Finding Efficient Circuits for Ensemble Computation*

Matti Järvisalo[1], Petteri Kaski[2], Mikko Koivisto[1], and Janne H. Korhonen[1]

[1] HIIT & Department of Computer Science, University of Helsinki, Finland
[2] HIIT & Department of Information and Computer Science, Aalto University, Finland

Abstract. Given a Boolean function as input, a fundamental problem is to find a Boolean circuit with the least number of elementary gates (AND, OR, NOT) that computes the function. The problem generalises naturally to the setting of *multiple* Boolean functions: find the smallest Boolean circuit that computes *all* the functions simultaneously. We study an NP-complete variant of this problem titled *Ensemble Computation* and, especially, its relationship to the Boolean satisfiability (SAT) problem from both the theoretical and practical perspectives, under the two monotone circuit classes: OR-circuits and SUM-circuits. Our main result relates the existence of nontrivial algorithms for CNF-SAT with the problem of rewriting in subquadratic time a given OR-circuit to a SUM-circuit. Furthermore, by developing a SAT encoding for the ensemble computation problem and by employing state-of-the-art SAT solvers, we search for concrete instances that would witness a substantial separation between the size of optimal OR-circuits and optimal SUM-circuits. Our encoding allows for exhaustively checking all small witness candidates. Searching over larger witness candidates presents an interesting challenge for current SAT solver technology.

1 Introduction

A fundamental problem in computer science both from the theoretical and practical perspectives is program optimisation, i.e., the task of finding the most efficient sequence of elementary operations that carries out a specified computation. As a concrete example, suppose we have eight variables $x_1, x_2, \ldots, x_8$ and our task is to compute each of the eight sums depicted in Fig. 1. What is the minimum number of SUM gates that implement this computation?

This is an instance of a problem that plays a key role in Valiant's study [18] of circuit complexity over a monotone versus a universal basis; Fig. 1 displays Valiant's solution. More generally, the problem is an instantiation of the NP-complete *Ensemble Computation* problem [8]:

> *(SUM-)Ensemble Computation.* Given as input a collection $\mathcal{Q}$ of nonempty subsets of a finite set P and a nonnegative integer b, decide (yes/no) whether

* This research is supported in part by Academy of Finland (grants 132812 and 251170 (MJ), 252083 and 256287 (PK), and 125637 (MK)), and by Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (JK).

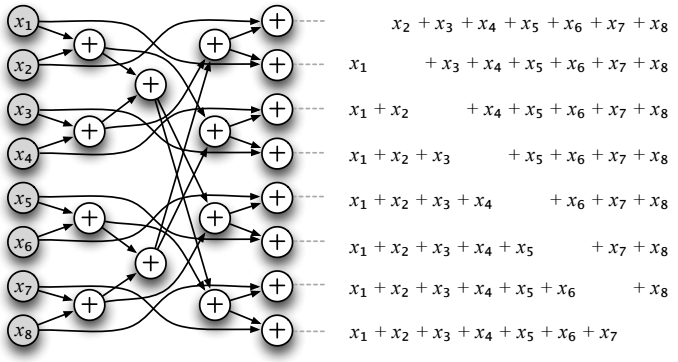

Fig. 1. An instance of ensemble computation (right) and a circuit that solves it (left).

there is a sequence

$$Z_1 \leftarrow L_1 \cup R_1, \quad Z_2 \leftarrow L_2 \cup R_2, \quad \ldots \quad , \quad Z_b \leftarrow L_b \cup R_b$$

of union operations, where

(a) for all $1 \leq j \leq b$ the sets L_j and R_j belong to $\{\{x\} : x \in P\} \cup \{Z_1, Z_2, \ldots, Z_{j-1}\}$,
(b) for all $1 \leq j \leq b$ the sets L_j and R_j are disjoint, and
(c) the collection $\{Z_1, Z_2, \ldots, Z_b\}$ contains $\mathcal{Q}$.

It is also known that *SUM-Ensemble Computation* remains NP-complete even if the requirement (b) is removed, that is, the unions need not be disjoint [8]; we call this variant *OR-Ensemble Computation*. Stated in different but equivalent terms, each set A in $\mathcal{Q}$ in an instance of *SUM-Ensemble Computation* specifies a subset of the variables in P whose sum must be computed. The question is to decide whether b arithmetic gates suffice to evaluate all the sums in the ensemble. An instance of *OR-Ensemble Computation* asks the same question but with sums replaced by ORs of Boolean variables, and with SUM-gates replaced by OR-gates. We will refer to the corresponding circuits as *SUM-circuits* and *OR-circuits*.

Despite the fundamental nature of these two variants of monotone computation, little seems to be known about their relative power. In particular, here we focus the following open questions:

(Q1) Given an OR-circuit for a collection $\mathcal{Q}$, how efficiently can it be rewritten as a SUM-circuit?
(Q2) Are there collections $\mathcal{Q}$ that require a significantly larger SUM-circuit than an OR-circuit?

Answering these questions would advance our understanding of the computational advantage of, in algebraic terms, idempotent computation (e.g. the maximum of variables) over non-idempotent computation (e.g. the sum of variables); the ability to express the

former succinctly in terms of the latter underlies recent advances in algebraic and combinatorial algorithms [2]. Interestingly, it turns out that the questions have strong connections to Boolean satisfiability (SAT) both from the theoretical and practical perspectives, as will be shown in this paper.

