TOWARDS CO-DISCOVERY BY EXTRACTING A SYMBOLIC REPRESENTATION FROM A NETWORK LAYER: COMPLEXITY, ALGORITHMS AND EXPERIMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learned models (DLMs) have been shown to be useful in a wide variety of situations. However, most XAI methods provide simple instance-level heatmap-style explanations that neither reveal the computation performed by the deep learner nor allow a human to modify the computation. Given that most DLMs comprise simple computational units, we explore extracting out a symbolic representation of the computation performed by the model as a formula in disjunctive normal form (DNF). We then show how the DNF can be used not only for explanation but also for simplification based on fairness/complexity/accuracy and even model editing. To achieve a symbolic representation, we restrict the layer's inputs to binary values often found in graphical data and some architectures. We explore three core co-discovery problems: extraction, simplification and editing, showing complexity results, algorithms and experimental results for each.

1 INTRODUCTION

Deep learned models (DLM) have been shown to be useful in a wide variety of settings (Goodfellow et al., 2016). However, they have one common limitation: they are essentially black boxes whose computational abilities are not well understood. Popular XAI methods such as LIME (Ribeiro et al., 2016) typically only produce instance-level explanations and even then just show a heatmap of which parts of the input the deep learner is using when making decisions (Belinkov, 2022). This falls far short of the human level explanation standards (Miller, 2019) which require understandable descriptions of the computation/reasoning that occurs. Given that most networks use simple computational units and considering a simplified data setting where inputs are binary, we explore whether more compelling and detailed explanations can be produced that a human can directly interact with and edit. We envision humans extracting out a layer's weights/computation in symbolic form, simplify-

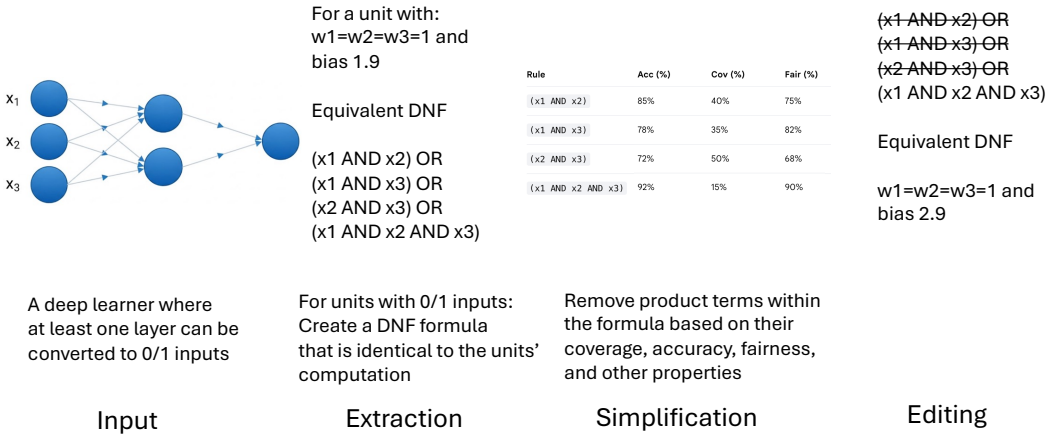


Figure 1: An overview of the core problems addressed in the paper

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing and even editing the symbolic representation, and then reinserting the equivalent weights into the DLM to facilitate what we term co-discovery.

Our approach extracts a DLM layer's computation as a formula in disjunctive normal form[1] (DNF). To achieve this, we limit ourselves to a simple but common binary data setting where some layer accepts only 0/1 as inputs which are processed by a network of rectified linear units (RELUs). Such data settings can model graphs that commonly occur in scientific data[2] and observational/categorical data common in social science settings (Scarselli et al., 2009). Some architectures such as self explanatory networks and binarized/quantized networks (Frosst & Hinton, 2017; Courbariaux et al., 2016; Qin et al., 2020) even allow 0/1 coding in the intermediate layers.

Illustrative Example. For simplicity, consider a single RELU unit with weights w_1, w_2 and w_3 all set to 1 and a bias value $b = 1.9$. Since the inputs to the neuron are limited to 0/1 values (which we denote by x_1, x_2 and x_3), a DNF formula (consisting of four product terms) equivalent to the RELU activating is $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$ since the weight sums of each product term is larger than the bias value 1.9. Note this simple approach can yield many product terms, with some rarely occurring in the data.

We make several core contributions which are diagrammatically shown in Figure 1.

- Extraction Problem: We formulate the DNF formula extraction problem from a RELU layer and show that extracting all product terms requires exponential time in the worst case. Even counting the number of appropriate product terms is computationally intractable; technically, it is **#P**-hard. We then show how to extract out a subset of highly probable product terms from a single neuron using a Gibbs sampler.
- Simplification Problem: Each product term can be measured by a variety of facets such as error, coverage, fairness and complexity[3]. The general problem of selecting a desirable subset of product terms (to obtain an accurate/representative/fairer/simpler DNF formula) is in general **NP**-hard. However, we develop a dynamic programming formulation for settings involving complexity.
- Editing Problem: We discuss how one can edit a DNF formula and then generate an equivalent RELU unit by constructing simple linear programs (LPs) to show that the problem is efficiently solvable; that is, it is in **P**.

We begin this paper by motivating DNF as our symbolic representation of a DLM. We then formulate the extraction, simplification and editing problems, show their complexity and discuss efficient approximations or other approaches to each problem. Finally, we perform experiments on several standard data sets and a challenging neuroscience data set.

2 WHY DNF EXTRACTION

We focus on DNF extraction for several reasons. Our work aims to explain the cognitive process of the DL in a language that allows interaction with a human expert. Our purpose in doing this is that the human can not only interpret the model easily but can edit it as needed; this follows the notion of co-discovery in DL where the human and machine work together.

Clearly, a DNF formula is easy to interpret for a human as it can be understood as a sum of simpler parts. Importantly, the human can edit the formula appropriately. For example, the domain expert may see some product terms and realize that they are inappropriate in that they are biased, not-novel or perhaps just inconsistent with existing domain knowledge. Conversely, the product terms in the DNF formula may trigger ideas for better combinations of the terms; creating new combinations may lead to deletion, combination or addition of product terms.

In addition, DNF formulae can be viewed as being closely related to concept theory (Murphy, 2002) which suggests how a human takes in raw data and processes it to make decisions. Its two main branches are exemplar theory and prototype theory. In the former, concepts are stored by exemplars/examples. For instance, the concept of a dog would be stored by a small but diverse collection of different types of dogs. In prototype theory, a concept is stored by a simple set of properties of the concept; for the concept of a dog, the properties may include items such as four legs, tail and head.

