MCGS: A Minimax-Based Combinatorial Game Solver

Taylor Folkersen, Haoyu Du, Martin Müller

University of Alberta

Motivation

- MCGS: A framework for solving (sums of) *short* 2 player games
  - “Short” games are finite and loop-free
  - Given a sum of games and first player, find the winner
  - “Normal play” convention: last player to move wins
- Existing CGT tools: CGSuite
  - Lots of features
  - Based on canonical forms. Very slow!
- Example: empty 1x16 NoGo board
  - CGSuite: finds canonical form in almost 8 minutes
  - MCGS: finds outcome class in ~1.6 seconds
    - Without transposition table or database enabled!

Motivation (Continued)

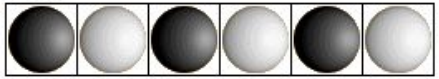
> G1

*

> G2

+-{*,^}

> G3



0

> G4

+-{^[2],{^*,^^|0,^*,+-{0,^*}}}

> G5

+-{^<2>*,^,{0|^,+-{*,^}},{^^*|v,+-{*,^}}}

> G6

+-{0,^*}

Motivation (Continued)

> G7

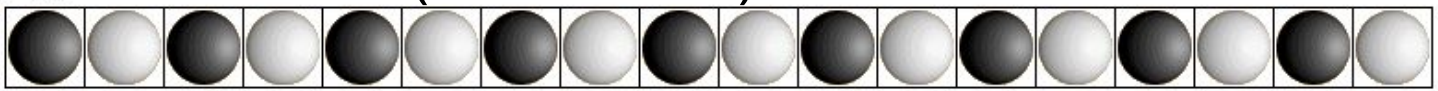
+-{{^[2]*,^^,{0|{0|^*,+-{0,^*}},{^*,^^|0,^*,+-{0,^*}}},{^3*|0,{^*,^^|0,^*,+-{0,^*}}}|0,{^^|0,^*},{0,{^*,^^|0,^*,+-{0,^*}}||v*,+-{0,^*}|0},+-{^<2>,^*,{0|^*,+-{0,^*}},{^^|v*,+-{0,^*}}},{^^,{^*,^^|0,^*,+-{0,^*}}|0,v*}},{{^^*|^},{^3*|0,{^*,^^|0,^*,+-{0,^*}}}|0,*,{0,*,{^,^*|0,v*}|vv,v*}},{0|{0|^*,+-{0,^*}},{^*,^^|0,^*,+-{0,^*}}||+-{^[2],{^*,^^|0,^*,+-{0,^*}}},{^,^^*|*,^,+-{*,^}||{0,v*,+-{0,^*}|vv,v*},+-{0,^*}}},{0|{0|^*,+-{0,^*}},{^*,^^|0,^*,+-{0,^*}}||{0|^,+-{*,^}||*,{*,v,+-{*,^}|vv*,v}},{0|^*,+-{0,^*}||0,{0,v*,+-{0,^*}|vv,v*}},{+-{0,^*},{^*,^^|0,^*,+-{0,^*}}||*,v,+-{*,^}|vv*,v}},{{0|+-{0,^*}},{^^*,{^^|^*}|^,^*,{^,^[2]*|0,*,{0,^*|v,v*}},{^*,^^|0,*,{0,^*|v,v*}}}|+-{0,^*},{*,^<2>,{^,^[2]*|0,*,{0,^*|v,v*}}|v,v*,{0,v*,{0,^*|v,v*}|vv,vv*}},+-{*,^},{+-{0,^*},{^*,^^|0,^*,+-{0,^*}}||*,v,+-{*,^}|vv*,v}}}

