AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

"Design and Analysis of algorithms – Activity – Case Study"

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

In

COMPUTER ENGINEERING

Submitted By

Name of the Student: Piyusha Rajendra Supe

Roll No: 23CO315

Under the Guidance of

Prof. N. R. Talhar

ALL INDIA SHRI SHIVAJI MEMORIAL SOCIETY'S COLLEGE OF ENGINEERING PUNE-411001

Academic Year: 2025-26 (Term-I)

Savitribai Phule Pune University

1) Problem data & assumptions

Cost matrix:

	F1	F2	F3	F4	F5
C1	5	∞	6	∞	4
C2	∞	3	∞	7	5
C3	4	6	5	∞	∞

Interpretation / options per flight:

- F1: C1(5) or C3(4)
- F2: C2(3) or C3(6)
- F3: C1(6) or C3(5)
- F4: C2(7) only
- F5: C1(4) or C2(5)

Objective: assign **one** crew to each flight (crew may serve multiple flights), minimize total cost, respect "∞" (not allowed).

2) Backtracking (exhaustive) — demonstration & full enumeration

Backtracking here means: try all feasible assignments (prune only when a flight has no allowed crew). Because every flight has at least one allowed crew (F4 has only C2), there are no infeasible-by-qualification dead ends; plain backtracking will enumerate all combinations.

Possible choices per flight multiply to:
2 (F1) × 2 (F2) × 2 (F3) × 1 (F4) × 2 (F5) = **16** feasible complete assignments.

I list the 16 assignments (sorted by total cost) so you can see how backtracking would examine them and then pick the minimum.

1. F1=C3, F2=C2, F3=C3, F4=C2, F5=C1 → cost = 4 + 3 + 5 + 7 + 4 = 23 **(minimum)**
2. F1=C1, F2=C2, F3=C3, F4=C2, F5=C1 → cost = 5 + 3 + 5 + 7 + 4 = 24
3. F1=C3, F2=C2, F3=C1, F4=C2, F5=C1 → cost = 4 + 3 + 6 + 7 + 4 = 24
4. F1=C3, F2=C2, F3=C3, F4=C2, F5=C2 → cost = 4 + 3 + 5 + 7 + 5 = 24
5. F1=C1, F2=C2, F3=C1, F4=C2, F5=C1 → cost = 5 + 3 + 6 + 7 + 4 = 25
6. F1=C1, F2=C2, F3=C3, F4=C2, F5=C2 → cost = 5 + 3 + 5 + 7 + 5 = 25
7. F1=C3, F2=C2, F3=C1, F4=C2, F5=C2 → cost = 4 + 3 + 6 + 7 + 5 = 25
8. F1=C1, F2=C2, F3=C1, F4=C2, F5=C2 → cost = 5 + 3 + 6 + 7 + 5 = 26
9. F1=C3, F2=C3, F3=C3, F4=C2, F5=C1 → cost = 4 + 6 + 5 + 7 + 4 = 26
10. F1=C1, F2=C3, F3=C3, F4=C2, F5=C1 → cost = 5 + 6 + 5 + 7 + 4 = 27
11. F1=C3, F2=C3, F3=C1, F4=C2, F5=C1 → cost = 4 + 6 + 6 + 7 + 4 = 27
12. F1=C1, F2=C3, F3=C3, F4=C2, F5=C2 → cost = 5 + 6 + 5 + 7 + 5 = 28
13. F1=C3, F2=C3, F3=C3, F4=C2, F5=C2 → cost = 4 + 6 + 5 + 7 + 5 = 27
14. F1=C3, F2=C3, F3=C1, F4=C2, F5=C2 → cost = 4 + 6 + 6 + 7 + 5 = 28
15. F1=C1, F2=C3, F3=C1, F4=C2, F5=C1 → cost = 5 + 6 + 6 + 7 + 4 = 28
16. F1=C1, F2=C3, F3=C1, F4=C2, F5=C2 → cost = 5 + 6 + 6 + 7 + 5 = 29

(you can verify each sum; e.g. best one is 4+3+5+7+4 = 23: 4+3=7, 7+5=12, 12+7=19, 19+4=23).

Backtracking conclusion: enumerating all 16 candidates shows the unique minimum-cost assignment is:

- F1 → C3 (cost 4)
- F2 → C2 (cost 3)
- F3 → C3 (cost 5)
- F4 → C2 (cost 7)
- F5 → C1 (cost 4)

Total cost = **23**.

(Backtracking will find this after exploring combinations; it guarantees correctness but explores every feasible leaf in this small example.)

3) Branch-and-Bound (LC = Least-Cost expansion) — demonstrate pruning and lower bounds

Bounding function used (simple and admissible):

At any partial assignment:

- cost_so_far = sum of costs assigned so far.
- For each *unassigned* flight, add the minimum possible cost among allowed crew for that flight (a cheap admissible estimate).
  Lower bound (LB) = cost_so_far + sum(min-cost for each unassigned flight).

