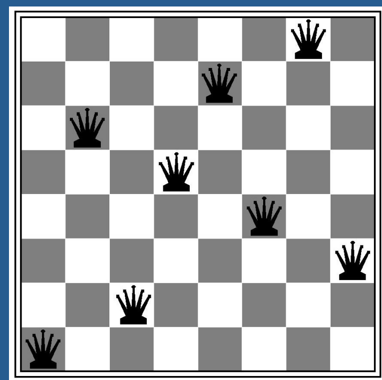


Constraint Satisfaction

EECS 492
January 25th, 2011

The story up to now…

- Uninformed Search
  - BFS, DFS, IDS
- Informed Search
  - A*, SMA*
- Local Search
  - Hill Climbing, Genetic Algorithms

Today

- Constraint Satisfaction Problems
  - Examples
  - Definitions
- Making smart choices going forward
  - Minimum Remaining Values heuristic
  - Forward checking
  - Constraint Propagation
- Making smart choices going backward (when we get stuck)
  - Backjumping
- Local Search Strategies

States as Black Boxes

- Search methods so far impose minimal requirements on states:
  - Generate successors
  - Evaluate domain-specific heuristics
  - Apply goal test
- From point of view of search algorithm, states are **black boxes** — no relevant internal structure

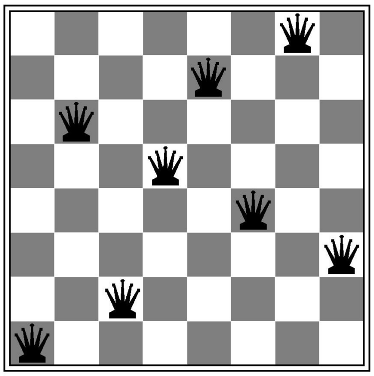
Exploiting Structure in States

- Constraint Satisfaction Problems (CSPs)
  - Standard, structured, and simple representation
  - Enabling use of general-purpose algorithms
  - Achieving performance improvements without domain-specific heuristics

Variables, Domains, Constraints

- Variables
  - {entrée, dessert}
- Domains
  - The set of values a variable can take
  - entrée \in { steak, fish, lasagna }
  - dessert \in { pie, jello, ice cream }
- Constraints
  - A relationship between two or more variables
  - calories(entrée) + calories(dessert) < 1000
  - calcium(entrée) + calcium(dessert) > 100

Example: 8 Queens

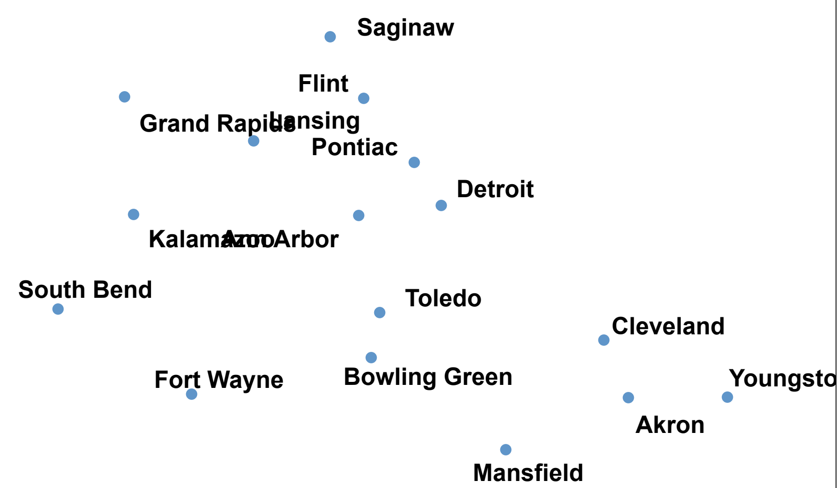


Find an arrangements of queens such that no queen attacks another

8 Queens as CSP

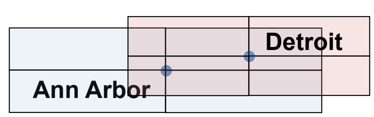
- Variables, $X_1,\ldots,X_n$
  - One for each queen (n=8)
  - Assume one queen per column
- Variable domains
  - Row location of queen in column i, $X_i \in \{1,\ldots,8\}$
- Constraints, $C_1,\ldots,C_m$
  - No queens can attack each other
    - $X_i \neq X_j$, $i \neq j$
    - $X_i \neq X_j + k$, $|i - j| = k$