> G8

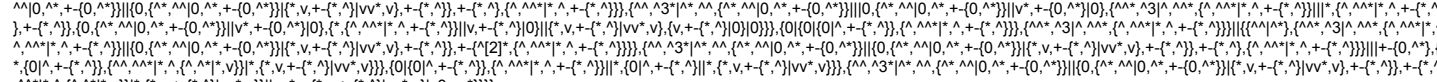
+-{{^,^^*|*,^,+-{*,^}},{^^*|+-{^<2>*,^,{0|^,+-{*,^}},{^^*|v,+-{*,^}}},{^[2]*,{0|^,+-{*,^}},{^^,^^*|*,^,{^,^*|*,v}}|*,{*,v,+-{*,^}|vv*,v}}}}

Motivation (Omitted)

> G9



> G10



MCGS

- Goal: create an efficient, general framework for solving games
  - Apply lessons learned from previous specialized solvers
    - Linear NoGo and NoGo (Du et al. 2023, 2024)
    - Linear Clobber (Tavakoli et al. 2022, Folkersen et al. 2022, 2025)
  - Implement rules of a game, benefit from MCGS's search engine
    - Easier than writing a game-specific solver from scratch
  - C++ source code
- Support for both *partizan* and *impartial* games
  - Partizan games: boolean minimax search to find winner
  - Impartial games: specialized algorithms to find nim-value
  - Wrapper creates impartial versions of any implemented game

MCGS (Continued)

- Test framework
  - “.test” input files: specify positions as sums of subgames, with first players and expected results
  - .csv/.html output
    - Search results
    - Search statistics: time, search node count, etc.
  - Thousands of test cases
    - Many verified by external solvers (i.e. CGSuite, game-specific solvers)
- Documentation
  - For general users
  - For developers
    - Instructions for adding new games
    - Comprehensive discussion of implementation details

Supported Games

- “Basic” CGT games:
  - Integer
  - Dyadic Rational
  - Nimber
  - Switch
  - Up/Star
- Strip (1D) games:
  - Linear Clobber
  - Linear NoGo
  - Elephants and Rhinos
  - * Generalized Toads and Frogs
  - * Toppling Dominoes
- Grid (2D) games:
  - Clobber
  - NoGo
  - * Amazons
  - * Fission
  - * Domineering
  - * Battle Sheep
- Others:
  - Kayles

* Denotes games coming in future releases

Hashing

- 64 bit Zobrist hashes
  - Used to index into transposition table and endgame database
- Two-part hashing scheme
  - Local hash h (one subgame)
  - Global hash H (entire sum)
    - Computed from local hashes (next slide)
- Zobrist table Z
  - Table of random 64 bit integers
  - Indexed by pair: (location, color)
  - Specialized solver may only represent 3 colors (Clobber, NoGo)
  - MCGS's Zobrist table represents all 128 bit color values

h(Clobber:■□_■) := Z(0,■) ⊕ Z(1,□) ⊕ Z(2,_) ⊕ Z(3,■) ⊕ gametype(Clobber)

Global Hash

- Compute global hash H(S, P) for sum S and player P:
  a. Sort games of sum: S = g0 + g1 ... + gn
     - Sums differing only in order should have the same global hash
     - H(g0 + g1, P) = H(g1 + g0, P)
  b. Compute list of local hashes: [h(g0), h(g1), ..., h(gn)] = [h0, h1, ..., hn]
  c. H(S, P) := Z_sum(0, h0) ⊕ Z_sum(1, h1) ⊕ ... ⊕ Z_sum(n, hn) ⊕ player(P)
     - Transform local hashes using another Zobrist table Z_sum
     - Two-level re-coding scheme prevents contents of local hashes cancelling out
     - Yields different hashes even for sums with thousands of copies of one game

Game-specific Functions

- Abstract base type *game*
- Required functions (each new game must implement):
  - Play and undo move
  - Compute local hash
  - Print (produces textual representation of the game)
  - Create move generator
    - Move generators produce all moves for a subgame

Game-specific Functions (Continued)

