"Only Process Changes"

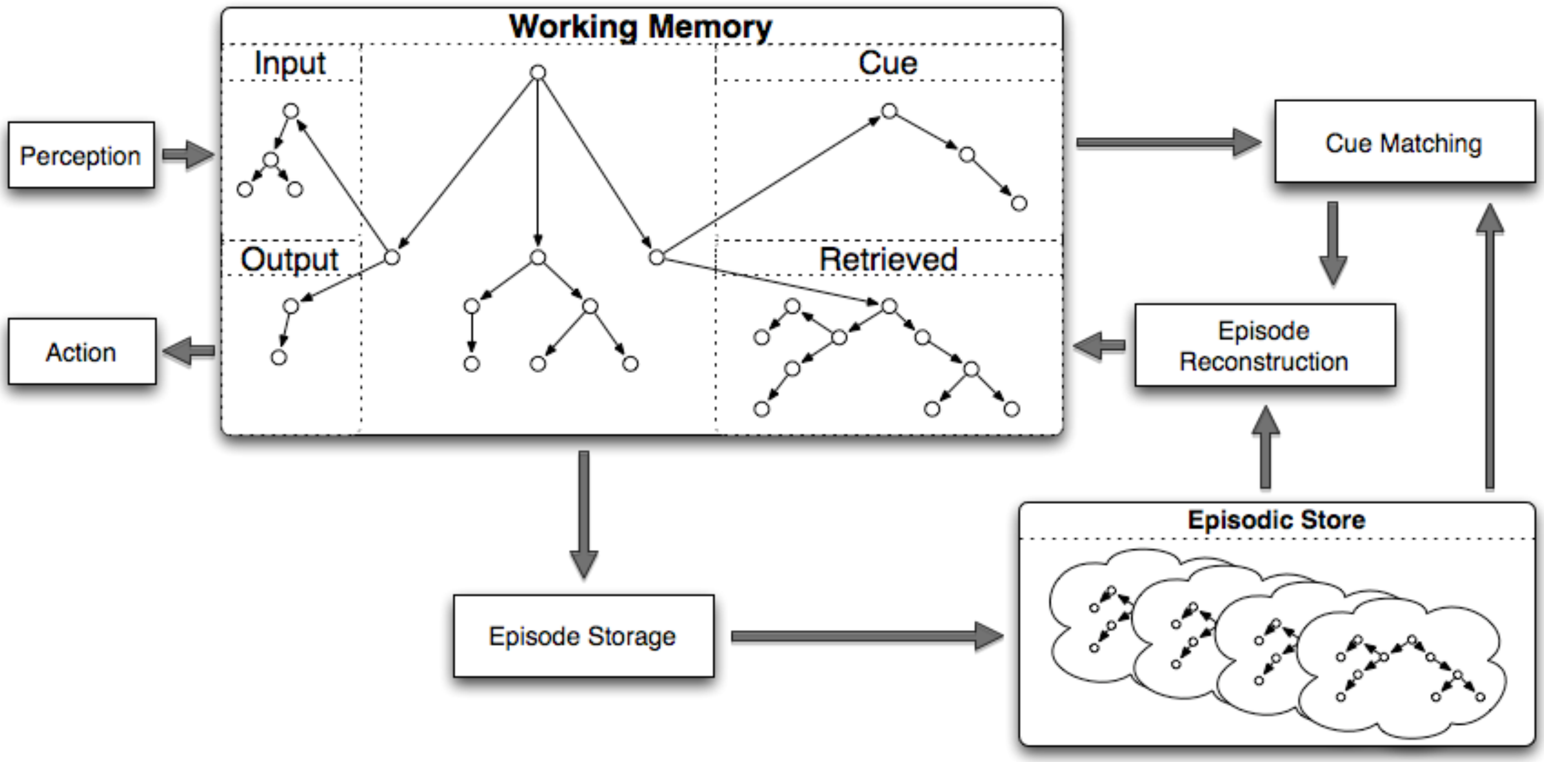
Efficiently Implementing Episodic Memory

