Unsupervised Learning of Local Updates for Maximum Independent Set in Dynamic Graphs

Devendra Parkar[1,2*] **Anya Chaturvedi**[1*] **Joshua J. Daymude**[1,2]
[1] School of Computing and Augmented Intelligence
[2] Biodesign Center for Biocomputing, Security and Society
Arizona State University
Tempe, AZ 85281, USA
{dparkar1, anya.chaturvedi, jdaymude}@asu.edu

Abstract

We present the first unsupervised learning model for finding Maximum Independent Sets (MaxIS) in dynamic graphs where edges change over time. Our method combines structural learning from graph neural networks (GNNs) with a learned distributed update mechanism that, given an edge addition or deletion event, modifies nodes' internal memories and infers their MaxIS membership in a single, parallel step. We parameterize our model by the update mechanism's radius and investigate the resulting performance–runtime tradeoffs for various dynamic graph topologies. We evaluate our model against a mixed integer programming solver and the state-of-the-art learning-based methods for MaxIS on static graphs (ICML 2020; NeurIPS 2020, 2023). Across synthetic and empirical dynamic graphs of 50–1,000 nodes, our model achieves competitive approximation ratios with excellent scalability; on large graphs, it significantly outperforms the state-of-the-art learning methods in solution quality, runtime, and memory usage. When generalizing to graphs of 10,000 nodes (100x larger than the ones used for training), our model produces MaxIS solutions 1.05–1.18x larger than any other learning method, even while maintaining competitive runtimes.

1 Introduction

Combinatorial optimization problems on graphs (e.g., GRAPH-COLORING, TRAVELING-SALESMAN, VEHICLE-ROUTING, etc.) arise naturally in many practical domains [3, 33]. However, many of these problems are classically intractable to solve exactly, and some even resist efficient approximation. The situation is exacerbated by the fact that many real-world applications are *dynamic*, requiring algorithms to not only find solutions but also maintain them as the underlying structure evolves. While one could iteratively apply algorithms designed for static graphs at each time step, this approach is often computationally expensive and inefficient. Critically, this approach ignores the often close relationship between successive graph structures. A more effective strategy might avoid repeatedly recomputing solutions from scratch by instead leveraging these small structural changes to incrementally update an existing solution. Such *update algorithms* are an active area of theoretical algorithms research [4, 6, 11, 15, 19, 40], but no learning algorithms have yet taken this approach.

In this paper, we consider the MAXIMUM-INDEPENDENT-SET (MaxIS) problem in a dynamic setting. For static graphs, MaxIS is classically known to be NP-hard [16] and hard to approximate: for general graphs, deterministic approximation within a constant factor is impossible [30] and within an $n^{1-\varepsilon}$ factor is NP-hard [20].

*These authors contributed equally to this work.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Differentiable Learning of Combinatorial Algorithms.

Nevertheless, MaxIS has a diverse range of real-world applications, from identifying functional nodes in brain networks [1] to constructing diversified investment portfolios [21]. Our goal is to solve MaxIS on such dynamically evolving graphs.

Recently, the learning community has developed several methods for solving combinatorial optimization problems on static graphs [7, 24]. Central to many of these methods are Graph Neural Networks (GNNs) due to their efficiency in capturing the structural information present in real-world networks such as the Internet, social networks, and molecular interactions [36, 39]. In the dynamic setting, recent models such as temporal GNNs are primarily concerned with tasks such as edge prediction, node classification, graph classification, etc. [25, 27, 31, 34, 37]. Work considering combinatorial optimization problems in a dynamic setting remains limited, with few examples such as Gunarathna et al. [17], which is an indirect heuristic learning approach. No existing models learn update mechanisms comparable to theoretical, rule-based update algorithms.

We propose a model that solves MaxIS in dynamic graphs by directly learning an update mechanism instead of a heuristic. We utilize the power of message-passing GNNs [5] combined with sequence learning to learn an update mechanism that maintains an approximate MaxIS solution as the underlying graph changes. Having learned an update mechanism, the model is able to update a MaxIS solution following a change in the dynamic graph in a single inference step, unlike more popular heuristic learning methods which require multiple inference steps and thus scale poorly with graph size. We underscore this as a necessary shift in perspective required to address some of the common criticisms of learning approaches to solving combinatorial optimization problems, namely, lack of scalability, computation inefficiency, and huge training sample requirements.

Contributions. We present the first unsupervised learning model for solving MAXIMUM-INDEPENDENT-SET (MaxIS) on dynamic graphs. The central idea of learning local, distributed update mechanisms for combinatorial optimization problems on dynamic graphs is novel, and we demonstrate its efficiency in our evaluations. Compared to a commercial mixed integer programming solver [18], our model maintains favorable approximation ratios across a variety of dynamic graph topologies and sizes. For larger datasets, our model conclusively outperforms the three prior learning methods for MaxIS on static graphs: the current state-of-the-art, DP-GNN [8]; a single-shot method, Erdős-GNN [22]; and a deep reinforcement learning method, LwDMIS [2]. Resource scaling for our method vs. DP-GNN is drastic; we obtain solutions up to 10^4x faster and use up to 27.5x less memory while producing up to 5.64x larger MaxIS solutions. Compared to Erdős-GNN and LwDMIS, our model delivers 1.07–1.22x larger MaxIS solutions than either method while also running 6.18–24.8x faster and saving up to 2.78–12.97x memory. These gains hold even in generalization experiments, where we evaluate our model on graphs 100x the size of those it was trained on.

