A Lower Bound for Boolean Permanent in Bijective Boolean Circuits and its Consequences *

Rimli Sengupta and H. Venkateswaran
e-mail : {rimli,venkat}@cc.gatech.edu

GIT–CC–94-55

September, 1994

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280

Abstract

We identify a new restriction on Boolean circuits called bijectivity and prove that bijective Boolean circuits require exponential size to compute the Boolean permanent function. As consequences of this lower bound, we show exponential size lower bounds for: (a) computing the Boolean permanent using monotone multilinear circuits; (b) computing the 0-1 permanent function using monotone arithmetic circuits; and (c) computing the lexicographically first bipartite perfect matching function using circuits over $(min, concat)$. The lower bound arguments for the Boolean permanent function are adapted to prove an exponential lower bound for computing the Hamiltonian cycle function using bijective circuits. We identify a class of monotone functions such that if their counting version is $\sharp\mathcal{P}$-hard, then there are no polynomial size bijective circuits for such functions unless $\mathcal{PH}$ collapses.

*This work was supported by NSF grant CCR-9200878.

1 Introduction

We identify a new restriction called bijectivity on Boolean circuits and prove an exponential size lower bound for computing the Boolean permanent function in this model. As consequences of this lower bound, we show exponential size lower bounds for: (a) computing the Boolean permanent function using monotone multilinear circuits; (b) computing the 0-1 permanent function using monotone arithmetic circuits; and (c) computing the lexicographically first Boolean permanent function using circuits over $(min, concat)$.

Arithmetic circuits with $\{+, \times\}$ nodes have traditionally been defined as algebraic circuits over a field. Several lower bounds are known on the size and depth of algebraic circuits over positive reals that compute certain multilinear polynomials [6, 11, 16, 14]. Our interest is in *counting* arithmetic circuits: those that compute functions of the form $f : \{0,1\}^* \rightarrow \mathcal{N}$. One of the main differences between counting arithmetic circuits and algebraic circuits over the positive reals is that while computing a multilinear polynomial, the formal polynomial associated with the circuit need not be multilinear in the former case whereas in the latter case it must. This is because the inputs to a counting arithmetic circuit receive only 0-1 values and $0, 1$ are both idempotent with respect to the $\times$ operator.

The reason for our interest in counting arithmetic circuits is that there are characterizations of popular counting classes such as $\sharp\mathcal{P}$ and $\sharp\mathcal{LOGCFL}$ in terms of these circuits [19, 20]. Therefore, proving non-trivial lower bounds on the size of arithmetic circuits that compute natural functions have implications for separating counting classes. For example, if one could prove a super-polynomial size lower bound for computing the 0-1 permanent function using counting arithmetic circuits, one would separate $\sharp\mathcal{LOGCFL}$ from $\sharp\mathcal{P}$. Moreover, it would also have the more severe implication that the 0-1 permanent function is not in $\mathcal{FP}$. Thus, the above lower bound seems hard to prove. Instead, we consider the question of proving non-trivial size lower bounds for computing the Boolean permanent function using Boolean circuits that are restricted such that their arithmetization (that is, the arithmetic circuit obtained by replacing the $\vee$ nodes with $+$ and $\wedge$ nodes with $\times$) computes the 0-1 permanent function.

To this end, we identify a new restriction called bijectivity on Boolean circuits. Defining a monomial to be consistent if it does not have both the positive literal and its complement appearing in it, we say that a Boolean circuit is *bijective* if the number of consistent monomials in the formal polynomial associated with the circuit is the same as the number of prime implicants of the function it computes. We prove that bijective Boolean circuits require exponential size to compute the Boolean permanent function.

Computing the Boolean permanent of a matrix is equivalent to deciding whether the bipartite graph represented by the matrix has a perfect matching. This problem can be solved by polynomial size Boolean circuits since it is known to be in $\mathcal{P}$ [5]. But no non-trivial lower bounds are known for the size or depth of general Boolean circuits that compute this function. For monotone Boolean circuits computing this function, a super-polynomial size lower bound [8] and a linear depth lower bound [7] are known. For constant depth unbounded fan-in circuits an exponential size lower bound for this function follows since it is constant depth reducible to PARITY [3].

The bijectivity restriction is interesting for the following reasons:

(a) Bijective circuits can compute all Boolean functions. This is because the circuit based on the representation of a Boolean function f as a disjunction of its prime implicants, is bijective.

(b) There are natural circuits that are bijective, such as an $\mathcal{NC}^1$ circuit for PARITY.

(c) Arithmetization of bijective circuits for a class of monotone functions, called *suitable* functions, computes their counting version. Bipartite perfect matching is an example of a suitable function, whose counting version is the 0-1 permanent function.

(d) Bijective circuits for suitable functions can be efficiently made monotone. It is worth noting however that monotone circuits are not necessarily bijective. For example, the arithmetization of a monotone circuit for the Boolean permanent function does not necessarily compute the 0-1 permanent.

The proof of our lower bound is based on a combinatorial framework developed by Jerrum and Snir [6] to obtain size lower bounds for computing polynomials using circuits defined over semi-rings. In [6], Jerrum and Snir prove an exponential size lower bound for computing the permanent polynomial using algebraic circuits over positive reals. The combinatorial framework they develop uses restricted Boolean circuits that correspond to such algebraic circuits. We observe that these Boolean circuits are monotone, multilinear and bijective. Because of our interest in lower bounds for computing the 0-1 permanent function, we extend their framework for Boolean circuits that are monotone and bijective.

The following are some noteworthy features of the results in this paper:

- The lower bound for computing the Boolean permanent function using bijective circuits is obtained by a simple extension of the framework in [6]. As noted earlier, our lower bound is for circuits whose formal polynomials need not be multilinear. This is not the case for circuits considered in [6, 11]. As a consequence of our bound, we obtain an exponential size lower bound for computing the 0-1 permanent function using monotone arithmetic circuits.

- Razborov [8] showed a super-polynomial size lower bound for monotone circuits that compute the Boolean permanent function. While the question as to whether matching requires exponential size in the monotone Boolean circuit model remains open, using the lower bound on bijective circuits we show that monotone circuits require exponential size when further restricted with multilinearity or homogeneity.

- As another consequence of the bijectivity lower bound, we prove an exponential size lower bound for computing the lexicographically first Boolean permanent function using circuits over $(min, concat)$.

- While Boolean permanent is known to be in $\mathcal{P}$ and Hamiltonian cycle is $\mathcal{NP}$-complete, the counting versions of both functions are complete for $\sharp\mathcal{P}$. This suggests that both these functions ought to be equally hard in some sense. By showing that both of them require exponential size on bijective Boolean circuits, we exhibit one such setting and thereby take a step in the direction of understanding the phenomenon of easy decision problems having hard counting versions. In fact, we identify a class of monotone functions such that if their counting version is $\sharp\mathcal{P}$-hard, then there are no polynomial size bijective circuits for such functions unless $\mathcal{PH}$ collapses.

