rows and columns we place a copy of* S*.*

(2) We insert an extra row after each horizontal block of the new copy of S*. In these extra rows we place one extra 1, under each leading column.*

(3) Finally we add v_i *new columns (after the old ones). In the new space we place a copy of the* i^{th} *vertical block of* B *using the extra rows.*

The constructed matrix $M = M(s,t)$ has properties (a)-(d).

Let us introduce a few notations. *Ordinary rows* and *ordinary columns* are the rows and columns introduced in step (1). *Extra rows* are the rows introduced in step (2). *Extra columns* are the ones introduced in step (3). The 1's introduced in step (1) are the *ordinary 1's*. The 1 entries introduced in step (2) are called the *extra 1's*. The 1's introduced in step (3) are the *new 1's*.

The previous notations give a partition of 1's into new, ordinary and extra 1's. There are similar partitions for rows and columns.

Any extra 1 is in an ordinary column and in an extra row.

The next lemma summarizes a few simple statements about the matrix $M(s,t)$.

Lemma 8.3.

(1) If s *and* t *are chosen appropriately and* $n = sC(s,t)$ *then* $M(s,t)$ *is an* $n \times n$ *matrix with* $n\alpha(n)$ *many 1's.*

(2) The $(c_i)^{th}$ *column contains 1's inside* H_i *and no other 1's.*

(3) Inside H_i*, after the leading column the 1's are decreasing, i.e. if* k *and* l *are two 1's in the same horizontal block and they are not leading 1's then* $k \nwarrow l$ *or* $l \nwarrow k$*. (Recall that* $q \nwarrow p$ *vaguely means that* p *is south, east or south-east direction from* q*.)*

(4) If l *is a new 1 and* k *is a 1 such that* $l \nearrow k$ *then* k *is a new 1 too. (Recall that* $q \nearrow p$ *vaguely means* p *is north, east or north-east direction from* q*.)*

(5) If l *is an ordinary 1 and* k *is a 1 in* l*'s column or in* l*'s row then* k *is an ordinary 1 in the same horizontal block with the one exception when* l *is a leading 1 and* k *is the extra 1 in its column.*

(6) If l *is an extra 1 or an ordinary 1 and* k *is an ordinary 1 such that* $l \nearrow k$ *then* l *and* k *is in the same horizontal block.*

Proof. For (1) we refer the reader to [HS86] or [Wi86].

The proof of (2)-(6) is easy induction following the definition of $M(s,t)$. ∎

Now we are ready to discuss the missing configurations in $M(s,t)$.

Theorem 8.4. $M(s,t)$ *does not have the following configurations: (i)* $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$*, (ii)* $\begin{pmatrix} & 1 & 1 \\ 1 & & 1 \end{pmatrix}$*, (iii)* $\begin{pmatrix} 1 & & \\ & & 1 \\ 1 & 1 & \end{pmatrix}$*, (iv)* $\begin{pmatrix} 1 & & & 1 \\ 1 & 1 & & \end{pmatrix}$*, (v)* $\begin{pmatrix} 1 & 1 \\ & 1 \\ 1 & \end{pmatrix}$*, (vi)* $\begin{pmatrix} & 1 & & 1 \\ 1 & & 1 & \end{pmatrix}$*, (vii)* $\begin{pmatrix} 1 & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}$*, (viii)* $\begin{pmatrix} & 1 & & 1 \\ & & 1 & \\ 1 & & & \end{pmatrix}$.

Proof. Each configuration in the statement has four 1's in it. Let us order these 1's. A 1 is earlier than an other if its row is earlier or if they are in the same row and it is left from the other. In the case of each configuration name the four 1's as a, b, c and d following the previously defined order.

Our proof is by induction following the definition of $M(s,t)$. The initial case is $s=1$ or $t=1$. Then the statement is clear.

The induction step is proved by contradiction. Let us assume that in $M(s,t)$ we can find four different 1's: the image of a, b, c and d, such that they obey the configuration. The individual configurations are considered separately.

(i) We distinguish cases depending on what kind of entry corresponds to c. Now on we don't distinguish a, b, c, d and their images.

Case 1: c is an extra 1. Then d is a new 1. a is in a leading column but it is not an extra 1. So a's row is an ordinary row. On the other hand $c \nearrow b$, hence (by 8.3.(4)) b is a new 1. So b's row (what is the same as a's row) is an extra row. Contradiction.

Case 2: c is a new 1. Using 8.3.(4) the whole configuration consists of new 1's. So it can be recognized inside $M(s-1, C(s,t-1))$. Contradiction with the inductional hypothesis.

Case 3: c is an ordinary 1. Using 8.3.(5) the whole configuration consists of ordinary 1's from the same horizontal block. So our configuration can be recognized in a copy of $M(s,t-1)$.

(ii) *Case 1:* c is an extra 1. Then a, b and d are new 1's. Let c' the first 1 after c in its row (that row is an extra row and c' is a new 1). Easy to check that a, b, c' and d give us a configuration C_1 or one what is the same as the original configuration. So using (i) or the inductional hypothesis we get a contradiction.

Case 2: c is a new 1. a, b, c and d are all new 1's. So our configuration is in a copy of $M(s-1, C(s,t-1))$.

Case 3: c is an ordinary 1. Using 8.3.(5) our configuration is inside a copy of $M(s,t-1)$.

(iii)-(vi) Using the same case analysis based on he bottom left 1 (what is not necessarily c).

(vii) *Case1:* d is an extra 1. Then a, b and c are ordinary 1's in the same horizontal block (using 8.3.(5) and the fact that ordinary columns and rows in the same block are consecutive ones). Then the positions of b and c are contradictory with 8.3.(3).

Case 2: d is a new 1. The same as the previous second cases.

Case 3: d is an ordinary 1. The same as the previous third cases.

(viii) *Case 1:* d is an extra 1 and c is a new 1. $c \nearrow b$ hence b is a new 1 too, in particular a's and b's row is an extra row. $d \nearrow a$ so a cannot be an extra 1. Hence all four 1's are new except d. Move d right to the first 1 in its row. Then we obtain four new ones (hence they are in a copy of $M(s-1, C(s,t-1))$ such that their configuration is the one described in (vii) or in (viii).

Case 2: d is an extra 1 and c is an ordinary 1. Using similar arguments as before we have that all four 1's are ordinary except d and they are in the same horizontal block. Move d up by one position. We obtain four ordinary 1's (inside a copy of $M(s,t-1)$) such that their configuration is the one described in (vi) or in (viii).

