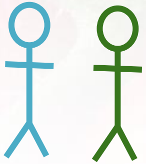
Simulating Dilemmas

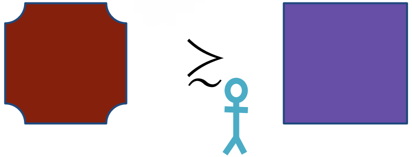
Justus Hibshman - 12/10/18

Games

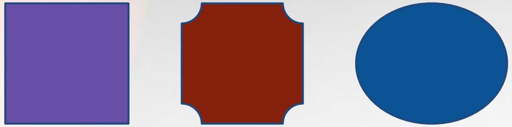
Players/Agents i=1...n



Preferences $\succsim_i$



Consequence set $C = \{c_1, \ldots, c_m\}$



Strategy Set $S = S_1 \times \ldots \times S_n$

$f: S \to C$

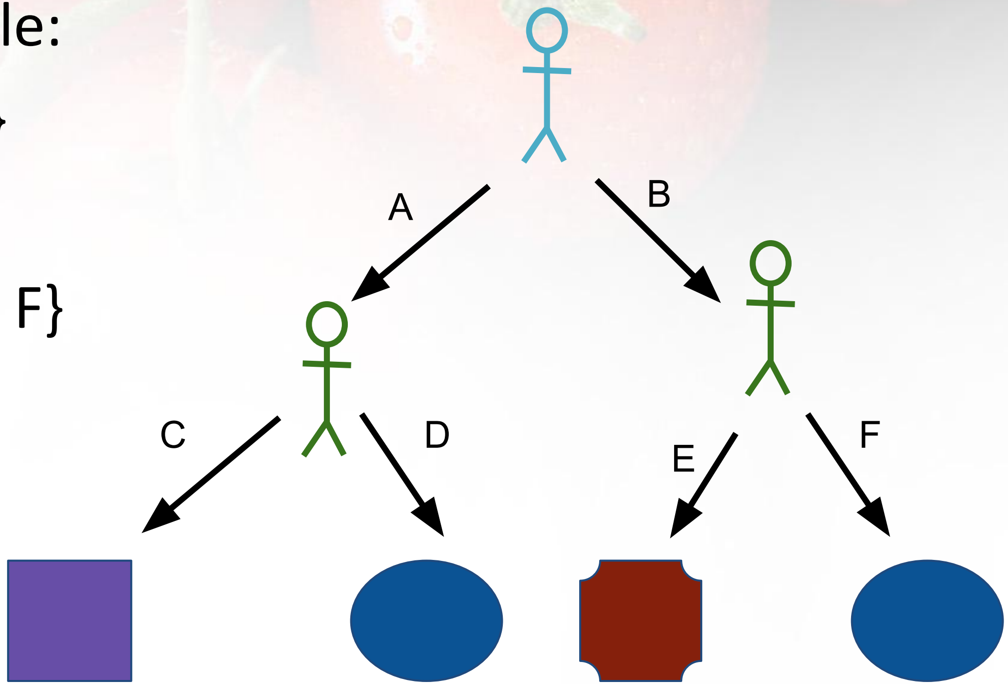
Extensive Form Games

Strategies are composed of decisions at tree nodes.

Example:

S = {A}

S = {C, F}



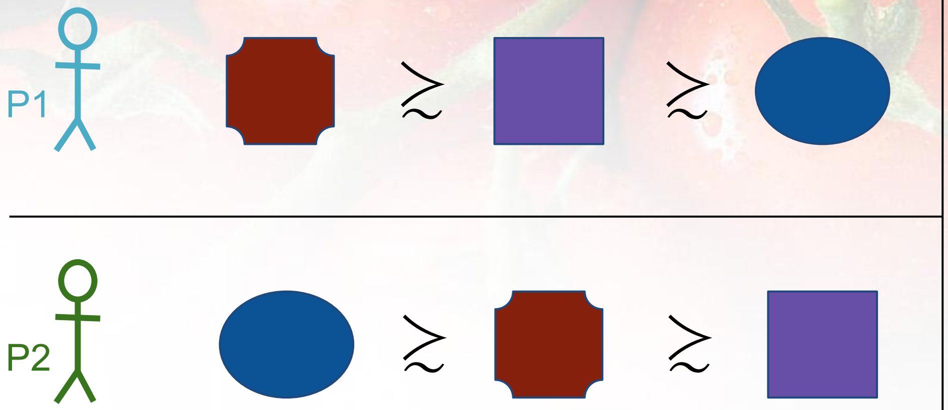
Dilemmas

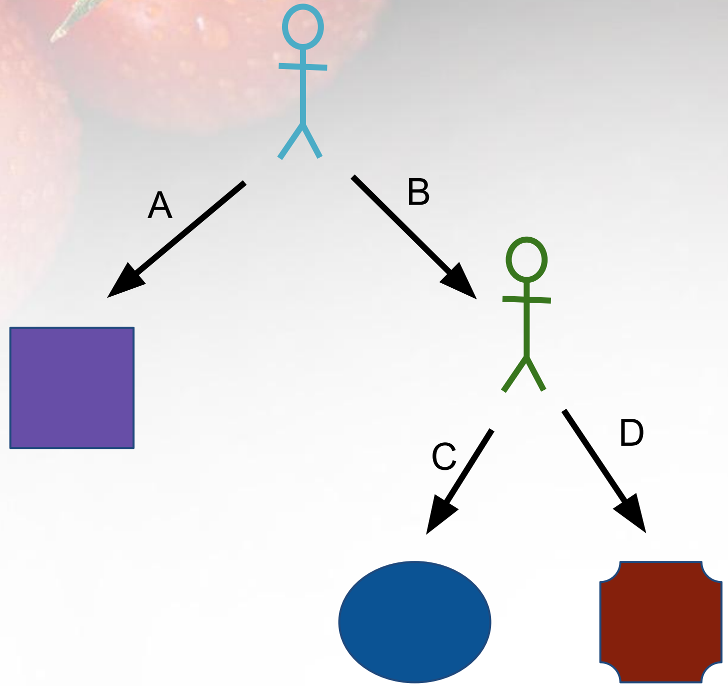
Assumptions:

- All players are rational.
- All players know the other players' preferences.
- Everything a player cares about has been encapsulated in the preference relation.
- These facts are "Common Knowledge."
    - Basically means "I know that you know that I know…"

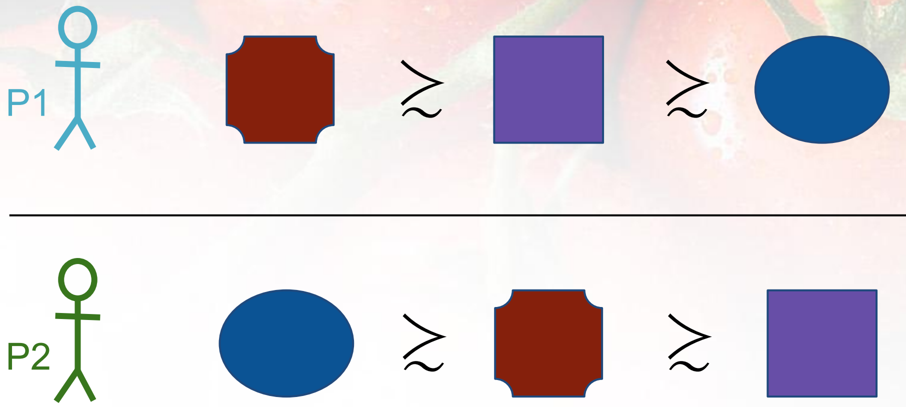
Dilemmas

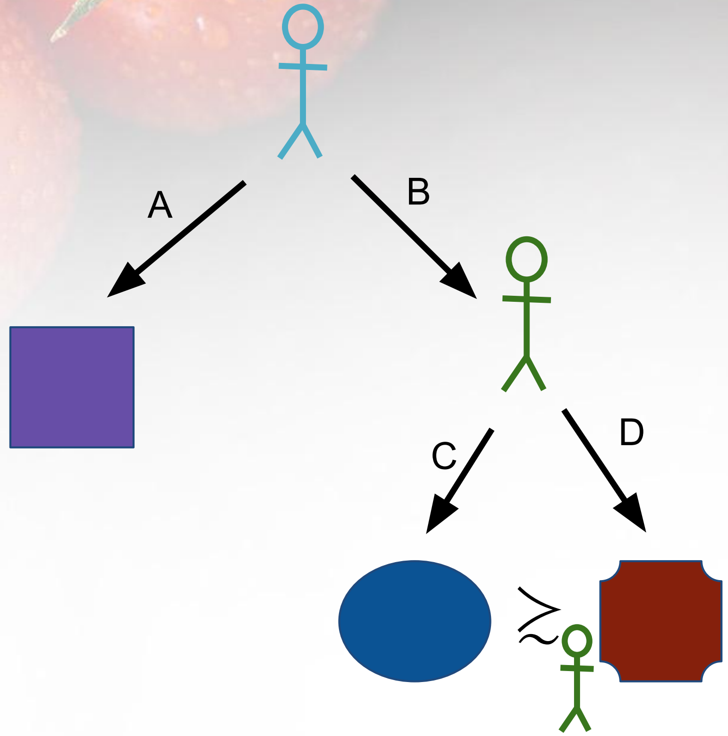
Dilemmas



