Introduction to

Artificial Intelligence (AI)

Computer Science cpsc502, Lecture 3

Sep, 15, 2011

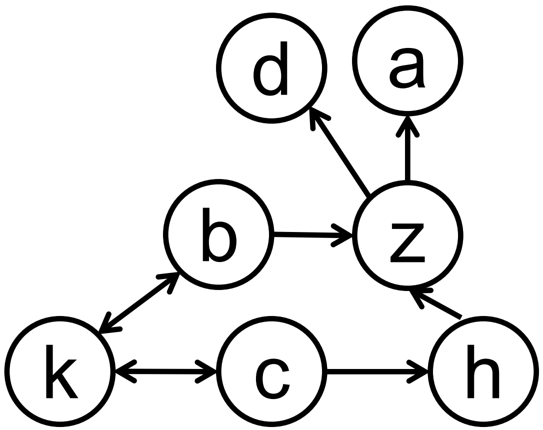
Today Sept 15

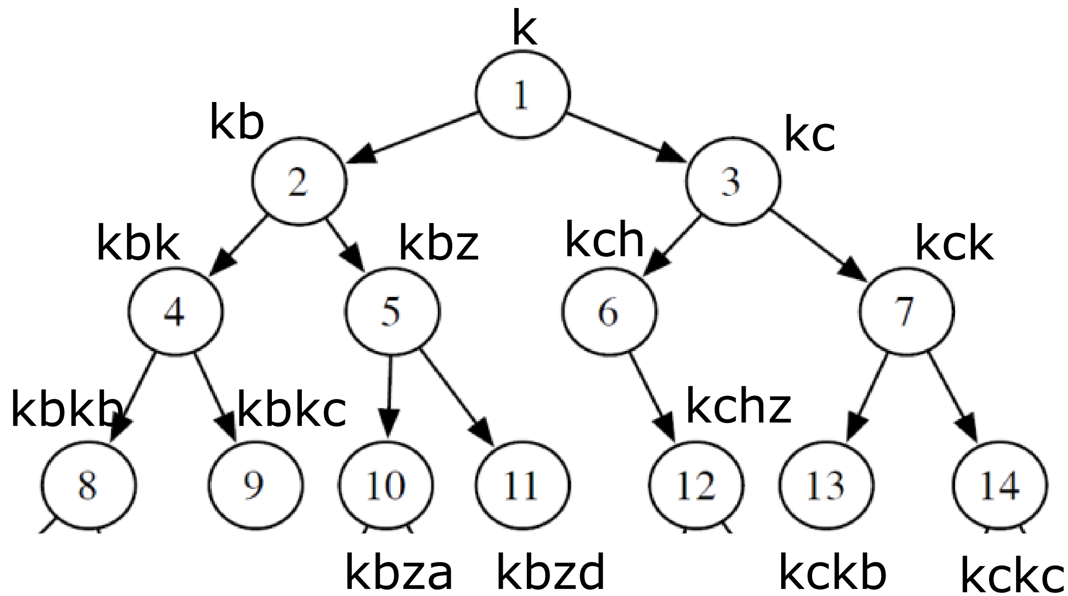
- Finish Search
- Constraint Satisfaction Problems
- ….
- ….

Office Hours

- My office hours – Thurs 11-12
- Shafiq's office hours

Clarification: state space graph vs search tree





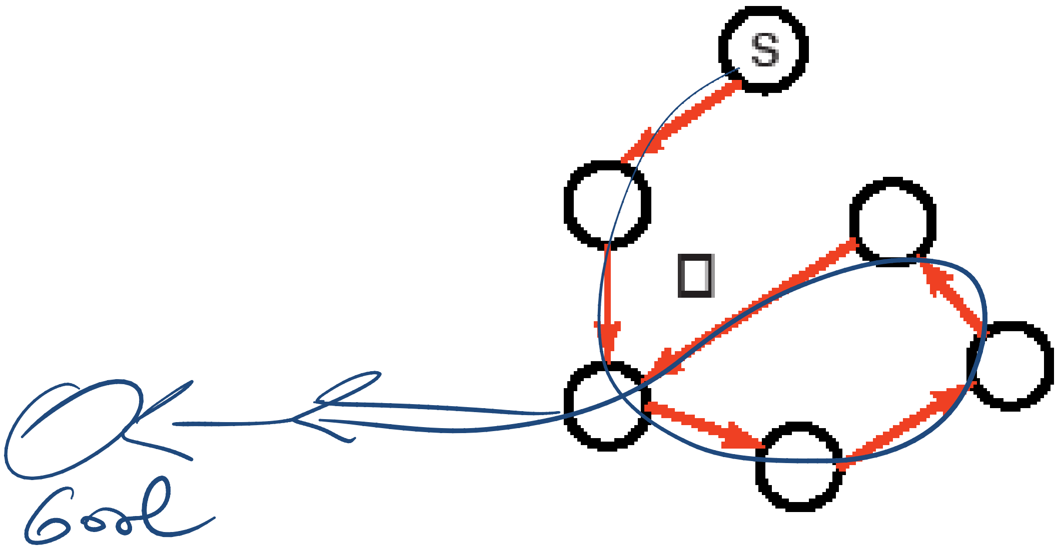
State space graph.

(For most problems, we are not explicitly given the whole graph)

Search tree.

Nodes in this tree correspond to paths in the state space graph

Cycle Checking



You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

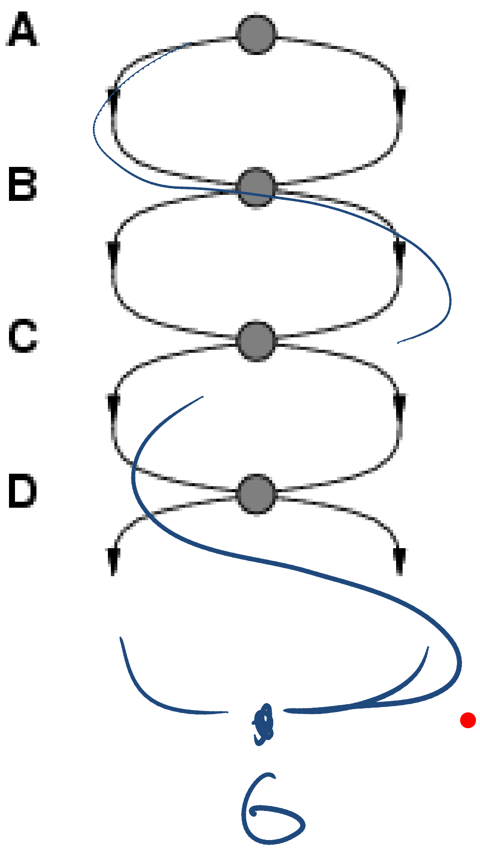
- The time is linear in path length.

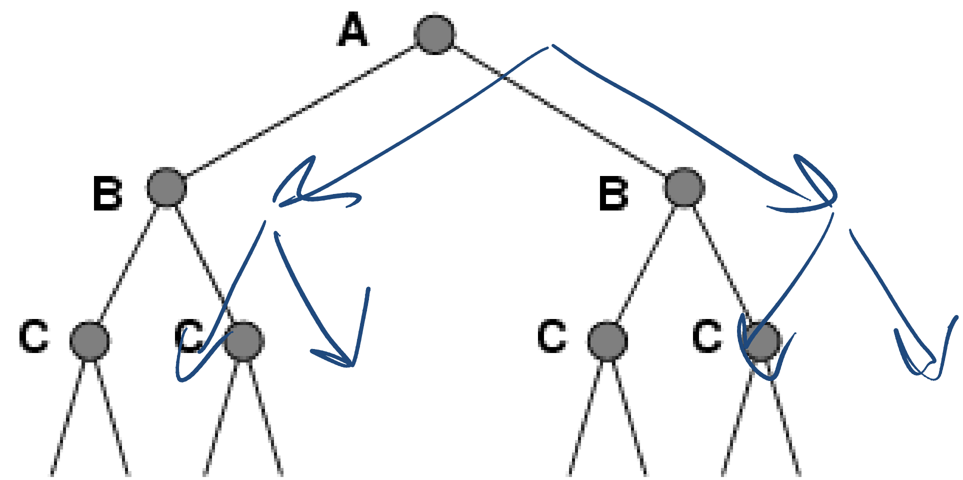
$\langle n_0, n_1, n_2 \ldots n_k \rangle$ n_2 n_i n_j

Repeated States / Multiple Paths

Failure to detect repeated states can turn a linear problem into an exponential one!