Case 3: d is not an extra 1. In this case take the bottom left 1 (d) and replace it with

another 1 by shifting it to the leading 1 in its row and sinking it to the bottom 1 in that column. This way we obtain the same configuration but the new d is an extra 1. That was handled in the previous cases. ∎

The previous theorem gives lower bounds on the complexity of several configurations and sets of configurations.

Corollary 8.5.
(1) $f(n;C_6) = \Omega(n\alpha(n))$,
(2) $f(n;C_8) = \Omega(n\alpha(n))$,
(3) $f(n;\{\begin{pmatrix} & 1 & 1 \\ 1 & & 1 \end{pmatrix}, \begin{pmatrix} 1 & & \\ & & 1 \\ 1 & 1 & \end{pmatrix}, \begin{pmatrix} 1 & & & 1 \\ 1 & 1 & & \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ & 1 \\ 1 & \end{pmatrix}\}) = \Omega(n\alpha(n))$. ∎

9. Upper bound on Davenport-Schinzel matrices

In this section we prove that

Theorem 9.1. $f(n;\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}) \leq O(n\alpha(n))$.

Proof. Let $A' = (a'_{ij})$ be an $n \times m$ 0–1 matrix not having a subconfiguration of $C_6 = \begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}$. Delete the first and the last 1 in each row, and keep only the columns with at least 2 entries. The obtained matrix is denoted by $A = (a_{ij})$, and obviously, for the number of entries we have $||A'|| \leq ||A|| + 2n + m$. Form a sequence s_j from the j'th column $(a_{ij})_{1\leq i\leq n}$ of length $||(a_{ij})|| =: l(j)$ in the following way

$$s_j = (s_1^j,\ s_2^j, \ldots, s_{l(j)}^j),$$

where $s_1^j < s_2^j < \ldots < s_{l(j)}^j$ and $a_{s_i j} = 1$ for $1 \leq i \leq l(j)$. Form one sequence $s' =: s_1 s_2 \ldots s_m$ in this order. Delete from s' the element $s_{l(j)}^j$ if it equals to s_1^{j+1}. In the obtained sequence, s, there are no equal consequtive elements. We claim that s does not contain a subsequence $ababa$, i.e. it is a $DS_5(n)$ sequence.

Suppose on the contrary. Then there exists a subsequence $abab$ of s with $a < b$. So there are $j_1 \leq \ldots \leq j_4$ such that $a \in s_{j_1}$, $b \in s_{j_2}$, $a \in s_{j_3}$, $b \in s_{j_4}$. Here $j_2 < j_3$, otherwise the first b in $abab$ could not preceed the second a in s. Consider the submatrix defined by the rows a and b and the columns $\{j_1, \ldots, j_4\}$. There are four possibilities.

$j_1 < j_2 < j_3 < j_4$

$$\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}$$

$j_1 = j_2 < j_3 < j_4$

$$\begin{pmatrix} 1 & 1 & & \\ 1 & & & 1 \end{pmatrix}$$

$j_1 = j_2$ and $j_3 = j_4$

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$j_1 < j_2 < j_3 = j_4$

$$\begin{pmatrix} 1 & & & 1 \\ & 1 & 1 & \end{pmatrix}.$$

In each cases A will contain a C_6, a contradiction.
So $||A|| \leq 2n + 2m + ds_5(n)$. ∎

Corollary 9.2. $f(n; C_7), f(n; C_9) = O(n\alpha(n))$. ∎

In the very same way we can obtain the following theorem. Let C^{2k} be a partial $2 \times 2k$ matrix with $c_{1,2i-1} = 1$, $c_{2,2i} = 1$ for $1 \leq i \leq k$.

Theorem 9.3. $f(n; C^{2k}) \leq O(ds_{4k-3}(n))$. ∎

It is not difficult to give a lower bound for $f(n; C^{2k})$ which is probably closer to f as the upper bound.

Theorem 9.4. $f(n; C^{2k}) \geq \Omega\left(\frac{ds_{2k+1}(n)}{\alpha(\alpha(n))^{O(\alpha(\alpha(n))^{2k-4})}}\right)$.

Remark. Here the right hand side is superlinear. For the best bound on $ds_{2k+1}(n)$ see [ASS].

Proof. Let s be a Davenport-Schinzel sequence $s \in DS_{2k+1}(n)$ of length $ds_{2k+1}(n)$ such that the element i appears earlier than j for $i < j$. It is well-known [Sh87], that

$$ds_{2k+1}(n) = O(n\alpha(n)^{O(\alpha(n)^{2k-4})}).$$

Split s into n almost equal parts $s = s_1 \dots s_n$. Let $s_i^{'}$ be a set of distinct values of s_i, $||s_i^{'}|| = x$. We have that

$$y =: \frac{ds_{2k+1}(n)}{n} - 1 \leq ||s_i|| \leq xO(\alpha(x)^{O(\alpha(x)^{2k-4})}) \leq xO(\alpha(y)^{O(\alpha(y)^{2k-4})}).$$

Here $y = O(\alpha(n)^{O(\alpha(n)^{2k-4})})$, so $\alpha(y) = \alpha(\alpha(n)) + O(1)$.

Finally, forming the i^{th} column of an $n \times n$ matrix A from $s_i^{'}$ we obtain the desired configuration without C^{2k}. ∎

10. Conclusions and open problems

The next table summarizes our results.

Configurations	Lower bound	Upper bound
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\Theta(n^{\frac{3}{2}})$ see Theorem 1.1.	
$\begin{pmatrix} 1 & 1 & \\ 1 & & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}$	$\Theta(n \log n)$ see Theorem 3.1. and Corollary 3.4.	
$\begin{pmatrix} 1 & & \\ & 1 & \\ 1 & & 1 \end{pmatrix}$	$\Omega(\frac{n \log n}{\log \log n})$ see Corollary 4.4.	$O(n \log n)$ see Corollary 3.4.
$\begin{pmatrix} 1 & & 1 \\ 1 & & \\ & 1 & \end{pmatrix}$	$\Omega(n)$	$O(n \log n)$ see Corollary 3.4.
$\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix}, \begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & 1 \end{pmatrix}$	$\Theta(n\alpha(n))$ see Corollary 7.5. and Theorem 8.1.	
$\begin{pmatrix} & 1 & & \\ & & & 1 \\ 1 & & 1 & \end{pmatrix}, \begin{pmatrix} & 1 & & \\ & & & 1 \\ 1 & & & \\ & & 1 & \end{pmatrix}$	$\Omega(n)$	$O(n\alpha(n))$ see Corollary 8.2.
All the other 28 matrices with 4 entries	$\Theta(n)$	

Finally we mention several open problems. The first few ones are suggested by the previous table. Even in the case of configurations with four 1's there are several unknown complexities.