As the main theoretical contribution, we establish a connection between (Q1) and the existence of non-trivial algorithms for CNF-SAT. In particular, we show (Theorem 2) that the existence of a subquadratic-time rewriting algorithm implies a nontrivial algorithm for general CNF-SAT (without restrictions on clause length), i.e., an algorithm for CNF-SAT that runs in time $O(2^{cn}m^2n)$ for a constant $0 < c < 1$ that is independent of the number of variables n and the number of clauses m. It should be noted that the existence of such an algorithm for CNF-SAT is a question that has attracted substantial theoretical interest recently [3,14,16,21]. In particular, such an algorithm would contradict the *Strong Exponential Time Hypothesis* [11], and would have significant implications also for the exponential-time complexity of other hard problems beyond SAT. Intuitively, our result suggests that the relationship of the two circuit classes may be complicated and that the difference in the circuit sizes could be large for some collections $\mathcal{Q}$. Furthermore, we show (Proposition 2) that our main result is tight in the sense that (Q1) admits an quadratic-time algorithm.

Complementing our main theoretical result, we address (Q2) from the practical perspective. While it is easy to present concrete instances for which the difference in size between optimal SUM-circuits and OR-circuits is small, finding instances that witness even a factor-2 separation between the number of arithmetic gates is a non-trivial challenge. In fact, our best construction (Theorem 1) achieves this factor only asymptotically, leaving open the question whether there are small witnesses achieving factor 2. As the main practical contribution, we employ state-of-the-art SAT solvers for studying this witness finding task by developing a SAT encoding for finding the optimal circuits for a given ensemble. We show experimentally that our encoding allows for exhaustively checking all small witness candidates. On the other hand, searching over larger witness candidates presents an interesting challenge for current SAT solvers.

As for related earlier work, SAT solvers have been suggested for designing small circuits [4,6,7,12,13], albeit of different types than the ones studied in this work. However, our focus here is especially in circuits implementing an *ensemble* of Boolean functions. A further key motivation that sets this work apart from earlier work is that our interest is not only to find efficient circuits, but also to discover witnesses (ensembles) that separate SUM-circuits and OR-circuits.

2 OR-Circuits, SUM-Circuits, and Rewriting

We begin with some key definitions and basic results related to OR- and SUM-circuits and the task of rewriting an OR-circuit into a SUM-circuit: We show that a SUM-circuit may require asymptotically at least twice as many arithmetic gates as an OR-circuit, and present two rewriting algorithms, one of which rewrites a given OR-circuit with g gates in $O(g^2)$ time into a SUM-circuit. In particular, a SUM-circuit requires at most g times as many arithmetic gates as an OR-circuit.

2.1 Definitions

For basic graph-theoretic terminology we refer to West's introduction [19]. A *circuit* is a directed acyclic graph C whose every node has in-degree either 0 or 2. Each node of C is a *gate*. The gates of C are partitioned into two sets: each gate with in-degree 0 is an *input gate*, and each gate with in-degree 2 is an *arithmetic gate*. The *size* of C is the number $g = g(C)$ of gates in C. We write $p = p(C)$ for the number of input gates in C. For example, the directed acyclic graph depicted on the left in Fig. 1 is a circuit with 26 gates that partition into 8 input gates and 18 arithmetic gates.

The *support* of a gate z in C is the set of all input gates x such that there is a directed path in C from x to z. The *weight* of a gate z is the size of its support. All gates have weight at least one, with equality if and only if a gate is an input gate. For example, in Fig. 1 the five columns of gates consist of gates that have weight 1, 2, 4, 6, and 7, respectively.

In what follows we study two classes of circuits, where the second class is properly contained within the first class. First, every circuit is an *OR-circuit*. Second, a circuit C is a *SUM-circuit* if for every gate z and for every input gate x it holds that there is at most one directed path in C from x to z.

We adopt the convention of using the operator symbols "$\vee$" and "$+$" on the arithmetic gates to indicate the type of a circuit. Fig. 2 below displays an example of both types of circuits. We observe that the circuit on the left in Fig. 2 is not a SUM-circuit because the bottom right gate can be reached from the input x_1 along two distinct directed paths.

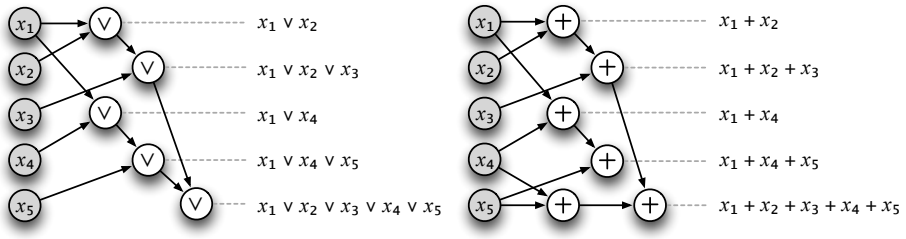

Fig. 2. An OR-circuit (left) and a SUM-circuit (right).

Let $(P, \mathcal{Q})$ be an instance of ensemble computation, that is, let P be a finite set and let $\mathcal{Q}$ be a set of nonempty subsets of P. We adopt the convention that for a SUM-ensemble all circuits considered are SUM-circuits, and for an OR-ensemble all circuits considered are OR-circuits. We say that a circuit C *solves* the instance $(P, \mathcal{Q})$ if (a) the set of input gates of C is P; and (b) for each $A \in \mathcal{Q}$, there exists a gate in C whose support is A. The *size* of the solution is the size of C. A solution to $(P, \mathcal{Q})$ is *optimal* if it has the minimum size over all possible solutions. A circuit C' *implements* a circuit C if for every gate z of C there is a gate z' of C' such that z and z' have the same support. A *circuit rewriting algorithm* takes as input a circuit C and outputs (i) a circuit C' that implements C; and (ii) a mapping $z \mapsto z'$ that identifies each gate z in C with a corresponding gate z' in C'.

