Chapter 3

Automated Patrol Planning using Graph Plan and POP

This report presents a comprehensive analysis of two classical automated planning algorithms Partial Order Planning (POP) and GraphPlan—applied to the wildlife poaching prevention problem in a 500 km^2 wildlife reserve in central India. Both algorithms were successfully executed to generate optimal patrol deployment strategies under resource constraints. The report details the theoretical foundations, implementation details, operational outputs, and comparative analysis of these planning methodologies.

3.1 Introduction

3.1.1 Problem Context

The wildlife reserve faces three critical poaching alerts during peak dry season:

- Gunshot-like audio detected near a riverbed.
- Repeated thermal movement near an elephant corridor past midnight
- Anonymous intelligence indicating planned poaching in tiger habitat.

With only two ranger teams and one drone available, the command center requires intelligent reasoning to prioritize deployment and generate explainable operational decisions.

3.1.2 Planning Problem Formulation

State Space:

Configurations of alert statuses, risk assessments, resource availability, and patrol completion.

Goal State:

All alerts analyzed, all high-risk zones patrolled or under surveillance, all alerts cleared.

Operators:

Intelligence analysis, resource assignment, patrol dispatch, and surveillance activation.

Constraints:

Limited ranger teams and drone availability; causal dependencies between actions.

3.2.Partial Order Planning (POP)

3.2.1 Theoretical Foundation

Partial Order Planning is a classical AI planning algorithm that maintains a partial order of actions rather than committing to a total order upfront. This "least-commitment" strategy allows flexibility in execution while maintaining causal dependencies between actions

3.2.2 Core Principles

1. **Least Commitment:** Actions are only ordered when logically necessary
2. **Causal Links:** Explicit connections between action effects and preconditions
3. **Threat Resolution:** Handles conflicts through promotion or demotion
4. **Open Preconditions:** Iteratively resolved until all goals are achieved

3.2.3 How POP Works

Step 1: Initialize Minimal Plan

- **Start action :** Establishes initial state with all alert flags active and resources available.
- **Finish action:** Requires goal state (all zones patrolled, all alerts cleared)
- **Open pre conditions:** All goals added to the "to-be-resolved" list

Step 2: Select and Resolve Subgoals

- Pick an unresolved precondition
- Choose an operator that produces this precondition
- Establish a causal link: Operator → Precondition → Consumer Action

Step 3: Add Ordering Constraints

- The chosen operator must execute before the action requiring its effects.
- All operators establishing preconditions for a new action must execute before it.

Step 4: Threat Detection and Resolution

- Identify actions that could delete (negate) a causal link's condition
- Threat resolving strategies:
    - **Promotion:** Insert threat before the link's producer
    - **Demotion:** Insert threat after the link's consumer
    - Choose strategy that maintains ordering consistency (no cycles)

Step 5: Iterate Until Complete

- Repeat steps 2-4 until all preconditions are resolved.
- Return the partial order plan with causal dependencies

3.2.4 POP Implementation for Wildlife Poaching Prevention

Initial State:

- All rangers and drones at base with resources available.
- Three active alerts for gunshot, thermal, and intel.
- High poaching risk in all three zones.

```
initial _state = {
      'At(Ranger1, Base)', 'At(Ranger2, Base)', 'At(Drone, Base)',
      'ResourceAvailable(Ranger1)','ResourceAvailable(Ranger2)',
      'ResourceAvailable(Drone)','AlertActive(Gunshot)',
      'AlertActive(Thermal)','AlertActive(Intel)',
      'PoachingRisk(Riverbed, High)','PoachingRisk(ElephantCorridor,High)' ,
      'PoachingRisk(TigerHabitat, High)'
 }
```

Final State:

- All zones patrolled or under surveillance.
- All alerts cleared.
- Poaching risks are reduced.

```
goal_state = {
     'Patrolled(Riverbed)', 'Patrolled(ElephantCorridor)',
     'SurveillanceActive(TigerHabitat)',
     'NOT(AlertActive(Gunshot))', 'NOT(AlertActive(Thermal))',
     'NOT(AlertActive(Intel))'
}
```

