# Constraint Satisfaction Problems (CSP)

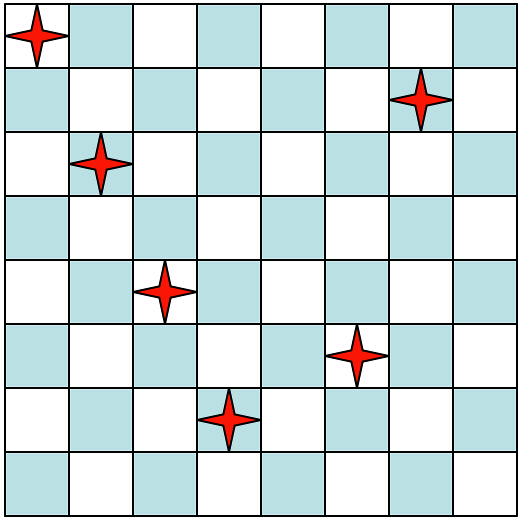
(Where we postpone making difficult decisions until they become easy to make)

R&N: Chap. 5

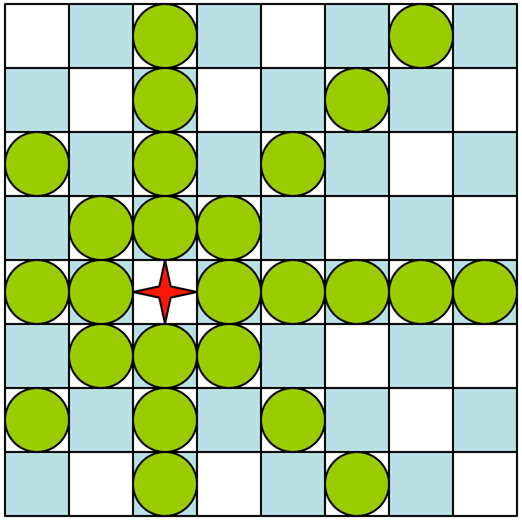
# What we will try to do ...

- Search techniques make choices in an often arbitrary order. Often little information is available to make each of them
- In many problems, the same states can be reached independent of the order in which choices are made ("commutative" actions)
- Can we solve such problems more efficiently by picking the order appropriately? Can we even avoid making any choice?

# Uninformed Search

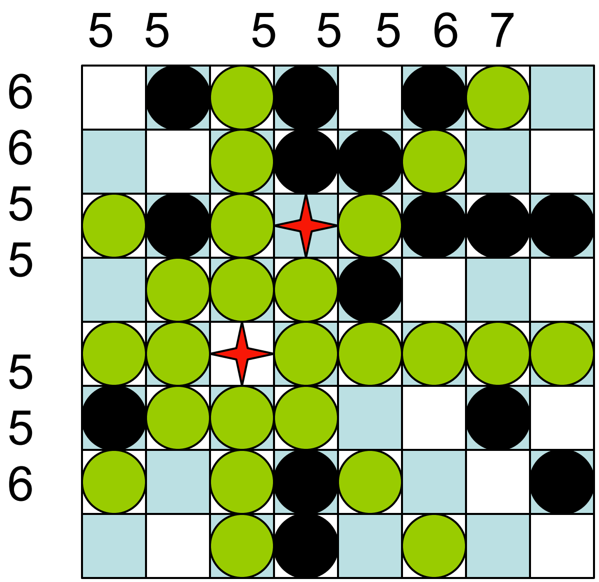


# Constraint Propagation



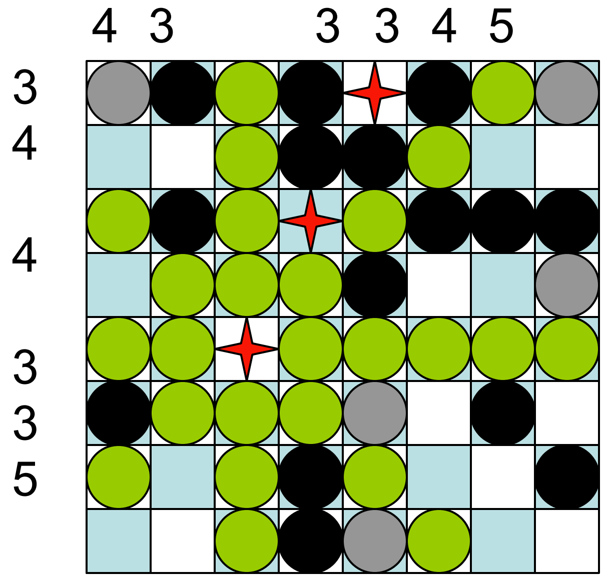
- Place a queen in a square
- Remove the attacked squares from future consideration

# Constraint Propagation



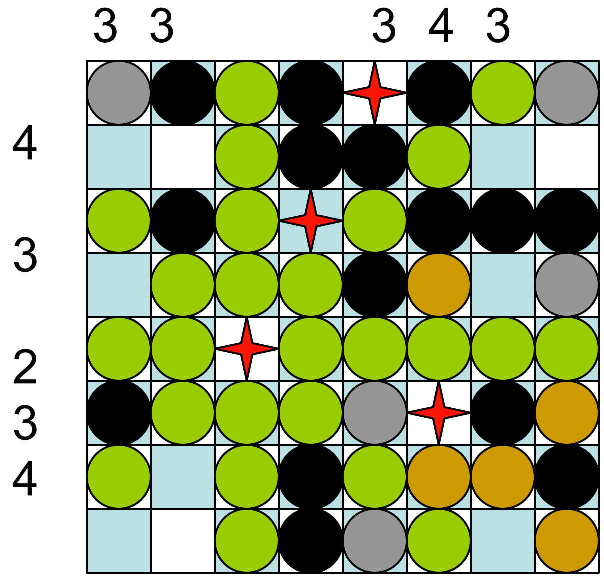

- Count the number of non-attacked squares in every row and column
- Place a queen in a row or column with minimum number
- Remove the attacked squares from future consideration

# Constraint Propagation

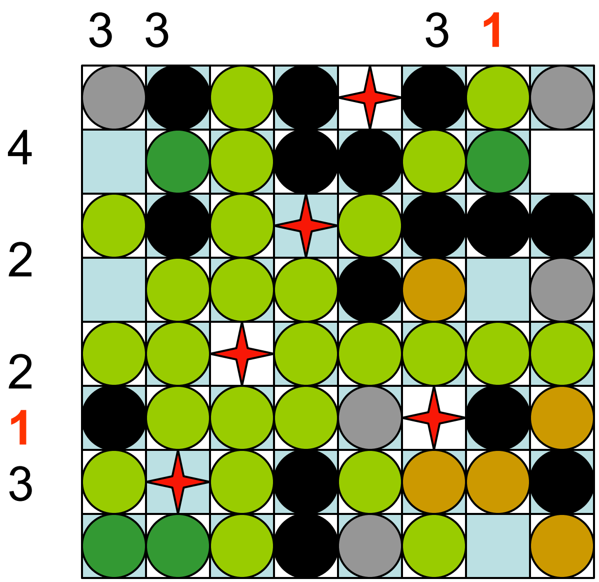



- Repeat

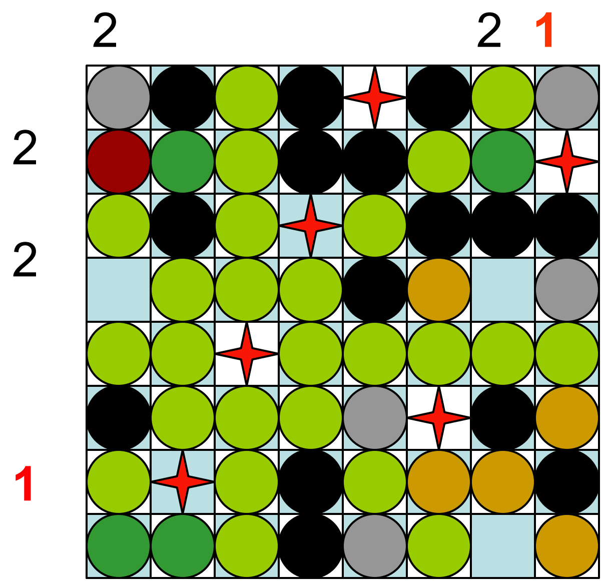
# Constraint Propagation



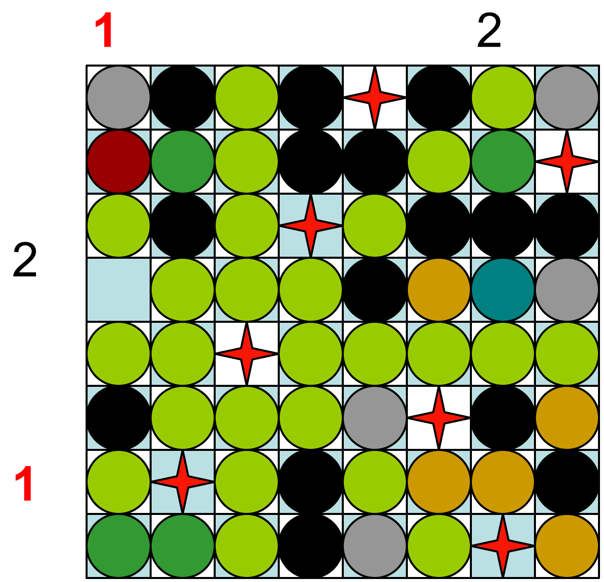
# Constraint Propagation



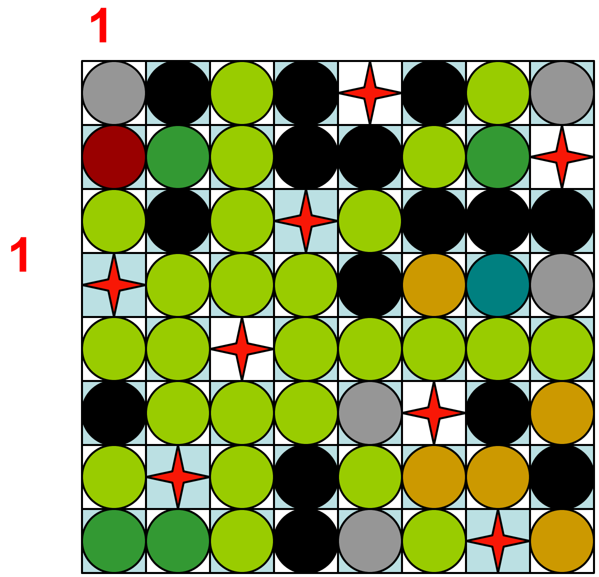
# Constraint Propagation



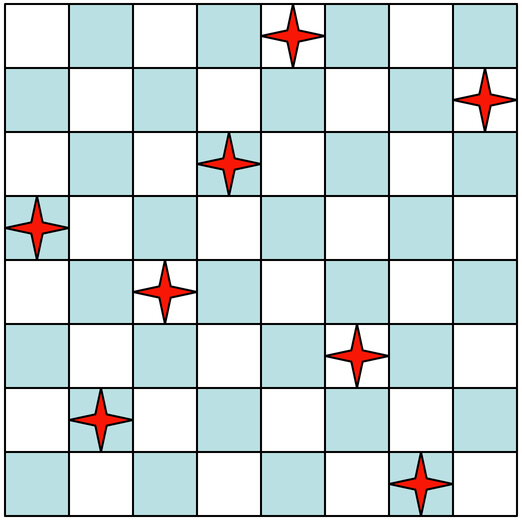
# Constraint Propagation



# Constraint Propagation



# Constraint Propagation



# What do we need?

- More than just a successor function and a goal test
- We also need:
  - A means to propagate the constraints imposed by one queen's position on the positions of the other queens
  - An early failure test

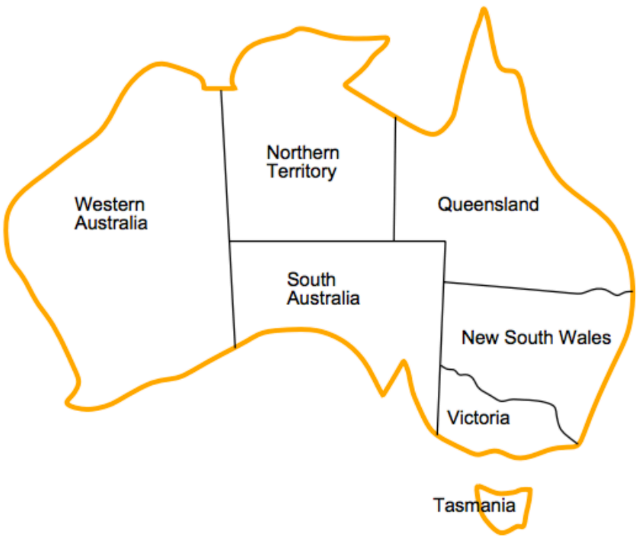
→ Explicit representation of constraints

→ Constraint propagation algorithms

# Constraint Satisfaction Problem (CSP)

- Set of variables $\{X_1, X_2, \ldots, X_n\}$
- Each variable $X_i$ has a domain $D_i$ of possible values. Usually, $D_i$ is finite
- Set of constraints $\{C_1, C_2, \ldots, C_p\}$
- Each constraint relates a subset of variables by specifying the valid combinations of their values
- Goal: Assign a value to every variable such that all constraints are satisfied

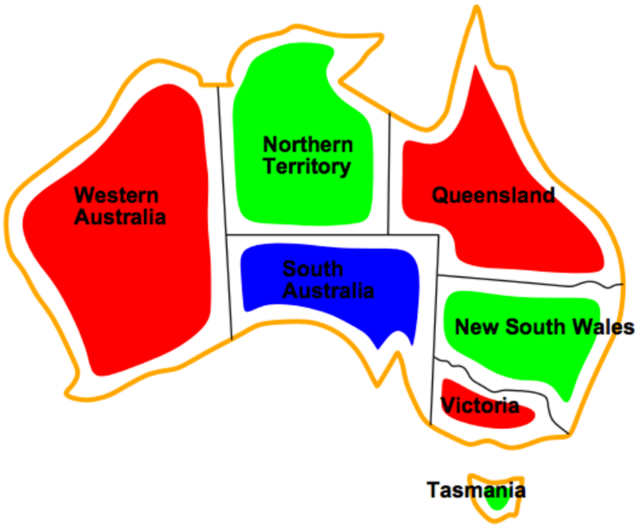
# Map Coloring




- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain:
  {red, green, blue}
- No two adjacent variables have the same value:

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

# Map Coloring




- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain:

  {red, green, blue}
- No two adjacent variables have the same value:

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

# 8-Queen Problem

- 8 variables $X_i$, $i = 1$ to 8
- The domain of each variable is: {1,2,…,8}
- Constraints are of the forms:
  - $X_i = k \rightarrow X_j \neq k$ for all $j = 1$ to 8, $j \neq i$
  - Similar constraints for diagonals

All constraints are binary

# Street Puzzle

1 2 3 4 5

$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}
$C_i$ = {Red, Green, White, Yellow, Blue}
$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}
$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}
$A_i$ = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

Who owns the Zebra?
Who drinks Water?

