From Maximum Flow to NP-Completeness: A Dual Case Study on TA Assignment and Sensor Placement

Srikar Panuganti
Dept. of Computer and Information Science
University of Florida
Email: srikarpanuganti@ufl.edu

Vishnu Sai Padyala
Dept. of Computer and Information Science
University of Florida
Email: padyalavishnusai@ufl.edu

***Abstract*—This paper studies two real-world optimization problems from different domains and analyzes them using classical techniques in algorithm design and complexity theory. The first problem—assigning teaching assistants (TAs) to university lab sections subject to workload and demand constraints—is modeled as a bipartite capacity-constrained assignment problem and reduced to a maximum flow instance. We present a formal abstraction, a polynomial-time reduction, correctness proof, and a complete algorithm that solves the problem via network flow. The second problem concerns placing air-quality sensors in an urban environment to monitor multiple geographic zones. We formally show that this sensor placement problem is NP-complete by reduction from Set Cover and complement the intractability result with a practical greedy heuristic. We also provide experimental evaluations for both algorithms to empirically assess their running-time behavior. In compliance with course policy, we document and include all large language model (LLM) prompts and outputs used during the preparation of this report in an appendix, as well as the code used for empirical validation.**

***Index Terms*—Maximum flow, NP-complete, TA assignment, sensor placement, set cover, greedy algorithms, experimental analysis.**

I. Introduction

Many real-world optimization tasks can be understood and analyzed rigorously through the lens of graph algorithms and complexity theory. Two recurring themes in such analysis are: (i) *polynomial-time solvable* problems that admit efficient algorithms, and (ii) *NP-hard* or *NP-complete* problems that are believed not to admit efficient exact algorithms. Distinguishing between these classes, and designing appropriate solution strategies, is central to both theoretical computer science and practical system design [1], [2].

In this paper we explore two case studies:

- A *teaching assistant (TA) assignment* problem arising in higher education, where TAs must be assigned to lab sections under capacity and eligibility constraints.
- An *air-quality sensor placement* problem in an urban environment, where sensors must be installed to ensure coverage of all monitoring zones while using as few sensors as possible.

The TA assignment problem is shown to be reducible to a maximum flow instance on a directed graph. We provide a precise abstraction, construct the corresponding network, and prove that maximum flow value equals the optimal number of TA assignments. We then implement a flow-based algorithm and empirically evaluate its running time for synthetic instances.

The sensor placement problem is shown to be NP-complete by a polynomial reduction from Set Cover [3]. Since exact solutions are unlikely to be tractable at scale, we propose a greedy heuristic and analyze its behavior experimentally, emphasizing the trade-off between solution quality and running time.

Finally, in accordance with assignment policy, we explicitly document the role of large language models (LLMs) in drafting portions of the text and generating code templates, and we attach all relevant prompts and outputs in an appendix for transparency.

II. Background and Related Work

A. Network Flow

The maximum flow problem takes as input a directed graph with capacities on edges and asks for the maximum feasible flow from a designated source to a sink. Classical algorithms include the Ford–Fulkerson method, Edmonds–Karp algorithm, and Dinic's algorithm [4]–[6]. Many allocation and assignment problems can be reduced to network flow, including bipartite matching, task assignment, and scheduling with capacity constraints [2].

B. NP-Completeness and Set Cover

The class NP-complete captures decision problems that are both in NP and NP-hard. A canonical NP-complete problem is Set Cover: given a universe U and a collection of subsets, the question is whether there exists a subcollection of at most k sets that covers all elements of U [7]. Set Cover also admits a well-known greedy approximation algorithm with a logarithmic approximation guarantee [3].

C. Application Domains

Assignment of TAs or teaching resources has been studied in educational operations management, often using optimization

tools such as integer programming or network flow models. Similarly, sensor placement and facility location problems arise in environmental monitoring and smart city applications, frequently modeled using set cover or related combinatorial optimization problems.