[1] A formula in disjunctive normal form consists of product terms (i.e., literals/inputs combined by AND) which are combined using OR.

[2] We explore a neuroscience dataset in our experiments.

[3] Here, "complexity" refers to the number of literals in a product term.

A DNF formula fits both these branches well. A single product term can be considered to define an exemplar and the literals in the product term define the properties of the concept. Extracting out the computation performed by a DL in a form similar to how a human stores/processes concepts has many advantages (Miller, 2019).

Example. Consider the application context of medical imaging where the input into the DL is a 0/1 connectivity matrix of brain regions and the aim is to predict a medical condition (such as Alzheimer's). Here co-discovery is important as there is a huge body of knowledge on the topic and any results the DL finds should be consistent with it; these requirements could be used to edit the model. Importantly, a DNF formula for each unit that is fed the 0/1 connectivity graph has a very natural interpretation. Each product term can be considered to be a subgraph and the entire DNF can be interpreted as the subgraphs/features extracted by the DL. Editing this DNF and generating new weights to reinsert into the network could be used to fix broken fundamental rules such as symmetry (e.g., the left and right pre-frontal cingular are always part of the same biomarker).

3 DNF Formula Extraction Problem

We begin by formalizing the problem of DNF extraction and show that extracting out all appropriate product terms for a single unit is intractable. We also show that even a counting version of the problem is computationally hard. However, extraction of all product terms is undesirable as they may contain rarely found product terms in the training data. To overcome this, we introduce a Markov Chain Monte Carlo (MCMC) sampler (Hastie et al., 2009) (a Gibbs sampler), that will sample highly probable product terms. This can be used to produce a smaller subset of highly probable (per the training data) product terms efficiently.

3.1 Problem Definition and Complexity

To present a precise definition of the DNF extraction problem and establish complexity results, we begin with some basic definitions.

Definition 3.1. ***Valid and Minimally Valid Product Terms for A Single Unit***

Let $\mathbb{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ be a set of N literals (of Boolean variables). Let $W[1..N]$ be a weight vector, with $W[i] \geq 0$ being the weight of literal ℓ_i, $1 \leq i \leq N$. Let $b > 0$ denote the bias.

*A subset $\mathbb{L}' \subseteq \mathbb{L}$ is a **valid product term** if the total weight of the literals in $\mathbb{L}'$, denoted by $W[\mathbb{L}']$,* exceeds *b; otherwise, $\mathbb{L}'$ is **invalid**.*

*A valid subset $\mathbb{L}'$ is **minimally valid** if for any literal $\ell_j \in \mathbb{L}'$, $W[\mathbb{L}'] - W[j] \leq b$. In other words, a subset $\mathbb{L}'$ of literals is minimally valid when $\mathbb{L}'$ is valid but removing any single literal makes it invalid. (Equivalently, $\mathbb{L}'$ is minimally valid if $\mathbb{L}'$ is valid but* every proper subset *of $\mathbb{L}'$ is invalid.)*

Example: Suppose we have three variables x_1, x_2 and x_3 with weight vector $W = (0.6, 0.5, 0.4)$ and bias $b = 0.9$. Here, the set $\mathbb{L}$ of literals is given by $\mathbb{L} = \{x_1, x_2, x_3\}$.

- Since the subset $\{x_1, x_2\}$ has a weight of $1.1 > 0.9$, it is valid. Removing either variable from the subset gets the weight below 0.9. Thus, $\{x_1, x_2\}$ is also a minimally valid set. Likewise, the subset $\{x_1, x_3\}$ is also minimally valid.
- The subset $\{x_2, x_3\}$ is invalid since its total weight 0.9 *doesn't* exceed the bias (0.9).
- The subset $\{x_1, x_2, x_3\}$ is valid as its total weight 1.5 exceeds the bias. However, this subset is not minimally valid since removing x_3 gives the subset $\{x_1, x_2\}$ which is valid.

Finding *one* minimally valid subset (if it exists) can be done efficiently; see Algorithm 2 in the Appendix (Section A). However, when we consider the following problem which requires listing all the minimally valid subsets, the situation changes significantly.

Problem 3.1. ***Listing All Minimally Valid Product Terms.***

GIVEN: *A set $\mathbb{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ of N literals (of Boolean variables), a weight vector $W[1..N]$, where $W[i] \geq 0$ is the weight of literal ℓ_i, $1 \leq i \leq N$, and a bias $b > 0$.*

REQUIREMENT: *List all the minimally valid subsets of $\mathbb{L}$.*

For this problem we have the following result.

Proposition 3.1. *Listing all the minimally valid subsets of literals requires exponential time in the worst case.* ***Proof:*** *See Appendix (Section B).*

In the Appendix (Section C) we also show that even the problem of just finding the number of valid subsets is computationally intractable (i.e., **#P**-hard) in general.

3.2 A Gibbs Sampler to Extract a Highly Probable DNF

Given the previous complexity results that point out the computational intractability regarding enumeration of valid product terms, we explore the extraction of highly probable product terms using a Markov Chain Monte Carlo (MCMC) sampler. Here, the event space is defined as the set of all possible product terms (conjunctions of literals). We wish to create a random walk on this event space so that the highly probable product terms are sampled accordingly.

The sampler begins by using Algorithm 2 to generate an initial state (i.e., one minimally valid product term). Then each literal in turn is sampled conditionally based on an empirical estimate calculated from the training data.

Algorithm 1: A Gibbs Sampler that Generates Propositions in Accordance With Their Probabilities in a Data Set

Input : D is the dataset of n instances each of m propositions (literals).
$MATCH$ function counts how often a proposition combination occurs in D.
Output: A probability distribution over all potential product terms.

Function GibbsSampler ($D, n, m, NumSamples, BurnIn$)
- Initialize array $State[1 \ldots m]$ using Algorithm 2
- Initialize array $Samples[1 \ldots 2^m - 1]$ to 0
- **for** $step = 1$ **to** $(BurnIn + NumSamples)$ **do**
  - **for** $i = 1$ **to** m **do**
    - $ConditionVars \leftarrow State - \{State[i]\}$
    - $Count_X_i \leftarrow$ Match$(D, X_i = 1 \wedge X_{\neq i} = ConditionVars)$
    - $Count_Condition \leftarrow$ Match$(D, X_{\neq i} = ConditionVars)$
    - $Prob_X_i \leftarrow (Count_X_i + 1)/(Count_Condition + 2)$ (Laplace smoothing fallback)
    - $u \sim$ Uniform$(0.0, 1.0)$
    - **if** $(u < Prob_X_i)$ **then** $State[i] \leftarrow 1$ **else** $State[i] \leftarrow 0$
    - **if** $(step > BurnIn)$ **then** $Samples[State]$ ++
  - **end**
- **end**
- **return** $Samples/NumSamples$

4 The DNF Simplification Problem

Even if we were to use our Gibbs sampler from the previous section we can still generate a DNF with hundreds of probable product terms for each unit. We may wish to further simplify/reduce this collection based on other properties such as fairness, coverage and accuracy.

