Facility Location Problem

By: Christian, Joshua, Blake, and Dustin

Our Problem and Practical Applications

- The basic premise is that we want to decide where to place a set of facilities (like warehouses, factories, etc.) to minimize the amount of facilities opened to service customers
- The goal for our variation is to select a minimum number of facilities such that every client is within the coverage distance of at least one facility

- Practical Applications:
    - Healthcare Facilities
        - Deciding where to open clinics or hospitals so that all residents are within a maximum travel distance
        - Example: Placing vaccination centers in a city to ensure coverage for all neighborhoods
    - Emergency Services
        - Determining locations for fire stations, police stations, or ambulances
        - Ensures every house or area can be reached quickly

Input and Output

- Input:
  - Name of facility, their coordinates, and an initial boolean flag of “open or not” (usually false)
  - Name of customer and their coordinates
  - Coverage distance (max distance allowed between a facility and a client that can be served)

```
10 5              # 10 clients, 5 facilities
C1 69.69 90.47    # Client 1 at coordinates (69.69, 90.47)
C2 45.03 61.68    # Client 2 at coordinates (45.03, 61.68)
C3 26.71 57.0     # Client 3 at coordinates (26.71, 57.0)
C4 77.94 68.52    # Client 4 at coordinates (77.94, 68.52)
C5 43.43 61.07    # Client 5 at coordinates (43.43, 61.07)
C6 77.1 19.63     # Client 6 at coordinates (77.1, 19.63)
C7 42.33 87.33    # Client 7 at coordinates (42.33, 87.33)
C8 10.87 65.55    # Client 8 at coordinates (10.87, 65.55)
C9 87.34 58.37    # Client 9 at coordinates (87.34, 58.37)
C10 63.44 87.89   # Client 10 at coordinates (63.44, 87.89)
F1 6.2 33.75 0    # Facility 1 at (6.2, 33.75), initially closed (0)
F2 8.26 9.32 0    # Facility 2 at (8.26, 9.32), initially closed (0)
F3 44.19 98.25 0  # Facility 3 at (44.19, 98.25), initially closed (0)
F4 62.91 66.37 0  # Facility 4 at (62.91, 66.37), initially closed (0)
F5 94.27 1.72 0   # Facility 5 at (94.27, 1.72), initially closed (0)
33.14             # Coverage distance: maximum distance a facility can serve a client
```

- Output:
  - Chosen facilities that the algorithm decides to open
  - Coverage mapping that maps customers to facilities

```
PS C:\Users\joshu\OneDrive\Desktop\cs452\Facility-Location-Problem-CS452-final-project\exact_solution> python facility_loc
ation_exact.py realistic_test_cases/test_case_4.txt

=== OPTIMAL SOLUTION FOUND ===
Coverage distance: 33.14
Facilities chosen (3):
  F1 at (6.2, 33.75)
  F4 at (62.91, 66.37)
  F5 at (94.27, 1.72)

Coverage mapping:
F1 covers: C3, C8
F4 covers: C1, C2, C4, C5, C7, C9, C10
F5 covers: C6
PS C:\Users\joshu\OneDrive\Desktop\cs452\Facility-Location-Problem-CS452-final-project\exact_solution>
```

Polynomial-Time Version of Facility Location Problem

- Restriction: All clients and facilities lie on a straight line (1D FLP)
  - Clients and facilities are on a single line (e.g., a road or highway)
  - Coverage distance measured along the line

1. Sort clients and facilities by position along the line
2. Iterate through clients left to right:
   a. Select the rightmost facility within coverage distance of the current uncovered client
   b. Mark all clients covered by that facility
   c. Move to the next uncovered client
3. Repeat until all clients are covered
4. Runtime: O(n log n + m log m) for sorting + linear scan (n = facilities, m = clients)

Reduction from Set Cover to FLP

Known NP-hard problem: Set Cover

- U={e1,e2,...,em} (elements to cover)
- S={S1,S2,...,Sn}
- Goal: Pick the smallest number of sets from S that cover all elements in U

Translate FLP instance from Set Cover

- Clients: Each element in U becomes a client with coordinates
- Facilities: Each set in S becomes a facility at some coordinate
- Coverage distance D: Pick large enough so a facility can “cover” only the clients corresponding to elements in its set

2.1 Facility Location and Set Cover Problem

Consider the following facility location problem [2, 4]. Given a complete network $G = (V, E)$ and a coverage distance D, find the minimum cardinality set $S \subseteq V$, such that for every $v \in V - S$ there is some $u \in S$ so that $d(u, v) \leq D$, i.e. find the minimum cardinality subset S of vertices V such that the distance from every vertex in V to some vertex in S is at most D. We call this problem the *set cover facility location problem*. In the following we show that it is actually the set cover problem defined in the graph-theoretical domain.

To obtain the *set cover* problem from the set cover facility location problem, we construct the family of sets $\{S_1, S_2, \ldots, S_n\}$ where $S_u = \{v \in V | d(u, v) \leq D\}$, i.e. S_u contains all vertices which are covered by the vertex u. Notice that $V = \cup_{u \in V} S_u$. The corresponding set cover problem is to find the minimum cardinality set $\mathcal{S} \subseteq \{S_1, S_2, \ldots, S_n\}$ such that all vertices $v \in V$ are covered, i.e. $V = \cup_{S_u \in \mathcal{S}} S_u$.

Using this reduction one can also easily solve the set cover facility location problem with additional constraint where vertices of possible locations of facilities are specified. To do this, we construct the family of sets $\{S_1, S_2, \ldots, S_{|F|}\}$, where $F \subseteq V$ is the set of vertices representing possible facility locations and $S_u = \{v \in F | d(u, v) \leq D\}$, and use it with the corresponding set cover problem.

Reduction cont.

