On the Isomorphism Problem for Decision Trees and Decision Lists*

V. Arvind[1], Johannes Köbler[2], Sebastian Kuhnert[2], Gaurav Rattan[1], and Yadu Vasudev[1]

[1] The Institute of Mathematical Sciences, Chennai, India
{arvind,grattan,yadu}@imsc.res.in
[2] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
{koebler,kuhnert}@informatik.hu-berlin.de

Abstract. We study the complexity of isomorphism testing for Boolean functions that are represented by decision trees or decision lists. Our results include a $2^{\sqrt{s}(\lg s)^{O(1)}}$ time algorithm for isomorphism testing of decision trees of size s. Additionally, we show:

- Isomorphism testing of rank-1 decision trees is complete for logspace.
- For $r \geq 2$, isomorphism testing for rank-r decision trees is polynomial-time equivalent to Graph Isomorphism. As a consequence we obtain a $2^{\sqrt{s}(\lg s)^{O(1)}}$ time algorithm for isomorphism testing of decision trees of size s.
- The isomorphism problem for decision lists admits a Schaefer-type dichotomy: depending on the class of base functions, the isomorphism problem is either in polynomial time, or equivalent to Graph Isomorphism, or coNP-hard.

1 Introduction

Two Boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$ are said to be *isomorphic* if there is a permutation π of the input variables $x_1, x_2, \ldots, x_n$ so that $f(x_1, x_2, \ldots, x_n)$ and $g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ are equivalent Boolean functions. The *Boolean function isomorphism problem* is to test if two given Boolean functions f and g are isomorphic. The complexity of this problem, when f and g are given as either Boolean circuits, formulas, or branching programs, has been studied before [AT96]. The isomorphism problem for Boolean circuits is in Σ_2^p and is coNP-hard even for DNF formulas. It is also known [AT96] that the problem is not hard for Σ_2^p unless the polynomial hierarchy collapses. Thierauf [Thi00] further studied isomorphism and equivalence for various models of Boolean functions. He has shown that the isomorphism problem for read-once branching programs is not NP-complete unless the polynomial hierarchy collapses to Σ_2^p. From an algorithmic perspective, Boolean Function Isomorphism can be solved in $2^{O(n)}$ time [Luk99] by reducing it to Hypergraph Isomorphism, and this is the best known algorithm in general. The

* This work was supported by Alexander von Humboldt Foundation in its research group linkage program. The third author was supported by DFG grant KO 1053/7-1.

best known algorithm for Graph Isomorphism, on the other hand, has running time $2^{O(\sqrt{n \lg n})}$ [BL83].

In this paper, our aim is to explore Boolean function representations for which the isomorphism problem has faster algorithms. We focus on the problem when the functions are given as decision trees and decision lists.

Definition 1.1. *A decision tree T_f on variables $X = \{x_1, \ldots, x_n\}$ is an ordered binary tree in which each leaf is labeled with a Boolean value and each inner node is labeled with a variable in X and has exactly two children. Any assignment $b_1, \ldots, b_n$ defines a path from the root of T_f to a leaf: At an inner node labeled with x_i, proceed to the left child if $b_i = 0$ and to the right child otherwise. The function value $T_f(b_1, \ldots, b_n)$ is the label of the leaf node reached along this path.*

Decision trees are a natural representation for Boolean functions and are fundamental to Boolean function complexity. The *size* of a decision tree is the number of its leaves.

The satisfiability and equivalence problems for decision trees have simple polynomial-time algorithms. Thus, the isomorphism problem for decision trees, denoted DT-Iso, is in NP.

Our main result is a $2^{\sqrt{s}(\lg s)^{O(1)}}$ time algorithm for isomorphism testing of size-s decision trees. We obtain this algorithm by examining the connection between bounded rank decision trees and hypergraphs of bounded rank. The rank of a hypergraph is the maximum hyperedge size in the hypergraph, and the rank of a decision tree T is the depth of the largest full binary tree that can be embedded in T. It turns out that rank-r decision trees can be encoded as hypergraphs of rank $O(r)$ and this transformation can be carried out in time $n^{O(r)}$. Since decision trees of size s have rank at most $\lg s$, this gives the $2^{\sqrt{s}(\lg s)^{O(1)}}$ time algorithm for isomorphism by applying the algorithm for bounded rank Hypergraph Isomorphism described in [BC08]. Further, it turns out that isomorphism of rank-1 decision trees is complete for deterministic logspace.

The next main topic of the paper is the isomorphism problem for decision lists, which were originally introduced by Rivest [Riv87] in learning theory.

Definition 1.2 ([Riv87]). *A $\mathcal{C}$-decision list ($\mathcal{C}$-DL) L, where $\mathcal{C}$ is a class of Boolean functions, is a sequence of the form $(f_i, b_i)_{i \leq m}$ where $f_i \in \mathcal{C}$, $b_i \in \{0, 1\}$ and $f_m = 1$. For a Boolean assignment x, the value computed by the decision list $L(x)$ is defined as b_i, where $i = \min\{j \geq 1 \mid f_j(x) = 1\}$.*

If $\mathcal{C}$ consists of single literals then $\mathcal{C}$-DL coincides with rank-1 decision trees. Similarly, if $\mathcal{C}$ consists of conjunctions of r literals then every r-CNF or r-DNF formula has a $\mathcal{C}$-decision list. We call such decision lists r-decision lists (or r-DLs, in short). For $r \geq 3$, the satisfiability problem for r-DLs is clearly NP-complete, and the equivalence problem is coNP-complete. Furthermore, every rank-r decision tree of size s has an r-decision list of size $O(s)$. Our results on isomorphism testing for decision lists are summarized below. We restrict our attention to classes $\mathcal{C}$ of Boolean functions depending only on at most k variables, for constant k.

