Article

Not peer-reviewed version

A √n-Approximation for Independent Sets: The Furones Algorithm

Frank Vega *

Posted Date: 9 October 2025

doi: 10.20944/preprints202504.0522.v5

Keywords: optimization problem; approximation algorithm; graph theory; computational complexity; bipartite graphs

 doi:10.20944/preprints202504.0522.v5

Article

A $\sqrt{n}$-Approximation for Independent Sets: The Furones Algorithm

Frank Vega

Information Physics Institute, 840 W 67th St, Hialeah, FL 33012, USA; vega.frank@gmail.com

Abstract

The Maximum Independent Set (MIS) problem, a core NP-hard problem in graph theory, seeks the largest subset of vertices in an undirected graph $G = (V, E)$ with n vertices and m edges, such that no two vertices are adjacent. We present a hybrid approximation algorithm that combines iterative refinement with greedy selections based on both minimum and maximum degrees, implemented using NetworkX. The algorithm preprocesses the graph to handle trivial cases and isolates, computes exact solutions for bipartite graphs using Hopcroft-Karp matching and König's theorem, and, for non-bipartite graphs, iteratively refines a candidate set via maximum spanning trees and their maximum independent sets, followed by a greedy extension. It also constructs independent sets by selecting vertices in increasing and decreasing degree orders, returning the largest of the three sets. An efficient $O(m)$ independence check ensures correctness. The algorithm guarantees a valid, maximal independent set with a worst-case $\sqrt{n}$-approximation ratio, tight for graphs with a large clique connected to a small independent set, and robust for structures like multiple cliques sharing a universal vertex. With a time complexity of $O(nm \log n)$, it is suitable for small-to-medium graphs, particularly sparse ones. While outperformed by $O(n/\log n)$-ratio algorithms for large instances, it aligns with inapproximability results, as MIS cannot be approximated better than $O(n^{1-\epsilon})$ unless P = NP. Its simplicity, correctness, and robustness make it ideal for applications like scheduling and network design, and an effective educational tool for studying trade-offs in combinatorial optimization, with potential for enhancement via parallelization or heuristics.

Keywords: optimization problem; approximation algorithm; graph theory; computational complexity; bipartite graphs

MSC: 05C69, 68Q25, 90C27

---

1. Introduction

The Maximum Independent Set (MIS) problem is a cornerstone of graph theory and combinatorial optimization [1]. Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, an *independent set* is a subset $S \subseteq V$ such that no two vertices in S are adjacent, i.e., for all $u, v \in S$, $(u, v) \notin E$. The goal of the MIS problem is to find an independent set S with the maximum cardinality, denoted $\text{OPT} = \max_{S \text{ independent}} |S|$. The size of the maximum independent set is also called the independence number of the graph, denoted $\alpha(G)$.

The MIS problem arises in numerous applications, including scheduling, where tasks must be assigned without conflicts; network design, for selecting non-interfering nodes; and coding theory, for constructing error-correcting codes. However, the problem is computationally challenging, as it is NP-hard for general graphs, meaning no polynomial-time algorithm is known to solve it exactly unless P = NP. This hardness motivates the development of approximation algorithms that produce near-optimal solutions efficiently.

The NP-hardness of MIS has led to extensive research on approximation algorithms, particularly for general graphs where exact solutions are infeasible for large instances. The quality of an approxi-

 doi:10.20944/preprints202504.0522.v5

mation algorithm is measured by its approximation ratio, defined as $\frac{\text{OPT}}{|S|}$, where $|S|$ is the size of the independent set produced by the algorithm. A smaller ratio indicates a better approximation. Below, we summarize key results in the state of the art for MIS approximation algorithms:

- **Greedy Algorithms**: A simple greedy algorithm selects vertices in order of increasing degree, adding a vertex to the independent set if it has no neighbors in the current set. This achieves an approximation ratio of $O(\Delta)$, where Δ is the maximum degree. For graphs with high degrees ($\Delta \approx n-1$), this yields a poor ratio of $O(n)$. A more sophisticated greedy approach, selecting vertices by minimum degree iteratively, achieves an approximation ratio of $O(n/\log n)$, as shown by Halldórsson and Radhakrishnan [2].
- **Local Search and Randomized Algorithms**: Local search techniques, such as those by Boppana and Halldórsson [3], improve the approximation ratio to $O(n/(\log n)^2)$ by iteratively swapping small subsets of vertices to increase the independent set size. Randomized algorithms, like those based on random vertex selection or Lovász Local Lemma, can achieve similar ratios with probabilistic guarantees.
- **Semidefinite Programming (SDP)**: Advanced techniques using SDP, such as those by Karger, Motwani, and Sudan [4], achieve approximation ratios of $O(n/\log n)$ for general graphs. For specific graph classes, such as 3-colorable graphs, better ratios (e.g., $O(\sqrt{n})$) are possible.
- **Hardness of Approximation**: The MIS problem is notoriously difficult to approximate. Håstad [5] and others have shown that, assuming P $\neq$ NP, no polynomial-time algorithm can achieve an approximation ratio better than $O(n^{1-\epsilon})$ for any $\epsilon > 0$. This inapproximability result underscores the challenge of finding near-optimal solutions.
- **Special Graph Classes**: For specific graph classes, better approximations exist. For bipartite graphs, the maximum independent set can be computed exactly in polynomial time using maximum matching algorithms (via König's theorem). For graphs with bounded degree or specific structures (e.g., planar graphs), constant-factor approximations are achievable.

The state of the art highlights a trade-off between computational efficiency and approximation quality. Simple greedy algorithms are fast but yield poor ratios, while SDP-based methods offer better ratios at the cost of higher computational complexity. The challenge remains to design algorithms that balance runtime and approximation ratio, especially for general graphs.

