DFS* and the Traveling Tournament Problem

David C. Uthus, Patricia J. Riddle, and Hans W. Guesgen

Traveling Tournament Problem

- Sports scheduling combinatorial optimization problem.
- Objective is to create double round robin tournament with minimal travel distance.

Team \ Round	1	2	3	4	5	6
A	@B	@C	@D	B	C	D
B	A	D	@C	@A	@D	C
C	@D	A	B	D	@A	@B
D	C	@B	A	@C	B	@A

Traveling Tournament Problem

- *at_most:* Restricts consecutive number of home or away games to 3.
- *no_repeat:* No back-to-back games against same team.

Round / Team	1	2	3	4	5	6
A	@B	@C	@D	B	C	D
B	A	D	@C	@A	@D	C
C	@D	A	B	D	@A	@B
D	C	@B	A	@C	B	@A

Traveling Tournament Problem

- Objective: Minimize total travel distance.
- Distances calculated individually for each team, then summed together.

Round / Team	1	2	3	4	5	6
A	@B	@C	@D	B	C	D
B	A	D	@C	@A	@D	C
C	@D	A	B	D	@A	@B
D	C	@B	A	@C	B	@A

Traveling Tournament Problem

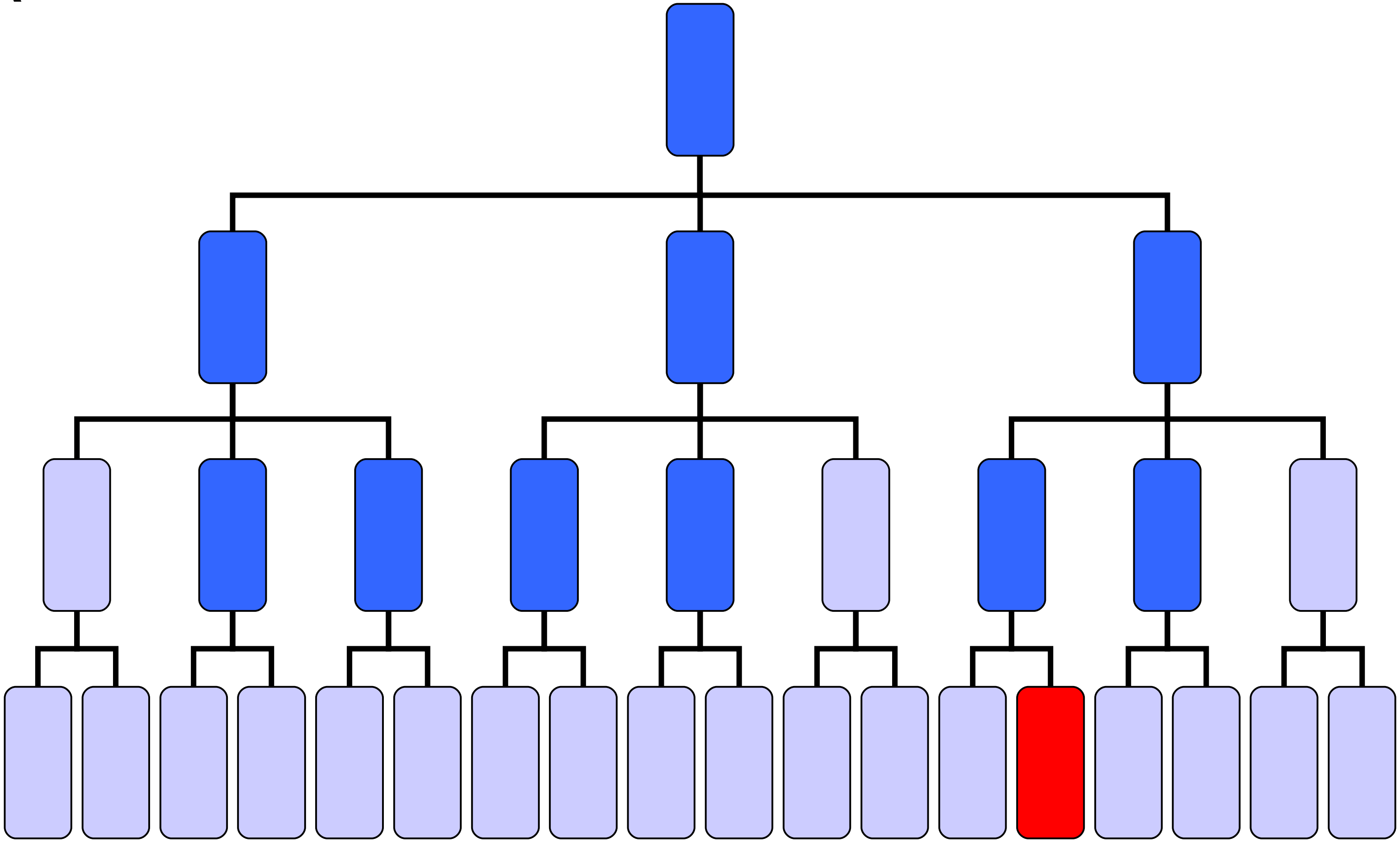
- Abstraction of Major League Baseball.
- Very difficult problem. Essentially parallel Traveling Saleman Problems.
- To date, only smallest few instances have been solved to optimality. 8 teams can take over a day of CPU time.
- Best known solutions found with metaheuristics, also require long running times to find high-quality solutions.

