Open Vocabulary Compositional Explanations for Neurons

Biagio La Rosa
Department of Computer Science and Engineering
University of California, Santa Cruz
bilarosa@ucsc.edu

Leilani H. Gilpin
Department of Computer Science and Engineering
University of California, Santa Cruz
lgilpin@ucsc.edu

Abstract

While neurons are the basic units of deep neural networks, it is still unclear what they learn and if their knowledge is aligned with that of humans. Compositional explanations aim to answer this question by describing the spatial alignment between neuron activations and concepts through logical rules. These logical descriptions are typically computed via a search over all possible concept combinations. Since computing the alignment over the entire state space is computationally infeasible, the literature commonly adopts beam search to restrict the space. However, beam search cannot provide any theoretical guarantees of optimality, and it remains unclear how close current explanations are to the true optimum. In this theoretical paper, we address this gap by introducing the first framework for computing guaranteed optimal compositional explanations. Specifically, we propose: (i) a decomposition that identifies the factors influencing the alignment, (ii) a heuristic to estimate alignment at any stage of the search, and (iii) the first algorithm that can compute optimal compositional explanations within a feasible time. Using this framework, we analyze the differences between optimal and non-optimal explanations and demonstrate that 10–40% of explanations obtained with beam search are suboptimal when overlapping concepts are involved. Finally, we evaluate a beam-search variant guided by our proposed decomposition and heuristic, showing that it matches or improves runtime over prior methods while offering greater flexibility in hyperparameters and computational resources.

1 Introduction

Compositional explanations [14] are a method for interpreting how individual units or neurons contribute to spatial alignment between input features and higher-level representations. The key idea is capturing the interaction between low-level (e.g., colors, textures, shapes) and high-level concepts (e.g., objects, entities) via propositional logic formulas able to express the alignment between these complex relationships and neuron activations.

One of the key problems to achieve this goal is that the full search space encompassing all of the possible combinations between concepts cannot typically be exhaustively explored due to its size. As a result, prior work has relied on beam search to identify plausible alignments [14]. Although the resulting explanations are valid, it is unclear whether these explanations are, in fact, optimal. While beam search does not guarantee optimality, it might converge to it due to unstudied or unknown

Preprint. Under review.

properties of the underlying datasets. If not, the explanations produced by beam search may represent only a subset of the alignment structure, offering a partial view of neuron behavior.

The main contribution of this work is to make navigating the state space tractable. To achieve this, we propose a decomposition of the Intersection-over-Union (IoU) metric that reveals a set of fundamental quantities governing alignment quality, and we design a heuristic and a corresponding algorithm that jointly reduce the size of the state space and guide the search process. This approach enables the computation of optimal compositional explanations in feasible time. To the best of our knowledge, this is the first attempt in this research direction. As a first step, we apply our method to the computer vision domain, given its prominence in compositional explanation research.

Our contributions are as follows:

- We propose a decomposed Intersection over Union score (dIoU) that identifies fundamental quantities for alignment quality and enables a better characterization of the impact of logical operators on spatial alignment.
- We design a heuristic and a corresponding algorithm that jointly reduce the size of the state space and guide the search process. We show that this algorithm computes guaranteed optimal explanations in feasible time and we analyze the differences between optimal and non-optimal explanations.
- We show that part of our proposed heuristic can be used directly to guide beam search with significant gains in flexibility and competitive or better performance than competitors. Specifically, our variant scales more effectively than competitors with respect to explanation length and beam size, and it is less resource-intensive and easier to parallelize.

In Section 2, we give an introduction to the relevant literature in the background and specify our framework for optimal compositional explanations. In Section 3, we analyze our contribution.

2 Optimal Compositional Explanations

2.1 Background and Related Work

Neuron explanations aim to understand what individual neurons learn during the training process. Different categories of methods have been proposed to decode different behaviors. Among the most popular in computer vision, we can cite the one that generates samples that capture features recognized by a neuron [5, 18, 15] and the ones that generate textual descriptions that correlate neuron activations and samples associated with a given concept [8, 16**? ? ?** , 17] through foundational models.

Differently from them, compositional explanations are a family of neuron explanations that specifically focus on the **spatial alignment** between a neuron activation range and concepts and express them through propositional logic formulas. The seminal work in this area is Network Dissection [1, 2], which associates each neuron with the single concept that maximizes this alignment. This approach was extended by Mu and Andreas [14] to associate relationships between multiple concepts, in an attempt to capture a higher degree of polysemantic behavior [4]. Relationships explored in the literature include co-occurrence [1], exclusion [14], relative position [6], and hierarchy [13].

Formally, let $\mathfrak{D} = \{x_1, x_2, ..., x_n\}$ be a probing dataset. Let z be a neuron to be explained in a probed model, and let d be both the dimension of its activations and the dimensionality of each sample in $\mathfrak{D}$[1]. Let $\mathfrak{L}^1$ be the concept set used to annotate samples in $\mathfrak{D}$, and $\mathfrak{L}^n$ be the set of all possible logical formulas of arity at maximum n between concepts in $\mathfrak{L}^1$. Compositional explanations aim to assign to z the logical combination $L \in \mathfrak{L}^n$ of concepts in $\mathfrak{L}^1$ (e.g., ((Cat OR Car) AND White)) that maximizes the alignment between the localization of a given neuron's activation range $[\tau_1, \tau_2]$ and the localization of the concepts within the probing dataset.

To reduce the search space, compositional explanations typically make two assumptions: concepts in the explanation are distinct (**Assumption 1**), and concepts are combined incrementally to form labels (e.g., $(((((A \oplus B) \oplus C) \ldots) \oplus Z)$)(**Assumption 2**), where $\oplus$ indicates a logical connective. Even with these assumptions, $\mathfrak{L}^n$ is too large to be explored exhaustively. The total number of combinations

[1] In the literature, this is typically obtained by upscaling/downscaling the activations.

is $\sum_{k=1}^{n} n_o^{k-1} \prod_k (|\mathfrak{L}^1| - k)$, where n_o is the number of logic connectives, and each combination requires the comparison of $2d$ values to compute the alignment. In the settings considered by Mu and Andreas [14], this leads to 2.8×10^{14} operations, rendering both storage and runtime infeasible. To cope with this, prior work adopts vanilla [14, 6, 13, 12] or informed [11] beam search with a small beam size. However, beam search does not guarantee optimality, leaving open the question of whether the resulting compositional explanations are the best possible or if better-aligned explanations exist. In the following, we characterize the fundamental quantities that influence alignment and propose both a heuristic and an algorithm that guarantees the optimality of explanations.

2.2 Fundamental Quantities

This section introduces our decomposed Intersection over Union score (dIoU) and the corresponding fundamental quantities. Alongside the assumptions introduced earlier, we adopt the following:

Assumption 3. *The logic operators connecting concepts are 00-preserving (i.e., they cannot produce a 1 from two zeros).* [2]

Terminology and Notation We use the term "the neuron fires" to indicate when its output lies within a specified activation range (e.g., top activations) and use $|\cdot|$ to indicate the cardinality of a set. According to previous literature, we define the *Concept Tensor* $\mathbf{M}^{\mathfrak{L}^1} \in \{0,1\}^{|\mathfrak{L}^1| \times |\mathfrak{D}| \times d}$ as the binary tensor corresponding to the localization of each concept within the dataset samples. From the Concept Tensor, we can extract, for each concept k, the *Concept Matrix* $\boldsymbol{M}_k \in 0, 1^{|\mathfrak{D}| \times d}$. Similarly, we define the *Neuron Activation Matrix* $\boldsymbol{N} \in 0, 1^{|\mathfrak{D}| \times d}$ as the binary matrix indicating the locations where the neuron fires within the dataset samples. Given the above notations, we propose the definitions of the following quantities.

Definition 2.1. We define the set U of *unique elements* of $\mathfrak{D}$ as the set of locations associated with exactly one annotation, and the set C of *common elements* of $\mathfrak{D}$ as the set of locations associated with multiple annotations.

$$U = \{(x,j) \mid \exists! k \in \mathfrak{L}^1 \text{ s.t. } x \in \mathfrak{D}, j \in [d], \mathbf{M}^{\mathfrak{L}^1}[k,x,j] = 1$$

$$C = \{(x,j) \mid \exists k_1, k_2 \in \mathfrak{L}^1 \text{ s.t. } x \in \mathfrak{D}, j \in [d], k_1 \neq k_2, \mathbf{M}^{\mathfrak{L}^1}[k_1,x,j] = 1, \mathbf{M}^{\mathfrak{L}^1}[k_2,x,j] = 1$$

Definition 2.2. Given a neuron activation matrix $\boldsymbol{N}$, we define the *unique activation* set N^U as the set of locations where the neuron fires on unique elements, and the *common activations* N^C as the set of locations where the neuron fires on common elements.

$$N^U = \{(x,j) \mid (x,j) \in U, \boldsymbol{N}[x,j] = 1\}$$
$$N^C = \{(x,j) \mid (x,j) \in C, \boldsymbol{N}[x,j] = 1\}$$

Definition 2.3. Given a neuron activation matrix $\boldsymbol{N}$ and a concept $k \in \mathfrak{L}^1$, we define the *unique intersection* set I^U as the set of unique elements that are annotated with k and where the neuron fires, and the *common intersection* set I^C as the set of common elements in annotated with k where the neuron fires.

$$I^U(k) = \{(x,j) \mid \boldsymbol{M}_k[x,j] = 1, (x,j) \in N^U\}$$
$$I^C(k) = \{(x,j) \mid \boldsymbol{M}_k[x,j] = 1, (x,j) \in N^C\}$$

Definition 2.4. Given a neuron activation matrix $\boldsymbol{N}$ and a concept $k \in \mathfrak{L}^1$, we define the *unique extras* set E^U as the set of all unique elements in $\mathfrak{D}$ annotated with the concept k and where the neuron does not fire, and the *common extras* set E^C as the set of all common elements in $\mathfrak{D}$ annotated with the concept k where the neuron does not fire.

$$E^U(k) = \{(x,j) \mid \boldsymbol{M}_k[x,j] = 1, (x,j) \in U, (x,j) \notin N^U\}$$
$$E^C(k) = \{(x,j) \mid \boldsymbol{M}_k[x,j] = 1, (x,j) \in C, (x,j) \notin N^C\}$$

Definition 2.3 and Definition 2.4 can be generalized to the case of label $L \in \mathfrak{L}^{\mathfrak{n}}$. In this case, the binary label matrix $\boldsymbol{M}_L$ is obtained as the result of the bitwise logic operations induced by the logic operators connecting single concepts $k \in L$.