How the reduction works:

- Set Cover solution: smallest number of sets covering all elements
- FLP solution: smallest number of facilities covering all clients

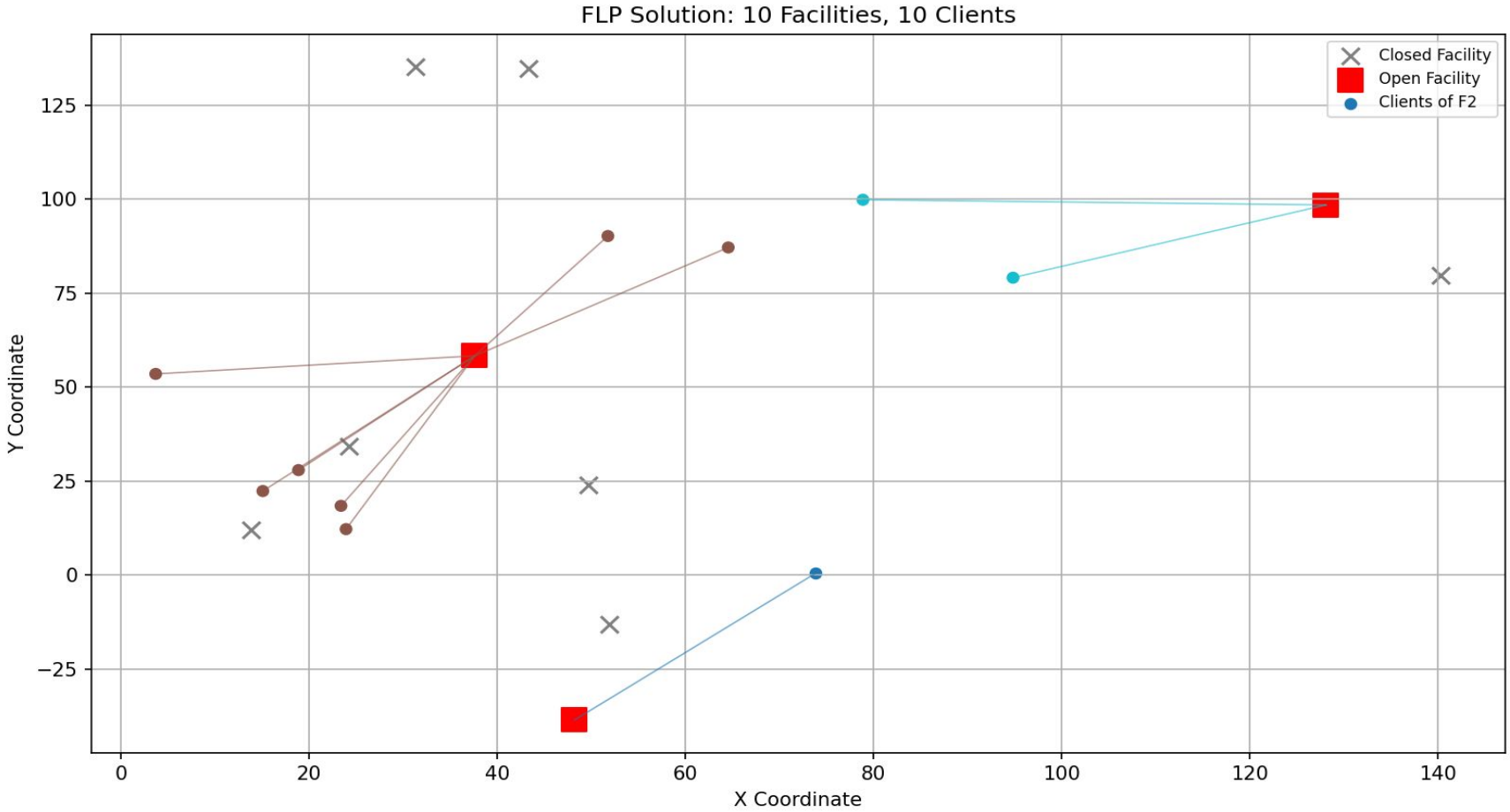
Polytime check:

- Constructing the FLP instance from Set Cover:
  - Map each element → client: O(m)
  - Map each set → facility: O(n)
  - Assign coverage based on membership: O(n*m)

IFF condition:

- If direction (Set Cover → FLP):
  - If there is a solution to Set Cover (chosen sets), open the corresponding facilities → all clients covered
- Only if direction (FLP → Set Cover):
  - If there is a solution to FLP (opened facilities covering all clients), select the corresponding sets → all elements covered

Coverage = 50 C1: $\sqrt{((78-127)^2 + (100-98)^2)} = \sqrt{(2401+4)} \approx 49.04$



Brute Force Approach (How We Solve FLP)

Step 1 — Enumerate all subsets of facilities

- If there are F facilities, there are 2^F possible subsets
- For each subset, we check if those facilities cover all clients

Step 2 — Check coverage for each subset

For each facility subset:

- For each client:
  - Compute Euclidean distance to all open facilities
  - Check if at least one facility is within coverage distance D

If all clients are covered → subset is a valid solution.