2 Maximum Independent Sets in Dynamic Graphs

Consider an undirected, static graph $G = (V, E)$ with nodes V and edges E. An *independent set* $I \subseteq V$ of G is a set of nodes no two of which are adjacent, i.e., for all $u, v \in I$, we have $(u, v) \notin E$. Its *size* is its number of nodes, denoted $|I|$. A *maximum independent set* (MaxIS) of G is any independent set I of G satisfying $|I| = \max\{|I'| : I' \text{ is an independent set of } G\}$.

Dynamic graphs (also called dynamic networks, temporal graphs, or evolving graphs) are graphs whose nodes and/or edges change over time [9, 10, 23, 31]. For this work, we consider dynamic graphs where at most one edge changes per time and the underlying set of nodes is fixed. Formally, a dynamic graph $\mathcal{G}$ is a finite sequence of *graph snapshots* $(G_0, G_1, \ldots, G_T)$ where each snapshot $G_t = (V, E_t)$ is an undirected, static graph on nodes V and, for all time steps $0 < t \leq T$, we have $|E_{t-1} \oplus E_t| = 1$, i.e., there is exactly one edge addition or one edge deletion per time step. We denote the *edge event* at time t which transitions G_{t-1} to G_t as $\mathcal{E}_t$. The *neighborhood* of a node v at time t is $\mathcal{N}_t(v) = \{u \in V \mid (u, v) \in E_t\}$ and its *degree* is denoted $d_t(v) = |\mathcal{N}_t(v)|$. The diameter of graph snapshot G_t, denoted $\text{diam}(G_t)$, is the maximum hop-distance between any two nodes in G_t.

We define the DYNAMIC-MAXIMUM-INDEPENDENT-SET (Dynamic MaxIS) problem as follows: Given a dynamic graph $\mathcal{G} = (G_0, G_1, \ldots, G_T)$—or, equivalently, an initial graph snapshot G_0 and a sequence of edge events $(\mathcal{E}_1, \ldots, \mathcal{E}_T)$—obtain a MaxIS for each snapshot G_t. This problem is clearly NP-hard since it contains the NP-complete MaxIS problem as a special case ($T = 0$). Given a solution for the MaxIS problem, there exist a polynomial-time conversions to obtain solutions

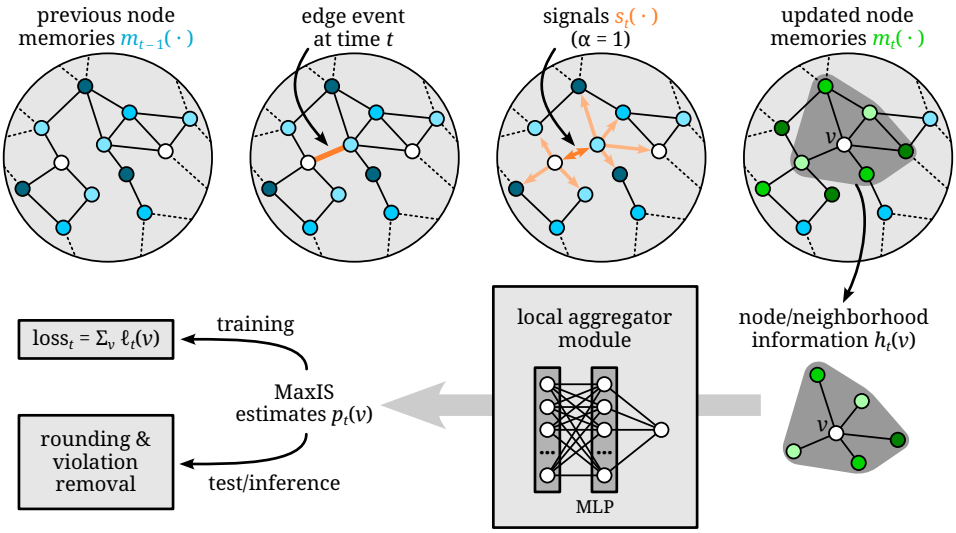

Figure 1: An overview of our unsupervised learning model for MaxIS on dynamic graphs. Nodes maintain memories (node colors) which are updated by signals propagating in an α-radius from edge events (orange arrows). Each node within a β-radius of the edge event then locally aggregates its updated memory and those of its neighbors to estimate its new probability of MaxIS membership. These estimates are then used to compute loss (during training) or to generate an integral MaxIS solution (during testing and inference).

for the MINIMUM-VERTEX-COVER and MAXIMUM-CLIQUE problems; hence, achieving good approximations for Dynamic MaxIS can also aid in approximating these problems' dynamic versions.