1. If $\mathcal{C}$ consists of parities on 2 literals, isomorphism testing for $\mathcal{C}$-DLs is in polynomial time.
2. Isomorphism testing for $\mathcal{C}$-DLs is GI-complete,[3] when $\mathcal{C}$ is one of the following: (i) the conjunction of two literals, (ii) a conjunction of literals with at most *one negative literal*, (iii) a conjunction of literals with at most *one positive literal*, and (iv) parities on three or more literals.
3. In all other cases for $\mathcal{C}$, isomorphism testing for $\mathcal{C}$-DLs is coNP-hard.

The above results show a Schaefer-type dichotomy for the $\mathcal{C}$-DL isomorphism problem. It is interesting to compare with the dichotomy results for $\mathcal{C}$-CSP isomorphism obtained by Böhler et al. [BHRV04]. In their paper, the dichotomy exactly corresponds to Schaefer's original classification [Sch78]. In our results above, we have the Boolean complements of the Schaefer classes.

We observe that any $\mathcal{C}$-CSP $F = C_1 \wedge C_2 \wedge \ldots \wedge C_s$ is equivalent to the $\mathcal{C}$-DL given by $L = (\neg C_1, 0), \ldots, (\neg C_s, 0), (1, 1)$, proving the following lemma.

Lemma 1.3. *Let $\mathcal{C}$ be any class of Boolean functions closed under negation. Given a $\mathcal{C}$-CSP F of size s, there is a $\mathcal{C}$-decision list L of size $s + 1$ that is equivalent to F.*

We now recall the notion of rank for decision trees [EH89]. Let T be a decision tree and v be a node in T. If v is a leaf node then its rank is $\mathrm{rk}(v) = 0$. Otherwise, suppose v has children v_1 and v_2 in T. If $\mathrm{rk}(v_1) \neq \mathrm{rk}(v_2)$, define $\mathrm{rk}(v) = \max\{\mathrm{rk}(v_1), \mathrm{rk}(v_2)\}$; if $\mathrm{rk}(v_1) = \mathrm{rk}(v_2)$, define $\mathrm{rk}(v) = \mathrm{rk}(v_1) + 1$. The *rank* of the decision tree $\mathrm{rk}(T)$ is the rank of its root node. The rank $\mathrm{rk}(f)$ of a Boolean function f is the minimum rank over all the decision trees computing f.

In general, by a *representation* of a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ we mean a finite description R for f, such that for any input $x \in \{0,1\}^n$ we can evaluate $R(x) = f(x)$ in time polynomial in n. Examples of representations include circuits, branching programs, formulas, decision trees, decision lists etc.

Let π be a permutation of the input variables $x_1, x_2, \ldots, x_n$. Then f^π denotes the Boolean function $f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$. Similarly, for any representation R of the function f we denote by R^π the representation for f^π obtained by replacing each input variable x_i in R by $x_{\pi(i)}$.

Let $\mathcal{R}$ and $\mathcal{R}'$ be sets of representations of Boolean functions. A *permutation preserving normal form representation* (in short, *normal form*) for $\mathcal{R}$ is a mapping $N\colon \mathcal{R} \to \mathcal{R}'$ such that (i) for any $R \in \mathcal{R}$, N_R and R describe the same function, (ii) if R_1 and R_2 describe the same function then $N_{R_1} = N_{R_2}$, and (iii) for each permutation π we have $N_{R^\pi} = (N_R)^\pi$.

A *canonical form representation* for $\mathcal{R}$ is a mapping $C\colon \mathcal{R} \to \mathcal{R}'$ such that (i) for any $R \in \mathcal{R}$, the function represented by C_R is isomorphic to the one described by R, and (ii) for any two representations R_1 and R_2, the functions described by R_1 and by R_2 are isomorphic if and only if $C_{R_1} = C_{R_2}$.

Suppose f is a rank-r Boolean function, and f is given as a decision tree T_f which is not necessarily of rank r. There is a recursive $n^{O(r)}$ time algorithm for

[3] We say that a decision problem is GI-complete if it is polynomial-time equivalent to Graph Isomorphism.

computing a rank-r decision tree for f. This procedure is useful in isomorphism testing for bounded rank decision trees. As base case, suppose f is rank 1 and is given by T_f. Then there is a variable x such that $f|_{x\leftarrow 0}$ or $f|_{x\leftarrow 1}$ is constant, where $f|_{x\leftarrow b}$ denotes f with x set to b. This can be checked by setting $x = b$ in T_f and verifying that all leaves in the modified decision tree are labeled by the same constant. Suppose x_i is a variable such that $f|_{x_i\leftarrow 0}$ is the constant function 1, then the function $f|_{x_i\leftarrow 1}$ is of rank 1 and has only $n-1$ variables. Proceeding thus, in time polynomial in the size of T_f we can check if f is of rank 1 and also compute a rank-1 decision tree for it. For checking if f has rank r, we sketch a simple recursive procedure: Find a variable x_i such that $f|_{x\leftarrow b}$ is of rank at most $r-1$ (checked recursively), where $b \in \{0,1\}$. If $f|_{x\leftarrow \overline{b}}$ has rank at most r (checked recursively) then f is of rank r, else f has rank more than r. If no such variable exists then f is not rank r. The correctness and running time bounds are straightforward by induction arguments.

