Graph families, pebbling formulas and CNF encodings

Matti Järvisalo[1], Arie Matsliah[2], Jakob Nordström[3], and Stanislav Živný[4]

[1] Department of Computer Science & HIIT, University of Helsinki, Finland
[2] IBM Research and Technion, Haifa, Israel
[3] KTH Royal Institute of Technology, Sweden
[4] University of Oxford, United Kingdom

1 Pebbling Formulas

To study the proof complexity measures of length, width and space, and to relate them to the practical hardness of CNF formulas, we focus on so-called *pebbling formulas* (also known as *pebbling contradictions*). Our main motivation for using pebbling formulas is that, as explained below, recent theoretical results allow us to construct such formulas with varying (and fully specified) space complexity properties while keeping the length and width complexity constant.

Pebbling formulas are so called since they encode instances of *pebble games* played on directed acyclic graphs, or DAGs (see the survey [22] for more information). The pebbling formula over G associates one variable with each vertex, postulates the source vertices (with no incoming edges) to be true and the (unique) sink vertex (with no outgoing edges) to be false, and then specifies that truth propagates from the sources to the sink. More formally, as defined in [10] the pebbling formula Peb_G consists of:

- for all source vertices s, a unit clause s (*source axioms*),
- for all non-sources v with incoming edges from the immediate predecessors in $pred(v)$, the clause $\bigvee_{u \in pred(v)} \overline{u} \vee v$ (*pebbling axioms*),
- for the (unique) sink z, the unit clause $\overline{z}$ (*sink axiom*).

If G has n vertices and max fan-in ℓ, then Peb_G is an unsatisfiable $(1+\ell)$-CNF formula with $n+1$ clauses over n variables. For the graphs in this paper, we have $\ell = 2$.

Pebbling formulas are not of much use to us as such, however—in particular, they are very easy with respect to all proof complexity measure we have discussed, and are easily seen to be solvable simply by unit propagation. But they can be transformed into much more interesting formulas by simple substitution. Let us elaborate on this.

Given any CNF formula F, we can fix a Boolean function $f\colon \{0,1\}^d \mapsto \{0,1\}$ and substitute every variable x in F by $f(x_1, \ldots, x_d)$, where $x_1, \ldots, x_d$ are new variables that do not appear anywhere else. Then we expand this out to get an equivalent CNF formula over this new set of variables. For a small example, if we let $\oplus$ denote binary exclusive or, then the clause $\overline{x} \vee y$ after substitution becomes $\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)$ which is expanded out to the set of clauses

$$\{x_1 \vee \overline{x}_2 \vee y_1 \vee y_2,\ x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2,\ \overline{x}_1 \vee x_2 \vee y_1 \vee y_2,\ \overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2\}\ . \quad (1)$$

(For general f there might be some choices in exactly how to do this expansion, but such implementation details do not affect this discussion so we ignore them for brevity.)

Table 1. DAG families and properties of the resulting CNF formula families

Name	Description	Space cplx
pyr$\langle h \rangle$seq	Sequence of pyramid graphs of (constant) height h	$\Theta(h)$
width$\langle w \rangle$chain	Chain graph of (constant) width w	$\Theta(w)$
bintree	Complete binary tree	$\Theta(\log n)$
pyrofpyr	Pyramid of height $\sqrt[4]{n}$ with each node expanded to pyramid	$\Theta(\sqrt[4]{n})$
pyrseqsqrt	Sequence of pyramids of (growing) height $\sqrt[4]{n}$	$\Theta(\sqrt[4]{n})$
pyramid	Pyramid graph of (growing) height $\sqrt{n}$	$\Theta(\sqrt{n})$
gtb	DAGs from [26] with butterfly graphs as superconcentrators	$\Theta(n/\log^2 n)$

Every DAG G has a *pebbling price* measuring how much space is needed to pebble G. As shown in [12, 23], making substitutions in pebbling formulas using *robust* functions $f\colon \{0,1\}^d \mapsto \{0,1\}$, meaning that the truth value of f can never be fixed by just assigning to one variable, yields substituted CNF formulas for which the *space complexity of the formula is equal to the pebbling price of* G.[5] The canonical example of a robust function is exclusive or. A non-robust function is ordinary or, but [24, 25] show that even for this function the same connection holds for certain families of graphs.

This means that if we pick the right graphs G, we can generate CNF formulas with known space complexity. In addition, it is easy to show that any pebbling formula, even after substitution, can be refuted in (small) constant width and (small) linear length. Thus, in this way we can get benchmark formulas that are uniformly *very* easy with respect to length and width, but for which the space complexity varies. Such formulas would seem like excellent benchmarks for testing whether space or width is a good measure of hardness in practice. If width is the more important parameter, then we would expect all formulas to be essentially equally easy. If space is more relevant, then we would expect running time to correlate with space complexity. Carrying out large-scale experiments along these lines and analyzing the results is the main practical contribution of this paper. When designing such experiments, one needs to choose (a) graphs from which to generate the benchmarks, and (b) substitution functions to apply. We discuss this next.

An overview of our choice of graph families and their space complexities is given in Table 1. Let us first explain two important building blocks. A *pyramid* of height h is a layered DAG with $h+1$ layers, where there is one vertex in the highest layer, two vertices in the next layer, et cetera, down to $h+1$ vertices in the lowest layer. The ith vertex at layer L has incoming edges from the ith and $(i+1)$st vertices at layer $L-1$. A *chain* of width w is a layered graph with w vertices at each layer, and with vertices i and $i-1$ at layer $L-1$ having edges to vertex i in layer $L \pmod w$.