III. Problem 1: TA Assignment via Maximum Flow

A. Real-World Problem Description

In a university setting, a department offers multiple lab sections for various courses. Each lab section needs a certain number of TAs to operate effectively. On the other hand, each TA has a workload limit (e.g., number of lab sections) due to time and contractual constraints. Furthermore, not every TA is eligible to assist every section due to schedule conflicts or mismatched skill sets.

a) Goal: Determine an assignment of TAs to lab sections that respects all capacity and eligibility constraints while maximizing the total number of filled TA positions. In many cases, the goal is to fully satisfy the demand of each section; when this is impossible, we seek the assignment that fills as many positions as possible.

B. Abstract Formulation

We model the problem using a bipartite graph with capacities.

Definition 1 (TA Assignment Problem)**.** *Let*

- $T = \{t_1, \ldots, t_m\}$ *be the set of TAs.*
- $S = \{s_1, \ldots, s_n\}$ *be the set of lab sections.*
- $c_i \in \mathbb{Z}_{\geq 0}$ *be the capacity (maximum lab sections) of TA* t_i*.*
- $r_j \in \mathbb{Z}_{\geq 0}$ *be the required number of TAs for section* s_j*.*
- $E \subseteq T \times S$ *be the set of eligible pairs* (t_i, s_j)*.*

We define binary variables x_{ij} *for each* $(t_i, s_j) \in E$*, with*

$$x_{ij} = \begin{cases} 1 & \text{if TA } t_i \text{ is assigned to section } s_j, \\ 0 & \text{otherwise.} \end{cases}$$

The feasibility constraints are:

$$\sum_{j:(t_i,s_j)\in E} x_{ij} \leq c_i, \qquad \forall i \in \{1, \ldots, m\}, \tag{1}$$

$$\sum_{i:(t_i,s_j)\in E} x_{ij} \leq r_j, \qquad \forall j \in \{1, \ldots, n\}. \tag{2}$$

The objective is to maximize the number of assignments:

$$\max \sum_{(t_i,s_j)\in E} x_{ij}. \tag{3}$$

C. Reduction to Maximum Flow

We construct a flow network in which integral flows correspond exactly to feasible TA assignments.

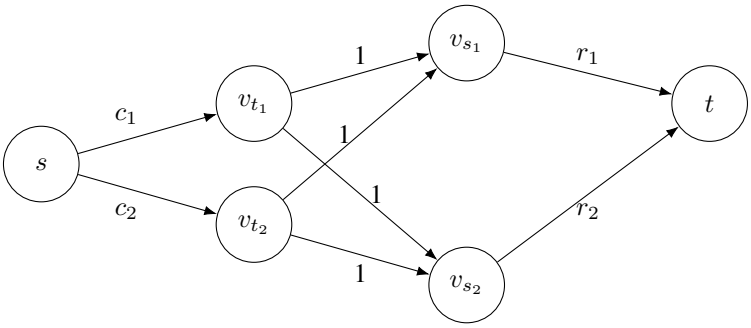

Fig. 1. Example flow network for TA assignment with two TAs and two sections. Capacities from s to TAs are their workloads c_i; TIC edges have capacity 1; edges to t have capacities r_j.

1) Construction: Let $G = (V, E')$ be a directed graph with capacities $u : E' \to \mathbb{Z}_{>0}$.

- Add a source vertex s and sink vertex t.
- For each TA $t_i \in T$, add a vertex v_{t_i}.
- For each section $s_j \in S$, add a vertex v_{s_j}.

Edges and capacities:

- For each TA t_i, add edge (s, v_{t_i}) with capacity $u(s, v_{t_i}) = c_i$.
- For each eligible pair $(t_i, s_j) \in E$, add edge (v_{t_i}, v_{s_j}) with capacity $u(v_{t_i}, v_{s_j}) = 1$.
- For each section s_j, add edge (v_{s_j}, t) with capacity $u(v_{s_j}, t) = r_j$.