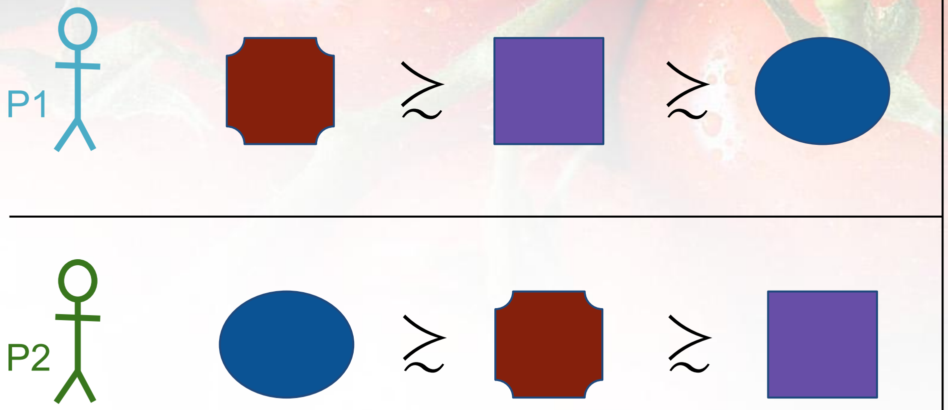


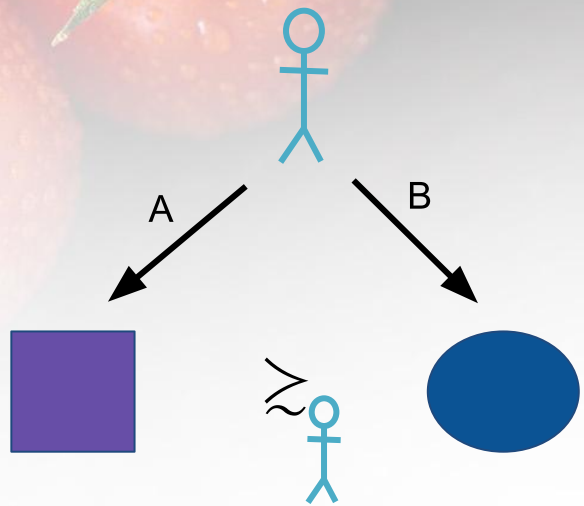
Dilemmas



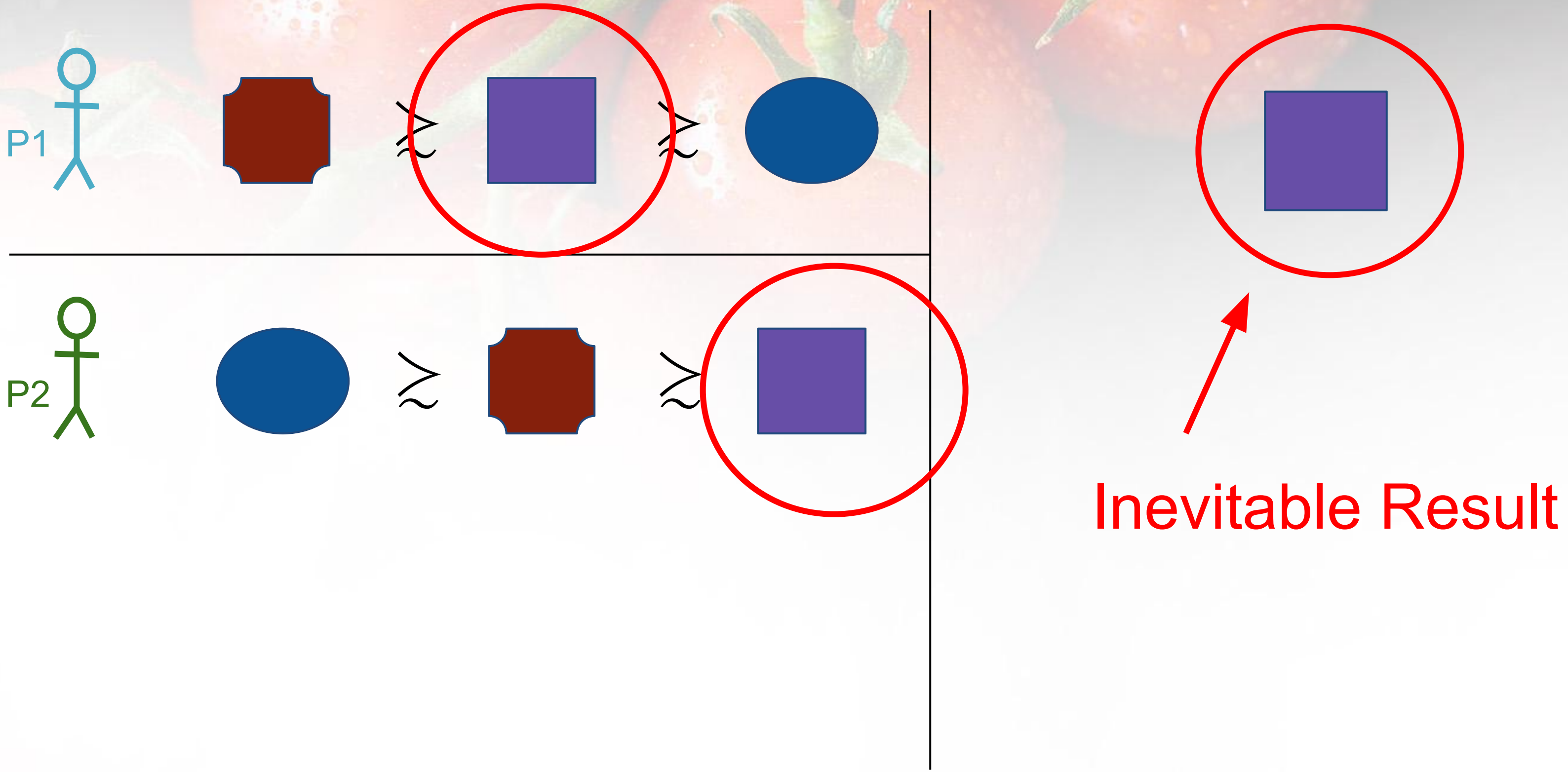


Dilemmas

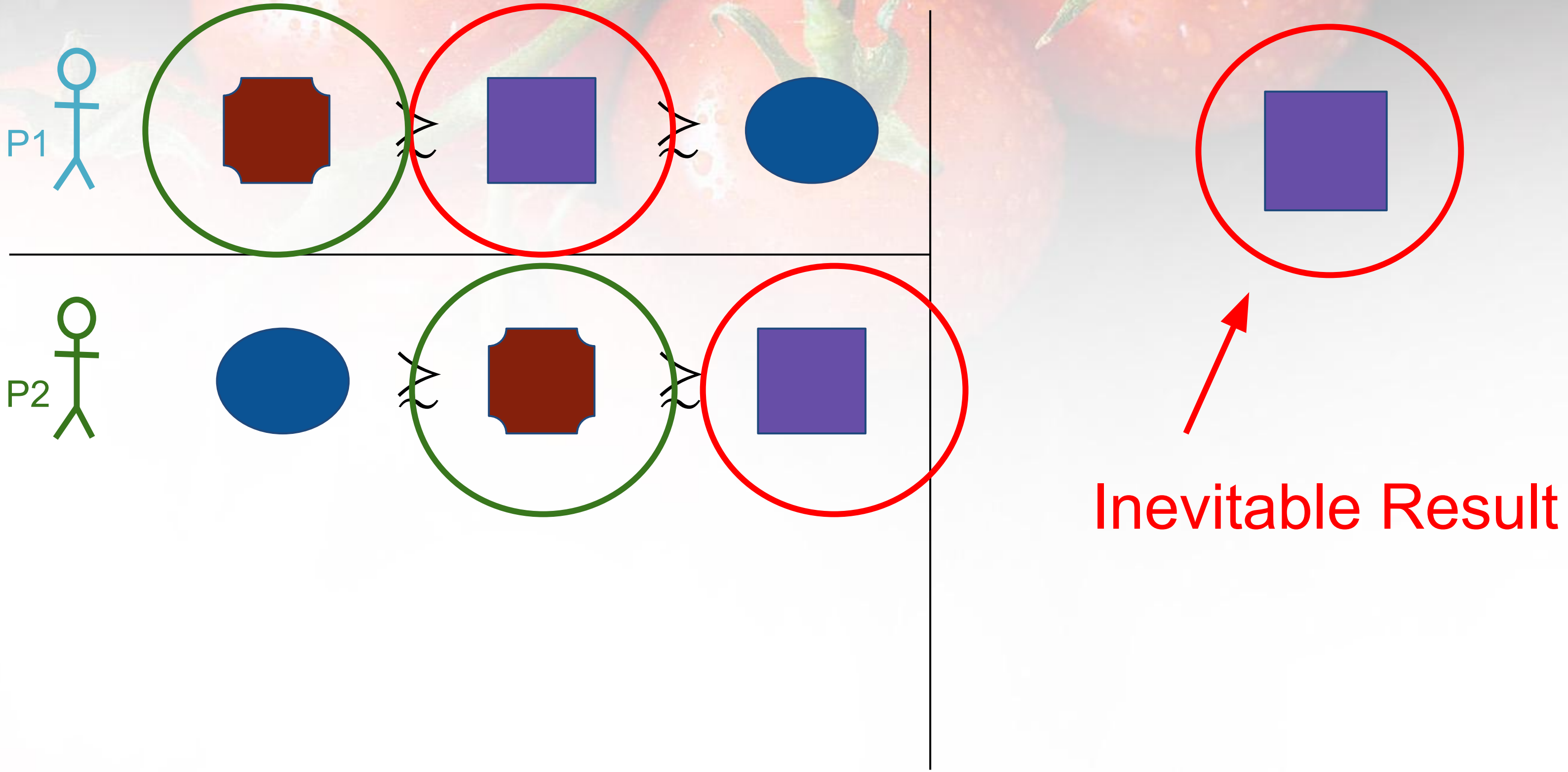




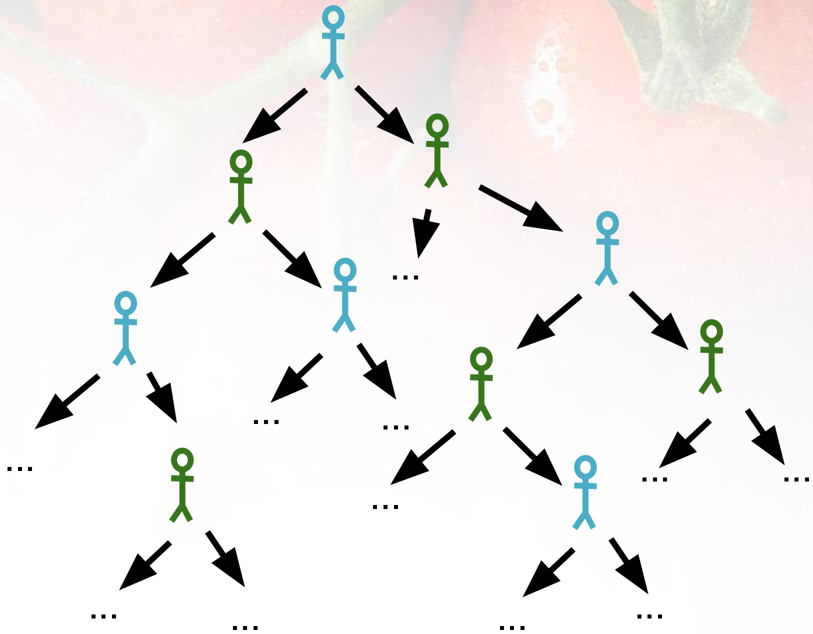
Dilemmas



Dilemmas



Do Dilemmas Occur Naturally?



- Big Trees
- Random Preferences

Kernel: Mini-Max with DFS

```
Def mini_max(node)
    If node.is_leaf():
        Return node.consequence
    result = null
    For child In node.children
        child_consequence = mini_max(child)
        If result Is null Or node.player.prefers(child_consequence, result)
            result = child_consequence
    Return result
```

Complexity

Number of players: **p**

Number of vertices: **v**

Number of edges: **e** = **v - 1** because its a tree.