Operators Defined:

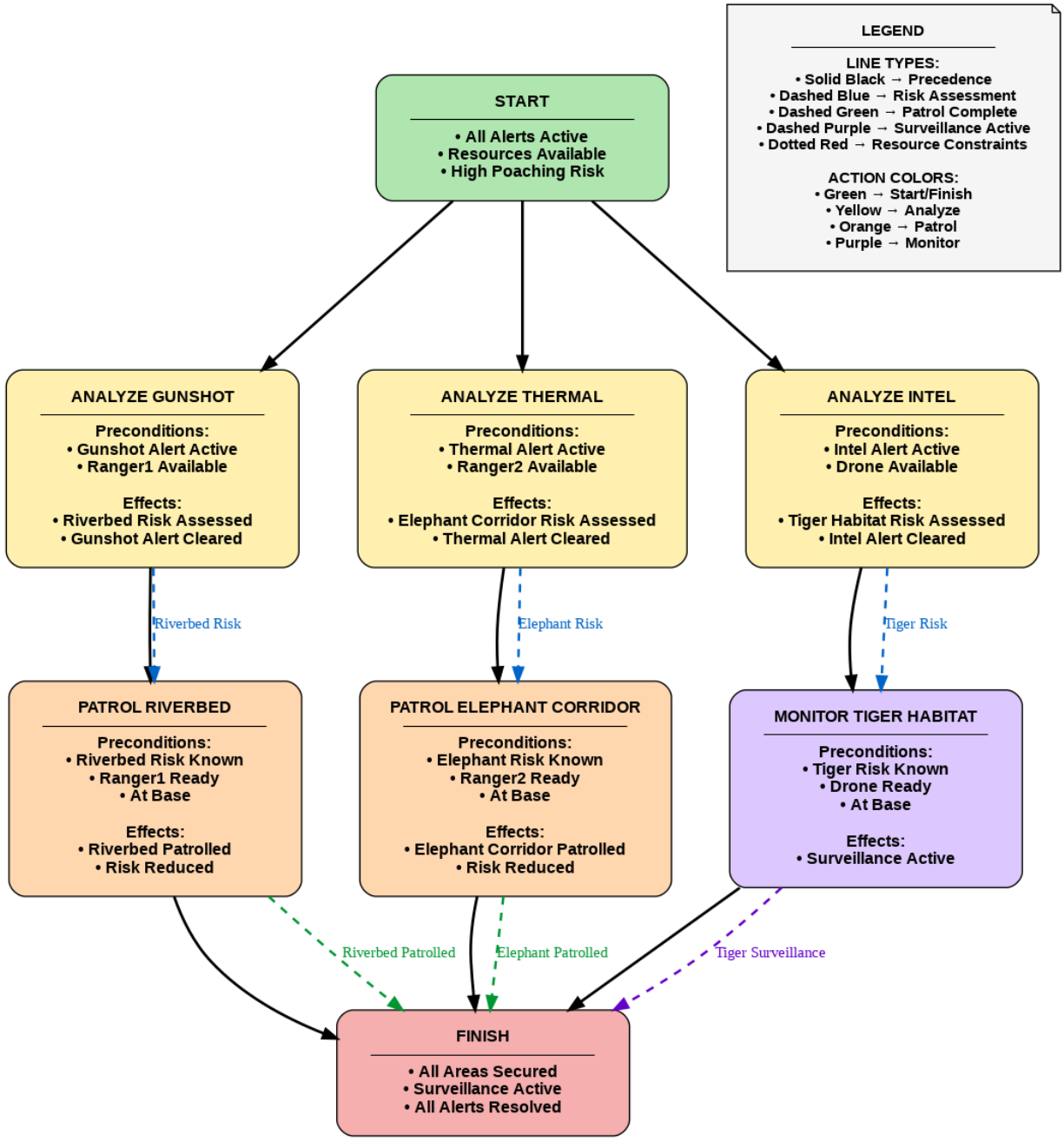
```
  'Analyze_Gunshot': {
            'preconditions': {'AlertActive(Gunshot)', 'ResourceAvailable(Ranger1)'},
            'effects': {'RiskAssessed(Riverbed)', 'NOT(AlertActive(Gunshot))'}
        },

'Analyze_Thermal': {
            'preconditions': {'AlertActive(Thermal)', 'ResourceAvailable(Ranger2)'},
            'effects': {'RiskAssessed(ElephantCorridor)', 'NOT(AlertActive(Thermal))'}
        },

'Analyze_Intel': {
            'preconditions': {'AlertActive(Intel)', 'ResourceAvailable(Drone)'},
            'effects': {'RiskAssessed(TigerHabitat)', 'NOT(AlertActive(Intel))'}
    },

'Patrol_Riverbed': {
            'preconditions': {'RiskAssessed(Riverbed)','ResourceAvailable(Ranger1)',
'At(Ranger1, Base)'},
            'effects': {'Patrolled(Riverbed)', 'NOT(PoachingRisk(Riverbed, High))'}
  },

'Patrol_ElephantCorridor': {
       'preconditions':{'RiskAssessed(ElephantCorridor)','ResourceAvailable(Ranger2)',
'At(Ranger2,Base)'},
       'effects': {'Patrolled(ElephantCorridor)','NOT(PoachingRisk(ElephantCorridor,
High))'}
  },

'Monitor_TigerHabitat': {
        'preconditions': {'RiskAssessed(TigerHabitat)', 'ResourceAvailable(Drone)',
'At(Drone, Base)'},
        'effects': {'SurveillanceActive(TigerHabitat)'}
  }
```

