Logic circuits from zero forcing

Daniel Burgarth*
Institute of Mathematics and Physics, Aberystwyth University, SY23 3BZ Aberystwyth, United Kingdom

Vittorio Giovannetti†
NEST, Scuola Normale Superiore and Istituto Nanoscienze-CNR, Piazza dei Cavalieri 7, I-56126 Pisa, Italy

Leslie Hogben‡
Department of Mathematics, Iowa State University, Ames, IA 50011, USA,
and American Institute of Mathematics, 360 Portage Ave, Palo Alto, CA 94306, USA

Simone Severini§
Department of Computer Science, and Department of Physics & Astronomy,
University College London, WC1E 6BT London, United Kingdom

Michael Young¶
Department of Mathematics, Iowa State University, Ames, IA 50011, USA

We design logical circuits based on the notion of zero forcing on graphs; each gate of the circuits is a gadget in which zero forcing is performed. We show that such circuits can evaluate every monotone Boolean function. By using two vertices to encode each logical bit, we obtain universal computation. We also highlight a phenomenon of "back forcing" as a property of each function. Such a phenomenon occurs in a circuit when the input of gates which have been already used at a given time step is further modified by a computation actually performed at a later stage. Finally, we show that zero forcing can be also used to implement reversible computation. The model introduced here provides a potentially new tool in the analysis of Boolean functions, with particular attention to monotonicity.

I. INTRODUCTION

We order the two elements of a set $\Sigma = \{0, 1\}$ such that $0 < 1$. This extends to a partial ordering on the set $\Sigma^n = \{0, 1\}^n$ by comparing words coordinate-wise. Let $x = x_1...x_n$ and $y = y_1...y_n$. Here, $x \succeq y$ means that $x_i \geq y_i$, for every $i = 1, ..., n$. A Boolean function $f : \Sigma^n \longrightarrow \Sigma$ is *monotone* when $f(x) \geq f(y)$ if $x \succeq y$, for every $x, y \in \Sigma^n$.

Monotone Boolean functions have an important role for proving lower bounds of circuit complexity (see, *e.g.*, Leeuwen [9], Chapter 14.4). Any function obtained by composition of monotone Boolean functions is itself monotone. Examples of monotone Boolean functions are the conjuction AND and the disjunction OR. Indeed, every monotone Boolean function can be realized by AND and OR operations (but without NOT). Boolean functions are important in applications, for example, in the implementation of a class of non-linear digital filters called stack filters [4]. Important methods for obtaining nontrivial bounds on specific monotone Boolean functions have been studied (see, *e.g.*, [2]).

---

*Electronic address: daniel@burgarth.de
†Electronic address: v.giovannetti@sns.it
‡Electronic address: lhogben@iastate.edu;hogben@aimath.org
§Electronic address: simoseve@gmail.com
¶Electronic address: myoung@iastate.edu

The concept of *zero forcing* on graphs is a recent idea that is part of a program studying minimum ranks of matrices with specific combinatorial constraints. Zero forcing has been also called graph infection and graph propagation [6, 10]. Notice that, in the context described here, the term "zero forcing" seems to be unfortunate, because we are forcing ones, not zeros. However, we keep the term given that this is now the most commonly used in the literature. In order to define zero forcing, we first need to define a *color-change rule*: if $G = (V, E)$ is a graph with each vertex colored either white or black, u is a black vertex of G, and exactly one neighbor v of u is white, then change the color of v to black. Given a coloring of G, the *final coloring* is the result of applying the color-change rule until no more changes are possible. A *zero forcing set* for G is a set $Z \subseteq V(G)$ such that if the elements of Z are initially colored black and the elements of $V(G) \backslash Z$ are colored white, the final coloring of G is all black.

Zero forcing is related to certain minimum rank/maximum nullity problems of matrices associated to graphs (see [3]) and to the controllability of quantum spin systems [5, 6]. Minimimizing the size of zero forcing sets is a difficult combinatorial optimization problem [1].

The remainder of this paper is organized as follows. In Section 2, we prove that zero forcing on graphs realizes all monotone Boolean functions, and highlight some simple related facts. The connection between zero forcing and circuits is obtained by associating a graph to each

logic gate. We will show that the functions AND and OR are indeed easily realized by two different gadgets with a few vertices. This is not the first work observing that monotone Boolean functions can be realized in a combinatorial setting. For example, Demaine *et al.* [7] have used the movements of a collections of simple interlocked polygons.

In Section 3, we describe the phenomenon of *back forcing* in the circuit. The phenomenon occurs when the color-change rule acts to modify the color of a vertex which has been already used during the computation. In some cases, back forcing implies that information about the output of a Boolean circuit can be read not just by looking at the color of a *target* vertex corresponding to the final output of the process, but at the color of the vertices in certain intermediate or initial gadgets. The idea opens a simple but intriguing scenario consisting of many parties that perform computation in a distributed way: each party holds a subset of the gates and it is able to read certain information about the input of other parties, since the color of its gates may have been modified by back forcing. Back forcing can be avoided by including some extra gadget acting as a filter.

