ELSEVIER

Discrete Applied Mathematics 54 (1994) 37–53

DISCRETE APPLIED MATHEMATICS

A lower bound for tree resolution*

David J. McClurkin

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, M5S 1A4

Received 15 January 1991; revised 18 December 1992

Abstract

In this article, highly expanding degree-3 bipartite graphs are generated randomly. Every graph gives rise to a contradictory set of clauses, and these particular graphs provide us with a highly interconnected set of 3SAT clauses. If n is the number of nodes in each side of the graph, then there are $3n$ variables and $8n$ clauses. We use this set to prove a lower bound for tree resolution. The lower bound obtained is $2^{(2/3)\lambda n}$ where $\lambda \approx 0.3166$. Letting $N = 3n$ be the number of variables, this bound is $\approx 2^{.070355N}$. This is contrasted with the best-known upper bound for 3SAT, from the algorithm in Monien and Speckenmeyer (1985), which is $\approx 2^{.6943N}$ where again N is the number of variables. Exponential lower bounds have been proved for stronger forms of resolution, but with significantly smaller constants.

1. Introduction

The problem of determining whether a given set of propositional logic clauses is satisfiable (SAT) is an important one in Computer Science. SAT was first proved by Cook [7] to be NP-complete, thus showing that it is representative of the difficulties of solving other problems in NP. In spite of the lack of a proof that P $\neq$ NP, it seems unlikely that a polynomial-time algorithm for SAT exists. Recent research has been directed at providing exponential-time algorithms where the exponential-time constants are as small as possible. Such algorithms are not feasible in the technical sense because of their asymptotic exponential-time behavior. However, even exponential-time algorithms may be "feasible", in the sense that if the problems we are interested in are not too large, and the exponential-time constants are small enough, then the problems may be solved in a reasonable amount of time on a powerful computer. So it is important to look at what constants are possible.

To study whether satisfiability or tautology can be practically determined, we must study proof systems that correspond in some way to actual computation, i.e., systems

* A somewhat expanded version of this result was submitted as a M.Sc. thesis at the University of Toronto.

which do not exploit nondeterminism as a computational resource. This rules out many systems like Frege systems because of their very nondeterministic nature. Tree resolution is important because the nondeterminism is limited, and because it corresponds in a very natural way to several algorithms for determining satisfiability or tautology, in particular the Davis–Putnam procedure. In fact, the fastest algorithms for SAT known at this time are equivalent to tree resolution, in the sense that they can be represented as algorithms which search for a resolution proof in tree form. Tree resolution has long been known to require exponential size proofs, but it is important to find as good a lower bound as possible to bound the behavior of this class of algorithms.

In this article, we first show, following [14], how each connected graph gives rise to an inconsistent family of clauses. We then state and prove Tseitin's theorem which shows the equivalence between tree resolution refutations of that set and edge deletion processes on the graph. Next, we randomly generate a family of highly expanding graphs. Finally, we show that members of this family require a large (exponential-size) edge deletion process, proving that a resolution refutation of the associated family of clauses must be large as well.

2. Definitions

We begin with formal definitions of the terminology used in this article.

Definition 2.1. A *literal* is either a variable or its negation, where the negation of a variable p is denoted by $\bar{p}$.

Definition 2.2. A *clause* is a disjunction of literals. The disjunction is implicitly assumed, so the clause $p \vee \bar{q} \vee r$, for example, is simply written $p\bar{q}r$. The empty, or contradictory, clause is written ϕ.

We now describe in detail the resolution proof system. The input to resolution is a set of clauses and the goal is to determine whether all clauses can be simultaneously satisfied by some truth value assignment (t.v.a.) to the variables. There is only one derivation rule for resolution: if there is a clause A containing a variable p, and another clause B containing its negation $\bar{p}$, then these 2 clauses may be resolved together, producing a third clause C, called the "resolvent", which is the disjunction of all literals in A together with those in B, except for p and $\bar{p}$ (any literals appearing identically in both A and B are simply merged, so that they occur only once in C). Clearly, this rule is sound, i.e., if there is a t.v.a. which simultaneously satisfies both A and B, then it must also satisfy C. The rule is also complete (see [14]), so if the original set of clauses is inconsistent, then we can eventually get the empty, or contradictory clause, ϕ.

$$\begin{tikzcd}
& ab \arrow[dr] & & \bar{a}bd \arrow[dl] & & ab \arrow[dr] & & \bar{a}b\bar{d} \arrow[dl] & & \bar{b}c \arrow[dr] & & \bar{b}\bar{c} \arrow[dl] \\
& & bd \arrow[drr] & & & & b\bar{d} \arrow[dll] & & & & \bar{b} \arrow[ddlll] & \\
& & & & b \arrow[drrr] & & & & & & & \\
& & & & & & & \phi & & & &
\end{tikzcd}$$

Fig. 1. A tree resolution refutation of the set $\{ab, \bar{a}bd, \bar{a}b\bar{d}, \bar{b}c, \bar{b}\bar{c}\}$.

There are several varieties of resolution for which results have been proved. The weakest is *tree resolution*, with which this article is concerned. In tree resolution, the proof is actually a binary tree, so each clause is either in the input set or is the resolvent of its left and right children. There is no overlap between the left and right subtrees of any node, so if a clause is used more than once, it must be reproved each time it is used (see Fig. 1).

The complexity measure of a resolution proof tree is simply the number of nodes in the tree. Equivalently, we count the number of leaf nodes, since the total number of nodes is 2(number of leaf nodes) − 1. In [14], a $2^{k\sqrt{n}}$ lower bound is proved, but vast improvements have been made since then.

A stronger system is *unrestricted resolution*. Here the proof is a sequence of clauses, each of which is either in the input set, or is the resolvent of 2 clauses appearing earlier in the sequence. In [12] a $2^{c\sqrt{n}}$ lower bound is proved using pigeon-hole clauses. Ref. [15] contains a 2^{cn} lower bound proof using Tseitin's graph clauses. Moreover, in [6] it is proved that almost all random clauses satisfy a generalization of Urquhart's criteria, and thus require exponential-sized proofs. All of these proofs yield very tiny constants.

An unrestricted resolution proof can be thought of as a directed acyclic graph (DAG), where each node is adjacent to all of its resolvents. This reflects the fact that if a clause is used many times, it does *not* have to be reproved each time it is used.