# Street Puzzle

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}
$C_i$ = {Red, Green, White, Yellow, Blue}
$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}
$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}
$A_i$ = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor’s
The Horse is next to the Diplomat’s

$\forall i,j \in [1,5],\ i \neq j,\ N_i \neq N_j$

$\forall i,j \in [1,5],\ i \neq j,\ C_i \neq C_j$

…

# Street Puzzle

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

**$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}**
**$C_i$ = {Red, Green, White, Yellow, Blue}**
**$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}**
**$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}**
**$A_i$ = {Dog, Snails, Fox, Horse, Zebra}**

**The Englishman lives in the Red house** → $(N_i = \text{English}) \Leftrightarrow (C_i = \text{Red})$
**The Spaniard has a Dog**
**The Japanese is a Painter** → $(N_i = \text{Japanese}) \Leftrightarrow (J_i = \text{Painter})$
**The Italian drinks Tea**
**The Norwegian lives in the first house on the left** → $(N_1 = \text{Norwegian})$
**The owner of the Green house drinks Coffee**
**The Green house is on the right of the White house** → 

$$\begin{cases} (C_i = \text{White}) \Leftrightarrow (C_{i+1} = \text{Green}) \\ (C_5 \neq \text{White}) \\ (C_1 \neq \text{Green}) \end{cases}$$

**The Sculptor breeds Snails**
**The Diplomat lives in the Yellow house**
**The owner of the middle house drinks Milk**
**The Norwegian lives next door to the Blue house**
**The Violinist drinks Fruit juice**
**The Fox is in the house next to the Doctor's**
**The Horse is next to the Diplomat's**

→ left as an exercise

# Street Puzzle

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}
$C_i$ = {Red, Green, White, Yellow, Blue}
$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}
$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}
$A_i$ = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house → $(N_i = \text{English}) \Leftrightarrow (C_i = \text{Red})$
The Spaniard has a Dog
The Japanese is a Painter → $(N_i = \text{Japanese}) \Leftrightarrow (J_i = \text{Painter})$
The Italian drinks Tea
The Norwegian lives in the first house on the left → $(N_1 = \text{Norwegian})$
The owner of the Green house drinks Coffee
The Green house is on the right of the White house → $(C_i = \text{White}) \Leftrightarrow (C_{i+1} = \text{Green})$, $(C_5 \neq \text{White})$, $(C_1 \neq \text{Green})$
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

unary constraints: $(N_1 = \text{Norwegian})$, $(C_5 \neq \text{White})$, $(C_1 \neq \text{Green})$

# Street Puzzle

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}
$C_i$ = {Red, Green, White, Yellow, Blue}
$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}
$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}
$A_i$ = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left → $N_1$ = Norwegian
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk → $D_3$ = Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

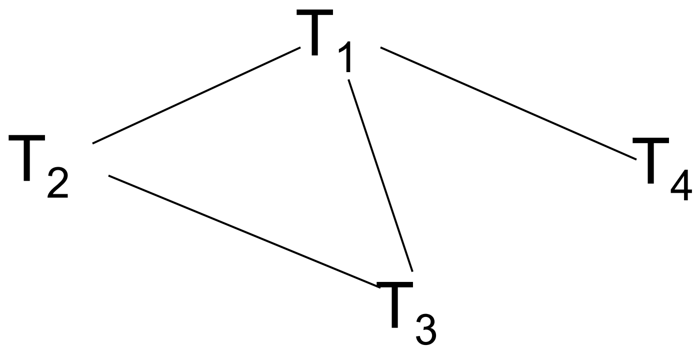
# Street Puzzle

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

$N_i$ = {English, Spaniard, Japanese, Italian, Norwegian}
$C_i$ = {Red, Green, White, Yellow, Blue}
$D_i$ = {Tea, Coffee, Milk, Fruit-juice, Water}
$J_i$ = {Painter, Sculptor, Diplomat, Violinist, Doctor}
$A_i$ = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house → $C_1 \neq$ Red
The Spaniard has a Dog → $A_1 \neq$ Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left → $N_1$ = Norwegian
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk → $D_3$ = Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice → $J_3 \neq$ Violinist
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

# Task Scheduling




Four tasks $T_1$, $T_2$, $T_3$, and $T_4$ are related by time constraints:

- $T_1$ must be done during $T_3$
- $T_2$ must be achieved before $T_1$ starts
- $T_2$ must overlap with $T_3$
- $T_4$ must start after $T_1$ is complete

- Are the constraints compatible?
- What are the possible time relations between two tasks?
- What if the tasks use resources in limited supply?

How to formulate this problem as a CSP?

# 3-SAT

- n Boolean variables $u_1, \ldots, u_n$
- p constraints of the form

$$u_i^* \vee u_j^* \vee u_k^* = 1$$

  where $u^*$ stands for either $u$ or $\neg u$

- Known to be NP-complete

# Finite vs. Infinite CSP

- Finite CSP: each variable has a finite domain of values
- Infinite CSP: some or all variables have an infinite domain

  E.g., linear programming problems over the reals:

  for $i = 1, 2, \ldots, p : a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n = a_{i,0}$

  for $j = 1, 2, \ldots, q : b_{j,1}x_1 + b_{j,2}x_2 + \ldots + b_{j,n}x_n \leq b_{j,0}$

- We will only consider finite CSP

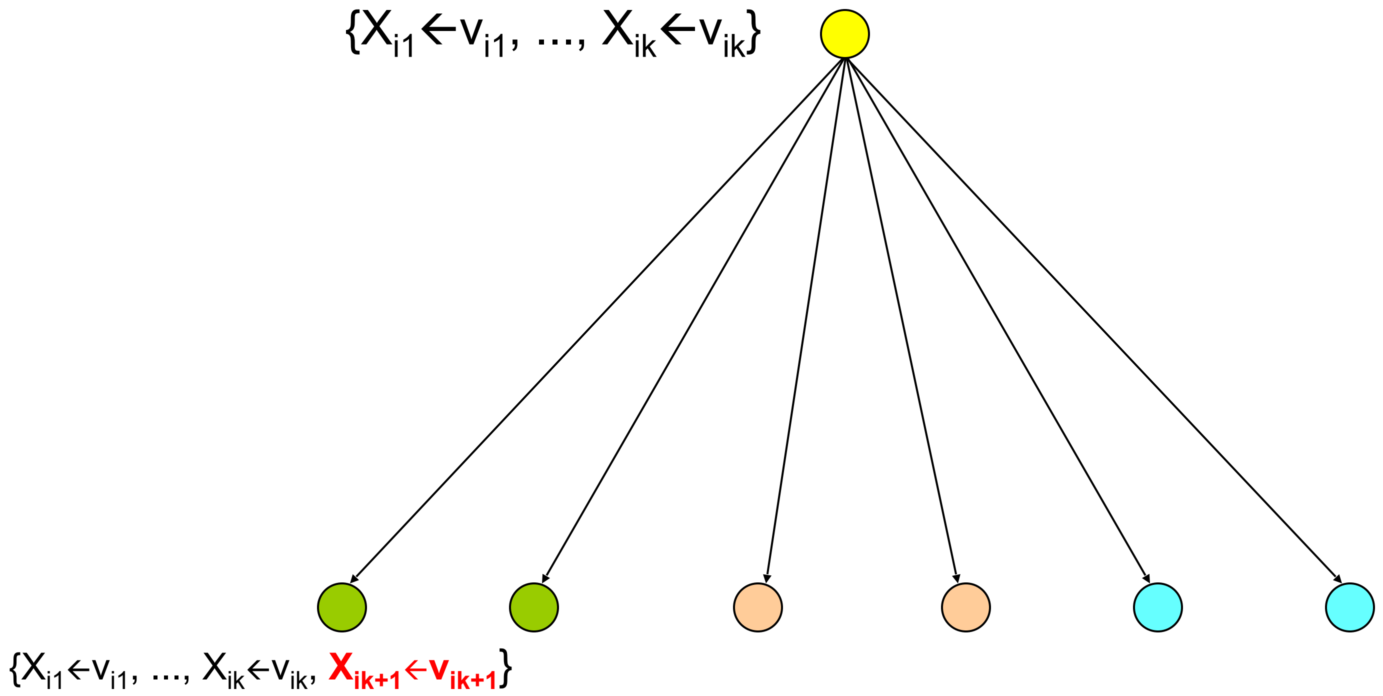
# CSP as a Search Problem

- n variables $X_1, \ldots, X_n$
- Valid assignment: $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\}$, $0 \leq k \leq n$,

  such that the values $v_{i1}, \ldots, v_{ik}$ satisfy all constraints relating the variables $X_{i1}, \ldots, X_{ik}$
- Complete assignment: one where k = n

  [if all variable domains have size d, there are $O(d^n)$ complete assignments]
- States: valid assignments
- Initial state: empty assignment {}, i.e. k = 0
- Successor of a state:

  $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\} \rightarrow \{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}, \mathbf{X_{ik+1} \leftarrow v_{ik+1}}\}$
- Goal test: k = n



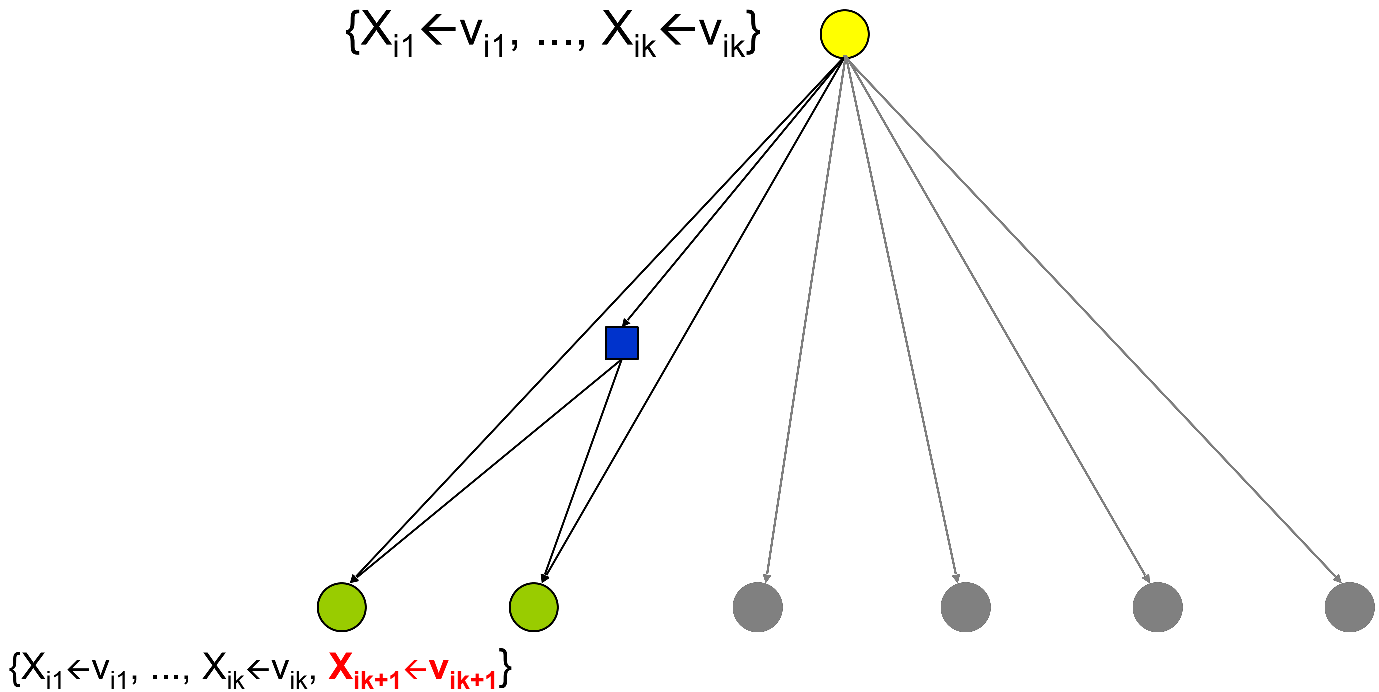

$r = n-k$ variables with $s$ values $\rightarrow$ $r \times s$ branching factor

# A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments

Hence:

1) One can expand a node N by first selecting **one** variable X not in the assignment A associated with N and then assigning every value v in the domain of X
   [→ big reduction in branching factor]




r = n-k variables with s values → r×s branching factor

n-k variables with s values → **s** branching factor

The depth of the solutions in the search tree is un-changed (n)

- 4 variables $X_1, \ldots, X_4$
- Let the valid assignment of N be:

  $$A = \{X_1 \leftarrow v_1, X_3 \leftarrow v_3\}$$

- For example pick variable $X_4$
- Let the domain of $X_4$ be $\{v_{4,1}, v_{4,2}, v_{4,3}\}$
- The successors of *A* are all the valid assignments among:

  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3\ , X_4 \leftarrow v_{4,1}\}$$

  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3\ , X_4 \leftarrow v_{4,2}\}$$

  $$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3\ , X_4 \leftarrow v_{4,3}\}$$

# A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments

Hence:

1) One can expand a node N by first selecting **one** variable X not in the assignment A associated with N and then assigning every value v in the domain of X

   [→ big reduction in branching factor]

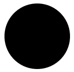
2) One need not store the path to a node

   → Backtracking search algorithm

# Backtracking Search

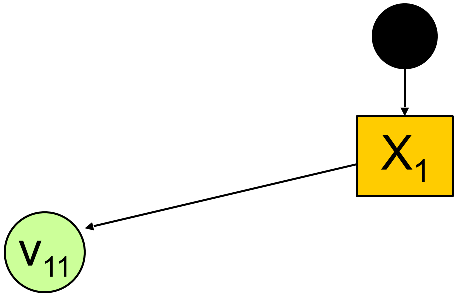
Essentially a simplified depth-first algorithm using recursion

# Backtracking Search
## (3 variables)



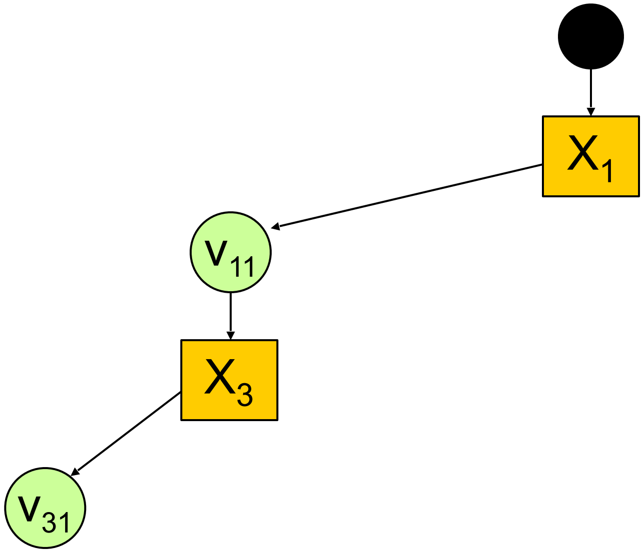
Assignment = {}

# Backtracking Search
## (3 variables)



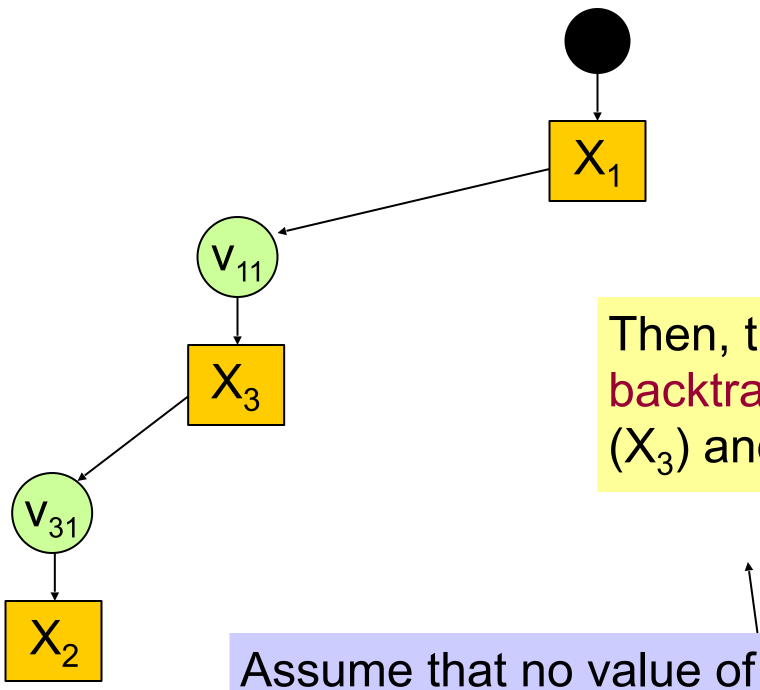

Assignment = $\{(X_1, v_{11})\}$

# Backtracking Search
## (3 variables)




Assignment = $\{(X_1,v_{11}), (X_3,v_{31})\}$

# Backtracking Search
## (3 variables)

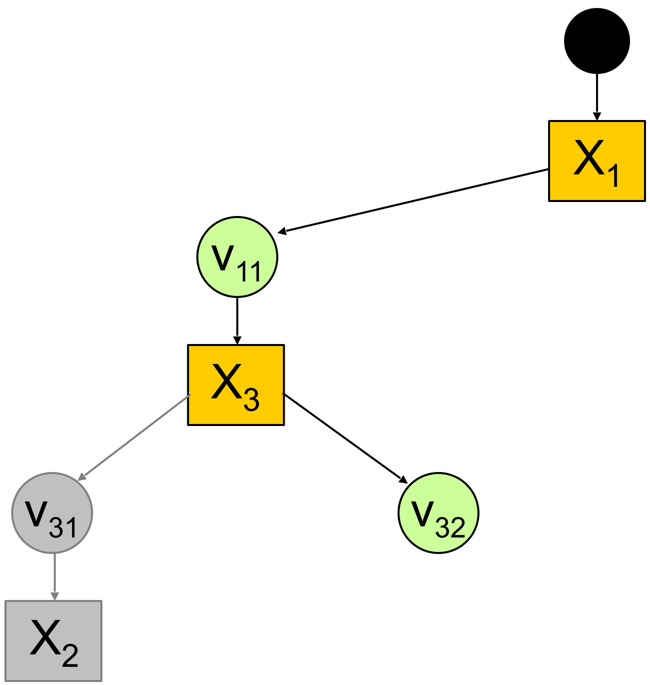



Then, the search algorithm backtracks to the previous variable ($X_3$) and tries another value

Assume that no value of $X_2$ leads to a valid assignment

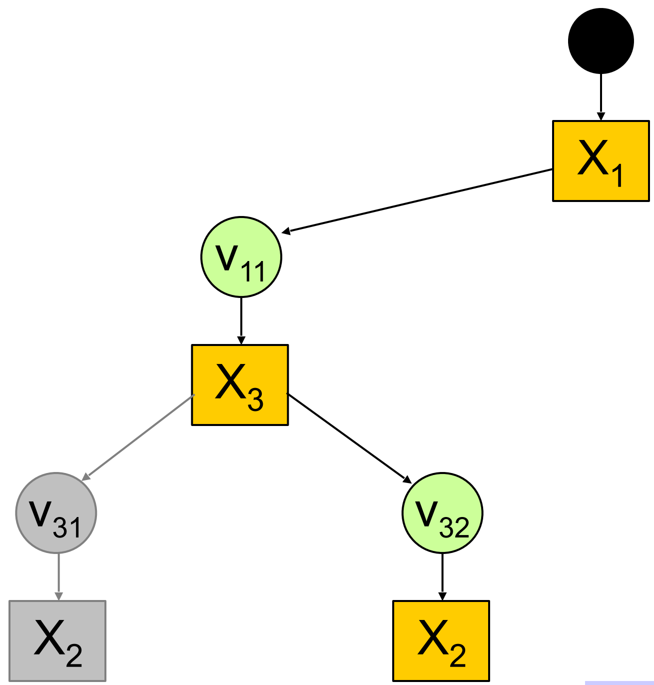
Assignment = $\{(X_1, v_{11}), (X_3, v_{31})\}$

# Backtracking Search
## (3 variables)




Assignment = $\{(X_1, v_{11}), (X_3, v_{32})\}$

# Backtracking Search
## (3 variables)

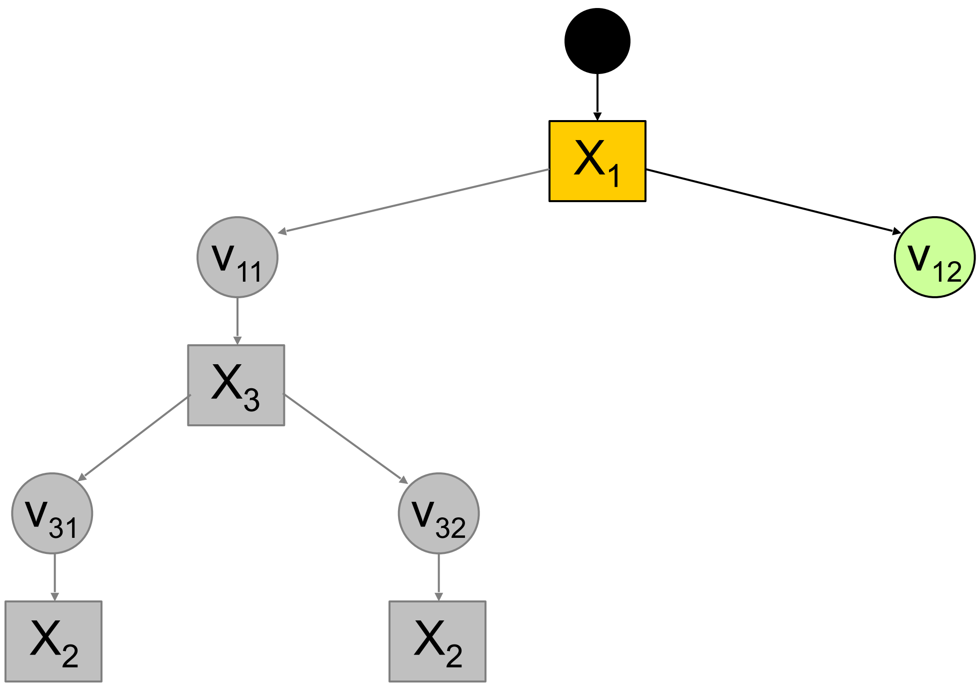



The search algorithm backtracks to the previous variable ($X_3$) and tries another value. But assume that $X_3$ has only two possible values. The algorithm backtracks to $X_1$

Assume again that no value of $X_2$ leads to a valid assignment

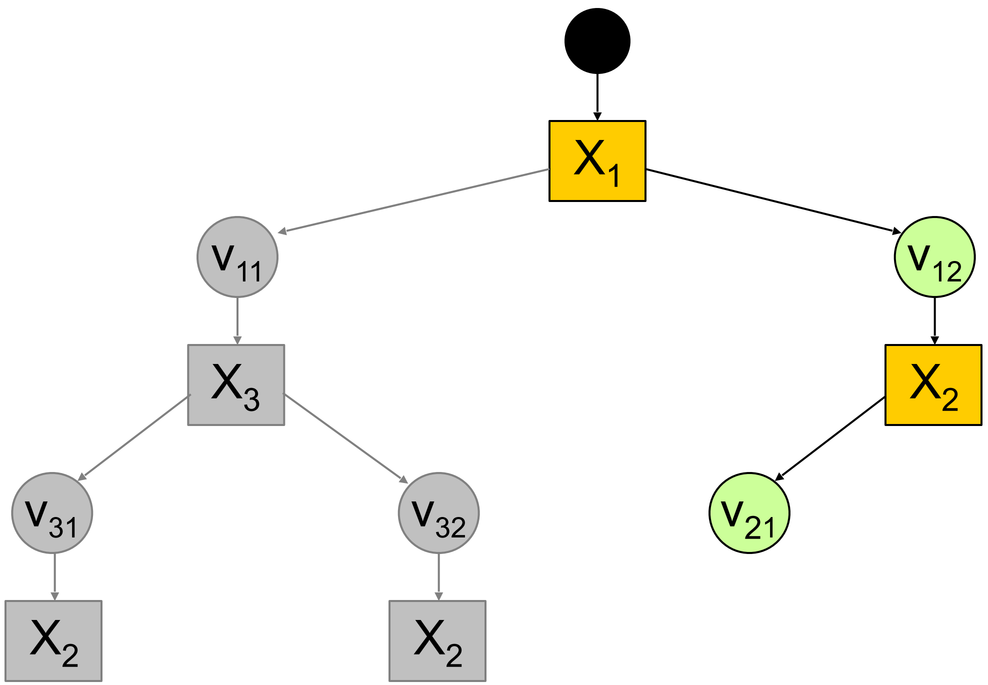
Assignment = $\{(X_1,v_{11}), (X_3,v_{32})\}$