Is it true that the complexity of all permutation configurations are linear?

What is the characterization of configurations with linear complexity? In extremal graph theory the forbidden subgraphs with linear threshold are exactly the trees.

Is it true, that if G is the (bipartite) graph corresponding the configuration C then

$$f(n;C) < O(T(n;G)\log n)? \tag{10.1}$$

Does (10.1) hold at least for trees?

There are several combinatorial structures with an underlying order where the similar extremal question is interesting. An example is a set of intervals on a given line. How many intervals (over n endpoints) guarantee the existence of a given interval configuration? Similar question can be asked about diagonals in a cycle. Davenport and Schinzel's original question can be extended to arbitrary forbidden subsequence. As far we know there is no organized account of these questions.

7. PARTITION OF GRAPHS

1. Introduction

E. Győri suggested the following problem in the sixth Hungarian Colloquium on Combinatorics held at Eger, 1981: given integers $s, t \geq 3$, does there exist a number $f(s,t)$ such that every $f(s,t)$-connected graph G admits a proper partition $\{ S, T \}$ of the vertex-set $V(G)$ so that the induced subgraphs $G(S)$ and $G(T)$ are s-connected and t-connected, respectively? Thomassen [Tho82], and independently M. Szegedy [Sz82] established the existence of the function $f(s,t)$. In his proof Thomassen showed the existence of a function $g(s,t)$ (where s, t are natural numbers) such that the vertex-set of any graph G with minimum degree $g(s,t)$ has a decomposition $S \cup T$ such that $G(S)$ and $G(T)$ have minimum degree at least s and t, respectively. Let $f(s,t)$ and $g(s,t)$ be the minimal numbers under the above conditions. In his paper Thomassen gave rather weak bounds. We sharpen the estimates of the functions $f(s,t)$ and $g(s,t)$.

Let us recall a few graph theoretical notations. Let G be a simple graph, $V(G)$ is the set of points of G, $E(G)$ is the set of edges of G. Let S be a subset of the vertices and $x \in V(G)$. $G(S)$ is the subgraphs of G induced by S, $e(S)$ is the number of edges of $G(S)$, $d(S)$ denotes the minimum degree in $G(S)$, $d(x)$ is the degree of point x, $d(x,S)$ denotes the number of edges xy, where $y \in S$. $\{ S, T \}$ is a partition of the set $V(G)$ if S and T are non-empty, disjoint subsets of $V(G)$ such that $S \cup T = V(G)$.

2. Estimating the function $f(s,t)$

Theorem 2.1. *If $s \geq 4$, then $g(s,t) \leq t + 2s - 3$.*

Proof. We have to prove that $d(V(G)) \geq t + 2s - 3$, then there exists a partition $\{ S, T \}$ of the set $V(G)$ such that $d(S) \geq s$ and $d(T) \geq t$.

Let S be a vertex-set such that $|S|$ is minimal and (if there are more than such S) $e(S)$ is maximal under the conditions:

$$(*) \qquad \begin{cases} e(S) > (s-1)|S| - \frac{s(s-1)}{2} \\ |S| \geq s \end{cases}$$

Such an S exists (for example $V(G)$ satisfies condition $(*)$). $S \neq V(G)$ as for any $x \in V(G)$,

$$d\big(V(G) - \{\, x \,\}\big) \geq t + 2s - 4,$$

so if $t \geq 2$, then

$$e\big(V(G) - \{\, x \,\}\big) \geq \frac{t+2s-4}{2}\big|V(G) - \{\, x \,\}\big| \geq (s-1)|V(G) - \{\, x \,\}|.$$

(If $t = 1$, the assertion is trivial.) So $\{\, S, T = V(G) - S \,\}$ is a partition of $V(G)$. We shall prove that this is an appropriate partition.

First we note that $|S| \geq s+1$. Suppose that $|S| = s$, then from $(*)$ we obtain

$$e(S) > (s-1)s - \frac{s(s-1)}{2} = \frac{s(s-1)}{2},$$

a contradiction.

Now let x be a point of S with minimum degree in $G(S)$. We claim that $d(x, S) \geq s$, i.e. $d(S) \geq s$. Assume indirectly that it is not the case:

$$\begin{aligned} |S - \{\, x \,\}| \geq & s, \\ e(S - \{\, x \,\}) > & (s-1)|S| - \frac{s(s-1)}{2} - (s-1) = (s-1)|S - \{\, x \,\}| - \frac{s(s-1)}{2}. \end{aligned}$$

This contradicts the minimality of $|S|$.

On the other hand, if x is the vertex above, then by the minimality of $|S|$,

$$(2.2) \quad e(S) - d(S) = e(S - \{\, x \,\}) \leq (s-1)|S - \{\, x \,\}| - \frac{s(s-1)}{2} = (s-1)(|S|-1) - \frac{s(s-1)}{2}.$$

We have

$$\frac{|S|d(S)}{2} \leq e(S) \leq (s-1)\Big(|S| - 1 - \frac{s}{2}\Big) + d(S), \quad d(S) \leq (2s-2)\frac{|S| - 1 - \frac{s}{2}}{|S| - 2}.$$

So if $s \geq 3$, then $d(S) < 2s - 2$, i.e. $d(S) \leq 2s - 3$.

Now we show only a "few" edges connect any point of $T = V(G) - S$ to S. Let $D = \max_{x \in T} d(x, S)$ and let y be a point of T for which $d(y, S) = D$. We claim that $D \leq 2s - 3$.

There are two cases to consider.

Case 1. $d(S) \leq 2s - 4$.

Suppose that $D > 2s-3$. Let $S' = S - \{x\} \cup \{y\}$. $e(S') > e(S)$ and $|S'| = |S|$, so S' satisfies the condition $(*)$. This contradicts the maximality of $e(S)$. This proves that $D \leq 2s-3$.

Case 2. $D \geq 2s-1$.

The same exchange as above gives us a contradiction.

Case 3. If $D = 2s-2$ and there is an $x \in S$ such that $d(x,S) = 2s-3$ and $xy \notin E(G)$.

The same exchange as above gives us a contradiction.

Case 4. If $D = 2s-2$ and there exist no $x \in S$ such that $d(x,S) = 2s-3$ and $xy \notin E(G)$.