Tseitin defines the notion of "regularity" as follows: a resolution DAG is *regular* if on each path from the root to a leaf node, no variable is eliminated more than once. This is equivalent to saying: once a variable is eliminated it is never reintroduced by a subsequent resolution. In [10] a 2^{cn} lower bound is proved for regular resolution.

The regularity condition can be applied to tree resolution proofs. However, every tree resolution proof can be transformed into a regular tree resolution proof with no increase in size (see [14]). So, without loss of generality, we can (and shall) assume that the optimal tree resolution proof for a set of clauses is regular.

There is an even more powerful resolution system called *extended resolution* in which new variables can be introduced to stand for the disjunction of other variables. No nonpolynomial lower bound has been proved for extended resolution.

The stronger forms of resolution have a nondeterministic aspect to them which does not seem to correspond to any sort of deterministic algorithm. Tree resolution, on the other hand, is of much interest because it corresponds more closely to algorithms for determining the satisfiability of sets of clauses, the most famous of which is the *Davis–Putnam procedure.*

There are actually two very different algorithms known as the Davis–Putnam procedure. The one that corresponds to tree resolution, and the one to which we are referring whenever we use that terminology, works as follows: A variable p in the input set is selected and set to 0 (false). This transforms the set of clauses by removing p from all clauses containing p, and removing all clauses containing $\bar{p}$. It is then recursively determined whether the resulting set of clauses is satisfiable. If the answer is no, then the algorithm backtracks to the point where p was set to 0, and tries setting it to 1 instead, again recursively determining whether the resulting set of clauses is statisfiable.

It is not hard to see that the Davis–Putnam procedure is equivalent to a depth-first traversal of a resolution proof tree. Much effort has been devoted to studying the Davis–Putnam procedure in terms of its computational complexity, mainly in the area of probabilistic analysis and average time complexity. Ref. [11] shows that for certain probability distributions, the procedure has polynomial average time behavior. Ref. [9] shows that all of the distributions in the above are, in a sense, unreasonable, and shows that for a family of reasonable distributions, the procedure requires exponential time with probability 1.

The algorithm in [13] is similar to the Davis–Putnam procedure, except that they check for monotonicity, i.e., variables which occur only positively or negatively. They discard clauses containing such variables because they are all trivially satisfiable by simply setting that variable appropriately. From the tree resolution perspective, no such clause can appear in a resolution proof tree, because it would introduce a variable that could never be subsequently eliminated. Hence, their algorithm also corresponds to a search strategy for tree resolution.

3. Tseitin's theorem

This section contains a greatly simplified proof of the result in [14]. We first describe Tseitin's technique for obtaining an unsatisfiable set of clauses from a graph.

Let G be a connected graph (or multigraph), and label each vertex $v \in G$ by a "charge" $\varepsilon(v) \in \{0, 1\}$ so that $\sum_{v \in G} \varepsilon(v) \equiv 1 \pmod 2$ (such a graph is said to have an "odd labelling", see Fig. 2). Assign to each edge in G a unique literal, i.e., either a variable or its negation. Denote by $\Sigma(v)$ the set of clauses which represents the statement that $\sum_i x_i \equiv \varepsilon(v) \pmod 2$ where the sum is taken over all literals x_i incident with v. The exact requirement on $\Sigma(v)$ is that $\Sigma(v)$ contain all clauses C involving literals incident with v such that the number of complemented literals in C is opposite in parity to $\varepsilon(v)$. Notice that $|\Sigma(v)| = 2^{\delta(v)-1}$ where $\delta(v)$ is the degree of vertex v. Let $\Sigma(G) = \bigcup_{v \in G} \Sigma(v)$.

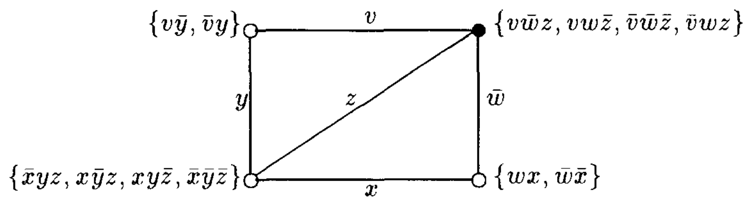
$\{v\bar{y}, \bar{v}y\}$
v
$\{v\bar{w}z, vw\bar{z}, \bar{v}\bar{w}\bar{z}, \bar{v}wz\}$
y
z
$\bar{w}$
$\{\bar{x}yz, x\bar{y}z, xy\bar{z}, \bar{x}\bar{y}\bar{z}\}$
x
$\{wx, \bar{w}\bar{x}\}$

Fig. 2. A graph G with an odd labelling and the associated set of clauses. Solid nodes are labelled 1; the rest are labelled 0.

Summing these equivalences over all $v \in G$, each literal in G occurs exactly twice, so LHS $\equiv 0$, while the graph has an odd labelling, and so RHS $\equiv 1$. The consequence of this contradiction is that the set of clauses $\Sigma(G)$ is inconsistent and therefore has a tree resolution refutation. We shall now prove Tseitin's theorem which describes the equivalence between regular proof trees of $\Sigma(G)$ and edge deletion processes for G.

Definition 3.1. If A is a clause which does not contain the literal $\bar{p}$, then $A - p$ is the same clause with the literal p removed if it occurs.

Definition 3.2. If Γ is a set of clauses, then Γ/p is the result of deleting p from clauses in Γ containing p, and deleting all clauses containing $\bar{p}$. Thus, $\Gamma/p = \{A - p\colon A \in \Gamma, \bar{p} \notin A\}$.

Definition 3.3. If Γ is a set of clauses, then $\Gamma \overset{k}{\vdash} A$ means that there is a size k tree resolution proof of the clause A from clauses in Γ.

Lemma 3.4. *If $\Sigma(G)\Gamma \overset{k}{\vdash} A$ then there is some connected component of G, G', such that $\Sigma(G')\Gamma \overset{k}{\vdash} A$.*

Proof (as in [16], by induction on the proof tree of A). If $A \in \Sigma(G)$ then the connected component is simply the component containing the vertex associated with A; otherwise, A is the resolvant of Bp and $C\bar{p}$. By induction, Bp and $C\bar{p}$ are provable from connected components of G. Also, each of these components must contain a vertex incident with the edge labelled p. Therefore, the components are not disjoint, since they are connected at least by the edge p. □

Definition 3.5. *Transferring the charge* at an edge e means replacing the associated literal p with its complement $\bar{p}$, and complementing the charges at the two nodes incident with e.

