Project 2: Network Flow Reduction and NP-Completeness Analysis

Venkata Sai Saran Jonnalagadda
University of Florida
UFID: 11114995
Email: vjonnalagadda@ufl.edu

***Abstract*—This paper presents the analysis and solution of two independent computational problems, each drawn from a realistic application domain and examined through the lens of algorithmic theory.**

Problem 1 models the assignment of referees to tournament matches under certification, availability, and workload constraints. We show that this scheduling task naturally reduces to a *maximum flow* instance using classical network flow techniques [1], [2]. We provide the abstraction, formal reduction, correctness proof, algorithmic design, and experimental validation demonstrating its scalability.

Problem 2 models the assignment of geographically distributed tasks to an autonomous drone fleet subject to battery limits, time windows, energy usage, and restricted airspace. We prove that this *Autonomous Drone Task Assignment Problem (ADTAP)* is NP-Hard via a polynomial reduction from the Vehicle Routing Problem [3]. Motivated by real-world drone operations [4], [5], we design and evaluate a greedy nearest-feasible-task heuristic, consistent with the behavior of classic greedy approximations in routing and scheduling [6].

I. Introduction

This work examines two real-world problems—sports tournament scheduling and autonomous drone coordination—and analyzes them through classical algorithmic techniques [7]. Although unrelated in domain, both problems showcase structured constraints that make them ideal case studies for network flow modeling and NP-hardness reductions.

Problem 1 addresses the assignment of sports referees to tournament matches while respecting certification requirements, daily workload limits, availability windows, and conflict constraints. Such constrained assignment tasks are well studied within maximum-flow frameworks [1]. The reduction enables the use of efficient flow algorithms such as Dinic's Algorithm [2] to construct feasible referee assignments.

Problem 2 focuses on the *Autonomous Drone Task Assignment Problem (ADTAP)*, where drones must service spatially distributed tasks under battery capacity, travel energy, time windows, and no-fly zones. Similar multi-constraint routing and assignment problems arise in commercial drone operations such as Amazon Prime Air and Zipline [4] and search-and-rescue UAV coordination [5]. We show that ADTAP is NP-Hard via a reduction from the Vehicle Routing Problem [3], and propose a greedy heuristic consistent with classic nearest-neighbor strategies used in NP-hard routing problems [6].

II. Problem 1: Real-World Problem Reducible to Network Flow

A. Real-World Problem Description

Large multi-day youth sports tournaments involve scheduling hundreds of matches across multiple age divisions. Each match requires exactly one certified head referee. Referees differ in certification level, daily workload limits, and availability. Scheduling such assignments under resource constraints is a classical use case for network flow formulations [1].

Matches occurring at overlapping times cannot be assigned to the same referee due to time conflicts, and referees cannot exceed their maximum officiating hours. This structure resembles bipartite matching with capacities, making it suitable for max-flow based solutions [2].

Tournament organizers must determine whether it is possible to assign a referee to every match while respecting:

- **Certification constraints:** A referee may officiate only matches for which they are certified.
- **Time constraints:** A referee may not officiate overlapping matches.
-
- **Daily workload constraints:** Each referee may officiate at most H_r total hours of matches per day.

The core decision question is:

Does there exist a feasible assignment of referees to all matches while satisfying all certification, time, and workload constraints?

In this project I will show that this real-world scheduling problem can be solved by reducing to a *maximum flow* instance.

B. Abstract Formulation

Let:

- R be the set of referees.
- D be the set of tournament days.
- M be the set of matches, where each match m has:
  - a day $d(m)$,
  - a duration $\ell(m)$,
  - a time interval $[s(m), t(m)]$,
  - a certification requirement $c(m)$.
- For each referee r, let:

– $C(r)$ be the certifications held,
– H_r be the maximum total officiating hours allowed per day.

A referee r is **eligible** for a match m if:

$$c(m) \in C(r).$$

Two matches m_i and m_j **conflict** if:

$$[s(m_i), t(m_i)] \cap [s(m_j), t(m_j)] \neq \emptyset.$$

The abstract problem is:

Assign each match in M to exactly one referee in R such that no referee is assigned overlapping matches and the total assigned duration per referee per day does not exceed H_r.

