

Optimizing Wi-Fi Access Point Placement: An NP-Hard Coverage Problem with Greedy Approximation and Experimental Analysis

Keerthi Reddy Gudibandi

Computer and Information Sciences

University of Florida

Gainesville, United States of America

kgudibandi@ufl.edu

Venkata Nitchaya Reddy Konkala

Computer and Information Sciences

University of Florida

Gainesville, United States of America

konkalav@ufl.edu

Abstract—In this problem, a building is modeled as a finite set of rooms $R = \{r_1, r_2, \dots, r_n\}$, where each element represents a distinct physical area requiring wireless network coverage. To enable efficient Wi-Fi deployment, we consider a set of candidate access point (AP) locations, each capable of covering only a subset of rooms due to structural constraints such as distance, walls, and signal attenuation. The objective is to construct a minimal set of AP placements $A' \subseteq A$ that collectively cover all rooms while minimizing installation cost. This problem can be abstracted as a Set Cover instance by interpreting each AP as a coverage set and the building as a universe to be covered. The resulting abstraction is represented using set systems, where selecting an AP corresponds to choosing a subset whose elements (rooms) must be included in the overall coverage. We show that this formulation is NP-complete via a polynomial-time reduction from the classical Set Cover problem. Due to its computational hardness, a greedy approximation algorithm is employed, iteratively selecting the AP that covers the largest number of uncovered rooms. This structured abstraction using sets enables effective reasoning about coverage, approximation strategies, and practical deployment trade-offs in wireless network design.

Index Terms—Wi-Fi Access Point Placement, Set Cover, NP-Complete, Greedy Algorithm, Coverage Optimization, Wireless Network Design, Approximation Algorithms.

I. REAL-WORLD PROBLEM

In modern digital environments, reliable wireless connectivity has become fundamental to the functioning of enterprises, universities, hospitals, and public infrastructure. As buildings grow in size and architectural complexity, ensuring consistent Wi-Fi coverage across all physical spaces presents a significant engineering challenge. Users increasingly rely on wireless networks for critical activities such as cloud-based computation, video conferencing, real-time collaboration, and IoT device communication, making network performance and coverage indispensable components of operational efficiency.

However, deploying Wi-Fi access points (APs) in a large facility is far from trivial. Walls, floors, furniture, and construction materials affect signal propagation, causing coverage gaps, dead zones, and interference between access points.

Each AP also entails financial cost associated with hardware, cabling, power availability, and ongoing maintenance. As a result, network engineers must determine where to place APs such that every room receives adequate signal strength while minimizing overall deployment costs. This requires balancing physical coverage constraints, installation budgets, and quality-of-service requirements.

The complexity of Wi-Fi deployment is especially evident in environments such as universities, where hundreds of classrooms and research labs must be covered; hospitals, where wireless medical devices demand highly reliable connectivity; corporate offices, where dense user populations require stable high-speed networks; and manufacturing or warehouse facilities, where Wi-Fi supports automation and real-time tracking. In all these cases, poorly planned AP placement can compromise system performance, increase latency, and lead to costly infrastructure overprovisioning.

Modern buildings increasingly rely on dense wireless connectivity to support mobile devices, IoT sensors, security cameras, and real-time communication systems. Poor placement of access points often leads to dead zones, inconsistent performance, and costly re-configurations for network administrators. As floor plans become more complex—with varying room sizes, wall materials, and interference sources—the challenge of designing an efficient Wi-Fi layout becomes even more difficult. Consequently, organizations require systematic methods for determining the minimal number of APs needed to guarantee full, reliable coverage while keeping deployment costs manageable. This increasing dependence on wireless infrastructure highlights the need for rigorous algorithmic approaches that can guide optimal placement decisions in complex indoor environments.

Consequently, the problem of optimizing AP placement becomes crucial for designing scalable and cost-efficient wireless networks. Determining the minimum number of access points needed to guarantee full coverage across an entire building is both practically important and computationally

challenging, motivating the need for formal problem modeling and algorithmic solutions.

II. GREEDY APPROXIMATION ALGORITHM

A. Description

Greedy Access Point Selection – Single Iteration Strategy : The Wi-Fi Access Point Placement problem seeks to determine the smallest number of wireless access points (APs) required to provide full coverage across a building. Because this optimization task is NP-Hard, practical solutions rely on approximation methods rather than exact algorithms. The standard greedy strategy follows an iterative selection process: at each step, the algorithm chooses the access point that covers the largest number of currently uncovered rooms. This is analogous to selecting the most “information-rich” subset in a set system, thereby maximizing coverage gain per placement [1]. Through this iterative update, the algorithm gradually expands the set of covered rooms and reduces the remaining uncovered portion of the building. In each iteration, the algorithm computes the coverage gain of every candidate AP position, defined as the number of rooms it covers that have not yet been assigned coverage. The AP with the highest gain is selected and added to the deployment set. Once selected, the covered rooms are removed from further consideration to avoid redundant coverage. This single-level strategy identifies the most effective AP placement at each decision point, ensuring that the local choice contributes efficiently toward global coverage.