To obtain two different types of graphs of constant space complexity, we consider sequences of pyramids of constant height h with the sink of each pyramid connected to the leftmost source of next pyramid (pyr$\langle h \rangle$seq) and chains of constant width w (width$\langle w \rangle$chain). Another graph family that should yield easy formulas are complete binary trees (bintree), the space complexity of which is equal to the height of the tree.

[5] Actually, this is oversimplifying a bit—the space will be somewhere in between the deterministic black and the (smaller) non-deterministic black-white pebbling price, but these two measures are within a small constant factor for all graphs in this paper so this is immaterial.

Table 2. Substitution functions

Name	Description	Output	CNF encoding (for $d = 3$ variables)
or_d	OR of d vars	$x_1 \vee \cdots \vee x_d$	$x_1 \vee x_2 \vee x_3$
xor_d	parity of d vars	$x_1 \oplus \cdots \oplus x_n$	$x_1 \vee x_2 \vee x_3$, $x_i \bigvee_{j \neq i} \overline{x}_j$, $i = 1, 2, 3$
maj_d	majority of d vars	$2(x_1 + \cdots + x_d) > d$	$x_1 \vee x_2$, $x_1 \vee x_3$, $x_2 \vee x_3$
eq_d	all d vars equal	$x_1 = \cdots = x_d$	$x_1 \vee \overline{x}_2$, $\overline{x}_1 \vee x_2$, $x_1 \vee \overline{x}_3$, $\overline{x}_1 \vee x_3$
e1_d	exactly one of d	$x_1 + \cdots + x_d = 1$	$x_1 \vee x_2 \vee x_3$, $\overline{x}_1 \vee \overline{x}_2$, $\overline{x}_1 \vee \overline{x}_3$, $\overline{x}_2 \vee \overline{x}_3$
s_id	if-then-else	$x_1 ? x_2 : x_3$	$\overline{x}_1 \vee x_2$, $x_1 \vee x_2$

To get "medium-hard" DAGs, we use pyramids in two different ways. In pyramid-of-pyramids graphs (pyrofpyr) we take a pyramid of height h and expand each of its vertices v to pyramid of same height h with sink z_v. Every incoming edge to v is drawn to all sources of the pyramid, and all outgoing vertices from v are drawn from z_v. It is an easy argument that the space complexity is roughly $2h$ and the size of the graph is roughly h^4, so the space complexity grows like $\sqrt[4]{n}$ for graphs of size n. Another way of getting graphs of the same space complexity is to use the same construction as in pyr$\langle h \rangle$seq above but employ graphs of height $\sqrt[4]{n}$. These are our pyrseqsqrt graphs.

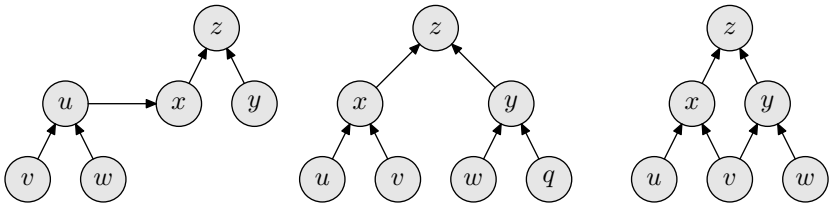
Finally to get "really hard" graphs we consider two well-known graph families. The first one is simply pyramids of height (and hence space complexity) $\sqrt{n}$. Our second hard family is from [26], which exhibits graphs with maximal space complexity $\Theta(n/\log n)$. These graphs cannot quite be used as-is, however. The bound in [26] is asymptotic with the smallest instances having huge size (due to the need for so-called *superconcentrators* of linear size). Therefore, we modify the construction to use much simpler, but asymptotically worse, superconcentrators made from butterfly graphs. It is not hard to verify that the proofs in [26] still go through, and we get much smaller graphs (gtb) that we can actually use to generate CNF formulas, at the price of paying a log factor in the space complexity.

When choosing the substitution functions to apply for pebbling formulas generated from graphs in these families, we have two major concerns. On the one hand, we would like the functions to be robust (as explained above). On on the other hand, however, we do not want too large a blow-up in formula size when substituting functions for variables. Again due to space constraints, we cannot go into much details, but Table 2 presents our choice of substitution functions, which seem to provide a good trade-off between the two concerns, and describes their CNF encodings. Note that we also use binary or, which provably preserves the space complexity (albeit with worse guarantees for the hidden constants) for all graph families used here except the gtb family.

Examples of small graphs are depicted in Figure 2.

$$
\begin{aligned}
&(u_1 \vee u_2) \\
\wedge\ &(v_1 \vee v_2) \\
\wedge\ &(w_1 \vee w_2) \\
\wedge\ &(\overline{u}_1 \vee \overline{v}_1 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_1 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_2 \vee \overline{v}_1 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_2 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{v}_1 \vee \overline{w}_1 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_1 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_2 \vee \overline{w}_1 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_2 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{x}_1 \vee \overline{y}_1 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_2 \vee \overline{y}_1 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_2 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &\overline{z}_1 \\
\wedge\ &\overline{z}_2
\end{aligned}
$$

Fig. 1. The substitution pebbling formula using binary disjunction for the pyramid graph Π_2.



(a) pyr$\langle 1 \rangle$seq $P = 2$ (b) bintree $P = 2$ (c) pyramid $P = 2$

Fig. 2. Examples of pebbling graphs