3 An Unsupervised Learning Model for Dynamic MaxIS

Memory-based models such as [25, 31, 34, 38] have been quite successful in prediction and classification tasks in dynamic graphs in recent years. Motivated by their effectiveness in predictive learning of dynamics, we present an unsupervised learning model focused on the MaxIS problem in dynamic graphs. We take inspiration from the Temporal Graph Networks (TGN) model proposed by Rossi et al. [31], which generalizes memory-based models by including a GNN-based graph embedding module. We repurpose the two key modules of TGNs, memory and graph embedding, to learn an update mechanism that maintains a candidate MaxIS solution as the underlying graph changes. Moreover, inspired by the use of messages (seen as random perturbations) to improve GNN performance for NP-hard problems [29], we couple node memory to an event handling module to inform nodes about edge events.

Our model responds to edge events (i.e., edge additions or deletions) over time (see Figure 1). At each time step, the *event handling module* is responsible for signaling nodes within an α hop-distance of the edge event and *memory module* updates their internal representations using this signal. Finally, the *local aggregation (graph embedding) module* produces a probability of membership in the MaxIS solution (i.e., an *estimate*) based on node's memory, its degree, and the memories of its neighbors. We describe each module and the training-related details of our model in the following subsections; pseudocode can be found in Appendix B.

Event Handling Module. This module provides each node v within an α hop-distance of the edge event $\mathcal{E}_t$ with a signal $s_t(v)$ encoding the type of event (addition or deletion) and the node's normalized distance from the event. Formally, the signal for node v at time t is

$$s_t(v) = [\texttt{enc}(\mathcal{E}_t) \,||\, r_t(v)], \tag{1}$$

where enc encodes edge deletions as $[0, 1]$ and edge additions as $[1, 0]$, and $r_t(v)$ is the hop-distance from v to $\mathcal{E}_t$ normalized by α and linearly interpolated into $[-1, 1]$.

Memory Module. A node's memory is a high-dimensional representation maintained throughout the model's execution that captures relevant structural changes affecting the node. At each time t, this module updates the memories of nodes within an α hop-distance of the edge event $\mathcal{E}_t$. Updates are performed using a GRU cell [12]. Formally, the memory of a node v at time t is

$$m_t(v) = \begin{cases} \texttt{GRU}\big([m_{t-1}(v) \,||\, s_t(v)]\big) & \text{if } v \text{ is within an } \alpha \text{ hop-distance of } \mathcal{E}_t; \\ m_{t-1}(v) & \text{otherwise.} \end{cases} \tag{2}$$

Local Aggregation Module. This module updates the estimates—i.e., probabilities of membership in the MaxIS solution—of all nodes within a β hop-distance of the edge event $\mathcal{E}_t$. First, each node v aggregates its immediate neighbors' memories:

$$\tilde{h}_t(v) = \texttt{ReLU}\left(\sum_{u \in \mathcal{N}_t(v)} \mathbf{W}_1 m_t(u)\right), \tag{3}$$

where $\mathbf{W}_1$ are learnable weights. Next, node v combines these aggregated memories with its own memory and degree information to obtain a local embedding

$$h_t(v) = \mathbf{W}_2 \left[\tilde{h}_t(v) \,||\, m_t(v) \,||\, d_t(v)\right], \tag{4}$$

where again $\mathbf{W}_2$ are learnable weights. Finally, node v passes its embedding through a step-down MLP and a sigmoid function σ to obtain its estimate (probability of MaxIS membership) in $[0, 1]$:

$$p_t(v) = \sigma\big(\texttt{MLP}(h_t(v))\big). \tag{5}$$

Any node v beyond the β hop-distance of $\mathcal{E}_t$ simply retains its previous estimate, i.e., $p_t(v) = p_{t-1}(v)$.

Training. We first define a loss function from a node's local perspective and then define a cumulative loss function from these local losses. Since the memory and local aggregation modules operate on individual nodes, a local loss function enables these modules to synergize and learn a distributed update mechanism. The loss for a node v at time t is

$$\ell_t(v) = -p_t(v) + \frac{c}{2d_t(v)} \sum_{u \in \mathcal{N}_t(v)} p_t(u)p_t(v). \tag{6}$$

The $-p_t(v)$ term rewards nodes with large estimates (i.e., probabilities of MaxIS membership closer to 1), aligning with the goal of finding a large independent set. This is counterbalanced by the $\sum_{u \in \mathcal{N}_t(v)} p_t(u)p_t(v)$ term which penalizes violations of independence, i.e., when neighbors u and v both have large estimates. Finally, the constant c is a hyperparameter balancing the two terms' loss contributions; in practice, $c = 3$ worked reasonably well across our evaluations (Section 4).