Lemma 3.6. *If G' is obtained from G by transferring the charge at an edge e, then $\Sigma(G') = \Sigma(G)$.*

Proof. This is clear since any clause $A \in \Sigma_G(v)$ must satisfy that the number of complemented literals in A is opposite in parity to $\varepsilon_G(v)$. By replacing p with $\bar{p}$ and reversing the charge at p's two endpoints, any such clause A will again satisfy this requirement, and thus $A \in \Sigma_{G'}(v)$. By symmetry we have $\Sigma_G(v) = \Sigma_{G'}(v)$. □

Corollary 3.7. *If G is a connected graph then any two odd labellings of G are equivalent, in the sense that the associated sets of clauses yield isomorphic tree resolution refutations.*

Proof. Since G is connected, there is a path between any pair of nodes in G. We can thus repeatedly transfer the charge along the path between two nodes, effectively complementing only the charges at the endpoints of the path. Since two odd labellings of G differ by an even number of charges, we can transform one labelling into the other by iterating the above process a finite number of times. □

Lemma 3.8. *If $\Gamma \overset{k}{\vdash} A$ where $p \in A$, then $\Gamma/p \overset{k}{\vdash} A - p$.*

Proof (by induction on the proof tree). If $A \in \Gamma$ then $A - p \in \Gamma/p$; otherwise, A is the resolvent of Bq and $C\bar{q}$ where $\Gamma \overset{l}{\vdash} Bq$ and $\Gamma \overset{m}{\vdash} C\bar{q}$. If $p \in B$ then by induction $\Gamma/p \overset{l}{\vdash} Bq - p$; if $p \notin B$ then no clause containing p occurs in the proof tree for Bq because of the regularity condition, so $\Gamma/p \overset{l}{\vdash} Bq - p = Bq$. Similarly, $\Gamma/p \overset{m}{\vdash} C\bar{q} - p$. Therefore, we can resolve $Bq - p$ and $C\bar{q} - p$ to obtain $BC - p = A - p$. □

Lemma 3.9. *If p is the literal labelling edge e and $\Gamma \subseteq \Sigma(G)$ then $\Gamma/p \subseteq \Sigma(G - e)$.*

Proof. Suppose that $C \in \Gamma/p$. Then $C = A - p$ for some clause $A \in \Gamma$, where either p occurs positively in A or does not occur at all in A. If $p \in A$ then $A \in \Sigma(v)$ where v is one of the two vertices in G incident with the edge e. Since A contains $1 - \varepsilon(v)$ negated variables, and p occurs positively, $A - p$ also contains $1 - \varepsilon(v)$ negated variables; hence, $A - p = C \in \Sigma_{G-e}(v)$. If p does not occur in A then $A \in \Sigma_G(v)$ for some vertex v not incident with the edge e. Then $\Sigma_{G-e}(v) = \Sigma_G(v)$ and thus $C = A - p = A \in \Sigma(G - e)$. □

Definition 3.10. If Γ is an inconsistent set of clauses, then $\mathscr{C}(\Gamma)$ is the number of leaves in the optimal tree refutation of Γ. If G is a graph, then $\mathscr{C}(G)$ denotes the size of the optimal tree refutation of $\Sigma(G)$. This definition is permitted because, as we have seen, the actual labelling of the graph is irrelevant.

Theorem 3.11 (Tseitin's theorem). *Let G be a connected graph. Then*

$$\mathscr{C}(G) = \begin{cases} 1 & \textit{if } G \textit{ is a single vertex,} \\ \min\limits_{e \in G} \mathscr{C}^e(G) & \textit{otherwise,} \end{cases}$$

where $\mathscr{C}^e(G)$ is given by

$$\mathscr{C}(G) = \begin{cases} \mathscr{C}(G_1) + \mathscr{C}(G_2) & \textit{if } e \textit{ ruptures } G, \\ 2\mathscr{C}(G') & \textit{otherwise,} \end{cases}$$

where $G' = G - e$, and in the case that deleting e ruptures G, G_1 and G_2 are the two connected components of G'.

Proof (by induction on the number of edges in the graph G). If G consists of a single vertex with no edges, then $\Sigma(G) = \{\phi\}$, and $\mathscr{C}(G) = 1$.

Now, suppose that G has one or more edges. $\phi \notin \Sigma(G)$, so the last step in the refutation is a resolution of, say, p and $\bar{p}$ where p is the literal labelling edge e. There are two cases to consider.

Case 1: Deleting e from G ruptures it into 2 components, G_1 and G_2, where G_1 has an odd labelling and G_2 has an even labelling. We have $C_1 \overset{k}{\vdash} p$ and $C_2 \overset{l}{\vdash} \bar{p}$ where $C_1, C_2 \subseteq \Sigma(G)$. p is the literal labelling edge e, so by Lemma 3.8, $C_1/p \overset{k}{\vdash} \phi$. However, $C_1/p \subseteq \Sigma(G - e) = \Sigma(G_1) \cup \Sigma(G_2)$, and so $\Sigma(G_1) \overset{k}{\vdash} \phi$ since it is provable from a single connected component of G' and G_1 is the only component with an odd labelling.

Now, construct G^* from G by transferring the charge at e. $\Sigma(G^*) = \Sigma(G)$, but this time G_2 has an odd labelling and G_1 an even labelling. $C_2 \overset{l}{\vdash} \bar{p}$ and, proceding as above, we conclude that $\Sigma(G_2) \overset{l}{\vdash} \phi$.

Thus, the complexity of $\mathscr{C}(G) \geqslant \mathscr{C}(G_1) + \mathscr{C}(G_2)$. Notice now that if $C_1/p \overset{m}{\vdash} \phi$ with $m < k + l$, then this proof makes use of some clause $A \in C_1/p$ where $A \notin C_1$, but $Ap \in C_1$ (for otherwise $\Gamma(G) \overset{m}{\vdash} \phi$ contradicting the optimality of the proof tree for $\Sigma(G)$). There is an isomorphic proof tree $C_1 \overset{m}{\vdash} p$ formed by adjoining p to the clauses A in the refutation $C_1/p \overset{m}{\vdash} \phi$ for which $Ap \in C_1$. Thus, we have the equality, $\mathscr{C}(G) = \mathscr{C}(G_1) + \mathscr{C}(G_2)$.