# Backtracking Search
## (3 variables)



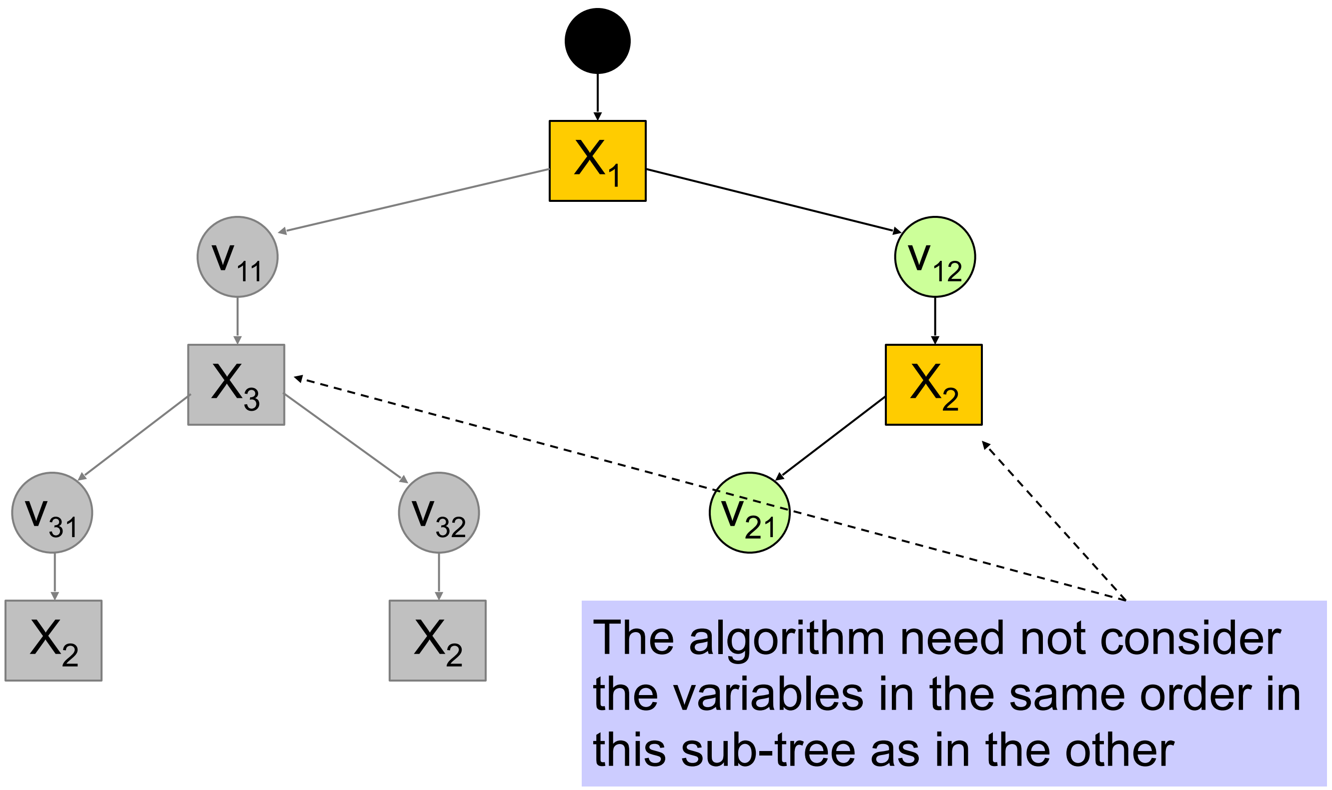

Assignment = $\{(X_1, v_{12})\}$

# Backtracking Search
## (3 variables)



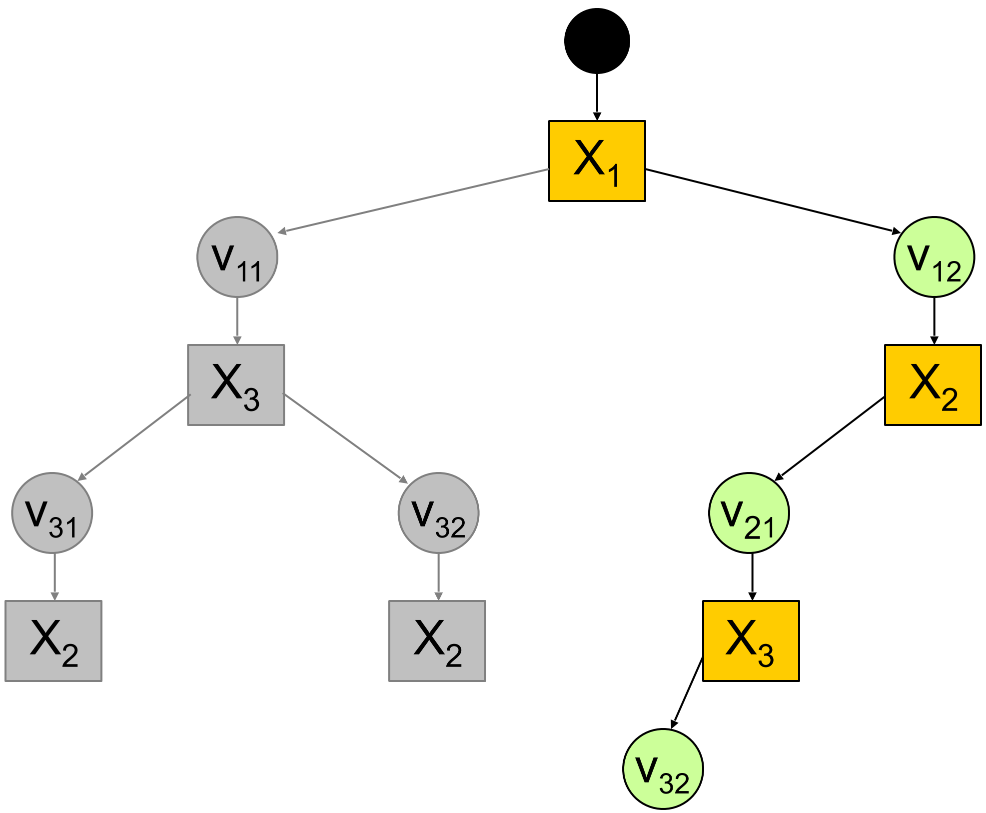

Assignment = $\{(X_1, v_{12}), (X_2, v_{21})\}$

# Backtracking Search
## (3 variables)



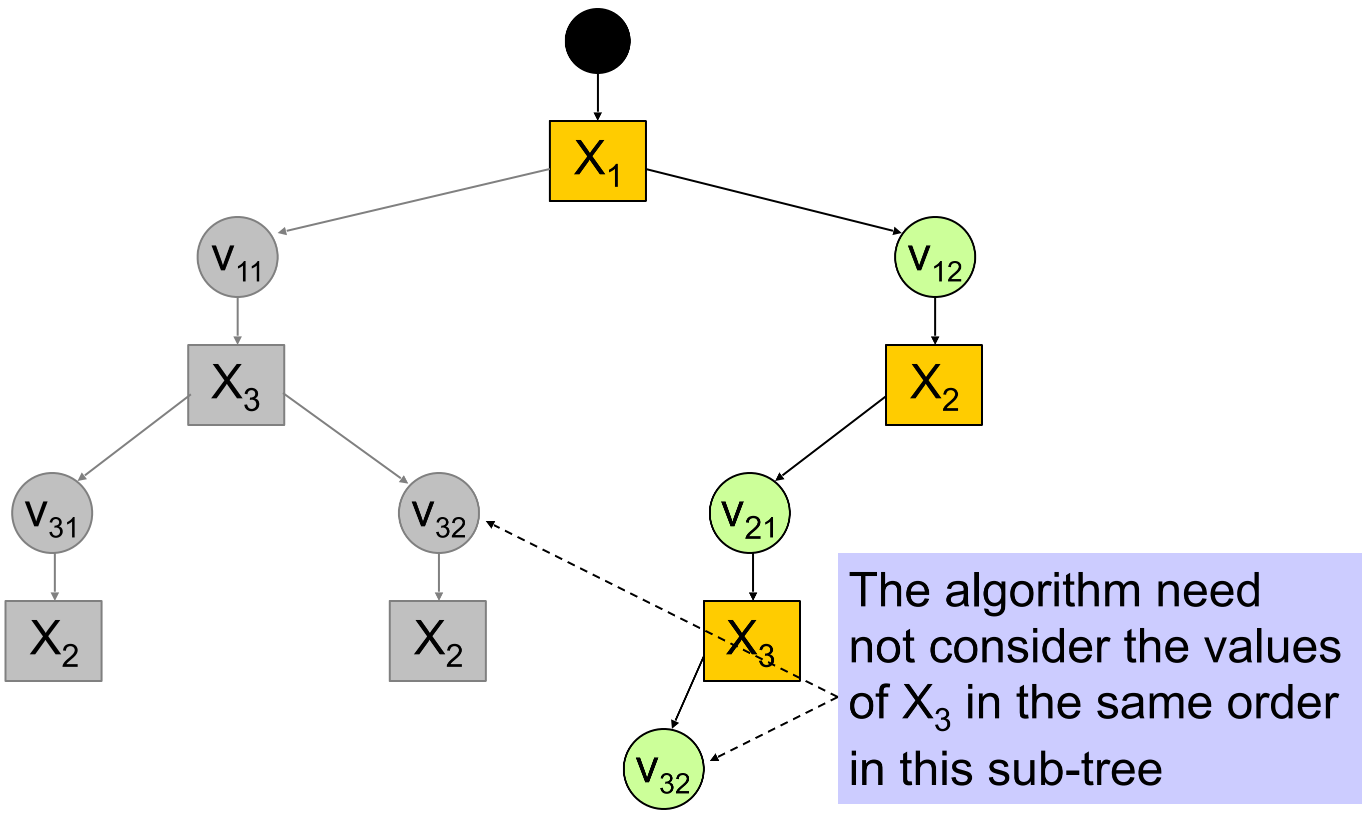

Assignment = $\{(X_1,v_{12}), (X_2,v_{21})\}$

# Backtracking Search
## (3 variables)



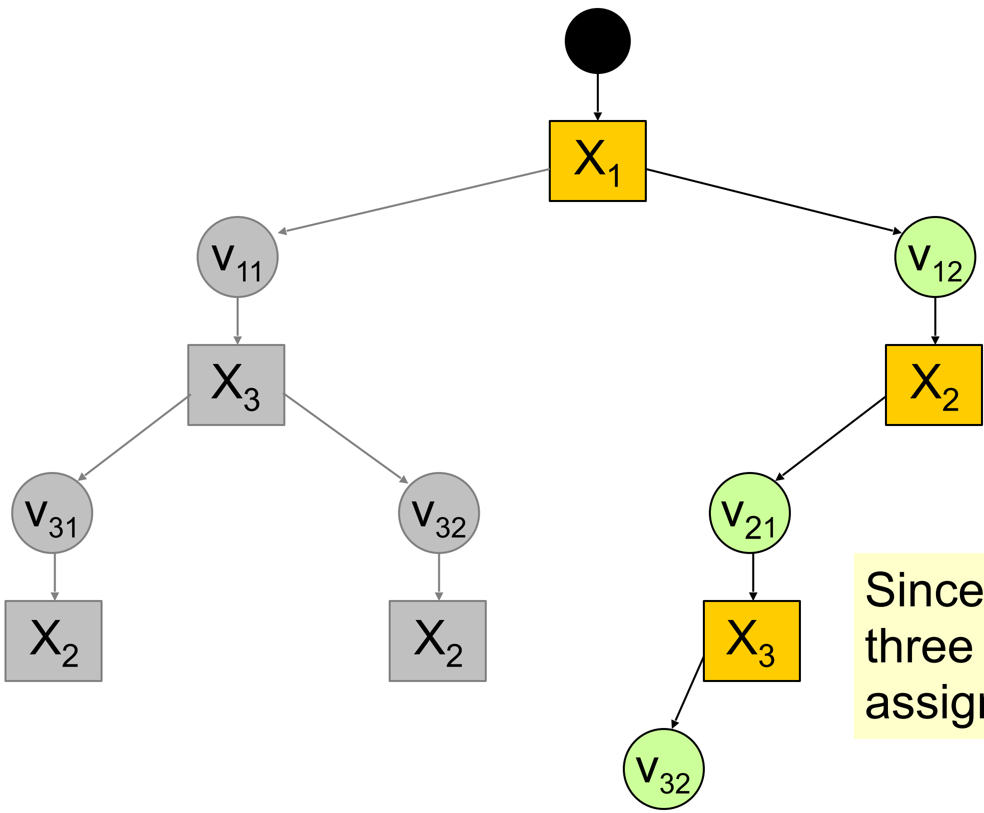

Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

# Backtracking Search
## (3 variables)




Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

# Backtracking Search
## (3 variables)




Since there are only three variables, the assignment is complete

Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

# Backtracking Algorithm

CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
    1. Add (X←v) to A
    2. If A is valid then
        1. result ← CSP-BACKTRACKING(A)
        - » If result ≠ failure then return result
    3. Remove (X←v) from A
- Return failure

Call CSP-BACKTRACKING({})

[This recursive algorithm keeps too much data in memory.
An iterative version could save memory (left as an exercise)]

# Critical Questions for the Efficiency of CSP-Backtracking

CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. X ← **select** a variable not in A
3. D ← **select** an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. If a is valid then
      i. result ← CSP-BACKTRACKING(A)
      ii. If result ≠ failure then return result
   - Remove (X←v) from A
5. Return failure

# Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?

