Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Boolean Nearest Neighbor Language in the Knowledge Compilation Map

Ondřej Čepek[1] and **Jelena Glišić**[2]
[1]Department of Theoretical Computer Science and Mathematical Logic, Charles University
[2]Department of Applied Mathematics, Charles University
ondrej.cepek@mff.cuni.cz, glisic@kam.mff.cuni.cz

Abstract

The Boolean Nearest Neighbor (BNN) representation of Boolean functions was recently introduced by Hajnal, Liu and Turán. A BNN representation of f is a pair (P, N) of sets of Boolean vectors (called positive and negative prototypes) where $f(x) = 1$ for every positive prototype $x \in P$, $f(x) = 0$ for every negative prototype $x \in N$, and the value $f(x)$ for $x \notin P \cup N$ is determined by the type of the closest prototype. The main aim of this paper is to determine the position of the BNN language in the Knowledge Compilation Map (KCM). To this end, we settle the complexity status of most standard queries and transformations (those listed in KCM) for BNN inputs. We also compare the succinctness of the BNN language with several languages considered in KCM.

1 Introduction

Boolean functions constitute a fundamental concept in computer science, which often serves as the backbone of computation tasks and decision-making processes. Their significance extends across diverse domains including circuit design, artificial intelligence and cryptography. However, as the complexity of systems increases, the need for efficient representation and manipulation of Boolean functions becomes more important. There are many different ways in which a Boolean function may be represented. Common representations include truth tables (TT) (where a function value is explicitly given for every binary vector), list of models (MODS), i.e., a list of binary vectors on which the function evaluates to 1, various types of Boolean formulas (including CNF and DNF representations), various types of binary decision diagrams (BDDs, FBDDs, OBDDs), negation normal forms (NNF, DNNF, d-DNNF), and other Boolean circuits.

The task of transforming one of the representations of a given function f into another representation of f (e.g. transforming a DNF representation into an OBDD or a DNNF into a CNF) is called knowledge compilation. This task emerged as an important ingredient of modern computation, aiming to transform Boolean functions into more compact and tractable forms. This allows us to use techniques from logic, algorithms, and data structures in order to bridge the gap between the expressive power of Boolean functions and the efficiency requirements of practical applications. For a comprehensive review paper on knowledge compilation see Darwiche and Marquis (2002), where the Knowledge Compilation Map (KCM) is introduced. KCM systematically investigates different knowledge representation languages with respect to (1) their relative succinctness, (2) the complexity of common transformations, and (3) the complexity of common queries. The *succinctness* of the representations describes how large the output representation in language B is with respect to the size of the input representation in language A when compiling from A to B. A precise definition of this notion will be given later in this text. Transformations include negation, conjunction, disjunction, conditioning, and forgetting. The complexity of such transformations may differ dramatically from trivial to computationally hard depending on the chosen representation language. The same is true for queries such as consistency check, validity check, clausal and sentential entailment, equivalence check, model counting, and model enumeration.

The number of knowledge representation languages in the Knowledge Compilation Map gradually increases. In Berre et al. (2018) the authors included Pseudo-Boolean constraint (PBC) and Cardinality constraint (CARD) languages into KCM by showing succinctness relations among PBC, CARD, and languages already in the map, and by proving the complexity status of all queries and transformations introduced in Darwiche and Marquis (2002). The same goal was achieved for the switch-list (SL) language in Čepek and Chromý (2020). In this paper, our aim is to solve the same task for Boolean Nearest Neighbor (BNN) representations introduced recently in Hajnal, Liu, and Turán (2022). Let us denote $\mathcal{B} = \{0, 1\}$ and d_H the Hamming distance on $\mathcal{B}^n$.

Definition 1 (Hajnal, Liu, and Turán (2022))**.** *A* Boolean nearest neighbor (BNN) *representation of a Boolean function f in n variables is a pair of disjoint subsets (P, N) of $\mathcal{B}^n$ (called* positive and negative prototypes*) such that for every $a \in \mathcal{B}^n$*

- *if $f(a) = 1$ then there exists $b \in P$ such that for every $c \in N$ it holds that $d_H(a, b) < d_H(a, c)$,*
- *if $f(a) = 0$ then there exists $b \in N$ such that for every $c \in P$ it holds that $d_H(a, b) < d_H(a, c)$.*

The Boolean nearest neighbor complexity of f, $BNN(f)$, is the minimum of the sizes of the BNN representations of f. That is, $BNN(f) = \min_{(P,N) \text{ represents } f}(|P| + |N|)$.

BNN representations of Boolean functions were introduced in Hajnal, Liu, and Turán (2022) as a special case of a more general Nearest Neighbor (NN) representations where the prototypes are not restricted to Boolean vectors but instead can be any real-valued vectors from $\mathbb{R}^n$. In this case, Euclidean distance d_E is used instead of the Hamming distance d_H. Since for any two vectors $x, y \in \mathcal{B}^n$, $d_E(x,y) = \sqrt{d_H(x,y)}$, any BNN representation, is also an NN representation. NN representations of Boolean functions were studied already more than thirty years ago Wilfong (1991), but there are also many recent works Banerjee, Bhore, and Chitnis (2018); Gottlieb, Kontorovich, and Nisnevitch (2018); Kilic, Sima, and Bruck (2023).

NN representations of Boolean functions are a special case of an even more general concept of NN classification problems in which prototypes can be of several classes. Such problems are studied in the areas of machine learning, learning theory, and computational geometry Luxburg and Bousquet (2004); Klenk, Aha, and Molineaux (2011); Anthony and Ratsaby (2018). When dealing with a nearest neighbor representation, the objective is usually to minimize the number of prototypes.

Another way of looking at BNN representations is to view them as a special case of partially defined Boolean functions (pdBf) Crama, Hammer, and Ibaraki (1988); Boros, Ibaraki, and Makino (1998) with an explicit way in which the function is extended to all unclassified vectors.

In the wide range of different knowledge representation languages, BNN representations belong to the group of languages based on lists of Boolean vectors. Other such languages are TT (truth table), MODS (list of models), IR (interval representations) Schieber, Geist, and Zaks (2005); Čepek, Kronus, and Kučera (2008) and SLR (switch list representations) Čepek and Hušek (2017); Čepek and Chromý (2020).

In this paper we shall concentrate on the time complexity of standard transformations (Section 3) and queries (Section 4) for the BNN language. We also establish several succinctness results that relate BNN to other knowledge representation languages in Section 5. First, we need to define several notions and recall few results in Section 2.