2.2 Bounds for Separation

The size of an optimal solution to an instance $(P, \mathcal{Q})$ is dependent on whether we are considering an OR-ensemble or a SUM-ensemble. To see this, let us consider Fig. 2. Observe that both circuits solve the same instance $(P, \mathcal{Q})$, but only the circuit on the right is a SUM-circuit. We claim that both circuits are optimal. Indeed, observe that the instance has five distinct sets of size at least 2. At least one arithmetic gate is required for each distinct set of size at least 2. Thus, the circuit on the left in Fig. 2 is optimal. Analogously, on the right in Fig. 2 at least four arithmetic gates are required to compute the first four sets in the instance, after which at least two further SUM-gates are required to produce the fifth set because the first four sets intersect pairwise.

The following construction shows that asymptotically (that is, by taking a large enough h and w) at least twice the number of arithmetic gates may be required in an optimal SUM-circuit compared with an optimal OR-circuit.

Theorem 1. *For all $h, w = 1, 2, \ldots$ there exists an ensemble whose optimal OR-circuit has $(h + 1)w - 1$ arithmetic gates and whose optimal SUM-circuit has $(2w - 1)h$ arithmetic gates.*

Proof. Take $P = \{x_0\} \cup \{x_{i,j} : i = 1, 2, \ldots, h; j = 1, 2, \ldots, w\}$ and let $\mathcal{Q}$ consist of the following sets. For each $j = 1, 2, \ldots, w$ and for each $i = 1, 2, \ldots, h$, insert the set $\{x_0, x_{1,j}, x_{2,j}, \ldots, x_{i,j}\}$ to $\mathcal{Q}$. Let us say that this set belongs to *chain* j. Finally, insert the set P into $\mathcal{Q}$. Let us call this set the *top*. In total $\mathcal{Q}$ thus has $hw + 1$ sets, and the largest set (that is, the top) has size $hw + 1$.

Every OR-circuit that solves $(P, \mathcal{Q})$ must use one OR-gate for each element in each chain for a total of hw gates. Excluding the element x_0 which occurs in all sets in $\mathcal{Q}$, the top has size hw, and the largest sets in each chain have size h. Thus, at least $w - 1$ OR-gates are required to construct the top. In particular, an optimum OR-circuit that solves $(P, \mathcal{Q})$ has $hw + w - 1 = (h + 1)w - 1$ arithmetic gates.

Next consider an arbitrary SUM-circuit that solves $(P, \mathcal{Q})$. Observe that each chain requires h distinct SUM-gates, each of which has x_0 in its support. There are hw such SUM-gates in total, at most one of which may be shared in the subcircuit that computes the top. Such a shared SUM-gate has weight at most $h + 1$, whereas the top has weight $hw+1$. Thus the subcircuit that computes the top can share weight at most $h+1$ and must use non-shared SUM-gates to accumulate the remaining weight (if any), which requires $h(w - 1)$ gates. Thus, the SUM-circuit requires at least $hw + h(w - 1) = (2w - 1)h$ arithmetic gates. □

Remark 1. Traditional nonconstructive tools for deriving lower bounds to circuit size appear difficult to employ for this type of separation between two monotone circuit classes. Indeed, it is easy to show using standard counting arguments that most ensembles $(P, \mathcal{Q})$ with $|P| = |\mathcal{Q}| = r$ require $\Omega(r^2/\log r)$ gates for both OR- and SUM-circuits, but showing that there exist ensembles where the required SUM-circuit is significantly larger than a sufficient OR-circuit appears inaccessible to such tools.

2.3 Upper Bounds for Rewriting

Let us now proceed to study the algorithmic task of rewriting a given OR-circuit into a SUM-circuit. In particular, our interest is to quantify the number of extra gates required. We start with the observation that no extra gates are required if all gates in the given OR-circuit have weight at most 4.

Proposition 1. *Every OR-circuit with g gates of weight at most 4 can be rewritten into a SUM-circuit with g gates. Moreover, there is an algorithm with running time $O(g)$ that rewrites the circuit.*

Proof. Let C be an OR-circuit with g gates given as input. First, topologically sort the nodes of C in time $O(g)$. Then, compute the support of each gate by assigning unique singleton sets at the input gates and evaluating the gates in topological order. Finally, proceed in topological order and rewrite the gates of the circuit using the following rules. Input gates do not require rewriting. Furthermore, every OR-gate of weight 2 can be trivially replaced with a SUM-gate. Each OR-gate z with weight 3 either has the property that the in-neighbours z_1, z_2 of z have disjoint supports (in which case we may trivially replace z with a SUM-gate) or z_1, z_2 have weight at least 2. In the latter case, if at least one of z_1, z_2 has weight 3 (say, z_1), we may delete z and replace it with z_1; otherwise rewrite z so that one of its in-neighbours is z_1 and the other in-neighbour is the appropriate input gate. Each OR-gate z with weight 4 either has in-neighbours z_1, z_2 with disjoint supports or z_1, z_2 have weight at least 3 and at least 2, respectively. Again we may either delete z or rewrite z so that one of its in-neighbours is z_1 and the other in-neighbour is the appropriate input gate. It is immediate that this rewriting can be carried out in time $O(g)$. □

Next we observe that an OR-circuit can always be rewritten into a SUM-circuit with at most g times the number of gates in the OR-circuit.