The rest of the paper is organized as follows. We begin by deriving a canonical form for the formal polynomial of a general Boolean circuit computing a monotone function (section 2.1). After motivating the bijectivity restriction in section 2.2 and 2.3, we show that bijective circuits for computing a class of monotone functions, that includes Boolean permanent and Hamiltonian cycle, must be monotone (section 3). In section 4, we adapt the Jerrum and Snir [6] lower bound framework to bijective circuits to obtain an exponential size lower bound for the Boolean permanent function. The three consequences of this lower bound are discussed in section 5. Section 6 concludes the paper.

2 Preliminaries

A Boolean circuit B_n is a rooted, directed, acyclic graph with interior nodes labeled from $\{\vee, \wedge\}$ and the leaf nodes labeled from $\{x_i, \bar{x}_i, 0, 1 | 1 \leq i \leq n\}$. B_n is *monotone* if the leaves are labeled only from $\{x_i, 0, 1 \mid 1 \leq i \leq n\}$. Without loss of generality, we will assume all $\wedge$-nodes have fanin 2. The *size* of a Boolean circuit is the number of non-leaf nodes in it, and its *depth* is the length of the longest path from any leaf to the root. Each B_n computes a unique Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$.

With every B_n, we can associate an algebraic expression $\mathcal{E}(B_n)$ defined inductively in a natural fashion.

Definition 2.1 If v is a leaf in B_n, let E_v be its label. If v is a $\vee$-node with children $v_1, v_2, \ldots, v_r$, let $E_v = \sum_{i=1}^{r} E_{v_i}$ and if v is a $\wedge$-node with children v_1, v_2, let $E_v = E_{v_1} \cdot E_{v_2}$. Then, $\mathcal{E}(B_n) = E_r$, where r is the root of B_n.

Definition 2.2 The *formal polynomial* $P(B_n)$ associated with a Boolean circuit B_n is obtained by only applying distributivity of $\wedge$ over $\vee$ in $\mathcal{E}(B_n)$.

Definition 2.3 A *parse-graph* G in B_n is defined inductively as follows: G includes the root of B_n; for any $\vee$-node v included in G, exactly one immediate predecessor of v in B_n is included as its only predecessor in G; and for any $\wedge$-node v included in G, all the immediate predecessors of v in B_n are included as its predecessors in G.

Parse-graphs are a generalization of the notion of parse-trees in [6]. Every node of a parse-graph G computes a formal multivariate monomial, defined in the obvious way. The monomial computed at the root of G is a monomial of $P(B_n)$. Thus, each monomial of $P(B_n)$ corresponds to a parse-graph in B_n.

Definition 2.4 A Boolean circuit B_n is said to be *multilinear* if $P(B_n)$ is multilinear.

Definition 2.5 Each element in the set $\{x_i \mid 1 \leq i \leq n\}$ is a *variable*. A *literal* is a variable x in positive form x or negative form $\bar{x}$. A *term* is a conjunction of literals, both positive and negative. Each term t can be expressed as $t_+.t_-$, where each literal in t_+ is positive and each literal in t_- is negative. For any term t, t_+ is the *positive term* of t and t_- is its *negative term*. $var(t_+)$ denotes the set of variables in t_+ and $var(t_-)$ is the analogous set for t_-.

Definition 2.6 A term t is said to be *consistent* if $var(t_+) \cap var(t_-) = \emptyset$. A parse-graph is *consistent* if the term it computes is consistent.

Definition 2.7 Each Boolean function f can be represented as a disjunction of its *prime implicants*, where each prime implicant is a term such that the set $\mathrm{PI}(f)$ of prime implicants satisfy the following properties: (i) for all $t \in \mathrm{PI}(f)$, t is consistent and each literal appears at most once in t; (ii) $\mathrm{PI}(f)$ contains all terms $t = t_+.t_-$, such that setting the variables in $var(t_+)$ to 1 and those in $var(t_-)$ to 0 causes f to evaluate to 1, regardless of the values of the other variables; (iii) there do not exist terms $t, t' \in \mathrm{PI}(f)$ such that both $var(t_+) \subseteq var(t'_+)$ and $var(t_-) \subseteq var(t'_-)$; (iv) when f is monotone, the terms in $\mathrm{PI}(f)$ do not contain negative literals.

Throughout this paper, we shall use the following functions:

Definition 2.8 The 0-1 permanent function $\mathrm{PERM} : \{0,1\}^{n^2} \to \mathcal{N}$ takes as input an $n \times n$ 0-1 matrix and outputs its permanent. The bipartite perfect matching function $\mathrm{BPM} : \{0,1\}^{n^2} \to \{0,1\}$, takes as input the standard $n \times n$ adjacency matrix representation of a bipartite graph G and outputs 1 if and only if G has a perfect matching. Note that BPM computes the Boolean permanent of the input matrix.

2.1 The Canonical Formal Polynomial

Given a general Boolean circuit B_n computing a monotone function f, we can establish some relationships between the consistent monomials of $P(B_n)$ and the terms of $\mathrm{PI}(f)$ leading to a canonical form for $P(B_n)$. The proofs of the following lemmas are based on the idea that $P(B_n)$ and f must agree on every input assignment, since B_n computes f.

Lemma 2.1 For each consistent monomial ρ of $P(B_n)$, there exists a term $t \in \mathrm{PI}(f)$ such that $var(t) \subseteq var(\rho_+)$.

Proof: Suppose there is a consistent monomial ρ for which this claim is not true. On the input assignment that sets the variables in $var(\rho_+)$ to 1 and all the rest to 0, B_n evaluates to 1 but f is 0, leading to a contradiction. $\Box$

In the other direction, we have the following lemma.

Lemma 2.2 For all terms $t \in \mathrm{PI}(f)$, there exists a consistent monomial ρ of $P(B_n)$, such that $var(t) = var(\rho_+)$.

Proof: Let t be any prime implicant of f. Consider the input that assigns 1 to the variables in t and 0 to all the rest. On this input, f evaluates to 1. Since B_n computes f, $P(B_n)$ must have a monomial ρ such that $var(\rho_+) \subseteq var(t)$, because otherwise B_n would evaluate to 0 on this input. Now, there cannot be any other $t' \in \mathrm{PI}(f)$ such that $var(t') \subseteq var(\rho_+)$, for otherwise $var(t') \subseteq var(t)$ which is impossible by fact 2.1 since $t, t' \in \mathrm{PI}(f)$. It follows from lemma 2.1 that $var(\rho_+) = var(t)$. $\Box$

Since each parse-graph in B_n computes a monomial in $P(B_n)$, by the above lemmas there is a term in $\mathrm{PI}(f)$ associated with each consistent parse-graph of B_n. By ordering the terms of $\mathrm{PI}(f)$, we associate a unique prime implicant with each consistent parse-graph of B_n. This allows us to partition the set of consistent parse-graphs of B_n into *parse-classes*, $PC_1, \ldots, PC_s$, where $s = |\mathrm{PI}(f)|$. By lemma 2.2, each parse-class has at least one parse-graph whose positive variables correspond exactly with those of the prime implicant associated with the parse-class. We shall refer to one such parse-graph as a *representative* of the parse-class.

Thus, for any B_n computing a monotone f, we can put $P(B_n)$ in the following normal form: the consistent monomials of $P(B_n)$ can be partitioned into $|\mathrm{PI}(f)|$ parse-classes; in each parse-class, there are one or more monomials whose positive variable set coincides with that of the prime-implicant corresponding to the class and each of the rest of the monomials in the parse-class contains this set as a subset of its positive variable set.