2 Preliminaries

We will use the following notation and notions:

- the symbol $\mathcal{B}$ denotes the set $\{0,1\}$ and $\mathcal{B}^n = \{0,1\}^n$ denotes the Boolean hypercube;
- by $d_H(x,y)$ for $x \in \mathcal{B}^n$ we denote the Hamming distance in the Boolean hypercube, i.e., the number of coordinates in which x and y differ;
- by $d_H(x,\mathcal{A})$ for $x \in \mathcal{B}^n$ and $\mathcal{A} \subseteq \mathcal{B}^n$ we denote $\min\{d_H(x,y) \mid y \in \mathcal{A}\}$;
- by $|x|$ for $x \in \mathcal{B}^n$ we denote the weight of x, i.e., the number of coordinates of x which are equal to 1;

Given a Boolean function $f : \mathcal{B}^n \to \mathcal{B}$ in n variables, we say that $x \in \mathcal{B}^n$ is a positive vector or a model of f (resp. negative vector or a non-model of f) if $f(x) = 1$ (resp. $f(x) = 0$).

Definition 2. *We call a Boolean function f in n variables a* symmetric function *if there exists a set $I_f \subseteq \{0,1,\ldots,n\}$ such that $f(x) = 1$ if and only if $|x| \in I_f$. We consider the following special cases of symmetric functions:*

1. PAR_n *(*parity function*) with $I_f = \{i \text{ is odd} \mid 1 \le i \le n\}$*
2. MAJ_n *(*majority function*) with $I_f = \{i \mid i \ge n/2\}$*
3. TH_n^t *(*threshold function *with unit weights and threshold t) with $I_f = \{i \mid i \ge t\}$ for $1 \le t \le n$.*

Definition 3. *The* Boolean hypercube graph *$\mathcal{B}_n = (V,E)$ is an undirected graph with vertex set $V = \mathcal{B}^n$ and edge set $E = \{(x,y) \mid d_H(x,y) = 1\}$. Finally, for a set of Boolean vectors $S \subseteq \mathcal{B}^n$ we define its neighborhood by*

$$\delta(S) = \{x \in \mathcal{B}^n \mid d_H(x,S) = 1\}.$$

In the following sections, we will often use the terms Boolean hypercube (denoted by $\mathcal{B}^n$) and Boolean hypercube graph (denoted by $\mathcal{B}_n$) interchangeably. In particular, a neighbor of a vector x in the Boolean hypercube will be any vector that is adjacent to it in the Boolean hypercube graph. Now we present two recent results which we will frequently use in the rest of this paper.

Theorem 1 (Hajnal, Liu, and Turán (2022))**.** *The following statements hold:*

(a) $BNN(PAR_n) = 2^n$,

(b) If n is odd then $BNN(MAJ_n) = 2$ and if n is even then $BNN(MAJ_n) \le \frac{n}{2} + 2$,

(c) $BNN(TH_n^{\lceil n/3 \rceil}) = 2^{\Omega(n)}$.

Lemma 1 (DiCicco, Podolskii, and Reichman (2024))**.** *Let f be a Boolean function on n variables. If the subgraph of $\mathcal{B}_n$ induced by the models of f has m connected components, then any BNN representation of f has at least one positive prototype in each connected component (and hence at least m positive prototypes in total). In particular, if x is a positive vector such that all of its neighbors have the opposite function value (i.e., x is isolated), then x must be a positive prototype in any BNN representation of f.*

Note that Lemma 1 holds by a symmetric argument also for non-models and negative prototypes.

3 Transformations

In this section we shall show that the **BNN** language unfortunately does not support any standard transformation from Darwiche and Marquis (2002) except negation (denoted $\neg$**C**), which is a trivial transformation for **BNN**, and singleton forgetting (denoted **SFO**), for which the complexity status remains open.

Observation 1. ***BNN** supports $\neg$**C**.*

Proof. Since no ties are allowed, i.e., for every $x \in \mathcal{B}^n$ all prototypes nearest to x must be of the same type, then it follows that f is represented by BNN (P,N) if and only if $\neg f$ is represented by BNN (N,P). Thus we may perform negation by outputting (N,P), which takes polynomial time. □

Now we shall consider conditioning on a set of variables (denoted **CD**) which transforms a given BNN by fixing a set of variables to constants, and forgetting a set of variables (denoted **FO**) which transforms a given BNN by existentially quantifying a set of variables. The fact that **BNN** does not support **CD** and **FO** can be proved using the exponential lower bound for threshold functions from Theorem 1.

Theorem 2. ***BNN*** *does not support* ***CD****.*

Proof. Let (P, N) be the smallest BNN representing the Boolean majority function on $n = 4k$ variables, for some $k \in \mathbb{N}$. That is, (P, N) represents the function TH^{2k}_{4k} and $|(P, N)| \leq \frac{n}{2} + 2 = 2(k + 1)$ holds by Theorem 1. Let $x_1, \ldots, x_n$ denote the variables of the threshold function. Notice that by setting some variable x_i to 1, i.e., conditioning on term $T = x_i$ we obtain a threshold function that has one variable fewer and threshold smaller by one.

Let $T_k = x_1 \wedge x_2 \wedge \cdots \wedge x_k$ be a consistent term. We denote by $TH^{2k}_{4k} \mid T_k$ the function resulting from TH^{2k}_{4k} by setting all literals in T_k to 1. Then $(TH^{2k}_{4k} \mid T_k) = TH^{k}_{3k}$. It follows from Theorem 1 that in order to represent such a function, we need a BNN of size $2^{\Omega(3k)} = 2^{\Omega(n)}$, and thus cannot produce it in polynomial time from the input (P, N) of size $O(n)$. □

The following lemma shows that for threshold functions conditioning (by a positive term) and forgetting work the same way producing the same result.

Lemma 2. *For $m, n \in \mathbb{N}$, $m \leq n$ consider the threshold function $f = TH^m_n$ and let $i \in \{1, 2, \ldots, n\}$ be arbitrary. Then*

$$f|x_i \equiv \exists x_i.f(x_1, x_2, \ldots, x_n)\,.$$

Proof. The proof of Theorem 2 implies $f|x_i \equiv TH^{m-1}_{n-1}$. Thus it suffices to show that also $\exists x_i.f(x_1, x_2, \ldots, x_n) \equiv TH^{m-1}_{n-1}$. By definition:

$$\begin{gathered}\exists x_i.f(x_1, x_2, \ldots, x_n) \equiv \\ f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee \\ f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \equiv \\ TH^m_{n-1} \vee TH^{m-1}_{n-1}\end{gathered}$$

Notice now that models of TH^m_{n-1} form a subset of models of TH^{m-1}_{n-1}. We may then omit TH^m_{n-1} and hence $\exists x_i.f(x_1, x_2, \ldots, x_n) \equiv TH^{m-1}_{n-1}$, as desired. □

Theorem 3. ***BNN*** *does not support* ***FO****.*

Proof. We combine Theorem 2 and Lemma 2. Again, $TH^{n/2}_n$ can be represented by a BNN of size $O(n)$, but we need a BNN of size $2^{\Omega(n)}$ after forgetting the first $n/4$ variables. □

Finally, we shall show that the **BNN** language does not support conjunction (denoted $\wedge$**C**) and disjunction (denoted $\vee$**C**) even in the bounded cases (denoted $\wedge$**BC** and $\vee$**BC**) in which only two conjuncts (disjuncts) are considered.

