RGF: Recursive Generative Framework for the Edge-Cut Separable Problems

Bo Peng **Rongzhen Ye** **Yaohua Li** **Weilin Luo***

Hai Wan* **Jiahao Xu** **Zexian Liang**

School of Computer Science and Engineering
Sun Yat-sen University, Guangzhou, China
{pengb8, yerzh, liyh363, xujh228, liangzx5}@mail2.sysu.edu.cn
{luowlin5, wanhai}@mail.sysu.edu.cn

Abstract

The Maximum Independent Set (MIS), Dominating Set (DS), and Vertex Cover (VC) problems are fundamental combinatorial optimization problems with wide-ranging applications. We identify a key property of these problems: by finding a cut in the graph and determining the states of vertices associated with the cut, the graph can be decomposed into multiple subgraphs whose solutions are independent of each other. Therefore, we classify MIS, DS, and VC as Edge-Cut Separable Problems (ECSPs). This property enables a natural divide-and-conquer strategy, which significantly reduces the computational cost and allows neural networks to scale to large-scale graphs that were previously intractable. Based on this property, we propose a recursive framework that leverages this property to approximately solve ECSPs, namely RGF. Our framework consists of two major components: a partitioner that identifies a cut in the graph, and a predictor that solves the cut-determination sub-problem (CDSP) to determine the states of vertices associated with the cut. By recursively applying these components, RGF generates a complete solution for the original ECSP instance. Additionally, through partial prediction, the memory consumption is significantly reduced, enabling RGF to handle real-world large graphs. Experimental results validate the advantages of RGF on real-world large graphs with up to 51M vertices: it reaches a 0.008–0.120% gap to the KaMIS heuristic and outperforms direct neural inference on all 12 benchmarks where the latter can be run.

1 Introduction

The *Maximum Independent Set* (MIS), the *Dominating Set* (DS), and the *Vertex Cover* (VC) problem are important *combinatorial optimization problems* (COPs). Many applications rely on solving these problems, such as very-large-scale integration [26], software verification [39], and bioinformatics [5].

We observed a key property of these problems: by finding a cut in the graph and determining the states of vertices associated with the cut, the graph can be decomposed into multiple subgraphs whose solutions are independent of each other. This property allows us to classify MIS, DS, and VC as *Edge-Cut Separable Problems* (ECSPs). For instance, in an MIS problem, identifying a cut and assigning values to its vertices enables us to solve smaller subgraphs separately, reducing computational cost.

*Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

ECSPs on general graphs are NP-hard [16, 2, 12], making exact solutions computationally expensive. Consequently, approximate solvers are widely used in practice.

The development of *Machine Learning* (ML) technologies has introduced new approximate methods for COPs, including ECSPs. Notable examples include `Intel` [33], `DIMES` [35], `DGL` [6], `LwD` [1], `GFlowNets` [42], `DIFUSCO` [37], `T2T` [31], and `FastT2T` [32]. Compared to traditional solvers, ML-based solvers can directly predict solutions according to the graph structure. However, this feature is not fully exploited in existing works. ML-based COP solvers can be broadly categorized into two types: *autoregressive* and *non-autoregressive* approaches, according to their solution generation methods. Autoregressive methods sequentially determine variables until a complete solution is constructed, resulting in $\mathcal{O}(n)$ neural network (NN) inferences. Non-autoregressive methods predict the soft constraints of the entire solution in one step NN inference, but they require post-processing algorithms to ensure feasibility.

The above ML-based solvers do not explicitly exploit the subgraph independence property of ECSPs. We hypothesize that this property supports a divide-and-conquer paradigm that focuses on local substructures, which can simplify the learning objective, improve scalability to large graphs, and enhance out-of-distribution generalization. Therefore, we propose a recursive framework that leverages this property to approximately solve ECSPs, namely `RGF`. Our framework consists of two major components: a partitioner that identifies a cut in the graph, and a predictor that solves the cut-determination sub-problem (CDSP) to determine the states of vertices associated with the cut. By recursively applying these components, our framework generates a complete solution for the original ECSP instance.

Compared to autoregressive approaches, which typically require $\mathcal{O}(n)$ NN inferences, `RGF` needs only $O(\log n)$ NN inferences. Since we apply non-autoregressive models as predictors, compared to native non-autoregressive approaches, `RGF` better leverages NN inference by solving a CDSP over cut-related vertices, achieving improved performance. Our experiments support the hypothesis that exploiting subgraph independence improves scalability and solution quality: on real-world large graphs with up to 51M vertices, `RGF` reaches a 0.008–0.120% gap to KaMIS, outperforms direct neural inference on all 12 benchmarks where the latter can be run, and keeps its memory bounded where a single forward pass runs out of memory; on synthetic benchmarks it consistently improves over `DIFUSCO`.

Our contributions are summarized as follows:

- We identified and analyzed the subproblem independence property of ECSPs.
- Based on this property, we proposed `RGF`, a recursive framework for solving ECSPs.
- We evaluated `RGF` on real-world large graphs with up to 51M vertices, substantially expanding the scale boundary of neural-network-based methods for these problems, where it achieves a small gap to the KaMIS heuristic and consistently outperforms direct neural inference; on synthetic MIS/DS/VC benchmarks it consistently improves over `DIFUSCO`.

2 Preliminaries

Consider an undirected graph $G = (V, E)$, where $V = \{v_0, \ldots, v_{n-1}\}$ is the set of vertices, n is the number of vertices, and E is the set of edges. Each edge $e \in E$ connects a pair of vertices, denoted as $e = (v_i, v_j)$ where $v_i, v_j \in V$. We formally define the three problems in ECSP, *i.e.* MIS, DS, VC problems as in Definitions 1, 2, and 3.

Definition 1 (MIS)**.** *Given G, the MIS problem seeks a subset of vertices $S \subseteq V$ of maximum size $|S|$ such that no two vertices in S are adjacent:*

$$\forall (v_i, v_j) \in E, \quad v_i \in S \implies v_j \notin S. \tag{1}$$

Intuitively, an independent set is a set of vertices in a graph, no two of which are connected by an edge. The MIS problem aims to find the largest such set.

Definition 2 (DS)**.** *Given G, the DS problem seeks a subset of vertices $S \subseteq V$ of minimum size $|S|$ such that every vertex not in S is adjacent to at least one vertex in S:*

$$\forall v_i \in V \setminus S, \quad \exists v_j \in S \text{ such that } (v_i, v_j) \in E. \tag{2}$$

In essence, a dominating set ensures global coverage through a single-hop adjacency. The DS problem minimizes this selection such that every vertex in G is either part of the set or neighboring one.

Definition 3 (VC)**.** *Given G, the VC problem seeks a subset of vertices $S \subseteq V$ of minimum size $|S|$ such that every edge in G is incident to at least one vertex in S:*

$$\forall (v_i, v_j) \in E, \quad v_i \in S \vee v_j \in S. \tag{3}$$

Essentially, VC targets the minimum vertex set required to guard or cover the entire edge set E, ensuring that no edge exists without at least one endpoint being selected.

Equipped with these three concrete problems, we can now formalize the class of problems to which our framework applies.

Definition 4 (Edge-Cut Separable Problem (ECSP))**.** *A combinatorial optimization problem on an undirected graph $G = (V, E)$ is an* Edge-Cut Separable Problem *(ECSP) if there exists a* partitioning rule $\mathcal{R}$ *such that, for any cut $C \subseteq E$ and any valid assignment $S_C \subseteq V_C$ (where V_C is the vertex set incident to C), the subgraphs $G_1, \ldots, G_m$ obtained by applying $\mathcal{R}(G, C, S_C)$ have mutually independent solutions: the optimal solution on each G_i can be computed independently of all G_j ($j \neq i$), and the union of the subgraph solutions together with S_C forms a valid solution for G.*

MIS, DS, and VC each instantiate this definition with a specific partitioning rule (Equations (5)–(6) below); Proposition 1 proves subgraph independence for all three, and Theorem 1 shows that recursively solving CDSPs preserves completeness for any ECSP.

A solution S can be represented by a binary vector $\mathbf{s} \in \{0, 1\}^n$, where $\mathbf{s}_i = 1$ if $v_i \in S$ and $\mathbf{s}_i = 0$ otherwise. This representation facilitates NN processing in our framework.

3 The Cut-Determination Sub-problem

We first introduce the theoretical foundation of our approach: the *cut-determination sub-problem* (CDSP) of ECSPs, as defined in Definition 5.

Definition 5 (The CDSP of ECSPs)**.** *Given an ECSP graph instance G and a cut $C \subseteq E$ (a subset of edges), let $V_C \subseteq V$ denote the set of vertices incident to edges in C, i.e., $V_C = \{v \in V \mid \exists (u, v) \in C\}$. The CDSP for (G, C) seeks a vertex subset $S_C \subseteq V_C$ such that there exists an optimal solution S^* for G satisfying the constraint:*

$$S_C \subseteq S^* \wedge (V_C \setminus S_C) \subseteq (V \setminus S^*). \tag{4}$$

In essence, the CDSP aims to identify a subset of vertices $S_C \subseteq V_C$ that forms part of an optimal solution S^* for G. This identified subset S_C is derived from the vertices associated with the cut C.

The fundamental challenge of CDSP is that the optimal solution S^* is unknown at the time of solving. This makes CDSP a prediction problem: we must infer the correct states of vertices in V_C based solely on the graph structure. This is the motivation for employing a neural network predictor $\mathtt{F}_\theta$ to approximate the solution of CDSP.

By removing the cut edges and their incident vertices, the graph G is partitioned into multiple connected subgraphs $G_1, G_2, \ldots$, with no edges connecting any pair of these subgraphs. Different ECSPs employ distinct partitioning rules, as detailed below.

MIS. For a CDSP (G, C) based on a MIS instance G with solution S_C, the graph G can be partitioned by removing the vertex set defined in Equation (5):

$$\{v \in V \mid v \in V_C \vee \exists (u, v) \in E \text{ such that } u \in S_C\}. \tag{5}$$

Under the MIS constraints, Equation (5) prunes the graph by removing both the fixed vertices V_C and those forced to zero by their neighbors. Specifically, if a vertex $u \in S_C$ is included in the independent set ($u = 1$), all its neighbors $v \notin V_C$ must be excluded to maintain feasibility, rendering them redundant for subsequent sub-problems.

