Characteristics of the Maximal Independent Set ZDD

David R. Morrison Edward C. Sewell Sheldon H. Jacobson

Abstract

Zero-suppressed binary decision diagrams (ZDDs) are important data structures that are used in a number of combinatorial optimization settings. This paper explores a ZDD characterization for the maximal independent sets of a graph; a necessary and sufficient condition for when nodes in the ZDD can be merged is provided, and vertex orderings of the graph are studied to determine which orderings produce smaller ZDDs. A bound on the width of the maximal independent set ZDD is obtained, relating it to the Fibonacci numbers. Finally, computational results are reported.

1 Introduction

A **binary decision diagram** (BDD) is a graph-based data structure that compactly encodes some binary function. BDDs were first described by Lee (1959) and Akers (1978), and have been used in a number of different applications, including circuit design and formal logic verification (Bryant, 1992). However, recently BDDs have also been shown to be quite useful in a number of different combinatorial optimization settings, due to their ability to completely characterize all valid solutions to the formula. This complete characterization gives rise to many useful techniques:

- Behle (2007) show how to use BDDs to enumerate vertices and facets of polyhedra, which is an important capability for integer programming theory and algorithms. Moreover, Behle and Eisenbrand (2007) uses BDDs to generate valid inequalities that can be used to cut off fractional solutions to an integer programming problem.

- Hadžić and Hooker (2008) use decision diagrams to perform post-optimality analysis on solutions to various optimization problems such as budgeting and network reliability. This post-optimality analysis uses a generalization of BDDs to characterize near-optimal solutions and perform sensitivity analysis for problems with uncertain input.

- Bergman et al. (2013) develop a method for using BDDs to compute upper and lower bounds for the maximal independent set problem. These bounds can then be used to prune off

subtrees in a branch-and-bound algorithm. In addition, the notion of an **approximate** BDD is introduced, which sacrifices an exact representation of the solution space for a smaller, easier-to-manage data structure which can still produce good bounds.

Zero-suppressed binary decision diagrams (ZDDs) are a extension of BDDs that are better suited for describing families of sets (Minato, 1993). For many combinatorial optimization problems, ZDDs are more effective because of the relative "sparsity" of the solution space; that is, given a family $\mathcal{F}$ of subsets of a universe $\mathcal{U}$, if the size of the family is small relative to $2^{|\mathcal{U}|}$, ZDDs can often describe $\mathcal{F}$ using much fewer nodes than the corresponding BDD. Coudert (1997) describes the use of ZDDs to perform a number of different set operations to compute maximal cliques, dominating sets, and other graphical structures. In addition, Minato (2001) shows how ZDDs can be used in VLSI CAD, Petri nets, and other optimization problems.

This paper extends the work of Bergman et al. (2012) by considering the application of ZDDs (instead of BDDs) to the maximal independent set problem, which is an important problem in many graph-theoretic settings. In particular, many applications are concerned with the generation of a sequence of maximal independent sets (or equivalently, maximal cliques). For instance, Eppstein and Strash (2011) describes an algorithm for listing all cliques in sparse, real-world graphs to provide information about communities within the network. Additionally, the well-known algorithm of Bron and Kerbosch (1973) is very effective at enumerating all maximal independent sets in a graph. While ZDDs can be used for enumeration, they also enable compact *storage* of all maximal independent sets. This offers a number of additional advantages, including the previously-discussed ability to perform sensitivity analysis. It also enables queries to be performed repeatedly about the family of all maximal independent sets, even in the presence of changing graph parameters.

Of particular interest in this setting is the weighted maximal independent set problem; given a ZDD characterizing the maximal independent sets of a graph, the maximum-weight maximal independent set can be found via the algorithm of Hadžić and Hooker (2008). Moreover, even if the vertex weights change, the ZDD does not have to be re-built, which can provide significant performance gains over branch-and-bound algorithms such as in Held et al. (2012). One example of the utility of this technique is given in Morrison et al. (2014), where maximal independent set ZDDs with changing weights are used to solve the pricing problem in a branch-and-price algorithm

for the graph coloring problem. Morrison et al. (2014) show significant performance gains over other algorithms that require repeated generation of maximal independent sets.

Therefore, the primary contribution of this paper is to show how ZDDs can be used to characterize all of the maximal independent sets of an input graph, as well as to analyze various properties of the maximal independent set ZDD. The remainder of this paper is organized as follows: in Section 2, a formal definition of the maximal independent set ZDD is presented, and Section 3 presents two different algorithms for constructing such a ZDD. Section 4 studies the effects of vertex order on the size of the ZDD, and Section 5 provides computational results showing the effects of construction algorithm and vertex ordering in practice. Data are presented for a subset of the graph instances from the DIMACS graph coloring challenge. Finally, Section 6 offers concluding remarks and future research directions.

2 Definitions and Notation

Given an undirected graph $G = (V, E)$ and vertices $u, v \in V$, u and v are **adjacent** (**non-adjacent**), denoted $u \leftrightarrow v$ ($u \not\leftrightarrow v$), if $\{u, v\} \in (\notin) E$. Further, given a set $X \subseteq V$, the **induced subgraph** of G with respect to X, denoted $G[X]$, is the subgraph of G defined on the vertex set X with all edges of G having both endpoints in X. The **neighbor set** of $X \subseteq V$, denoted $N(X)$, is the set of vertices adjacent to vertices in X, and the closed neighbor set $N[X]$ is $N(X) \cup X$ (if X contains a single vertex u, the set notation is dropped, e.g. $N(u)$). The degree $d(u)$ of a vertex u is $|N(u)|$; and $\Delta(G)$ is the largest degree of all vertices in G. An **independent set** $R \subseteq V$ is a set of vertices such that $G[R]$ has no edges, and R is a **maximal** independent set if there exists no vertex $v \in V \setminus R$ such that $R + v$ is an independent set. Let $\mathcal{S}$ be the family of all maximal independent sets in G. Given any set of vertices X, a node $v \in V$ is **dominated** by X if $v \in N[X]$; note that for any maximal independent set R, $N[R] = V$. The **independence number** $\alpha(G)$ is the size of the largest independent set in G.

Fix an ordering on the vertices of G, and label the vertices in this ordering as $v_1, v_2, \ldots, v_n$; a maximal independent set ZDD for G with respect to this ordering, denoted by $Z_{\mathcal{S}}$, is a directed acyclic graph (DAG) that satisfies the following properties, where $V(Z_{\mathcal{S}})$ denotes the nodes of $Z_{\mathcal{S}}$:

1. There are two special nodes in $Z_{\mathcal{S}}$ denoted **1** and **0**, called the **true** node and **false** node,

respectively. Additionally, there is exactly one "highest" node r in the topological ordering of $Z_{\mathcal{S}}$, called the **root** of $Z_{\mathcal{S}}$.