Theorem 4. ***BNN*** *does not support* $\wedge$***BC****.*

Proof. Consider the function MAJ_{2n}. This function has a BNN representation of size $O(n)$ by Theorem 1. Furthermore, we may consider the minority function on $2n$ variables MIN_{2n} with $I_{MIN_{2n}} = \{i \mid i \leq n\}$. Clearly, this function again admits a BNN representation of size $O(n)$. Now consider the function $f = MAJ_{2n} \wedge MIN_{2n}$. This function has exactly $\binom{2n}{n} = 2^{\Omega(n)}$ isolated models. It follows by Lemma 1 that any BNN representation of f has size $2^{\Omega(n)}$. □

The next corollary follows by considering the disjunction of $\neg MAJ_{2n}$ and $\neg MIN_{2n}$ (which has $\binom{2n}{n} = 2^{\Omega(n)}$ isolated non-models).

Corollary 1. ***BNN*** *does not support* $\vee$***BC****.*

The above results show that the set of supported transformations is very small. The biggest drawback is (in our opinion) the fact that unlike all other knowledge representation languages considered in Darwiche and Marquis (2002), Berre et al. (2018), and Čepek and Chromý (2020) the **BNN** language does not support conditioning. Conditioning is an essential transformation that is needed in many applications. In particular, if some of the variables are observable (and the others are decision variables) then queries are often asked after some (or all) of the observable variables are fixed to the observed values (which amounts to conditioning on this set of variables and only then answering the query).

In this context, we note that aside from singleton forgetting (**SFO**), i.e., forgetting a single variable rather than a set of variables, singleton conditioning (**SCD**) is also a transformation with an unknown complexity status. **SCD** was not listed among standard transformations in Darwiche and Marquis (2002) since all languages considered there satisfy even general conditioning (**CD**), but it makes sense to consider **SCD** for the **BNN** language. We conjecture that both **SCD** and **SFO** can be performed in polynomial time. This property (if true) together with the reasonably large set of supported queries (see the next section) would justify the usefulness of the **BNN** language as a target compilation language. Indeed, if **SCD** is supported, then any conditioning on a constant number of variables could lead to only a polynomial blow-up of the resulting BNN representation, while the current proof of hardness for **CD** requires conditioning on $\Omega(n)$ variables.

The complexity status of all standard transformations for **BNN** and selected standard languages is summarized in Table 1.

4 Queries

In this section, we show that the **BNN** language supports a reasonably large subset of standard queries from Darwiche and Marquis (2002). We begin with the trivial observation that both consistency (i.e., existence of a model, denoted **CO**) and validity (i.e., every vector is a model, denoted **VA**) can be checked in constant time.

Observation 2. ***BNN*** *supports* ***CO*** *and* ***VA****.*

L	CD	FO	SFO	∧C	∧BC	∨C	∨BC	¬C
NNF	✓	○	✓	✓	✓	✓	✓	✓
d-NNF	✓	○	✓	✓	✓	✓	✓	✓
DNNF	✓	✓	✓	○	○	✓	✓	○
d-DNNF	✓	○	○	○	○	○	○	?
BDD	✓	○	✓	✓	✓	✓	✓	✓
FBDD	✓	•	○	•	○	•	○	✓
OBDD	✓	•	✓	•	○	•	○	✓
CNF	✓	○	✓	✓	✓	•	✓	•
DNF	✓	✓	✓	•	✓	✓	✓	•
PI	✓	✓	✓	•	•	•	✓	•
IP	✓	•	•	•	✓	•	•	•
MODS	✓	✓	✓	•	✓	•	•	•
CARD	✓	•	•	✓	✓	•	•	•
PBC	✓	•	•	✓	✓	•	•	•
SL	✓	✓	✓	•	•	•	•	✓
BNN	•	•	?	•	•	•	•	✓

Table 1: The languages introduced in Darwiche and Marquis (2002), Berre et al. (2018), and Čepek and Chromý (2020) and the corresponding complexity of standard transformations. Here ✓ means "satisfies", • means "does not satisfy", ○ means "does not satisfy unless P=NP", and ? corresponds to an open problem.

Proof. For a BNN representation (P, N) consistency check is equivalent to checking that P is non-empty while validity check is equivalent to checking that N is empty. Both of these checks can be done in constant time. □

Next, we consider the implicant check query (denoted **IM**) which for a given term (conjunction of literals) T and a given BNN representing function f checks whether T is an implicant of f.

Theorem 5. ***BNN*** *supports* ***IM****.*

Proof. Let f be a Boolean function and (P, N) its BNN representation. Let $T = l_1 \wedge \cdots \wedge l_k$ be a consistent term. Without loss of generality, we may assume that $\forall 1 \leq i \leq k : l_i \in \{x_i, \neg x_i\}$, since otherwise we may relabel the variables. We aim to design a polynomial-time algorithm that checks whether $T \implies f$. Let us denote by $y \in \mathcal{B}^k$ the (partial) vector satisfying T, i.e., defined by $y_i = 1$ if $l_i = x_i$ and $y_i = 0$ if $l_i = \neg x_i$. Furthermore, let S denote the sub-cube of $\mathcal{B}^n$ determined by the vector y. That is,

$$S := \{x \in \mathcal{B}^n \mid \forall 1 \leq i \leq k : x_i = y_i\}.$$

Notice that $T \implies f$ if and only if there is no negative vector of f inside S (i.e., after fixing the values in y the resulting function is constantly 1). We shall show that this condition can be tested efficiently.

If there is a negative prototype in S, we are done and T is not an implicant of f. If all negative prototypes are outside of S, let us denote by N' the set of projections of all negative prototypes into S:

$$N' := \{\text{proj}_S(q) = (y_1, \ldots, y_k, q_{k+1}, \ldots, q_n) \mid q \in N\},$$

If there exists a negative prototype q which is at most as far from its projection $\text{proj}_S(q)$ than any positive prototype, i.e.

$$\exists q \in N : \forall p \in P : d_H(p, \text{proj}_S(q)) \geq d_H(q, \text{proj}_S(q))$$

then either there exists another negative prototype which is strictly closer to $\text{proj}_S(q)$ than q, or q is the closest negative prototype to $\text{proj}_S(q)$ in which case the above inequality must be strict for every positive prototype by the definition of BNN representation. In both cases $f(\text{proj}_S(q)) = 0$ follows, we have found a negative vector of f in S, and thus we can again conclude that T is not an implicant of f. Note, that this condition can be tested in polynomial time with respect to the size of P for any fixed q.