[2] This assumption has been implicitly adopted in prior literature, where the commonly used bitwise OR, AND, and AND NOT operators all satisfy the 00-preserving property.

Table 1: Visualization of identified quantities for a sample x. In pink the unique extras. In yellow the common extras. In green the unique intersection. In cyan the common intersection.

	Vector						$dIoU$
N(x)	1	1	1	0	0	0	
$\boldsymbol{M}_{c_1}[x]$	1	1	0	0	1	1	2/5
$\boldsymbol{M}_{c_2}[x]$	1	1	0	1	0	0	2/4
$\boldsymbol{M}_{c_3}[x]$	1	0	1	0	1	1	2/5

Theorem 2.1. Given a binary neuron activation matrix $\boldsymbol{N}$ and a label $L \in \mathfrak{L}^{\mathrm{n}}$, the decomposed alignment between the annotations associated with L and the neuron activations is defined as:

$$dIoU(\boldsymbol{N}, L, \mathfrak{D}, I^C, I^U, E^U, E^C) = \frac{\sum_{x\in\mathfrak{D}} |I^U(L)_x| + |I^C(L)_x|}{|\boldsymbol{N}| + \sum_{x\in\mathfrak{D}} |E^U(L)_x| + |E^C(L)_x|} \tag{1}$$

where the subscript indicates the quantity per sample.

Proof. See Section A for the proof of equivalence between $dIoU$ and the IoU score. □

We visualize all the identified quantities and the $dIoU$ score in Section 2.2.

Observation 1 (Impact of Operators on Quantities)**.** Given the above definitions, we can quantify the impact of the OR, AND, and AND NOT bitwise logic operators connecting two concepts k_1 and k_2. The OR operator is 1-preserving (i.e., any 1 in either concept remains 1), and thus it preserves the common elements shared by the two concepts and combines (i.e., sums) the non-shared ones and unique elements. The AND operator is 0-preserving (i.e., any 0 in either concept forces 0) and therefore removes all of the unique elements as well as common elements not shared by both concepts. Finally, the AND NOT operator preserves all of the unique elements but removes the common elements shared by both concepts. These operators will be the ones considered in this paper.

2.3 Heuristic

This section introduces our proposed heuristic. Given a label $L \in \mathfrak{L}^i$ s.t. $i \leq n$, the generic goal of this heuristic is to estimate the label alignment in a faster and less computational intensive way than directly computing it. Specifically, the proposed heuristic gives an estimation for the following maximum and minimum quantities: (i) given a logic operator $\oplus$, a label $\mathfrak{L}^i$, and a concept k, the estimate alignment of $L \oplus k$, (ii) alignment reachable starting from L and chaining additional concepts not yet included in L, up to a length of n.

2.3.1 Estimate Label Quantities

To facilitate the estimation of the alignment of $L \oplus k$, we introduce the binary *Disjoint Matrix* $D \in \{0,1\}^{|\mathfrak{L}^1|\times|\mathfrak{L}^1|}$, which encodes whether two concepts share any annotation overlap and can be computed once per dataset as a preprocessing step. Specifically, for every pair (k_1, k_2) of concepts:

$$D[k_1, k_2] = \begin{cases} 1, & \text{if } \forall (x,j) \in |\mathfrak{D}| \times d \mid \boldsymbol{M}_{k_1}[x,j] \neq \boldsymbol{M}_{k_2}[x,j], \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

In addition, we introduce two new quantities, *Space for Common Extras* and *Space for Unique Extras*, derived from Definitions 2.1, 2.2 and 2.4, to characterize and constrain the available space for the set of extra elements:

$$SE^C = \{(x,j)\} : (x,j) \in C, \boldsymbol{N}[x,j] = 0 \qquad SE^U = \{(x,j)\} : (x,j) \in U, \boldsymbol{N}[x,j] = 0 \tag{3}$$

By Assumption 2, we can separate the label into its left ($L_{\leftarrow}$) and right ($L_{\rightarrow}$) sides. Then, we can use D to check whether the left and right sides of L share common concepts or are disjoint and estimate alignment differently in the two cases.

Disjoint: If the two sides are disjoint, then the common quantities are set to 0 since there cannot be shared elements between disjoint vectors ($|I^C(L)| = |E^C(L)| = 0$). Regarding the unique elements, their value can be derived by Observation 1: the sum of the quantities of each side for OR and 0 for AND. For AND NOT, we set them to 0 because it degenerates into an uninformative case[3].

$$I^U(L) = \begin{cases} |I^U(L_\leftarrow)| + |I^U(L_\rightarrow)|, & \text{if } \oplus = OR \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$E^U(L) = \begin{cases} |E^U(L_\leftarrow)| + |E^U(L_\rightarrow)|, & \text{if } \oplus = OR \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Overlap: If the two sides overlap, we can estimate the exact value of the unique quantities for OR and AND using the same formulas as in the disjoint case. For the AND NOT operator, by Observation 1, the value equals the quantity of the left side. For the common quantities, we can derive them by combining the definitions in Section 2.2 and Observation 1, obtaining:

$$|I^C_{min}(L)_x| = \begin{cases} max(|I^C_{min}(L_\leftarrow)_x|, |I^C(L_\rightarrow)_x|) & \text{if } \oplus = OR \\ max(|I^C_{min}(L_\leftarrow)_x| + |I^C(L_\rightarrow)_x| - |N^C_x|, 0) & \text{if } \oplus = AND \\ max(|I^C_{min}(L_\leftarrow)_x| - |I^C(L_\rightarrow)_x, 0) & \text{if } \oplus = AND\ NOT \end{cases} \tag{6}$$

$$|I^C_{max}(L)_x| = \begin{cases} min(|I^C_{max}(L_\leftarrow)_x| + |I^C(L_\rightarrow)_x|, |N^C_x|) & \text{if } \oplus = OR \\ min(|I^C_{max}(L_\leftarrow)_x|, |I^C(L_\rightarrow)_x|) & \text{if } \oplus = AND \\ min(|I^C_{max}(L_\leftarrow)_x)|, |N^C_x| - |I^C(L_\rightarrow)_x)| & \text{if } \oplus = AND\ NOT \end{cases} \tag{7}$$

$$|E^C_{min}(L)_x| = \begin{cases} max(|E^C_{min}(L_\leftarrow)_x|, |E^C(L_\rightarrow)_x|) & \text{if } \oplus = OR \\ max(|E^C_{min}(L_\leftarrow)_x| + |E^C(L_\rightarrow)_x| - |SE^C_x|, 0) & \text{if } \oplus = AND \\ max(|E^C_{min}(L_\leftarrow)_x| - |E^C(L_\rightarrow)_x|, 0) & \text{if } \oplus = AND\ NOT \end{cases} \tag{8}$$

$$|E^C_{max}(L)_x| = \begin{cases} min(|E^C_{max}(L_\leftarrow)_x| + |E^C(L_\rightarrow)_x|, |SE^C_x|) & \text{if } \oplus = OR \\ min(|E^C_{max}(L_\leftarrow)_x|, |E^C(L_\rightarrow)_x|) & \text{if } \oplus = AND \\ min(|E^C_{max}(L_\leftarrow)_x|, |SE^C_x| - |E^C(L_\rightarrow)_x|) & \text{if } \oplus = AND\ NOT \end{cases} \tag{9}$$

Note that, since $L_\rightarrow$ is always an atomic concept (by Assumption 2), its quantities are exact. A detailed derivation is provided in Section E.

Aggregated Computation The above quantities are defined per-sample, requiring $|\mathfrak{D}|$ comparisons for each computation. This can still be costly for large state spaces. To mitigate this, we introduce a lighter aggregated computation, obtained by summing the values per label. For example, $\sum_{x\in\mathfrak{D}} min(|E^C_{max}(L_\leftarrow)_x|, |SE^C_x| - |E^C(L_\rightarrow)_x|)$ can be transformed in $min(\sum_{x\in\mathfrak{D}} |E^C_{max}(L_\leftarrow)_x|, \sum_{x\in\mathfrak{D}} |SE^C_x| - \sum_{x\in\mathfrak{D}} |E^C(L_\rightarrow)_x|)$. The trade-off is precision versus efficiency: the sample version is more accurate but computationally intensive, while the aggregated version can be pre-computed once per label, reducing the cost to a single comparison per quantity at the expense of precision (see Section C for details).