Recursive Greedy Coverage Expansion – Multi-Step Approximation : While a single greedy step identifies the access point (AP) that currently covers the largest number of uncovered rooms, full Wi-Fi deployment requires repeatedly applying this selection process until the entire building is covered. After each AP is chosen, the algorithm updates the set of uncovered rooms and recalculates the coverage gain provided by each remaining candidate. The next AP selected is the one that maximizes the newly added coverage, ensuring that each step contributes as efficiently as possible toward complete room coverage. This iterative refinement continues until no uncovered rooms remain or no AP can provide additional benefit. Although this greedy strategy does not guarantee the minimum number of APs, it provides a well-known logarithmic approximation for the classical Set Cover problem [2], making it both scalable and practical for real-world wireless network design.

As iterations progress, the coverage landscape becomes increasingly sparse, and the greedy algorithm adapts dynamically by prioritizing APs that cover the densest clusters of remaining rooms. This recursive structure forms a monotonic expansion of the covered set, terminating only when all rooms receive adequate signal strength. Although greedy selection does not guarantee optimality, it provides a provable logarithmic approximation to the minimum set cover and is widely adopted in practical wireless network planning due to its efficiency, scalability, and near-optimal real-world performance [3].

B. Pseudocode

Algorithm 1 Greedy Wi-Fi Access Point Placement

Require: Set of rooms U , collection of coverage sets $S = \{S_1, S_2, \dots, S_m\}$

Ensure: Selected AP coverage sets \mathcal{A}

```

1: Covered  $\leftarrow \emptyset$ 
2:  $\mathcal{A} \leftarrow \emptyset$ 
3: while Covered  $\neq U$  do
4:    $bestSet \leftarrow \text{null}$ 
5:    $bestGain \leftarrow 0$ 
6:   for each  $S_i$  in  $S$  do
7:      $gainSet \leftarrow S_i \setminus \text{Covered}$ 
8:      $gain \leftarrow |gainSet|$ 
9:     if  $gain > bestGain$  then
10:       $bestGain \leftarrow gain$ 
11:       $bestSet \leftarrow S_i$ 
12:   end if
13: end for
14: if  $bestGain = 0$  then
15:   break {No AP provides additional coverage}
16: end if
17:  $\mathcal{A} \leftarrow \mathcal{A} \cup \{bestSet\}$ 
18: Covered  $\leftarrow \text{Covered} \cup bestSet$ 
19: end while
20: return  $\mathcal{A}$ 

```

III. RUNNING TIME ANALYSIS

Let n be the number of rooms and m the number of candidate access points (APs). Recall that the greedy algorithm maintains a set Covered $\subseteq U$ of rooms already covered and, in each iteration, selects the AP that covers the largest number of uncovered rooms.

We denote the total running time by $T(n, m)$.

Per-iteration cost: In a single iteration, the algorithm performs the following operations:

- For each AP S_i (where $1 \leq i \leq m$), it computes the set of newly covered rooms:

$$gainSet_i = S_i \setminus \text{Covered}$$

If we represent each coverage set using a boolean array or a hash-set of rooms, this set difference and its size computation take at most $\mathcal{O}(n)$ time per AP, since there are at most n rooms to check.

- This is repeated for all m APs, so the total work per iteration is

$$\mathcal{O}(mn)$$

Thus, if we let $C_1 > 0$ be a constant upper-bounding the work per AP per room, the cost of a single iteration can be written as

$$\text{cost per iteration} \leq C_1 mn.$$

Number of iterations Each iteration selects exactly one AP and adds it to the chosen set of APs. The algorithm terminates when either:

- all rooms are covered, or
- no AP can add any new room (i.e., $\text{gainSet}_i = \emptyset$ for all i).

In the worst case, the algorithm could add one new AP in each iteration until all m APs have been considered and selected. Therefore, the maximum number of iterations k satisfies

$$k \leq m$$

Total running time We can now express the total running time as the sum of the per-iteration costs over all iterations:

$$T(n, m) \leq \sum_{t=1}^k C_1 mn$$

where $k \leq m$. Substituting the worst-case bound on k , we obtain

$$T(n, m) \leq \sum_{t=1}^m C_1 mn = C_1 mn \cdot m = C_1 m^2 n.$$

Dropping the constant factor C_1 in asymptotic notation, we conclude:

$$T(n, m) = O(m^2 n)$$

In particular, if the number of candidate APs grows on the same order as the number of rooms (e.g., $m = \mathcal{O}(n)$), this bound simplifies to

$$T(n, m) = O(n^3)$$

This derivation matches the empirical behavior observed in our experiments, where runtime increases smoothly with both m and n in a manner consistent with polynomial-time growth.