Work by Nate Derbinsky & John E. Laird

OUTLINE

- Soar-EpMem: Big Picture
  - Episode Storage
  - Cue Matching
  - Episode Reconstruction
- Summary Numbers
- Current State & Future Directions

SOAR-EPMEM: BIG PICTURE



Episode Storage

- Storage is <u>automatic</u>
  - Frequency set via the `trigger` parameter
    - none, *output*, dc
  - Time set via the `phase` parameter
    - *output*, selection
  - On-demand via `force` parameter
    - *off*, remember, ignore

- Storage is <u>experiential</u>
  - Captures all of top-state
    - Can prevent capture of particular attributes via the `exclusions` parameter
  - Episodes are *temporally* related

Where Do Episodes Go?

- SQLite v3 Relational DB
- Episodic store can be in-memory or on-disk
  - `database` parameter: *memory*, file
  - `path` parameter: *empty*, any file system path
- On-disk database files can be accessed/ manipulated by any SQLite3 client

- On-disk databases are *costly*
  - `commit` parameter controls number of episodes between transactional commit (does not include on-disk journaling)

ARITHMETIC: MEMORY VS. DISK

	Base	Mem, C=1	Mem, C=50k	Disk, C=1	Disk, C=50k
ms/dc	0.081	0.486	0.479	1.243	0.476
kernel (s)	3.419	20.288	19.994	51.893	19.879
RAM (MB)	-	5.38	5.34	2.37	2.36
Disk (MB)	-	-	-	4.5	4.5
		500%	1.4%	156%	2.0%

Mac OS 10.5.6, 2.8GHz, 4GB RAM
DCs: 41,756
Repetitions: 3
watch: 0, no result writes
trigger: dc
srand: 55512

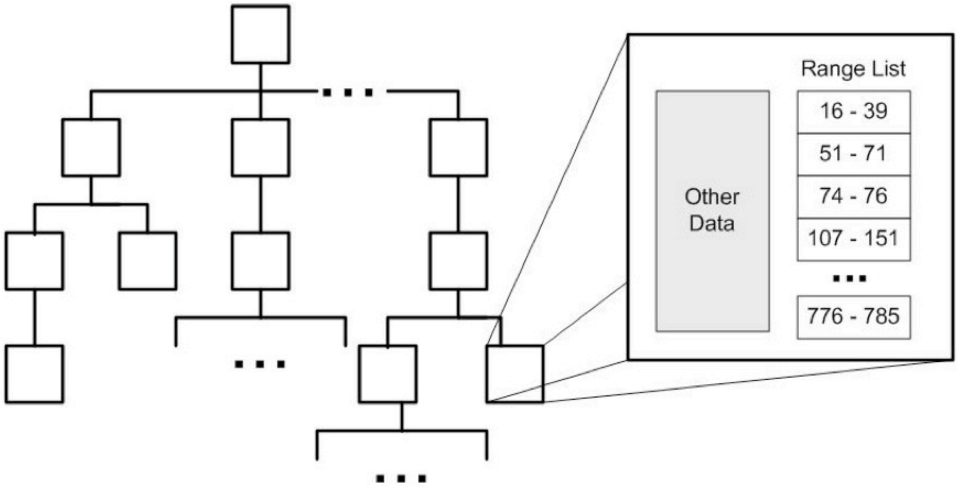
6-Blocks-World: Memory vs. Disk

	Base	Mem, C=1	Mem, C=50k	Disk, C=1	Disk, C=50k
ms/dc	0.099	0.286	0.269	1.425	0.273
kernel (s)	3.941	11.405	10.757	56.995	10.925
RAM (MB)	-	7.75	7.75	2.35	2.36
Disk (MB)	-	-	-	6.7	?
		189%	6%	398%	4.5%

Mac OS 10.5.6, 2.8GHz, 4GB RAM
DCs: 40,000
Repetitions: 3
watch: 0, no result writes
trigger: dc
srand: 55512

What is Stored?