DFS*

- Hybridization of IDA* and depth-first branch-and-bound.
- Also known as IDA*_CR and MIDA*.
- Each iteration, increase upper bound by greater amount than IDA*.
- Final iteration, after a solution is found, continue on as depth-first branch-and-bound.

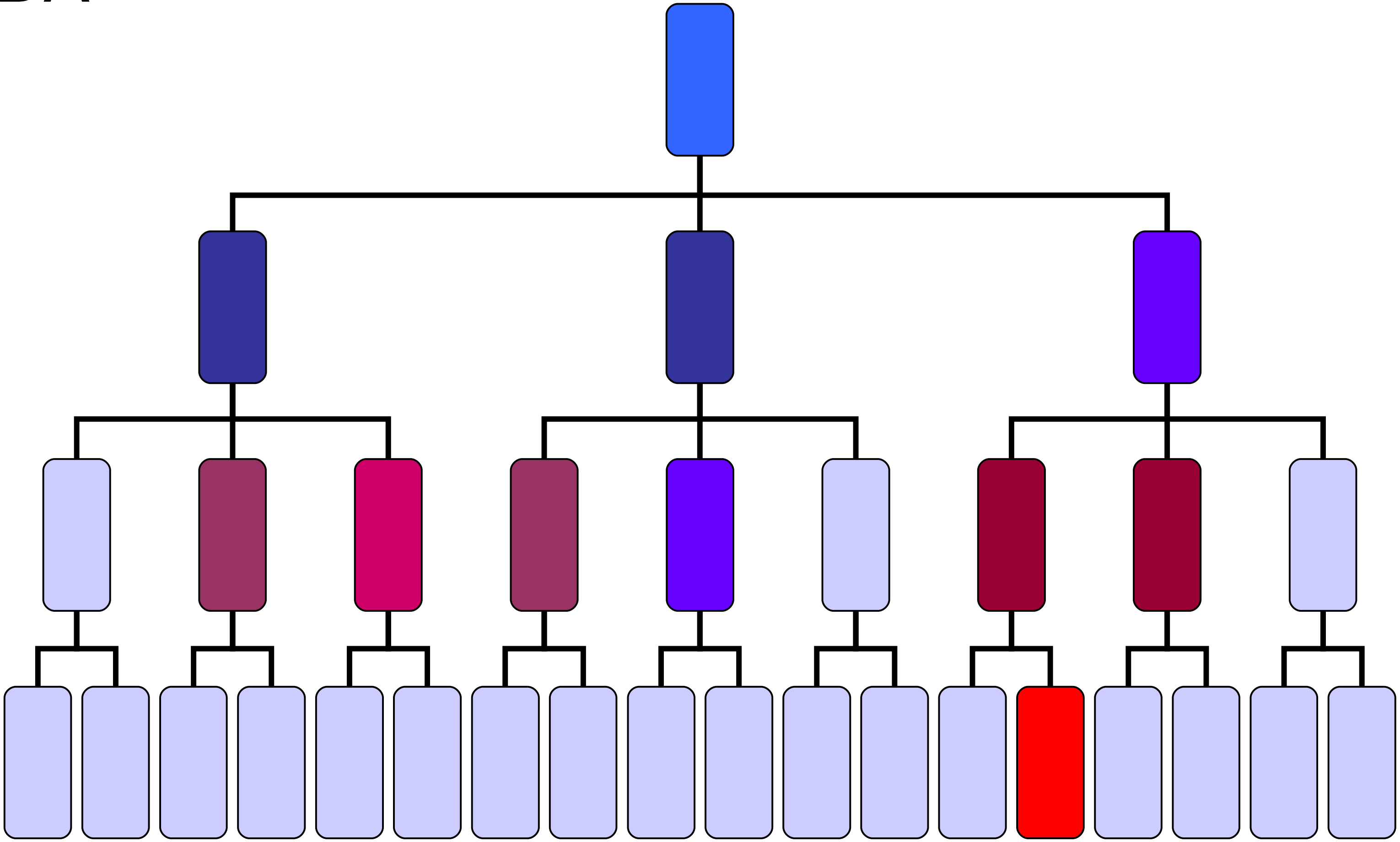
