Lecture 8

Intro to CSP as Search

Lecture Overview

- Recap of previous lecture
- Intro to CSP
- CSP algorithms using Search
  - Generate and test
  - Graph search
- Intro to Arc Consistency (time permitting)

Recap (Must Know How to Fill This)

	Selection	Complete	Optimal	Time	Space
DFS	LIFO	N	N	$O(b^m)$	$\boldsymbol{O(mb)}$
BFS	FIFO	Y	Y	$O(b^m)$	$O(b^m)$
IDS	LIFO	Y	Y	$O(b^m)$	$\boldsymbol{O(mb)}$
LCFS	min cost	Y **	Y **	$O(b^m)$	$O(b^m)$
Best First	min h	N	N	$O(b^m)$	$O(b^m)$
A*	min f	Y**	Y**	$O(b^m)$	$O(b^m)$
B&B	LIFO + pruning	Y**	Y**	$O(b^m)$	$\boldsymbol{O(mb)}$
IDA*	LIFO	Y**	Y**	$O(b^m)$	$\boldsymbol{O(mb)}$

MBA*	min f	Y**	Y**	$O(b^m)$	$O(b^m)$

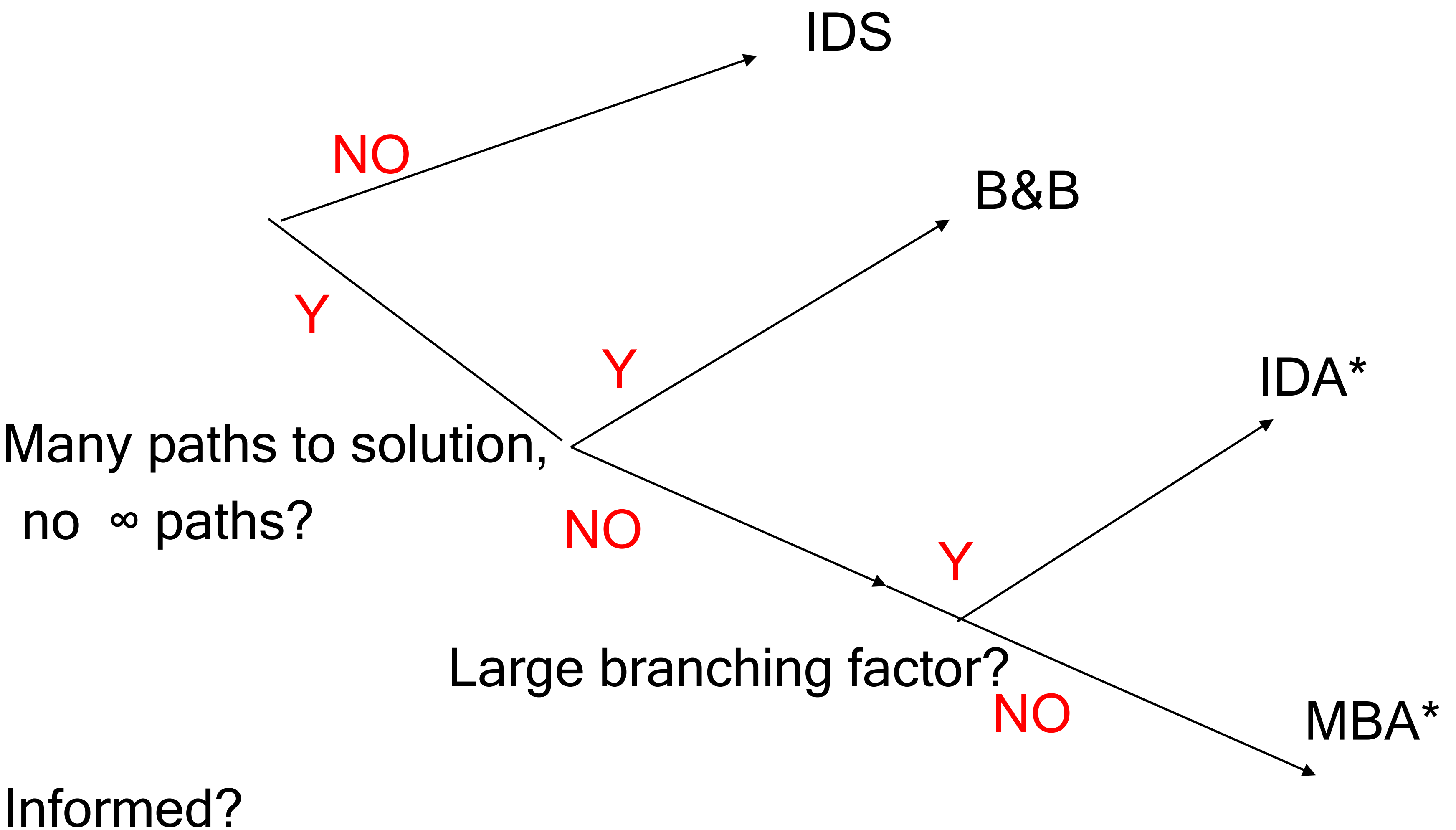
** Needs conditions: you need to know what they are[5]

Algorithms Often Used in Practice

	Selection	Complete	Optimal	Time	Space
DFS	LIFO	N	N	$O(b^m)$	$O(mb)$
BFS	FIFO	Y	Y	$O(b^m)$	$O(b^m)$
IDS	LIFO	Y	Y	$O(b^m)$	$O(mb)$
LCFS	min cost	Y **	Y **	$O(b^m)$	$O(b^m)$

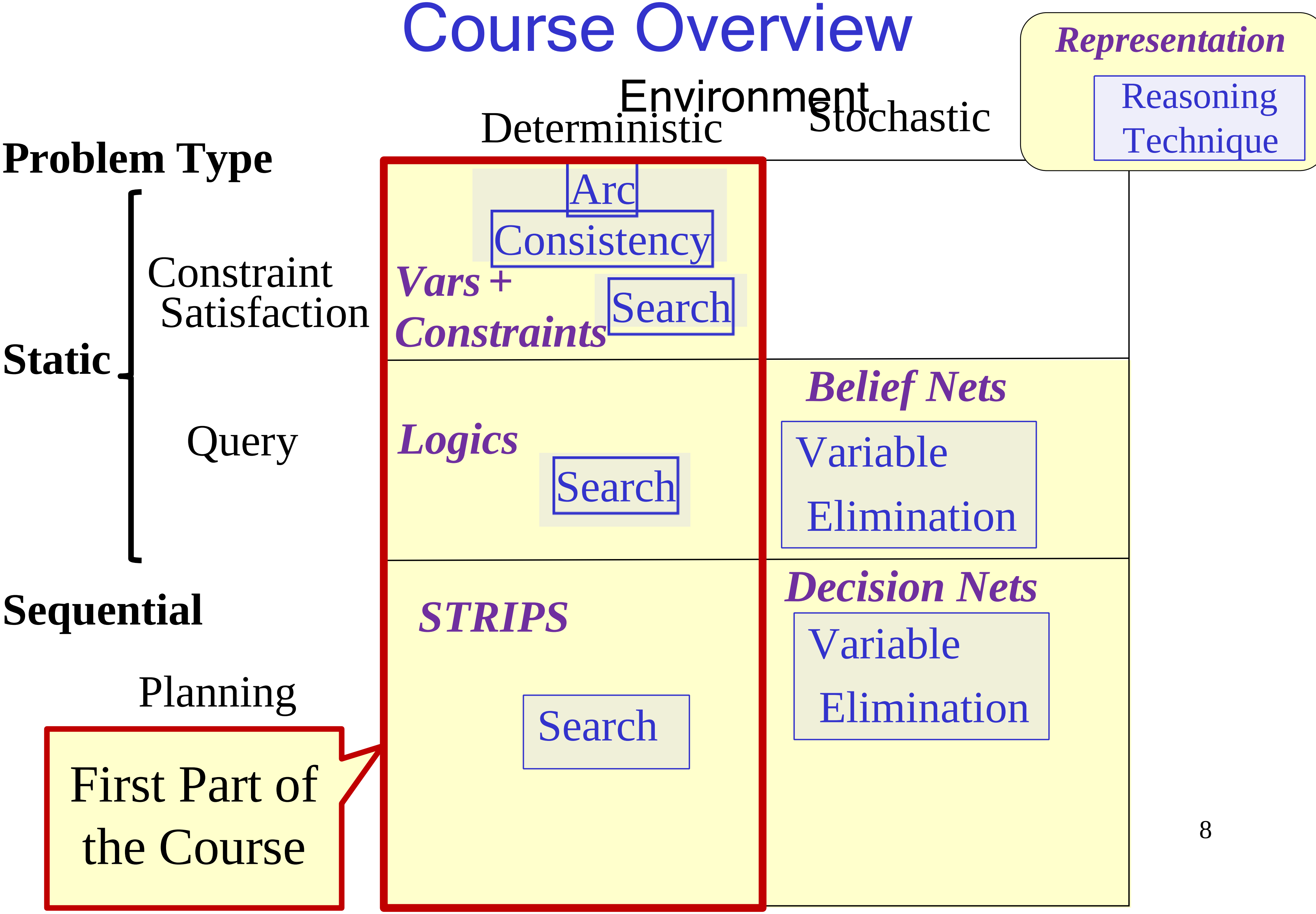
Best First	min h	N	N	$O(b^m)$	$O(b^m)$
A*	min f	Y**	Y**	$O(b^m)$	$O(b^m)$
B&B	LIFO + pruning	Y**	Y**	$O(b^m)$	$O(mb)$
IDA*	LIFO	Y	Y	$O(b^m)$	$O(mb)$
MBA*	min f	Y**	Y**	$O(b^m)$	$O(b^m)$