Example 2.1 Consider the Boolean function $f = x_1x_2 + x_2x_3$. The following is the formal polynomial of a possible circuit B_3 computing f: $P(B_3) = x_1^2x_2\bar{x}_3 + x_1x_2^2x_3 + x_2^2x_3\bar{x}_1 + x_2x_3x_1^2 + x_1x_2\bar{x}_2$.

2.2 Arithmetic Circuits that Count

Arithmetic circuits with $\{+, \times\}$ nodes have traditionally been defined as algebraic circuits over the field of reals. Valiant [16] studied the power of negations in this setting and showed that a single subtraction can lead to an exponential gain in size for counting the number of perfect matchings in triangular grid graphs. [11, 6] proved non-trivial lower bounds on the size and depth of arithmetic circuits that compute certain multilinear polynomials with non-negative 0-1 coefficients.

In this work, we are interested in a special type of arithmetic circuit, namely, those that compute functions of the form $f : \{0,1\}^* \rightarrow \mathcal{N}$. We define a *counting* arithmetic circuit A_n similarly to a Boolean circuit except that the interior nodes are labeled nodes labeled from $\{+, \times\}$ and the leaf nodes labeled from $\{x_i, (1 - x_i), 0, 1 \mid 1 \leq i \leq n\}$, where each $x_i \in \{0,1\}$. We say that A_n is *monotone* if the leaves are labeled only from $\{x_i, 0, 1 \mid 1 \leq i \leq n\}$. Each A_n computes a unique function $g : \{0,1\}^n \rightarrow \mathcal{N}$. The *size* and *depth* of A_n is defined as before. As in the case of Boolean circuits, we can associate a formal polynomial $P(A_n)$ with an arithmetic circuit A_n. We define the *degree* of A_n to be the degree of $P(A_n)$.

The reason for our interest in counting arithmetic circuits is that there are characterizations of popular counting classes such as $\sharp\mathcal{P}$ and $\sharp\mathcal{LOGCFL}$ in terms of these circuits [19, 20], summarized in the theorem below. The uniformity condition used below is the notion of U_D-uniformity defined by Ruzzo [9].

Theorem 2.1 [18, 19] $\sharp\mathcal{P}$ is the class of functions computable by uniform families of counting arithmetic circuits within polynomial depth and polynomial degree. $\sharp\mathcal{LOGCFL}$ is the class of functions computable by uniform families of arithmetic circuits within polynomial size and polynomial degree.

Therefore, proving non-trivial lower bounds on the size of counting arithmetic circuits that compute natural functions may have implications in separation of counting classes.

Unlike algebraic circuits over reals, counting arithmetic circuits cannot compute polynomials with negative coefficients since -1 is not available as a constant and the circuit receives 0-1 inputs. Further, for computing a function represented by a multilinear polynomial, the formal polynomial associated with a counting arithmetic circuit need not be multilinear, since 0 and 1 are the only input values and are both idempotent with respect to the $\times$ operator.

We would like to be able to prove the following result:

Conjecture 2.1 PERM cannot be computed by polynomial size arithmetic circuits.

We expect this to be true because if there were polynomial size arithmetic circuits for PERM, then Toda's result that $\mathcal{PH} \subseteq \mathcal{P}^{\sharp\mathcal{P}}$ [13] would lead to the collapse of $\mathcal{PH}$. Proving this conjecture seems hard since in conjunction with theorem 2.1, it would imply that PERM

$\notin \sharp\mathcal{LOGCFL}$. This is turn would imply that $\sharp\mathcal{LOGCFL}$ is properly contained in $\sharp\mathcal{P}$, since PERM is known to be in $\sharp\mathcal{P}$ [15]. Moreover, it would also have the more severe implication that PERM $\notin \mathcal{FP}$. So, we prove a weaker version of this statement which is motivated in the next two sections.

2.3 Arithmetic Circuits as Restricted Boolean Circuits

In this section, we consider Boolean circuits that are restricted just enough that they behave like arithmetic circuits.

For a given Boolean function f, let us define the function $\sharp f : \{0,1\}^* \to \mathcal{N}$ such that $\sharp f(x)$ is the number of prime implicants of f satisfied on input x.

Let A_n be an arithmetic circuit computing $\sharp f$. Consider the Boolean circuit B_n obtained by replacing each $+$-node of A_n with an $\vee$-node, each $\times$-node with a $\wedge$-nodde and the $(1-x_i)$ leaves with $\bar{x}_i$. It is easily verified that B_n computes f. But conversely, if we start with a Boolean circuit B_n computing f and generate an arithmetic circuit A_n by doing the reverse replacements, then on input x, A_n does not necessarily count the number of prime implicants of f that are satisfied on x. This is primarily due to the additive idempotence and absorption axioms of a Boolean algebra. Therefore, for the converse to hold, we need to restrict a Boolean circuit such that its arithmetization computes $\sharp f$. We call such circuits *parsimonious*. More precisely, a parsimonious Boolean circuit computes f if and only if its arithmetization computes $\sharp f$. Thus, instead of studying arithmetic circuits for $\sharp f$, one could equivalently study parsimonious Boolean circuits for f.

Since PERM is the counting version of BPM, the analogue of conjecture 2.1 is,

Conjecture 2.2 BPM cannot be computed by polynomial size parsimonious Boolean circuits.

Razborov's super-polynomial size lower bound [8] for monotone Boolean circuits computing BPM implies that there are no polynomial size *monotone* parsimonious circuits for BPM and therefore no polynomial size monotone arithmetic circuits for PERM. But the above result has the exact same consequences as before and therefore is hard to prove. So we consider a weaker restriction on Boolean circuits for BPM whose arithmetization computes PERM, for which we can prove an exponential size lower bound.

3 Bijective Boolean Circuits

By lemma 2.2 above, the formal polynomial of a general Boolean circuit computing a monotone function must contain *at least* $|\mathrm{PI}(f)|$ consistent monomials. Thus, requiring a bijection to exist between the set of consistent formal monomials and the set of prime implicants restricts the Boolean model of computation. This suggests the following restriction on Boolean circuits.

Definition 3.1 A Boolean circuit B_n computing the function f is said to be *bijective* if the number of consistent monomials in $P(B_n)$ equals $|\mathrm{PI}(f)|$.

Bijective circuits are in general powerful enough to compute all Boolean functions. This is because the circuit based on the representation of a Boolean function as a sum of its prime implicants is bijective. A natural question about bijective circuits is whether they can compute all Boolean functions within the same resources as general Boolean circuits. In section 4.1, we answer this in the negative by showing that bijective circuits require exponential size to compute bipartite perfect matching. This function is known to be in $\mathcal{P}$ and therefore can be computed within polynomial size using general Boolean circuits. Thus, there are functions for which the bijectivity restriction leads to an exponential blowup in size. However, there also are functions that bijective circuits can compute within the same resources as general Boolean circuits. For example, it can be checked that the $\mathcal{NC}^1$ circuit for PARITY is bijective.