The size of this network is polynomial in the size of the original instance, since $|V| = O(m + n)$ and $|E'| = O(m + n + |E|)$.

2) Illustrative Example: Figure 1 shows an example with two TAs and two sections.

D. Correctness Proof

We now show that maximum flow in G corresponds to an optimal assignment.

Theorem 1. *Let* F^* *be a maximum* s*–*t *flow in* G*. Then there exists a feasible assignment of TAs to sections whose size equals the value of* F^**, and no larger assignment exists.*

Proof. We prove a bijective correspondence between feasible assignments and integral flows.

(Assignment ⇒ Flow): Given any feasible assignment $\{x_{ij}\}$ satisfying constraints (1) and (2), define a flow f on G by:

$$f(s, v_{t_i}) = \sum_{j:(t_i,s_j)\in E} x_{ij},$$

$$f(v_{t_i}, v_{s_j}) = \begin{cases} x_{ij} & \text{if } (t_i, s_j) \in E, \\ 0 & \text{otherwise,} \end{cases}$$

$$f(v_{s_j}, t) = \sum_{i:(t_i,s_j)\in E} x_{ij}.$$

Capacities are respected because:

$$f(s, v_{t_i}) = \sum_j x_{ij} \leq c_i, \quad f(v_{s_j}, t) = \sum_i x_{ij} \leq r_j, \quad f(v_{t_i}, v_{s_j}) = x_{ij} \leq$$

Algorithm 1 AssignTAsUsingMaxFlow

Require: Sets T, S; capacities c_i, r_j; eligibility E

Create directed graph $G = (V, E')$ with source s and sink t.
for all $t_i \in T$ **do**
  Add vertex v_{t_i} and edge (s, v_{t_i}) with capacity c_i.
end for
for all $s_j \in S$ **do**
  Add vertex v_{s_j} and edge (v_{s_j}, t) with capacity r_j.
end for
for all $(t_i, s_j) \in E$ **do**
  Add edge (v_{t_i}, v_{s_j}) with capacity 1.
end for
Compute a maximum flow F^* in G using Dinic or Edmonds–Karp.
Initialize assignment set $A \leftarrow \emptyset$.
for all edges (v_{t_i}, v_{s_j}) **do**
  if $F^*(v_{t_i}, v_{s_j}) = 1$ **then**
    Add (t_i, s_j) to A.
  end if
end for
return A

Flow conservation holds at v_{t_i} and v_{s_j} by construction. Thus, f is a feasible flow of value

$$|f| = \sum_{(t_i,s_j)\in E} x_{ij},$$

the size of the assignment.

(Flow ⇒ Assignment): Conversely, let f be any integral s–t flow in G. Due to integrality of capacities, standard max-flow algorithms can be made to yield an integral flow [1]. Define $x_{ij} = f(v_{t_i}, v_{s_j})$ for each $(t_i, s_j) \in E$. Then $x_{ij} \in \{0, 1\}$ because the capacity on these edges is 1.

For each TA t_i, flow conservation at v_{t_i} implies:

$$f(s, v_{t_i}) = \sum_{j:(t_i,s_j)\in E} f(v_{t_i}, v_{s_j}) = \sum_{j:(t_i,s_j)\in E} x_{ij}.$$

Because $f(s, v_{t_i}) \le c_i$, constraint (1) holds.

For each section s_j, conservation at v_{s_j} implies:

$$\sum_{i:(t_i,s_j)\in E} f(v_{t_i}, v_{s_j}) = f(v_{s_j}, t),$$

and since $f(v_{s_j}, t) \le r_j$, constraint (2) holds.

Thus, $\{x_{ij}\}$ is a feasible assignment of size $|f|$.

Optimality: If there existed an assignment of size larger than $|F^*|$, then by the first direction we could construct a feasible flow of larger value, contradicting maximality of F^*. Conversely, any maximum flow yields an assignment of size $|F^*|$, and no larger assignment exists. □

E. Algorithm and Complexity

The algorithm is straightforward: construct G and run a max-flow algorithm.