Case 2: Deleting e from G produces a connected graph G'. The proof here is the same as case 1, except that G remains connected after the edge e is deleted, and so $\mathscr{C}(T) = 2\mathscr{C}(G')$. □

Definition 3.12. A *deletion process* on a graph G is a binary tree T whose nodes are connected subgraphs of G such that the root of T is G itself, and each node H in T which is not a single node has an edge $e \in H$ associated with it such that if deleting e ruptures H then the two children of H are the two connected components of $H - e$, and if deleting e does not rupture H then both children of H are $H - e$. The leaves of T are single nodes of G.

By Tseitin's theorem, when studying the size of resolution proof trees for families of graph-based clauses, it is sufficient to consider deletion processes on the graph. If the last step in a refutation is resolving the variables p and $\bar{p}$ together, then there is an isomorphic edge deletion process in which the edge labelled p is deleted first. Since this rule can be applied recursively, it follows that the size of the optimal proof tree is equal to the size of the smallest edge deletion process.

4. Generating expanders

In this section we show the existence of a family of highly expanding graphs, using the techniques of random graphs. The proof is very technical and may be skipped; only the statement of Theorem 4.5 is used in the lower bound proof in the following section.

Notation 4.1. If G is a bipartite graph, then G_L and G_R are the sets of vertices in the left and right sides of G, respectively. If H is a subgraph of G then $H_L = H \cap G_L$ and $H_R = H \cap G_R$.

Definition 4.2. If G is a graph and X is a subset of the vertices of G, then the neighbourhood set of X, $\mathcal{N}(X)$, is the set of all vertices of G adjacent to some node in X. In the case that G is bipartite and $X \subseteq G_L$ or $X \subseteq G_R$, X and $\mathcal{N}(X)$ do not intersect, although this is not true in general.

Definition 4.3. A bipartite graph G, where $|G_L| = |G_R| = n$, is a *λ-expander* if whenever $X \subset G_L$ or $X \subset G_R$ and $|X| \leqslant n/2$ then $|\mathcal{N}(X)| \geqslant (1+\lambda)|X|$.

Definition 4.4. A bipartite graph G, where $|G_L| = |G_R| = n$, is a *strong λ-expander* (see [2]) if whenever $X \subseteq G_L$ or $X \subseteq G_R$ then $|\mathcal{N}(X)| \geqslant (1 + \lambda(1-\alpha))|X|$ where $|X| = \alpha n$.

This definition captures the notion that small sets expand more than large ones. We shall sometimes refer to the expanders in Definition 4.3 as *regular* expanders when it is important to distinguish them from strong expanders. Notice that every strong λ-expander is a regular $\lambda/2$-expander; however, the converse is not true.

We now prove the existence of a family of strong expanders, which have the additional property that very small subsets have an expansion factor close to 2, which is required in the final stages of the lower bound proof in the next section. The expansion factor is $2-\delta$ where δ is specified in the next section. We actually prove that such a family must exist for each $\delta > 0$, although we are interested only in the particular δ of Section 5.

Theorem 4.5. *Fix $\delta > 0$. There exists a family of strong λ-expanders with expansion constant $\lambda = 0.3166$, which in addition satisfies the following property* (P):

There exists $\varepsilon > 0$ such that if $X \subset G$ with $|X| \leqslant \varepsilon n$, then $|\mathcal{N}(X)| \geqslant (2-\delta)|X|$.

Proof (following the method in [3]). Let G be a random bipartite graph on $n+n$ nodes with $3n$ edges, defined by choosing 3 random matchings from G_L to G_R[1]. We shall prove that with nonzero probability, G is a strong λ-expander.

We must show that with nonzero probability, all subsets $X \subseteq G_L$ or $X \subseteq G_R$ have the property that $|\mathcal{N}(X)| \geqslant (1+\lambda(1-\alpha))|X|$, where $\alpha = |X|/n$. Equivalently, we show that the probability is less than $\frac{1}{2}$ that there is a subset $X \subseteq G_L$ and $U \subseteq G_R$ with $|X| = k$ and $|U| < (1+\lambda(1-k/n))k$ such that $|\mathcal{N}(X)| \subseteq U$. The proof that all subsets of G_R expand is symmetric.

Define $P(k)$ = the probability that there is a set of size k which does not expand by a factor of β, where $\beta(k) = 1 + \lambda(1-k/n)$ for large subsets of G and $\beta(k) = 2-\delta$ for small subsets. In what follows, we shall simply write β rather than $\beta(k)$ to simplify the formulas, but it should be emphasized that β is not constant; it is a function of k. Then, the probability that G is not a strong λ-expander is no greater than $2\sum_{k=1}^{n} P(k)$.

For a given k, the number of choices of X and U as defined above are $\binom{n}{k}$ and $\binom{n}{\lceil \beta k \rceil - 1}$, respectively. For given X and U, the probability that, for a given one of the three random matchings. $\mathcal{N}(X) \subseteq U$ is

$$\frac{\binom{\lceil \beta k \rceil - 1}{k}}{\binom{n}{k}}.$$

Thus,

$$P(k) \leqslant \binom{n}{k}\binom{n}{\lceil \beta k \rceil - 1}\left[\frac{\binom{\lceil \beta k \rceil - 1}{k}}{\binom{n}{k}}\right]^3.$$

For small values of k we use the approximations

$$\binom{a}{i} \leqslant \left(\frac{ea}{i}\right)^i \quad \text{and} \quad \frac{\binom{a}{i}}{\binom{b}{i}} \leqslant \left(\frac{a}{b}\right)^i \quad \text{for } a < b.$$

Since the approximation for $\binom{a}{i}$ is both an increasing function of a, and an increasing function of i for small i, we have

$$\begin{aligned} P(k) &\leqslant \left(\frac{en}{k}\right)^k \left(\frac{en}{\beta k}\right)^{\beta k} \left(\frac{\beta k}{n}\right)^{3k} \\ &= \left[\beta e^{1+\beta}\left(\frac{\beta k}{n}\right)^{2-\beta}\right]^k. \end{aligned}$$

[1] Note that in general this gives a multigraph, but all definitions and theorems regarding graph-based clauses hold for multigraphs as well as graphs.