The cumulative loss at time t is the sum of all local losses $\ell_t(v)$ for nodes v within a β hop-distance of the edge event $\mathcal{E}_t$ (i.e., all nodes that updated their estimates at time t). This set of nodes at this one time step forms a single batch in our training process. The sequential processing of all edge events in a dynamic graph's training set forms a single epoch in our training process.

Model Variants. As explained above, the hop-distances α and β control which nodes utilize an edge event to update their memories and which nodes update their estimates and contribute to cumulative loss, respectively. In the *bounded cascading* (BCAS) model variant, we set $\alpha = \beta = \gamma \cdot \text{diam}(G_0)$, where $\gamma \ll 1$, strictly bounding nodes' memory and estimate updates to a local region around each edge event.

In the *non-cascading* (No-CAS) model variant, we set $\alpha = 0$ and $\beta = \text{diam}(G_0)$; i.e., only the two nodes directly involved in the edge event update their memories, but all nodes in the graph update their estimates and contribute to cumulative loss. In this variant, the local aggregation module is primarily responsible for the graph-wide learning of the update mechanism, with node memory aiding in the locality of the edge event.

Initial MaxIS Generation and Unsupervised Learning. Update algorithms for MaxIS crucially require an existing MaxIS solution for the initial snapshot G_0 which is later updated as the graph structure changes. We mitigate this (potentially very expensive) prerequisite by utilizing an initial

MaxIS generation phase that efficiently produces memories for all nodes in G_0 and model weights that training can warm-start from. The idea is to build up G_0 one edge at a time (as a sequence of edge addition events), updating only the memories of the nodes involved in each edge addition. Once G_0 is fully constructed, estimates and corresponding losses are computed for all nodes. This constitutes a single epoch of this generation phase; in a single run of the generation phase, we execute epochs until the cumulative loss has stabilized. After executing multiple runs with random seeds, we extract the node memories and model weights from the run with the least cumulative loss to warm-start model training. Thus, the model as a whole learns to first find a candidate MaxIS (generation phase) and then maintain it (training phase) in an entirely unsupervised manner.

Integral Solution Generation. During testing and deployment, we use a simple rounding and violation removal procedure to convert the model's relaxed node estimates into a candidate MaxIS. Specifically, all nodes with estimates 0.5 or larger are initially included in the candidate solution; then, nodes violating independence are removed in decreasing order of number of violations until an independent set is obtained. Such a procedure is a necessary part of single-shot learning methods that directly use relaxation to solve optimization problems with hard constraints [14, 22, 32, 35].

4 Experiments

Datasets. We evaluate model performance across both synthetic and empirical dynamic graphs. For synthetic graphs, we consider Erdős–Rényi (ER) and Power Law (PL) graphs with randomly generated initial structures. For empirical graphs, we used initial structures given by GERMANY [28], a geographical network with a mesh topology; TWITTER [26], a social network subgraph with a dense small-world topology; and BRAIN [13], a biological network with a sparse small-world topology. Across all graphs, dynamics are produced by sampling edge additions or deletions according to the initial snapshot's degree distribution for a desired number of time steps. The dynamic graphs we consider span a range of sizes: small ($\approx$100 nodes, 50,000 time steps), medium ($\approx$1,000 nodes, 100,000 time steps), and large ($\approx$10,000 nodes, 5,000 time steps).

Our Model and Comparisons. We ran our complete training pipeline using the two model variants: BCAS and No-CAS. For the BCAS variant, we set $\alpha = \beta = 0.25 \cdot \text{diam}(G_0)$, i.e., half of the radius of the first snapshot of the dynamic graph. For each variant, we performed three training runs with different random seeds and report results for the median run. Since ours is the only learning method for MaxIS in the dynamic setting, we compare our method against a commercial mixed integer programming solver and existing learning methods for MaxIS on static graphs, where we apply them to each snapshot of a dynamic graph independently.

- Gurobi [18] is the most commonly used and commercially available mixed integer programming solver. We use the default settings of branch-and-bound search for solving.
- DP-GNN [8] is the state-of-the-art learning model for MaxIS in static graphs, combining dynamic programming with GNNs as comparators to form a multi-step model.
- Erdős-GNN [22] is an earlier GNN-based, single-shot, probabilistic learning model for combinatorial optimization in static graphs.
- LwDMIS [2] is a deep reinforcement learning method that adaptively adjusts its stages and defers decisions about nodes' MaxIS membership to improve scalability.

Metrics. We evaluate each method using three metrics: approximation ratio (performance), per-graph solution generation time (runtime), and peak memory usage. Performance refers to the mean ratio of a method's candidate MaxIS size against that of Gurobi's across all snapshots in a dynamic graph. Runtime refers to the mean time taken by a method to process a single snapshot and generate a candidate MaxIS across all snapshots in a dynamic graph (reported as seconds per graph, s/g). Peak memory usage denotes the maximum memory used by a method during training.

We provide further details about the datasets and evaluation of methods in Appendix C.