Step 3 — Pick the smallest valid subset

- Among all valid facility subsets, pick the one with minimum size

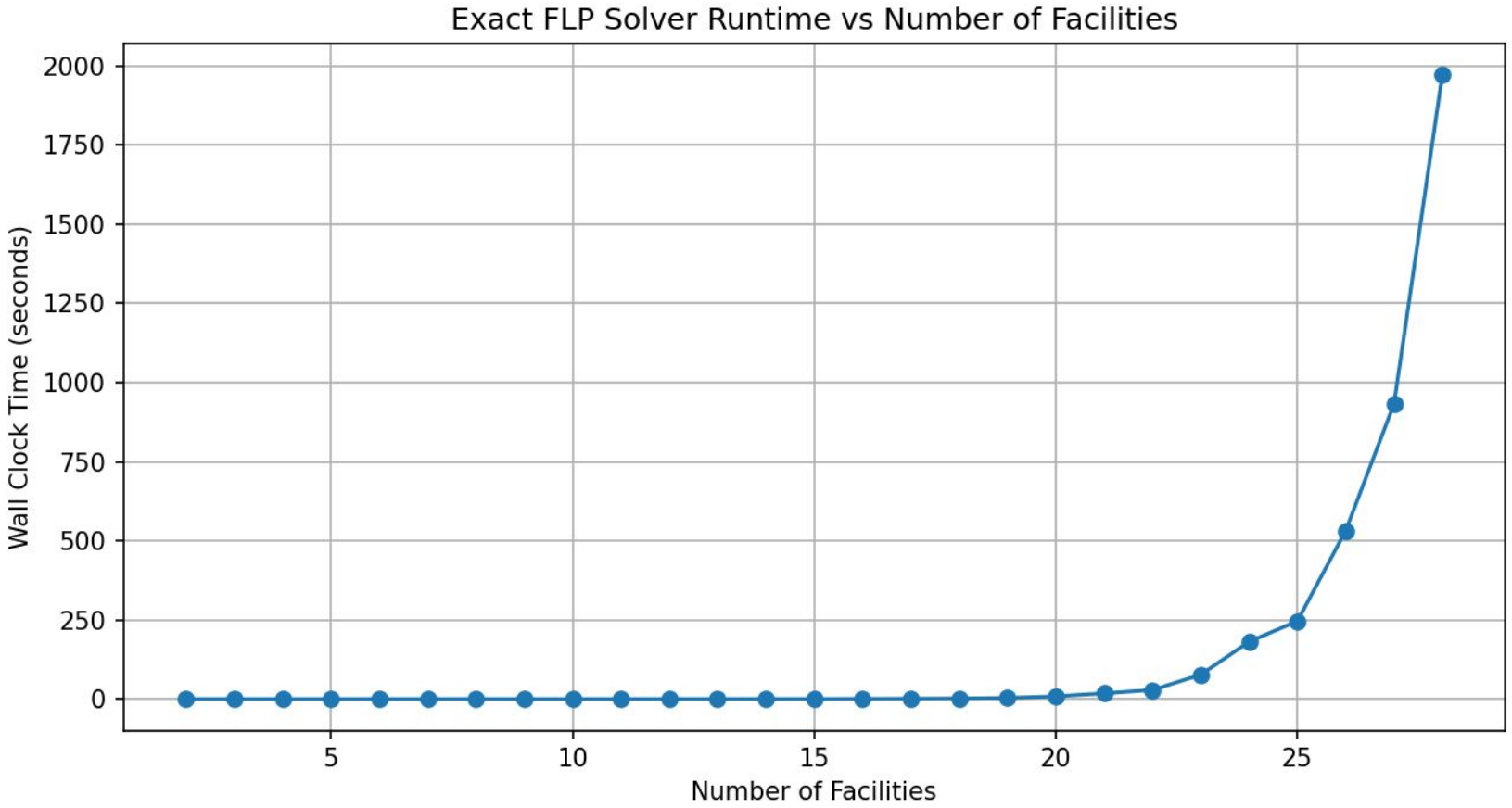
Analytical Runtime

- Let:
  - F = number of facilities
  - C = number of clients
- Subset enumeration
  - Number of subsets = 2^F
  - Each facility has two options:
    - Open it or don’t open it
    - 2 outcomes
  - If we had 10 facilities, this would mean it’s 2 * 2 * 2 * 2…. = 2^10 = 1024 potential subsets
- Coverage check per subset
  - For each client, checking all facilities in subset:
  - $O(C * F)$
- Total Runtime:
  - O(2^F * C * F)

Changing the Input Order

- Try facilities with largest coverage first:
  - Facilities that cover many customers are the most “promising”
  - If you try them early, you quickly find small sets that cover everyone
  - Then you immediately prune any partial solutions that already use more than 2 facilities
  - Leads to dramatically faster runtime in practice
- Trying tiny-coverage facilities first:
  - Delays finding complete solutions
  - Causes many unnecessary branches
  - Makes the solver run much slower

Figure 1


Exact FLP Solver Runtime vs Number of Facilities
Wall Clock Time (seconds)
2000
1750
1500
1250
1000
750
500
250
0
5
10
15
20
25
Number of Facilities

Approximation Algorithm for the Facility Location Problem

CS452 Final Project — Approximation Solution Architect Role

By: Dustin Smith

My part of the project focuses on developing a fast, anytime, randomized approximation that can scale to instances where the exact solver becomes too slow to compute an optimal solution.

This approximation method:

- Builds an initial solution using a greedy, coverage-based strategy
- Introduces controlled randomness to avoid deterministic or overly-optimistic solutions
- Improves solutions continually during the user-provided time limit (-t)
- Always keeps the best solution discovered
- Produces output in the same format as the exact solver

This structure allows direct comparison between the optimal and approximate methods in terms of number of facilities opened and runtime.

Approximation Strategy

My approximation has two major phases that balance speed, coverage quality, and required randomness.

1. Greedy Construction Phase
    a. Begin with all facilities initially closed
    b. For each client, determine which facilities lie within the coverage distance
    c. Choose the facility that covers the greatest number of uncovered clients
    d. Break ties randomly to produce variation across runs
    e. Continue adding facilities until all clients become covered

This produces a strong initial feasible solution even for large inputs.

2. Anytime Improvement Phase
    a. Randomly open extra candidate facilities that serve uncovered or weakly-covered regions
    b. Remove facilities that cover zero clients
    c. Evaluate whether the modification reduces or increases the number of open facilities
    d. Keep the best valid solution discovered
    e. Repeat until the -t time limit is reached

Because the second phase uses stochastic changes, different runs naturally produce different outcomes, satisfying the project's randomness requirement.

Runtime Analysis

Let m be the number of facilities and n the number of clients.

