TDDC17

Seminar 4
Adversarial Search
Constraint Satisfaction Problems

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Adverserial Search

Chapter 5

minmax algorithm
alpha-beta pruning

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Why Board Games?

Board games are one of the oldest branches of AI (Shannon and Turing 1950).

- Board games present a very abstract and pure form of competition between two opponents and clearly require a form of "intelligence"
- The states of a game are easy to represent
- The possible actions of the players are well-defined

→ Realization of the game as a search problem

→ It is nonetheless a contingency problem, because the characteristics of the opponent are not known in advance

Problems

Board games are not only difficult because they are contingency problems, but also because the search trees can become astronomically large.

Examples:

- **Chess**: On average 35 possible actions from every position, 100 possible moves (50 each player) → $35^{100} \approx 10^{150}$ nodes in the search tree (with "only" 10^{40} distinct chess positions (nodes)).
- **Go**: On average 200 possible actions with ca. 300 moves → $200^{300} \approx 10^{700}$ nodes.

Good game programs have the properties that they

- delete irrelevant branches of the game tree,
- use good evaluation functions for in-between states, and
- look ahead as many moves as possible.

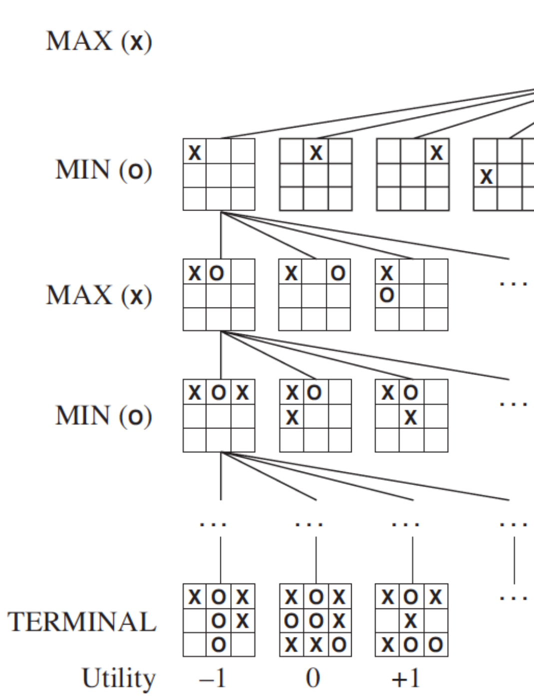
Adverserial Search

- Multi-Agent Environments
  - agents must consider the actions of other agents and how these agents affect or constrain their own actions.
  - environments can be cooperative or competitive.
  - One can view this interaction as a "game" and if the agents are competitive, their search strategies may be viewed as "adversarial".
- Two-agent, zero-sum games of perfect information
  - Each player has a complete and perfect model of the environment and of its own and other agents actions and effects
  - Each player moves until one wins and the other loses, or there is a draw.
  - The utility values at the end of the game are always equal and opposite, thus the name zero-sum.
  - Chess, checkers, Go, Backgammon (uncertainty)

Games as Search

- The Game
  - Two players: One called MIN, the other MAX. MAX moves first.
  - Each player takes an alternate turn until the game is over.
  - At the end of the game points are awarded to the winner, penalties to the loser.
- Adversarial Search:
  - Initial State – Board position and player to move
  - Successor Function – returns a list of (move, state) pairs indicating a legal move and resulting state.
    - Search space is a *game tree*.
    - A *ply* is a half move.
  - Terminal Test – When the game is over.
  - Utility Function – Gives a numeric value for terminal states. For example, in Chess: win (1), lose (-1), draw (0):

Simple Game Tree for Tic-Tac-Toe



- Game trees can be infinite
- Often large! Chess has:
  - 10^{40} distinct states
  - average of 50 moves
  - average b-factor of 35
  - $35^{100} = 10^{154}$ nodes

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Minimax

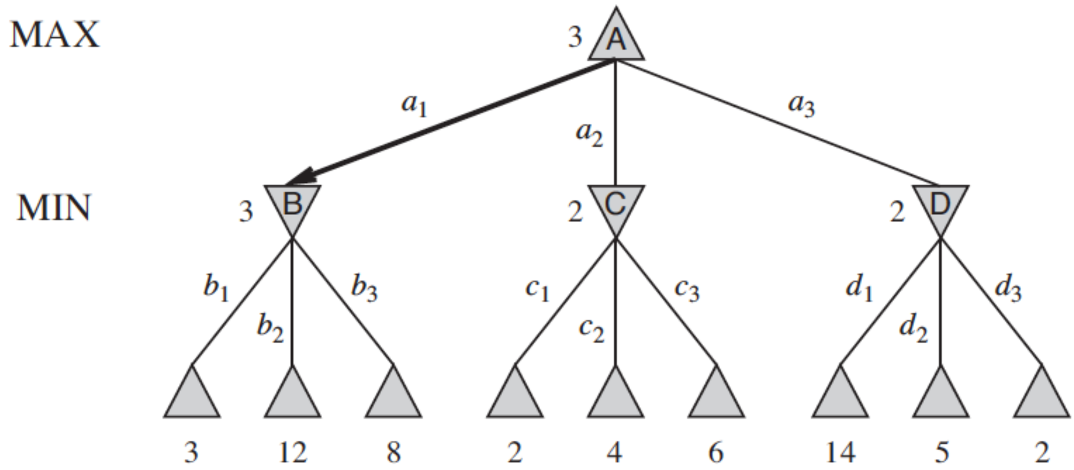
1. Generate the complete game tree using depth-first search.
2. Apply the utility function to each terminal state.
3. Beginning with the terminal states, determine the utility of the predecessor nodes as follows:

- Node is a MIN-node
  Value is the minimum of the successor nodes
- Node is a MAX-node
  Value is the maximum of the successor nodes
- From the initial state (root of the game tree), MAX chooses the move that leads to the highest value (minimax decision).