Let us now assume the opposite, namely that for every $\text{proj}_S(q) \in N'$ there exists a positive prototype $p \in P$ which is strictly closer to it than q:

$$\forall q \in N : \exists p \in P : d_H(p, \text{proj}_S(q)) < d_H(q, \text{proj}_S(q))$$

We claim that in this case there are no negative vectors of f inside S, and we can conclude that T is an implicant of f. Let us assume for contradiction that there exists such a vector $x \in S$ for which $f(x) = 0$. Let q be a closest negative prototype to x and let $q' = \text{proj}_S(q)$ be its projection on S. Furthermore, let p be the positive prototype closest to q'. Then

$$\begin{aligned} d_H(q, x) &= d_H(q, q') + d_H(q', x) \\ &> d_H(p, q') + d_H(q', x) \geq d_H(p, x). \end{aligned}$$

The first equality holds because $d_H(q, q')$ depends only on the first k coordinates while $d_H(q', x)$ depends only on the remaining coordinates. The first inequality follows from the assumption and the second from the triangle inequality for Hamming distance. However, $d_H(q, x) > d_H(p, x)$ implies $f(x) = 1$ which is a contradiction.

The above discussion is summarized in Algorithm 1. The algorithm runs in time $O(n|P||N|)$ (which is polynomial in the size of the input $n(|P|+|N|)$), as its main part consists of two nested *for* loops. In the worst case, the algorithm considers projections of all negative prototypes and calculates their distances to each of the positive prototypes. □

Algorithm 1 Implicant check for a BNN representation

Input: (P, N) representing f and a consistent term T
Output: **IM**(f, T)

```
for q ∈ N do
    if q ∈ S then
        return 0
    q' ← proj_S(q)
    d ← +∞
    for p ∈ P do
        d ← min{d, d_H(p, q')}
    if d_H(q, q') ≤ d then
        return 0
return 1
```

Since negation is trivial for the **BNN** language by Observation 1, we immediately get the following result for clausal entailment (denoted **CE**).

Corollary 2. ***BNN** supports **CE**.*

Proof. Let f be a Boolean function and (P, N) its BNN representation. Let $C = l_1 \vee \cdots \vee l_k$ be a consistent clause. Our aim is to design a polynomial-time algorithm that checks whether $f \implies C$. Clearly $(f \implies C) \equiv (\neg C \implies \neg f)$. Moreover, $\neg f$ is readily available (recall Observation 1), and by DeMorgan laws the negation of a clause C is a term T. So let $T = \neg C$. Now, we may check whether f entails C by calling Algorithm 1 for inputs $\neg f = (N, P)$ and $\neg C = T$, getting the correct answer in polynomial time. □

It is interesting to note that for most standard languages which support **CE** this property stems from the fact that such languages support **CD** and **CO**. In these cases, the **CE** algorithm first performs the required conditioning, then tests consistency, and then outputs yes if and only if the partial function is not consistent. This is not the case for **BNN**, as it supports **CE** despite not supporting **CD**. We are not aware of any other knowledge representation language that supports clausal entailment without supporting conditioning[1].

The fact that **BNN** supports clausal entailment also directly implies that **BNN** supports model enumeration.

Corollary 3. ***BNN** supports **ME**.*

Proof. Models can be enumerated using a tree search such as the simple DPLL algorithm by Davis, Logemann, and Loveland (1962) where at every step before a value is assigned to a variable and a branch to a node on the next level is built, it is first tested whether the current partial assignment of values to variables plus the considered assignment yields a subfunction which is identically zero (and if yes then the branch is not built). This amounts to a clausal entailment test. Therefore, all branches of the built tree have full length (all variables are fixed to constants) and terminate at models. Thus, both the size of the tree and the total work required is upper bounded by a polynomial in the number of models. □

In the rest of this section, we shall prove that **BNN** does not support the remaining standard queries, i.e., that there are no polynomial-time algorithms for equivalence check (denoted **EQ**), sentential entailment (denoted **SE**), and model counting (denoted **CT**) queries unless P=NP. To do so, we consider the following decision problem:

Half-Size Independent Set (HSIS)
Input: An undirected graph $G = (V, E)$ with n vertices.
Question: Does there exist an independent set with exactly $n/2$ vertices in G?

Although IS, the general independent set problem (in which a parameter k is part of the input and the question asks for the existence of an independent set of size k), is widely known to be NP-complete, we have to argue that also the restricted version HSIS is hard. To see this, consider the textbook reduction from 3-SAT to IS which for every cubic clause creates a clique of size three and then connects vertices that correspond to complementary literals (one edge for each such pair). It is easy to see that the input 3-SAT instance (with m clauses) is satisfiable if and only if the constructed graph on $3m$ vertices contains an independent set of size exactly m. Modifying this construction by adding m isolated vertices yields a reduction in which the input 3-SAT instance is satisfiable if and only if the constructed graph on $4m$ vertices contains an independent set of size exactly $2m$, which is an instance of HSIS.

Now we are ready to prove the first hardness result.

Theorem 6. ***EQ** is co-NP complete for **BNN**.*

Proof. First, observe that the equivalence query for **BNN** belongs to co-NP. A certificate for a negative answer is a vector on which the two input BNNs give opposite function values.

For the hardness part, let $G = (V, E)$ with $n = 2k$ vertices be an instance of HSIS. We assume without loss of generality that $k > 2$. We define two input BNN representations for the EQ query as follows:

1. $F = (P_f, N_f)$ is a BNN representation of the majority function $f = MAJ_{2k}$. We assume that F is the representation of f from the proof of Theorem 1, namely:
   - $P_f = \{x \in \mathcal{B}^{2k} \mid |x| = 2k - 1\}$, i.e all[2] vectors of weight $2k - 1$, and
   - $N_f = \{(0, 0, \ldots, 0)\}$.
2. $H = (P_h, N_h)$ is a BNN representation of function h defined as follows:
   - $P_h = \{p\} \cup \{p^e \mid e \in E\}$, where $p = (1, 1, \ldots, 1)$ and for every $e = (i, j) \in E$ the vector p^e has weight $2k - 2$ with $p^e_i = p^e_j = 0$.
   - $N_h = \{x \in \mathcal{B}^{2k} \mid |x| = 1\}$. Let us denote these vectors by $n^1, \ldots, n^{2k}$ with $n^i_i = 1$.

First, we shall show that $h(x) = f(x)$ holds for all vectors with weight different from k.