DFS*

- A*



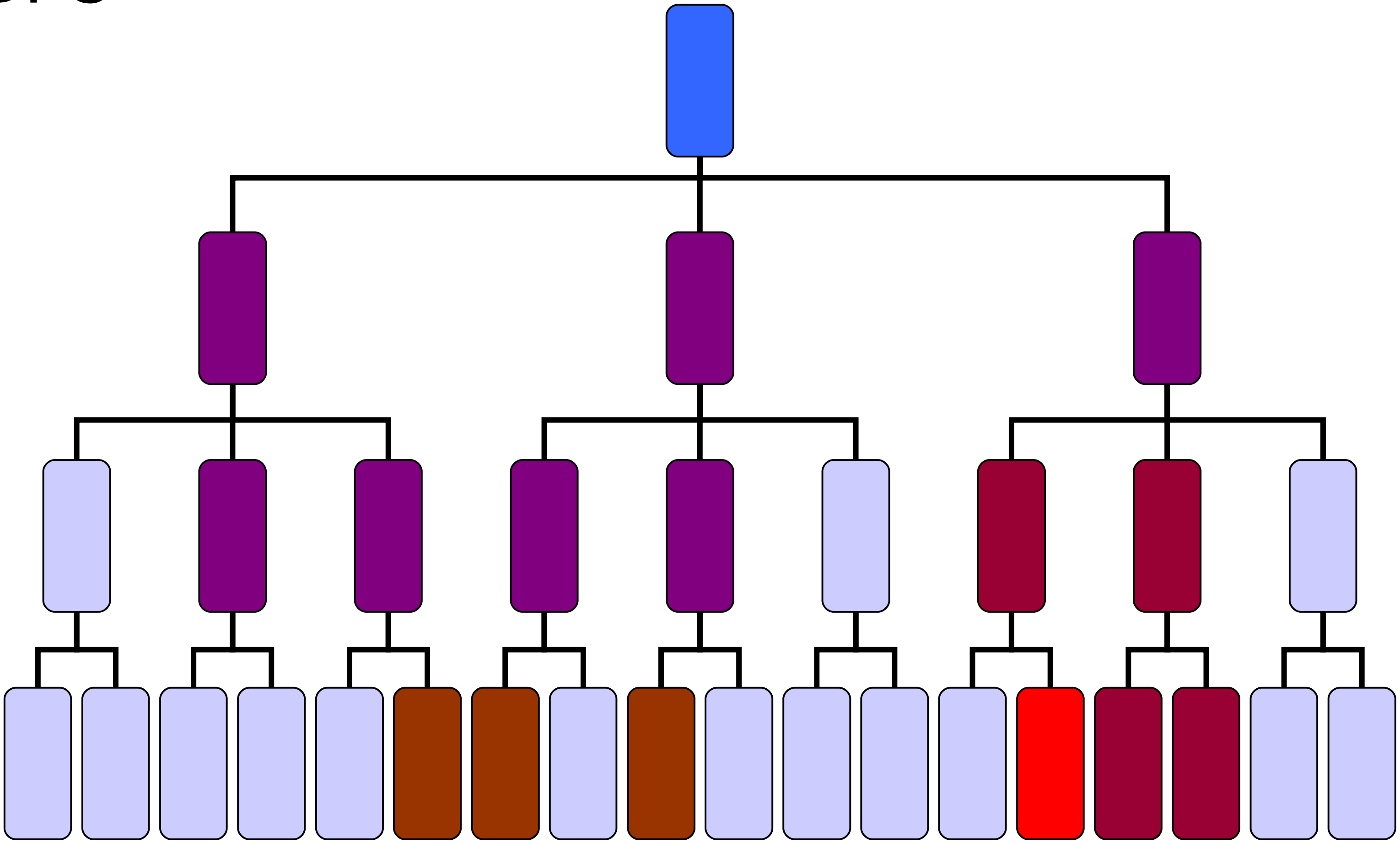
DFS*

- IDA*



DFS*

- DFS*



DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

Depth-First Search

- Pair up teams one round at a time from round 1 to *r*.
- Finish pairings teams within a round before moving to next round.