** Needs conditions: you need to know what they are[6]


IDS
NO
B&B
Y
Y
IDA*
Many paths to solution,
no ∞ paths?
NO
Y
Large branching factor?
NO
MBA*
Informed?

These are indeed general guidelines, specific problems might yield different choices

Course Overview



Standard vs Specialized Search

- We studied general state space search in isolation
- Standard search problem: search in a state space

- State is a "black box" - any arbitrary data structure that supports three problem-specific routines:
- goal test: goal(state)
- finding successor nodes: neighbors(state)
- if applicable, heuristic evaluation function: h(state)

Course Overview

Environment

- We will see more specialized versions of search for various problems

Markov Processes

Value Iteration

Representation

Reasoning Technique

Problem Type		Deterministic	Stochastic
Static	Constraint Satisfaction	*Vars + Constraints* Arc Consistency Search	
	Query	*Logics* Search	*Belief Nets* Variable Elimination
Sequential	Planning	*STRIPS* Search	*Decision Nets* Variable Elimination

Course Overview

Environment

We will look at Search in Specific R&R Systems

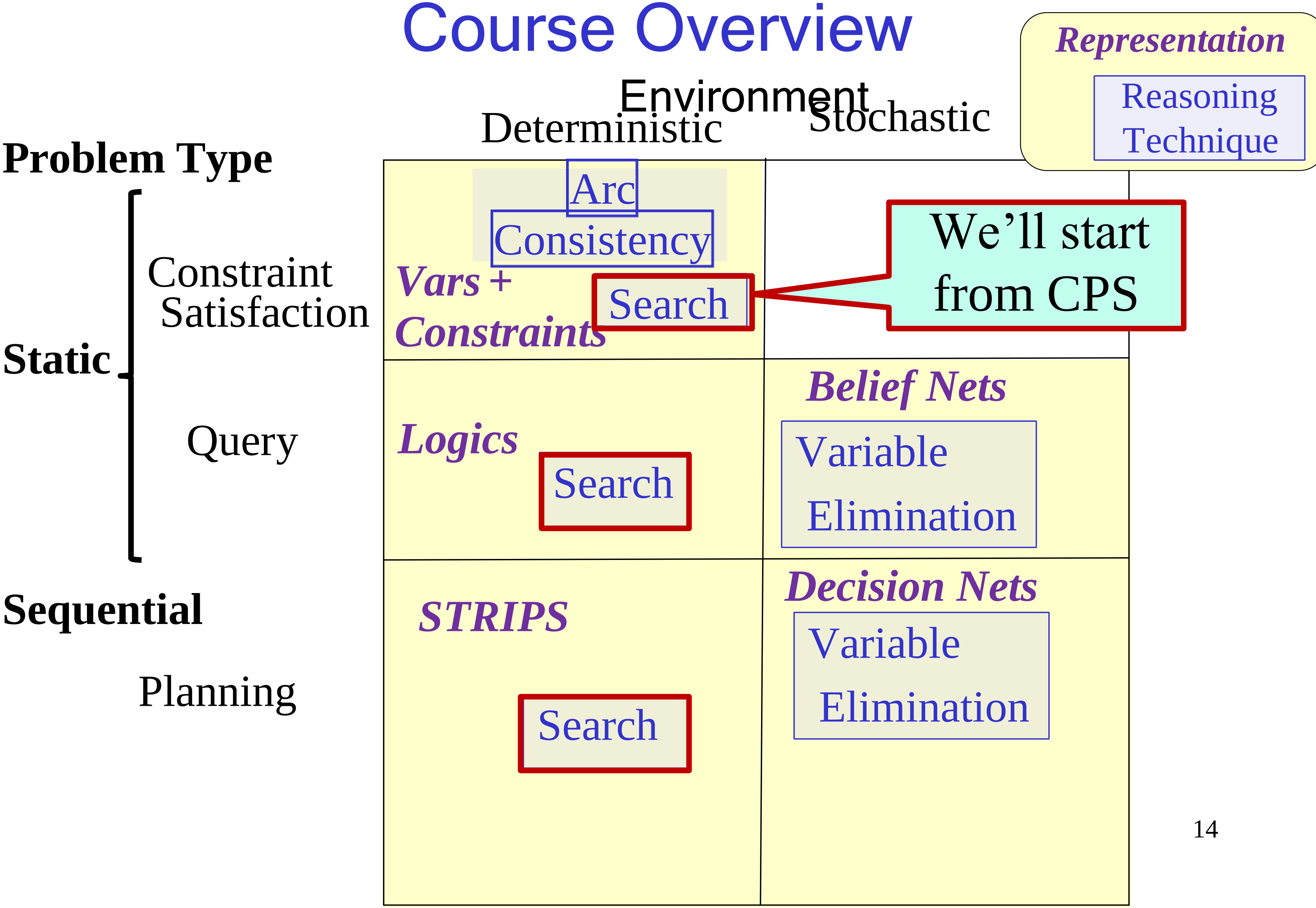
- Constraint Satisfaction Problems (CPS):
- State
- Successor function
- Goal test
- Solution
- Heuristic function • Query:
- State
- Successor function
- Goal test
- Solution
- Heuristic function

Markov Processes

Value Iteration

- Planning
- State
- Successor function
- Goal test
- Solution
- Heuristic function

Course Overview



- Constraint Satisfaction Problems (CPS):
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

We will look at Search for CSP

- Query:
- State
- Successor function
- Goal test
- Solution
- Heuristic function
- Planning

Course Overview

Environment

- State
- Successor function
- Goal test
- Solution
- Heuristic function

Markov Processes

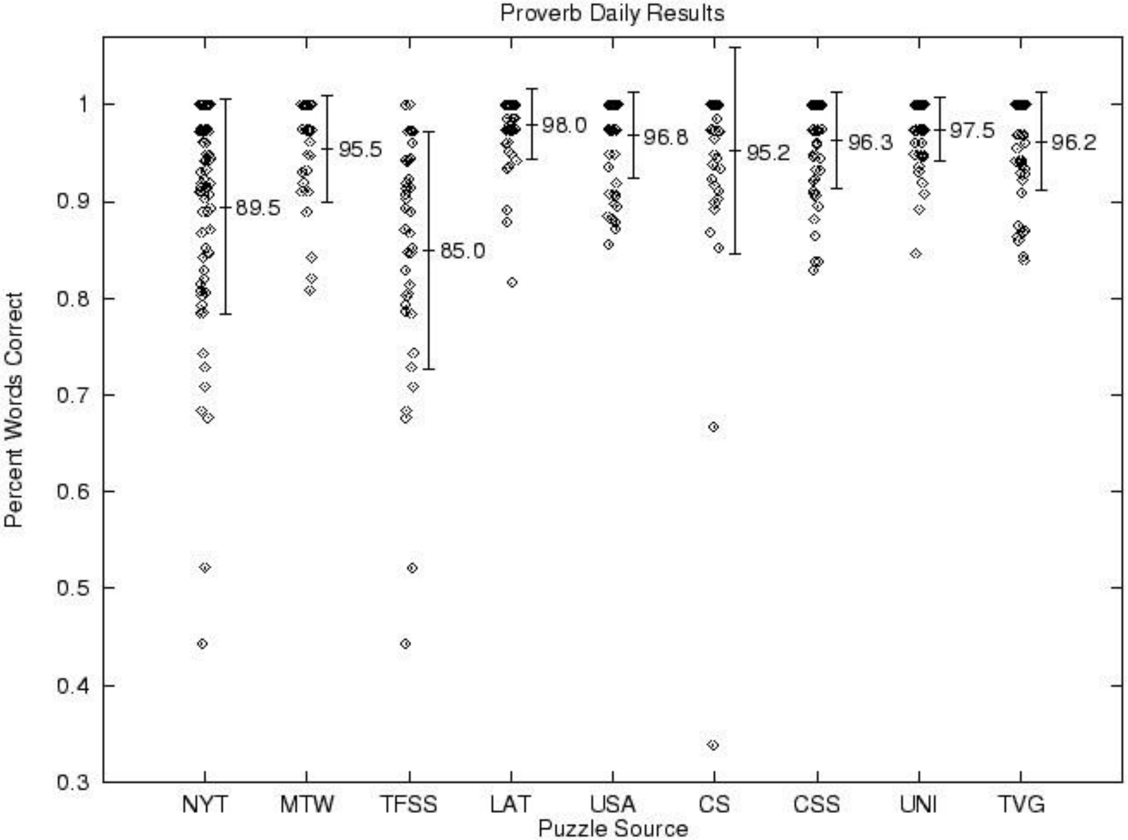
Value Iteration

Daily Puzzles

370 puzzles from 7 sources.