If $k \leqslant en$ then the expression is

$$\leqslant [\beta e^{1+\beta}(\beta\varepsilon)^{2-\beta}]^{\varepsilon n}.$$

If $\beta = 2 - \delta$ then the quantity inside the parentheses is constant, and there is some $\varepsilon > 0$ for which this constant is < 1. The expression is exponential in n and thus is $< 1/(2n)$ for sufficiently large n. This takes care of the additional property (P).

Next we must approximate $P(k)$ for $k > (1 - \varepsilon)n$. As above, we wish to show that the probability is less than $1/(2n)$ that there are sets $X \subset G_{\mathrm{L}}$ and $U \subset G_{\mathrm{R}}$, $|X| = k$, $|U| = \lceil (1 + \lambda(1 - k/n))k \rceil - 1$, such that $\mathcal{N}(X) \subseteq U$. Equivalently, we show that the probability is $< 1/(2n)$ that there are sets $\bar{U} \subset G_{\mathrm{R}}$ and $\bar{X} \subset G_{\mathrm{L}}$, $|\bar{U}| = n - \lceil (1 + \lambda(1 - k/n))k \rceil + 1$, $|\bar{X}| = n - k$, such that $\mathcal{N}(\bar{U}) \subseteq \bar{X}$.

Since $k > (1 - \varepsilon)n$,

$$\begin{aligned} |\bar{U}| &= n - \lceil (1 + \lambda(1 - k/n))k \rceil + 1 \\ &= \lfloor n - (1 + \lambda(1 - k/n))k \rfloor + 1 \\ &< \lfloor n - \alpha n \rfloor + 1 \\ &< \lfloor \varepsilon n \rfloor + 1 \end{aligned}$$

and so $|\bar{U}| \leqslant \varepsilon n$. Applying the result of the preceding proof of property (P), we know that the probability that $|\mathcal{N}(\bar{U})| < (2 - \delta)|\bar{U}|$ is less than $1/(2n)$ for arbitrarily small δ:

$$\begin{aligned} (2 - \delta)|\bar{U}| &\geqslant (2 - \delta)[n - (1 + \lambda(1 - k/n))k] \\ &= (2 - \delta)(1 - \lambda k/n)(1 - k/n)n \\ &> (1 - k/n)n \\ &= n - k. \end{aligned}$$

Thus, the probability that such $\bar{U}$ and $\bar{X}$ exist is $< 1/(2n)$.

Finally, we must approximate $P(k)$ for $\varepsilon n \leqslant k \leqslant (1 - \varepsilon)n$. Hence we show that $nP(k) < \frac{1}{2}$ for such k and for n sufficiently large. Using Stirling's formula, we write

$$\log \binom{n}{k} = n \log n - k \log k - (n - k) \log (n - k) + \mathrm{O}(\log n).$$

Noting also that

$$\log \binom{n-1}{k} = \log \binom{n}{k} + \mathrm{O}(\log n),$$

$$\log \binom{n}{k-1} = \log \binom{n}{k} + \mathrm{O}(\log n),$$

we can write, as before,

$$nP(k) = n\binom{n}{k}\binom{n}{\lceil\beta k\rceil - 1}\left[\frac{\binom{\lceil\beta k\rceil - 1}{k}}{\binom{n}{k}}\right]^3$$

$$= \exp\left\{\log n + \log\binom{n}{\lceil\beta k\rceil - 1} + 3\log\binom{\lceil\beta k\rceil - 1}{k} - 2\log\binom{n}{k}\right\}$$

$$= \exp\{-n\log n + 2\beta k\log\beta k - k\log k - (n-\beta k)\log(n-\beta k)$$
$$- 3(\beta k - k)\log(\beta k - k) + 2(n-k)\log(n-k) + \mathrm{O}(\log n)\}.$$

Substituting $\alpha = k/n$ and simplifying, we have

$$= \exp\{-n\log n + 2\beta\alpha n\log\beta\alpha n - \alpha n\log\alpha n - (n-\beta\alpha n)\log(n-\beta\alpha n)$$
$$-3(\beta\alpha n - \alpha n)\log(\beta\alpha n - \alpha n) + 2(n-\alpha n)\log(n-\alpha n) + \mathrm{O}(\log n)\},$$
$$= \exp\{n\log n[-1 + 2\beta\alpha - \alpha - (1-\beta\alpha) - 3(\beta\alpha - \alpha) + 2(1-\alpha)]$$
$$+ 2\beta\alpha n\log\beta\alpha - \alpha n\log\alpha - (n-\beta\alpha n)\log(1-\beta\alpha)$$
$$- 3(\beta\alpha n - \alpha n)\log(\beta\alpha - \alpha) + 2(n-\alpha n)\log(1-\alpha) + \mathrm{O}(\log n)\}.$$

The first line of this expression is zero, and we are left with

$$= \exp\{n[2\beta\alpha\log\beta\alpha - \alpha\log\alpha - (1-\beta\alpha)\log(1-\beta\alpha)$$
$$- 3(\beta\alpha - \alpha)\log(\beta\alpha - \alpha) + 2(1-\alpha)\log(1-\alpha)] + \mathrm{O}(\log n)\}.$$

The entropy function $H(x) = -x\log x - (1-x)\log(1-x)$, so we can write this as

$$= \exp\{n[H(\beta\alpha) - 2H(\alpha) + 3\beta\alpha\log\beta - 3\alpha(\beta-1)\log(\beta-1)]$$
$$+ \mathrm{O}(\log n)\}.$$

One can show that $H(1/\beta) = \log\beta - (1-1/\beta)\log(\beta-1)$ and so we conclude that the expression is

$$= \exp\{n[H(\beta\alpha) - 2H(\alpha) + 3\beta\alpha H(1/\beta)] + \mathrm{O}(\log n)\}.$$

Letting $F(\alpha) = H(\beta\alpha) - 2H(\alpha) + 3\beta\alpha H(1/\beta)$, it is sufficient to prove that $F(\alpha) < 0$, and is bounded away from 0, for the required values of β. Recall that $\beta(\alpha) = 1 + \lambda(1-\alpha)$ where $\lambda = 0.3166$.

