Journal on Satisfiability, Boolean Modeling, and Computation 10 (2016) 11–22

Hard satisfiable 3-SAT instances via autocorrelation

Srinivasan Arunachalam[*] arunacha@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam
The Netherlands

Ilias Kotsireas[†] ikotsire@wlu.ca
Wilfrid Laurier University
Waterloo, Ontario
Canada

Abstract

We establish a reduction of a combinatorial problem defined via autocorrelation to an instance of Boolean satisfiability. As a consequence, we obtain a family of hard satisfiable 3-SAT instances. The combinatorial problem that we reduce is the D-optimal matrices problem. We generated a family of 3-SAT instances from the D-optimal matrices problem with the motivation to solve interesting cases using the power of SAT solvers. We give a detailed construction of the generated instances that were submitted to SAT competition 2014. Our reduction techniques is fairly straightforward and can be adapted to various other problems that are defined via autocorrelation.

Keywords: *SAT-solver, autocorrelation, D-optimal matrices*

Submitted January 2016; revised September 2016; published December 2016

1. Introduction

The Boolean satisfiability problem, commonly referred to as SAT, is a well-known NP-Complete problem and in fact the first problem that was proved to be NP-complete. It is defined as the problem of determining whether a Boolean formula has an assignment of its variables satisfying all its clauses. The Boolean formula is defined in terms of conjunctions of clauses, each clause individually being a disjunction of literals. This is the well-known conjunctive normal form (CNF).

Various combinatorial problems have been encoded as SAT problems in the past, see for instance Chapter 17 in [5]. In this paper we establish a reduction of a combinatorial problem defined via autocorrelation to an instance of 3-SAT for the first time. More specifically, we consider the problem of finding D-optimal matrices for odd n, namely $2n \times 2n$ square matrices with entries $\{-1, +1\}$, whose determinant is maximal among all other matrices with entries $\{-1, +1\}$ of the same order. A widely believed conjecture is that D-optimal matrices exist for all orders $2n$ that are not excluded by a constraint. See [9] and [6] for this

[*] Affiliated with Institute of Quantum Computing during most of this work and is grateful to Dr. Michele Mosca for support from NSERC.

[†] Research supported by an NSERC grant. I am grateful to Dr. Michele Mosca for introducing me to the research of the first Author.

and related problems. A combinatorial optimization formulation of the D-optimal matrices problem can be found in [12]. D-optimal matrices can be used to construct Hadamard matrices, which are important in Quantum Information and occur in many Engineering applications. We are interested in particular in a specific kind of D-optimal matrices, that arise from two n-bit sequences from $\{-1,+1\}^n$ whose corresponding autocorrelation coefficients sum to the constant 2. There are many open cases in the D-optimal matrices problem, which is one of the motivations behind formulating this problem in terms of Boolean Satisfiability.

Often, we come across the *dichotomy* that although Boolean Satisfiability is NP-complete, algorithmic progress and optimized implementations have resulted in SAT solvers which have been able to solve huge instances in *realistic time* contrary to scaling of algorithms. The annual SAT solving competitions [7] is a platform where people compete globally to solve benchmark hard problems using their latest state-of-the-art SAT solvers. We submitted some of the 3-SAT instances [2] of the D-optimal matrices problem that we generated to the SAT competition. The results were encouraging, especially given the time-limit constraint imposed by the organizers, and definitely demonstrated that our satisfiable 3-SAT instances pose a significant challenge to SAT solvers. We believe that with the vast study in Satisfiability solving algorithm, our 3-SAT reduction along with custom-tailored SAT solvers have the potential to solve very large instances of the D-optimal matrices problem and eventually be competitive enough with classical methods to solve open instances of the D-optimal problem.

Another independent interest in our 3-SAT formulation of the D-optimal matrices problem is in the light of the paper by Cook and Mitchell [8] in which they claim that finding hard satisfiable instances is a difficult problem. Experimental evidence employing various heuristics has shown that the difficulty of SAT instances depends intricately on the number of clauses and variables m, n. It is well known the hardest SAT instances occur near a critical phase transition region $\alpha_c = m/n \approx 4.25$, for which the given SAT instance is considered to be the toughest to solve. In this paper, the asymptotic clause/variable ratio for the generated 3-SAT instances for the D-optimal matrices problem is approximately 3.25, which intuitively implies that the generated instances should neither be too hard nor too easy for SAT solvers. The widely believed conjecture that D-optimal matrices exist for every odd n which is not excluded by a constraint, gives evidence for the existence of a satisfying assignment for the corresponding 3-SAT instance and hence gives a natural technique to generate hard satisfiable 3-SAT instances.