Then in S only the neighbors of y may have degree $2s-3$. Hence

$$e(S) \geq \frac{(2s-2)(2s-3) + \big[|S| - (2s-2)\big](2s-2)}{2} = (s-1)\big(|S|-1\big).$$

But from (2.2)

$$e(S) \leq (s-1)\big(|S|-1\big) - \frac{s(s-1)}{2} + d(S) = (s-1)\big(|S|-1\big) - \frac{s(s-1)}{2} + 2s - 3.$$

So we have

$$2s - 3 - \frac{s(s-1)}{2} \geq 0,$$

$$0 \geq s^2 - 5s + 6 = (s-2)(s-3).$$

This contradicts $s \geq 4$.

This proves that $D \leq 2s-3$, hence $d(T) \geq t$. ∎

In fact we proved the following claim:

Theorem 2.3. *Let G be a graph with minimum degree at least $2s-1$ $(s \geq 4)$ and let S be a vertex-set such that $|S|$ is minimal and $e(S)$ maximal under condition $(*)$. Then $S, V(G) - S$ are non-empty sets and in S the degree of any vertex is at least s and from any vertex in $V(G) - S$ at most $2s-3$ edges go to S.* ∎

Exchanging s and t we obtain $g(s,t) \leq 2t + s - 3$. So if $\min(s,t) \geq 4$, then $g(s,t) \leq s + t - 3 + \min(s,t)$. It can be easily seen from the proof that if $s = 3$, then $g(s,t) \leq t+4$ and if $s = 2$, then $g(s,t) \leq t+3$. The complete graph K_{s+t+1}, shows that $g(s,t) \geq s+t+1$. The following corollary sums up our results.

Corollary 2.4. *If $\min(s,t) \leq 4$, then $g(s,t) = s+t+1$, if $\min(s,t) \geq 4$, then $s+t+1 \leq g(s,t) \leq s+t+1+\big(\min(s,t)-4\big)$.* ∎

In the proof we used only the following conditions on the set S:

(1) S satisfies $(*)$,

(2) if $x \in S$, then $S - \{x\}$ does not satisfy $(*)$,

(3) if $x \in S$ and $y \notin S$, then $e\big(S - \{x\} \cup \{y\}\big) \leq e(S)$.

Based on this observation we give an algorithm which has a graph G with minimum degree at least $2s + t - 3$ as input and a partition $\{S, T\}$ as output.

Algorithms 2.5. *Step 0.* Let S=V(G).

Step 1. If there exists an $x \in S$ such that $S - \{ x \}$ satisfies $(*)$, then let $S = S - \{ x \}$ and restart the algorithm with the new S.

If there is no appropriate x, go to step 2.

Step 2. If there is an $x \in S$ and $y \notin S$ such that $e(S - \{ x \} \cup \{ y \}) > e(S)$, then let $S = S - \{ x \} \cup \{ y \}$ and restart step 2 with the new S.

If there is no (x, y) pair as above, then from the above remark $\{ S, V(G) - S \}$ is a valid partition.

Step 1 is executed $\mathcal{O}(n)$ many times. Between two executions of step 1 the step 2 is called $\mathcal{O}(n^2)$ many times. The execution of a step takes $\mathcal{O}(n^2)$ inspections. Consequently the running time of the whole algorithm is $\mathcal{O}(n^5)$.

3. Estimating the function $f(s, t)$

Theorem 3.1. [Mad72] *If $|V(G)| \geq 2n - 1$ ($n \geq 2$, natural number) and $e(G) > (2n - 3)\big(|V(G)| - (n - 1)\big)$ for a simple graph G, then G has an n-connected induced subgraph.* ∎

In fact the proof of this theorem gives the following result:

Theorem 3.2. [Mad72] *If G is a graph as above, $V_0 \subset V(G)$ and $|V_0|$ is minimal such that $|V_0| \geq 2n - 2$ and*

$$e(V_0) > (2n - 3)\big(|V_0| - n + 1\big) = (2n - 3)|V_0| - \frac{(2n - 2)(2n - 3)}{2},$$

then V_0 induces an n-connected subgraph. ∎

Using this result, we prove the following theorem:

Theorem 3.3. *If $s \geq 3, t \geq 2$ and G is an $(s + t - 1)$-connected graph with $d(V(G)) \geq 4s + 4t - 13$, then there exists a partition $\{ S, T \}$ of $V(G)$ such that $G(S)$ and $G(T)$ are s-connected and t-connected, respectively.*

Proof. Consider the following conditions on $S \subset V(G)$:

$$|S| \geq 2s - 2,$$

$$e(S) > (2s - 3)|S| - \frac{(2s - 2)(2s - 3)}{2}.$$

Choose an S such that $|S|$ is minimal and (if there are more than one such S) $e(S)$ is maximal under the conditions above. $\{ S, T = V(G) - S \}$ is a partition of $V(G)$ which can be proved the same way as it was done in theorem 2.1.

By theorem 3.2 S induces an s connected subgraph.

On the other hand by theorem 2.3 $d(y, S) \leq 2(2s-2) - 3 = 4s - 7$ for $y \in T$. Consequently, $d(T) \geq 4t - 6$. So

$$|T| \geq 2t - 2, \quad \text{and}$$

$$e(T) \geq \frac{(4t-6)|T|}{2} > (2t-3)\big[|T| - (t-1)\big].$$

By Mader's theorem 3.2 there exists a subset $T_0 \subset T$ with $G(T_0)$ t-connected.

After this, we can follow the proof of Thomassen. Let S and T be non-empty, disjoint sets such that $G(S)$ and $G(T)$ are s-connected and t-connected, respectively, and $|S \cup T|$ is maximal.

We are going to prove that $S \cup T = V(G)$. Suppose that $A = V(G) - (S \cup T) \neq \emptyset$. Since $|(S \cup A) \cup T| > |S \cup T|$ by the maximality of $|S \cup T|$ we have that $G(S \cup A)$ is not an s-connected graph, i.e. $G(S \cup A)$ has a cut-set B such that $|B| \leq s - 1$. Let G_1 be a component of $G\big((S \cup T) - B\big)$ such that $C = V(G_1) \subset A$. $|(T \cup C) \cup S| > |S \cup T|$, so $G(T \cup C)$ is not a t-connected graph. Let D be a cut-set of $G(T \cup C)$ such that $|D| \leq t - 1$.