It can be verified numerically that $F(\alpha) < 0$ for $0 < \alpha < 1$. F is continuous on the closed interval $[\varepsilon, 1-\varepsilon]$, so it is bound away from 0. Therefore, $\sum_{k=1}^{n} P(k) < \frac{1}{2}$ for sufficiently large n, so there exists a family of strong λ-expanders.

It can be checked that F has a root if $\lambda \approx 0.3167$, and so the best possible value of λ using these methods lies between 0.3166 and 0.3167.

It is worth noting that this method can be used to generate regular expanders by finding a constant value of β such that $F(\alpha) < 0$ for $0 \leqslant \alpha \leqslant \frac{1}{2}$. It can be verified by taking derivatives that F is a convex function, and so its maximum value occurs at an endpoint. One can also verify that $F(\frac{1}{2}) \geqslant F(\varepsilon)$, and so it suffices to simply set $\alpha = \frac{1}{2}$. The maximum β for which $F(\frac{1}{2}) < 0$ is approximately 1.162835, and thus we can conclude that there is a regular expander with expansion factor $\lambda = 0.162835$.

5. Lower bound for tree resolution

Central to the lower bound proof is the idea that there are graphs with the property that they have relatively few edges but nevertheless require large edge deletion processes. Intuitively, the best strategy for an edge deletion process is to cut the graph roughly into half, to reduce the subsequent amount of duplicated work. Expanders should therefore have this property because they are difficult to cut into half; if $X \subset G$ where $|X_\mathrm{L}| = |X_\mathrm{R}| = n/2$ then $|\mathcal{N}(X_\mathrm{L})|, |\mathcal{N}(X_\mathrm{R})| \geqslant (1 + \lambda)n/2$, and thus the number of edges required to disconnect X from G is at least $\lambda n/2 + \lambda n/2 = \lambda n$.

Strong expanders should require even more work, because after bisecting the original graph, the resulting components retain much of their original expansion. However, a problem arises because there is nothing guaranteeing that the optimal strategy is to bisect the graph exactly. The problem is to express the notion that if little work is done early (by cutting the graph into unequally sized components) then much work must be done later (the larger component will retain most of its original expansion).

There are several lower bounds for resolution which use expanders (see [11, 15]). Something like the $\frac{1}{3} - \frac{2}{3}$ rule is invariably used: in an edge deletion process, there will be some subgraph H of size m where $(\frac{1}{3})n \leqslant m < (\frac{2}{3})n$. A bound is then placed on the number of edges k, that must be deleted to disconnect H from G, and lower bound is expressed in terms of k. Obviously, this is sound reasoning, but unfortunately it loses much information. If $m \approx (\frac{1}{3})n$ then much more work has been done than simply deleting edges between H and $\bar{H}$. Also, it does not account for the work that must subsequently be done within H to complete the edge deletion process.

What follows is an inductive proof that takes all of that into account.

Lemma 5.1. *If G is a graph, then an optimal deletion process for G will have the following property: if H is ruptured into H_1 and H_2 then any edge e deleted from an ancestor of H will not be contained entirely within either H_1 or H_2.*

Proof. Suppose that a deletion process does not have the property. So there is some subgraph H which is ruptured into H_1 and H_2, and an edge e deleted from some ancestor of H which lies entirely in H_1 or H_2. We can rearrange the deletion process by postponing the deletion of this edge until after H_i is ruptured. This has the effect of strictly decreasing the number of nodes in the tree. □

So, we can think of the deletion process as a recursive application of the following rule: delete k edges from G, rupturing it into two connected components, G_1 and G_2, where each of the k edges joins a vertex in G_1 to a vertex in G_2.

Lemma 5.2. *Let G be a d-regular strong λ-expander, with $d \geqslant 3$. If $H \subseteq G$ where $|H| = m = \alpha(2n)$, then $\lambda(1-\alpha)m$ edges must be deleted from G to disconnect H from G.*

Proof. Say that there are $m_l = \alpha_l n$ edges in H_{L}, and $m_r = \alpha_r n$ edges in H_{R}, and w.l.o.g. $\alpha_l \leqslant \alpha_r$. H_{L} has at least $(1+\lambda(1-\alpha_l))m_l$ neighbours, and so H_{L} has at least $(1+\lambda(1-\alpha_l))m_l - m_r$ neighbours outside of H_{R}. Let t be the number of edges between H_{L} and H_{R}. Then

$$\begin{aligned} t &\leqslant dm_l - [(1+\lambda(1-\alpha_l))m_l - m_r] \\ &= (d-1)m_l - \lambda(1-\alpha_l)m_l + m_r. \end{aligned}$$

Therefore, if k is the number of edges required to disconnect H from G, then

$$\begin{aligned} k &= (dm_l - t) + (dm_r - t) \\ &\geqslant -(d-2)m_l + (d-2)m_r + 2\lambda(1-\alpha_l)m_l \\ &= [2(d-2)r + 2\lambda(1-\alpha+r)(\alpha-r)]n, \end{aligned}$$

where $\alpha_l = \alpha - r$ and $\alpha_r = \alpha + r$ for $r \geqslant 0$. Let

$$g(r) = 2(d-2)r + 2\lambda(1-\alpha+r)(\alpha-r).$$

Differentiating with respect to r, we have

$$g'(r) = 2(d-2) - 2\lambda + 4\lambda\alpha - 4\lambda r.$$

Since $d \geqslant 3$, $\lambda < 1$ and $r \leqslant \alpha$, we have $g'(r) \geqslant 0$ for all r. This implies that the minimum value for $g(r)$ occurs when $r = 0$, i.e., when $\alpha_l = \alpha_r$. Hence, we have $k \geqslant g(0)n = 2\lambda(1-\alpha)\alpha n = \lambda(1-\alpha)m$. □

Lemma 5.3. *Let G be a strong λ-expander on $2n$ nodes. If $H \subseteq G$ where $|H| = \alpha(2n)$ and l edges have been deleted from H in the process of disconnecting it from G, then*

$$\mathscr{C}(H) \geqslant 2^{\lambda(1-\alpha/3)\alpha n - l/2 + O(1)}.$$

Proof (by induction on the size of H). Suppose that the optimal deletion strategy for H is splitting it into H_1 and H_2, where k edges from H_1 to H_2 must be deleted to accomplish this. Let $m_1 = |H_1| = \alpha_1(2n)$ and $m_2 = |H_2| = \alpha_2(2n)$. Of the l edges deleted from H, l_1 of them come from H_1 and l_2 come from H_2. So $\alpha = \alpha_1 + \alpha_2$ and