Summary statistics:

- 95.3% words correct (miss three or four words per puzzle)
- 98.1% letters correct
- 46.2% puzzles completely correct



Source: *Michael Littman*

Constraint Satisfaction Problems (CSP)

- In a CSP
  - state is defined by a set of variables V_i with values from domain D_i
  - goal test is a set of constraints specifying

1. allowable combinations of values for subsets of variables (hard constraints)
2. preferences over values of variables (soft constraints)

Dimensions of Representational Complexity (from lecture 2)

- Reasoning tasks (Constraint Satisfaction / Logic&Probabilistic Inference / Planning)
- Deterministic versus stochastic domains

Some other important dimensions of complexity:

•• Explicit state or features or relations Explicit state or features or relations

- Flat or hierarchical representation

- Knowledge given versus knowledge learned from experience
- Goals versus complex preferences
- Single-agent vs. multi-agent

Variables/Features and Possible Worlds

- Variable: a synonym for feature
- We denote variables using capital letters
- Each variable V has a domain dom(V) of possible values
- Variables can be of several main kinds:
  - ✓ Boolean: |dom(V)| = 2
  - ✓ Finite: |dom(V)| is finite
  - ✓ Infinite but discrete: the domain is countably infinite
  - ✓ Continuous: e.g., real numbers between 0 and 1
- Possible world:
- Complete assignment of values to each variable

- This is equivalent to a state as we have defined it so far
  - ✓ Soon, however, we will give a broader definition of state, so it is best to start distinguishing the two concepts.

17

Example (lecture 2)

Mars Explorer Example

Weather	{S, C}
Temperature	[-40, 40]
Longitude [0, 359]	
Latitude	[0, 179]

One possible world (state) $\boxed{\{S, -30, 320, 210\}}$

Number of possible (mutually exclusive) worlds (states)

$$\boxed{2 \times 81 \times 360 \times 180}$$

Product of cardinality of each domain

… always exponential in the number of variables

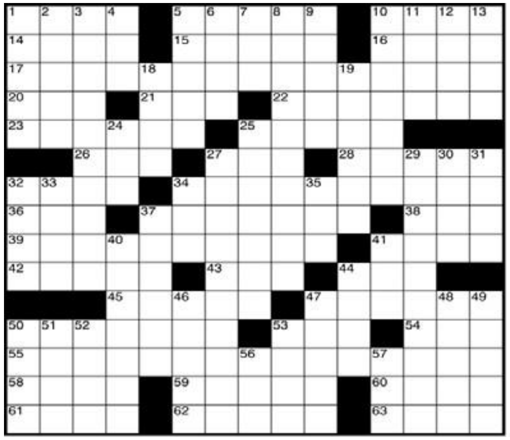
Lecture Overview

- Recap of previous lecture

- CSP: possible worlds, constraints and models
- CSP algorithms using Search
  - Generate and test
  - Graph search
- Intro to Arc Consistency (time permitting)

How many possible worlds?

- Crossword Puzzle 1:
- variables are words that have to be filled in
- domains are English words of correct length
- possible worlds: all ways of assigning words



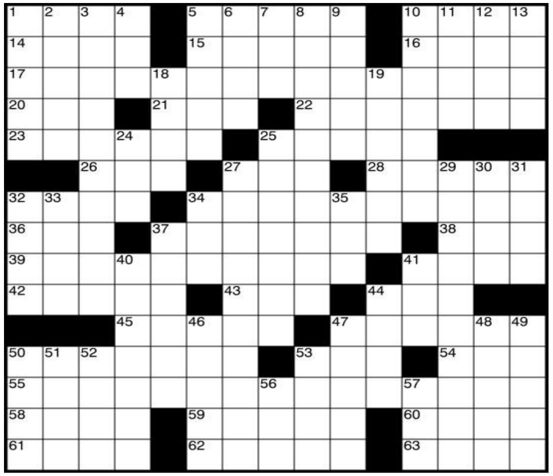
- Number of English words? Let's say 150,000
- Of the right length? Assume for simplicity: 15,000 for each length

How many possible worlds?

- Number of words to be filled in? 63
- How many possible worlds? (assume any combination is ok)

Crossword Puzzle 1:

- variables are words that have to be filled in
- domains are English words of correct length
- possible worlds: all ways of assigning words



- Number of English words? Let's say 150,000 • Of the right length? Assume for simplicity: 15,000 for each length
- Number of words to be filled in? 63

How many possible worlds?

- How many possible worlds? (assume any combination is ok)

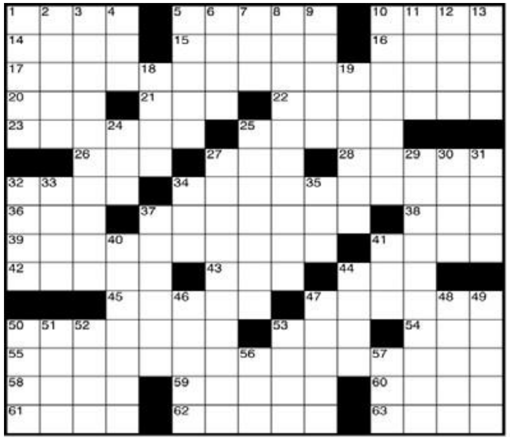
A. $15{,}000 * 63^{21}$

B. $15{,}000^{63}$

C $63^{15{,}000}$

D. $1{,}563^{63}$

- Crossword Puzzle:
- variables are words that have to be filled in
- domains are English words of correct length
- possible worlds: all ways of assigning words



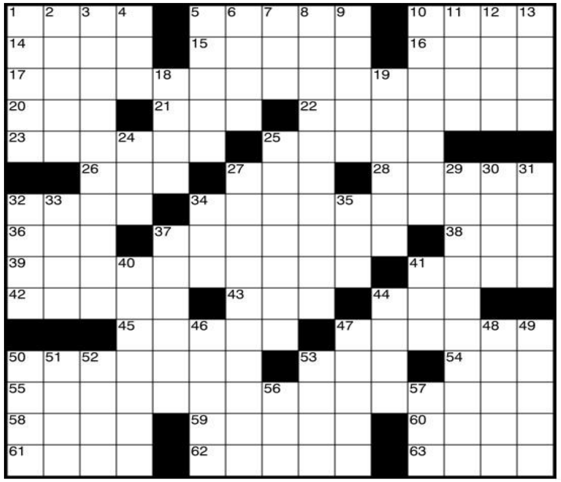
How many possible worlds?

- Number of English words? Let's say 150,000
- Of the right length? Assume for simplicity: 15,000 for each word
- Number of words to be filled in? 63
- How many possible worlds? (assume any combination is ok)

15000^{63}

How many possible worlds?

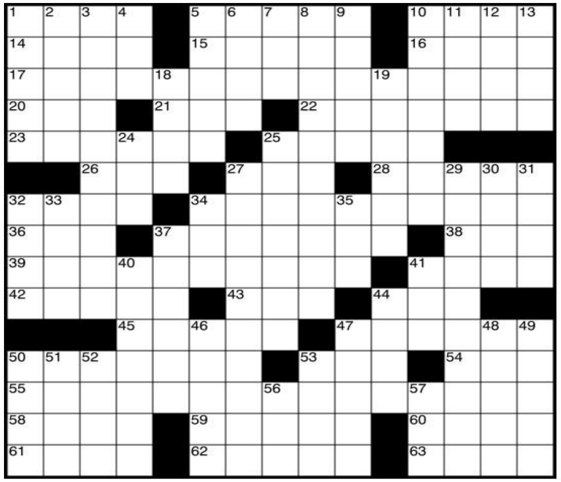
- Crossword 2:
- variables are cells (individual squares)
- domains are letters of the alphabet
- possible worlds: all ways of assigning letters to cells



- Number of empty cells? 15*15 – 32 = 193
- Number of letters in the alphabet? 26
- How many possible worlds? (assume any combination is ok)

How many possible worlds?