DS. For a CDSP (G, C) based on a DS instance G with solution S_C, the graph G is partitioned by removing the cut edges C. No vertices are removed, but those in V_C must be labeled with their determined value. That is, the CDSP of DS requires vertex labels as input, with three possible labels: $\{\text{Unknown}, 1, 0\}$.

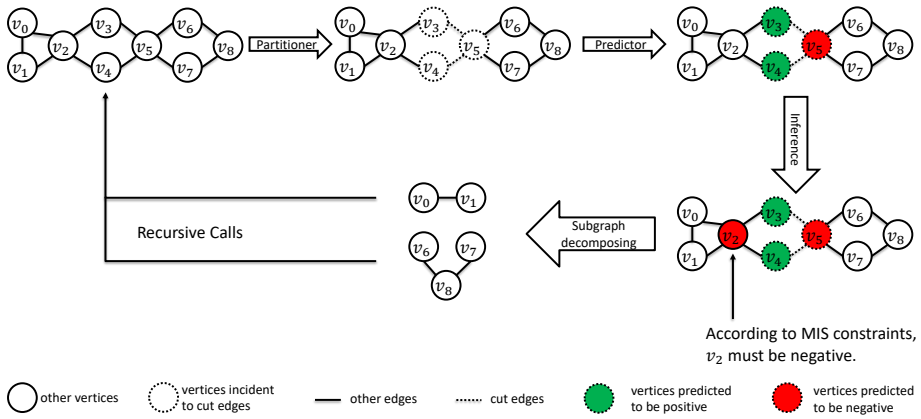

Figure 1: The overall process of RGF. Given a graph G, we first identify a cut using the partitioner. Cut edges and vertices connected by cut edges are marked with dotted lines. Predicting these vertices forms a CDSP. We then apply a predictor to solve the CDSP, *i.e.*, predict the states of these vertices. Vertices assigned 1 are marked in green, while those assigned 0 are marked in red. After prediction, we infer the states of nearby vertices, according to the MIS constraints. Then we remove all vertices that are predicted or inferenced in previous steps, and the corresponding edges. The remaining graph forms at least two connected subgraphs. We recursively apply this framework to each subgraph.

VC. For a CDSP (G, C) based on a VC instance G with solution S_C, the graph G is partitioned by removing the vertex set defined in Equation (6):

$$\{v \in V \mid v \in V_C \vee \exists (u, v) \in E \text{ such that } u \in V_C \setminus S_C\}. \tag{6}$$

Equation (6) indicates that two types of vertices are removed: (1) $v \in V_C$, which are already determined, and (2) $v \notin V_C$ that have a neighbor $u \in V_C \setminus S_C$. According to Definition 3, the latter vertices must be assigned 1 because their determined neighbor u is assigned 0.

Proposition 1 (Independence of Subgraph Solutions)**.** *The solution for any subgraph G_i is independent of the solutions for all other subgraphs G_j where $i \neq j$.*

This elimination process effectively decouples the remaining graph into disconnected components. As Proposition 1 formalizes, the independence of these subgraphs provides the theoretical foundation for a scalable, recursive, or parallelized solution strategy.

4 Recursive Framework for ECSPs

The subgraph independence property is critical for ML-based ECSP solvers. It enables a divide-and-conquer strategy that simplifies the learning objective, enhances model scalability for large graphs, and improves out-of-distribution generalization by focusing on local substructures. Based on this property, we propose a recursive framework that leverages this property to approximately solve ECSPs, namely RGF.

Figure 1 illustrates RGF using an MIS example. RGF consists of two main components: a graph partitioner that identifies a cut in the graph, and a predictor that predicts the states of vertices in the cut. In RGF, each recursive process consists of three main steps: (1) identifying a cut, (2) solving the CDSP, and (3) obtaining subgraphs.

Here we provide an example of our framework solving a simple MIS instance in Example 1.

Example 1 (Example of our framework solving a simple MIS instance)**.** *Given a graph $G = (V, E)$ where $V = \{v_0, v_1, \ldots, v_8\}$ and $E = \{(v_0, v_1), (v_0, v_2), (v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_5), (v_4, v_5), (v_5, v_6), (v_5, v_7), (v_6, v_8), (v_7, v_8)\}$. This is an undirected graph, and for conciseness we omit listing the reverse edges in E. The graph structure matches Figure 1.*

To solve this instance, our framework first finds a cut C. In this example, $C = \{(v_3, v_5), (v_4, v_5)\}$, so $V_C = \{v_3, v_4, v_5\}$. We then invoke $\mathtt{F}_\theta$ to predict the states of vertices in V_C. Suppose $\mathtt{F}_\theta$ correctly predicts $s_3 = s_4 = 1$ and $s_5 = 0$. Although v_2 is not in V_C, it is connected to v_3 (which has $s_3 = 1$). According to the MIS constraints, v_2 must have $s_2 = 0$. Next, we remove all vertices that are predicted or inferred in previous steps, and their corresponding edges from the graph.

The remaining graph forms two subgraphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$, where $V_0 = \{v_0, v_1\}$, $E_0 = \{(v_0, v_1)\}$, $V_1 = \{v_6, v_7, v_8\}$, and $E_1 = \{(v_6, v_8), (v_7, v_8)\}$. The MIS solutions for G_0 and G_1 are independent, so we recursively apply our framework to solve these subgraphs.

More details of our framework are introduced below.

4.1 Framework Details and Complexity Analysis

Algorithm 1 Recursive Generative Framework for ECSPs

Require: Graph $G = (V, E)$, threshold α, exact solver $\mathtt{S}^*$, learnable solver $\mathtt{F}_\theta$.
Ensure: Approximate solution S.

```
1: function RGF(G = (V, E), α, S*, F_θ)
2:     if |V| ≤ α then
3:         return S*(G)
4:     end if
5:     C, V_C ← PARTITIONER(G)
6:     S_C ← F_θ(G, V_C)
7:     G_1, G_2, ..., G_m ← GETSUBGRAPHS(G, V_C, S_C)
8:     for all G_i ∈ {G_1, G_2, ..., G_m} do
9:         S_i ← RGF(G_i, α, S*, F_θ)
10:    end for
11:    return ⋃_{i=1}^{m} S_i
12: end function
```

The framework is detailed in Algorithm 1.

The hyperparameter α determines when the exact solver $\mathtt{S}^*$ is invoked. As shown in line 2, if the number of vertices in the current graph G is less than or equal to α, $\mathtt{S}^*$ is called to compute an exact solution for G. While $\mathtt{S}^*$ has exponential time complexity, it may still be faster than invoking the neural network $\mathtt{F}_\theta$ for very small graphs.

Line 5 involves calling the partitioner to split G into (at least) two parts, returning the corresponding cut C and the vertices V_C. The objective of this partitioning is to balance the size of the resulting subgraphs while minimizing the number of cut edges. Finding an optimal partition is NP-hard; fortunately, we do not require an exact solution. Thus, approximate partitioning methods are suitable. Line 6 invokes the predictor $\mathtt{F}_\theta$ to obtain an approximate solution for the CDSP instance (G, V_C).

Line 7 entails generating subgraphs based on the cut C and the approximate solution S_C. The specific implementation of this subgraph generation varies depending on the particular ECSP variant, as detailed in the preceding section.

Line 8 recursively applies the framework to solve each subgraph's ECSP instance.

We now analyze the time complexity of the framework. The key is determining the recursion depth. Since we use an approximate graph partitioning tool that cannot guarantee balanced partitions, the worst-case iteration depth reaches n (when each partition separates a single vertex). However, the $\mathcal{O}(\log n)$ recursion depth requires only a weak assumption: there exists a constant $\epsilon \in [0.5, 1)$ such that every partition produced by the partitioner yields subgraphs of size at most $\epsilon|V|$. Under this assumption, the maximum subgraph size shrinks by a factor of at most ϵ at each recursion level, giving a maximum depth of $\log_{1/\epsilon} n = -\frac{1}{\log \epsilon} \log n = \mathcal{O}(\log n)$. This condition is satisfied by any non-degenerate partitioner, including the size-balanced random partitioning we use.

Another critical factor is the number of neural network (NN) inferences. At first glance, this appears to be $\mathcal{O}(n)$ since depth i would require 2^i inferences. However: (1) subgraphs at the same depth are independent; (2) the combined size of subgraphs at any depth does not exceed the original graph size. Thus, we can parallelize inferences across all subgraphs at a given depth by treating them as a single

(disconnected) graph, requiring only one NN inference. This approach leverages a common batching technique in GNNs.

The complete complexity analysis is summarized as follows: given an ECSP instance $G = (V, E)$ with $|V| = n$, the time complexity of Algorithm 1 is $\mathcal{O}((\mathcal{T}_C(G) + \mathcal{T}_F(G)) \log n)$, where $\mathcal{T}_C(G)$ denotes the time complexity of the graph partitioning algorithm, and $\mathcal{T}_F(G)$ denotes the time complexity of the learnable solver $\mathtt{F}_\theta$. A rigorous proof is provided in Appendix A.

Since approximate graph partitioning algorithms with polynomial time complexity are readily available [22], and assuming the learnable solver $\mathtt{F}_\theta$ (typically a neural network) also operates in polynomial time, the overall framework operates in polynomial time complexity as an approximation algorithm. We emphasize that this is a runtime guarantee for our approximate solver, not a claim of solving NP-hard problems exactly.

A learned predictor is approximate, so it may output an invalid CDSP solution S_C that cannot be extended to any valid solution of G. We therefore insert a legality-repair hook that only modifies decisions on V_C, enforcing independence inside V_C for MIS, coverage of internal V_C edges for VC, and domination of predicted-0 cut vertices for DS. It runs in $\mathcal{O}(|V_C| \log |V_C| + |E(V_C)|)$, preserves the algorithmic structure and the completeness argument because the repaired S_C is again a valid CDSP solution, and is detailed in Appendix F.

