Random Search Algorithms

James H. Brodeur and Benjamin E. Childs

Budapest, Do8

Goal

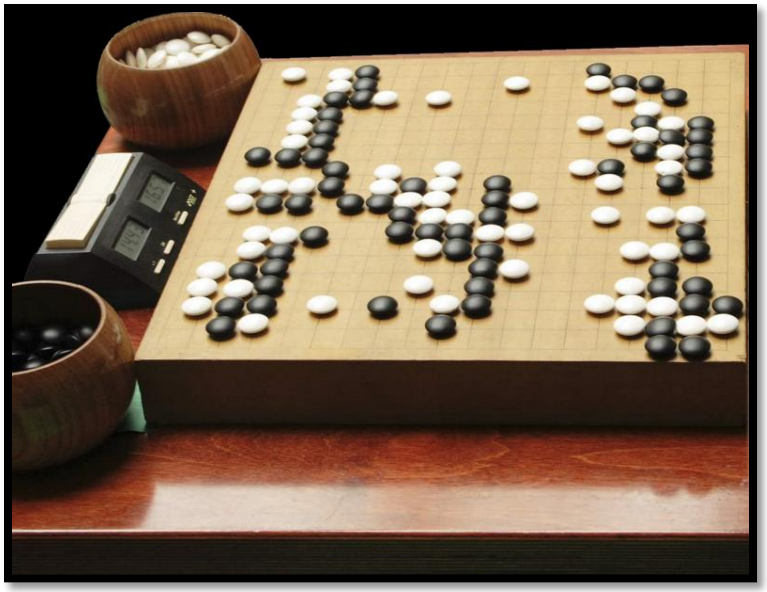
Design, develop and test improvements to the UCT random search algorithm for directed acyclic graphs and grouping

Why?

- Deep Blue defeated Kasparov 10 years ago
- Since then:
  - Standard workstation computer can now beat chess grandmasters
- Problems still remain where computers are not competitive vs. humans
- Computers must be able to deal with much higher branching factors

Introduction to the Rules of Go

- 2 Players
- 19 x 19 Board
  - may be smaller for beginners / computers
- Place pieces in turn



- Goal
  - Capture Territory
  - Keep Pieces Alive

Image courtesy of Wikipedia ('http://en.wikipedia.org/wiki/Go_(board_game)')

Search Algorithms

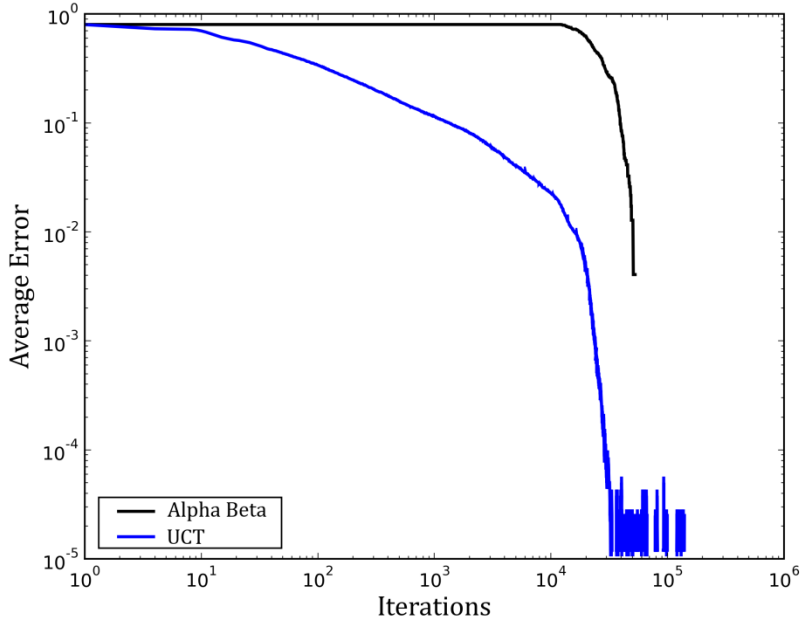
- Motivation
  - Game trees that are large can be hard to search
- Traditional Algorithms must search a relatively high % of the tree
  - Minimax, Alpha-Beta Pruning, Iterative Deepening
- Monte-Carlo Search offers alternative
  - Essentially random search from a point in the tree
  - Do many random searches and combine the outcomes

Random Search Algorithms

- Basic Monte Carlo Search
  - Sample nodes randomly, select move based on mean result
- Upper Confidence Bounds Applied to Trees (UCT)
  - Sample nodes based on the upper bound of a confidence interval.
  - Node with highest upper confidence bound is sampled next
  - Improves sample to look at nodes that have the greatest potential for good results

Advantage of UCT

- Balances exploration-exploitation tradeoff
- Converges to correct answer sooner
- Provides acceptable results even after a short time