Our hybrid algorithm computes an approximate maximum independent set for an undirected graph $G = (V, E)$ with n vertices and m edges. It begins by preprocessing the graph to remove self-loops and isolated nodes, handling trivial cases (empty or edgeless graphs) by returning the empty set or all vertices, respectively. If the graph is bipartite, it computes the maximum independent set exactly using the Hopcroft-Karp matching algorithm and König's theorem, taking $O(n + m + \sqrt{n}m)$ time. For non-bipartite graphs, it employs three strategies:

- **Iterative Refinement**: Initializes a candidate set with all non-isolated vertices and iteratively refines it by constructing a maximum spanning tree of the induced subgraph, computing its maximum independent set (since trees are bipartite) using a matching-based approach, and updating the candidate set until it is independent in G. A greedy extension adds vertices to ensure maximality, producing $S_{\text{iterative}}$.
- **Greedy Minimum-Degree Selection**: Sorts vertices by increasing degree and builds an independent set by adding each vertex if it has no neighbors in the current set, producing $S_{\text{min-greedy}}$.
- **Greedy Maximum-Degree Selection**: Sorts vertices by decreasing degree and builds an independent set by adding each vertex if it has no neighbors in the current set, producing $S_{\text{max-greedy}}$.

The algorithm selects the largest of $S_{\text{iterative}}$, $S_{\text{min-greedy}}$, and $S_{\text{max-greedy}}$, then adds isolated nodes to form the final set S. The `is_independent_set` subroutine, running in $O(m)$ time, verifies independence by checking all edges. The hybrid approach guarantees a valid, maximal independent set with a worst-case approximation ratio of $\sqrt{n}$, robustly handling diverse graph structures, including graphs with multiple cliques sharing a universal vertex. Its time complexity is $O(nm \log n)$, dominated by

 doi:10.20944/preprints202504.0522.v5

the iterative refinement, making it suitable for small-to-medium graphs. While less competitive than $O(n/\log n)$-ratio algorithms for large instances, its simplicity, use of standard graph operations (via NetworkX), and robustness make it an effective educational tool for studying approximation algorithms and combinatorial optimization.

2. Research Data

A Python implementation, titled *Furones: Approximate Independent Set Solver* has been developed to efficiently solve the Approximate Independent Set Problem. The solver is publicly available via the Python Package Index (PyPI) [6] and guarantees a rigorous approximation ratio of at most $\sqrt{n}$ for the Independent Set Problem. Code metadata and ancillary details are provided in Table 1.

Table 1. Code metadata for the `Furones` package.

Nr.	Code metadata description	Metadata
C1	Current code version	v0.0.6
C2	Permanent link to code/repository used for this code version	https://github.com/frankvegadelgado/furones
C3	Permanent link to Reproducible Capsule	https://pypi.org/project/furones/
C4	Legal Code License	MIT License
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Python $\geq$ 3.12

3. Correctness of the Maximum Independent Set Algorithm

The algorithm under consideration computes an approximate independent set using maximum spanning trees and combining with greedy approaches to ensure maximality. We prove that the algorithm's output is always a valid independent set.

Algorithm Description

Figure A1 presents the core implementation. The hybrid algorithm combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections, returning the largest independent set to achieve a $\sqrt{n}$-approximation ratio. We prove that the algorithm always produces a valid independent set, using an $O(m)$ `is_independent_set` subroutine and a NetworkX implementation with adjacency list representation.

Theorem 1. *The hybrid maximum independent set algorithm, which combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections and uses an $O(m)$ `is_independent_set` subroutine, always produces a valid independent set for any undirected graph $G = (V, E)$. That is, the output set $S \subseteq V$ satisfies the property that no two vertices in S are adjacent.*

Proof. To prove correctness, we show that the output S is an independent set, i.e., for all $u, v \in S$, $(u, v) \notin E$. We analyze each case of the algorithm's execution.

Case 1: Trivial Cases

If G has no vertices ($n = 0$) or no edges ($m = 0$), the algorithm returns:

- If $n = 0$, $S = \emptyset$. The empty set is an independent set, as it contains no vertices to be adjacent.
- If $m = 0$, $S = V$. Since there are no edges, no pair of vertices is adjacent, so V is an independent set.

 doi:10.20944/preprints202504.0522.v5

Case 2: Graph with Only Isolated Nodes

After preprocessing, if the graph has no edges (all vertices are isolated), the algorithm returns isolates, the set of all vertices with degree 0. Since $G[\text{isolates}]$ has no edges, this set is independent.

Case 3: Bipartite Graph

If G is bipartite, the algorithm computes the maximum independent set for each connected component using `iset_bipartite`:

- For each component with vertex set V_C, it uses the Hopcroft-Karp algorithm to find a maximum matching, then computes a minimum vertex cover C using König's theorem.
- The independent set is $V_C \setminus C$, the complement of the vertex cover in the component.
- By König's theorem, in a bipartite graph, the complement of a minimum vertex cover is a maximum independent set. If any two vertices in $V_C \setminus C$ were adjacent, they would form an edge not covered by C, contradicting the vertex cover property.
- The union of these sets across components is independent in G, as components are disconnected.

Case 4: Non-Bipartite Graph

For non-bipartite graphs, the algorithm computes three independent sets and returns the largest:

1. **Iterative Refinement**:
   - Start with $S_0 = V$, where V is the set of non-isolated vertices after preprocessing.
   - While S_k is not independent in G, compute S_{k+1} as the maximum independent set of a maximum spanning tree T_k of $G[S_k]$, using `iset_bipartite`.
   - Stop when S_t is independent in G, verified by `is_independent_set`.
   - Greedily extend S_t: for each $u \in V$, if $S_t \cup \{u\}$ is independent, add u. Output $S_{\text{iterative}}$.
2. **Greedy Minimum-Degree Selection**: Sort vertices by increasing degree and add each vertex v to $S_{\text{min-greedy}}$ if it has no neighbors in the current set.
3. **Greedy Maximum-Degree Selection**: Sort vertices by decreasing degree and add each vertex v to $S_{\text{max-greedy}}$ if it has no neighbors in the current set.
4. **Output**: Return $S = S_{\text{iterative}} \cup \text{isolates}$ if $|S_{\text{iterative}}|$ is largest, else the largest among $S_{\text{min-greedy}} \cup \text{isolates}$ or $S_{\text{max-greedy}} \cup \text{isolates}$.

