Extended Action Spaces for MCTS-Based LLM Reasoning: A Research Roadmap

Working Draft

January 23, 2026

Abstract

This working paper explores research directions for extending Monte Carlo Tree Search (MCTS) applied to Large Language Model reasoning. While the canonical MCTS-Reasoning specification uses only the CONTINUE action, richer action spaces may improve reasoning efficiency and quality. We discuss state management actions (COMPRESS, VERIFY, BACKTRACK), problem decomposition patterns, graph-based reasoning structures, and algorithm variants. This document serves as a living roadmap for future development rather than a specification of implemented functionality.

Contents

1 Introduction

The canonical MCTS-Reasoning specification [1] defines a single action: Continue, which prompts the LLM to generate the next reasoning step. While simple and effective, this minimal action space limits the search algorithm's ability to manage complex reasoning patterns.

1.1 Motivation

Several limitations of the single-action approach motivate richer action spaces:

1. **Unbounded State Growth**: Reasoning traces accumulate text without bound, eventually exceeding context limits
2. **No Explicit Verification**: Self-checking happens only if the LLM spontaneously includes it
3. **Implicit Backtracking**: Poor paths are deprioritized by UCB1, but never explicitly abandoned
4. **Sequential Decomposition**: Complex problems must be solved step-by-step within a single path

1.2 Scope of This Document

This working paper sketches potential extensions without claiming they are implemented or fully specified. The goal is to:

- Document promising research directions
- Compare with related work in structured reasoning
- Identify open questions requiring further investigation
- Guide future development priorities

2 Related Work

Recent work has explored various approaches to structured reasoning with LLMs.

2.1 Tree of Thoughts

Yao et al. [2] introduced Tree of Thoughts (ToT), which explores multiple reasoning paths through deliberate problem decomposition. Key differences from MCTS-Reasoning:

- ToT uses breadth-first or depth-first search rather than MCTS
- Evaluation can occur at intermediate states, not just terminal states
- "Thoughts" are explicitly structured rather than free-form continuations

2.2 Graph of Thoughts

Besta et al. [3] extend ToT to graph structures (GoT), allowing:

- Aggregation of multiple reasoning paths
- Refinement loops that revisit earlier thoughts
- Non-linear reasoning topologies

GoT demonstrates that tree structures may be insufficient for complex reasoning, motivating our discussion of graph-based extensions in Section 5.

2.3 Reasoning via Planning

Hao et al. [4] apply MCTS to reasoning with an explicit world model:

- States are structured representations, not just text
- Actions include domain-specific operators
- Rewards come from a learned value function

This approach trades generality for domain-specific efficiency.

2.4 Self-Consistency

Wang et al. [5] demonstrate that sampling multiple reasoning paths and voting among answers improves reliability. The canonical MCTS-Reasoning already implements self-consistency via `PathSampler`, but deeper integration with the search process remains unexplored.

3 State Management Actions

State management actions help control reasoning trace length and quality.

3.1 Compress Action

When reasoning traces become long, the Compress action summarizes the trace into a more compact representation:

$$\textsc{Compress}(s) = \text{LLM}_{\text{summarize}}(s)$$

3.1.1 Action Space Extension

$$\mathcal{A}(s) = \begin{cases} \{\textsc{Continue}, \textsc{Compress}\} & \text{if } |s| > \text{threshold} \\ \{\textsc{Continue}\} & \text{otherwise} \end{cases}$$

3.1.2 Design Considerations

- **Information Loss**: Compression inevitably loses detail; when is this acceptable?
- **Threshold Selection**: Fixed threshold vs. adaptive (based on problem complexity)?
- **Compression Quality**: How to ensure key insights are preserved?
- **Backpropagation**: Should compressed states receive different value updates?

3.1.3 Implementation Sketch

The `ExtendedActionSpace` class in the canonical implementation provides a prototype:

```
action_space = ExtendedActionSpace(
    generator=generator,
    llm=llm,
    compress_threshold=2000  # characters
)
```

Remark 3.1 (Partial Implementation)**.** *While `ExtendedActionSpace` and `CompressAction` exist in the codebase, they are not extensively tested and the action selection policy (when to compress vs. continue) remains simplistic.*

3.2 Verify Action

The VERIFY action asks the LLM to check the current reasoning before continuing:

$$\textsc{Verify}(s) = s \oplus \text{LLM}_{\text{verify}}(s)$$

The verification might:

- Confirm the reasoning is correct (continue normally)
- Identify errors (potentially triggering backtrack)
- Suggest corrections (modify the state)

3.2.1 Open Questions

- When should verification be triggered? (Every k steps? When confidence drops?)
- How should verification failure affect UCB1 values?
- Should verification results influence sibling nodes?

