Lecture 6:
Metagaming

- Game Optimization
- Symmetry and Factoring
- Structure Identification
- Evaluation Functions
- Monte Carlo Tree Search

Rule Ordering

Example:

```
ancestor(X,Z) <= parent(X,Y) ∧ ancestor(Y,Z)
ancestor(X,Y) <= parent(X,Y)
```

Better Version (for generating solutions to `ancestor`):

```
ancestor(X,Y) <= parent(X,Y)
ancestor(X,Z) <= parent(X,Y) ∧ ancestor(Y,Z)
```

Conjunct Ordering (1)

Example:

```
goal(X,Z) <= p(X,Y) ∧ q(Y,Z) ∧ distinct(Y,b)
```

Better:

```
goal(X,Z) <= p(X,Y) ∧ distinct(Y,b) ∧ q(Y,Z)
```

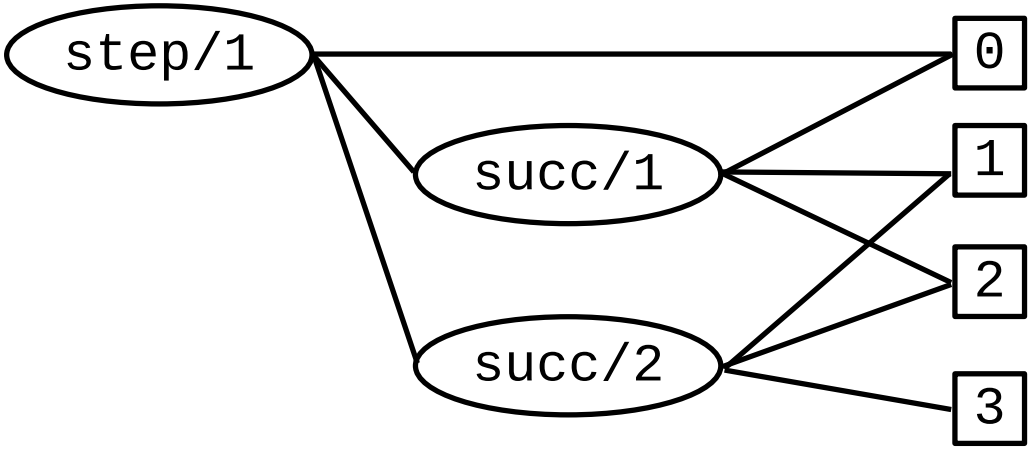
Worse:

```
goal(X,Z) <= distinct(Y,b) ∧ p(X,Y) ∧ q(Y,Z)
```

Domains

The domains of the fluents can be determined from the roles of the game with the help of the dependency graph

```
succ(0,1) ∧ succ(1,2) ∧ succ(2,3)
init(step(0))
next(step(X)) <= true(step(Y)) ∧ succ(Y,X)
```



Conjunct Ordering (2)

Example:

```
wins(P) <= true(cell(X,Y,P)) ∧ corner(X,Y) ∧ queen(P)
```

Solution Set Sizes:

```
|true(cell(X,Y,P))| = 768
|corner(X,Y)|      =   4
|queen(P)|         =   2
```

Better Version:

```
wins(P) <= queen(P) ∧ corner(X,Y) ∧ true(cell(X,Y,P))
```

Data Extraction

Original:

```
p(10) <= q(a)
p(20) <= q(b)
p(30) <= q(c)
q(X)  <= ...
```

Assumptions:

- q is expensive to compute
- as easy to generate answers as to check answers

New, improved Version:

```
p(Y) <= q(X) ∧ r(X,Y)
r(a,10)
r(b,20)
r(c,30)
q(X) <= ...
```

Database Views

The ancestor relation `a` is the transitive closure of the parent relation `p`:

```
a(X,Y) <= p(X,Y)
a(X,Z) <= a(X,Y) ∧ a(Y,Z)
```

The “samefamily” relation `sf` is true of all pairs of people that are relatives, i.e., that have a common ancestor:

```
sf(Y,Z) <= a(X,Y) ∧ a(X,Z)
```

Using Materialized Views

```
a(X,Y) <= p(X,Y)
a(X,Z) <= a(X,Y) ∧ a(Y,Z)

sf(Y,Z) <= a(X,Y) ∧ a(X,Z)
```

If we materialize the view `a` or `sf` then we increase the computational efficiency of answering the query `sf`.

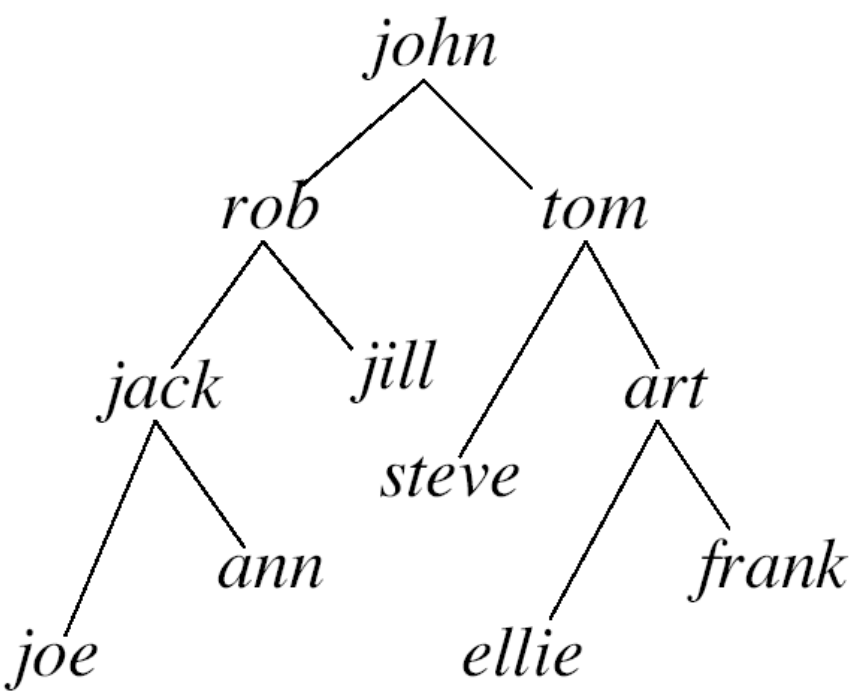
If we do not materialize the views `a` or `sf` then we decrease the amount of database storage space (space economy).