Theorem 1.4. *Given as input a decision tree T, we can check if the computed Boolean function has rank r and, if so, construct a rank-r decision tree for it in time $(n^r \cdot |T|)^{O(1)}$.*

2 GI-Hardness of DT-Iso and $\mathcal{C}$-DL-Iso

We will show that isomorphism testing even for rank-2 decision trees is GI-hard.

Let $G = (V, E)$ be a graph with $V = \{v_1, v_2 \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. We encode G as a Boolean function f_G on $n+m$ Boolean variables $v_1, \ldots, v_n$ and $e_1, \ldots, e_m$ as follows: $f_G(e_1, \ldots, e_m, v_1, \ldots, v_n) = 1$ if and only if exactly three variables e_i, v_j, v_k are 1, all remaining variables are 0, and $e_i = (v_j, v_k) \in E$. Here the Boolean variables v_i and e_j correspond, by abuse of notation, to elements of $V \cup E$. The proofs for the following simple observations are omitted for lack of space.

Lemma 2.1. *For any graph $G(V, E)$, the function f_G is of rank 2 and can be represented by a rank-2 decision tree of size $O(|G|^2)$.*

Theorem 2.2. *Let G and H be two graphs and let f_G and f_H be the functions as defined above. Then, $G \cong H$ if and only if $f_G \cong f_H$.*

Corollary 2.3. GI $\leq_p^m$ DT-Iso.

We now give a simple reduction from Graph Isomorphism to 2-DNF Isomorphism. Since $\mathcal{C}$-DLs, where $\mathcal{C}$ is the class of conjunctions of 2 literals or of k literals with at most 1 negative literal, contains 2-DNFs [Riv87], this will prove $\mathcal{C}$-DL-Iso is GI-hard for this choice of $\mathcal{C}$.

Given a graph $G(V, E)$, define the following functions over the variable set V: $\check{f}_G = \bigvee_{e=(u,v)\in E} u \wedge v$, $\hat{f}_G = \bigvee_{e=(u,v)\in E} \overline{u} \wedge \overline{v}$.

Lemma 2.4. *Let G, H be two graphs. Then $G \cong H$ if and only if $\check{f}_G \cong \check{f}_H$ if and only if $\hat{f}_G \cong \hat{f}_H$.*

Böhler et al. [BHRV04] have considered $\mathcal{C}$-CSP Isomorphism, where the constraints in $\mathcal{C}$ are all XORs of k literals for constant $k \geq 3$. They have shown that this problem is GI-hard. Combined with Lemma 1.3 and Lemma 2.4, this yields the following.

Proposition 2.5. GI $\leq_p^m$ *$\mathcal{C}$-DL-Iso, where $\mathcal{C}$ consists either of (i) conjunctions of two literals, (ii) conjunctions of k literals with at most one positive literal each, (iii) conjunctions of k literals with at most one negative literal each or (iv) XORs of $k \geq 3$ literals, for some constant k.*

In Section 4 we will show that these problems are GI-complete.

3 Isomorphism for Bounded Rank Decision Trees

We first show that the isomorphism problem for rank-1 Boolean functions is in polynomial time. In fact, we will give a polynomial-time algorithm for computing a canonical form representation for rank-1 Boolean functions. If the rank-1 function is given as a rank-1 decision tree, we show that the isomorphism problem is complete for deterministic logspace. Building on the rank-1 case, we give a polynomial-time reduction from the isomorphism problem for bounded rank Boolean functions to isomorphism of bounded rank hypergraphs. This yields a moderately exponential time algorithm for isomorphism testing of bounded rank decision trees.

Let f be a rank-1 Boolean function given by some decision tree T_f which is not necessarily rank 1.

Since $\mathrm{rk}(f) = 1$, for some variable x_i the function $f|_{x_i \leftarrow b}$ is a constant. Let $V_1(f)$ be the subset of variables $x_i \in \{x_1, x_2, \ldots, x_n\}$ such that $f|_{x_i \leftarrow 1}$ or $f|_{x_i \leftarrow 0}$ is a constant function. We define two subsets of $V_1(f)$: $V_{1,0}(f) = \{x_i \mid f|_{x_i \leftarrow 0} \text{ is constant}\}$ and $V_{1,1}(f) = \{x_i \mid f|_{x_i \leftarrow 1} \text{ is constant}\}$. The set $V_1(f)$ is computable in polynomial time from T_f: to check if $x_i \in V_1(f)$ we fix x_i to a constant in T_f and see if all leaves in the resulting decision tree have the same label.

In general, if T is a decision tree computing a function f, x_i is some variable and b is a Boolean constant, we can obtain the decision tree $T|_{x_i \leftarrow b}$ for the function $f|_{x_i \leftarrow b}$ by removing the subtree corresponding to the $\overline{b}$ path of any node labeled with the variable x_i. To check if this is the constant function, it is enough to verify if all the leaves of this modified decision tree have the same constant.