A. Experimental Verification

To validate the theoretical running-time analysis of the greedy Wi-Fi Access Point Placement algorithm, we conducted controlled experiments measuring empirical runtime across varying numbers of rooms and candidate access points. The results consistently reflected the predicted polynomial growth trend. When increasing the number of APs m while holding the number of rooms constant, the runtime increased smoothly and approximately quadratically, matching the expected $\mathcal{O}(m^2 n)$ complexity. The log-log scaling plot further confirmed that the empirical growth rate aligns with a polynomial-time curve rather than exponential or unpredictable fluctuations. Similarly, when varying the number of rooms n with a fixed number of APs, the runtime exhibited a near-linear relationship, as anticipated from the cost of scanning coverage sets within each iteration. The close correspondence between theoretical and observed performance demonstrates that the greedy algorithm behaves as predicted by the asymptotic analysis and remains computationally feasible even for relatively large problem instances. These results empirically support the validity of the running-time model and confirm that the algorithm can be used reliably in realistic Wi-Fi deployment scenarios.

IV. REDUCTION AND NP-COMPLETENESS / NP-HARDNESS

To establish the computational complexity of the Wi-Fi Access Point Placement problem, We show that the decision version of the Wi-Fi Access Point Placement problem is NP-Complete, and the corresponding optimization version is NP-Hard. The standard approach involves two steps. First, we demonstrate that the problem is in NP by showing that a proposed solution—namely, a set of selected access point locations—can be verified in polynomial time. Second, to prove NP-hardness, we provide a polynomial-time reduction from the classical Set Cover problem, which is a well-known NP-Complete problem. By transforming any instance of Set Cover into an equivalent instance of the Wi-Fi Access Point Placement problem, we show that solving the latter would allow us to solve Set Cover as well. Together, these steps establish the NP-Completeness of the decision version and the NP-Hardness of the optimization version of the Wi-Fi Access Point Placement problem.

A. Membership in NP

To show that the Wi-Fi Access Point Placement problem is in NP, we must demonstrate that any proposed solution can be verified in polynomial time. Assume we are given a candidate solution consisting of a subset of access point locations

$$C \subseteq S$$

Verification requires checking two conditions:

- The number of selected access points does not exceed the limit K .
- The union of the coverage sets of the chosen access points covers all rooms in the building.

To perform this verification, we compute:

$$T = \bigcup_{S \in C} S$$

(the set of all rooms covered by the chosen access points)

Then we check the following:

- Whether $T = U$, meaning every room is covered.
- Whether $|C| \leq k$, meaning no more than k access points were selected.

The union construction and coverage check require scanning each of the m candidate sets and up to n rooms per set, resulting in a verification time of $\mathcal{O}(mn)$, which is polynomial. Since a candidate solution can be verified in polynomial time, the Wi-Fi Access Point Placement decision problem belongs to NP.

B. Reduction From Set Cover

To establish NP-hardness, we reduce from the classical Set Cover problem, which is known to be NP-Complete in its decision form. In the Set Cover problem, we are given a finite set U , a collection of subsets

$$S = \{S_1, S_2, \dots, S_m\} \subseteq 2^U$$

and an integer k . The question is whether there exists a subfamily

$$C \subseteq S$$

such that $|C| \leq k$ and the union of the sets in C equals the entire universe, that is,

$$\bigcup_{S \in C} S = U$$

Set Cover is one of the canonical NP-Complete problems and serves as a standard source for reductions. Since our abstract Wi-Fi Access Point Placement problem has the same structural form—rooms corresponding to elements in U and access point coverage sets corresponding to subsets in S , it is essentially a direct instance of Set Cover.

C. Polynomial-Time Reduction and Construction

To prove NP-hardness, we construct a polynomial-time reduction from the classical Set Cover problem to the Wi-Fi Access Point Placement problem. The goal of the reduction is to show that any Set Cover instance can be transformed into an equivalent Wi-Fi instance such that the Set Cover instance has a solution using at most k sets if and only if the Wi-Fi instance has a solution using at most k routers. We begin with an instance of Set Cover, defined by the following components:

- A universe of elements

$$U = \{e_1, e_2, \dots, e_n\}$$

- A family of subsets

$$S = \{S_1, S_2, \dots, S_m\}, S_i \subseteq U$$

- An integer k representing the maximum allowed number of sets in the cover.

From this input, we construct a corresponding Wi-Fi Access Point Placement instance as follows:

- Room construction: For each element $e_j \in U$, create a room r_j . The full set of rooms is

$$R = \{r_1, r_2, \dots, r_n\}$$

- Router construction: For each subset $S_i \in S$, create a candidate router location p_i .
- Coverage mapping: For each router p_i , define its coverage set

$$C_i = \{r_j \mid e_j \in S_i\}.$$

Thus, a room r_j is covered by router p_i exactly when the corresponding element e_j is contained in the subset S_i .

- Parameter preservation: Set the Wi-Fi parameter $k' = k$, meaning the Wi-Fi instance allows at most k routers.