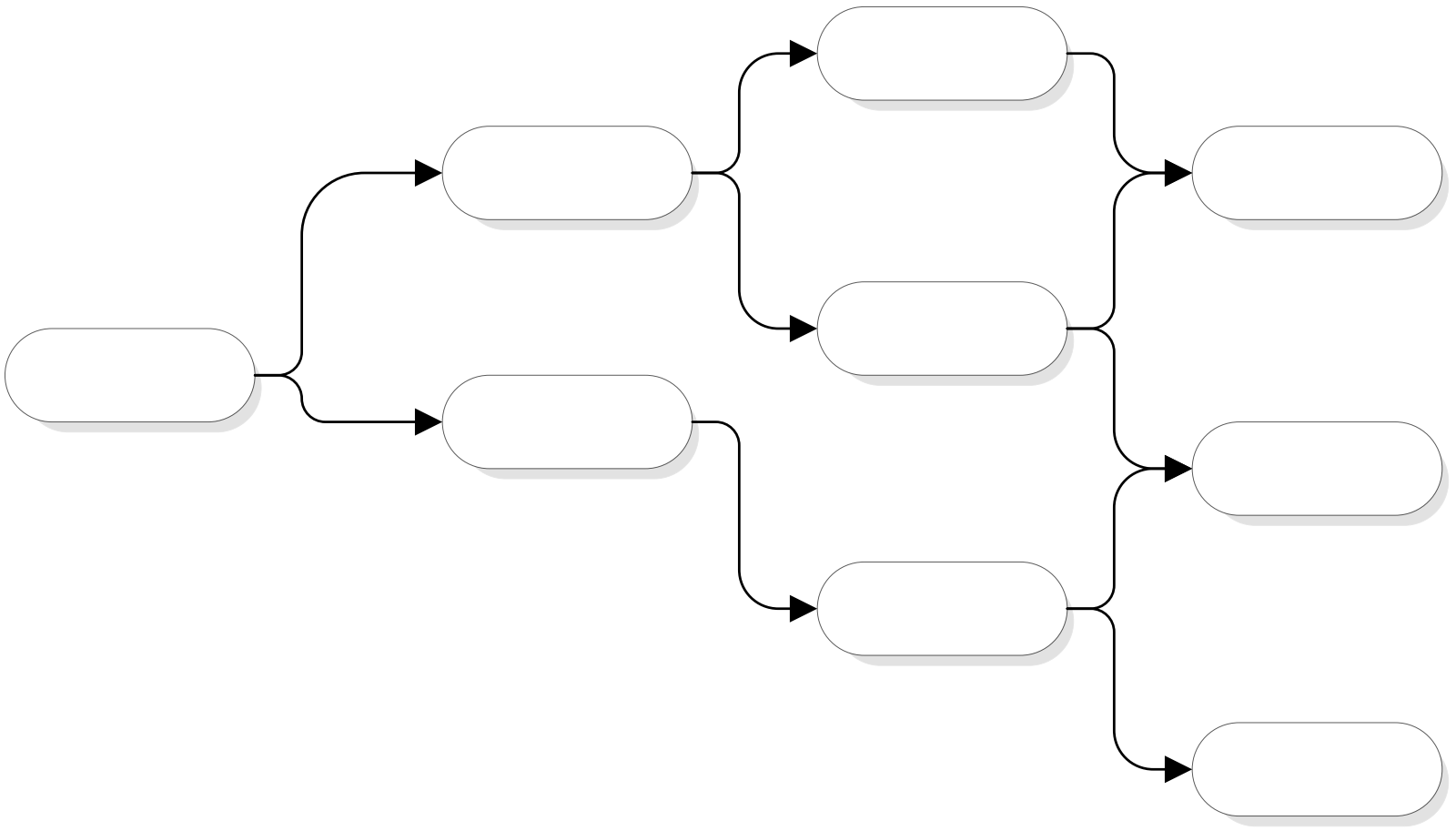
Development

UCT-DAG and Grouping

Directed Acyclic Graphs (DAGs)

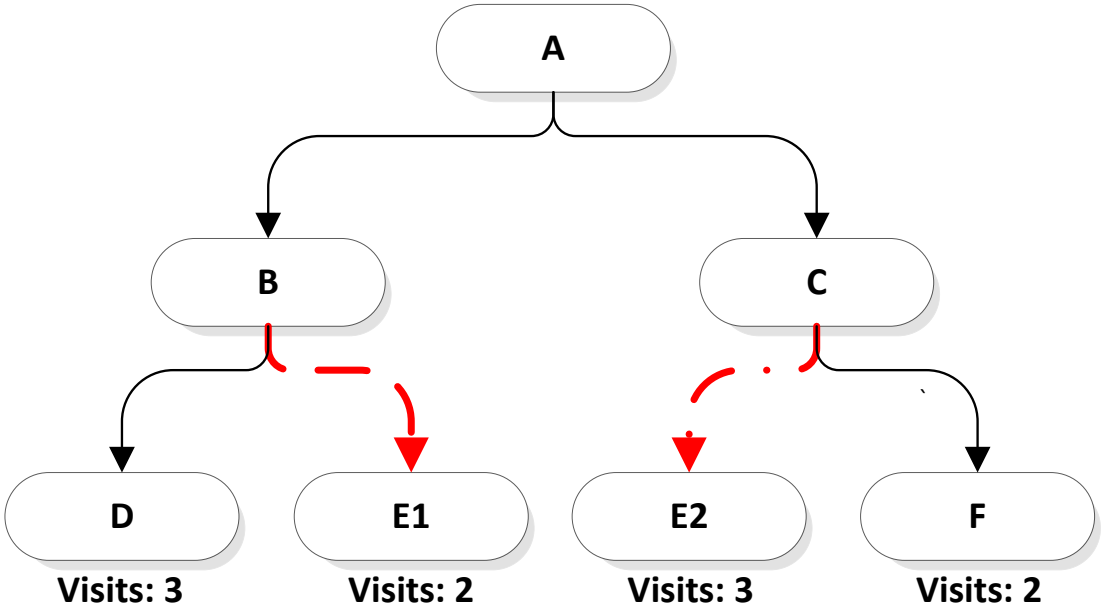
- Many game trees are actually graphs
  - Game position may be arrived at from more than one path.
  - Directed Acyclic graph is a better approximation of this than a Tree
- Game of Go expressly forbids cycles (ko, superko)
  - So, DAG is a perfect representation

Directed Acyclic Graphs (DAGs)