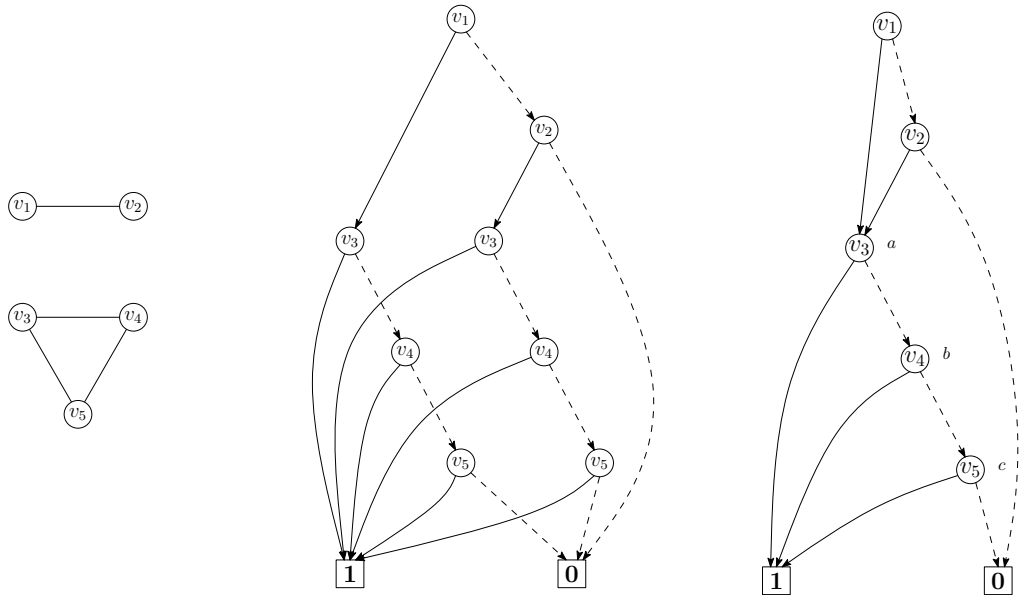
2. Every node $z \in V(Z_{\mathcal{S}}) \setminus \{\mathbf{1}, \mathbf{0}\}$ has two outgoing edges, the **high edge** and the **low edge**. The node at the head of the high (low) edge is called the high (low) child, and is denoted by $\text{hi}(z)$ ($\text{lo}(z)$). The true and false nodes have no outgoing edges.

3. Every node $z \in V(Z_{\mathcal{S}}) \setminus \{\mathbf{1}, \mathbf{0}\}$ is associated with a vertex $v_i \in V$; the index of the associated vertex is given by $\text{var}(z)$, that is, $\text{var}(z) = i$. By convention, $\text{var}(\mathbf{1}) = \text{var}(\mathbf{0}) = n+1$. Finally, if $\text{var}(z) = i$, then $\text{var}(\text{hi}(z)) > i$ and $\text{var}(\text{lo}(z)) > i$.

4. No $z \in V(Z_{\mathcal{S}})$ has $\text{hi}(z) = \mathbf{0}$.

5. Every path from the root of $Z_{\mathcal{S}}$ to $\mathbf{1}$ is in $1-1$ correspondence with a maximal independent set of G; furthermore, given such a path P, the corresponding independent set can be reconstructed by taking each v_i for which there exists $z \in P$ with $\text{var}(z) = i$ and $(z, \text{hi}(z)) \in E(P)$, where $E(P)$ denotes the edges of P.

Any DAG satisfying conditions (1)-(5) is said to **characterize** the family of all maximal independent sets of G. Condition (4) is called the **zero-suppressed** condition, and is what differentiates a ZDD from a BDD. An example graph together with a maximal independent set ZDD is given in Figure 1. Given a node $z \in V(Z_{\mathcal{S}})$, a **valid path** is a path from z to $\mathbf{1}$ in Z; such a path corresponds to a (not-necessarily-maximal in G) independent set R on the vertices $\{v_{\text{var}(z)}, v_{\text{var}(z)+1}, \ldots, v_n\}$ where $v_j \in R$ if and only if v_j is the tail of a high edge on the valid path. In this case, z **yields** R, denoted $z \vdash R$; in a slight abuse of notation, say that a vertex $v \in V$ is **dominated** by the valid path from z corresponding to R if $v \in N[R]$. Finally, note that the (weighted) independence number $\alpha(G)$ can be determined from $Z_{\mathcal{S}}$ by computing the longest root-to-$\mathbf{1}$ path in $Z_{\mathcal{S}}$; since $Z_{\mathcal{S}}$ is a DAG, this can be done in linear time with respect to the number of nodes and edges in $Z_{\mathcal{S}}$ (Sedgewick and Wayne, 2011).



(a) The graph G, together with a vertex ordering

(b) A ZDD characterizing all maximal independent sets of G

(c) The reduced ZDD for G with respect to the vertex ordering

Figure 1

3 Construction Algorithms for $Z_{\mathcal{S}}$

For any given graph and vertex ordering, there are many different ZDDs that can be constructed. However, as shown in Bryant (1986), there is a unique smallest BDD for any binary function and variable ordering; this result extends naturally to the maximal independent set ZDD. This observation is motivated by the fact that in any non-reduced ZDD, there exist isomorphic subgraphs which can be merged together to form a smaller ZDD (see Figure 1c). Furthermore, there is a natural recursive algorithm that can be used to construct $Z_{\mathcal{S}}$; this algorithm, given in Morrison et al. (2014), stores each node $z \in V(Z_{\mathcal{S}})$ in a hash table which can be searched in constant time. If there exists a node $z \in V(Z_{\mathcal{S}})$ with the same associated vertex, high child, and low child as some node z', z and z' can be merged, with the parent of z' adjusted to point to z. In this case, z and z' are equivalent, denoted $z \cong z'$. Pseudocode for this algorithm is given in Algorithm 1.

The algorithm takes as input a set of k undominated vertices U, and a current index $i \leq k+1$, and it behaves as follows: first, two base cases are checked (lines 1 and 2). In the first case, some vertex in $\{u_1, u_2, \ldots, u_{i-1}\}$ is not adjacent to any vertex in $\{u_i, u_{i+1}, \ldots, u_k\}$. In this case, there is no way to construct a maximal independent set with the remaining vertices. Alternately, if there

Algorithm 1: RecursiveBuildZDD(U, i)

input: A set $U = \{u_1, u_2, \ldots, u_k\}$ of undominated vertices such that $u_j < u_{j+1}$ with respect to the vertex ordering on V, and a "current index" i

output: The root node of a ZDD characterizing all the maximal independent sets in $G[U]$ that can be formed with vertices in $\{u_i, u_{i+1}, \ldots, u_k\}$