$l = l_1 + l_2$. Ignoring the O(1) for now, we have

$$\begin{aligned}\mathscr{C}(H) &= 2^{k-1}[\mathscr{C}(H_1) + \mathscr{C}(H_2)]\\ &\geqslant 2^{k-1}\left[2^{\lambda(1-\alpha_1/3)\alpha_1 n-(l_1+k)/2} + 2^{\lambda(1-\alpha_2/3)\alpha_2 n-(l_2+k)/2}\right]\\ &= 2^{\lambda(1-\alpha_1/3)\alpha_1 n-l_1/2+k/2-1} + 2^{\lambda(1-\alpha_2/3)\alpha_2 n-l_2/2+k/2-1}.\end{aligned}$$

From Lemma 5.2 we know that $k \geqslant \lambda(1-\alpha_1)m_1 - l_1$ and $k \geqslant \lambda(1-\alpha_2)m_2 - l_2$, which gives

$$\begin{aligned}&\geqslant 2^{\lambda(1-\alpha_1/3)\alpha_1 n-l_1/2+\lambda(1-\alpha_2)\alpha_2 n-l_2/2-1}\\ &\quad + 2^{\lambda(1-\alpha_2/3)\alpha_2 n-l_2/2+\lambda(1-\alpha_1)\alpha_1 n-l_1/2-1}\\ &= 2^{\lambda\alpha n-\lambda n(\alpha_1^2/3+\alpha_2^2)-l/2-1} + 2^{\lambda\alpha n-\lambda n(\alpha_2^2/3+\alpha_1^2)-l/2-1}.\end{aligned}$$

It is easy to show that

$$\frac{\alpha_1^2}{3} + \alpha_2^2 = \frac{(\alpha_1+\alpha_2)^2}{3} + \frac{2}{3}\alpha_2(\alpha_2-\alpha_1)$$

and hence

$$\begin{aligned}\mathscr{C}(H) &\geqslant 2^{\lambda(1-\alpha/3)\alpha n-2/3\lambda n\alpha_2(\alpha_2-\alpha_1)-l/2-1}\\ &\quad + 2^{\lambda(1-\alpha/3)\alpha n-2/3\lambda n\alpha_1(\alpha_1-\alpha_2)-l/2-1}\\ &= 2^{\lambda(1-\alpha/3)\alpha n-l/2-1}\left[2^{2/3\lambda n\alpha_2(\alpha_1-\alpha_2)} + 2^{2/3\lambda n\alpha_1(\alpha_2-\alpha_1)}\right]\end{aligned}$$

The last expression is always greater than 1. We must therefore bound the number of times it can be less than 2, thereby eliminating the -1 earlier in the exponent, and giving us the O(1) as in the statement of the lemma.

As was noted in the proof of the existence of strong expanders, for each $\delta > 0$, there is some constant ε and a family of strong λ-expanders for which all sets of size less than εn expand by a factor of $2-\delta$. We shall show that if $\alpha_2 \leqslant \varepsilon$ then the -1 does not appear in the expression. This implies that the number of times the -1 is introduced is bounded by $1/\varepsilon$.

To show this, we examine the following three cases

Case 1: $\alpha_2 = 1/(2n)$. Here, H_2 is a single node and degree of each vertex in G is 3, so $k = 3 - l_2$. Hence,

$$\begin{aligned}\mathscr{C}(H) &\geqslant 2^{\lambda(1-\alpha/3)\alpha n-l/2-\lambda/2+\lambda n/3(\alpha^2-\alpha_1^2)+3/2-1}\\ &> 2^{\lambda(1-\alpha/3)\alpha n-l/2}.\end{aligned}$$

Case 2: $\alpha_2 = 1/n$. Here H_2 is exactly two nodes, so $k = 4 - l_2$:

$$\begin{aligned}\mathscr{C}(H) &\geqslant 2^{\lambda(1-\alpha/3)\alpha n-l/2-\lambda+\lambda n/3(\alpha^2-\alpha_1^2)+4/2-1}\\ &> 2^{\lambda(1-\alpha/3)\alpha n-l/2}.\end{aligned}$$

Case 3: $3/(2n) \leqslant \alpha_2 \leqslant \varepsilon$. Then $k \geqslant (2-\delta)\alpha_2 n - l_2$:

$$\begin{aligned}\mathscr{C}(H) &\geqslant 2^{\lambda(1-\alpha/3)\alpha n - l/2 - \lambda\alpha_2 n + \lambda n/3(\alpha^2-\alpha_1^2) + (2-\delta)\alpha_2 n/2 - 1}\\ &> 2^{\lambda(1-\alpha/3)\alpha n - l/2 + (3/2)((2-\delta)/2-\lambda) - 1}\\ &> 2^{\lambda(1-\alpha/3)\alpha n - l/2}.\end{aligned}$$

The last line follows because $\lambda < \frac{1}{3}$ and so the expression $(2-\delta)/2 - \lambda$ can be made larger than $\frac{2}{3}$ by choosing δ small enough, and from property (P) in Theorem 4.5 we know that δ can be chosen arbitrarily small. □

Corollary 5.4. *If G is a strong λ-expander, then*

$$\mathscr{C}(G) \geqslant 2^{(2/3)\lambda n + O(1)}.$$

Note that this lower bound is expressed in terms of the number of vertices in the graph. We wish to express the bound in terms of the number of variables in the input set. Substituting the best known value for λ from the previous section, and noting that the number of variables in the input set $N = 3n$, we have the following lower bound for tree resolution:

$$2^{\lambda(2/3)N/3} \approx 2^{0.070355N}.$$

Some may feel that simply counting the number of variables in the set of input clauses is an unreasonable way of measuring the size of the problem; the number of clauses in the input set is a more realistic measure of the problem's input size. If we let M be the number of clauses in the input set, then $M = 8n$, and so we have the following lower bound:

$$2^{\lambda(2/3)M/8} \approx 2^{0.026383M}.$$

Note that the above argument does not depend in any way on the degree of the graph. However, as we now show, degree 3 graphs give the best lower bound. Degree 2 graphs provide sets of 2SAT clauses which are solvable in polynomial time (see [7]). Increasing the degree of the graph will allow us graphs with a higher expansion factor, and an apparently better lower bound. However, increasing the degree by one will also double the number of clauses in the input set. Since increasing the degree by one will not double the expansion factor, we know that degree 3 graphs will provide the best lower bound in terms of the number of input clauses.