Table 1: Test dataset performance (w.r.t. Gurobi) for small-sized (left half) and medium-sized (right half) datasets. We report the means and the std. deviations of the performance along with the runtime (in seconds per graph), averaged across the test dataset. We report the peak memory utilized while training by learning methods in GigaBytes (GB). We separate our model variants from other learning based methods and highlight the methods with the best performance (bold-black) and the best runtime (blue) for each dataset.

Dataset (→) Method (↓)	ER ($n = 100$)	PL ($n = 100$)	TWITTER ($n = 119$)	GERMANY ($n = 50$)	ER ($n = 1{,}000$)	PL ($n = 1{,}000$)	BRAIN ($n = 638$)
BCAS	0.82 ± 0.049 $(0.029s/g)$ (2.3 GB)	0.93 ± 0.029 $(0.033s/g)$ (3 GB)	0.94 ± 0.028 $(0.064s/g)$ (3 GB)	0.88 ± 0.059 $(0.046s/g)$ (2 GB)	0.99 ± 0.025 $(0.024s/g)$ (2.1 GB)	0.96 ± 0.010 $(0.082s/g)$ (2.2 GB)	$\mathbf{1.41 \pm 0.043}$ $(0.050s/g)$ (6.1 GB)
No-CAS	0.93 ± 0.030 $(0.075s/g)$ (3.5 GB)	0.79 ± 0.063 $(0.068s/g)$ (3.4 GB)	0.94 ± 0.030 $(0.085s/g)$ (2.6 GB)	0.94 ± 0.042 $(0.062s/g)$ (2 GB)	$\mathbf{1.08 \pm 0.025}$ $(0.288s/g)$ (3 GB)	$\mathbf{0.99 \pm 0.005}$ $(0.235s/g)$ (2.3 GB)	1.34 ± 0.047 $(0.523s/g)$ (4.7 GB)
DP-GNN	$\mathbf{0.98 \pm 0.058}$ $(0.079s/g)$ (3.7 GB)	$\mathbf{0.99 \pm 0.051}$ $(0.086s/g)$ (3.6 GB)	$\mathbf{0.97 \pm 0.036}$ $(0.080s/g)$ (6.4 GB)	$\mathbf{0.99 \pm 0.035}$ $(0.071s/g)$ (1.9 GB)	0.45 ± 0.021[2] $(29.51s/g)$ (44 GB)	0.69 ± 0.024[1] $(44.09s/g)$ (26.4 GB)	0.24 ± 0.054[1] $(11.36s/g)$ (129.7 GB)
Erdos-GNN	0.88 ± 0.034 $(0.235s/g)$ (3.6 GB)	0.95 ± 0.026 $(0.241s/g)$ (3.4 GB)	0.90 ± 0.036 $(0.248s/g)$ (4.4 GB)	0.94 ± 0.035 $(0.142s/g)$ (2.6 GB)	0.95 ± 0.021 $(1.78s/g)$ (16 GB)	0.92 ± 0.014 $(2.00s/g)$ (6.4 GB)	1.15 ± 0.055 $(1.24s/g)$ (61 GB)
LwDMIS[3]	0.82 ± 0.051 $(0.024s/g)$ (N/A)	0.91 ± 0.044 $(0.057s/g)$ (N/A)	0.77 ± 0.065 $(0.076s/g)$ (N/A)	0.85 ± 0.062 $(0.053s/g)$ (N/A)	0.95 ± 0.019 $(0.207s/g)$ (N/A)	0.91 ± 0.022 $(0.037s/g)$ (N/A)	1.03 ± 0.039 $(0.302s/g)$ (N/A)

4.1 Results

The results highlight the efficient scalability and overall performance of our model. Both the BCAS and No-CAS variants of our model consistently outperform other learning-based methods in terms of runtime and peak memory usage, while also achieving comparable or superior performance across all graph sizes. Surprisingly, DP-GNN—although the current state-of-the-art—performs well only on small datasets (Table 1, left half). On the medium datasets (Table 1, right half), its performance is 1.30–4.20x worse than any next best method across datasets while running more than 1,000x slower and consuming up to 27.44x more memory than other methods. In contrast, earlier methods such as LwDMIS and Erdős-GNN exhibit better tradeoffs over different graph sizes: LwDMIS is reasonably fast but performs similarly or worse than Erdős-GNN, especially on small graphs, while Erdős-GNN achieves its stronger performance at the cost of slower speeds and worse scalability. Our model variants strike an effective balance between these, delivering 1.07–1.22x better performance than either method while also reducing runtime (6.18–24.8x faster), and saving up to 2.78–12.97x memory. This demonstrates the efficiency and scalability of our model when compared to any other existing learning method.