If B_n is a bijective circuit that computes a monotone function f, then by lemmas 2.1 and 2.2, every consistent monomial ρ of $P(B_n)$ corresponds to a term $t \in \mathrm{PI}(f)$ such that $var(\rho_+) = var(t)$. It is then natural to ask whether negations are useful for bijective circuits computing a monotone function. We begin by showing that for any bijective circuit computing a certain class of monotone functions, that includes BPM, there is an equivalent monotone Boolean circuit of the same size.

Definition 3.2 A Boolean function f is defined to be *suitable* if it is monotone and for any pair of terms $t, t' \in \mathrm{PI}(f)$, $|\, var(t') - var(t) \,| \geq 2$.

Consider the monotone function BPM. Let S_n be the set of all permutation functions on n elements and for each $\pi \in S_n$ let $p_\pi = \bigwedge_{i=1}^{n} x_{i,\pi(i)}$. Then, PI(BPM) is exactly $\{p_\pi \,|\, \pi \in S_n\}$. We shall refer to the set of variables in p_π as $var(p_\pi)$. It is easy to verify that for any $\pi, \sigma \in S_n$, $|\, var(p_\pi) - var(p_\sigma) \,| \geq 2$, showing that BPM is a suitable function.

A natural example of a non-suitable function is UCONN, which takes as input the adjacency matrix of a graph G and outputs a 1 if and only if G is connected.

Lemma 3.1 If B_n is a bijective Boolean circuit computing a suitable function f, then for all consistent monomials ρ in $P(B_n)$, $var(\rho_-) = \emptyset$.

Proof: Let ρ be a consistent monomial in $P(B_n)$ with a negative literal $\bar{x}$. Consider the assignment $\mathcal{I}$ that sets all the variables in $var(\rho_+) \cup \{x\}$ to 1 and all the rest to 0. Since B_n is bijective and f is monotone, there exists a term t in $\mathrm{PI}(f)$, such that $var(t) = var(\rho_+)$. Therefore, f evaluates to 1 on input $\mathcal{I}$. But clearly, ρ evaluates to 0.

Since B_n computes f, there must be another consistent monomial ρ' in $P(B_n)$, such that ρ' evaluates to 1 on input $\mathcal{I}$. Since B_n is bijective, it follows from lemma 2.2 that there exists a term t' in $\mathrm{PI}(f)$ distinct from t, such that $var(t') = var(\rho'_+)$. In order for ρ' to evaluate to 1, the variables in $var(t')$ must be set to 1 on input $\mathcal{I}$. This implies that $var(t') \subseteq var(t) \cup \{x\}$, which is impossible since by definition of suitability, $|\, var(t') - var(t) \,| \geq 2$. This gives the desired contradiction. $\Box$

Given a bijective circuit B_n computing a suitable function f, lemmas 2.1, 2.2 and 3.1 completely determine the variable sets of each consistent monomial in $P(B_n)$. Moreover, since all consistent monomials of $P(B_n)$ must be monotone, given B_n we can produce a *monotone* bijective circuit for f of the same size by simply tying all the $\{\bar{x}_i\}$ inputs of B_n to the constant 0. Therefore, we have

Theorem 3.1 If B_n is a bijective Boolean circuit of size s that computes a suitable function f, then there is an equivalent monotone Boolean circuit B'_n of size s such that,

$$P(B'_n) = \bigvee_{t \in PI(f)} \Big(\bigwedge_{x_i \in var(t)} x_i^{k_i} \Big)$$

where each k_i is a natural number.

It is worth noting that monotone circuits for suitable functions need not be bijective. It follows from theorem 3.1 that bijective circuits for suitable functions must be parsimonious. Therefore, we have

Corollary 3.1 Given a bijective circuit B_n computing a suitable function f, the arithmetization of B_n computes $\sharp f$.

Consider a suitable function f such that $\sharp f$ is $\sharp\mathcal{P}$-hard. A polynomial size bijective circuit for f would imply a polynomial size arithmetic circuit for $\sharp f$. This in turn would lead to the collapse of $\mathcal{PH}$ due to Toda's result that $\mathcal{PH} \subseteq \mathcal{P}^{\sharp\mathcal{P}}$ [13].

Corollary 3.2 Given a suitable function f such that $\sharp f$ is $\sharp\mathcal{P}$-hard, there is no polynomial size bijective circuit for f unless $\mathcal{PH}$ collapses.

In particular, since BPM is a suitable function and since PERM is $\sharp\mathcal{P}$-hard, there is no polynomial size bijective circuit for BPM unless $\mathcal{PH}$ collapses. We make this lower bound unconditional in the next section.

4 A Lower Bound for Bipartite Perfect Matching

In this section, we prove an exponential size lower bound for computing the bipartite perfect matching function BPM using bijective circuits.

Let $m = n^2$ and let B_m be a bijective Boolean circuit with $2m + 2$ input nodes labeled from the set $\{x_{i,j}, \bar{x}_{i,j}, 0, 1 \mid 1 \leq i, j \leq n\}$, computing BPM. By theorem 3.1 and the fact that BPM is a suitable function, we have,

Proposition 4.1 For any bijective circuit computing BPM, there is a monotone bijective circuit for BPM of the same size.

Thus B_m is monotone and $P(B_m)$ has exactly $n!$ monomials of the form $\bigwedge_{i=1}^{n} x_{i,\pi(i)}^{k_i}$, one for each $\pi \in S_n$, where each k_i is an integer. We shall loosely refer to each consistent monomial in $P(B_m)$ as a *permutation*. Since BPM is a suitable function and its counting version PERM is known to be $\sharp\mathcal{P}$-complete [15], by corollary 3.2 we have a conditional lower bound on the size of bijective circuits for BPM. But since bijective circuits for BPM are monotone, Razborov's super-polynomial size lower bound [8] for monotone Boolean circuits computing the Boolean permanent applies and we have,

Fact 4.1 Bijective circuits for BPM require super-polynomial size.

In the next section, we improve this size lower bound to exponential by extending the lower bound arguments of [6] to hold for bijective circuits.

4.1 Adaptation of Jerrum and Snir's Framework

The main difference between this model and that used in [6] is that the circuits used here need not be multilinear. Throughout this section, we shall use the fact that B_m has exactly $n!$ parse-graphs, each computing a permutation.

Definition 4.1 For an $\wedge$-node α, let $m(\alpha)$ be the number of parse-graphs of B_m in which α appears.

Definition 4.2 An $\wedge$-node is said to be *(r,d)-significant* for $1 \leq r \leq n$ and $0 \leq d \leq \lfloor \frac{r}{2} \rfloor$, if it participates in a parse-graph with a term that has r variables, d of which are contributed by one of its immediate predecessors alone.

We note that if an $\wedge$-node α is not (r,d)-significant for any (r,d), then $m(\alpha) = 0$. Moreover, as a part of the proof of lemma 4.1 below, we show that α cannot be (r,d)-significant for more than one (r,d) pair.

Definition 4.3 Let H be a subgraph of a parse-graph G. Define the *weight* of H as follows: $W(H) = \sum_{\alpha \in \wedge-\text{nodes}(H)} \frac{1}{m(\alpha)}$, where $\wedge$-$nodes(H)$ denotes the set of $\wedge$-nodes in H.