Let the DNF formula Y to be simplified consist of a set of n product terms $\{Y_1, Y_2, \ldots, Y_n\}$ over a set of m Boolean literals/variables $X_1, X_2, \ldots, X_m$. We explore two ways of simplifying Y: (i) by removing some product terms, that is, by finding a subset $Y' \subset Y$ and (ii) by keeping all/most product terms but removing some literals from each of them; that is, retaining a small subset $X' \subset X$ of important literals. Under optimization objectives, we show that both approaches lead to computationally intractable problems in general. However, we provide efficient methods for special cases.

4.1 Removing Product Terms: Intractability and an Efficient Special Case

We now formulate the problem of DNF Simplification through Product Term Removal, which we abbreviate as DNFS-PTR.

Problem 4.1. ***DNFS-PTR***

INSTANCE: *A set $X = \{x_1, x_2, \ldots, x_m\}$ of m Boolean variables; a set $Y = \{Y_1, Y_2, \ldots, Y_n\}$ of n product terms formed using the literals of X; for each product term Y_i, a benefit β_i and weight c_i, $1 \leq i \leq n$; an upper bound of $\ell \leq n$ of product terms to be chosen; positive values β and γ.*

QUESTION: *Is there a subset $Y' \subseteq Y$ of product terms such that (i) $|Y'| \leq \ell$, (ii) the benefit of $Y' \geq \beta$ and (iii) the weight of $Y' \leq \gamma$?*

Here, benefit could be a measure of product terms' properties such as accuracy, fairness and coverage that we wish to maximize. Weight refers to an attribute that we wish to upper bound; this can also be the reciprocal of a measure such as error or accuracy. In our experiments, the weight of each product term Y_i is its complexity: the number of literals in Y_i. We begin with a remark about the complexity of this problem in general. We then mention a special case that can be solved efficiently.

From the formulation, one can readily see that the DNFS-PTR problem with $\ell = n$, corresponds to the KNAPSACK problem (Garey & Johnson, 1979): a subset of the n product terms must be added to the knapsack so that the weight of the chosen subset is at most γ and the benefit is at least β. Using this idea, it is easy to get a reduction from the KNAPSACK problem to DNFS-PTR. Since the KNAPSACK problem is **NP**-hard, we have the following observation.

Observation 4.1. *In general, DNFS-PTR is **NP**-hard.* ■

A Dynamic Programming Algorithm for DNF Simplification. A special case of the DNFS-PTR problem arises when the weight of each product term Y_i is measured by the number of literals in Y_i. As mentioned earlier, we use this weight measure in our experiments. If there are m variables, even though there are $2m$ possible literals, each product term may have at most m literals. The reason is that if a product term has $m + 1$ or more literals, it must include some variable and its complement; such a product term has the Boolean value 0 and so is not useful. Thus, the weight of any product term is an integer in the range 1 to m. Since our goal is to choose at most $\ell \leq n$ product terms, the weight of any solution is at most $m\ell$. In other words, we may assume that the weight bound γ is an integer satisfying the condition $0 \leq \gamma \leq m\ell$. This restriction allows us to use dynamic programming to obtain a polynomial algorithm for this special case of the DNFS-PTR problem.

The dynamic programming table that stores solutions to subproblems is a three-dimensional array B of size $n \times (\ell+1) \times (\gamma+1)$ with the following significance: each entry $B[i, j, k]$ stores the maximum possible benefit when exactly j product terms are chosen from the subset $\{Y_1, Y_2, \ldots, Y_i\}$ (i.e., from the first i product terms) with a total weight of at most k. Recall that the weight and benefit of product term Y_i are respectively c_i and β_i, $1 \leq i \leq n$.

For $i = 1$, the array B can be initialized as follows.

$$\begin{array}{lcll} B[1,0,k] & = & 0 & \text{for all } k,\ 0 \leq k \leq \gamma \\ B[1,1,k] & = & -\infty & \text{for all } k < c_1 \\ B[1,1,k] & = & \beta_1 & \text{for all } k \geq c_1 \end{array}$$

The equation for $B[1, 0, k]$ above holds because when no product term is chosen from $\{Y_1\}$, the benefit is zero, regardless of the weight. The equation for $B[1, 1, k]$ shows that it is infeasible (i.e., the benefit is $-\infty$) to choose the product term Y_1 at any weight *less than* c_1; however, for any weight $\geq c_1$, the benefit of choosing Y_1 is β_1.

Now, suppose for some $i \geq 2$, we have computed all the entries of B for the subset $\{Y_1, \ldots, Y_{i-1}\}$. We can compute the entries for the subset $\{Y_1, \ldots, Y_{i-1}, Y_i\}$ as follows.

$$\begin{array}{lcl} B[i,j,k] & = & \max\{B[i-1,j,k], B[i-1,j-1,k-c_i] + \beta_i\}, \\ & & \text{for all } j,\ 1 \leq k \leq \ell \text{ and for all } k,\ 0 \leq k \leq \gamma. \end{array}$$

In the above equation for $B[i, j, k]$, the first value $B[i - 1, j - 1, k]$ of the max operator represents the best benefit when product term Y_i is not chosen in the solution. The second value $B[i - 1, j - 1, k - c_i] + \beta_i$ shows that when product term Y_i is chosen, the solution has exactly $j - 1$ product terms from $\{Y_1, \ldots, Y_{i-1}\}$, the weight of the product terms chosen from $\{Y_1, \ldots, Y_{i-1}\}$ is at most $k - c_i$ and that the product term Y_i offers an additional benefit of β_i. (As is the standard practice, we use the convention that when an index of the array B falls outside the specified range, the value of such an entry is ignored and not used in the computation.) Once all the entries of B are available, the optimum benefit is given by $\max\{B[n, j, \gamma] : 1 \leq j \leq \ell\}$. A pseudocode description of the above dynamic programming approach is in the Appendix (Algorithm 3 in Section D).

