# Curating Embodied Navigation Datasets for Lifelong Learning

Damon George & Jacob Krantz

June 9 2021

**Abstract**

Embodied navigation datasets commonly exist in an episodic format where individual episodes are treated as i.i.d. samples. However, this i.i.d. assumption is incorrect – episodes from the same scene are correlated. In the real world, this treatment would look like: 1) ask a robot to navigate in a scene, 2) reset its memory, and 3) ask the robot to navigate again in the same scene. This procedure is ineffective since past experience can inform future decision-making in the same scene. In this work, we develop a method to transform existing embodied navigation datasets to "tour"-based datasets; each tour in a transformed dataset contains a set of episodes such that for all pairs of episodes in the set, an oracle agent can inter-navigate between them. We show that this transformation can be solved via a reduction to the minimum vertex cover problem (MVC) or alternatively the maximum independent set problem (MIS). We can then reduce both of these formulations to SAT or PMAX-SAT. In analyzing the performance of these reduction methods on a real navigation dataset (VLN-CE [1]), we find that our best method (MVC-PMAX-SAT) solves the dataset transformation problem faster than an off-the-shelf MVC solver.

## 1 Introduction

In embodied navigation tasks, an agent is instantiated in either a simulated or physical environment and must take actions to follow a path or reach a goal location. Recently, there has been a heavy focus on evaluating navigation agents in novel environments and in an episodic fashion, *i.e.* once an agent calls `STOP`, memory is reset and the agent is teleported to the start of another episode [2–5]. However, this is neither realistic (we would like our robots to not rely on teleportation), nor is it effective from a life-long learning perspective (memory from past episodes can inform decision-making in future episodes). For these reasons, we aim to construct an embodied navigation dataset that is a subset of the original, such that if episodes are taken in random permutation, an agent can inter-navigate between them all. In other words, we wish to remove all episodes from the dataset that can't be navigated to/from other episodes within the same scene. We choose our problem to be the culling of existing datasets rather than the curation of new ones because we can leverage the wealth of existing embodied navigation datasets that took significant effort to create. Our method is general enough to transform these datasets in an automated and efficient manner.

The dataset transformation problem can be viewed as a graph problem in which each episode is a vertex, and an edge exists between two episodes if it is NOT possible for an agent to navigate between them. In this way, the minimum vertex cover problem finds the minimum number of episodes to remove while the maximum independent set problem finds the largest set of episodes to keep. Both of these formulations are classic NP-Hard problems that can be solved via reduction to SAT.

We apply our method to the Vision-and-Language Navigation in Continuous Environments dataset (VLN-CE [1]), which specifies paths through a simulated 3D environment using natural language instructions. An example of such an instruction could be "*Walk to the end of the hallway. Turn left into the kitchen and stop near the stove.*" Agents are further given RGB and depth observations and must take a sequence of low-level discrete actions (`FORWARD`, `LEFT`, `RIGHT`, `STOP`) to navigate to within 3.0 meters of the goal location.

# 2 Method

In this section, we formalize our dataset transformation problem and detail our solution that reduces the problem to SAT. We will consider the case where all episodes in the dataset exist in the same scene. For datasets with multiple scenes, a transformation can be applied to each scene's episode set. The resulting dataset would be the union of each scene's transformed episode set.

## 2.1 Problem Formulation

Each episode in the dataset, $t \in \mathcal{D}$, has a starting location $t^{(s)}$ and goal location $t^{(g)}$. The input to our problem can intuitively be formulated as a directed graph $G = (V, E)$ where $\forall t: \ t^{(s)} \in V$ and $t^{(g)} \in V$. There are two types of edges: edges from start to goal within a single episode ($\forall t: \ (t^{(s)}, t^{(g)}) \in E$) and edges from the end of one episode to the start of another ($\forall t_1, t_2 \ s.t. \ t_1 \neq t_2: \ (t_1^{(g)}, t_2^{(s)}) \in E$).

**Goal:** compute a minimal subset of vertices $S \subseteq V$, resulting in a graph $\hat{G}$ with vertices $V \setminus S$ such that all remaining edges $e \in \hat{E}$ can be navigated by an agent.