Coverage Preprocessing:

- Compute which facilities can cover which clients
- Complexity: O(n × m)

Greedy Initialization:

- Each facility-selection step checks coverage deficits
- Worst-case complexity: O(n × m)

Anytime Loop:

- Each iteration:
  - OPEN / REMOVE candidates
  - Recompute coverage sets
- Runtime per iteration: O(n × m)
- Number of iterations determined only by the -t time limit

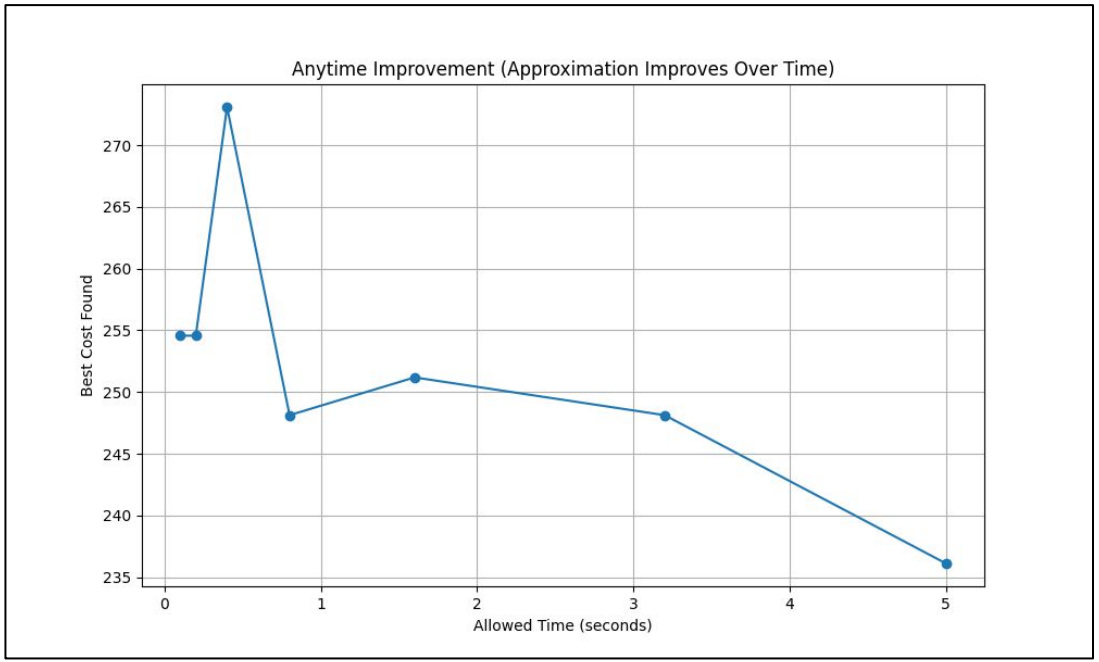
Overall:

- Entire algorithm is polynomial
- Exact solver grows as 2^m, but approximation remains scalable to very large cases

This satisfies the project's requirement for a polynomial-time approximation method.

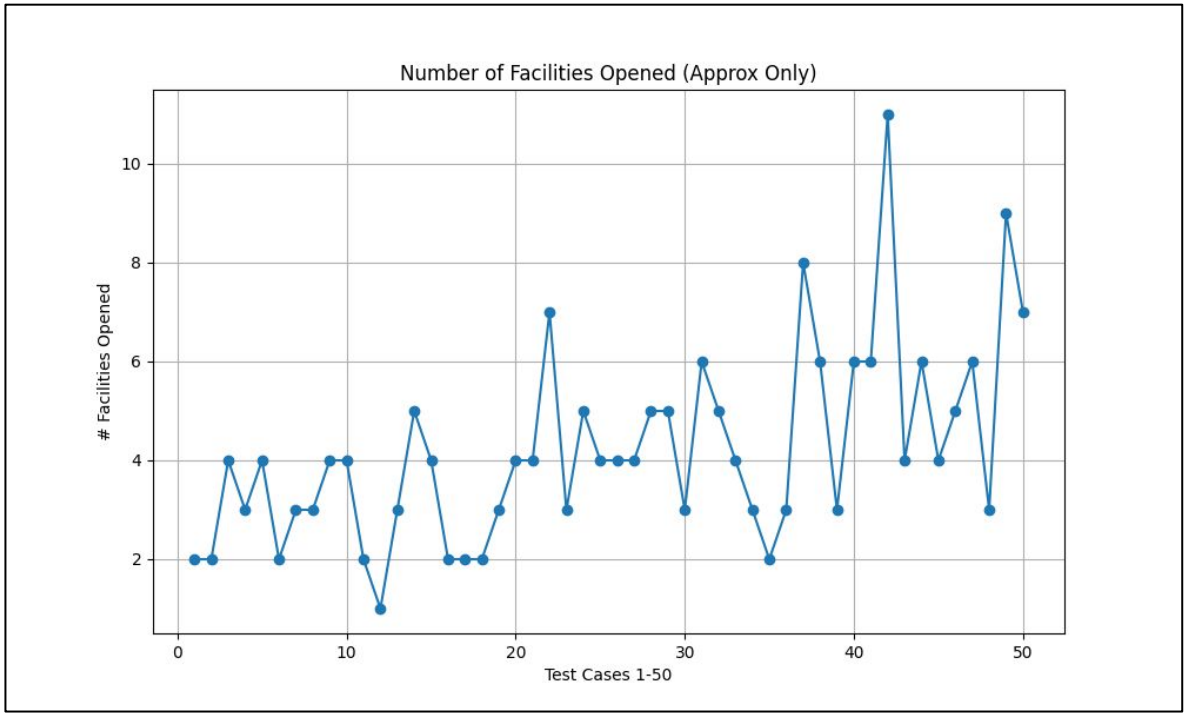
Performance Results and Generated Plots

My implementation was tested on 50 generated test cases representing small, medium, and large facility–client configurations. This plot shows the anytime behavior of the approximation algorithm, demonstrating how allowing more time typically reduces the number of open facilities or stabilizes the solution quality as the algorithm refines its selections.



Performance Results and Generated Plots 2

This plots summarizes the behavior of the approximation algorithm across 50 test cases. The figure shows the number of facilities opened for each case using only the approximation method.