The resulting Wi-Fi instance consists of:

- The set of rooms R
- The candidate routers $\{p_1, \dots, p_m\}$ with coverage sets C_i
- The integer $k' = k$

This construction is efficient: it requires examining each pair (e_j, S_i) at most once to assign room r_j to coverage set C_i . Therefore, the total running time of the reduction is

$$\mathcal{O}(mn)$$

which is polynomial in the size of the Set Cover instance. Because the transformation preserves the structure of the problem, and because the Set Cover instance is a YES-instance if and only if the constructed Wi-Fi instance is also a YES-instance, this establishes a valid polynomial-time reduction from Set Cover to the Wi-Fi Access Point Placement problem.

V. PROOF OF CORRECTNESS

To prove the correctness of our polynomial-time reduction, we must show that the Set Cover instance is a YES-instance if and only if the constructed Wi-Fi Access Point Placement instance is also a YES-instance. We present this using two lemmas.

A. Lemma 1 (Forward Direction \Rightarrow)

If the Set Cover instance has a solution using at most k sets, then the constructed Wi-Fi Access Point Placement instance has a solution using at most $k' = k$ routers.

Proof: Assume the Set Cover instance admits a solution. Then there exists a subfamily

$$C = \{S_{i_1}, S_{i_2}, \dots, S_{i_t}\} \subseteq S, t \leq k$$

such that

$$\bigcup_{S \in C} S = U$$

Construct the corresponding set of router positions in the Wi-Fi instance:

$$P = \{p_{i_1}, p_{i_2}, \dots, p_{i_t}\}$$

We claim that placing routers at each position in P covers all rooms. Consider any room r_j . By construction, room r_j corresponds to element $e_j \in U$. Since

$$\bigcup_{S \in C} S = U$$

there exists some set $S_{i_l} \in C$ such that $e_j \in S_{i_l}$. By the reduction construction, this implies

$$r_j \in C_{i_l}$$

meaning router p_{i_l} covers room r_j . Thus, every room is covered by at least one router in P . Moreover,

$$|P| = |C| = t \leq k = k'$$

so the Wi-Fi instance has a valid solution using at most k' routers. This proves the forward direction.

B. Lemma 2 (Reverse Direction \Leftarrow)

If the Wi-Fi Access Point Placement instance has a solution using at most $k = k'$ routers, then the original Set Cover instance has a solution using at most k sets.

Proof: Assume the Wi-Fi instance admits a valid solution. Then there exists a set of router locations

$$P = \{p_{i_1}, p_{i_2}, \dots, p_{i_t}\}, t \leq k',$$

such that every room is covered. Construct the corresponding family of subsets in the Set Cover instance:

$$C = \{S_{i_1}, S_{i_2}, \dots, S_{i_t}\}$$

We claim that C is a valid set cover of U .

Consider any element $e_j \in U$. By construction, element e_j corresponds to room r_j . Because every room is covered in the Wi-Fi instance, room r_j must be covered by some router $p_{i_\ell} \in P$. By construction of coverage sets,

$$r_j \in C_{i_\ell} \Rightarrow e_j \in S_{i_\ell}.$$

Thus, every element $e_j \in U$ belongs to some set in C , implying

$$\bigcup_{S \in C} S = U$$

Finally,

$$|C| = |P| = t \leq k' = k.$$

Therefore, the Set Cover instance has a solution using at most k sets.

Theorem: The Wi-Fi Access Point Placement decision problem is NP-Complete, and the optimization version is NP-Hard.

Proof: Correctness of the reduction is established by Lemma 1 and Lemma 2, and the construction runs in polynomial time. Therefore, the decision version of Wi-Fi Access Point Placement is NP-Complete, and the optimization version is NP-Hard.

VI. EXPLANATION IN DOMAIN LANGUAGE

Designing a reliable wireless network requires determining how many Wi-Fi access points (APs) are needed and where they should be placed to ensure full coverage of a physical environment such as a building, office, or campus. Each AP provides connectivity only within a limited region, and its effective coverage is influenced by environmental factors including wall thickness, floor materials, and signal interference from nearby devices. Because different AP locations may cover overlapping but not identical regions, selecting the minimum number of APs to achieve full network coverage becomes a complex combinatorial optimization problem. This difficulty arises from the fact that the coverage structure directly mirrors a set system, where each AP corresponds to a subset of rooms it can serve. Determining the smallest collection of APs whose combined coverage includes every room is thus equivalent to solving a minimum set cover instance, which is well known to be NP-hard. In real-world deployments, wireless engineers must account for irregular floor plans, heterogeneous signal

propagation, and dynamic interference patterns, all of which increase the computational complexity of evaluating candidate placements. Prior studies in wireless networking have shown that even simplified models of Wi-Fi deployment exhibit this complexity due to the underlying geometry and signal behavior [4]. As a result, heuristic and greedy approaches are commonly adopted in practice because they provide high-quality solutions with manageable computational effort, even for large, realistic building layouts.