- Episode contents set via `mode` parameter
  - *graph* = full structure
  - tree = Andy's Working Memory Tree
    - No shared identifiers or multi-valued attributes
- Only Process Changes!
  - Keep global record of unique paths
  - Maintain valid temporal ranges



Storage Algorithm

for each WME in WM:

1. if WME points to global structure, ignore
2. else:
   a) if does not exist in global structure, add
   b) point WME to global structure
   c) start new interval in the global structure

The Problem of Unknown Identifiers

- To find a new identifier in global structure = combinatorial deep-structure comparison
  - *I see a car in front of me. Is it the same as any other car I've ever seen? Let's compare color, model, make, year, tires, scratches…*
- Our solution: push to cue matching
  - if: not multi-valued attribute and only ever seen 1, then match
    - *If I've only seen one car in my life, and I only see one now, I have no reason not to believe they are the same.*
  - else: assume new
    - *If I've seen many cars in my life, or I see more than one now, I can't be sure of this car's identity.*

Storage Conclusion

- TankSoar (mapping-bot)
  - ~2500-2600 WMEs/episode
  - 100K episodes = 3.37ms/dc, 260MB
  - 500K episodes = 3.45ms/dc, 1.3GB

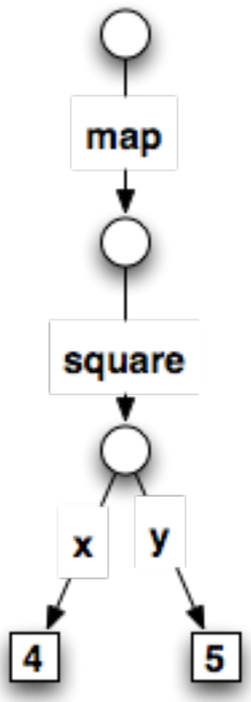
- Assuming about constant episode size, storage is linear in the *changes* in Working Memory

Cue Matching

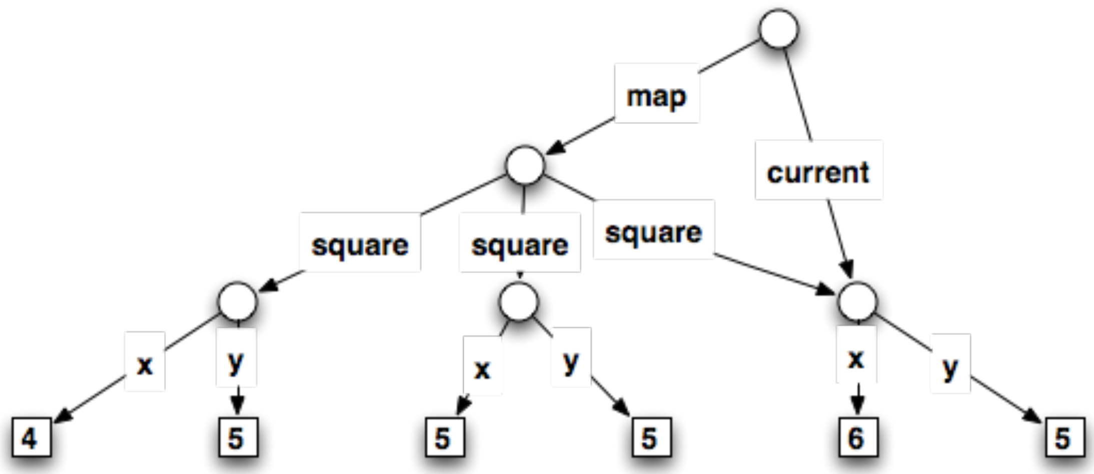
- Input: acyclic cue(s), modifiers (before/after, prohibit)
- 2-Phase Nearest Neighbor Cue Matching
  - Identify candidate episodes based upon *surface* analysis
  - Perform *structural* cue analysis on *perfect surface* matches
- Bias by episode recency

