SPECTER LABS RESEARCH BLOG

Wonton Soup: Proof Structures Under Interventions

Specter Labs

1. Wonton Soup: Proof Structures Under Interventions

`wonton-soup` is our intervention harness for proof-search experiments. The core question is simple:
when we perturb a solver's search process, does it return to the same proof structure, or settle into a different one?
We run wild-type and intervention sweeps over deterministic theorem samples, capture full search artifacts (`*_history.json`, `*_mcts_tree.json`, `*_graph.json`, `*_comparison.json`), and compare outcomes across seeds, tactics, providers, and backends.

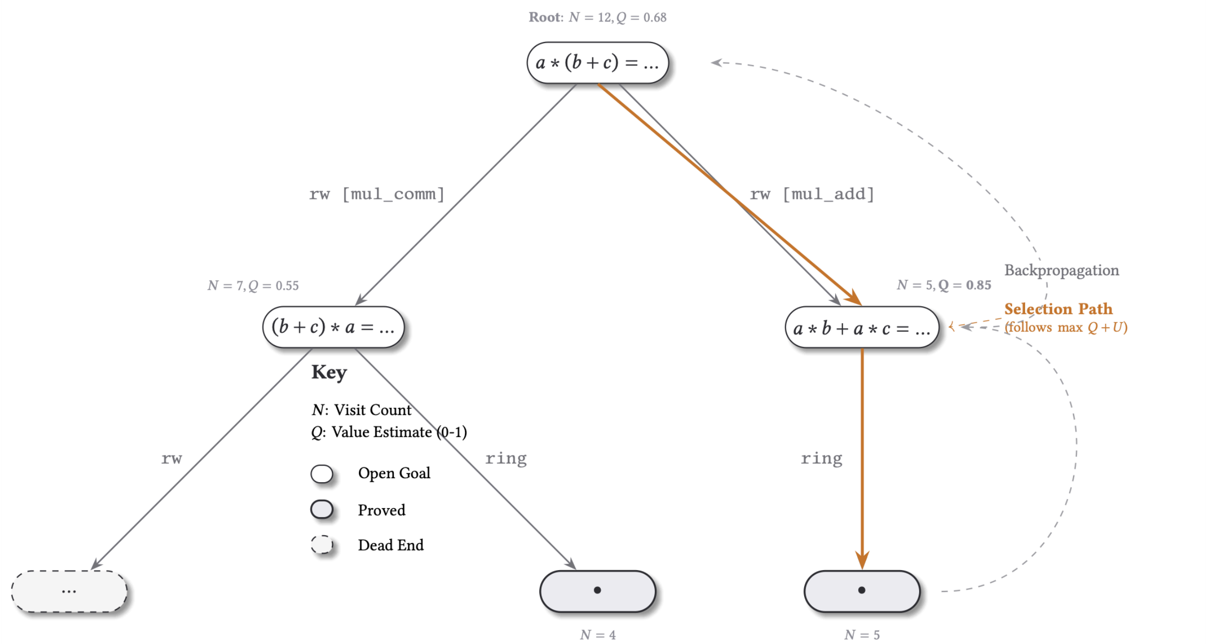

Figure 1: MCTS Proof Search Tree

1.1. 1. What We're Looking For

We treat proof search as a stochastic process over structured states.

- A perturbation can be a blocked tactic or tactic family, a seed change, or a policy/scheduler change.
- A response can be recovery to the same structural family or migration into a different attractor.
- The object of study is not only solve rate; it is the shape and stability of search trajectories.

This is why we log enough structure to replay and compare runs months later under fixed configuration.

1.2. 2. Search, Lesions, and Structural Invariance

Three threads from the Diverse Intelligence research program motivate this setup.

1.2.1. Search efficiency as a measurable quantity

Chis-Ciure and Levin (2025) formalize biological intelligence as search efficiency in multi-scale problem spaces. Their central metric is the $log10$ of the ratio between the cost of a random walk and that of the observed agent: how many orders of magnitude of work does a directed policy save over maximal-entropy search? Even under conservative assumptions, they show that organisms as simple as amoebae navigating chemical gradients operate hundreds to sextillions of times more efficiently than a blind baseline.

We borrow this framing directly. Our K metric is the same log-ratio, applied to proof search: τ_{blind} is the expected edge count for a null policy over the tactic action surface, τ_{agent} is the observed count, and $K = \log_{10}(\tau_{blind}/\tau_{agent})$. A positive K means the solver is exploiting structure in the problem space rather than brute-forcing it.