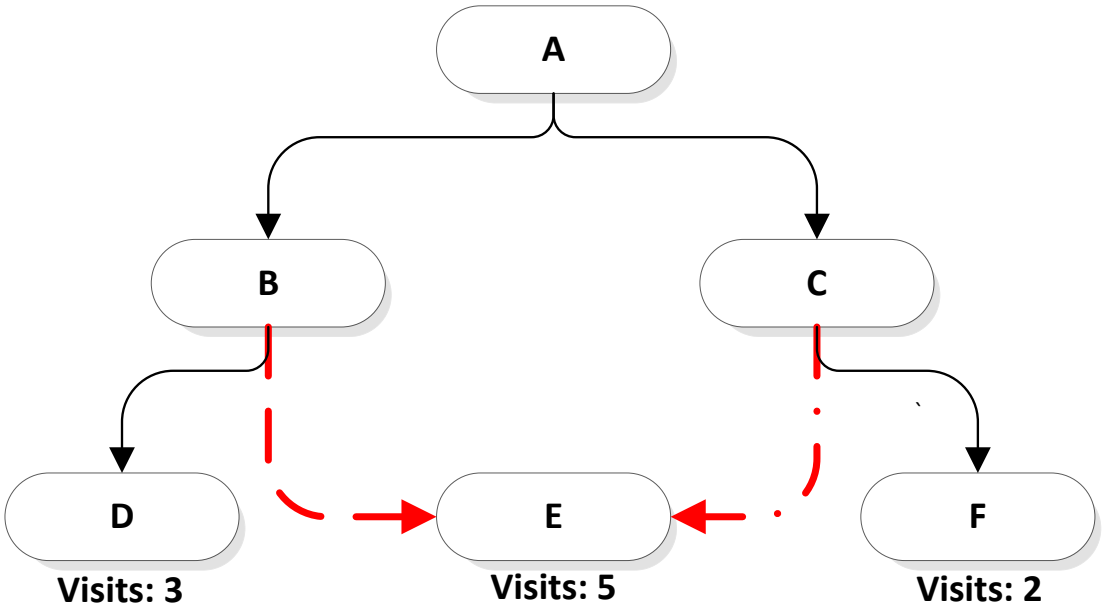
UCT on DAGs

- Previous Approach
  - Treat DAG as Tree
  - Problem: more nodes to explore, does not share information



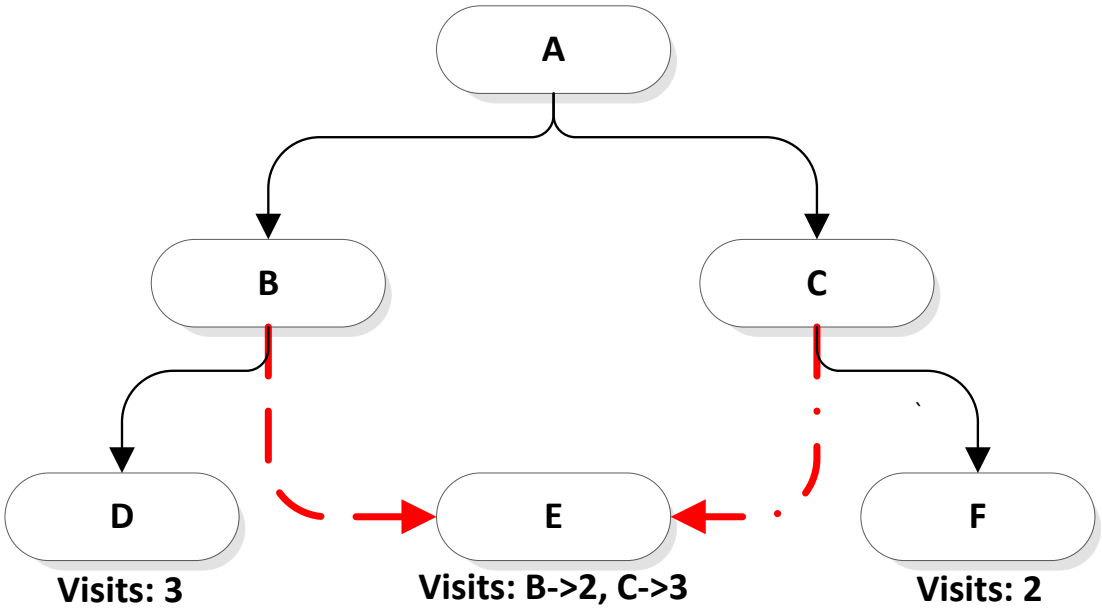
UCT on DAGs

- Naïve Approach
  - Keep visit count, value in each node.
  - Problem when there are multiple paths to nodes,



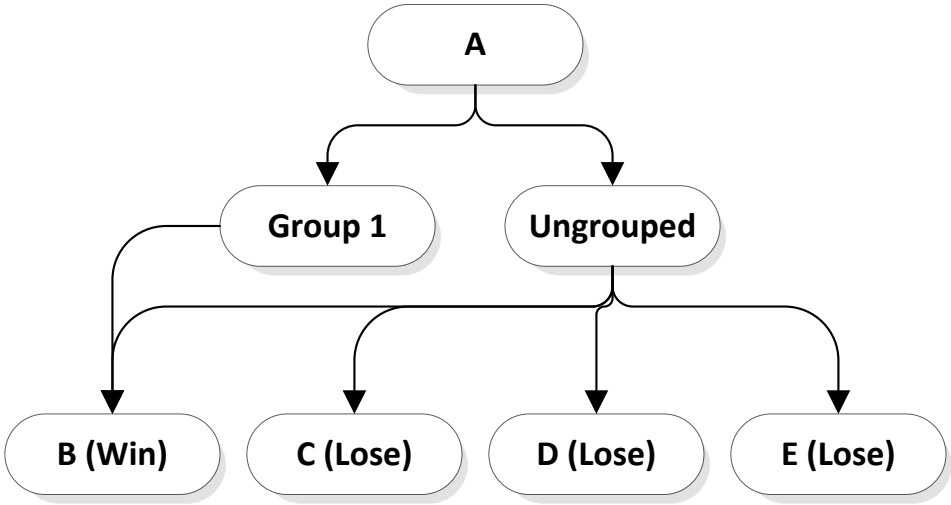
UCT on DAGs

- Current Solution
  - Share values between nodes on DAG
  - Keep individual counts for each parent