- Optional functions:
  - Split
    - Break game into subgames
    - Clobber:■□_■■□ = Clobber:■□ + Clobber:■■□
    - Clobber:■□_■■□ = * + ↑
      - Takes advantage of “basic” CGT game optimizations
  - Normalize
    - Convert equivalent games to one common, unique representation
    - Clobber:■■□__ = Clobber:□■■
  - Split and normalize functions increase transposition table hits
- Optional incremental hash update
  - Update local hash when changing a subgame’s state
  - Avoids recomputing local hash in its entirety

Database (DB)

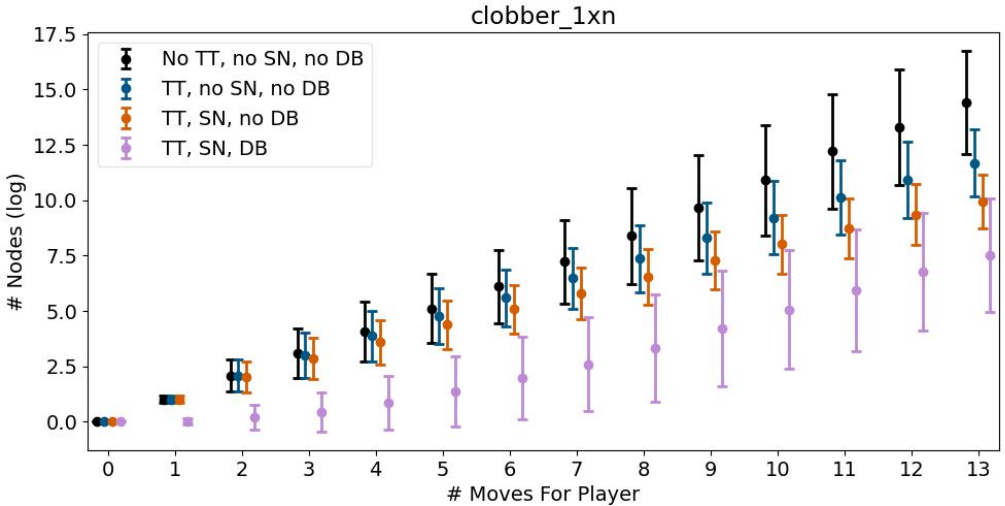
- Stores pre-computed information about subgames
  - DB entries contain outcome classes
  - Assists with sum simplification, and earlier search termination
  - Currently: contains single subgames, which are normalized and don't split
    - Reduces size of DB
    - No sums yet
  - Queried using local hashes
- DB creation uses abstract base type *database game generator*
  - Generates subgames of some game type, for the DB to create their entries
  - Generates in order of increasing "size"
    - Ensures DB creation is fast: after 1 move on generated game, resulting subgames are in DB
  - Modular design with reusable components
    - Easy to add database support for a game type