Next, we define $f_0 = f$ and $f_i = f_{i-1}|_{V_{1,0}(f_{i-1}) \leftarrow 1, V_{1,1}(f_{i-1}) \leftarrow 0}$. We also define the variable sets $V_i(f) = V_1(f_{i-1})$, which are again classified into $V_{i,0}(f) = V_{1,0}(f_{i-1})$ and $V_{i,1}(f) = V_{1,1}(f_{i-1})$. The variable set $\{x_1, x_2, \ldots, x_n\}$ is thus partitioned into $V_1(f), \ldots, V_k(f)$ for some $k \leq n$. The *level* of a variable x_i is the index j such that $x_i \in V_j(f)$.

The normal form for the rank-1 function f is defined as the sequence of pairs $\langle l_i, c_i \rangle_{1 \leq i \leq r}$ where l_i is a variable or its complement, and $c_i \in \{0, 1\}$. The pairs $\langle l_i, c_i \rangle$ in the sequence are ordered from left to right in increasing order of variable levels. Within each level they are in increasing order of variable

indices. For $x_i \in V_{j,0}(f)$, add $\langle \overline{x_i}, f_{j-1}|_{x_i \leftarrow 0} \rangle$ to the sequence; for $x_i \in V_{j,1}(f)$, add $\langle x_i, f_{j-1}|_{x_i \leftarrow 1} \rangle$ to the sequence.

Suppose f is a rank 1 function given by a decision tree T. Let the above sequence computed from T be N_T (which is actually a decision list for f). This sequence N_T defines a normal form representation for T. This follows from the next two lemmas.

Lemma 3.1. *Suppose f and g are two isomorphic rank-1 Boolean functions and π is an isomorphism from f to g. For any input variable x, if x is in level j for f then variable $\pi(x)$ is in level j for g.*

This implies that the number of variables in each level coincides for two isomorphic Boolean functions of rank 1. The next lemma is in the converse direction.

Lemma 3.2. *Let T_1 and T_2 be decision trees for two Boolean functions of rank 1, defined on the n variables $x_1, \ldots, x_n$. Let N_{T_1} and N_{T_2} be the corresponding normal form sequences obtained as in the discussion above. Suppose for each level i, $|V_{i,0}(f)| = |V_{i,0}(g)|$ and $|V_{i,1}(f)| = |V_{i,1}(g)|$, then T_1 and T_2 are isomorphic.*

Hence, the defined sequences are normal forms for rank-1 functions and in time polynomial in the size of the input decision tree we can compute the normal form. Given T_1 and T_2, Lemma 3.2 shows that by comparing the sizes of the sets $V_{i,0}$ and $V_{i,1}$ for the two functions we can check if the Boolean functions are isomorphic or not. This gives us the following theorem.

Theorem 3.3. *Given Boolean functions of rank 1 by decision trees, there is a polynomial time algorithm that checks if the functions are isomorphic.*

We now show that if the rank-1 function f is given as a decision tree T_f which is of rank 1, then the canonization problem is in logspace. For each internal node of T_f at least one of its children is a leaf (labeled by a constant). We can partition the internal nodes in the tree into subsets $L_1, \ldots, L_m$, where L_1 has consecutive nodes starting from the root that have a leaf child labelled with the same Boolean constant as at the root, L_2 is the next set of consecutive nodes with a leaf child labelled with the Boolean constant opposite to the one at the root, and so on. We further classify the variables in each L_i into the subset $L_{i,0}$ of nodes in L_i whose left child is a leaf and subset $L_{i,1}$ of nodes whose right child is a leaf. These sets can be computed in logspace by inspection of the input decision tree. Since for each $x \in L_{1,b}$, the restriction $f|_{x \leftarrow b}$ is a constant, $L_{1,b} \subseteq V_{1,b}(f)$. Also, notice that no variable outside the set $L_{1,b}$ has this property. Hence $L_{1,b} = V_{1,b}(f)$. Likewise, we can argue that $L_{i,b} = V_{i,b}(f)$ for all i and b. As a consequence we can state the following lemma.

Lemma 3.4. *Let T and T' be two rank-1 decision trees computing equivalent Boolean functions. Then $L_{i,b} = L'_{i,b}$ for all i and b.*

In order to obtain the canonical form for rank-1 decision trees, we first order the variables of L_i such that all the nodes whose left child is a constant come

first followed by the nodes whose right child is a constant. We do this for all the sets $L_1, \ldots, L_m$. Now starting from the root, rename the variables with the root node getting the variable x_1 followed by x_2 and so on. This new decision tree T_c, which is clearly computable in logspace, will be the canonical form for the original decision tree. By Lemma 3.4, a rank-1 decision tree isomorphic to the given tree is obtained by permuting the variables in L_i in some way for each i. Hence given two rank-1 decision trees computing isomorphic Boolean functions, the above procedure outputs the same rank-1 decision tree proving that it is a canonical form.

We now show the logspace completeness. We will give a reduction from the problem PathCenter, which is known to be complete for L [ADKK12]. The input to PathCenter is a directed path P of odd length and a vertex u. The problem is to test if u is the center of the path. We construct two decision trees T_1 and T_2 from P: For each $v \in V$ there is a variable x_v, and both T_1 and T_2 contain one internal node for each x_v. If v is the successor of v' in P, x_v becomes the right child of $x_{v'}$ in T_1, and $x_{v'}$ becomes the right child of x_v in T_2. The right child of x_v, where v is the vertex without successor (or without predecessor, respectively) is a leaf labeled with 1. In both trees, the left child of x_u is a leaf labeled with 1. The left children of all other variables are leaves labeled with 0.