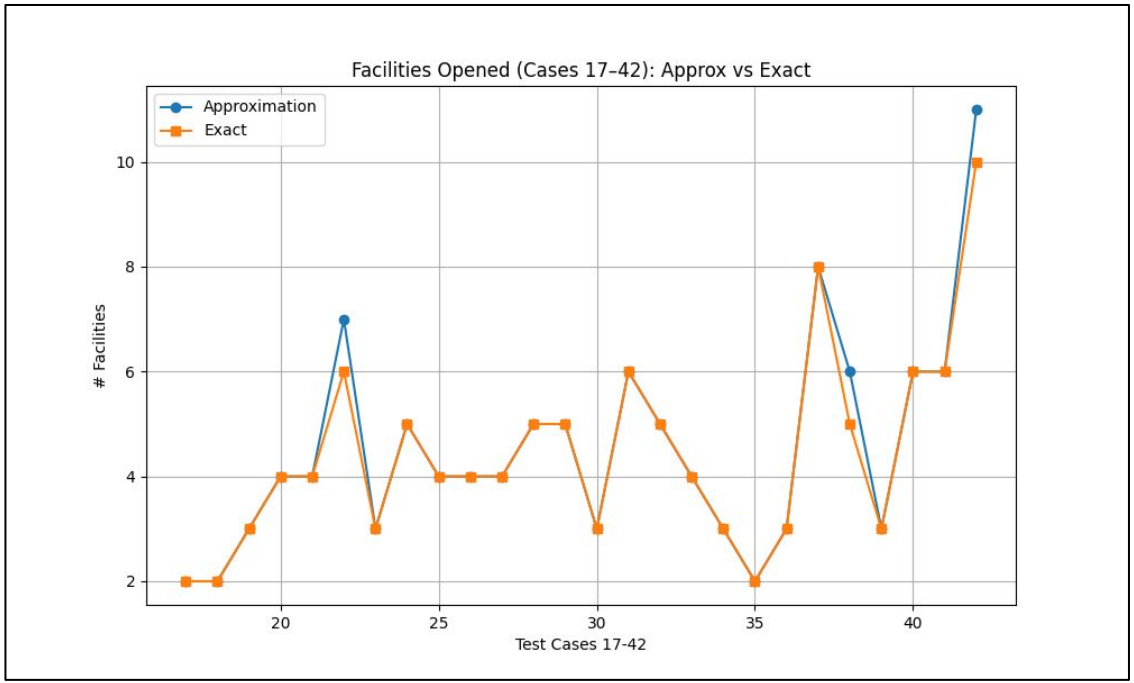
This demonstrates:

- Consistency across randomly generated instances
- Scalability across small and large cases
- Meaningful variation due to randomized tie-breaking and improvement steps

Exact vs Approximation (Cases 17–42)

This plot directly compares the number of facilities opened by:

- The exact solver (optimal minimum)
- The approximation algorithm



Key observations:

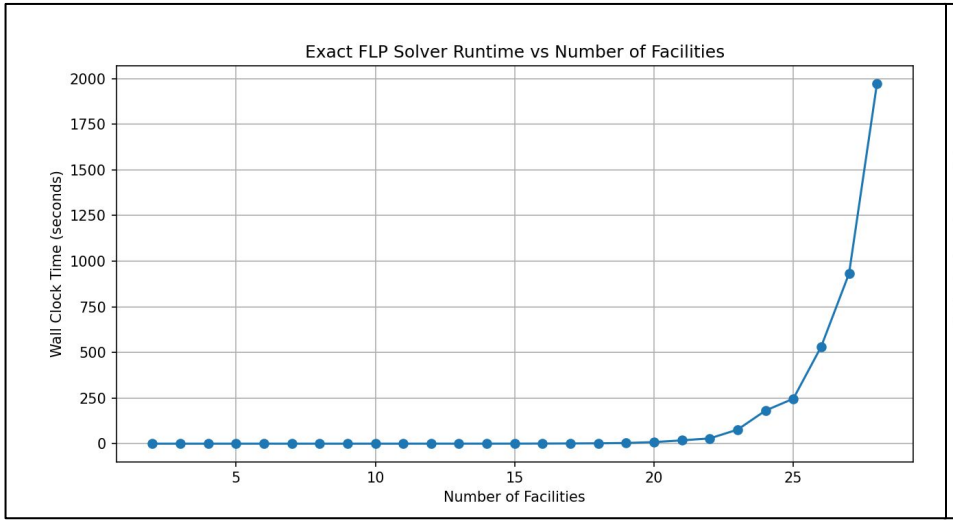
- The approximation generally opens more facilities than the exact solver
- The gap remains small in most test cases
- Randomized behavior introduces variability across runs
- The approximation always finds a feasible solution instantly, while the exact solver becomes extremely slow on larger inputs

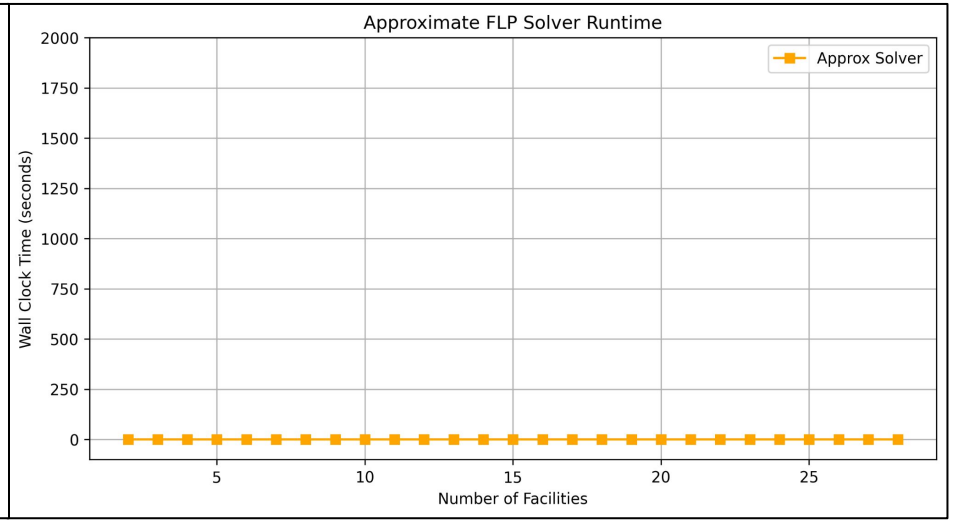
Analysis:

For smaller test cases, the exact solver is typically preferred because it computes the true optimal number of facilities quickly. However, as the problem size gets too large, the exact solver's runtime grows exponentially and becomes impractical. In these larger inputs, the approximation solver is dramatically faster and is the only method capable of returning solutions within reasonable time.