4.2 Scaling to Real-world Large Graphs

Industrial-scale instances often involve millions of vertices, which can impose excessive memory demands on the learnable solver $\mathtt{F}_\theta$, particularly on resource-constrained computing nodes. Our framework addresses this challenge by modifying line 6 in Algorithm 1. Instead of supplying the entire graph G to $\mathtt{F}_\theta$, we only provide the subgraph $\mathcal{N}_G(V_C)$ comprising the neighborhood of the selected vertex set V_C, details are shown in Algorithm 2. This localized approach ensures memory requirements remain within feasible limits.

Algorithm 2 Finding $\mathcal{N}_G(V_C)$

Require: Graph $G = (V, E)$, size of neighborhood β, selected vertex set V_C.
Ensure: The neighborhood subgraph $\mathcal{N}_G(V_C)$ of size β.

```
1: function GetNeighborhood(G = (V, E), V_C, β)
2:     V_N ← V_C
3:     Q ← EmptyQueue()
4:     for all v ∈ V_C do
5:         Q.Insert(v)
6:     end for
7:     while |V_N| < β ∧ ¬Q.IsEmpty() do
8:         v ← Q.Front()
9:         Q.Pop()
10:        for all u ∈ {u ∈ V \ V_N | (u, v) ∈ E} do
11:            V_N ← V_N ∪ {u}
12:            Q.Insert(u)
13:        end for
14:    end while
15:    E_N ← {(u, v) ∈ E | u ∈ V_N ∧ v ∈ V_N}
16:    return N_G(V_C) = (V_N, E_N)
17: end function
```

The hyperparameter β in Algorithm 2 controls the maximum size of the neighborhood subgraph $\mathcal{N}_G(V_C)$ and is an engineering parameter determined by hardware memory constraints, not a theoretical one. A smaller β reduces memory consumption but restricts the receptive field of the predictor $\mathtt{F}_\theta$, as vertices beyond the β-bounded neighborhood are invisible to it, potentially degrading prediction accuracy. Conversely, a larger β provides richer contextual information and improves accuracy, but increases memory usage. In practice, we recommend setting β to the largest value that fits within the available GPU memory. This design allows RGF to trade off between prediction quality and hardware constraints, enabling deployment on resource-limited devices while preserving scalability.

5 Experiments and Results

5.1 Datasets and Experimental Settings

Datasets. Our main benchmarks are real-world large graphs, whose edge-cut structure keeps the CDSP small relative to the instance: street networks from the 10th DIMACS Implementation Challenge benchmark[1] [4], road networks, and web graphs. Table 2 lists the evaluated instances together with their sizes. To evaluate all three ECSP variants and the full pipeline on synthetic data, we also use RB graphs with 200–300 vertices [41], and Erdős–Rényi (ER) graphs [14] generated with 700–800 vertices and connection probability 0.15.

Evaluation metrics. We use KaMIS [22] as the reference heuristic and report the drop, i.e., the relative gap to the KaMIS solution, together with wall-clock time. For DS and VC, reference solutions are computed with a MILP solver and by complementing the MIS reference, respectively.

Experimental setup. We use the pre-trained `FastT2T` (hidden size 128 , $(T_s, T_g) = (1, 0)$) as the predictor, METIS 2-way partitioning, the neighborhood radius β set to the largest value permitted by the GPU memory budget of each run (12–24 GB; $\beta = 512{,}000$ in all real-world runs), and $\alpha = 500$ by default for the synthetic ablations; the real-world results in Table 2 use the tuned $\alpha = 10^4$, and the effect of α is studied in RQ3. Unless stated otherwise, the real-world graph results are averaged over 5 seeds (seeds $\{0, 1, 2, 3, 4\}$) and use a deterministic execution mode: the random seeds of METIS, the exact solver and the predictor are fixed, CUDA kernels are required to be deterministic, and TF32 is disabled. The synthetic RB/ER experiments follow the original training and inference procedures of prior work, are averaged over 5 independent runs, and are not run in a deterministic mode. Experiments run on a workstation with an AMD Threadripper 3970X CPU, two NVIDIA RTX 3090 GPUs, and 128GB RAM. Our code is available at `https://github.com/citrontyan/RGF`.

5.2 Research Questions and Results

RQ1: Overall Performance (Gap, Time). Table 1 reports the overall performance of `RGF` on the MIS, DS and VC tasks, on RB and ER graphs. Compared with direct `DIFUSCO` [37], `RGF+DIFUSCO` improves the drop in all six settings, most clearly on DS/ER. DS is harder than MIS/VC for one-shot predictors: domination is a covering constraint, so a single missed vertex can force many additional vertices and local decoding errors amplify. The standalone predictors are therefore weak on DS, especially on dense ER graphs. RGF confines decisions to cuts and solves leaves exactly, reducing the DS/ER drop from 66.76% to 38.82%; the residual gap tracks the strength of the underlying DS predictor. DS is also the most expensive row per ER graph because its recursion is deeper: MIS/ER and VC/ER terminate after fewer levels with fewer predictor calls and no exact leaves, whereas DS removes no vertices when the cut is applied, so the remaining subgraphs stay above α and the recursion continues for more levels, requiring more predictor calls and many exact MILP leaves per graph. Against the stronger `FastT2T` [32] baseline, `RGF+FastT2T` is better on DS and VC/RB, close on VC/ER, and slightly worse on MIS/RB. The recent task-specific predictors `COExpander` [34] and `GenSCO` [30] are stronger than DIFUSCO/FastT2T on MIS; RGF with either is comparable within noise to its standalone counterpart (Table 1), and neither predictor was trained for DS or VC.

RQ2: Quality and Scalability on Real-World Large Graphs . Table 2 reports the results. On street, road and web graphs, `RGF` reaches small relative gaps, while the direct neural baseline is far worse wherever it can be run; over all feasible benchmarks, `RGF` is better on every one. For larger graphs, a single full-graph forward pass no longer fits in GPU memory and either spills or runs out of memory, whereas `RGF` is bounded by the CDSP and scales to graphs with 51M vertices. The same table also reports a same-wall-clock plain random-walk baseline: it is surprisingly competitive on several smaller street networks and even achieves a lower drop than `RGF` on six of the sixteen instances, but `RGF` is better on the larger road networks, the two web graphs, and most other instances, with an average gap roughly half that of RW. This indicates that simple local search can be competitive on small, low-degree graphs, whereas the advantage of `RGF` grows with graph scale and degree.

[1] https://sites.cc.gatech.edu/dimacs10/downloads.shtml

Table 1: Overall performance on RB and ER graph. The better drop is marked in **bold**.

Dataset	Method	MIS		DS		VC	
		Drop (%)	Time (s)	Drop (%)	Time (s)	Drop (%)	Time (s)
RB	DIFUSCO	2.92±0.10	7.5	18.12±0.93	7.4	1.57±0.05	7.7
	RGF+DIFUSCO	**2.41±0.15**	8.9	**18.06±0.31**	10.2	**1.30±0.05**	8.7
	FastT2T	**0.88±0.19**	1.9	26.28±0.42	0.1	0.48±0.01	0.1
	RGF+FastT2T	0.99±0.12	5.7	**25.76±0.28**	0.9	**0.46±0.01**	0.8
	COExpander	**2.44±0.62**	0.3	–	–	–	–
	RGF+COExpander	2.50±0.33	0.7	–	–	–	–
	GenSCO	0.82±0.22	7.4	–	–	–	–
	RGF+GenSCO	**0.80±0.27**	7.5	–	–	–	–
ER	DIFUSCO	8.46±0.28	21.5	66.76±0.44	21.4	0.54±0.02	21.2
	RGF+DIFUSCO	**8.26±0.23**	22.3	**38.82±0.49**	122.9	**0.53±0.01**	21.2
	FastT2T	6.56±0.26	3.5	59.25±0.63	0.2	**1.03±0.02**	0.2
	RGF+FastT2T	**6.48±0.28**	9.0	**36.78±0.55**	7.6	1.04±0.01	1.5
	COExpander	6.68±0.43	1.2	–	–	–	–
	RGF+COExpander	**6.55±0.32**	1.6	–	–	–	–
	GenSCO	5.18±0.61	7.7	–	–	–	–
	RGF+GenSCO	**5.06±0.45**	9.2	–	–	–	–

– indicates that the predictor was not trained on DS or VC.

Table 2: Quality and scalability on real-world large MIS graphs. d is the average degree; "direct' applies FastT2T to the whole graph in a single forward pass; RW is a plain random walk with the same wall-clock budget as RGF. The better of RGF and RW is marked in **bold**.

Instance	$\|V\|$	$\|E\|$	d	Time (s)	RGF drop (%)	RW drop (%)	direct drop (%)
luxembourg.osm	114,599	119,666	2.09	9.0	0.060±0.006	**0.025±0.004**	5.26
belgium.osm	1,441,295	1,549,970	2.15	108.3	0.090±0.003	**0.049±0.002**	4.47
netherlands.osm	2,216,688	2,441,238	2.20	169.1	0.062±0.001	**0.046±0.001**	3.63†
italy.osm	6,686,493	7,013,978	2.10	475.8	**0.052±0.002**	0.116±0.002	6.26†
great-britain.osm	7,733,822	8,156,517	2.11	539.4	0.058±0.001	**0.049±0.001**	3.81†
germany.osm	11,548,845	12,369,181	2.14	886.5	0.066±0.001	**0.053±0.001**	×
asia.osm	11,950,757	12,711,603	2.13	940.2	**0.035±0.001**	0.137±0.005	×
europe.osm	50,912,018	54,054,660	2.12	3 767.6	**0.056±0.001**	0.103±0.002	×
GAP-road	23,947,347	28,854,312	2.41	2 272.5	**0.042±0.001**	0.111±0.002	×
roadNet-TX	1,379,917	1,921,660	2.79	152.5	**0.120±0.006**	0.237±0.006	2.60†
roadNet-CA	1,965,206	2,766,607	2.82	242.6	**0.120±0.002**	0.261±0.004	2.81†
roadNet-PA	1,088,092	1,541,898	2.83	150.4	**0.115±0.002**	0.255±0.008	2.60
web-NotreDame	325,729	1,090,108	6.69	73.1	**0.008±0.001**	0.103±0.028	0.45
web-Stanford	255,265	1,941,926	15.21	40.3	**0.078±0.016**	0.377±0.033	2.15
web-BerkStan	654,782	6,581,871	20.11	1,076.2	**0.188±0.030**	0.501±0.057	1.76†
web-Google	855,802	4,291,352	10.03	175.6	0.155±0.006	**0.123±0.013**	0.59†

† Direct run required spilling to host memory.
× Direct run out of memory even with host-memory spilling.