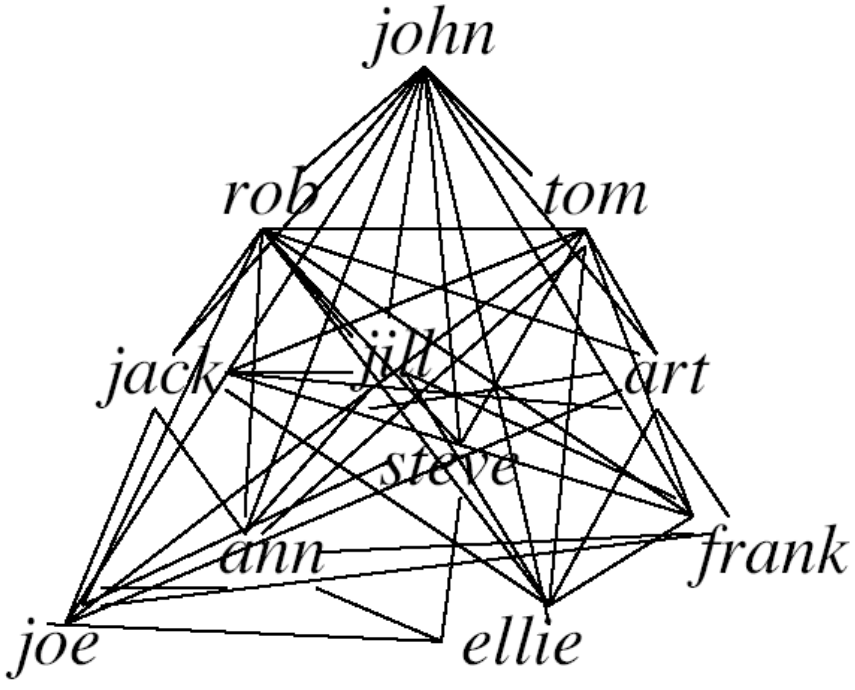
What are the optimal views to materialize? Database reformulation gives answers.

Querying Data Faster: Ideas

How about precomputing all elementary queries?

Not always a great idea (materializing `sf`)

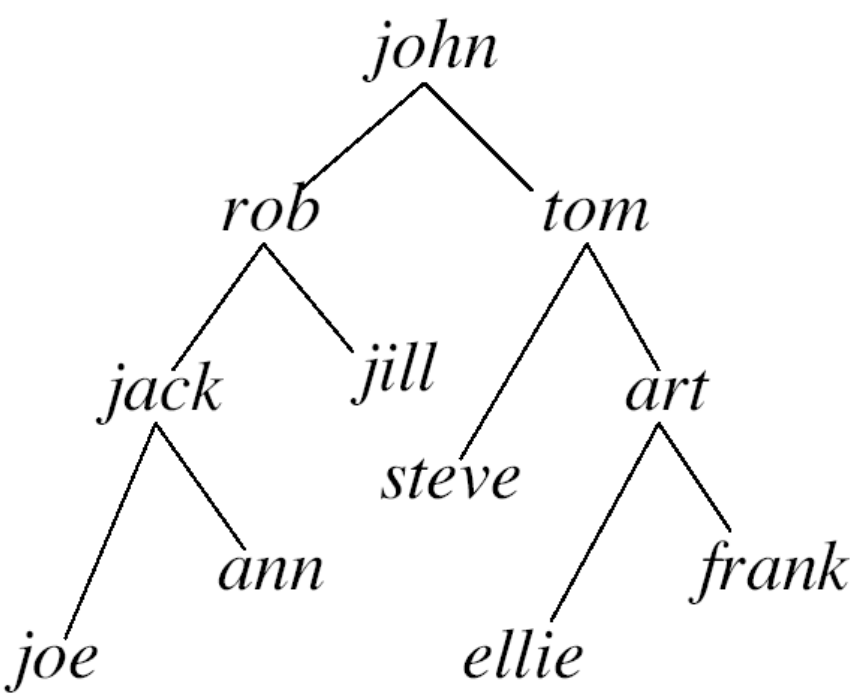


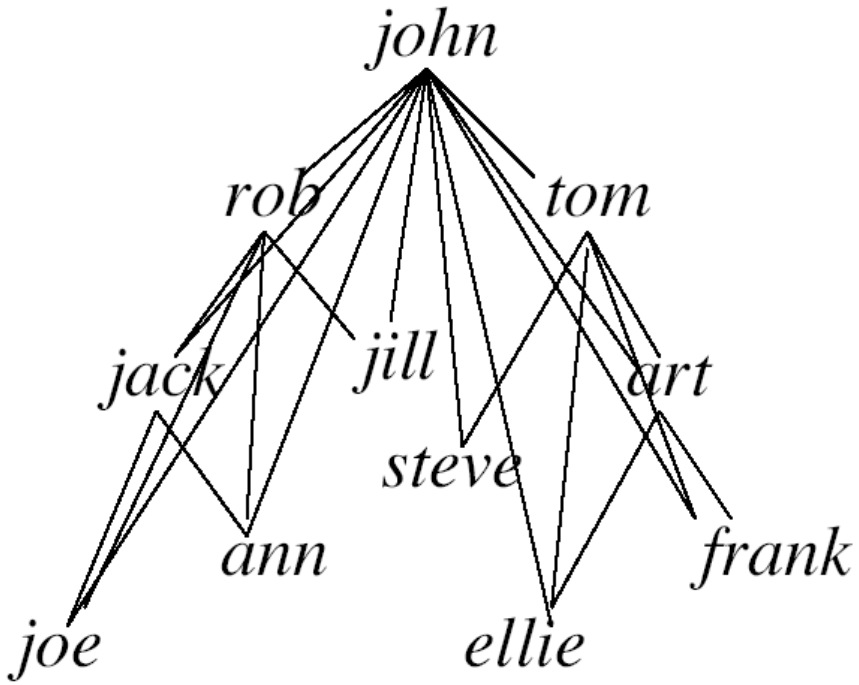


Querying Data Faster: Ideas

How about precomputing predefined views?

