OMEGA*: An Ontology-Driven Tool for Explaining Multi-Agent Path Finding

Bharath Muppasani, Ritirupa Dey, Biplav Srivastava, Vignesh Narayanan

AI Institute, University of South Carolina, Columbia, SC, USA

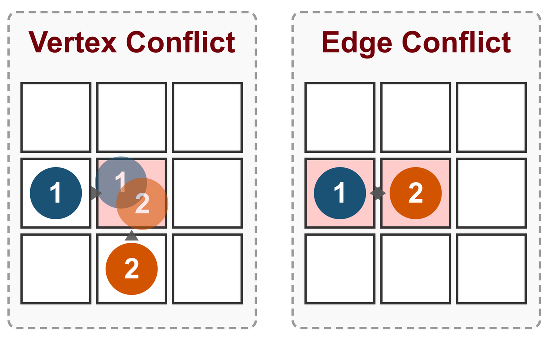
Paper

AI INSTITUTE #AIISC UNIVERSITY OF SOUTH CAROLINA

Motivation & Why OMEGA*

Problem

Multi-Agent Path Finding (MAPF) considers n agents on a graph $G = (V, E)$, each with start s_i and goal g_i. A solution $\Pi = \{\pi_i\}$ assigns collision-free paths over discrete time, avoiding vertex conflicts $(v_t^i = v_t^j)$ and edge swaps $(v_t^i, v_{t+1}^i) = (v_{t+1}^j, v_t^j)$.



- Robots may **wait or detour** in ways that feel like a "black box" to operators
- Raw planner logs list cells and timesteps, but **don't encode causal links** (who blocked whom, what conflict triggered replanning)
- Existing explanation approaches provide only **partial coverage** (visual segmentation or logic-based queries, but not both)

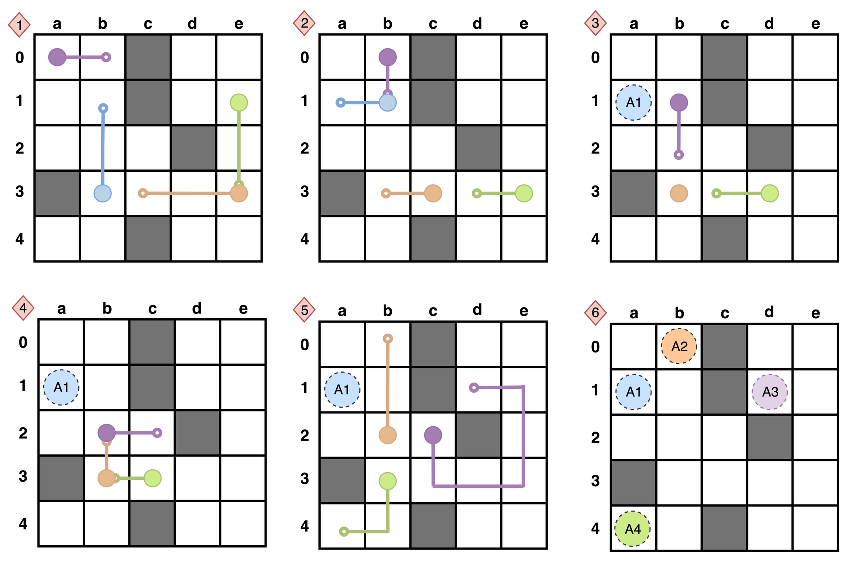


Figure 1: Plan Segmentation based Visualization

Solution

OMEGA* converts planner execution logs into a semantic knowledge graph using the Multi-Agent Planning Ontology (maPO):

- Users ask **queries**; OMEGA answers with grounded rationales and synchronized grid highlights
- Transforms opaque execution traces into **queryable, auditable explanations**
- Works across MAPF algorithms with minimal modifications for canonical JSON logging

What's New

- **Planner-agnostic normalization:** canonical JSON log schema for any MAPF solver
- **Interoperable semantics:** reuses OWL-Time, PROV-O, SOSA/SSN, CORA standards
- **Causal chain captured as events:** CollisionEvent → ConflictAlert → ReplanningStrategy → (Original/Resolved)SubPlan