A lemma similar to the one below was proven in [6] for their model. To prove it for bijective circuits, we need to take into account the fact that the parse-graphs are not necessarily trees.

Lemma 4.1 If α is an (r,d)-significant $\wedge$-node of B_m, then $m(\alpha) \leq d!(r-d)!(n-r)!$.

Proof: Let β and γ be the immediate predecessors of α in B_m. Let G be a parse-graph in which α appears with an r-variable term. Let a be the term formed at β and b be the term formed at γ such that $a \cdot b$ has $r \geq 1$ variables and a has $0 \leq d \leq \lfloor r/2 \rfloor$ variables that are not in b (see figure 1). Let c be a term such that $a \cdot b \cdot c$ is the n-variable term formed at the root of G. Clearly, c has $n - r$ variables that are not in a or b. Note that the sets $var(a)$, $var(b)$ and $var(c)$ may have non-empty intersections since G is a parse-graph as opposed to a parse-tree.

Let α participate in another parse-graph G'. Since α is an $\wedge$-node, β and γ participate in G' as well. In G', let a' be the term formed at β and b' be the term formed at γ. Let $a' \cdot b' \cdot c'$ be the term formed at the root of G'. Now since B_m is bijective, the term computed by G' must be different from that computed by G. Moreover, since α is an $\wedge$ node, there must be parse-graphs in B_m that compute the rest of the terms in the product $(a+a')(b+b')(c+c')$. But every term computed by a parse-graph of B_m must be a permutation. Thus, the number of parse-graphs in which α participates is the product of the number of distinct a's, b's and c's, where a is a term formed at β, b is a term formed at γ and c is a term such that $a \cdot b \cdot c$ is a permutation.

We first show that for a fixed r and d, $m(\alpha)$ cannot exceed $d!(r-d)!(n-r)!$. Let

$$\begin{aligned} A &= var(a) - \{var(b) \cup var(c)\} \\ B &= var(b) - \{var(a) \cup var(c)\} \\ C &= var(c) - \{var(a) \cup var(b)\} \end{aligned}$$

It is easy to see that α can participate in any parse-graph that computes the term $a' \cdot b' \cdot c'$, where the variables in a' are obtainable by permuting within the indices of the variables in A and b', c' are obtainable similarly from B and C, respectively. Therefore, α can participate in $|A|!\,|B|!\,|C|!$ parse-graphs. Clearly, $|A|!\,|B|!\,|C|! \leq d!\,(r-d)!\,(n-r)!$.

Suppose α participates in a parse-graph G' different from the $|A|!\,|B|!\,|C|!$ parse-graphs described above such that in G', β computes the term a', γ computes the term b' and the output node computes the term $a' \cdot b' \cdot c'$. Now, by choice of G' and bijectivity of B_m, $a' \cdot b' \cdot c'$ must be a permutation different from the $|A|!\,|B|!\,|C|!$ permutations above. Thus, the indices

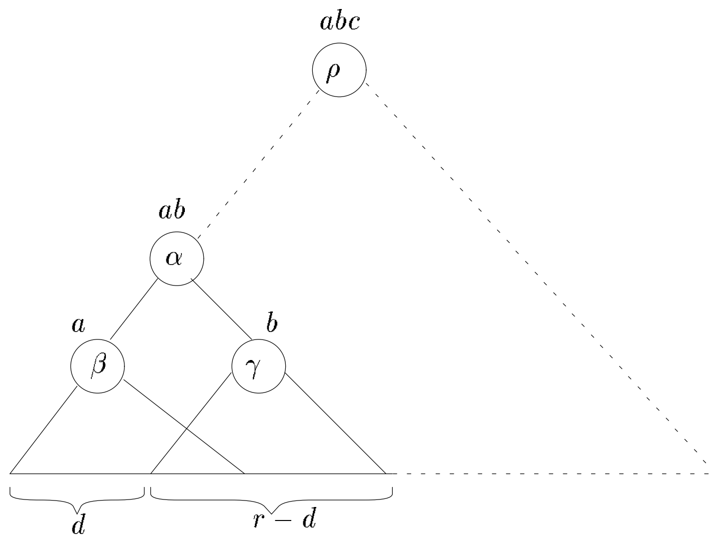

Figure 1: (r,d)-significant node α embedded in bijective circuit B_m with output node ρ.

of the variables in at least one of a', b' or c' are not obtainable by permuting within the indices of the variables in A, B or C, respectively. Without loss of generality, suppose this is true for a' (the argument is symmetrical for b' or c'). Then, there must be a variable x_{ij} in $var(a')$ such that either i is not the row index of any variable in A, or j is not the column index of any variable in A or both. But then, the term $a' \cdot b \cdot c$ is not a permutation and hence cannot be a monomial of $P(B_m)$, by theorem 3.1. Therefore, G' cannot exist. Note that we did not make any assumptions about a' or b' other than that $a' \cdot b' \cdot c'$ is a permutation different from those counted earlier. Therefore, this argument holds even if $|var(a' \cdot b')| = r'$ and $|var(a') - var(b')| = d'$, where $r' \neq r$ and $d' \neq d$. Since this means that the node α contributes to $m(\alpha)$ only for a single value of r and a single value of d, $m(\alpha)$ is bounded above by $d!(r-d)!(n-r)!$. □

The lemma below is motivated by theorem 3.3 in [6], where $\{G_i \mid 1 \leq i \leq n!\}$, are the parse-graphs of B_m. Let $\Lambda = \{\wedge\text{-node } \alpha \mid m(\alpha) \geq 1\}$.

Lemma 4.2 $\sum_{i=1}^{n!} W(G_i) = \mid \Lambda \mid$.

Proof: By definition,

$$\sum_{i=1}^{n!} W(G_i) = \sum_{i=1}^{n!} \sum_{\alpha \in \wedge-\text{nodes}(G_i)} \frac{1}{m(\alpha)}.$$

Fix an $\wedge$-node α. For each parse-graph G_i, the contribution by α to the sum on the right-hand side of the above equation is either 0 (if α does not occur in it) or $\frac{1}{m(\alpha)}$. Thus, the total contribution by α is $m(\alpha)\frac{1}{m(\alpha)} = 1$ and therefore the right-hand side is the number of $\wedge$-nodes in Λ. □

To obtain a lower bound on the weight of a parse-graph G, we consider the number of input variables covered by G, instead of the notion of degree used in [6]. This is primarily because $P(B_m)$ is not necessarily multilinear in our model.

For any subgraph H of a parse-graph of B_m, let $v(H)$ denote the number of variables in the term associated with H. Let $c(r,d) = d!(r-d)!(n-r)!$. The lemma below is adapted from theorem 3.4 in [6].

Lemma 4.3 Let H be a subgraph of any parse-graph G. Then, $W(H) \geq \sum_{i=2}^{v(H)} \frac{1}{c(i,1)}$.

Proof: The proof is by induction on the number of nodes in H. For the base case, H has a single node. Since it must be a leaf, $v(H) = 1$, and since H has no $\wedge$-nodes, $W(H) = 0$. Thus, the lemma holds.

