**Min Conflicts and Backtracking; both search**

\_ Initial state: {} – all variables are unassigned

\_ Successor function: a value is assigned to one of the unassigned variables with no conflict

\_ Goal test: a complete assignment

\_ Path cost: a constant cost for each step

\_ Solution appears at depth *n* if there are *n* variables

\_ Depth-first or local search methods work well

\_ Discrete variables- finite domains:

\_ *n* variables, domain size *d* \_ *O* (*d n*) complete assignments

\_ e.g., Boolean CSPs, such as 3-SAT (NP-complete)

\_ Worst case, can’t solve finite-domain CSPs in less than exponential time

infinite domains: integers, strings, etc.

ex.job scheduling, variables are start/end days for each job

\_ need a constraint language, e.g., *StartJob1 + 5 ≤ StartJob3*

\_ Continuous variables

\_ e.g., start/end times for Hubble Space Telescope observations

\_ linear constraints solvable in polynomial time by linear programming

\_ Unary constraints involve a single variable,

\_ e.g., SA ≠ green

\_ Binary constraints involve pairs of variables,

\_ e.g., SA ≠ WA

\_ Higher-order constraints involve 3 or more

variables\_ e.g., crypt arithmetic column constraints

\_ Since we can formulate CSP problems as standard search problems, we can apply search algorithms from chapter 3,4

\_ If breadth-first search were applied, branching factor? *nd*

tree size? *nd* \* (*n*-1)*d* \* … \* *d* = *n*! \* *dn* leaves

\_ complete assignments? *dn*

\_ A crucial property to all CSPs: commutativity

\_ the order of application of any given set of actions has no

effect on the outcome

\_ Variable assignments are commutative, i.e., [ WA = red

then NT = green ] same as [ NT = green then WA = red ]

\_ Only need to consider assignments to a single variable at each node \_ *b* = *d* and there are *d n* leaves

\_ Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign

**Minimum remaining values** (MRV)

\_ choose the variable with the fewest “legal” values

\_ also called most constrained variable or fail-first heuristic

\_ does it help in choosing the first variable?

**Most constraining variable**:

\_ selecting the variable that has the largest number of constraints on other unassigned variables

\_ also called degree heuristics

\_ Tie-breaker among MRV

Given a variable, choose the **least constraining value**:

\_ the one that rules out the fewest values in the remaining variables

\_ Combining these heuristics makes 1000 queens feasible

**Forward Checking**

-Keep track of remaining legal values for unassigned

variables

-Terminate search when any variable has no legal values

-Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures

**Constraint propagation** repeatedly enforces constraints locally by propagating implications of a constraint of one variable onto other variables

**AC-3 Time complexity** *O* (*n* 2*d* 3) - Simplest form of propagation makes each arc consistent X *Y* is consistent iff for every value *x* of *X* there is some allowed *y*

\_ If *X* loses a value, neighbors of *X* need to be rechecked

\_ Arc consistency detects failure earlier than forward checking

\_ Can be run as a preprocessor or after each assignment

Hill-climbing, simulated annealing typically work

with "complete" states, i.e., all variables assigned

\_ To apply to CSPs:\_ allow states with unsatisfied constraints

\_ operators reassign variable values

\_ Variable selection: randomly select any conflicted variable

\_ Value selection by min-conflicts heuristic:\_ choose value that violates the fewest constraints

\_ i.e., hill-climb with *h*(*n*) = total number of violated constraints

**Adversarial Search**

Optimal play for MAX assumes that MIN also plays

optimally, what if MIN does not play optimally?

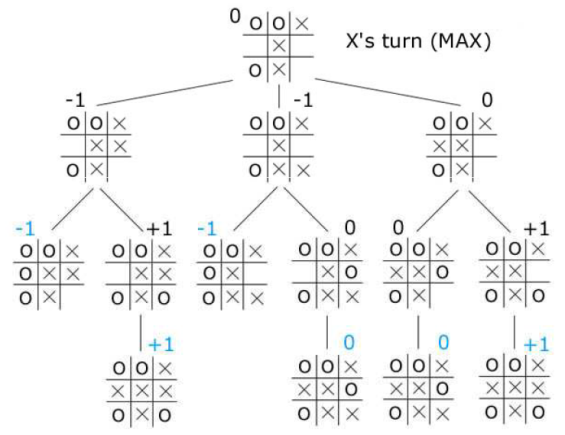
\_ A complete depth-first search? Yes

\_ Time complexity? O(bm)

\_ Space complexity? O(bm) (depth-first exploration)

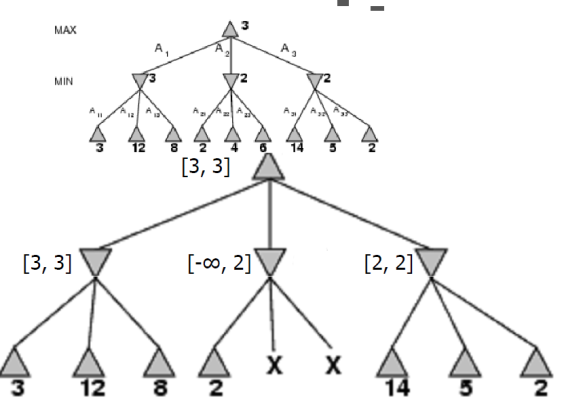
\_ For chess, b ≈ 35, m ≈ 100 for "reasonable" games