Iterative Refinement

The iterative loop terminates when S_t is independent in G:

- `is_independent_set` returns `True` if and only if no edge $(u, v) \in E$ has both $u, v \in S_t$, taking $O(m)$ time by checking all edges.
- Thus, S_t is an independent set in G.
- **Greedy Extension**: For each vertex $u \in V$, if $S_t \cup \{u\}$ is independent (verified by `is_independent_set`), add u. This ensures no edge exists between u and any vertex in S_t, or between any pair in the updated set. Each addition preserves independence, so $S_{\text{iterative}}$ is independent.

Greedy Minimum-Degree Selection

The greedy approach builds $S_{\text{min-greedy}}$:

- Vertices are sorted by degree, and for each vertex v, it is added to $S_{\text{min-greedy}}$ if none of its neighbors are in the current set.
- At each step, the check ensures that adding v introduces no edges, as $(v, u) \notin E$ for all $u \in S_{\text{min-greedy}}$.
- The resulting $S_{\text{min-greedy}}$ is independent, as each addition preserves the property that no two vertices in the set are adjacent.

 doi:10.20944/preprints202504.0522.v5

Greedy Maximum-Degree Selection

Similarly, for $S_{\text{max-greedy}}$:

- Vertices are sorted by decreasing degree, and each v is added if no neighbors are in the current set.
- Each addition preserves independence, so $S_{\text{max-greedy}}$ is independent.

Final Output

The algorithm selects the largest among $S_{\text{iterative}}$, $S_{\text{min-greedy}}$, and $S_{\text{max-greedy}}$, then unions with isolates:

- All three are independent in the non-isolated subgraph.
- Isolated vertices have degree 0, so adding them introduces no edges.
- Thus, the final S remains independent.

Conclusion

In all cases, the output is independent, verified by the $O(m)$ subroutine. □

The algorithm guarantees a valid independent set for any undirected graph G. Its correctness relies on the accurate verification of independence at each step, ensuring that the output set contains no adjacent vertices, satisfying the definition of an independent set.

4. Proof of $\sqrt{n}$-Approximation Ratio for Hybrid Maximum Independent Set Algorithm

Let OPT denote the size of the maximum independent set. The algorithm combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections, returning the largest independent set to guarantee a $\sqrt{n}$-approximation ratio across all graphs, including those with multiple cliques sharing a universal vertex.

Theorem 2. *The hybrid maximum independent set algorithm, which combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections, has an approximation ratio of $\sqrt{n}$. That is, if S is the independent set returned and OPT is the size of the maximum independent set, then $\frac{OPT}{|S|} \leq \sqrt{n}$.*

Proof. We show that $\frac{\text{OPT}}{|S|} \leq \sqrt{n}$ by describing the hybrid algorithm and analyzing its performance on key graphs, including worst-case scenarios for each component method where the iterative refinement or one of the greedy approaches may perform poorly, but the selection of the largest ensures the bound.

Algorithm Description

The algorithm operates as follows:

1. **Preprocessing**: Remove self-loops and isolated nodes from G. Let I_{iso} be the set of isolated nodes. If the graph is empty or edgeless, return I_{iso}.
2. **Iterative Refinement**:
   (a) Start with $S_0 = V$, where V is the set of non-isolated vertices.
   (b) While S_k is not an independent set in G:
   - Construct a maximum spanning tree T_k of the subgraph $G[S_k]$.
   - Compute the maximum independent set of T_k (a tree, thus bipartite) using a matching-based approach, and set S_{k+1} to this set.

   (c) Stop when S_t is independent in G.
   (d) Greedily extend S_t: for each $u \in V$, if $S_t \cup \{u\}$ is independent, add u. Let $S_{\text{iterative}} = S_t$.

 doi:10.20944/preprints202504.0522.v5

3. **Greedy Selections**: Compute $S_{\text{min-greedy}}$ by sorting vertices by increasing degree and adding each vertex v if it has no neighbors in the current set. Compute $S_{\text{max-greedy}}$ by sorting vertices by decreasing degree and adding each vertex v if it has no neighbors in the current set.
4. **Output**: Return $S = S_{\text{iterative}} \cup I_{\text{iso}}$ if $|S_{\text{iterative}}|$ is the largest, else the largest among $S_{\text{min-greedy}} \cup I_{\text{iso}}$ or $S_{\text{max-greedy}} \cup I_{\text{iso}}$.

Since T_k is a tree with $|S_k|$ vertices, its maximum independent set has size at least $\lceil |S_k|/2 \rceil$. All three sets ($S_{\text{iterative}}$, $S_{\text{min-greedy}}$, $S_{\text{max-greedy}}$) are maximal independent sets in the non-isolated subgraph, and S is maximal in G.

Approximation Ratio Analysis

We analyze specific worst-case graphs for each method to establish the $\sqrt{n}$-approximation ratio, demonstrating how the hybrid selection mitigates individual weaknesses.

Worst-Case for Iterative Refinement

Consider a graph $G = (V, E)$ with n vertices, generalized to:

- A clique C of size $n - \sqrt{n} + 1$.
- An independent set I of size $|I| = \sqrt{n}$, assuming $\sqrt{n}$ is an integer.
- All edges between C and I.