1.2.2. Local lesions, global behavioral readout

Zhang, Goldstein, and Levin (2024) reframe classical sorting algorithms as models of morphogenesis. Rather than treating algorithms as fixed computational procedures, they let each array element exert minimal local agency and implement sorting policies from the bottom up. The key finding is what happens under perturbation: when elements are "damaged" and cannot execute perfectly, the decentralized approach outperforms traditional implementations. Arrays with defective elements sort themselves more reliably than top-down implementations facing the same errors. The system exhibits unexpected competencies never explicitly programmed, including the ability to temporarily reduce local progress in order to navigate around a defect.

This is the template for our intervention protocol. We block one tactic or tactic family from a known solution path and rerun. The question is the same one Zhang et al. ask of their self-sorting arrays: does the system reroute around the lesion and still reach the goal, or does it collapse? And if it reroutes, is the resulting structure the same or different?

1.2.3. Pattern-level invariants under perturbation

Levin (2022) introduces the TAME framework for understanding cognition across radically different substrates. The core insight is a deep symmetry between problem-solving in anatomical, physiological, transcriptional, and behavioral spaces: the same patterns of multi-scale competency appear regardless of whether the substrate is a cell colony, a regenerating planarian, or a neural network. TAME argues that what matters is not whether a particular mechanism is present, but whether a behavioral structure persists under perturbation across scales.

We adopt this as our primary object of study. In the same vein, we ask whether the proof-search trajectory belongs to the same structural family as the wild-type run. Basin analysis, GED families, and attractor clustering are all ways of asking the TAME question in a formal-methods setting: is the pattern invariant, or did the perturbation push the system into a genuinely different attractor?

1.2.4. Mapping to proof search

In proof search, we try and have these three threads converge: block parts of tactic space, rerun under controlled budgets, and measure how structure changes. We then have:

- K quantifies efficiency relative to blind.
- GED quantifies structural distance from wild-type.
- Basin analysis quantifies whether the system has one stable attractor or many.

Together, they let us ask whether a proof-search process exhibits the kind of robust, multi-path competency that Levin and collaborators study in biological systems, or whether it is brittle and path-dependent.

1.3. 3. Harness and Corpus Design

Corpus management is incredibly important for us, both to have the necessary data to run these experiments, but also to make sure they are easily reproducible and stretch what our solver setup is capable of.

`wonton-soup` treats corpus and run configuration as first-class artifacts. Corpus builds are manifest-backed and run configs are snapshot-pinned. All downstream analysis reads those artifacts directly, which keeps the baseline fixed while we perturb one axis of behavior.

We also separate validity from capability. Gate A checks that an item is structurally processable for the chosen backend and schema contract. Gate B checks whether the provider and search policy can do meaningful work on that valid slice under the configured budget. Intervention studies run on slices that pass both gates, so failure modes are easier to interpret.

Deterministic selection (`--sample` with `--seed`) is how we ensure replayability. If you rerun later with the same corpus ref and selector inputs, you should recover the same theorem slice and comparable outputs.

Run-level schemas complete the provenance contract with `run_config.json`, `run_status.json`, and `summary.json.gz` being responsible for postprocess, lake extraction, and cross-run audits.

1.3.1. What is fixed per comparison run

- Corpus reference plus build provenance (`manifest.json`, item ordering, hash identity).
- Selection procedure (`--sample`, `--seed`, `--offset`, `--limit`) and resulting theorem slice.
- Search budget and core execution knobs (mode, iteration budget, intervention declaration).
- Analysis-facing artifact contract (`run_config.json`, `run_status.json`, `summary.json.gz`, theorem subartifacts).

1.4. 4. Search Core: Centralized and Distributed MCTS

The search core is one proof-space model with two control regimes, not two unrelated systems. Both modes walk a tactic-conditioned state graph and use compatible logging contracts, so a mode switch changes execution dynamics without changing what downstream analysis reads.

Centralized mode is the structural baseline. A single global selection loop owns frontier choice and expansion order, which gives one policy view over one queue and minimizes coordination effects. This is the cleanest surface for proof families, basin structure, reroute versus collapse, recovery after intervention, and blind-relative efficiency.

Distributed mode is the collective-control layer over that same proof-space morphology. Multiple local agents operate over a shared frontier. Inflight reservations are used to reduce duplicate expansion pressure, and scheduling controls let us perturb coordination directly: block, delay, reroute, virtual loss, and depth/path bias interventions change who explores what and when.