Example: Map Labeling



Map Labeling as CSP

- Variables
  - City label locations
- Domains
  - {NW, NE, SW, SE}
- Constraints
  - Labels of nearby cities do not overlap



AA	Detroit
NW	NE, SE
NE	*none*
SW	NW, NE, SE
SE	NW, NE

Legal assignments

Example: 3SAT

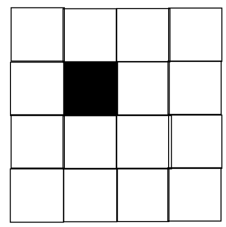
$(P_1 \vee \neg P_2 \vee \neg P_3) \wedge (P_1 \vee \neg P_2 \vee \neg P_4) \wedge (P_1 \vee \neg P_3 \vee \neg P_4) \wedge$
$(\neg P_1 \vee P_2 \vee \neg P_3) \wedge (P_2 \vee \neg P_3 \vee \neg P_4) \wedge (\neg P_1 \vee P_2 \vee \neg P_4) \wedge$
$(\neg P_1 \vee \neg P_2 \vee P_3) \wedge (\neg P_1 \vee P_3 \vee \neg P_4) \wedge (\neg P_2 \vee P_3 \vee \neg P_4) \wedge$
$(\neg P_1 \vee \neg P_2 \vee P_4) \wedge (\neg P_1 \vee \neg P_3 \vee P_4) \wedge (\neg P_2 \vee \neg P_3 \vee P_4) \wedge$
$(\neg P_1 \vee \neg P_2 \vee \neg P_3)$

Variables?
Domains?
Constraints?

Other examples?

- What other CSPs can you think of?

Example: Crossword Puzzle



1A: A number
2A: A number
…

1D: A metal
2D: A number

Variables?
Domains?
Constraints?

Example: Sudoku

								2
1			3	6		5	4	
			7		8	9		
	4			2		1		
	6		9		1		8	
		2		8			3	
		4	1		5			
	5	8		3	4			6
2								

Variables?
Domains?
Constraints?

Cryptarithmetic

- Variables
  - letters
- Domains
  - {0,…,9}
- Constraints
  - Columns have to add up right, including carries
  - Letters stand for distinct digits
  - S, M are non-zero

```
  S E N D
+ M O R E
---------
M O N E Y
```

CSPs as Search ("F" grade)

- For uninformed search formulation we need: {state_0, successors(n), is-goal(n), path-cost(n) }
  - state: Assignments for each variable.
  - state_0: No variables yet assigned
  - successors(n): All possible variable assignments for all unassigned variables
  - is-goal(n): All variables assigned satisfying all constraints?
  - path-cost(n): any constant value
- Which search algorithm?
- Run time?

CSPs as Search ("D")

- For uninformed search formulation we need:

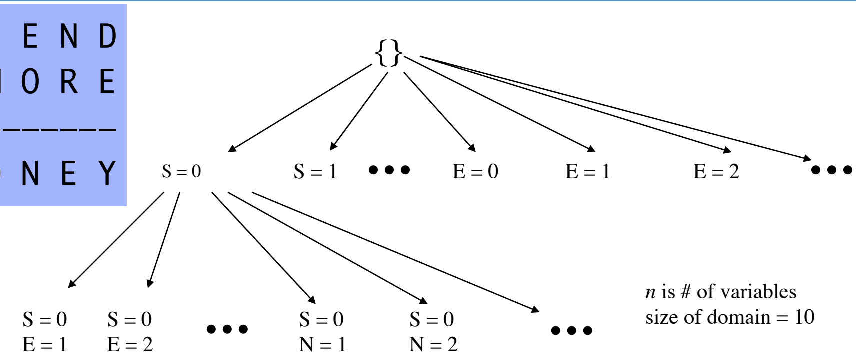
{$state_0$, successors(n), is-goal(n), path-cost(n) }

  - state: Assignments for each variable.
  - $state_0$: No variables yet assigned
  - successors(n): All *consistent* possible variable assignments for all unassigned variables
  - is-goal(n): All variables assigned?
  - path-cost(n): any constant value

- Run time?