We briefly highlight some possible objections that one could pose while reading this paper. The reduction from the D-optimal matrices to Boolean satisfiability is fairly straightforward. Fair enough, we still find it interesting that small open case instances of the D-optimal matrices problem could not be solved by a naive reduction. In fact, the naivety in our reduction implies there is a lot of symmetry and structure in our SAT instances which could have been used by SAT solvers based on DPLL in the SAT competition 2014. However, that was not the case. The hardness of instances are solely based on the ratio between the clauses and variables, which is not a good indication when the construction is optimized. That is indeed quite true, but we believe from earlier non-optimized constructions that when the ratio (or the hardness in the SAT instances) will in fact increase from the proposed 3.25. We find it interesting already that such “relatively“ hard instances of satisfiability can be

obtained by such natural problems and our paper is the first to show the reduction from the D-optimal matrices problem to satisfiability. We would also like to highlight here that, definitely techniques such as Sequential Counter [13] and Totalizer [4] could be employed to make our construction more efficient. Note that we are also dealing with various pseudo-boolean constraints and there has been a lot of work ([1], [4], [11]) which could possibly employed to further improve our construction. We leave this as a direction for future research.
The rest of the paper is organized as follows. In Section 2, we provide the necessary background to define the D-optimal matrices problem in terms of the concept of autocorrelation. In Section 3, we provide all the tools that are required for reducing the D-optimal matrices problem to Boolean Satisfiability. In Section 4 we discuss in more detail the structure of the generated 3-SAT instances, namely their number of variables and their number of clauses. In [3], we consider all cases of the D-optimal matrices problem up until size $n \leq 100$ and construct the 3-SAT formulas for the corresponding instance. Finally in Section 5 we summarize our results and mention some future directions which would be interesting to pursue.

2. Background

In this section, we introduce the autocorrelation formulation for the *D-optimal matrices* problem, see [9] for more details.

Definition 1. Let n be odd and consider two n-bit sequences $A = [a_1, \ldots, a_n]$ and $B = [b_1, \ldots, b_n]$ both having elements from $\{-1, +1\}$. Let $\alpha = a_1 + a_2 + \ldots + a_n$, $\beta = b_1 + b_2 \ldots + b_n$ and let us assume (α, β) is a solution to the equation $x^2 + y^2 = 4n - 2$. A solution to the *D-optimal matrices* problem consists of an assignment of the $2n$ binary variables $a_1, \ldots, a_n, b_1, \ldots, b_n$, such that the following 2 linear and $\frac{n-1}{2}$ quadratic equations are satisfied:

$$\left\{ \begin{array}{ll} a_1 + \cdots + a_n = \alpha, & \\ b_1 + \cdots + b_n = \beta, & \\ \displaystyle\sum_{i=1}^{n} (a_i a_{i+j} + b_i b_{i+j}) = 2, & \qquad j = 1, \ldots, \dfrac{n-1}{2}, \end{array} \right\}. \tag{1}$$

where the index $i + j$ is taken modulo n, when it exceeds n. The quantity $\sum_{i=1}^{n} a_i a_{i+j}$ for $j = 1, \ldots, \frac{n-1}{2}$ is called the (periodic) autocorrelation of the sequence A, at lag j.

Example 2. Let $n = 9$ and consider $A = [1, 1, 1, 1, 1, 1, 1, -1, -1]$ and
$B = [1, 1, -1, 1, -1, 1, 1, 1, -1]$. Then we have $\alpha = 5$, $\beta = 3$ satisfying $\alpha^2 + \beta^2 = 4n - 2$ and $\sum_{i=1}^{n} (a_i a_{i+j} + b_i b_{i+j}) = 2$ for $j = 1, \ldots, 4$. Therefore these two sequences form a solution of the D-optimal matrices problem for $n = 9$. Note that any other cyclic permutation of A, B is a solution as well.

Example 3. Let $n = 11$ and note that the equation $\alpha^2 + \beta^2 = 4n - 2 = 42$ does not have any integer solutions. Therefore there are no solutions to the D-optimal matrices problem for $n = 11$.