RQ3: The Exact-Solver Threshold α. Table 3 reports the per-instance effect of increasing α from 500 to 10^4 on the seven smaller street networks. The larger threshold reduces the drop on every instance and shrinks the gap several-fold, with a comparable wall clock; the paired improvement is statistically significant and is explained by the reduction of $|V_{\mathrm{NN}}|$ predicted by the bound of Appendix E. The gain is instance-dependent, so α should be calibrated per graph family.

RQ4: Partitioning Strategies. We vary the number of METIS parts k at each recursive step using the same three street networks, $\alpha = 500$, FastT2T, five seeds; this ablation is separate from the $\alpha = 10^4$ protocol of Table 2. Increasing k from 2 to 3 reduces the recursion depth and shortens the wall clock, but it enlarges the first-level cut, so the predictor must decide more vertices in a single call. Table 4 shows that the net effect on quality is negative on the two larger networks and neutral on the smallest one, consistent with the bound of Theorem 3, which is linear in the number of predicted vertices. The deeper recursion of 2-way partitioning is therefore the better trade-off, and

Table 3: Effect of the exact-solver threshold α on the seven smaller street networks

Instance	$\lvert V\rvert$	d	Drop α=500 (%)	Drop α=10^4 (%)	Time α=500 (s)	Time α=10^4 (s)	Δ drop
luxembourg.osm	114,599	2.09	0.331±0.023	0.060±0.006	25.6	21	+0.271
belgium.osm	1,441,295	2.15	0.424±0.007	0.090±0.003	436.6	325	+0.334
netherlands.osm	2,216,688	2.20	0.366±0.005	0.062±0.001	529.6	578	+0.304
italy.osm	6,686,493	2.10	0.310±0.005	0.052±0.002	1 533.2	1 522	+0.258
great-britain.osm	7,733,822	2.11	0.288±0.005	0.058±0.001	1 774.0	1 376	+0.230
germany.osm	11,548,845	2.14	0.320±0.006	0.066±0.001	2 907.8	2 317	+0.254
asia.osm	11,950,757	2.13	0.284±0.004	0.035±0.001	3 091.7	3 845	+0.249

we keep $k = 2$ in all experiments. The partitioner ablation in Table 5 compares METIS, random size-balanced partitioning, and single-vertex partitioning: METIS is best or tied on MIS and VC/RB; random partitioning is better on ER DS but METIS is better on RB DS; and single-vertex partitioning is competitive only on VC/ER.

Table 4: Effect of the METIS partition count k on three street networks.

Instance	Drop k=2 (%)	Drop k=3 (%)	Time k=2 (s)	Time k=3 (s)
luxembourg.osm	0.331±0.023	0.329±0.031	25.6	14.9
belgium.osm	0.424±0.007	0.528±0.009	436.6	259.4
netherlands.osm	0.366±0.005	0.427±0.009	529.6	397.2

Table 5: Impact of different partitioning strategies. Best results in each column are marked in **bold**.

Dataset	Method	MIS		DS		VC	
		Drop (%)	Time (s)	Drop (%)	Time (s)	Drop (%)	Time (s)
RB	RGF+METIS	**1.01±0.13**	2.39±0.06	**25.76±0.28**	0.66±0.02	**0.46±0.01**	0.52±0.01
	RGF+Random	4.85±0.15	**0.21±0.07**	28.60±0.19	**0.18±0.03**	3.22±0.01	**0.24±0.08**
	RGF+Single-vertex	6.38±0.29	1.60±0.01	27.24±0.37	6.14±0.20	2.07±0.03	0.33±0.02
ER	RGF+METIS	**6.48±0.28**	4.18±0.04	36.78±0.55	5.63±0.26	1.04±0.01	0.84±0.00
	RGF+Random	16.42±0.21	**0.52±0.14**	**34.87±0.67**	**1.45±0.07**	1.04±0.01	**0.69±0.02**
	RGF+Single-vertex	15.25±0.47	12.47±0.21	36.90±1.36	33.77±2.78	**0.98±0.03**	27.70±24.44

6 Related Work

6.1 ECSPs

The ECSPs, namely MIS, DS, and VC, as well as the *Maximum Clique* problem, are polynomial-time reducible to one another. As important COPs, these problems have been extensively studied for decades. Exact algorithms typically employ branch and bound [36, 38]. Due to their NP-hard complexity, numerous approximation algorithms have been proposed [3, 28, 8, 18]. In recent years, several ML-based approaches focusing on MIS have been proposed [1, 7], while others model multiple COPs, which we discuss in the next subsection.

6.2 ML for COPs

Current learning-based solvers for COPs are broadly categorized into constructive approaches and improvement-based approaches [32]. Constructive methods comprise: (1) autoregressive techniques [23, 25, 27, 24, 19] that sequentially construct solutions through iterative variable assignments, and (2) non-autoregressive techniques [21, 15, 17, 35, 37, 43] that generate soft-constrained solutions in a single inference pass, requiring subsequent post-processing to ensure feasibility. Improvement-based solvers [11, 40, 9, 29, 20] employ learned policies to iteratively refine solutions via local search operators, directly optimizing the objective function.

Recent advances leverage generative modeling for COPs, capitalizing on its representational capacity to estimate solution distributions conditioned on problem instances [19, 10, 37, 13, 42, 31]. Notably, diffusion-based frameworks like `DIFUSCO` [37] achieve state-of-the-art performance on tasks such as the Traveling Salesman Problem (TSP) and MIS. However, these methods exhibit a critical

limitation: they fail to integrate instance-specific search paradigms to exploit the full potential of learned distributions. While the T2T framework [31] addresses this partially through objective-guided gradient search, its step-by-step inference remains computationally intensive. FastT2T [32] alleviates efficiency concerns via one-step inference but relies on heuristic post-processing. More recent diffusion-based solvers target stronger solution quality: COExpander [34] adapts solution expansion during decoding, and GenSCO [30] treats generation as a search operator for test-time scaling. In our experiments, both are used as strong MIS predictors.

A parallel line of work explores divide-and-conquer strategies for scaling ML-based COP solvers. Notably, UDC [44] proposes a unified neural divide-and-conquer framework that first generates a global initial solution, then extracts sub-problems constrained by this initial state, and iteratively refines the sub-problem solutions to improve the global solution. This mechanism is fundamentally an iterative refinement process: the sub-problem boundaries are intrinsically tied to the current global solution, and the conquering stage depends on full-graph information. Consequently, UDC inevitably loads all vertices and edges to generate the global initial solution, which becomes a scalability bottleneck. In contrast, RGF is a topologically recursive framework that structurally decomposes the graph without relying on any global initial solution. At each recursive step, RGF identifies a cut based on the graph topology and solves the CDSP using only local neighborhoods of the cut, allowing sub-problems to be processed independently and in parallel.

7 Limitations

RGF trades raw speed for scale: KaMIS is much faster on moderate graphs, and the gap of RGF remains nonzero on most real-world instances even with a tuned threshold. Our method is most effective on sparse graphs with small cuts: on dense graphs, the cut can be large, so the CDSP can approach full-graph inference and the scalability advantage diminishes. The best α is instance-dependent and chosen empirically; an automatic rule is left for future work. On several small street networks, a same-wall-clock random walk matches or beats RGF. On synthetic benchmarks, strong task-specific predictors such as COExpander and GenSCO retain their quality inside RGF but RGF gives no gain at that scale; the error analysis covers MIS, and large-scale DS/VC remain future work.

8 Conclusion and Future Work

We identified that the Maximum Independent Set, Dominating Set, and Vertex Cover problems share a subgraph independence property and categorized them as Edge-Cut Separable Problems (ECSPs). Based on this property, we proposed RGF, a recursive framework that uses a partitioner to identify a cut and a neural predictor to solve the resulting Cut-Determination Sub-Problem (CDSP). Experiments on real-world large graphs with up to 51M vertices show that RGF reaches a 0.008–0.120% gap to KaMIS, outperforms direct neural inference on all 12 benchmarks where the latter can be run, and keeps its memory bounded where a single forward pass runs out of memory; on synthetic benchmarks it consistently improves over DIFUSCO . Crucially, because the CDSP only requires subgraphs near cut edges rather than the full graph for inference, RGF substantially reduces memory consumption, which is what enables this scalability. Moreover, predictor errors accumulate linearly rather than exponentially, with the exact solver acting as an error barrier .

For future work, we plan to apply RGF to industrial problems such as Set Cover (reducible to VC) , to develop post-processing methods for further quality improvement.

Acknowledgments and Disclosure of Funding

Our experiments were conducted on an RTAI cluster, which is supported by the School of Computer Science and Engineering as well as the Institute of Artificial Intelligence, Sun Yat-sen University. This article was supported by the National Natural Science Foundation of China (No. 62276284). It was also supported by the Guangxi Key R&D Program (No. guikeFN2600640488), the National Natural Science Foundation of China (No. 62502554), the Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515011470, 2025A1515011639, 2026A1515012083), the Guangzhou Science and Technology Project (No. 202201011699), and the Shenzhen Science and Technology Program (No. KJZD20230923114059O2). The authors declare no competing interests.

References

[1] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 134–144, 2020.

[2] Robert B. Allan and Renu C. Laskar. On domination and independent domination numbers of a graph. *Discret. Math.*, 23(2):73–76, 1978.

[3] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012.

[4] David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. 2018.

[5] Francisco Barahona, Andrés Weintraub, and Rafael Epstein. Habitat dispersion in forest planning and the stable set problem. *Oper. Res.*, 40(Supplement-1):S14–S21, 1992.