Lemma 3.5. *Let T_1 and T_2 be the decision trees constructed from an instance $(P(V,E),u)$ of* PathCenter. *Let f_1 and f_2 be the functions computed by the decision trees, respectively. Then, $f_1 \cong f_2$ if and only if $(P(V,E),u) \in$* PathCenter.

We now consider rank-r Decision Tree Isomorphism. Using the normal form representation for rank-1 Boolean functions, we will obtain normal forms for Boolean functions of rank r. Similar to rank-1 decision trees, where the normal form consists of literal and constant pairs, for bounded rank functions the normal form will consist of pairs of literals and a normal form of a rank $r-1$ Boolean function. Let $V_1(f)$ be the subset of variables x_i such that $f|_{x_i \leftarrow 0}$ or $f|_{x_i \leftarrow 1}$ has rank at most $r-1$. Let $V_{1,b}(f) \subseteq V_1(f)$ consist of x_i such that $f|_{x_i \leftarrow b}$ has rank at most $r-1$ for $b \in \{0,1\}$. Further partition $V_{1,b}$ into subsets $V^{\ell}_{1,b}$ for $1 \leq \ell \leq r-1$ where $V^{\ell}_{1,b} = \{x_i \in V_1(f) \mid f|_{x_i \leftarrow b} \text{ has rank } \ell\}$.

For each $x_i \in V^{\ell}_{1,0}$ in increasing order of index i, include the pair $\langle \overline{x_i}, N_{x_i} \rangle$ in the sequence, where N_{x_i} is the normal form for $f|_{x_i \leftarrow 0}$, defined recursively. Similarly for $x_i \in V^{\ell}_{1,1}$ in the increasing order of the variable name, add the tuple $\langle x_i, N_{x_i} \rangle$ to the sequence. The above procedure is carried out for ℓ increasing from 1 to $r-1$. Now define, $f_1 = f|_{V_{1,0}(f) \leftarrow 1, V_{1,1}(f) \leftarrow 0}$, and continue constructing the normal form for f_1 as explained above. In general, we define the variable subsets $V^{\ell}_{i,b}(f) = V^{\ell}_{1,b}(f_{i-1})$. Since for each x_i, checking if $f|_{x_i \leftarrow b}$ is a rank-ℓ function and to compute it takes $\text{poly}(n^{r-1}|T|)$ time by Theorem 1.4, and since this process has to be repeated for at most n steps, the normal form can be constructed in time $\text{poly}(n^r|T|)$. The normal form consists of a sequence $\langle l_i, N_{x_i} \rangle_{i \leq m}$ where $m \leq n$ and $l_i \in \{x_i, \overline{x_i}\}$. We summarize the discussion in the following lemma (without proof).

Lemma 3.6. *Given a decision tree T computing a Boolean function f of rank r, a normal form representation N_T for the function can be computed in time* $\mathrm{poly}(n^r|T|)$.

We now describe our reduction of rank-r Decision Tree Isomorphism to bounded rank Hypergraph Isomorphism, where the rank of a hypergraph is the maximum size of any hyperedge in it.

Given a rank-r Boolean function as a decision tree, we first construct the normal form for f, N_f in time $n^{O(r)}$ as described earlier. The next step is to construct a vertex-colored hypergraph corresponding to the normal form. We will encode all the information in the normal form using hyperedges. The construction is inductive.

Rank-1 functions: The case of rank-1 functions is easy, since the normal form for a rank-1 Boolean function consists of a decision tree where for each node, one of its children is a constant. In the hypergraph corresponding to the rank-1 function f, for each variable x_i that appears in the normal form we add a vertex v_i. Add the vertices (i, b) where $1 \leq i \leq n$ and $b \in \{0, 1\}$. We also add two vertices $\mathbf{0}$ and $\mathbf{1}$ corresponding to the constants. Now for each variable x_i, if $x_i \in V_{j,b}(f)$ and one of its children is labeled with the constant c, add the hyperedge $(v_i, (j, b), c)$ in the hypergraph. We color all the vertices corresponding to the variables with one color and each (j, b) with a separate color. The vertices $\mathbf{0}$ and $\mathbf{1}$ are colored with different colors as well. Call the resulting rank-3 hypergraph $\mathcal{H}_f$. We have the following lemma.