It is immediate to see that the above formulation of the D-optimal matrices problem is a constraint satisfaction problem (CSP) with the following parameters:

- $2n$ Variables: $a_1, a_2, \ldots, a_n, b_1, \ldots, b_n$
- Domain of all variables: $\{-1, +1\}$
- Constraints: $(n-1)/2$ quadratic constraints and 2 linear constraints.

2.1 Transformation to $\{0, 1\}$ variables

In order to formulate the D-optimal matrices problem as an instance of Boolean satisfiability, we first apply a linear transformation to switch from $\{-1, +1\}$ variables to $\{0, 1\}$ variables. This transformation is simply given by:

$$A_i = \frac{a_i + 1}{2};\ B_i = \frac{b_i + 1}{2}, \qquad i = 1, \ldots, n.$$

Hence the linear and quadratic equations (1) of the D-optimal matrices problem become:

$$\left\{ \begin{array}{l} A_1 + \cdots + A_n = \dfrac{\alpha + n}{2}; \\ B_1 + \cdots + B_n = \dfrac{\beta + n}{2}; \\ \displaystyle\sum_{i=1}^{n} (A_i A_{i+j} + B_i B_{i+j}) = \frac{n + 1 + \alpha + \beta}{2}, \qquad j = 1, \ldots, \frac{n-1}{2}. \end{array} \right\}. \tag{2}$$

Once we have solved equations (2) in the $\{0, 1\}$ variables, we can use the inverse of the above linear transformation to obtain the solution of the original equations (1) in the $\{-1, +1\}$ variables.

3. Tools for transformation

The essential tool that we employ for converting the CSP in the previous section to an instance of SAT is the Tseitin transformation. Tseitin transformation [14] is a powerful technique to convert *any* Boolean gate to a satisfiability instance. The idea is to introduce a new variable for every formula, where the new variable evaluates to 1, if the formula is satisfied. Any Boolean formula can be converted to a satisfiability clause with only a linear increase in size of the formula. The following table shows the logical operators and their respective SAT formulas.

Table 1: SAT formulation for every binary gate.

Boolean Gate	*Formula*	*SAT formulation*
AND gate	$x_o = x_1 . x_2$	$(\overline{x_1} \vee \overline{x_2} \vee x_o) \wedge (x_1 \vee \overline{x_o}) \wedge (x_2 \vee \overline{x_o})$
NAND gate	$x_o = \overline{x_1 . x_2}$	$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_o}) \wedge (x_1 \vee x_o) \wedge (x_2 \vee x_o)$
OR gate	$x_o = x_1 + x_2$	$(x_1 \vee x_2 \vee \overline{x_o}) \wedge (\overline{x_1} \vee x_o) \wedge (\overline{x_2} \vee x_o)$
NOR gate	$x_o = \overline{x_1 + x_2}$	$(x_1 \vee x_2 \vee x_o) \wedge (\overline{x_1} \vee \overline{x_o}) \wedge (\overline{x_2} \vee \overline{x_o})$
NOT gate	$x_o = \overline{x_1}$	$(\overline{x_1} \vee \overline{x_o}) \wedge (x_1 \vee x_o)$
XOR gate	$x_o = x_1 \oplus x_2$	$(\overline{x_1} \vee \overline{x_2} \vee \overline{x_o}) \wedge (x_1 \vee x_2 \vee \overline{x_o}) \wedge (x_1 \vee \overline{x_2} \vee x_o) \wedge (\overline{x_1} \vee x_2 \vee x_o)$

In this section, we explain and construct circuits for the comparator, half-adder, full-adder, which will be required to transform the CSP in Section 2 to a SAT instance. On the way we shall also compute the number of clauses and variables that will be introduced by employing each gate.

1. **Comparator**: The equality between two numbers A and B with binary representation $A_1A_1A_2 \dots A_N$ and $B_1B_2B_3 \dots B_N$ respectively, can be performed by element wise comparison of all the N bits. The following circuit is used to perform the operation $x_i = A_i.B_i + \overline{A_i}.\overline{B_i}$. Note that $x_i = 1$ if and only if $A_i = B_i$ and otherwise $x_i = 0$.

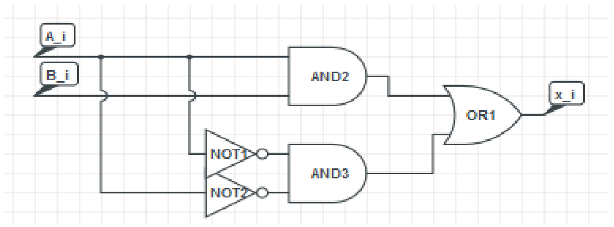


Figure 1: Comparator circuit.