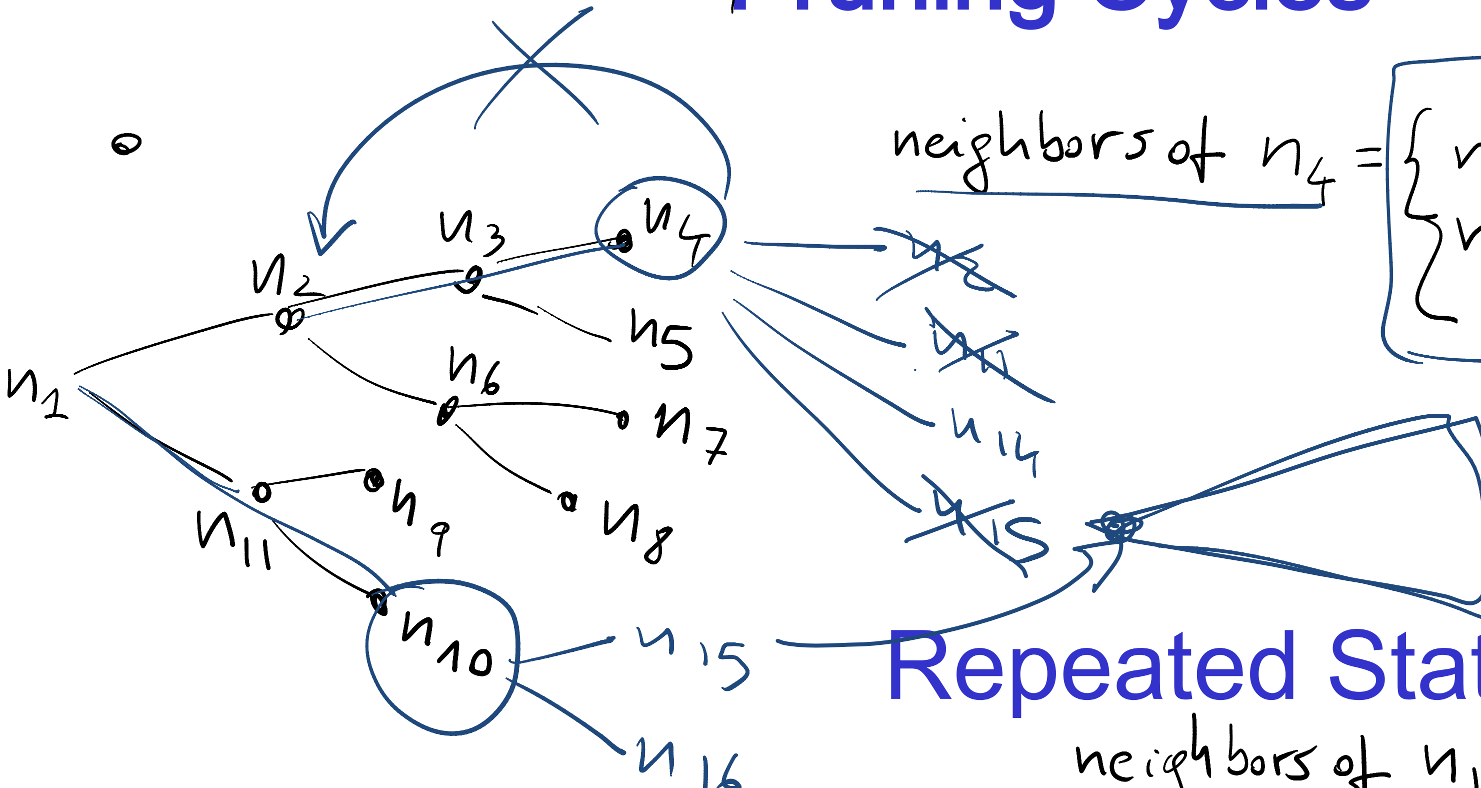
E.g. state space with 2 actions from each state to next





- **2^d possible paths through the state graph => exponentially larger search tree!**

Pruning Cycles



neighbors of n_4 = $\boxed{\{n_2, n_{11}, n_{14}, n_{15}\}}$

Repeated States

neighbors of n_{10} = $\{n_{15}, n_{16}\}$

R&Rsys we'll cover in this course

Environment

Problem		Deterministic	Stochastic
Static	Constraint Satisfaction	*Vars + Constraints* Arc Consistency Search	aka Bayesian Networks
	Query	*Logics* → Propositional, → First Order, → ...... Search	*Belief Nets* Var. Elimination Approx. Inference Temporal. Inference
Sequential	Planning	*STRIPS* actions precs effects Search	aka influence diagrams *Decision Nets* Var. Elimination *Markov Processes* Value Iteration

Representation

Reasoning Technique

Standard Search vs. Specific R&R systems

Constraint Satisfaction (Problems):

- State (and start state)
- Successor function
- Goal test
- Solution

two next lectures

Planning :

- State
- Successor function
- Goal test
- Solution

following weeks

Inference

- State
- Successor function
- Goal test
- Solution

Today Sept 15

- Finish Search
- **Constraint Satisfaction Problems**
  - Variables/Features
  - Constraints
  - CSPs
  - Generate-and-Test
  - Search
  - Consistency
  - Arc Consistency
- ….

Variables/Features, domains and Possible Worlds

- Variables / features
  - we denote variables using capital letters A, B
  - each variable V has a domain *dom*(V) of possible values

    $dom(B) = dom(A) = \{0, 1\}$

- Variables can be of several main kinds:
  - Boolean: $|dom(V)| = 2$ propositions
  - Finite: the domain contains a finite number of values
  - Infinite but Discrete: the domain is countably infinite
  - Continuous: e.g., real numbers between 0 and 1

- Possible world: a complete assignment of values to a set of variables

  e.g. $\{A = 1, B = 0\}$

Possible Worlds

2 81 360 180

one possible state {S, +35, 30°, 110°}

sunny cloudy

Mars Explorer Example

Weather {S, C}

Temperature {-40 – +40}

longitude latitude

LocX 0° 359° *LocY* 0° 179°

2 * 81 * 360 * 180

number of possible worlds

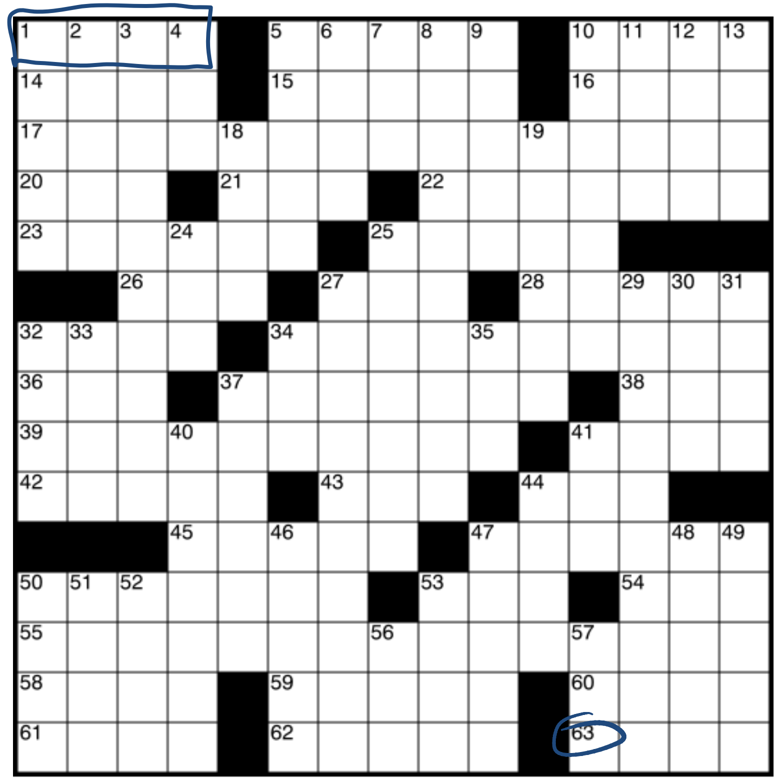
mutually exclusive

Product of cardinality of each domain

… always exponential in the number of variables

Examples

h_1



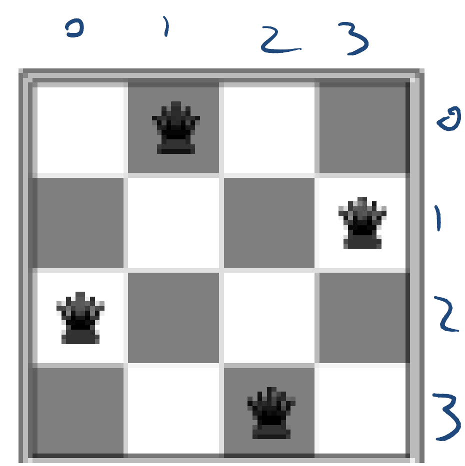
63

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

- *Number of English words?* $150 * 10^3$
- *Number of words of length k ?* 1-10 15×10^3
- *So, how many possible worlds?* $(15 \times 10^3)^{63}$

More examples

- n-Queens problem
  - variable: location of a queen on a chess board
    - there are *n* of them in total, hence the name
  - domains: grid coordinates n^2
  - possible worlds: locations of all queens



$(n^2)^n$

no overlaps

$\longrightarrow \binom{n^2}{n} = \frac{n^2!}{(n^2-n)!\, n!}$

possible ways to choose n location out of n^2

$\frac{16!}{12!\, 4!}$

More examples

- Scheduling Problem: task 1, task 2, ...
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job) (start_time, end-time, location)
  - possible worlds: time/location assignments for each task

e.g. $task_1 = \{11am.., room\ 310\}$
$task_2 = \{12pm, room101\}$
. . . . . .

Scheduling possible world

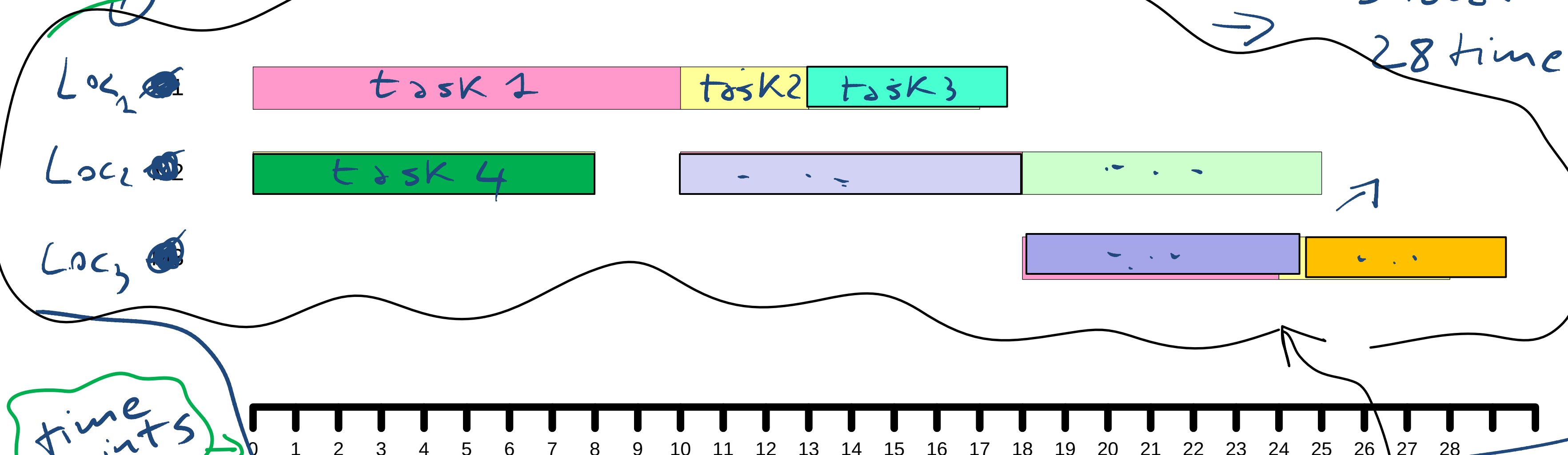
- *how many possible worlds?*

in general

$$(\#\text{locs} * \text{timepoints} * \text{time points})^{\#\text{tasks}}$$

$$(3 * 28 * 28)^{8}$$

8 tasks
3 locations
28 timepoints



time points *

task1 (start-t=0; end-t=10; loc=Loc1)

task2 (start-t=10; end-t=13; loc=Loc1)

. . . .

one possible world

More examples….