## 2.2 Framing Dataset Culling as a Minimum Vertex Cover

At a high level, we want to transform our directed graph $G$ into a graph that only contains edges along which an agent cannot navigate. From such a graph, the minimum vertex cover gives us the smallest subset of vertices such that if removed, no edges remain in the graph. The resulting graph after removing such vertices is our final transformed "tour"-based dataset.

**1. Graph Simplification.** We begin by observing that $G$ can be simplified given general knowledge of navigation datasets. We assume that all edges of the form $(t^{(s)}, t^{(g)}) \in E$ are navigable – the datasets we are interested in do not contain impossible episodes. As such, we can remove these edges from $E$, leaving us only with edges of the form $\forall t_1, t_2 \ s.t. \ t_1 \neq t_2: \ (t_1^{(g)}, t_2^{(s)}) \in E$.

We note that a "tour"-based episode set should be navigable taken in any permutation. For this reason, we can collapse start and goal vertices of the same episode into a single vertex, replacing $t^{(s)}, t^{(g)} \in G$ with $t$ for all episodes $t$. This gives us a fully connected, undirected graph where each episode is a vertex and each episode pair is an edge. We call this modified graph $G^{(t)} = (V^{(t)}, E^{(t)})$.

**2. Episode-to-Episode Navigability.** Given $G^{(t)}$, we run an A*-based path planning algorithm from the end of each episode to the start of each episode. If the algorithm is not able to compute a navigation plan under oracle conditions (given a known and static navigation mesh), we deem the two episodes not navigable. If the path plan is an action sequence of over 500 steps, we also deem the two episodes not navigable. These exceedingly long action sequences can be the product of inconsistencies in the navigation mesh or navigation that is only feasibly after collision with obstacles. At this point we know for each edge $e \in E^{(t)}$ whether or not it is navigable. We construct a new graph $G^m = (V^{(t)}, E^{(m)})$ such that all edges are not navigable: $E^{(m)} = \{e \mid e \in E^{(t)} \text{ and } e \text{ is not navigable}\}$.

**3a. The Minimum Vertex Cover.** Finally, $G^m$ is a graph whose minimum vertex cover is a minimal set of episodes such that if removed from the dataset, makes all episode sets in the dataset a valid tour.

**3b. The Maximum Independent Set.** Equivalently, $G^m$ is a graph whose maximum independent set is the largest set of episodes such that if culled from the dataset, makes all episode sets in the dataset a valid tour. This formulation is equivalent because the complement of a solution to the maximum independent set problem is a valid solution to the minimum vertex cover problem.

## 2.3 Encoding Minimum Vertex Cover as SAT

The k-vertex cover problem, *i.e.* $\{G \mid G = (V, E)$ is a graph with a vertex cover of size $\leq k\}$, is a classic NP-complete problem, and the reduction of this decision problem to SAT is straightforward. A vertex cover is a subset of vertices in a graph that touch at least one endpoint of every edge in the graph. To reduce the k-vertex cover problem to SAT, we must create a boolean formula $\phi_{vc}$ that is satisfiable iff $G$ contains a vertex cover of size $k$. To create this formula, we first define a boolean variable $x_i$ for every vertex $v_i \in V$. In $\phi_{vc}$, we simply add clauses of the form $(x_i \vee x_j)$ if an edge exists between those two vertices, *i.e.* if $(v_i, v_j) \in E$. Each clause enforces the constraint that at least one endpoint of each edge in the graph must exist in the

vertex cover. Lastly, we add a final condition that $x_1 + x_2 + \ldots + x_{|V|} \leq k$. The logical conjunction (and) of theses clauses is a boolean formula easily translateable to SAT/SMT and solveable by standard libraries.

$$\phi_{vc}(G,k) = \left(\sum_{v_i \in V} x_i \leq k\right) \wedge \left(\bigwedge_{(v_i,v_j)\in E} (x_i \vee x_j)\right) \tag{1}$$

If this formula is satisfiable, $G$ has a vertex cover of size $\leq k$, and the corresponding true assignments to the variables determines the vertices included in the vertex cover. To prove that this formula works as intended, let's first inspect the first clause, the at-most constraint. Clearly this clause enforces the maximum number of true variables to $k$, and since the true variables correspond to vertices in the vertex cover, this clause enforces the maximum size of the vertex cover. The second set of clauses ensures that for each edge, at least one of the endpoints is in the vertex cover. Since $\phi_{vc}$ is the logical conjunction of all these clauses, this formula is clearly satisfiable iff $G$ has a vertex cover of size $\leq k$.