It is easy verify that $B \cup D$ is a cut-set of G. But $|B \cup D| \leq s + t - 2$, contradicting the $(s + t - 1)$-connectivity of G. ∎

If G is $(4s + 4t - 13)$-connected, then G satisfies the conditions of theorem 3.3. Hence we obtained that if $s \geq 3$, $t \geq 2$, then $f(s, t) \leq 4s + 4t - 13$. The graph K_{s+t+1} shows that $f(s, t) \geq s + t + 1$. The following corollary sums up our bounds on $f(s, t)$:

Corollary 3.4. *If $s \geq 3$ and $t \geq 2$, then*

$$s + t + 1 \leq f(s, t) \leq 4s + 4t - 13. \quad ∎$$

REFERENCES

[A28] W. Ackermann, *Zum Hilbertschen Aufbau der reelen Zahlen*, Math. Ann. **99** (1928), 118-133.

[AKV] R. Adamec, M. Klazar and P. Valtr, *Forbidden words*, Preprint, Department of Mathematics, Karlovy University, Prague, Czechoslovakia.

[Ad78] L. Adleman, *Two theorems on random polynomial time*, Proc. 19th IEEE FOCS (1978), 75–83.

[ASS] P. Agarwal, M. Sharir and P. Shor, *Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences*, Preprint.

[AA88] A. Aggarwal and R.J. Anderson, *A random NC-algorithm for depth first search*, Combinatorica **8** (1988), 1–12.

[AAK89] A. Aggarwal, R.J. Anderson and M.Y. Kao, *Parallel depth-first search in general directed graphs*, Proc. 21st ACM STOC (1989), 297–308.

[AHU74] A. Aho, J.Hopcroft and J. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Menlo Park, CA, 1974.

[ABHKPRSzT86] M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. Rödl, E. Szemerédi, Gy. Turán, *Two lower bounds for branching programs*, Proc. 18th ACM STOC (1986), 30–38.

[AKS87] M. Ajtai, J. Komlós and E. Szemerédi, *Deterministic simulation in LOGSPACE*, Proc. 19th ACM STOC (1987), 132–140.

[AW85] M. Ajtai and A. Wigderson, *Deterministic simulation of probabilistic constant depth circuits*, Proc. 26th IEEE FOCS (1985), 11–19.

[ABI86] N. Alon, L. Babai and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, Journal of Algorithms **7** (1986), 567-583.

[AB87] N. Alon and R. Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica **7** (1987), 1–22.

[AM86] N. Alon and W. Maass, *Ramsey theory and lower bounds for branching programs*, Proc. 27th IEEE FOCS (1986), 410–417.

[And87] R.J. Anderson, *A parallel algorithm for the maximal path problem*, Combinatorica **7** (1987), 315–326.

[An85] A. E. Andreev, *On a method of obtaining lower bounds for the complexity of individual monotone functions*, in Russian, Dokl. Akad. Nauk SSSR **282/5** (1985), 1033–1037.

[AV79] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, Journal of Computer and System Sciences **19** (1979), 155–193.

[BHSzT87] L. Babai, P. Hajnal, E. Szemerédi and Gy. Turán, *A lower bound for read-only-once branching programs*, J.C.S.S. **35** (1987), 153–162.

[BPRSz] L. Babai, P. Pudlák, V. Rödl and E. Szemerédi, *Lower bounds to the complexity of symmetric Boolean functions*, submitted for publication.

[Ba85] D. A. Barrington, *Width-3 permutation branching programs*, draft, MIT (1985).

[Ba86] D. A. Barrington, *Bounded-width polynomial size branching programs recognize exactly those languages in NC^1*, Proc. 18th ACM STOC (1986), 1–5.

[Be86] P. Beame, *Limits on the power of concurrent-write parallel machines*, Proc. 18th ACM STOC (1986), 169–176.

[BC85] P. Beame and S. Cook, private communication.

[BO83] M. Ben-Or, *Lower bounds for algebraic computational trees*, Proc. 15th ACM STOC (1983), 247–248.

[Be73] C. Berge, "Graphs and Hypergraphs," North-Holland - American Elsevier, 1973.

[BBL74] M.R. Best, P. van Emde Boas and H.W. Lenstra, Jr., *A sharpened version of the Aanderaa–Rosenberg conjecture*, Report ZW 30/74, Mathematish Centrum, Amsterdam (1974).

[BGy] D. Bienstock and E. Győri, *An extremal problem on sparse $0-1$ matrices*, to appear in *SIAM J. Disc. Math.*.

[Bl84] N. Blum, *A boolean function requiring $3n$ network size*, Theoritical Computer Science **28** (1984), 337–345.

[BI87] M. Blum and R. Impagliazzo, *Generic oracles and oracle classes*, Proc. 28th IEEE FOCS (1987), 118–126.

[Bo77] B. Bollobás, *Complete subgraphs are elusive*, J. Combinatorial Theory Ser. B **20** (1976), 1–7.

[Bo78] B. Bollobás, "Extremal Graph theory," Academic Press, London, 1978.

[Bo85] B. Bollobás, "Random graphs," Academic Press, London, 1985.

[BE78] B. Bollobás and S. E. Eldridge, *Packing of graphs and applications to computational complexity*, J. of Combinatorial Theory Ser. B **25**, 105–124.

[Bo] R. Boppana, for a description see [SW86], unpublished.

[BoS88] R. Boppana and M. Sipser, *The complexity of finite functions*, preprint, 1988, to appear in "The Handbook of Theoretical Computer Science", edited by J. van Leewen et al., North-Holland, Amsterdam.

[Bor77] A. Borodin, *On relating time and space to size and depth*, SIAM J. Comput. **6** (1977), 733–744.

[BFKLT81] A. Borodin, M.J. Fischer, D.G. Kirkpatrick, N.A. Lynch and M. Tompa, *A time-space tradeoff for sorting on nonoblivious machines*, J.C.S.S. **22** (1981), 351–364.

[BDFP83] A. Borodin, D. Dolev, F. E. Fich and W. Paul, *Bounds for width-2 branching programs*, Proc. 15th ACM STOC (1983), 87–93.

[BK87] J. Boyar and H. Karloff, *Coloring planar graphs in parallel*, J. Algorithms **8** (1987), 470–479.

[Br86] A.Z. Broder, *How hard is it to marry at random*, Proc. 18th ACM STOC (1986), 50–58.

[Br66] W.G. Brown, *On graphs that do not contain a Thomsen graph*, Canad. Math. Bull. **9** (1966), 281–285.