UCT Grouping

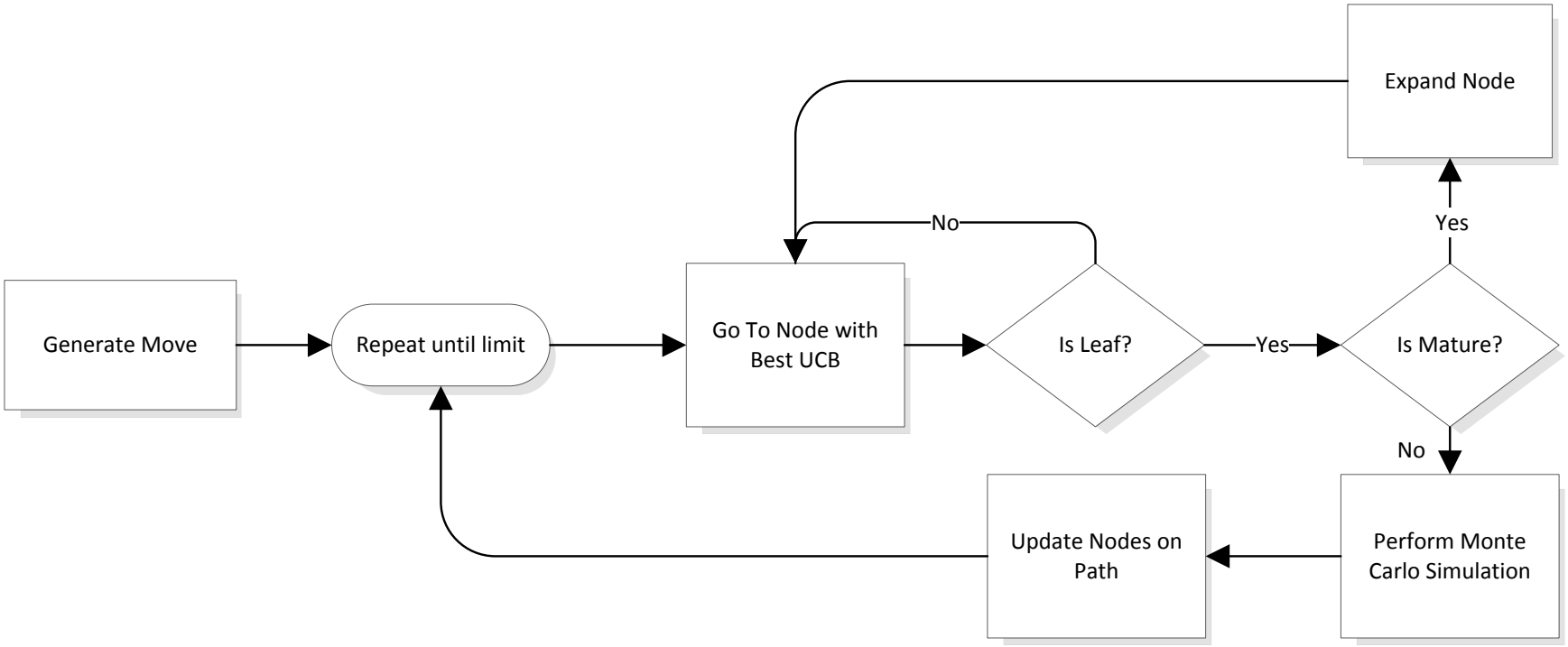
- Add structure to the tree,
  - Groups correlated with winning or losing moves
  - Proposed by Saito et al.
- In this example:
  - Equal chance of exploring winning move to
  - Higher chance of exploring winning move
- However,
  - Creates ~3x nodes
  - Deeper Trees



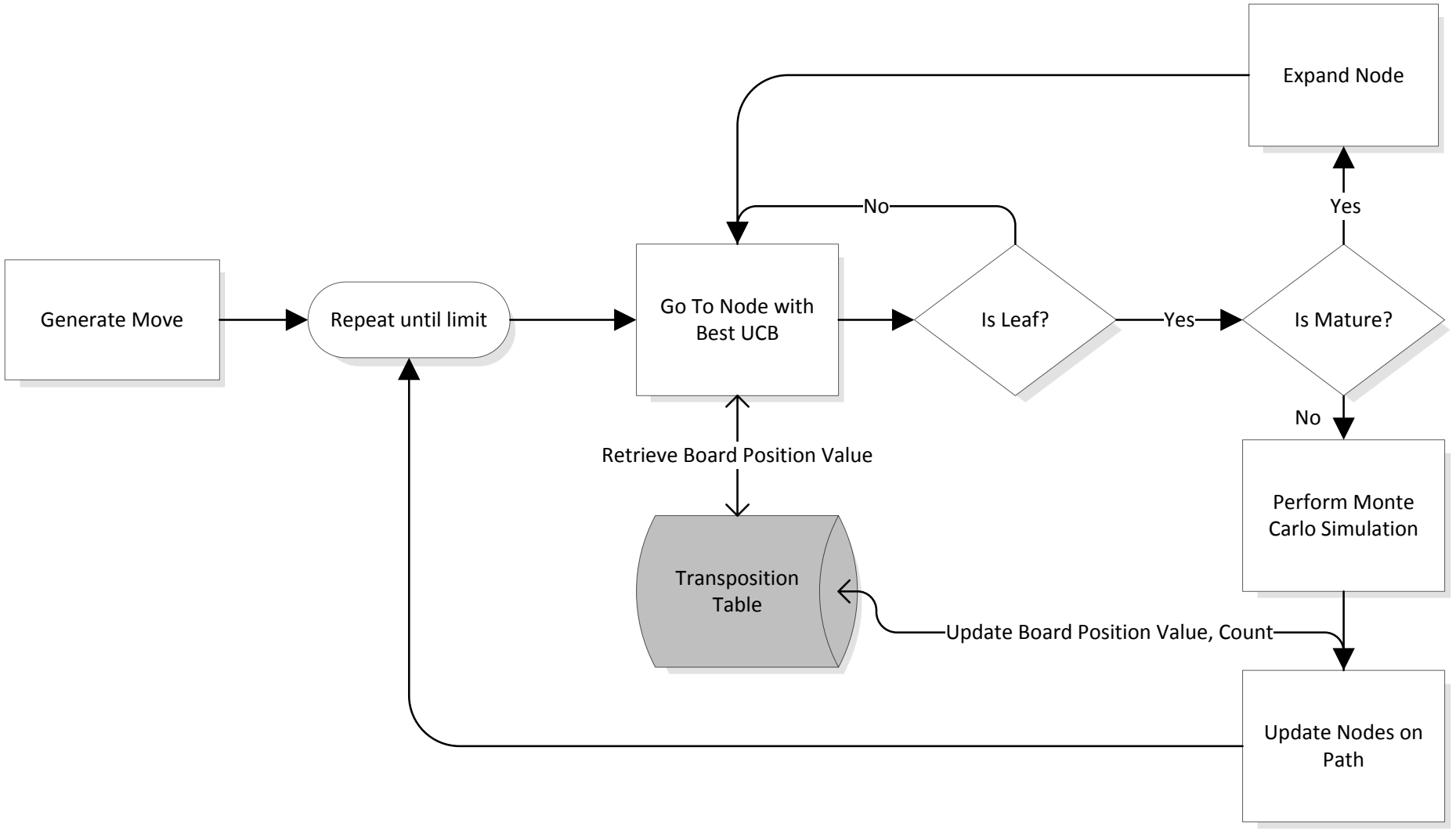
Artificial Grouping

- Generate artificial groupings similar to those used in Go.
- Use existing trees
  - Add n groups, biased by parameter α
  - With higher α
    - Winning moves more likely to be in groups
    - Losing moves less likely to be in groups