Proposition 2. *There exists an algorithm that in time $O(g^2)$ rewrites a given OR-circuit with g gates into a SUM-circuit.*

Proof. The algorithm operates as follows. Let C be an OR-circuit with g gates and p input gates given as input. Topologically sort the nodes of C in time $O(g)$. Suppose the input gates of C are $x_1, x_2, \ldots, x_p$. Associate with each of the g gates an array of p bits. Then, iterate through the gates of C in topological order. For each input gate x_j, initialise the bit array associated with x_j so that the jth bit is set to 1 and the other bits are set to 0. For each OR-gate z with in-neighbours z_1, z_2, assign the bit array associated with z to be the union of the bit arrays associated with z_1 and z_2. This step takes time $O(gp)$. Finally, iterate through the gates of C. For each arithmetic gate z, output a SUM-circuit that computes the sum of the at most p inputs specified by the bit array associated with z. This requires at most $p - 1$ SUM-gates for each z. The algorithm takes $O(gp)$ time and outputs a circuit with $O(gp)$ gates. The claim follows because $p \leq g$. □

3 Subquadratic Rewriting Implies Faster CNF-SAT

Complementing the quadratic-time algorithm in Proposition 2, this section studies the possibility of developing fast (subquadratic-time) algorithms for rewriting OR-circuits as SUM-circuits. In particular, we show that the existence of such a subquadratic-time rewriting algorithm would, surprisingly, yield a non-trivial algorithm for general CNF-SAT (cf. Refs. [16,21] and [20, Theorem 5]).

Theorem 2. *Let $0 < \epsilon \leq 1$. If there is an algorithm that in time $O(g^{2-\epsilon})$ rewrites a given OR-circuit with g gates into a SUM-circuit, then there is an algorithm that solves CNF-SAT in time $O\big(2^{(1-\epsilon/2)n} m^{2-\epsilon} n\big)$, where n is the number of variables and m is the number of clauses.*

Proof. Let $0 < \epsilon \leq 1$ be fixed and let A be a circuit rewriting algorithm with the stated properties. We present an algorithm for CNF-SAT. Let an instance of CNF-SAT given as input consist of the variables $x_1, x_2, \ldots, x_n$ and the clauses $C_1, C_2, \ldots, C_m$. Without loss of generality (by inserting one variable as necessary), we may assume that n is even. Call the variables $x_1, x_2, \ldots, x_{n/2}$ *low* variables and the variables $x_{n/2+1}, x_{n/2+2}, \ldots, x_n$ *high* variables. The algorithm operates in three steps.

In the first step, the algorithm constructs the following OR-circuit. First let us observe that there are $2^{n/2}$ distinct ways to assign truth values (0 or 1) to the low variables. Each of these assignments indexes an input gate to the circuit. Next, for each clause C_i, we construct a subcircuit that takes the OR of all input gates that *do not satisfy* the clause C_i, that is, the input gate indexed by an assignment a to the low variables is in the OR if and only if no literal in C_i is satisfied by a. For each C_i, this subcircuit requires at most $2^{n/2} - 1$ OR-gates. Let us refer to the output gate of this subcircuit as gate C_i. Finally, for each assignment b to the high variables, construct a subcircuit that takes the OR of all gates C_i such that the clause C_i is *not satisfied* by b. Let us refer to the output gate of this subcircuit as gate b. The constructed circuit has $p = 2^{n/2}$ inputs and $g \leq m(2^{n/2} - 1) + 2^{n/2}(m - 1) = O(2^{n/2} m)$ gates. The construction time for the circuit is $O(2^{n/2} mn)$.

In the second step, the algorithm rewrites the constructed OR-circuit using algorithm A as a subroutine into a SUM-circuit in time $O(g^{2-\epsilon})$, that is, in time $O(2^{(1-\epsilon/2)n} m^{2-\epsilon})$. In particular, the number of gates in the SUM-circuit is $G = O(2^{(1-\epsilon/2)n} m^{2-\epsilon})$. For a gate z in the OR-circuit, let us write z' for the corresponding gate in the SUM-circuit.

In the third step, the algorithm assigns the value 1 to each input a' in the SUM-circuit (any other inputs are assigned to 0), and evaluates the SUM-circuit over the integers using $O(2^{(1-\epsilon/2)n} m^{2-\epsilon})$ additions of $O(n)$-bit integers. If there exists a gate b' that evaluates to a value less than $2^{n/2}$, the algorithm outputs "satisfiable"; otherwise the algorithm outputs "unsatisfiable". The running time of the algorithm is $O(2^{(1-\epsilon/2)n} m^{2-\epsilon} n)$.

To see that the algorithm is correct, observe that in the OR-circuit, the input a occurs in the support of b if and only if there is a clause C_i such that neither a nor b satisfies C_i. Equivalently, the assignment (a, b) into the n variables is not satisfying (because it does not satisfy the clause C_i). The rewrite into a SUM-circuit enables us to infer the presence of an a' that does not occur in the support of b' by counting the number of a' that do occur in the support of b'. SUM-gates ensure that each input in the support of b' is counted exactly once. □

Theorem 2 thus demonstrates that unless the strong exponential time hypothesis [11] fails, there is no subquadratic-time algorithm for rewriting arbitrary OR-circuits into SUM-circuits.

4 Finding Small Circuits Using SAT Solvers

We next develop a SAT encoding for deciding whether a given ensemble has a circuit of a given size.

4.1 SAT Encoding

We start by giving a representation of an OR- or SUM-circuit as a binary matrix. This representation then gives us a straightforward way to encode the circuit existence problem as a propositional formula.