To estimate the running time of this dynamic programming algorithm, note that the array B has a total of $n \times (\ell + 1) \times (\gamma + 1) = O(n\ell\gamma)$ entries and the equations shown above use $O(1)$ time to compute each entry. Thus, the running time of the algorithm is $O(n\ell\gamma)$. Since $\ell \leq n$ and $\gamma \leq m\ell$, the running time is polynomial in the size of the DNFS-PTR problem instance. We thus conclude:

Theorem 4.1. *When the weight of each product term is the number of literals in the product term, the DNFS-PTR problem can be solved in polynomial time.* ■

4.2 Removing Literals: Intractability Results and Approximation Algorithm

To allow a human to delete literals, we first identify an important set X' of literals that are crucial to the unit; in other words, only the literals *not* in X' may be deleted. The definition of the **Important Literal Discovery Problem** (ILDP) is as follows.

Problem 4.2. ***ILDP***

INSTANCE: *A set $X = \{x_1, x_2, \ldots, x_m\}$ of m Boolean variables, a positive integer $r \leq m$, a set $Y = \{Y_1, Y_2, \ldots, Y_n\}$ of product terms, where each Y_j is a conjunction (i.e., product term) of a subset of at least r variables from X, $1 \leq j \leq n$, integers k and ℓ, $1 \leq k, \ell \leq m$.*

QUESTION: *Is there subset $X' \subseteq X$ such that $|X'| \leq \ell$ and each product term Y_j contains at least k variables from X', $1 \leq j \leq n$?*

The following result establishes the complexity of ILDP. Our provable efficient approximation algorithm for ILDP is given in the Appendix (Section F).

Theorem 4.2. *The ILDP problem is **NP**-complete. **Proof:** See Appendix (Section E).*

Editing Operation	**Additional Constraint**
Product Term Addition ($Y' = Y \cup \{y_{n+1}\}$)	$W'^T \mathbf{x_{n+1}} > b'$
Product Term Deletion ($Y' = Y - \{y_i\}$)	$W'^T \mathbf{x_i} \leq b'$
Literal Addition ($X' = X \cup \{x_{m+1}\}$)	$W'^T \mathbf{x} + x_{m+1} w'_{m+1} > b'$
Literal Deletion ($X' = X - \{x_j\}$)	$W'^T \mathbf{x} > b' + w_j$

Table 1: Additional constraints to obtain suitable weights after editing a DNF formula. Let Y be the set of product terms which were extracted using weights W and bias b. Let the literals associated with the added/deleted term be defined by the binary vector $\mathbf{x_i}$ and the added literal's weight be w_j. W' and b' denote the new weight vector and bias to be computed after an editing step.

5 The DNF Editing Problem

The DNF extraction and simplification problems produce Y, a finite set of product terms. Here we explore the problem of editing Y based on human feedback to obtain Y' and then creating a new set of weights to reflect the edited DNF formula Y'. For example, the user may exclude product terms in Y that are biased, trivial or do not match human knowledge. Conversely, the existing product terms (concepts) may motivate new product terms to add.

The objective of the DNF Editing Problem is to compute the parameters (W', b') that realize a structurally modified product term set Y'. We explore four possible modifications below and show the corresponding linear constraints in Table 1.

- Product Term Addition: A new product term is added yielding $Y' = Y \cup \{y_{n+1}\}$.
- Product Term Deletion: A product term is deleted yielding $Y' = Y - \{y_i\}$.
- Literal Addition: A literal is added yielding $X' = X \cup \{x_{m+1}\}$.
- Literal Deletion: A literal is deleted yielding $X' = X - \{x_j\}$.

Efficient Product-Term/Literal Insertions and Deletions via Linear Programs

For an arbitrary product-term and/or literal addition/deletion, determining the existence of (W', b') can be formulated as a Linear Program (LP) as shown below. Since there are polynomial time algorithms for solving LPs (Kleinberg & Tardos, 2006), these editing problems can be solved efficiently.

Property	Mushroom	Breast Cancer	Healthy Aging
Five most probable product terms found	31.4% (93.2%)	7.4% (98.3%)	8.4% (93.1%)
Five most accurate product terms found	42.0% (53.1%)	13.1% (23.1%)	14.4% (25.9%)
Five simplest product terms found	16.1% (34.2%)	5.4% (14.5%)	2.6% (7.8%)

Table 2: Over 500 learned models, we compare how often the five most probable/accurate/simplest product terms are found in 100 product terms chosen randomly versus via our Gibbs sampler (Algorithm 1) reported second in parentheses. As expected, the Gibbs sampler yields significantly more popular and simpler terms, but surprisingly slightly higher accuracy.

Data Set	Coverage Of Training Set	Accuracy	Complexity: No. Literals
Mushroom	56.7% (80.6%)	96.5% (97.7%)	9.0 (4.6)
Breast Cancer	87.2% (95.2%)	63.9% (69.1%)	13.2 (6.3)
Healthy Aging	71.2% (95.6%)	64.3% (68.2%)	13.9 (6.9)

Table 3: Over 500 models, the accuracy, and complexity for 100 terms chosen by our Gibbs sampler, alongside 25 terms from Algorithm 3 reported second in parentheses. The simplification method yields slightly higher accuracy, but with significantly greater coverage and complexity.

Let $W' = (w'_1, w'_2, \ldots, w'_n)^T$ and b' be the real-valued decision variables to solve for. We wish them to satisfy several types of constraints. Each product term in the edited DNF (Y') should be minimally valid and the new weights W' should be similar to the previous weights (W). The corresponding LP is below in Equation (1). Note that W is a given constant vector, and hence the objective is linear.

$$
\begin{aligned}
\arg\max_{W'} \quad & W^T W' \\
\text{s.t.} \quad & \mathbf{x_i}^T W' \;\geq\; b' + \epsilon, \quad \forall y_i \in Y' && \text{(Validity)} \\
& \sum_{x_j \in Y_k \setminus \{x_i\}} w'_j \;\leq\; b', \quad \forall x_i \in Y_k, \quad \forall Y_k \in Y' && \text{(Minimality)} \\
& \textit{Appropriate edit constraints from Table 1} && \text{(Edits)} \\
\text{where} \quad & W' \in \mathbb{R}^n_{\geq 0},\; b' \in \mathbb{R}_{\geq 0},\;\; \epsilon > 0 \;\; \textit{is a small constant.}
\end{aligned}
\tag{1}
$$

6 Experimental Questions

Our experiments explore two sets of questions. For the extraction and simplification problems, we use three standard datasets to explore the properties of the extracted and simplified DNF formulae compared to an exhaustive enumeration of all minimal and valid product terms. Then we explore a challenging situation with a focus on DNF editing for co-discovery. Given the formal results, we do not provide experimental results surrounding efficiency. Our experimental questions are:

1. Does our Gibbs sampler for the DNF extraction problem (Algorithm 1) produce highly probable product terms compared to all valid and minimal product terms? (see Table 2).
2. Does applying our simplification algorithm (Algorithm 3) to the Gibbs sampler's output result in simpler but also accurate and well covered terms? (see Table 3).
3. How can we use the extraction, simplification and editing on a challenging real world neuroscience dataset where the input to the DL is a 0/1 brain connectivity graph? (see Table 4).