[6] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

[7] Lorenzo Brusca, Lars C. P. M. Quaedvlieg, Stratis Skoulakis, Grigorios Chrysos, and Volkan Cevher. Maximum independent set: Self-training through dynamic programming. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[8] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1181–1196, 2017.

[9] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6278–6289, 2019.

[10] Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. The policy-gradient placement and generative routing neural networks for chip design. In *Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[11] Paulo R. de O. da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Proceedings of The 12th Asian Conference on Machine Learning, ACML 2020, 18-20 November 2020, Bangkok, Thailand*, pages 465–480, 2020.

[12] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[13] Xingbo Du, Chonghua Wang, Ruizhe Zhong, and Junchi Yan. Hubrouter: Learning global routing via hub generation and pin-hub connection. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[14] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[15] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *AAAI 2021, IAAI 2021, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7474–7482, 2021.

[16] M. R. Garey and David S. Johnson. "strong" np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[17] Simon Geisler, Johanna Sommer, Jan Schuchardt, Aleksandar Bojchevski, and Stephan Günnemann. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

[18] Demian Hespe, Christian Schulz, and Darren Strash. Scalable kernelization for maximum independent sets. *ACM J. Exp. Algorithmics*, 24(1):1.16:1–1.16:22, 2019.

[19] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[20] Qingchun Hou, Jingwei Yang, Yiqiang Su, Xiaoqing Wang, and Yuming Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

[21] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *CoRR*, abs/1906.01227, 2019.

[22] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[23] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017.

[24] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In *Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[25] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[26] Sun-Yuan Kung, Shiann-Ning Jean, and Chih-Wei Jim Chang. Fault-tolerant array processors using single-track switches. *IEEE Trans. Computers*, 38(4):501–514, 1989.

[27] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: policy optimization with multiple optima for reinforcement learning. In *Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[28] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017.

[29] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 26198–26211, 2021.

[30] Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *Advances in Neural Information Processing Systems (NeurIPS 2025)*, 2025.

[31] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[32] Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast T2T: optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[33] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 537–546, 2018.

[34] Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. COExpander: Adaptive solution expansion for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, 2025.

[35] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[36] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.*, 38(2):571–581, 2011.

[37] Zhiqing Sun and Yiming Yang. DIFUSCO: graph-based diffusion solvers for combinatorial optimization. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[38] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings*, pages 191–203, 2010.

[39] Miroslav N. Velev. Using rewriting rules and positive equality to formally verify wide-issue out-of-order microprocessors with a reorder buffer. In *2002 Design, Automation and Test in Europe Conference and Exposition (DATE 2002), 4-8 March 2002, Paris, France*, pages 28–35, 2002.

[40] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Trans. Neural Networks Learn. Syst.*, 33(9):5057–5069, 2022.

[41] Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.

[42] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C. Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. In *Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[43] Xinhao Zheng, Yang Li, Cunxin Fan, Huaijin Wu, Xinhao Song, and Junchi Yan. Learning plaintext-ciphertext cryptographic problems via anf-based SAT instance representation. In *Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[44] Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. Udc: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 6081–6125. Curran Associates, Inc., 2024.

A Detailed Proofs

A.1 Proof of Proposition 1 (Independence of Subgraph Solutions)

Proof. We prove the proposition for each ECSP variant separately. Let $G = (V, E)$ be the original graph, $C \subseteq E$ be a cut, $V_C \subseteq V$ be the vertices incident to edges in C, and $S_C \subseteq V_C$ be a solution to the CDSP for (G, C). After removing cut edges and determined vertices according to the partitioning rules in § 3, the remaining graph decomposes into connected components $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_m = (V_m, E_m)$.

MIS. By Equation (5), the removed vertex set is $R = \{v \in V \mid v \in V_C \vee \exists (u, v) \in E \text{ such that } u \in S_C\}$. We show that no edge exists between any V_i and V_j for $i \neq j$. Suppose, for contradiction, there exists an edge $(a, b) \in E$ with $a \in V_i$, $b \in V_j$. Since E_i only contains edges within V_i, we have $(a, b) \notin E_i$ and $(a, b) \notin E_j$, meaning (a, b) must be a cut edge that was removed. But then both a and b would belong to $V_C \subseteq R$, contradicting $a \in V_i$ and $b \in V_j$. Furthermore, suppose $a \in V_i$ and $b \in R$ due to being a neighbor of some $u \in S_C$. Then a would also be removed since it is incident to a cut edge (a, b), implying $a \in V_C \subseteq R$. Thus, no inter-component edges remain.

Since G_i and G_j share no vertices or edges, the MIS constraint (Eq. 1) for vertices in V_i only involves edges within E_i. The selection of vertices in $S_i \subseteq V_i$ does not affect the feasibility of $S_j \subseteq V_j$, because no edge connects a vertex in V_i to a vertex in V_j. Therefore, the MIS solutions for G_i and G_j are independent.

DS. By the DS partitioning rule, only cut edges C are removed; all vertices remain. We show that after removing C, G_i and G_j are disconnected. By definition of a cut, removing C disconnects the graph into at least two components. Any edge (a, b) with $a \in V_i$, $b \in V_j$ must have been in C, and is therefore removed. Thus, no edge connects V_i to V_j.

Since no edges connect different subgraphs, the domination constraint (Eq. 2) for a vertex $v \in V_i$ only depends on edges within E_i and the labels of vertices in V_i (vertices in $V_C \cap V_i$ have known labels from S_C). The selection of dominating vertices in V_i does not affect the domination of vertices in V_j. Therefore, the DS solutions are independent across subgraphs.

One subtlety concerns *cross-component domination*. Definition 5 decides the label of every vertex in V_C, and a component V_i may contain a vertex that is adjacent, through a removed cut edge, to a decided vertex of another component whose label is 1. In that case the domination constraint of the vertex is already satisfied by the decided vertex, and the subproblem must treat it as dominated instead of satisfying it a second time; otherwise the subproblem optimum need not be consistent with the global optimum. Our implementation therefore propagates an "already dominated" flag into each subproblem (see Appendix F). With this flag, the domination constraint for $v \in V_i$ depends only on edges within E_i and on the labels and domination status of vertices in V_i, so the independence argument and the completeness proof below both hold.

VC. By Equation (6), the removed vertex set is $R = \{v \in V \mid v \in V_C \vee \exists (u, v) \in E \text{ such that } u \in V_C \setminus S_C\}$. Similar to the MIS case, any edge connecting V_i to V_j would be a cut edge, making both endpoints belong to $V_C \subseteq R$. Any edge connecting V_i to a vertex b that is removed because its neighbor $u \in V_C \setminus S_C$ is assigned 0 would imply b is incident to u, making b adjacent to a vertex in V_C, hence $b \in V_C \subseteq R$. Therefore, no inter-component edges remain.

Since G_i and G_j share no edges, the VC constraint (Eq. 3) for edges in E_i only depends on vertices in V_i. The selection of vertices in $S_i \subseteq V_i$ does not affect the coverage of edges in E_j. Thus, the VC solutions are independent across subgraphs.

In all three cases, the solution for G_i is independent of the solution for G_j ($i \neq j$), completing the proof. □

A.2 Completeness of Our Approach

We present the completeness analysis of our proposed framework in Theorem 1.

Theorem 1 (Completeness of Algorithm 1)**.** *Algorithm 1 is guaranteed to find a complete solution, provided that the learnable solver F_θ is complete.*

Theorem 1 demonstrates that our framework preserves the completeness of the underlying solver. It does not introduce additional errors or failure points beyond those inherent in the solver itself. The detailed proof is provided below.

A.3 Proof of Theorem 1 (Completeness of Algorithm 1)

Proof. We prove by strong induction on $|V|$, the number of vertices in the input graph.

Base case: $|V| \leq \alpha$. By line 2 of Algorithm 1, the exact solver $\mathtt{S}^*$ is invoked. By definition, $\mathtt{S}^*$ returns a complete solution for any ECSP instance. Thus, the base case holds.

Inductive hypothesis. Assume Algorithm 1 returns a complete solution for all input graphs with strictly fewer than k vertices, for some $k > \alpha$.

Inductive step. Consider a graph $G = (V, E)$ with $|V| = k > \alpha$. Algorithm 1 proceeds as follows:

1. **Cut identification** (line 5): A partitioner identifies a cut $C \subseteq E$ and the associated vertex set V_C. Since $k > \alpha$, a non-trivial cut is produced (otherwise the graph would be trivial and could be solved directly).
2. **CDSP solving** (line 6): The learnable solver $\mathtt{F}_\theta$ solves the CDSP (G, V_C), producing $S_C \subseteq V_C$. By assumption, $\mathtt{F}_\theta$ is complete, meaning S_C satisfies Eq. (4): there exists an optimal solution S^* for G such that $S_C \subseteq S^*$ and $(V_C \setminus S_C) \subseteq (V \setminus S^*)$.
3. **Subgraph generation** (line 7): Based on S_C and C, subgraphs $G_1, \ldots, G_m$ are generated according to the partitioning rules in § 3. Since $V_C \neq \emptyset$, at least one vertex is removed from G, so $|V_i| < |V| = k$ for all i.
4. **Recursive solving** (line 8): For each G_i, the algorithm recursively calls itself. By the inductive hypothesis, each recursive call returns a complete solution S_i for G_i.
5. **Solution assembly**: The final solution is $S = S_C \cup \bigcup_{i=1}^{m} S_i \cup S_{\text{inf}}$, where S_{inf} denotes vertices whose states are deterministically inferred from S_C (e.g., in MIS, neighbors of vertices assigned 1 in S_C are assigned 0).

We now show S is complete for G. By Theorem 1, the subgraphs $G_1, \ldots, G_m$ have mutually independent solutions. The CDSP solution S_C identifies vertices in V_C that are consistent with some optimal solution S^*. The inferred vertices S_{inf} are deterministically forced by S_C and the problem constraints (Definitions 1–3), so they too are consistent with S^*.

Every vertex $v \in V$ falls into exactly one of the following categories: (a) $v \in V_C$, covered by S_C; (b) v is deterministically inferred from S_C, covered by S_{inf}; (c) $v \in V_i$ for some i, covered by S_i. Therefore, every vertex receives an assignment, and by the inductive hypothesis together with the CDSP guarantee, all assignments are consistent with the problem constraints. Hence, S is a complete solution for G.