That gives us a clean research stack. First use centralized MCTS to establish the structural landscape of a theorem slice. Then use distributed MCTS to test how coordination changes access to that landscape. The scheduling interventions in distributed mode are deliberate lesions on search dynamics and let us ask whether solve behavior depends on one narrow expansion regime or remains robust under controlled changes in local-global coordination.

Both modes emit compatible tree and trace artifacts, so comparisons can stay within the same analysis contract (`*_mcts_tree.json`, traces, run summaries) instead of requiring mode-specific postprocess logic.

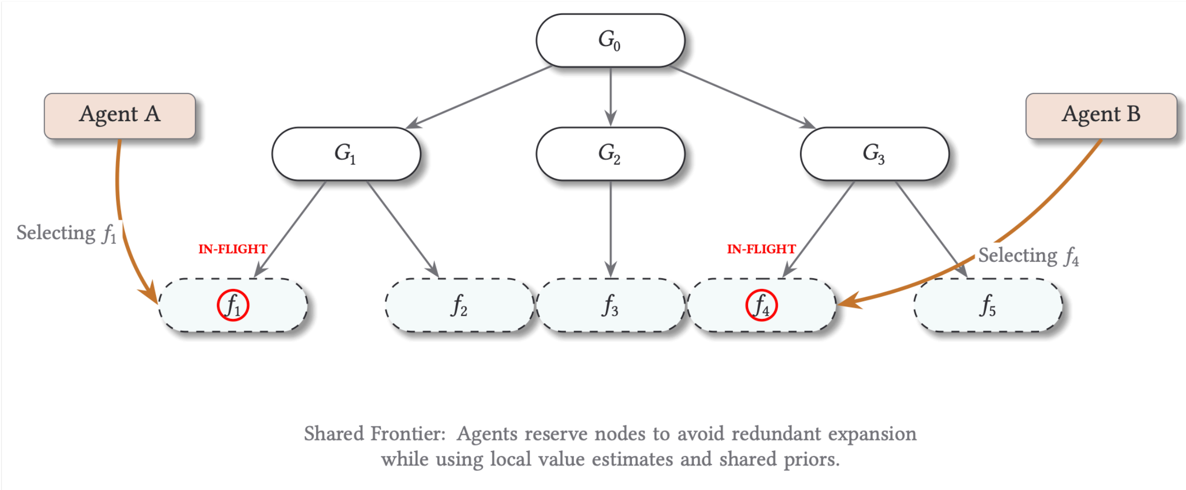

Figure 2: Distributed Frontier

How to read the figure: each worker lane represents local agent activity against a shared frontier; reservations and scheduler policy shape contention and handoff. Dense synchronized bands suggest strong coupling, while staggered bands indicate looser parallel exploration.

1.5. 5. Backend Families and Artifact Compatibility

We use a multi-backend harness to measure whether behavioral patterns appear across different backends or are only implementation artifacts. If a pattern recurs across backend families with different proof objects and trace surfaces, it is a stronger candidate invariant.

`wonton-soup` currently supports five execution backends:

- `lean`
- `coq`
- `e`
- `vampire`
- `z3`

Run-level schemas are shared (`run_config.json`, `run_status.json`, `summary.json.gz`), and backend-specific capabilities are explicit via `run_status.json` flags plus file-presence checks. That means a downstream consumer can fail loud on unavailable artifact families instead of guessing.

This contract matters for mixed analyses. For example, `ged_search_graph` is meaningful only when a true search graph exists; external solver traces may map to `ged_trace_graph` or proof-object families instead. Capability flags and validity metadata keep those distinctions visible.

1.5.1. Backend Artifact Families (Typical)

Backend	Search-graph family	Proof family	Trace family	Practical note
`lean`	`ged_search_graph`	proof-term artifacts when enabled	MCTS traces	Full search-graph comparisons are strongest here.
`coq`	usually unavailable	proof object family (integration dependent)	backend trace varies	Treat proof/trace availability as capability-gated.
`e`	unavailable	proof object family	`ged_trace_graph` from TSTP-style traces	Mark trace completeness explicitly.
`vampire`	unavailable	proof object family	optional trace family	Proof-centric comparison is typical.
`z3`	unavailable	proof object family	optional trace family	Search-graph GED is not the primary signal.

Cross-backend comparisons are safest on shared run-level outcomes and explicitly labeled metric families. Structure-level comparisons should be grouped by compatible artifact families, not collapsed into one undifferentiated score.