3.3 Backtrack Action

The BACKTRACK action explicitly marks a reasoning path as unpromising:

$$\textsc{Backtrack}(s) = \text{TERMINAL}_{\text{failed}}$$

Unlike natural search abandonment (where UCB1 simply deprioritizes low-value nodes), explicit backtracking:

- Immediately terminates exploration of this branch
- Can propagate negative signal to parent nodes
- Frees resources for more promising alternatives

3.3.1 Triggering Conditions

Backtracking might be appropriate when:

- The LLM explicitly states the approach won't work
- Verification repeatedly fails
- Resource limits (depth, tokens) are reached
- A contradiction is detected

4 Problem Decomposition

Complex problems often benefit from decomposition into subproblems. This section sketches how MCTS might support explicit decomposition.

4.1 Decompose Action

A Decompose action splits the current problem into independent subproblems:

$$\textsc{Decompose}(s) = \{s_1, s_2, \ldots, s_k\}$$

where each s_i is a subproblem state that can be solved independently.

4.1.1 Structural Implications

Decomposition fundamentally changes the search structure:

- The node has multiple "children" that must *all* be solved (AND node)
- Standard MCTS children represent *alternatives* (OR node)
- Combining solutions requires an Aggregate operation

This creates an AND-OR tree or DAG structure, not a simple OR tree.

4.2 Recursive MCTS

One approach: run independent MCTS searches for each subproblem:

Algorithm 1 Recursive MCTS for Decomposition (Sketch)

```
Require: State $s$ to decompose
Ensure: Combined solution
 1: procedure SolveWithDecomposition($s$)
 2:     if IsSimple($s$) then
 3:         return MCTS($s$)                          ▷ Base case: standard MCTS
 4:     end if
 5:     $\{s_1, \ldots, s_k\} \leftarrow$ Decompose($s$)
 6:     for $i = 1$ to $k$ do
 7:         $\text{result}_i \leftarrow$ SolveWithDecomposition($s_i$)      ▷ Recursive
 8:     end for
 9:     return Aggregate($\text{result}_1, \ldots, \text{result}_k$)
10: end procedure
```

4.2.1 Open Questions

1. **Backpropagation**: How should values from subproblem MCTS searches propagate to the decomposition node?
2. **Resource Allocation**: How many simulations for each subproblem?
3. **Failure Handling**: What if one subproblem cannot be solved?
4. **Decomposition Quality**: How to evaluate whether a decomposition is good?
5. **When to Decompose**: How does the search decide between Continue and Decompose?

4.3 Aggregation

Combining subproblem solutions is non-trivial:

- Simple concatenation may miss interactions
- The aggregation step itself may require reasoning
- Inconsistencies between subproblem solutions must be resolved

An Aggregate action might be another LLM call:

$$\textsc{Aggregate}(r_1, \ldots, r_k) = \text{LLM}_{\text{combine}}(r_1, \ldots, r_k, s_{\text{original}})$$

5 Beyond Trees: Graph-Based Reasoning

The tree structure of MCTS imposes fundamental constraints on reasoning patterns. Certain operations naturally suggest non-tree structures.

5.1 Limitations of Tree Structure

Trees enforce:

- **Single Parent**: Each state has exactly one predecessor
- **No Cycles**: Cannot revisit earlier states
- **Independent Branches**: No information sharing across paths

These constraints conflict with natural reasoning patterns:

5.1.1 Problem Decomposition

Splitting a problem into independent subproblems, solving each separately, and combining results creates a DAG structure where the "combine" node has multiple parents (the solved subproblems).

5.1.2 Critique-Revision Cycles

Evaluating a solution, identifying issues, and refining creates potential cycles that trees cannot represent. In a tree, we can only approximate this by continuing forward.

5.1.3 Cross-Path Information Sharing

Using insights discovered in one reasoning path to inform another requires edges between branches—impossible in a tree.

5.2 Directed Acyclic Graphs (DAGs)

DAGs relax the single-parent constraint while maintaining acyclicity:

- Multiple paths can converge to the same insight
- Decomposition and aggregation are naturally represented
- Backpropagation becomes more complex (multiple paths to root)

5.2.1 DAG Backpropagation

With multiple parents, value updates must handle:

- Credit assignment: which parent gets credit for a good outcome?
- Visit counting: how to count visits with multiple paths?
- Convergence: ensuring values stabilize despite cycles in the update graph

5.3 Hypergraph Structures

Hypergraphs generalize graphs by allowing edges to connect more than two nodes:

- A Decompose action creates a hyperedge from one node to multiple children
- An Aggregate action creates a hyperedge from multiple nodes to one
- Standard Continue remains a binary edge

This provides a unified representation for all action types but requires:

- Generalized UCB1 for hyperedge selection
- Synchronization semantics (when are all subproblems "ready"?)
- More complex backpropagation algorithms

Remark 5.1 (Natural Decomposition within MCTS)**.** *Note that the LLM within a Continue action can naturally perform soft versions of these operations: reasoning about subproblems sequentially within a single path, self-critiquing before continuing, or reconsidering earlier steps. The tree structure captures* alternative *reasoning paths, while complex reasoning* within *a path remains expressible through the continuation mechanism.*

Graph structures become necessary only when we want the search algorithm *to explicitly manage decomposition, aggregation, and cross-path communication.*

6 Algorithm Variants

Several MCTS variants may improve performance for LLM reasoning.