In Section 4, we show that zero forcing implements reversible computation, if we augment the gadgets for AND and OR with extra vertices. The input of a reversible circuit can be read from the output. In natural way, zero forcing becomes *universal, i.e.,* it can realize any Boolean function, if we an ecoding. A *dual rail encoding, i.e.,* two vertices for each logical bit, is a method to construct the NOT gate and therefore to obtain universal computation. Conclusions are in Section 5.

II. MAIN RESULT

Our main result is easy to prove:

Theorem 1 *Zero forcing realizes all monotone Boolean functions.*

Proof. It is sufficient to show that zero forcing realizes the functions AND and OR.

Claim 1. The gate AND is realized by the gadget G_{AND} with vertices $\{1, 2, 3\}$ and edges $\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$, where 1 and 2 are the input vertices and 3 is the output vertex, containing the result and being able to propagate the color. All vertices are initially colored white. An illustration of the gadget G_{AND} is below:
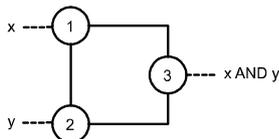
Proof of Claim 1. If no action is taken then the final coloring of the gadget is white. If we color vertex 1 black then the final coloring is all white but for vertex 1. The

same holds for vertex 2. However, if we color vertex 1 and vertex 2 black then the color-change rule implies that vertex 3 is black at step 2. In fact, $\{1, 2\}$ is a zero forcing set for G_{AND}.

Claim 2. The gate OR is realized by the gadget G_{OR} with vertices $\{1, 2, 3, 4\}$ and edges $\{\{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\}$, where 1 and 2 are the input vertices. The output vertex is vertex 4. Vertex 3 is initially colored black:
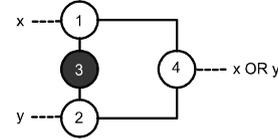

Proof of Claim 2. If no action is taken then the final coloring of the gadget is all white, but for vertex 3. If we color vertex 1 black then the color-change rule implies that vertex 4 is black at step 2. The same holds for vertex 2 and for vertex 1 and vertex 2 together. In fact, $\{1, 3\}, \{2, 3\}, \{1, 2, 3\}$ are zero forcing sets for G_{OR}, able to propagate the color for inducing the next step of the computation.

It is important to observe that zero forcing does not realize the function NOT, since when a vertex is colored black, it can not change color anymore. The consequence is that zero forcing does not realize universal computation (any Boolean function can be implemented using AND, OR and NOT gates) but monotone Boolean functions only. This concludes the proof. ∎

It may be worth observing the following points:

- Notice that extra vertices forming *delay lines* may be needed to assemble a circuit such that the output produced by zero forcing in parallel gates is syncronous. However, given our choice of gadgets, exactly 2 time steps are required for output of zero forcing in G_{AND} and G_{OR}. At time step 3 the color-change rule acts on the next gate in the circuit. There is then a convenient distinction between internal and external time: *internal time* refers to the zero forcing steps inside the gadgets/gates; *external time* refers to the time steps of the computation.

- The gadgets G_{AND} and G_{OR} have three and four vertices, respectively. By inspection on all possible combinations of white and black vertices for graphs with at most four vertices, we can observe that we have chosen the smallest possible gadgets, in terms of number of vertices and edges, realizing the two functions. One might think that the gate OR is realized also by the gadget with three vertices in the figure:
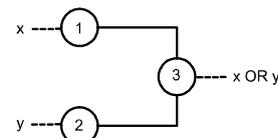
Although the gadget implements the OR correctly, it cannot be used as an initial or intermediate gate of a circuit, since in this gadget the color-change rule does not move fowards the output to the next gate, but it halts at vertex 3:
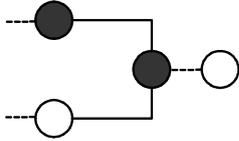
- Let us consider the gadget G_{OR}. If we color vertex 1 black then the color-change rule implies that vertex 4 is black at step 2. Suppose that vertex 2 is colored white at step 1. At step 2 the gate has computed the OR function in vertex 4 with input $\{0, 1\}$. At step 2 vertex 2 is also colored black under the action of the color-change rule, because this is the unique white neighbour of vertex 3. This is necessary in order for the computation to proceed using the output (black vertex 4). So, for all inputs with output 1, the vertices of G_{OR} are black after two steps of the internal time. Such behaviour is discussed in more detail in the next section.