Let $(P, \mathcal{Q})$ be an OR- or SUM-ensemble and let C be a circuit of size g that solves $(P, \mathcal{Q})$. For convenience, let us assume that $|P| = p$, $|\mathcal{Q}| = q$ and $P = \{1, 2, \ldots, p\}$. Furthermore, we note that outputs corresponding to sets of size 1 are directly provided by the input gates, and we may thus assume that $\mathcal{Q}$ does not contain sets of size 1. The circuit C can be represented as a $g \times p$ binary matrix M as follows. Fix a topological ordering $z_1, z_2, \ldots, z_g$ of the gates of C such that $z_i = i$ for all i with $1 \leq i \leq p$ (recall that we identify the input gates with elements of P). Each row i of the matrix M now corresponds to the support of the gate z_i so that for all $1 \leq j \leq p$ we have $M_{i,j} = 1$ if j is in the support of z_i and $M_{i,j} = 0$ otherwise. In particular, for all $1 \leq i \leq p$ we have $M_{i,i} = 1$ and $M_{i,j} = 0$ for all $j \neq i$. Figure 3 displays an example.

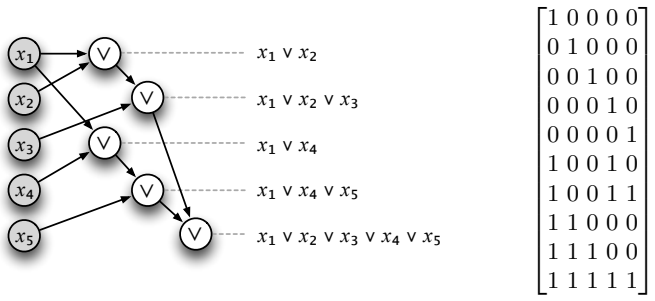

Fig. 3. An OR-circuit (left) and a matrix describing the circuit (right).

Now, C (viewed as an OR-circuit) solves $(P, \mathcal{Q})$ if and only if the matrix M satisfies

(a) for all i with $1 \leq i \leq p$ it holds that $M_{i,i} = 1$ and $M_{i,j} = 0$ for all $j \neq i$,
(b) for all i with $p + 1 \leq i \leq g$ there exist k and ℓ such that $1 \leq k < \ell < i$ and for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if and only if $M_{k,j} = 1$ or $M_{\ell,j} = 1$, and

(c) for every set A in $\mathcal{Q}$ there exists an i with $1 \leq i \leq g$ such that for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if $j \in A$ and $M_{i,j} = 0$ otherwise.

Similarly, C (viewed as a SUM-circuit) solves $(P, \mathcal{Q})$ if and only if the matrix M satisfies conditions (a), (c), and

(b') for all i with $p + 1 \leq i \leq g$ there exist k and ℓ such that $1 \leq k < \ell < i$ and for all j with $1 \leq j \leq p$ it holds that $M_{i,j} = 1$ if and only if $M_{k,j} = 1$ or $M_{\ell,j} = 1$ and that $M_{k,j} = 0$ or $M_{\ell,j} = 0$.

Based on the above observations, we encode an ensemble computation instance as SAT instance as follows. Given an OR-ensemble $(P, \mathcal{Q})$ and integer g as input, we construct a propositional logic formula φ over variables $M_{i,j}$, where $1 \leq i \leq g$ and $1 \leq j \leq p$, so that any assignment into variables $M_{i,j}$ satisfying φ gives us a matrix that satisfies conditions (a)–(c). We encode condition (a) as

$$\alpha = \bigwedge_{i=1}^{p} \Big(M_{i,i} \wedge \bigwedge_{j \neq i} \neg M_{i,j} \Big) .$$

Similarly, we encode the conditions (b) and (c), respectively, as

$$\beta = \bigwedge_{i=p+1}^{g} \bigvee_{k=1}^{i-2} \bigvee_{\ell=k+1}^{i-1} \bigwedge_{j=1}^{p} \Big((M_{k,j} \vee M_{\ell,j}) \leftrightarrow M_{i,j} \Big) , \quad \text{and}$$

$$\gamma = \bigwedge_{A \in \mathcal{Q}} \bigvee_{i=p+1}^{g} \Big[\Big(\bigwedge_{j \in A} M_{i,j} \Big) \wedge \Big(\bigwedge_{j \notin A} \neg M_{i,j} \Big) \Big] .$$

The desired formula φ is then $\varphi = \alpha \wedge \beta \wedge \gamma$. For a SUM-ensemble, we replace β with

$$\beta' = \bigwedge_{i=p+1}^{g} \bigvee_{k=1}^{i-2} \bigvee_{\ell=k+1}^{i-1} \bigwedge_{j=1}^{p} \Big(((M_{k,j} \vee M_{\ell,j}) \leftrightarrow M_{i,j}) \wedge (\neg M_{k,j} \vee \neg M_{\ell,j}) \Big) .$$

4.2 Practical Considerations

There are several optimisations that can be used to tune this encoding to speed up SAT solving. The resulting SAT instances have a high number of symmetries, as any circuit can be represented as a matrix using any topological ordering of the gates. This makes especially the unsatisfiable instances difficult to tackle with SAT solver. To alleviate this problem, we constrain the rows i for $p + 1 \leq i \leq g$ appear in lexicographic order, so that any circuit that solves $(P, \mathcal{Q})$ has a unique valid matrix representation. Indeed, we note that the lexicographic ordering of the gate supports (viewed as binary strings) is a topological ordering. We insert this constraint to the SAT encoding as the formula

$$\bigwedge_{i=p+2}^{g} \bigwedge_{k=p+1}^{i-1} \Big[(M_{i,1} \vee \neg M_{k,1}) \wedge \bigwedge_{j_1=2}^{p} \Big(\Big(\bigwedge_{j_2=1}^{j_1-1} (M_{i,j_2} \leftrightarrow M_{k,j_2}) \Big) \rightarrow (M_{i,j_1} \vee \neg M_{k,j_1}) \Big) \Big] .$$