Round / Team	1	2	3	4	5	6
A	@B	@C				
B	A					
C	@D	A				
D	C					

Depth-First Search

- Easy to propagate both double round robin structure constraints and additional *at_most* and *no_repeat* constraints.
- Easy to calculate distance travelled so far.

Round / Team	1	2	3	4	5	6
A	@B	@C				
B	A					
C	@D	A				
D	C					

DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

Memory & Heuristic Estimates

- Heuristic estimates are minimal travel distance for each individual team.
- Sum individual estimates with distance traveled to get estimated total distance.

Round / Team	1	2	3	4	5	6
A	@B	@C				
B	A					
C	@D	A				
D	C					

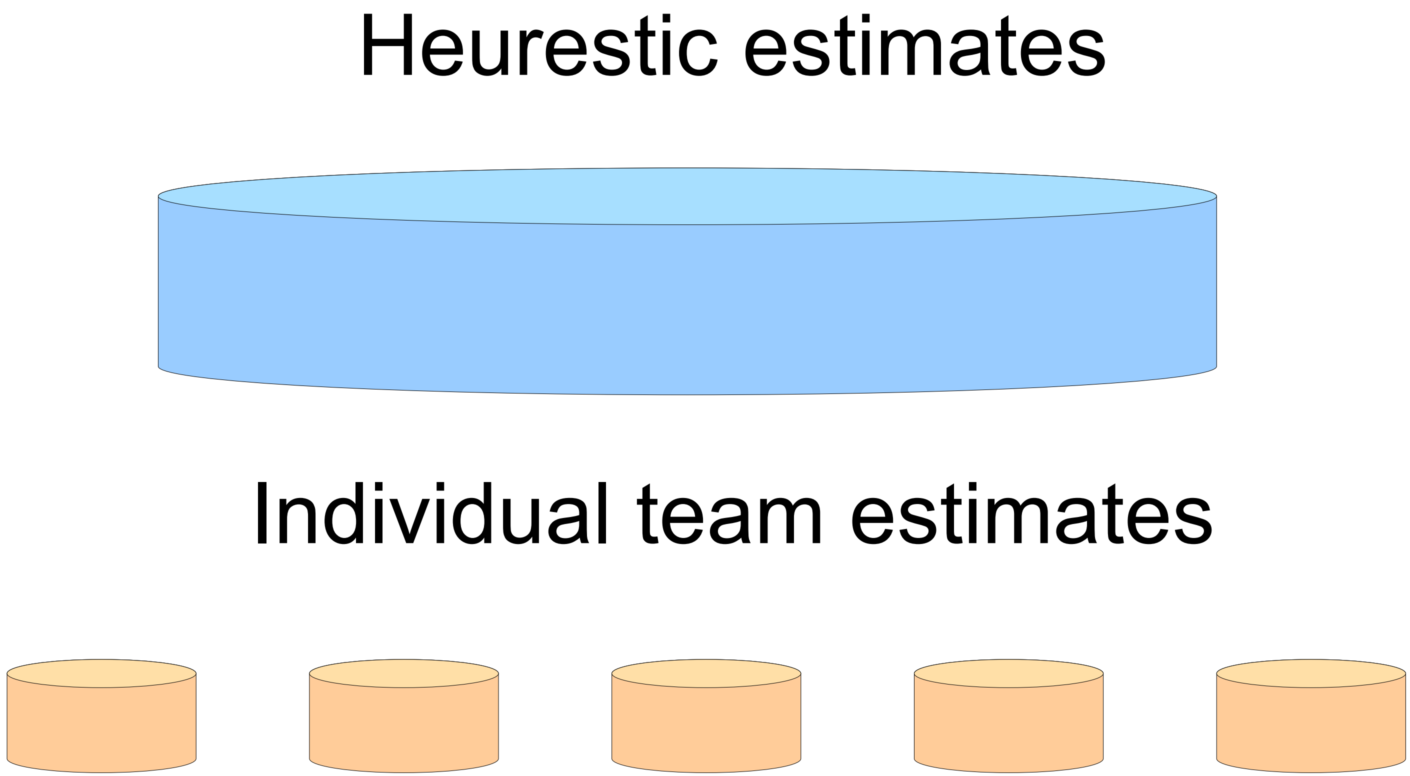
Memory & Heuristic Estimates