MiniMax with DFS: O(**p**(**v** + **e**)) = O(**pv**)

Game Tree Generation: O(**pv**)

Checking Optimality of Result: O(**pv**)

Original Implementation

- DFS: Boost Graph Library (C++)
  - Overload “DFS Visitor” class
- Game Generation
  - Custom C++ code
  - Boost graph format

Enhanced Implementation

- Custom Graph Format (C++)
  - Adjacency List Implementation
- Custom Tree Traversal Code
- Parallelization done with pthreads
- ~550 Lines of Code

Enhanced Implementation

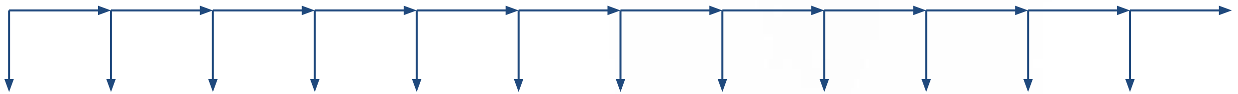
Basic idea:

- Available worker-thread pool
- Workers do sub-traversals
- Any worker can assign sub-traversal to other workers
- When a worker finishes:
  - Wake up the thread waiting on results
  - Re-enter the pool

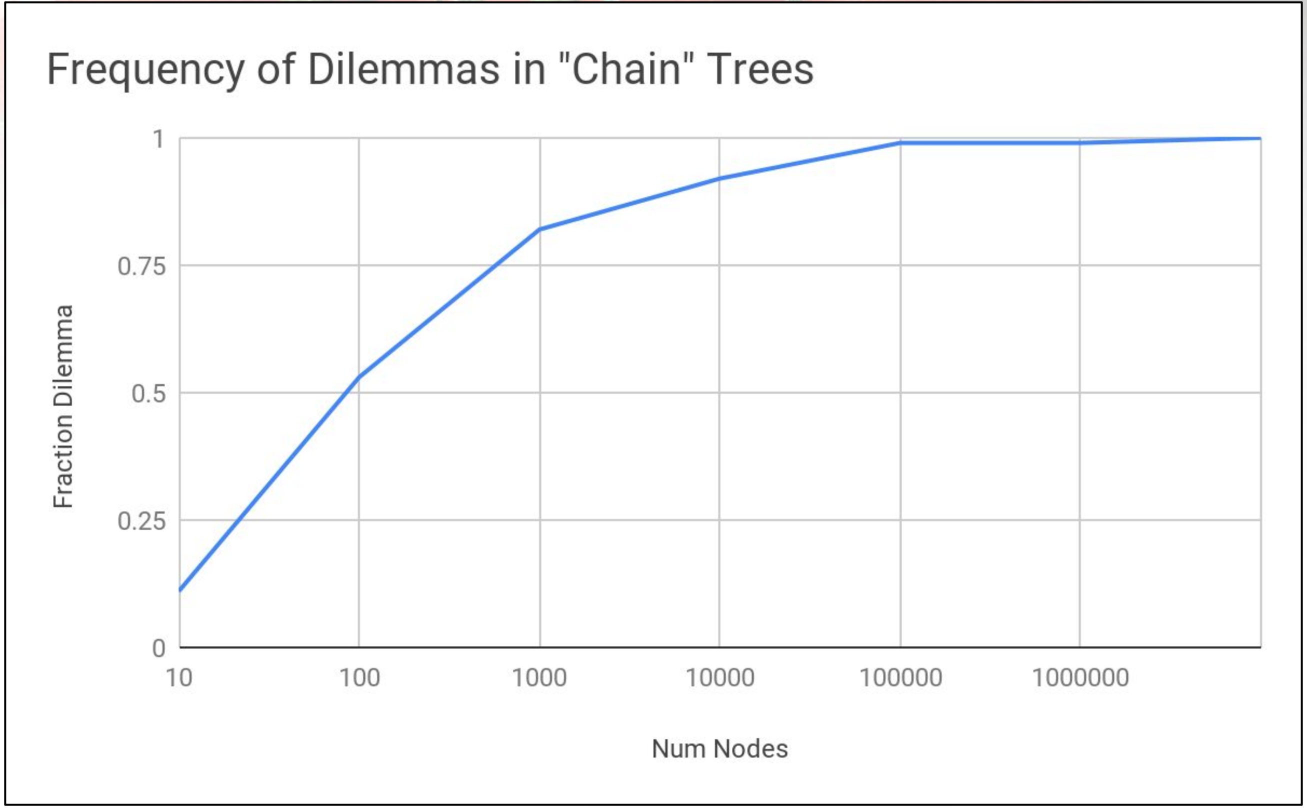
Experiment Setup

Run 100 trials for each parametrization:

- Number of players: 4
- Vary number of game tree nodes from 10 to 10,000,000
- Balanced Trees (degree of 8)
- “Chain” Trees
  - Every decision node has one stop-edge leading to a final outcome and one continue-edge.
  - Example:

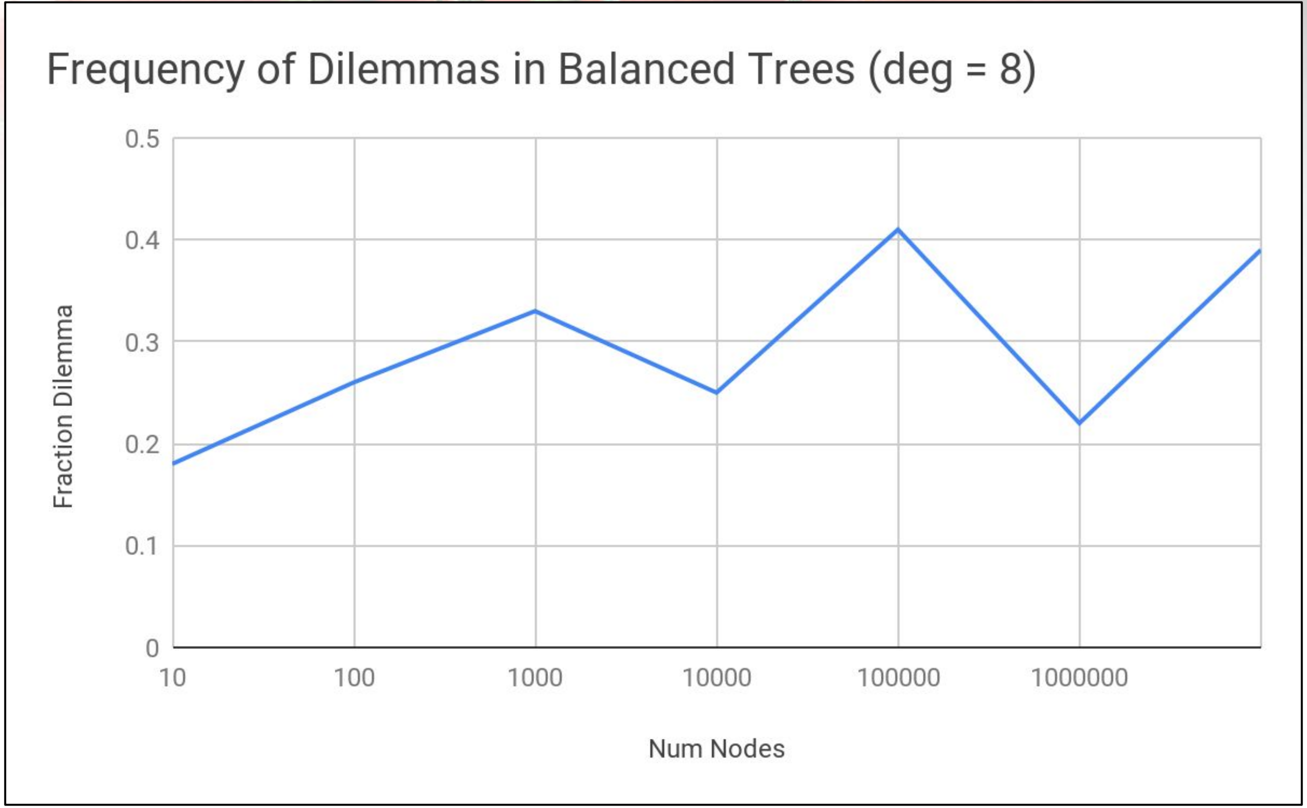


Results

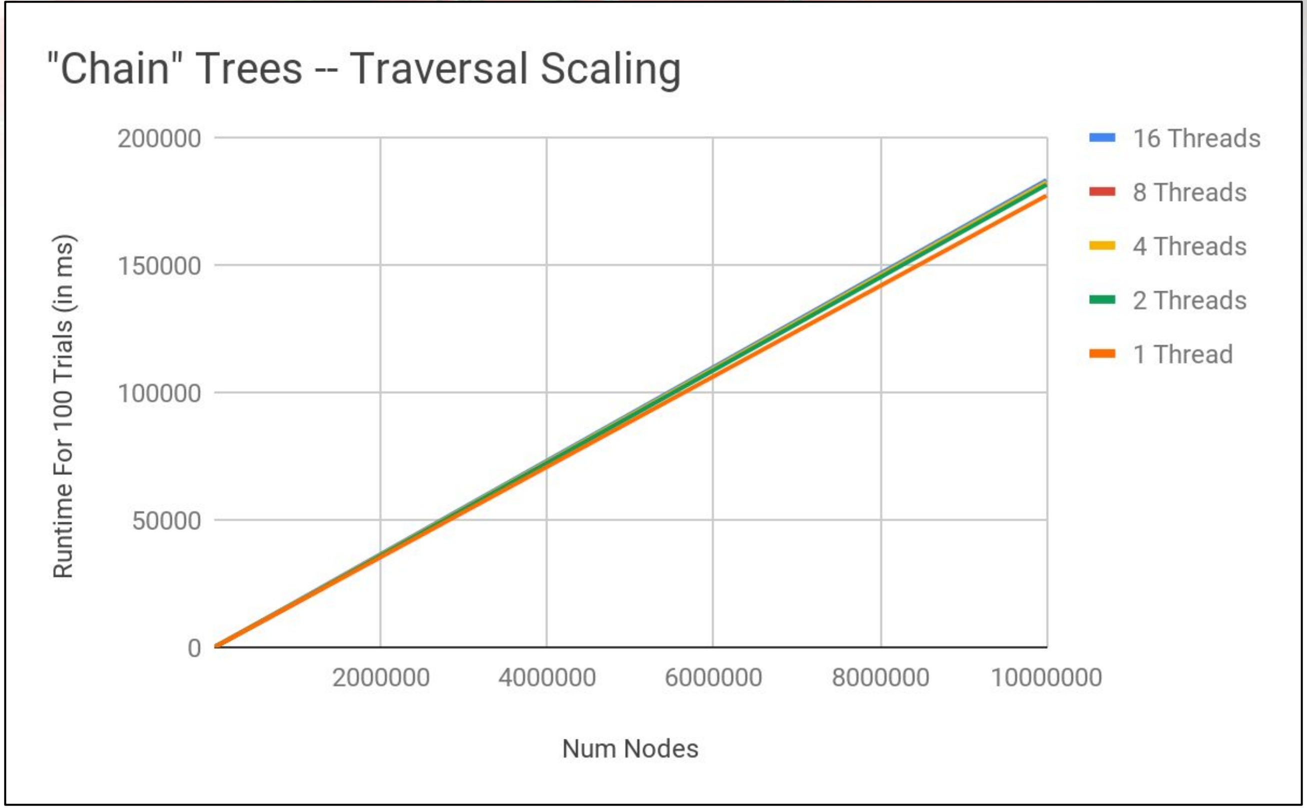


Frequency of Dilemmas in "Chain" Trees
Fraction Dilemma
1
0.75
0.5
0.25
0
10
100
1000
10000
100000
1000000
Num Nodes