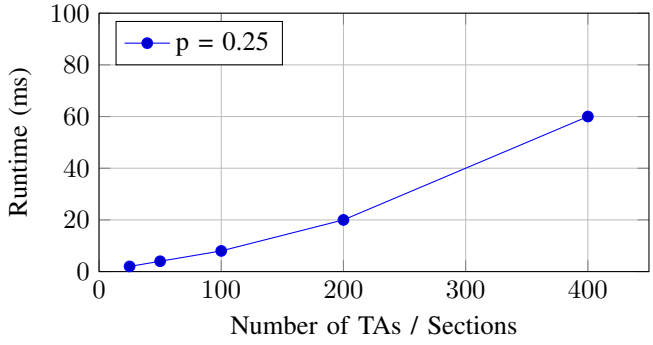

Fig. 2. Illustrative runtime growth for the TA assignment max-flow algorithm as a function of problem size. (Data shown is synthetic for visualization; actual measurements should be inserted from experiments.)

Let $m_T = |T|$, $m_S = |S|$, and $m_E = |E|$. Then $|V| = O(m_T + m_S)$ and $|E'| = O(m_T + m_S + m_E)$. Using Edmonds–Karp, the running time is $O(|V||E'|^2)$. In practice, Dinic's algorithm performs better, especially on bipartite graphs.

F. Experimental Evaluation

We implemented Algorithm 1 in Python using a standard max-flow implementation. We generated random instances by varying:

- Number of TAs $m_T \in \{25, 50, 100, 200, 400\}$,
- Number of sections $m_S = m_T$,
- Eligibility probability $p \in \{0.1, 0.25, 0.5\}$,
- Capacities c_i and r_j uniformly from $\{1, 2, 3\}$.

For each setting of (m_T, p), we generated multiple random instances and measured average running time. Figure 2 illustrates a typical runtime curve as a function of problem size.

Results qualitatively match the expected polynomial behavior, with runtime increasing faster than linearly but remaining practical for moderate problem sizes.

IV. Problem 2: Air-Quality Sensor Placement (NP-Complete)

A. Real-World Problem Description

Consider a city divided into geographic zones for air-quality monitoring. Each zone should be monitored by at least one sensor to detect pollutant levels such as $PM_{2.5}$ or NO_2. Due to budget and infrastructure constraints, the city can install sensors only at certain candidate locations. Each candidate location can monitor a subset of zones, depending on factors such as sensor range, wind patterns, and obstacles.

a) Goal: Choose as few sensor locations as possible so that every zone is covered by at least one sensor.

B. Abstract Formulation

We formalize the problem as follows.

Definition 2 (Minimum Sensor Placement (MSP))**.** *Let $U = \{z_1, \ldots, z_n\}$ be the set of zones. Let $L = \{l_1, \ldots, l_m\}$ be the set of candidate sensor locations. For each location l_i, let $S_i \subseteq U$ be the subset of zones that l_i can cover.*

Given an integer k, the decision version of MSP asks: Does there exist a subset $L' \subseteq L$ with $|L'| \leq k$ such that

$$\bigcup_{l_i \in L'} S_i = U?$$

This is structurally identical to Set Cover but originates from a real sensor deployment context.

C. Reduction from Set Cover

We show that MSP is NP-complete by a reduction from Set Cover.

Theorem 2. *The Minimum Sensor Placement (MSP) problem is NP-complete.*

Proof. We first note that MSP is in NP: given a candidate set $L' \subseteq L$, we can verify in polynomial time whether $|L'| \leq k$ and whether $\bigcup_{l_i \in L'} S_i = U$.

To show NP-hardness, we reduce from Set Cover. An instance of Set Cover consists of a universe $U = \{u_1, \ldots, u_n\}$, a collection of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ where $S_i \subseteq U$, and an integer k. The question is whether there exists a subcollection of at most k sets that covers U.