1 **if** $G[U]$ has a vertex with no neighbor in $\{u_i, u_{i+1}, \ldots, u_k\}$**:** return **0**
2 **if** $U == \varnothing$**:** return **1**
3 $U_H = U \setminus N[u_i]$ $\langle\!\langle$ *Use vertex u_i; remove it and its neighbors from U* $\rangle\!\rangle$
4 $h = \min\{j \mid j > i \text{ and } u_j \in U_H\}$ or $|U_H| + 1$ if no j exists
5 $z_h =$ `RecursiveBuildZDD`(U_H, h)
6 $z_l =$ `RecursiveBuildZDD`$(U, i + 1)$
7 **if** $z_h == \mathbf{0}$**:** return z_l
8 $\langle\!\langle$ *Look up element in reverse hash table* $\rangle\!\rangle$
9 **if** $\exists\ z \in V(Z_\mathcal{S})$ s.t. $\text{var}(z) = i, \text{lo}(z) = z_l,$ and $\text{hi}(z) = z_h$**:** return z
10 **else:** insert a new node z' into $Z_\mathcal{S}$, and return z'

are no undominated vertices, the set is maximal by definition. If neither base case is met, two branches are constructed. The high branch is constructed by removing the current vertex u_i and all its neighbors from the set of undominated vertices, and advancing the current index to the next undominated vertex in U (if no such vertex exists, the current index is set to an "off-the-end" value of $k + 1$). Similarly, the low branch is constructed by advancing the current index to the next undominated vertex in U, if one exists. Once the high and low children have been computed (recursively constructing them if necessary), line 7 enforces the zero-suppressed condition; line 9 checks for the existence of a node with the same label, high child, and low child, and line 10 inserts a new node into the ZDD if it does not exist. To construct the complete maximal independent set for a graph G, `RecursiveBuildZDD` $(V, 1)$ is called.

The theoretical running time of Algorithm 1 is $O(n|\overline{Z_\mathcal{S}}|)$, where $\overline{Z_\mathcal{S}}$ is the size of the ZDD produced if the merge step (line 9) is not present. A crude upper bound on $\overline{Z_\mathcal{S}}$ is 2^n; the results in Section 4 will improve this bound slightly, and in many practical cases $\overline{Z_\mathcal{S}}$ is much smaller. In comparison, the branching enumeration algorithm of Bron and Kerbosch (1973) runs in $O(3^{n/3})$ worst-case time (Tomita et al., 2006).

In this section, a necessary and sufficient equivalence condition for nodes in $Z_\mathcal{S}$ is shown that does not rely on a hash table or a recursive algorithm. In particular, this condition allows for a non-recursive top-down construction of the smallest $Z_\mathcal{S}$ for some graph G and a fixed variable

ordering. The resulting equivalence condition is similar to Theorem 1 from Bergman et al. (2012), but is somewhat more complicated due to the maximality requirement for sets in $\mathcal{S}$. Specifically, the independent set BDD described in Bergman et al. (2012) stores a single set of vertices E, called the **eligible set**, at every node in the BDD; it is shown that two nodes in the BDD are equivalent if they have the same eligible set (to avoid confusion with the edge set of the graph, this paper denotes the eligibility set by L). This idea is extended for the maximal independent set ZDD; here, however, two sets are needed, the **totally dominated set** and the **reduced eligibility set**. The main result of this section is to show that two nodes in $Z_{\mathcal{S}}$ are equivalent if and only if their totally dominated sets and reduced eligibility sets are identical. First, a few definitions are needed. For the remainder of this section, let $z \in V(Z_{\mathcal{S}})$ with $\text{var}(z) = i$.

Definition 1. *Consider an independent set $R \subseteq \{v_i, v_{i+1}, \ldots, v_n\}$ such that $z \vdash R$. The* ***i*-dominated set** *for R, denoted $\Gamma_i(R)$, is $N(R) \cap \{v_1, v_2, \ldots, v_{i-1}\}$, that is, the neighbors of R smaller than v_i.*

Definition 2. *The* **totally dominated set** *of vertices at z, denoted T_z, is the subset of $\{v_1, v_2, \ldots, v_{i-1}\}$ such that every valid path from z dominates all vertices in T_z. In other words,*

$$T_z = \bigcap_{R : z \vdash R} \Gamma_i(R).$$

For example, in Figure 1c, $T_a = \varnothing$, $T_b = \{v_3\}$, and $T_c = \{v_3, v_4\}$. Note that, given some root-to-z path, it is not necessary for T_z to actually be undominated at z; the definition of totally dominated only requires that all valid paths starting from z dominate v. However, any undominated nodes along such a root-to-z path must belong to T_z.

Additionally, note that for nodes $z, z' \in V(Z_{\mathcal{S}})$, if there exists a vertex $v \in T_{z'}$ and $v \notin T_z$, then v is undominated by at least one valid path from z, but is dominated by all valid paths from z'. This proves the following lemma:

Lemma 1. *Any two equivalent nodes $z, z' \in V(Z_{\mathcal{S}})$ must satisfy $T_z = T_{z'}$.*

The next definitions describe the other necessary set that must be tracked to determine node equivalence in $Z_{\mathcal{S}}$.

Definition 3. *Define an* **eligibility set** *at z to be $L_z \subseteq \{v_i, v_{i+1}, \ldots, v_n\}$, where $v_j \in L_z$ if $v_j \geq v_i$ is available for inclusion in an independent set, given some root-to-z path.*

Note that the eligibility set at z is not necessarily unique. Depending on the path taken from the root to z, some vertices could be undominated at z, but not usable in any valid path from z. Such a vertex is called an **eligible impostor**. For example, suppose that there exists a vertex $v \in L_z$ such that $N(v) = L_z \setminus v$. Additionally, suppose that at z, some vertex $u \in T_z$ with $u \not\leftrightarrow v$ has not been dominated. Then, any path from z that uses v cannot dominate u, and thus cannot be a valid path. In this case, v is an eligible impostor. The reduced eligibility set at z removes all such impostors.

Definition 4. *The* **reduced eligibility set** *at z, denoted $\overline{L}_z$, is the set of vertices which are used by at least one valid path from z. Equivalently, $\overline{L}_z = L_z \setminus I$, where I is the set of eligible impostors with respect to some path from the root of $Z_{\mathcal{S}}$ to z.*

The following theorem gives a necessary and sufficient condition for when two nodes of a maximal independent set ZDD are equivalent and can be merged:

Theorem 1. *Nodes $z, z' \in V(Z_{\mathcal{S}})$ are equivalent if and only if $T_z = T_{z'}$ and $\overline{L}_z = \overline{L}_{z'}$.*

Proof. ($\Rightarrow$) Suppose $z' \cong z$; the requirement that $T_z = T_{z'}$ is established by Lemma 1. Furthermore, every valid path from z must exactly correspond to a valid path from z', which means that every vertex used on a valid path from z must be eligible at z', and vice versa. The only eligible vertices present at z that do not need to be present at z' are those vertices which are used in no valid paths from z—that is, the eligible impostors. Thus, it must be the case that $\overline{L}_z = \overline{L}_{z'}$.