However, the minimum vertex cover problem (MVC) is not a decision problem; it is an optimization problem to find the minimum $k$ such that a vertex cover of size $k$ exists in $G$. As such, this problem is NP-hard and is not so easily reducible to SAT. Using the above formulation of the $k$-vertex cover decision problem, we can write the MVC problem as

$$\min \sum_{v_i \in V} x_i \quad ; \; s.t. \bigwedge_{(v_i,v_j)\in E} (x_i \vee x_j) \tag{2}$$

As stated, this maximization problem cannot be encoded in SAT and solved with a standard SAT solver. The simplest solution to this issue is to search for $k$ by testing values of $k$ in $\phi_{vc}(G,k)$ with a SAT solver. In this work, we implement a simple binary search over the range of possible vertex cover sizes $k \in [1, |V|]$. We will call this method MVC-Search. With this method, $O(\log |V|)$ queries to the SAT solver are made during a search, making this solution clearly inefficient.

**Efficiency Improvement: Reducing to PMAX-SAT.**

Despite the limitation that the MVC problem cannot be encoded in SAT, the MVC problem can easily be encoded in PMAX-SAT. MAX-SAT is a generalization of SAT, with the problem being to find the maximum number of satisfiable clauses in a CNF boolean formula. It is obvious that MAX-SAT is NP-Hard because SAT could easily be decided by MAX-SAT (simply check if the output of MAX-SAT is the total number of clauses). PMAX-SAT extends MAX-SAT by differentiating between soft and hard clauses, with the goal being to maximize the number of satisfiable soft clauses while ensuring all hard clauses are satisfied.

To reduce the MVC problem to PMAX-SAT, we must first rewrite the minimization in (3) as a maximization.

$$\max \sum_{v_i \in V} \neg x_i \quad ; \; s.t. \bigwedge_{(v_i,v_j)\in E} (x_i \vee x_j) \tag{3}$$

To convert this maximization to a PMAX-SAT expression, we can simply define the summation as a set of soft clauses and the edge constraint as a set of hard clauses.

$$\phi_{mvc}(G) = \left(\bigwedge_{v_i \in V} (\neg x_i)^*\right) \wedge \left(\bigwedge_{(v_i,v_j)\in E} (x_i \vee x_j)\right) \tag{4}$$

where the $(\cdot)^*$ indicates a soft clause. All other clauses are hard and must be satisfied. With this formula $\phi_{mvc}$, a standard PMAX-SAT or Weighted MAX-SAT solver can be used to find the minimum vertex cover. The solver will maximize the number of vertices not included in the vertex cover (thereby minimizing the vertex cover size), while ensuring that endpoints from every edge are included in the vertex cover. We will call this method MVC-MAXSAT.

## 2.4 Encoding Maximum Independent Set as SAT

The m-independent set problem, ie $\{G \mid G = (V,E)$ is a graph with an independent set of size $\geq m\}$, is another classic NP-complete problem related to the vertex cover problem. An independent set is a subset

of vertices in a graph such that none of the vertices in the set are adjacent. The independent set is the complement of the vertex cover, which makes its SAT reduction a simple variation of the vertex cover reduction. The edge constraint clauses in the formula $\phi_{is}$ take the form of $(\neg x_i \vee \neg x_j)$ to ensure that both endpoints of an edge don't exist in the independent set. And similarly, instead of including an at-most constraint, this reduction uses an at-least constraint. The resulting SAT reduction of the $m$-independent set problem is therefore

$$\phi_{is}(G, m) = \left( \sum_{v_i \in V} x_i \geq m \right) \wedge \left( \bigwedge_{(v_i, v_j) \in E} (\neg x_i \vee \neg x_j) \right) \tag{5}$$

This formula is satisfiable iff $G$ has an independent set of size $\geq m$, and the true variable assignments determine the independent set membership.