- Crossword 2:
- variables are cells (individual squares)
- domains are letters of the alphabet
- possible worlds: all ways of assigning letters to cells



- Number of empty cells? 15*15 – 32 = 193
- Number of letters in the alphabet? 26
- How many possible worlds? (assume any combination is ok)

26^{193}

- In general: (domain size)$^{\#\text{variables}}$

Constraint Satisfaction Problems (CSP)

- Allow for usage of useful general-purpose algorithms with more power than standard search algorithms

- They exploit the multi-dimensional nature of the problem and the structure provided by the goal

How many possible worlds?

set of constraints, *not* black box.

Constraints

- Constraints are restrictions on the values that one or more variables can take
- Unary constraint: restriction involving a single variable
- k-ary constraint: restriction involving k different variables
  - ✓We will mostly deal with binary constraints
- Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
  2. giving a function that returns true when given values for each variable which satisfy the constraint

Constraints: Simple Example

- Unary constraint: $V_2 \neq 2$ $V = \{V_1, V_2\}$

  - $dom(V_1) = \{1,2,3\}$;

- k-ary constraint: – $dom(V_2) = \{1,2\}$
- binary (k=2): $V_1 + V_2 < 5$
- 3-ary: $V_1 + V_2 + V_4 < 5$

  We will mostly deal with binary constraints

- Constraints can be specified by

1. listing all combinations of valid values for the variables participating in the constraint
   - for constraint $V_1 > V_2$ and dom(V_1) = {1,2,3} and dom(V_2) = {1,2}:

domain

V_1	V_2

2. giving a function (predicate) that returns true if given values for each variable satisfy the constraint else false:

Constraints: Simple Example

- Unary constraint: $V_2 \neq 2$ $V = \{V_1, V_2\}$
  - dom(V_1) = {1,2,3};
- k-ary constraint:
  - dom(V_2) = {1,2}

  ✓binary (k=2): $V_1 + V_2 < 5$

✓3-ary: $V_1 + V_2 + V_4 < 5$

We will mostly deal with binary constraints

- Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
     - for constraint $V_1 > V_2$ and $\mathrm{dom}(V_1) = \{1,2,3\}$ and $\mathrm{dom}(V_2) = \{1,2\}$:

V_1	V_2
2	1
3	1
3	2

  2. giving a function (predicate) that returns true if given values for each variable satisfy the constraint else false:

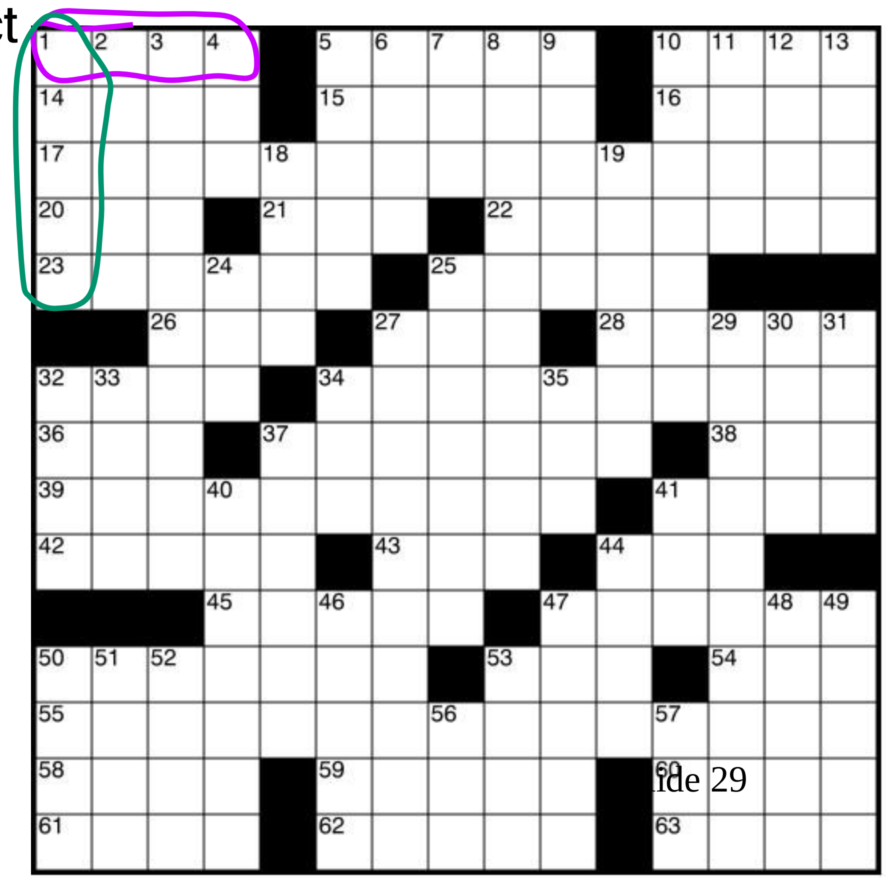
$$V_1 > V_2$$

Examples

- Crossword Puzzle 1:
- variables are words that have to be filled in

letters at points where they intersect

v_1[ ]



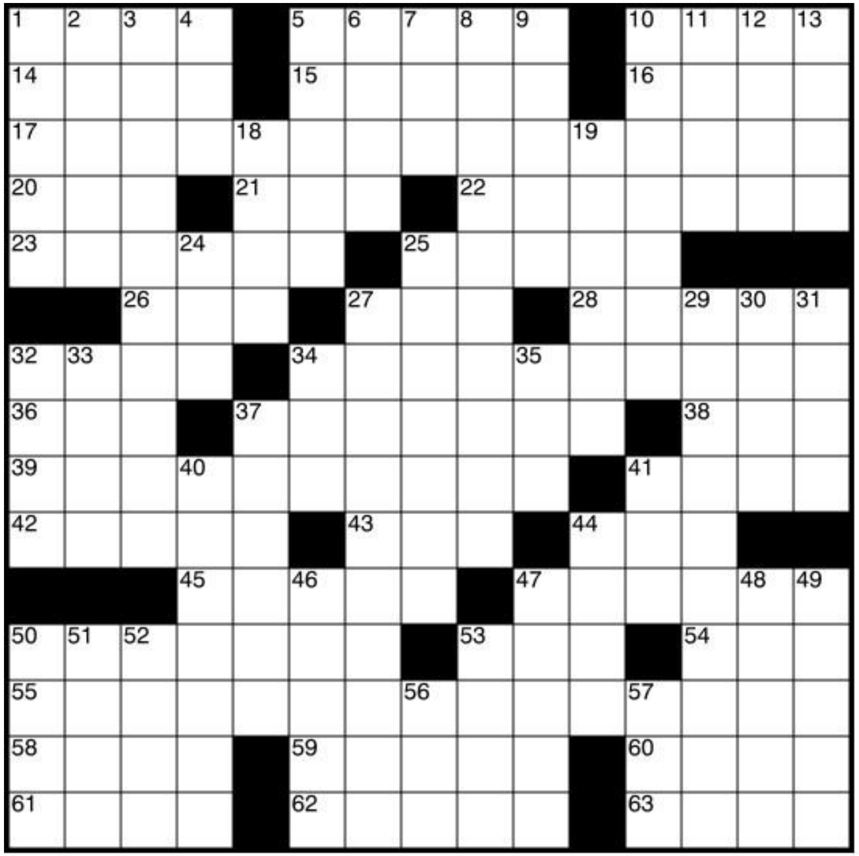
- domains are valid English words
- constraints:words have the same $h_1[]$

Examples

- Crossword Puzzle 1:
- variables are words that have to be filled in • domains are valid English words
- *constraints:* words have the same letters at points where they intersect

$h_1[]$

$v^1[\ ]$



$h_1[0] = v_1[0]$ $h_1[1] = v_2[0]$ …..

~225 constraints

Example: Map-Coloring

Variables WA, NT, Q, NSW, V, SA, T

Domains D_i= {red, green, blue}

Constraints: adjacent regions must have different colors e.g.,

Or

WA	NT

NT	SA

NT	QU

. . . .