Note: Minimax assumes that MIN plays perfectly. Every weakness (i.e. every mistake MIN makes) can only improve the result for MAX.

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

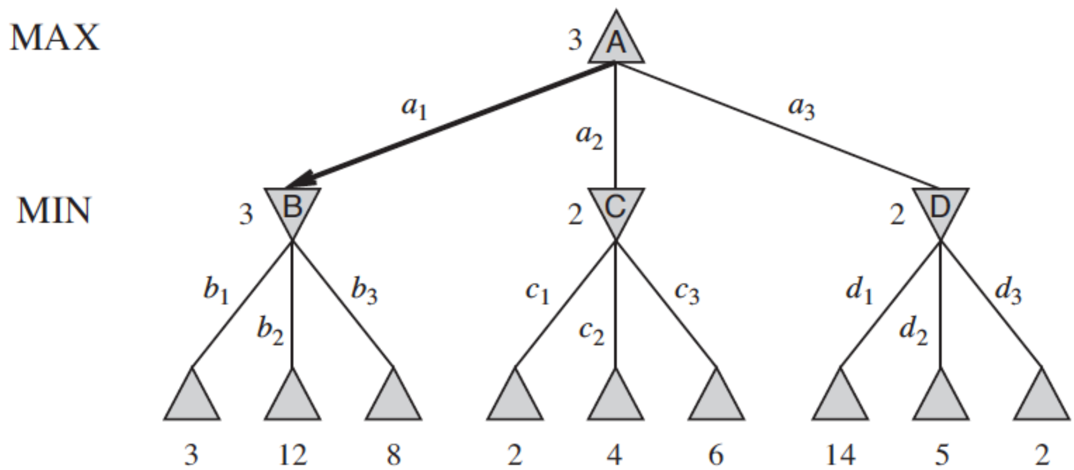
A MINMAX Tree



- Interpreted from MAX's perspective
- Assumption is that MIN plays optimally
- The minimax value of a node is the utility for MAX
- MAX prefers to move to a state of maximum value and MIN prefers minimum value

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

MINMAX Algorithm



What move should MAX make from the initial state?

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

MINIMAX Algorithm

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Note: Minimax only works when the game tree is not too deep. Otherwise, the minimax value must be approximated.

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Alpha-Beta Pruning

- Minimax search examines a number of game states that is <u>exponential</u> in the number of moves.
- Can be improved by using *Alpha-Beta Pruning*.
  - The same move is returned as minmax would
  - Can effectively cut the number of nodes visited in half (still exponential, but a great improvement).
  - Prunes branches that can not possibly influence the final decision.
  - Can be applied to infinite game trees using cutoffs.

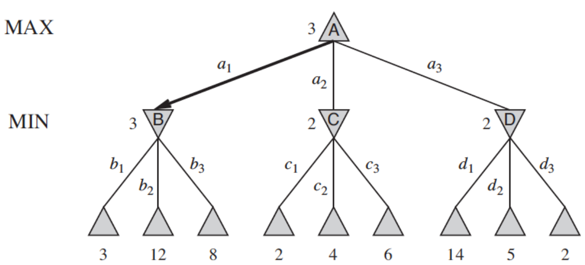
Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

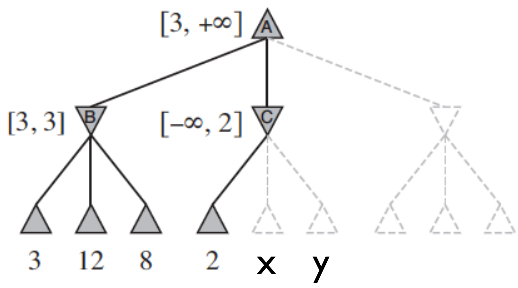
Alpha-Beta Values

alpha – the value of the best (i.e., highest value) choice we have found so far at any choice point along the path for MAX. (actual value is <u>at least</u>)....lower bound

beta - the value of the best (i.e., lowest value) choice we have found so far at any choice point along the path for MIN. (actual value is <u>at most</u>)...upper bound

Intuitions





x and y: *the two unevaluated successors of node C*

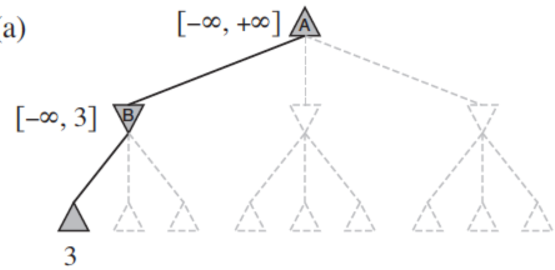
MINMAX(root) $= \max(\min(3,12,8), \min(2, x, y), \min(14, 5, 2))$
$= \max(3, \min(2, x, y), 2)$
$= \max(3, z, 2)$ where $z = \min(2, x, y) < 2$
$= 3$

- *Since C can maximally be 2, x and y become irrelevant because we have already something better (3 @ B)!*

Often possible to prune entire subtrees rather than just leaf nodes!

Alpha-Beta Progress

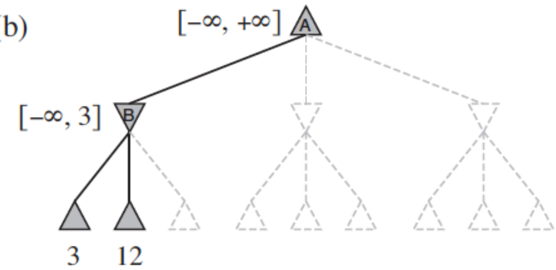
(a)



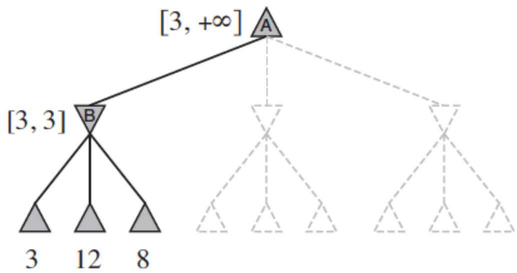
[alpha, beta]

