Posthoc:
A Visualisation Framework for Understanding Search

Kevin Zheng[1]**, Daniel Harabor**[1]**, Michael Wybrow**[1]

[1]Monash University, Australia

Abstract

We present POSTHOC, a debugging and visualisation framework that helps users better understand how search algorithms work. POSTHOC takes as input *search traces*, human-readable output logs produced by an algorithmic problem solving program. The logs are are used for subsequent playback, analysis and visualisation. Our system does not depend on any specific type of visualisation nor any particular decision-making schema. Being independent, POSTHOC readily complements new and existing solvers: for AI planning, pathfinding, and heuristic search, and it can be integrated as a complementary problem-solving tool alongside.

Introduction

Search algorithms are considered a foundational topic in the field of Artificial Intelligence (Russell and Norvig 2021) and they are often found at the heart of leading solvers, for a variety of important practical settings; e.g., AI Planning (Wilkins 2014), Game Development (Rabin 2019), Robotics (Kavraki and LaValle 2016), Routing (Bast et al. 2016) and more. Substantial interest in the topic has also produced a variety of complementary tools that try to help practitioners better understand search programs. Recent examples include MAES (Andreasen et al. 2022), a visualisation and debugging environment for robotics applications, PDSim (De Pellegrin and Petrick 2023), a tool which visualises and simulates planning domains, and Sturtevant's collection of Single Agent Search demos (Sturtevant 2021), which help newcomers to the area understand how influential and foundational algorithms actually work. Yet, difficulties arise when attempting to extend these tools beyond their original context; e.g., to visualise output from a new type of algorithm, to examine solutions for a new kind of domain, or simply trying to plot existing information in a new way. This is because search procedures and outputs vary widely from one problem to the next. Moreover, the core insights which make algorithms successful can often depend on details from the domain in which they are applied.

In this demo we present POSTHOC, a new visualisation framework which can address these shortcomings. Like some other tools, our system allows practitioners to quickly profile a wide variety of search procedures, then contextualise their results with a corresponding visual representation.

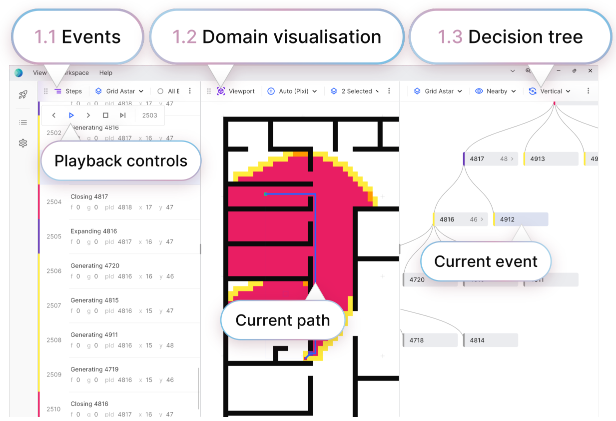

Figure 1: Basic A* grid search in POSTHOC

Listing 1: Simple search trace `example.trace.yaml`

```
version: 1.1.0
view:
  main:
  - $: rect
    x: ${{x}}
    y: ${{y}}
    color: ${{palette[type]}}
events:
- {type: expand, id: 2, x: 8, y: 15, f:
    2, g: 3}
- {type: generate, id: 3, pid: 2, x: 9,
    y: 15, f: 2, g: 4}
```

Unlike similar tools, POSTHOC is technology-agnostic, being independent of any specific domain, solving program or algorithmic strategy. We achieve this using *search traces*, solver produced output logs that help to decouple search from visualisation. Our framework has near-zero upfront cost: it has no installation, no set-up, and requires only minimal knowledge of search procedures to get started. The main goals of POSTHOC are twofold: (i) reduce the barrier-to-entry for producing visualisations and; (ii) assist non-experts to engage with cutting-edge developments in the area of state-space search. A video demonstration is available at https://shorturl.at/pJKT5.

Figure 2: JPSW

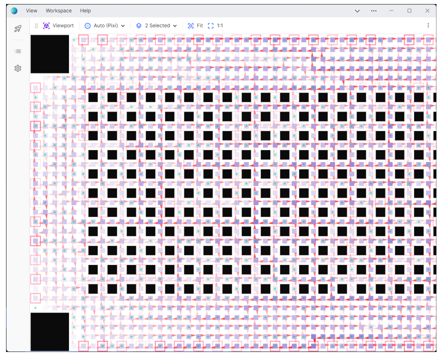

Figure 3: Guided PIBT

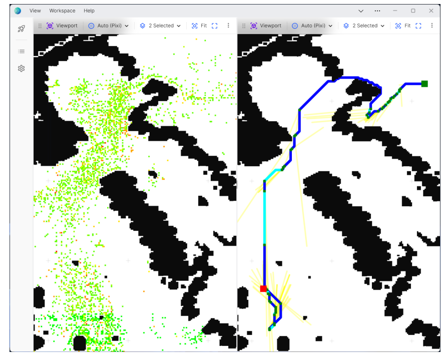

Figure 4: StarCraft game analysis

System Description

POSTHOC takes as input a structured event log, which we call a *search trace*. The logs describe fundamental search operations and search outputs; e.g., `expand`, `generate`, `relax` and `goal`. Each event has a corresponding set of labels, which record *node-id*, *cost* and *parent* information. Search programs are often instrumented to produce such logs, during algorithmic development, and some solvers output event logs as part of their core functionality; e.g., WARTHOG (Harabor 2024), a library for pathfinding search. Our system requires event logs follow a specific `YAML`-format, which we illustrate in Listing 1. The basic form (lines 8–11) suffices to visualise the search trace as a decision tree. Events can be further annotated with arbitrary metadata, such as state descriptors (e.g., agent position) and drawing primitives (lines 2–7). These are used to produce a custom visualisation of the domain. Figure 1 shows an example for a grid-based pathfinding problem. The visualisation was generated using a search trace similar to Listing 1.