- Let $x \in \mathcal{B}^{2k}$ be a vector with $|x| \leq k - 1$. If we pick any index i with $x_i = 1$ then the negative prototype n^i of weight 1 satisfies $d_H(x, n^i) \leq k - 2$ (and if x is the all-zero vector then $d_H(x, n^i) = 1 \leq k - 2$ as $k > 2$ was assumed). On the other hand, all positive prototypes are at a Hamming distance at least $k - 1$ from x since at least that many zero-bits in x have to be flipped to get a vector of weight $2k - 2$ or more. Thus $h(x) = 0 = f(x)$.
- Let $x \in \mathcal{B}^{2k}$ be a vector with $|x| \geq k + 1$. In this case, we have $d_H(x, p) \leq k - 1$. All negative prototypes are at a Hamming distance of at least k from x since at least that many one-bits in x have to be flipped to get a vector of weight 1. Thus $h(x) = 1 = f(x)$.

We now investigate the middle level of the Boolean lattice.

[1] It is possible that compressed SDDs (sentential decision diagrams) do not support unbounded conditioning while supporting clausal entailment. This was remarked in Bova (2016), but we are not sure if this question was ever resolved.

[2] The proof of Theorem 1 in fact uses only arbitrarily chosen $k+1$ vectors of weight $2k-1$, however, we do not need a minimal representation here and taking all vectors of weight $2k - 1$ does not change the represented function.

- Assume that there are no independent sets of size k in G and let $x \in \mathcal{B}^{2k}$ be a vector with $|x| = k$. Vector x has exactly k coordinates with $x_i = 0$ and by our assumption, this index set cannot be an independent set of G. Thus there must exist $e = (i, j) \in E$ such that $x_i = x_j = 0$ and hence $d_H(x, p^e) = k - 2$ since only $k - 2$ zero-bits in x have to be flipped to arrive to p^e. On the other hand, all negative prototypes are at a Hamming distance at least $k - 1$ from x since exactly that many one-bits in x have to be flipped to get a vector of weight 1. Thus $h(x) = 1 = f(x)$ for every vector x of weight k.
- Now assume that there exists an independent set of size k in G, defined by an index set S. Let $x^S \in \mathcal{B}^{2k}$ be a vector with $|x^S| = k$ where S defines the zero-bits of x^S. Clearly, $d_H(x^S, p) = k$ and also $d_H(x^S, p^e) \geq k$ for every $e \in E$. To see the latter, notice that if $|e \cap S| = 1$ we need to flip $k - 1$ zero-bits and one one-bit in x^S to arrive to p^e, and if $|e \cap S| = 0$ we need to flip all k zero-bits and two one-bits in x^S to arrive to p^e. On the other hand, if we pick any index i with $x^S_i = 1$ then the negative prototype n^i of weight 1 satisfies $d_H(x^S, n^i) = k - 1$. Thus $h(x^S) = 0$ while $f(x^S) = 1$.

If we summarize the above observations we get that $h \equiv f$ if and only if there exists no independent set of size k in G. In other words, the answer to the equivalence query on BNN representations F and H is yes if and only if the answer to the input HSIS instance G is no. Since both F and H can be constructed from G in polynomial time, this finishes the proof of NP-hardness of **EQ**. □

Theorem 6 immediately gives us the following corollary.

Corollary 4. ***SE*** *is co-NP complete for* ***BNN****.*

Proof. It is again easy to see that the sentential entailment query for **BNN** belongs to co-NP. Given a query $F \models H$, a certificate for a negative answer is a vector x such that F classifies x as a positive vector while H classifies x as a negative vector (both properties can be checked in polynomial time).

The hardness part is a direct consequence of Theorem 6 since $F \equiv H$ if and only if $F \models H$ and $H \models F$, so any equivalence query can be answered by asking two sentential entailment queries on the same pair of input BNN representations[3]. □

Finally, the proof of Theorem 6 implies the following.

Corollary 5. ***CT*** *is NP-hard for* ***BNN****.*

Proof. Notice that the number of models for $F = MAJ_{2k}$ is easy to compute, in particular, this number equals half of all vectors plus half of the middle level. That is, $CT(F) = 2^{2k-1} + \frac{1}{2}\binom{2k}{k}$.

[3]While the reduction from EQ to SE requires two SE queries, the reduction from HSIS to SE in fact requires a single check $F \models H$, because the models of H always form a subset of models of F, and therefore $H \models F$ holds by the construction in the proof of Theorem 6.

It follows from the proof of Theorem 6 that $CT(H) = CT(F)$ if and only if $H \equiv F$ (recall that the models of H always form a subset of models of F), and hence the ability to compute $CT(H)$ in polynomial time would answer the EQ query for F and H and thus decide the input HSIS instance G. □

We remark that the reduction in the proof of Theorem 6 is number preserving: any two distinct independent sets of size k in G correspond to two distinct non-models of h of weight k. Therefore, $CT(F) - CT(H)$ equals the number of independent sets of size k in G. This means that if the counting version of the HSIS problem is #P hard (which we do not know, but it likely is), then the model counting query for BNN is also #P hard.

The complexity status of all standard queries for **BNN** and selected standard languages is summarized in Table 2.

L	CO	VA	CE	IM	EQ	SE	CT	ME
NNF	∘	∘	∘	∘	∘	∘	∘	∘
d-NNF	∘	∘	∘	∘	∘	∘	∘	∘
DNNF	✓	∘	✓	∘	∘	∘	∘	✓
d-DNNF	✓	✓	✓	✓	?	∘	✓	✓
BDD	∘	∘	∘	∘	∘	∘	∘	∘
FBDD	✓	✓	✓	✓	?	∘	✓	✓
OBDD	✓	✓	✓	✓	✓	∘	✓	✓
CNF	∘	✓	∘	✓	∘	∘	∘	∘
DNF	✓	∘	✓	∘	∘	∘	∘	✓
PI	✓	✓	✓	✓	✓	✓	∘	✓
IP	✓	✓	✓	✓	✓	✓	∘	✓
MODS	✓	✓	✓	✓	✓	✓	✓	✓
CARD	∘	✓	∘	✓	∘	∘	∘	∘
PBC	∘	✓	∘	✓	∘	∘	∘	∘
SL	✓	✓	✓	✓	✓	✓	✓	✓
BNN	✓	✓	✓	✓	∘	∘	∘	✓

Table 2: The languages from Darwiche and Marquis (2002), Berre et al. (2018), and Čepek and Chromý (2020) and the complexity of standard queries. Here ✓ means "satisfies", ∘ means "does not satisfy unless P=NP", and ? corresponds to an open problem.

5 Succinctness

In this section, we partially establish the position of the **BNN** language in the succinctness diagram presented in Darwiche and Marquis (2002). Let us start with the formal definition of succinctness.