By the principle of strong induction, Algorithm 1 returns a complete solution for any input graph, provided $\mathtt{F}_\theta$ is complete. □

A.4 Proof of the Time Complexity of Algorithm 1

Proof. Let $T(n)$ denote the worst-case time complexity of Algorithm 1 on a graph with n vertices.

Base case. When $n \leq \alpha$, the algorithm invokes the exact solver $\mathtt{S}^*$ (line 2). Let $\mathcal{T}_S(n)$ be the time complexity of $\mathtt{S}^*$. Since α is a constant, $\mathcal{T}_S(n) = \mathcal{O}(1)$ for $n \leq \alpha$.

Recursive case. For $n > \alpha$, the algorithm executes:

1. **Partitioning** (line 5): Invokes the partitioner on G, taking time $\mathcal{T}_C(n)$.
2. **CDSP prediction** (line 6): Invokes the learnable solver on (G, V_C), taking time $\mathcal{T}_F(n)$.

3. **Subgraph generation** (line 7): Extracts subgraphs in time $\mathcal{O}(n+m)$ (linear in vertices and edges).

4. **Recursive calls** (line 8): Recursively solves each subgraph G_i with n_i vertices, where $\sum_i n_i < n$ (since $|V_C| \geq 1$ for any non-trivial cut).

Recursion depth. Under the assumption that the partitioner produces a cut such that each subgraph has at most ϵn vertices for some constant $\epsilon \in [0.5, 1)$, the size of the maximum subgraph shrinks by a factor of at most ϵ at each recursion level. Starting from n, after d levels the maximum subgraph size is at most $\epsilon^d n$. The recursion terminates when $\epsilon^d n \leq \alpha$, giving $d \leq \log_{1/\epsilon}(n/\alpha) = \mathcal{O}(\log n)$. Thus, the recursion depth is $\mathcal{O}(\log n)$.

Work per depth. At depth d, let the subgraphs be $G_1^{(d)}, G_2^{(d)}, \ldots$ with vertex counts $n_1^{(d)}, n_2^{(d)}, \ldots$. These subgraphs are vertex-disjoint, so $\sum_j n_j^{(d)} \leq n$. The partitioner and learnable solver are invoked on each subgraph at depth d. Assuming both $\mathcal{T}_C$ and $\mathcal{T}_F$ are at least linear in the graph size (a standard assumption for graph algorithms and neural networks), we have:

$$\sum\nolimits_j \mathcal{T}_C\left(n_j^{(d)}\right) \leq \mathcal{T}_C(n), \tag{7}$$

$$\sum\nolimits_j \mathcal{T}_F\left(n_j^{(d)}\right) \leq \mathcal{T}_F(n). \tag{8}$$

The linear-time subgraph extraction at each depth also sums to $\mathcal{O}(n)$. Thus, the total work at each depth is bounded by $\mathcal{T}_C(n) + \mathcal{T}_F(n) + \mathcal{O}(n)$.

Batching of NN inferences. As noted in the main text, the subgraphs at the same depth are independent and can be treated as a single disconnected graph for NN inference. This means $\mathtt{F}_\theta$ is invoked only once per depth, not once per subgraph. The combined graph has at most n vertices, so $\mathcal{T}_F$ at each depth is $\mathcal{T}_F(n)$, consistent with the analysis above.

Total complexity. Summing over all $\mathcal{O}(\log n)$ depth levels:

$$T(n) = \sum_{d=0}^{\mathcal{O}(\log n)} \big(\mathcal{T}_C(n) + \mathcal{T}_F(n) + \mathcal{O}(n)\big) = \mathcal{O}\big((\mathcal{T}_C(n) + \mathcal{T}_F(n)) \log n\big). \tag{9}$$

This completes the proof. □

B Training Details and Predictor Configurations

Unless stated otherwise, all predictors follow the original training procedures, and for MIS we directly use the pretrained weights released by the authors. For DS and VC, we retrain the predictors on the corresponding problem following the original procedures, replacing the MIS training samples with DS/VC data.

`DIFUSCO`. `DIFUSCO` [37] is a discrete diffusion model. It defines a forward noising process $q(x_t \mid x_0)$ over $T = 1000$ steps, progressively corrupting the optimal solution x^* into uniform noise via a doubly stochastic transition matrix. An anisotropic GNN with edge gating is trained to reverse this process, predicting $p_\theta(x_{t-1} \mid x_t, G)$, by minimizing the variational upper bound of the negative log-likelihood. Inference uses a shortened cosine schedule (typically 50–100 steps). We refer the reader to the original paper for full details.

`FastT2T`. `FastT2T` [32] is an optimization consistency model that shares the forward noising process of diffusion models, but its training objective is closer to supervised learning: instead of predicting the previous noise level x_{t-1}, the network directly predicts the optimal solution x^* from a noised input. Concretely, the loss is $\mathrm{BCE}(f_\theta(x_t, t, G), x^*)$ computed over two independently noised copies of x^* at different noise levels. This enables single-step inference with an anisotropic GNN backbone.

Training variants of `FastT2T` for the CDSP interface. During `RGF` inference the predictor only outputs the states of the cut vertices V_C, whereas the released predictors are trained on complete solutions x^*. We evaluate two retraining variants, both keeping the `FastT2T` architecture and consistency objective. *CDSP-mask retraining (MIS):* random cuts C are drawn from the known solution, V_C is masked as the prediction target, and the loss is restricted to V_C; it converged only on RB (20 epochs). *Task-specific retraining (DS, VC):* we retrain `FastT2T` on the target problem following the original procedure, replacing the MIS training samples with DS or VC data. No `DIFUSCO` weights were retrained with CDSP supervision; these `FastT2T` variants are summarized in Table 6 and kept separate from the synthetic results. MIS/ER and VC/ER have no archived weights: CDSP-mask was not trained on ER because the METIS cut covers essentially all vertices, so partial-solution supervision degenerates to the complete-solution objective, and only an RB version of the VC-retrained checkpoint exists; these combinations are not reproducible and are omitted rather than filled with the official `FastT2T` weights.

Table 6: Reproducible retrained variants of `FastT2T` reported separately from the synthetic results. Drop is the relative gap to KaMIS in percent and time is the wall clock per run on the same fixed subset (first 100 RB / 64 ER test graphs; mean $\pm$ standard deviation over five runs). DS drops use the HiGHS MILP reference. MIS/ER and VC/ER are not reproducible.

Variant	Dataset	Drop (%)	Time (s) $\pm$ std
MIS, CDSP-mask retrained	RB	1.76$\pm$0.21	232.7$\pm$1.4 s
DS, task-retrained	RB	29.51$\pm$0.14	144.5$\pm$1.5 s
DS, task-retrained	ER	36.13$\pm$0.54	359.1$\pm$23.0 s
VC, task-retrained	RB	2.61$\pm$0.01	51.5$\pm$1.6 s

C Memory Usage Comparison

Table 7 reports the GPU memory consumption of the predictors and of the full `RGF` pipeline on the MIS task with full-graph inference (subgraph inference disabled). In this full-graph setting the recursion adds no peak memory: the `RGF` rows match their standalone predictors, because the same full-graph forward pass dominates the peak; enabling CDSP subgraph inference is what decouples memory from graph size on large graphs, where standalone models fail entirely (Section 4.2).

Table 7: GPU memory consumption (MB) for full-graph inference on MIS (RB and ER). Peak allocated memory is measured with `torch.cuda.max_memory_allocated`; for each graph we take the median over three repeats and report the median over the first 20 test graphs of each dataset (batch size 1, subgraph inference disabled).

Method	RB	ER
DIFUSCO (standalone)	68.43	698.80
RGF + DIFUSCO	68.43	698.80
FastT2T (standalone)	34.56	355.08
RGF + FastT2T	34.56	355.08

D Random Walk Baseline

Algorithm 3 specifies the Random Walk (RW) baseline used in the experiments. RW is a randomized local search heuristic: it constructs a maximal independent set by random greedy insertion, then repeatedly attempts local swaps; every time no further vertex can be inserted, the improvement phase is executed. In all large-graph experiments, RW is allocated exactly the same wall-clock budget as `RGF`.

On the real-world benchmarks, this baseline is surprisingly strong and can reach the KaMIS reference solution on several co-author and citation graphs.

Algorithm 3 Random Walk baseline for MIS

Require: Graph $G = (V, E)$, wall-clock budget T
Ensure: Independent set S

$S \leftarrow \emptyset$
while time $< T$ **do**
  $Q \leftarrow \{v \in V \setminus S \mid \forall u \in S, (u, v) \notin E\}$ ▷ candidate set
  if $Q \neq \emptyset$ **then**
    pick $v \in Q$ uniformly at random; $S \leftarrow S \cup \{v\}$
  else
    for all $v \in S$ **do**
      $C_v \leftarrow \{u \in N(v) \setminus S \mid \forall w \in S \setminus \{v\}, (u, w) \notin E\}$
      if $|C_v| \geq 2$ **then**
        $S \leftarrow (S \setminus \{v\}) \cup C_v$; **break**
      end if
    end for
  end if
end while
return S

E Error Propagation Analysis

This section analyzes how errors of the predictor propagate through the recursion. We focus on MIS for concreteness; the DS and VC cases differ only in the error-asymmetry direction and are summarized in Remark 1.

E.1 Notation and Error Model

Let $G = (V, E)$ with $|V| = n$ and let S^* be a fixed optimal solution. The recursion of Algorithm 1 forms a tree $\mathcal{T}$ whose internal nodes have more than α vertices and invoke $\mathtt{F}_\theta$, while leaf nodes have at most α vertices and invoke the exact solver $\mathtt{S}^*$. Let D_{NN} be the number of levels that invoke the predictor; under the ϵ-shrinkage assumption,

$$D_{\mathrm{NN}} = \left\lceil \frac{\log(n/\alpha)}{\log(1/\epsilon)} \right\rceil = \mathcal{O}(\log n). \tag{10}$$

For level d, let $V_j^{(d)}$ be the vertex set of the j-th subgraph, $n_d = \sum_j |V_j^{(d)}|$, and let $V_C^{(d)}$ be the union of the cut-vertex sets at that level; write $\gamma_d = |V_C^{(d)}|/n_d$ for the fraction of vertices predicted at level d, and let η be the fraction of vertices removed at each level (predicted and inferred), so that $n_d \leq (1-\eta)^d n$.