2) In which order should X's values be assigned?

# Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?

# Critical Questions for the Efficiency of CSP-Backtracking

1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?
   The current assignment may be part of a solution. Selecting the right value to assign to X may help discover this solution more quickly

# Critical Questions for the Efficiency of CSP-Backtracking

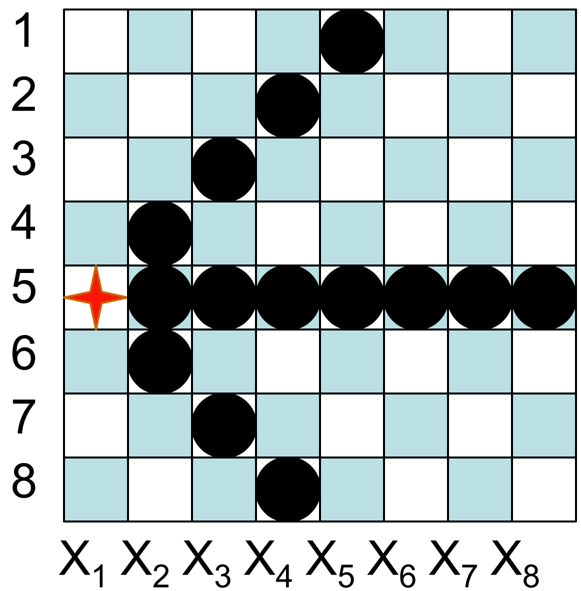
1) Which variable X should be assigned a value next?
   The current assignment may not lead to any solution, but the algorithm does not know it yet. Selecting the right variable X may help discover the contradiction more quickly

2) In which order should X's values be assigned?
   The current assignment may be part of a solution. Selecting the right value to assign to X may help discover this solution more quickly

More on these questions very soon ...

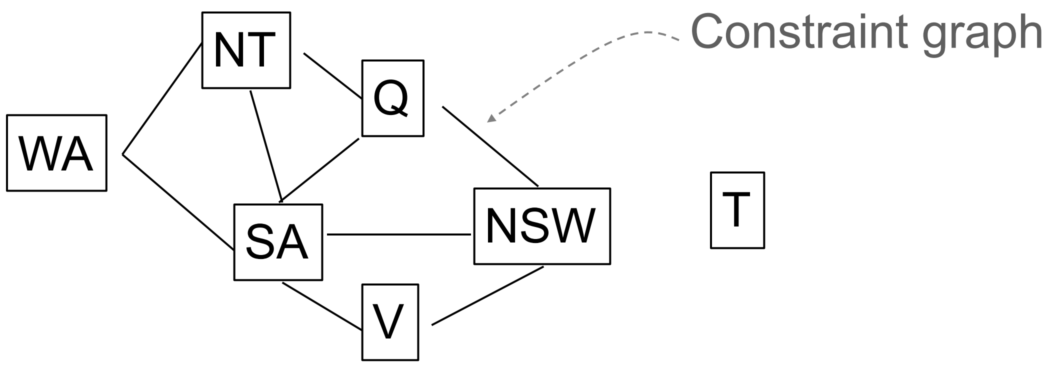
# Forward Checking

A simple constraint-propagation technique:



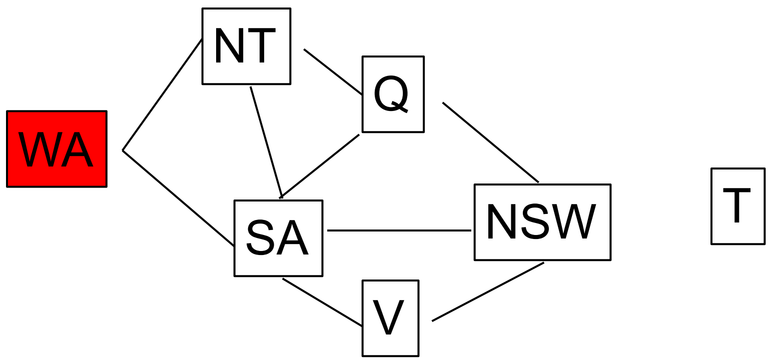

Assigning the value 5 to $X_1$ leads to removing values from the domains of $X_2$, $X_3$, ..., $X_8$

# Forward Checking in Map Coloring




| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| RGB | RGB | RGB | RGB | RGB | RGB | RGB |

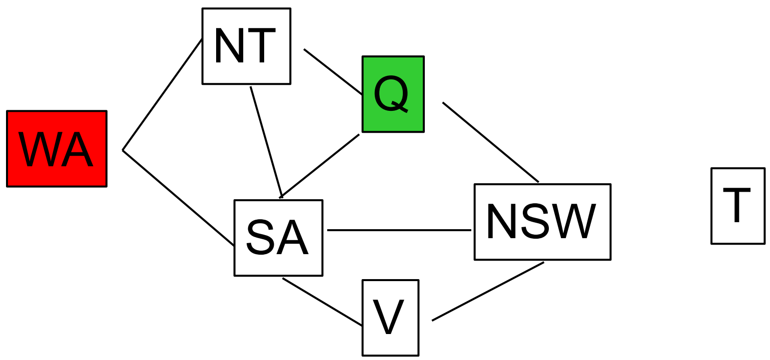
# Forward Checking in Map Coloring




| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| R | ~~R~~GB | RGB | RGB | RGB | ~~R~~GB | RGB |

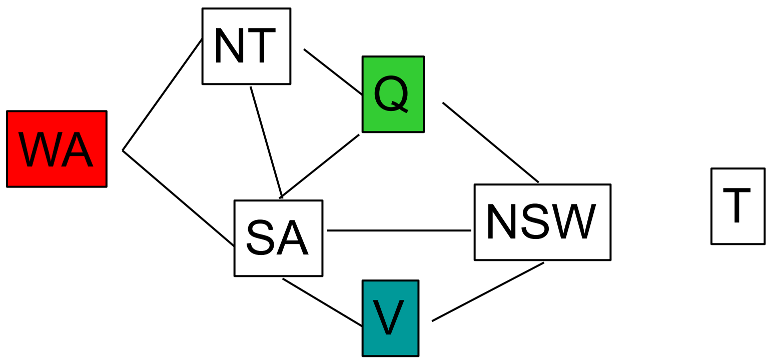
Forward checking removes the value Red of NT and of SA

# Forward Checking in Map Coloring




| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| R | GB | RGB | RGB | RGB | GB | RGB |
| R | ~~G~~B | G | R~~G~~B | RGB | ~~G~~B | RGB |

# Forward Checking in Map Coloring




| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| R | GB | RGB | RGB | RGB | GB | RGB |
| R | B | G | RB | RGB | B | RGB |
| R | B | G | R~~B~~ | B | ~~B~~ | RGB |

# Forward Checking in Map Coloring

Empty set: the current assignment
{(WA ← R), (Q ← G), (V ← B)}
does not lead to a solution

| WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|
| RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| R | GB | RGB | RGB | RGB | GB | RGB |
| R | B | G | RB | RGB | B | RGB |
| R | B | G | R~~B~~ | B | ~~B~~ | RGB |

# Forward Checking (General Form)

Whenever a pair (X←v) is added to assignment *A* do:

For each variable Y not in *A* do:

For every constraint *C* relating Y to the variables in *A* do:

Remove all values from Y's domain that do not satisfy *C*

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
    a. Add (X←v) to A
    b. var-domains ← forward checking(var-domains, X, v, A)
    c. If no variable has an empty domain then
        (i) result ← CSP-BACKTRACKING(A, var-domains)
        (ii) If result ≠ failure then return result
    d. Remove (X←v) from A
5. Return failure

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A)

1. If assignment A is complete then return A
2. X ← **select** a variable not in A
3. D ← **select** an ordering on the domain of X
4. For each value v in D do
    a. Add (X←v) to A
    b. If a is valid then
        i. result ← CSP-BACKTRACKING(A)
        ii. If result ≠ failure then return result
    - Remove (X←v) from A
5. Return failure

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   d. Remove (X←v) from A
5. Return failure

No need any more to verify that A is valid

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   d. Remove (X←v) from A
5. Return failure

Need to pass down the updated variable domains

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   1. Remove (X←v) from A
5. Return failure

1) Which variable $X_i$ should be assigned a value next?
   → Most-constrained-variable heuristic
   → Most-constraining-variable heuristic

2) In which order should its values be assigned?
   → Least-constraining-value heuristic

These heuristics can be quite confusing

Keep in mind that **all** variables must eventually get a value, while only **one** value from a domain must be assigned to each variable

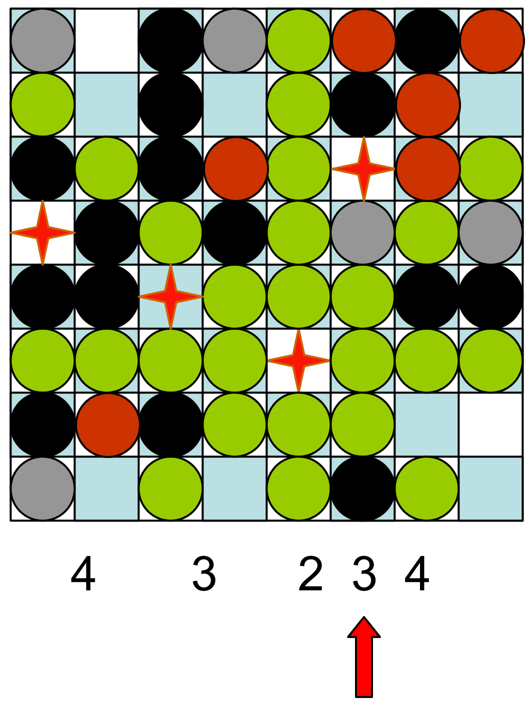
# Most-Constrained-Variable Heuristic

1) Which variable $X_i$ should be assigned a value next?

Select the variable with the smallest remaining domain

[Rationale: Minimize the branching factor]

# 8-Queens

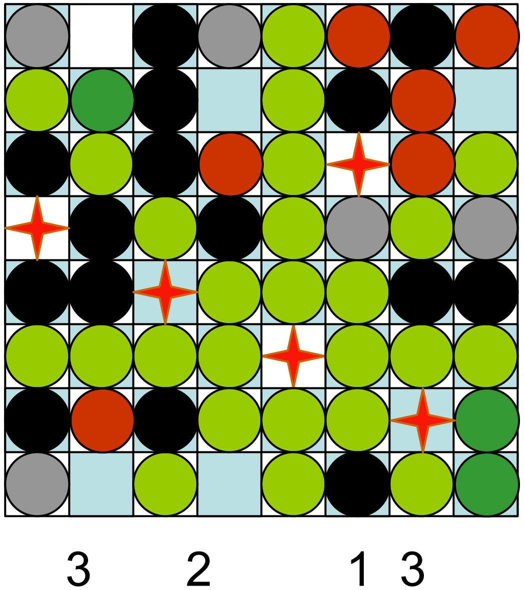



Forward checking

New assignment

Numbers of values for each un-assigned variable

# 8-Queens

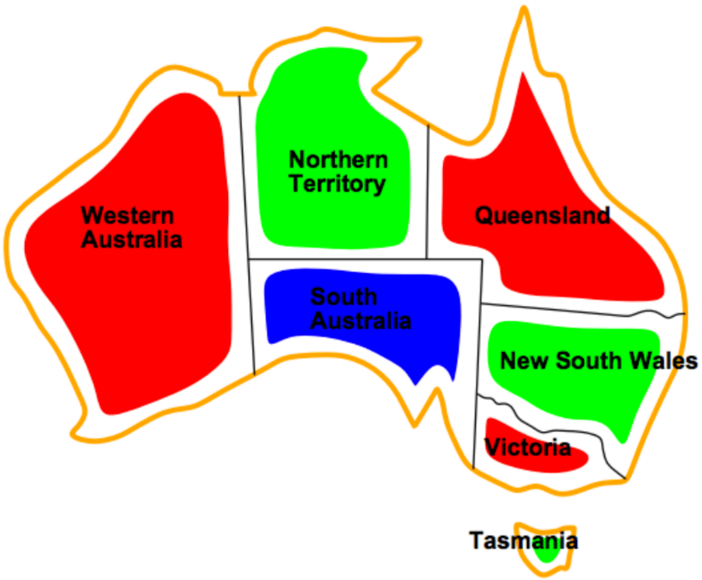



Forward checking

New assignment

New numbers of values for each un-assigned variable

# Map Coloring




- SA's remaining domain has size 1 (value Blue remaining)
- Q's remaining domain has size 2
- NSW's, V's, and T's remaining domains have size 3

→ Select SA

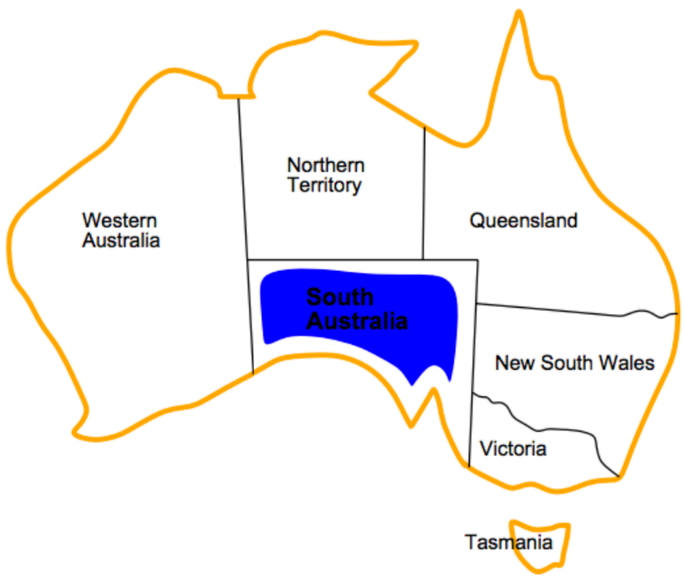
# Most-Constraining-Variable Heuristic

1) Which variable $X_i$ should be assigned a value next?