($\Leftarrow$) Suppose $T_z = T_{z'}$ and $\overline{L}_z = \overline{L}_{z'}$ for $z, z' \in Z_{\mathcal{S}}$. It suffices to show that valid paths from z are in bijection with valid paths from z'. Note that in any root-to-z path, the undominated vertices at z are a subset of T_z, and similarly for z'. Therefore, since $T_z = T_{z'}$, any valid path from z (z') to $\mathbf{1}$ must dominate all undominated vertices at z (z'). Additionally, since $\overline{L}_z = \overline{L}_{z'}$, all valid paths from z are also valid paths from z', and vice versa. Thus, there is a bijection between valid paths from z and valid paths from z', proving the result. ∎

Theorem 1 provides a method for building $Z_{\mathcal{S}}$ for a particular graph by computing T_z and $\overline{L}_z$ for every node in the graph. Pseudo-code for algorithms to compute T_z and $\overline{L}_z$ at a node z is

Algorithm 2: ComputeTotalDomination(U, i)

input: A set of undominated vertices $U = \{u_1, u_2, \ldots, u_k\}$, and an index i
output: The totally dominated set given U and i

1 $\langle\!\langle$ *The only nodes that could be in* T_z *are the neighbors of eligible vertices* $\rangle\!\rangle$
2 $L_z = \{u_i, u_{i+1}, \ldots, u_k\}$
3 $T_z = \left(\bigcup_i^k N(u_i)\right) \setminus L_z$
4 **for each** $v \in T_z$ **:**
5 $F = L_z \setminus N(v)$
6 Search for an independent set with vertices from F dominating all of L_z
7 **if** such a set exists**:** remove v from T_z
8 return T_z

given in Algorithms 2 and 3, respectively. The first algorithm is motivated by the observation that if a vertex v is totally dominated at z, no maximal independent set can be found using vertices in $L_z \setminus N(v)$. Thus, Algorithm 2 searches for such an independent set for each v, and removes v from T_z if one can be found. The second algorithm operates similarly; it takes as input a list of undominated vertices at z, and checks each of them to see if they are contained in some maximal independent set from z. If no such set is found, the vertex is marked as an eligible impostor. Both of these algorithms use depth-first search to find independent sets.

Note that a few enhancements to Algorithm 2 can be made; namely, all of $\{u_1, u_2, \ldots, u_{i-1}\}$ must be dominated in a valid path from z, and thus, do not have to be checked in line 4. Additionally, if the reduced eligibility set $\overline{L}_z$ is known at the start, it can be used in place of L_z. Finally, note that multiple vertices can be checked at once—in particular, if an independent set is found in one iteration of the loop that leaves multiple vertices from T_z undominated, all of them can be removed from T_z and not considered further. The worst-case running times for Algorithms 2 and 3 are $O(k2^k)$, but these bounds are often quite weak.

The maximal independent set ZDD can now be constructed. Whenever a node z is inserted into $Z_{\mathcal{S}}$, store T_z and $\overline{L}_z$ for that node. Then, when a new node is being considered for insertion, compute the totally dominated set and the reduced eligibility set for the node, and compare it to all nodes at the current level in the ZDD. If one is found that satisfies the equivalence conditions, merge the new node into the current node. Algorithm 4 gives a top-down construction routine for $Z_{\mathcal{S}}$; in this algorithm, K_i is the collection of nodes z for which $\mathrm{var}(z) = i$, and U_z is the set of undominated vertices at a node z.

Algorithm 3: ComputeReducedEligibility(U, i)

input: A set of undominated vertices $U = \{u_1, u_2, \ldots, u_k\}$, and an index i
output: The reduced eligibility set given U and i

$L_z = \{u_i, u_{i+1}, \ldots, u_k\}$
$\overline{L}_z = \varnothing$
for each $u_j \in \{u_i, u_{i+1}, \ldots, u_k\}$**:**
    $F = L_z \setminus N[u_j]$
    Search for an independent set dominating U using u_j and vertices from F
    if such a set exists**:** add u_j to $\overline{L}_z$
return $\overline{L}_z$

To store the totally dominated sets and reduced eligibility sets, note that for any node z, all $v_i \in T_z$ have $i < \text{var}(z)$ and all $v_i \in \overline{L}_z$ satisfy $i \geq \text{var}(z)$. Therefore, T_z and $\overline{L}_z$ can be stored in a single boolean array, where all true entries in the array with index less than $\text{var}(z)$ are members of T_z, and similarly for $\overline{L}_z$.

Algorithm 4: BuildZDD$(G = (V, E))$

output: The root node of $Z_{\mathcal{S}}$

Insert the root of $Z_{\mathcal{S}}$ into K_1, with $U_{\text{root}} = V$
for each $i \in \{1, 2, \ldots, n\}$**:**
    for each $z \in K_i$**:**
        $U_H = U_z \setminus N[v_i]$
        $h = \min\{j \mid j > i \text{ and } v_j \in U_H\}$
        $\overline{L}_{z'} =$ `ComputeReducedEligibility`(U_H, h)
        $T_{z'} =$ `ComputeTotalDomination`(U_H, h)
        if $\exists$ node in K_h with $T_z = T_{z'}$ and $\overline{L}_z = \overline{L}_{z'}$**:** merge z and z'
        else: insert z' into K_h
        $U_L = U$
        $l = \min\{j \mid j > i \text{ and } v_j \in U_L\}$
        $\overline{L}_{z'} =$ `ComputeReducedEligibility`(U_L, l)
        $T_{z'} =$ `ComputeTotalDomination`(U_L, l)
        if $\exists$ node in K_l with $T_z = T_{z'}$ and $\overline{L}_z = \overline{L}_{z'}$**:** merge z and z'
        else: insert z' into K_l
return the root of $Z_{\mathcal{S}}$

Algorithm 4 is a top-down construction, in the sense that a node is inserted into $Z_{\mathcal{S}}$ before either of its children. A bottom-up construction can also be used, in which both the high and low child of a node are inserted into the ZDD before the parent is inserted. The bottom-up construction is a natural modification of Algorithm 1, where a check is inserted after line 2 to see if the current node can be merged with any other node at the current level.

Note that for such a construction, the totally dominated set and reduced eligibility set at a node can be computed recursively and stored with each node in the ZDD, as follows:

$$T_z = \big((N(z) \cap \{1, \ldots, v_{i-1}\}) \cup T_{\mathrm{hi}(z)}\big) \cap T_{\mathrm{lo}(z)},$$

where $T_{\mathbf{0}} = \{1, 2, \ldots, v_n\}$ and $T_{\mathbf{1}} = \varnothing$. Additionally, $\overline{L}_z$ can be computed by

$$\overline{L}_z = \{v_{\mathrm{var}(z)}\} \cup \overline{L}_{\mathrm{hi}(z)} \cup \overline{L}_{\mathrm{lo}(z)}$$

where $\overline{L}_{\mathbf{1}} = \overline{L}_{\mathbf{0}} = \varnothing$. The first condition follows since any totally dominated vertex at z must be dominated by all paths from $\mathrm{hi}(z)$ and from $\mathrm{lo}(z)$; the second follows because a vertex at z is in $\overline{L}_z$ if it is used at z, or at $\mathrm{hi}(z)$, or at $\mathrm{lo}(z)$. Computing these sets in this manner requires constant time. The bottom-up variant of Algorithm 4 using these recursive conditions is called the **merging algorithm**.