1.5.2. Showcase

We pin two high-signal views used repeatedly in analysis: attractor separation and blind-relative efficiency.

Attractor view: structural families and basin mass concentration. K view: efficiency over blind baseline for intervention comparisons.

1.6. 6. Metrics and Comparison Families

We use a metric stack, not a single score:

- K-style search efficiency (`k_search_efficiency`) from trace-derived blind nulls.
- Paper-style paired blind baseline (`paper_k`) from basin runs with `--basin-blind`.
- GED families (`ged_search_graph`, `ged_search_graph_soft`, `ged_proof_graph`, `ged_trace_graph`) with explicit validity metadata.

- Trajectory comparison (divergence, reconvergence, recovery iterations).
- Basin analysis (solve rate, structure hash diversity, dominant basin frequency).
- Sheaf analyses (equivalence consistency and tactic-transform residuals).
- Cross-run lake exports for reproducible, cross-experiment aggregation.

1.6.1. Quick Metric Interpretation

Metric	What changed in the intervention run	How to read it
`k_search_efficiency` / `paper_k`	Attempted edge count before first solve (τ_{agent}) vs blind baseline (τ_{blind})	Higher is better; $K > 0$ means fewer attempts than blind
`normalized GED_search`	Search-graph structure relative to wild-type	Near `0` means structurally similar search; larger values mean stronger reroute
shared prefix	Number of early wild-type steps replayed before divergence	High prefix means late divergence; low prefix means early policy/path change
divergence iteration/depth	First step where intervention path differs	Lower means early structural perturbation; higher means late perturbation
solve status under block	Whether constrained run still reaches terminal proof	Distinguishes robust reroute from true tactic dependency
basin mass + attractor ID	Fraction of seeds ending in each clustered trajectory family	Concentrated mass indicates stable basin; split mass indicates multimodal behavior

K is reported as:

$$K = \log_{10}\left(\frac{\tau_{blind}}{\tau_{agent}}\right)$$

Example calibration: $K = \log_{10}(120/9) = 1.12$ (about 13 × fewer attempts than blind).

Figure 3: K-Metric Visualization

- τ_{agent}: attempted tactic edges until first terminal solve in the observed search graph.
- τ_{blind}: expected attempted edges for a matched blind null policy over the same action surface.
- K: orders-of-magnitude efficiency over blind (`K > 0` is better than blind).

Two related outputs:

- `k_search_efficiency`: trace-derived null model from postprocess.
- `paper_k`: paired blind baseline from basin runs with `--basin-blind`.

1.7. 7. Intervention Protocol

For each theorem, we first solve a wild-type run and extract the solution path $\pi = \{\tau_1, ..., \tau_n\}$. We then run controlled lesions by blocking one tactic (or tactic family) from that path and rerun under the same budget and configuration.

This gives a clean comparison: same theorem, same search budget, one constrained action channel, repeated across all path tactics.