Given the outputs $x_1, x_2, \dots, x_N$, the equality of the numbers A, B can be performed by simply verifying if $x_1 \wedge x_2 \wedge \dots \wedge x_N = 1$. If this clause is satisfied, it implies $A = B$ else $A \neq B$. Satisfiability representation of the above circuit can be done using the Tseitin transformation as follows:

$$\begin{aligned}(\overline{A_i} \vee \overline{B_i} \vee x_b^1) \wedge (A_i \vee \overline{x_b^1}) \wedge (B_i \vee \overline{x_b^1}) &= 1;\\ (\overline{A_i} \vee \overline{x_b^2}) \wedge (A_i \vee x_b^2) &= 1;\\ (\overline{B_i} \vee \overline{x_b^3}) \wedge (B_i \vee x_b^3) &= 1;\\ (\overline{x_b^2} \vee \overline{x_b^3} \vee x_b^5) \wedge (x_b^2 \vee \overline{x_b^5}) \wedge (x_b^3 \vee \overline{x_b^5}) &= 1;\\ (x_b^1 \vee x_b^5 \vee \overline{x_i}) \wedge (\overline{x_b^1} \vee x_i) \wedge (\overline{x_b^5} \vee x_i) &= 1,\end{aligned} \tag{3}$$

where the variables $x_b^1, x_b^2, x_b^3, x_b^4, x_b^5$ are buffer variables required for the computation of the output bit x_i. Totally there are 5 clauses to be satisfied for the equality of a single bit with 7 variables, and hence equality of strings A and B would overall require $5N + 1$ constraints with $7N$ variables.

2. **Addition:** A half-adder and full-adder are often employed to compute the sum of two binary numbers. The full-adder involves the addition of 2 bits along with a carry-in bit, which is absent for the half-adder. In order to compute the sum $A_{12} + A_{13} + \dots + A_{n1} + B_{12} + B_{13} + \dots + B_{n1}$ in Equation (1) we can compute it by pairwise addition of the terms through a binary tree of depth $\log_2(2N)$. The half-adder to compute the sum of bits A_i, B_i can be described in the following figure,

   The Tseitin transformation of the circuit to a SAT instance can be written as follows

$$\begin{aligned}(\overline{A_i} \vee \overline{B_i} \vee \overline{x_s}) \wedge (A_i \vee B_i \vee \overline{x_s}) \wedge (A_i \vee \overline{B_i} \vee x_s) \wedge (\overline{A_i} \vee B_i \vee x_s) &= 1;\\ (\overline{A_i} \vee \overline{B_i} \vee x_c) \wedge (A_i \vee \overline{x_c}) \wedge (B_i \vee \overline{x_c}) &= 1.\end{aligned} \tag{4}$$

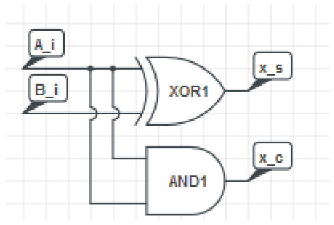


Figure 2: Half-adder circuit to compute sum of two bits A_i, B_i.

Similarly the full-adder circuit to compute the sum of bits A_i, B_i with carry-in bit C_i can be described as follows, The Tseitin transformation of the full-adder circuit to a

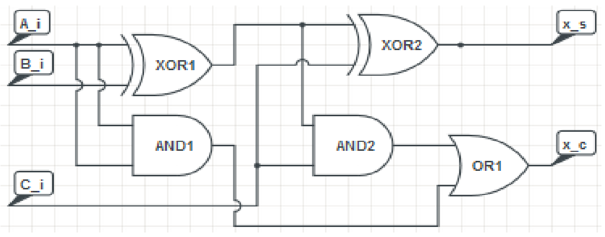


Figure 3: Full-adder circuit to compute sum of two bits A_i, B_i with a carry-in bit C_i.