- Heuristic estimates expensive to calculate, each estimate similar to solving Traveling Salesman Problem.
- DFS* uses minimal memory.
- Use available memory to store all heuristic estimates in a multi-dimensional matrix.
- All that is then required is calculating index in matrix when each estimate is required.

Memory & Heuristic Estimates

- Further improve heuristic estimate usage with two-level approach for memory.
- Upper level stores all estimates.
- At lower level, each team has its own cache of current estimates.
- Calculating indices:
  - Upper level – O(n)
  - Lower level – O(1)

Memory & Heuristic Estimates



Memory & Heuristic Estimates

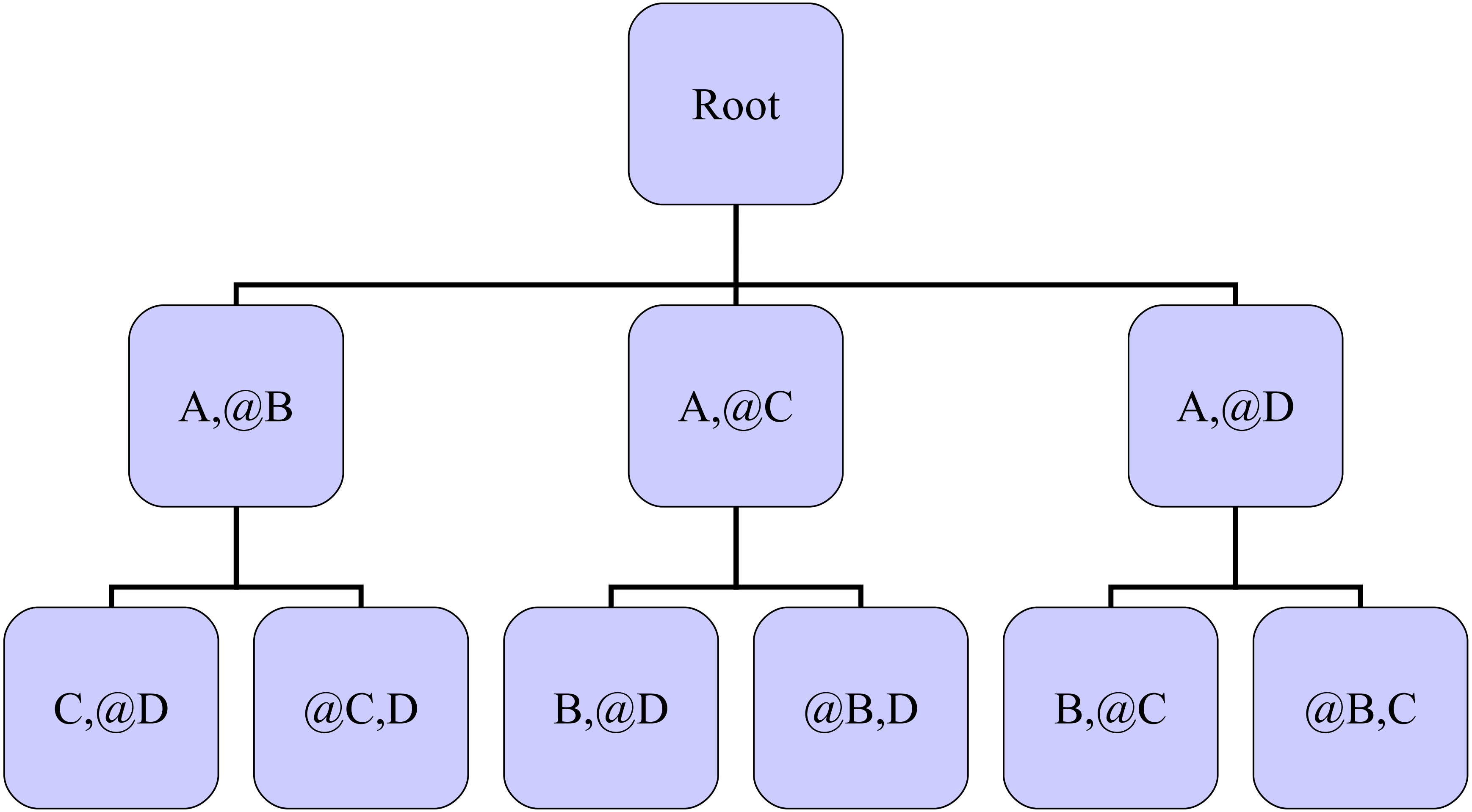
- A team's estimates not effected by other team pairings.