4 Vertex Ordering Heuristics for $Z_{\mathcal{S}}$

For a fixed variable ordering, there exists a smallest reduced ZDD characterizing any family of sets; however, different variable orderings can lead to drastically different data structure sizes, and in fact it is NP-hard to determine the variable ordering that leads to the smallest possible ZDD (Bollig and Wegener, 1996). While this result does not necessarily imply that finding the optimal variable ordering is NP-hard for the maximal independent set ZDD, the authors conjecture this to be the case. Therefore, in this section, several different heuristic ordering rules are explored for the maximal independent set ZDD.

- **Random Order**: In this ordering, the variables are randomly permuted and used to construct the ZDD. Several different permutations can be tried, and the one yielding the smallest ZDD can be used.
- **Degree Ordering**: This rule orders the vertices of G by the increasing or decreasing degree sequence of G.
- **Degeneracy Ordering**: This rule orders the vertices of G by increasing or decreasing **in-**

duced degree sequence. In other words, the i^{th} vertex in the ordering v_i has the highest or lowest degree in $G[V \setminus \{v_1, v_2, \ldots, v_{i-1}\}]$.

- **Clique Cover Ordering**: This rule computes a covering of G by maximal cliques; the cliques are sorted by size, and the vertices within each clique are ordered arbitrarily.

- **Maximal Path Decomposition Ordering**: This rule computes a set of paths $P_1, P_2, \ldots, P_q$ such that P_i is maximal in $G[V \setminus \bigcup_{j=1}^{i-1} P_j]$. The vertices are then ordered as

$$v_1^1, v_2^1, \ldots, v_{l_1}^1, v_1^2, v_2^2, \ldots, v_{l_2}^2, v_1^q, v_2^q, \ldots, v_{l_q}^q,$$

  where v_i^j is the i^{th} vertex along the path P_j, and l_j is the length of path P_j.

Eppstein and Strash (2011) use the degeneracy ordering in their algorithm for finding maximal cliques. Moreover, Bergman et al. (2012) describe the maximal path decomposition rule for the independent set BDD, and prove that the width of the i^{th} level of a BDD using this ordering is bounded by the $(i+1)^{st}$ Fibonacci number F_{i+1}. The proof of this result does not directly translate to the maximal independent set case; however, a slightly tighter bound can be proven in this setting:

Theorem 2. *The width of the i^{th} level of $Z_{\mathcal{S}}$ is bounded by F_i when the vertices of G are ordered according to a maximal path decomposition.*

Proof. Let $P_1, P_2, \ldots, P_q$ be a maximal path decomposition of G; without loss of generality, assume that G is connected with at least 3 vertices, so $|P_1| \geq 3$. Let $v_1, v_2, \ldots, v_n$ be the vertices of G in the order induced by the path decomposition, and let L_i be the level corresponding to vertex v_i. Finally, let w_i be the number of nodes in level L_i. To prove that $w_i \leq F_i$, induction on i is used.

Note that L_1 has one node, the root. L_2 also contains one node since $v_1 \leftrightarrow v_2$. If $v_1 \not\leftrightarrow v_3$ and $v_2 \not\leftrightarrow v_3$, L_3 contains at most 3 nodes (if v_1 or v_2 must be in every maximal independent set of G, then $w_3 < 3$). In general, fix $i \in \{1, 2, \ldots, n\}$, and assume that for any connected graph G with at least 3 vertices, and any ordering of $V(G)$ induced by a maximal path decomposition, $w_j \leq F_j$ for all $j < i$. It suffices to show that $w_i \leq F_i$.

To see that this is the case, consider two cases: first, v_i is not the first vertex in some path in the decomposition. In this case, $v_{i-1} \leftrightarrow v_i$, so the only edges that can exist between L_{i-1} and L_i

are low edges. Thus, there are at most w_{i-1} nodes in L_i with parents in L_{i-1}. Additionally, there can be at most w_{i-2} high edges from L_{i-2} to L_i. If there are no edges from L_k to L_i for $k < i-2$, and there are no low edges from L_{i-2}, the result follows from the inductive hypothesis. On the other hand, suppose such an edge exists; call such an edge a **bad** edge. Since the bad edge skips L_{i-1}, there must exist $k < i-1$ such that $v_k \leftrightarrow v_{i-1}$. Delete all such edges in G; this increases the size of L_{i-1} by at least one, so by the inductive hypothesis, $w_{i-1} \leq F_{i-1} - 1$ for the unmodified graph. By the same reasoning, if there are k bad edges, $w_{i-1} \leq F_{i-1} - k$. Therefore, the total number of nodes at level i is $w_i \leq F_{i-1} - k + F_{i-2} + k = F_i$, as desired.

In the second case, v_i is the first vertex along some path in the decomposition. In this case, all low edges from nodes in L_{i-1} go to $\mathbf{0}$, because v_{i-1} has no neighbor in $\{v_i, v_{i+1}, \ldots, v_n\}$. A similar argument as in the first case then follows to show that $L_i \leq F_i$. ■

Note that Theorem 2 is entirely a structural result—it makes no use of the equivalence conditions presented in Section 3. Therefore, there is the opportunity for this result to be significantly tighter, if many nodes in the ZDD are equivalent and can be merged. Furthermore, Theorem 2 implies that when using the maximal path decomposition ordering, the worst-case running time of Algorithm 1 is improved to $O(n\varphi^n)$, where $\varphi = (1+\sqrt{5})/2$, since $\sum_{i=1}^{n} F_i = F_{n+2} - 1$, and F_i is bounded by $O(\varphi^n)$ (Weisstein, 2013).

5 Computational Results

Both the recursive construction algorithm (Algorithm 1) and the merging algorithm variant were implemented in C++, and run on an Intel Core i7-930 2.8GHz quad-core processor with 12 GB of available memory. However, these algorithms utilized only a single processor core. For the sake of comparison, the *dfmax* benchmark program was run on the r500.5 instance provided by Trick (2005). The systems used for these experiments took 6.60s user time to solve this benchmark instance. All results discussed in this section are given in CPU seconds, and have a time limit of two hours. Notation used in this section is given in Table 1.

Table 2 provides a comparison between the recursive construction algorithm and the merging algorithm on a subset of the problems from the DIMACS graph coloring database (Johnson and Trick, 1996; Trick, 2005). The recursive construction algorithm was given a limit of 100 000 000

Instance	The graph instance under consideration
n	The number of vertices in the instance
m	The number of edges in the instance
α	The independence number of the instance, as computed by $Z_{\mathcal{S}}$
$\lvert\mathcal{S}\rvert$	The total number of maximal independent sets in the instance
t	The time needed to construct $Z_{\mathcal{S}}$
$\lvert Z_{\mathcal{S}}\rvert$	The number of nodes in $Z_{\mathcal{S}}$
μ	The average size of $Z_{\mathcal{S}}$ for the random vertex ordering
σ	The standard deviation in the size of $Z_{\mathcal{S}}$ for the random vertex ordering
μ gap	The gap between the average randomly-ordered ZDD size and the smallest-found size
min gap	The gap between the minimum randomly-ordered ZDD size and the smallest-found size
max gap	The gap between the maximum randomly-ordered ZDD size and the smallest-found size

Table 1: Notation used in Tables 2-4.