Among the variables with the smallest remaining domains (ties with respect to the most-constrained-variable heuristic), select the one that appears in the largest number of constraints on variables not in the current assignment

[Rationale: Increase future elimination of values, to reduce future branching factors]

# Map Coloring




- Before any value has been assigned, all variables have a domain of size 3, but SA is involved in more constraints (5) than any other variable

→ Select SA and assign a value to it (e.g., Blue)

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   1. Remove (X←v) from A
5. Return failure

1) Most-constrained-variable heuristic
2) Most-constraining-variable heuristic

3) Least-constraining-value heuristic

1) Select the variable with the smallest remaining domain
2) Select the variable that appears in the largest number of constraints on variables not in the current assignment

# Least-Constraining-Value Heuristic

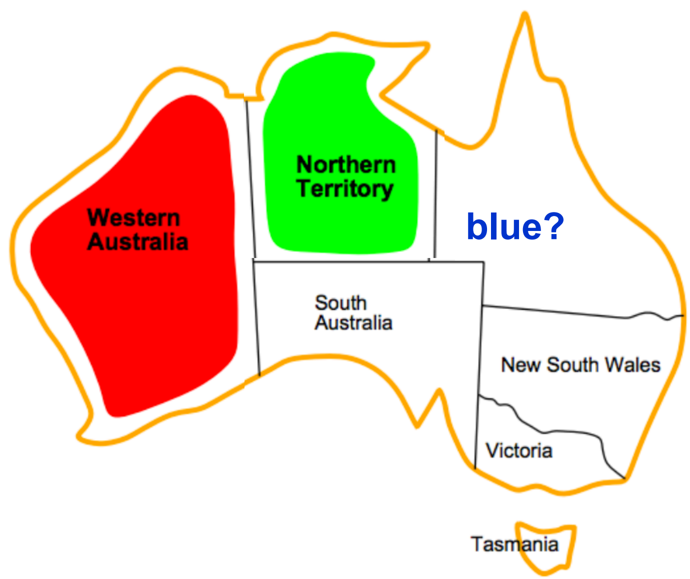
2) In which order should X's values be assigned?

Select the value of X that removes the smallest number of values from the domains of those variables which are not in the current assignment

[Rationale: Since only one value will eventually be assigned to X, pick the least-constraining value first, since it is the most likely not to lead to an invalid assignment]

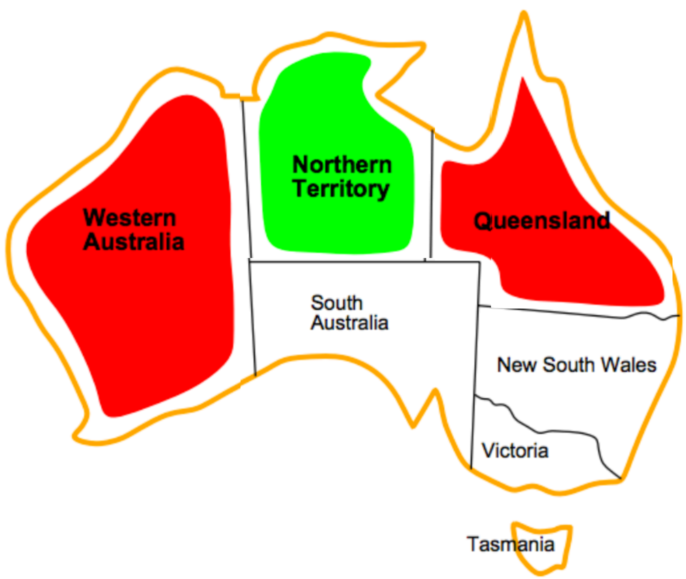
[Note: Using this heuristic requires performing a forward-checking step for every value, not just for the selected value]

# Map Coloring




- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

# Map Coloring




- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

→ So, assign Red to Q

# Modified Backtracking Algorithm

CSP-BACKTRACKING(A, var-domains)

1. If assignment A is complete then return A
2. X ← select a variable not in A
3. D ← select an ordering on the domain of X
4. For each value v in D do
   a. Add (X←v) to A
   b. var-domains ← forward checking(var-domains, X, v, A)
   c. If no variable has an empty domain then
      (i) result ← CSP-BACKTRACKING(A, var-domains)
      (ii) If result ≠ failure then return result
   1. Remove (X←v) from A
5. Return failure

1) Most-constrained-variable heuristic
2) Most-constraining-variable heuristic

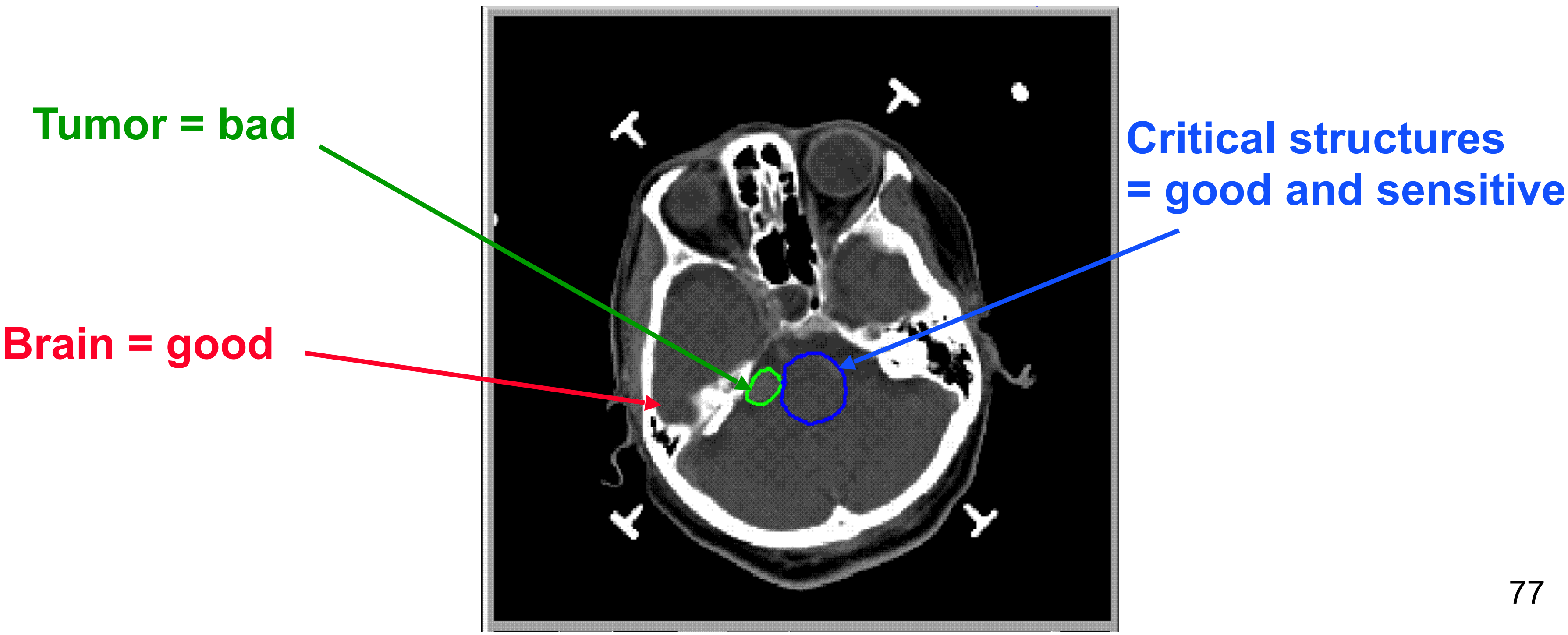
3) Least-constraining-value heuristic

# Applications of CSP

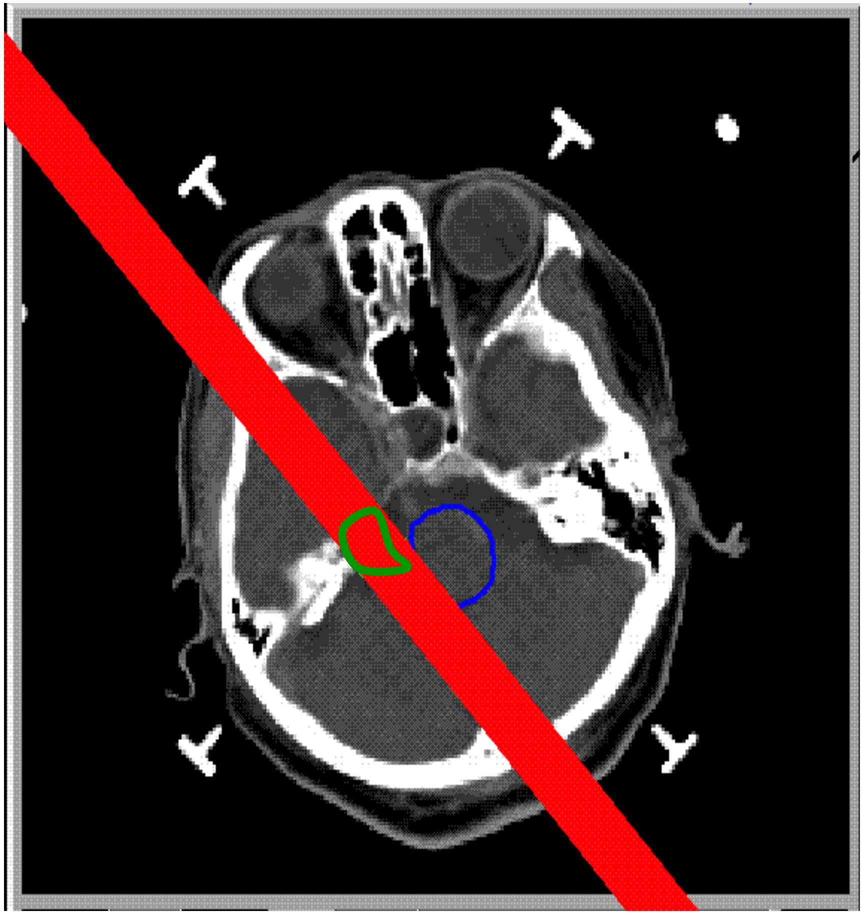
- CSP techniques are widely used
- Applications include:
  - Crew assignments to flights
  - Management of transportation fleet
  - Flight/rail schedules
  - Job shop scheduling
  - Task scheduling in port operations
  - Design, including spatial layout design
  - Radiosurgical procedures

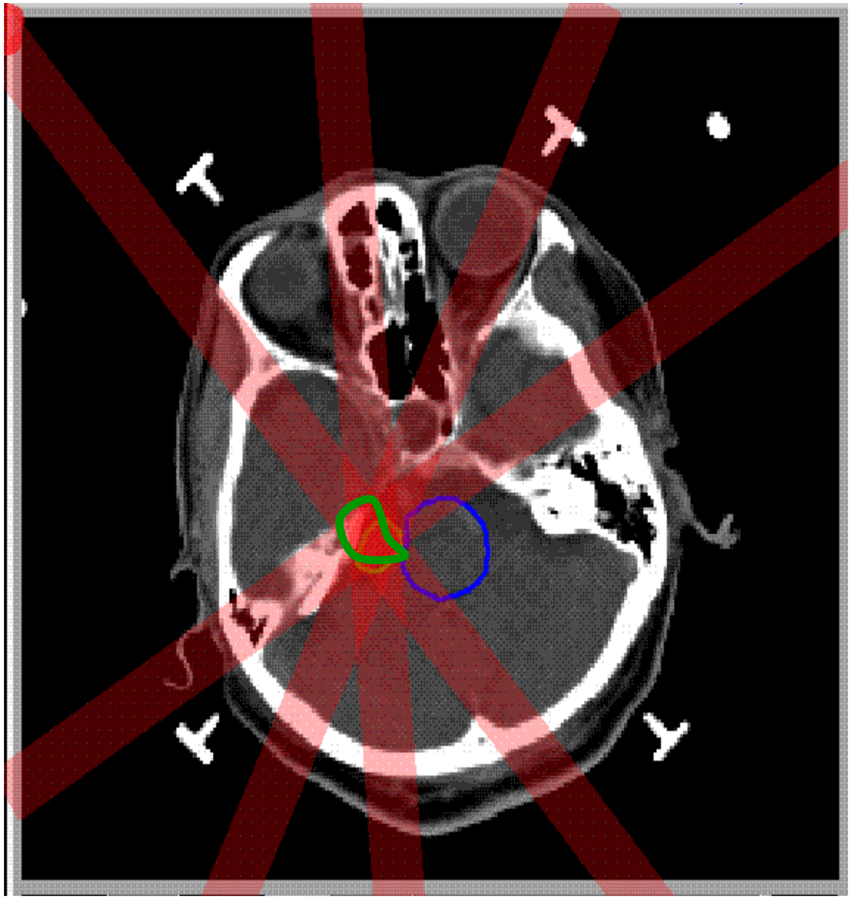
# Radiosurgery

Minimally invasive procedure that uses a beam of radiation as an ablative surgical instrument to destroy tumors