Example: Map-Coloring

Variables WA, NT, Q, NSW, V, SA, T

Domains D_i= {red, green, blue}

Constraints: adjacent regions must have different colors

e.g., WA ≠ NT, NT ≠ SA, NT ≠ QU, ….,

Or

WA	NT
Red	Green
Red	Bue
Green	Red
Green	Blue
Blue	Red
Blue	Green

NT	SA
Red	Green
Red	Bue
Green	Red
Green	Blue
Blue	Red
Blue	Green

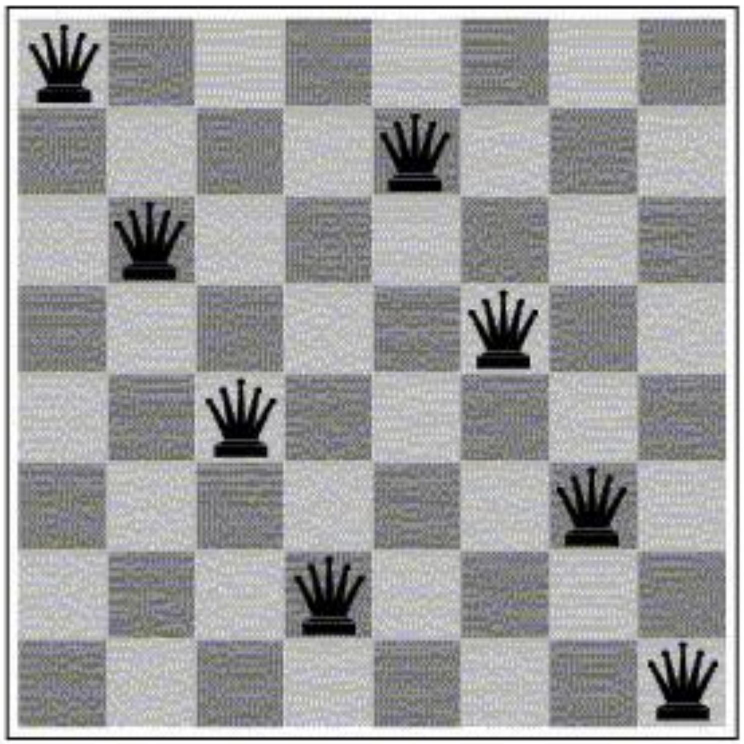
NT	QU
Red	Green
Red	Bue
Green	Red
Green	Blue
Blue	Red
Blue	Green

……………..

•

Example: Eight Queen problem

Eight Queen problem: place 8 queens on a chessboard so that no queen can attack the others



Example 2: 8 queens

- Variables: V_1,.. $V_{n.}$
- Constraints: No queens can be in the same row, column or diagonals

V_i = Row occupied by the I thqueen in the I thcolumn

- Domains: D_{Vi}= {1,2,3,4,5,6,7,8}
- Constraints: Two queens cannot be on the same row, column or on the same diagonal

Example 2: 8 queens

- Variables: $V_1, .. V_{n.}$

V_i = Row occupied by the i^{th} queen in the i^{th} column

- Domains: $D_{Vi} = \{1,2,3,4,5,6,7,8\}$

- Constraints: Two queens cannot be on the same row, column or on the same diagonal

- We can specify the constraints by enumerating explicitly, for each pair of columns, which positions are allowed. Ex:

  $Constr(V_1, V_2) =$

Example 2: 8 queens

- Variables: $V_1, .. V_n.$

V_i = Row occupied by the i^{th} queen in the i^{th} column

- Domains: $D_{Vi} = \{1,2,3,4,5,6,7,8\}$

- Constraints: : Two queens cannot be on the same row or on the same diagonal

- We can specify the constraints by enumerating explicitly, for each pair of columns, what positions are allowed. Ex:

  $Constr(V_1, V_2) = \{(1,3), (1,4),..(1,8)$

Example 2: 8 queens

- Variables: V_1,.. $V_{n.}$

(2,4)(2,5)…

(8,1)…. (8,5), (8,6)}

Constraints: one more concept

- Constraints are restrictions on the values that one or more variables can take
- Unary constraint: restriction involving a single variable
- k-ary constraint: restriction involving k different variables
  - ✓We will mostly deal with binary constraints
- Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
  2. giving a function that returns true when given values for each variable which satisfy the constraint

- A possible world satisfies a set of constraints
  - if the values for the variables involved in each constraint are consistent with that constraint, i.e.
    1. They are elements of the list of valid domain values
    2. Function for that constraint returns true for those values

Constraint Satisfaction Problems (CSPs): Definitions

Definition:
A constraint satisfaction problem (CSP) consists of:

- a set of variables V
- a domain dom(V) for each variable
- a set of constraints C

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e., a possible world) that satisfies all of its constraints.

Simple example:

- $V = \{V_1\}$ All models for this CSP:
  - $dom(V_1) = \{1,2,3,4\}$
- $C = \{C_1, C_2\}$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 > 1$

Constraint Satisfaction Problems (CSPs): Definitions

Definition:

A constraint satisfaction problem (CSP) consists of:

- a set of variables V
- a domain dom(V) for each variable
- a set of constraints C

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e., a possible world) that satisfies all of its constraints.

Simple example:

- $V = \{V_1\}$ All models for this CSP:
  - $dom(V_1) = \{1,2,3,4\}$ $V_1 = 3$
- $C = \{C_1, C_2\}$ $V_1 = 4$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 > 1$

Models and Possible Worlds

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e. a possible world) that satisfies all of its constraints.

Another example:

- $V = \{V_1, V_2\}$
- $dom(V_1) = \{1,2,3\}$
- $dom(V_2) = \{1,2\}$
- $C = \{C_1, C_2, C_3\}$

- C_1: $V_2 \neq 2$
- C_2: $V_1 + V_2 < 5$
- C_3: $V_1 > V_2$

How many models do we have?

A	None
B	1
C	2
D	3
E	4

Models and Possible Worlds

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e. a possible world) that satisfies all of its constraints.

$V = \{V_1, V_2\}$

- $dom(V_1) = \{1,2,3\}$
- $dom(V_2) = \{1,2\}$

• $C = \{C_1, C_2, C_3\}$

- C_1: $V_2 \neq 2$

Possible worlds for this CSP:

$\{V_1=1, V_2=1\}$

$\{V_1=1, V_2=2\}$

{V{$_1$V=2$_1$=2, V, V$_2$=1}$_2$=1} (a model)

$\{V_1=2, V_2=2\}$

- C_2: $V_1 + V_2 < 5$ $\{V_1=3\{V, V_1=3_2=1\}$' V $_2=1\}$ (a model)
- C_3: $V_1 > V_2$ $\{V_1=3, V_2=2\}$

end 42

Example: Map-Coloring

Definition:
A model of a CSP is an assignment of values to all of its variables (i.e. a possible world) that satisfies all of its constraints.

WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Scope of a constraint

Definition:

The scope of a constraint is the set of variables that are involved in the constraint

- Examples:
- $V_2 \neq 2$ has scope $\{V_2\}$
- $V_1 > V_2$ has scope $\{V_1,V_2\}$
- $V_1 + V_2 + V_4 < 5$ has scope $\{V_1,V_2,V_4\}$

- Number of variables in the scope of a k-ary constraint is k

Finite Constraint Satisfaction Problem: Definition

Definition:
A finite constraint satisfaction problem is a CSP with a finite set of variables and a finite domain for each variable.

- We will only study finite CSPs here
- but many of the techniques carry over to countably infinite and continuous domains.

Constraint Satisfaction Problems: Variants

- We may want to solve the following problems with a CSP:
- determine whether or not a model exists
- find a model
- find all of the models
- count the number of models
- find the best model, given some measure of model quality
  - ✓this is now an optimization problem

- determine whether some property of the variables holds in all models

Solving Constraint Satisfaction Problems

- Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is NPhard
- There is no known algorithm with worst case polynomial runtime.
- We can't hope to find an algorithm that is polynomial for all CSPs.

- However, we can try to:

- find efficient (polynomial) consistency algorithms that reduce the size of the search space
- identify special cases for which algorithms are efficient
- work on approximation algorithms that can find good solutions quickly, even though they may offer no theoretical guarantees • find algorithms that are fast on typical (not worst case) cases

Lecture Overview

- Recap of previous lecture

- CSP: possible worlds, constraints and models
- CSP algorithms using Search
  - Generate and test
  - Graph search
- Intro to Arc Consistency (time permitting)

Generate and Test (GT) Algorithms

- Systematically check all possible worlds - Possible worlds: cross product of domains $dom(V_1) \times dom(V_2) \times \cdots \times dom(V_n)$ • Generate and Test:
  - Generate possible worlds one at a time - Test constraints for each one.