Lemma 3.7. *Let f and g be Boolean functions of rank 1 given by decision trees, and let $\mathcal{H}_f$ and $\mathcal{H}_g$ be the hypergraphs constructed as above. Then, f and g are isomorphic as functions if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Rank-r functions: Let f be a rank-r function, and let $N_f = \langle l_i, N_{x_i} \rangle_{i \leq k}$, where $k \leq n$ and $l_i \in \{x_i, \overline{x_i}\}$, be the normal form for f. The vertex set for the hypergraph $\mathcal{H}_f$ is $\{u_1, \ldots, u_n\} \cup \{v_1^d, \ldots, v_n^d \mid 1 \leq d \leq r\} \cup \{(l, i, b, j) \mid 1 \leq l \leq r,\ 1 \leq i \leq n,\ 1 \leq j \leq r,\ b \in \{0, 1\}\} \cup \{\mathbf{0}, \mathbf{1}\}$. Intuitively, the vertices $u_1, \ldots, u_n$ will encode the variables $x_1, \ldots, x_n$ and $v_1^1, \ldots, v_n^1$ will encode the variables $x_1, \ldots, x_n$ at the outermost level in (l_1, N_{x_i}) pairs. Let $\mathcal{H}_i$ denote the hypergraph encoding N_{x_i}, constructed inductively. The vertex set of $\mathcal{H}_i$ will be $\{v_1^d, \ldots, v_n^d \mid 2 \leq d \leq r\} \cup \{(l, i, b, j) \mid 1 \leq l \leq r-1,\ 1 \leq i \leq n,\ b \in \{0, 1\},\ 2 \leq j \leq r\} \cup \{\mathbf{0}, \mathbf{1}\}$. We define the edge set for $\mathcal{H}_f$ as follows: For every $\langle l_i, N_{x_i} \rangle$ in the normal form and every edge $e \in \mathcal{H}_i$, we include $e \cup \{v_i^1\} \cup \{(l, j, b, 1)\}$ in the edge set if $x_i \in V_{j,b}^l(f)$ (where b encodes whether l_i is x_i or $\overline{x_i}$) and the edges $\{u_i, v_i^1\}$ for all i. Assume, inductively, that $\mathcal{H}_i$ is of rank at most $2(r-1)+1$. Then clearly H_f is of rank at most $2r+1$. If $f \cong g$ via $\pi \in S_n$, then since N_f and N_g are their normal form representations $(N_f)^\pi = N_g$, where $(N_f)^\pi$ is obtained by replacing x_i by $x_{\pi(i)}$ for all i in N_f. By induction on the rank r, we can easily argue that there is a $\pi \in S_n$ such that $(N_f)^\pi = N_g$ if and only if the hypergraphs H_f and H_g are isomorphic.

Lemma 3.8. *Let f and g be Boolean functions of rank r given by decision trees. Let $\mathcal{H}_f$ and $\mathcal{H}_g$ be the hypergraphs constructed as above. Then, f and g are isomorphic as functions if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

According to the construction, the hypergraph $\mathcal{H}_f$ corresponding to the rank-r function f has $2nr$ vertices and rank $2r+1$. The size of the hypergraph is at most $n^{O(r)}$ since any rank-r Boolean function has a rank-r decision tree of size $n^{O(r)}$. In particular, the normal form that we construct is of size at most $n^{O(r)}$. We formulate these observations in the following theorem.

Theorem 3.9. *Let f and g be Boolean functions of rank r given by decision trees T_f and T_g. There is an algorithm running in time $n^{O(r)}$ that outputs two hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ of rank $2r+1$ and size $n^{O(r)}$ such that f and g are isomorphic if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Since any decision tree of size s has rank at most $O(\log s)$, it has a normal form representation of size $n^{O(\log s)}$ which can be computed in $n^{O(\log s)} \cdot s^{O(1)}$. Hence we have the following corollary.

Corollary 3.10. *Let f and g be two decision trees of size s. There is an $s^{O(\log s)}$ time algorithm which computes hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ of logarithmic rank and size $s^{O(\log s)}$ such that f and g are isomorphic if and only if $\mathcal{H}_f \cong \mathcal{H}_g$.*

Combining this with the isomorphism algorithm for hypergraphs of bounded rank due to Babai and Codenotti [BC08], we observe the following:

Corollary 3.11. *Given two Boolean functions f and g as decision trees of size s, there is a $2^{\sqrt{s}(\log s)^{O(1)}}$ time algorithm to check if $f \cong g$.*

4 Isomorphism for Decision Lists

We now consider $\mathcal{C}$-DL Isomorphism (defined in Section 1), where $\mathcal{C}$ consists either of (i) conjunctions of k literals with at most one negated each, or (ii) conjunctions of k literals with at most one positive each, or (iii) conjunctions of 2 literals, or (iv) XORs of k literals. In all these cases, the $\mathcal{C}$-DL isomorphism problem is reducible to GI. Moreover, when $\mathcal{C}$ consists of XORs of two literals this isomorphism problem is in polynomial time. We will refer to this last case as $2\oplus$-DL. We have shown that Graph Isomorphism is reducible to $\mathcal{C}$-DL-Iso when $\mathcal{C}$ is any of the above four classes. For all other $\mathcal{C}$, by Lemma 1.3, $\mathcal{C}$-DL-Iso is coNP-hard. This shows the Schaefer-type dichotomy for the isomorphism problem of decision lists.

Let L be a $2\oplus$-DL, i.e., L is given by a sequence of pairs (p_i, b_i) where $b_i \in \{0,1\}$ and each p_i is an XOR of two literals. We say that a pair (p_i, b_i) *fires* on an assignment x, if i is the least index such that $p_i(x) = 1$. Let f_L denote the function computed by L.

We first construct a normal form representation for $2\oplus$-DLs to obtain an equivalent decision list where the tuples are partitioned into the sets $B_1, \ldots, B_m$,

where the second component of each pair in B_i is 0 if i is odd, and is 1 if i is even, for all i (in this normal form the set B_1 could possibly be empty). We then exploit the structure of the normal form and in polynomial time transform the 2⊕-DL isomorphism problem to Tree Isomorphism which can be solved efficiently.