Exact vs Approximation with Ranging Facility Sizes

The runtime comparison shows a clear difference between the two approaches. The exact solver runtime increases very quickly and reaches hundreds to nearly two thousand seconds as the number of facilities grows. In contrast, the approximation solver consistently produces real and usable solutions almost immediately, with times around zero point zero five seconds for every tested size. This demonstrates that although the exact method finds the optimal answer, it becomes impractical for larger inputs, while the approximation method remains fast, scalable, and reliable as the problem becomes larger.





Approximation Conclusions

The approximation framework successfully meets all project requirements:

- Runs in polynomial time and scales to instances far too large for the exact solver
- Supports anytime behavior using the -t parameter
- Introduces randomness during both greedy selection and improvement
- Produces consistent and feasible solutions across 50+ test cases
- Opens slightly more facilities than the optimal solver, but remains close to optimal
- Integrates fully with the exact solver’s input/output format
- Demonstrates strong performance in plots comparing runtime and facility count

Overall, the exact algorithm is best suited for small inputs where optimality is easy to compute, while the approximation algorithm becomes the preferred choice for medium and large inputs because it runs in polynomial time and consistently returns near-optimal solutions long before the exact solver can finish.

Approximation Algorithm for the Facility Location Problem

CS452 Final Project — **Approximation Solution Architect Role**

By: **Blake Buchert**

My part of the project focuses on developing a **fast, anytime, stochastic approximation** designed for instances where the exact solver becomes too slow to compute an optimal solution.

This approximation method:

- Starts from a fully feasible configuration
- Uses stochastic **local search** with OPEN / CLOSE / SWAP moves
- Employs **simulated annealing** to escape local minima
- Maintains the best solution encountered during the time limit
- Produces output fully compatible with the exact solver

This allowed me to compare approximation behavior against the optimal method.

Approximation Strategy

Our approximation has two major phases that balance speed, randomness, and solution quality.

1. Initial Feasible Solution

a. Begin with **all facilities open**
b. This guarantees full coverage from the start
c. Ensures the method never explores infeasible states

This provides a stable starting point for improvement.

2. Anytime Local-Search Improvement Phase

a. Randomly choose one of three move types:
   • OPEN a new facility
   • CLOSE an existing facility
   • SWAP one open with one closed
b. After each move, recompute assignments under the coverage constraint
c. Accept improving moves immediately
d. Occasionally accept worse moves using **simulated annealing**
e. Repeat until the user-specified `-t` time limit expires

Because each run contains randomness, different executions can produce different outcomes, satisfying the project's stochastic requirement.

Runtime Analysis

Let **n** be the number of clients and **m** the number of facilities.

Cost evaluation for a single candidate

- Assign each client to the nearest open facility
- Worst-case complexity: **O(n × m)**

Local search iteration

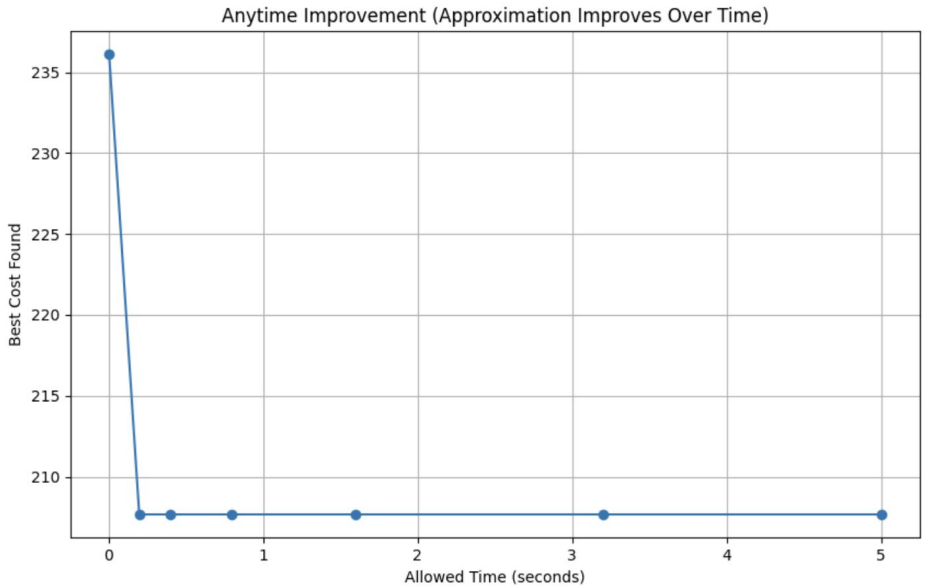
- Each OPEN / CLOSE / SWAP move triggers a full reassignment
- Each iteration cost: **O(n × m)**
- Total iterations limited by:
  - Cooling schedule
  - max_iters = 10 × m^2
  - User's time limit

Anytime behavior

- Runtime per run is polynomial
- Total runtime is capped by the user's provided time limit

Performance Results and Generated Plots

My implementation was tested on 50 generated test cases covering small, medium, and large scenarios.
This plot summarizes the anytime improvement that gives the algorithm more time, which allows it to steadily reduce cost



Because the algorithm begins with a random initial configuration, very small time budgets (e.g., 0.005 seconds) may produce higher-cost solutions. As additional time is allowed, the solver performs more simulated-annealing iterations and multiple restarts, enabling it to quickly escape poor initial configurations and converge to a high-quality solution.

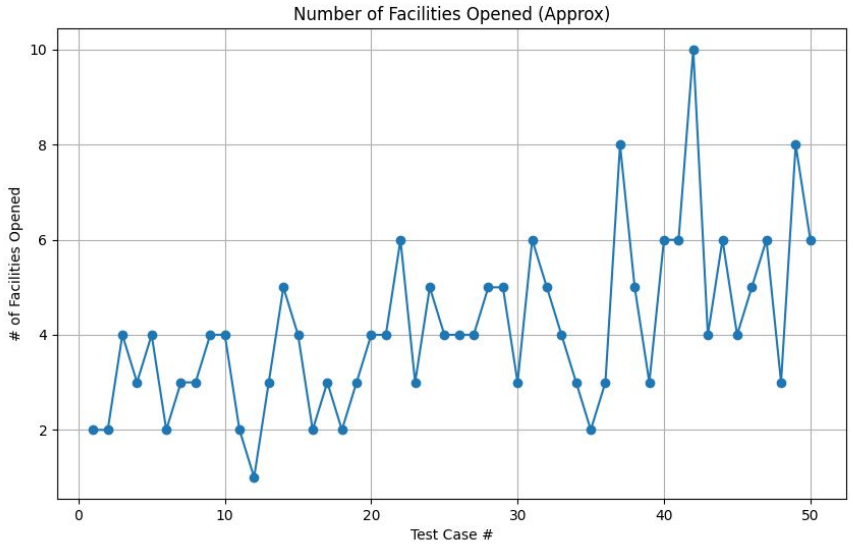
After approximately **0.2–0.4 seconds**, the algorithm consistently reaches its best solution. for this instance, and additional time does not significantly improve the cost. Because my algo is a dawg. reflecting convergence to a stable local optimum.