Example: 3 variables A,B,C

Generate and Test (GT) Algorithms

```
For a in dom(A)
    For b in dom(B)
        For c in dom(C)
            if {A=a, B=b, C=c} satisfies all constraints
                return {A=a, B=b, C=c}
fail
```


For a in dom(A)
For b in dom(B)
For c in dom(C)
if {A=a, B=b, C=c} satisfies all constraints
return {A=a, B=b, C=c}
fail

Generate and Test (GT) Algorithms

- If there are k variables, each with domain size d, and there are c constraints, the complexity of Generate &

A. $O(ckd)$

Test

C. $O(cd^k)$

B. $O(ck^d)$

D. $O(d^{ck})$

- If there are k variables, each with domain size d, and there are c constraints, the complexity of Generate & Test is

$O(cd^k)$

Generate and Test (GT) Algorithms

- There are d^k possible worlds
- For each one need to check c constraints

- Need to find a better way, that exploits the "insights" that we have on the goal expressed in terms of constraints
- Why does GT fail to do this?

Generate and Test (GT) Algorithms

- Need to find a better way, that exploits the “insights” that we have on the goal expressed in terms of constraints
- Why does GT fail to do this?

- It checks contraints only after a full assignment of values to variables has been made
- Fails to leverage the modular/explicit nature of the goal

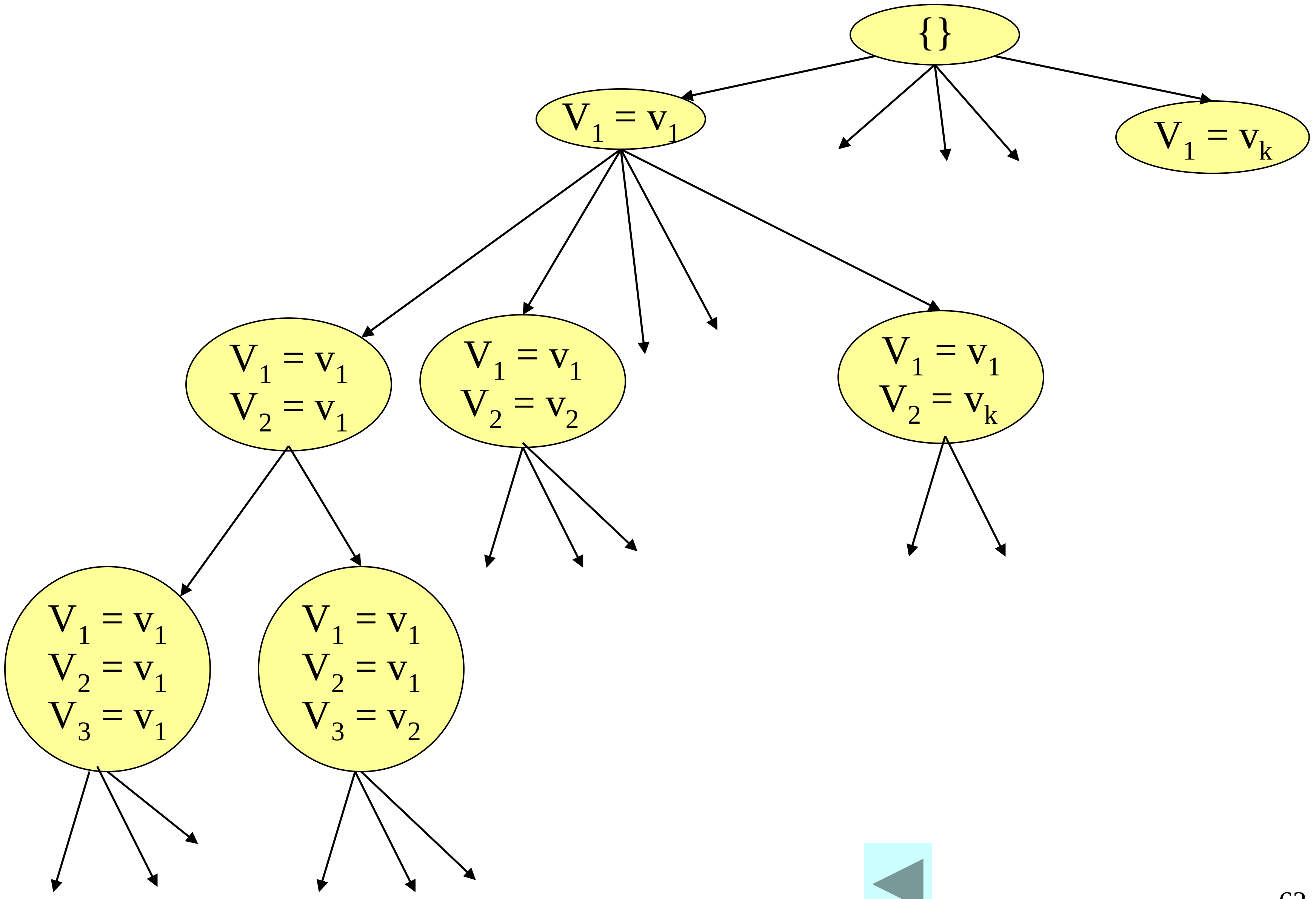
Generate and Test (GT) Algorithms

CSP as a Search Problem: one formulation

- States: partial assignments of values to variables
- Start state: empty assignment
- Successor function: states with the next variable assigned
- Follow a total order of the variables $V_1, \ldots, V_n$ • A state assigns values to the first k variables:
  - ✓$\{V_1 = v_1, \ldots, V_k = v_k\}$
- Neighbors of node $\{V_1 = v_1, \ldots, V_k = v_k\}$: nodes $\{V_1 = v_1, \ldots, V_k = v_k, V_{k+1} = x\}$ for each $x \in dom(V_{k+1})$

- Goal states: complete assignments of values to variables that satisfy all constraints
- That is, models
- Solution: assignment (the path does not matter)

CSP as a Search Problem: one formulation


{}
V1 = v1
V1 = vk
V1 = v1
V2 = v1
V1 = v1
V2 = v2
V1 = v1
V2 = vk
V1 = v1
V2 = v1
V3 = v1
V1 = v1
V2 = v1
V3 = v2

Which search algorithm would be most appropriate for this formulation of CSP?

A. Depth First Search

B. BFS

C. A *

D. IDS

Which search algorithm would be most appropriate for this formulation of CSP?

A. Depth First Search

- the search tree is always finite and has no cycles
- If there are n variables every solution is at depth n.
- Possibly very large branching factor b

Dealing with complexity

- CSP problems can be huge - Thousands of variables
- Exponentially more search states