Similarly to the MVC problem, the maximum independent set problem (MIS) is an optimization problem, not a decision problem, making this problem NP-hard. The corresponding maximization problem for MIS is

$$\max \sum_{v_i \in V} x_i \quad ; \; s.t. \bigwedge_{(v_i, v_j) \in E} (\neg x_i \vee \neg x_j) \tag{6}$$

Since this optimization problem cannot be encoded in SAT, we implemented binary search for this problem as well, which we call the MIS-Search method. Just like the MVC-Search method, this search requires $O(\log |V|)$ queries to the SAT solver using the boolean formula $\phi_{is}$.

**Efficiency Improvement: Reducing to PMAX-SAT.**

The MIS problem is also easily encodeable in PMAX-SAT. Since the MIS problem is already a maximization, no change is needed to the MIS formulation in (6) to create the PMAX-SAT formula. Once again, we define the at-least constraint as a set of soft clauses.

$$\phi_{mis}(G) = \left( \bigwedge_{v_i \in V} (x_i)^* \right) \wedge \left( \bigwedge_{(v_i, v_j) \in E} (\neg x_i \vee \neg x_j) \right) \tag{7}$$

where the $(\cdot)^*$ indicates a soft clause and all other clauses are hard, *i.e.* must be satisfied. With this formula $\phi_{pmax}$, a standard PMAX-SAT or Weighted MAX-SAT solver can be used to find the maximum independent set. We call this method MIS-MAXSAT.

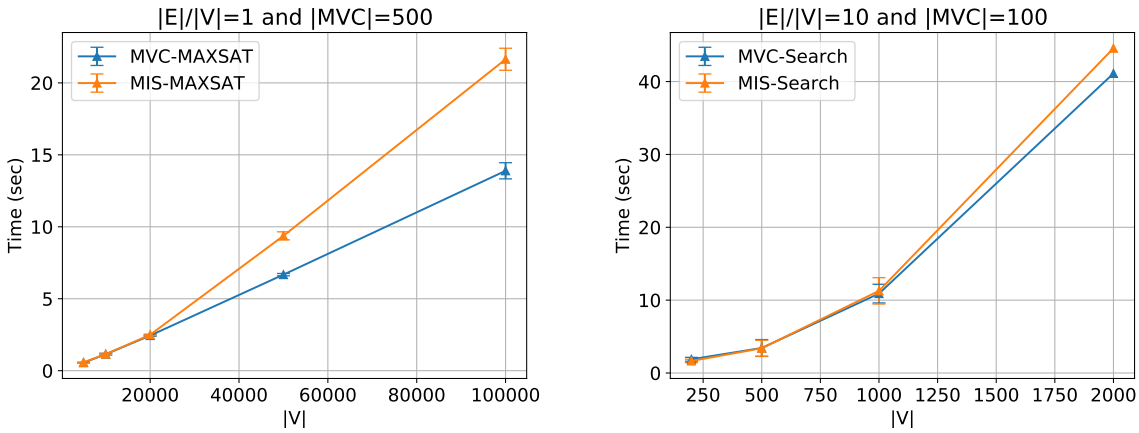
# 3 Experiments

We run experiments on both synthetic datasets and a real dataset to answer the following questions: (3.1) How does the performance of the MVC reduction compare to the performance of the MIS reduction? (3.2) How does the performance of the PMAX-SAT reduction compare to the standard SAT reduction? (3.3) How do the dimensions of episode count (graph vertices) and the number of non-navigable episode pairs (graph edges) contribute to the cost of our solution? (3.4) How efficient are these methods when applied to a real-world dataset?