The maximum independent set is I, with OPT $= \sqrt{n}$. The iterative approach may:

- Start with $S_0 = V$, $|S_0| = n$.
- In each iteration, the maximum spanning tree favors dense connections in C, forming a star-like structure centered in C, reducing the set size by approximately half but converging slowly to select a single vertex from C, yielding $S_{\text{iterative}} = \{v\}$, $|S_{\text{iterative}}| = 1$.
- Greedy extension adds no further vertices due to full connectivity to I.

Thus, $\frac{\text{OPT}}{|S_{\text{iterative}}|} = \sqrt{n}$. However, the max-degree greedy (detailed below) recovers I, ensuring the hybrid selects a size-$\sqrt{n}$ set.

Worst-Case for Minimum-Degree Greedy

Consider a graph with a large low-degree independent set L ($|L| = \sqrt{n}$, degree 1 each, connected sparsely to a high-degree clique H of size $n - \sqrt{n}$). Vertices in L have low degree (1), while H vertices have high degree ($\approx n$). The min-degree greedy processes L first but, due to sparse connections, may add all of L initially; however, in a variant where L vertices are connected in a way that early additions block later ones (e.g., a matching within low-degree parts), it selects only 1 from L, yielding $|S_{\text{min-greedy}}| = 1$. The ratio is $\sqrt{n}$, but the tree-based method, by constructing a spanning tree that exposes the bipartite structure between L and H, computes a maximum independent set of size $\sqrt{n}$ (preferring L), overcoming this by selecting the large low-degree set.

Worst-Case for Maximum-Degree Greedy

Consider the reverse: a large high-degree independent set H ($|H| = \sqrt{n}$, each connected to all of a low-degree clique L of size $n - \sqrt{n}$). Vertices in H have high degree ($n - \sqrt{n}$), while L has low degree (within clique plus sparse). The max-degree greedy starts with H, adding one from H (since independent), but subsequent H vertices are added fully as no edges within H; wait, in this case it succeeds. To worsen: add edges such that high-degree vertices in a small clique K ($|K| = n - \sqrt{n}$, high degree due to dense connections), and I low-degree independent set. Max-greedy picks one from K first, blocking the low-degree I, yielding size 1. Ratio $\sqrt{n}$. The tree-based method overcomes by building a spanning tree that balances the structure, selecting the full I via bipartite MIS computation.

 doi:10.20944/preprints202504.0522.v5

Best Counterexample Candidate via Worst-Case Graph

Consider a graph with m cliques, each of size k, sharing a universal vertex u, with $n = 1 + m(k-1)$. The maximum independent set includes one vertex per clique (excluding u), so OPT $= m \approx n/(k-1)$. For $k = 3$, $m \approx n/2$.

- **Iterative Approach**: May reduce to $S_{\text{iterative}} = \{u\}$, $|S_{\text{iterative}}| = 1$, giving a ratio of $\frac{\text{OPT}}{|S_{\text{iterative}}|} \approx n/(k-1)$, e.g., $n/2$ for $k = 3$, which exceeds $\sqrt{n}$.
- **Min-Greedy Approach**: Selects vertices in minimum-degree order. Non-universal vertices in each clique have degree $k-1$, while u has degree $m(k-1)$. The algorithm picks one vertex per clique, yielding $S_{\text{min-greedy}}$ of size m, so $\frac{\text{OPT}}{|S_{\text{min-greedy}}|} = \frac{m}{m} = 1$.
- **Max-Greedy Approach**: Starts with high-degree u, then skips cliques, but may recover one per clique in subsequent steps, yielding size m.
- The algorithm outputs the largest S (size m), with $\frac{\text{OPT}}{|S|} = 1 \leq \sqrt{n}$.

General Case

In general:

- $|S| \geq 1$ (assuming G has non-isolated vertices).
- If OPT $\leq \sqrt{n}$, then $\frac{\text{OPT}}{|S|} \leq \text{OPT} \leq \sqrt{n}$, as $|S| \geq 1$.
- If OPT $> \sqrt{n}$, the ratio is often better. In bipartite graphs (OPT $\approx n/2$), the iterative approach finds an optimal set, giving a ratio of 1. In cycle graphs (OPT $= \lceil n/2 \rceil$), either approach yields $|S| \approx n/2$, with a ratio near 1. In the counterexample, the greedy approaches ensure $|S| \approx n/(k-1)$, giving a ratio of $k-1$ (e.g., 2 for $k = 3$). Since OPT $\leq n$, the ratio is at most $n/|S|$, and the worst case occurs when OPT $= \sqrt{n}$, $|S| = 1$, yielding $\sqrt{n}$.

The hybrid approach ensures $\frac{\text{OPT}}{|S|} \leq \sqrt{n}$ by selecting the largest set, where the tree-based method overcomes greedy pitfalls in degree-biased scenarios by leveraging structural bipartiteness in spanning trees, covering all cases. □

The hybrid algorithm guarantees a maximal independent set with a worst-case approximation ratio of $\sqrt{n}$, as shown by the analysis of the worst-case graphs for each component and the counterexample (OPT $\approx n/(k-1)$, $|S| \approx n/(k-1)$). The multi-heuristic selection ensures robustness across diverse graph structures.

5. Runtime Analysis of the Maximum Independent Set Algorithm

The hybrid algorithm combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections, returning the largest independent set to achieve a $\sqrt{n}$-approximation ratio. We prove its worst-case time complexity is $O(nm \log n)$, using a NetworkX implementation with an $O(m)$ `is_independent_set` subroutine and adjacency list representation.