SAT instance can be written as follows

$$\begin{aligned}(\overline{A_i} \vee \overline{B_i} \vee \overline{x_b^1}) \wedge (A_i \vee B_i \vee \overline{x_b^1}) \wedge (A_i \vee \overline{B_i} \vee x_b^1) \wedge (\overline{A_i} \vee B_i \vee x_b^1) = 1;\\ (\overline{A_i} \vee \overline{B_i} \vee x_b^2) \wedge (A_i \vee \overline{x_b^2}) \wedge (B_i \vee \overline{x_b^2}) = 1;\\ (\overline{C_i} \vee \overline{x_b^1} \vee \overline{x_s}) \wedge (C_i \vee x_b^1 \vee \overline{x_s}) \wedge (C_i \vee \overline{x_b^1} \vee x_s) \wedge (\overline{C_i} \vee x_b^1 \vee x_s) = 1;\\ (\overline{x_b^1} \vee \overline{C_i} \vee x_b^3) \wedge (x_b^1 \vee \overline{x_b^3}) \wedge (C_i \vee \overline{x_b^3}) = 1;\\ (x_b^3 \vee x_b^2 \vee \overline{x_c}) \wedge (\overline{x_b^3} \vee x_c) \wedge (\overline{x_b^2} \vee x_c) = 1.\end{aligned} \tag{5}$$

Employing the tools of the half-adder and full-adder, we can compute the sum of two numbers A, B with binary representation $A_1A_2 \dots A_N$ and $B_1B_2 \dots B_N$ respectively. The following circuit shows one such example containing one half-adder and $N-1$ full-adders

The binary representation of $A+B$ can be read out as $Out{=}C_NS_NS_{N-1}\dots S_1$. With circuit constructions of the comparator, half-adder, full-adder we are ready to describe the reduction from the CSP in Section 2.1 to Boolean satisfiability. In the next section we explain the reduction to satisfiability and calculate the total number of clauses and variables required for one such addition circuit.

4. Reduction to Boolean satisfiability

In this section we are finally ready to discuss the reduction of the CSP in Section 2 to Boolean satisfiability. In the process, we employ some combinatorial computations to calculate the

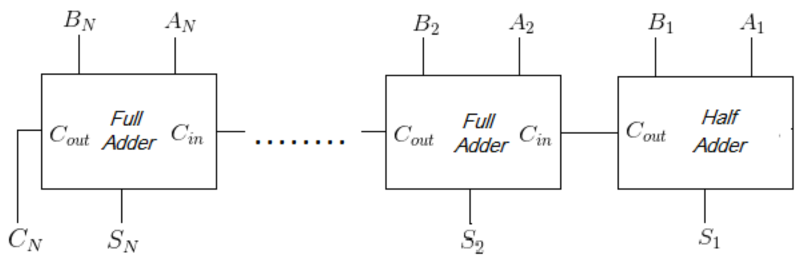


Figure 4: Adder circuit to compute sum of two N digit numbers.

number of clauses and variables in the final SAT instance. This computation is helpful to get an idea of the hardness of the instances generated through the Tseitin transformation.

Firstly, by symmetry it can be noted that instead of analyzing all $(n-1)/2$ periodic auto correlation in Equations (1), it suffices to analyze just once of them and generalize the results to all of them. Secondly, the arguments for the $(n-1)/2$ quadratic equations can be applied to the 2 linear equations as well. Hence we restrict our attention to analyzing one of the quadratic equation. The basic idea is to perform the summation in the shape of a binary tree where the nodes represent gates to perform pairwise addition. The binary binary tree takes inputs a sequence of $2N$ inputs and outputs the pairwise sum of these terms. In order to obtain the exact number of instances of variables we break down the reduction into three steps.[1.]

1. Given the sequence $(A_1, A_2, \ldots, A_N, B_1, B_2, \ldots, B_N)$, we first need to compute the auto correlation terms (or pairwise products) i.e. A_iA_j which requires an AND gate. Since there are totally $2N$ such terms to be accounted for, we require $2N$ AND gates. Without loss of generality, we denote the layer containing the $2N$ AND gates as the 0^{th} level of the binary tree.

2. Given the auto correlation terms A_iA_j, B_iB_j we are required to compute the sum of these $2N$ terms. The most efficient technique to compute the sum of these $2N$ terms is through a binary tree of depth $\lfloor \log_2(2N) \rfloor$. Naturally $2N$ need not be a whole power of 2, so we need to account for the extra terms. Before we continue to quantify the number of clauses and variables, we introduce some notation.

   - The binary tree consists of nodes which are effectively gates consisting of compositions of (Half adders denoted by $\mathcal{H}$ and Full-adders denoted as $\mathcal{F}$).
   - We say two gates interact, if the outputs of the individual gates are inputs to the gate in the subsequent level of the binary tree.