- Exhaustive search is typically infeasible

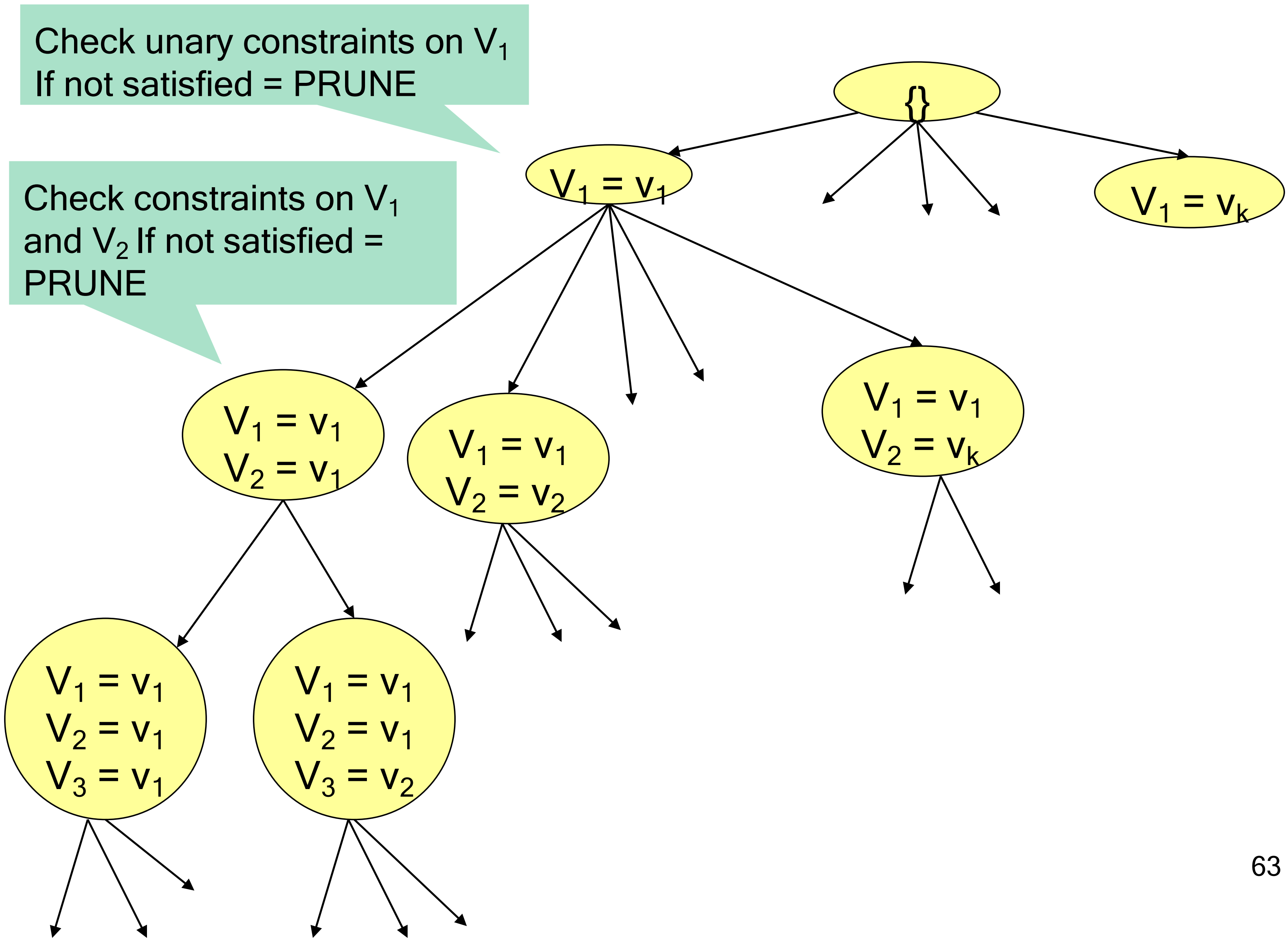
- Many algorithms exploit the structure provided by the goal ⇒ set of constraints, *not* black box

Backtracking algorithms

- Explore search space via DFS but evaluate each constraint as soon as all its variables are bound.
- Any partial assignment that does not satisfy the constraint can be pruned.
- Example:
  - 3 variables A, B, C each with domain {1,2,3,4}

- $\{A = 1, B = 1\}$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables $\Rightarrow$ Fail! Prune!

CSP as Graph Searching

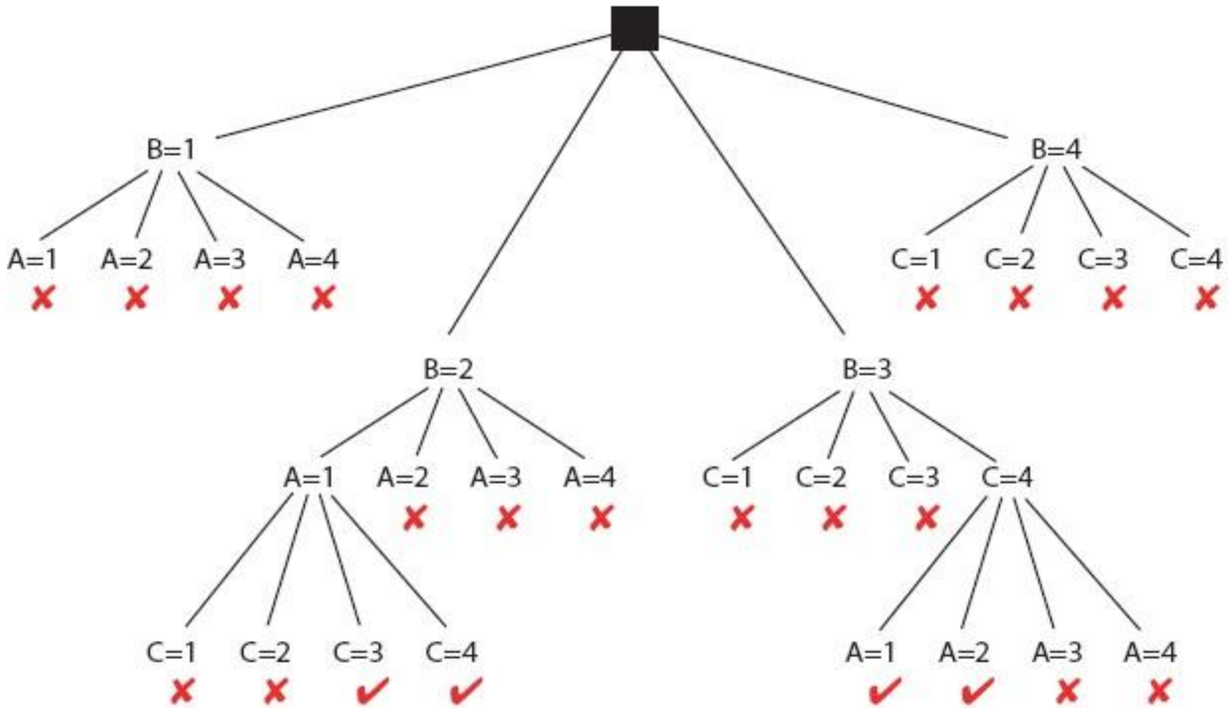


Check unary constraints on V_1
If not satisfied = PRUNE
Check constraints on V_1 and V_2 If not satisfied = PRUNE
{}
$V_1 = v_1$
$V_1 = v_k$
$V_1 = v_1$ $V_2 = v_1$
$V_1 = v_1$ $V_2 = v_2$
$V_1 = v_1$ $V_2 = v_k$
$V_1 = v_1$ $V_2 = v_1$ $V_3 = v_1$
$V_1 = v_1$ $V_2 = v_1$ $V_3 = v_2$

Solving CSPs by DFS: Example

- Variables: A,B,C
- Domains: {1, 2, 3, 4}
- Constraints: A < B, B < C

Good ordering, lots of pruning happens right away



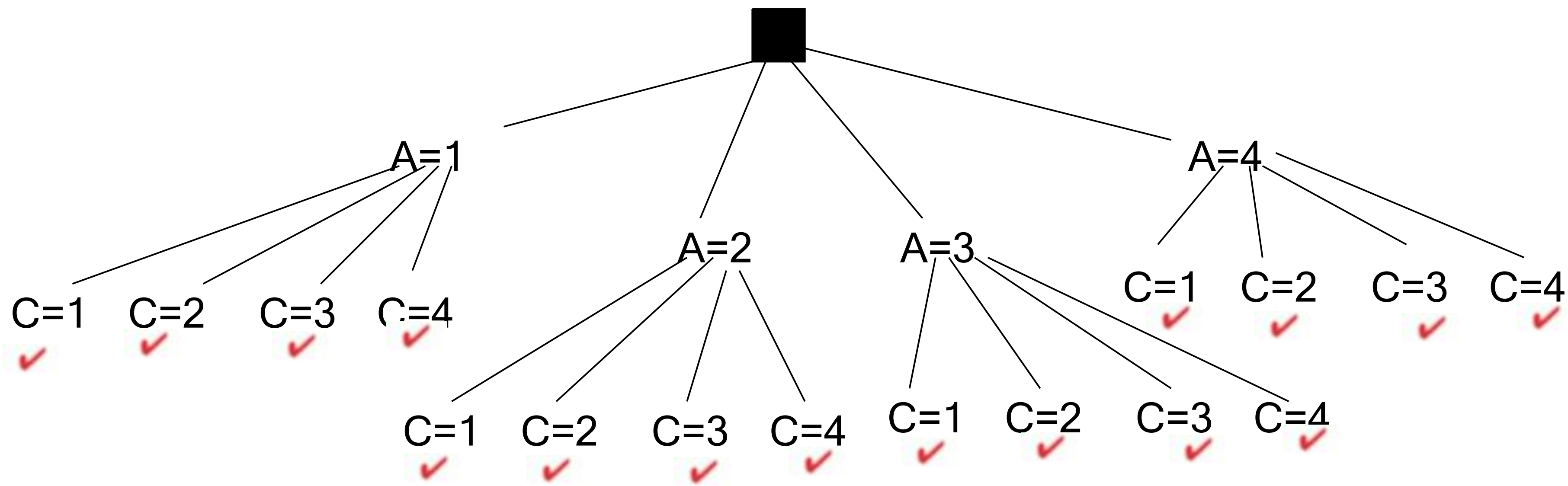
B=1
A=1
A=2
A=3
A=4
B=2
A=1
A=2
A=3
A=4
C=1
C=2
C=3
C=4
B=3
C=1
C=2
C=3
C=4
A=1
A=2
A=3
A=4
B=4
C=1
C=2
C=3
C=4