We first explain the normal form N_L (which is also a 2⊕-DL) for a given 2⊕-DL L. For each pair of literals l_i and l_j, if $l_i \oplus l_j = 1$ implies that the function value is 0, we add $(l_i \oplus l_j, 0)$ to the set of pairs B_1 of N_L. We can find such pairs by replacing all occurrences of l_j by $\overline{l_i}$ in the decision list and checking if it computes the constant function 0. After B_1 is computed, we compute the set B_2 as follows: find literals l_r and l_p such that $(l_p \oplus l_r) \wedge \bigwedge_{p_i \in B_1} \neg p_i \Rightarrow f_L$. It is easy to see that such pairs can be found efficiently. For each such pair we include $(l_r \oplus l_p, 1)$ in B_2. Continuing this construction, we obtain sets of pairs $B_1, B_2, \ldots, B_m$. This 2⊕-DL is the normal form N_L for L. The following lemma summarizes this normal form construction.

Lemma 4.1. *If L is a 2⊕-DL then N_L is a normal form representation for L. Moreover, N_L is computable in time polynomial in $|L|$.*

Let L be a 2⊕-DL and N_L be its normal form. We will efficiently encode N_L as a rooted tree T_L so that the following holds: For two 2⊕-DLs L_1 and L_2, f_{L_1} and f_{L_2} are isomorphic if and only if the trees T_{L_1} and T_{L_2} are isomorphic. Recall that the normal form N_L consists of sets of pairs $B_1, B_2, \ldots, B_m$. Consider inputs for which none of the pairs $(l_i \oplus l_j, 0) \in B_1$ fire. For all such inputs $l_i = l_j$ holds for each pair $(l_i \oplus l_j, 0) \in B_1$. These equalities induces a partition on the set of all variables into subsets $A_{1,p}, A_{1,n}A_{2,p}, A_{2,n} \ldots, A_{k,p}, A_{k,n}$, so that on any input for which the decision list reaches B_2 and for each ℓ, all variables in $A_{\ell,p}$ are equal and all variables in $A_{\ell,n}$ are equal and complementary to the variables in $A_{\ell,p}$. Let $A_\ell = A_{\ell,p} \cup A_{\ell,n}$. Notice that some of the A_ℓ could be singletons, e.g. when the corresponding literals do not occur in B_1, and some $A_{\ell,p}$ or $A_{\ell,n}$ could be empty.

Now, we construct a new 2⊕-DL from N_L as follows: Delete B_1 from N_L. Introduce a new variables $y_\ell, 1 \leq \ell \leq k$, and in the decision list $B_2, \ldots, B_m$ replace by the variable y_ℓ all occurrences of variables in $A_{\ell,p}$ and replace by $\overline{y_\ell}$ all occurrences of variables in $A_{\ell,n}$, for each ℓ.

Let $\hat{L} = (\hat{B_2}, \ldots, \hat{B_m})$ denote this new 2⊕-DL defined on the new input variables $y_\ell, 1 \leq \ell \leq k$. Recursively, we obtain a rooted tree $T_{\hat{L}}$ with leaves labeled by $y_\ell, 1 \leq \ell \leq k$ corresponding to $\hat{L}$. Now, to obtain the rooted tree T_L from $T_{\hat{L}}$, we insert two children (ℓ, p) and (ℓ, n) to the leaf labeled y_ℓ, and make the elements of $A_{\ell,p}$ children of (ℓ, p) and the elements of $A_{\ell,n}$ children of (ℓ, n). To complete the construction note that the base case when N_L consists of only B_1 is easy to handle (since it is a constant function we create a depth one rooted tree T_L with one leaf for each variable).

Since N_L is a normal form representation for L, for any permutation π of the variables $x_1, x_2, \ldots, x_n$ we have $N_{L^\pi} = (N_L)^\pi$. Suppose L_1 and L_2 are 2⊕-DLs computing isomorphic functions and π is an isomorphism: L_1^π and L_2 are equivalent functions. Now, since N_{L_1} and N_{L_2} are normal forms for L_1 and L_2

we have $(N_{L_1})^\pi = N_{L_1^\pi} = N_{L_2}$. By the above construction it is easy to see that the rooted trees T_{L_1} and T_{L_2} are isomorphic via a permutation ψ such that ψ restricted to the leaves of T_{L_1} is the permutation π (indeed, ψ is the unique extension of π to the internal nodes of T_{L_1}). Conversely, if T_{L_1} and T_{L_2} are isomorphic via a permutation ψ then, from our construction we can argue that L_1 and L_2 are isomorphic, where the isomorphism is given by ψ restricted to the leaves of T_{L_1}. We summarize the above discussion in the following lemma.

Lemma 4.2. *The Boolean functions computed by two $2\oplus$-DLs L_1 and L_2 are isomorphic if and only if the rooted trees T_{L_1} and T_{L_2} are isomorphic. Furthermore, given an isomorphism from T_{L_1} to T_{L_2} we can recover in polynomial time an isomorphism from f_{L_1} to f_{L_2}.*

Since testing isomorphism between trees can be done efficiently, we have the following theorem.