Because the min-cost per flight are:

- min(F1)=4, min(F2)=3, min(F3)=5, min(F4)=7, min(F5)=4

Root LB = 4+3+5+7+4 = **23**. That already equals the value we eventually find — that tells us the theoretical minimum cannot be less than 23.

Now LC strategy expands the node with the smallest LB first. I show the key nodes / bounds and pruning decisions (we expand only nodes with the lowest LB first).

Search (textual partial solution tree):

1. Root: (no assignment) LB = 23. Expand (split by choices for F1).
   - Node A: F1 = C3 → cost_so_far = 4, LB = 4 + (min of F2..F5) = 4 + (3+5+7+4) = **23**
   - Node B: F1 = C1 → cost_so_far = 5, LB = 5 + (3+5+7+4) = **24**

   LC picks Node A (LB=23) to expand next (smallest LB)

2. Expand Node A (F1=C3): branch on F2
   - A1: F2 = C2 → cost_so_far = 4+3 = 7. LB = 7 + (min of F3..F5 = 5+7+4) = **23**
   - A2: F2 = C3 → cost_so_far = 4+6 = 10. LB = 10 + (5+7+4) = **26** → LC will not expand this while cheaper LB nodes exist; once best = 23 found, A2 will be pruned since LB>best.

   LC expands A1 next (LB=23).

3. Expand A1 (F1=C3, F2=C2): branch on F3
   - A1a: F3 = C3 → cost_so_far = 4+3+5 = 12. LB = 12 + (min of F4,F5 = 7+4) = **23**
   - A1b: F3 = C1 → cost_so_far = 4+3+6 = 13. LB = 13 + (7+4) = **24**

   LC expands A1a (LB=23).

4. Expand A1a (F1=C3, F2=C2, F3=C3): F4 only choice C2
   - A1a1: F4 = C2 → cost_so_far = 12 + 7 = 19. LB = 19 + (min for F5 = 4) = **23**

   Expand A1a1: branch on F5

   - A1a1i: F5 = C1 → total cost = 19 + 4 = **23** → **complete solution found**. Current best (upper bound) = 23.
   - A1a1ii: F5 = C2 → total cost = 19 + 5 = 24 → (worse than best)
5. **Pruning after best = 23**:
   - Any open node with LB ≥ 23 can be pruned (we are searching for strictly better than current best). In particular:
     - Node B (F1=C1) LB = 24 → prune.
     - Node A2 (F1=C3,F2=C3) LB = 26 → prune.
     - Node A1b (F1=C3,F2=C2,F3=C1) LB = 24 → prune.
   - No further expansions needed; we already have an optimal solution of cost equal to the root LB, so optimality is proven quickly.

What this shows: Branch-and-Bound with LC found the optimal solution while exploring only a **tiny fraction** of the 16 leaves (it visited the promising nodes until it hit the solution) and pruned other branches using LB computations. In this example the root LB equalled the final optimum (23), so the algorithm converges quickly.

4) Final answer (explicit)

Minimum cost schedule (optimal):

- F1 → **C3** (cost 4)
- F2 → **C2** (cost 3)
- F3 → **C3** (cost 5)
- F4 → **C2** (cost 7)
- F5 → **C1** (cost 4)

Total = 4 + 3 + 5 + 7 + 4 = **23**.

5) Comparative analysis (Backtracking vs Branch-and-Bound)

- **Backtracking**
  - Guarantees correctness by enumerating all feasible assignments (complete search).
  - Simple to implement; good for small problems.
  - **Cost:** explores all feasible leaves in worst case → combinatorial explosion as flights and crew grow (exponential). In our example it explored 16 leaves.
- **Branch-and-Bound**
  - Uses bounds (lower bounds) to *prune* subtrees that cannot contain a better solution than the best found so far.
  - LC strategy expands the most promising (lowest LB) node first; often finds good solutions early and prunes much of the tree.
  - **Cost:** still worst-case exponential, but in practice far fewer nodes are explored; scales much better than plain backtracking on typical scheduling instances.

Why Branch-and-Bound is better suited for large-scale airline scheduling

- Real airline scheduling includes many additional constraints (max work hours, required rest, consecutive pairings, crew qualifications, pairing continuity) and objective terms (overtime, hotel costs). These make the feasible space huge.
- B&B can incorporate strong lower bounds (LP relaxations, assignment relaxations, or problem-specific heuristics) which enable powerful pruning.
- Commercial solvers (ILP/MIP solvers) use branch-and-bound (plus cutting planes, pre-solve, heuristics) and are the standard for realistic scheduling.
- For very large instances, B&B is combined with decomposition, column generation (crew pairing problems), or metaheuristics to produce solutions in acceptable time.