3.2.5 POP Execution Output

Total Steps: 8 (Start + Finish + 6 core actions)

Ordering Constraints: 21 precedence relationship

Causal Links: 21 producer-consumer relationships



Core Action Sequence (Flexible Partial Order):

1. **Analyze_Gunshot** (Parallel possible with other analysis)
2. **Analyze_Thermal** (Parallel possible with other analysis)
3. **Analyze_Intel** (Parallel possible with other analysis)

4. **Patrol_Riverbed** (Must follow Analyze_Gunshot)
5. **Patrol_ElephantCorridor** (Must follow Analyze_Thermal)
6. **Monitor_TigerHabitat** (Must follow Analyze_Intel)

Causal Dependencies Established:

- Analyze_Gunshot → [RiskAssessed(Riverbed)] → Patrol_Riverbed
- Analyze_Thermal → [RiskAssessed(ElephantCorridor)] → Patrol_ElephantCorridor
- Analyze_Intel → [RiskAssessed(TigerHabitat)] → Monitor_TigerHabitat

3.2.6 Advantages of POP for This Application

1. **Parallelizability**: Analysis actions can execute simultaneously without explicit ordering
2. **Resource Efficiency**: Explicitly tracks resource requirements; prevents over allocation
3. **Flexibility**: Multiple valid execution orderings support adaptability to field conditions
4. **Explainability**: Causal links provide clear justification for action sequences
5. **Minimal Commitment**: Avoids unnecessary constraints that could limit deployment options

3.2.7 Key Limitations

1. **Scalability**: Threat resolution becomes complex with large action sets
2. **Computational Overhead**: Backtracking required when orderings create cycles
3. **Limited Mutex Handling**: Does not explicitly compute mutually exclusive actions

3.3 GraphPlan

3.3.1 Theoretical Foundation

GraphPlan is a forward-chaining planning algorithm that constructs a planning graph layer by layer, alternating between state levels (propositions) and action levels. It explicitly computes mutex (mutually exclusive) relations to identify conflicts and constraints.