# Problem





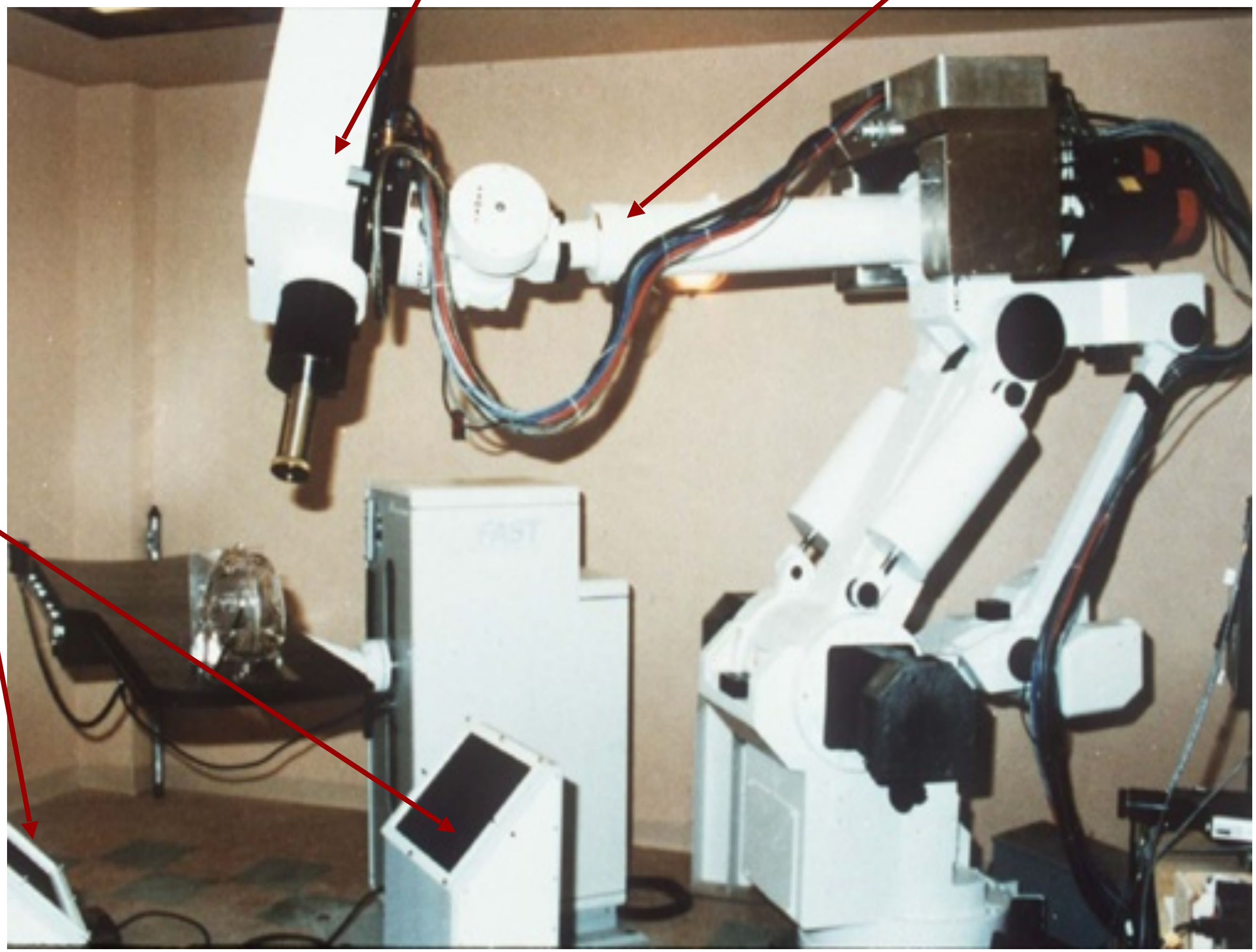
Burn tumor without damaging healthy tissue