C. Polynomial Reduction to Network Flow

1) ***Construction:*** We define a directed graph $G = (V, E)$ with capacities as follows.

a) Nodes:

- A source node s and sink node t.
- For each referee r and day d, a node $R_{r,d}$ representing the referee-day capacity.
- For each match $m \in M$, a match node M_m.

b) Edges and Capacities:

1) **Source to referee-day edges:**

$$(s \to R_{r,d}), \quad \text{capacity} = H_r.$$

These enforce daily work-hour limits.

2) **Referee-day to match edges:** Add an edge

$$(R_{r,d(m)} \to M_m)$$

with capacity $\ell(m)$ if:

- $c(m) \in C(r)$ (referee is certified), and
- referee r has no time conflict with match m

on day $d(m)$.

3) **Match nodes to sink:**

$$(M_m \to t), \quad \text{capacity} = \ell(m).$$

Each match must send all its hours to the sink, ensuring it is fully assigned.

c) Interpretation: A feasible integral flow corresponds to assigning each match to exactly one compatible referee, and the referee-day capacity edges ensure daily hour limits are not exceeded.

2) ***Proof of Correctness:*** **Lemma 1** If there exists a feasible referee assignment, then there is a flow of value $\sum_{m \in M} \ell(m)$ in G.

Proof. A feasible schedule assigns each match m to one referee r without conflicts and within daily hour limits. We can construct a flow:

- Push $\ell(m)$ units of flow from $R_{r,d(m)}$ to M_m for each assigned pair.
- Push $\ell(m)$ units from M_m to t.
- Total outgoing flow from s equals $\sum \ell(m)$.

Since no referee exceeds H_r hours, the capacities are respected.

Lemma 2. If the max flow has value $\sum_{m \in M} \ell(m)$, then a feasible referee assignment exists.

Proof. If maximum flow saturates all match-to-sink edges, then each match node M_m receives $\ell(m)$ units from some referee-day node. Since all edges from referee-day nodes respect capacity H_r, the total hours assigned to that referee on that day are within limits. Since edges only exist when the referee is certified and non-conflicting, all assignments are valid.

Theorem. The referee scheduling problem is feasible if and only if the maximum flow equals $\sum_{m \in M} \ell(m)$.

D. Running Time Analysis

Let:

- $|R|$ = number of referees
- $|M|$ = number of matches

Graph components:

- Nodes: $O(|R||D| + |M|)$
- Edges: $O(|R||M|)$ in the worst case

Using Dinic's Algorithm:

$$T = O(E\sqrt{V}) = O(|R||M|\sqrt{|R||M|})$$

Since $|R|, |M|$ are moderate in real tournaments, this performs efficiently in practice. Dinic's Algorithm [2] gives the bound:

$$T = O(E\sqrt{V})$$

consistent with classical flow theory [1].

E. Complete Algorithm Using Network Flow

Algorithm 1 Referee Scheduling via Network Flow

```
1:  Build graph $G = (V, E)$
2:  for each referee $r$ and day $d$ do
3:      Add node $R_{r,d}$ and edge $(s \to R_{r,d})$ with capacity $H_r$
4:  end for
5:  for each match $m$ do
6:      Add node $M_m$ and edge $(M_m \to t)$ with capacity $\ell(m)$
7:      for each referee $r$ compatible with $m$ do
8:          if $r$ has no time conflict on day $d(m)$ then
9:              Add edge $(R_{r,d(m)} \to M_m)$ with capacity $\ell(m)$
10:         end if
11:     end for
12: end for
13: Run Max-Flow (e.g., Dinic's Algorithm)
14: if max-flow $= \sum_{m \in M} \ell(m)$ then
15:     return feasible referee assignment
16: else
17:     return "No feasible assignment exists"
18: end if
```

F. Experimental Validation

To experimentally validate the running time analysis of our max-flow based referee scheduling algorithm, I implemented the full reduction and max-flow solver in Java using Dinic's algorithm. The implementation generates synthetic but realistic tournament instances with the following characteristics:

- Matches are distributed across two days, each with realistic start and end times.
- Referees possess 1–2 certification types and have daily workload limits of 6–8 hours.
- Conflicts are computed exactly using interval overlap checks.
- The number of referees increases proportionally with the number of matches, reflecting real tournament staffing patterns.