For the induction step, let α be the root node of H and let $v(H) = r$. Without loss of generality, we assume that α is an $\wedge$-node. Since α appears in G, it must be (r,d)-significant for some d, $0 \leq d \leq \lfloor \frac{r}{2} \rfloor$. Let α have immediate predecessors β and γ, with β contributing d variables alone (see figure 1). Let H_γ be the subgraph of H rooted at γ. Clearly, $v(H_\gamma) = (r-d)$. Now, there is a subgraph $\tilde{H}_\beta$ rooted at β such that $v(\tilde{H}_\beta) = d$. For each leaf x in the set of d leaves in figure 1, there is a path P_x from x to β that is disjoint from H_γ. Let $\tilde{H}_\beta$ be the edge-induced subgraph of H_β defined on the union of the edge sets of P_x, for all x. We have,

$$W(H) \geq W(\tilde{H}_\beta) + W(H_\gamma) + \frac{1}{m(\alpha)}.$$

Note that $m(\alpha) \geq 1$ since α appears in G. Applying the induction hypothesis to the subgraphs $\tilde{H}_\beta$ and H_γ and using lemma 4.1, we get:

$$W(H) \geq \sum_{i=2}^{d} \frac{1}{c(i,1)} + \sum_{i=2}^{(r-d)} \frac{1}{c(i,1)} + \frac{1}{c(r,d)}.$$

The expression on the right is shown in [6] to be minimum at $d = 1$ in the range $1 \leq d \leq \lfloor \frac{r}{2} \rfloor$. In the range $0 \leq d \leq \lfloor \frac{r}{2} \rfloor$, this expression attains its minimum value at $d = 1$ as well, since $\frac{1}{c(r,0)} \geq 0$. □

The above lemmas lead to the lower bound for BPM using bijective circuits.

Theorem 4.1 Any bijective Boolean circuit B_m requires size $\geq n(2^{n-1} - 1)$ to decide whether a bipartite graph has a perfect matching.

Proof: From lemmas 4.2 and 4.3 it follows that the size of any bijective Boolean circuit for this problem is at least $\sum_{j=1}^{n!} \sum_{i=2}^{v(G_j)} \frac{1}{c(i,1)}$. But $v(G_j) = n$ since every parse-graph of B_m computes a permutation. Therefore, the above expression is equivalent to $n! \sum_{i=2}^{n} \frac{1}{c(i,1)}$. Substituting for $c(i,1)$ in $\sum_{i=2}^{n} \frac{1}{c(i,1)}$ we get, $\sum_{i=2}^{n} \frac{1}{(n-i)!\,(i-1)!}$ which is exactly $\frac{2^{n-1}-1}{(n-1)!}$, from which the theorem follows.□

4.2 Notes on the Lower Bound

A few points are worth noting:

Upper Bound The Boolean circuit based on the permanent analogue of Laplace's expansion rule for determinants computes BPM within size $O(n(2^{n-1} - 1))$ [6]. This circuit happens to be bijective. Therefore, the lower bound presented above is tight.

Depth Lower Bound The above size lower bound immediately implies a linear depth lower bound for computing the Boolean permanent function with bijective Boolean circuits. This depth bound also follows from proposition 4.1 and a linear depth bound for this function using monotone Boolean circuits proven by Raz and Wigderson [7].

Lower Bound for Hamiltonian Cycle The Hamiltonian cycle function HC : $\{0,1\}^{n^2} \rightarrow \{0,1\}$ takes as input the standard $n \times n$ adjacency matrix representation of a graph G and outputs 1 if and only if G has a Hamiltonian cycle. HC is a monotone function and each of its prime implicants corresponds to a Hamiltonian cycle of the complete graph on n vertices. Let C_n be the set of all cyclic permutation functions on n elements and for each $\pi \in C_n$ let $p_\pi = \bigwedge_{i=1}^{n} x_{i,\pi(i)}$. Then, *PI*(HC) is exactly $\{p_\pi \,|\, \pi \in C_n\}$. As in the case of BPM, HC is a suitable function.

Since HC is a monotone function, lemmas 2.1 and 2.2 hold and since it is also suitable, lemma 3.1 and theorem 3.1 hold. The arguments used in section 4 can then be easily adapted to obtain an exponential size lower bound for computing HC using bijective Boolean circuits, by simply considering cyclic permutations in place of all permutations. Since HC is a suitable function, this lower bound also follows from the fact that bijective circuits for HC must be monotone (theorem 3.1) and the exponential size lower bound for computing HC using monotone Boolean circuits [1].

4.3 Generalizing the Lower Bound

We now show that the lower bound presented above holds for a model that is slightly more general than bijective circuits.

Definition 4.4 A Boolean circuit B_n is said to be *homogeneous* if all the consistent monomials of $P(B_n)$ have the same number of positive variables.

Let B_m be a monotone homogeneous circuit for BPM. Recall the canonical formal polynomial of a general Boolean circuit computing a monotone function (section 2.1). The monotonicity restriction rids it of negative literals and the homogeneity restriction allows only those monomials to survive whose variable sets correspond exactly to that of some prime implicant. Thus, whereas a bijective circuit for BPM has exactly $n!$ parse-graphs, a monotone homogeneous circuit could have more than one parse-graph in each parse-class although each of them cover the same set of variables. In this sense, bijective circuits for BPM are a special case of monotone homogeneous circuits for BPM.

The lower bound is proved using a set of *representatives* $\{G_i | 1 \leq i \leq n!\}$, one from each parse-class, instead of the $n!$ parse-graphs used in section 4.1. Definitions 4.1, 4.2, 4.3, the statement of lemma 4.3 and the proofs of lemmas 4.1, 4.2 and 4.3 are appropriately altered for representative parse-graphs.

Theorem 4.2 Monotone homogeneous circuits require size $\geq n(2^{n-1} - 1)$ to compute BPM.

For any monotone Boolean circuit B_n computing BPM, $P(B_n)$ can be simplified into the form $\vee_{\pi \in S_n} p_\pi$ by applying a sequence of semi-ring axioms and axioms of Boolean algebra such as, idempotence ($x.x = x$; $x + x = x$) and absorption ($x + xy = x$). Now the absorption axiom is used in the simplification if and only if $P(B_n)$ has one or more monomials that have more than n variables. This is because none of these monomials appear in the final reduced form and therefore must be absorbed. Thus, monotone homogeneous Boolean circuits correspond in some sense to the restricted model of Boolean computation obtained by taking away the absorption axiom.

5 Consequences

In this section, we present consequences of the lower bound result in three models of computation.

5.1 Monotone Arithmetic Circuits

We establish a close connection between monotone arithmetic circuits for 0-1 permanent and bijective Boolean circuits for BPM. The lower bound result for bijective circuits in section 4 then implies an exponential size lower bound for monotone arithmetic circuits that compute the 0-1 permanent function.

Let $m = n^2$ and let A_m be a monotone arithmetic circuit that computes PERM : $\{0,1\}^m \to \mathcal{N}$. Since A_m is monotone, the monomials of $P(A_m)$ are trivially consistent.