[at least, at most]

(b)

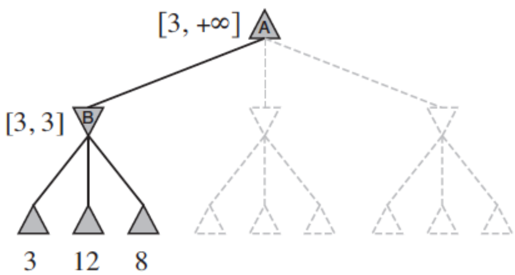


(c)



Alpha-Beta Progress

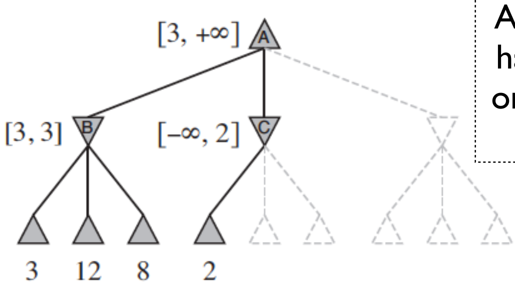
(c)



[alpha, beta]

[at least, at most]

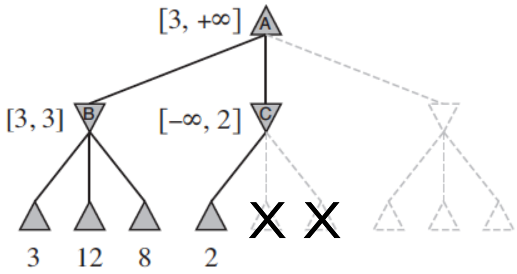
(d)



A has a better choice at B then it would ever have at C because further exploration would only make beta= 2 at C smaller, so prune the remaining branches

Alpha-Beta Progress

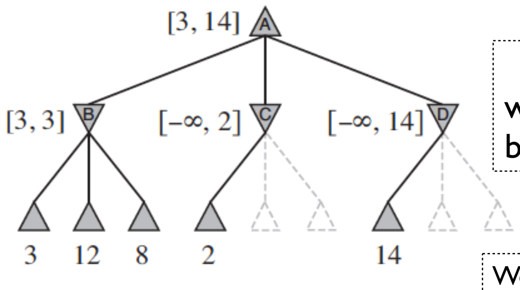
(d)



[alpha, beta]

[at least, at most]

(e)

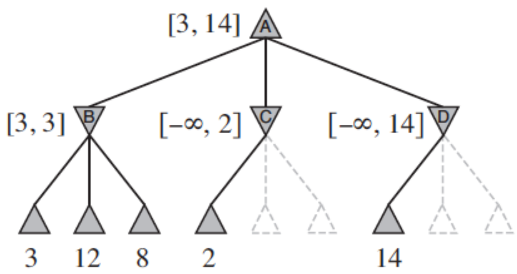


The 1st leaf below D is 14, so D is *at most* worth 14 which is higher than MAX's current best alternative (at least 3), so keep exploring

We now have bounds on all successors of the root so A's beta value is at most 14.

Alpha-Beta Progress

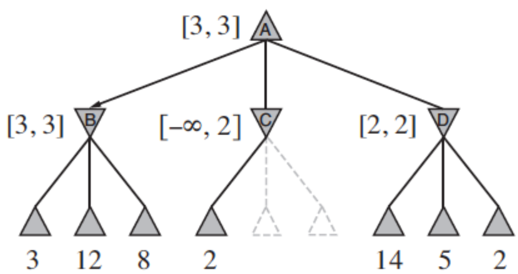
(e)



[alpha, beta]

[at least, at most]

(f)



Similar for the 2nd leaf with value 5 so keep exploring. Final leaf gives exact value of 2 for D

MAX's decision at root is to move to B with value 3.

Alpha-Beta Search Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in ACTIONS(*state*) with value v

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Similar to MINMAX algorithm but here we keep track of and propagate alpha and beta values

Chess (1)

In 1997, world chess master G. Kasparov was beaten by a computer in a match of 6 games.

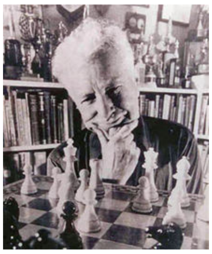
Deep Blue (IBM Thomas J. Watson Research Center)

- Special hardware (30 processors with 48 special purpose VLSI chess chips, could evaluate 200 million chess positions per second)
- Heuristic search
- Case-based reasoning and learning techniques
  - 1996 Knowledge based on 600 000 chess games
  - 1997 Knowledge based on 2 million chess games
  - Training through grand masters

Chess (2)

Nowadays, ordinary PC hardware is enough …

Name	Strength (ELO)
Rybka 2.3.1 (50$ @ Amazon)	2962
G. Kasperov	2828
V. Anand	2758
A. Karpov	2710
Deep Blue	2680



Arpad Elo - Creator of ELO system

But note that the machine ELO points are not strictly comparable to human ELO points …

ELO rating system is a method for calculating relative skill levels of players in competitor versus competitor games such as chess

The Reasons for Success…

- Alpha-Beta-Search

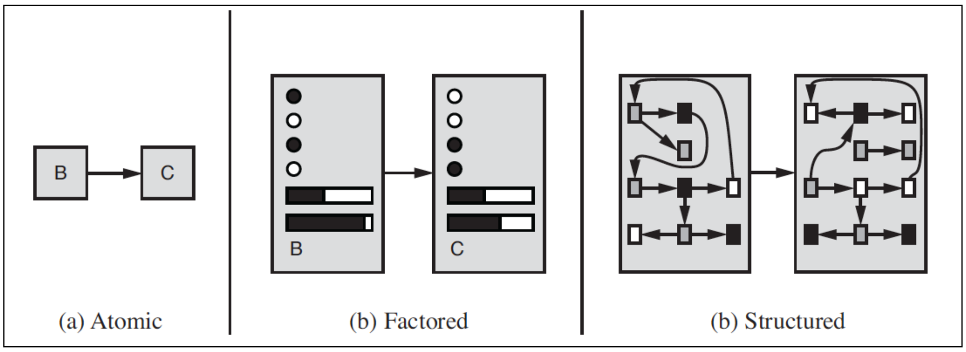
  … with dynamic decision-making for uncertain positions

- Good (but usually simple) evaluation functions
- Large databases of opening moves
- And very fast and parallel processors!
- (For Go, Monte-Carlo techniques proved to be successful rather than Alpha-Beta Search!)