2.3.2 Estimate Paths

In this section, we estimate the maximum score achievable by concatenating additional concepts to a label. If the explanation length were unbounded, this maximum would converge to 1, making the heuristic uninformative. However, compositional explanations are inherently bounded, as long explanations would not aid user understanding; in practice, the maximum length is typically small (e.g., 3). This boundedness allows us to produce tighter estimates. Given a label L, our goal is to estimate the numerator and denominator in Theorem 2.1 for all possible paths obtained by concatenating additional concepts through logic operators. To this end, we introduce the *maximum and minimum improvement*. For each sample, we extract the $top - n$ and $bottom - n$ values for each quantity and compute cumulative sums into the Top and $Bott$ vectors, starting from the largest or

[3]For example, "cat AND NOT dog" is always true, and the AND NOT side does not add meaningful information to the explanation.

smallest values, respectively. For example, $Top_k(I^C)_x$ represents the cumulative sum of the common intersection values of the k concepts with the highest scores for that quantity. These quantities, combined with Observation 1 and the equations in Section 2.3.1, allow us to compute the maximum and minimum factors for OR, AND, and AND NOT exclusive paths (i.e., paths where only one operator is used from that point onward to concatenate additional concepts):

$$|I_{min}(L)_x| = \begin{cases} max(|I^C_{min}(L)_x| + |I^U_{min}(L)_x|, Bott_1(I^C)_x + Bott_1(I^U)_x) & \text{OR Path} \\ 0 & \text{AND Path} \\ |I^U_{min}(L)_x| & \text{AND NOT Path} \end{cases} \tag{10}$$

$$|I_{max}(L)_x| = \begin{cases} min(|I^C_{max}(L)_x| + Top_t(I^C)_x, |N^C_x|) & \\ \quad +min(|I^U_{max}(L)_x| + Top_t(I^U)_x, |N^U_x|) & \text{OR Path} \\ min(|I^C_{max}(L)_x|, Top_1(I^C)_x) & \text{AND Path} \\ |I^U_{max}(L)_x| + min(|I^C_{max}(L)_x|, |N^C_x| - Bott_1(I^C)_x) & \text{AND NOT Path} \end{cases} \tag{11}$$

$$|Union_{min}(L)_x| = |\boldsymbol{N}_x| + \begin{cases} max(|E^C_{min}(L)_x| + |E^U_{min}(L)_x|, & \\ \quad Bott_1(E^C)_x + Bott_1(E^U)_x) & \text{OR Path} \\ 0 & \text{AND Path} \\ |E^U_{min}(L)_x| & \text{AND NOT Path} \end{cases} \tag{12}$$

$$|Union_{max}(L)_x| = |\boldsymbol{N}_x| + \begin{cases} min(|E^C_{max}(L)_x| + Top_t(E^C)_x, |SE^C_x|) & \\ \quad +min(|E^U_{max}(L)_x| + Top_t(E^U)_x, |SE^U_x|) & \text{OR Path} \\ min(|E^C_{max}(L)_x|, Top_1(E^C)_x) & \text{AND Path} \\ |E^C_{max}(L)_x| + min(|E^U_{max}(L)_x|, & \\ \quad |SE^C_x| - Bott_1(E^C)_x) & \text{AND NOT Path} \end{cases} \tag{13}$$

(14)

where t denotes the difference between the maximum length and the length of the label L. To estimate the values for paths involving multiple operators, we take the maximum and minimum of each quantity across the operators considered (see Section E.3 for a discussion about explicitly modeling every possible combination). Among the paths, we also include the *final path*, computed by Theorem 2.1, to denote the case where the label is not further expanded.

These estimates are finally used to compute the maximum and the minimum dIoU:

$$dIoU_{max} = \frac{\sum_{x \in \mathfrak{D}} |I_{max}(L)_x|}{\sum_{x \in \mathfrak{D}} |Union_{min}(L)_x|} \qquad dIoU_{min} = \frac{\sum_{x \in \mathfrak{D}} |I_{min}(L)_x|}{\sum_{x \in \mathfrak{D}} |Union_{max}(L)_x|} \tag{15}$$

Aggregated Computation As in Section 2.3.1, the aggregated computation estimates the quantities more efficiently by operating on aggregate values (i.e., sums) per label instead of computing them on a per-sample basis. For example, rather than using $|Union^{sample}_{max}(L)_x| = |N_x| + min(|E^C_{max}(L)_x|, Top_1(E^C)_x)$ the aggregated formulation uses $|Union^{aggr}_{max}(L)| = N + min(\sum_{x \in \mathfrak{D}} |E^C_{max}(L)_x|, Top^A_1(E^C))$, where Top^A is computed concept-wise.

2.4 Optimal Algorithm

The optimal algorithm is a best-first search guided by our proposed heuristic. We provide a textual overview of the main steps below and a more detailed discussion and pseudocode in Section C.

1. Compute the exact quantities (Section 2.2) for every concept in the dataset.
2. For each concept, compute $dIoU_{max}$ and $dIoU_{min}$ for all possible paths starting from it, using the heuristic aggregated computation.
3. Initialize the frontier with all paths whose estimated $dIoU_{max}$ is greater than the global maximum of the minimum estimates.

Table 2: Average number of visited, expanded, and estimated nodes, along with runtime per unit (in minutes), by the optimal algorithm, a beam search guided by our heuristic, and two alternative beam search algorithms.

Algorithm	Optimal	Visited	Expanded	Estimated	Time (sec)
Low Complexity					
Optimal (our)	✓	1	100	778	0.08
Beam + Our H.	✗	1	11	405	0.09
MMESH Beam	✗	135	14.94	716	10.01
Vanilla Beam	✗	716	14.94	-	2.77
Intermediate Complexity					
Optimal (our)	✓	1	2885	1.76×10^{6}	69.27
Beam + Our H.	✗	1.94	11	24730	9.46
MMESH Beam	✗	41.94	15	37978	37.31
Vanilla Beam	✗	37979	15	-	450
High Complexity					
Optimal (our)	✓	47.36	$3.54\ \times10^{5}$	1.28×10^{8}	5768
Beam + Our H.	✗	5.82	11	28947	139
MMESH Beam	✗	44.98	13.47	$53774\ \pm 2$	106
Vanilla Beam	✗	53775	13.47	-	5929

4. Iteratively pop nodes from the frontier, starting from those with the highest estimated $dIoU_{max}$.
   (a) If the estimate is aggregated, refine it by computing the sample-based estimate and reinsert the node. Otherwise, proceed to the next step.
   (b) If the node path is not a final one (i.e., can still be extended), expand its label by concatenating every possible concept with every allowed connective. For each new label, compute $dIoU_{max}$ and $dIoU_{min}$ via the heuristic aggregated computation and insert them into the frontier.
   (c) If the node path is a final one, compute its exact IoU. During this step, the algorithm stores the intermediate quantities of the sub-labels composing the label. This information is then backpropagated to the frontier: nodes that share sub-labels with the evaluated node update their estimates using these exact quantities.
   (d) Continue until the frontier is empty. During the process, whenever a new maximum of the $dIoU_{min}$ estimates is found, prune the frontier by removing all nodes whose $dIoU_{max}$ falls below this threshold.

Additionally, to reduce redundant computation, the algorithm incorporates a limited set of logical equivalence rules, which are applied before and during the expansion phase, and maintains a buffer to cache recently explored nodes associated with the same estimated $dIoU_{max}$. We discuss these details and justify the design choices in Section C. Because the maximum $dIoU$ of each path is an overestimation, and since the algorithm is designed to visit all nodes whose $dIoU$ exceeds that of the current best explanation, **the algorithm is guaranteed to return the most aligned explanation**. In other words, the returned explanation is always the optimal one.

3 Analysis

3.1 Feasibility of Optimality and Heuristic-Guided Beam Search

This section evaluates the feasibility of the proposed optimal algorithm and the effectiveness of the heuristic when used to guide beam search. We consider three scenarios that vary in annotation complexity: **low, moderate, and high**. The low-complexity setting (Cityscapes [3]) includes a small number of concepts (25), all of which are disjoint. The moderate-complexity setting (the extended

Table 3: Average number of changed explanations and percentage of explanations falling into categories 1, 2, and 3 for different models.

Dataset	Diff	Beam IoU	Optimal IoU	Cat 1 (%)	Cat 2 (%)	Cat 3 (%)
ResNet	8%	0.077	0.083	85	4	11
AlexNet	22%	0.045	0.047	93	3	4
DenseNet	39%	0.039	0.041	73	0	27

version of Ade20K [21]) includes a much larger number of concepts (847) but with no overlapping annotations. Finally, the high complexity setting (Broden [1]) involves frequent overlaps combined with a large number of concepts (1198). As reference baselines, we include the MMESH-guided beam search [11] and the vanilla beam search [14]. The beam search variant that uses our heuristic (*Beam + Our H.*), replaces MMESH with label-quantity estimation (see Section D for details). For each setting, we report the average number (over 50 units) of visited nodes (i.e., those for which the exact IoU is computed), expanded nodes, estimated nodes, and the computation time per unit (std. dev. can be found in Section B). Following Mu and Andreas [14], we extract 50 random units of the last convolutional layer in a ResNet [7] model trained on Places365 [20] and use the highest activations (top 0.005 percentile) as activation ranges.

Table 2 shows that the optimal algorithm consistently finds the optimal solution within feasible runtimes across all scenarios. As expected, informed beam search methods are faster, since they explore a much smaller portion of the state space (i.e., estimated nodes). However, the performance of the optimal algorithm remains comparable to the vanilla beam search of Mu and Andreas [14], even in the most complex settings. Importantly, the number of expanded states is a small fraction of the overall state space (less than 0.1%) and is significantly smaller than the number of estimated states. This property is crucial for refining heuristic estimates without compromising runtime efficiency (see Section C).

More notably, beam search guided by our heuristic outperforms all baselines across all settings in terms of visited states while achieving comparable or better runtimes. Compared to MMESH, our approach offers several improvements. First, MMESH relies on annotations during beam expansion, which requires annotations to be kept in memory and GPU resources, thus limiting scalability and parallelization. Conversely, our variant uses annotations only when visiting states, allowing them to be loaded from disk on demand. This design, combined with its higher efficiency, removes the need to store annotations in memory and enables easier parallelization in low-resource scenarios. Our beam variant also scales efficiently with changes in hyperparameters. For example, when varying explanation length (3,5,10, and 20) and beam size (5, 10, and 20) in the moderate settings, the runtime of our variant is stable between 0.16 and 0.18 min/unit. By contrast, MMESH slows down progressively with both explanation length [0.62, 1.28, 4.55, > 240] and beam size [0.62, 1.24, > 240], starting from 0.62 for 3-concepts explanations and beam size fixed to 5 to 4.5 min/units (see Table 5 for a table of results). For explanation lengths or beam sizes higher than 10, MMESH becomes infeasible, despite the limited complexity of the considered settings.

3.2 Explanation Analysis

This section addresses the question we originally posed about beam search–based algorithms: are the explanations they compute optimal? In general, the answer is no. Although beam search often identifies valid explanations, our analysis (Table 3) shows that between 10% and 40% of them differ from the optimal ones across several models studied in previous literature [14]. We classify these differences into three categories: (1) explanations differ in both concepts and IoU, (2) explanations involve the same concepts but differ in how they are connected, resulting in different IoU, and (3) explanations share the same IoU but differ in the way the concepts are connected.