Playback and Interrogation

POSTHOC allows users to explore recorded data via simple playback mechanisms. Events are parsed and visualised in input order, which allows the user inspect the process: step by step, to verify the correctness of each operation, or holistically, to acquire general insights into the search process (e.g., where were the “hard bits”). Another possibility is to interrogate the search process using *breakpoints*, pausing playback when specific conditions are met; e.g., when a certain node is expanded, when a new solution is found or when a specified invariant (such as monotonicity) is violated.

For each step of the search the system offers two views: a domain-independent *decision-tree* and a domain-specific *rendered view*. In each view the user can select elements to better understand the search process; e.g., selecting a tree-node shows the metadata associated with that node and the sequence of decisions to that node, from the root. Figure 1 shows an example. When multiple search traces are loaded, POSTHOC facilitates comparisons between different solver outputs. Figure 4 shows an example where we analyse planning decisions for a game of StarCraft. The right shows the planned path for one agent; the left shows the locations of temporal obstacles, which helps the user understand why the path looks as it does.

Integrated Search

In pedagogical settings it is often desirable to interact directly with a solver program and observe its output. Interactivity helps users to better understand search algorithms by observing how they tackle different problems, in real time. Integrating a solver with POSTHOC is straightforward: the user specifies which executable to invoke and which input problem file. A more complex use-case allows the user to specify particular start and target states, directly from the visualiser. In this case the integration with the solver requires an additional schema, which maps input from the visualiser to the problem format expected by the solver, so that new problem instances can be created on-the-fly.

Posthoc In Practice

To evaluate POSTHOC we undertook a range of user studies with postgraduate students whose projects involve problem solving using state-space search. We conclude with three real-world examples from these studies.

Case 1: Debugging an implementation of Weighted Terrain Jump Point Search (JPSW; Carlson et al. 2023). In rare cases the implementation incorrectly pruned optimal solutions from the search space. Direct inspection of output logs did not produce any clues about the cause of the error. Using POSTHOC, the researcher created a visualisation of the search (Figure 2) and found the error in minutes.

Case 2: Optimising a MAPF algorithm. In this complex use case, a researcher used POSTHOC to better understand the effectiveness of a recently propopsed MAPF algorithm (Zhang et al. 2024). The resulting visualisation (Figure 3) allowed the researcher to identify opportunities for further improvement: it shows agent occupancy and directional flows as a heat map.

Case 3: Game analysis. In this use case, a researcher wanted to analyse the behaviour of path-planning agents in a game of StarCraft. The game involves many agents and thousands of path planning episodes among numerous dynamic obstacles. The researcher produced several visualisations (Figure 4) including trajectories for each individual agent, heatmaps of all paths, and all locations appearing as a start or target.

References

Andreasen, M. Z.; Holler, P. I.; Jensen, M. K.; and Albano, M. 2022. MAES: A Realistic Simulator for Multi-Agent Exploration and Coverage. In *Sysem Demonstration, 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Bast, H.; Delling, D.; Goldberg, A.; Müller-Hannemann, M.; Pajor, T.; Sanders, P.; Wagner, D.; and Werneck, R. F. 2016. Route planning in transportation networks. *Algorithm engineering: Selected results and surveys*, 19–80.

Carlson, M.; Moghadam, S. K.; Harabor, D. D.; Stuckey, P. J.; and Ebrahimi, M. 2023. Optimal Pathfinding on Weighted Grid Maps. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10): 12373–12380.

De Pellegrin, E.; and Petrick, R. P. 2023. PDSim: Simulate Plans. In *System Demonstration, 33rd International Conference on Automated Planning and Scheduling (ICAPS)*.

Harabor, D. 2024. The Warthog Pathfinding Library. https://bitbucket.org/dharabor/pathfinding. Last accessed 2024-03-25.

Kavraki, L. E.; and LaValle, S. M. 2016. Motion planning. In *Springer handbook of robotics*, 139–162. Springer.

Rabin, S. 2019. *Game AI Pro 360: Guide to Movement and Pathfinding*. CRC Press.

Russell, S.; and Norvig, P. 2021. *Artificial Intelligence: A Modern Approach (4th Ed)*. Pearson Education.

Sturtevant, N. 2021. Demos for a Course in Single-Agent Heuristic Search. In *System Demonstration, 31st International Conference on Automated Planning and Scheduling (ICAPS)*.

Wilkins, D. E. 2014. *Practical Planning: Extending The Classical AI Planning Paradigm*. Elsevier.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. *ArXiv*, abs/2402.01446.