We obtain further speedup by constraining the first t arithmetic gates to have small supports. Indeed, the ith arithmetic gate in any topological order has weight at most $i+1$. Thus, we fix $t = 6$ in the experiments and insert the formula

$$\bigwedge_{i=1}^{t} \bigwedge_{\substack{S \subseteq P \\ |S|=i+2}} \neg\Big(\bigwedge_{j \in S} M_{p+i,j}\Big) .$$

Further tuning is possible if $\mathcal{Q}$ is an *antichain*, that is, if there are no distinct $A, B \in \mathcal{Q}$ with $A \subseteq B$. In this case an optimal circuit C has the property that every gate whose support is in $\mathcal{Q}$ has out-degree 0. Thus, provided that we do not use the lexicographical ordering of gates as above, we may assume that the gates corresponding to sets in $\mathcal{Q}$ are the last gates in the circuit, and moreover, their respective order is any fixed order. Thus, if $\mathcal{Q} = \{A_1, A_2, \ldots, A_q\}$ is an antichain, we can replace γ with

$$\bigwedge_{i=1}^{q} \Big[\Big(\bigwedge_{j \in A_j} M_{g-q+i,j}\Big) \wedge \Big(\bigwedge_{j \notin A_j} \neg M_{g-q+i,j}\Big)\Big]$$

to obtain a smaller formula. Finally, we note that we can be combine this with the lexicographic ordering by requiring that only rows i for $p+1 \leq i \leq g-q$ are in lexicographic order.

5 Experiments

We report on two series of experiments with the developed encoding and state-of-the-art SAT solvers: (a) an exhaustive study of small ensembles aimed at understanding the separation between OR-circuits and SUM-circuits, and (b) a study of the scalability of our encoding by benchmarking different solvers on specific structured ensembles.

5.1 Instance Generation and Experimental Setup

For both series of experiments, the problem instances given to SAT solvers were generated by translating the encoding in Sect. 4 into CNF. We used the symmetry breaking constraints and antichain optimisations described in Sect. 4.2; without these, most instances could not solved by any of the solvers.

The formula encoding an input ensemble $(P, \mathcal{Q})$ and a target number of gates g was first translated into a Boolean circuit and then into CNF using the `bc2cnf` encoder (`http://users.ics.tkk.fi/tjunttil/circuits/`), which implements the standard Tseitin encoding [17]. The instance generator and a set of interesting handpicked CNF-level benchmark instances are available at

`http://cs.helsinki.fi/u/jazkorho/sat2012/`.

When working with an ensemble, the size of the optimal OR-circuit or optimal SUM-circuit is not generally known. Thus, we structured the experiments for a given ensemble $(P, \mathcal{Q})$ with $|P| = p$ and $|\mathcal{Q}| = q$ as a sequence of jobs that keeps the ensemble $(P, \mathcal{Q})$

fixed and varies the target number of gates g. We start from a value of g for which a circuit is known to exist ($p(1+q)$) and then decrease the value in steps of 1 until we hit an unsatisfiable instance at $g = u$; an optimal circuit then has $g = u + 1$ gates.

The experiments were run on Dell PowerEdge M610 blade servers with two quad-core 2.53-GHz Intel Xeon processors and 32 GB of memory. We report the user times recorded via `time` under Linux (kernel version 2.6.38). In the timed benchmarking runs we ran one simultaneous job on a single server, but in the explorative experiments we ran multiple jobs per server in parallel. SAT solvers used were Minisat 2.2.0 [5] and Lingeling 587f [1] (two CDCL solvers among the best for application instances), Clasp 2.0.4 [9] (CDCL solver, one of the best for crafted instances), and March_rw [10] (a DPLL-lookahead solver, one of the best for unsatisfiable random instances).

5.2 Optimal Circuits for All Small Ensembles

We say that two ensembles $(P, \mathcal{Q}_1)$ and $(P, \mathcal{Q}_2)$ are *isomorphic* if there is a permutation of P that takes $\mathcal{Q}_1$ to $\mathcal{Q}_2$. The optimal circuit size is clearly an isomorphism invariant of an ensemble, implying that in an exhaustive study it suffices to consider one ensemble from each isomorphism class.

We carried out an exhaustive study of all nonisomorphic ensembles $(P, \mathcal{Q})$ across the three parameter ranges (i) $p = 5$ and $2 \leq q \leq 7$, (ii) $p = 6$ and $2 \leq q \leq 7$, and (iii) $p = 7$ and $2 \leq q \leq 6$ subject to the following additional constraints: (a) every set in $\mathcal{Q}$ has size at least 2, (b) every set in $\mathcal{Q}$ contains at least two points in P that each occur in at least two sets in $\mathcal{Q}$, and (c) the ensemble is connected (when viewed as a hypergraph with vertex set P and edge set $\mathcal{Q}$). We generated the ensembles using the `genbg` tool that is part of the canonical labelling package `nauty` [15].