ZDD nodes; the merging algorithm had a node limit that changed with the size of the graph (since a larger graph requires a larger totally dominated and reduced eligibility set). Each of the instances in Table 2 use the maximal path decomposition ordering for the vertices. When $Z_{\mathcal{S}}$ could be constructed for an instance, the independence number for that instance was computed by finding the longest root-to-**1** path in $Z_{\mathcal{S}}$; this value is reported in column 4. Instances for which the algorithms were unable to construct $Z_{\mathcal{S}}$ in the two-hour time limit report the number of nodes inserted at termination in columns 6 and 8.

There are 50 instances in Table 2 for which $Z_{\mathcal{S}}$ could be constructed (or the node limits hit) within the time limit by one of the algorithms. Of these 50, the recursive algorithm significantly outperformed the merging algorithm in 28 cases (these instances are highlighted in grey in column 7). Conversely, the merging algorithm outperformed the recursive algorithm in 17 cases (highlighted in column 9). Of the remaining 7 instances which failed to complete within the time limit by either algorithm, the recursive algorithm was able to construct a larger partial ZDD in 4 cases (highlighted in column 6), and the merging algorithm was able to construct a larger partial ZDD in 3 cases (highlighted in column 8).

This difference in performance can be explained by the fact that for some instances, the recursive algorithm needs to make many identical recursive calls to determine node equivalence in the ZDD. In these cases, memoizing the totally dominated set and reduced eligibility set allows the algorithm to short-cut these recursive calls. However, because computing T_z and $\overline{L}_z$ is itself computationally challenging, the recursive algorithm will outperform the merging algorithm for instances which do

Table 2: A comparison of the recursive construction algorithm and the merging algorithm for a subset of the DIMACS graph coloring database. Grey cells indicate the faster algorithm, or the algorithm that generated more nodes if both took more than 2 hours.

					Recursive alg.		Merging alg.	
Instance	n	m	α	$\lvert\mathcal{S}\rvert$	$\lvert Z_{\mathcal{S}}\rvert$	t	$\lvert Z_{\mathcal{S}}\rvert$	t
DSJC125.5	125	3891	10	43268	48328	0.61	48328	14.55
DSJC125.9	125	6961	4	524	623	0.01	623	0.02
DSJC250.5	250	15668	12	1470363	1476916	34.50	>1153590	>2hrs
DSJC250.9	250	27897	5	2580	2893	0.04	2893	0.36
DSJC500.5	500	62624	13	91664597	83467418	3818.16	>1057125	>2hrs
DSJC500.9	500	112437	5	14560	15397	0.39	15397	7.03
DSJC1000.5	1000	249826	?	?	$>10^8$	6937.39	>716976	>2hrs
DSJC1000.9	1000	449449	6	100389	102909	5.77	102909	184.60
DSJR500.1	500	3555	?	?	>72383	>2hrs	>592268	>2hrs
DSJR500.1c	500	121275	13	643	2443	0.15	2443	0.46
DSJR500.5	500	58862	7	38551855	1809872	711.83	>1057125	>2hrs
queen8_8	64	728	8	10188	9951	0.08	9951	1.35
queen8_12	96	1368	8	334806	221524	3.42	221524	476.55
queen9_9	81	1056	9	57600	50746	0.60	50746	32.44
queen10_10	100	2940	10	376692	295493	5.27	295493	1078.65
queen11_11	121	3960	11	2640422	1870782	42.95	>778484	>2hrs
queen12_12	144	5192	12	19469324	12443637	368.98	>777125	>2hrs
queen13_13	169	6656	13	151978440	88885235	3351.61	>709664	>2hrs
queen14_14	196	8372	?	?	$>10^8$	3795.47	>673151	>2hrs
queen15_15	225	10360	?	?	$>10^8$	3740.66	>664525	>2hrs
queen16_16	256	12640	?	?	$>10^8$	3972.67	>564070	>2hrs
myciel3	11	23	5	16	29	0.00	29	0.00
myciel4	20	71	11	79	152	0.00	152	0.00
myciel5	47	236	23	857	1429	0.00	1429	0.04
myciel6	95	755	47	49049	40191	0.99	40191	18.96
myciel7	191	2360	95	75511755	7191878	2635.97	>317430	>2hrs
1-Insertions_4	67	232	32	56641	85122	0.93	85122	155.42
3-Insertions_3	56	110	27	228439	81050	2.46	81050	146.24
4-Insertions_3	79	156	39	37833929	5518516	430.29	>522455	>2hrs
1-FullIns_4	93	593	45	129042	137761	8.78	137761	881.28
2-FullIns_3	52	201	25	15966	7975	0.25	7975	3.26
3-FullIns_3	80	346	37	1454750	363408	55.76	>251538	>2hrs
4-FullIns_3	114	541	?	?	>4527300	>2hrs	>250095	>2hrs
5-FullIns_3	154	792	?	?	>6042790	>2hrs	>212485	>2hrs
fpsol2.i.1	496	11654	307	1.67×10^{14}	>521	>2hrs	3969	1.59
fpsol2.i.2	451	8691	261	8.49×10^{18}	>174	>2hrs	16923	8.51
fpsol2.i.3	425	8688	238	7.43×10^{18}	>375	>2hrs	17405	8.73
latin_square_10	900	307350	10	30240	52742	35.93	52742	134.38
school1	385	19	?	?	>608862	>2hrs	>93887	>2hrs
school1_nsh	352	14612	?	?	>2148287	>2hrs	>473049	>2hrs
mulsol.i.1	197	3925	100	98404	644	1.37	664	0.04
mulsol.i.2	188	3885	90	2669597327	>1650	>2hrs	2021	0.15
mulsol.i.3	184	3916	86	2669597327	>1663	>2hrs	2029	0.16
mulsol.i.4	185	3946	86	4650922127	>1665	>2hrs	2033	0.15
mulsol.i.5	185	3973	88	3330038927	>1828	>2hrs	2196	0.17
miles250	128	774	?	?	>275	>2hrs	>7552	>2hrs
miles500	128	2340	?	?	>53123	>2hrs	>85851	>2hrs
miles750	128	4226	12	33208742	6112	111.80	6112	1.39
miles1000	128	6342	8	775281	8520	4.64	8520	1.80
miles1500	128	10396	5	7802	1695	0.06	1695	0.07
anna	138	986	?	?	>9316	>2hrs	23296	4.52
david	87	812	36	44149508	6901	174.35	6901	2.53
jean	80	508	38	1251960	1360	2.33	1360	0.06
huck	74	602	27	7272300	283	15.25	27	0.01
zeroin.i.1	211	4100	120	79170	731	0.66	731	0.05
zeroin.i.2	211	3541	127	18189098	1114	138.59	1114	0.11
zeroin.i.3	206	3540	123	12912650	1112	111.29	1112	0.12

not require many identical recursive calls.