# The CyberKnife

linear accelerator

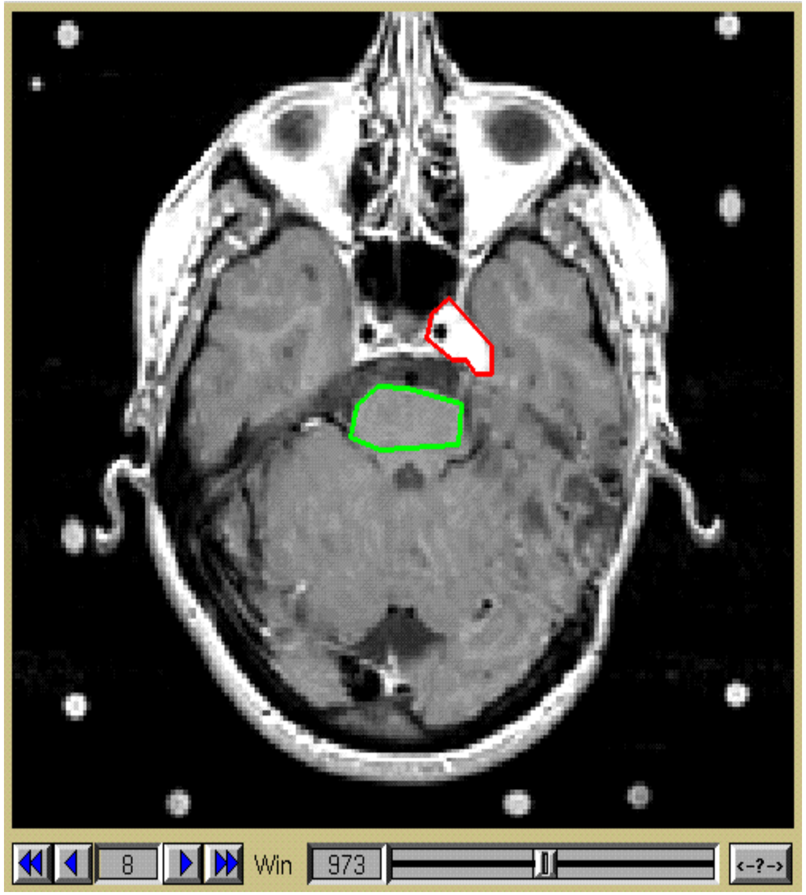
robot arm

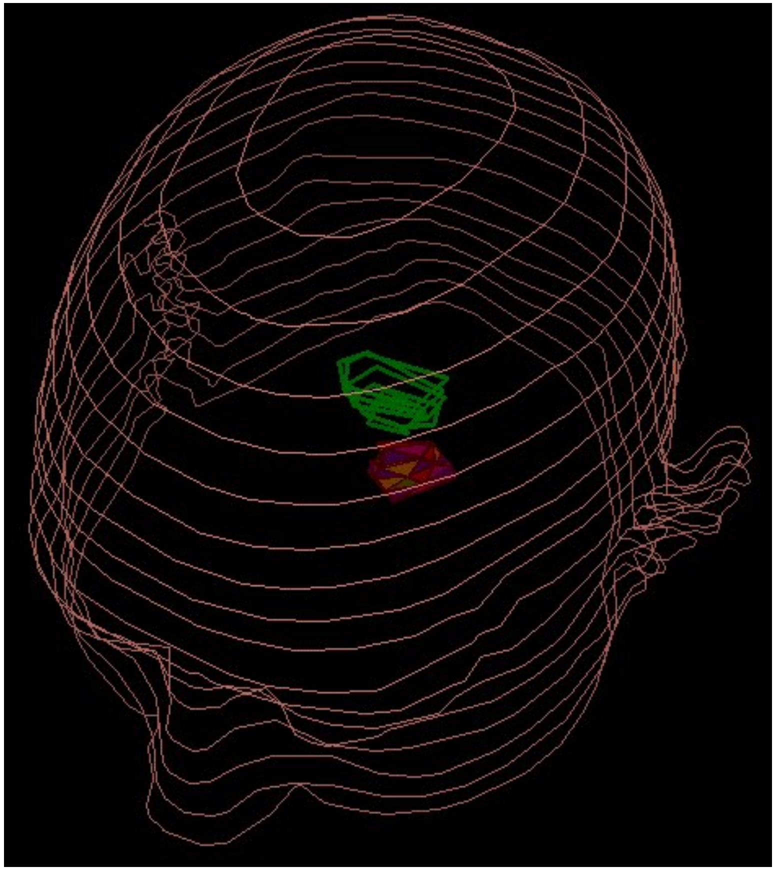
X-Ray
cameras



# Inputs

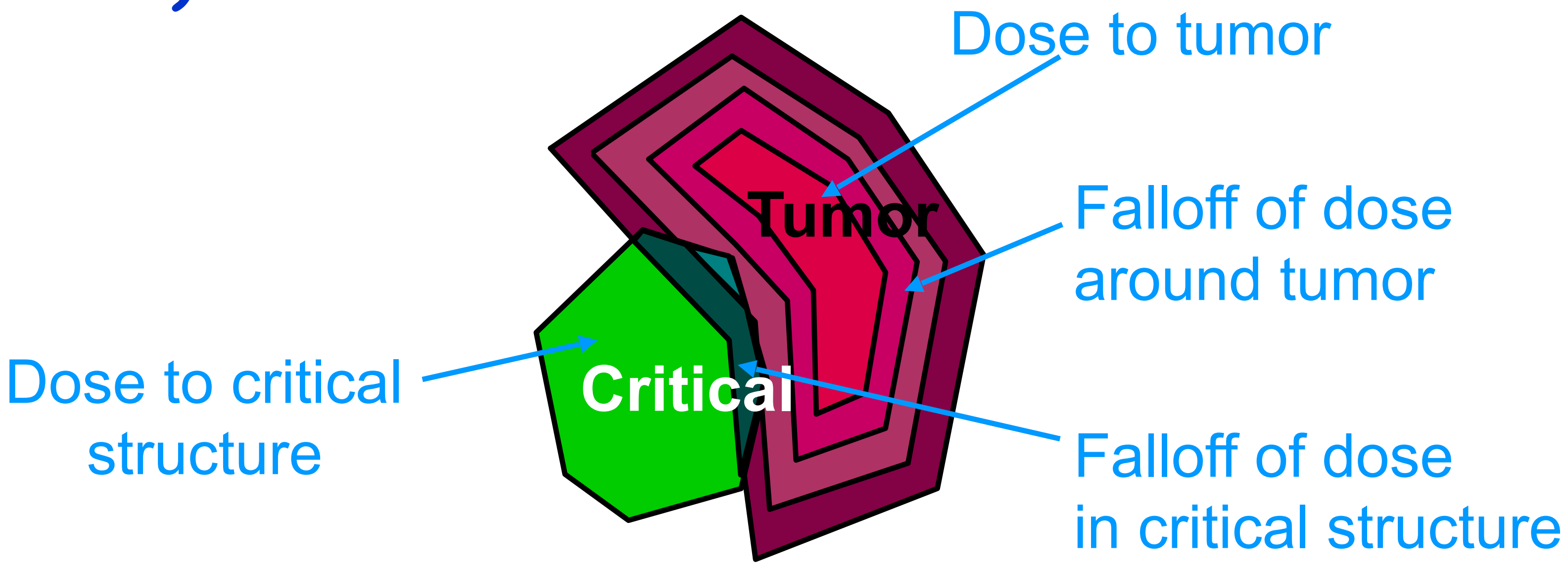
1) Regions of interest



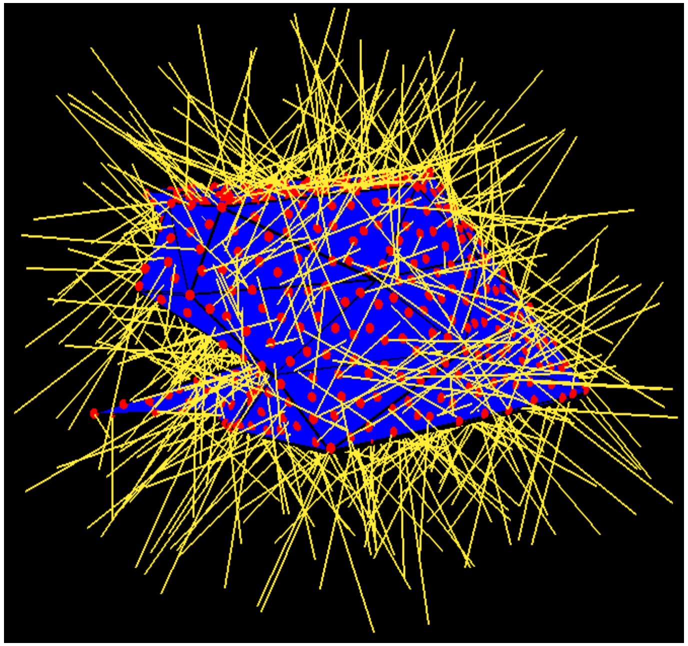



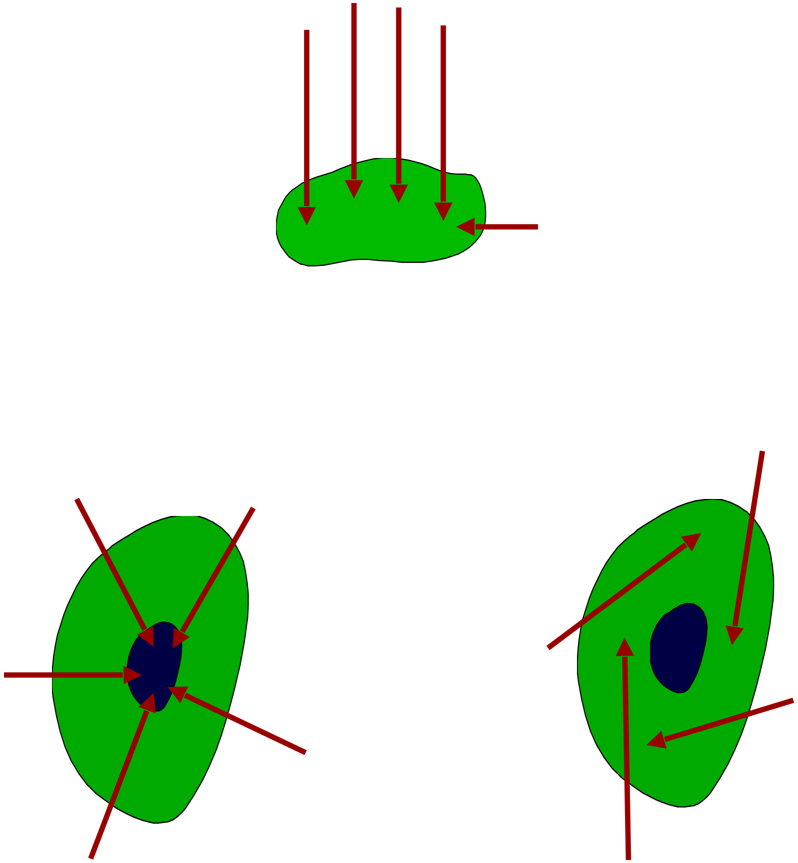
# Inputs

## 2) Dose constraints

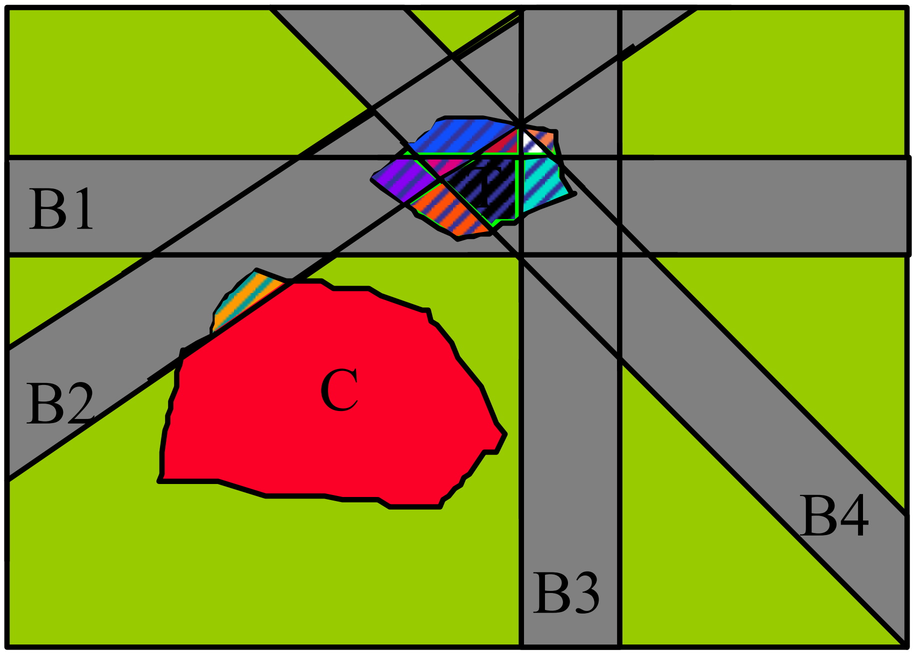


# Beam Sampling





# Constraints




- **2000 ≤ Tumor ≤ 2200**

  **2000 ≤ B2 + B4 ≤ 2200**
  **2000 ≤ B4 ≤ 2200**
  **2000 ≤ B3 + B4 ≤ 2200**
  **2000 ≤ B3 ≤ 2200**
  **2000 ≤ B1 + B3 + B4 ≤ 2200**
  **2000 ≤ B1 + B4 ≤ 2200**
  **2000 ≤ B1 + B2 + B4 ≤ 2200**
  **2000 ≤ B1 ≤ 2200**
  **2000 ≤ B1 + B2 ≤ 2200**

- **0 ≤ Critical ≤ 500**

  **0 ≤ B2 ≤ 500**

**2000 < Tumor < 2200**

**2000 < B2 + B4 < 2200**
**2000 < B4 < 2200**
**2000 < B3 + B4 < 2200**
**2000 < B3 < 2200**
**2000 < B1 + B3 + B4 < 2200**
**2000 < B1 + B4 < 2200**
**2000 < B1 + B2 + B4 < 2200**
**2000 < B1 < 2200**
**2000 < B1 + B2 < 2200**

⇨

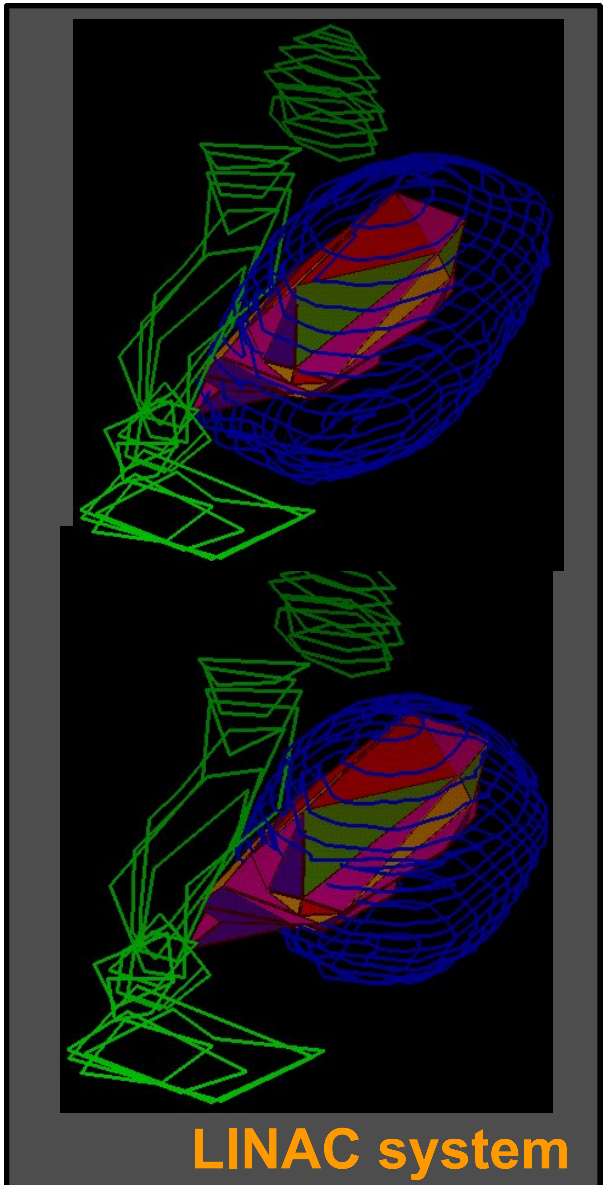
**2000 < Tumor < 2200**

**2000 < B4**

**2000 < B3**
**B1 + B3 + B4 < 2200**

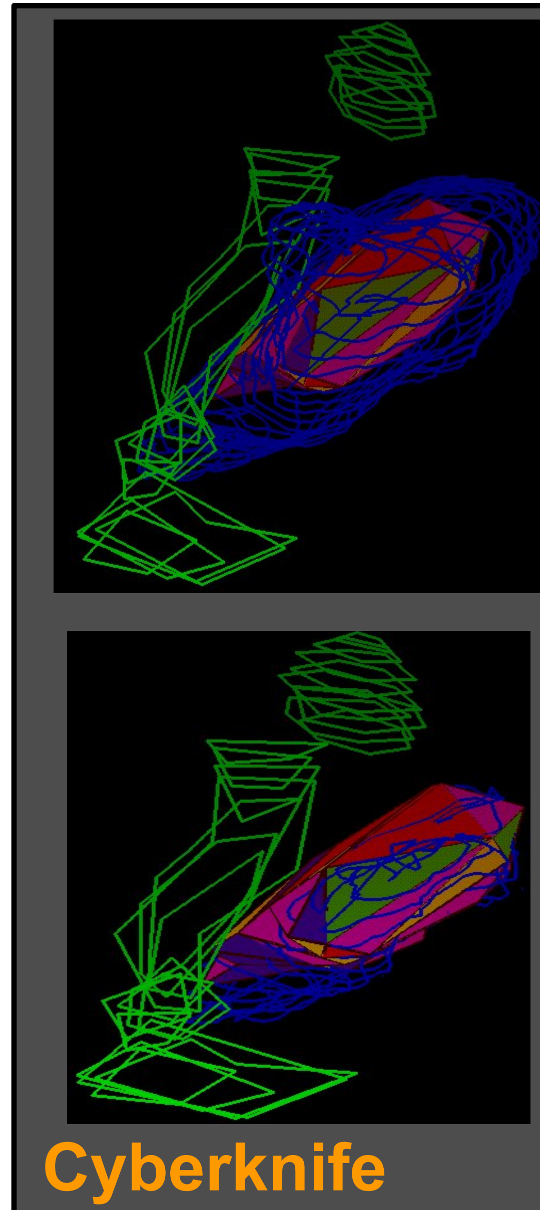
**B1 + B2 + B4 < 2200**
**2000 < B1**

# Case Results



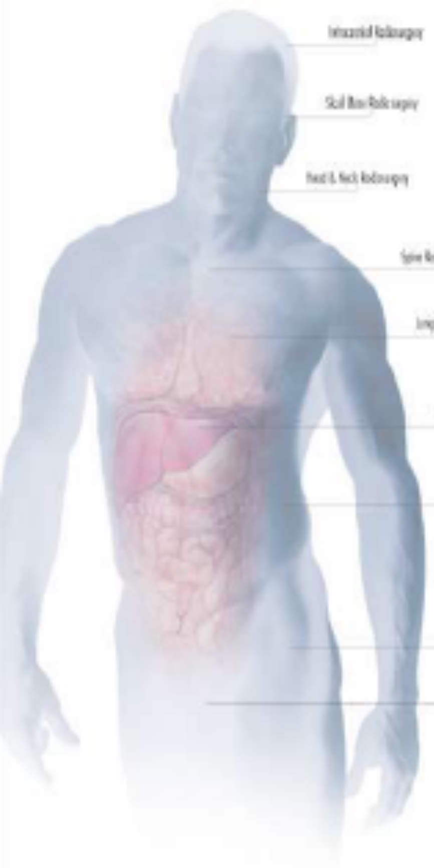

**50% Isodose Surface**

**80% Isodose Surface**



# THE POWER OF T4 TECHNOLOGY

CyberKnife® Tight-to-the-Tumor (T4) Radiosurgery with Ultimate Conformality



## FULL-BODY
**100% Frameless**
**T4 Radiosurgery**

## INTEGRATION OF TWO REVOLUTIONARY TECHNOLOGIES

**Proprietary Image-Guidance System**
Tracks and verifies tumor location to enable automatic compensation for tumor movement.

**Multi-Jointed Robotic Arm**
Enables access to previously unreachable tumors and reduces damage to surrounding critical structures.

Integration of these unique technologies allows physicians to treat complex-shaped tumors with clinically proven accuracy that has been demonstrated to be comparable, if not superior, to frame-based radiosurgical systems.[1]

### Simple Outpatient Treatment Process

**Planning:** CT scanning and advanced treatment planning are utilized.
**Positioning:** The patient lies on a table with only a face mask or body mold used for immobilization.
**Verification:** The image-guidance system verifies tumor location and compares it to previously stored data.
**Targeting:** When tumor movement is detected, the robotic arm is repositioned within a fraction of a second.
**Repeat:** This verification process is repeated prior to delivery of each radiation beam.
**Treatment:** Hundreds of finely collimated radiation beams deliver precision radiosurgery to the tumor.
**Completion:** Following CyberKnife® treatment, the patient goes home. There is zero recovery time.

**CyberKnife® T4 Radiosurgery**
A new standard in IMRT conformality