Example

Cue

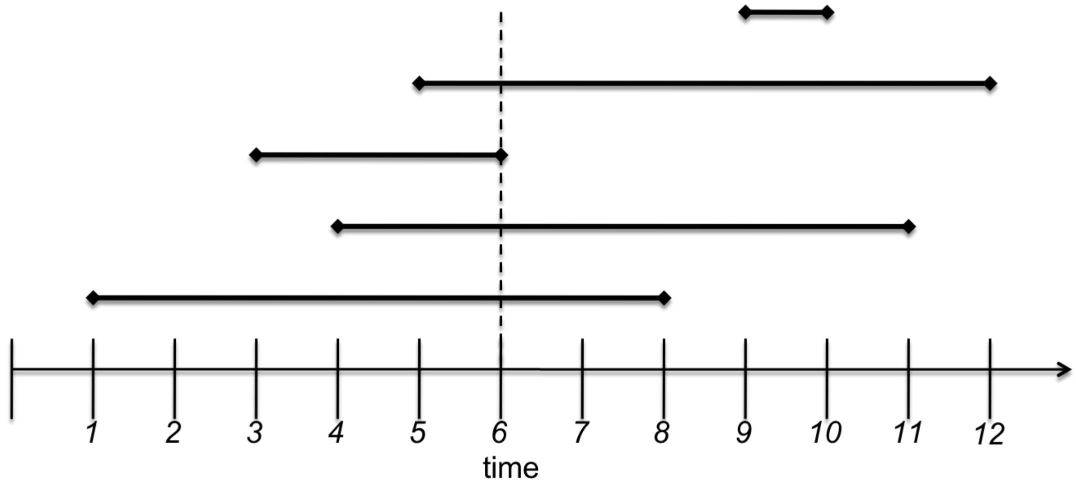


Working Memory

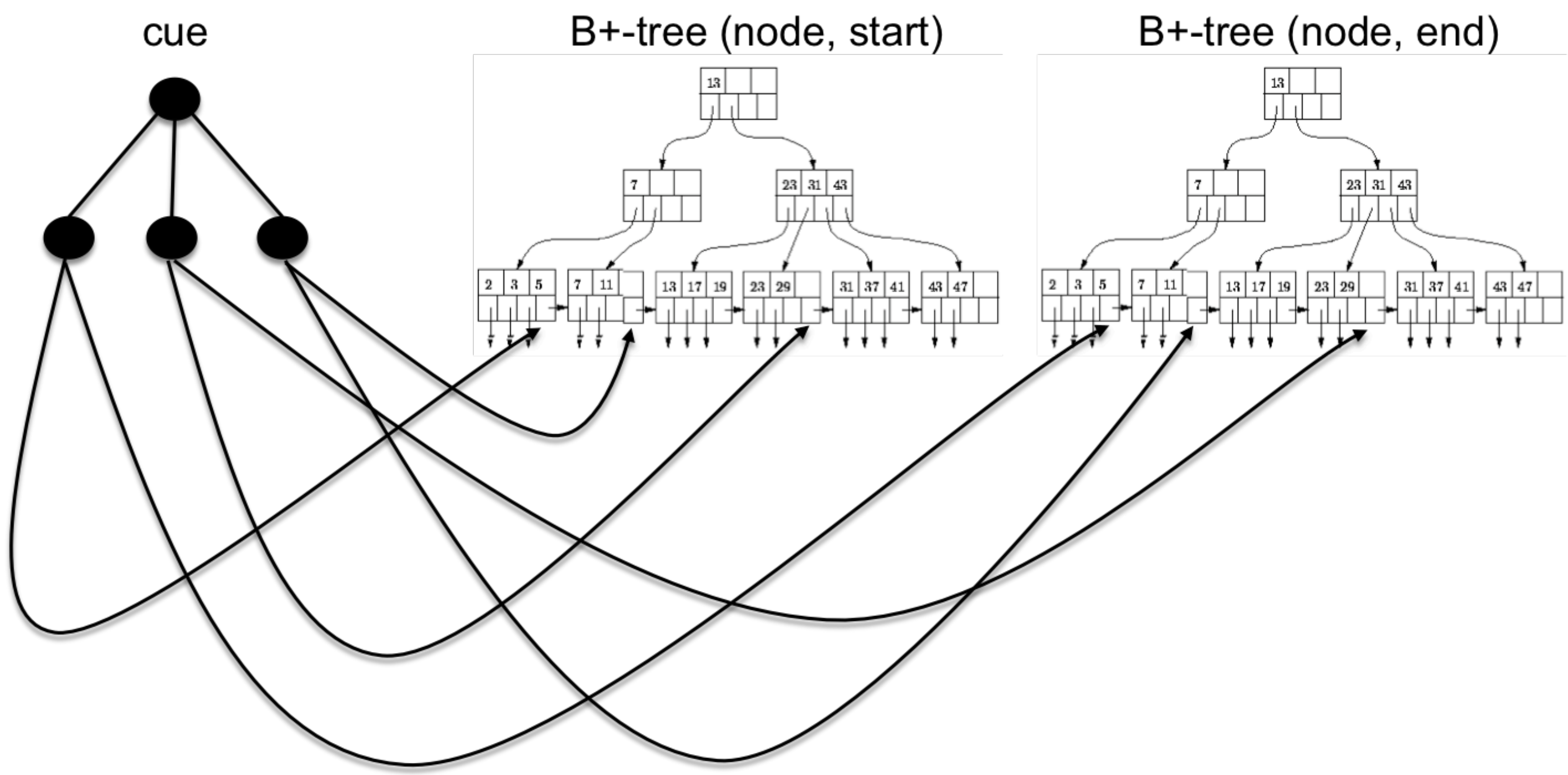


NN: Phase 1

- Match Score = (B)(Cardinality) + (1-B)(Weight)
  - Cardinality = # matching cue leaf WMEs
  - Weight = Working Memory Activation
  - B = `balance` parameter [0, 1]