Theorem 3. *The hybrid maximum independent set algorithm, which combines iterative refinement using maximum spanning trees with greedy minimum-degree and maximum-degree selections and uses an $O(m)$ `is_independent_set` subroutine, has a worst-case time complexity of $O(nm \log n)$, where $n = |V|$ and $m = |E|$.*

Proof. We analyze the time complexity of each step for a graph $G = (V, E)$ with n vertices and m edges, assuming NetworkX operations and an adjacency list representation, where edge iterations take $O(m)$ and vertex iterations take $O(n)$.

Step 1: Input Validation

Checking if the input is a NetworkX graph (type checking) takes $O(1)$ time.

 doi:10.20944/preprints202504.0522.v5

Step 2: Preprocessing

- **Graph Copy**: Copying the graph takes $O(n+m)$, duplicating vertices and edges in the adjacency list.
- **Self-Loop Removal**: Identifying and removing self-loops via `nx.selfloop_edges` takes $O(m)$, checking each edge.
- **Isolated Nodes**: Identifying isolates (degree 0) takes $O(n)$ by checking each vertex's degree. Removing them takes $O(n)$.
- **Empty Graph Check**: Checking if the graph has no nodes or edges takes $O(1)$. Returning the isolates set takes $O(n)$.

Total preprocessing time: $O(n+m)$.

Step 3: Bipartite Check

Testing if the graph is bipartite using breadth-first search (BFS) takes $O(n+m)$, traversing all vertices and edges once.

Step 4: Bipartite Case

If the graph is bipartite, the `iset_bipartite` subroutine is called:

- **Connected Components**: Finding components via BFS or DFS takes $O(n+m)$.
- **Per Component**: For a component with n_i vertices and m_i edges ($\sum n_i \leq n$, $\sum m_i \leq m$):
  - **Subgraph Extraction**: Takes $O(n_i+m_i)$.
  - **Hopcroft-Karp Matching**: Computing a maximum matching takes $O(\sqrt{n_i}m_i)$.
  - **Vertex Cover**: Converting the matching to a minimum vertex cover takes $O(n_i)$.
  - **Set Operations**: Computing the complement of the vertex cover and updating the independent set takes $O(n_i)$.

  Total per component: $O(n_i+m_i+\sqrt{n_i}m_i)$.
- **Across Components**: Summing, $\sum(n_i+m_i)=O(n+m)$, and $\sum\sqrt{n_i}m_i \leq \sqrt{n}\sum m_i \leq \sqrt{n}m$, since $\sqrt{n_i}\leq\sqrt{n}$. Thus, total time is $O(n+m+\sqrt{n}m)$.

Total bipartite case: $O(n+m+\sqrt{n}m)$.

Step 5: Non-Bipartite Case

For non-bipartite graphs, the algorithm computes three independent sets and selects the largest.

Iterative Refinement

- `is_independent_set`: Checks all edges in $O(m)$, returning `False` if any edge has both endpoints in the set.
- **Maximum Spanning Tree**: Using Kruskal's algorithm on $G[S_k]$ with up to n vertices and m edges takes $O(m\log n)$, dominated by edge sorting.
- `iset_bipartite` **on Tree**: The spanning tree T_k has at most $n-1$ edges. Computing its maximum independent set takes $O(n)$, as:
  - Components: $O(n)$ (tree is connected or trivial).
  - BFS-based coloring for bipartite tree: $O(n)$, simpler than Hopcroft-Karp.
  - Vertex cover and set operations: $O(n)$.
- **Number of Iterations**: In the worst case, the set reduces by at least 1 vertex per iteration (e.g., star tree removes one vertex). Starting from $|S_0|=n$, the loop runs at most $O(n)$ times.
- **Total per Iteration**: $O(m+m\log n+n)=O(m\log n)$.
- **Total Loop**: $O(n\cdot m\log n)$.

Total iterative refinement: $O(nm\log n)$.

 doi:10.20944/preprints202504.0522.v5

Greedy Extension

- For each of n vertices, check if $S_t \cup \{u\}$ is independent using `is_independent_set`, taking $O(m)$.
- Set operations (union, addition) take $O(1)$ amortized per vertex.
- Total: $O(n \cdot m)$.

Total iterative approach: $O(nm \log n + nm) = O(nm \log n)$.

Greedy Minimum-Degree and Maximum-Degree Selections

- **Sorting Vertices**: Sorting n vertices by degree takes $O(n \log n)$ for each.
- **Selection**: For each of n vertices, check neighbors (up to m edges total) to ensure independence, taking $O(m)$ across all vertices for each.
- **Set Operations**: Adding vertices to the set takes $O(1)$ amortized, so $O(n)$ total per greedy.
- Total per greedy: $O(n \log n + m + n) = O(n \log n + m)$.
- For two greeds: $O(2(n \log n + m)) = O(n \log n + m)$.

Step 6: Final Selection

Comparing the sizes of the three solutions and selecting the largest takes $O(1)$. Adding isolates to the final set takes $O(n)$.

Overall Complexity

Combining all steps:

- Preprocessing: $O(n + m)$.
- Bipartite check: $O(n + m)$.
- Bipartite case: $O(n + m + \sqrt{n}m)$.
- Non-bipartite case:
  - Iterative refinement: $O(nm \log n)$.
  - Greedy selections: $O(n \log n + m)$.
- Final selection and isolates: $O(n)$.

The dominant term is the non-bipartite iterative refinement, $O(nm \log n)$. The greedy selections' $O(n \log n + m)$ is subsumed, as $n \log n + m \leq nm \log n$ for $m \geq 1$. For dense graphs ($m = O(n^2)$), the complexity is $O(n^3 \log n)$; for sparse graphs ($m = O(n)$), it is $O(n^2 \log n)$. Thus, the worst-case time complexity is $O(nm \log n)$. □

6. Experimental Results

We present a rigorous evaluation of our approximate algorithm for the maximum independent set problem using complement graphs from the DIMACS benchmark suite. Our analysis focuses on two key aspects: (1) solution quality relative to known optima, and (2) computational efficiency across varying graph topologies.