Constraint Satisfaction Problems
Chapter 6.

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Representing States



No internal structure

Vector of attribute values
attribute-value pairs

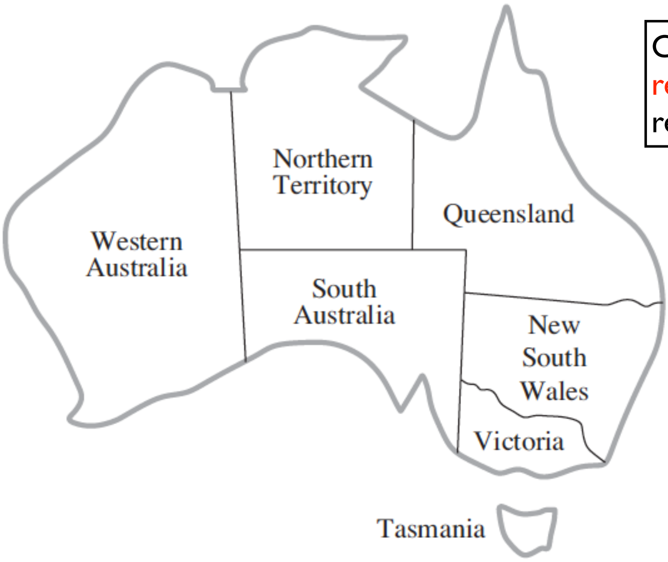
Objects (possibly with attributes)
Relations between and properties of objects

So far:
uninformed search
heuristic search

Today:
Constraint Satisfaction

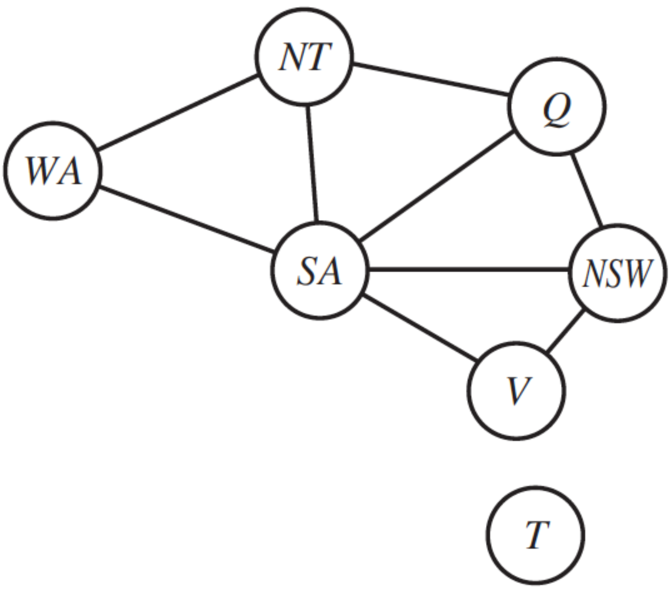
Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Map Coloring: Australian States and Territories



Color each of the territories/states red, green or blue with no neighboring region having the same color

Let's Abstract!



<u>Constraint Graph</u>

Nodes are *variables*

Arcs are *constraints*

Our Representation

- •Associate a variable with each region.
- •Introduce a set of values the variables can be bound to.
- •Define constraints on the variable/value pairs

Goal:

<u>Find</u> a set of legal bindings satisfying the constraints!

Constraint Satisfaction Problem

<u>Problem Specification</u>

3 Components:

- X is a set of *variables* $\{X_1, ..., X_n\}$
- D is a set of *domains* $\{D_1, ..., D_n\}$, one for each variable
- C is a set of *constraints* on X restricting the values variables can simultaneously take

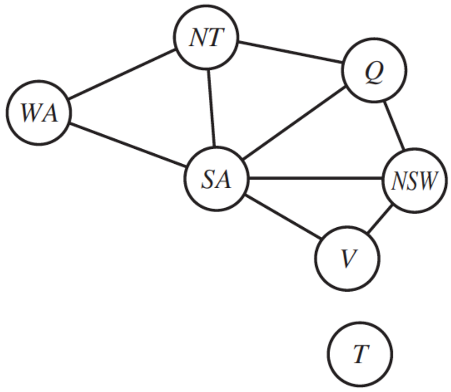
<u>Solution to a CSP</u>

An assignment of a value from its domain to each variable, in such a way that all the constraints are satisfied

One may want to find 1 solution, all solutions, an optimal solution, or a good solution based on an objective function defined in terms of some or all variables.

Map Coloring: Australia

Map Coloring Specification
- X = {WA, NT, SA, Q, NSW, V, T}
- D = {{red, green, blue}, ..., {red, green, blue}}
- C is a set of *constraints* on X



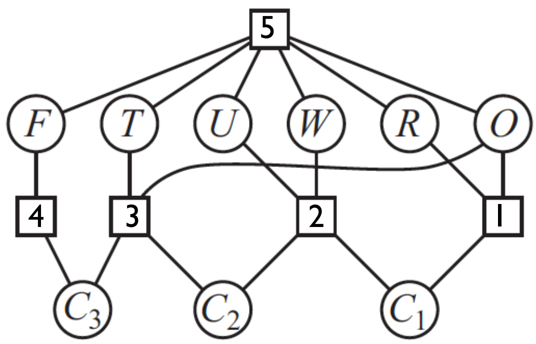
Binary Constraints

Constraints
WA ≠ NT,
WA ≠ SA,
NT ≠ Q,
NT ≠ SA,
Q ≠ SA,
Q ≠ NSW,
V ≠ SA,
V ≠ NSW,

Crypto-Arithmetic Problems

Specification
- X = {F, T, U, W, R, O, C1, C2, C3}
- D = {{0,...,9}, ..., {0,...,9}, {0,1}, {0,1}, {0,1}}
- C is a set of *constraints* on X