Definition 4. *Let L_1 and L_2 be two knowledge representation languages. We say that L_1 is* at least as succinct as *L_2, denoted $L_1 \leq L_2$, if and only if there exists a polynomial p such that for every sentence $\alpha \in L_2$, there exists an equivalent sentence $\beta \in L_1$ where $|\beta| \leq p(|\alpha|)$. Furthermore, L_1 is* strictly more succinct than *L_2, denoted $L_1 < L_2$, if $L_1 \leq L_2$ but $L_2 \not\leq L_1$.*

We show a diagram representing the known succinctness results as per Čepek (2022) in Figure 1. In this paper we restrict our attention to the seven relations that connect **BNN** with standard KCM languages in the diagram in Figure 2.

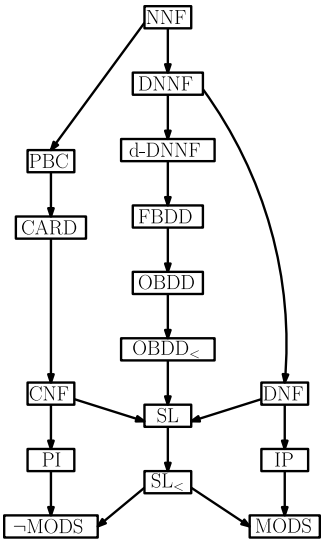

Figure 1: Diagram representing known succinctness results. The languages included are from Darwiche and Marquis (2002), Berre et al. (2018) and Čepek and Chromý (2020) An edge $L_1 \to L_2$ indicates that L_1 is strictly more succinct than L_2.

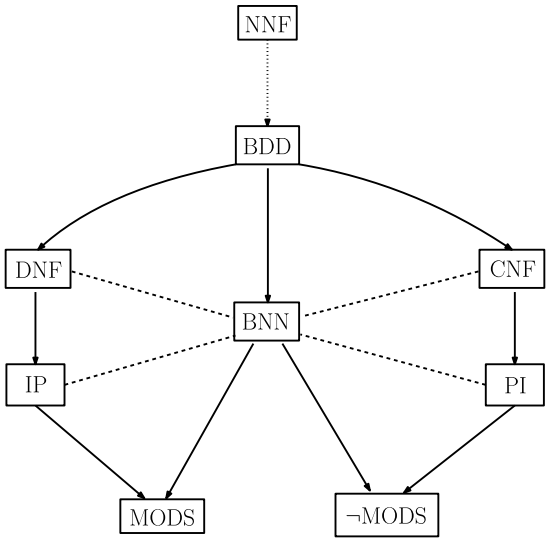

Figure 2: A solid directed edge from L_1 to L_2 indicates that L_1 is strictly more succinct than L_2. A dotted directed edge from L_1 to L_2 indicates that L_1 is at least as succinct as L_2. A dashed undirected edge between L_1 and L_2 means that L_1 and L_2 are incomparable.

These results determine the position of the **BNN** language in the original succinctness diagram from Darwiche and Marquis (2002).

It should be noted that the **BDD** language (BDD = binary decision diagram) is missing in Figure 1, so we should also argue about the relations that concern **BDD**. The relation **NNF** $\leq$ **BDD** follows from the fact that **BDD** is a strict subset of **NNF** (Darwiche and Marquis (2002)). To show the relations **BDD** $<$ **CNF** and **BDD** $<$ **DNF** we utilize the fact that any Boolean formula with negations at the variables, which uses only conjunction and disjunction as connectives, can be compiled into a BDD with only a linear blowup (Wegener (2000)) (which proves $\leq$), and the fact that **CNF** and **DNF** are incomparable (proving the strict relation). We begin our study of the relations concerning the **BNN** language by comparing it to **BDD**.

Theorem 7. ***BDD*** $<$ ***BNN***.

Proof. For the non-strict relation **BDD** $\leq$ **BNN** it suffices to show that there exists a polynomial p such that for every sentence $\alpha \in \mathbf{BNN}$, there exists an equivalent sentence $\beta \in \mathbf{BDD}$ where $|\beta| \leq p(|\alpha|)$. For any $\alpha = (P, N)$, we will construct such a binary decision diagram β. Let us assume that $P \cup N = \{p^1, \ldots, p^k\}$ and let $x \in \mathcal{B}^n$. We build β in two steps:

1. We construct a gadget which for two fixed prototypes p^i and p^j decides which one is closer to input x.
2. We put a number of these gadgets together so that a prototype closest to x is found and its value outputted.

We begin by building the BDD gadget. For a fixed $p^i, p^j \in P \cup N$, we construct a diagram $G_{i,j}$, as shown in Figure 3 for $n = 3$. The gadget first compares each coordinate of x with the corresponding coordinate of p^i. Thus the $n + 1$ nodes on level $n + 1$ of the gadget reflect the value $d_H(x, p^i)$ from $d_H(x, p^i) = 0$ on the right (in this case $x = p^i$) to $d_H(x, p^i) = n$ on the left (both vectors differ in every coordinate).

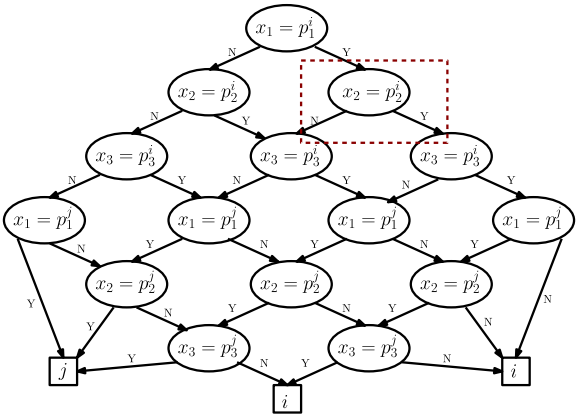

Figure 3: The gadget $G_{i,j}$ for $n = 3$. For a description of what each comparison node looks like, see Figure 4.

In the next n levels (starting at level $n + 1$), the gadget sequentially compares coordinates of x with the corresponding coordinates of p^j. The comparisons stop as soon as it is decided which of $d_H(x, p^i)$ and $d_H(x, p^j)$ is smaller (e.g. $d_H(x, p^i) = n$ and $x_1 = p_1^j$ already implies $d_H(x, p^i) > d_H(x, p^j)$ without testing the remaining coordinates). As soon as the gadget determines which of the two prototypes p^i, p^j is closer, it points either to the root node of a next gadget $G_{i,k}$ (if p^i is closer to x) or $G_{j,k}$ (if p^j is closer to x) for some k, or to one of the terminals $0, 1$ if the closest

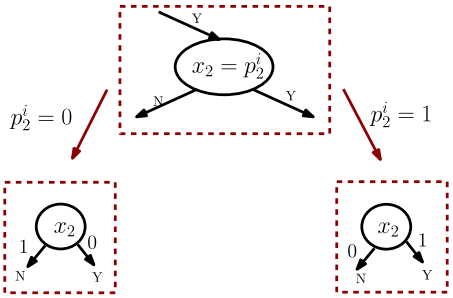

Figure 4: Decision nodes that correspond to the comparison node $x_2 = p_2^i$ if $p_2^i = 0$ (left) and if $p_2^i = 1$ (right).

prototype was already determined. In Figure 3 the corresponding directed edges go to the index of the closer prototype. The gadgets can break ties $d_H(x, p^i) = d_H(x, p^j)$ arbitrarily (tie is achieved at the two edges in the center of the bottom level), since the aim is to find one of the nearest prototypes, and those must all necessarily belong to the same class, by the definition of BNN.