Three random graph generation methods were used in the synthetic datasets. The simplest method generated random graphs according to the $G_{n,m}$ variant of the Erdős-Rényi model [6]. This model generates graphs randomly and uniformly from the set of all graphs with $n$ nodes and $m$ edges. The second method used the Barabási-Alpert model [7] to generate scale-free graphs. This method was preferable to the first because it generates graphs with a few high degree vertices, which more likely approximates real life datasets in which some episodes are more problematic than others. The last graph generation method we developed to create random graphs with an approximate MVC size $k$. To generate these graphs, we randomly selected $k$ vertices to be in the MVC, and then randomly generated the desired $m$ edges with the added constraint that at least one endpoint of every edge must be in the MVC. Since edges were allowed within the selected vertex cover, the desired MVC size $k$ was only an upper bound. This method allowed us to easily test the effect of the MVC size on the run time of the SAT reductions.

All timing tests were performed on a custom Ubuntu PC with an Intel i7-10700K cpu running at 3.8GHz.

### 3.1 MVC vs. MIS Reductions




(a) (b)

Figure 1: Comparisons of MVC vs. MIS reductions for both PMAX-SAT and SAT-search reductions. Each trial uses a constant ratio of edges to vertices and a constant MVC size, which are shown in the Figures.

In Figure 1, we compare the performance of reducing our dataset transformation problem to the MVC and the MIS problems. We make this comparison across a scaled number of vertices using the randomly-generated synthetic problem instances. While scaling the number of vertices, we choose a constant ratio of edges to vertices and a constant MVC size in order to prevent those additional variables from influencing the results. We present results for both the PMAX-SAT reduction (Figure 1a) and the SAT reduction with binary search (Figure 1b). For the SAT-search algorithms, we find that MVC and MIS reductions perform similarly without any significant difference in computational speed. However, for the PMAX-SAT methods, MVC and MIS reductions perform similarly only for low $|V|$. As $|V|$ is scaled higher, MVC begins to outperform MIS. This performance improvement becomes more pronounced as the problem difficulty increases, as we can see in Figure 1a. For instance, at $|V| = 1000$, MVC is only 1.05 times faster than MIS (averaged over 300 trials) whereas at $|V| = 10000$, MVC is 1.4 times faster than MIS (averaged over 50 trials).

Although the PMAX-SAT boolean formulas for MVC and MIS, $\phi_{mvc}$ and $\phi_{mis}$ contain the same number of clauses, MVC is likely faster because typically $|\text{MVC}| < |V|/2$. Because the MVC is the complement of the MIS, this means that the MVC size (the number of episodes in a minimal vertex cover) is typically less than the MIS size (the number of episodes in a maximal independent set). As a result, the number of satisfiable soft clauses in the MVC-MAXSAT formula is less than corresponding number of satisfiable soft clauses in the MIS-MAXSAT formula. This may give the MAX-SAT solver an easier time in finding the MVC solution, but this is speculation.

### 3.2 MaxSAT vs. SAT Reductions

In Figure 2, we compare the performance of the PMAX-SAT reduction to SAT with binary search. As in Section 3.1, we make this comparison across a scaled number of vertices using randomly-generated synthetic problem instances, a constant ratio of edges to vertices ($|E|/|V| = 10$) and a constant cover of 100. We present results for both the MVC and MIS reductions. We find that for both MVC and MIS, the PMAX-SAT reduction significantly outperforms the SAT-Search reduction for all tested problem instances – not just by a constant factor across scaled vertices, but exponentially so. The poor scaling of the SAT-search reduction makes it infeasible to run for $|V| > 2000$, whereas the PMAX-SAT reduction is able to handle graphs up to $|V| = 100000$.

The PMAX-SAT methods see such a signficant improvement in performance because they query the PMAX-SAT solver only once, whereas the SAT-Search methods make $O(\log |V|)$ queries to their SAT solver. Although a single query to the SAT solver is likely faster than a single query to the -SAT solver (because