Definition 6 (Prediction error)**.** *At an internal node with cut-vertex set V_C, let $\hat{S}_C$ be the predicted assignment and let S_C^* be the restriction of S^* to V_C. The false-positive set is $\mathcal{E}_{\mathrm{FP}} = \hat{S}_C \setminus S_C^*$, the false-negative set is $\mathcal{E}_{\mathrm{FN}} = S_C^* \setminus \hat{S}_C$, and the error rate is $\rho_{V_C} = (|\mathcal{E}_{\mathrm{FP}}| + |\mathcal{E}_{\mathrm{FN}}|)/|V_C|$. We assume a uniform bound $\rho_{V_C} \leq \rho$ for all internal nodes and denote by β an upper bound on the fraction of false positives among all errors.*

E.2 Single-Node Error Impact

Lemma 1 (Local loss of an internal node, MIS)**.** *Consider an internal node whose subgraph is $G' = (V', E')$ with cut-vertex set V_C, average degree $\bar{\Delta}'$, and prediction errors $\mathcal{E}_{\mathrm{FP}}, \mathcal{E}_{\mathrm{FN}}$. The loss in the final solution size that is attributable to the decisions at this node satisfies*

$$L(G') \;\leq\; |\mathcal{E}_{\mathrm{FN}}| + \sum_{v \in \mathcal{E}_{\mathrm{FP}}} \big(\deg_{G'}(v) - 1\big) \;\leq\; \rho\,|V_C|\,\big((1-\beta) + \beta\bar{\Delta}'\big). \tag{11}$$

Proof. A false negative excludes a vertex $v \in S^*$ from the independent set and fixes it to 0; since v is not added to the solution, its neighbors are not removed from the instance and remain available to the subproblems, so the only loss caused by this decision is the vertex v itself, i.e., at most 1. A false

positive adds $v \notin S^*$ to the solution and forces all neighbors of v that are not in V_C to 0, removing them from the instance; because S^* is a maximal independent set, v has at least one neighbor in S^*, and by the legality repair of Appendix F no such neighbor can be a vertex of V_C assigned 1, so the false positive gains the vertex v but excludes at least one S^* vertex, and at most $\deg_{G'}(v) - 1$ further S^* vertices can be lost. Summing the two terms and using $|\mathcal{E}_{\mathrm{FN}}| + |\mathcal{E}_{\mathrm{FP}}| = \rho_{V_C}|V_C|$ with false-positive fraction at most β gives the second inequality. Overlapping neighborhoods of several false positives can only make the actual loss smaller than the sum of individual bounds, so the union bound remains valid. □

Theorem 2 (Additive error accumulation)**.** *Let $\hat{S}$ be the solution produced by Algorithm 1 under the legality repair, and let $\mathcal{I}$ be the set of internal nodes of the recursion tree. Then*

$$|S^*| - |\hat{S}| \;\le\; \sum_{u \in \mathcal{I}} L(u), \tag{12}$$

and every leaf node contributes zero loss.

Proof. By Proposition 1, the subproblems of different branches are solved independently, so an error made in one branch cannot affect the feasibility or the optimality of another branch; errors propagate only from a node to its own descendants. Every vertex is assigned exactly once during the recursion: it is either predicted at some internal node, or deterministically inferred from such a prediction, or solved by the exact solver at a leaf. A vertex that reaches a leaf is solved optimally for its subproblem, conditioned on the boundary values fixed by its ancestors. Hence the total loss equals the sum of the losses attributable to the predictions at internal nodes, and leaf nodes—solved exactly—introduce no additional loss. □

E.3 Total Expected Loss

Theorem 3 (Expected loss bound)**.** *Assume that (i) the partitioner satisfies the ϵ-shrinkage assumption with $\epsilon \in [0.5, 1)$; (ii) each internal node has error rate at most ρ with false-positive fraction at most β; (iii) the average degree of the subgraphs at level d is $\bar{\Delta}^{(d)}$, and let $\bar{\Delta}$ bound it uniformly; (iv) a fraction γ of vertices per level belongs to V_C and a fraction $\eta \ge \gamma$ is removed. Let α be the exact-solver threshold and $r = \log(1-\eta)/\log\epsilon$. Then*

$$\mathbb{E}\big[|S^*| - |\hat{S}|\big] \;\le\; \rho \cdot |V_{\mathrm{NN}}| \cdot \big((1-\beta) + \beta\bar{\Delta}\big), \tag{13}$$

where the number of vertices decided by the predictor is

$$|V_{\mathrm{NN}}| = \frac{\gamma}{\eta}\, n\, \left(1 - \left(\frac{\alpha}{n}\right)^r\right), \qquad \textit{and} \qquad |V_{\mathrm{exact}}| = n\left(\frac{\alpha}{n}\right)^r. \tag{14}$$

Proof. By Theorem 2 and linearity of expectation, the expected loss equals the sum over internal nodes in $\mathcal{T}$ of their expected local losses. By Lemma 1, a node at level d with cut-vertex set V_C contributes at most $\rho|V_C|((1-\beta) + \beta\bar{\Delta})$. Summing over the nodes of level d gives $\rho\,|V_C^{(d)}|((1-\beta) + \beta\bar{\Delta}) = \rho\gamma_d n_d((1-\beta) + \beta\bar{\Delta})$, and summing over $d = 0, \ldots, D_{\mathrm{NN}} - 1$ yields

$$\mathbb{E}[|S^*| - |\hat{S}|] \le \rho\big((1-\beta) + \beta\bar{\Delta}\big) \sum_{d=0}^{D_{\mathrm{NN}}-1} \gamma_d n_d. \tag{15}$$

With $\gamma_d \le \gamma$ and $n_d = (1-\eta)^d n$, the geometric sum is $\gamma n\,(1 - (1-\eta)^{D_{\mathrm{NN}}})/\eta$. Since $D_{\mathrm{NN}} = \lceil \log(n/\alpha)/\log(1/\epsilon) \rceil$, we have $(1-\eta)^{D_{\mathrm{NN}}} = (\alpha/n)^{\log(1-\eta)/\log\epsilon} = (\alpha/n)^r$, which gives Eq. (14). The relation $|V_{\mathrm{exact}}| = n(\alpha/n)^r$ follows because the total number of vertices in the leaf subgraphs is $n_{D_{\mathrm{NN}}} = (1-\eta)^{D_{\mathrm{NN}}} n$. □

Corollary 1 (No exponential error amplification)**.** *The expected loss is linear in $|V_{\mathrm{NN}}|$ and in ρ; in particular, since $|V_{\mathrm{NN}}| \le n$, $\mathbb{E}[|S^*| - |\hat{S}|] \le \rho\, n\,((1-\beta) + \beta\bar{\Delta})$. Prediction errors are therefore accumulated additively, not multiplicatively: the exact solver at the leaves truncates error propagation along every branch of the recursion.*

Corollary 2 (Sublinear effect of α)**.** *Since $r < 1$ in the relevant regime ($\epsilon < 1$, $\eta < 1$), $|V_{\mathrm{exact}}|$ grows only as α^r and $|V_{\mathrm{NN}}|$ decreases sublinearly with α; the marginal benefit of increasing α therefore diminishes.*

Corollary 3 (Sparse-graph bound and the level-0 cut ratio)**.** *For sparse graphs, $\bar{\Delta} = \mathcal{O}(1)$, so the expected loss reduces to $\mathcal{O}(\rho\,|V_{\mathrm{NN}}|)$. Moreover, increasing α can only reduce the deep levels of the recursion: if $\lambda_0 = |V_C^{(0)}|/|V_{\mathrm{NN}}|$ is the fraction of predicted vertices that come from the first level, the fraction of $|V_{\mathrm{NN}}|$ that α can eliminate is at most $1 - \lambda_0$. This explains the empirically observed asymmetry between street networks (small level-0 cuts, $\lambda_0 \approx 2\%$, so a larger α is very effective) and citation/co-author graphs (large level-0 cuts, $\lambda_0 \approx 53$–68%, so a larger α has little effect); see Appendix G.*

Remark 1 (DS and VC)**.** *The same decomposition applies to DS and VC, with the roles of false positives and false negatives exchanged. For MIS, a false positive is the expensive error (it forces all neighbors to 0, costing up to $O(\bar{\Delta})$), while a false negative costs $O(1)$; for VC the situation is reversed. The corresponding local bound replaces $((1-\beta) + \beta\bar{\Delta})$ by $(\beta + (1-\beta)\bar{\Delta})$ for VC, and by a term bounded by the maximum closed-neighborhood size for DS, where a missed dominating vertex can require additional vertices to dominate its neighborhood. Theorem 3 therefore holds for all three ECSPs with the appropriate constants.*

F Legality of Approximate CDSP Solutions

Theorem 1 assumes that the predictor $\mathtt{F}_\theta$ returns a valid CDSP solution, that is, one that can be extended to an optimal solution of G. Learned predictors are approximate, and an invalid assignment S_C that cannot be extended to any valid solution is worse than a mere quality loss: it can make the recursion ill-defined.

- **MIS.** If two adjacent vertices in V_C are predicted as 1, the independence constraint is violated, and every subproblem created after removing them is inconsistent with the original instance.
- **VC.** If an edge with both endpoints in V_C is predicted as 0–0, both endpoints leave the instance and no later subproblem can cover that edge.
- **DS.** If a cut vertex predicted as 0 has no predicted-1 neighbor inside V_C, then after the cut edges are removed it can never be dominated, and the associated subproblem MILP is infeasible. This failure was observed empirically before the safeguard described below was introduced.