- Map Coloring Problem
  - variable: regions on the map
  - domains: possible colors
  - possible worlds: color assignments for each region

- *how many possible worlds?*

$(\#\text{colors})^{\#\text{regions}}$

Constraints

Constraints are restrictions on the values that one or more variables can take $A \quad B \quad C \quad \{0, 1\}$

- Unary constraint: restriction involving a single variable
  - $\{A = 1\} \quad \{B < 1\}$
- k-ary constraint: restriction involving the domains of k different variables $\underline{A = B} \quad A > B + C$
  - it turns out that k-ary constraints can always be represented as binary constraints, so we'll *mainly* only talk about this case

- **Constraints can be specified by**
  - giving a function that returns true when given values for each variable which satisfy the constraint
  - giving a list of valid domain values for each variable participating in the constraint $\{A = 0 \quad B = 0\}$ $\{A = 1 \quad B = 1\}$

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, SA ≠ NT, SA ≠ WA . . . .

or, (WA,NT) in {(red,green),(red,blue),(green,red),
(green,blue),(blue,red),(blue,green)}

Constraints (cont.)

- A **possible world** satisfies **a set of constraints** if the set of variables involved in each constraint take values that are consistent with that constraint

possible world

- A,B,C domains [1 .. 10]
- A= 1 , B = 2, C = 10 pw

False satisfies True

does not satisfy

- Constraint set1 {A = B, C>B}
- Constraint set2 {A $\neq$ B, C>B}

Examples

- Crossword Puzzle:
  - 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

h1

15 x 15

V1

$h_1[0] = V_1[0]$, ...

~ 225 constraints

63

- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - *constraints:* sequences of letters form valid English words

A

63 constraints

concatenate(A[0,0] ... A[0,3]) ∈ English word of length 4

More examples

e.g. two Queens cannot be on the same column / row

$Q_1 = \{x_1, y_1\}$
$Q_2 = \{x_2, y_2\}$
$x_1 \neq x_2$ and $y_1 \neq y_2$

- n-Queens problem
  - variable: location of a queen on a chess board
    - there are *n* of them in total, hence the name
  - domains: grid coordinates
  - *constraints*: no queen can attack another
- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - *constraints*: e.g. $task_1(loc_1, start\text{-}t_1)$ if $start\text{-}t_1 = start\text{-}t_2$ $task_2(loc_2, start\text{-}t_2)$ then $loc_1 \neq loc_2$
    - ✓ tasks can't be scheduled in the same location at the same time;
    - ✓ certain tasks can be scheduled only in certain locations;
    - ✓ some tasks must come earlier than others; etc.

Constraint Satisfaction Problems: definitions

Definition (Constraint Satisfaction Problem)

A constraint satisfaction problem consists of

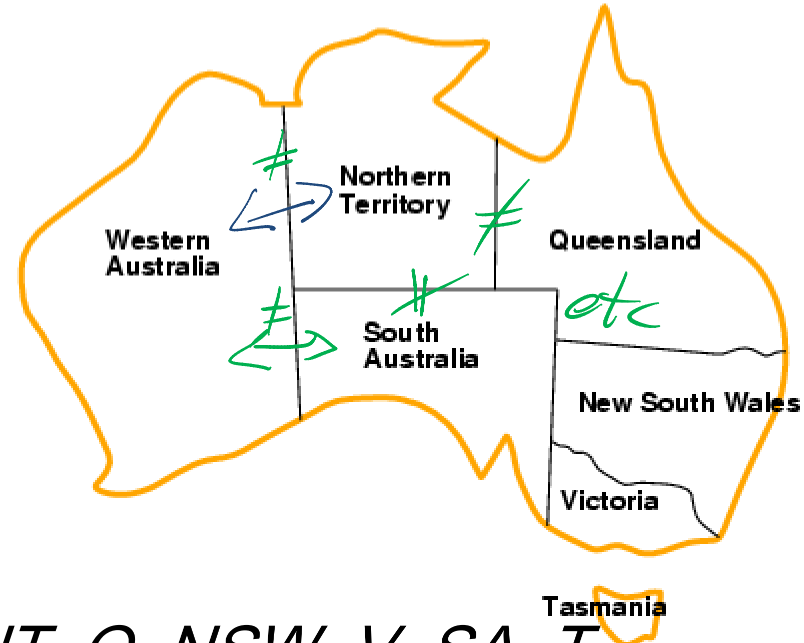
- a set of variables
- a domain for each variable
- a set of constraints

Definition (model / solution)

A possible world

A model of a CSP is an assignment of values to variables that satisfies all of the 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., WA ≠ NT, or

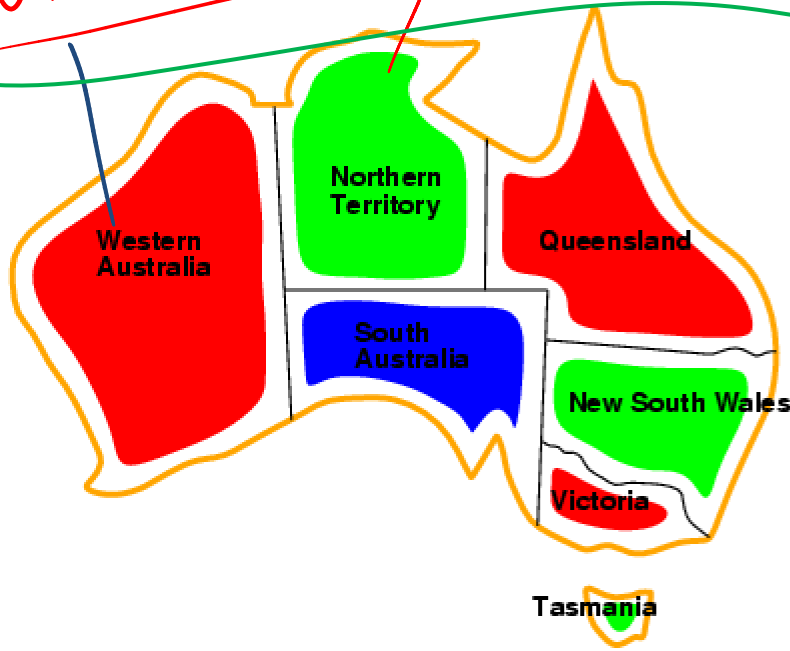
(WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

Example: Map-Coloring

UNIQUE ?

WA = red   NT = green

with these two it becomes unique



Models / Solutions are complete and consistent
assignments, e.g., WA = red, NT = green, Q = red, NSW = green, V = red,SA = blue, T = green

Constraint Satisfaction Problem: Variants

We may want to solve the following problems using a CSP

A. determine whether or not a model exists

useful to avoid wasting time on B

B. find a model

C. find all of the models

D. count the number of the models

E. find the best model given some model quality

- this is now an optimization problem

F. determine whether some properties of the variables hold in all models

Constraint Satisfaction Problems: Game Plan

- Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is NP-hard
  - There is no known algorithm with worst case polynomial runtime
  - We can't hope to find an algorithm that is efficient for all CSPs
- However, we can try to:
  - identify special cases for which algorithms are efficient (polynomial)
  - 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 cases

Today Sept 15

- Finish Search
- **Constraint Satisfaction Problems**
  - Variables/Features
  - Constraints
  - CSPs
  - Generate-and-Test
  - Search
  - Consistency
  - Arc Consistency

- ….