Additionally, for each instance the ZDD was used to compute the total number of maximal independent sets present in the graph, with the results shown in column 5 of Table 2. In most cases, the number of maximal independent sets is close to the size of $Z_{\mathcal{S}}$, though some notable exceptions exist (particularly `fpsol2.i.1`, `fpsol2.i.2`, and `fpsol2.i.3`). This value can be used as a surrogate to see how many nodes are merged in the ZDD—ZDDs that are small relative to the total number of maximal independent sets are able to perform many merges, whereas ZDDs that are larger than the total number of independent sets do not have as many similar paths that can be merged together.

Next, a study of the different vertex orderings discussed in Section 4 was performed, with the results presented in Table 3. For these results, only the recursive algorithm was used, since it outperformed the merging algorithm in a majority of cases. Instances for which the recursive algorithm in Table 2 took more than two hours of computation time were not tested. For each instance, a clique cover was computed by iteratively applying a branch-and-bound algorithm to search for maximal cliques, until all vertices are contained in some clique. At each iteration, the branch-and-bound algorithm terminated after exploring $2\Delta(G)$ states, and the best clique found was returned.

Of the 38 instances shown in Table 3 which were below the node limit, the maximal path decomposition ordering produced the minimally-sized ZDD in 17 cases. The clique cover ordering produced the smallest ZDD in 11 instances, and the degree list ordering and the reverse degeneracy ordering gave the smallest ZDD for 5 instances each. This difference in size appears to be related to the ease of finding large cliques in G; for instance, in the `DSJ` graphs and the `queen` graphs, large cliques can be easily found. In these cases, the clique cover ordering generally performs best. However, in instances that do not contain large cliques (such as the triangle-free `myciel` graphs), the maximal path ordering performs better. The reverse degree ordering and the forward degeneracy ordering did not produce the minimally-sized ZDD for any instance, and thus are not shown in Table 3.

Additionally, note that the vertex ordering affects the time to complete construction, as well as the size of the final ZDD. For example, note that for `3-FullIns_3`, despite the fact that the maximal path decomposition ordering yields a ZDD that is less than half the size of any other ordering, the

Table 3: A comparison of the maximal independent set ZDD size with the vertex orderings from Section 4. Grey cells indicate the smallest ordering.

	Degree List		Rev. Induced Degree List		Clique Cover		Maximal Path Decomp.		Random	
Instance	$\|Z_S\|$	t	$\|Z_S\|$	t	$\|Z_S\|$	t	$\|Z_S\|$	t	$\|Z_S\|$	t
DSJC125.5	47183	0.50	44809	0.51	45559	0.55	48328	0.61	46445	0.55
DSJC125.9	627	0.00	622	0.01	607	0.00	623	0.01	627	0.01
DSJC250.5	1416093	27.69	1419868	28.89	1410240	30.40	1476916	34.50	1447760	30.53
DSJC250.9	2880	0.04	2852	0.04	2866	0.07	2893	0.04	2913	0.04
DSJC500.5	79745416	3054.26	80620093	3290.64	80307678	3378.64	83467418	3818.16	81549931	3310.20
DSJC500.9	15252	0.39	15271	0.37	15279	0.54	15397	0.39	15329	0.39
DSJC1000.5	$>10^8$	6371.72	78359919	>2hrs	$>10^8$	6652.68	$>10^8$	6937.39	87381935	>2hrs
DSJC1000.9	102060	5.34	102584	5.12	102368	6.67	102909	5.77	102451	5.96
DSJR500.1c	2437	0.15	2085	0.13	2012	0.24	2443	0.15	2138	0.14
DSJR500.5	612313	313.21	1832145	308.96	1188614	373.67	1809872	711.83	2169788	380.22
queen8_8	9611	0.08	9689	0.09	9341	0.09	9951	0.08	11056	0.09
queen8_12	209878	3.35	220198	3.06	169104	3.31	221524	3.42	253668	2.97
queen9_9	47591	0.55	49159	0.56	46899	0.60	50746	0.60	61160	0.54
queen10_10	280640	4.35	281622	4.35	272460	4.76	295493	5.27	345768	4.34
queen11_11	1715945	44.24	1728852	35.72	1678570	38.29	1870782	42.95	2223191	35.18
queen12_12	11474606	322.40	11488955	310.23	11097792	379.80	12443637	368.98	15285013	290.45
queen13_13	80618162	2791.45	79625893	3093.27	77364686	3228.99	88885235	3351.61	$>10^8$	2277.94
queen14_14	$>10^8$	3412.03	$>10^8$	4122.76	$>10^8$	4280.20	$>10^8$	3795.47	$>10^8$	2346.78
queen15_15	$>10^8$	3355.18	$>10^8$	4795.93	$>10^8$	4176.67	$>10^8$	3740.66	$>10^8$	2530.21
queen16_16	$>10^8$	3817.44	$>10^8$	5416.81	$>10^8$	4913.85	$>10^8$	3972.67	$>10^8$	2712.34
myciel3	38	0.00	30	0.00	33	0.00	29	0.00	35	0.00
myciel4	230	0.00	176	0.00	210	0.00	152	0.00	202	0.00
myciel5	3435	0.02	1734	0.03	2948	0.02	1429	0.00	3479	0.03
myciel6	221879	4.33	97346	4.38	193969	4.15	40191	0.99	177746	10.68
myciel7	$>10^8$	5461.37	2297922	>2hrs	$>10^8$	6644.03	7191878	2635.97	19031053	>2hrs
1-Insertions_4	227317	2.04	58411	2.02	85456	1.61	85122	0.93	181447	4.38
3-Insertions_3	592429	5.42	32975	3.52	52489	2.70	81050	2.46	260534	4.84
4-Insertions_3	43549657	1322.11	586435	665.40	773005	544.35	5518516	430.29	22883116	1606.48
1-FullIns_4	294052	5.80	142075	7.67	304315	8.96	137761	8.78	402067	19.97
2-FullIns_3	11759	0.18	11161	0.27	20161	0.27	7975	0.25	20473	0.37
3-FullIns_3	814178	28.22	739391	49.65	1206277	36.32	363408	55.76	1420292	61.18
latin_square_10	93203	27.72	110236	47.06	113209	34.22	52742	35.93	162995	32.92
mulsol.i.1	1012	18.40	804	17.68	1447	21.63	644	1.37	63963	15.20
miles750	22994	146.39	21626	172.69	19078	160.43	6112	111.80	463524	160.68
miles1000	4435	3.09	9618	3.28	4367	3.20	8520	4.64	62315	3.12
miles1500	717	0.04	1584	0.03	991	0.03	1695	0.06	2402	0.05
david	33493	1279.88	23687	1753.32	11057	1252.25	6901	174.35	608664	727.55
jean	6534	46.41	6471	75.11	3760	50.32	1360	2.33	80955	29.73
huck	4553	69.90	2478	182.87	2576	78.13	283	15.25	56630	63.37
zeroin.i.1	1129	16.68	830	21.84	2504	21.02	731	0.66	116632	17.09
zeroin.i.2	87372	3556.30	3297	4646.50	68695	4178.08	1114	138.59	6921096	6342.30
zeroin.i.3	79356	2323.45	3227	3339.63	58670	2858.85	1112	111.29	2110951	2245.66

length of time needed to compute $Z_{\mathcal{S}}$ is approximately twice the length of time needed for the clique cover ordering or the degree list ordering. This again demonstrates the effect the vertex ordering can have on the number of recursive calls made by the construction algorithm.