- Maximize Match Score: Only Process Changes!

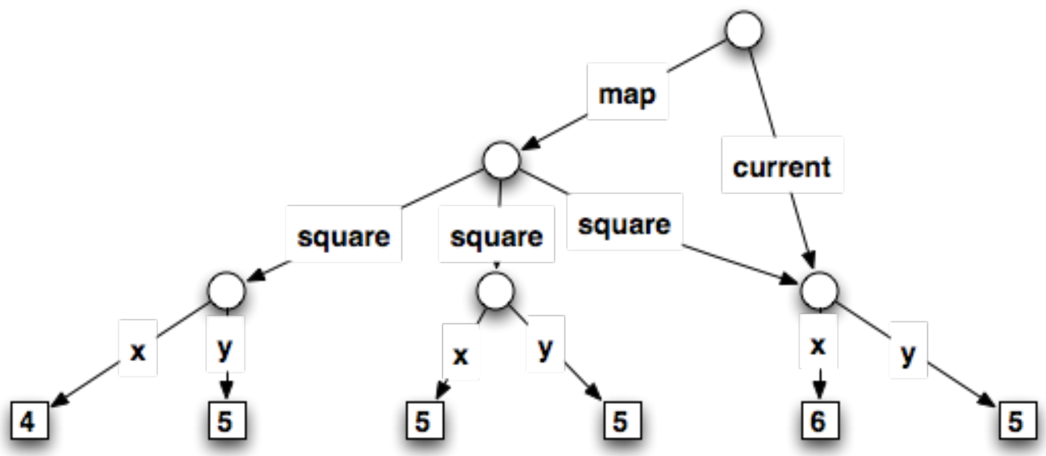


Efficient Interval Search



Cue Leaf Node Ambiguity

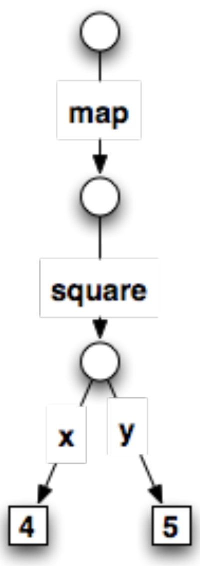
- Because Working Memory is a *graph*, cue leaf nodes do not uniquely identify WMEs of interest
- Simple example: (x=6)