For all of the generated 1,434,897 nonisomorphic ensembles, we successfully determined the optimum OR-circuit size and the optimum SUM-circuit size in approximately 4 months of total CPU time using Minisat. Among the instances considered, we found no instance where the gap between the two optima is more than one gate. The smallest ensembles in terms of the parameters p and q where we observed a gap of one gate occurred for $p = 5$ and $q = 5$, for exactly 3 nonisomorphic ensembles; one of the ensembles with accompanying optimal circuits is displayed in Fig. 2. A further analysis of the results led to Theorem 1 and Proposition 1.

After this work the next open parameters for exhaustive study are $p = 7$ and $q = 7$ with 13,180,128 nonisomorphic ensembles.

In general, the large number of isomorphism classes for larger p and q makes an exhaustive search prohibitively time-consuming. A natural idea would be to randomly sample ensembles with given parameters to find an ensemble witnessing a large separation between optimal OR- and SUM-circuits. However, as highlighted in Remark 1, most ensembles require both a large OR-circuit and a large SUM-circuit, suggesting that random sampling would mostly give instances with small difference between optimal OR- and SUM-circuits. This intuition was experimentally supported as follows. We generated random ensembles $(P, \mathcal{Q})$ by setting $P = \{1, 2, \ldots, p\}$ and drawing uniformly at random a $\mathcal{Q}$ consisting of q subsets of P of size at least 2. We generated 1,000 instances for $p = q = 9$ and for $p = q = 10$. Among these instances, we found only one instance (with $p = q = 10$) where the gap between the optimal OR-circuit and and the optimal

SUM-circuit was 2, while we know that instances with larger separation do exist for these parameters. However, there were 49 instances with $p = q = 10$ where the optimal circuit sizes were not found within a 6-hour time limit.

5.3 Scaling on Structured Ensembles

To test the scalability of our encoding and to benchmark different solvers, we also studied two parameterised families of structured ensembles for varying family parameters and target number of gates g. The first family is illustrated by the Valiant's construction in Fig. 1 for $p = 8$. This family is parameterised by the number of inputs p, with $P = \{1, 2, \ldots, p\}$ and $\mathcal{Q} = \{P \setminus \{i\} \colon i \in P\}$. As benchmarks we generated CNF instances for $p = 8, 9, 10, 11$ and $g = 2p, 2p + 1, \ldots, 2p + 20$ using the SUM-encoding and the antichain optimisation. The second family is given in Theorem 1 and is parameterised by two parameters h and w. As benchmarks we generated CNF instances for $h = 3$ and $w = 5$ and $g = 32, 33, \ldots, 52$ using both the OR-encoding and the SUM-encoding.

The results for the two benchmark families are reported in Figs. 4 and 5. The solver March_rw was omitted from the second benchmark due to its poor performance on the first benchmark family. In an attempt to facilitate finding upper bounds for even larger instances, we also tested the local search solver SATTIME2011, which performed notably well on satisfiable crafted instances in the 2011 SAT Competition. However, in our experiments on instances from the satisfiable regime, SATTIME2011 was unable to find the solution within the 3600-second time limit already for the ensembles in Fig. 4 with $p = 8$ and $g = 26, 27, 28$.

6 Conclusions

We studied the relative power of OR-circuits and SUM-circuits for ensemble computation, and developed tight connections to Boolean satisfiability from both the theoretical and practical perspectives. As the main theoretical contribution, we showed that, while OR-circuits can be rewritten in quadratic-time into SUM-circuits, a subquadratic-time rewriting algorithm would imply that general CNF-SAT has non-trivial algorithms, which would contradict the strong exponential time hypothesis. From the practical perspective, we developed a SAT encoding for finding smallest SUM- and OR-circuits for a given ensemble. State-of-the-art SAT solvers proved to be a highly useful tool for studying the separation of these two circuit classes. Using the developed encoding, we were able to exhaustively establish the optimum OR-circuit and SUM-circuit sizes for all small instances, which contributed to our analytical understanding of the problem and led to the theoretical results presented in this paper. Our publicly available instance generator may also be of independent interest as a means of generating interesting benchmarks.

Larger, structured instances provide interesting challenges for current state-of-the-art SAT solver technology. Further developments either on the encoding or the solver level—including tuning SAT solvers especially for this problem—would allow for providing further understanding to the problem of separating different circuit classes.

Acknowledgment. We thank Teppo Niinimäki for insight concerning the construction in Theorem 1.