Generalization. We highlight our model's generalization ability by training our model variants on small synthetic datasets and then evaluating them on corresponding datasets 100x their size (Table 2). DP-GNN, Erdős-GNN, and LwDMIS display the same strengths and weaknesses as for the smaller datasets: DP-GNN scales so poorly that its runtime is four orders of magnitude worse than the others, Erdős-GNN achieves decent performance but is slower than the remaining methods, and LwDMIS runs very quickly at the cost of inferior performance. Our No-CAS variant, although not as fast as LwDMIS, achieves far superior performance to any other learning method and even ties Gurobi's performance in a third of the runtime. On the PL dataset specifically, BCAS achieves the same great performance, but is 4.97x faster than No-CAS and 14.73x faster than Gurobi.

References

[1] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, Jan. 2011. doi:10.1126/science.1193210.

[2]Restricted evaluation, 10,000 snapshots of test dataset

[3]LwDMIS fails to train on some graph snapshots; thus, we do not report its peak training memory usage

[4]Restricted evaluation, 6 snapshots of test dataset

Table 2: Generalization results for large-sized datasets (10,000 nodes), reported similarly as Table 1. Note, we do not specify the peak memory usage as no bespoke training was executed. Additionally, we report performance only for those methods that could be evaluated on the entire datasets.

Method (↓) Dataset (→)	ER	PL
BCAS	0.63 ± 0.085 $(0.015s/g)$	0.99 ± 0.002 $(0.679s/g)$
No-CAS	$\mathbf{1.17 \pm 0.024}$ $(5.349s/g)$	$\mathbf{0.99 \pm 0.001}$ $(3.372s/g)$
DP-GNN[4]	N/A $(36239.508s/g)$	N/A $(39994.408s/g)$
Erdos-GNN	1.07 ± 0.023 $(19.2s/g)$	0.94 ± 0.009 $(18.41s/g)$
LwDMIS	0.99 ± 0.020 (0.541 s/g)	0.92 ± 0.003 $(0.296s/g)$

[2] S. Ahn, Y. Seo, and J. Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning*, page 134–144. PMLR, Nov. 2020. URL https://proceedings.mlr.press/v119/ahn20a.html.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, 2007. ISBN 9780691129938.

[4] S. Assadi, K. Onak, B. Schieber, and S. Solomon. Fully dynamic maximal independent set with sublinear in n update time. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116. URL https://doi.org/10.1137/1.9781611975482.116.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. (arXiv:1806.01261), Oct. 2018. doi:10.48550/arXiv.1806.01261. URL http://arxiv.org/abs/1806.01261. arXiv:1806.01261 [cs].

[6] S. Behnezhad, M. Derakhshan, M. Hajiaghayi, C. Stein, and M. Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, page 382–405, Baltimore, MD, USA, Nov. 2019. IEEE. ISBN 978-1-72814-952-3. doi:10.1109/FOCS.2019.00032. URL https://ieeexplore.ieee.org/document/8948654/.

[7] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, Apr. 2021. ISSN 0377-2217. doi:10.1016/j.ejor.2020.07.063.

[8] L. Brusca, L. C. Quaedvlieg, S. Skoulakis, G. G. Chrysos, and V. Cevher. Maximum independent set: self-training through dynamic programming. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

[9] B.-M. Bui-Xuan, A. Ferreira, and A. Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(02): 267–285, 2003. doi:10.1142/S0129054103001728.

[10] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emerg. Distrib. Syst.*, 27(5):387–408, Oct. 2012. ISSN 1744-5760. doi:10.1080/17445760.2012.668546.

[11] S. Chechik and T. Zhang. Fully dynamic maximal independent set in expected poly-log update time. (arXiv:1909.03445), Apr. 2021. doi:10.48550/arXiv.1909.03445. URL http://arxiv.org/abs/1909.03445. arXiv:1909.03445 [cs].

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. (arXiv:1412.3555), Dec. 2014. doi:10.48550/arXiv.1412.3555. URL http://arxiv.org/abs/1412.3555. arXiv:1412.3555 [cs].

[13] N. A. Crossley, A. Mechelli, P. E. Vértes, T. T. Winton-Brown, A. X. Patel, C. E. Ginestet, P. McGuire, and E. T. Bullmore. Cognitive relevance of the community structure of the human brain functional coactivation network. *Proceedings of the National Academy of Sciences*, 110 (28):11583–11588, Jun 2013. doi:https://doi.org/10.1073/pnas.1220826110. URL https://www.pnas.org/doi/10.1073/pnas.1220826110.

[14] P. L. Donti, D. Rolnick, and J. Z. Kolter. Dc3: A learning method for optimization with hard constraints. Oct. 2020. URL https://openreview.net/forum?id=V1ZHVxJ6dSS.

[15] X. Gao, J. Li, and D. Miao. Dynamic approximate maximum independent set on massive graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1835–1847, 2022. doi:10.1109/ICDE53745.2022.00183.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., United States, 1979.

[17] U. Gunarathna, R. Borovica-Gajic, S. Karunasekera, and E. Tanin. Dynamic graph combinatorial optimization with multi-attention deep reinforcement learning. In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '22, page 1–12, New York, NY, USA, Nov 2022. Association for Computing Machinery. ISBN 978-1-4503-9529-8. doi:10.1145/3557915.3560956. URL https://dl.acm.org/doi/10.1145/3557915.3560956.