   It can easily be seen that U_j is the j^{th} bit in the binary representation of $2N$, in which case the total number of unaccounted nodes U_n is effectively the hamming weight of the binary representation of $2N$. There are many possibilities of how these unaccounted nodes can interact, but by some elementary calculations it can be seen that the optimal interaction (in terms of least number of introduced variables and clauses) occurs if

1. In Section 4.1 we give an example for $N = 5$

the unaccounted node in the k^{th} level interacts with the first unaccounted node at $(k+j)^{th}$ level and this process carries on till the $\lfloor \log_2(2N) \rfloor^{th}$ level. It remains to account for the unaccounted nodes.

Without loss of generality, let us assume there were m unaccounted nodes in the binary tree at levels $k_1, k_2, \ldots, k_m$. Note that the last node of the binary tree always corresponds to $m = \lfloor \log_2(2N) \rfloor$. The number of half-adders and full-adders required for the summation of any two nodes in the $x^{th}, (x+y)^{th}$ level is $\mathcal{H}^y \mathcal{F}^x \mathcal{H}$ (or totally $y+1$ $\mathcal{H}$ gates and x $\mathcal{F}$ gates). Hence the total number of adders required for the summation of the unaccounted nodes are

$$\sum_{i=1}^{m} \mathcal{H}^{k_{i+1}-k_i} \mathcal{F}^{k_i} \mathcal{H}. \tag{6}$$

3. The last step for the reduction is to ensure the left and right hand side (LHS/RHS) of the PAF Equations (1) are equal. We use the comparator as discussed in Section 3 to compare every bit. It should be noted that the overall number of comparators depends on the number of bits in the constant of the Equation (1). If the number of bits for the quantity on the LHS has more bits than the RHS, we could simply flip the extra bits and check if they are 1 (effectively checking if the extra bits are 0). This bit flip can simply be accomplished by the NOT gate.

Having analyzed the exact number of gates required to write down the PAF equations as a SAT formulation we are ready to to analyze the exact number of variables and clauses required for the SAT formulation. The following table summarises the number of clauses and variables from Section 3 required by the individual gates for the SAT formulation.

Table 2: Number of clauses and variables for binary gates.

Binary gate	*No. of clauses introduced*	*No. of variables introduces*
AND gate	3	1
Comparator	11	5
Full Adder	17	5
Half Adder	7	2
NOT gate	2	1

The total number of clauses and variables for computing the summation in the binary tree is

$$N_c = \sum_{j=1}^{\lfloor \log_2(2N) \rfloor} \left\lfloor \frac{2N}{2^j} \right\rfloor (17(j-1)+7); \qquad N_v = \sum_{j=1}^{\lfloor \log_2(2N) \rfloor} \left\lfloor \frac{2N}{2^j} \right\rfloor (5(j-1)+2). \tag{7}$$

Depending on the binary representation (for the unaccounted nodes) of $2N$ we need to compute the number of half-adders and full-adders required for the overall sum (which can be computed from Equation (6)). Correspondingly clauses and variables need to be added to the equation above. The number of comparators required for comparing both sides of Equation (1) is

Hence the total number of clauses and variables in the quadratic equations are $N_c + N_c^c + 6N$ ($6N$ extra clauses for the $2N$ AND gates) and $N_v + N_v^c + 2N$ ($2N$ terms for the initial $2N$ variables in the problem) respectively.

The final combinatorial calculation involves the linear equations that are required to be satisfied in Equation (1). Effectively we can repeat the calculations performed earlier, instead now we consider the sum of N terms instead of $2N$ terms. The overall number of clauses and variables for one of the two linear equations is

$$\begin{aligned}
N_c &= \sum_{j=1}^{\lfloor \log_2 N \rfloor} \left\lfloor \frac{N}{2^j} \right\rfloor (17(j-1)+7) + 11C^1_{count} + 2\lceil \log_2(N) \rceil + 1 - C^1_{count}, \\
N_v &= \sum_{j=1}^{\lfloor \log_2 N \rfloor} \left\lfloor \frac{N}{2^j} \right\rfloor (5(j-1)+2) + 5C^1_{count} + \lceil \log_2 N \rceil + 1 - C^1_{count}.
\end{aligned} \tag{8}$$

where $C^1_{count} = \left\lfloor \log_2 \frac{\alpha+N}{2} \right\rfloor + 1$. Correspondingly for the other equation, we change the variable α to β. As mentioned earlier, depending on the binary representation (for the unaccounted nodes) of N we need to add the clauses and variables for the half-adders and full-adders (which can be computed from Equation (6)). We have used these quantities to compute the entries for the table in Section [3].