Cryptarithmetic Search Tree ("D")

```
   S E N D
 + M O R E
 ---------
 M O N E Y
```



Branching factor

10 n at level 1

9(n – 1) at level 2

8(n – 2) at level 3,…

→ # of leaves = n!10!

But only 10^n complete assignments!

Another Key Observation

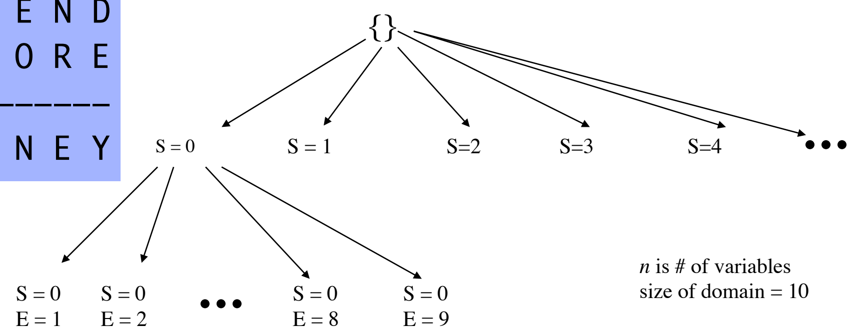
- The consistency of an assignment depends only on the values assigned to the variables
- The order in which the variables were assigned is irrelevant! (Commutivity)
  - Ignoring this leads to additional n! complexity
  - Solution: expand only *one* variable per node.

CSPs as Search ("C")

- For uninformed search formulation we need:
{$state_0$, successors(n), is-goal(n), path-cost(n) }
  - state: Assignments for each variable.
  - $state_0$: No variables yet assigned
  - successors(n): All *consistent* possible variable assignments for a *single unassigned* variables
  - is-goal(n): All variables assigned?
  - path-cost(n): any constant value
- Run time?

Cryptarithmetic Search Tree ("C")

```
  S E N D
+ M O R E
---------
M O N E Y
```



n is # of variables
size of domain = 10

Branching factor
- 10 at level 1
- 9 at level 2
- 8 at level 3,…

Picking which variable to expand

- Picking next variable arbitrarily is often inefficient
- MRV (minimum remaining values) heuristic
  - Choose variable with fewest legal values remaining
  - aka most constrained variable
  - If any variable has no legal values, MRV will choose that and detect failure immediately
- Degree heuristic
  - choose variable with largest number of constraints on unassigned variables

How many values remain?

- How do we compute which values remain for a variable?
  - Determining this exactly requires solving the problem!
  - How can we efficiently reduce the size of the domain?

How many values remain?

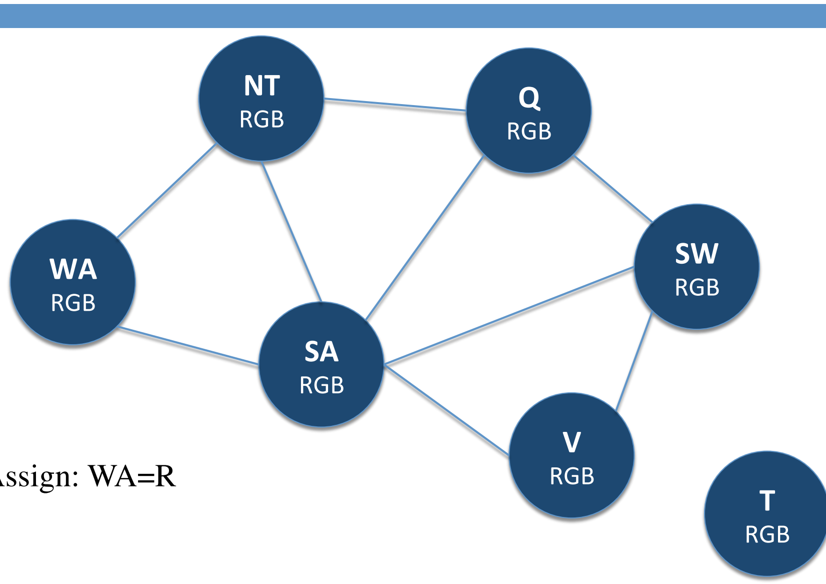
- Forward Checking
- Arc Consistency
- k-Consistency
- MAC Consistency