Not too good, either (materializing a)

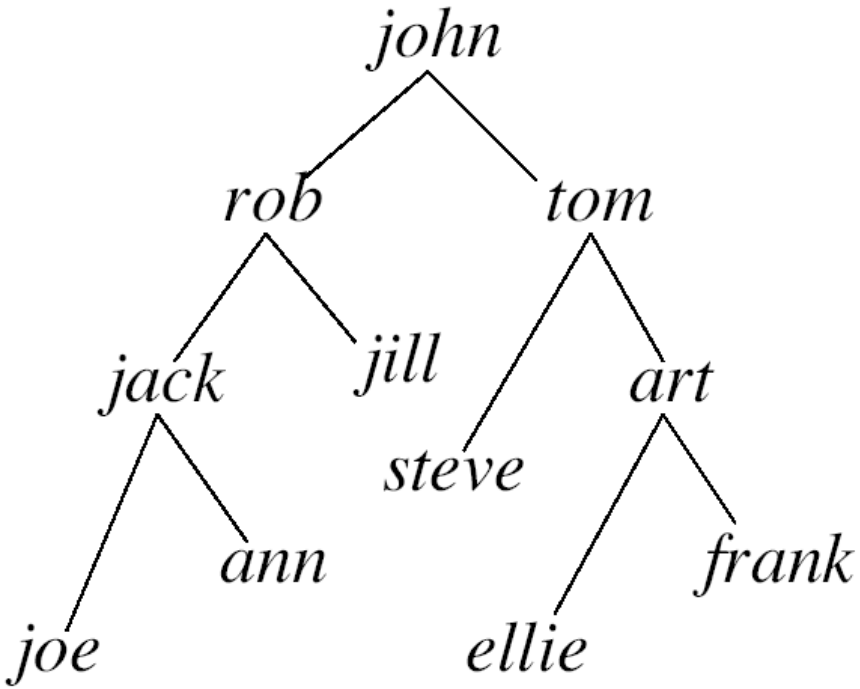


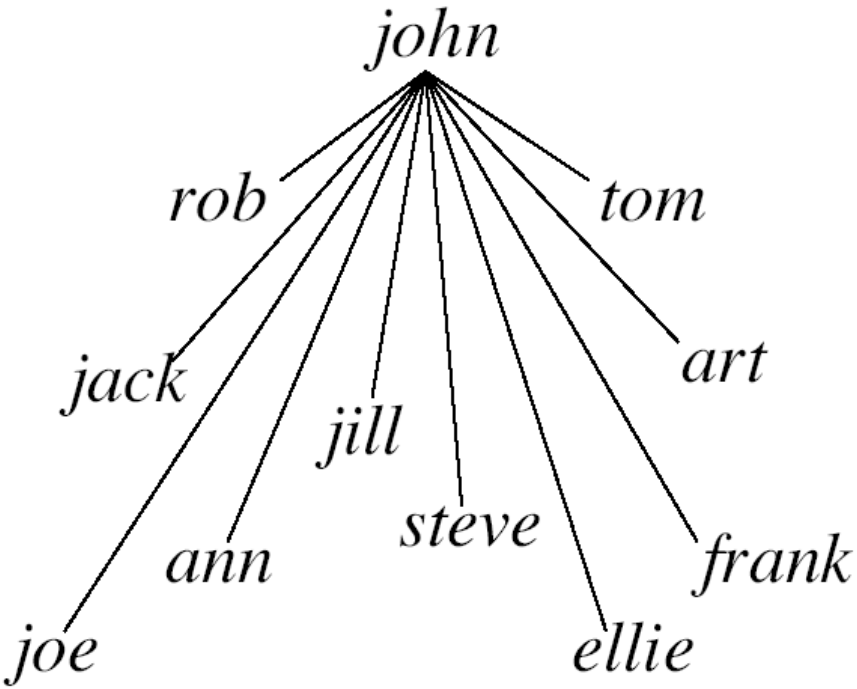


Querying Data Faster: Ideas

How about precomputing “pieces” of the elementary queries?

Materializing new views that are not already defined on the database: *relational reformulation*





May Need to Invent New Relations

Ancestor a: `a(X,Y) <= p(X,Y)`
`a(X,Y) <= a(X,Z) ∧ a(Z,Y)`

Same family sf: `sf(X,Y) <= a(Z,X) ∧ a(Z,Y)`

New: “has parent” `hp(X) <= p(Z,X)`

New: “founding father” `ff(X,Y) <= a(X,Y) ∧ ¬hp(X)`

New: a rewriting of sf in terms of ff:

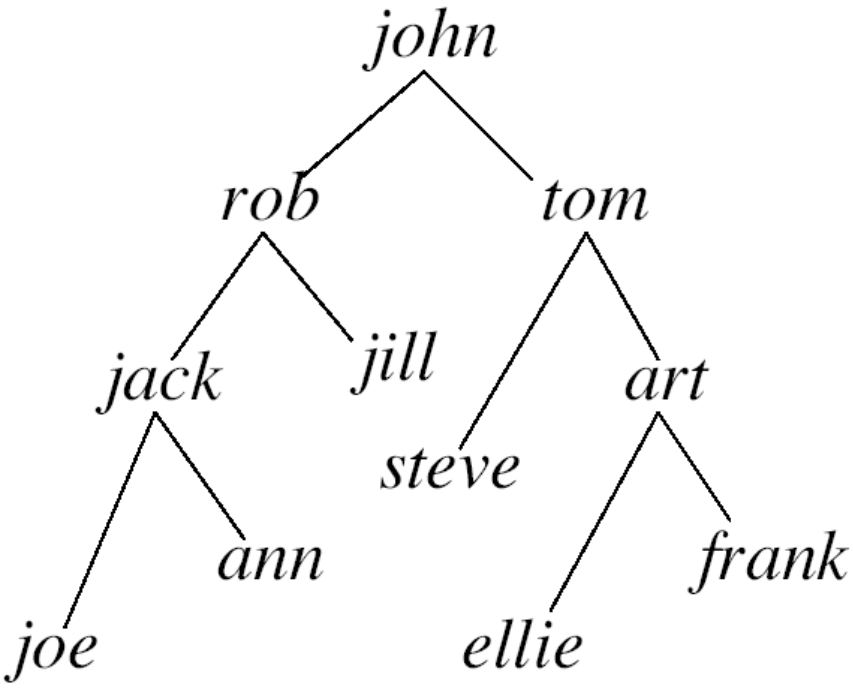
`sf(X,Y) <= ff(Z,X) ∧ ff(Z,Y)`

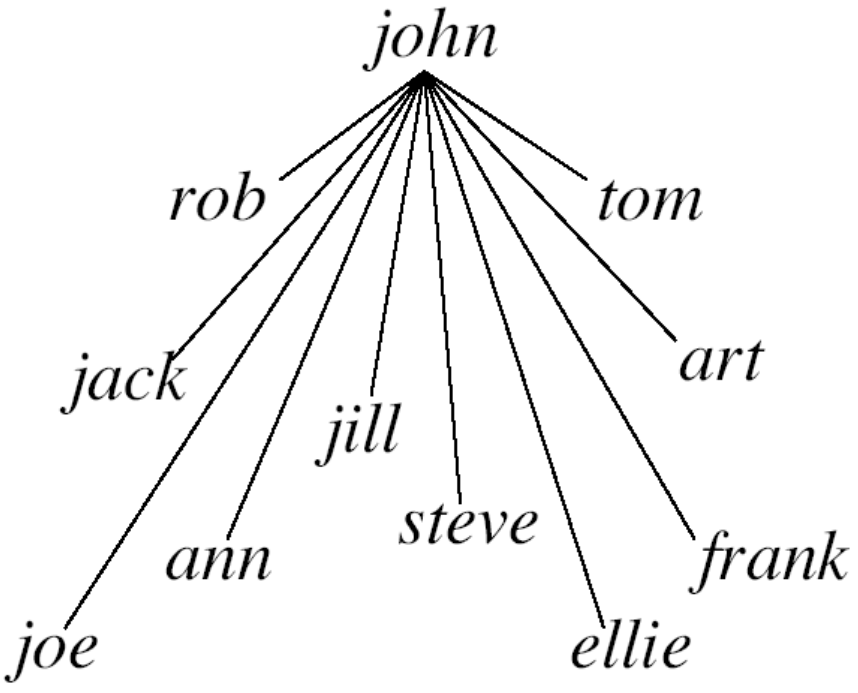
Reformulating Samefamily

Has parent: `hp(X) <= p(Z,X)`

Founding father: `ff(X,Y) <= a(X,Y) ∧ ¬hp(X)`

A rewriting of `sf` in terms of `ff`: `sf(X,Y) <= ff(Z,X) ∧ ff(Z,Y)`





Symmetry

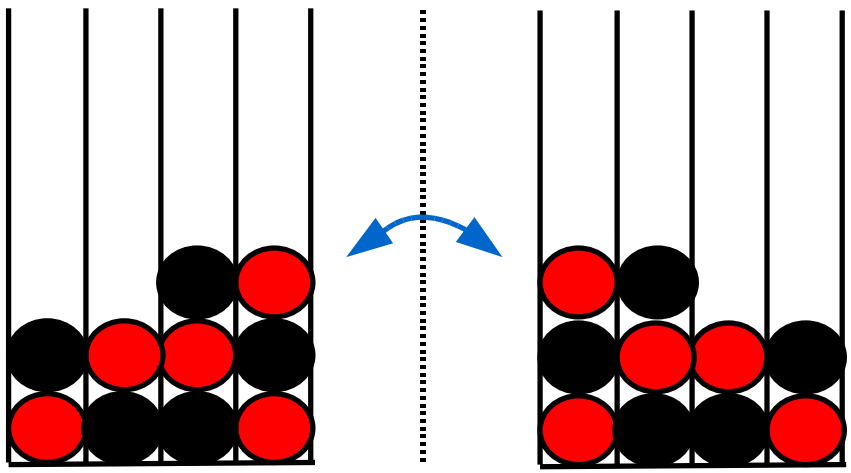
Symmetries can be logically derived from the rules of a game.

A symmetry relation over the elements of a domain is an equivalence relation such that

- two symmetric states are either both terminal or non-terminal
- if they are terminal, they have the same goal value
- if they are non-terminal, the legal moves in each of them are symmetric and yield symmetric states

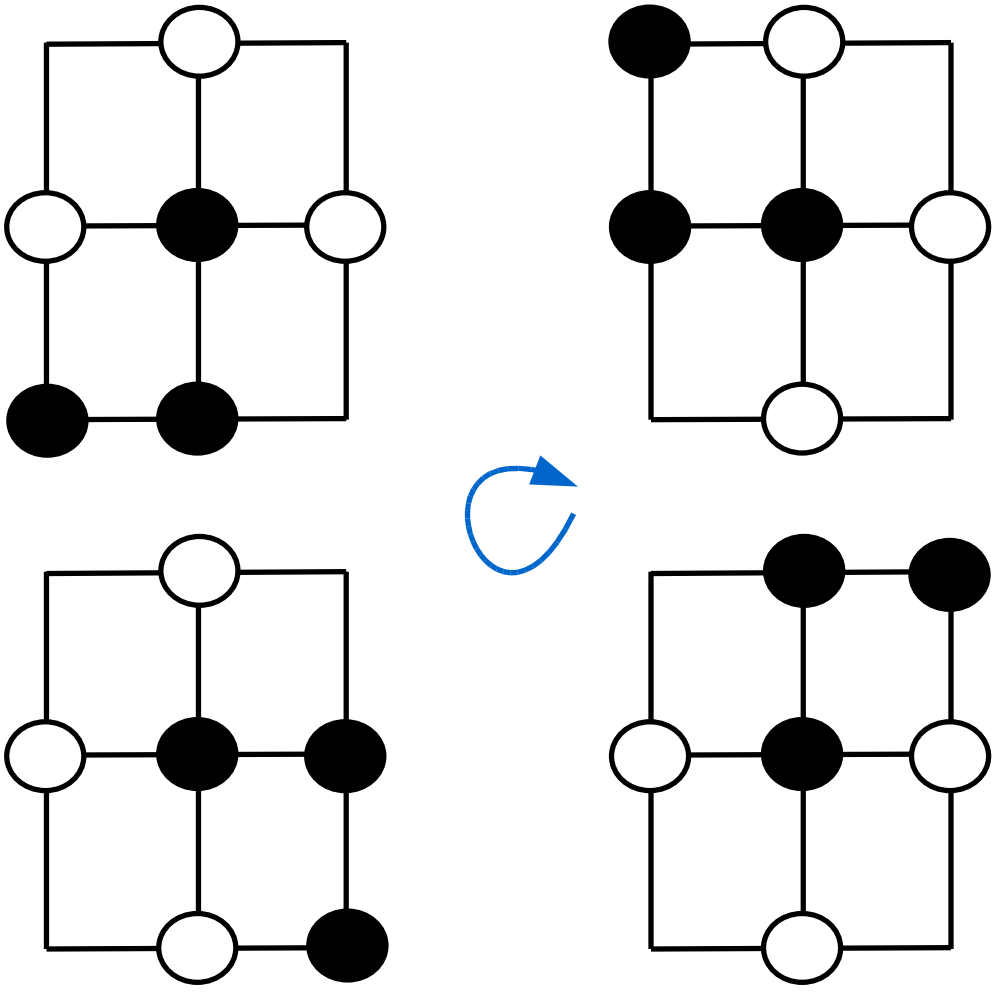
Example: Individual pebbles in Othello or Go