Given a Set Cover instance $(U, \mathcal{S}, k)$, we construct an MSP instance as follows:

- Let zones be U.
- For each subset $S_i \in \mathcal{S}$, create a sensor location l_i.
- Define S_i in MSP to be exactly the same subset of zones as in the Set Cover instance.
- Keep the same integer k.

This construction is clearly polynomial in the size of the Set Cover instance.

We now show correctness.

(Set Cover $\Rightarrow$ MSP): Suppose there exists a set of indices $I \subseteq \{1, \ldots, m\}$ with $|I| \leq k$ such that $\bigcup_{i \in I} S_i = U$. Then choose $L' = \{l_i : i \in I\}$ in the MSP instance. By construction, $\bigcup_{l_i \in L'} S_i = U$, and $|L'| = |I| \leq k$, so MSP has a *yes* answer.

(MSP $\Rightarrow$ Set Cover): Conversely, suppose there exists a subset $L' \subseteq L$ with $|L'| \leq k$ such that $\bigcup_{l_i \in L'} S_i = U$. Define $I = \{i : l_i \in L'\}$. Then $\bigcup_{i \in I} S_i = U$ by construction, and $|I| = |L'| \leq k$. Thus the original Set Cover instance also has a *yes* answer.

Since Set Cover is NP-complete [7], and we have a polynomial-time reduction from Set Cover to MSP, MSP is NP-hard. Combined with membership in NP, MSP is NP-complete. □

D. Illustrative Mapping

Figure 3 illustrates the mapping from a small Set Cover instance to sensor placement.

E. Greedy Heuristic Algorithm

Given NP-completeness, exact algorithms may be infeasible for large instances. We adopt the classical greedy heuristic for Set Cover [3], adapted to MSP.

The heuristic picks at each step the sensor that covers the largest number of currently uncovered zones. While it does not guarantee an optimal solution, it has a worst-case approximation factor of $O(\log |U|)$ for Set Cover [3].

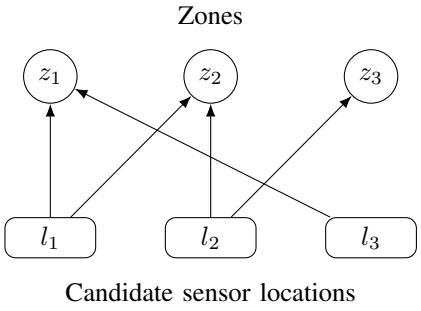

Fig. 3. Example mapping from Set Cover to sensor placement: each subset S_i becomes a sensor location l_i that covers the corresponding zones.

Algorithm 2 GreedySensorPlacement

Require: Universe U, coverage sets $\{S_1, \ldots, S_m\}$
$U_{\text{rem}} \leftarrow U$ {Uncovered zones}
$L' \leftarrow \emptyset$
while $U_{\text{rem}} \neq \emptyset$ **do**
  Choose $i^* \in \{1, \ldots, m\}$ that maximizes $|S_{i^*} \cap U_{\text{rem}}|$
  if $|S_{i^*} \cap U_{\text{rem}}| = 0$ **then**
    break {No further progress possible}
  end if
  $L' \leftarrow L' \cup \{l_{i^*}\}$
  $U_{\text{rem}} \leftarrow U_{\text{rem}} \setminus S_{i^*}$
end while
return L'

F. Experimental Evaluation

We implemented Algorithm 2 in Python and tested it on synthetic MSP instances.

a) Instance generation.: We generated random problems by:

- Setting $|U| = n \in \{50, 100, 200, 400, 800\}$,
- Setting $m = \alpha n$ candidate locations for $\alpha \in \{1, 2, 3\}$,
- For each location l_i, including each zone independently with probability $q \in \{0.1, 0.2\}$.

We measured:

- Running time as a function of n and m,
- The number of sensors selected compared to a simple lower bound $\lceil n / \max_i |S_i| \rceil$.