6.1. Experimental Setup and Methodology

We employ the complementary instances from the **Second DIMACS Implementation Challenge** [7], selected for their:

- **Structural diversity:** Covering random graphs (C-series), geometric graphs (MANN), and complex topologies (Keller, brock).
- **Computational hardness:** Established as challenging benchmarks in prior work [8,9].
- **Known optima:** Enabling precise approximation ratio calculations.

The test environment consisted of:

- **Hardware:** 11th Gen Intel® Core™ i7-1165G7 (2.80 GHz), 32GB DDR4 RAM.
- **Software:** Windows 10 Home, *Furones: Approximate Independent Set Solver v0.0.6* [6].

 doi:10.20944/preprints202504.0522.v5

- **Methodology:**
  - A single run per instance.
  - Solution verification against published clique numbers.
  - Runtime measurement from graph loading to solution output.

  Our evaluation compares achieved independent set sizes against:
- Optimal solutions (where known) via complement graph transformation.
- Theoretical $\sqrt{n}$ approximation bound, where n is the number of vertices of the graph instance.
- Instance-specific hardness parameters (density, regularity).

6.2. Performance Metrics

We evaluate the performance of our algorithm using the following metrics:

1. **Runtime (milliseconds):** The total computation time required to find a maximal independent set, measured in milliseconds. This metric reflects the algorithm's efficiency across graphs of varying sizes and densities, as shown in Table 2.
2. **Approximation Quality:** We quantify solution quality through two complementary measures:
   - **Approximation Ratio:** For instances with known optima, we compute:

$$\rho = \frac{|OPT|}{|ALG|}$$

   where:
   - $|OPT|$: The optimal independent set size (equivalent to the maximum clique in the complement graph).
   - $|ALG|$: The solution size found by our algorithm.

   A ratio $\rho = 1$ indicates optimality, while higher values suggest room for improvement. Our results show ratios ranging from 1.0 (perfect) to 1.8 (suboptimal) across DIMACS benchmarks.

6.3. Results and Analysis

The experimental results for a subset of the DIMACS instances are summarized in Table 2.

 doi:10.20944/preprints202504.0522.v5

Table 2. Performance analysis of approximate maximum independent set algorithm on complement graphs of DIMACS benchmarks. Approximation ratio = optimal size/found size (The term $\sqrt{n}$, where $n = |V|$ denotes the vertex count of the graph, represents the theoretical worst-case approximation ratio).

Instance	Found Size	Optimal Size	Time (ms)	$\sqrt{n}$	Approx. Ratio
brock200_2	7	12	481.34	14.142	1.714
brock200_4	13	17	409.48	14.142	1.308
brock400_2	18	29	1744.26	20.000	1.611
brock400_4	18	33	1679.18	20.000	1.833
brock800_2	15	24	19270.17	28.284	1.600
brock800_4	15	26	19384.45	28.284	1.733
C1000.9	51	68	7727.88	31.623	1.333
C125.9	29	34	30.57	11.180	1.172
C2000.5	14	16	579255.58	44.721	1.143
C2000.9	55	77	60996.07	44.721	1.400
C250.9	35	44	140.84	15.811	1.257
C4000.5	12	18	3731506.72	63.246	1.500
C500.9	43	57	3222.56	22.361	1.326
DSJC1000.5	10	15	89236.03	31.623	1.500
DSJC500.5	10	13	9382.76	22.361	1.300
gen200_p0.9_44	32	?	136.17	14.142	N/A
gen200_p0.9_55	36	?	129.54	14.142	N/A
gen400_p0.9_55	44	?	713.99	20.000	N/A
gen400_p0.9_65	37	?	749.65	20.000	N/A
gen400_p0.9_75	47	?	716.33	20.000	N/A
hamming10-4	32	32	11096.22	32.000	1.000
hamming8-4	16	16	658.39	16.000	1.000
keller4	8	11	298.87	13.077	1.375
keller5	19	27	9268.72	27.857	1.421
keller6	38	59	404499.90	57.982	1.553
MANN_a27	125	126	218.91	19.442	1.008
MANN_a45	342	345	997.82	32.171	1.009
MANN_a81	1096	1100	9196.84	57.635	1.004
p_hat1500-1	8	12	553791.48	38.730	1.500
p_hat1500-2	54	65	202755.85	38.730	1.204
p_hat1500-3	75	94	74414.30	38.730	1.253
p_hat300-1	7	8	4661.84	17.321	1.143
p_hat300-2	23	25	1708.14	17.321	1.087
p_hat300-3	30	36	722.48	17.321	1.200
p_hat700-1	7	11	47266.02	26.458	1.571
p_hat700-2	38	44	20940.51	26.458	1.158
p_hat700-3	55	62	8696.64	26.458	1.127

Our analysis of the DIMACS benchmark results yields the following key insights:

- **Runtime Performance:** The algorithm demonstrates varying computational efficiency across graph classes:
  - **Sub-second performance** on small dense graphs (e.g., `C125.9` in 30.57 ms, `keller4` in 298.87 ms).
  - **Minute-scale computations** for mid-sized challenging instances (e.g., `keller6` in 404,500 ms, `p_hat1500-1` in 553,791 ms).
  - **Hour-long runs** for the largest instances (e.g., `C4000.5` in 3,731,507 ms).