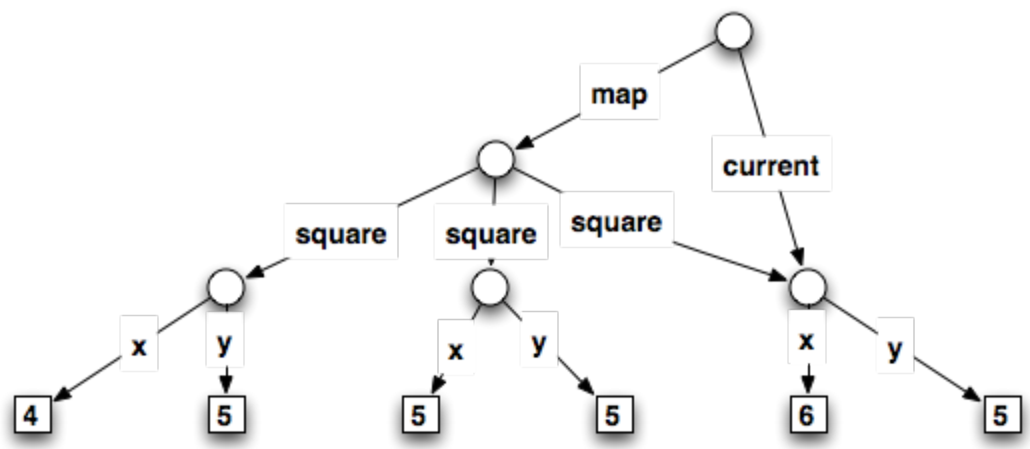


- Path is important!

Addressing Cue Leaf Node Ambiguity

- Cue leaf node paths can be expressed as monotonic, disjunctive normal form (DNF) boolean statements
  - *sat*(x=4) := (root AND map[1] AND square[1] AND x=4[1])
  - *sat*(y=5) := (root AND map[1] AND square[1] AND y=5[1]) OR
    (root AND map[1] AND square[2] AND y=5[2]) OR
    (root AND map[1] AND square[3] AND y=5[3])

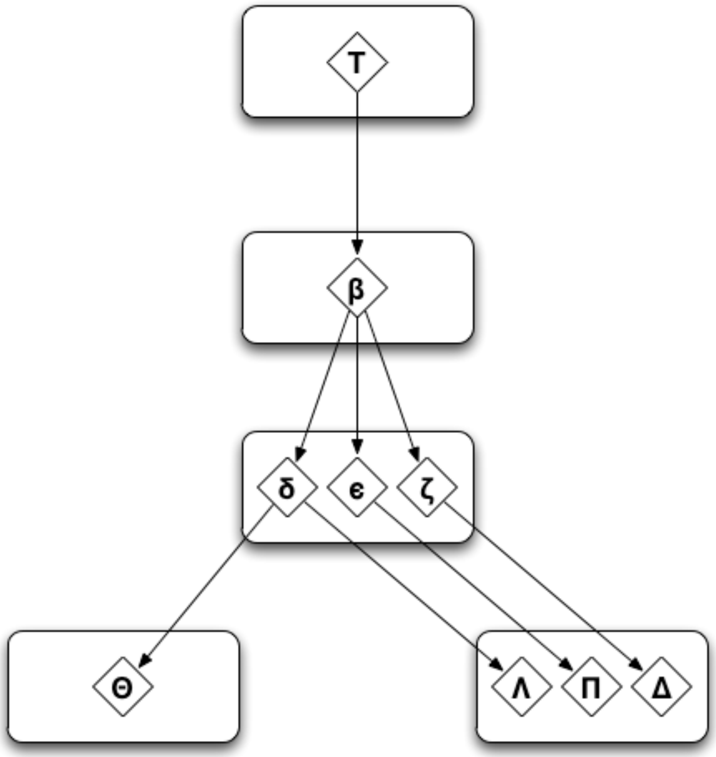




Efficiently Tracking DNF SAT

- Map cue to DNF Graph
- Keep counter at each literal, each clause
- While walking endpoints, propagate along paths (i.e. within clauses)…

Only Process Changes!