Reflectional Symmetry



Connect-3

Rotational Symmetry

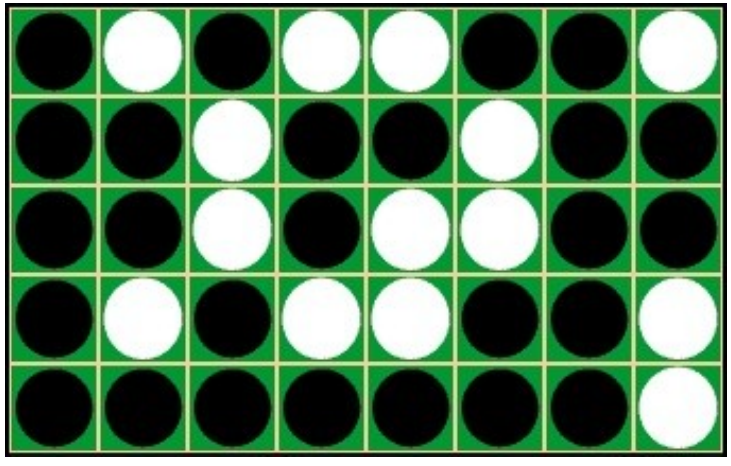


Capture Go

Factoring Example

Hodgepodge = Chess + Othello

Branching factor: a



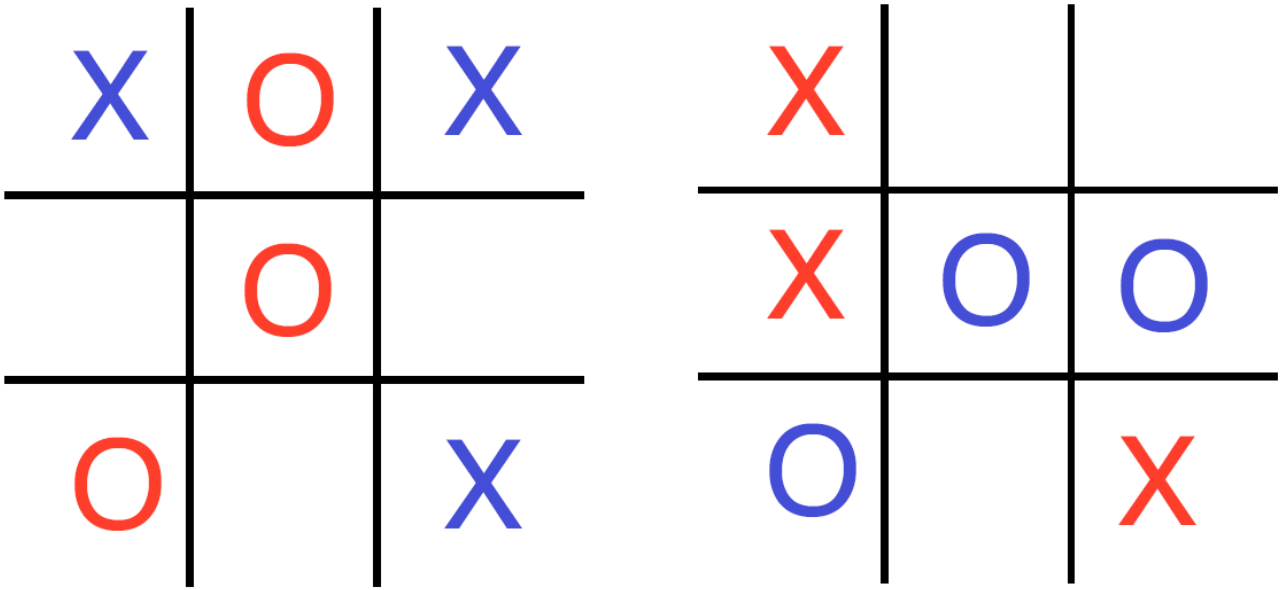
Branching factor: b

Branching factor as given to players: $a * b$

Fringe of tree at depth n as given: $(a * b)^n$

Fringe of tree at depth n factored: $a^n + b^n$

Double Tic-Tac-Toe



Branching factor: 81, 64, 49, 36, 25, 16, 9, 4, 1

Branching factor (factored): 9, 8, 7, 6, 5, 4, 3, 2, 1 (x 2)

Game Factoring and its Use

1. Compute factors
   - Behavioral factoring
   - Goal factoring
2. Play factors
3. Reassemble solution
   - Append plans
   - Interleave plans
   - Parallelize plans with simultaneous actions

Behavioral Factoring

A set $\mathscr{F}$ of fluents and moves is a *behavioral factor* if and only if there are no connections between the fluents and moves in $\mathscr{F}$ and those outside of $\mathscr{F}$.

Goal Factoring

Simple Case – goal is a conjunction

- Partition conjuncts over behavioral factors
- Create new goals for each factor

Complex Case – goal is a disjunction of conjunctions

- Split each conjunct as above
- Check for lossless joins, i.e. when recombined, we get the same results

Good:

$(p1 \wedge q1) \vee (p1 \wedge q2)$

Bad:

$(p1 \wedge q1) \vee (p2 \wedge q2)$

Blind Search

- Blind search: only assign scores to nodes based on the evaluation of the complete subtrees at those nodes
- Problem: can relatively rarely see all the way to the bottom of the tree for a single node, even less so for every successor node
- Solution: improve efficiency of inference
- Solution: assign intermediate scores to nodes based on an evaluation function
- *Metagaming* means to reason about properties of games

Designing Evaluation Functions

- Typically designed by programmers/humans
- A great deal of thought and empirical testing goes into choosing one or more good functions
- E.g.
  - piece count, piece values in chess
  - holding corners in Othello
- But this requires knowledge of the game's structure, semantics, play order, etc.