3.3.2 Core Principles

1. **Forward Graph Expansion**: Builds layers from initial state toward goals.
2. **Level-Based Progression**: Each level represents achievable propositions and applicable actions.
3. **Explicit Mutex Detection**: Identifies actions/propositions that cannot coexist.
4. **Backward Plan Extraction**: Extracts plan from final state back to initial state.
5. **Graphical Representation**: Planning graph explicitly models reachability.

3.3.3 How GraphPlan Works

Phase 1: Graph Expansion

For each level k:

Action Level Construction (S_k → A_k):

- Include all domain actions whose preconditions are present in S_k.
- Include NOOP (no-operation) actions for persistence of propositions.
- Compute pairwise mutex relations between actions.

State Level Construction (A_k → S_{k+1}):

- Union all effects of applicable actions in A_k.
- Add NOOP-produced persistence propositions.
- Compute pairwise mutex relations between propositions.

Mutex Computation:

1. **Action-Level Mutex:** Two actions are mutex if:
    - **Inconsistent Effects:** One deletes what the other adds.
    - **Interference:** One deletes a precondition of the other.
    - **Competing Needs:** Their preconditions are mutex at previous level.
2. **State-Level Mutex:** Two propositions are mutex if:
    - **Negation:** One is the negation of the other.
    - **Inconsistent Support**: All actions producing the propositions are pairwise mutex.

Termination Condition:

- Goals are present and pairwise non-mutex at level k.
- OR graph levels off (no new propositions added).

Phase 2: Plan Extraction

- Start from the final state level containing all goals (non-mutex).
- Backtrack level by level, selecting non - mutex actions that produce unmet goals.
- Build preconditions for the previous level.
- Continue until the initial state is reached.

3.3.4 GraphPlan Implementation for Wildlife Poaching Prevention

Domain Actions (same logical structure as POP):

```
Analyze_Gunshot, Analyze_Thermal, Analyze_Intel
Assign_Ranger1_Riverbed, Dispatch_Ranger1_Riverbed
Assign_Ranger2_ECorridor, Dispatch_Ranger2_ECorridor
Monitor_Tiger_Drone
```