Don't worry, these are all basically the same idea, applied to varying extremes!

Forward Checking (FC)

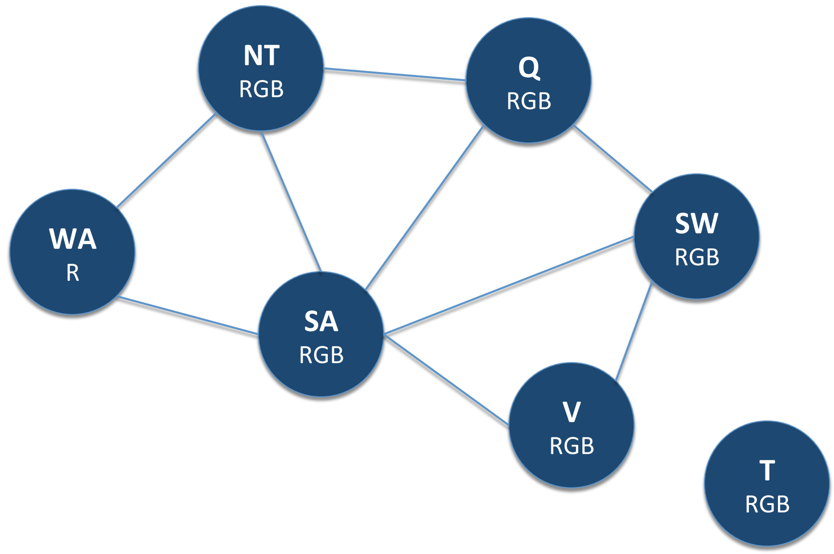
- Whenever a variable *X* is assigned
  - Examine each unassigned variable *Y* connected to *X* by a constraint
  - Delete from *Y*'s domain any value inconsistent with the value chosen for *X*
  - If assignment becomes impossible (anywhere), backtrack.