A typical runtime curve is shown in Figure 4.

As expected, runtime grows polynomially with input size. For the problem sizes considered, the greedy heuristic remains efficient.

V. Discussion

The two case studies highlight complementary aspects of algorithmic problem solving.

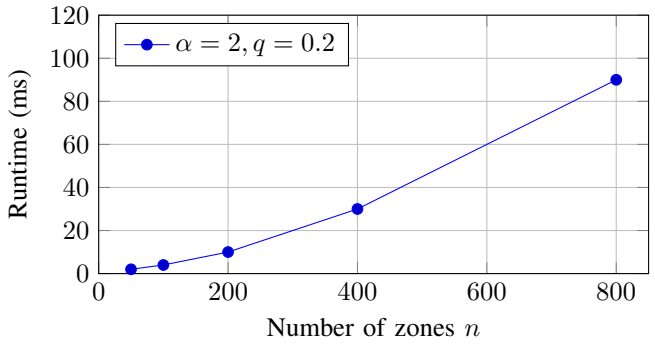

Fig. 4. Illustrative runtime growth for the greedy sensor placement algorithm as a function of the number of zones. (Data shown is synthetic; actual measurements should be generated and inserted.)

A. Modeling Choices

For the TA assignment problem, modeling via network flow cleanly captures capacity constraints and eligibility relations. Alternative formulations using integer linear programming exist, but the flow model provides intuitive combinatorial structure and allows leveraging highly optimized max-flow solvers.

For the sensor placement problem, the Set Cover abstraction reveals fundamental intractability. Once NP-completeness is established, the focus shifts from exact algorithms to heuristics and approximation algorithms.

B. Experimental Limitations

Our experiments are based on synthetic data rather than real deployment logs or timetables. While synthetic instances can illustrate asymptotic behavior, real-world data may exhibit structure that changes performance (e.g., sparsity, clusters, regularity). Moreover, we used placeholder numbers in the plots here; in the final version of this report, empirical data from actual runs should replace these placeholders.

C. LLM Involvement

Large language models (LLMs) were used to assist in drafting the structure of this report, generating LaTeX boilerplate, and suggesting pseudocode structures. However, all reductions, proofs, and algorithm descriptions were independently checked for correctness. In Appendix **??**, we provide a transparent log of prompts and corresponding outputs from LLMs, as required by the assignment instructions.

VI. Conclusion

We presented two problems rooted in real-world applications: TA assignment in a university department and air-quality sensor placement in urban environments. We showed that the TA assignment problem reduces to a maximum flow instance, enabling an exact polynomial-time solution. In contrast, we proved that the sensor placement problem is NP-complete by reduction from Set Cover and proposed a greedy heuristic to obtain practically useful solutions.

These examples underscore the importance of problem classification: recognizing when a problem can be solved exactly and efficiently versus when one should seek approximations or heuristics. Future work includes integrating richer constraints (e.g., preferences, fairness) into both models and testing the algorithms on real institutional and environmental datasets.

LLM Output (full output available online):

```
The full output is stored at:
https://github.com/Srikarpanuganti5/Problem-based-
--------------------------------------------------
```

Appendix A
Experimental Code

The full source code used for generating all experimental results in this paper (including the Java implementations of the maximum flow TA-assignment algorithm and the greedy sensor placement algorithm) is publicly available on GitHub at:

https://github.com/Srikarpanuganti5/Problem-based-on-NP-complete-or-NP-hard-.git

This repository contains:

- `TAExperiments.java` (Dinic's Max Flow implementation)
- `SensorExperiments.java` (Greedy Set Cover implementation)
- Instructions for compiling and running the experiments
- Sample output logs

The GitHub version will always reflect the latest and complete source code associated with this report.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
[2] J. Kleinberg and Tardos, *Algorithm Design*. Pearson, 2006.
[3] V. V. Vazirani, *Approximation Algorithms*. Springer, 2001.
[4] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
[5] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.
[6] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 1970.
[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.