The first category is the most prominent and often involves explanations connected by AND and AND NOT operators, suggesting that beam search struggles to express explanations that describe units specialized in recognizing complex scenarios. This discrepancy does not imply that the explanations produced by beam search are incorrect; they still capture meaningful alignments between the unit and the concepts expressed in the formulas. However, they often fall short of reflecting the highest degree of alignment that the unit actually exhibits. The second category includes cases where the

optimal explanations are more precise and more aligned (e.g., from ((table OR sink) AND white-c) (IoU=0.036) to ((white-c AND table) OR sink) (IoU=0.040)). These differences are small and, in this case, we can consider the beam search explanations quite close and faithful to the optimal explanations.

Finally, the third category includes cases where the semantics of the explanations changes. For example, consider the explanation ((ball_pit-s OR flower) AND NOT dining_room-s). At first glance, one might interpret this unit as being specialized in recognizing $ball\_pit$ not located in dining rooms. However, inspecting the dataset reveals that these concepts never co-occur (i.e., they are disjoint), whereas there are instances including both flowers and dining rooms. Thus, part of the explanation is effectively "unverified". In contrast, the optimal algorithm correctly identifies the alignment as ((flower AND NOT dining_room-s) OR ball_pit-s) and exposes a key limitation of beam search: it cannot backtrack on earlier decisions, and the search may compensate for errors by producing explanations that rely on unverified scenarios (further details and examples for 10 units per model are provided in Section F).

3.3 Algorithm Analysis: Insights and Limitations

This section briefly discusses the key design choices behind the optimal algorithm and provides insights into its limitations.

Overall, the beam search guided by our heuristic represents a safe choice when the reduced time for computation is a priority. Conversely, the optimal algorithm represents a first promising step towards the guarantee of optimality on neural explanations and a better choice when the optimality is considered more important than the time of execution. Regarding design choices, Section C includes a detailed discussion of them. In summary, all the quantities and steps proposed in this algorithm are necessary for the search to be tractable in high complexity scenarios. For example, we use aggregated computations to decide which nodes enter the frontier and sample-based ones to parse it, since estimated nodes far outnumber those actually expanded. Importantly, nodes added to the frontier may still be pruned if their estimated IoU falls below the best found so far. Backpropagation and minimum estimation are crucial in this setting, since they reduce redundant nodes and prevent exhaustive exploration of overestimated candidates. By empirically analyzing the algorithm's execution traces, we identified the following insights and areas of improvement for future research:

Convergence Towards Breath-first search We noted that the state space is explored similarly to breadth-first search, since the top vector dominates the search and roughly overestimates the maximum improvement. In theory, this reliance could make it infeasible for long explanations (although shorter explanations are generally preferred). To mitigate this problem, future research could explore vectors that are (1) specific to a label (which is costly) or (2) novel and better representations of the maximum improvement.

Unmeaningful Units Our algorithm could become much slower when either a unit is not interpretable (the IoU < 0.04 [2]) or it is unspecialized (using default rules) and the probing dataset is of high complexity. In these extreme cases, the space to be explored is very large since there is no clear alignment and the combinations of concepts are all similar. One possible solution is to run beam search and then refine the units deemed interpretable for optimality. Alternatively, one could automatically switch from the optimal algorithm to beam search once the frontier grows too large, and use the beam search output to initialize and guide the optimal search.

4 Conclusion

This paper presents the first attempt to guarantee optimality in compositional explanations. Specifically, we identified and formalized the fundamental quantities governing spatial alignment, proposed a heuristic to estimate the potential alignment from any label in the search space, and developed an algorithm capable of computing optimal compositional explanations within feasible runtimes. We further demonstrated that our heuristic can also improve existing beam search–based approaches for non-optimal explanations. Since our method does not rely on spatial information, the proposed heuristic is broadly applicable across domains. Moreover, our theoretical contribution may extend beyond neural explanations, as bitwise computations are of interest in areas such as semantic seg-

mentation and communication. Finally, we call for further research on refining this heuristic and on designing new algorithms for computing alternative forms of compositional explanations.

5 Reproducibility Statement

To ensure reproducibility, we will release the code upon acceptance. Additionally, we describe all the assumptions of this work in Section 2.1 and Section 2.2. The full pseudo-code for both the optimal algorithm and the beam search guided by our heuristic is provided in Section C and Section D. Proofs and derivations of the estimations are given in Section 2.2, Section 2.3.1, and Section 2.3.2 and Sections A and E. We detail the rationale behind design choices in Section C and provide example outputs of our algorithm in Section F. These examples can serve as gold references when re-implementing the algorithm. Finally, we describe the datasets and additional setup information in Section B.

References

[1] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

[2] D. Bau, J.-Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou, and A. Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1907375117. URL `https://www.pnas.org/content/early/2020/08/31/1907375117`.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022.

[5] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. 2009.

[6] R. Harth. *Understanding Individual Neurons of ResNet Through Improved Compositional Formulas*, pages 283–294. Springer International Publishing, 2022. ISBN 9783031092824. doi: 10.1007/978-3-031-09282-4_24.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[8] E. Hernandez, S. Schwettmann, D. Bau, T. Bagashvili, A. Torralba, and J. Andreas. Natural language descriptions of deep features. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=NudBMY-tzDr`.

[9] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[11] B. La Rosa, L. H. Gilpin, and R. Capobianco. Towards a fuller understanding of neurons with clustered compositional explanations. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=51PLYhMFWz`.

[12] S. M. Makinwa, B. La Rosa, and R. Capobianco. Detection accuracy for evaluating compositional explanations of units. In *AIxIA 2021 - Advances in Artificial Intelligence*, pages 550–563. Springer International Publishing, 2022. doi: 10.1007/978-3-031-08421-8_38.

[13] R. Massidda and D. Bacciu. Knowledge-driven interpretation of convolutional neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 356–371. Springer International Publishing, 2023. doi: 10.1007/978-3-031-26387-3_22.

[14] J. Mu and J. Andreas. Compositional explanations of neurons. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[15] A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *Visualization for Deep Learning workshop, ICML 2016*, Feb. 2016.

[16] T. Oikarinen and T.-W. Weng. CLIP-dissect: Automatic description of neuron representations in deep vision networks. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=iPWiwWHc1V`.

[17] T. Oikarinen and T.-W. Weng. Linear explanations for individual neurons. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=WIbntm28cM`.

[18] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11), nov 2017. doi: 10.23915/distill.00007.

[19] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[20] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[21] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. doi: 10.1109/cvpr.2017.544.

A Proof Equivalence Alignment-IoU Score

Proof. Let $\boldsymbol{M}_L$ be the binary label tensor representing the result of the bitwise logic operations induced by the logic operators connecting single concepts $k \in L$. The Intersection over Union (IoU) score is then defined as:

$$IoU(\boldsymbol{N}, L, \mathfrak{D}, \boldsymbol{M}_L) = \frac{|\boldsymbol{N} \cap \boldsymbol{M}_L|}{|\boldsymbol{N} \cup \boldsymbol{M}_L|} \tag{16}$$

We first observe that $|\boldsymbol{N} \cap \boldsymbol{M}_L|$ is necessarily equal to $\sum_{x \in \mathfrak{D}} |I^U(L)_x| + |I^C(L)_x|$. Indeed, all elements in $|\boldsymbol{N} \cap \boldsymbol{M}_L|$ are associated with both the neuron and the label L. Consider a generic element $j \in |\boldsymbol{N} \cap \boldsymbol{M}_L|$. Since the logic operators are 00-preserving (Assumption 2), this element must be associated with at least one annotation of one of the concepts in L. But if the element is associated with at least a concept, then it is either associated with exactly one, and thus $j \in I^U$, or with multiple concepts, and thus $j \in I^C$. This also holds in the case of concepts connected by the AND NOT operator, because by design this operator must be chained to a positive concept, and AND NOT is 00-preserving.

Regarding the denominator, we can note that

$$|\boldsymbol{N} \cup \boldsymbol{M}_L| = |\boldsymbol{N}| + |\boldsymbol{M}_L| - |\boldsymbol{N} \cap \boldsymbol{M}_L| \tag{17}$$

Observe that for all $j \in \boldsymbol{N} \cap \boldsymbol{M}_L$, the contribution $|\boldsymbol{M}_L| - |\boldsymbol{N} \cap \boldsymbol{M}_L| = 0$, since these elements are counted in both sets. Hence, $|\boldsymbol{M}_L| - |\boldsymbol{N} \cap \boldsymbol{M}_L|$ represents the number of elements that are labeled

Table 4: Average number of visited, expanded, and estimated nodes, along with runtime per unit (in minutes). Statistics are computed across three settings and datasets: low complexity (Cityscapes), intermediate complexity (Ade20K Full), and high complexity (Broden). The table reports results for our two proposed algorithms, the beam search powered by MMESH, and the vanilla beam search.

Algorithm	Optimal	Visited	Expanded	Estimated	Time (sec)
Low Complexity					
Optimal (our)	✓	1	100 $\pm$ 34	778 $\pm$ 221	0.08 $\pm$ 0.01
Beam + Our H.	✗	1 $\pm$ 0.30	11 $\pm$ 0.42	405 $\pm$ 25	0.09 $\pm$ 0.01
MMESH Beam	✗	135 $\pm$ 3	14.94 $\pm$ 0.42	716 $\pm$ 17	10.01 $\pm$ 0.30
Vanilla Beam	✗	716 $\pm$17	14.94 $\pm$ 0.42	-	2.77 $\pm$ 0.16
Intermediate Complexity					
Optimal (our)	✓	1	2885 $\pm$ 2583	1.76×10^6	69.27 $\pm$ 40.60
Beam + Our H.	✗	1.94 $\pm$0.24	11	24730 $\pm$ 1090	9.46 $\pm$ 0.38
MMESH Beam	✗	41.94 $\pm$156	15	37978 $\pm$ 1.79	37.31 $\pm$ 2.87
Vanilla Beam	✗	37979 $\pm$ 1.74	15	-	450 $\pm$ 10
High Complexity					
Optimal (our)	✓	47.36 $\pm$ 99.27	3.54 $\times10^5$	1.28×10^8	5768 $\pm$ 1297
Beam + Our H.	✗	5.82 $\pm$ 6.47	11	28947 $\pm$ 2261	139 $\pm$ 23
MMESH Beam	✗	44.98 $\pm$ 69.11	13.47 $\pm$ 4.54	53774 $\pm$ 2	106 $\pm$ 18
Vanilla Beam	✗	53775 $\pm$ 1	13.47 $\pm$ 4.54	-	5929 $\pm$ 548

as 1 in $\boldsymbol{M}_L$ but for which the neuron does not fire. This definition coincides with Definition 2.4. Therefore, we can rewrite the denominator as

$$|\boldsymbol{N} \cup \boldsymbol{M}_L| = |\boldsymbol{N}| + \sum_{x \in \mathfrak{D}} |E(L)_x|. \tag{18}$$