Round Team	1	2	3	4	5	6
A	@B	@C				
B	A					
C	@D	A				
D	C					

DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

Subtrees



Subtrees

- Each subtree consists of the first 4 pairings of depth-first search.
- Order subtrees after each iteration so most promising are tried first in final iteration.
- Used for calculating new upper bounds.
- Allows for DFS* to be parallelized.

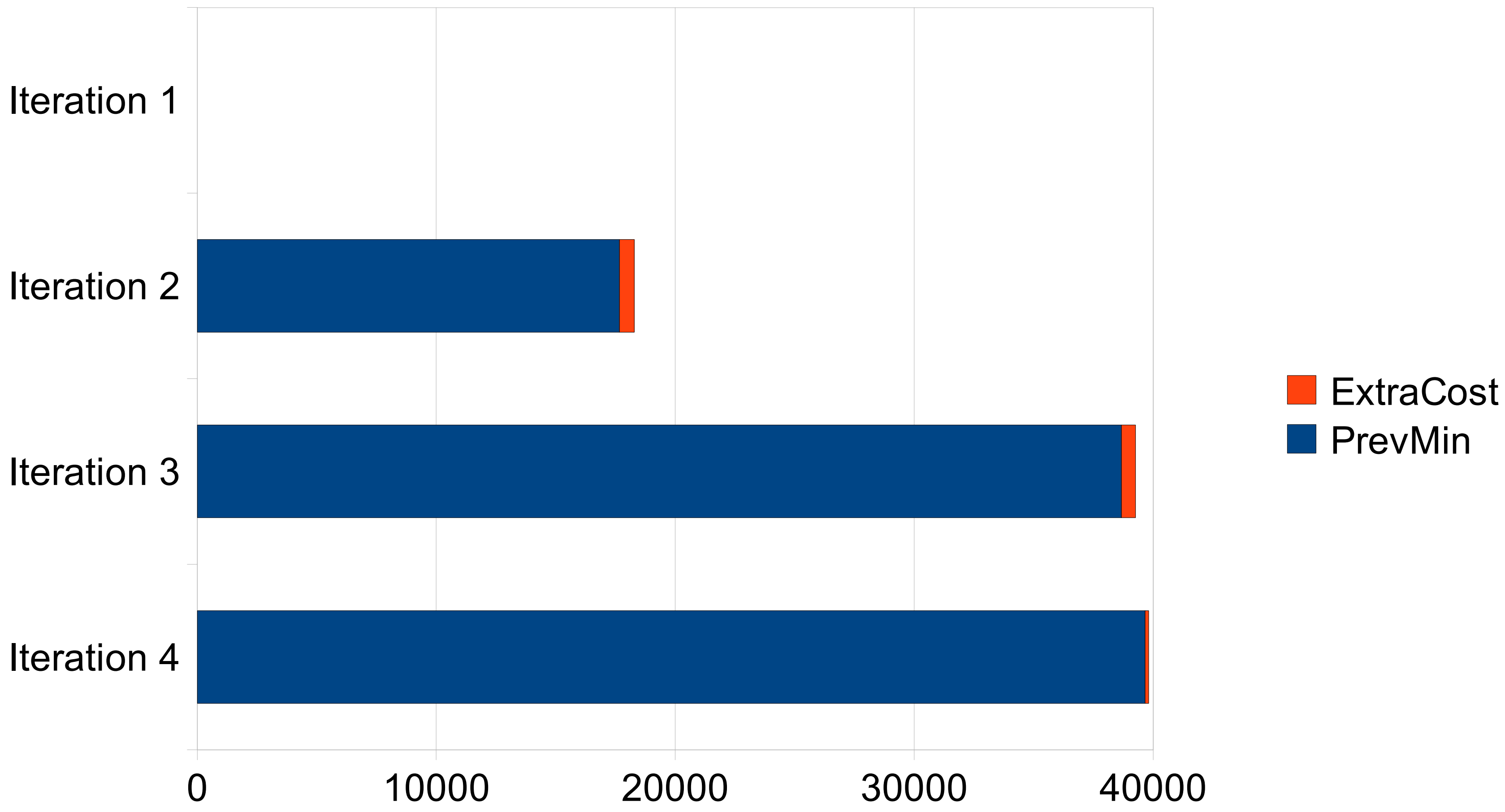
DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