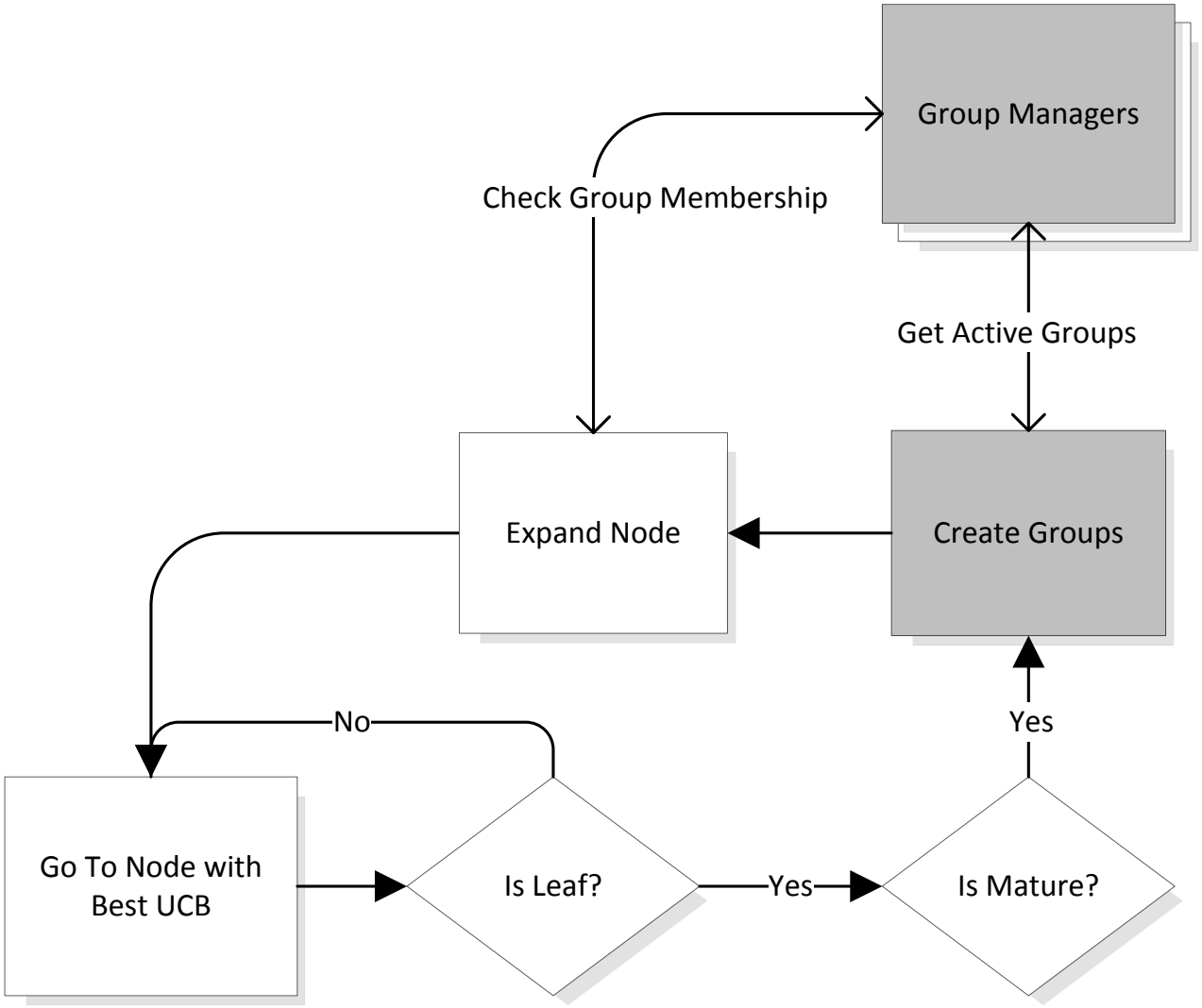
Initial LibEGO UCT Implementation



Added Transposition Table



Added Grouping



Implemented Groupings

Group Name
Else Group
Border Group*
Manhattan Distance (Enemy)*
Manhattan Distance (Friendly)
Manhattan Distance (Total)
Friendly Saves
Enemy Kills
Many Liberties
Friendly Chainmaker
Enemy Chainmaker
Chain Group

* - Previously Implemented by Saito et. al

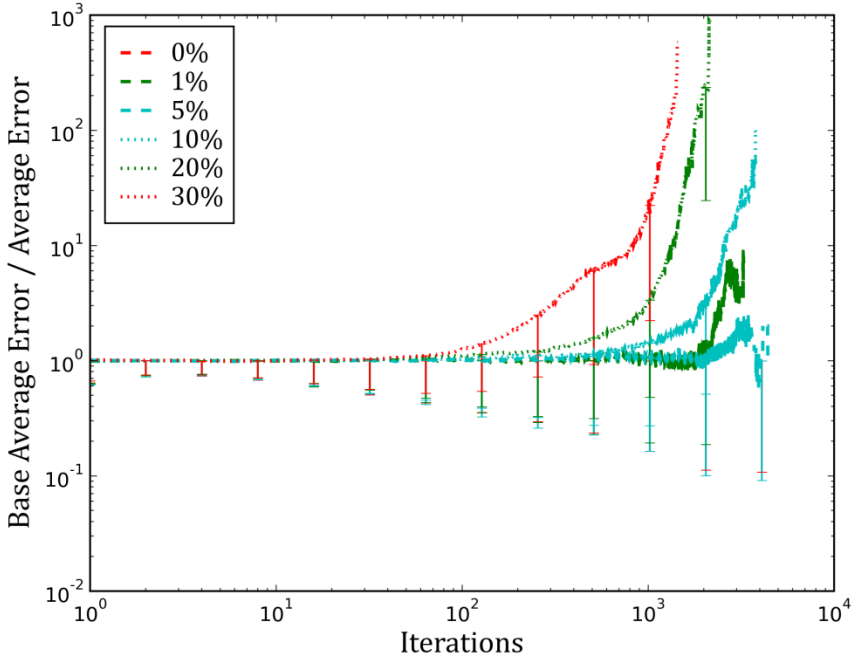
Artificial Game Tree Experiments

Why Artificial Game Trees?