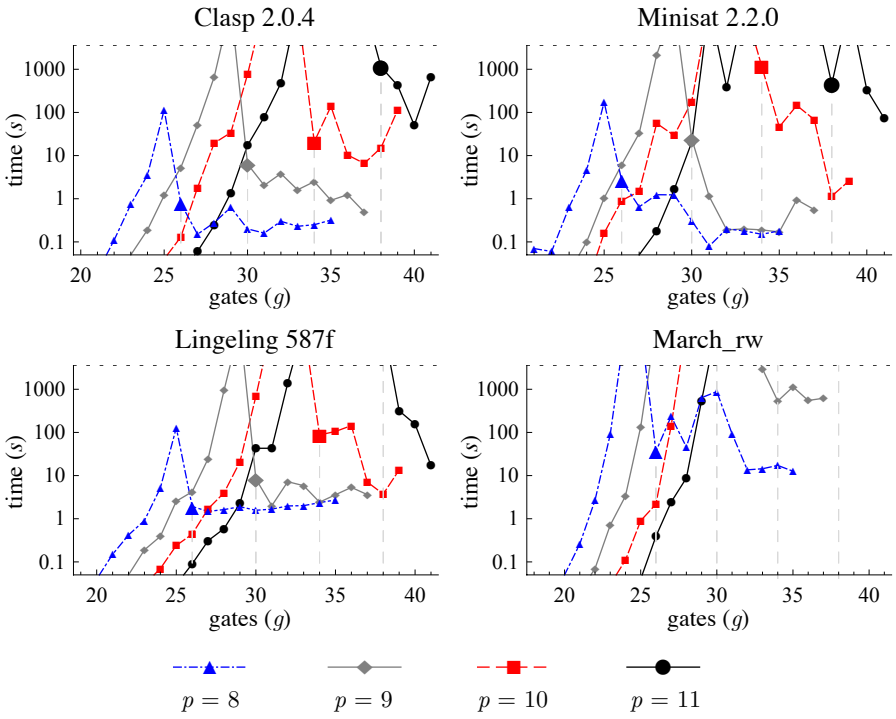

Fig. 4. Solution times for different SAT solvers as a function of the number of gates on SUM-ensembles corresponding to Valiant's construction (Fig. 1). The data points highlighted with larger markers and a vertical dashed line indicate the smallest circuits found. The horizontal dashed line at 3600 seconds is the timeout limit for each run. As the instance size p grows, the unsatisfiable instances with g just below the size of the optimal circuit rapidly become very difficult to solve.

References

1. Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Technical Report 10/1, Johannes Kepler University, Linz, Austria (2010)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M., Nederlof, J., Parviainen, P.: Fast zeta transforms for lattices with few irreducibles. In: Proc. SODA, pp. 1436–1444. SIAM (2012)
3. Dantsin, E., Wolpert, A.: On moderately exponential time for SAT. In: Proc. SAT, LNCS, vol. 6175, pp. 313–325. Springer (2010)
4. Demenkov, E., Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: New upper bounds on the Boolean circuit complexity of symmetric functions. Inf. Process. Lett. 110(7), 264–267 (2010)
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. SAT. LNCS, vol. 2919, pp. 502–518. Springer (2004)
6. Estrada, G.G.: A note on designing logical circuits using SAT. In: Proc. ICES. LNCS, vol. 2606, pp. 410–421. Springer (2003)
7. Fuhs, C., Schneider-Kamp, P.: Synthesizing shortest linear straight-line programs over GF(2) using SAT. In: Proc. SAT. LNCS, vol. 6175, pp. 71–84. Springer (2010)

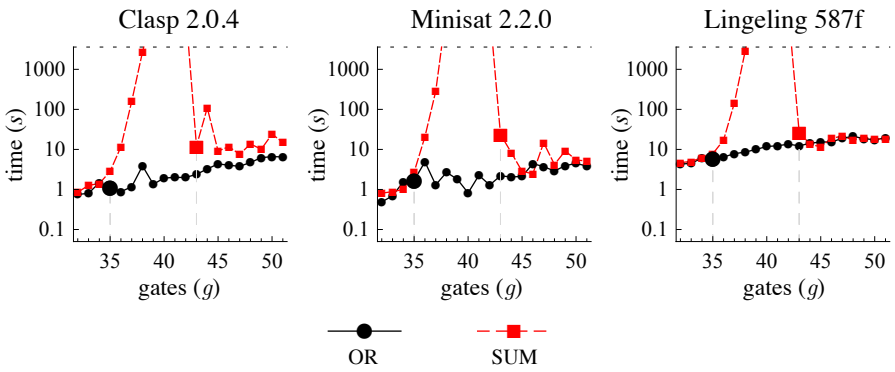

Fig. 5. Solution times for different SAT solvers as a function of the number of gates on OR- and SUM-ensembles from Theorem 1 with parameters $w = 5$ and $h = 3$. The data points highlighted with larger markers and a vertical dashed line indicate the smallest circuits found. The horizontal dashed line at 3600 seconds is the timeout limit for each run. The optimal OR circuit is small, and SAT solvers have no difficulty in finding it.

8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
9. Gebser, M., Kaufmann, B., Schaub, T.: The conflict-driven answer set solver clasp: Progress report. In: Proc. LPNMR. LNCS, vol. 5753, pp. 509–514. Springer (2009)
10. Heule, M., Dufour, M., van Zwieten, J., van Maaren, H.: March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In: SAT 2004 Selected Papers. LNCS, vol. 3542, pp. 345–359. Springer (2005)
11. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. System Sci. 62(2), 367–375 (2001)
12. Kamath, A., Karmarkar, N., Ramakrishnan, K., Resende, M.: An interior point approach to Boolean vector function synthesis. In: Proc. MWSCAS. pp. 185–189. IEEE (1993)
13. Kojevnikov, A., Kulikov, A.S., Yaroslavtsev, G.: Finding efficient circuits using SAT-solvers. In: Proc. SAT. LNCS, vol. 5584, pp. 32–44. Springer (2009)
14. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Proc. SODA, pp. 777–789. SIAM (2010)
15. McKay, B.: nauty user's guide. Tech. Rep. TR-CS-90-02, Australian National University, Department of Computer Science (1990)
16. Pătraşcu, M., Williams, R.: On the possibility of faster SAT algorithms. In: Proc. SODA. pp. 1065–1075. SIAM (2010)
17. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp. 466–483. Springer (1983)
18. Valiant, L.G.: Negation is powerless for Boolean slice functions. SIAM J. Comput. 15(2), 531–535 (1986)
19. West, D.B.: Introduction to graph theory. Prentice Hall Inc., Upper Saddle River, NJ (1996)
20. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoret. Comput. Sci. 348(2–3), 357–365 (2005)
21. Williams, R.: Improving exhaustive search implies superpolynomial lower bounds. In: Proc. STOC. pp. 231–240. ACM (2010)