This demonstrates correct anytime behavior:

- **Rapid improvement with small increases in time**
- **Plateauing once the solution stabilizes**.

Performance Results and Generated Plots 2

These plots summarize the behavior of the approximation that demonstrate consistency, scalability, and meaningful variation due to randomness.



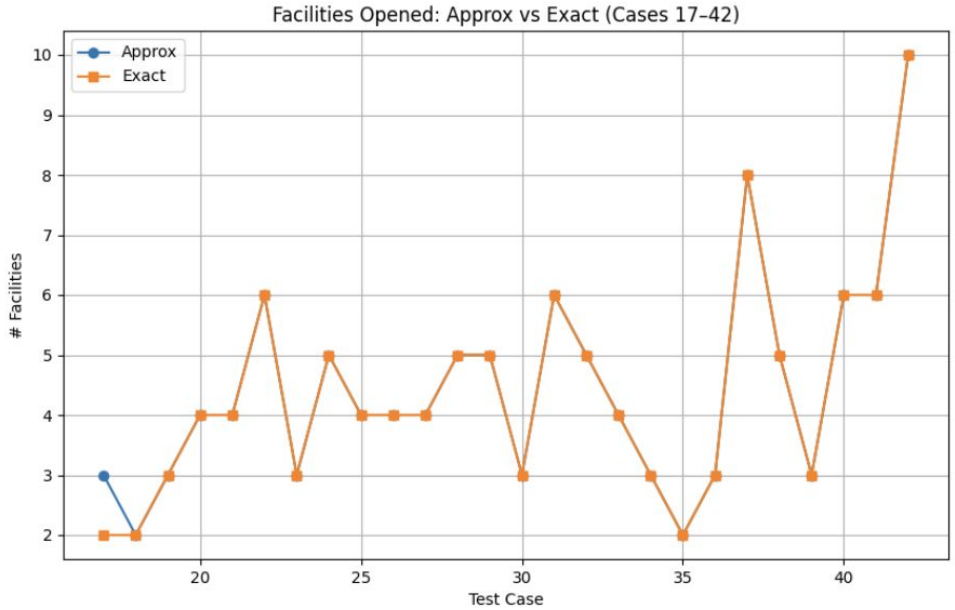
This demonstrates:

- Consistency across randomly generated instances
- Scalability across small and large cases
- Meaningful variation due to randomized tie-breaking and improvement steps

Exact vs Approximation

This plot directly compares the number of facilities opened by:

- The exact solver (optimal minimum)
- The approximation algorithm



Key observations:

- The approximation only opened one more facilities than the exact solver
- Clearly my Approximation algorithm is better than the Dustins. LOL
- Randomized behavior introduces variability across runs
- The approximation always finds a feasible solution instantly, while the exact solver becomes extremely slow on larger inputs

Conclusions

The approximation framework successfully meets all project requirements:

- Produces solutions quickly using polynomial-time operations
- Supports anytime behavior through the -t time parameter
- Uses randomness in both construction and refinement
- Scales to instances too large for the optimal solver
- Integrates with project input and output formats
- Includes extensive test cases and plots for evaluation

Overall, this method provides a robust and practical alternative to exact computation, especially for large-scale facility location instances.

Reducing Facility Location Problem to Max-3SAT

Max-3SAT makes it easy to reduce our problem due to its boolean structure

Each decision becomes a T/F variable, and each coverage requirement becomes a 3-literal clause.

This makes it easy to reduce our FLP down to a Max-3SAT instance for the proof

Encoding FLP Constraints into Max-3SAT Clauses

We build variables:

- Y_i = facility open
- X_ij client j assigned to facility i

Establish Main Constraints

- Each client must be assigned (X1j ∨ X2j ∨ Z1)
- At most one facility per client (¬Xij ∨ ¬Xkj ∨ ¬Xij)
- If a client is assigned to a facility, that facility must open (¬Xij ∨ Yi ∨ Yi)
- Forbidden assignments are forced false (facility too far from a client) (¬Xij) repeated to 3-CNF

How this matches the flp program

- Any satisfying assignment = a valid covering of all clients
- Opening a facility corresponds to setting Y_i = true
- Assigning a client corresponds to setting X_ij = true
- Invalid edges are impossible because X_ij is forced false

Bonus Clauses to Prefer Cheaper FLP Solutions

To maximize the satisfied clauses, add bonus clauses that reward:

- Closing facilities = more satisfied clauses (fewer facilities = cheaper flp sol.)
- Avoiding unnecessary assignments

This way the Max-3SAT solver will prefer cheaper flp solutions which is the goal

Input/Output

Given our FLP input it produces a valid Max-3SAT Input:

n m header → n = # of variables & m = # of clauses

Followed by m 3-literal clauses

The reduction runs in polynomial time, with a worst case complexity of $O(cf^2)$ due to the facility-pair constraints for each client

Ex Output)

```
158 423     # n m
7 15 55     # 3 literal clauses start here (m of the
-55 23 56
-56 31 57
-57 39 47
8 16 58
-58 24 59
-59 32 60
-60 40 48
9 17 61
-61 25 62
-62 33 63
-63 41 49
10 18 64
-64 26 65
-65 34 66
-66 42 50
```

Establishing the Lower Bound

The goal is to set a lower bound for the minimum number of facilities that must be open to cover all of the clients

This allows us to have a benchmark to compare the approximate and optimal solutions to