- Provides controlled environment to test UCT modifications
  - Run the experiments on exactly the same trees
  - Small enough trees that the correct moves can be calculated
- Quantify improvements

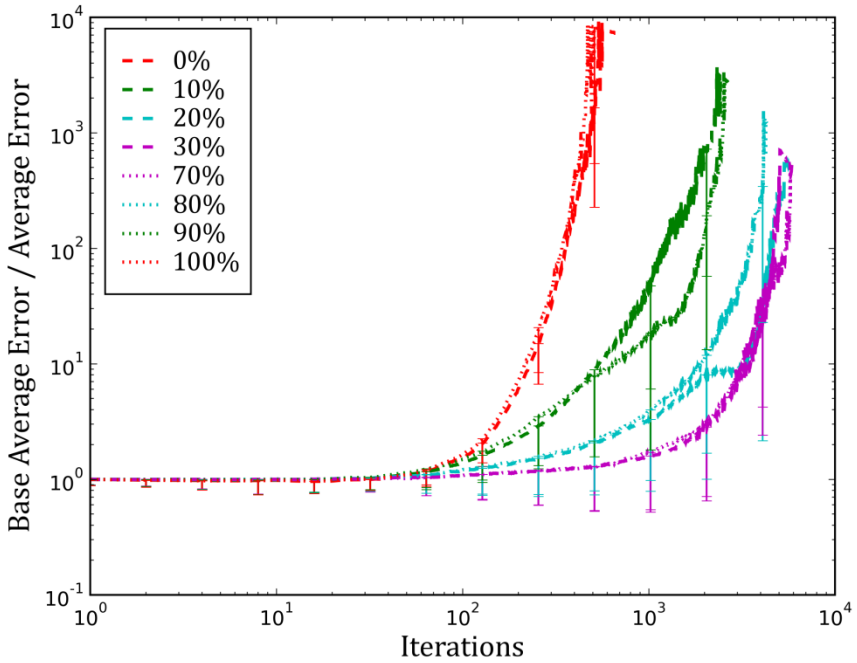
Findings

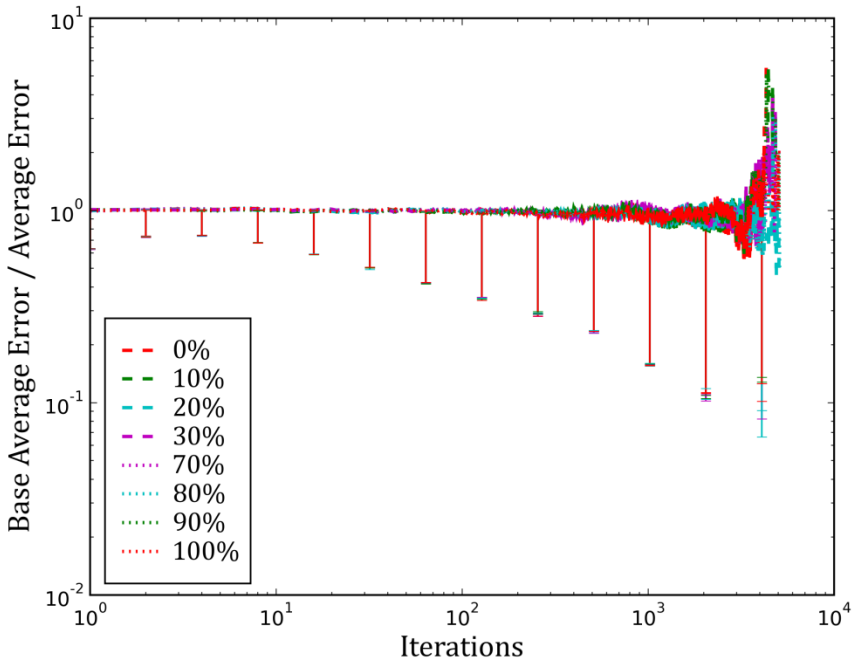
UCT-DAG can improve performance of UCT on complex DAGs and performs equally to UCT on simple DAGs.



Findings

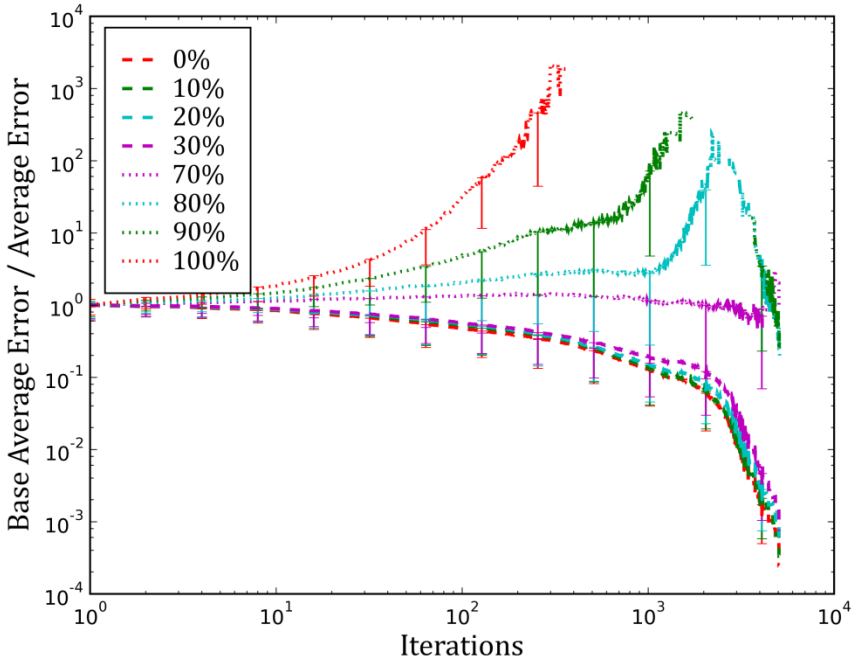
Highly correlated, complete groupings dramatically improve performance of UCT on trees with high branching factors.