Now, let B_m be the monotone Boolean circuit obtained by "Booleanizing" A_m, that is, replacing each $\times$ node with an $\wedge$-node and each $+$ node with an $\vee$ node. It is easy to verify that B_m computes BPM. By the analogues of lemmas 2.1 and 2.2, $P(A_m)$ has at least $n!$ monomials. If it had any more, then on the input that assigns all variables to 1, PERM evaluates to $n!$ but A_m would compute a value strictly greater than $n!$. Thus, $P(A_m)$ has exactly $n!$ monomials. Now, since $P(B_m)$ is formally identical to $P(A_m)$ by construction, it follows that B_m must be bijective. The lower bound below now follows from theorem 4.1.

Theorem 5.1 Any monotone arithmetic circuit A_m requires size $\geq n(2^{(n-1)} - 1)$ to compute PERM.

Proof: Suppose A_m had smaller size. Then, the bijective Boolean circuit obtained by Booleanizing A_m, would compute BPM within size smaller than $n(2^{(n-1)} - 1)$. But by theorem 4.1, this is impossible. □

We note that this lower bound on monotone arithmetic circuits computing PERM extends that obtained by [6] to the 0-1 permanent function. It is also worth noting that this bound does not follow from the work of Valiant [16].

5.2 Monotone Multilinear Circuits

Let B_n be a monotone multilinear Boolean circuit for BPM. In this section, we show that given such a B_n, it is possible to construct a circuit B'_n computing BPM, while at most squaring the size, such that B'_n is monotone homogeneous. An exponential lower bound on the size of monotone multilinear circuits for BPM then follows from the arguments in section 4.3.

Recall the canonical formal polynomial of a general Boolean circuit for a monotone function (section 2.1). Monotonicity rids it of negative literals and multilinearity rids each monomial of its degree. Thus, each parse-class has one or more monomials that look exactly like the prime implicant corresponding to the class. We now provide a construction that takes a monotone multilinear circuit and produces one in which only these monomials survive. Thus, the resulting circuit is monotone, homogeneous, multilinear and computes the same function as the original circuit.

Lemma 5.1 Given a monotone multilinear circuit B_n of size s computing a homogeneous function f with p variables, there is a monotone, multilinear, homogeneous circuit B'_n that computes f within size $O(s^2)$.

Proof: Given B_n, we first construct an equivalent circuit C_n that has the following normal form: (i) C_n has alternating $\vee$ and $\wedge$ layers; (ii) the output node is an $\vee$ node and all circuit inputs are inputs to $\vee$ nodes; and (iii) each $\wedge$ node has fan-in two. It is easily verified that the size of C_n is at most twice that of B_n.

Given C_n, we construct an equivalent circuit B'_n such that each monomial of $P(B'_n)$ has exactly p variables. This is achieved by essentially keeping a count of the number of variables covered at a node.

- For every leaf node A in C_n create a leaf node A in B'_n.
- For every $\vee$-node A in C_n create the $\vee$-nodes $[A, i, 0]$, $0 \le i \le p$, in B'_n.
- For every $\wedge$-node A in C_n create the $\vee$-nodes $[A, i, 1]$, $0 \le i \le p$, in B'_n.
- For all $0 \le i \le p$, the inputs to an $\vee$-node of the form $[A, i, 1]$ are $\wedge$-nodes $[A, i, j, k]$, for all j, k such that $0 \le j, k \le p$ and $j + k = i$.
- For all i, j, k, inputs to the $\wedge$-node $[A, i, j, k]$ are the $\vee$-nodes $[B, j, 0]$ and $[C, k, 0]$, where B and C are the inputs of the $\wedge$-node A in C_n.
- For all $0 \le i \le p$, the inputs to an $\vee$-node of the form $[A, i, 0]$ are set as follows: for each input B of the $\vee$-node A in C_n, (a) if B is an $\wedge$-node, make $[B, i, 1]$ an input of $[A, i, 0]$, for all $0 \le i \le p$; (b) if B is a leaf node labeled with a variable x, $[A, 1, 0]$ has x as its input, and for all $i \ne 1$, $[A, i, 0]$ gets the constant 0 as an input; and (c) if B is a leaf node labeled with a constant c, $[A, 0, 0]$ has c as its input, and for all $1 \le i \le p$, $[A, i, 0]$ gets the constant 0 as an input.

The size of B'_n is at most a square of that of C_n. It is also easily verified that the formal monomials of B'_n are those of B_n that have exactly p variables. Since the construction preserves monotonicity and multilinearity, B'_n is a monotone, multilinear and homogeneous circuit that computes f. $\Box$

Using lemma 5.1 and theorem 4.2, we have

Theorem 5.2 Monotone multilinear circuits require size $\Omega(\sqrt{n(2^{n-1}-1)})$ to compute BPM.

Note that since the construction above essentially pulls out the monomials of $P(B_n)$ that have algebraic degree n, the lower bound in theorem 5.2 also holds for monotone Boolean circuits in which not all formal monomials are multilinear but there is at least one multilinear formal monomial in each parse-class whose variable set is exactly that of the prime implicant corresponding to the class. Let us call such circuits *nearly-multilinear*. Then we have,

Theorem 5.3 Monotone nearly-multilinear circuits require size $\Omega(\sqrt{n(2^{n-1}-1)})$ to compute BPM.

5.3 Circuits Over $(min, concat)$

We consider the semiring MIN $= (\Sigma^* \cup \{\perp\}, +, \times)$, where Σ is any alphabet, $\times$ denotes concatenation ($\perp \times x = x \times \perp = \perp$, for all x) and $+$ denotes lexicographic minimum ($x + \perp = \perp + x = x$, for all x).

A circuit M_n over MIN is a rooted, acyclic, digraph with interior nodes labeled with $+$ or $\times$. The leaf nodes of the circuit are labeled either with an input variable x_i, $1 \leq i \leq n$, or with some element of $\Sigma \cup \{\perp\}$. The size and depth of M_n and the formal polynomial $P(M_n)$ associated with M_n are defined as before.

Consider the function LMBPM: $\{\Sigma \cup \perp\}^{n^2} \rightarrow \Sigma^n \cup \{\perp\}$, which takes as input an $n \times n$ matrix $\mathrm{X} = [x_{ij}]$ with $x_{ij} \in \Sigma \cup \{\perp\}$. The input encodes a bipartite graph G such that if $x_{ij} \in \Sigma$, then the value of x_{ij} denotes the order on the edge (i, j) in G; if $x_{ij} = \perp$, then there is no edge (i, j) in G. On this input, if G has a perfect matching, LMBPM outputs a string of length n over the alphabet Σ, that encodes the lexicographically minimum perfect matching. If G does not have a perfect matching, the function outputs $\perp$.

The algorithm below computes LMBPM, where BPM(G) decides whether G has a perfect matching and $\times$ denotes concatenation. Since there is a polynomial time algorithm for BPM [5], LMBPM is therefore in $\mathcal{P}$. We will now show that circuits over MIN require exponential size to compute this function.

```
begin
1. res = ⊥;
2. for i = 1, |E(G)| do
   /* Each edge in G is visited in order */
      2.1 if BPM(G − e_i) then
      /* G − e_i is the graph obtained from G by deleting the vertices
          of e_i and all incident edges */
          2.1.1 res = res ×{e_i};
          2.1.2 G = G − e_i;
          end-if
   end-for
3. return(res);
end
```

Figure 2: LMBPM is in $\mathcal{P}$.