[18] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

[19] K. Hanauer, M. Henzinger, and C. Schulz. Recent advances in fully dynamic graph algorithms – a quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, Dec. 2022. ISSN 1084-6654. doi:10.1145/3555806.

[20] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636, Burlington, VT, USA, 1996. IEEE. doi:10.1109/SFCS.1996.548522.

[21] R. Hidaka, Y. Hamakawa, J. Nakayama, and K. Tatsumura. Correlation-diversified portfolio construction by finding maximum independent set in large-scale market graph. *IEEE Access*, 11:142979–142991, Jan 2023. doi:https://doi.org/10.1109/access.2023.3341422. URL https://arxiv.org/abs/2308.04769.

[22] N. Karalias and A. Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems*, volume 33, page 6659–6672. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/49f85a9ed090b20c8bed85a5923c669f-Abstract.html.

[23] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart. Representation learning for dynamic graphs: a survey. *J. Mach. Learn. Res.*, 21(1):70:2648–70:2720, Jan. 2020. ISSN 1532-4435.

[24] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html.

[25] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1269–1278, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6. doi:10.1145/3292500.3330895. URL https://dl.acm.org/doi/10.1145/3292500.3330895.

[26] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[27] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. (arXiv:2302.01018), July 2023. doi:10.48550/arXiv.2302.01018. URL http://arxiv.org/abs/2302.01018. arXiv:2302.01018 [cs].

[28] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. Sndlib 1.0—survivable network design library. *Netw.*, 55(3):276–286, 2010. ISSN 0028-3045.

[29] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, page 21997–22009. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/b8b2926bd27d4307569ad119b6025f94-Abstract.html.

[30] J. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986. doi:10.1016/0196-6774(86)90032-5.

[31] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020. URL https://arxiv.org/abs/2006.10637.

[32] M. J. A. Schuetz, J. K. Brubaker, and H. G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, Apr. 2022. ISSN 2522-5839. doi:10.1038/s42256-022-00468-6.

[33] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition, 2014. ISBN 9781611973594.

[34] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha. Dyrep: Learning representations over dynamic graphs. Sept. 2018. URL https://openreview.net/forum?id=HyePrhR5KX.

[35] H. P. Wang, N. Wu, H. Yang, C. Hao, and P. Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. Oct. 2022. URL https://openreview.net/forum?id=HjNn9oD_v47.

[36] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi:10.1109/TNNLS.2020.2978386.

[37] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs. Sept. 2019. URL https://openreview.net/forum?id=rJeW1yHYwH.

[38] L. Yu, L. Sun, B. Du, and W. Lv. Towards better dynamic graph learning: New architecture and unified library. 36:67686–67700, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d611019afba70d547bd595e8a4158f55-Abstract-Conference.html.

[39] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, 2019. ISSN 2197-4314. doi:10.1186/s40649-019-0069-y.

[40] W. Zheng, C. Piao, H. Cheng, and J. X. Yu. Computing a near-maximum independent set in dynamic graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 76–87, 2019. doi:10.1109/ICDE.2019.00016.

Table 3: Pearson correlation for model performance and runtime vs. properties of graph snapshots. Correlation coefficients are calculated over all 75,000 graph snapshots in the testing splits of our medium datasets. Due to the large number of samples, we set our threshold for significance at $p < 10^{-10}$. All reported correlations are significant with $p < 10^{-40}$, except the one value marked with a (†), which has $p = 0.00095$.

	Performance		Runtime	
Graph Properties	BCAS	No-CAS	BCAS	No-CAS
# Edges	0.99	0.97	–0.09	0.98
Min. Degree	0.91	0.94	–0.15	0.92
Max. Degree	0.96	0.89	0.12	0.92
Ave. Degree	0.99	0.97	–0.06	0.98
Ave. Clustering Coefficient	0.99	0.95	–0.01†	0.97
Diameter	–0.68	–0.79	0.43	–0.74
Ave. Node Distance	–0.75	–0.85	0.39	–0.81

A Comparision of BCAS vs. No-CAS

Table 3 shows correlations between our BCAS and No-CAS models' performance and runtime vs. properties of the graph snapshots in the testing datasets. For performance, the same graph properties help and hurt both BCAS and No-CAS: both models perform better on graphs with many edges, high-degree nodes, and dense clustering but perform worse when diameter and node distance increase. This is intuitive, as both variants aggregate information from only their local neighbors and thus benefit from dense structure but suffer from distant changes that are not propagated. For runtime, there is a sharp contrast between the variants. BCAS runtimes are largely uncorrelated with graph properties—owing to its strictly local cascades and aggregation—exhibiting only a weak positive correlation with diameter and distance (stemming from the fact that $\alpha = \beta$ are set as a fixed fraction of diameter). No-CAS runtimes, on the other hand, are strongly determined by density (i.e., edge counts, degrees, and clustering), since every node in the graph must perform local aggregation at every time step and the complexity of local aggregation is entirely dependent on neighborhood size.