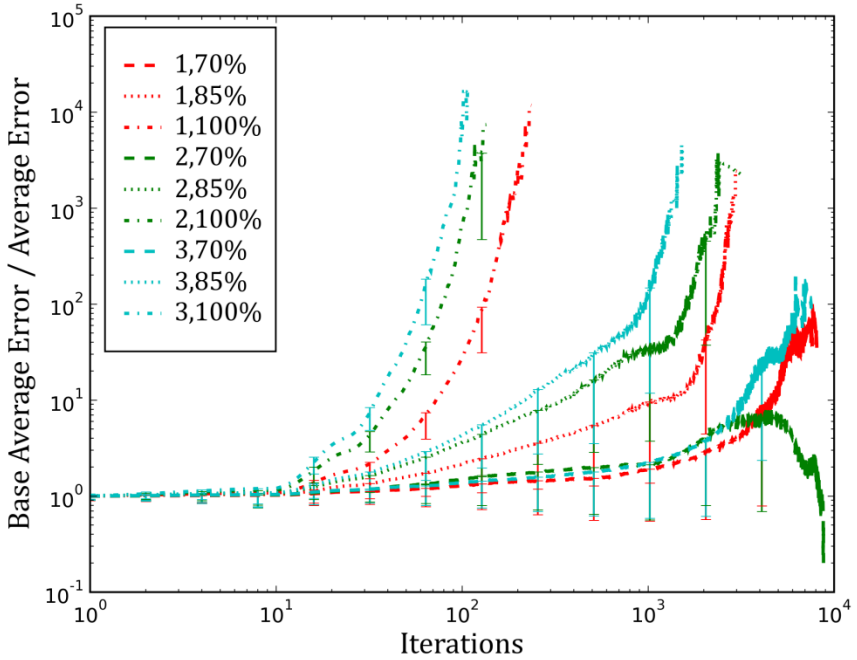
Findings

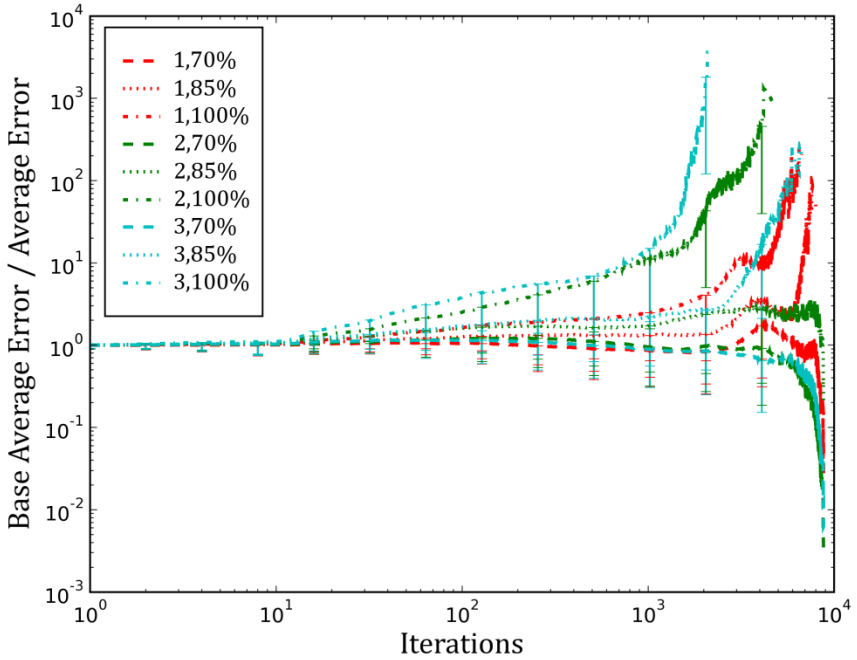
Highly accurate, complete groupings with group overlap dramatically improve performance of UCT on any trees.



Findings

Multiple less-accurate groups, with group overlap, perform similarly to a single more-accurate group.





Computer Go Experiments

Computer Go Experiments

Experiments	Base	No Groupings	Groupings (4)
Board Size: Play Time	9x9: 10s, 20s	13x13: 40s, 80s, 160s	
Number of Games	1000 games split over 5 processors		

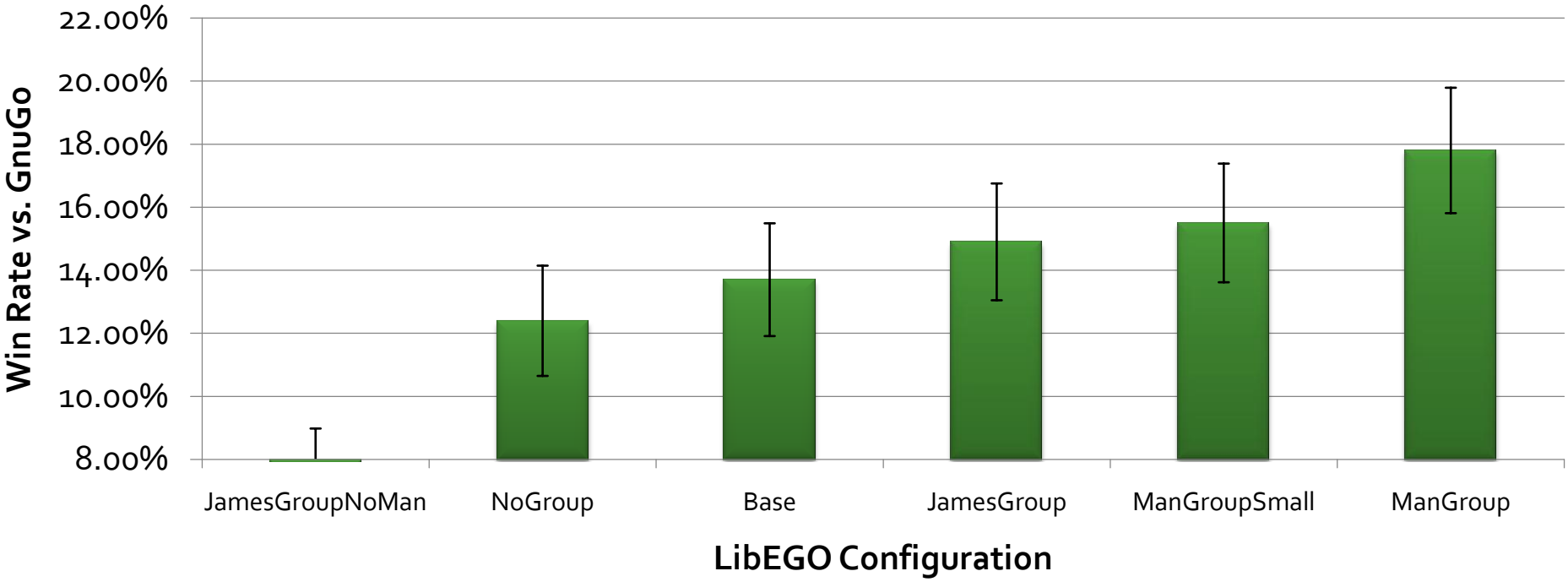
- Using GnuGo as opponent
- TwoGTP as referee