[BS77] D. Burns and S. Schuster, *Every* $(p, p-2)$ *graph is contained in its complement*, J. Graph Theory **1** (1977), 277–279.

[BS78] D. Burns and S. Schuster, *Embedding* $(p, p-1)$ *graphs in their complements*, Israel J. Math. **30** (1978), 313–320.

[Ca74] P. A. Catlin, *Subgraphs of graphs I.*, Discrete Math. **10** (1974), 225–233.

[CFL83] A. K. Chandra, M. L. Furst and R. J. Lipton, *Multiparty protocols*, Proc. 15th ACM STOC (1983), 94–99.

[Ch52] H. Chernoff, *A measure of asymptotic effiency for tests of a hypothesis based on the sum of observations*, Annals of Math. Stat. **23** (1952), 493–509.

[Co85] S. Cook, *A Taxonomy of Problems with Fast Parallel Algorithms*, Information and Control **64** (1985), 2–22.

[CDR86] S. Cook, C. Dwork and R. Reischuk, *Upper and lower bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput. **15** (1986), 87–97.

[DS65] H. Davenport and A. Schinzel, *A combinatorial problem connected with differential equations, I and II,*, *Amer. J. Math.*, **87** (1965), 684-694 and *Acta Arithmetica*, **17** (1971), 363-372.

[DM86] M. Dietzfelbinger and W. Maass, *Two lower bound arguments with 'inaccessible' number*, Structure in Complexity Theory, Lecture Notes in Computer Science, **223**, Springer, Berlin - New york, 1986, 163–183.

[Di52] G.A. Dirac, *Some theorems on abstract graphs*, Proc. London Math. Soc. **2** (1952), 69–81.

[Du85] P.E. Dunne, *Lower bounds on the complexity of* 1*-time-only branching programs*, FCT Proc., Lect. Notes in Comp. Sci. **199** (1985), 90–99.

[EH91] H. Edelsbrunner and P. Hajnal, *A lower bound on the number of unit distances between the vertices of a convex polygon*, JCT ser. A **56** (1991), 312–316.

[Er47] P. Erdős, *Some remarks on the theory of graphs*, Bulletin Amer. Math. Soc. **53** (1947), 242–294.

[ERS66] P. Erdős, A Rényi and V.T. Sós, *On a problem of graph theory*, Studia Sci. Math. Hungar. **1** (1966), 215–235.

[ESi66] P. Erdős and M. Simonovits, *A limit theorm in graph theory*, Studia Sci. Math. Hungar. **1** (1966), 51–57.

[ES74] P. Erdős and J. Spencer, "Probabilistic methods in combinatorics," Akadémia Kiadó, Budapest, 1974.

[ESt46] P. Erdős and A.H. Stone, *On the structure of linear graphs*, Bull. Amer. Math. Soc. **52** (1946), 1087–1091.

[FT87] U. Faigle and Gy Turán, *The complexity of interval orders and semiorders*, Discrete Math **63** (1987), 131–141.

[FT88] U. Faigle and Gy. Turán, *Sorting and recognition problems for ordered sets*, SIAM J. Comput. **17** (1988), 100–113.

[FRSS81] R.J. Faudree, C.C. Rousseau, R.H. Schelp and S. Schuster, *Embedding graphs in their complements*, Csechoslovak Math J. **31** (1981), 53–62.

[FMP] M. J. Fischer, A. Meyer and M. S. Paterson, $\Omega(n \log n)$ *lower bounds on length of Boolean formulas*, SIAM J. Computing **11** (1982), 416–427.

[FHS78] S. Fortune, J. Hopcroft, E. M. Schmidt, *The complexity of equivalence and containment free single variable program schemes*, Fifth Internat. Colloq., Udine, Lecture Notes in Computer Science, **62**, Springer, Berlin - New York, 1978, 227–240.

[FW78] S. Fortune and J. Wyllie, *Parallelism in random access mashines*, Proc. 10th ACM STOC (1978), 114-118.

[Fü] Z. Füredi, *The maximum number of unit distances in a convex n-gon*, to appear in *J. Combinatorial Th., A.*.

[FH] Z. Füredi and P. Hajnal, *Davenport-Schinzel theory of matrices*, to appear in *Discrete Mathematics*.

[GJ79] M. Garey and D. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," W.H. Freeman and Company, San Francisco, 1979.

[GPS87] A. Goldberg, S. Plotkin and G. Shannon, *Parallel symmetry-breaking in sparse graphs*, Proc. 19th ACM STOC (1987), 315–324.

[GS87] M. Goldberg and T. Spencer, *A new parallel algorithm for the maximal independent set problem*, Proc. 28th IEEE FOCS (1987), 161–165.

[Go77] L.M. Goldschlager, *Synchronous parallel computation*, Ph. D. Thesis, University of Toronto (1977); see also, J. ACM **29** (1982), 1073–1086.

[GRS80] R. L. Graham, B. Rothschild and J. Spencer, "Ramsey Theory," Wiley, New York, 1980.

[GyL76] A. Gyárfás and J. Lehel, *Packing trees of different order into K_n*, in: "Combinatorics", Akadémia kiadó, Budapest, 1976, 463–469.

[Gy81] E. Győri, *An n-dimensional search problem with resticted questions*, Combinatorica **1** (1981), 377–380.

[HMT88] A. Hajnal, W. Maas and Gy. Turán, *On the communication complexity of graph properties*, Proc. 20th ACM STOC (1988), 186–191.

[Ha83] P. Hajnal, *Partition of graphs with condition on the connectivity and minimum degree*, Combinatorica **3** (1983), 95–99.

[Ha] P. Hajnal, *On the number of unit distances between vertices of a convex polygon*, Manuscript.

[Ha88] P. Hajnal, *Fast parallel algorithm for finding a Hamiltonian cycle in dense graphs*, The University of Chicago, Technical Report 88–003, April 1988.

[Ha90] P. Hajnal, *On the power of randomness in the decision tree model*, Proc. 5th Structure in Complexity Theory Conf. (1990), 66–77.

[Ha91] P. Hajnal, *An $\Omega\left(n^{4/3}\right)$ lower bound on the randomized complexity of graph properties*, Combinatorica (11(2)), 131–143.

[HaSz] P. Hajnal and M. Szegedy, *On packing bipartite graphs*, To appear in *Combinatorica*.

[HSz88] P. Hajnal and E. Szemerédi, *Parallel Brooks coloring*, SIAM J. Disc. Math **3** (1990), 74–80.