Forward Checking Example

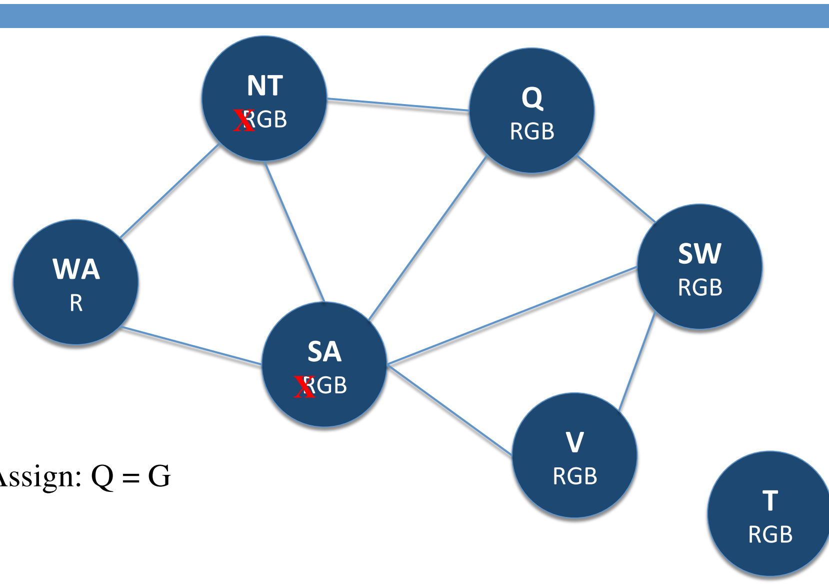


Assign: WA=R

Forward Checking Example

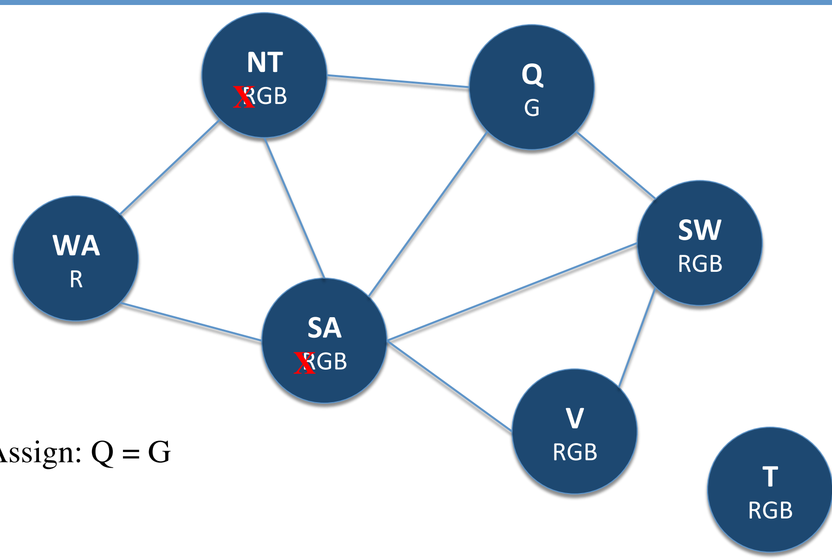


Forward Checking Example



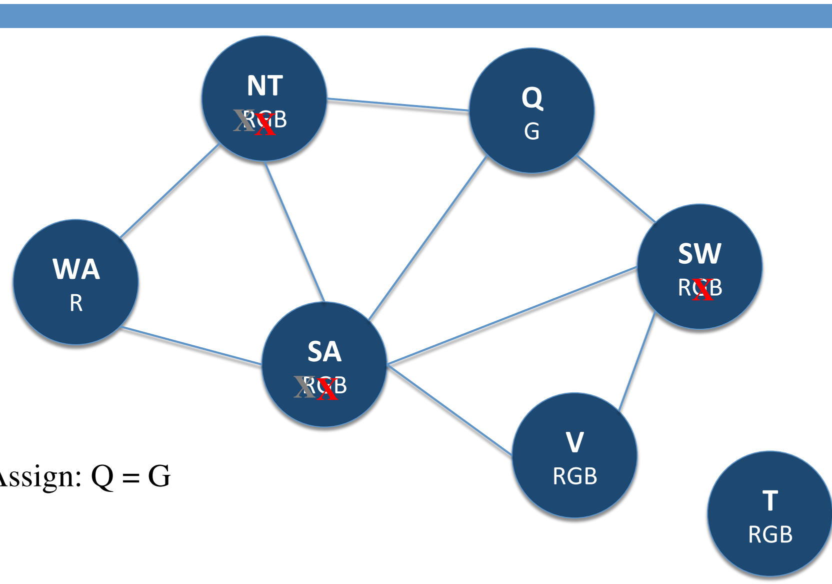
Assign: Q = G

Forward Checking Example



Assign: Q = G

Forward Checking Example



Assign: Q = G

Arc Consistency

- Basic idea:
  - Whenever we reduce the domain for a node, reprocess the edges it's connected to.
  - Reprocessing: for an edge between (A,B)
    - remove any values from Dom{A} for which there is no value in Dom{B} that satisfies the edge.
    - and vice-versa
    - (If domains got smaller, we must reprocess more edges!)
- Arc Consistency is very powerful
  - Can solve many problems by itself!