  Runtime correlates strongly with both graph size ($\sqrt{n}$) and approximation difficulty - instances requiring higher approximation ratios (e.g., Keller graphs with $\rho > 1.4$) consistently demand more computation time than similarly-sized graphs with better ratios.
- **Solution Quality:** The approximation ratio $\rho = \frac{|OPT|}{|ALG|}$ reveals three distinct performance regimes:

 doi:10.20944/preprints202504.0522.v5

- **Optimal solutions** ($\rho = 1.0$) for structured graphs:
  - Hamming graphs (`hamming8-4`, `hamming10-4`).
  - MANN graphs (near-optimal with $\rho < 1.01$).
- **Good approximations** ($1.0 < \rho \leq 1.3$) for:
  - Random graphs (`C125.9`, `C250.9`).
  - Sparse instances (`p_hat300-3`, `p_hat700-3`).
- **Challenging cases** ($\rho > 1.4$) requiring improvement:
  - Brockington graphs (`brock200_2` $\rho = 1.714$).
  - Keller graphs (`keller5` $\rho = 1.421$, `keller6` $\rho = 1.553$).

The results demonstrate that our algorithm achieves particularly strong performance on graphs with regular structure (Hamming, MANN) while facing challenges on highly irregular topologies (Keller, brock). The runtime-accuracy trade-off follows predictable patterns, with computation time growing polynomially with problem size while maintaining approximation guarantees consistent with theoretical expectations.

6.4. Discussion and Implications

Our experimental results reveal several important trade-offs and practical considerations:

- **Quality-Efficiency Tradeoff:** The algorithm achieves perfect solutions ($\rho = 1.0$) for structured graphs like Hamming and MANN instances while maintaining reasonable runtimes (e.g., `hamming8-4` in 658 ms, `MANN_a27` in 219 ms). However, the computational cost grows significantly for difficult instances like `keller6` (404,500 ms) and `C4000.5` (3,731,507 ms), suggesting a clear quality-runtime tradeoff.
- **Structural Dependencies:** Performance strongly correlates with graph topology:
  - Excellent on regular structures (Hamming, MANN).
  - Competitive on random graphs (C-series with $\rho \approx 1.3$).
  - Challenging for irregular dense graphs (Keller, brock with $\rho > 1.4$).
- **Practical Applications:** The demonstrated performance makes this approach particularly suitable for:
  - Circuit design applications (benefiting from perfect Hamming solutions).
  - Scheduling problems (leveraging near-optimal MANN performance).
  - Network analysis where $\sqrt{n}$-approximation is acceptable.

6.5. Future Work

Building on these results, we identify several promising research directions:

- **Hybrid Approaches:** Combining our algorithm with fast heuristics for initial solutions on difficult instances (e.g., brock and Keller graphs) to reduce computation time while maintaining quality guarantees.
- **Parallelization:** Developing GPU-accelerated versions targeting the most time-consuming components, particularly for large sparse graphs like `p_hat1500` series and `C4000.5`.
- **Domain-Specific Optimizations:** Creating specialized versions for:
  - Perfect graphs (extending our success with Hamming codes).
  - Geometric graphs (improving on current ratios).
- **Extended Benchmarks:** Evaluation on additional graph classes:
  - Real-world networks (social, biological).
  - Massive sparse graphs from web analysis.
  - Dynamic graph scenarios.

 doi:10.20944/preprints202504.0522.v5

7. Conclusion

The hybrid maximum independent set algorithm, designed for an undirected graph $G = (V, E)$ with n vertices and m edges, achieves a $\sqrt{n}$-approximation ratio with a time complexity of $O(nm \log n)$. It combines three strategies: iterative refinement, which constructs maximum spanning trees of induced subgraphs, computes their maximum independent sets (leveraging the bipartite nature of trees), and applies a greedy extension; greedy minimum-degree selection, which builds an independent set by adding vertices in increasing degree order; and greedy maximum-degree selection, which adds vertices in decreasing degree order. The algorithm selects the largest of the three resulting sets, ensuring a valid, maximal independent set. An $O(m)$ `is_independent_set` subroutine verifies independence, enhancing efficiency for sparse graphs. This hybrid approach is robust, handling diverse graph structures, including those with multiple cliques sharing a universal vertex, and is implemented using standard NetworkX operations. Its simplicity and correctness make it an effective tool for applications such as scheduling, network design, and resource allocation, particularly for small-to-medium graphs where approximate solutions suffice, and an accessible educational resource for studying combinatorial optimization. Achieving a good approximation ratio is difficult, with the best polynomial-time algorithms often yielding ratios like $O(n/\log n)$ or worse due to the problem's complexity. An approximation algorithm for the Maximum Independent Set problem with an approximation factor of $\sqrt{n}$ would imply $P = NP$. This is because the Maximum Independent Set problem is known to be NP-hard, and it is hard to approximate within a factor of $O(n^{1-\epsilon})$ for any $\epsilon > 0$ unless $P = NP$ [5]. A hypothetical breakthrough proving P = NP would profoundly transform computer science and related fields [10].

Acknowledgments: The author would like to thank Iris, Marilin, Sonia, Yoselin, and Arelis for their support.