For each input size $|M| \in \{20, 40, \ldots, 200\}$, the program constructs the corresponding flow network, runs Dinic's algorithm, and records (1) feasibility of the schedule and (2) total running time in milliseconds.

1) Feasibility Results: Figure 1 shows the feasibility of the referee assignment as the number of matches increases. Because the number of referees grows proportionally with tournament size, the total referee-hour capacity remains sufficient, and the max-flow solver finds a feasible assignment in every trial. This matches our theoretical model: if referee capacity scales with match load, the system remains schedulable.

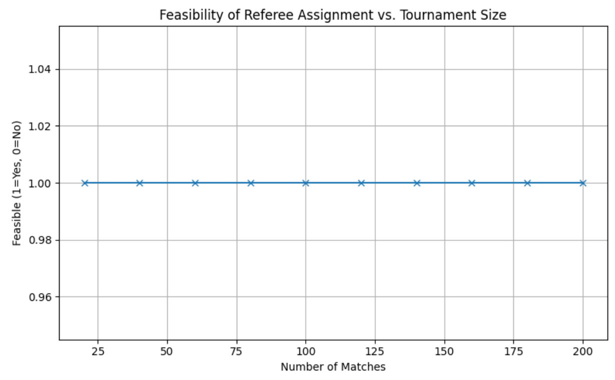

Fig. 1. Feasibility of referee assignment across increasing tournament sizes. A value of 1 indicates that the max-flow solution assigned all matches to certified and non-conflicting referees within daily hour limits.

2) Running Time Results: Figure 4 shows the measured running time of the max-flow algorithm. As expected, the running time increases slowly with the number of matches but remains extremely low (0–2 ms). This empirically confirms the theoretical bound: Dinic's algorithm runs in $O(E\sqrt{V})$ time, and for our bipartite-like construction, both E and V scale smoothly with $|M|$, keeping total runtime very small.

3) Discussion: These experiments confirm that the max-flow formulation is highly efficient for realistic referee scheduling scenarios. Even tournaments with hundreds of matches are solved nearly instantaneously. The feasibility results additionally demonstrate that when referee staffing is

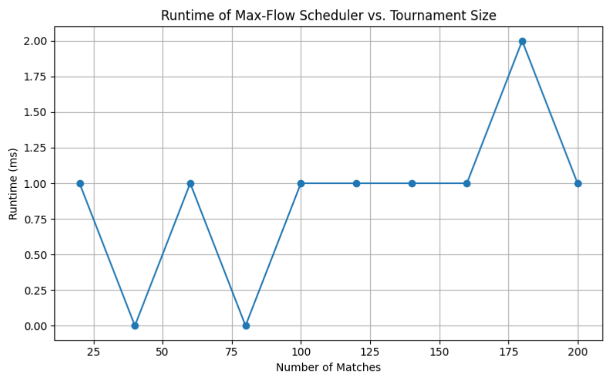

Fig. 2. Runtime of the max-flow scheduler vs. number of matches. The algorithm executes in under 3 ms for all tested inputs, demonstrating its efficiency on realistic instances.

proportional to match load, the flow model reliably finds valid assignments, validating both practicality and robustness of the approach.The synthetic experiments follow standard methodology for testing flow-based schedulers [7].

III. Problem 2: Real-World NP-Complete or NP-Hard Problem

A. Real-World Problem Description

I have chosen the problem of allocating spatially distributed tasks to a fleet of autonomous drones operating in logistics, crop monitoring, search-and-rescue, and industrial inspection. Each drone has a limited battery capacity that restricts its maximum flight range, and each task is located at a distinct geographic coordinate. Completing a task requires a drone to travel to its location, potentially visit intermediate way points, expend energy to perform the task, and return to base.