LibEGO Experiments

Name	Description
Base	Original LibEGO algorithm
NoGroup	Added 500,000 entry transposition table saved between moves
ManGroup	NoGroup with Manhattan Enemy and Friendly Groupings
ManGroupSmall	ManGroup except only 50,000 entries in the transposition table
JamesGroup	NoGroup with Manhattan Total Group, Enemy Captures Group, Friendly Saves Group and Many Liberties Group
JamesGroupNoMan	James Group without Manhattan Group.

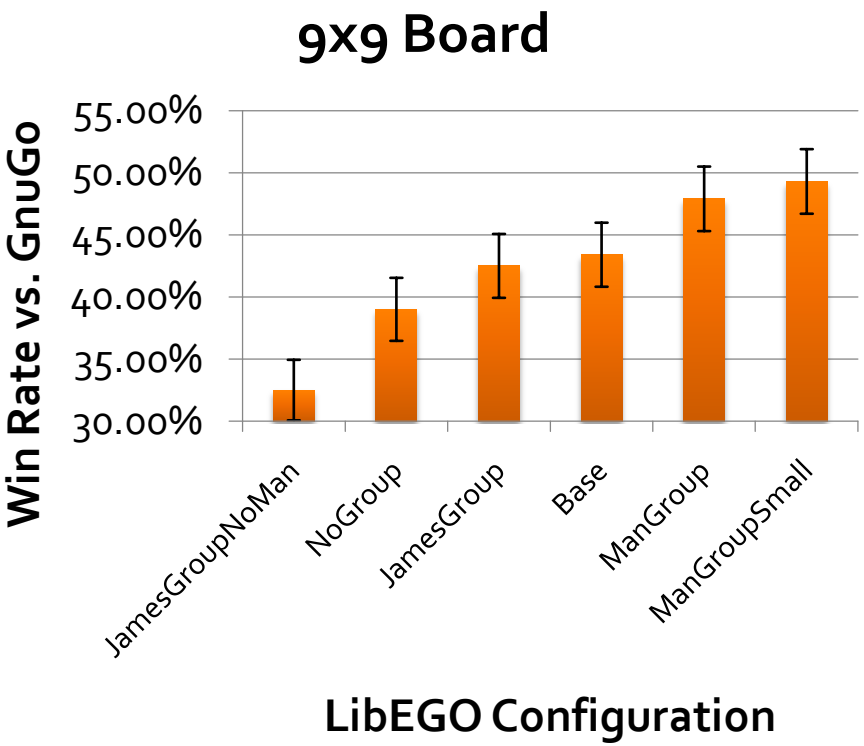
Findings

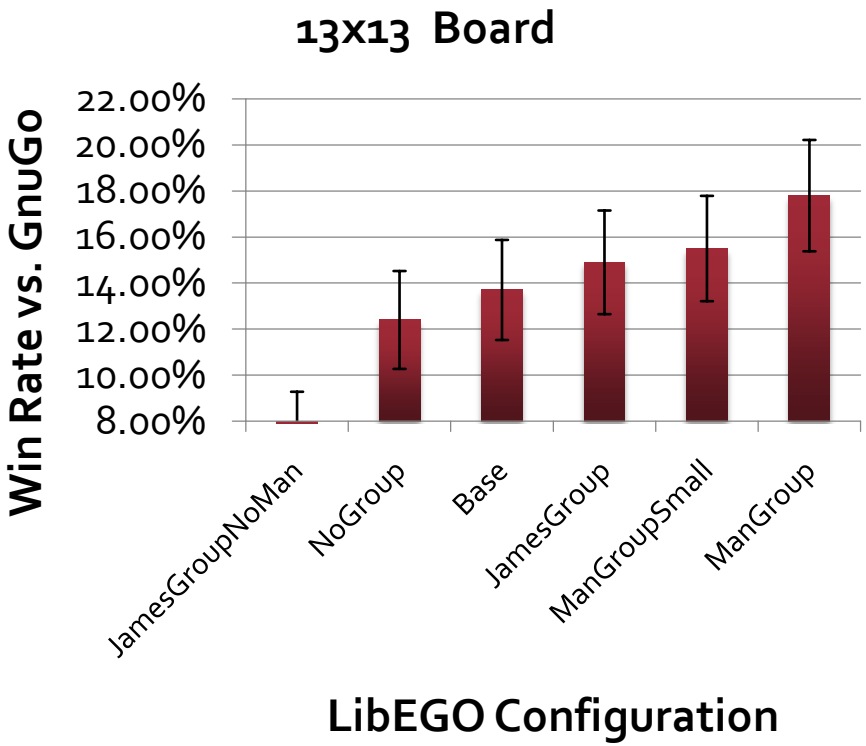
Accurate groupings in conjunction with UCT DAG improves performance of LibEGO vs GnuGo on larger boards.



Findings

The size of the UCT-DAG transposition table may be sensitive to branching factor.





Future Research

Future Research

UCT-DAG with Multi-Path Update

Future Research

Transposition Table

Future Research

Online or Offline Determination of Group Biases & Parameter Tuning

Future Research

Using the GRID as a mechanism for Machine Learning with UCT / Grouping

Future Research

UCT/Monte-Carlo Simulation Split: LibEGO & PGame

End Result

Acknowledgments

- **Advisor**: Gábor Sárközy
- **Co-Advisor:** Stanley Selkow
- **SZTAKI**
  - **MLHCI Group**
    - Levente Kocsis, András György
    - Petronella Hajdú
  - **Internet Applications Department**
  - Adam Kornafeld
- **Worcester Polytechnic Institute**

Thank You

Questions or Comments?