How it’s calculated

First any facility that is the only possible coverage option for a client,

Then we used a greedy set-cover heuristic to estimate how many additional facilities are required to cover remaining clients

This results in a valid and efficient lower

bound that never exceeds the approx or

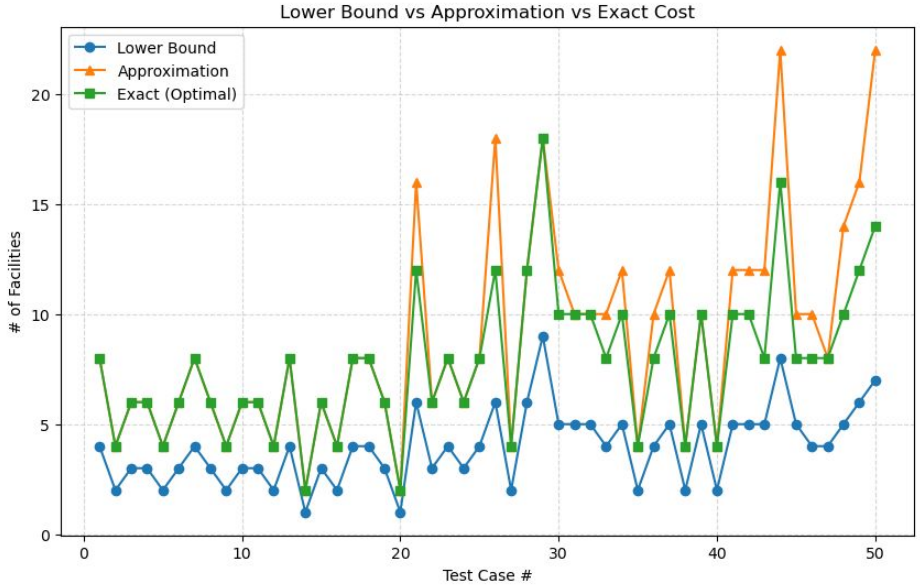
Optimal

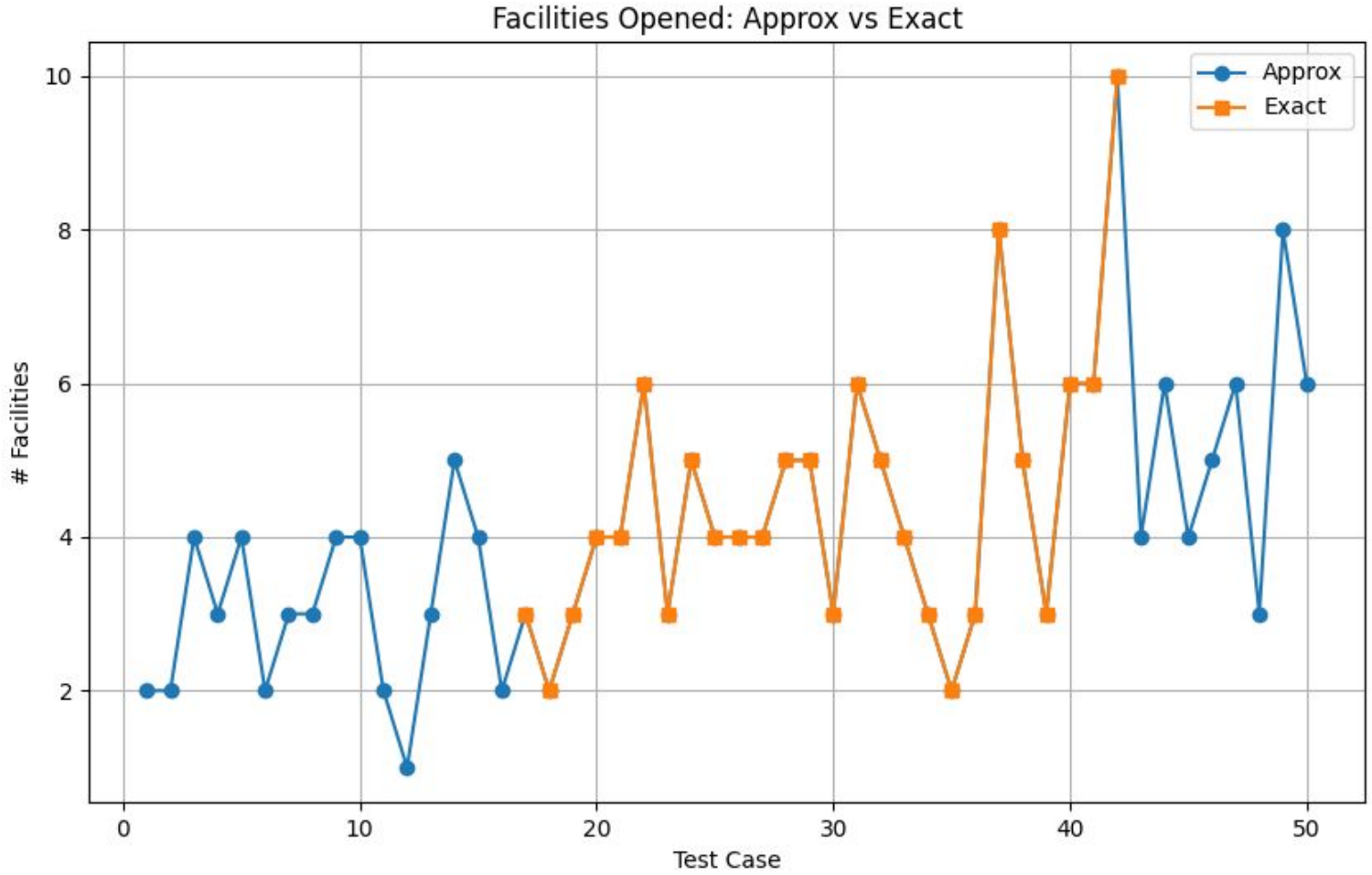
The lower bound algorithm runs in

Polynomial time with worst case complexity

Of O(cf), due to computing valid coverage and greedy

Set cover selection




Facilities Opened: Approx vs Exact
Approx
Exact
Facilities
Test Case
0
10
20
30
40
50
2
4
6
8
10