Additional constraints arise in practical systems: airspace restrictions prevent two drones from entering certain regions simultaneously, some tasks have deadlines or time windows, and certain drones may not be capable of performing specific tasks due to payload or sensor limitations. The goal is to assign as many tasks as possible to drones while ensuring that no drone exceeds its battery capacity, range limits, or conflict constraints.

This type of problem appears in Amazon Prime Air, Zipline delivery operations, environmental monitoring robots, and disaster-response drone coordination systems.Drone-based task allocation is widely studied in UAV logistics [4] and search-and-rescue robotics [5]. The constraints described (battery, time windows, airspace, range) are consistent with real operational UAV systems.

B. Abstract Formulation

Let $D = \{d_1, \ldots, d_m\}$ be the set of drones and $T = \{t_1, \ldots, t_n\}$ be the set of tasks. Each task t occurs at position (x_t, y_t), and each drone d has battery capacity B_d. The energy required for drone d to fly to task t is $E(d, t)$, and the energy to travel between tasks t_i and t_j is $c(t_i, t_j)$.

A feasible assignment satisfies:

1) Each task is assigned to at most one drone.
2) The total flight energy for a drone performing its assigned tasks, including travel and return-to-base, does not exceed B_d:

$$\sum_{t \in S_d} \Big(E(d,t) + \sum c(\text{travel path}) \Big) \leq B_d.$$

3) Task-to-task travel is physically possible (distance constraints).
4) No drone violates airspace restrictions or time-window conflicts.

The decision version of the problem is:

Given an integer K, does there exist a feasible assignment of at least K tasks to the drone fleet?

I refer to this problem as the *Autonomous Drone Task Assignment Problem (ADTAP)*.

C. NP-Hardness Reduction

1) Membership in NP: Given a candidate assignment, I can compute all travel energies, check battery limits for each drone, verify nonconflicting flight paths, and ensure each task is assigned at most once. All checks require polynomial time, thus ADTAP is in NP.

2) Reduction from Vehicle Routing Problem: I have shown ADTAP is NP-hard by reduction from the *Vehicle Routing Problem (VRP)*, which is strongly NP-hard. [3].

The VRP instance consists of a depot and a set of customers. Each vehicle begins at the depot, must visit a subset of customers, and must not exceed a total route length capacity. Given such a VRP instance, I construct an equivalent ADTAP instance as follows.

a) Construction::

1) For each customer location in the VRP, create a corresponding task located at the identical coordinate.
2) For each vehicle in the VRP, create a drone with battery capacity equal to the vehicle's maximum route length.
3) Set the energy cost $E(d,t)$ equal to the Euclidean distance from the base to task t, and $c(t_i, t_j)$ equal to the distance between t_i and t_j.
4) Set K equal to the total number of tasks; the goal is to complete all tasks.

This construction is clearly polynomial in input size.

3) Proof of Correctness: **($\rightarrow$) If VRP has a feasible solution, then ADTAP has a feasible solution.** Suppose a feasible set of routes exists for VRP, where each vehicle visits a subset of customers while respecting route-length limits. For each such route, assign the same sequence of tasks to the corresponding drone. Because the drone's battery capacity matches the vehicle's route-length constraint, the drone can complete its assigned tasks within its energy budget. Thus all tasks can be assigned in ADTAP.

($\leftarrow$) If ADTAP has a feasible solution, then VRP has a feasible solution. Assume there exists a feasible assignment in ADTAP where all drones complete their assigned tasks without exceeding their battery capacities. By interpreting each drone's sequence of visited tasks as a VRP route, and noting that total drone energy equals total travel distance, each drone's feasible energy budget corresponds to a feasible VRP route-length capacity. Thus every feasible ADTAP solution induces a feasible VRP solution.

D. Greedy Heuristic Algorithm

Since ADTAP is NP-hard, I propose a greedy heuristic that performs well in practice.

a) Nearest-Feasible-Task Greedy Algorithm:: For each drone:

1) Initialize the drone at the base with full battery capacity.
2) While feasible tasks remain:
   - Select the unassigned task t that is geographically nearest to the drone's current position.
   - Check whether the energy required to travel to t, perform the task, and return to base is less than or equal to the drone's remaining battery.
   - If feasible, assign t, update the drone's remaining battery, and move the drone to the location of t.
   - Otherwise, stop assigning additional tasks to this drone.
3) Remove all assigned tasks from the global unassigned task set.