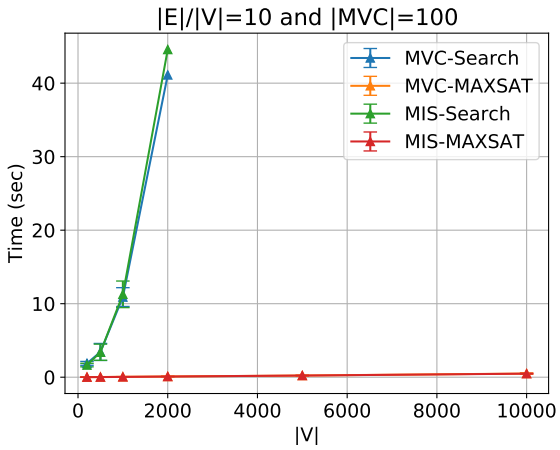
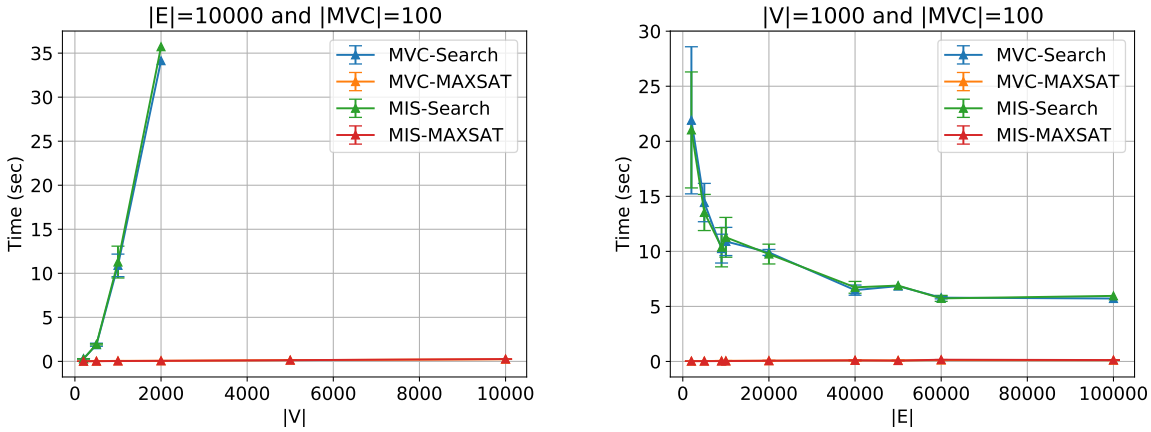



Figure 2: Comparisons of PMAX-SAT vs. SAT-search reductions for both MVC and MIS reductions. Each trial uses the constant ratio $|E|/|V| = 10$ and a constant MVC $= 100$.

PMAX-SAT is a more difficult complexity class), this is insignificant compared to the performance loss of making multiple queries to the SAT solver. Furthermore, the SAT formula $\phi_{vc}$ is likely more complex than the PMAX-SAT formula $\phi_{mvc}$ because of the at-most constraint. Ignoring the $|E|$ clauses required for both formulas, the PMAX-SAT expression requires exactly $|V|$ soft clauses whereas typical at-most SAT encodings require more than a linear number of clauses, as well as extra variables [8]. This increased formula complexity could actually make each SAT query slower than the single PMAX-SAT query, though this was not tested.

### 3.3 Solution Cost



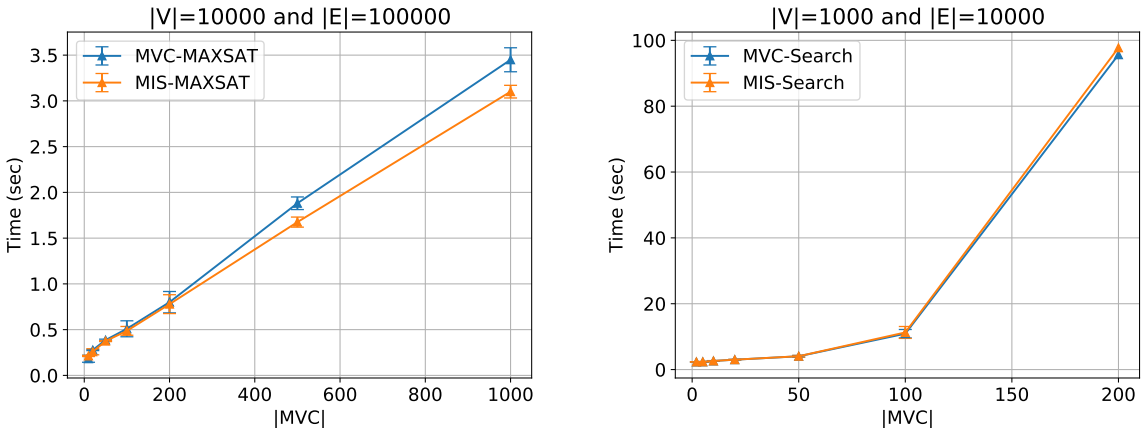

(a) (b)

Figure 3: Scaling of various methods due to vertices (left) and edges (right).