7 Experimental Methods and Results

7.1 For Questions 1 and 2

Datasets. We used 3 datasets with categorical features from the UC Irvine machine learning repository for each of our experiments: Mushroom, Breast Cancer and Healthy Aging.

Models. We use three RELU with input dimensions equivalent to the number of one-hot encoded features, with a binary classification output. DNF extraction is only performed on the first layer as the inputs are 0/1. During a forward pass:

1. **Non-negative weights:** The unconstrained weights are passed through a ReLU activation function to ensure that all effective weights are strictly non-negative.
2. **Non-negative bias:** The unconstrained bias is passed through a Softplus function and then negated to ensure the effective threshold is strictly non-negative.

These constraints ensure that the inclusion of an input feature (an input of 1) can only positively contribute to exceeding the activation threshold for classification.

Experiment Procedure. The training process occurs using 5-fold cross-validation to ensure robustness. The models were trained using the Adam optimizer with a learning rate of 0.01 and Binary Cross-Entropy loss. Following 20 epochs of training, a DNF was extracted using the Gibbs sampler.

Experiment Results. Table 2 shows that our Gibbs sampler produces across all three data sets far more probable product terms. Of the 5 most probable product terms (found from complete enumeration), our method discovered all of them in over 90% of the 500 trials. This of course is expected given it is the purpose of the sampler. Similarly, our sampler was able to find the 5 most accurate product terms often. Finally, the sampler finds the simplest (in terms of number of literals) product terms slightly more often.

Table 3 shows the results of post-processing the output of the Gibbs sampler using Algorithm 3 to select terms by trading off accuracy, simplicity and coverage. The method is able to trade-off the three by finding slightly more accurate DNF formulae with greater coverage but which are substantially less complex. This is particularly important for the DNF editing stage.

7.2 QUESTION 3

Dataset. The Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset (Mueller et al., 2005) provides comprehensive longitudinal neuroimaging, clinical, and biomarker data across normal aging, mild cognitive impairment (MCI), and Alzheimer's disease (AD). Predicting the former (a precursor to AD) is particularly challenging (Tzourio-Mazoyer et al., 2002).

Our dataset consists of approximately 500 individuals drawn primarily from ADNI-1 and ADNI-2. We represent each individual with a brain connectivity graph consisting of nodes from the AAL atlas (Tzourio-Mazoyer et al., 2002) and an edge between two nodes if and only if the absolute Pearson correlation of the BOLD (blood oxygenization level dependent) measurements between the two regions is greater than the 0.25. This provides a graph of 116 anatomical regions with over 13,000 potential edges/propositions which is the input into the DL.

Models. Our PDL consists of 4 layers (32 x 16 x 8 x 1) of RELU units whose weights are constrained to be positive and bias negative. Clearly there are more sophisticated networks (such as graph convolutional networks) that could have been used but we are still able to get predictive accuracy of over 80% at predicting AD and 60% at predicting MCI.

Experiment Procedure. The training process occurs using 10-fold cross-validation. The models were trained using the Adam optimizer with a learning rate of 0.01, and employ Binary Cross-Entropy with Logits Loss. Following 20 epochs of training, a DNF was extracted using the Gibbs sampler (Algorithm 1) and simplified using the dynamic program in Algorithm 3.

Each product term here can be interpreted as a subgraph and hence the DNF formula for a single unit is a collection of different subgraphs. For all subgraphs processed by all 32 units, we simulate a human editing them by looking for and correcting well known inconsistencies such as:

- Enforcing symmetry: For example, the Cerebral Regions consist of both left and right sides; if only one side appears in the extracted product term, we automatically add the other.
- Closing triangles: For example, if there are edges from Cingulate Post (CP) to Hippocampus (H) to ParaHippocampal (HP), then we add an edge from CP to HP.
- Ensuring connectivity: If product terms produce a graph that has more than one connected component, then we place an edge between two elements in different connected components with the highest absolute correlation to enforce a connected graph.

Applying these three rules to create Y' clearly produces a more complex DNF formula and we wish to determine how this affects accuracy.

Experimental Results. Table 4 shows our results. It is important to note that each of our problems: extraction, simplification and editing, effectively rewrite the weights of the 32 units connected to the input layer. For example, the extraction problem's use of a Gibbs sampler effectively uses only

Property	Extraction	Simplification	Editing
Total Number of Product Terms	153.4 (232.1)	121.6 (173.9)	156.5 (186.4)
Average Complexity of Product Term	1321.3 (1867.8)	308.2 (431.7)	478.9 (467.8)
Average Accuracy	88.1% (63.1%)	86.3% (65.3%)	93.1% (66.8%)

Table 4: We performed 10-fold cross validation on our ADNI data set 500 times to produce predictive results for AD and MCI (in parentheses). The average accuracy was 89.0% for AD and 63.3% for MCI. We then reused each of the networks but replaced the weights on the units from the input layer with weights from our three methods. We found extraction, simplification and editing methods changed the overall complexity of the subgraphs used as bio-markers and the predictive performance. Note that editing the DNF produced the highest predictive accuracy.

weights associated with product terms common in the data set. Similarly, the simplification problem further reduces those product terms (subgraphs) to those that are simple and accurate. Finally, our editing adds complexity to the weights by adding edges.

8 Related Work

The work most superficially similar to our own is *Deep learning repair* (Yu et al., 2021). This is an emerging area that aims at correcting unsafe, biased, or incorrect behaviors without the need for expensive, time-consuming retraining (Goldberger et al., 2021). This field is vital for safety-critical applications such as autonomous driving and flight control, where AI models must adhere to strict behavioral constraints. Early research focused on "provable repair," which guarantees that the model will mathematically satisfy specified safety conditions (Sotoudeh & Thakur, 2021). Though superficially similar, the motivation and implementation are fundamentally different. In particular, this work is not designed for human-in-the-loop style settings, so there is no need for any extracted insights from the DL to be interpretable and neither does the human directly edit the DL. Instead this work is a post-processing method to overcome critical flaws, not co-discovery.