Generate-and-Test Algorithm

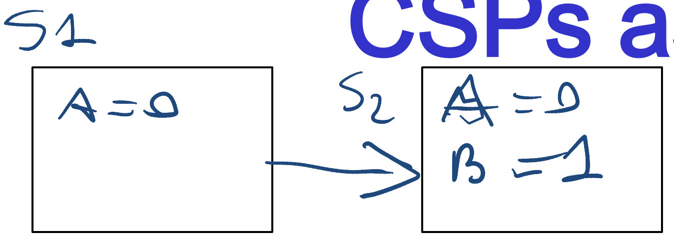
$\rightarrow$ domA = {1, 2, 3, 4, 5}
$\rightarrow$ domB = {1, 2, 3, 4, 5}
$\rightarrow$ domC = {1, 2, 3}

- **Algorithm:**
  - Generate possible worlds one at a time
  - Test them to see if they violate any constraints

```
For a in domA
   For b in domB
       For c in domC
       if (a b c) satisfies all constraints
       return (abc)
return fail
```

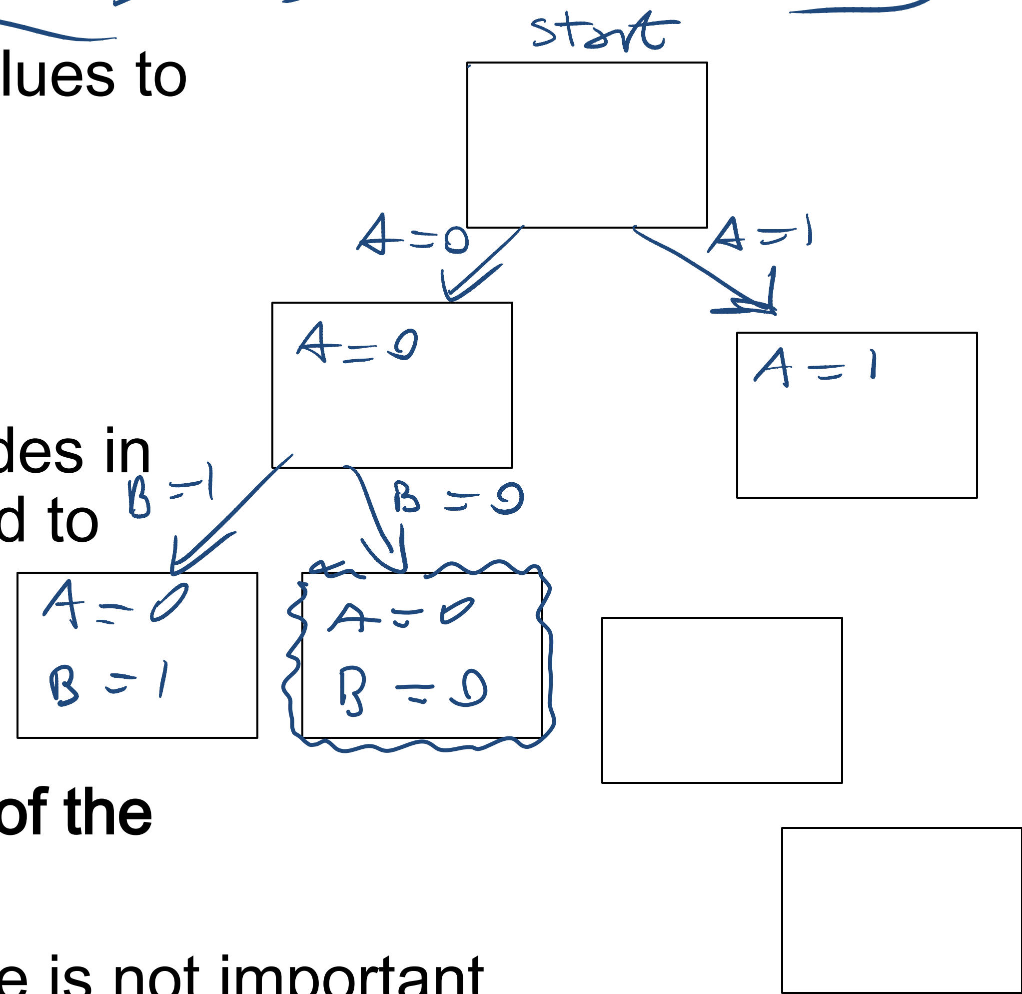
- This procedure is able to solve any CSP
- However, the running time is proportional to the number of possible worlds
  - always exponential in the number of variables
  - far too long for many CSPs ☹

CSPs as search problems



A, B   dom A = dom B = {0,1}

A = B



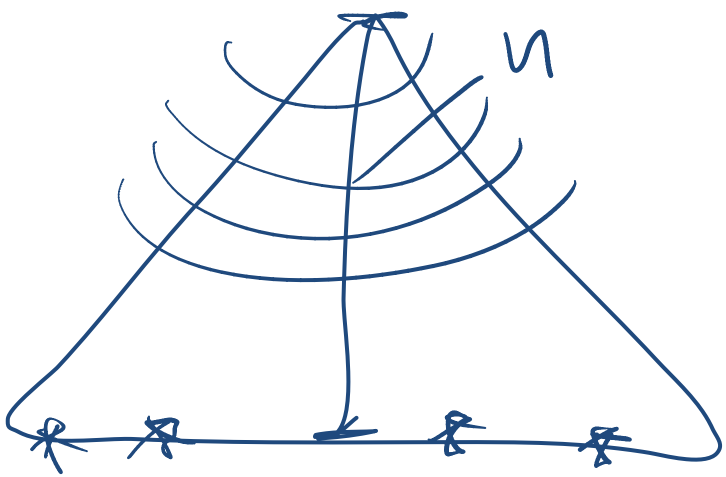
- **states:** assignments of values to a subset of the variables
- **start state:** the empty assignment (no variables assigned values)
- **neighbours** of a state: nodes in which values are assigned to one additional variable
- **goal state:** a state which **assigns a value to each variable**, and **satisfies all of the constraints**

Note: the path to a goal node is not important

CSPs as Search Problems

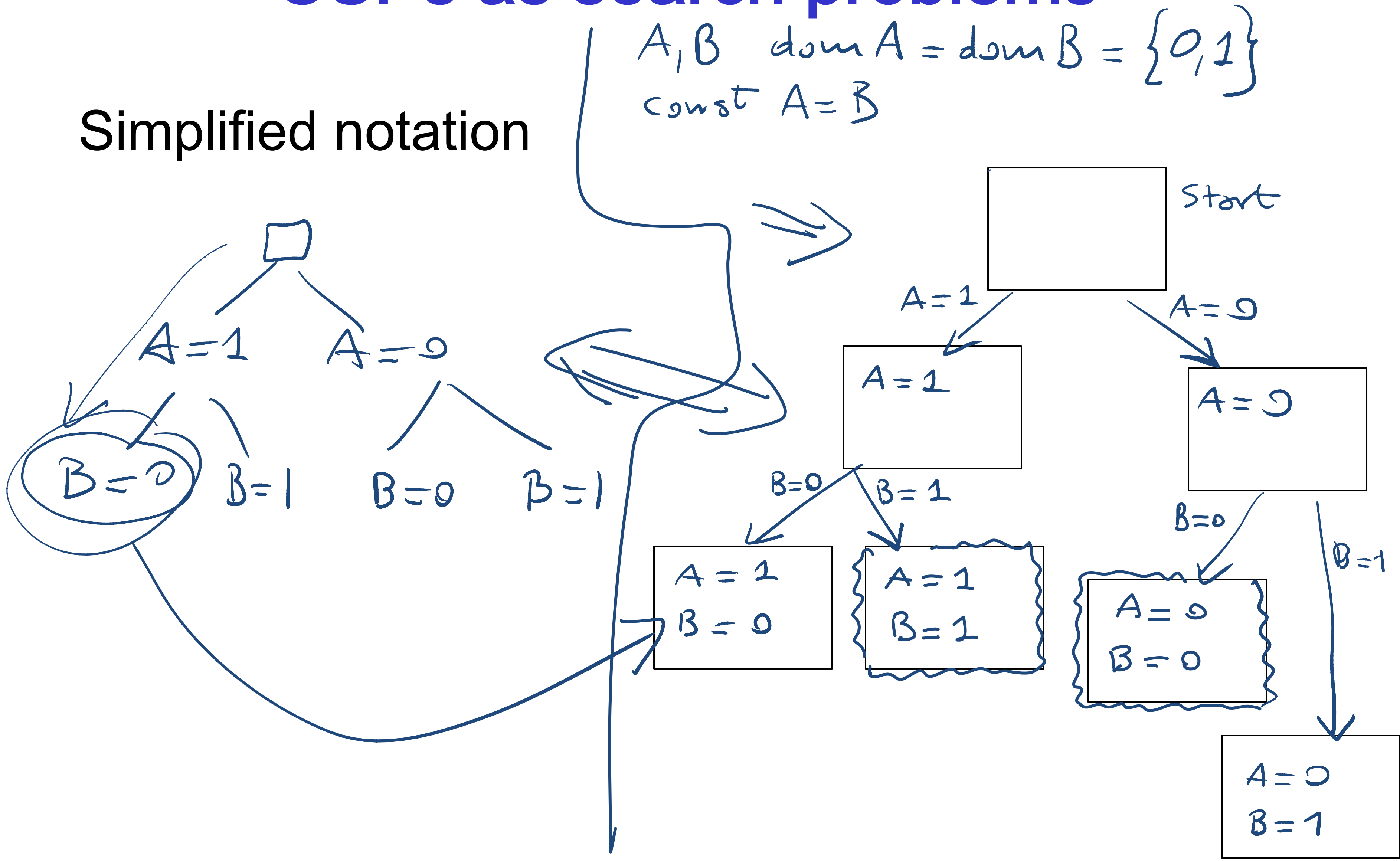
What search strategy will work well for a CSP?

- If there are n variables every solution is at depth……. n
- Is there a role for a heuristic function?
- the tree is always finite and has no cycles, so which one is better BFS or IDS or DFS?



CSPs as search problems

Simplified notation

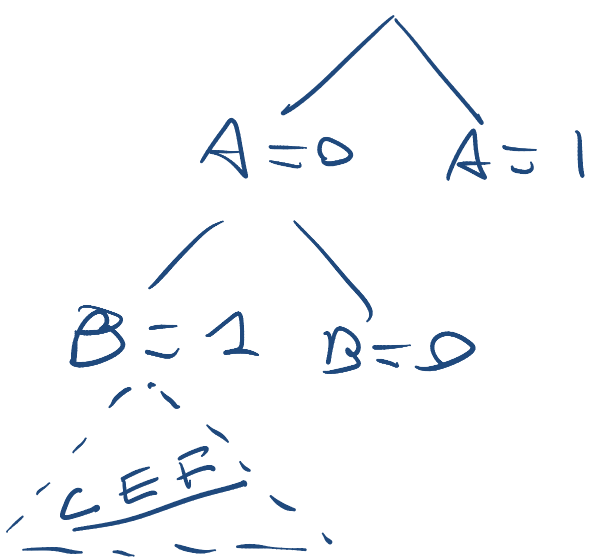


CSPs as Search Problems

How can we avoid exploring some sub-trees i.e.,

prune the DFS Search tree?