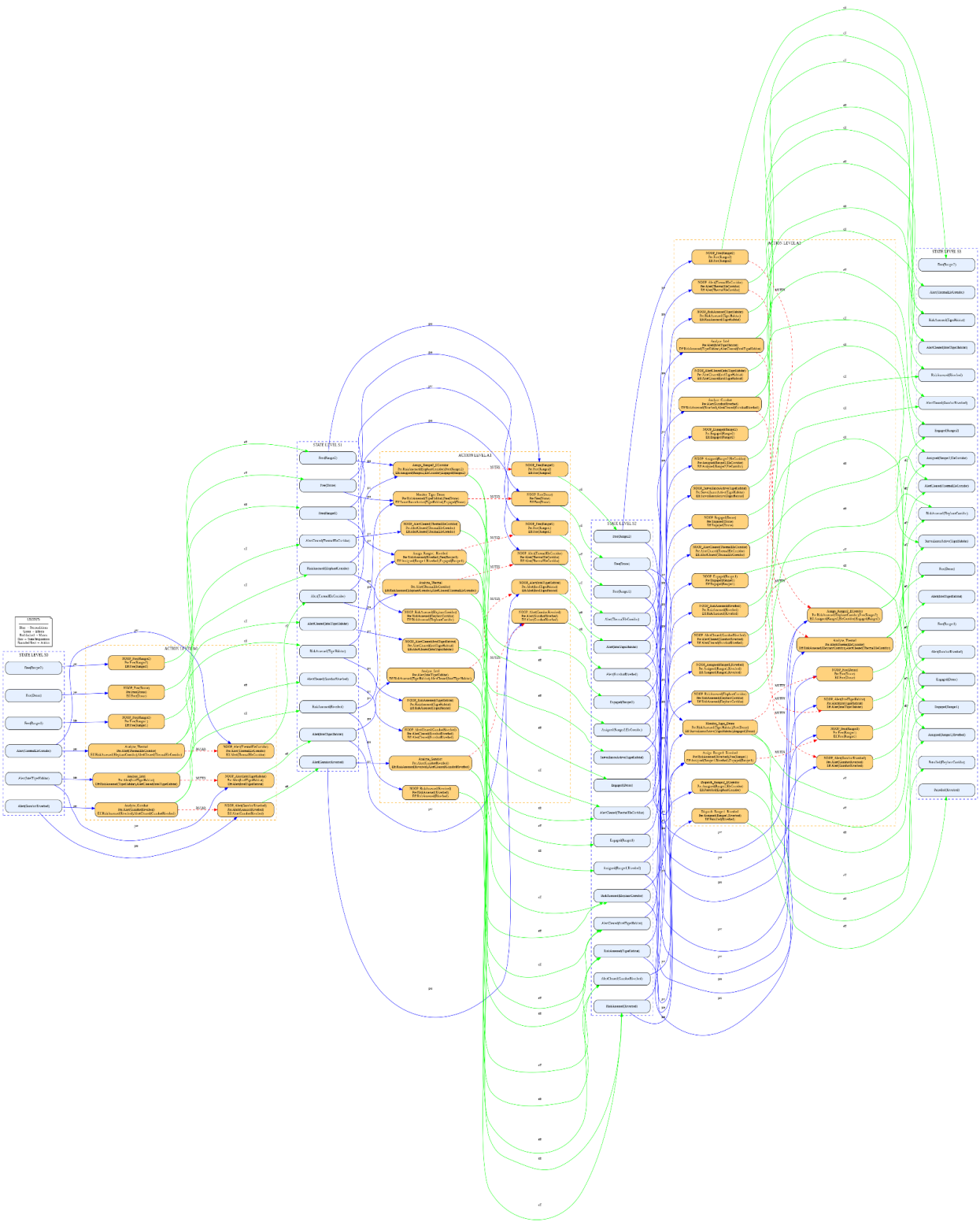
Initial State (S_0):

```
Alert(GunshotRiverbed), Alert(ThermalEleCorridor), Alert(IntelTigerHabitat)
Free(Ranger1), Free(Ranger2), Free(Drone)
```

Goal State:

```
Patrolled(Riverbed), Patrolled(ElephantCorridor), SurveillanceActive(TigerHabitat)
AlertCleared(GunshotRiverbed), AlertCleared(ThermalEleCorridor),
AlertCleared(IntelTigerH
```

3.3.5 GraphPlan Execution Output



Planning Graph Statistics:

State Levels: 4 levels ($S_0 \rightarrow S_3$)

Action Levels: 3 levels ($A_0 \rightarrow A_2$)

Proposition Growth: 6 → 12 → 18 → 20 propositions

Action Growth: 9 → 18 → 26 actions (including NOOPs)

Planning Graph Layer Details:

Level	Type	Count	Key Components
S_0	Initial State	6	All alerts active, resources free
A_0	Actions	9	3 analysis + 6 NOOPs
S_1	Intermediate	12	Alerts + risks + resources
A_1	Actions	18	6 domains + 12 NOOPs
S_2	Intermediate	18	Added assignments + engagements
A_2	Actions	26	8 domains + 18 NOOPs

Level	Type	Count	Key Components
S_3	Goal-Satisfying	20	All goals achievable, non-mutex

Extracted Plan (Core Actions - Non-NOOP):

Step 1: Analyze_Gunshot

- Pre: Alert(GunshotRiverbed)
- Eff : RiskAssessed(Riverbed), AlertCleared(GunshotRiverbed)

Step 2: Analyze_Thermal

- Pre: Alert(ThermalEleCorridor)
- Eff : RiskAssessed(ElephantCorridor), AlertCleared(ThermalEleCorridor)

Step 3: Analyze_Intel

- Pre: Alert(IntelTigerHabitat)
- Eff : RiskAssessed(TigerHabitat), AlertCleared(IntelTigerHabitat)

Step 4: Assign_Ranger1_Riverbed

- Pre: RiskAssessed(Riverbed), Free(Ranger1)
- Eff : Assigned(Ranger1,Riverbed), Engaged(Ranger1)