New Upper Bounds

- Based off of information from subtrees.
- Takes minimal lower bound of subtree which achieved deepest depth.
- Adds small extra cost associated with deepest depth and average distance in distance matrix.
- Purpose is to decrease number of iterations.

New Upper Bounds



DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

Symmetry

- Problem is horizontally symmetrical.
- Eliminate symmetry by using first team as pivot, check make sure number remaining away is greater than remaining number home games.

Round / Team	1	2	3	4	5	6
A	@B	@C	@D	B	C	D
B	A	D	@C	@A	@D	C
C	@D	A	B	D	@A	@B
D	C	@B	A	@C	B	@A

DFS* - Components

- Depth-First Search
- Memory & Heuristic Estimates
- Subtrees
- New Upper Bounds
- Symmetry
- Parallelization

Parallelization

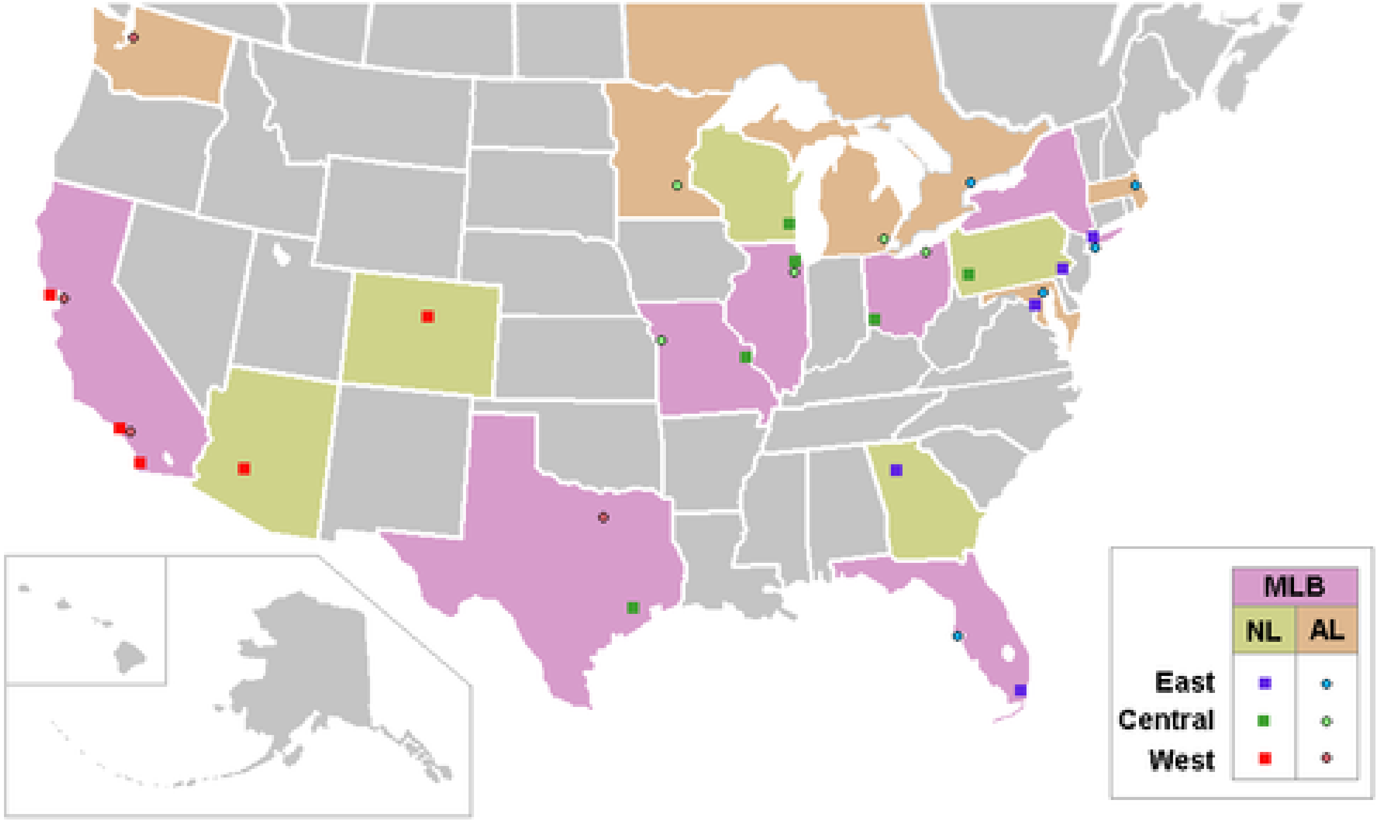
- DFS* with subtrees ideal for parallelization: very few race conditions.
- Implemented on a shared-memory multi-cpu compute server.
- Each cpu will work on a single subtree at one time.