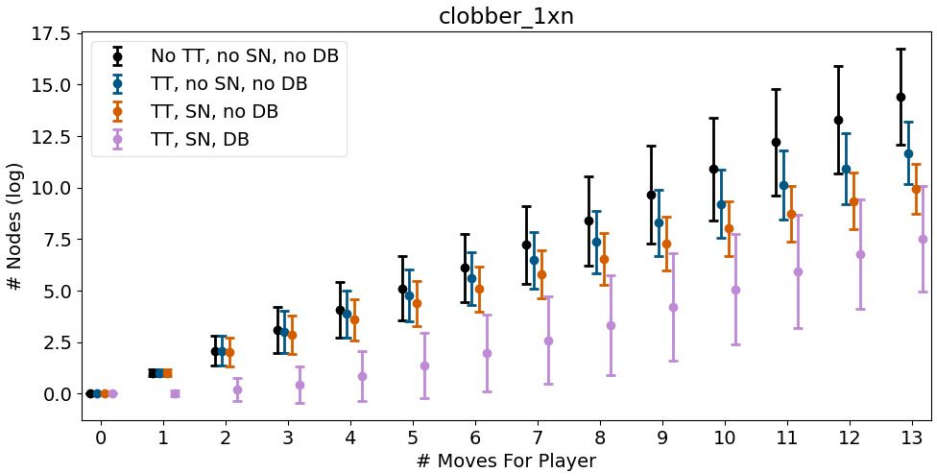
Sum Optimizations

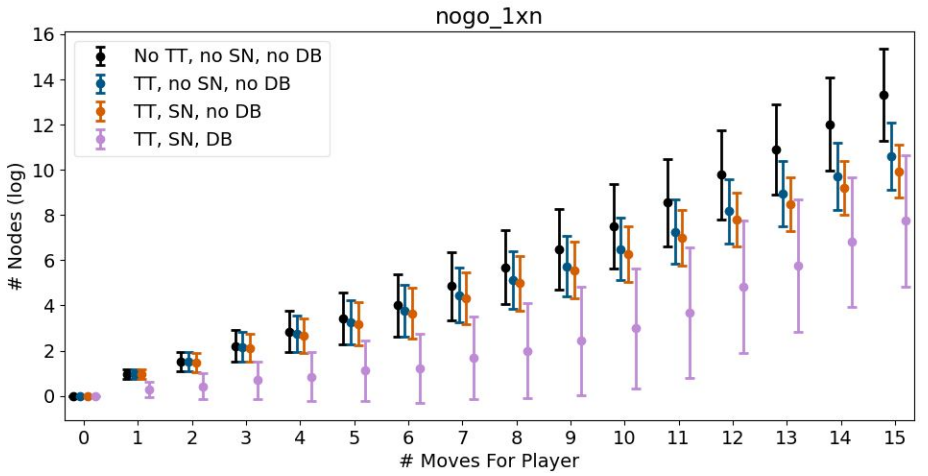
- Basic CGT game simplification
  - Switches: {A | B} for rationals A and B
    - Extract mean: {4 | -8} = -2 + {6 | -6}
    - Convert to number: {2 | 3} = 5/2
  - Sum up integers and rationals:
    - 1 + 2 + 1/2 = 7/2
  - Nim-add nimbers:
    - *3 + *2 = *1
  - Sum together multiples of ↑ and ↓:
    - 2↑ + 3↓ = ↓
- Look up subgames in database:
  - Prune games with outcome class P
  - Sometimes: outcome classes determine result without further search
- After playing a move on a subgame, use its split and normalize functions
  - Increase transposition table hits

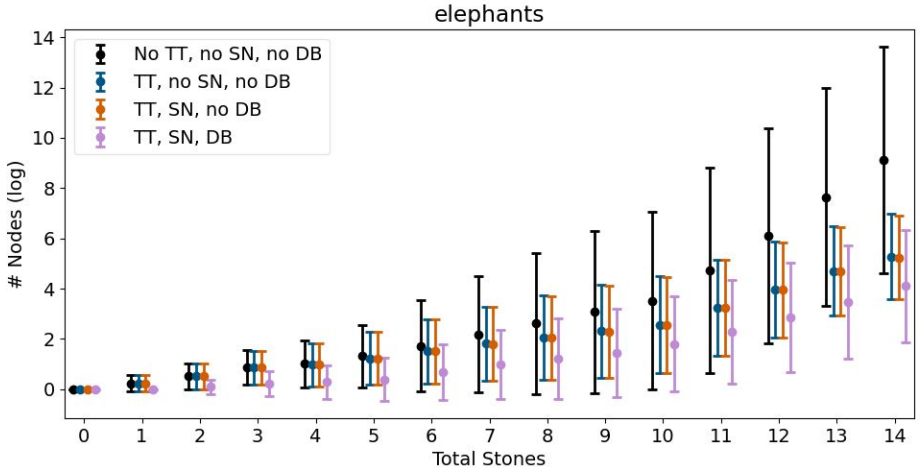
Experiments

- Scaling results for several game types
  - Generate random games, sort into buckets (of up to 2000 games) along X axis
  - Y axis: (natural log of) search node counts
    - Node count: Invocations of recursive minimax search function
  - Same games across 4 ablations, changing 3 variables:
    - TT (transposition table)
    - SN (split and normalize)
    - DB (database)
  - 60 second timeout
    - A game which times out for any ablation setting is excluded from the data