Cue Matching Performance Model

- Cue Match = DNF + Interval Search + Graph Match
  - DNF = (A)(log[U * R](L))
    - U = # unique WMEs
    - R = # intervals
    - L = # cue literals
  - Interval Search = (B)(1/T)(Distance)(Δ)
    - T = # episodes
    - Distance = # episodes searched
    - Δ = # cue-relevant intervals
  - Graph Match = CSP backtracking

Cue Matching Performance Issues

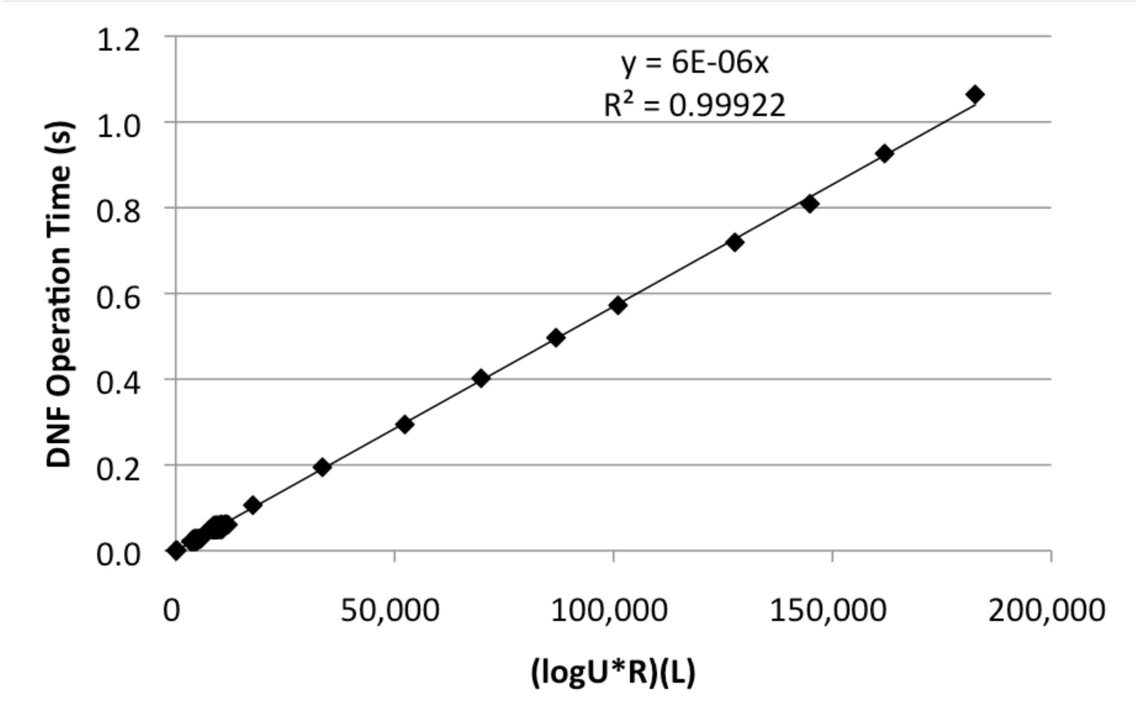
- L = # literals
  - Multi-valued attributes (map squares)
  - Ambiguous blinking identifiers (radar "open")
    - Agent solution: search on elaborations

- Distance = minimal cue co-occurrence
  - Cues with low probability of perfect cardinality can cause *linear* walk of endpoints

- Model predicts performance *linear* with agent changes!