Problem Instances

- Problem sets can be found at TTP website maintained by Michael Trick, creator of the problem.
- Problem sets vary in size.
- Smaller instances in a set are subsets of the next larger instance in set, i.e. NL4 subset of NL6.

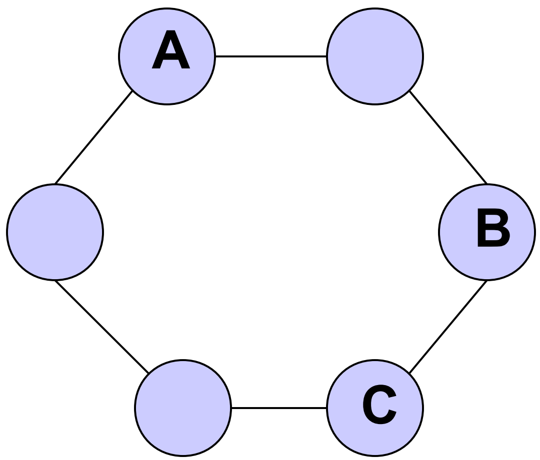
Problem Instances - NL

- NL instances: Based on real world distances of NL teams in MLB.



Problem Instances - CIRC

- CIRC instances: All teams placed on a circle, distance is minimal distance to another team going through neighbors.



Problem Instances - SUPER

- SUPER instances: Based on real world distances of Super 14 Rugby League.

Problem Instances - GALAXY

- GALAXY instances: Based on distances between 39 stars with exoplanets plus Sol.
- First problem set where distances are in a 3D plane instead of 2D.

Performance

- Using memory reduced time on NL8 from ~94,000 seconds to ~400 seconds.
- Eliminating symmetry helped to improve performance for NL instances by almost half, had smaller impact with CIRC instances.
- Parallelization helped to further reduce running time, but not 100% efficient.

Comparison

	Irnich and Schrempp	Us
NL4	<0.3 secs	0.0 secs
NL6	<19 mins	0.98 secs
NL8	<18 hrs	262.42 secs
CIRC4	<0.2 secs	0.0 secs
CIRC6	<18 hrs	2.05 secs

Other results

- First to solve CIRC8, 337 seconds required across 4 processors.
- New lower bounds found for NL10, NL12, and CIRC10.
- Introduced SUPER instances, solved team sizes 4 – 10, lower bounds only for 12 and 14 teams.
- Introduced GALAXY instances, solved team sizes 4 – 8, most difficult problem set so far.

Future – Pattern Matching

- Look into pattern matching for constraint propagation.
- Addresses the *at_most* constraint.
- Used for Ant Colony Optimization approach, found to be great for speeding up the construction of solutions.

Future – Heuristic Estimates

- Look into stronger heuristic estimates.
- Hope this will lead to better pruning of the search space for larger team sets.
- Use ideas from planning such as coupling and inconsistent heuristics.
- Possibly tie in with pattern matching.

Future – Distributed Computing

- Look into distributed computing.
- Would eliminate CPU cache misses caused by different threads needing different parts of heuristic estimates.
- Only message passing needed is for passing new/finished subroots and when new, better solutions are found.

Conclusions

- DFS* approach fastest to find known optimal solutions.
- Biggest impacts were storing heuristic estimates in memory and eliminating symmetry.
- DFS* can be easily parallelized, potential for distributed computing.

Thank You