6.1 Progressive Widening

Instead of a fixed branching factor B, progressive widening adjusts B based on visit count:

$$B(n) = \lceil k \cdot \nu(n)^{\alpha} \rceil$$

where k, α are parameters. This allows:

- Early exploration with few children
- More alternatives for promising nodes
- Adaptive resource allocation

6.2 RAVE (Rapid Action Value Estimation)

RAVE shares value information across the tree based on action similarity. For LLM reasoning, this might mean:

- Similar reasoning patterns are valued similarly
- Requires defining "action similarity" for text continuations
- May help with sparse rewards

6.3 Parallel MCTS

Parallelization strategies for MCTS:

- **Root Parallelization**: Run independent trees, combine results
- **Leaf Parallelization**: Parallel rollouts from same node
- **Tree Parallelization**: Concurrent tree modifications with virtual loss

For LLM reasoning, root parallelization is simplest and naturally combines with self-consistency voting.

6.4 Learned Components

6.4.1 Value Networks

Learn to evaluate intermediate states:

$$V(s) = \text{Network}(s)$$

This allows evaluation before reaching terminal states, potentially accelerating search. Challenges:

- Training data: which intermediate states are "good"?
- Generalization: will the value network transfer across problem types?
- Integration: how to combine learned values with terminal evaluations?

6.4.2 Policy Networks

Learn to prioritize actions/continuations:

$$P(a|s) = \text{Network}(s, a)$$

This guides expansion toward promising regions. Challenges:

- The action space is infinite (all possible text continuations)
- Must integrate with LLM generation, not replace it
- May limit exploration if policy is overconfident

6.5 Beam Search Hybrid

Combine MCTS exploration with beam search efficiency:

- Use beam search for rapid generation of candidates
- Use MCTS for deep exploration of promising candidates
- Maintain beam-width top nodes, but allow UCB1 selection within beam

7 Open Questions

This section collects fundamental questions requiring further investigation.

7.1 Action Selection

1. How should the search algorithm choose between Continue, Compress, Verify, and Decompose?
2. Should action selection be learned or heuristic?
3. How do we balance action space richness against search efficiency?

7.2 Decomposition

1. How should backpropagation work from recursive MCTS searches?
2. What determines whether decomposition improves or harms performance?
3. How to detect when decomposition is appropriate?
4. How to aggregate solutions with conflicting assumptions?

7.3 Graph Structures

1. When are graph structures worth the added complexity?
2. How to extend UCB1 for DAG or hypergraph structures?
3. What synchronization semantics for multi-parent aggregation?
4. How to prevent cycles in revision/critique patterns?

7.4 Learning

1. What training signal for intermediate state value estimation?
2. How to learn action selection policies over infinite action spaces?
3. Can value/policy networks transfer across reasoning domains?
4. How much training data is needed for reasonable performance?

7.5 Evaluation

1. What benchmarks best test extended action spaces?
2. How to measure the benefit of decomposition vs. continued reasoning?
3. What metrics capture reasoning quality beyond answer correctness?

8 Conclusion

This working paper sketches research directions for extending MCTS-based LLM reasoning beyond the canonical single-action specification. Key themes include:

1. **State Management**: Actions like COMPRESS, VERIFY, and BACKTRACK help manage reasoning trace length and quality
2. **Problem Decomposition**: Explicit DECOMPOSE and AGGREGATE actions may handle complex problems more efficiently than sequential reasoning
3. **Beyond Trees**: DAG and hypergraph structures can represent reasoning patterns that trees cannot
4. **Algorithm Variants**: Progressive widening, parallel MCTS, and learned components offer performance improvements

These ideas remain research directions rather than implemented features. The canonical MCTS-Reasoning specification provides a solid foundation; the extensions described here represent potential future evolution as understanding of LLM reasoning deepens.

References

[1] MCTS-Reasoning Technical Report v0.5. *Canonical Specification of Monte Carlo Tree Search for LLM Reasoning.*

[2] Yao, S., et al. (2023). Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601.*

[3] Besta, M., et al. (2023). Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687.*

[4] Hao, S., et al. (2023). Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992.*

[5] Wang, X., et al. (2023). Self-consistency improves chain of thought reasoning in language models. *ICLR 2023.*

[6] Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. *European Conference on Machine Learning*, 282–293.

[7] Browne, C. B., et al. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1–43.