Another related field is trying to extract out rules from DL (Andrews et al., 1995; Zilke et al., 2016). The most similar work in this area to our own is LogiX-GIN (Ragno et al., 2025), an inherently interpretable Graph Isomorphism Network (GIN) architecture. Their architecture directly incorporates logic layers alongside a specialized feature-binarization preprocessing layer. This configuration forces the network to automatically extract logic rules alongside its node and graph classifications. However, this work is *only* focused on XAI and does not attempt to simplify or edit the rules which are central to our aim of co-discovery.

9 Conclusion and Future Work

In this paper, we began to explore the novel area of co-discovery by introducing a framework to extract, simplify, and edit symbolic computations as disjunctive normal form (DNF) formulae from network layers restricted to binary inputs. We analyzed the computational complexity of these tasks, proving that full enumeration and general simplification are intractable, and proposed efficient practical solutions—including a Gibbs sampler for extracting highly probable product terms, a dynamic programming algorithm for simplification based on coverage, accuracy, and complexity, and linear programming formulations for model editing. Our experimental results across standard benchmark datasets and a challenging Alzheimer's disease neuroimaging application demonstrate that our approach helps bridge the gap between deep learning performance and human interpretability, enabling effective human-AI co-discovery.

Future research directions include exploring more sophisticated network architectures (such as graph convolutional networks), extending DNF extraction and editing to deeper network layers, and refining automated domain-specific editing rules to further improve biomarker discovery and clinical interpretability in complex biomedical applications.

REFERENCES

Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

Yonatan Belinkov. Probing classifiers: Hype, pitfalls, and what to do about them. *Computational Linguistics*, 48(4):1047–1052, 2022.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, Cambridge, MA, second edition, 2009.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freemand and Co., San Francisco, CA, 1979.

Yishai Goldberger, Eran Shoham, et al. Nnrepair: Constraint-based repair of neural network classifiers. In *Computer Aided Verification*, 2021. doi: 10.1007/978-3-030-81685-8_1.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, Reading, MA, 2006.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019. doi: 10.1016/j.artint.2018.07.007.

Susanne G. Mueller, Michael W. Weiner, Leon J. Thal, Ronald C. Petersen, Clifford R. Jack, William Jagust, John Q. Trojanowski, Arthur W. Toga, and Laurel Beckett. Ways toward an early diagnosis in alzheimer's disease: The alzheimer's disease neuroimaging initiative (adni). *Alzheimer's & Dementia*, 1(1):55–66, 2005. doi: 10.1016/j.jalz.2005.06.003.

Gregory L. Murphy. *The Big Book of Concepts*. MIT Press, 2002.

Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020. doi: 10.1016/j.patcog.2020.107281.

Alessio Ragno, Marc Plantevit, and Celine Robardet. On logic-based self-explainable graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Matthew Sotoudeh and Aditya Thakur. Provable repair of deep neural networks. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 590–603, 2021.

Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, Nathalie Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *NeuroImage*, 15(1):273–289, 2002. doi: 10.1006/nimg.2001.0978.

Vijay V. Vazirani. *Approximation Algorithms*. Springer, New York, NY, 2001.

Bing Yu, Hua Qi, Qing Guo, Felix Juefei-Xu, Ming Xue, Lei Ma, and Jianjun Zhao. Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment. *IEEE Transactions on Reliability*, 71(4):1401–1416, 2021. doi: 10.1109/TR.2021.3096332.

Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. Deepred – rule extraction from deep neural networks. In *Discovery Science*, pp. 457–473. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46307-0_29.

A Finding a Minimally Valid Product Term

Algorithm 2 finds a minimally valid subset of literals (i.e., a product term) if one exists. It is a simple greedy algorithm that starts with all the literals and repeatedly deletes literals until reaching a subset from which no literal can be deleted without violating validity. It is easy to see that the algorithm runs in $O(N)$ time, where N is the initial number of literals.

Algorithm 2: Finding a minimally valid product term

Input : A set $\mathbb{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ of N literals; a weight vector $W[1..N]$, where $W[i] \geq 0$ is the weight of ℓ_i, $1 \leq i \leq N$; a bias value $b > 0$.
Output: A minimally valid subset $\mathbb{L}'$ of $\mathbb{L}$, if one exists.

Let CurrentTotal = $\sum_{i=1}^{N} W[i]$ and $\mathbb{L}' = \mathbb{L}$.
if $(\text{CurrentTotal} \leq b)$ **then**
  print "No valid subset" and **stop**.
end
for $(i = 1 \text{ to } N)$ **do**
  temp = CurrentTotal $-W[i]$.
  if $(\text{temp} > b)$ **then**
    Delete literal ℓ_i from $\mathbb{L}'$.
    CurrentTotal = temp.
  end
end
Return $\mathbb{L}'$.

B Listing All Minimally Valid Subsets: Statement and Proof of Proposition 3.1

Statement of Proposition 3.1: Listing all the minimally valid subsets of literals requires exponential time in the worst case.

Proof: We show that there are problem instances where the number of minimally valid subsets is exponentially large. To construct one such instance, let $\mathbb{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ be a set of N literals, where N is even. Let each literal have a weight of 1 and let the bias be $N/2 - 1$. In this case, it can be seen that every subset of $\mathbb{L}$ with $N/2$ literals is a minimally valid subset. Hence, for this example, the number of minimally valid subsets is given by the binomial coefficient $\binom{N}{N/2}$. It is well known that for any k, $1 \leq k \leq N$, $\binom{N}{k} \geq (N/k)^k$ (Cormen et al., 2009). Thus, $\binom{N}{N/2} \geq 2^{N/2}$, which is exponential in N. The proposition follows. ■

C The Complexity of Counting the Number of Valid Subsets

Proposition 3.1 shows that listing all minimally valid subsets is intractable. Here, we explore whether simplifying the problem can lead to an efficient algorithm. Unfortunately this is not the case.

Suppose we ignore the minimality requirement and simply want to *find the number of valid subsets*. Note that the goal here is to merely compute the number of valid subsets but *not* to list all of them. We will now show that this counting problem is also computationally intractable. Formally, we show that the problem is **#P**-hard; in particular, it is at least as hard as the problem of counting the number of solutions to the Subset-Sum problem (Garey & Johnson, 1979). The definition of this counting problem associated with Subset-Sum, which we denote by #Subset-Sum, is as follows: given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n positive integers and another integer B, find the number of subsets of X whose sum is equal to B. The following is a well known result (Garey & Johnson, 1979).