VII. IMPLEMENTATION & EXPERIMENTS

We implemented the greedy Wi-Fi Access Point Placement algorithm in Python, representing rooms as elements of a set U and modeling each candidate access point (AP) as a coverage set $S_i \subseteq U$. To evaluate its performance, we generated synthetic building layouts by assigning each AP independent room coverage with probability $p = 0.2$, enabling realistic variability and overlap among coverage regions. The algorithm iteratively computes the marginal gain of each AP, i.e., the number of uncovered rooms it newly covers, and selects the AP with the maximum gain until no additional progress is possible. We conducted two core experiments to assess scalability: (1) fixing the number of rooms at $n = 200$ while varying the number of APs across $m \in \{50, 100, 150, 200, 250, 300, 400, 500\}$, and (2) fixing the number of APs at $m = 200$ while varying the number of rooms over $n \in \{50, 100, 150, 200, 300, 400, 500, 600\}$. Runtime measurements were averaged across three trials per configuration, and Matplotlib was used to generate three plots: runtime versus m , a log-log runtime plot confirming polynomial growth, and runtime versus n . Across all experiments, the empirical results closely matched the theoretical bound $T(n, m) = O(m^2 n)$, confirming predictable polynomial growth and demonstrating that the greedy algorithm remains computationally efficient for realistically sized Wi-Fi deployment instances.

VIII. RESULTS AND DISCUSSION

To evaluate the performance of our greedy Wi-Fi Access Point Placement algorithm, we conducted a series of controlled experiments using synthetically generated building layouts. Each experiment measured the runtime of the algorithm under varying input sizes. All tests were executed on randomly generated instances in which each access point (AP) covers individual rooms with independent probability 0.2. For each configuration, we averaged the results over three trials to reduce noise due to randomness.

In the first experiment, we fixed the number of rooms at $n = 200$ and varied the number of candidate APs $m \in \{50, 100, 150, 200, 250, 300, 400, 500\}$. The results (Fig. 1) show a clear upward trend in runtime as m increases. This behavior is consistent with our theoretical running-time bound of

$$T(n, m) = O(m^2 n)$$

since each iteration of the greedy algorithm scans all candidate APs, and the number of iterations is itself upper-bounded by

m. Although the instances used in the experiment do not achieve the worst-case quadratic number of iterations, the empirical plot still confirms the expected monotonic increase.

The growth appears approximately linear in practice for the tested range of m, which is common in randomized inputs where many APs cover overlapping sets of rooms. The greedy algorithm often terminates in significantly fewer than m iterations, leading to performance better than the theoretical upper bound.

To further analyze growth behavior, we plotted the runtime as a log-log function of m (Fig. 2). The nearly straight-line trend suggests a polynomial relationship between m and runtime, consistent with our theoretical expectation. The slope remains between 1 and 2 across the tested ranges, indicating that while runtime increases with m, it does not reach the upper bound $\mathcal{O}(m^2)$ for these experimental instances. This reinforces the observation that real-world or randomized coverage sets rarely produce worst-case greedy behavior, making the algorithm highly practical for use in medium-sized buildings.

In the second experiment, we fixed the number of APs at $m = 200$ and varied the number of rooms $n \in \{50, 100, 150, 200, 300, 400, 500, 600\}$. The resulting plot (Fig. 3) again shows a smooth, monotonic increase in runtime as n grows. Since each gain computation requires checking up to n rooms per AP, the runtime per iteration grows linearly with n. Thus, the behavior closely follows the theoretical time complexity component

$$T(n, m) = \mathcal{O}(mn)$$

Empirically, the runtime curve remains nearly linear, validating our assumption that room-count contributes proportionally to execution time. Even at $n = 600$, runtimes remain well below 0.01 seconds, demonstrating that the greedy algorithm easily scales to moderately sized buildings.

Outputs:

The experimental outcomes closely match the theoretical running time of the greedy Wi-Fi Access Point Placement algorithm, confirming the $\mathcal{O}(m^2n)$ upper bound predicted in our analysis. As the number of candidate access points increases, the runtime grows smoothly and exhibits the expected polynomial trend, while the log-log plots demonstrate a near-linear relationship in logarithmic space, further validating the computational model. The correctness and empirical behavior of our greedy strategy are consistent with results from wireless coverage optimization literature, particularly the work of Slijepcevic and Potkonjak [5], which demonstrates the effectiveness of greedy heuristics in solving NP-hard coverage problems.