4.1 Example

In this section, we give an example of the SAT formulation for the specific case $N = 5$ corresponding to parameters $\alpha = 3, \beta = 3$. We have employed tools employed in Section 3 and the reduction in the previous section to compute the number of clauses and variables for this case. The PAF equations to be satisfied for this case are

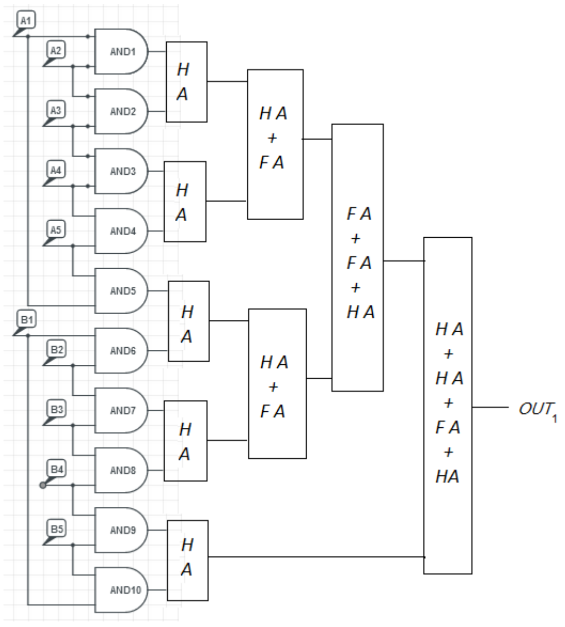
$$\begin{aligned}
&A_1A_2 + A_2A_3 + A_3A_4 + A_4A_5 + A_5A_1 + B_1B_2 + B_2B_3 + B_3B_4 + B_4B_5 + B_5B_1 = 2;\\
&A_1A_3 + A_2A_4 + A_3A_5 + A_4A_1 + A_5A_2 + B_1B_3 + B_2B_4 + B_3B_5 + B_4B_1 + B_5B_2 = 2;\\
&A_1 + A_2 + A_3 + A_4 + A_5 = 3;\\
&B_1 + B_2 + B_3 + B_4 + B_5 = 3.
\end{aligned}$$

It remains to be checked if the outputs of the quadratic and linear equations are equal to 2 and 3 respectively which is done through the following comparator circuit.

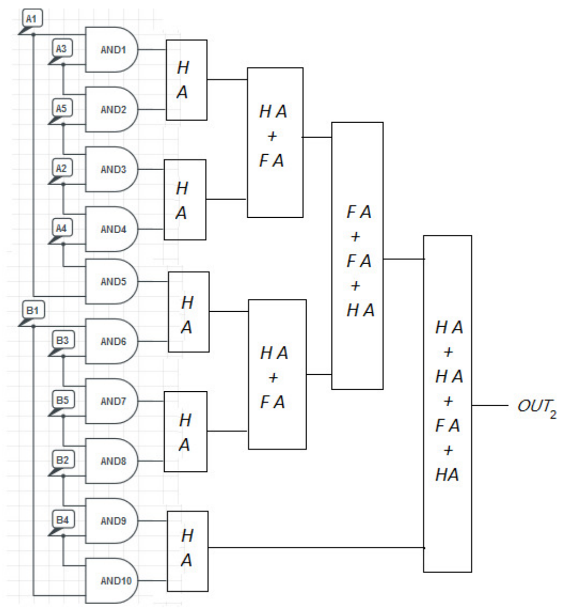
Hence the SAT formulation concludes by checking if $Out = 1$, which is the case only if the PAF equations are satisfied. A solution to the instance is $A = (1, 1, 1, 0, 0)$ and $B = (1, 1, 1, 0, 0)$ which translates to the solution of $a = (1, 1, 1, -1, -1)$ and $b = (1, 1, 1, -1, -1)$ to the D-Optimal matrices problem for $N = 5$.