The General Case

- No knowledge of features
- No insight into game structure
- No intuition about what is a good feature for this particular game

- Some general ideas work in many cases – but sometimes they don't ...
- E.g. mobility heuristics, novelty heuristics, goal distance

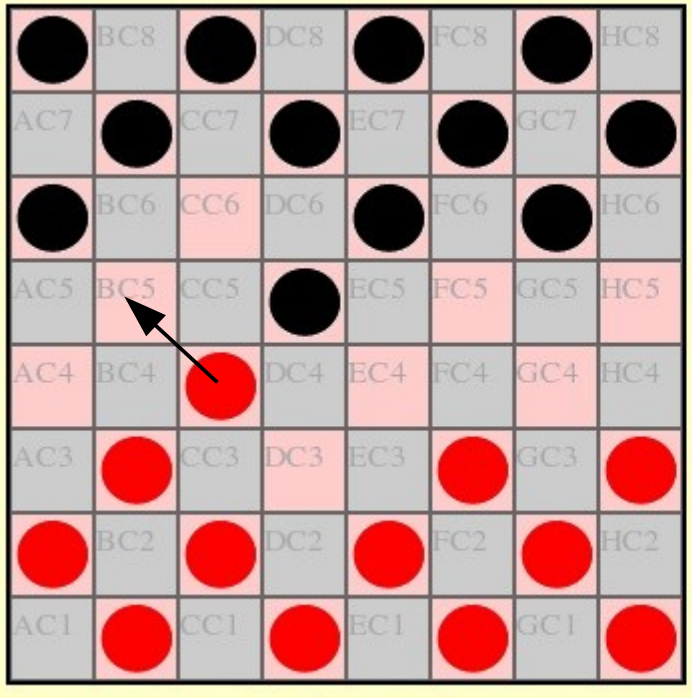
Mobility

- More moves means better state
  Optionally: limiting opponent moves is better too

- The good:
  In many games, being cornered or forced into making a move is quite bad
  - In Chess, when you are in check, you can do relatively few things compared to not being in check
  - In Othello, having few moves means you have little control of the board

- The bad: Mobility is disastrous for Checkers

Worldcup 2006: Cluneplayer vs. Fluxplayer



Piece Count BLACK: 12 RED: 12

Playclock:

Roles:

Red	Black
CLUNEPLAYER	FLUXPLAYER

Last Moves (step 2):

Red	Black
noop	move(bp,c,c6,d,c5)

Inverse Mobility

- Having fewer things to do is better
  Optionally: giving opponent things to do is better
- This works in some games, like Nothello, where you might in fact want to *lose pieces*
- How to decide between mobility and inverse mobility heuristics?

Novelty

- Changing the game state is better
- The good:
  - Changing things as much as possible can help avoid getting stuck
  - When it is unclear what to do, maybe the best thing is to throw in some directed randomness
- The bad:
  - Changing the game state can happen if you throw away your own pieces ...
  - Unclear if novelty per se actually goes anywhere useful for anybody

Identifying Structures: Relations

A successor relation is a binary relation that is antisymmetric, functional, and injective.

Example:

```
succ(1,2) ∧ succ(2,3) ∧ succ(3,4) ∧ ...
next(a,b) ∧ next(b,c) ∧ next(c,d) ∧ ...
```

An order relation is a binary relation that is antisymmetric and transitive.

Example:

```
lessthan(A,B) <= succ(A,B)
lessthan(A,C) <= succ(A,B) ∧ lessthan(B,C)
```

Boards and Pieces

An (*m*-dimensional) board is an *n*-ary fluent ($n \geq m+1$) with

- *m* arguments whose domains are successor relations
- 1 output argument

Example:

`cell(a,1,whiterook)` $\wedge$ `cell(b,1,whiteknight)` $\wedge$ `...`

A marker is an element of the domain of a board's output argument.
A piece is a marker which is in at most one board cell at a time.

Example: Pebbles in Othello, White King in Chess

Goal Distance

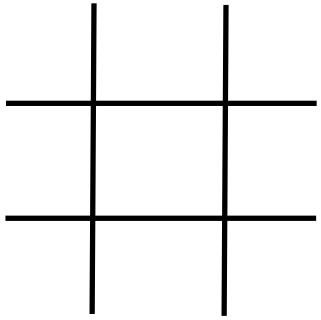
- The better an intermediate state satisfies the goal specification, the better it is
- Fuzzy Logic (t-norms) to evaluate a goal formula
- Value *0.5 < p <1.0* for true literals (and *1-p* for false literals)
- Non-standard t-norm with threshold to avoid values close to 0 for large conjunctions

$$\text{eval}(F \wedge G) = T(\text{eval}(F),\text{eval}(G))$$
$$\text{eval}(F \vee G) = \text{co-}T(\text{eval}(F),\text{eval}(G))$$
$$\text{eval}(\neg F) = 1 - \text{eval}(F)$$

Example: Tic-Tac-Toe

```
goal(xplayer,100) <= true(cell(M,1,x)) ∧
                     true(cell(M,2,x)) ∧
                     true(cell(M,3,x))
                     ∨
                     true(cell(1,N,x)) ∧
                     true(cell(2,N,x)) ∧
                     true(cell(3,N,x))
                     ∨
                     true(cell(1,1,x)) ∧
                     true(cell(2,2,x)) ∧
                     true(cell(3,3,x))
                     ∨
                     true(cell(1,3,x)) ∧
                     true(cell(2,2,x)) ∧
                     true(cell(3,1,x))
```

Evaluation of Intermediate States



eval(goal(xplayer,100)) after does(xplayer,mark(2,2))
> eval(goal(xplayer,100)) after does(xplayer,mark(1,1))
> eval(goal(xplayer,100)) after does(xplayer,mark(1,2))