Let us for a moment compare this result with the lower bound for regular resolution in [11]. In the case $d = 3$, the best expansion factor known for a regular λ-expander is $\lambda \approx 0.162835$ (see the previous section; see also [5, 3]). Galil's bound is

$$\geqslant 2^{\lambda N/96} \approx 2^{0.001696198N}.$$

Of course, this is not a fair comparison because regular resolution is more powerful than tree resolution.

6. Conclusions

The expanders generated in Section 4 are not the best possible; it is possible to obtain even more highly expanding graphs, although the expansion factor will not be of the simple form

$$|\mathcal{N}(X)| \geqslant (1 + \lambda(1 - \alpha))|X|$$

for $|X| = \alpha n$. So it would be necessary to reformulate Lemma 5.3 to accommodate the more complex form of the expansion factor. The best possible expansion factor would come from solving the equation

$$F(\alpha) = H(\beta\alpha) - 2H(\alpha) + 3\beta\alpha H(1/\beta) = 0$$

for β at each value of α between ε and 1. Of course, it would be too much to expect the solution to be an analytic function $\beta(\alpha)$. However, something greater than $\beta(\alpha) = 1 + \lambda(1 - \alpha)$ is possible. The optimal solution should then be approximately

$$\sum_{k \geqslant 1} \psi(k),$$

where $\psi(k)$ is given by the following recurrence:

$$\psi(k) = 2\left(\beta\left(\frac{1}{2^k}\right) - 1\right)\frac{n}{2^k} - \sum_{0<i<k} \frac{\psi(i)}{2^{k-i}}.$$

If $\beta(\alpha) = 1 + (1 - \alpha)\lambda$ then the above expression is exactly equal to the expression in Corollary 5.1.

The lower bound obtained in this article is approximately $2^{0.070355N}$, where N is the number of variables in the input set. The bound is on the size of a resolution proof tree, but it also applies to a wide class of backtracking algorithms for solving SAT. The best known upper bound for SAT is due to Monien and Speckenmeyer [13]. They solve 3SAT in time no greater than $2^{0.6943N}$ using a backtracking algorithm which can be shown to be identical to traversing a resolution tree.

It is believed by the author that providing a deterministic branching algorithm that runs in a certain time bound is more difficult than showing the existence of tree resolution proofs of a certain size, because when analyzing an algorithm, we measure not only the number of resolutions, but also the other computational aspects of the algorithm. In particular, we must measure the time spent computing the next variable on which to branch. This problem has been studied extensively by the automatic theorem proving community, and many heuristics for choosing the variable on which to branch have been proposed.

The problem of deciding the best branching strategy is probably intractable, and hence it is likely that no algorithm will ever come very close to the optimal lower bound on the size of resolution proof trees. There is a trade-off: the better the branching strategy we want, the more work the algorithm will have to do to find that strategy. In other words, if a deterministic algorithm for solving SAT which effectively generates a tree reso[illegible]n proof spends only a polynomial amount of time choosing the next variab[illegible] it will be worse than optimal because of its poor variable selection heuristi[illegible]akes the time to find the best variable at each stage, then again it will not be optim[illegible]ecause it spends so much time figuring out a good variable strategy in addition t[illegible]oing the resolution itself. The gap between this paper's lower bound and the upper bound in [13] can certainly be narrowed somewhat, but it would be surprising if they were brought significantly closer together. The current state of knowledge does not allow us to study this discrepancy quantitatively.

References

[1] M. Ajtai, The complexity of the pigeonhole principle, in: Proceedings 29th Annual Symposium on the Foundations of Computer Science (1988) 346–355.

[2] N. Alon, Eigenvalues and expanders, Combinatorica 6 (1986) 83–96.

[3] L.A. Bassalygo, Asymptotically optimal switching circuits, Problems of Inform. Transmission 17 (1981) 206–211

[4] S. Buss, Polynomial size proofs of the propositional pigeonhole principle, J. Symbolic Logic 52 (1987) 916–927.

[5] F.R.K. Chung, On concentrators, superconcentrators, generalizers, and nonblocking networks, The Bell System Tech. J. 58 (1979) 1765–1777.

[6] V. Chvátal and E. Szemerédi, Many hard examples for resolution, ACM J. 35 (1988) 759–768.

[7] S. Cook, The complexity of theorem proving procedures, Proceedings 3rd Annual Symposium on the Theory of Computing, Shaker Heights, OH, May 3–7 (ACM, New York, 1971).

[8] S. Cook and R. Reckhow, The relative efficiency of propositional proof systems, J. Symbolic Logic 44 (1977) 36–50.

[9] J. Franco and M. Paull, Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem, Discrete Appl. Math. 5 (1983) 77–87

[10] Z. Galil, On the complexity of regular resolution and the Davis–Putnam procedure, Theoret. Comput. Sci. 4 (1977) 23–46.

[11] A. Goldberg, P. Purdom and C. Brown, Average time analyses of simplified Davis–Putnam procedures, Inform. Process. Lett. 15 (1982) 72–75.

[12] A. Haken, The intractibility of resolution, Theoret. Comput. Sci. 39 (1985) 297–308.

[13] B. Monien and E. Speckenmeyer, Solving satisfiability in less than 2^n steps, Discrete Appl. Math. 1 (1985) 287–295.

[14] G.S. Tseitin, On the complexity of derivation in propositional calculus, in: A.O. Slisenko, ed., Studies in Constructive Mathematics and Mathematical Logic, Part II (translated from Russian) (Consultants Bureau, New York, 1970) 115–125.

[15] A. Urquhart, Hard examples for resolution, ACM J. 34 (1987) 209–219.

[16] A. Vellino, The complexity of automated reasoning, Ph.D. Thesis, University of Toronto, Toronto, Ontario, Canada (1989).