B Model Pseudocode

Algorithms 1–3 show pseudocode for our model's initial MaxIS generation procedure, update training process, and integral solution generation procedure, respectively. Throughout, graph snapshots G_t, edge events $\mathcal{E}_t$, signals $s_t(\cdot)$, node memories $m_t(\cdot)$, node estimates $p_t(\cdot)$, and node losses $\ell_t(\cdot)$ are defined as they were in the main text. Bold versions of these notations refer to the vector of all node properties of that type (e.g., $\mathbf{m}_t$ refers to all node memories). In a slight abuse of notation, we use $\mathcal{N}_t^x(\mathcal{E}_t)$ to denote all nodes within an x hop-distance from the edge event $\mathcal{E}_t$; i.e., the endpoints of the edge event are in $\mathcal{N}_t^0(\mathcal{E}_t)$, those endpoints and their neighbors are in $\mathcal{N}_t^1(\mathcal{E}_t)$, and so on.

Algorithm 1 Initial MaxIS Generation Epoch

Initialize all node memories as $\mathbf{m}_0 \leftarrow \mathbf{0}$.
for all edges $(u, v) \in E_0$ **do**
    Let $\mathcal{E}$ be the edge event of adding (u, v).
    Do event handling with signal $s \leftarrow [\texttt{enc}(\mathcal{E}) \,||\, 0]$ as in Eq. 1.
    Update $m_0(u) \leftarrow \texttt{GRU}([m_0(u) \,||\, s])$ and $m_0(v) \leftarrow \texttt{GRU}([m_0(v) \,||\, s])$ as in Eq. 2.
Compute node estimates $\mathbf{p}_0$ from the updated node memories $\mathbf{m}_0$ as in Eqs. 3–5.
Compute node losses ℓ_0 from $\mathbf{p}_0$ as in Eq. 6.
Compute cumulative loss $L_0 \leftarrow \sum_{v \in V} \ell_0(v)$.

C Experiment and Details

Each dynamic graph is divided into its training, evaluation, and testing splits, arranged chronologically with increasing time steps. We use a 70:15:15 training, evaluation, testing split for small datasets and

Algorithm 2 Update Training Epoch

Initialize node memories $\mathbf{m}_0$ using Algorithm 1.
for all edge events $\mathcal{E}_t \in (\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T)$ in the training set **do**
  for all nodes $v \in \mathcal{N}_t^{\alpha}(\mathcal{E}_t)$ **do**
    Do event handling with $s_t(v) \leftarrow [\texttt{enc}(\mathcal{E}_t) \mid\mid r_t(v)]$ as in Eq. 1.
    Update $m_t(v)$ using $s_t(v)$ as in Eq. 2.
  for all nodes $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ **do**
    Compute estimate $p_t(v)$ from $m_t(v)$ as in Eqs. 3–5.
  for all nodes $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ **do**
    Compute node loss $\ell_t(v)$ from $\mathbf{p}_t$ as in Eq. 6.
  Compute cumulative loss $L_t \leftarrow \sum_{v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)} \ell_t(v)$.

Algorithm 3 Integral Solution Generation (Testing & Inference)

Initialize node memories $\mathbf{m}_0$ using Algorithm 1.
for all edge events $\mathcal{E}_t \in (\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T)$ in the testing set **do**
  for all nodes $v \in \mathcal{N}_t^{\alpha}(\mathcal{E}_t)$ **do**
    Do event handling with $s_t(v) \leftarrow [\texttt{enc}(\mathcal{E}_t) \mid\mid r_t(v)]$ as in Eq. 1.
    Update $m_t(v)$ using $s_t(v)$ as in Eq. 2.
  for all nodes $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ **do**
    Compute estimate $p_t(v)$ from $m_t(v)$ as in Eqs. 3–5.
  for all nodes $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ **do** ▷ Estimate Rounding
    if $p_t(v) \geq 0.5$ **then** $I_t(v) \leftarrow 1$.
    else $I_t(v) \leftarrow 0$.
  while there exists a node $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ with violations **do** ▷ Violation Removal
    Choose the node $v \in \mathcal{N}_t^{\beta}(\mathcal{E}_t)$ with the most violations.
    Remove v from the MaxIS wtih $I_t(v) \leftarrow 0$.

a 50:25:25 split for medium datasets. The large datasets only have 5,000 time steps because we use them solely for generalization experiments and do not train our model on them.

All methods were trained (if applicable) and tested on a machine with an AMD EPYC 7413 CPU and a 20 GB slice of one NVIDIA A100 GPU. For the learning methods, training was halted after seven days. For Gurobi, we enforced a time limit of 1 s per snapshot; this is sufficient for producing exact solutions for small datasets, but yields approximations for medium and large ones.