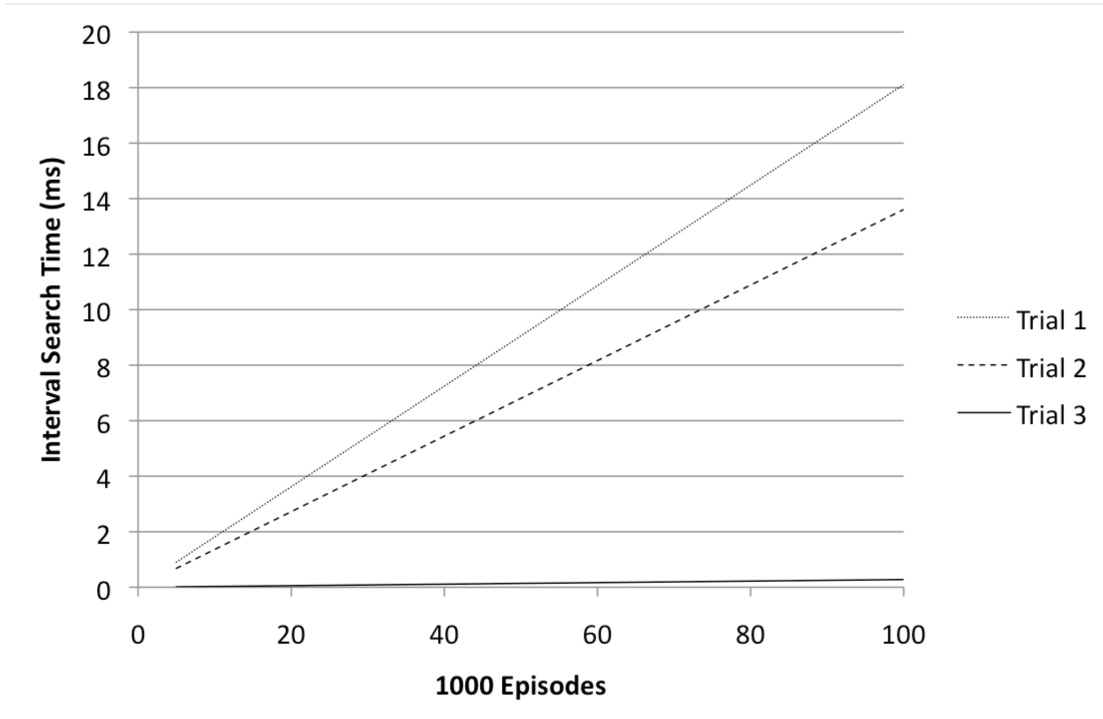
DNF Performance

- Model: 5.69 μ s
- Typical: 0.5ms
- MVA: 15ms
- Radar: 75s



Interval Search Performance

- Model: 1.53 μ s
- Typical: 0.88 μ s
- Distant: 135ms



Graph Match Performance

- Not thoroughly tested
- Informally
  - Typical: 2.1% of total cue matching time
  - Worst: 21%

Episode Reconstruction

- Used following successful cue matching or `retrieve/next/previous`
- Collecting episode WMEs from a set of intervals is an *interval intersection query*
  - *Find me all intervals that started before and ended after time t*
- We implement a Relational Interval Tree
  - Maps an Interval Tree onto RDBMS b+-trees and SQL queries
  - Intersection queries are log(# intervals)

Reconstruction Performance Model

- Reconstruction = RI-tree + Collect + Add
  - RI-tree = (C)(logR) ~ 76 μ s
    - R = # intervals
  - Collect = (D)(logU)(M) + (E)(M) ~ 22.4ms
    - U = # unique WMEs
    - M = # WMEs in episode
  - Add = (F)(M) ~ 1.29ms
    - M = # WMEs in episode
  - In our experiments, M dominates!
    - Assuming constant M, grows with *changes*!

Summary Numbers: mapping-bot

Episodes	Storage	Cue Matching	Reconstruction
100K	3.37ms, 260MB	43.9ms	23.8ms
500K	3.45ms, 1.3GB	64ms*	26.3ms

* If distant cue retrievals are omitted, cue matching time is indistiguishable.

Concluding Tips

- Don't change recorded deep structure
  - And definitely don't query!
  - Exclusions and elaborations are your friend!
- Graph matching returns meta-data
  - 0/1 = unified with cue
  - *mapping* = first cue/retrieval WME association
- Timers!
  - Can help debug slow performance, but costly
  - Implemented in levels to ameliorate cost vs. benefit

ISSUES

- Ref counts @ top-state
  - Can cause seg-fault in deep stack
  - Needed for WMA across WMEs in back-trace?

- Acceptable preference WMEs
  - What *could* I have done?
  - Causes linear MVA storage/cue matching explosion

ISSUES

- Structural analysis of imperfect candidates
  - *Get a good dissimilar-match, or a bad similar-match?*
- Unbounded search
  - Via cue or environment/agent
  - Explore: effort-bounding, heuristics
- Limited evaluation
  - Need longer, more diverse & calculated experimentation

To Do

- ICCBR 2009 (submitted)

- Soar 9.1.1-beta
  - Updated manual
  - Tutorial