Theorem C.1. *The #Subset-Sum problem is **#P**-hard.*

Thus, unless the complexity classes **#P** and **P** coincide, there is no efficient algorithm for the #SUBSET-SUM problem. We can show a similar result for the problem of counting valid subsets.

Proposition C.1. *The problem of counting the number of valid product terms is **#P**-hard.*

Proof: The idea is to show that if there is an efficient algorithm for counting the number of valid subsets, then #SUBSET-SUM can also be solved efficiently.

Suppose $\mathcal{A}$ is an efficient algorithm for counting the number of valid subsets. Consider an arbitrary instance of the SUBSET-SUM problem specified by the set $X = \{x_1, x_2, \ldots, x_n\}$ of n positive integers and another positive integer B. We construct two instances I_1 and I_2 of the valid subset problem as follows. For both instances, we have a literal ℓ_i for each integer x_i, $1 \leq i \leq n$, and the weight vector W is such that $W[i] = x_i$, $1 \leq i \leq n$. For the instance I_1, the bias b_1 is chosen as $B - 1$ and for the instance I_2, the bias b_2 is chosen as B. When we run Algorithm $\mathcal{A}$ on I_1 and I_2, let the answers returned by the algorithm be η_1 and η_2 respectively. We have the following claim.

Claim: The number of solutions to the counting problem associated with SUBSET-SUM is $\eta_1 - \eta_2$.

Proof of Claim: Note that all the weights are positive integers. Thus, η_1 counts the number of valid subsets whose weight is at least B. This is the number of subsets of the SUBSET-SUM instance with sum at least B. Likewise, η_2 counts the number of subsets of the SUBSET-SUM instance with sum at least $B + 1$. Again, since the set A consists of positive integers, the quantity $\eta_1 - \eta_2$ gives exactly the number of subsets whose sum is equal to B. The claim follows.

Thus, if we have a polynomial time algorithm for counting the number of valid subsets, we can solve the counting problem associated with SUBSET-SUM in polynomial time. It follows from Theorem C.1 that the problem of counting the number of valid subsets is **#P**-hard. ■

D PSEUDOCODE DESCRIPTION OF THE DYNAMIC PROGRAMMING ALGORITHM FOR DNFS-PTR

The discussion on a dynamic programming algorithm for the special case of DNFS-PTR where the weight of each product term is equal to the number of literals in the product term was given in Section 4.1. A pseudocode description of this algorithm is given below as Algorithm 3. Please refer to Section 4.1 for the significance of the array B used in this pseudocode description.

E REMOVING LITERALS: STATEMENT AND PROOF OF THEOREM 4.2

Statement of Theorem 4.2: The ILDP problem is **NP**-complete.

Proof: We will actually show that the ILDP problem is **NP**-complete. even when $r = 2$ and $k = 1$.

It is easy to see that ILDP is in **NP** since one can guess a subset X' and verify in polynomial time that $|X'| \leq \ell$ and that each product term Y_j contains at least k variables from X'. We prove **NP**-hardness by a reduction from the **Minimum Vertex Cover** (MVC) problem: given an undirected graph $G(V, E)$ and an integer $\beta \leq |V|$, is there a subset $V' \subseteq V$ such that $|V'| \leq \beta$ and for each edge $\{u, v\} \in E$, at least of the nodes u and v is in V'? The MVC problem is known to be **NP**-complete (Garey & Johnson, 1979). The reduction from MVC to ILDP is as follows.

(a) Let $V = \{v_1, v_2, \ldots, v_m\}$. For each node $v_i \in V$, we create a Boolean variable x_i, $1 \leq i \leq m$.

(b) For each edge $e_j = \{v_a, v_b\} \in E$, we create a product term $Y_j = x_a \wedge x_b$ using the Boolean variables corresponding to the nodes in e_j. (Thus each product term has exactly two variables; i.e., $r = 2$.)

(c) We set $\ell = \beta$ and $k = 1$.

This completes the reduction. It is easy to see that the reduction can be done in polynomial time.

Suppose G has a vertex cover V' with at most β vertices. Construct a subset X' of X as follows. X' is initially empty. For each node $v_i \in V'$, add the variable x_i to X'. Thus $|X'| \leq \beta = \ell$. Consider any product term $Y_j = x_a \wedge x_b$. Since the product term Y_j corresponds to the edge $e_j = \{v_a, v_b\}$,

Algorithm 3: Outline for the Dynamic Programming Algorithm for DNFS-PTR when the weight of each product term is the number of literals in it.

Input : A set $Y = \{Y_1, Y_2, \ldots, Y_n\}$, a benefit $\beta_i \geq 0$ and weight c_i for product term Y_i, $1 \leq i \leq n$, an integer $\ell \leq n$ and positive integer γ. (The weight c_i is the number of literals in Y_i.)

Output: MaxBenefit, the best benefit value obtainable from a subset of Y consisting of at most ℓ product terms and whose total weight is at most γ. (Please refer to Section 4.1 for the significance of the array B used in this pseudocode description.)

```
/* Initializing appropriate entries of the B array. */
for k = 0 to γ do
    Set B[1, 0, k] to 0.
    if (k < c_1) set B[1, 1, k] to −∞ else set B[1, 1, k] to β_1.
end
/* Computing the remaining entries of the B array. */
for i = 2 to n do
    for j = 1 to ℓ do
        for k = 0 to γ do
            B[i, j, k] = max{B[i − 1, j, k], B[i − 1, j − 1, k − c_i] + β_i}.
        end
    end
end
/* Computing the maximum benefit value. */
MaxBenefit = B[n, 1, γ].
for j = 2 to ℓ do
    if (MaxBenefit < B[n, j, γ]) then MaxBenefit = B[n, j, γ].
end
Output MaxBenefit.
```

and V' contains at least one of v_a and v_b, it follows that X' contains at least one of the variables x_a and x_b. Thus, X' is a solution to the ILDP instance.

For the converse, suppose there is a solution X' with at most β variables to the ILDP problem. Construct a subset V' of V as follows. V' is initially empty. For each variable $x_i \in X'$, add the node v_i to V'. Thus $|V'| \leq \beta = \ell$. Consider any edge $e_j = \{v_a, v_b\}$ of G. Since the product term $Y_j = x_a \wedge x_b$ corresponds to the edge e_j and X' contains at least one of x_a and x_b, it follows that V' contains at least one of the nodes v_a and v_b. Thus, V' is a solution to the MVC instance, and this completes the proof. ■

F An Approximation Algorithm for the ILDP Problem

This section provides an efficient approximation algorithm for the ILDP problem with a provable performance guarantee. In particular, the size of the solution set X' found by the approximation algorithm is within the factor $(1 + \ln p)$ of an optimal solution, where p is the maximum number of literals in any product term.