Since our aim is to construct a BDD, we must convert the comparison nodes in the gadget to standard decision nodes. Each gadget is defined for fixed prototypes, so there is a natural way to convert the Y and N labels on the outgoing edges from a $x_k = p_k^i$ node into 0 and 1 labels from a decision node on variable x_k, as depicted in Figure 3 for $k = 2$.

It now remains to build the output BDD β using the gadgets. We can sequentially test pairs of prototypes until the closest one is found. We do so by making a rooted acyclic directed graph of gadgets with $k-1$ levels. At level i, prototype p^{i+1} is compared with every p^j for $j \leq i$, and directed edges are pointed to appropriate gadgets on the next level. After level $k-1$, a closest prototype has been determined, and so the directed edges going out of these gadgets point to terminal 1 (resp. 0) depending on whether the found closest prototype is positive (resp. negative). The output BDD β for a function with k prototypes is shown in Figure 5.

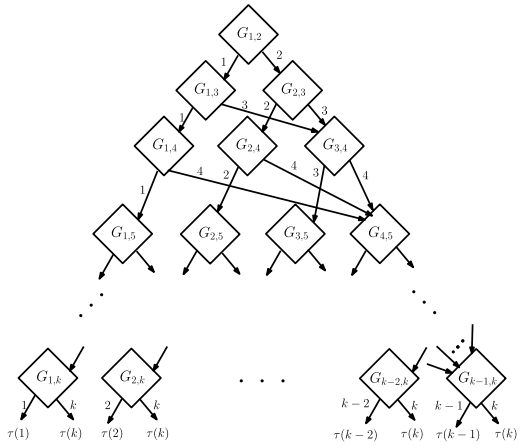

Figure 5: BDD of a function with k prototypes, where $\tau(i) = 1$ if prototype p^i is positive and $\tau(i) = 0$ if prototype p^i is negative.

We now show that the constructed BDD β is of polynomial size with respect to $|\alpha| = kn$. For vectors of length n, each gadget consists of $(n+1)^2 - 1 = O(n^2)$ decision nodes. As there are k prototypes in α, BDD β consists of $\frac{1}{2}(k-1)k = O(k^2)$ gadgets. Hence the size of the constructed BDD β is $O(n^2k^2) = O(|\alpha|^2)$, as desired.

To show that the inequality is strict, it suffices to consider the parity function f on n variables. It has a unique BNN representation with 2^n prototypes by Theorem 1. On the other hand, it is well-known Wegener (2000) that f admits a BDD representation of size $O(n)$. □

We now continue with proving the remaining two strict inequalities in Figure 2 which tie the **BNN** language to languages **MODS** (all models) and ¬**MODS** (all non-models).

Proposition 1. ***BNN*** $<$ ***MODS***.

Proof. We first show that **BNN** $\leq$ **MODS**. Consider a Boolean function f with a set of models $M \subseteq \mathcal{B}^n$ where $|M| = m$. Define $(P, N) = (M, \delta(M))$. We claim that (P, N) is a BNN representation of f with size polynomial in mn, which is the number of bits to store M.

Since all models of f are prototypes in P it suffices to check non-models to verify that (P, N) indeed represents f. Let x be an arbitrary vector such that $f(x) = 0$. If $x \in N$, then x is classified correctly. Consider $x \notin N$ (which implies $d_H(x, M) \geq 2$) and let $z \in P$ be a positive prototype closest to x. Any shortest path from x to z must pass through some $y \in \delta(M)$ and thus through a negative prototype which is closer to x than z which means that x is classified correctly.

In the worst case, all n neighbors of every model need to be picked as negative prototypes. Thus the constructed BNN (P, N) has at most $m + nm$ prototypes.

To see that the inequality is strict it suffices to consider the constant 1 function on n-dimensional vectors which has 2^n models but can be represented by a single positive prototype. □

An immediate consequence follows by an entirely symmetric argument.

Corollary 6. ***BNN*** $<$ ***¬MODS***.

Now we are ready to show that **BNN** is incomparable to both **CNF** and **DNF**.

Lemma 3. ***BNN*** $\not\leq$ ***CNF***.

Proof. It suffices to show that there is a Boolean function with CNF representation of size polynomial in the number of variables, which cannot be represented by a BNN of polynomial size. We will prove that the family of functions defined by

$$f_n(x_1, \ldots, x_n, y_1, \ldots, y_n) = \bigwedge_{i=1}^{n} (x_i \oplus y_i)$$

(where $\oplus$ is the XOR operator) has the desired property. Clearly, each f_n has a CNF representation F_n of size $O(n)$

$$F_n = \bigwedge_{i=1}^{n} (x_i \vee y_i) \wedge (\neg x_i \vee \neg y_i).$$

On the other hand, f_n has 2^n models (for each i we can decide whether to satisfy x_i or y_i) and every model of f_n is isolated (flipping a single bit falsifies the corresponding XOR). Therefore any BNN representation of f_n has at least 2^n prototypes by Lemma 1. It should be noted that this construction is a special case of the more general construction in DiCicco, Podolskii, and Reichman (2024) (Theorem 11). □

Lemma 4. ***CNF*** $\not\leq$ ***BNN***.

Proof. Consider the majority function MAJ_n on n variables which can be represented by a BNN of size $O(n)$ by Theorem 1. On the other hand, any CNF of this monotone function must contain at least as many clauses as is the number of its maximal false vectors Crama and Hammer (2011), and so the claim follows. □

Theorem 8. ***BNN*** *is incomparable with* ***CNF***.

Proof. We combine Lemma 3 and Lemma 4 and obtain the result. □

Corollary 7. ***BNN*** *is incomparable with* ***DNF***.