[HS86] S. Hart and M. Sharir, *Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes*, Combinatorica **6** (1986), 151–177.

[Ha86] J. Hastad, *Improved lower bounds for small depth circuits*, Proc. 18th ACM STOC (1986), 6–20.

[HHS81] S.M. Hedetniemi, S.T. Hedetniemi and P.J. Slater, *A note on packing two trees into K_n*, Ars Combinatorica **11** (1981), 149–153.

[HR72] R.C. Holt and E.M. Reingold, *On the time required to detect cycles and connectivity in directed graphs*, Math. Systems Theory **6** (1972), 103–107.

[HB84] K. Hwang and F.A. Briggs, "Computer architecture and parallel processing," McGraw-Hill, New York, 1984.

[Il78] N. Illies, *A counterexample to the generalized Aanderaa−Rosenberg conjecture*, Info. Proc. Letters **7** (1978), 154–155.

[IS86] A. Israeli and Y. Shiloach, *An improved paralle algorithm for maximal matching*, Inf. Proc. Letters **22** (1986), 57–60.

[Ju87] S.P. Jukna, *Lower bounds on the complexity of local circuits*, preprint, 1987.

[KSS84] J. Kahn, M. Saks and D. Sturtevant, *A topological aproach to evasiveness*, Combinatorica **4** (1984), 297–306.

[KN88] M. Karchmer and J. Naor, *A fast parallel algorithm to color a graph with Δ colors*, J. of Algorithms **9** (1988), 83–91.

[Ka85] H. J. Karloff, *Fast parallel algorithms for graph theoretical problems*, M.Sc. Thesis, University of California, Berkeley.

[Ka86] H. Karloff, *A Las Vegas RNC algorithm for maximum matching*, Combinatorica **6** (1986), 387–392.

[Ka88] H. Karloff, *An NC algorithm for Brooks' theorem*, Theoretical Computer Science **68** (1988), 89–103.

[KUW86] R.M. Karp, E. Upfal and A. Wigderson, *Constructing a perfect matching is in random NC*, Combinatorica **6** (1986), 35–48.

[KaW85] R.M. Karp and A. Wigderson, *A fast parallel algorithm for the maximal independent set problem*, JACM **32** (1985), 762–773.

[Ki88] V. King, *Lower bounds on the complexity of graph properties*, Proc. 20th ACM STOC (1988), 468–476.

[Kir74] D. Kirkpatrick, *Determining graph properties from matrix represantation,*, Proc. 6th SIGACT Conf. (1974), 84-90.

[KS86] D. Kirkpatrick and R. Seidel, *The ultimate planar convex hull algorithm*, SIAM J. Comput. **15** (1986), 287–299.

[KK80] D.J. Kleitman and D.J. Kwiatkowski, *Further results on the Aanderaa−Rosenberg conjecture*, J. Combinatorial Theory **28** (1980), 85–95.

[KM75] D.E. Knuth and R.W. Moore, *An analysis of alpha-beta pruning*, Artificial Intelligence **6** (1975), 293–326.

[Ko88] P. Komjáth, *A simplified construction of nonlinear Davenport-Schinzel sequences,*, J. of Comb. Theory **A49** (1988), 262–267.

[KST54] T. Kővári, V.T. Sós and P. Turán, *On a problem of Zarankiewicz*, Colloq. Math. **3** (1954), 50–57.

[Kr87] M. Krause, *Lower bounds for depth-restricted branching programs*, preprint, 1987.

[Kr88] M. Krause, *Exponential lower bounds on the complexity of local and real-time branching programs*, J. Inform. Process. Cybernet. **24** (1988), 99-110.

[KW86] K. Kriegel and S. Waack, *Lower bounds on the complexity of real-time branching programs*, preprint,1986.

[Le59] C. Y. Lee, *Representation of switching functions by binary decision programs*, Bell Syst. Tech. Journal **38** (1959), 985–999.

[Lo66] L. Lovász, *On decomposition of graphs*, Studia Sci. Math. Hung. **1** (1966), 237–238.

[Lo79] L. Lovász, "Combinatorial Problems and Exercises," North Holland, Amsterdam, 1979.

[Lo79b] L. Lovász, *Determinants matchings and random algorithms*, in: "Foundamentals of Computation Theory FCT '79" (ed. L. Budach), Akademie-Verlag, Berlin, 1979, 56–574.

[Lu86] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput. **15** (1986), 1036–1053.

[Mad72] W. Mader, *Existenz n-fach zussamenhängender Teilgraphen in Graphen genügend grossen Kantendichte*, Abh. Math. Sem. Hamburg Univ. **37** (1972), 86–97.

[MT85] U. Manber and M. Tompa, *The complexity of problems on probabilistic non-deterministic and alternating decision trees*, J. ACM **32** (1985), 732–740.

[Ma74] Z. Manna, "Mathematical Theory of Computation," McGraw-Hill, New York, 1974.

[Mas76] W. Masek, *A fast algorithm for the string editing problem and decision graph complexity*, M.Sc. Thesis, MIT (1976).

[M84] F. Meyer auf der Heide, *A polynomial linear search algorithm for the n-dimensional knapsack problem*, J. ACM **31** (1984), 668–676.

[MH85a] F. Meyer auf der Heide, *Fast algorithms for n-dimensional restrictions of hard problems*, J. Assoc. Comput. Mach. **35** (1988), 185–203.

[MH85b] F. Meyer auf der Heide, *Non-deterministic versus probabilistic linaer search algorithms*, Proc. 26th IEEE FOCS (1985), 65–73.

[MW74] E.C. Milner and D.J.A. Welsh, *On the computational complexity of graph theoritical properties*, Univ. of Calgary, Res. Paper No.232 (1974).

[MW76] E.C. Milner and D.J.A. Welsh, *On the computational complexity of graph theoritical properties*, in: Proc. Fifth British Combinatorial Conf. (ed: C.St.J.A. Nash-Williams and J. Sheehan), Utilitas Math., Winnipeg, Ontario, Canada, 1976, 471-487.

[M] J. Mitchell, *Shortest rectilinear paths among obstacles*, *SORIE Technical report No. 739*, Cornell University, 1987.

[Ne66] E. I. Nečiporuk, *On a Boolean function*, Dokl. Akad. Nauk SSSR **169** (1966), 765–766; English translation: Soviet Math Doklady **7** (1966), 999–1000.

[vN28] J. von Neumann, *Zur Theorie der Gesellschaftsspiele*, Math. Annalen **100** (1928), 295–320.