Arc Consistency: Run-Time

- Processing a single arc:
  - $O(d^2)$: (for each value in A, check each value in B)
- Each arc processed at most _____ times
  - 2d-1 : Arcs only reprocessed when Dom{A} or Dom{B} gets smaller… at worst one value at a time. (But we know to stop when Dom{}=0).
- At most ____ arcs/edges (fully connected)
  - n(n-1)/2 (fully connected)
- Total runtime: $O(n^2d^3)$
  - Remember: CSP includes 3SAT, which is NP-complete.
  - How can Arc Consistency be polynomial time?

k-Consistency

- *k*-consistent:
  - for any consistent assignment of $k - 1$ variables, exists consistent value of any kth
- Strongly *k*-consistent:
  - j-consistent for all $j \leq k$
- Special cases
  - $k = 1$: node consistency (maintained by FC)
  - $k = 2$: arc consistency
  - k = n: Problem is (almost) solved.
- Can choose to enforce higher-order consistency after each assignment.
  - Albeit at greater computational costs.

Your turn: Special Constraints

- Goal: Want to be able to enforce constraints at the highest possible level in the search tree in order to maximize pruning.
  - Assume all variables have an integer domain {1,9} and that you know the current set of permissible values for each variable.
  - Reformulate these constraints so that they can be applied as early as possible in the search tree:
    - (Note: there may be different constraints that you can apply at different levels!)
  - Assume domain of all variables is initially {1-9}
    - 1. All-Different(x_1, x_2, x_3, x_4)
    - 2. All-Different(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)
    - 3. All-Same(x_1, x_2, x_3);
    - 4. Sum(x_1, x_2, x_3, x_4) = 30
    - 5. Sum(x_1, x_2, x_3, x_4) is Odd
    - 6. Product(x1, x2, x3, x4) = 18
    - 7. IsPrime(100*x1 + 10*x2 + x3)

Challenge Winners

Your turn: Cryptarithmetic Problem

- What the graph look like?
- Solve it using MRV, forward checking
  - Variable ordering: {c1, c2, T, W, O, F, U, R}
  - Static analysis: F = 1.

```
    c2 c1
    T  W  O
  + T  W  O
 ----------
  F O  U  R
```

(removing this problem… I've never gotten through it without making about six mistakes… very very grungy.)

Picking values

- We've talked a lot about which variable to substitute…
- Does it matter which order we try the values in the domain?
  - Yes! If we try the *likely* values first, we'll find a solution faster.

Picking Values

- Least constrained value
  - Which value rules out the fewest values nearby?
  - Pursue most promising directions first
- Other heuristics
  - Most probable *a priori*
  - Cryptograms: for a given ciphertext word, try common plaintext words first.

MRV vs. LCV?

- Minimum Remaining Values
  - Pick variable with *fewest* values left in its domain
- Least Constrained Value:
  - Pick value with *most* possible children
- Why do we maximize one and minimize the other?
  - To solve the problem, we must eventually assign every variable, so pick the one with the smallest branching factor (MRV).
  - Once we've picked a variable, we must ultimately rule out all possibilities, so look for most promising values first.
    - Hope that we won't have to try other values later on....

BT Refinement: Perspectives

Look Back

- Reasoning about what to do after failure
- Backjumping
  - Backtrack to some decision *before* most recent

Look Ahead

- Reasoning about how to make better assignments
- Examples
  - Ordering heuristics
  - Constraint propagation: FC, MAC,… MkC

Basic Back Goal

- Our goal is to jump back up as far as possible
  - Safe jump: don't miss a solution
  - Know that the sub-tree we skip is unsolvable

Basic Back Jumping

- *Conflict Set(X_i) :*
  - For each value in Dom{X_i}, what is the earliest variable that is inconsistent with it?
    - Suppose Dom{X_7} = {v_1, v_2}
    - Suppose $X_7=v_1$ is incompatible with the current values of X_3 and X_5. We'd have to go all the way back up to X_3 to make $X_7=v_1$ possible.
    - Suppose $X_7=v_2$ is incompatible with the current values of X_4, X_5, and X_6. We'd have to go all the way back up to X_4 to make $X_7=v_2$ possible.
    - Conflict Set(X_7) = {X_3, X_4}
  - We backjump to X_4: a different value of X_4 might allow an assignment of X_7 (X_7 = v_2).

Forward Checking?