- once we consider a path whose end node violates one or more constraints, we know that a solution cannot exist below that point
- thus we should **remove that path** rather than continuing to search



A B C E F {1, 0}

constraints (A = B) (F > A)

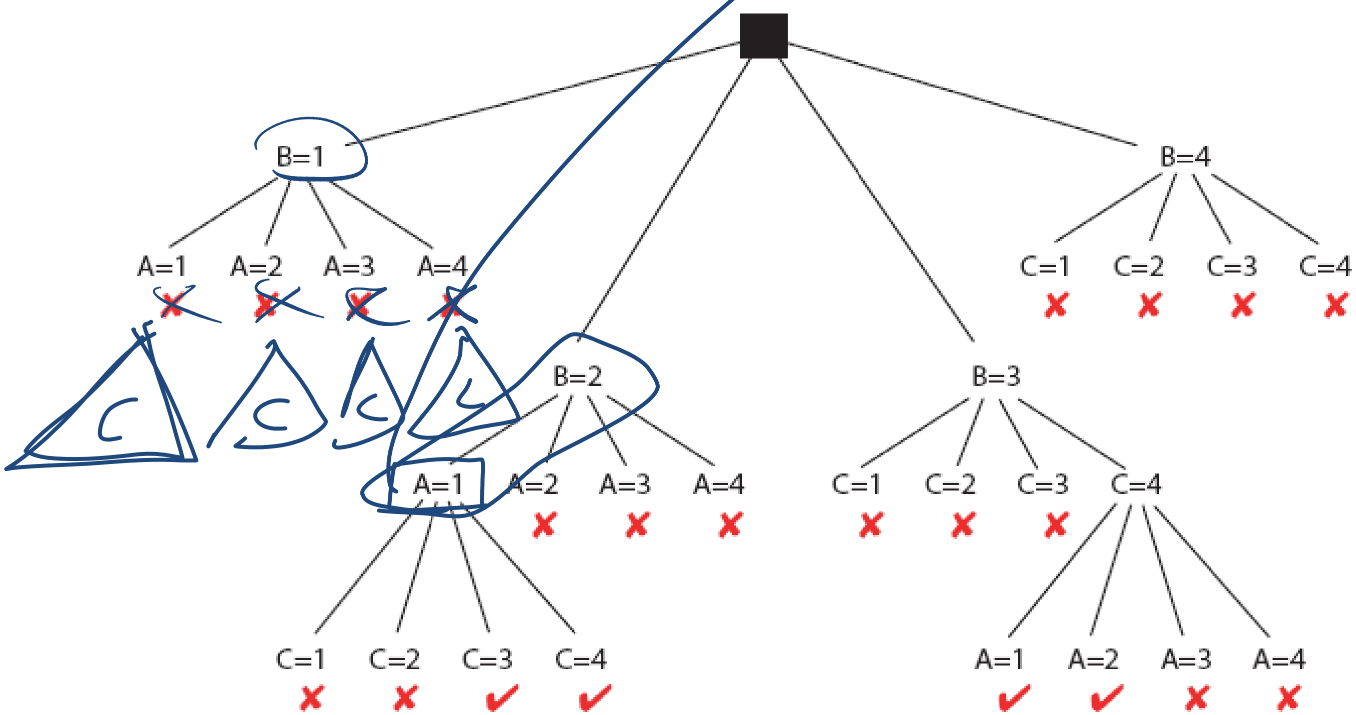
(C = E)

Solving CSPs by DFS: Example

Problem:

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

B = 2
A = 1



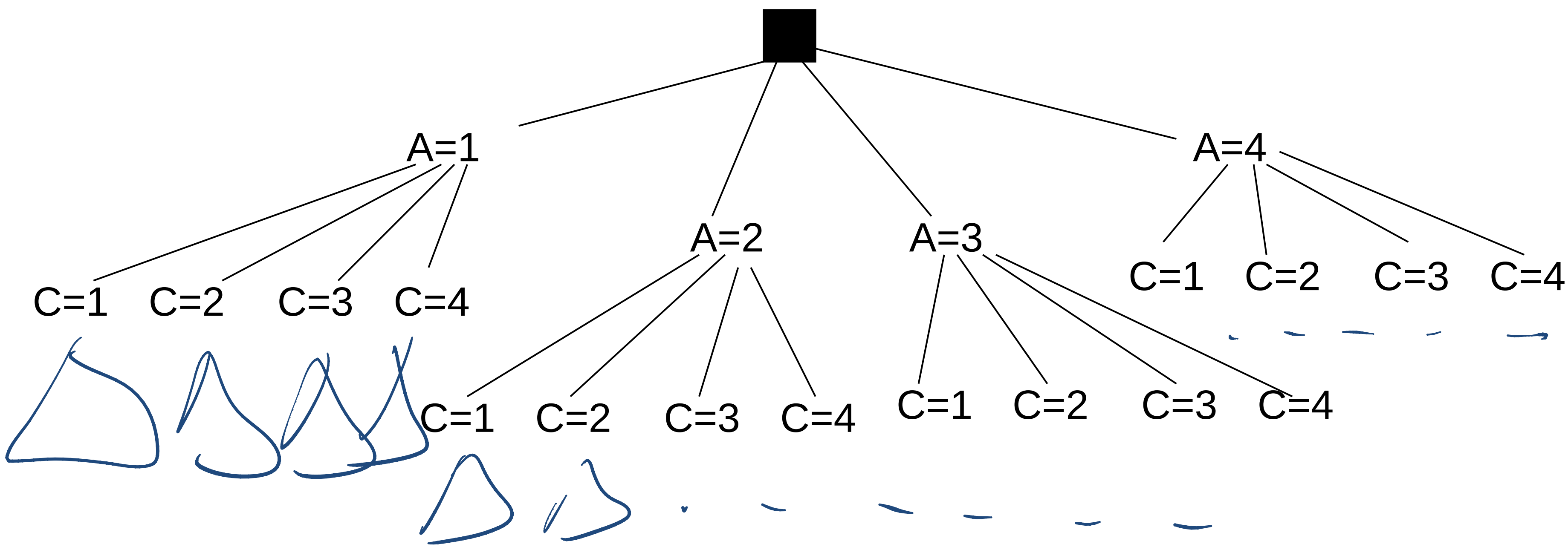
Solving CSPs by DFS: Example Efficiency

Problem:

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

Note: the algorithm's efficiency depends on the order in which variables are expanded

Degree "Heuristics"



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: set of constraints
- 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

Can we do better than Search?

Key ideas:

- **prune the domains** as much as possible **before "searching"** for a solution.

Simple when using constraints involving single variables (technically enforcing **domain consistency**)

- Example: $\mathbf{D}_B = \{1, 2, \cancel{3}, 4\}$ with constraint $B \neq 3$.

How do we deal with constraints involving multiple variables?

Definition (**constraint network**)

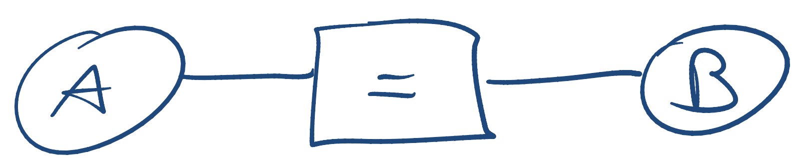
A constraint network is defined by a graph, with

- one **node** for every **variable**
- one **node** for every **constraint**

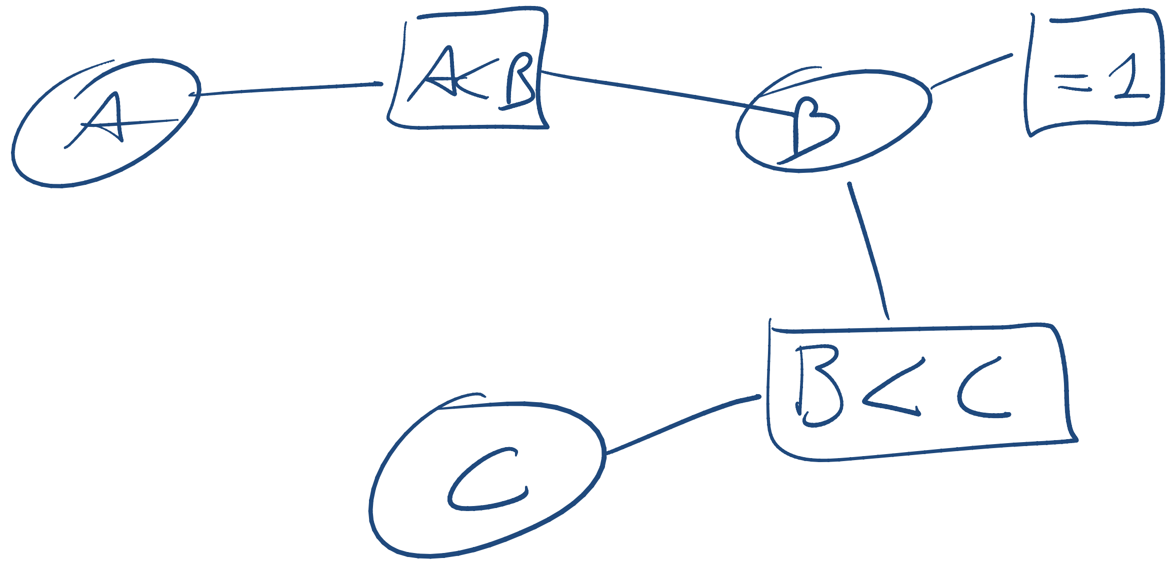
and undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