OMEGA* Explanations for MAPF

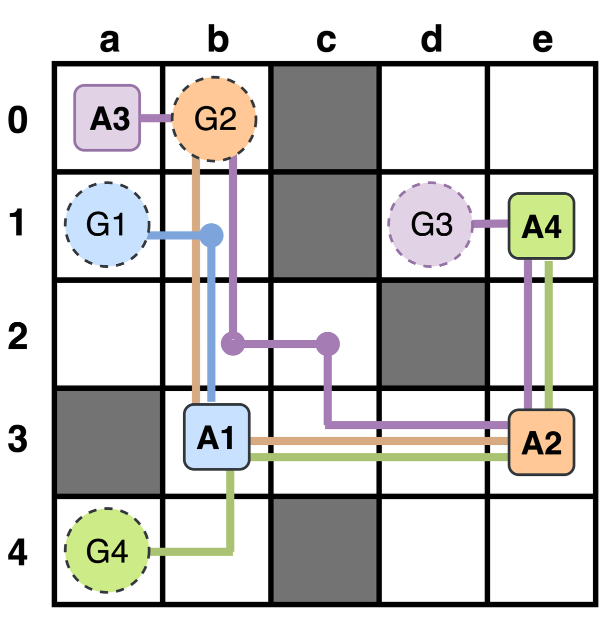


Figure 2: MAPF instance with 4 agents.

Question 1: What conflicts was Agent 3 in?

Answer 1: Agent 3 was involved in the following:

- *t=1* — **vertex** conflict at *(b1)* with Agent 1.
- *t=3* — **vertex** conflict at *(b2)* with Agent 2.

Question 2: How was the conflict at (b2) resolved?

Answer 2: A vertex conflict occurred at location *(b2)*. To resolve the conflict with **Agent 3**, **Agent 2** changed its plan.

New plan for Agent 2:

- *t=3* — Wait at *(b3)* (Duration: 1 step).
- *t=4* — Move to *(b2)*.
- *t=5* — Replan complete.

Question 3: Provide a global plan summary.

Answer 3:

- **Global Makespan:** 12.0
- **Sum of Costs:** 28.0
- **Replans Triggered:** 2

Question 4: Why did Agent 3 wait at t=1?

Answer 4: Agent 3 waited to avoid an **Vertex Conflict** with Agent 1 traversing (b2) → (b1).