Similarly to the numerator case, because the logic operators are 00-preserving, every element in $E(L)_x$ is either included in $E^U(L)_x$ or in $E^C(L)_x$. Thus,

$$|\boldsymbol{N} \cup \boldsymbol{M}_L| = |\boldsymbol{N}| + \sum_{x \in \mathfrak{D}} \big(|E^U(L)_x| + |E^C(L)_x|\big). \tag{19}$$

and thus

$$dIoU(\boldsymbol{N}, L, \mathfrak{D}, I^C, I^U, E^U, E^C) = IoU(\boldsymbol{N}, L, \mathfrak{D}, \mathbf{M}). \tag{20}$$

□

B Complete Results

Table 4 reports the averages and standard deviations of the results presented in Table 2. Table 5 shows differences in scalability between beam search guided by our proposed heuristic and beam search guided by MMESH. Both tables are computed over 50 units randomly extracted from the penultimate layer of a ResNet18 model trained on Places365, consistent with prior work on compositional explanations [14, 6, 12, 11]. We follow Bau et al. [2] and Mu and Andreas [14] and use the highest activations corresponding to the top 0.005 percentile across the probing dataset as the activation range and fix the maximum explanation length to 3. As probing datasets, we used Cityscapes [3] (accessible at: `https://www.cityscapes-dataset.com/` under MIT License) for the low complexity settings, Ade20kFull [21] (accessible via the Detectron2 [19] framwork) for the moderate settings, and Broden [1] (accessible at `https://github.com/CSAILVision/NetDissect` under MIT license) for the highest complexity settings. All results were computed on a workstation equipped with an NVIDIA GTX 3090 GPU, without parallelization, in order to avoid timing overhead.

Table 5: Avg. Time across 50 units (min) per hyperparameters on moderate settings.

Value	Our	MMESH
	Explanation Len	
3	0.16	0.62
5	0.16	1.28
10	0.17	4.55
20	0.18	> 240
	Beam Size	
5	0.16	0.62
10	0.17	1.24
20	0.18	> 240

C Optimal Algorithm

This sections describe the optimal algorithm we introduced in the main paper. Algorithm 1 shows the pseudocode of our algorithm. The optimal algorithm is a best-first search guided by our proposed heuristic and consists of the following steps:

1. Compute the exact quantities (Section 2.2) for every concept in the dataset (line 5).
2. For each concept, compute $dIoU_{max}$ and $dIoU_{min}$ for all possible paths starting from it, using the heuristic aggregated computation (line 6).
3. Initialize the frontier with all paths (line 7) and keep track of the greatest $dIoU_{min}$ (line 8). The frontier is sorted by the estimated $dIoU_{max}$.
4. Reduce the frontier by removing nodes whose estimated $dIoU_{max}$ is lower than the global maximum of the minimum estimates (line 10).
5. Iteratively pop nodes from the frontier, starting from those with the highest estimated $dIoU$ (lines 11-12).
   (a) If the estimate is aggregated, refine it by computing the sample-based estimate and reinsert the node (lines 13-20). Otherwise, proceed to the next step.
   (b) Apply logical equivalence rules when possible (lines 21-26), recompute the sample-based estimate for the equivalent expression, and reinsert the node if the estimate has changed. Otherwise, proceed to the next step. Currently, we only check distributive properties as logical equivalences. Note that even though the expressions are logically equivalent, their estimates may differ due to overestimation. The goal of this step is to select the form with the smallest overestimation among all possible equivalents.
   (c) Check whether the same node has been recently explored by comparing its maximum IoU with the most recent one. If they match but the node has not yet been explored, add it to memory; if it has already been explored, skip it. Otherwise, clear the memory and initialize the most recent IoU with the node's maximum IoU (lines 27–38).
   (d) If the node is not a final one (i.e., can still be extended), expand its label by concatenating every possible concept with every allowed connective (line 49). For each new label, compute $dIoU_{max}$ and $dIoU_{min}$ via the heuristic aggregated computation and insert them into the frontier (lines 50-51). If a new maximum is found among $dIoU_{min}$ of the new paths, update the global maximum and reduce the frontier (lines 52-55).
   (e) If the node is a final one, compute its exact IoU (line 42). During this step, the algorithm stores the intermediate quantities of the sub-labels composing the label (line 40). This information is then backpropagated to the frontier: nodes that share sub-labels with the evaluated node update their estimates using these exact quantities (line 41). If the IoU is greater than that of the best label found so far, update the best label with the current node (lines 43–45).
   (f) Continue until the frontier is empty (line 11).

In the following, we provide the rationale behind several design choices made during the development of this algorithm.

Memory Mechanism The memory mechanism (lines 27–38) was introduced to account for logical equivalences not handled elsewhere in the algorithm and to prevent redundant computation. We considered several alternatives, but this solution proved to be the least computationally expensive. Without such a mechanism, some logically equivalent rules could be expanded multiple times (lines 49–55), leading to a significantly larger search space. Alternative options would be to check for logical equivalences or the presence of a node directly when adding nodes to the frontier. However, this would make exploration prohibitively expensive: verifying logical equivalence or membership would offset the efficiency gains of the heuristic. In addition, since the frontier is implemented as a heap, checking whether a node is already present is slower than with a sorted list. Maintaining a sorted frontier would require re-sorting on every insertion, which would be too costly and therefore infeasible.

Concept Quantities A key design choice concerns the handling of concept quantities. We store quantities only for atomic concepts and do not cache them when estimating (lines 6, 14, and 50) or computing the $dIoU$ (line 40). Consequently, the intermediate quantities of labels are recomputed multiple times during the search. This decision was made for two main reasons. First, storing additional quantities increases access time, which is critical because the algorithm frequently accesses these values; this overhead could easily exceed the time required to recompute them from scratch. Second, caching more quantities increases memory usage, potentially limiting the ability to run parallel processes. Given the substantial runtime required for the most complex datasets, preserving the ability to parallelize is essential. Moreover, avoiding extensive caching keeps the approach lightweight in terms of memory and resource requirements. In conclusion, the marginal gains from more precise estimations are outweighed by the overhead, especially given that the algorithm tends to converge toward a breadth-first search (Section 3.3).

Backpropagation This process was introduced to reduce the number of visited states. While it has no (or bad) impact on low and moderate-complexity settings, it significantly reduces runtime in high complexity scenarios, especially when running on CPUs. Specifically, the frontier in the later stages of the search often contains similar explanations that differ only in the last added concept (e.g., (A AND B) OR C) and (A AND B) OR D). These nodes typically appear in the frontier because the left-hand terms provide a rough overestimation relative to the current maximum. When a node is visited, this overestimation is temporarily corrected, and the backpropagation step updates the estimates of all remaining nodes. This correction helps the algorithm to avoid visiting unpromising states. On average, 2000–3000 nodes per unit are updated via backpropagation, highlighting the importance of this mechanism.

Sample vs Aggregated Computation The optimal algorithm leverages both sample-based and aggregated computations for path and label estimations. This design is necessary due to the large state space and the overestimation introduced by the aggregated computation. We explored several alternatives during the development of this work, but this trade-off is the only one that ensures feasibility in high complexity settings. As explained in the main text, sample computation requires $|\mathfrak{D}|$ comparisons per calculation, whereas aggregated computation relies solely on the sum of concept quantities computed at the first step of the algorithm, combining them in a single operation. These two approaches represent a trade-off between precision and efficiency: the sample version is more accurate but computationally intensive, while the aggregated version is faster but less precise. In practice, using only aggregated computation for both node expansion and frontier exploration would yield faster per-node computations but result in a frontier that is orders of magnitude larger than that explored by our combined approach. Conversely, using only sample computation slightly reduces the frontier (by roughly 50,000 nodes per unit in preliminary experiments) but incurs significantly higher computation time per node, effectively nullifying the gains from the smaller frontier.

MinIoU As explained previously, the algorithm computes both the minimum and maximum IoU for each label and path. This increases the time required per estimation, since the maximum and minimum calculations are distinct and rarely share terms, effectively doubling the computation time per node. An alternative design could omit the MinIoU computation and explore only nodes whose maximum IoU exceeds the best visited so far. However, without the MinIoU, it becomes impossible

to dynamically reduce the frontier at runtime: the frontier can only grow, and the only way to remove a single node is to fully explore it. This would result in a frontier orders of magnitude larger than the one explored by the proposed design, especially in the early phase of the search, when the MinIoU is updated multiple times and allows significant pruning. Future work could explore adaptive strategies that decide dynamically whether to compute MinIoU based on the search space or external information, potentially improving overall runtime.

Code Optimization In addition to the previously mentioned design choices, we implemented several code optimizations to avoid unnecessary computation of quantities and to handle logical equivalences. For instance, we enforce an order on concept indices when chaining two consecutive applications of the same operator during node expansion. For example, for a concept with index 10, it can only be chained with concepts having an index greater than 10, since all other combinations will be captured when expanding nodes with lower indices. The same rule applies to consecutive operators of the same type. For example, if we have $(3 \text{ OR } 15)$, we can only chain concepts with indices greater than 15 when applying the OR operator again (e.g., $(3 \text{ OR } 15) \text{ OR } 18$), while we are free to choose any concept for other operators (AND or AND NOT). Other code optimizations involve avoiding the computations of paths when the intersection of the right or left sides is 0 and avoiding the sample computation related to equations involving $Bott_1$ in datasets where this vector is always 0 (see Section E)

D Beam Search Algorithm

In this section, we provide a high-level description of the beam search guided by our heuristic and present its pseudo-code in Algorithm 2. This algorithm replaces the MMESH heuristic [11] with the label quantity estimates introduced in Section 2.2, within a standard heuristic-guided beam search framework. The procedure begins by following the initial two steps of the optimal algorithm but keeps only the best b concepts to form the initial beam. It then proceeds iteratively for n steps: at each iteration, the algorithm expands the nodes in the current beam, sorts the resulting state space according to the heuristic, and evaluates candidate nodes by computing their IoU. The top b nodes are then selected to form the next beam. The process terminates when no candidate improves the current best IoU^{max}, or when no nodes remain to be expanded or visited. Differently from the optimal algorithm, this algorithm relies solely on sample computations and does not make use of, or estimate, the paths introduced in Section 2.3.2. Beyond the advantages discussed in Section 3.1, our heuristic does not rely on spatial information and can therefore be applied across multiple domains without requiring modifications to the code or formulation.