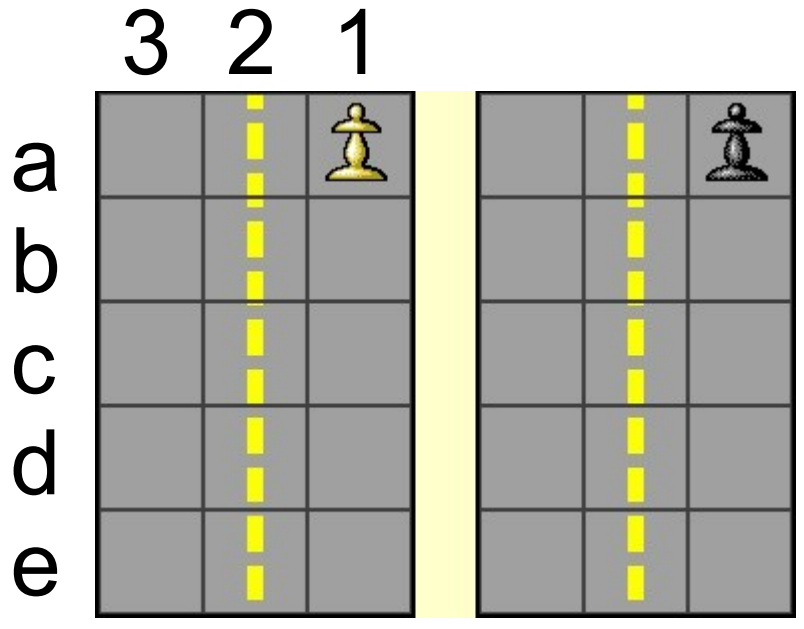
Goal Distance (2)

The closer the current value of a functional fluent to the target value, the “less false” is the corresponding goal literal

- Remember how successor relations and order relations can be identified
- These relations define metrics Δ on functional fluents wrt. the output argument
- Truth degree of `true(f(x,a))` given that `true(f(x,b))`:

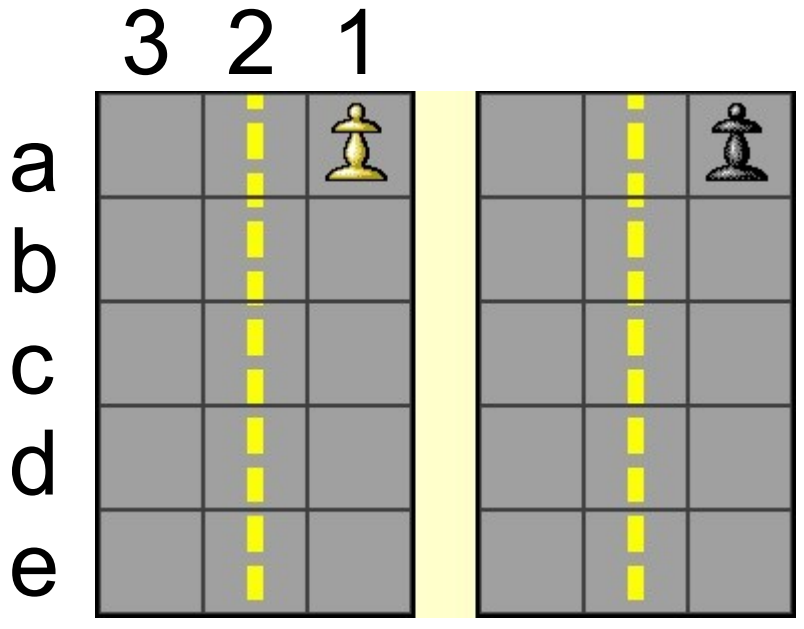
$$(1-p)-(1-p)*\frac{\Delta(b,a)}{|dom(f(\vec{x}))|}$$

Example: The Goal in Racetrack




```
goal(white,100) <= true(lane(white,e))
init(lane(white,a))
```

Evaluation of Intermediate States

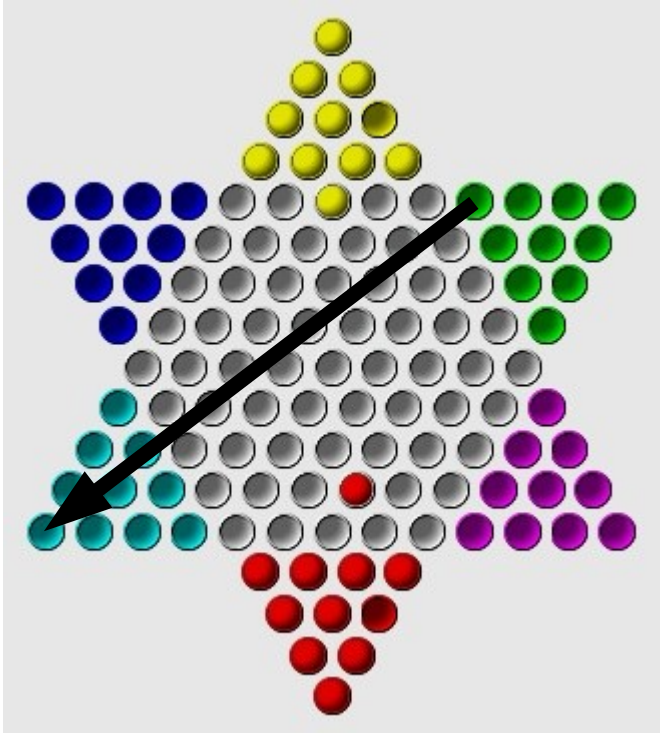


Δ(b,e) = 3 < Δ(a,e) = 4, hence:

eval(`goal(white,100)`) after `does(white,move(a,1,a,2))`

< eval(`goal(white,100)`) after `does(white,move(a,1,b,1))`

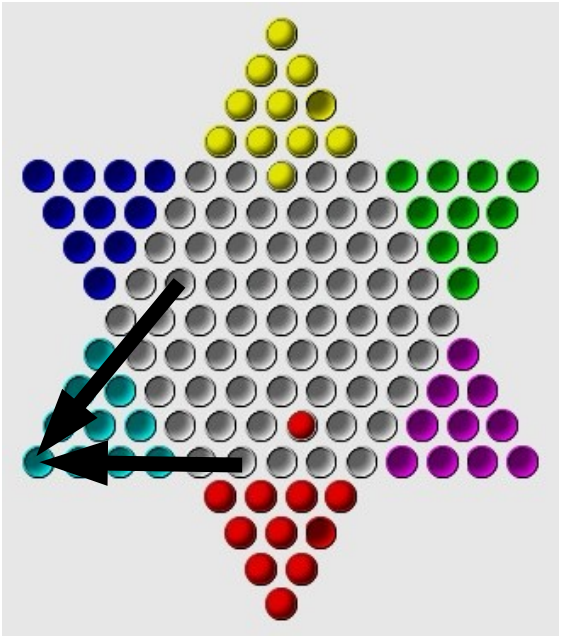
Another Example




```
init(cell(green,j,13))∧ ...
goal(green,100) <= true(cell(green,e,5)) ∧ ...
```