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

from (Kondrak & van Beek, 1997)

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	Q	5
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	Q	5
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

Q_6 conflict set = {1,2,3,5}, Jump back to 5…

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

Q_5: nothing left to try. back up.

6-Queens Example

1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		**4**
5		Q	2	1	2	2
6			2	**Q**	1	3
	1	2	3	4	5	6

Q_4: Try row 6.

6-Queens Example

1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	**Q**	*4*
5		Q	2	1	2	2
6			2	**Q**	1	3
	1	2	3	4	5	6

Q_5: Try row 4.

6-Queens Example

1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	**Q**	*4*
5		Q	2	1	2	2
6			2	**Q**	1	3
	1	2	3	4	5	6

Q_6 conflict set = {1,2,3,4}

6-Queens Example

1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		*4*
5		Q	2	1	2	2
6			2	**Q**	1	3
	1	2	3	4	5	6

Backjump to Q_4

Conflict-Directed Backjumping (CBJ)

- In ordinary back-jumping, we consider the conflict set at just the current search node
  - When we jump back, we "forget" why we did it.
- We can do better by propagating conflict set information back *up* the tree
  - Allows us to "remember" the constraints of *future* variable assignments.

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

from (Kondrak & van Beek, 1997)

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	Q	5
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	Q	5
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

Q_6 conflict set = {1,2,3,5}, Jump back to 5…

6-Queens Example

1		1		Q	3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3	{1,2,3}	{1,2,3,5}
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

Combine Q6 conflicts with local conflicts
{1,2,3} = {1,2,3,5} U {1,2,3} - 5

6-Queens Example

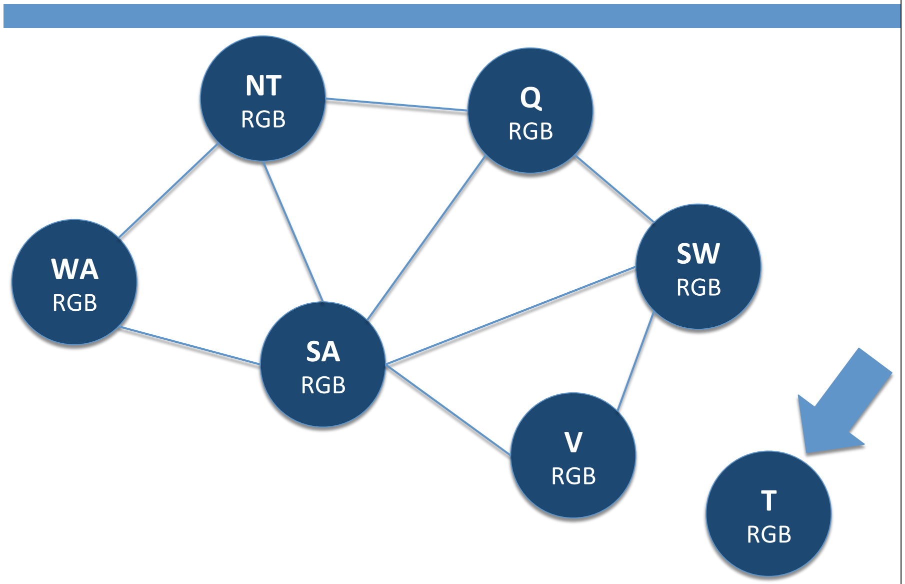
1		1			3	2
2	Q	1	1	1	1	1
3		1	Q	2	3	3
4			1	3		
5		Q	2	1	2	2
6			2		1	3
	1	2	3	4	5	6

Backjump to Q_3 !