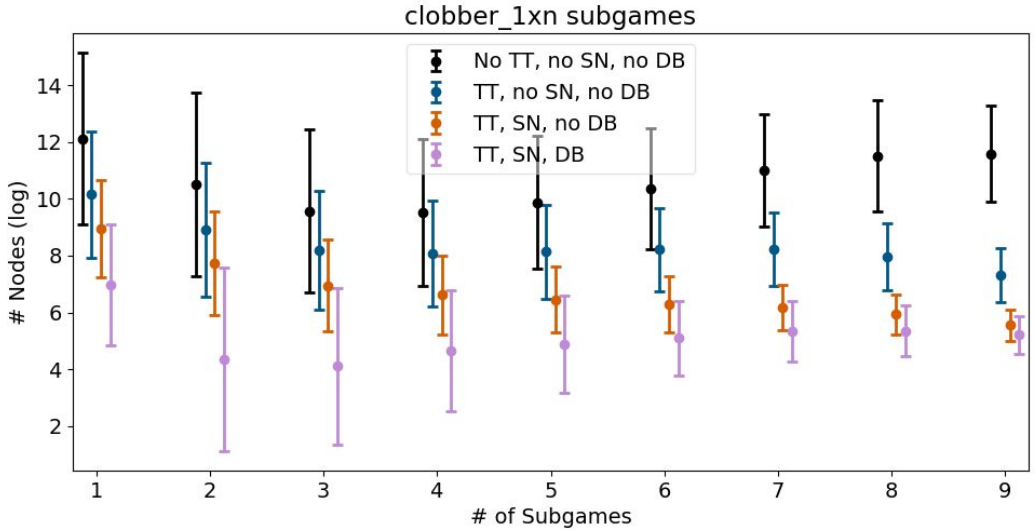








- Games:
  - Linear Clobber, Linear NoGo
    - X axis: Move count for first player
  - Elephants and Rhinos
    - X axis: Stone count at start
- Performance scales best with DB
  - Effect diminishes along X axis. Due to DB only storing outcome classes?



- Linear Clobber
  - X axis: Number of subgames at start
  - Again, benefit of DB diminishes with increasing number of subgames

Conclusions/Future Work

- MCGS fills the gap between:
  - CGT-based software relying on canonical forms
  - Solvers based on monolithic states
- Generic, pre-implemented optimizations are useful for a variety of games
  - Transposition table
  - Subgame structure
  - Database
- Future work:
  - Database
    - Support storing sums
    - Add more fields
      - Thermographs
      - Bounds
      - Simpler equal games (for substitution during search)
      - Dominated moves
  - More games
  - Move ordering heuristics
  - And much more!

Thanks For Listening!

- https://github.com/ualberta-mueller-group/MCGS
  - MIT license

* Incremental Hash Update

- Functions: Play, undo move, normalize, undo normalize
  - Play example: Clobber:■□■□ ➔ Clobber:_■■□
  - h(Clobber:_■■□) = h(Clobber:■□■□) ⊕ Z(0,■) ⊕ Z(0,_) ⊕ Z(1,□) ⊕ Z(1,■)

Update location 0 Update location 1

* Database Game Generator

- Generates in order of increasing “size”
  - Ensures DB creation is fast: after 1 move on generated game, resulting subgames are in DB
  - Ordered first by grid dimensions, then by game-dependent order:
    - Grid dimension ordering: 1x1 ➔ 1x2 ➔ 2x1 ➔ 2x2 ➔ ... ➔ 3x3
    - For given dimensions, Clobber generates in order of increasing stone count:
      - 1x3: ___ ➔ ... ➔ ■□■ ➔ ...
    - Similarly, NoGo generates in order of decreasing stone count:
      - 1x3: ■■■ ➔ ... ➔ ___