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



hypergraph

n-ary constraints

1. $O + O = R + 10 \times C_1$
2. $C_1 + W + W = U + 10 \times C_2$
3. $C_2 + T + T = O + 10 \times C_3$
4. $C_3 = F$
5. Alldiff(F,T,U,W,R,O)

Variable, Domain and Constraint Types

Types of variables/domains

- Discrete variables
  - Finite or infinite domains
- Boolean variables
  - Finite domain
- (Continuous variables)
  - Infinite domain

Types of constraints

- Unary constraints (1)
- Binary constraints (2)
- Higher-Order contraints (>2)
- Linear constraints
- Nonlinear constraints

Some Special cases

- Linear programming
  - Linear inequalities forming a convex region. Continuous domains.
  - Solutions in time polynomial to the number of variables
- Integer programming
  - Linear constraints on integer variables.

Any higher-order/finite domain csp's can be translated into binary/finite domain CSPs! (In the book, R/N stick to these)

Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Variables

81(one for each cell)

Constraints:

Alldiff() for each row

Alldiff() for each column

Alldiff for each 9 cell area

Advantages of CSPs

- Representation is closer to the original problem.
- Representation is the same for all constraint problems.
- Algorithms used are domain independent with the same general purpose heuristics for all problems
- Algorithms are simple and often find solutions quite rapidly for large problems
  - CSPs often more efficient than regular state-space search because it can quickly eliminate large parts of the search space
  - Many problems intractable for regular state-space search can be solved efficiently with a CSP formulation.

Solving a CSP: Types of Algorithms

Search
(choose a new variable assignment)

Inference
Constraint Propagation
(reduce the # of legal values for a variable and propagate to other variables)

Constraint Propagation

Sometimes solves the problem without search!

Constraint Propagation

Pre-Processing

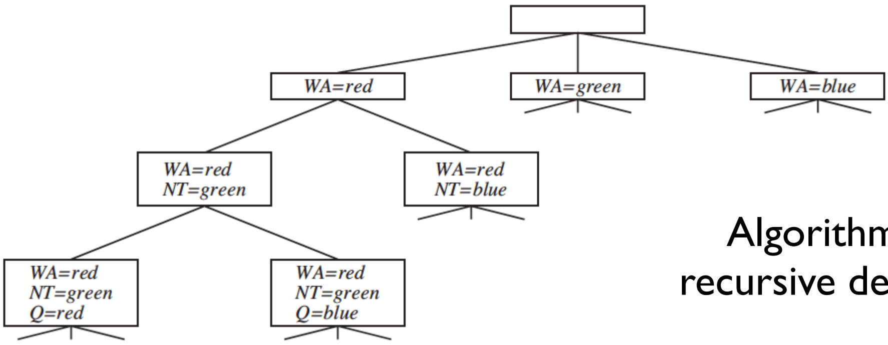
Search

Constraint Propagation

Interleave

Search

Simple Backtracking Search Algorithm for CSPs



Algorithm is based on recursive depth-first search

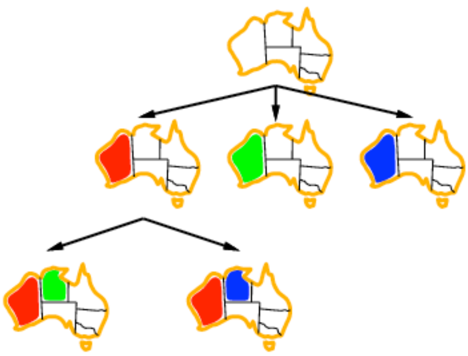
If a value assignment to a variable leads to failure then it is removed from the current assignment and a new value is tried (backtrack)

The algorithm will interleave inference with search

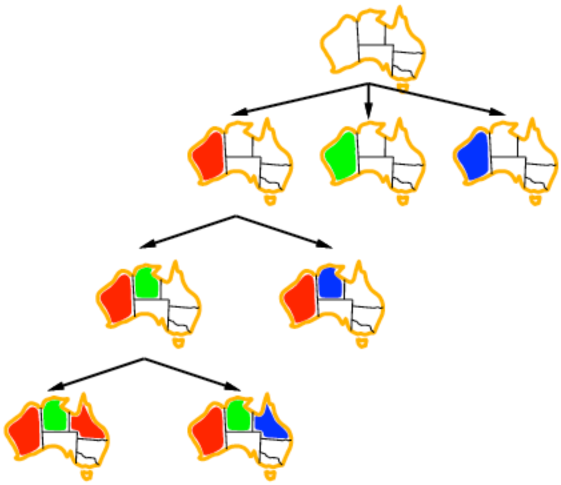
Example (1)

Example (2)

Example (3)



Example (4)



Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Backtracking Algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
   return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment  then
         add {var = value} to assignment
         inferences ← INFERENCE(csp, var, value)
         if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
               return result
      remove {var = value} and inferences from assignment
   return failure
```

Domain Independent Heuristics

Inference

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Potential Problems with Backtracking

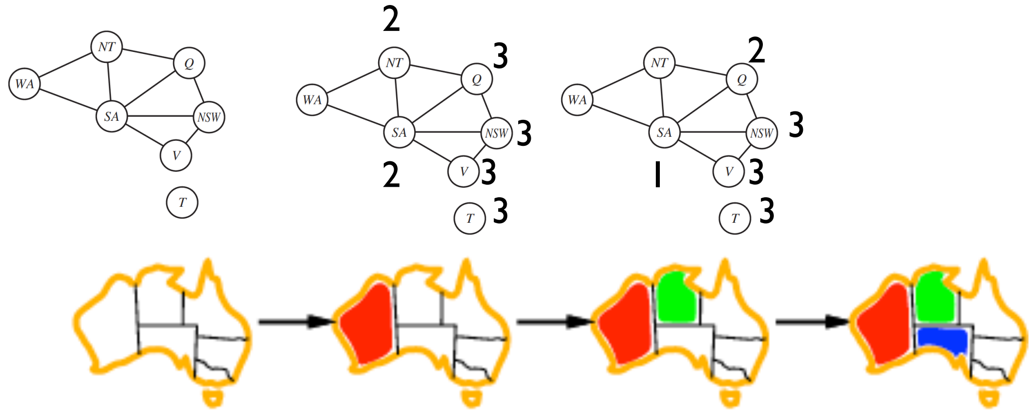
- Variable choice and value assignment is arbitrary
  - *Which variable should be assigned?*
    - SELECT-UNASSIGNED-VARIABLE()
  - *Which values should be assigned first?*
    - ORDER-DOMAIN-VALUES()
- Conflicts detected too late (empty value domain)
  - Conflicts not detected until they actually occur.
  - *What are the implications of current variable assignments for the other unassigned variables?*
    - *INFERENCE()*
- Thrashing
  - Real reason for failure is conflicting variables, but these conflicts are continually repeated throughout the search
  - *When a path fails, can the search avoide repeating the failure in subsequent paths?*
    - One solution: Intelligent Backtracking

Variable Selection Strategies

- Variable Selection Strategy
  - SELECT-UNASSIGNED-VARIABLE()
  - Minimum Remaining Values (MRV) heuristic
    - Choose the variable with the fewest remaining legal values.
    - Try first where you are most likely to fail (fail early!..hard cases 1st)
      - Will knock out large parts of the search tree.
  - Degree Heuristic
    - Select the variable that is involved in the largest number of constraints on other unassigned variables.
    - Hard cases first!
    - Tie breaker when MRV can't be applied.

Minimum Remaining Values (MRV) heuristic

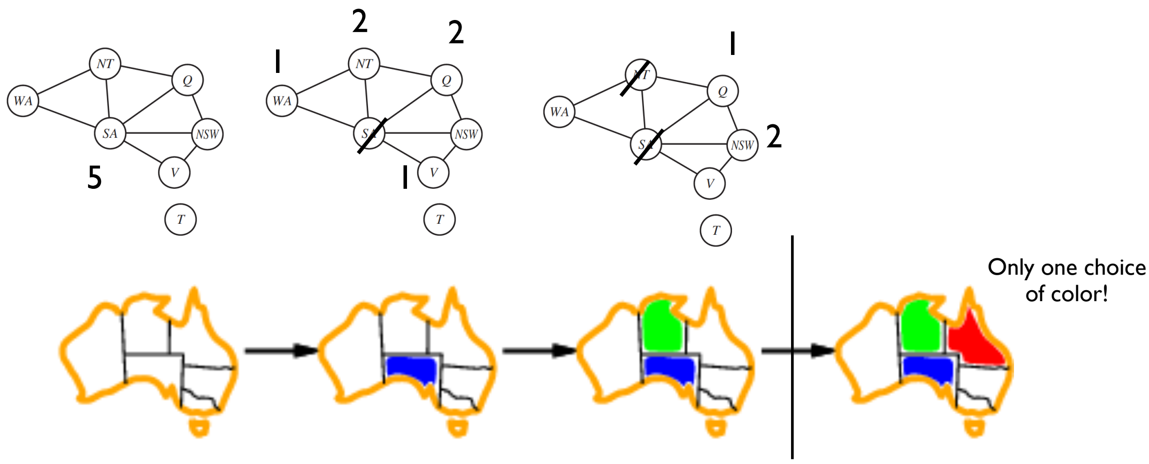
"Attempts to fail early, thus removing parts of the search tree"



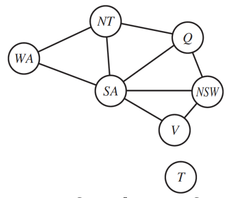
Actually, if we used
degree heuristic
to break ties, then SA
would be chosen here
instead of NT

Degree Heuristic

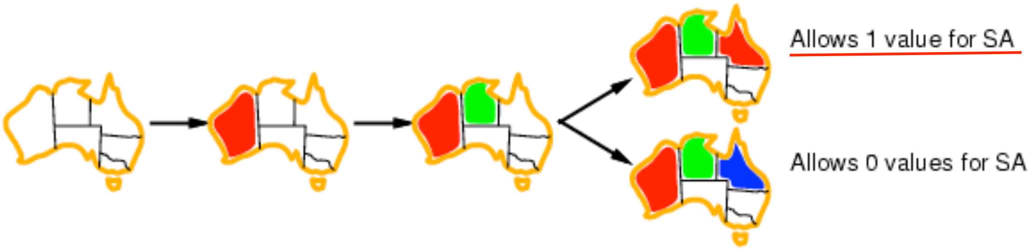
"Attempts to reduce the branching factor in search tree"



Value Selection Strategies



- Value Selection Strategy
  - ORDER-DOMAIN-VALUES()
  - Least-constraining-value heuristic
    - Choose the value that rules out the fewest choices of values for the neighboring variables in the constraint graph.
    - Maximize the number of options….least commitment.
      - Only useful when searching for one solution.



Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Inference in CSP's

Key Idea:
- Treat each variable as a node and each binary constraint as an arc in our constraint graph.
- Enforcing local consistency in each part of the graph eliminates inconsistent values throughout the graph.
- The less local we get when propagating the more expensive inference becomes.

Node Consistency
A single variable is node consistent if all values in the variable's domain satisfy the variables unary constraints