We are interested in determining how our solutions scale with increasing vertices and increasing edges studied independently. In Figure 3a, we fix the number of edges to 10000 and the cover to 100. We find that both MVC and MIS PMAX-SAT reductions scale near-linearly with the number of vertices. On the other hand, the SAT-search reductions for both MVC and MIS scale quite poorly in what appears to be another exponential curve.

In Figure 3b, we fix the number of vertices to 1000 and the cover to 100. We find once again that both

MVC and MIS PMAX-SAT reductions scale near-linearly with the number of edges. On the other hand, we observe the SAT-search reductions to actually speed up for higher $|E|$ and converge, following an exponential decay. Although this behavior is counter intuitive, it occurs because the size of the minimum vertex cover is fixed. As $|E|$ increases, the problem becomes simpler because the vertices in the vertex cover become more obvious. For example, when $|E| = 100000$, each vertex in the cover must have a high degree (on average 100). Thus finding the vertex cover is nearly as simple as finding the vertices with the highest degrees. Although this is an oversimplification of the computation that the SAT solver is performing, the solver likely exploits this information in some other way, resulting in the SAT-Search methods actually speeding up when $|E|$ increases for a fixed value of the MVC size.




(a) (b)

Figure 4: Scaling of PMAX-SAT methods (left) and the SAT-Search methods (right) due to the minimum vertex cover size |MVC|.

In addition to scaling with $|V|$ and $|E|$, we also note that our solutions scale with the solution size, *i.e.* the size of the minimum vertex cover. This explains why we fixed |MVC| to constant values in all the previous analyses. In Figure 4, we analyze the scaling performance of the SAT-Search and PMAX-SAT methods against increasing values of |MVC|, while maintaining fixed values of $|V|$ and $|E|$. In Figure 4a, we see that the PMAX-SAT reductions scale linearly with increasing MVC size, although this linear slope is larger than any of the slopes we've seen thus far, suggesting that the MVC size is one of the most important factors in determing the PMAX-SAT runtimes. The SAT-Search methods once again scale exponentially, as shown in Figure 4b. In fact, the SAT-methods scale so poorly with the MVC size that we were unable to test any values above $|\text{MVC}| = 200$, which makes these methods practically unusable for datasets of significant complexity.

The difficulty of these algorithms increases with |MVC| for likely the same reason that the running time of the SAT-Search methods decreased in Figure 3b. Small vertex cover sizes (relative to the graph size) correspond to more obvious solutions to the minimum vertex cover problem. And vice-versa, large MVC sizes result in a more uniform distribution of degrees among the vertices, increasing the difficulty of finding the specific vertices to minimize the cover size. Unfortunately, since the purpose of our methods is to find the minimum vertex cover, it is impossible to know beforehand whether a given dataset will have a large cover size, and therefore a high computational complexity.

## 3.4 Real-world Application

In Figure 5, we show how our tool performs on a real-world dataset (VLN-CE). We find that our tool can effectively transform this dataset into a "tour"-based one in a computationally feasible time. Specifically, the largest split of the dataset (Train), can be transformed in well under a second using our MAXSAT reduction from either MIS or MVC. As is consistent with results in Section 3.4, we find that the SAT reduction is significantly slower than the MAXSAT reduction. We find that an exact branch and bound algorithm from

| Reduction 1 | Reduction 2 | VLN-CE Transformation Time (s) | | |
|---|---|---|---|---|
| | | Train | Val-Seen | Val-Unseen |
| MIS | Cliquer | 0.316 | 0.000 | 0.001 |
| | SAT | 21.195 | 0.002 | 0.010 |
| | MAXSAT | 0.042 | 0.001 | 0.004 |
| MVC | Cliquer | 0.324 | 0.000 | 0.001 |
| | SAT | 21.035 | 0.001 | 0.007 |
| | MAXSAT | 0.038 | 0.001 | 0.004 |