\_ exact solution completely infeasible



**Alpha Beta Pruning**

The number of game states with minimax search is exponential in the # of moves thus need to prune away branches that cannot possibly influence the final decision



Consider a node n such that Player has a choice of moving to

\_ If Player has a better choice m either at the parent of n or at any choice point further up, then n will never be reached in actual play

\_ α-β pruning gets it name from the two parameters that describe bounds on the backed-up values

α = the value of the best (highest-value) choice we have found so far at any choice point along the path for MAX

\_ β = the value of the best (lowest-value) choice we have found so far at any choice point along the path for MIN

\_ α-β search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is worse than the current α or β for MAX or MIN respectively

\_ Pruning does not affect final result

\_ The effectiveness of alpha-beta pruning is highly dependent on the order of successors

\_ It might be worthwhile to try to examine first the successors that are likely to be best

\_ With "**perfect ordering**," time complexity = O(bm/2)

-> effective branching factor becomes square root of b

\_ For chess, 6 instead of 35

\_ it can look ahead roughly twice as far as minimax in the same amount of time

\_ Ordering in chess: captures, threats, forward moves, and then backward moves

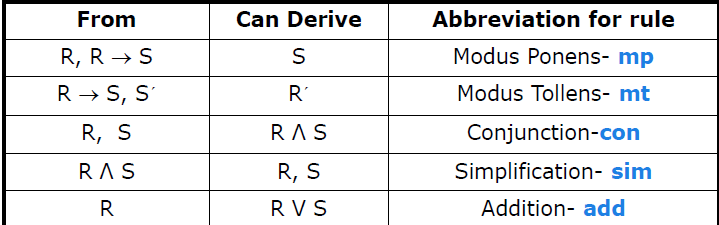
**If successors are examined in random order rather**

**than best-first**, the complexity will be roughly O(b3m/4)

Adding dynamic move-ordering schemes, such as trying first the moves that were found to be best last time, brings us close to the theoretical limit

The best moves are often called killer moves (killer move heuristic)

**Logical Agents**

****

A sentence is valid if it is true in all models, also known as tautology e.g., True, A ∨ ¬A, A ⇒ A, (A ∧ (A ⇒ B)) ⇒ B

Validity is connected to inference via the Deduction Theorem

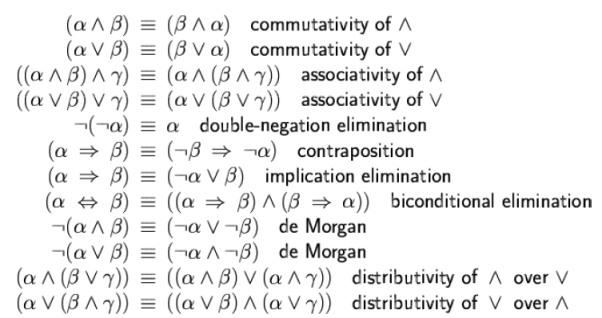
KB ╞ α if and only if (KB ⇒ α) is valid

A sentence is satisfiable if it is true in some model e.g., A ∨ B, C

A sentence is unsatisfiable if it is true in no models e.g., A ∧ ¬A

Satisfiability is connected to inference via the following:

KB ╞ α if and only if (KB ∧ ¬α) is unsatisfiable thus proof by contradiction

****

****

**Conjunctive Normal Form**(CNF): conjunction of disjunctions of literals clauses E.g., (A ∨ ¬B) ∧ (B ∨ ¬C ∨ ¬D)

**Horn Form** KB = conjunction of Horn clauses

Horn clause = disjunction of literals of which at most one is positive E.g., (¬L1,1 V ¬Breeze V B1,1)

Every horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal E.g., (L1,1 ^ Breeze) ⇒ B1,1

**Definite clause**: horn clauses with exactly one positive literal. The positive literal is called the head and the negative literals form the body. Definite clauses form the basis for logic programming. Can be used for FC and BC. Linear time

Resolution is complete for propositional logic **Forward, backward chaining** are linear-time, complete for Horn clauses

Propositional logic lacks expressive power.

semantics: truth of sentences wrt models

entailment: necessary truth of one sentence given another

inference: deriving sentences from other sentences

soundness: derivations produce only entailed sentences

completeness: derivations can produce all entailed sentences

**Resolution Algorithm**

To show that KB╞α,we show that (KB^¬α)is unsatisfiable

!First,(KB^¬α) is converted into CNF. Then the solution rule is applied to the resulting clauses. Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not present. Process until: 1.no new clauses can be added thus kb does not entail a. two clauses resolve to yield {} thus kb entails a. {} is a disjunction of no disjoints is equivalent to false thus the contradiction

Wumpus world properties:

fully observable? no, only local perception

deterministic? yes, outcome exactly specified

episodic? n, sequential at the level of actions

static? yes, wumpus and pits do not move

discrete?yes, single agent? Yes

An inference algorithm that derives only entailed sentences is called sound

An inference algorithm is complete if it can derive any sentence that is entailed

If KB is true in the real world, then any sentence derived from KB by a sound inference procedure is also true in the real world