5. Conclusion and future research

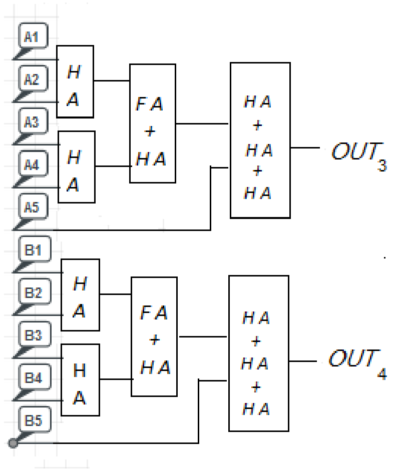
We have presented for the first time a technique for reducing a hard combinatorial problem defined via autocorrelation to an instance of Boolean Satisfiability. The reduction was



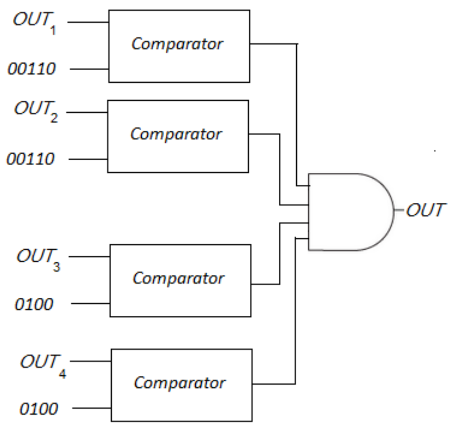
(a) Computing LHS of the quadratic PAF Equation (1) with periodicity 1.



(b) Computing LHS of the quadratic PAF Equation (1) with periodicity 2.



(a) Computing LHS of the linear PAF Equations (1).



(b) Validation of equality in the quadratic and linear Equations (1).

performed through a series of transformations, namely a linear change of variables followed by the Tseitin transformation on the underlying CSP. Our technique is applicable to many other combinatorial problems that can be defined via autocorrelation. The specific problem of D-optimal matrices we consider in this paper, provides a rich source of hard satisfiable 3-SAT instances, which can also be used as challenging benchmarks for SAT solvers. It would be interesting to see if further research in improving SAT solvers will help solve some of the open cases in the D-optimal matrices problem. Our construction helps in intuitively realizing that there is a lot of structure in the D-optimal matrices problem, which is not respected by Random Walk (or Walksat) based solvers. Consequently, they are bound to take longer (or fail in a constrained execution time scenario) on the generated SAT instances, however DPLL based solvers such as Minisat should technically perform much better on such instances. Additional research is needed, in order to gauge whether the SAT encoding should or should not include the vertical and horizontal combinatorial constraints of [12] and in general the compression constraints of [10].

Acknowledgments

We are grateful to the Fields Institute for hosting the "Quantum Optimization" workshop in October 2014. During the workshop we were able to discuss, clarify and finalize several key aspects of this paper. We are grateful to the anonymous referees for their thorough reading of our original submission, and their insightful comments and detailed suggestions, that led us to improve our work.

References

[1] Amir Aavani, David G. Mitchell, and Eugenia Ternovska. New encoding for translating pseudo-boolean constraints into SAT. In *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013, 11-12 July 2013, Leavenworth, Washington, USA.*, 2013.

[2] Srinivasan Arunachalam and Ilias Kotsireas, 2014. Satisfiability through auto correlation, in SAT competition.

[3] Srinivasan Arunachalam and Ilias Kotsireas, 2014. Satisfiability through autocorrelation. https://sites.google.com/site/autocorthroughsat/home.

[4] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, pages 108–122, 2003.

[5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[6] Richard P. Brent and Judy-anne H. Osborn. General lower bounds on maximal determinants of binary matrices. *Electr. J. Comb.*, **20**(2):P15, 2013.

[7] SAT Conference. Sat Competition. http://www.satcompetition.org/.

[8] Stephen A. Cook and David G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11-13, 1996*, pages 1–18, 1996. Edited by: Dingzhu Du, Jun Gu, Panos M. Pardalos.

[9] Dragomir Z. Djokovic and Ilias S. Kotsireas. New results on D-optimal matrices. *J. Combin. Des.*, **20**(6):278–289, 2012.

[10] Dragomir Z. Djokovic and Ilias S. Kotsireas. Compression of periodic complementary sequences and applications. *Des. Codes Cryptogr.*, **74**(2):365–377, 2015.

[11] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, **2**(1-4):1–26, 2006.

[12] Ilias S. Kotsireas and Panos M. Pardalos. D-optimal matrices via quadratic integer optimization. *J. Heuristics*, **19**(4):617–627, 2013.

[13] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, pages 827–831, 2005.

[14] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, pages 466–483. Springer, 1983.