System Architecture

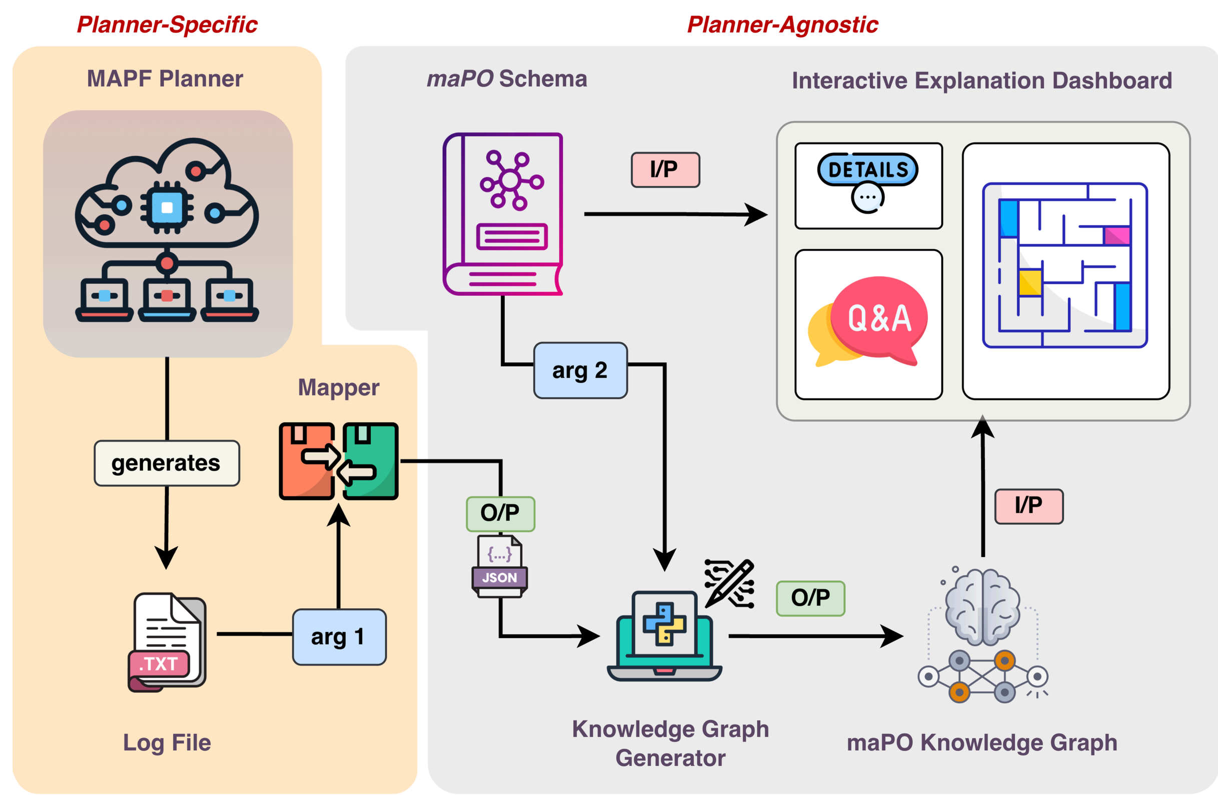


Figure 3: System architecture showing how planner-specific outputs are converted into the maPO knowledge graph via a generic layer, keeping the method planner-agnostic for interactive, ontology-driven explanations.

Pipeline

The framework operates in three stages that ensure planner-agnostic operation:

- **Log Generation:** Any MAPF planner (CBS, ICBS, RL) outputs a standardized JSON trace containing environment, agents, paths, collisions, and alerts
- **Knowledge Graph Creation:** Python mapper asserts RDF triples consistent with maPO schema and exports RDF/Turtle
- **Explanation Generation:** SPARQL queries over the knowledge graph; results populated into natural language templates
- **Interactive Dashboard:** Text + visual overlays synchronized with grid simulation for inspection

Planner Agnosticism: Canonical JSON

- **Planner-Agnostic by Design:** OMEGA* does not rely on solver internals; it works with any MAPF solver.
- **Canonical JSON Export:** Solvers need to export a standardized execution trace (agents, paths, conflicts, alerts, and plan revisions).
- **Universal Debugger:** A generic log→maPO mapper converts log traces into instantiated maPO knowledge graph, enabling SPARQL-based explanations.

```
{
  "environment": {
    "gridSize": [R, C],
    "obstacles": [{ "id": obs_id, "cell": [r, c] }]
  },
  "agents": [{
    "id": agent_id,
    "start": { "t": t0, "cell": [rs, cs] },
    "goal":  { "cell": [rg, cg] }
  }],
  "collisions": [{
    "id": coll_id,
    "t": t,
    "type": "vertex" | "edge",
    "loc": [r, c],
    "agents": [ai, aj]
  }],
  "alerts": [{
    "id": alert_id,
    "conflict": coll_id,
    "agent": agent_id,
    "type": alert_type,
    "reason": text
  }],
  "jointPlan": {
    "subplans": [plan_id],
    "makespan": T
  }
}
```

Evaluation Highlights

User study (N=28) comparing maPO-generated explanations vs. raw planner logs across 3 scenarios (RL, CBS, ICBS):

94% PREFERENCE	4.44/5 CLARITY RATING

- Users chose maPO explanations in **159 of 167** recorded preferences (Binomial Test, $p < .001$)
- Wilcoxon signed-rank tests confirmed median clarity ratings significantly above neutral

Cognitive Load Metrics (N=18)

- **Flesch Reading Ease (FRE): 94.39** — "Very Easy" readability level
- **Automated Readability Index (ARI): 4.87** — Grade 4-5 comprehension
- **Coleman-Liau Index (CLI): 1.13** — Grade 1-2 level text

These scores confirm maPO explanations impose minimal linguistic burden and are well-suited for rapid comprehension.