E Estimating Quantities

This section presents the rationale and derivation of all estimations computed by our heuristic. We divide the discussion into per-sample estimations and aggregated computations.

E.1 Sample Computation

E.1.1 Label Quantities

In the following, we derive the estimations of the label quantities introduced in the main text for the sample-based computation. For clarity, we group the equations by operator and discuss each operator separately. Because the quantities for the unique elements are exact and are directly derived from the definition and Observation 1, here we focus on the derivation of the common quantities.

$$|I^C_{min}(L)_x| = max(|I^C_{min}(L_{\leftarrow})_x|, |I^C(L_{\rightarrow})_x|) \tag{21}$$

$$|I^C_{max}(L)_x| = min(|I^C_{max}(L_{\leftarrow})_x| + |I^C(L_{\rightarrow})_x|, |N^C_x|) \tag{22}$$

$$|E^C_{min}(L)_x| = max(|E^C_{min}(L_{\leftarrow})_x|, |E^C(L_{\rightarrow})_x|) \tag{23}$$

$$|E^C_{max}(L)_x| = min(|E^C_{max}(L_{\leftarrow})_x| + |E^C(L_{\rightarrow})_x|, |SE^C_x|) \tag{24}$$

Derivation Equations (21) *to* (24) *for the OR operator*: We can start by noting that Equation (21) corresponds to the case where one side is a subset of the other. In this case, the equation gives the maximum cardinality of the intersection between the left and right sides, since the minimum elements are already included in the maximum and the OR is 1-preserving (i.e., the number of ones cannot be lower than before the combination). Conversely, Equation (22) corresponds to the case where the sides are disjoint in their extras, adjusted by the $|N_x^C|$ quantity. Indeed, because the left side is an overestimation, it may happen that the sum exceeds the limits, requiring readjustment using $|N_x^C|$. The estimation of the maximum and minimum extras follows the same reasoning, with the only difference that the common space extra $|SE_x^C|$ is used to adjust the quantity of the maximum extras.

$$|I^C_{min}(L)_x| = max(|I^C_{min}(L_{\leftarrow})_x| + |I^C(L_{\rightarrow})_x| - |N_x^C|, 0) \tag{25}$$
$$|I^C_{max}(L)_x| = min(|I^C_{max}(L_{\leftarrow})_x|, |I^C(L_{\rightarrow})_x|) \tag{26}$$
$$|E^C_{min}(L)_x| = max(|E^C_{min}(L_{\leftarrow})_x| + |E^C(L_{\rightarrow})_x| - |SE_x^C|, 0) \tag{27}$$
$$|E^C_{max}(L)_x| = min(|E^C_{max}(L_{\leftarrow})_x|, |E^C(L_{\rightarrow})_x|) \tag{28}$$

Derivation Equations (25) *to* (28) *for the AND operator*: In this case, the AND operator is 0-preserving. Therefore, when computing the minimum, we consider the scenario where a guaranteed overlap (i.e., both sides equal to 1) occurs. This happens when the sum of the two sides exceeds the maximum available space, represented by $|N_x^C|$ and $|SE_x^C|$, respectively. In such cases, the minimum guaranteed overlap is given by the difference between the sum and the cardinality of the available space. In all other cases, the best estimation we can provide is simply 0. The computation of the maximum corresponds to the case of fully overlapping concepts. Since the operator is 0-preserving, the equation in this case selects the minimum of the two sides for each sample.

$$|I^C_{min}(L)_x| = max(|I^C_{min}(L_{\leftarrow})_x| - |I^C(L_{\rightarrow})_x|, 0) \tag{29}$$
$$|I^C_{max}(L)_x| = min(|I^C_{max}(L_{\leftarrow})_x)|, |N_x^C| - |I^C(L_{\rightarrow})_x|) \tag{30}$$
$$|E^C_{min}(L)_x| = max(|E^C_{min}(L_{\leftarrow})_x| - |E^C(L_{\rightarrow})_x|, 0) \tag{31}$$
$$|E^C_{max}(L)_x| = min(|E^C_{max}(L_{\leftarrow})_x|, |SE_x^C| - |E^C(L_{\rightarrow})_x|) \tag{32}$$

Derivation Equations (29) *to* (32) *for the AND NOT operator*: This operator behaves like the AND operator but combines a negated concept, flipping all the bits in the corresponding vectors. The maximum estimations follow the same reasoning as for the AND operator, except that in this case $|N_x^C| - |I^C(L_{\rightarrow})_x|$ and $|SE_x^C| - |E^C(L_{\rightarrow})_x|$ represent the bits that were originally 0 and are now 1, corresponding to the common intersection and the common extras of the negated concept. The minimum estimation corresponds to the case where the right side fully overlaps with the left side. In this case, due to the negation, all overlapping bits flip to 0. Since the AND operator is 0-preserving, this results in the loss of all left side elements shared with the right side.

E.1.2 Path Quantities

In the following, we derive the estimations of the path quantities introduced in the main text. For clarity, we group the equations by operator and discuss each operator separately. Note that we assume, for all the paths, that at least 1 concept is added to the label, since the case where no concepts are added is covered by the "final path" obtained by applying Theorem 2.1 to the label quantities. In all the following equations, t denotes the difference between the maximum length and the length of the label L.

$$|I_{min}(L)_x| = max(|I^C_{min}(L)_x| + |I^U_{min}(L)_x|, Bott_1(I^C)_x + Bott_1(I^U)_x) \tag{33}$$
$$|I_{max}(L)_x| = min(|I^C_{max}(L)_x| + Top_t(I^C)_x, |N_x^C|) + min(|I^U_{max}(L)_x| + Top_t(I^U)_x, |N_x^U|) \tag{34}$$
$$|Union_{min}(L)_x| = |\boldsymbol{N}_x| + max(|E^C_{min}(L)_x| + |E^U_{min}(L)_x|, Bott_1(E^C)_x + Bott_1(E^U)_x) \tag{35}$$
$$\begin{aligned} |Union_{max}(L)_x| = &|\boldsymbol{N}_x| + min(|E^C_{max}(L)_x| + Top_t(E^C)_x, |SE_x^C|) \\ &+ min(|E^U_{max}(L)_x| + Top_t(E^U)_x, |SE_x^U|) \end{aligned} \tag{36}$$

Derivation of Equations (33) *to* (36) *for the OR operator*: Equation (33) and Equation (35) follow the same derivation as Equations (21) and (23). In this case, the added concept is represented by $Bott_1(I^C)$, which corresponds to the minimum intersection of any concept in a given sample and reflects the case of fully overlapping concepts. In practice, for most datasets, this quantity is always equal to 0; thus, it reduces to $|I^C_{min}(L)_x| + |I^U min(L)_x|$. The same reasoning applies to $|Union_{min}(L)_x|$ and the extras. Note also that the label quantities can be lower than the bottom values, since they represent combinations of concepts that may further reduce these quantities. Finally, the maximum quantities correspond to the case where the concepts included in L are disjoint from those cumulated in the Top vectors. In this situation, the quantities are simply the sum of neuron activations, maximum quantities, and the Top vectors, adjusted by the available space $|N^U_x|$ and $|SE^U_x|$, respectively.

$$|I_{min}(L)_x| = 0 \tag{37}$$
$$|I_{max}(L)_x| = min(|I^C_{max}(L)_x|, Top_1(I^C)_x) \tag{38}$$
$$|Union_{min}(L)_x| = |\boldsymbol{N}_x| \tag{39}$$
$$|Union_{max}(L)_x| = |\boldsymbol{N}_x| + min(|E^C_{max}(L)_x|, Top_1(E^C)_x) \tag{40}$$

Derivation Equations (37) *to* (40) *for the AND operator*: This operator is simpler to derive. Specifically, Equations (25) and (27) corresponds to the case where the label and all the concepts included in the Top vectors are disjoint, and thus all the quantities reduce to 0. Conversely, Equation (39) and Equation (40) represent the case where the label fully overlaps with the quantities stored in Top_1. Therefore, for the AND operator, only the quantities from the smaller concept are preserved per sample. Note that Top_1 is the only applicable Top vector here, since higher indices sum the cardinality of multiple concepts, which would violate the operations defined by the AND operator.

$$|I_{min}(L)_x| = |I^U_{min}(L)_x| \tag{41}$$
$$|I_{max}(L)_x| = |I^U_{max}(L)_x| + min(|I^C_{max}(L)_x|, |N^C_x| - Bott_1(I^C)_x) \tag{42}$$
$$|Union_{min}(L)_x| = |\boldsymbol{N}_x| + |E^U_{min}(L)_x| \tag{43}$$
$$|Union_{max}(L)_x| = |\boldsymbol{N}_x| + |E^C_{max}(L)_x| + min(|E^U_{max}(L)_x|, |SE^C_x| - Bott_1(E^C)_x), \tag{44}$$

Derivation Equations (41) *to* (44) *for the AND NOT operator*: Equations (29) and (31) corresponds to the same case as in Equations (25) and (27). However, since the AND NOT operator preserves all unique elements (Observation 1), the minimum intersection corresponds to the minimum unique intersection of the label, and the minimum union includes the minimum unique extras. Conversely, Equations (43) and (44) represents the case where the label fully overlaps with the negated concept, represented by $|N^C_x| - Bott_1(I^C)_x$ and $|SE^C_x| - Bott_1(E^C)_x$, respectively. As previously noted, in practice, for most datasets, $Bott_1$ is always equal to 0. Thus, the equations simplify to $|I^U_{max}(L)_x| + |I^C_{max}(L)_x|$ for the intersection and $|\boldsymbol{N}_x| + |E^C_{max}(L)_x| + |E^U_{max}(L)_x|$ for the union, since by definition $|I^C_{max}(L)_x| < |N^C_x|$ and $|E^U_{max}(L)_x| < |SE^C_x|$ due to the fact that $|N^C_x|$ represents the total neuron common space and $|SE^C_x|$ represents the total available space for common extras.