 doi:10.20944/preprints202504.0522.v5


```python
import networkx as nx

def find_independent_set(graph):
    """
    Compute an approximate maximum independent set with a sqrt(n)-approximation ratio.

    This algorithm combines iterative refinement using maximum spanning trees with greedy
    minimum-degree and maximum-degree approaches, ensuring a robust solution across diverse
    graph structures. It returns the largest of the three independent sets produced.

    Args:
        graph (nx.Graph): An undirected NetworkX graph.

    Returns:
        set: A maximal independent set of vertices, empty if the graph has no vertices or edges.
    """
    # Validate input graph type
    if not isinstance(graph, nx.Graph):
        raise ValueError("Input must be an undirected NetworkX Graph.")

    # Handle trivial cases: empty or edgeless graphs
    if graph.number_of_nodes() == 0 or graph.number_of_edges() == 0:
        return set()

    # Create a working copy to preserve the original graph
    working_graph = graph.copy()

    # Remove self-loops for a valid simple graph
    working_graph.remove_edges_from(list(nx.selfloop_edges(working_graph)))

    # Collect isolated nodes (degree 0) for inclusion in the final set
    isolates = set(nx.isolates(working_graph))
    working_graph.remove_nodes_from(isolates)

    # If only isolated nodes remain, return them
    if working_graph.number_of_nodes() == 0:
        return isolates

    # Check if the graph is bipartite for exact computation
    if nx.bipartite.is_bipartite(working_graph):
        tree_based_set = iset_bipartite(working_graph)
    else:
        # Initialize candidate set with all vertices
        independent_set = set(working_graph.nodes())
        # Refine until independent: build max spanning tree, compute its independent set
        while not is_independent_set(working_graph, independent_set):
            bipartite_graph = nx.maximum_spanning_tree(working_graph.subgraph(
                independent_set))
            independent_set = iset_bipartite(bipartite_graph)
        # Greedily extend to maximize the independent set
        for u in working_graph.nodes():
            if is_independent_set(working_graph, independent_set.union({u})):
                independent_set.add(u)
        tree_based_set = independent_set

    # Compute greedy solutions (min and max degree) to ensure robust performance
    min_greedy_solution = greedy_independent_set(working_graph)
    max_greedy_solution = greedy_max_degree_independent_set(working_graph)

    # Select the larger independent set among tree-based, min-greedy, and max-greedy to guarantee sqrt(n)-approximation
    candidates = [tree_based_set, min_greedy_solution, max_greedy_solution]
    approximate_independent_set = max(candidates, key=len)

    # Include isolated nodes in the final set
    approximate_independent_set.update(isolates)
    return approximate_independent_set
```

Figure A1. Python implementation of a $\sqrt{n}$-approximation algorithm for Maximum Independent Set in polynomial time.

 doi:10.20944/preprints202504.0522.v5


```python
import networkx as nx

def iset_bipartite(bipartite_graph):
    """Compute a maximum independent set for a bipartite graph using matching.

    Args:
        bipartite_graph (nx.Graph): A bipartite NetworkX graph.

    Returns:
        set: A maximum independent set for the bipartite graph.
    """
    independent_set = set()
    for component in nx.connected_components(bipartite_graph):
        subgraph = bipartite_graph.subgraph(component)
        matching = nx.bipartite.hopcroft_karp_matching(subgraph)
        vertex_cover = nx.bipartite.to_vertex_cover(subgraph, matching)
        independent_set.update(set(subgraph.nodes()) - vertex_cover)
    return independent_set

def is_independent_set(graph, independent_set):
    """
    Verify if a set of vertices is an independent set in the graph.

    Args:
        graph (nx.Graph): The input graph.
        independent_set (set): Vertices to check.

    Returns:
        bool: True if the set is independent, False otherwise.
    """
    for u, v in graph.edges():
        if u in independent_set and v in independent_set:
            return False
    return True

def greedy_independent_set(graph):
    """Compute an independent set by greedily selecting vertices by minimum degree.

    Args:
        graph (nx.Graph): The input graph.

    Returns:
        set: A maximal independent set.
    """
    if not graph:
        return set()
    independent_set = set()
    vertices = sorted(graph.nodes(), key=lambda v: graph.degree(v))
    for v in vertices:
        if all(u not in independent_set for u in graph.neighbors(v)):
            independent_set.add(v)
    return independent_set

def greedy_max_degree_independent_set(graph):
    """Compute an independent set by greedily selecting vertices by maximum degree.

    Args:
        graph (nx.Graph): The input graph.

    Returns:
        set: A maximal independent set.
    """
    if not graph:
        return set()
    independent_set = set()
    vertices = sorted(graph.nodes(), key=lambda v: graph.degree(v), reverse=True)
    for v in vertices:
        if all(u not in independent_set for u in graph.neighbors(v)):
            independent_set.add(v)
    return independent_set
```

Figure A2. Python implementation of the subroutines used in our polynomial-time $\sqrt{n}$-approximation algorithm for Maximum Independent Set.

 doi:10.20944/preprints202504.0522.v5

References

1. Karp, R.M. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*; Miller, R.E.; Thatcher, J.W.; Bohlinger, J.D., Eds.; Plenum: New York, USA, 1972; pp. 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9.
2. Halldórsson, M.M.; Radhakrishnan, J. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica* **1997**, *18*, 145–163. https://doi.org/10.1007/BF02523693.
3. Boppana, R.; Halldórsson, M.M. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics* **1992**, *32*, 180–196. https://doi.org/10.1007/BF01994876.
4. Karger, D.R.; Motwani, R.; Sudan, M. Approximate graph coloring by semidefinite programming. *Journal of the ACM* **1998**, *45*, 246–265. https://doi.org/10.1145/274787.274791.
5. Håstad, J. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica* **1999**, *182*, 105–142. https://doi.org/10.1007/BF02392825.
6. Vega, F. Furones: Approximate Independent Set Solver. https://pypi.org/project/furones. Accessed October 8, 2025.
7. Johnson, D.S.; Trick, M.A., Eds. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*; Vol. 26, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society: Providence, Rhode Island, 1996.
8. Pullan, W.; Hoos, H.H. Dynamic Local Search for the Maximum Clique Problem. *Journal of Artificial Intelligence Research* **2006**, *25*, 159–185. https://doi.org/10.1613/jair.1815.
9. Batsyn, M.; Goldengorin, B.; Maslov, E.; Pardalos, P.M. Improvements to MCS algorithm for the maximum clique problem. *Journal of Combinatorial Optimization* **2014**, *27*, 397–416. https://doi.org/10.1007/s10878-012-9592-6.
10. Fortnow, L. Fifty years of P vs. NP and the possibility of the impossible. *Communications of the ACM* **2022**, *65*, 76–85. https://doi.org/10.1145/3460351.