A B {0, 1}

A = B



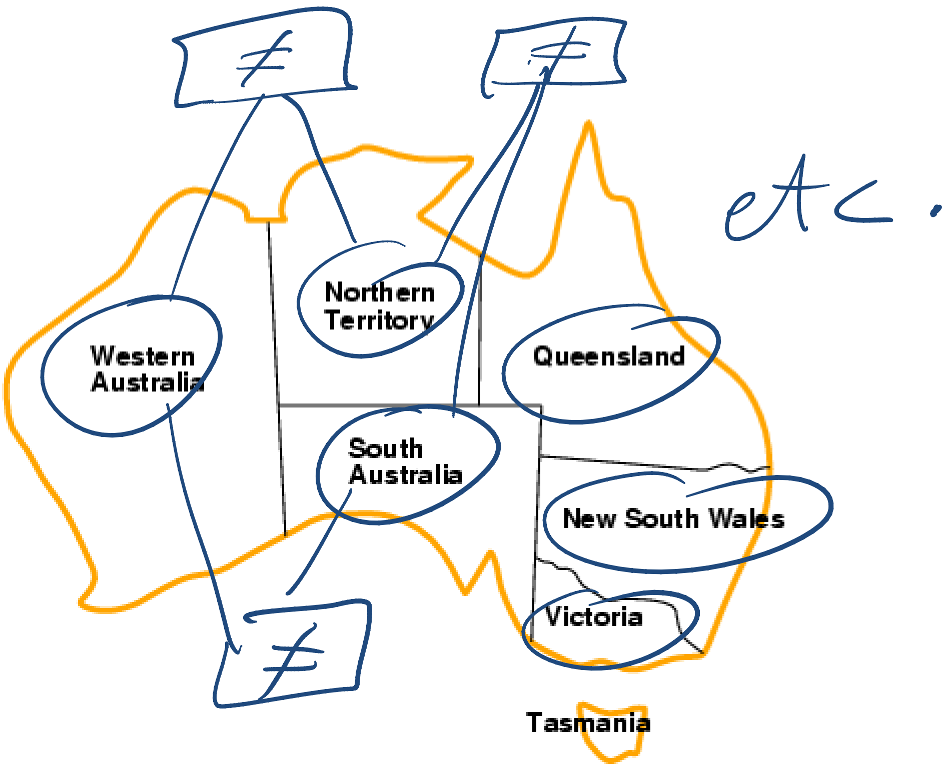
Example Constraint Network



Recall Example:

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

Example: Constraint Network for Map-Coloring



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

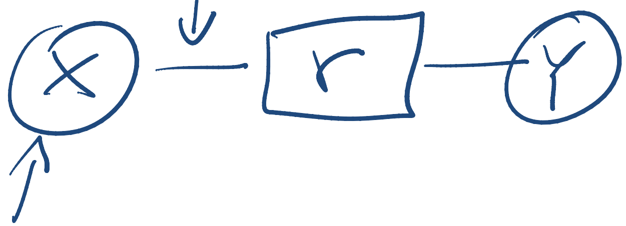
Domains D_i = {red,green,blue}

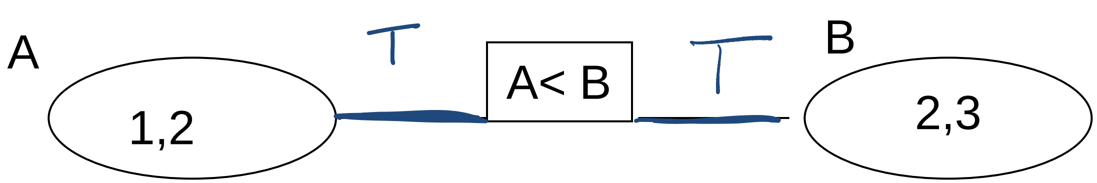
Constraints: adjacent regions must have different colors

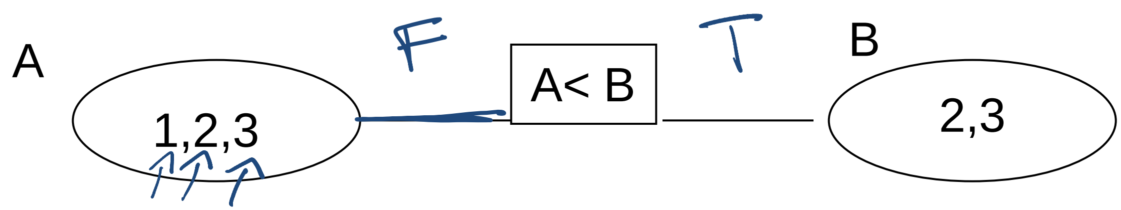
Arc Consistency

Definition (arc consistency)

An arc $\langle X, r(X,Y)\rangle$ is arc consistent if for each value x in $dom(X)$ there is some value y in $dom(Y)$ such that $r(x,y)$ is satisfied.

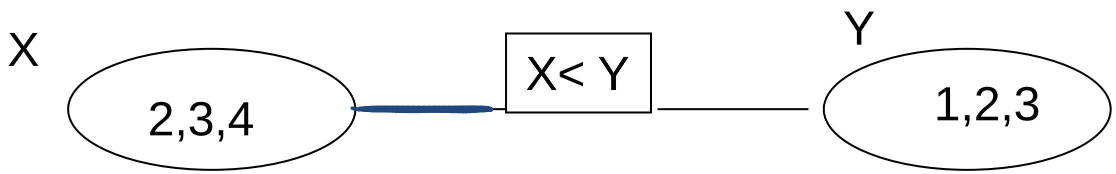






How can we enforce Arc Consistency?

- If an arc $\langle X, r(X,Y) \rangle$ is not arc consistent, all values x in $dom(X)$ for which there is no corresponding value in $dom(Y)$ may be deleted from $dom(X)$ to make the arc $\langle X, r(X,Y) \rangle$ consistent.
  - This removal can never rule out any models/solutions



- **A network is arc consistent** if all its arcs are arc consistent.

Arc Consistency Algorithm: high level strategy

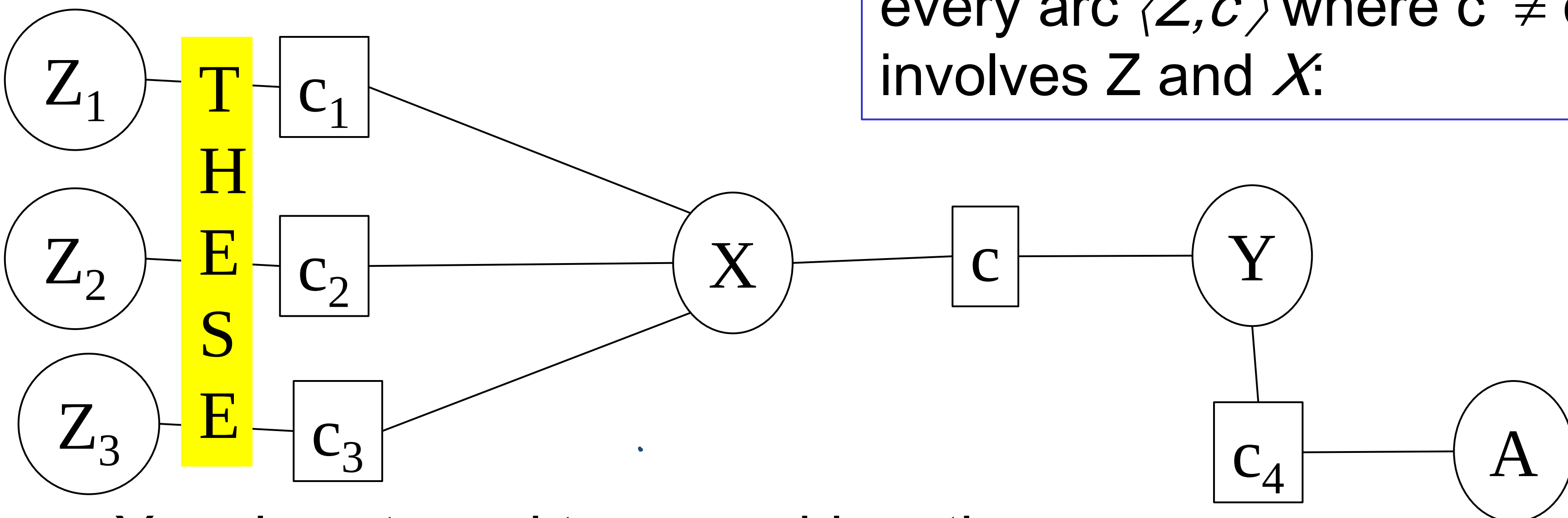
- Consider the arcs in turn, making each arc consistent.
- BUT, arcs may need to be revisited whenever….

  - NOTE - Regardless of the order in which arcs are considered, we will terminate with the same result

Which arcs need to be reconsidered?

- When we reduce the domain of a variable X to make an arc ⟨X,c⟩ arc consistent, which arcs do we need to reconsider?

every arc ⟨*Z,c'*⟩ where c' ≠ c involves Z and *X*:



- You do not need to reconsider other arcs
  - If an arc ⟨X,c'⟩ was arc consistent before, it will still be arc consistent
  - Nothing changes for arcs of constraints not involving X

Arc Consistency Algorithm: Complexity

- Let's determine Worst-case complexity of this procedure (compare with DFS d^n)
  - let the max size of a variable domain be ***d***
  - let the number of variables be ***n***
  - The max number of binary constraints is……. $n(n-1)/2$

- How many times the same arc can be inserted in the ToDoArc list? d

$\boxed{O(d^3 n^2)}$

- How many steps are involved in checking the consistency of an arc? d^2

$\{X_1 \dots X_d\}$ $\{Y_1 \dots Y_d\}$

OVERALL COMPLEXITY

Arc Consistency Algorithm: Interpreting Outcomes

- Three possible outcomes (when all arcs are arc consistent):
  - One domain is empty → no sol
  - Each domain has a single value → unique sol ☺
  - Some domains have more than one value → may or may not be a solution
    - in this case, arc consistency isn't enough to solve the problem: we need to perform search

see arc consistency (AC) practice exercise

Domain splitting (or case analysis)

- Arc consistency ends: Some domains have more than one value → may or may not be a solution
  - A. Apply Depth-First Search with Pruning
  - B. Split the problem in a number of disjoint cases

$$CSP = \{ X = \{x_1\ x_2\ x_3\ x_4\} \ldots \}$$

$$CSP_1 \{ X = \{x_1\ x_2\} \ldots \} \qquad CSP_2 \{ X \{x_3\ x_4\} \}$$

- Set of all solution equals to….

$$Sol(CSP) = \bigcup_i sol(CSP_i)$$

But what is the advantage?