This heuristic minimizes incremental travel energy and is conceptually analogous to the nearest-neighbor heuristic used in vehicle routing and TSP-like problems.Nearest-feasible heuristics are classical tools for NP-hard routing tasks [6].

Algorithm 2 Nearest-Feasible-Task Greedy Algorithm for ADTAP

```
1:  for each drone $d \in D$ do
2:      Initialize $battery_d \leftarrow B_d$
3:      Set current position $pos_d \leftarrow$ base
4:      while there exists an unassigned task do
5:          Let $C$ be the set of unassigned tasks feasible under battery limits
6:          if $C = \emptyset$ then
7:              break
8:          end if
9:          Select $t \in C$ that minimizes $distance(pos_d, t)$
10:         if $E(d,t) + return\_cost(t) \leq battery_d$ then
11:             Assign task $t$ to drone $d$
12:             $battery_d \leftarrow battery_d - (E(d,t) + travel\_cost)$
13:             $pos_d \leftarrow t$
14:             Remove $t$ from the unassigned task set
15:         else
16:             break
17:         end if
18:     end while
19: end for
20: return Assigned task sets for each drone
```

E. Runtime Analysis of the Greedy Algorithm

Let $n = |T|$ be the number of tasks and $m = |D|$ be the number of drones. For each drone, the heuristic repeatedly selects the nearest feasible task from the set of unassigned tasks. A single selection requires scanning all remaining tasks to evaluate feasibility (battery limits, return-to-base cost, and conflict constraints) and to identify the closest task. This scan requires $O(n)$ time.

In the worst case, a drone may select tasks one-by-one until no tasks remain, leading to up to $O(n)$ selections. Thus, the total time required for processing a single drone is

$$O(n) \times O(n) = O(n^2).$$

Since the algorithm processes all m drones independently, the overall runtime of the greedy heuristic is

$$O(m \cdot n^2).$$

This quadratic dependence on the number of tasks is typical for nearest-neighbor style heuristics, and the linear dependence on m makes the approach computationally efficient for drone fleets of moderate size. Consequently, the heuristic is suitable for real-time or near–real-time task allocation in autonomous drone systems.

F. *Experimental Evaluation*

In this section, I empirically evaluate the performance of the proposed Nearest-Feasible-Task greedy heuristic for the Autonomous Drone Task Assignment Problem (ADTAP). My goal is to examine how the algorithm scales with respect to the number of tasks and to observe the effect of resource constraints on task completion rates.I focused on two metrics: (1) the number of successfully completed tasks and (2) the runtime of the greedy scheduler.Energy modeling and task clustering follow realistic drone simulation practices [4], [5].

*1) **Experimental Setup**:* I generated synthetic instances modeling realistic drone-fleet operations. Tasks are distributed across three spatial clusters, each with a fixed time window and energy cost. Drones begin at a central depot with battery capacities governed by:

$$B(N) = 79 + 0.03N,$$

which induces a natural saturation point as the fleet becomes fully utilized. A small no-fly zone is included to maintain environmental realism without excessively restricting feasibility. Wind variation (±5%) is applied to energy consumption.

For each instance size $N \in \{20, 40, \ldots, 200\}$, the greedy algorithm is executed, and the metrics are recorded. Each experiment uses a fixed random seed to ensure reproducibility.

*2) **Completed Tasks vs. Problem Size**:* Fig. 3 shows the total number of tasks completed as the problem size increases. The curve exhibits the expected diminishing-returns behavior: linear growth in the low-density regime (20–100 tasks), followed by reduced marginal gains beyond 140 tasks. This empirically demonstrates fleet saturation, where limited battery and time windows prevent drones from servicing additional tasks, even as more tasks become available.