E.2 Aggregated Computation

This section describes the aggregated computation of the common quantities. To improve readability, we shorten the notation $\sum_{x \in \mathfrak{D}}$ to simply $\sum$, since there are no ambiguities and the summation is used exclusively for this purpose.

E.2.1 Label Quantities

In the case of disjoint concepts, we can use the same equation as in the sample-based case described in Section 2.3.1, since these are already aggregated. For the common quantities, the modification consists of pre-computing the aggregate value per label rather than per sample. This requires computing the dataset-wide sum only for atomic concepts, while all higher-arity labels can be derived through

arithmetic operations over these sums. Therefore:

$$|I^C_{min}(L)_x| = \begin{cases} min(\sum |I^C_{min}(L_{\leftarrow})_x|, \sum |I^C(L_{\rightarrow})_x|) & \text{if } \oplus = OR \\ max(\sum |I^C_{min}(L_{\leftarrow})_x| + \sum |I^C(L_{\rightarrow})_x| - |N^C|, 0) & \text{if } \oplus = AND \\ max(\sum |I^C_{min}(L_{\leftarrow})_x| - \sum |I^C(L_{\rightarrow})_x|, 0) & \text{if } \oplus = AND\ NOT \end{cases} \tag{45}$$

$$|I^C_{max}(L)_x| = \begin{cases} min(\sum |I^C_{max}(L_{\leftarrow})_x| + \sum |I^C(L_{\rightarrow})_x|, |N^C|) & \text{if } \oplus = OR \\ min(\sum |I^C_{max}(L_{\leftarrow})_x|, \sum |I^C(L_{\rightarrow})_x|) & \text{if } \oplus = AND \\ min(\sum |I^C_{max}(L_{\leftarrow})_x)|, |N^C| - \sum |I^C(L_{\rightarrow})_x)| & \text{if } \oplus = AND\ NOT \end{cases} \tag{46}$$

$$|E^C_{min}(L)_x| = \begin{cases} max(\sum |E^C_{min}(L_{\leftarrow})_x|, \sum |E^C(L_{\rightarrow})_x|) & \text{if } \oplus = OR \\ max(\sum |E^C(L_{\leftarrow})_x| + \sum |E^C(L_{\rightarrow})_x| - |SE^C|, 0) & \text{if } \oplus = AND \\ max(\sum |E^C_{min}(L_{\leftarrow})_x| - \sum |E^C(L_{\rightarrow})_x|, 0) & \text{if } \oplus = AND\ NOT \end{cases} \tag{47}$$

$$|E^C_{max}(L)_x| = \begin{cases} min(\sum |E^C_{max}(L_{\leftarrow})_x| + \sum |E^C(L_{\rightarrow})_x|, |SE^C|) & \text{if } \oplus = OR \\ min(\sum |E^C_{max}(L_{\leftarrow})_x|, \sum |E^C(L_{\rightarrow})_x|) & \text{if } \oplus = AND \\ min(\sum |E^C_{max}(L_{\leftarrow})_x|, |SE^C| - \sum |E^C(L_{\rightarrow})_x|) & \text{if } \oplus = AND\ NOT \end{cases} \tag{48}$$

The derivation of these quantities follows the same rationale as the sample computation, and the only difference is the larger overestimation produced by the aggregation of label-wise quantities.

E.2.2 Path Quantities

Similarly to the sample-based computation, computing the path $dIoU$ requires estimating the maximum possible improvement. In this case, however, the Top^A and $Bott^A$ vectors are computed concept-wise rather than per sample. Specifically, for each quantity, we sort the values of all individual concepts in the dataset and compute cumulative sums into the Top^A and $Bott^A$ vectors, starting from the largest and smallest values, respectively. Substituting these into the equations of the sample-based computation, we obtain:

$$I_{min}(L) = \begin{cases} max(\sum |I^C_{min}(L)_x| + \sum |I^U_{min}(L)_x|, & \\ \quad Bott^A_1(I^C) + Bott^A_1(I^U)) & \text{OR Path} \\ 0 & \text{AND Path} \\ \sum |I^U_{min}(L)_x| & \text{AND NOT Path} \end{cases} \tag{49}$$

$$I_{max}(L) = \begin{cases} min(\sum |I^C_{max}(L)_x| + Top^A_t(I^C), |N^C|) + & \\ \quad min(\sum |I^U_{max}(L)_x| + Top^A_t(I^U), |N^U|) & \text{OR Path} \\ min(\sum |I^C_{max}(L)_x|, Top^A_1(I^C)) & \text{AND Path} \\ \sum |I^U_{max}(L)_x| + min(\sum |I^C_{max}(L)_x|, & \\ \quad |N^C| - Bott^A_1(I^C)) & \text{AND NOT Path} \end{cases} \tag{50}$$

$$|Union_{min}(L)| = |\boldsymbol{N}| + \begin{cases} max(\sum |E^C_{min}(L)_x| + \sum |E^U_{min}(L)_x|, & \\ \quad Bott^A_1(E^C) + Bott^A_1(E^U)) & \text{OR Path} \\ max(\sum |E^C_{min}(L)_x| + Bott^A_1(E^C) - |SE^C|, 0) & \text{AND Path} \\ \sum |E^C_{min}(L)_x| & \text{AND NOT Path} \end{cases} \tag{51}$$

$$|Union_{max}(L)| = |\boldsymbol{N}| + \begin{cases} min(\sum |E^C_{max}(L)_x| + Top^A_t(E^C), |SE^C|) & \\ \quad + min(\sum |E^U_{max}(L)_x| + Top^A_t(E^U), |SE^U|) & \text{OR Path} \\ min(\sum |E^C_{max}(L)_x|, Top^A_1(E^C)) & \text{AND Path} \\ \sum |E^C_{max}(L)_x| + min(& \\ \quad \sum |E^U_{max}(L)_x|, |SE^C| - Bott^A_1(E^C)) & \text{AND NOT Path} \end{cases} \tag{52}$$

The derivation of these quantities follows the same rationale as the sample computation case, and the only difference is that these represent a larger overestimation. However, note in this case $|Union_{min}(L)|$ can be greater than 0, unlike in the sample computation where $\forall x \in \mathfrak{D}$, $Bott_1(E^C)_x = 0$, except in the rare degenerate case (not observed in any of the datasets tested in this paper) where a single sample contains all concepts in the dataset.

E.3 Path Combinations

As in the previous section, we shorten here the notation $\sum_{x \in \mathfrak{D}}$ to simply $\sum$. As mentioned in the main text, to estimate values for paths involving multiple operators, we select the maximum and minimum of each quantity across the operators considered. For example, when both OR and AND can appear along a path, we compute:

$$dIoU_{max}(OR, AND)_x = \frac{max(\sum |I^{OR}_{max}(L)_x|, \sum |I^{AND}_{max}(L)_x|)}{min(\sum |Union^{OR}_{min}(L)_x|, \sum |Union^{AND}_{min}(L)_x|)} \tag{53}$$

and

$$dIoU_{min}(OR, AND)_x = \frac{min(\sum |I^{OR}_{min}(L)_x|, \sum |I^{AND}_{min}(L)_x|)}{max(\sum |Union^{OR}_{max}(L)_x|, \sum |Union^{AND}_{max}(L)_x|)} \tag{54}$$

These expressions simplify to:

$$dIoU_{max}(OR, AND)_x = \frac{\sum |I^{OR}_{max}(L)_x|}{\sum |Union^{AND}_{min}(L)_x|} \tag{55}$$

and

$$dIoU_{min}(OR, AND)_x = \frac{\sum |I^{AND}_{min}(L)_x|}{\sum |Union^{AND}_{max}(L)_x|} \tag{56}$$

We can further rewrite them as:

$$dIoU_{max}(OR, AND)_x = \frac{min(|I^{C}_{max}(L)_x| + Top_t(I^C)_x, |N^C_x|) + min(|I^{U}_{max}(L)_x| + Top_t(I^U)_x, |N^U_x|)}{|\boldsymbol{N}_x|} \tag{57}$$

and

$$dIoU_{min} = 0 \tag{58}$$

Now, let us consider the case where we explicitly design this combined quantity. As before, we observe that $dIoU_{min} = 0$. For the numerator, however, we can refine the estimation: including an AND operator at any step will, by design, remove all the unique quantities of the current label. Thus:

$$dIoU_{max}(OR, AND) = \frac{min(|I^{C}_{max}(L)_x| + Top_t(I^C)_x, |N^C_x|) + Top_t(I^U)_x}{|\boldsymbol{N}_x| + (|E^{C}_{max}(L)_x| + |Bott_1(E^C)_x| - |SE^C_x|} \tag{59}$$

However, since $|Bott_1(E^C)_x|$ is 0 in most datasets, the denominator reduces to $|\boldsymbol{N}_x|$. Thus, the only practical difference lies in the numerator, where the $|I^U_{max}(L)_x|$ term is missing. However, in general, $Top_t(I^U)_x$ is much larger than $|I^U_{max}(L)_x|$, since it includes the maximum per sample across all concepts in the dataset. This makes the difference between $dIoU^{(OR,AND)}_{max}$ (from explicit design) and our proposed simplification relatively small. Similar observations can be made for all the other combinations of the operators considered in this paper.

From a practical perspective, and given that the optimal algorithm often converges toward breadth-first exploration (Section 3.3), the gain of the refined design is marginal. Conversely, our simplified approach, based on combining the values of exclusive paths of operators, scales more efficiently and facilitates future extensions. Indeed, new logic operators can be incorporated into the optimal algorithm simply by providing their estimates for the exclusive path.