WA $\neq$ green D_{WA}={red, gr~~e~~en,blue}

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Arc Consistency

Definition

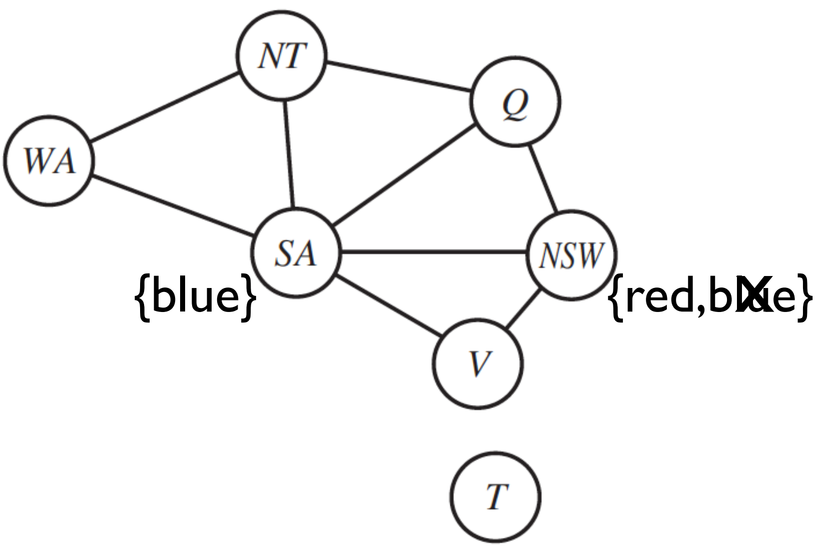
Arc (V_i,V_j) is arc consistent if for every value x in the domain of V_i there is some value y in the domain of V_j such that V_i = x and V_j = y satisfies the constraints between V_i and V_j.

A constraint graph is arc-consistent if all its arcs are arc consistent

- The property is not symmetric.
- Arc consistent constraint graphs do not guarantee consistency of the constraint graph and thus guarantee solutions. They do help in reducing search space and in early identification of inconsistency.
- **AC-3** ($O(n^{2}*d^{3})$), **AC-4** ($O(n^{2}*d^{2})$) are polynomial algorithms for arc consistency, but 3SAT (NP) is a special case of CSPs, so it is clear that **AC-3**, **AC-4** do not guarantee (full) consistency of the constraint graph.

n = # variables, d = domain size

Arc Consistency is not symmetric



SA ⟶ NSW

Is arc-consistent

NSW ⟶ SA

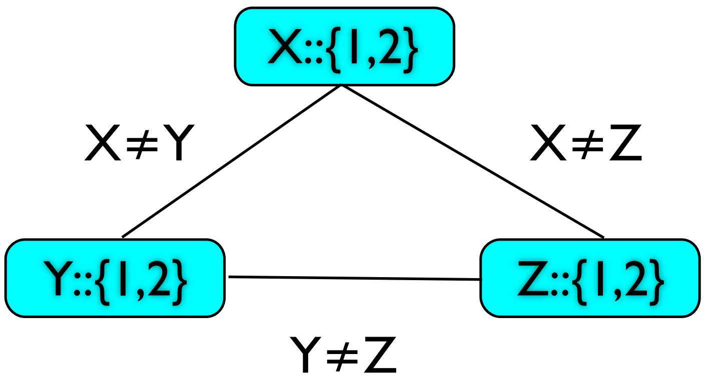
Is not arc-consistent

Remove blue from NSW

NSW ⟶ SA

Is now arc-consistent

Arc Consistency does not guarantee a solution

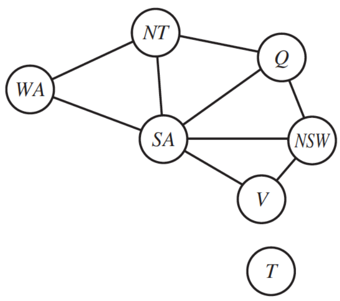


Arc consistent constraint graph with no solutions

Simple Inference: Forward Checking

Whenever a variable *X* is assigned, look at each unassigned variable *Y* that is connected to *X* by a constraint and delete from *Y*'s domain any value that is inconsistent with the value chosen for *X*. [make Y's arc consistent with X]

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

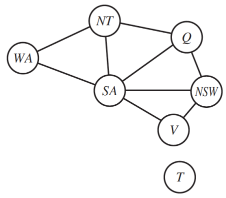


Note1: After WA=red, Q=green, NT and SA both have single values. This eliminates branching.

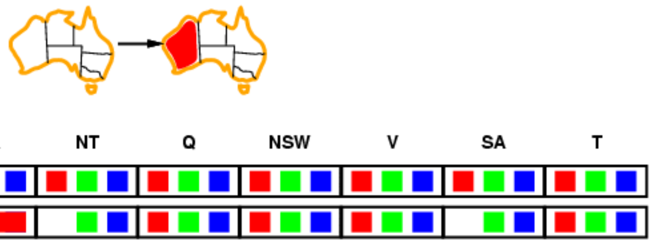
Note 2: WA=red, Q=green, there is an inconsistency between NT, SA, but it is not noticed.

Note 3: After V=blue, an inconsistency is detected

Forward Checking (2)



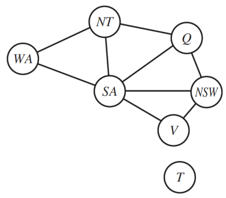
- Keep track of remaining values
- Stop if all have been removed



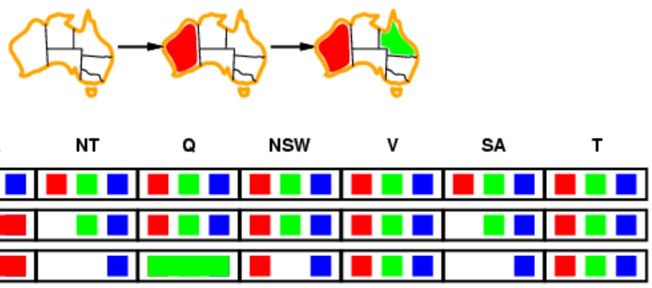
After inference, when searching: Branching decreased on NT and SA

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Forward Checking (3)



- Keep track of remaining values
- Stop if all have been removed

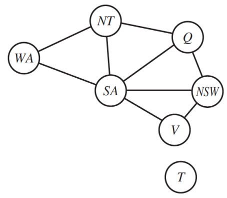


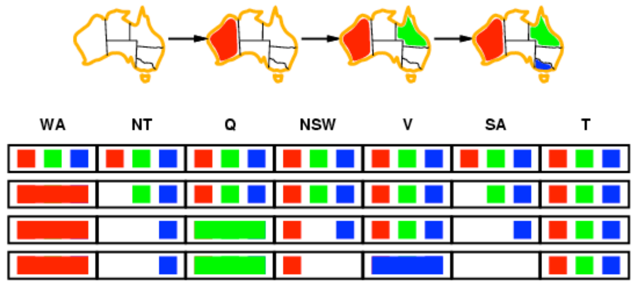
After inference, when searching: Branching eliminated on NT and SA

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Forward Checking (4)

- Keep track of remaining values
- Stop if all have been removed

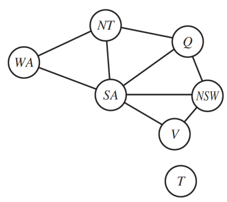


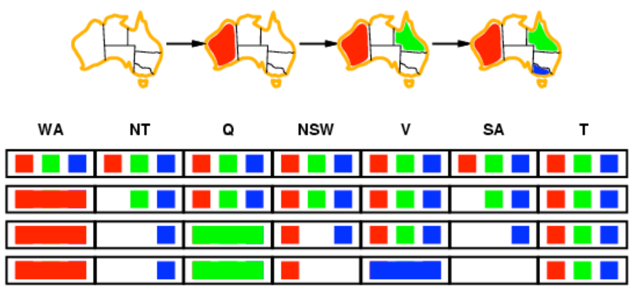


The partial assignment WA=R, Q=G, V=B is inconsistent (SA is empty) so the search algorithm will backtrack immediately.

Forward Checking: Sometimes it Misses Something

- Forward Checking makes the current variable arc-consistent but does not do any look ahead to make all other variables arc-consistent.





In row 3, when WA is red and Q is green, both NT and SA are forced to be blue. But they are adjacent so this can not happen, but it is not picked out by forward checking, the inference is too weak.