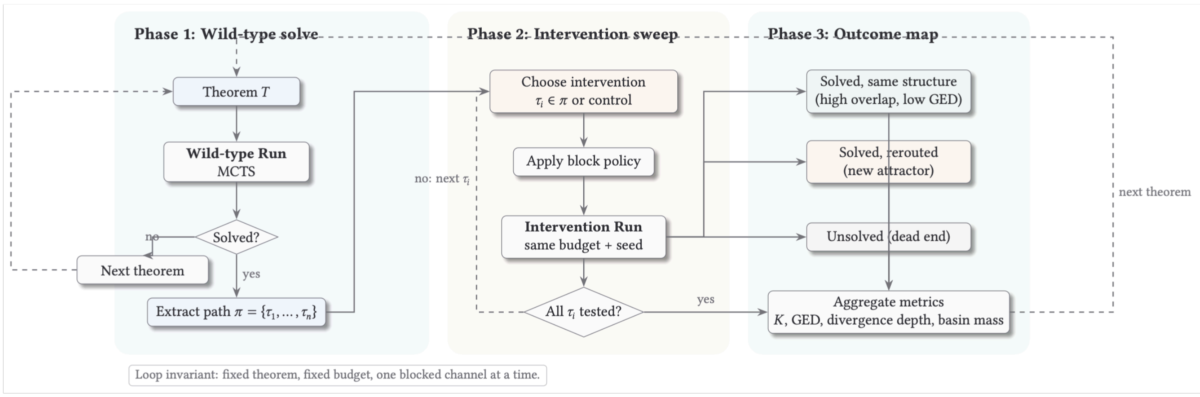

Figure 4: Canonical Loop

1.7.1. How to Read Attractor Analysis

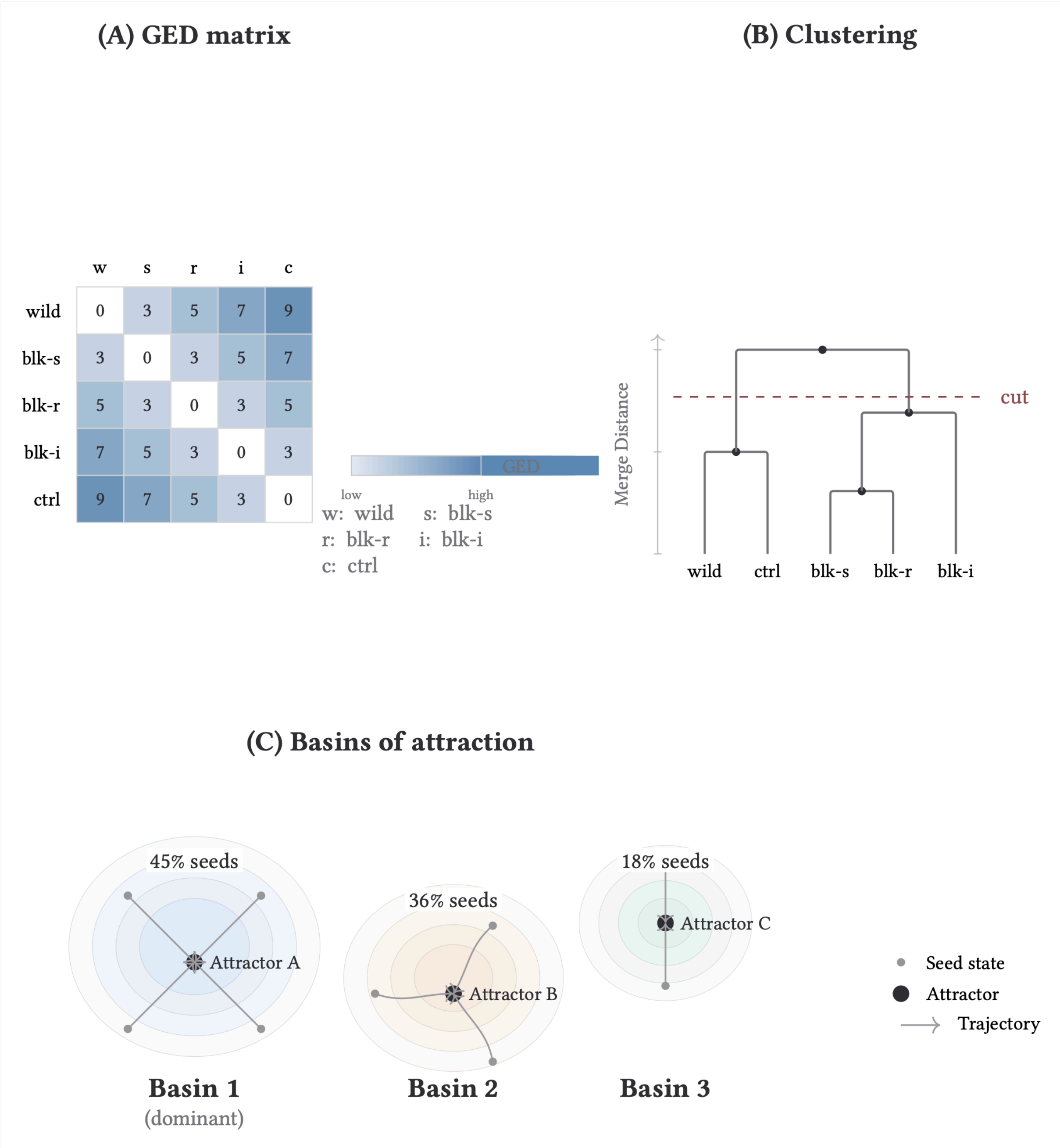

Figure 5: Attractor Analysis

- Panel A (GED matrix): pairwise structural distance between runs.
- Panel B (clustering + cut): where we place the cut determines attractor families.
- Panel C (basins): seed mass captured by each attractor family.

Interpretation: low GED + large shared basin mass implies robust proof structure; high GED with split mass implies genuine rerouting under intervention.

1.8. 8. Log-Derived Vignettes

1.8.1. Vignette Gallery Player

[Interactive element – see web version at specterlab.org/blog/wonton-soup/]