Conflict-based BackJump Wrap-Up

- By transferring conflict set, we preserve key information across backtracks, pruning a much larger part of the search space

Disconnected Sub-graphs



Sub problems

- Finding independent sub-problems is rare, but wonderful
  - Original problem: $O(d^n)$
  - Split in two equal-sized sub-problems: $O(d^{n/2})$

Your turn

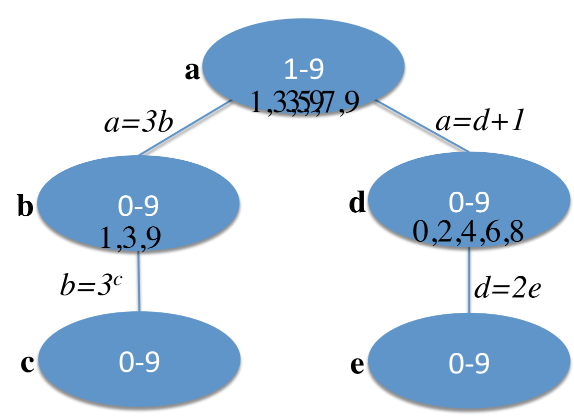
- Let 'abcde' be a 5 digit number (a!=0) such that:
  - a = 3b
  - b = 3^c
  - a = d+1
  - d = 2e
  - *Is this an easy or hard problem?*

Trees

- Trees are great too!
- Starting from the leaves:
  - Apply arc consistency to the parent, removing values from parent domain.
  - Now, the leaves can always find a value consistent with their parent.
- Start from the root:
  - Pick any value for the node consistent with its parent.
- Runtime?
  - $nd^2 + nd$

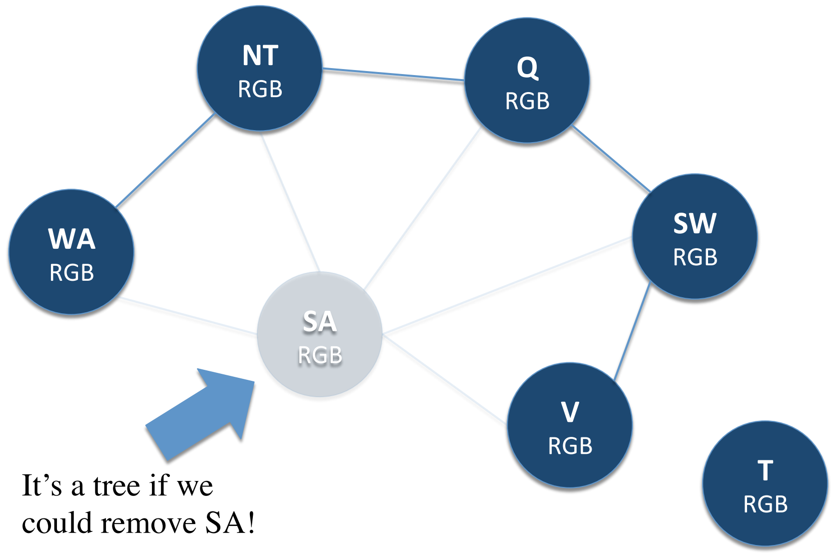
Trees

- Let 'abcde' be a 5 digit number (a!=0) such that:
  - a = 3b
  - b = 3^c
  - a = d+1
  - d = 2e



There are two solutions! 31021 and 93184

Transforming Problems into Trees



It's a tree if we could remove SA!

Tree-ification

- Pick nodes S that turn the problem into a tree
- For all possible assignments to S
  - Solve the induced tree
    - If solution found, return it

Local Search

- Local search is applicable to CSP too!
- Advantages
  - Can be very fast
  - Replanning
    - Produces solutions similar to earlier solutions

Local Search for CSPs

- Search in space of complete assignments
- Min-conflicts heuristic
  - Choose variable to reassign
  - Pick value minimizing number of conflicts with neighbors in constraint graph
- For *n*-queens, search time empirically independent of *n*
  - Solutions are fairly densely distributed around the state space: any initial guess never has far to go!

GSAT: Local Search for SAT

```
procedure GSAT(Σ)
    for i := 1 to Max-tries
        T := random truth assignment
        for j := 1 to Max-flips
            if T satisfies Σ then return T
            else Poss-flips := set of vars that increase satisfiability most
                V := a random element of Poss-flips
                T := T with V's truth assignment flipped
        end
    end
    return "no satisfying assignment found"
```

Next Time

- AI in Medicine, Prof. Syed
- Recitation: hints/suggestions for programming challenge!