By reducing dom(X) we may be able to run AC again

Complete Process

- Simplify the problem using **arc consistency**
- No unique solution i.e., for at least one var, |dom(X)|>1
- **Split X**
- For all the splits
  - Restart arc consistency on arcs <Z, r(Z,X)>

Initial TDA

these are the ones that are possibly **inconsistent**

- Disadvantage ☹: you need to keep all these CSPs around (vs. lean states of DFS)

Searching by domain splitting

CSP; apply AC

some domains have multiple values

split on X

CSP_1

apply AC

some domains . . . . .

split Y

CSP_2

apply AC

some domains . . . .

split Z

- Disadvantage ☹: you need to keep all these CSPs around (vs. lean states of DFS)

Systematically solving CSPs: Summary

- Build Constraint Network
- Apply Arc Consistency
  - One domain is empty → no sol
  - Each domain has a single value → unique sol
  - Some domains have more than one value → ?!

    may or may not have a solution

- Apply Depth-First Search with Pruning
- Split the problem in a number of disjoint cases
  - Apply Arc Consistency to each case

Local Search motivation: Scale

- Many CSPs (scheduling, DNA computing, more later) are simply too big for systematic approaches
- If you have 10^5 vars with dom(var_i) = 10^4

- Systematic Search

$b = 10^4$ branching factor

$d = 10^5$ depth

$(10^4)^{10^5}$

- Constraint Network

var nodes

constraint nodes

$10^5 + 10^5 * 10^5$

10^{10} max # of nodes

- but if solutions are densely distributed…….

TODO for this Thue

Read Chp 4 of textbook (especially from 4.8 to end)

Do exercises 4.A , 4.B available at http://www.aispace.org/exercises.shtml

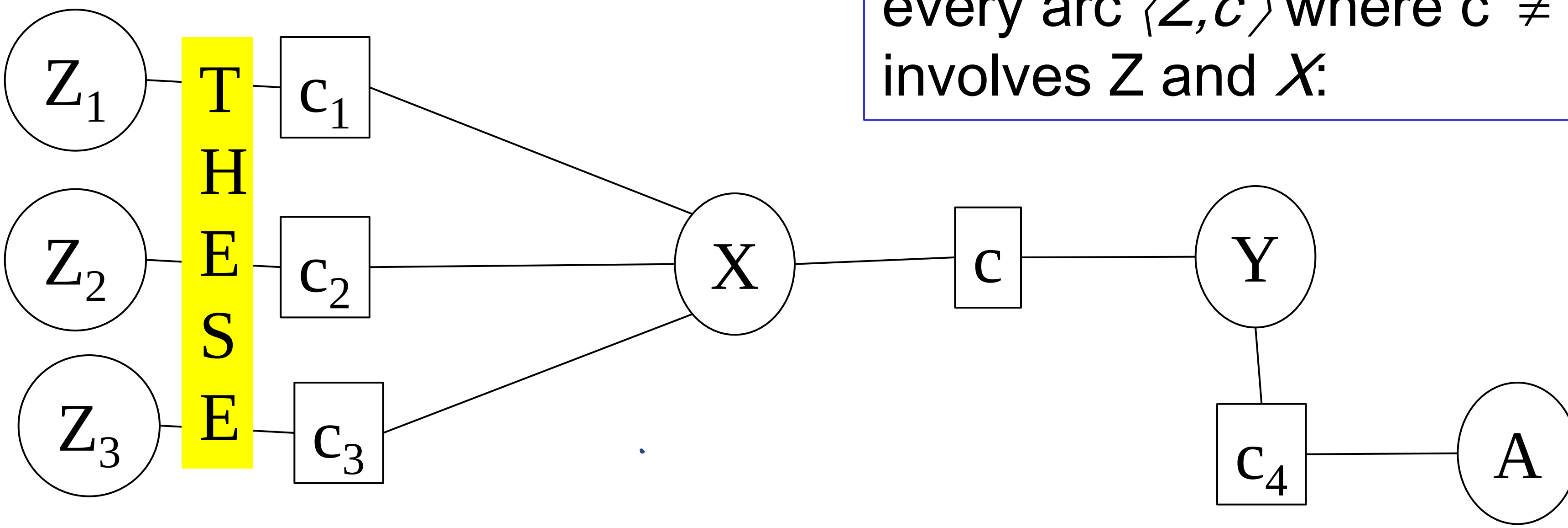
Please, look at solutions only after you have tried hard to solve them!

- Join piazza (the class discussion forum)

Which arcs need to reconsidered?

- When we reduce the domain of a variable X to make an arc ⟨X,c⟩ arc consistent, which arcs do we need to reconsider?

every arc ⟨*Z,c'*⟩ where c' $\neq$ c involves Z and *X*:



- You do not need to reconsider other arcs
  - If an arc ⟨X,c'⟩ was arc consistent before, it will still be arc consistent
  - Nothing changes for arcs of constraints not involving X

Arc consistency algorithm (for binary constraints)

Procedure GAC(V,dom,C)

Inputs

V: a set of variables

dom: a function such that dom(X) is the domain of variable X

C: set of constraints to be satisfied

Output

arc-consistent domains for each variable

Local

$\mathbf{D}_X$ is a set of values for each variable X

TDA is a set of arcs

for each variable X **do**

$\quad \mathbf{D}_X \leftarrow \mathrm{dom}(X)$

$\quad TDA \leftarrow \{\langle X,c \rangle \mid c \in C \text{ and } X \in \mathrm{scope}(c)\}$

while $(TDA \neq \{\})$

$\quad$ **select** $\langle X,c \rangle \in TDA$

$\quad TDA \leftarrow TDA \setminus \{\langle X,c \rangle\}$

$\quad ND_X \leftarrow \{x \mid x \in \mathbf{D}_X \text{ and } \exists\, y \in \mathbf{D}_Y \text{ s.t. } (x, y) \text{ satisfies } c\}$

$\quad$ **if** $(ND_X \neq \mathbf{D}_X)$ **then**

$\quad\quad TDA \leftarrow TDA \cup \{ \langle Z,c' \rangle \mid X \in \mathrm{scope}(c'),\ c' \neq c,\ Z \in \mathrm{scope}(c') \setminus \{X\} \}$

$\quad\quad \mathbf{D}_X \leftarrow ND_X$

return $\{\mathbf{D}_X \mid X \text{ is a variable}\}$

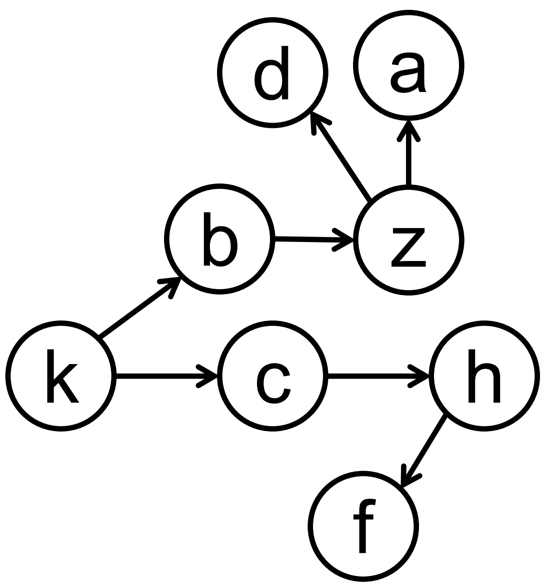
TDA: ToDoArcs, blue arcs in AIspace

Scope of constraint c is the set of variables involved in that constraint

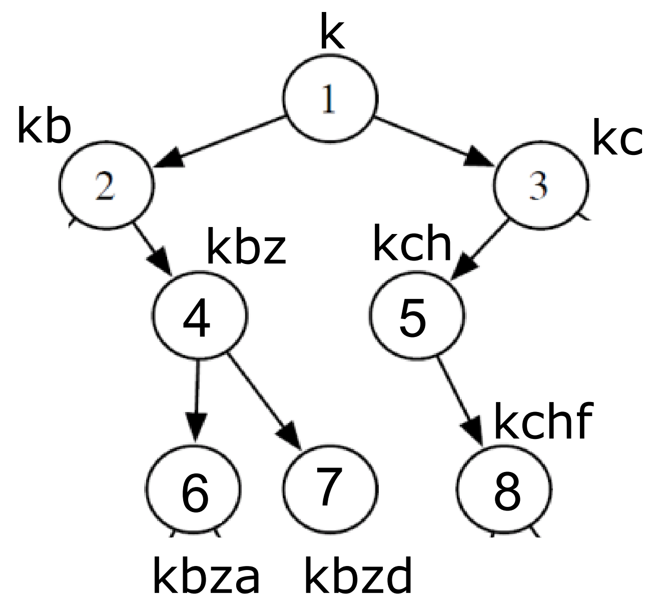
ND_X: values x for X for which there a value for y supporting x

X's domain changed: $\Rightarrow$ arcs (Z,c') for variables Z sharing a constraint c' with X could become inconsistent

Clarification: state space graph vs search tree



State space graph.