We therefore insert a legality-repair hook between the predictor and the cut. The hook only modifies decisions on V_C, and enforces the minimal conditions above:

- **MIS:** S_C is made independent inside V_C by degree-descending greedy deletion;
- **VC:** every edge internal to V_C has at least one endpoint in S_C;
- **DS:** every vertex predicted as 0 has a predicted-1 neighbor in V_C (vertices are added to S_C as needed).

The cost is $\mathcal{O}(|V_C| \log |V_C| + |E(V_C)|)$, the recursion structure of Algorithm 1 is unchanged, and the completeness statement is preserved, because the repaired S_C is again a valid CDSP solution as required by Definition 5. The hook changes only the values on the cut and never touches a subproblem; the remaining, legal prediction errors are what the analysis in Appendix E bounds. Experimentally, the repair is only needed for the VC and DS variants: without it, VC subproblems are always infeasible and DS subproblems are reported as infeasible, while with it all VC/DS solutions in our RB experiments are feasible.

G Empirical Validation of the Error Analysis

We validate Theorem 3 on real-world large graphs, where the recursion is deep and the exact solver operates on large leaf subgraphs. For every run we record the per-level statistics of the recursion: the number of levels, the number of vertices in the subgraphs at each level, the number of cut vertices $|V_C^{(d)}|$, the number of vertices predicted to be in the solution, and the decisions of the predictor before and after the legality repair. The total number of vertices decided by the predictor

is $|V_{\mathrm{NN}}| = \sum_d |V_C^{(d)}|$; the error rate is $\rho = (\sum_d |\mathcal{E}_{\mathrm{FP}}^{(d)}| + \sum_d |\mathcal{E}_{\mathrm{FN}}^{(d)}|)/|V_{\mathrm{NN}}|$, measured against the KaMIS reference solution, and the observed loss is $L = |S_{\mathrm{ref}}| - |S_{\mathrm{RGF}}|$.

Table 8 reports an α sweep on `luxembourg.osm` with the deterministic execution mode. As α grows from 500 to 50,000, the drop decreases from 0.357% to 0.024% while $|V_{\mathrm{NN}}|$ decreases from 2,092 to 120 (a factor of 17). The error rate increases from 35.3% at $\alpha = 500$ to 50.0% at $\alpha = 50{,}000$, but the normalized cost $L/(\rho|V_{\mathrm{NN}}|)$ stays in $[0.233,\ 0.288]$ (a factor of only 1.24) while $|V_{\mathrm{NN}}|$ changes by $17\times$, precisely the linear relation $L \approx c_{\mathrm{inst}} \cdot \rho \cdot |V_{\mathrm{NN}}|$ predicted by Theorem 3.

Table 8: Empirical validation of the error bound on `luxembourg.osm` (deterministic execution, seed 0). ρ is the measured per-vertex error rate, $L = |S_{\mathrm{ref}}| - |S_{\mathrm{RGF}}|$ the observed loss, and the last column the loss normalized by $\rho|V_{\mathrm{NN}}|$; ρ and L are measured at all listed thresholds.

α	$\lvert V_{\mathrm{NN}}\rvert$	depth	drop (%)	ρ (%)	L	$L/(\rho\lvert V_{\mathrm{NN}}\rvert)$
500	2,092	9	0.357	35.3	206	0.279
1,000	1,331	8	0.238	37.8	137	0.272
2,000	854	7	0.146	34.2	84	0.288
5,000	552	6	0.097	38.6	56	0.263
10,000	362	5	0.066	39.2	38	0.268
20,000	226	4	0.042	42.5	24	0.250
50,000	120	3	0.024	50.0	14	0.233

Table 9 reports the same quantities on two structurally different graph families. Theorem 3 holds in every run: the measured loss never exceeds the per-level bound, and the empirical factor $c_{\mathrm{inst}} = L/(\rho|V_{\mathrm{NN}}|)$ stays below the theoretical worst-case factor $(1-\beta) + \beta\bar{\Delta}$ (0.047–0.288 versus 1.67–8.69). The bound is nevertheless conservative for two reasons. First, it charges every false positive its worst-case cost $\bar{\Delta}$, whereas false positives and false negatives partly cancel in the objective: the identity $L = \sum_d |\mathcal{E}_{\mathrm{FN}}^{(d)}| - \sum_d |\mathcal{E}_{\mathrm{FP}}^{(d)}|$ holds exactly in our runs. Second, ρ is defined relative to one near-optimal KaMIS solution, and an optimal independent set is far from unique: the whole-graph symmetric difference between the `RGF` solution and the reference is 10–26% of the vertices, while the size gap is only 0.09–0.17 percentage points. Many of the counted "errors" are therefore ties between near-optimal solutions rather than harmful mistakes, so ρ should be read as an upper bound on the harmful error rate.

Table 9: Measured error statistics on two graph families (deterministic execution, seed 0). $\bar{\Delta}$ is the average degree of the cut vertices, β the fraction of false positives among all errors, and "bound" is $\sum_d \rho_d |V_C^{(d)}| ((1-\beta_d) + \beta_d \bar{\Delta}_d)$, the per-level form of Theorem 3.

Instance	α	$\lvert V_{\mathrm{NN}}\rvert$	ρ (%)	β (%)	$\bar{\Delta}$	L	c_{inst}	bound
luxembourg.osm	500	2,092	35.28	47.7	2.40	206	0.2791	1232.0
citationCiteseer	500	61,893	8.93	53.4	13.42	261	0.0472	40875.6
citationCiteseer	2,000	59,416	8.88	53.2	13.71	252	0.0477	39846.7
coAuthorsCiteseer	1,000	32,520	11.00	53.6	15.35	312	0.0872	30998.0

The contrast between families is instructive. The error rate is much higher on the street network than on the citation graphs, but this is not because the predictor is worse there: on the low-degree street networks, 66–73% of the cut vertices decided by the predictor have degree ≤ 2, and an independent set on a path has two almost equivalent parity choices, so a fixed near-optimal reference disagrees with the predictor on many such vertices without affecting the objective. On citation/co-author graphs, cut vertices are high-degree boundary vertices ($\bar{\Delta} = 13$–15), whose decisions are strongly constrained by their neighbors inside V_C. This is why ρ is not comparable across instances, and why the loss normalization by $\rho|V_{\mathrm{NN}}|$ is the meaningful test of the theorem. The deterministic error profiles also confirm that the legality repair does not mask prediction quality: in all instrumented runs the repair changed 0 decisions, i.e., the decoded predictor output was already a legal MIS assignment.

The same relation holds across instances within a graph family: on the street networks, increasing α from 500 to 10^4 reduces $|V_{\mathrm{NN}}|$ by 80% (belgium.osm, 41,333 → 8,152), 85% (netherlands.osm, 59,536 → 8,904) and 83% (luxembourg.osm, 2,092 → 362), and the drops decrease by almost exactly the same relative amounts (-79%, -83%, -82%). In contrast, on citation and co-author

graphs the level-0 cut already contains a large fraction of $|V_{\mathrm{NN}}|$ ($\lambda_0 \approx 50.9\%$ for citationCiteseer and 66.8% for coAuthorsCiteseer), so increasing α can only eliminate the remaining deep levels; this is exactly the limitation stated in Corollary 3, and it explains why the same threshold tuning is very effective on street networks but not on citation networks. The practical implication is that the most reliable way to reduce the loss is to reduce $|V_{\mathrm{NN}}|$ by raising α, rather than to chase a predictor with a lower raw error rate, unless that error rate can be reduced specifically on the constrained cut vertices that dominate the objective.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction accurately reflect the contributions and scope of our paper. The scope and limitations of the empirical evaluation are discussed in the new "Limitations" section , and the real-world graph experiments are reported in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in the "Limitations" section , including the wall-clock cost relative to classical heuristics, the instance-dependent exact-solver threshold, benchmark saturation, error propagation, and memory considerations .

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide sketches of the proofs in the main paper and complete proofs in the appendix , including the new error-propagation analysis (Appendix E) with its assumptions stated explicitly and its bound validated empirically (Appendix G) .

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide our code and data in the supplemental material ; the implementation, dataset-generation scripts and archived result files are available at `https://github.com/citrontyan/RGF`. Section 5 documents the exact configuration, seeds, deterministic execution mode, and the reference-solution protocol needed to reproduce the main sparse-graph results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide our code and data in the supplemental material. We document the hyperparameters and training details in Appendix B; the implementation, data-generation scripts and result archives are available at `https://github.com/citrontyan/RGF` .

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide detailed information about the experimental setup in Section 5 and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For the sparse-graph experiments we report the mean and standard deviation over 5 random seeds (3 for the α study) with deterministic execution, 95% bootstrap confidence intervals for paired differences, and two-sided Wilcoxon signed-rank tests; the

synthetic results average 5 independent runs and are not deterministic. Appendix G validates the error bound per level .

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 reports the workstation used (AMD Threadripper 3970X CPU, two NVIDIA RTX 3090 GPUs, 128 GB RAM) and the per-run wall-clock time and GPU memory consumption for all experiments. The large sparse-graph runs occupy at most 24 GB of GPU memory, and the full experimental campaign (including hyperparameter calibration and the five-seed runs) required several GPU-days on this workstation .

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and ensure that our research complies with all relevant guidelines.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: We do not foresee any significant societal impacts of our work, either positive or negative.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release any data or models that have a high risk for misuse, so we have not implemented any safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper uses public benchmarks and solvers that are credited and cited: the DIMACS10 and SNAP graph benchmarks, the METIS and KaMIS solvers, and the pretrained `DIFUSCO` and `FastT2T` checkpoints released by their authors. We use them under their public research licenses; no restricted asset is redistributed in this paper .

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We release the code implementing `RGF`, the dataset-generation and evaluation scripts, and the archived result files at `https://github.com/citrontyan/RGF`; Section 5 documents the configuration and the deterministic mode needed to reproduce the reported results .

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: We do not conduct any crowdsourcing experiments or research with human subjects, so we have not included any instructions or details about compensation.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not conduct any research with human subjects, so we have not described any potential risks or obtained any IRB approvals.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We use LLMs only for writing, editing, and formatting purposes, and they do not impact the core methodology, scientific rigor, or originality of our research. Therefore, we have not included a description of LLM usage in our paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.