Proof. To show that **BNN** $\not\leq$ **DNF** consider the function $\neg f_n$ where f_n is as defined in the proof of Lemma 3. After an application of DeMorgan laws, we obtain a DNF formula of linear size representing the function $\neg f_n$. However, we may repeat the argument from Lemma 3 for the negative vectors of $\neg f$ which are all isolated and conclude that an exponential number of negative prototypes is required. To show that **DNF** $\not\leq$ **BNN** it again suffices to consider the majority function. Any DNF of this monotone function must contain at least as many clauses as is the number of minimal true vectors Crama and Hammer (2011). □

The last two relations for **BNN** which tie it to languages **IP** (all prime implicants) and **PI** (all prime implicates) already follow for free.

Corollary 8. ***BNN*** *is incomparable with both* ***IP*** *and* ***PI***.

Proof. Consider the function f_n and its CNF F_n from the proof of Lemma 3. Notice that no two clauses in F_n are resolvable. Hence, F_n is the PI representation of f_n (consisting of all prime implicates of f_n), and **BNN** $\not\leq$ **PI** follows. The opposite relation **PI** $\not\leq$ **BNN** follows from the fact that **CNN** $\not\leq$ **BNN** because **PI** $\subset$**CNF**. By symmetric arguments, the result can also be shown for the **IP** language. □

The following succinctness relation (which does not appear in Figure 2) also holds.

Proposition 2. ***BNN*** $\not\leq$ ***OBDD***.

Proof. It suffices to show that there is a Boolean function in n variables with OBDD representation of polynomial size in n which cannot be represented by a BNN of polynomial size in n. This property is satisfied by the threshold function $TH_n^{\lfloor n/3 \rfloor}$ which is true if and only if at least one third of its inputs are set to one. This is a symmetric Boolean function and it is long known that every such function can be represented by an OBDD of polynomial size in the number of variables Wegener (2000). On the other hand, $TH_n^{\lfloor n/3 \rfloor}$ can only be represented by a BNN of exponential size in n by Theorem 1. □

The above result also implies **BNN** $\not\leq$ **FBDD** since the **OBDD** language is a subset of the **FBDD** language. We conjecture that **FBDD** $\not\leq$ **BNN** also holds (which would imply **OBDD** $\not\leq$ **BNN**) but finding a family of functions necessary for such a statement remains an open problem.

6 Conclusions

We study the properties of the **BNN** language with respect to its position in the Knowledge Compilation Map. First, we study the complexity status of standard transformations and queries. Although the **BNN** language supports a decent subset of queries in polynomial time and hence it passes the necessary condition for a target compilation language formulated in Darwiche and Marquis (2002)[4], the lack of supported transformations makes the **BNN** language less appealing at this moment. However, we conjecture singleton conditioning and singleton forgetting to be polynomial time transformations, and if this conjecture turns out to be true, then it will be easier to argue that the **BNN** language is in fact a good target language for knowledge compilation. In such a case, designing a compiler into **BNN** may make sense. Although the complexity of queries and transformations for **BNN** is the main subject of this paper, we have also established several succinctness relations of this language to standard languages. The most interesting questions with respect to succinctness that remain open are the relations of **BNN** to **OBDD** and **FBDD**. We conjecture that in both cases the languages are incomparable. Another subject of future study are the succinctness relations of **BNN** to knowledge representation languages, which were added to KCM later, in particular to the **CARD**, **PBC**, **SDD**, and **SL** languages.

Acknowledgments

We are grateful to Petr Kučera for several helpful suggestions and for the construction that is used in the proof of Theorem 4.

References

Anthony, M., and Ratsaby, J. 2018. Large width nearest prototype classification on general distance spaces. *Theoretical Computer Science* 738.

Banerjee, S.; Bhore, S.; and Chitnis, R. 2018. Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In Bender, M. A.; Farach-Colton, M.; and Mosteiro, M. A., eds., *LATIN 2018: Theoretical Informatics*, 80–93. Cham: Springer International Publishing.

Berre, D. L.; Marquis, P.; Mengel, S.; and Wallon, R. 2018. Pseudo-Boolean Constraints from a Knowledge Representation Perspective. In *IJCAI*, 1891–1897. ijcai.org.

[4]Here we refer to the following sentence from Darwiche and Marquis (2002): "For a language to qualify as a target compilation language we will require that it permits a polytime clausal entailment test."

Boros, E.; Ibaraki, T.; and Makino, K. 1998. Error-Free and Best-Fit Extensions of Partially Defined Boolean Functions. *Information and Computation* 140(2):254–283.

Bova, S. 2016. SDDs are exponentially more succinct than OBDDs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, 929–935. Association for the Advancement of Artificial Intelligence.

Čepek, O., and Chromý, M. 2020. Properties of Switch-List Representations of Boolean Functions. *J. Artif. Intell. Res.* 69:501–529.

Čepek, O., and Hušek, R. 2017. Recognition of tractable DNFs representable by a constant number of intervals. *Discrete Optimization* 23:1–19.

Čepek, O.; Kronus, D.; and Kučera, P. 2008. Recognition of interval Boolean functions. *Annals of Mathematics and Artificial Intelligence* 52(1):1–24.

Čepek, O. 2022. Switch lists in the landscape of knowledge representation languages. In *FLAIRS*.

Crama, Y., and Hammer, P. L. 2011. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Crama, Y.; Hammer, P. L.; and Ibaraki, T. 1988. Cause-effect relationships and partially defined Boolean functions. *Annals of Operations Research* 16(1):299–325.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *J. Artif. Intell. Res.* 17:229–264.

Davis, M.; Logemann, G.; and Loveland, D. W. 1962. A machine program for theorem-proving. *Commun. ACM* 5(7):394–397.

DiCicco, M.; Podolskii, V.; and Reichman, D. 2024. Nearest Neighbor Complexity and Boolean Circuits. *Electron. Colloquium Comput. Complex.* TR24–025.

Gottlieb, L.-A.; Kontorovich, A.; and Nisnevitch, P. 2018. Near-optimal sample compression for nearest neighbors. *IEEE Transactions on Information Theory* 64(6):4120–4128.

Hajnal, P.; Liu, Z.; and Turán, G. 2022. Nearest neighbor representations of Boolean functions. *Inf. Comput.* 285(Part B):104879.

Kilic, K. M.; Sima, J.; and Bruck, J. 2023. On the information capacity of nearest neighbor representations. In *ISIT*, 1663–1668. IEEE.

Klenk, M.; Aha, D. W.; and Molineaux, M. 2011. The case for case-based transfer learning. *AI Mag.* 32(1):54–69.

Luxburg, U. v., and Bousquet, O. 2004. Distance–Based Classification with Lipschitz Functions. *J. Mach. Learn. Res.* 5:669–695.

Schieber, B.; Geist, D.; and Zaks, A. 2005. Computing the minimum DNF representation of Boolean functions defined by intervals. *Discrete Applied Mathematics* 149:154–173.

Wegener, I. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM.

Wilfong, G. 1991. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, 224–233. New York, NY, USA: Association for Computing Machinery.