Chinese checkers continued

(f,10)



(e,5)

(j,5)

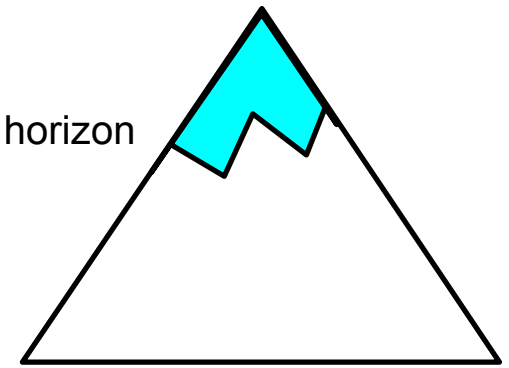
$\Delta((j,5),(e,5))=5 \quad < \quad \Delta((f,10),(e,5))=6$

Assessment

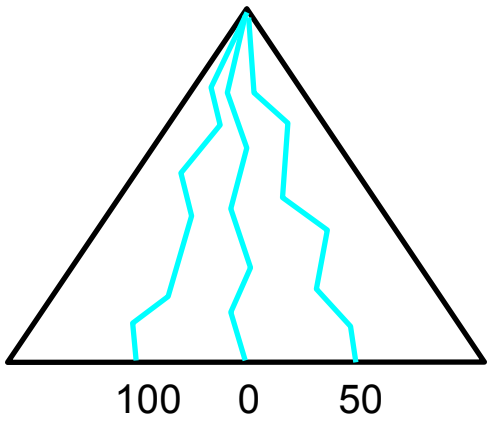
Fuzzy goal evaluation works particularly well for games with

- independent sub-goals
  15-Puzzle
- converge to the goal
  Chinese Checkers
- quantitative goal
  Othello
- partial goals
  Peg Jumping, Chinese Checkers with >2 players

Monte Carlo Tree Search (1)



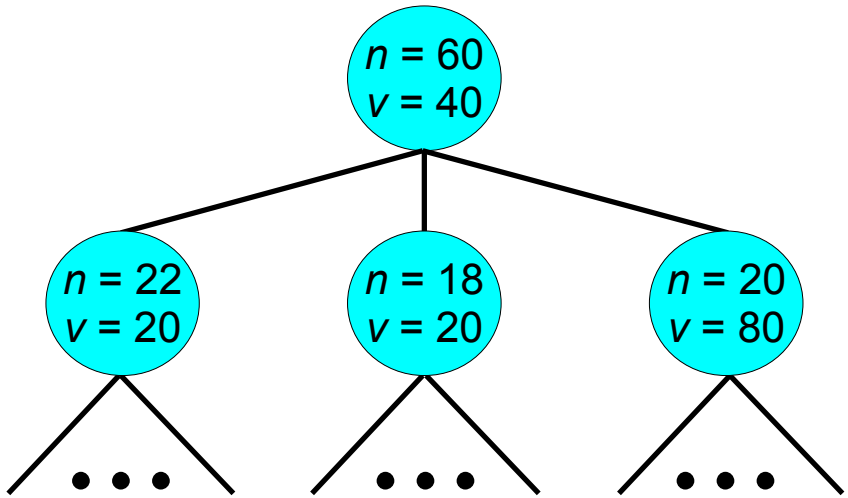
Game Tree Seach



MC Tree Search

Monte Carlo Tree Search (2)

Value of move = Average score returned by simulation

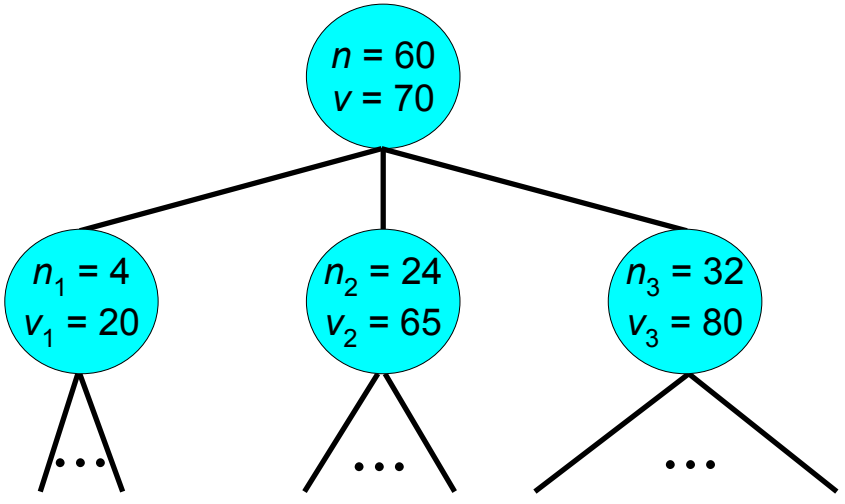


Confidence Bounds

- Play one random game for each move
- For next simulation choose move

$$argmax_i \left(v_i + C * \sqrt{\frac{\log n}{n_i}} \right)$$

confidence bound



Assessment

Monte Carlo Tree Search works particularly well for games which

- converge to the goal
  Checkers
- reward greedy behavior
- have a large branching factor
- do not admit a good heuristics

Schedule

Lectures		Tutorials
Oct, 13th, 2008	Introduction	Oct, 20th, 2008
Oct, 27th, 2008	Game Description Language	Nov, 3rd, 2008
Nov, 10th, 2008	State-Space Search and Planning	Nov, 17th, 2008
Nov, 24th, 2008	Incomplete Information	Dec, 1st, 2008
Dec, 8th, 2008	Automated Reasoning	Dec, 15th, 2008
Jan, 5th, 2009	Metagaming	Jan, 12th, 2009
Jan, 26th, 2009	Game Theory	Feb, 2nd, 2009
Apr, 2nd, 2009	Competition	