Step 5: Assign_Ranger2_ECorridor

- Pre: RiskAssessed(ElephantCorridor), Free(Ranger2)
- Eff : Assigned(Ranger2,EleCorridor), Engaged(Ranger2)

Step 6: Monitor_Tiger_Drone

- Pre: RiskAssessed(TigerHabitat), Free(Drone)
- Eff : SurveillanceActive(TigerHabitat), Engaged(Drone)

Step 7: Dispatch_Ranger1_Riverbed

- Pre: Assigned(Ranger1,Riverbed)
- Eff : Patrolled(Riverbed)

Step 8: Dispatch_Ranger2_ECorridor

- Pre: Assigned(Ranger2,EleCorridor)
- Eff : Patrolled(ElephantCorridor)

3.3.6. Advantages of GraphPlan for This Application

1. **Explicit Mutex Computation**: Clearly identifies resource conflicts (e.g., Ranger1 cannot be in two places)
2. **Completeness Guarantee**: If plan exists, GraphPlan will find it
3. **Reachability Analysis**: Directly identifies achievable goals vs. impossible ones
4. **Structured Representation**: Planning graph provides visual evidence of dependencies
5. **Performance**: Often faster than POP for large problems with many conflicts

3.3.7. Key Limitations

1. **Memory Consumption**: Planning graph can become very large
2. **Less Flexible Ordering**: Level-by-level structure enforces sequential progression
3. **NOOP Proliferation**: Many persistence actions clutter the graph
4. **Post-hoc Plan Extraction**: Must extract plan after full graph is built

3.4. Comparative Analysis

3.4.1. Algorithm Comparison

Characteristic	POP	GraphPlan
Planning Paradigm	Backward chaining, least-commitment	Forward chaining, graph-based
Action Ordering	Partial order (flexible)	Sequential levels (fixed progression)
Mutex Representation	Implicit (threat resolution)	Explicit (computed at each level)
Flexibility	High multiple valid orderings	Lower—level-based constraints
Parallelizability	High independent actions need not order	Limited—levels impose sequencing
Computational Approach	Iterative refinement, backtracking	Incremental layer expansion
Plan Extraction	Concurrent with planning	Post-planning backward search
Core Actions Generated	6 actions	8 actions
Scalability	Degrades with action/conflict complexity	Degrades with state space size
Memory Usage	Lower	Higher (full graph storage)

3.4.2 Critical Differences in Outputs

POP Partial Order vs. GraphPlan Sequential Plan:

POP: Start → [Analyze_Gunshot || Analyze_Thermal || Analyze_Intel] → [Patrol_Riverbed, Patrol_ElephantCorridor, Monitor_TigerHabitat] → Finish

This notation (|| = parallel) shows that three analysis actions have no ordering constraints between them.

GraphPlan: Start → Analysis Phase → Assignment Phase → Dispatch Phase → Finish

This sequential phase structure naturally maps to command-center decision cycles.

3.5. Conclusion

Interpretation and Insights

1. Both planners succeed because the domain is well-structured, deterministic, and free of contradictory goals.
    - POP highlights parallelism.
    - GraphPlan highlights reachability and conflict handling.
2. In practical deployment, ranger teams can follow POP-style flexibility, while command-center planning can use GraphPlan's phase-based breakdown.
3. The revised study confirms that both POP and GraphPlan effectively generate patrol strategies for the wildlife reserve.
4. Combining insights from both helps balance flexibility and structure during real patrol operations.

3.6. Individual Contributions

Konisa Sai Sriyuktha(25CS06016):

- Collaboratively constructed initial state, goal state and all action states required.
- Implemented GraphPlan algorithm and code and obtained the graph.
- Documented GraphPlan

Bikram Shahi(25CS06010):

- Collaboratively constructed initial state, goal state and all action states required.
- Implemented POP algorithm and code and obtained the graph.
- Documented POP