Theorem 4.3. *The $2\oplus$-DL isomorphism problem is in polynomial time.*

We now turn to the remaining variants of the $\mathcal{C}$-DL isomorphism problem. A $\mathcal{C}$-DL is a list of pairs $(C(x_{i_1}, \ldots, x_{i_r}), b)$, where r is some fixed constant and $C(x_{i_1}, \ldots, x_{i_r})$ is a $\mathcal{C}$-term. We will consider $\mathcal{C}$ to be one of the following classes of functions: (i) The $\mathcal{C}$-terms are of the form $l_i \wedge l_j$. (ii) The $\mathcal{C}$-terms are of the form $l_{i_1} \wedge \cdots \wedge l_{i_r}$, where r is a fixed constant and at most one literal is positive. (iii) The $\mathcal{C}$-terms are of the form $l_{i_1} \wedge \cdots \wedge l_{i_r}$, where r is a fixed constant and at most one literal is negative. (iv) The $\mathcal{C}$-terms are of the form $l_{i_1} \oplus \cdots \oplus l_{i_r}$, where $r \geq 3$.

We show that in all these cases $\mathcal{C}$-DL Isomorphism is reducible to Graph Isomorphism.

Theorem 4.4. *The $\mathcal{C}$-DL isomorphism problem, where $\mathcal{C}$ is one of the function classes above, is polynomial-time reducible to Graph Isomorphism.*

We give a reduction from $\mathcal{C}$-DL-Iso to the label-respecting isomorphism problem of labeled trees which is equivalent to Graph Isomorphism [RZ00]. In this problem, we are given two rooted trees and additionally each vertex has a label. We ask if there is an isomorphism between the trees which is label-respecting. I.e. if two vertices in the first tree have the same label, their images in the second tree have the same label. An equivalent generalized version is finding isomorphism of colored labeled trees, in which each vertex also has a color and we ask for color-preserving, label-respecting isomorphism.

Given a $\mathcal{C}$-DL L on variables $\{x_1, \ldots, x_n\}$, we compute a normal form in polynomial time for the associated Boolean function f_L.

We find all r-tuples of literals, T_1, such that setting the associated $\mathcal{C}$-term to true forces f_L to be constant. In general, we find all r-tuples of literals, T_i, such that setting the associated $\mathcal{C}$-term to true given the premise that all the $\mathcal{C}$-terms in $T_j, j < i$ are false forces f_L to be constant. For the cases of $\mathcal{C}$ we are considering, checking this amounts to solving either 2CNF or Horn formula or Anti-Horn formula satisfiability or solving linear equations modulo 2.

Such a process yields a sequence $T_1, \ldots, T_m$ of sets of r-tuples of literals which partitions all (the at most $(2n)^r$ many) r-tuples of literals. As we saw for $2\oplus$-DLs, this sequence $T_1, \ldots, T_m$ actually yields a $\mathcal{C}$-DL which is a normal form representation for L.

We will now encode N_L as a labeled tree T_L (in the sense of [RZ00]). It turns out that two $\mathcal{C}$-DLs L_1 and L_2 compute isomorphic functions if and only if there is a label-respecting tree isomorphism from T_{L_1} to T_{L_2}. We outline the encoding algorithm which takes N_L as input and computes a labeled tree T_L: Let $T_1, T_2, \ldots, T_m$ be the r-tuple sets defining N_L. We create a root node with m children corresponding to $T_1, T_2, \ldots, T_m$, where the node for T_i is colored i. In the subtree rooted at the node corresponding to T_i we create a child c for each r-tuple $C \in T_i$. The node c will have r children which are leaves labeled by the corresponding variable name (in $x_1, x_2, \ldots, x_n$) and colored p or n depending on whether that literal occurring in C is positive or negative.

This completes the construction of the labeled tree T_L. It is easy to verify that if the Boolean functions computed by L_1 and L_2 are isomorphic via a permutation π then, in fact, π acting on the leaf labels of T_{L_1} induces an isomorphism from T_{L_1} to T_{L_2}. Conversely, if there is a label-respecting isomorphism ψ from T_{L_1} to T_{L_2}, then ψ induces a permutation π on the leaf labels of T_{L_1} which turns out to be an isomorphism from f_{L_1} to f_{L_2}. This completes the proof sketch of Theorem 4.4.

References

[ADKK12] V. Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert, *The isomorphism problem for k-trees is complete for logspace*, Inf. Comput. **217** (2012), 1–11.

[AT96] Manindra Agrawal and Thomas Thierauf, *The Boolean isomorphism problem*, FOCS, 1996, pp. 422–430.

[BC08] László Babai and Paolo Codenotti, *Isomorhism of hypergraphs of low rank in moderately exponential time*, FOCS, 2008, pp. 667–676.

[BHRV04] Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer, *The complexity of Boolean constraint isomorphism*, STACS, 2004, pp. 164–175.

[BL83] László Babai and Eugene M. Luks, *Canonical labeling of graphs*, STOC, 1983, pp. 171–183.

[EH89] Andrzej Ehrenfeucht and David Haussler, *Learning decision trees from random examples*, Inf. Comput. **82** (1989), no. 3, 231–246.

[Luk99] Eugene M. Luks, *Hypergraph isomorphism and structural equivalence of Boolean functions*, STOC, 1999, pp. 652–658.

[Riv87] Ronald L. Rivest, *Learning decision lists*, Machine Learning **2** (1987), no. 3, 229–246.

[RZ00] Sarnath Ramnath and Peiyi Zhao, *On the isomorphism of expressions*, Inf. Process. Lett. **74** (2000), no. 3-4, 97–102.

[Sch78] Thomas J. Schaefer, *The complexity of satisfiability problems*, STOC, 1978, pp. 216–226.

[Thi00] Thomas Thierauf, *The computational complexity of equivalence and isomorphism problems*, LNCS, vol. 1852, Springer, 2000.