Idea behind the approximation algorithm: Our approximation algorithm relies on the following **Set Multi-Cover** (SMC) problem, which is well known in combinatorial optimization (Vazirani, 2001).

Set Multi-Cover (SMC)

INSTANCE: A universe $U = \{u_1, u_2, \ldots, u_n\}$ of n elements, a collection $C = \{C_1, C_2, \ldots, C_m\}$, where each C_j is a subset of U ($1 \leq j \leq m$) and a positive integer λ_i (the coverage requirement) for each element $u_i \in U$ ($1 \leq i \leq n$).

REQUIREMENT: A subcollection $C' \subseteq C$ with a *minimum* number of sets such that each element $u_i \in U$ appears in at least λ_i subsets in C', $1 \leq i \leq n$.

The SMC problem is **NP**-hard since it is a generalization of the standard Minimum Set Cover (MSC) problem in which the coverage requirement $\lambda_i = 1$ for each element $u_i \in U$.

We first show that the ILDP problem can be reduced to the SMC problem in an *approximation-preserving* manner. This allows us to use any approximation algorithm that provides a performance guarantee of ρ for the SMC problem as an approximation algorithm with the same performance guarantee for ILDP. The following proposition presents the details of the reduction from ILDP to SMC.

Proposition F.1. *The ILDP problem can be reduced to the SMC problem in an approximation-preserving manner; that is, the reduction ensures that any solution that uses q sets for the SMC problem can be efficiently converted into a solution that uses q variables for the ILDP problem and vice versa.*

Proof: Consider any instance of the ILDP problem given by the set $X = \{x_1, x_2, \ldots, x_m\}$ of m Boolean variables, a positive integer $r \leq m$, a set $Y = \{Y_1, Y_2, \ldots, Y_n\}$ of product terms, where each Y_j is a conjunction of a subset of at least r variables from X ($1 \leq j \leq n$) and integer k. We assume that each product term Y_j contains at least k variables; otherwise, there is no solution to the ILDP problem. Recall that in the ILDP problem, we are required to choose a subset X' of X such that each product term Y_j has at least k variables from X'. The reduction to the SMC problem is as follows.

1. For each product term Y_j, we construct an element u_j of the universal set U of the SMC instances, $1 \leq j \leq n$.
2. For each variable x_i of the ILDP instance, we create a set C_i ($1 \leq i \leq m$) as follows. C_i is initially empty. If x_i occurs in product term Y_j, then we add u_j to C_i, $1 \leq j \leq m$.
3. For each set C_i, the coverage requirement λ_i is set to the value k given in the ILDP instance.

This completes the reduction which can obviously be done in polynomial time. We now show that any solution that uses q sets for the SMC problem can be efficiently converted into a solution that uses q variables for the ILDP problem and vice versa.

Suppose there is a solution that uses q sets for the SMC problem. Let $C' = \{C_{i_1}, C_{i_2}, \ldots, C_{i_q}\}$ denote the solution. Consider the subset $X' = \{x_{i_1}, x_{i_2}, \ldots, x_{i_q}\}$ of variables of the ILDP instance. We claim that X' contains at least k variables form each product term Y_j. To see this, note that the element u_j of U corresponding to Y_j occurs in k or more subsets in C'. Each such subset $C_{i_p} \in C'$ corresponds to a variable that occurs in Y_j. Thus, at least k of the variables in Y_j are in X'. In other words, we have a solution with q variables for ILDP.

Now, suppose there is a solution $X' = \{x_{i_1}, x_{i_2}, \ldots, x_{i_q}\}$ consisting of q variables for ILDP. Consider the subcollection $C' = \{C_{i_1}, C_{i_2}, \ldots, C_{i_q}\}$ of C. We claim that each element $u_i \in U$ occurs in at least k sets in C'. To see this, note that each product term Y_i represented by u_i contains at least k variables from X'. Each such variable $x_{i_p} \in X'$ corresponds to the set C_{i_p} in C'. In other words, the element u_i occurs in at least k subsets in C', and this completes the proof of Proposition F.1. ■

A greedy heuristic for SMC: A well known greedy heuristic (see for example, (Vazirani, 2001)) for SMC proceeds as follows. Initially, the required solution C' and the set of elements whose coverage requirements have been met are both empty. In each iteration, the heuristic picks a set C_i from C such that among the sets in C, C_i has the maximum number of elements whose coverage requirement has not yet been met. (Ties are broken arbitrarily.) The chosen set C_i is added to the solution set C' and removed from C. Iterations are continued until the coverage requirement for every element is satisfied. The following result, which establishes the performance guarantee of the heuristic, is given in (Vazirani, 2001).

Theorem F.1. *The greedy heuristic for the SMC problem provides a performance guarantee of $1 + \ln(f)$, where f is the size of the largest set in the input.* ■

The following result states the performance guarantee that can be obtained for the ILDP problem.

Theorem F.2. *There is a polynomial time approximation algorithm for the ILDP problem with a performance guarantee of $1 + \ln(p)$, where p is the size of (i.e., the number of variables in) the largest product term in the input.*

Proof: Our approximation algorithm for ILDP first reduces the problem to SMC and then uses the known greedy heuristic for SMC. In the reduction from ILDP to SMC, the size of a largest set in the

SMC instance is the maximum number of literals p in any product term. Thus, Theorem F.2 follows directly from Proposition F.1 and Theorem F.1. ■

Our approximation algorithm resulting from the above discussion is outlined in Algorithm 4.

Algorithm 4: Greedy Heuristic for ILDP

Input : A set $X = \{x_1, x_2, \ldots, x_m\}$ of m Boolean variables, a set $Y = \{Y_1, Y_2, \ldots, Y_n\}$ of product terms (where each product term Y_i has a subset of at least r variables from X), an integer $k \leq r$.

Output: A subset X' of X such that each product term Y_i has at least k variables from X' and $|X'|$ is as small as possible.

Let $X' = D = \emptyset$. (D is the set of product terms which have at least k variables from X'.)

while $(D \neq Y)$ **do**

  Find a variable $x_j \in X$ such that among the variables in X, x_j occurs in the maximum number of product terms in $Y - D$ (i.e., product terms which have $< k$ variables from X'.)

  If a product term Y_i has k or more variables from X', then add Y_i to D.

  Add x_j to X' and delete x_j from X.

  Remove from Y each product term which has at least k variables from X'.

end

Output the set X'.