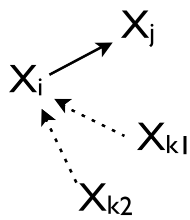
AC-3 Algorithm

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components (X, D, C)
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of $D_i = 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add (X_k, X_i) to *queue*
  return *true*

---

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*



Returns an arc consistent binary constraint graph or false because a variable domain is empty (and thus no solution)

INFERENCE() = AC-3

Example using nodes WA,NT,SA

{r,g,b} WA --> NT {r,g,b}
{r,g,b} NT --> WA {r,g,b}
{r,g,b} WA --> SA {r,g,b}
{r,g,b} SA --> WA {r,g,b}
{r,g,b} NT --> SA {r,g,b}
{r,g,b} SA --> NT {r,g,b}

During search:
Assign WA=red

{r,g,b} WA --> NT {r,g,b}
{r,g,b} NT --> WA {r,g,b}
{r,g,b} WA --> SA {r,g,b}
{r,g,b} SA --> WA {r,g,b}
{r,g,b} NT --> SA {r,g,b}
{r,g,b} SA --> NT {r,g,b}

Apply AC-3 to queue of arcs

ok: {r,g,b} WA --> NT {r,g,b}
no: {r,g,b} NT --> WA {r,g,b}
{r,g,b} WA --> SA {r,g,b}
{r,g,b} SA --> WA {r,g,b}
{r,g,b} NT --> SA {r,g,b}
{r,g,b} SA --> NT {r,g,b}

(Remove r from NT and) place NT's dependents on the queue

ok: ~~{r,g,b} WA --> NT {r,g,b}~~
ok: ~~{r,g,b} NT --> WA {r,g,b}~~
{r,g,b} WA --> SA {r,g,b}
{r,g,b} SA --> WA {r,g,b}
{r,g,b} NT --> SA {r,g,b}
~~{r,g,b} SA --> NT {r,g,b}~~

Next Problem

{r,g,b} SA --> NT {r,g,b}
{r,g,b} WA --> NT {r,g,b}

Continue until fail or no more changes are required

Path Consistency

{r,g,b} WA --> NT {r,g,b}
{r,g,b} NT --> WA {r,g,b}
{r,g,b} WA --> SA {r,g,b}
{r,g,b} SA --> WA {r,g,b}
{r,g,b} NT --> SA {r,g,b}
{r,g,b} SA --> NT {r,g,b}

Note that arc consistency does not help us out for the map coloring problem!

It only looks at pairs of variables

A two variable set $\{X_i, X_j\}$ is path consistent with respect to a 3rd variable X_m if, for every assignment $\{X_i=a, X_j=b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i,X_m\}$ and $\{X_m, X_j\}$.

$$X_i \longrightarrow X_m \longrightarrow X_j$$

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

K-Consistency

A CSP is *k*-consistent if, for any set of *k*-1 variables and for any consistent assignment to those variables, a consistent value can always be found for the *k*th variable.

1-consistency: node consistency
2-consistency: arc consistency
3-consistency: path consistency

A CSP is strongly *k*-consistent if it is *k*-consistent and is also *k*-1 consistent, *k*-2 consistent, ..., 1-consistent.

In this case, we can find a solution in $O(n^2d)$! but establishing n-consistency takes time exponential in n in the worst case and space exponential in n!

Artificial Intelligence & Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Real-Life example

- Suppose our territories are coverage areas, each with a sensor that monitors the area.
- Each sensor has three possible radio frequencies
- Sensors overlap if they are in adjacent areas
- If sensors overlap, they can not use the same frequency

Find a solution where each sensor uses a frequency that does not interfere with adjacent coverage areas

This is an N-map coloring problem!