- It is straightforward to realize the operation COPY:
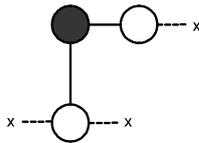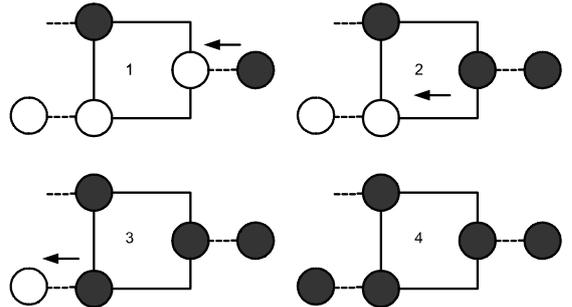
III.   BACK FORCING

If each Boolean variable in the input of a circuit is set to 1, then the vertices of the circuit that are initially colored black form a zero forcing set. However, this is not the only situation in which we have a zero forcing set. The next figure gives an example:
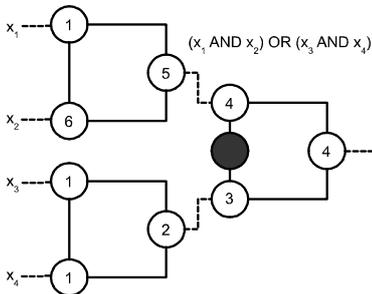
This is a circuit computing the Boolean function $(x_1$ AND $x_2)$ OR $(x_3$ AND $x_4)$. The number in the vertices of the figure specify the internal time step at which the vertex is black; the vertices labeled by 1 are initially colored black. The output of the circuit is 1 at step 4 and at step 6 of the internal time the vertices encoding the input of the function are all colored black. This can happen if and only if three of the input vertices are colored white at internal time 1.

The phenomenon will be called *back forcing*, because it is induced by the color-change rule acting backwards with respect to the direction from input to output in the whole circuit. The gadget G_{AND} exhibits back forcing conditionally on having input $\{0, 1\}$. The type of back forcing in G_{AND} can be called *transmittal back forcing*, because if something back forces its output black then the gate transmits the back force, *i.e.*, it modifies the color of the output vertex in a gate used previously. The figure clarifies the dynamics:

The gadget G_{OR} needs to force an input forward in order to color black one of the output vertices adjacent to its inputs and in another gate. In this sense, G_{OR} does not have transmittal back forcing. In other words, a gate at external time t, can not back force its color into G_{OR} at external time $t + 1$. In contrast, the circuit $(x_1$ AND $x_2)$ OR $(x_3$ AND $x_4)$ can initiate back forcing as described above (when it an intermediate element in the circuit).

We can also slow down back forcing, by including appropriate *delay lines* – for example, by adding extra vertices in each gadget or between them. Alternatively, we could consider delay lines directly embedded in the structure of the gadgets implementing the logical gates.

Also, back forcing can be avoided completely by including the gadget below. The gadget act as a *filter*. In some sense, the filter can be understood as an electronic diode allowing zero forcing only in one direction:
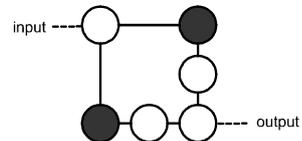
In relation to the circuit for the function $(x_1$ AND $x_2)$ OR $(x_3$ AND $x_4)$, it may be interesting to see that if there are two parties each one chosing the input of one of the two AND gates, and each one having access to only the corrisponding vertices, given the back forcing, the parties can then learn the output of the circuit by looking at the color of their vertices at the end of the computation, except when a party chooses $(0, 0)$ (*i.e.*, white, white).

IV. REVERSIBILITY AND UNIVERSALITY

A logical circuit is *reversible* if there is a one-to-one correspondence between input and output (see [11] for a review). A necessary and sufficient condition is that the input of each gate should be readable at the end of the gate evaluation together with the output. The importance of reversible computing is at least twofold: it overcomes the problem of energy dissipation in microprocessors, thus proposing an approach to scalability and optimization of resources; the idea of a thermodynamics of computation introduced with the study of reversibility was instrumental for the development of quantum computing. From the engineering perspective, reversible computing is a challenge, in particular because it is difficult to develop switching devices with adiabatic energy coefficients below those of transistors.

Despite the fact that the color-change rule induces a non-reversible process, we can use zero forcing to carry forward the input of each gate. Consider the gadgets in the figure:
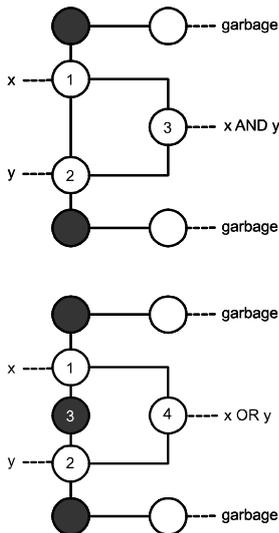

The gadgets realize the functions AND and OR. In a reversible gate, the outputs that are not used as primary outputs are called *garbages*. Each zero forcing AND and OR has two garbages. As a consequence, at the end of a computation there are $2n$ barbages, where n is the number of gates. The input of a circuit is in one-to-one correspondence with its output and the logical value of all garbages required during the computation.