Solving CSPs by DFS: Example Efficiency

- Variables: A,B,C
- Domains: {1, 2, 3, 4}
- Constraints: A < B, B < C

Much worse ordering, keeps more branches around

Problem? Performance heavily depends on the order in which variables are considered



EXAMPLE : $D_A = D_B = D_C = \{1,2,3,4\}$ $(A \neq B) \wedge (B \neq C) \wedge (C < D) \wedge (A=D) \wedge (B \neq D) \wedge (E < A) \wedge (E < B) \wedge (E < C) \wedge (E < D)$

A=1 B=1 failure

B=2 C=1 D=1 failure

D=2 failure

D=3 failure

D=4 failure

C=2 failure

C=3 D=1 failure

D=2 failure

D=3 failure

D=4 failure

C=4 D=1 failure

D=2 failure

D=3 failure

D=4 failure

⋮

EXAMPLE : $D_A = D_B = D_C = \{1,2,3,4\}$ (A ≠ D) ∧ (E < A) ∧ (E < B) ∧ (E < C) ∧ (E < D) ∧ (A ≠B) ∧ (B ≠C) ∧ (C < D) ∧ (A=D) ∧ (B

A=1 D=1 C= 1

failure

C=2

failure

C=3 failure

C=4 failure

D=2 failure

D=3 failure

D=4 failure

D=2 failure

D=3 failure

D=4 failure

D=2 failure

D=3 failure

D=4 failure

D=3 failure

D=4 failure

A=1 B=1 failure

B=2 C=1 D=1 failure

C=4 D=1 failure

D=2 failure

C=2 failure

C=3 D=1 failure

Selecting variables in a smart way

- Backtracking relies on one or more heuristics to select which variables to consider next.
  - E.g, variable involved in the largest number of constraints:

- Can also be smart about which values to consider first

- But we will look at an alternative approach that can do much better

Meaning of ‘heuristic’ in CSP as search

- This is a different use of the word ‘heuristic ‘from the definition we have given in the search module

- Qualifications of ‘heuristic’ still true in the context of CSP as search

  - Can be computed cheaply during the search
  - Provides guidance to the search algorithm

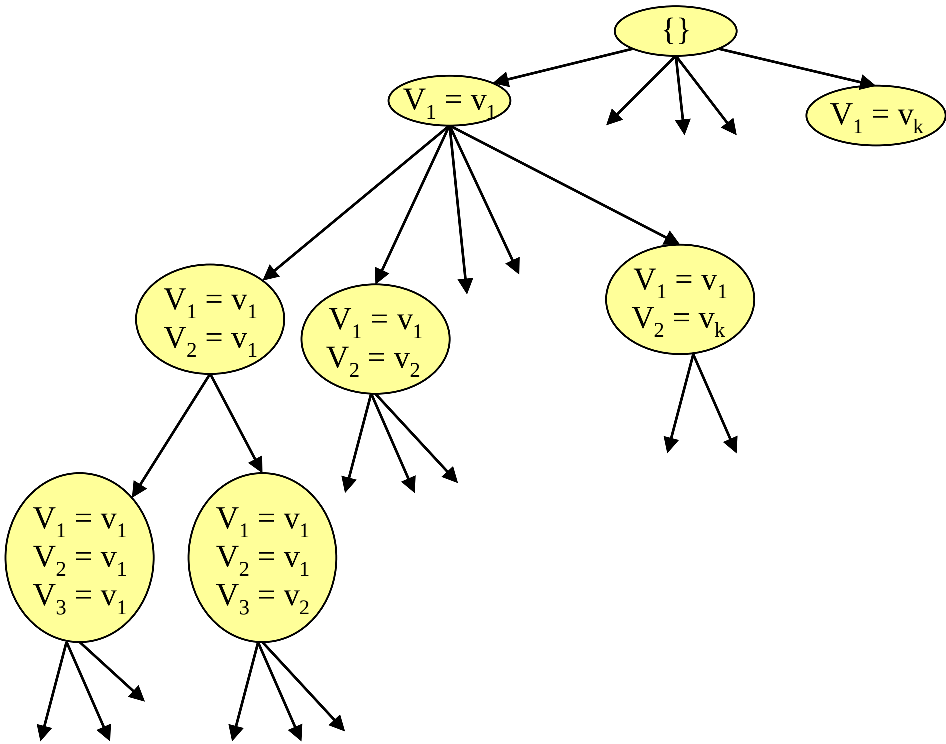
- Qualification that is not true anymore in this context

- ‘Estimate of the distance to the goal’

- Both meanings are used frequently in the AI literature.
- In general
- ‘heuristic ‘means ‘serves to discover’: goal-oriented.
- Does not mean ‘unreliable’!

Standard Search vs. Specific R&R systems

- Constraint Satisfaction (Problems):
- State: assignments of values to a subset of the variables
- Successor function: assign values to a "free" variable
- Goal test: all variables assigned a value and all constraints satisfied?
- Solution: possible world that satisfies the constraints
- Heuristic function: none (all solutions at the same distance from start)
- Planning :
- State
- Successor function
- Goal test

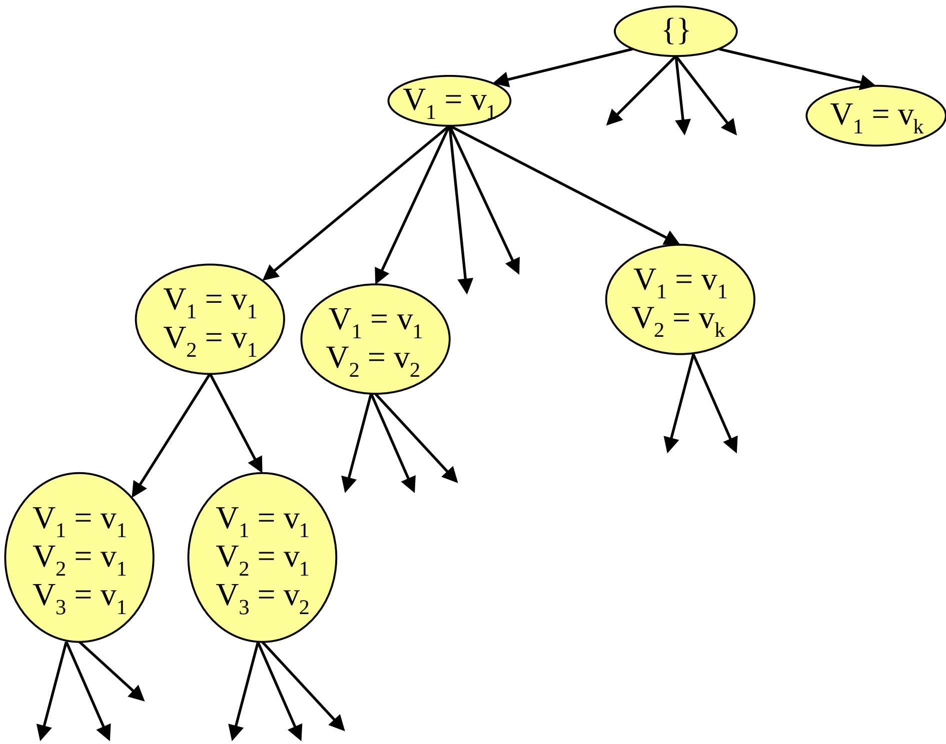


- Solution
- Heuristic function
- Inference
- State
- Successor function
- Goal test
- Solution
- Heuristic function

Standard Search vs. Specific R&R systems

- Constraint Satisfaction (Problems):
- State: assignments of values to a subset of the variables
- Successor function: assign values to a “free” variable
- Goal test: all variables assigned a value and all constraints satisfied?
- Solution: possible world that satisfies the constraints

- Heuristic function: none (all solutions at the same distance from start)
- Planning:
- State
- Successor function
- Goal test
- Solution
- Heuristic function
- Inference
- State
- Successor function
- Goal test
- Solution
- Heuristic function



Learning Goals for CSP

- Define possible worlds in term of variables and their domains
- Compute number of possible worlds on real examples
- Specify constraints to represent real world problems differentiating between:
- Unary and k-ary constraints
- List vs. function format
- Verify whether a possible world satisfies a set of constraints (i.e., whether it is a model, a solution)
- Implement the Generate-and-Test Algorithm. Explain its disadvantages.

- Solve a CSP by search (specify neighbors, states, start state, goal state).
- Compare strategies for CSP search.
- Implement pruning for DFS search in a CSP.