F Examples of Explanations Difference

This section presents examples of the differences between explanations computed by beam search and those obtained with the optimal algorithm. The examples are not cherry-picked; rather, they correspond to the first ten differing explanations for the last convolutional layer of each explained model (ResNet18 [7], AlexNet [10], and DenseNet [9]). All the models have been pretrained on the Place365 dataset.

#RESNET18

Unit 30
M-MESH:
((balcony-interior-s OR control_tower-indoor-s) OR dinette-home-s)
(0.087)
Optimal: ((table AND dining_room-s) OR balcony-interior-s) (0.100)

Unit 39
M-MESH: ((bed AND NOT black-c) OR pillow) (0.043)
Optimal: ((pillow OR pool table) OR swimming pool) (0.047)

Unit 41
M-MESH: ((butchers_shop-s OR rubble-s) OR meat) (0.069)
Optimal: ((pink-c AND mountain) OR butchers_shop-s) (0.069)

Unit 45
M-MESH: ((house AND NOT manufactured_home-s) OR roof) (0.163)
Optimal: ((house AND NOT garage-outdoor-s) OR roof) (0.163)

Unit 56
M-MESH: ((table AND dining_room-s) OR lamp) (0.045)
Optimal: ((table OR chair) AND dining_room-s) (0.051)

Unit 70
M-MESH: ((bed OR ball_pit-s) OR pillow) (0.061)
Optimal: ((bed AND NOT brown-c) OR ball_pit-s) (0.061)

Unit 87
M-MESH: ((alley-s OR corridor-s) AND NOT wall) (0.079)
Optimal: ((floor AND corridor-s) OR alley-s) (0.091)

Unit 94
M-MESH: ((plant OR field) AND green-c) (0.028)
Optimal:
((swimming_pool-indoor-s OR rope_bridge-s) OR hedge_maze-s) (0.029)

Unit 100
M-MESH: ((sink OR countertop) OR bathtub) (0.103)
Optimal: ((mirror OR sink) AND bathroom-s) (0.105)

Unit 109
M-MESH: ((art_gallery-s OR drawing) AND NOT ceiling) (0.231)
Optimal: ((painting AND museum-indoor-s) OR art_gallery-s) (0.238)

ALEXNET

Unit 2
M-MESH: ((floor AND yellow-c) OR ballroom-s) (0.043)
Optimal: ((yellow-c OR airport_terminal-s) AND floor) (0.052)

Unit 3
M-MESH: ((light OR podium-indoor-s) OR fluorescent) (0.050)
Optimal: ((green-c AND ceiling) OR light) (0.053)

Unit 20
M-MESH: ((person AND NOT black-c) AND NOT brown-c) (0.029)
Optimal: ((grey-c OR white-c) AND person) (0.029)

```
Unit 21
M-MESH: ((road AND street-s) AND NOT white-c) (0.028)
Optimal: ((waiting_room-s OR poolroom-home-s) AND floor) (0.031)

Unit 22
M-MESH: ((ceiling AND living_room-s) AND NOT black-c) (0.041)
Optimal: ((bedroom-s OR living_room-s) AND ceiling) (0.047)

Unit 26
M-MESH: ((pool table OR ball_pit-s) OR day_care_center-s) (0.037)
Optimal: ((purple-c AND ceiling) OR pool table) (0.038)

Unit 29
M-MESH: ((path OR platform) OR forest_road-s) (0.031)
Optimal: ((white-c AND person) OR path) (0.035)

Unit 31
M-MESH:
((skyscraper AND NOT building_facade-s) OR downtown-s) (0.058)
Optimal: ((blue-c AND skyscraper-s) OR skyscraper) (0.064)

Unit 37
M-MESH: ((ceiling AND black-c) OR pagoda-s) (0.053)
Optimal: ((black-c OR red-c) AND ceiling) (0.059)

Unit 48
M-MESH: ((bed AND NOT brown-c) AND NOT black-c) (0.050)
Optimal: ((white-c OR blue-c) AND bed) (0.051)


# DENSENET161

Unit 1
M-MESH: ((tree AND grey-c) OR ski_resort-s) (0.026)
Optimal: ((grey-c OR blue-c) AND tree) (0.028)

Unit 3
M-MESH: ((drawer OR boxing_ring-s) AND NOT cabinet) (0.074)
Optimal: ((drawer AND NOT cabinet) OR boxing_ring-s) (0.074)

Unit 4
M-MESH: ((tent OR batters_box-s) AND NOT grass) (0.028)
Optimal: ((earth AND batters_box-s) OR tent) (0.030)

Unit 5
M-MESH: ((mountain AND blue-c) AND NOT coast-s) (0.038)
Optimal: ((blue-c OR highway-s) AND mountain) (0.039)

Unit 7
M-MESH: ((floor AND black-c) OR swimming pool) (0.021)
Optimal: ((black-c OR blue-c) AND floor) (0.022)

Unit 8
M-MESH: ((floor AND bedroom-s) OR forest_road-s) (0.012)
Optimal: ((bedroom-s OR supermarket-s) AND floor) (0.013)

Unit 9
M-MESH: ((bakery-shop-s OR sconce) OR lighthouse) (0.026)
Optimal: ((food OR sconce) OR patty) (0.027)
```

Unit 15
M-MESH: ((harbor-s OR hay) AND NOT sky) (0.023)
Optimal: ((harbor-s AND NOT sky) OR hay) (0.023)

Unit 22
M-MESH: ((attic-s AND NOT floor) AND NOT bed) (0.047)
Optimal: ((wall OR ceiling) AND attic-s) (0.054)

Unit 24
M-MESH: ((drawer AND NOT grey-c) AND NOT white-c) (0.043)
Optimal: ((table AND bedroom-s) OR drawer) (0.045)

Algorithm 1: Optimal Algorithm

Input: $\mathfrak{L}^1$, $\boldsymbol{N}$, **M**, DisjointMatrix, length
Output: BestLabel,BestIoU

Frontier ← empty priority queue
ConceptQuantities, Memory ← empty lists
MinIoU, RecentIoU ← 0
for $c_{k,i}$ ***in*** $\mathfrak{L}^1$ **do**
    ConceptQuantities[$c_{k,i}$] ← `compute_quantities`($c_{k,i}$, **M**, $\boldsymbol{N}$)
    Paths ← `estimate_aggregate_paths`(ConceptQuantities[$c_{k,i}$], length, MinIoU)
    Frontier.`add`(Paths)
    MinIoU ← `update_min`(Paths, MinIoU)
end
Frontier ← `reduce_frontier`(Frontier,MinIoU)
while Frontier *is not empty* **do**
    Node ← Frontier.`pop()`
    if Node *is aggregate_estimation* **then**
        UpdatedNode ← `compute_sample_estimate`(Node,MinIoU)
        MinIoU ← `update_min`(UpdatedNode, MinIoU)
        if UpdatedNode.*max_iou* $>$ MinIoU **then**
            Frontier.`add`(UpdatedNode)
        end
        continue
    end
    UpdatedNode ← `apply_logic_equivalences`(Node)
    if UpdatedNode.*max_iou* $<$ Node.*max_iou* ***and*** UpdatedNode.*max_iou* $>$ MinIoU **then**
        MinIoU ← `update_min`(UpdatedNode, MinIoU)
        Frontier.`add`(UpdatedNode)
        continue
    end
    if Node.*iou* == RecentIoU **then**
        if Node ***in*** Memory **then**
            continue
        end
        else
            Memory.`add`(Node)
        end
    end
    else
        Memory ← empty list
        RecentIoU ← Node.max_iou
    end
    if Node *is final* **then**
        TreeQuantities = `compute_tree_quantities`(Node)
        Frontier ← `update_by_tree`(Frontier, TreeQuantities)
        IoU = `compute_iou`(TreeQuantities.`get`(Node)
        if IoU $>$ BestIoU **then**
            BestIoU = IoU
            BestLabel = Node.label
        end
    end
    else
        AdditionalNodes ← `expand`(Node)
        Paths ← `estimate_aggregate_paths`(AdditionalNodes, Quantities, length,MinIoU)
        Frontier.`add`(Paths)
        if `min(`Paths`)` $>$ MinIoU **then**
            MinIoU ← `min`(Paths)
            Frontier ← `reduce_frontier`(Frontier, MinIoU)
        end
    end
end
return *BestLabel, BestIoU*

Algorithm 2: Our Informed Beam Search Algorithm

Input: $\mathfrak{L}^1$, $\boldsymbol{N}$, $\mathbf{M}$, DisjointMatrix, b, length
Output: BestLabel,BestIoU

Beam $\leftarrow$ empty list
ConceptsQuantities $\leftarrow$ empty list
for $c_{k,i}$ ***in*** $\mathfrak{L}^1$ **do**
    Quantities $\leftarrow$ `compute_quantities`$(c_{k,i}, \boldsymbol{N}, \mathbf{M})$
    ConceptsQuantities`.append`(*Quantities*)
    IoU $\leftarrow$ `compute_dIoU`(*Quantities*)
    Beam`.add`$(label = c_{k,i}, iou = IoU)$
end
`sort`(*Beam*) `# Sort by IoU`
`# Select the best b candidates`
Beam $\leftarrow$ Beam[:b]
MinIoU $\leftarrow$ `find_min`(*Beam*)
for *2* ***to*** length **do**
    SearchSpace $\leftarrow$ `expand_beam`(*Beam*, $\mathfrak{L}^1$)
    Estimations $\leftarrow$ `estimate_labels_iou`(*SearchSpace*, *ConceptQuantities*, , *DisjointMatrix*)
    `sort`(*Estimations*)
    for L, EstIoU ***in*** Estimations **do**
        if EstIoU $<$ MinIoU **then**
            `# All the other labels cannot be added to the beam`
            break
        end
        Iou $\leftarrow$ `compute_iou`(L, $\boldsymbol{N}$, $\mathbf{M}$)
        Beam`.add`(label=L, iou=*Iou*)
    end
    `sort`(*Beam*)
    `# Select the best b candidates`
    Beam $\leftarrow$ Beam[:b]
    `# Compute and update info`
    MinIoU $\leftarrow$ `find_min`(*Beam*)
end
BestLabel, BestIoU $\leftarrow$ `max`(*Beam*)
return *BestLabel, BestIoU*