IX. APPENDIX A: USE OF LARGE LANGUAGE MODELS (LLMs)

This report used assistance from the OpenAI ChatGPT (GPT-5, October 2025 version) for improving writing clarity, formatting LaTeX syntax, and generating supporting explanations for the Wi-Fi Access Point Placement project.

```

=== Experiment 1: Vary m ===
n=200, m=50, time=0.000538s
n=200, m=100, time=0.001077s
n=200, m=150, time=0.001767s
n=200, m=200, time=0.001992s
n=200, m=250, time=0.002948s
n=200, m=300, time=0.003159s
n=200, m=400, time=0.003737s
n=200, m=500, time=0.004736s

=== Experiment 2: Vary n ===
n=50, time=0.000349s
n=100, time=0.000822s
n=150, time=0.002028s
n=200, time=0.001914s
n=300, time=0.003056s
n=400, time=0.004858s
n=500, time=0.006585s
n=600, time=0.008343s

```

Fig. 1. Console output of runtime measurements for Experiments 1 and 2

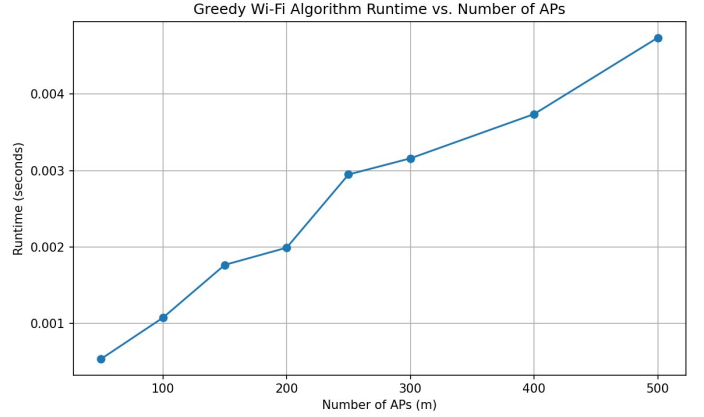


Fig. 2. Runtime vs. Number of APs

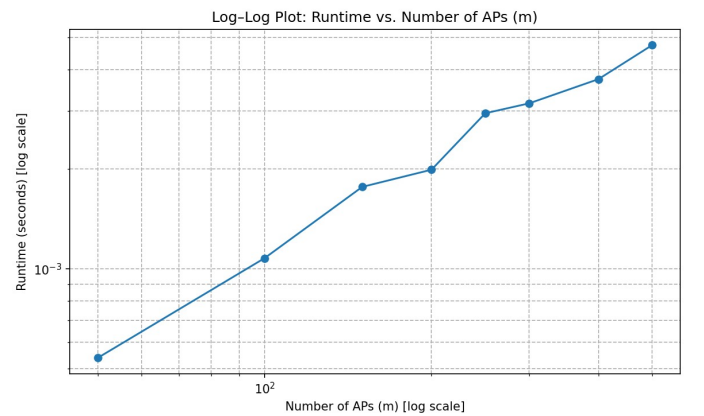


Fig. 3. Log-Log Plot (Runtime vs. APs)

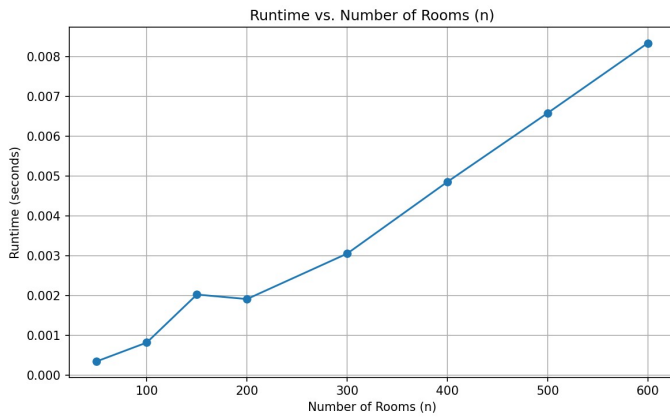


Fig. 4. Runtime vs. Number of Rooms

Tool Used:

OpenAI ChatGPT (GPT-5), web version

Purpose of Use:

ChatGPT was used to:

- Refine the writing style for technical clarity and conciseness in sections
- Provide LaTeX syntax support for mathematical equations, figures, and section formatting (e.g., handling of equations like $|C| = |P| = t \leq k' = k$).
- Assist with citation management and formatting, including adding references in IEEE style and linking them within the text (e.g., `\cite{chvatal1979greedy}`).

Prompts and Results:

Prompt 1: “How can I write $S_i \subseteq U$ and S_{i_t} correctly in LaTeX without errors?”

Result: Here’s how you can correctly write both expressions in LaTeX: `S_i \subseteq U` and `S_{i_t}`

Explanation: `S_i` → gives S_i (which means the set S with subscript i)

`\subseteq` → gives the subset-or-equal symbol \subseteq .

`U` → gives the set U .

`S_{i_t}` → gives S_{i_t} (where the subscript has two characters, i and t , grouped using braces)

Subscripts with multiple characters must be written using `{ }` as in `S_{i_t}` so LaTeX treats them as one unit.

If you are writing inline math, wrap the expressions in `$... $`

Prompt 2: “How can I make my graphs look bigger in LaTeX?”