Results

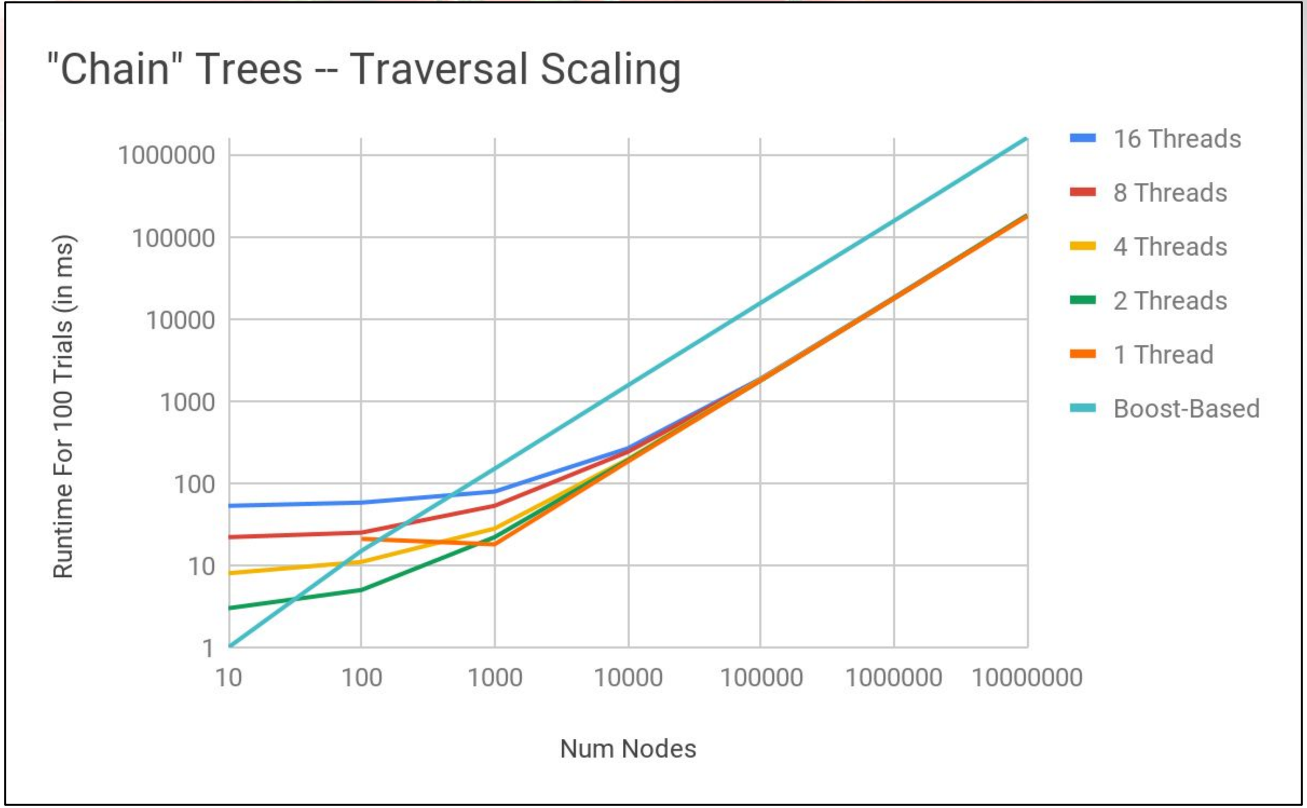

Frequency of Dilemmas in Balanced Trees (deg = 8)
0.5
0.4
0.3
0.2
0.1
0
Fraction Dilemma
10
100
1000
10000
100000
1000000
Num Nodes