Figure 5: Timings for transforming the VLN-CE dataset. We compare the MIS reduction to the MVC reduction. Further, we show a comparison of performance when reducing to SAT and MAXSAT and how these reductions compare to just solving MIS or MVC directly via an algorithm from Cliquer. Each reported time is an average across 5 trials. For reference, $|V_{\text{tr}}| = 10819$, $|E_{\text{tr}}| = 2778$ and $|\text{MVC}_{\text{tr}}| = 119$; $|V_{\text{v-s}}| = 778$, $|E_{\text{v-s}}| = 18$ and $|\text{MVC}_{\text{v-s}}| = 6$; and $|V_{\text{v-us}}| = 1839$, $|E_{\text{v-us}}| = 126$ and $|\text{MVC}_{\text{v-us}}| = 12$.

the Cliquer package, while it performs the fastest for small dataset splits (Val-Seen, Val-Unseen), does not scale as well as our MAXSAT reduction when solving a larger dataset split (Train).

### 3.5 Implementation Notes

We implement our dataset transformation tool in the Python language. The graphs generated in our program are stored and manipulated using the `networkx`[1] library. Prior to implementing our SAT and MaxSAT reductions, we used off-the-shelf MVC and MIS solvers in `sagemath`[2]. For the SAT and MaxSAT reductions, we used the `pysat`[3] library. We use the `pandas`[4] library for saving and comparing results between experiment runs. Along with our PDF report, we include a zip file containing code that can be used to replicate every experiment we ran in the paper. The code is general enough to apply to other real-world datasets beyond VLN-CE, which provides benefit to other researchers working in the embodied navigation space. Any standard graph format could be used to interface with our Python code, but currently the Python tool accepts as input any text file containing a newline-separated list of edges of the form: ⟨ `split-name, scene-id, [episode-id1, episode-id2]` ⟩, where each edge signifies a non-navigable transition between two episodes, as defined in our original problem formulation.

## 4 Conclusion

We have identified and addressed the problem of transforming an embodied navigation dataset from an i.i.d. episodic dataset into a corresponding "tour"-based dataset that links together episodes for lifelong learning and evaluation. We develop an array of solutions to this problem involving either a reduction to the minimum vertex cover problem (MVC) or the maximum independent set problem (MIS). From either of these reductions, we show that it is possible to reduce any problem instance into SAT, or, more efficiently, into PMAX-SAT. We find that our most efficient solution (MVC-PMAX-SAT) scales better than an off-the-shelf MVC solver and can solve an instance of a real-world dataset (VLN-CE) nearly instantaneously. We hope our method can inspire future work on lifelong learning for embodied navigation agents, which we view as a critical skill to operating agents in the vast and complicated real world.

## References

[1] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee, "Beyond the nav-graph: Vision-and-language navigation in continuous environments," in *ECCV*, 2020, pp. 104–120.

[1] `https://networkx.org/`
[2] `https://doc.sagemath.org/html/en/reference/graphs/sage/graphs/graph.html`
[3] `https://pysathq.github.io/`
[4] `https://pandas.pydata.org/`

[2] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[3] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: a platform for embodied ai research," 2019.

[4] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," 2018.

[5] D. Batra, A. Gokaslan, A. Kembhavi, O. Maksymets, R. Mottaghi, M. Savva, A. Toshev, and E. Wijmans, "Objectnav revisited: On evaluation of embodied agents navigating to objects," *arXiv preprint arXiv:2006.13171*, 2020.

[6] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[7] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999. [Online]. Available: https://science.sciencemag.org/content/286/5439/509

[8] P. M. Bittner, T. Thüm, and I. Schaefer, "Sat encodings of the at-most-k constraint," in *International Conference on Software Engineering and Formal Methods*. Springer, 2019, pp. 127–144.