maPO Ontology: Core Concepts

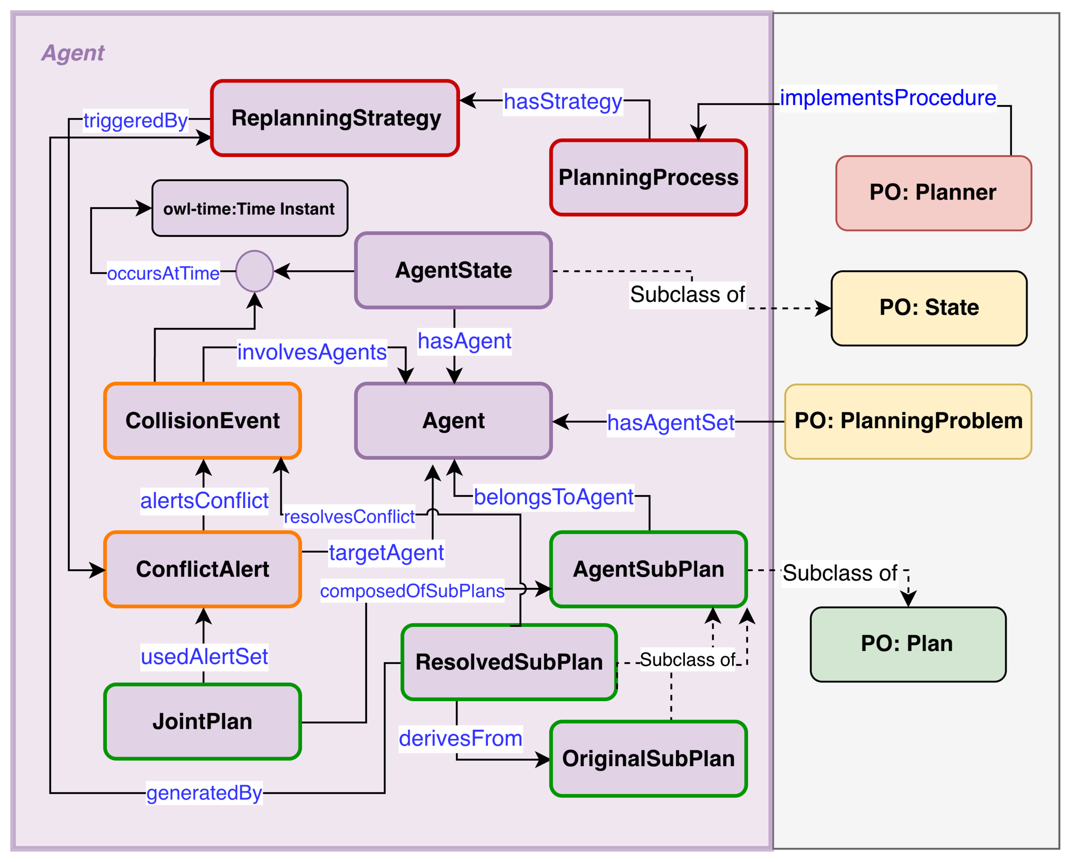


Figure 4: Illustrative Multi-Agent Planning Ontology (maPO) schema showing conflict–alert lifecycle (orange), replanning processes (red), and plan evolution (green). The full, up-to-date ontology is available on our explanation platform via its PURL URI.

Design: Competency Questions

C1: Which CollisionEvents (including their time, type, location, and involved agents) were detected during planning?

C2: For a given CollisionEvent, which agent(s) received a ConflictAlert?

C3: What was an agent's original, conflict-unaware plan, and how does it compare to its final, resolved plan?

C4: Why did a specific agent have to wait or reroute in its final plan?

C5: For a given ConflictAlert, which ReplanningStrategy did the agent use?

C6: What was the cost change associated with a revised AgentSubPlan?

C7: Why was a particular agent (from a set of conflicting agents) chosen to be the one to replan? (i.e., what was the planner's selectionRationale?)

C8: What is the final JointPlan after all conflicts are resolved, and what is its overall makespan?

Core Concepts

`ma:Agent, ma:AgentState`
Aligns with sosa:Platform/Sensor

`ma:CollisionEvent`
Captures conflict type, time, and location

`ma:ReplanningStrategy`
Provenance of how conflict was resolved

`ma:AgentSubPlan`
Linked series of Steps (time, cell)

Key Properties

- **alertsConflict, targetAgent:** who was alerted for which conflict
- **triggeredBy, derivesFrom:** PROV-O provenance chain
- **resolvesConflict, generatedBy:** causal links from problem to solution

KEY TAKEAWAYS

Human-Robot Interaction

- **Robots become explainable: waiting/detours get causal labels.**
- **Trust Calibration: operators can verify planner decisions.**
- **Diagnose congestion across robot teams.**

Semantic Web Impact

- **Interoperable KG via W3C standards (PROV-O, OWL-Time).**
- **Planner Agnostic: Decouples explanation from solver.**
- **Extensible: SPARQL-driven templates.**

Acknowledgements & References

*This work was supported in part by a **J.P. Morgan Chase Faculty Award**, by the **Air Force Office of Scientific Research** under Award No. FA9550-24-1-0228, and by the **National Science Foundation** under Award No. 2337998.*

[1] Muppasani, B., Dey, R., Srivastava, B. and Narayanan, V., 2026. OMEGA: An Ontology-Driven Tool for Explaining Multi-Agent Path Finding. AAAI 2026.

[2] Muppasani, B.C., Gupta, N., Pallagani, V., Srivastava, B., Mutharaju, R., Huhns, M.N. and Narayanan, V., 2025. Building a planning ontology to represent and exploit planning knowledge and its aplications. Discover Data, 3(1), p.55.

Contact: bharath@email.sc.edu, deyr@email.sc.edu

Video **Website**