Results

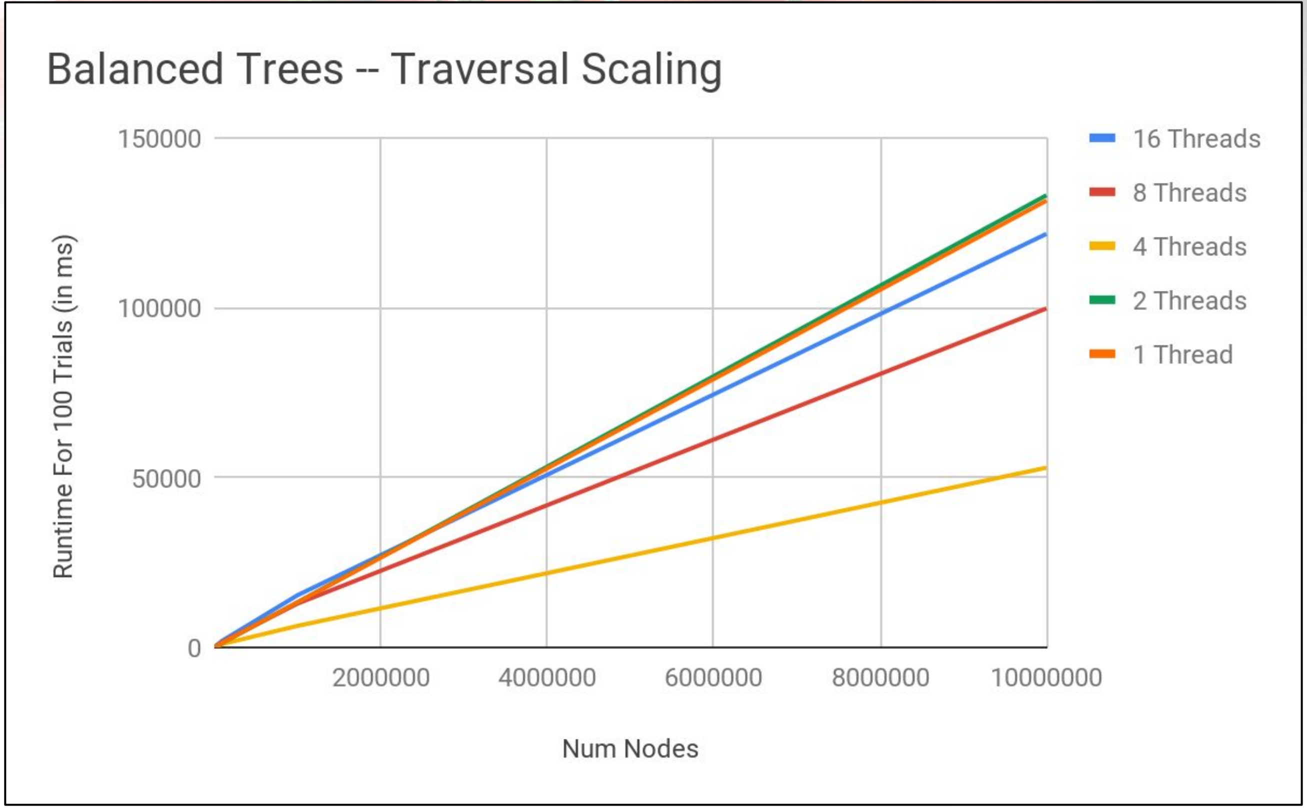


"Chain" Trees -- Traversal Scaling
Runtime For 100 Trials (in ms)
200000
150000
100000
50000
0
2000000
4000000
6000000
8000000
10000000
Num Nodes
16 Threads
8 Threads
4 Threads
2 Threads
1 Thread

Results

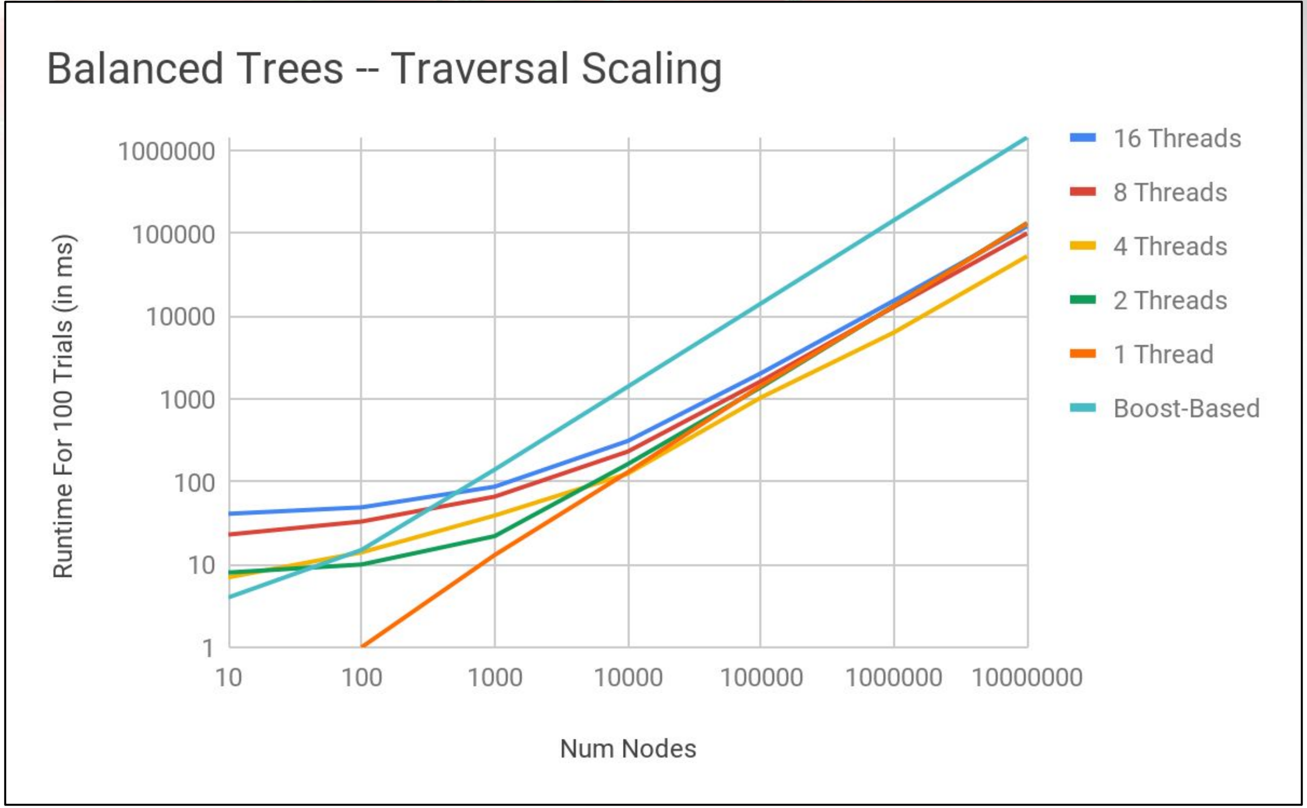

"Chain" Trees -- Traversal Scaling
Runtime For 100 Trials (in ms)
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
10000000
Num Nodes
16 Threads
8 Threads
4 Threads
2 Threads
1 Thread
Boost-Based

Results



Balanced Trees -- Traversal Scaling
150000
100000
50000
0
2000000
4000000
6000000
8000000
10000000
Runtime For 100 Trials (in ms)
Num Nodes
16 Threads
8 Threads
4 Threads
2 Threads
1 Thread

Results



Balanced Trees -- Traversal Scaling
Runtime For 100 Trials (in ms)
1000000
100000
10000
1000
100
10
1
10
100
1000
10000
100000
1000000
10000000
Num Nodes
16 Threads
8 Threads
4 Threads
2 Threads
1 Thread
Boost-Based

Reference

- Boost C++ Libraries
  - https://www.boost.org/