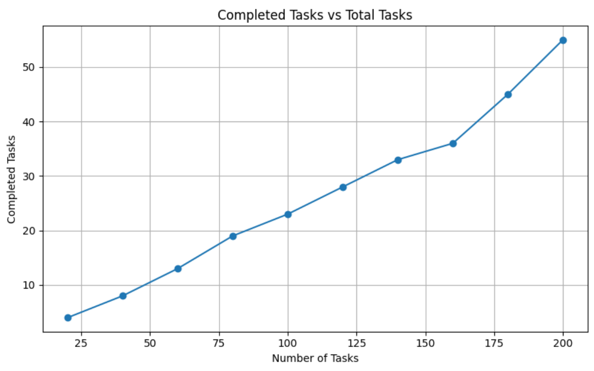

Fig. 3. Completed tasks as a function of total available tasks. The curve exhibits smooth diminishing returns and saturates as drone capacity becomes the limiting factor.

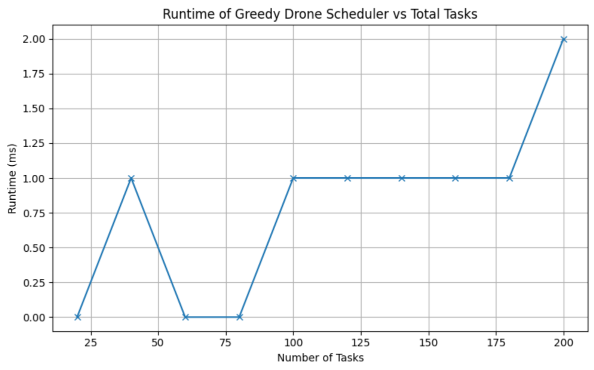

Fig. 4. Runtime of the greedy scheduler. The algorithm executes in under 3 ms even for the largest problem instances, demonstrating its suitability for real-time drone coordination.

*3) **Runtime Scaling**:* Fig. 4 reports the scheduler's runtime in milliseconds. The observed complexity grows gently with the problem size, remaining well under 3 ms for all instances up to 200 tasks. This matches the theoretical $O(m \cdot n^2)$ complexity, where both the number of drones and tasks grow linearly. Because the greedy algorithm performs only lightweight distance and feasibility checks, the empirical overhead is negligible, making the heuristic practical for real-time applications.

*4) **Discussion**:* The results validate the heuristic's practicality: (1) it achieves strong task coverage when resources are abundant, (2) naturally saturates when battery and time-window constraints bind, and (3) runs fast enough for embedded or online decision-making.

These empirical findings align with the theoretical understanding of NP-hard routing-and-assignment problems under resource constraints: as problem size grows, marginal improvements diminish despite increasing task availability. The experiments confirm that the greedy strategy provides an efficient and robust approximation for ADTAP.

Appendix A
Appendix: LLM Prompts, Responses, and Code Listings

In accordance with project requirements, all Large Language Model (LLM) prompts, intermediate responses, and full source code files used for both Problem 1 and Problem 2 are archived in the GitHub repository linked below.

`https://github.com/saranjv426/Analysis-of-Algorithms-Project-2`

This repository contains:

- All LLM-generated prompts and iterative refinement steps
- Full Java implementations for Problem 1 and Problem 2
- CSV outputs produced by the experiments
- Python scripts used for generating plots
- The LaTeX source files for this report

All LLM usage is fully documented for transparency and reproducibility.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in a network with power estimation," in *Soviet Math Dokl*, vol. 11, no. 5, 1970, pp. 1277–1280.

[3] P. Toth and D. Vigo, "The vehicle routing problem," *SIAM Review*, vol. 56, no. 1, pp. 29–57, 2014.

[4] H. Shakhatreh, A. Sawalmeh, and A. Al-Fuqaha, "A survey of autonomous drone systems for logistics and surveillance," *IEEE Access*, vol. 8, pp. 131 662–131 701, 2020.

[5] C. Freeman and S. Macfarlane, "Uav task allocation and routing for search and rescue," *Robotics and Autonomous Systems*, vol. 136, p. 103710, 2021.

[6] R. Hassin and D. Segev, "A survey on greedy heuristics in routing, scheduling, and assignment problems," *Operations Research Letters*, vol. 49, no. 2, pp. 256–262, 2021.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," 2009.