[Ni] N. Nisan, *CREW PRAMs and decision trees*, Proc. 21th ACM STOC (1989), 327–335.

[Pe80] J. Pearl, *Asymtotic properties of minimax trees and game-searching procedures*, Artif. Intell. **14** (1980), 113–126.

[Pe82] J. Pearl, *The solution for the branching factor of the alpha beta pruning algorithm and its optimality*, Comm. ACM **25** (1982), 559–564.

[Pr87] H.J. Prömel, *Counting unlabeled structures*, J. Combinatorial Th. Ser. A **44** (1987), 83–93.

[Pu84] P. Pudlák, *A lower bound on complexity of branching programs*, Proc. Conf. on the Mathematical Foundations of Computer Science, Springer Lecture Notes in Computer Science **176** (1984), 480–489.

[Ra85a] A. A. Razborov, *Lower bounds for the monotone complexity of some Boolean functions*, in Russian, Dokl. Akad. Nauk SSSR **281** (1985), 798–801; English translation: Soviet Mathematics Doklady **31** (1985), 354–357.

[Ra85b] A. A. Razborov, *A lower bound for the monotone network complexity of the logical permanent*, in Russian, Matematicheskie Zametki **37:6** (1985), 887–900; English translation: Math. Notes of the Acad. of Sci. of the USSR **37**, 485–493.

[Ra87] A. A. Razborov, *Lower bounds on the size of bounded depth networks over a complete basis with logical addition*, in Russian, Matematicheskie Zametki **41:4** (1987), 598–607; English translation: Math. Notes of the Acad. of Sci. of the USSR **41:4** (1987), 333–338.

[Re72] E. Reingold, *On the optimality of some set algorithms*, J. ACM **19** (1972), 649–659.

[RV76] R. Rivest and S. Vuillemin, *On recognizing graph properties from adjacency matrices*, Theor. Comp. Sci. **3** (1976), 371–384.

[Roi81] I. Roizen, *On the average number of terminal nodes examined by alpha-beta*, UCLA Cognitive Systems Laboratory Technical Report (1981).

[Ro73] A. L. Rosenberg, *On the time required to recognize properties of graphs: A problem*, SIG ACT News **5** (1973), 15–16.

[Sa] M. Saks, *Recognition problems for transitive relations*, submitted for publication.

[SW86] M. Saks and A. Wigderson, *Probabilistic boolean decision trees and the complexity of evaluating game trees*, Proc. 26th IEEE FOCS (1986), 29–38.

[SS78] N. Sauer and J. Spencer, *Edge-disjoint replacement of graphs*, J. of Combinatorial Theory Ser. B **25** (1978), 295–302.

[Sav76] J. E. Savage, "The Complexity of Computing," Wiley, New York, 1976.

[Sh49] C.E. Shannon, *The synthesis of two-terminal switching circuits*, Bell Syst. Techn. J. **28**, 59–98.

[Sh87] M. Sharir, *Almost linear upper bounds on the length of generalized Davenport-Schinzel sequences*, Combinatorica **7** (1987), 131–143.

[Sh] J. B. Shearer, announced in [Ba86].

[STY85] P.J. Slater, S.K. Teo and H.P. Yap, *Packing a tree with a graph of the same size*, J. Graph Theory **9** (1985), 213–216.

[Sm87] S. Smale, *On the topology of algorithms I.*, J. Compl. **3** (1987), 81–89.

[Sm87] R. Smolensky, *Algebraic methods in the theory of lower bound for Boolean circuit complexity*, Proc. 19th ACM STOC (1987), 77–82.

[Sn85] M. Snir, *Lower bounds for probabilistic linear decision trees*, Theor. Comp. Sci. **38** (1985), 69–82.

[Sp87] J. Spencer, "Ten lectures on the probabilistic method," SIAM, Philadelphia, 1987.

[SY] M. Steele and A. Yao, *Lower bounds for algebraic decision trees*, J. Algorithms **3** (1982), 1–8.

[Sz82] M. Szegedy, Personal communication.

[Sz74] E. Szemerédi, *On a problem by Davenport and Schinzel*, Acta Arithmetica **15** (1974), 213–224.

[Ta83] M. Tarsi, *Optimal search on some game trees*, J. ACM **3** (1983), 389–396.

[TY87] S.K. Teo and H.P.Yap, *Two theorems on packing of graphs*, Europ. J. Combinatorics **8** (1987), 199–207.

[Tho82] C. Thomassen, *Graph decomposition with constraints on the connectivity and minimum degree*, Journal of Graph Theory **7** (1983), 165–167.

[Tu37] A. Turing, *On computable numbers with an application to the entscheidungsproblem*, Proc. of the London Math. Soc. **42** (1936-7), 230–265.

[Ya77] A. Yao, *Probabilistic computation: towards a unified measure of complexity*, Proc. 18th IEEE FOCS (1977), 222-227.

[Ya79] A. Yao, *Some complexity questions related to distributed computations*, Proc. 11th ACM STOC (1979), 209–213.

[Ya81] A. Yao, *A lower bound to finding convex hulls*, J. ACM **28** (1981), 780–789.

[Y82] A. Yao, *Theory and applications of trapdoor functions*, Proc. 23the IEEE FOCS (1982), 80–91.

[Ya83] A. C. Yao, *Lower bounds by probabilistic arguments*, Proc. 24th IEEE FOCS (1983), 420–428.

[Ya85] A. C. Yao, *Separating the polynomial-time hierarchy by oracles*, Proc. 26th IEEE FOCS (1985), 1–10.

[Y87] A. Yao, *Lower bounds to randomized algorithms for graph properties*, Proc. 28th IEEE FOCS (1987), 393-400.

[Y88] A. Yao, *Monotone bipartite graph properties are evasive*, SIAM J. Comput. **17** (1988), 517–520.

[We87] I. Wegener, *On the complexity of branching programs and decision trees for clique functions*, TAPSOFT'87, Vol. 1., Lecture Notes in Computer Science, **249**, Springer, Berlin - New York, 1987, 1–12.

[Wi86] A. Wiernik, *Planar realizations of Nonlinear Davenport-Schinzel sequences by segments*, to appear in Discret and Comput. Geom., Procedings, 27th IEEE Found. of Comput. Sci. (1986), 97-106.

[Wi] A. Wigderson, unpublished.

[Zá84] S. Zák, *An exponential lower bound for one-time-only branching programs*, Proc. Conf. on Mathematical Foundations of Computer Science, Springer Lecture Notes in Computer Science **176** (1984), 562-566.