By making use of two vertices to encode a *logical bit*, we can design the gate NOT, which permits universal computation together with the gate AND (by constructing the gate NAND). The encoding uses the set $\{0, 1\}$, but now we assign 0 and 1 to the following two gadgets, respectively:
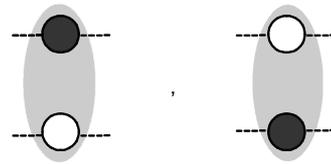
The vertices in these gadget can be seen as the *physical bits* of the econding. The gate NOT is realized by *swapping* the wires of the physical bits:
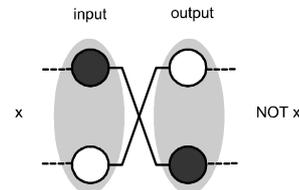
Next is a representation of the AND with the dual rail encoding:
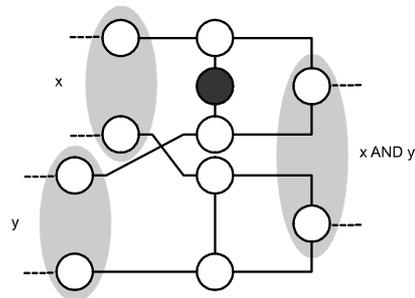
V. CONCLUSIONS

We have shown that all monotone Boolean functions can be realized by zero forcing in a graph constructed by *gluing* together the copies of two types of subgraphs/gadgets corresponding to the Boolean gates AND and OR. We have briefly discussed the minimality of such gadgets in terms of vertices and edges.

We have highlighted a back forcing action. Back forcing has an effect on the coloring of gates already used, as a function of what is happened in the "future", *i.e.*, at a later stage of the computation. Because of the relation between zero forcing and minimum ranks, the model described here is amenable to be studied with linear algebraic tools, potentially suggesting a novel direction in the analysis of monotone Boolean functions.

Finally, we have shown that computations obtained by zero forcing can also be implemented by carrying forward the input of each gate, and then without any information loss.

An open problem suggested by the paper is to understand the link between zero forcing and the dynamics at the basis of other unconventional models of computation, like, for example, the billiard ball computer – introduced as a model of reversible computing [8] –, models involving geometric objects, and dominos [7].

[1] A. Aazami, *Hardness results and approximation algorithms for some problems on graphs*, Ph.D. thesis, University of Waterloo, Dec. 2008.

[2] N. Alon, R. Boppana, The monotone circuit complexity of Boolean functions, *Combinatorica*, **7**, (1986), 1-22.

[3] American Institute of Mathematics (AIM) Minimum Rank-Special Graphs Work Group, AIM Minimum Rank – Special Graphs Work Group (F. Barioli, W. Barrett, S. Butler, S. M. Cioabă, D. Cvetković, S. M. Fallat, C. Godsil, W. Haemers, L. Hogben, R. Mikkelson, S. Narayan, O. Pryporova, I. Sciriha, W. So, D. Stevanović, H. van der Holst, K. Vander Meulen, A. Wangsness), Zero forcing sets and the minimum rank of graphs, *Linear Algebra and Its Application* **428** (2008), 1628-1648.

[4] J. Astola, D. Akopian, O. Vainio, and S. Agaian, New digit-serial implementation of stack filters, *Discrete Applied Mathematics*, **61**, pp. 181–197, 1997.

[5] D. Burgarth, S. Bose, C. Bruder, V. Giovannetti, Local controllability of quantum networks, *Phys. Rev. A* **79**, 060305(R) (2009).

[6] D. Burgarth and V. Giovannetti, Full control by locally induced relaxation, *Phys. Rev. Lett.* **99**, 100501 (2007).

[7] E. D. Demaine, M. L. Demaine, R. Uehara, Any monotone boolean function can be realized by interlocked polygons, *CCCG 2010*, 139-142.

[8] E. Fredkin and T. Toffoli, Conservative logic, *International Journal of Theoretical Physics*, vol. 21, no. 3, pp. 219–253, 1982.

[9] J. Leeuwen, *Handbook of Theoretical Computer Science*, Elsevier Science Publishers, 1990.

[10] S. Severini, Nondiscriminatory Propagation on Trees, *J. Phys. A: Math. Theor.* **41** 482002, 2008.

[11] P. Vitanyi, Time, Space, and Energy in Reversible Computing, *Proc. 2005 ACM International Conference on Computing Frontiers*, Ischia, Italy, 4-6 May 2005, 1st International Workshop on Reversible Computing.