Let $m = n^2$. Let M_m be a circuit over MIN that computes LMBPM. Let B_m be the Boolean circuit obtained from M_m by replacing each $+$-node with an $\vee$-node and each $\times$-node with an $\wedge$-node. Moreover, if M_m has the matrix $\mathrm{X} = [x_{ij}]$ as input, then $\mathrm{Y} = [y_{ij}]$, the input to B_m, is derived as follows : if $x_{ij} \in \Sigma$, then $y_{ij} = 1$, otherwise $y_{ij} = 0$. Clearly, B_m is a monotone Boolean circuit. We now show that B_m is also nearly-multilinear and computes BPM. This is primarily because the elements of Σ are not idempotent with respect to the $\times$ operator.

Lemma 5.2 If M_m computes LMBPM on input X, then B_m is a monotone nearly-multilinear circuit that computes BPM on input Y.

Proof : The bipartite graph G encoded by X is simply the one encoded by Y, with a total order on its edges. Let $P(M_m)$ and $P(B_m)$ be the formal polynomials associated with M_m and B_m respectively. There is clearly a bijection between the monomials of $P(M_m)$ and those of

$P(B_m)$. Now, if G has a perfect matching, then there is at least one monomial in $P(M_m)$ all of whose variables receive values from Σ. By construction, all the variables in the corresponding monomial in $P(B_m)$ receive the value 1 on input Y. Therefore, B_m evaluates to 1. Conversely, if G doesn't have a perfect matching, then for every monomial in $P(M_m)$, there is at least one variable that receives the value $\perp$. Therefore, for every monomial in $P(B_m)$, there is at least one variable that gets a 0 value on input Y, causing B_m to evaluate to 0. Thus, B_m computes BPM.

To show that B_m is nearly-multilinear, we need to show that for all $\pi \in S_n$, the monomial $p_\pi = \wedge_{i=1}^{n} y_{i,\pi(i)}$ appears in $P(B_m)$. Suppose for some π, p_π does not appear in $P(B_m)$. By construction, $q_\pi = \wedge_{i=1}^{n} x_{i,\pi(i)}$ does not appear in $P(M_m)$. On the input to M_m for which q_π is the lexicographically first bipartite perfect matching in the input graph, the output of M_m disagrees with that of the function LMBPM thereby giving the desired contradiction. $\Box$

By theorem 5.3 we have,

Corollary 5.1 Circuits over MIN require size $\geq n(2^{n-1} - 1)$ to compute LMBPM.

A linear depth lower bound immediately follows for circuits over MIN computing LMBPM.

6 Concluding Remarks

In this paper we introduce the notion of bijective Boolean circuits and show that such circuits require exponential size to compute the perfect matching function for bipartite graphs. Since this function is known to be in $\mathcal{P}$, it follows that general Boolean circuits are exponentially more powerful than bijective circuits and that the polynomial size circuit for this function cannot be bijective. This size bound also implies a linear depth lower bound in this model, which in turn implies that if an $\mathcal{NC}$ circuit for BPM exists, it cannot be bijective. We also showed some interesting consequences of this size bound for computations using monotone multilinear circuits, monotone arithmetic circuits and circuits over $(min, concat)$. While bipartite perfect matching is known to be in $\mathcal{P}$ and Hamiltonian cycle is $\mathcal{NP}$-complete, the counting versions of both functions are complete for $\sharp\mathcal{P}$. This suggests that both these functions ought to be equally hard in some sense. By showing that both of them require exponential size on bijective Boolean circuits, we exhibit one such setting and thereby take a step in the direction of understanding the phenomenon of easy decision problems having hard counting versions. In fact, we identify a class of monotone functions such that if their counting version is $\sharp\mathcal{P}$-hard, then there are no polynomial size bijective circuits for such functions unless $\mathcal{PH}$ collapses.

We conclude with a few open questions that this work raises:

- Are there other functions for which the approach presented here can be used to derive non-trivial lower bounds in the bijective circuit model? The functions BPM and HC considered in this paper are both monotone. What about non-monotone functions?
- What can be said about the class of natural decision versions of $\sharp\mathcal{P}$-complete functions that require exponential size bijective circuits?
- In all the examples of natural restricted circuits that we have looked at, bijectivity seems to appear only in conjunction with multilinearity. Are there natural circuits that are bijective but not multilinear?
- What would be an appropriate notion of reducibility for the bijective circuit model?

- Are there other interesting restrictions on Boolean circuits for which lower bounds for natural functions can be obtained?
- Can one obtain a super-polynomial size lower bound on parsimonious circuits for BPM?

References

[1] N. Alon and R.B. Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica, 7 (1987), pp. 1-22.

[2] R. Boppana and M. Sipser, *The complexity of finite functions*, In *The Handbook of Theoretical Computer Science*, J. van Leeuwen ed., Elsevier Science Publishers B.V. (1990) 759-804.

[3] A. Chandra, L. Stockmeyer and U. Vishkin, *Constant depth reducibility*, SIAM J. Comput., 13 (1984), pp. 423-439.

[4] M. Grigni and M. Sipser, *Monotone separation of Logspace from* NC^1, Proc. 6th Annual IEEE Conference on Structures in Complexity Theory (1991), pp. 294-298.

[5] J.E. Hopcroft and R.M. Karp, *A* $n^{5/2}$ *algorithm for maximum matching in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225-231.

[6] M. Jerrum and M. Snir, *Some exact complexity results for straight-line computations over semi-rings*, J. Assoc. Comput. Mach., 29 (1982), pp. 874-897.

[7] R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, Proc. 22nd Annual ACM Symposium on Theory of Computing (1990), pp. 287-292.

[8] A.A. Razborov, *A lower bound on the monotone network complexity of the logical permanent*, Mathematischi Zametki 37, (1985), pp. 887-900.

[9] W. L. Ruzzo, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365-383.

[10] J. E. Savage, *The complexity of computing*, R. E. Kreiger Publishing Co., Malabar, FL, 1987.

[11] E. Shamir and M. Snir, *On the depth complexity of formulas*, Math. Systems Theory, 13 (1980), pp. 301-322.

[12] É. Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica, 7 (1987), pp. 141-142.

[13] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), 865-877.

[14] M. Tompa and P. Tiwari, *A direct version of Shamir and Snir's lower bounds on monotone circuit depth*, Univ. of Washington Technical Report # 92-12-06, December 1992.

[15] L. Valiant, *The complexity of computing the permanent*, Theor. Comput. Sci., 8 (1979), pp. 189-201.

[16] L. Valiant, *Negation can be exponentially powerful*, Theor. Comput. Sci., 12 (1980), pp. 303-314.

[17] L. Valiant, *Why is Boolean complexity theory difficult?* In *Boolean Function Complexity Theory*, edited by M.S. Paterson, London Mathematical Society. Lecture Notes Series 169, Cambridge University Press, 1992.

[18] H. Venkateswaran, *Properties that characterize LOGCFL*, J. Comput. System Sci., 43 (1991), pp. 380-404.

[19] H. Venkateswaran, *Circuit definitions of nondeterministic complexity classes*, SIAM J. Comput., 21 (1992) pp. 655-670.

[20] V. Vinay, *Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits*, Proc. 6th Annual IEEE Conference on Structure in Complexity Theory (1991), pp. 270-284.