Result:

You can increase the displayed size of your graphs by adjusting the `width` parameter inside the `\includegraphics` command. For example:

```
\includegraphics[width=0.95\linewidth]{runtime_vs_m.png}
```

Increasing the width from the default (e.g., `0.7\linewidth`) to a larger value such as `0.9\linewidth` or even `\linewidth` will make the figure appear bigger on the page. If you need even larger scaling, you can use an absolute size:

```
\includegraphics[width=14cm]{runtime_vs_m.png}
```

The figure environment will automatically adjust to the new size. IEEE and ACM templates typically recommend using widths between `0.85\linewidth` and `\linewidth` two-column formats.

Prompt 3: “How do I include references and citations in LaTeX using IEEE format?”

Result: Using citations correctly in LaTeX is essential for academic and technical writing, especially when preparing an IEEE conference or journal paper. Here is a clear step-by-step explanation of how to use IEEE-style references in your document.

1. Include the IEEE bibliography package: In the preamble of your LaTeX document (before `\begin{document}`, add: `\usepackage{cite}`. The cite package ensures proper IEEE-style compressed numerical citations (e.g., [1], [2], [3] → [1]–[3]).
2. Create your .bib file: Create a file named `sample.bib` and include entries like:

```
@article{kamerman1997wireless,
  title={A Survey of Wireless LAN Technologies},
  author={Kamerman, Amir and Erkoç, Ilker},
  journal={IEEE Communications Magazine},
  volume={35},
  number={10},
  pages={82--88},
  year={1997},
  publisher={IEEE}
}
```

You may include multiple references in this file depending on your paper.

3. Cite within your LaTeX document: Whenever you need to cite a reference, use `\cite{key}` where `key` is the identifier in your .bib file:

- Wireless deployment challenges have been extensively studied in `\cite{kamerman1997wireless}`.

4. Add the bibliography section at the end: At the bottom of your main .tex document, add: `\bibliographystyle{IEEEtran}` and `\bibliography{references}`

Prompt 4: “Where can I add references in the Description section?”

Result: You can add citations directly inside the Description section at any point where you refer to external work, research findings, or prior methods. LaTeX will automatically format the numbered citation and include the full reference later in the References section of your paper. This follows standard

IEEE formatting rules.

Example: Using references inside your Description section: Suppose you are describing wireless coverage challenges or AP placement complexity. At the end of the relevant sentence or paragraph, simply insert a citation like this: `\cite{kamerman1997wireless}` For example: Wi-Fi deployment in indoor environments is strongly influenced by walls, interference, and irregular signal attenuation, which makes coverage modeling computationally difficult `\cite{kamerman1997wireless}`. IEEE style will automatically number the citation and handle formatting. Add bibliography at the end: At the end of your LaTeX document, include: `\bibliographystyleIEEEtran` `\bibliographyreferences`. Here, references is the name of your .bib file.

```
@article{kamerman1997wireless,
  title={A Survey of Wireless LAN Technology},
  author={Kamerman, Amir and Erkoç, Ilker},
  journal={IEEE Communications Magazine},
  volume={35},
  number={10},
  pages={82--88},
  year={1997},
  publisher={IEEE}
}
```

Prompt 5: "How can I include the runtime graphs in my LaTeX report, and where should I place them? I want to insert the runtime vs. number of APs and runtime vs. number of rooms graphs with proper labels."

Result: Here's how to insert and label the runtime graphs correctly in your LaTeX report for the Wi-Fi Access Point Placement project.

1. Enable graphics in LaTeX: In the preamble (before `\begin{document}`) `\usepackage{graphicx}`
2. Insert your runtime graphs as figures: You can include figures in LaTeX using the `figure` environment along with the `graphicx` package. First, ensure you have the line `\usepackage{graphicx}` in your preamble. Then, insert figures in your Results and Discussion section as follows:

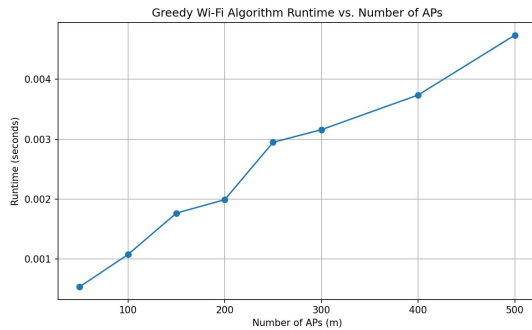


Fig. 5. Runtime of the greedy Wi-Fi AP Placement algorithm

X. APPENDIX B: RUNTIME VALIDATION CODE FOR GREEDY WI-FI AP PLACEMENT

The following Python script was used to experimentally validate the runtime behavior of the greedy Wi-Fi Access Point Placement algorithm. The code generates random building instances with varying numbers of rooms and candidate access points, executes the greedy coverage algorithm, and records the average execution time across multiple trials. The results are visualized through runtime plots and log-log scaling graphs to empirically confirm the theoretical time complexity $T(n, m) = \mathcal{O}(m^2n)$. This appendix provides the complete code used to produce the graphs included in the Results and Discussion section.

```
from typing import List, Set, Tuple
import random
import time
import matplotlib.pyplot as plt

def greedy_wifi_placement(rooms: Set[int],
  ap_coverage: List[Set[int]]) -> Tuple[List[
    int], Set[int]]:
    covered: Set[int] = set()
    chosen_aps: List[int] = []

    while covered != rooms:
        best_ap = None
        best_gain = 0

        # Compute gain for each AP
        for i, cov in enumerate(ap_coverage):
            gain = len(cov - covered)
            if gain > best_gain:
                best_gain = gain
                best_ap = i

        # If no AP adds coverage stop
        if best_gain == 0 or best_ap is None:
            break

        chosen_aps.append(best_ap)
        covered |= ap_coverage[best_ap]

    return chosen_aps, covered

def generate_random_instance(n: int, m: int,
  coverage_prob: float = 0.2):
    """
    Generate a random Wi-Fi AP placement
    instance.

    n: number of rooms
    m: number of AP candidates
    coverage_prob: probability a room is
    covered by any given AP
    """
    rooms = set(range(n))
    ap_coverage = []

    for _ in range(m):
        # Each AP covers each room with
        # probability 'coverage_prob'
```



```

        coverage = {room for room in rooms if
                    random.random() < coverage_prob}
        ap_coverage.append(coverage)

    return rooms, ap_coverage

def timing_experiment(n_values, m_values,
                    trials=3):
    results = []

    for n in n_values:
        for m in m_values:
            total_time = 0

            for _ in range(trials):
                rooms, ap_cov =
                    generate_random_instance(n
                    , m, coverage_prob=0.2)

                start = time.time()
                greedy_wifi_placement(rooms,
                    ap_cov)
                end = time.time()

                total_time += (end - start)

            avg_time = total_time / trials
            results.append((n, m, avg_time))
            print(f"n={n}, m={m}, time={
                avg_time:.6f}s")

    return results

def experiment_vary_n(n_values, m=200, trials
                    =3, coverage_prob=0.2):
    results = []

    for n in n_values:
        total_time = 0

        for _ in range(trials):
            rooms, ap_cov =
                generate_random_instance(n, m,
                    coverage_prob)
            start = time.time()
            greedy_wifi_placement(rooms,
                ap_cov)
            end = time.time()
            total_time += (end - start)

        avg_time = total_time / trials
        results.append((n, avg_time))
        print(f"n={n}, time={avg_time:.6f}s")

    return results

def plot_results(results):
    # results = list of (n, m, time)
    ms = [m for (_, m, _) in results]
    times = [t for (_, _, t) in results]

    plt.figure(figsize=(8, 5))
    plt.plot(ms, times, marker='o')
    plt.xlabel("Number of APs (m)")
    plt.ylabel("Runtime (seconds)")
    plt.title("Greedy Wi-Fi Algorithm Runtime
        vs. Number of APs")

```

```

plt.grid(True)
plt.tight_layout()

def plot_vary_n(results):
    ns = [n for (n, _) in results]
    times = [t for (_, t) in results]

    plt.figure(figsize=(8, 5))
    plt.plot(ns, times, marker='o')
    plt.xlabel("Number of Rooms (n)")
    plt.ylabel("Runtime (seconds)")
    plt.title("Runtime vs. Number of Rooms (n)
        ")
    plt.grid(True)
    plt.tight_layout()

def plot_loglog(results):
    ms = [m for (_, m, _) in results]
    times = [t for (_, _, t) in results]

    plt.figure(figsize=(8, 5))
    plt.loglog(ms, times, marker='o', base=10
    )
    plt.xlabel("Number of APs (m) [log scale]
        ")
    plt.ylabel("Runtime (seconds) [log scale]
        ")
    plt.title("Log Log Plot: Runtime vs.
        Number of APs (m)")
    plt.grid(True, which="both", ls="--")
    plt.tight_layout()

if __name__ == "__main__":
    print("\n=== Experiment 1: Vary m ===")
    n_values = [200] # Fix number of rooms
    m_values = [50, 100, 150, 200, 250, 300,
        400, 500]

    results_m = timing_experiment(n_values,
        m_values, trials=3)
    plot_results(results_m)
    plot_loglog(results_m)

    print("\n=== Experiment 2: Vary n ===")
    n_values_large = [50, 100, 150, 200, 300,
        400, 500, 600]

    results_n = experiment_vary_n(
        n_values_large, m=200)
    plot_vary_n(results_n)

    plt.show()

```

Listing 1. Python script for runtime validation of Greedy WI-FI AP Placement.

REFERENCES

- [1] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.
- [3] J. Zhang, Z. Zhang, and W. Li, "A survey of approximation algorithms for the power cover problem," *Mathematics*, vol. 13, no. 15, p. 2479, 2025.

- [4] A. Kamerman and I. Erkoç, "A survey of wireless lan technologies," *IEEE Communications Magazine*, vol. 35, no. 10, pp. 82–88, 1997.
- [5] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *IEEE International Conference on Communications (ICC)*. IEEE, 2001, pp. 472–476.