Instance	μ	σ	μ gap	min gap	max gap
DSJC125.5	47094	575	3.37	1.94	6.09
DSJC125.9	624	5	2.78	1.48	3.95
DSJC250.5	1449883	9298	2.81	1.74	3.62
DSJC250.9	2888	14	0.76	0.24	1.64
DSJC500.9	15303	27	0.34	0.02	0.60
DSJC1000.9	102603	68	0.53	0.38	0.62
DSJR500.1c	2145	27	6.62	4.08	8.40
DSJR500.5	2269446	147585	270.63	223.79	310.60
queen8_8	11418	231	22.23	18.36	26.91
queen8_12	260516	7242	54.06	47.73	59.24
queen9_9	60275	1190	28.52	24.39	32.53
queen10_10	353986	7440	29.92	25.68	34.62
queen11_11	2258415	41048	34.54	30.53	38.98
queen12_12	15293305	220211	37.80	34.74	41.20
myciel3	33	2	14.48	6.90	24.14
myciel4	205	17	34.74	17.76	53.29
myciel5	2944	294	106.03	79.85	143.46
myciel6	173144	15087	330.80	280.01	415.97
1-Insertions_4	163411	19369	91.97	62.74	129.30
3-Insertions_3	248769	26960	373.94	283.35	465.47
4-Insertions_3	21705543	3072441	2707.94	2219.95	3231.84
1-FullIns_4	417028	22898	202.72	184.45	234.41
2-FullIns_3	20813	3202	160.98	106.71	217.55
3-FullIns_3	1743903	398986	379.87	230.60	606.19
latin_square_10	162470	562	208.05	205.76	209.17
mulsol.i.1	144013	69842	22262.20	7552.95	43331.68
miles750	619798	89983	10040.67	7483.84	12659.78
miles1000	65365	7235	1396.80	1104.88	1646.76
miles1500	2454	155	242.30	211.99	278.52
david	1032703	562188	14864.55	4002.83	30099.94
jean	79025	27774	5710.69	2953.53	10109.56
huck	184398	151934	65058.37	10557.24	187694.35
zeroin.i.1	67696	34137	9160.77	3588.24	15855.13

Table 4: A comparison of random vertex orderings to the best orderings found.

Finally, to see how the size of $Z_{\mathcal{S}}$ varies under different random orderings, 10 trials were run for each of the instances in Table 3 for which the random ordering ZDD could be computed in under 30 minutes. The results from these trials are shown in Table 4; additionally, the gap between the average size, minimum size, and maximum size of the random ordering was computed for each instance (columns 4-6). This gap is computed as (rand − best)/best. For no instances did the random vertex ordering find a smaller ordering than the best ordering found; furthermore, the gap between the best ordering and the best random ordering is quite large in most cases. The results from these experiments show that in general, a random ordering is not an effective vertex ordering for $Z_{\mathcal{S}}$.

6 Conclusion

In this paper, characteristics of the maximal independent set ZDD for undirected graphs are studied. A necessary and sufficient condition for equivalence between two nodes in such a ZDD is proved, and various vertex orderings are studied to determine which orderings yield small ZDDs. A result is obtained showing that the maximal path decomposition ordering yields a ZDD whose width is bounded by the Fibonacci numbers. Finally, computational results are presented for the graphs in the DIMACS graph coloring challenge.

A number of open questions remain. First, since the bound on the width of the ZDD relies only on structural properties of the ZDD and assumes that no nodes are merged, it is likely that this bound can be tightened further. Additionally, it would be worthwhile to prove similar bounds for various other orderings described. Lastly, improving the performance of the merging algorithm may enable faster construction times overall. One approach may be to adapt the Bron and Kerbosch (1973) algorithm to produce a ZDD; however, this is challenging due to the non-binary branching factor of their algorithm.

7 Acknowledgments

The computational results reported were obtained at the Simulation and Optimization Laboratory at the University of Illinois, Urbana-Champaign. This research has been supported in part by the Air Force Office of Scientific Research (FA9550-10-1-0387), as well as fellowships through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program and National Science Foundation Graduate Research Fellowship Program. Additionally, the third author was supported in part by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government. The authors would like to thank two anonymous referees and the associate editor for their detailed review of our paper, which resulted in a significantly improved draft.

References

S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 100(6):509–516, 1978.

M. Behle. *Binary Decision Diagrams and Integer Programming.* PhD thesis, Universistät des Saarlands, 2007.

M. Behle and F. Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.

D. Bergman, A. A. Cire, W. V. Hoeve, and J. N. Hooker. Variable ordering for the application of BDDs to the maximum independent set problem. In N. Beldiceanu, N. Jussien, and . Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, number 7298 in Lecture Notes in Computer Science, pages 34–49. Springer Berlin Heidelberg, Jan 2012.

D. Bergman, A. A. Cire, W. V. Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, Articles in Advance, Aug 2013.

B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, Sep 1996.

C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, Sep 1973.

R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug 1986.

R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

O. Coudert. Solving graph optimization problems with ZBDDs. In *European Design and Test Conference*, pages 224–228, 1997.

D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In P. M. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, number 6630 in Lecture Notes in Computer Science, pages 364–375. Springer Berlin Heidelberg, Jan 2011.

T. Hadžić and J. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. Technical Report 167, Tepper School of Business, Apr 2008.

S. Held, W. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, Dec 2012.

D. S. Johnson and M. A. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*. American Mathematical Society, Jan 1996.

C. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *30th Conference on Design Automation*, pages 272–277, 1993.

S. Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer*, 3(2):156–170, May 2001.

D. R. Morrison, E. C. Sewell, and S. H. Jacobson. Solving the pricing problem in a generic branch-and-price algorithm using zero-suppressed binary decision diagrams. arXiv:1401.5820 [cs.DS], Jan 2014. URL `http://arxiv.org/abs/1401.5820`.

R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, Feb 2011.

E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, Oct 2006.

M. A. Trick. Computational series: Graph coloring and its generalizations, Mar 2005. URL `http://mat.gsia.cmu.edu/COLOR02/`.

E. W. Weisstein. Binet's fibonacci number formula– from wolfram MathWorld, 2013. URL `http://mathworld.wolfram.com/BinetsFibonacciNumberFormula.html`.