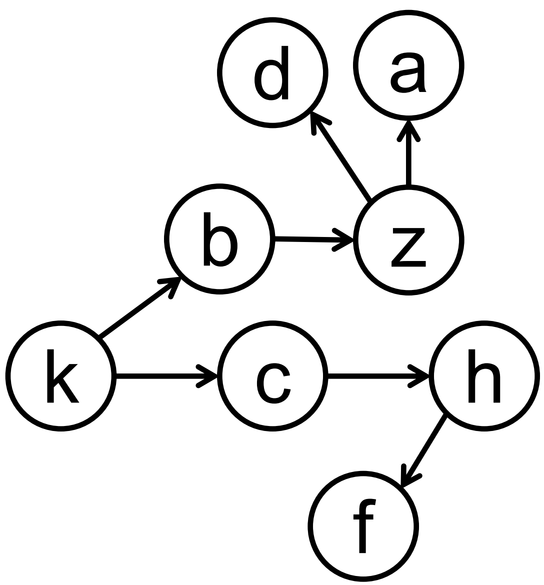


Search tree.

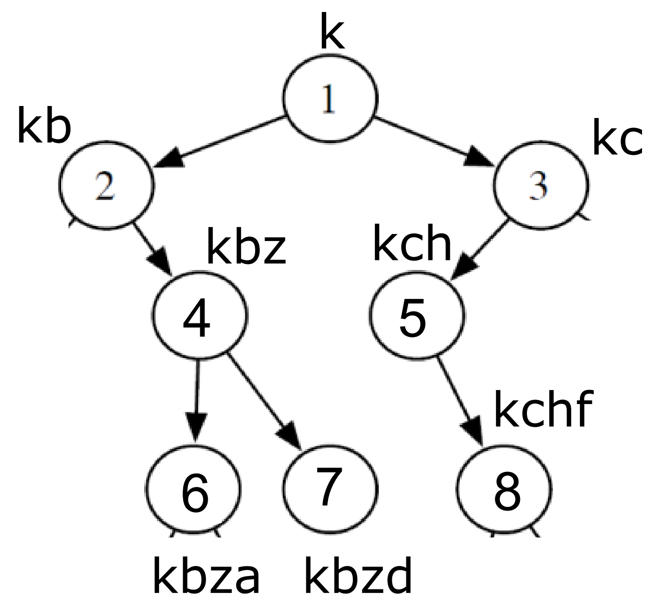
Nodes in this tree correspond to paths in the state space graph

If there are no cycles, the two look the same

Clarification: state space graph vs search tree



State space graph.



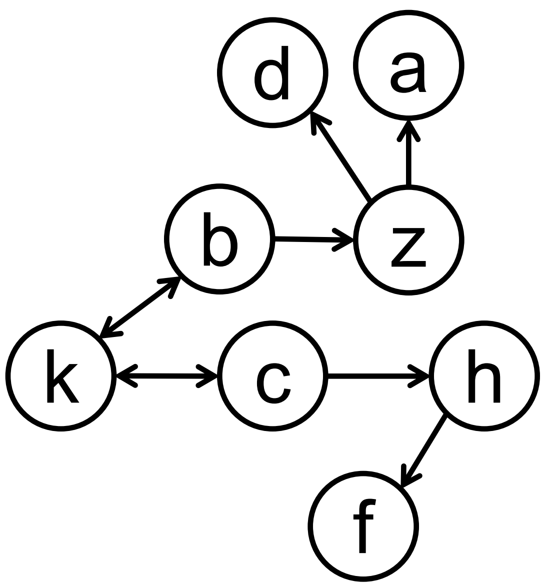
Search tree.

What do I mean by the numbers in the search tree's nodes?

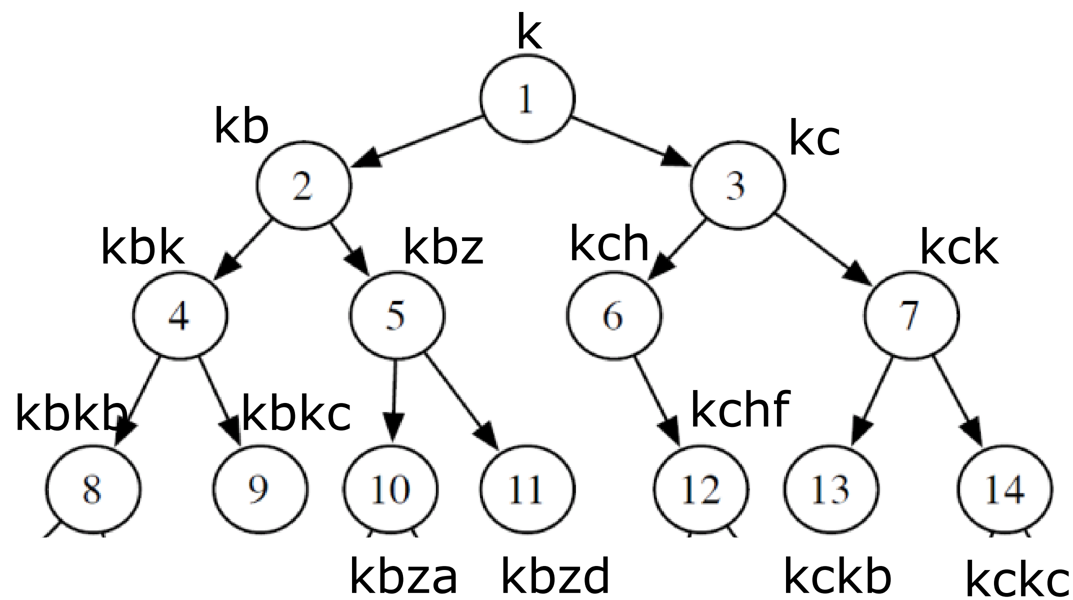
Node's name

Order in which a search algo. (here: BFS) expands nodes

Clarification: state space graph vs search tree



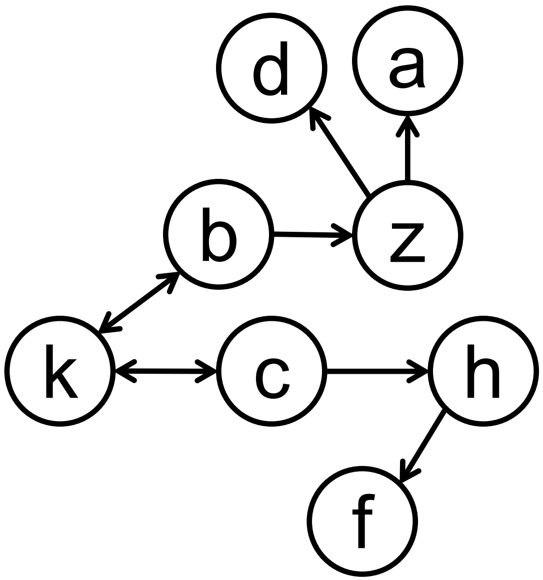
State space graph.



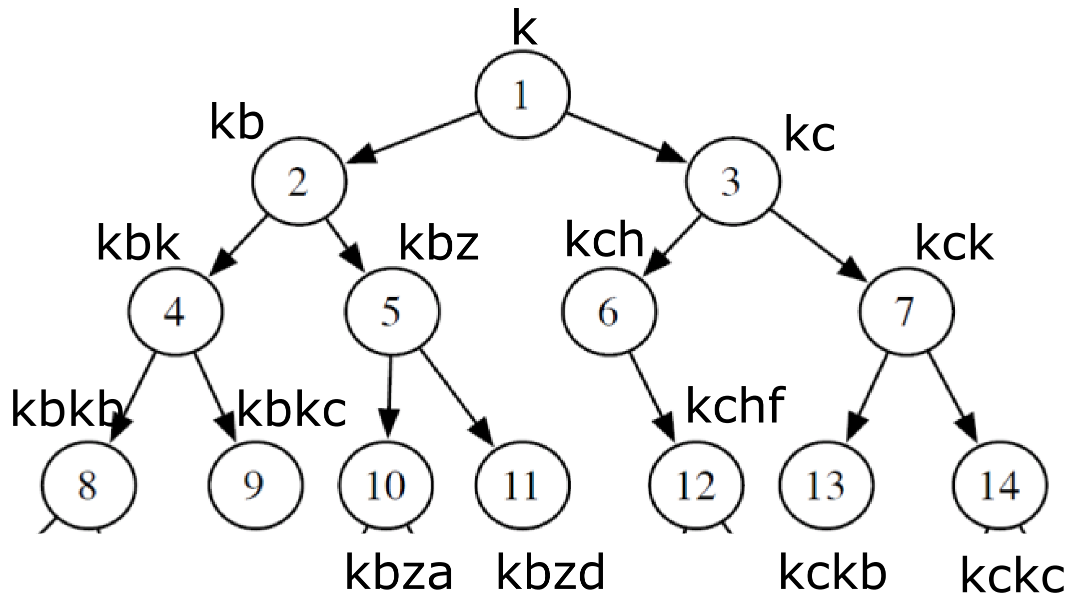
Search tree.
(only first 3 levels, of BFS)

- If there are cycles, the two look very different

Clarification: state space graph vs search tree



State space graph.



Search tree.
(only first 3 levels, of BFS)

What do nodes in the search tree represent in the state

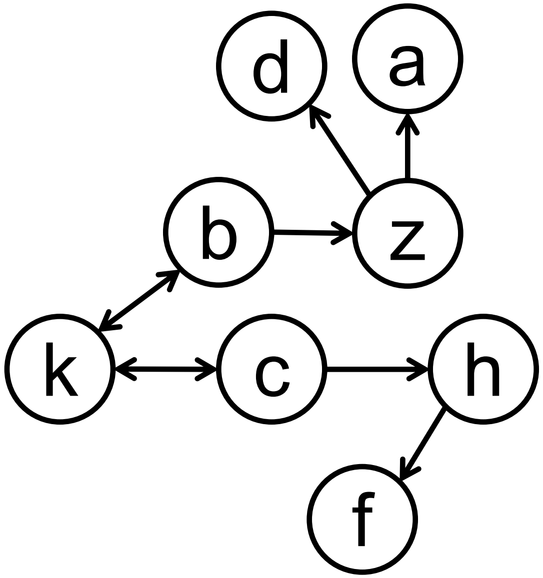
nodes

edges

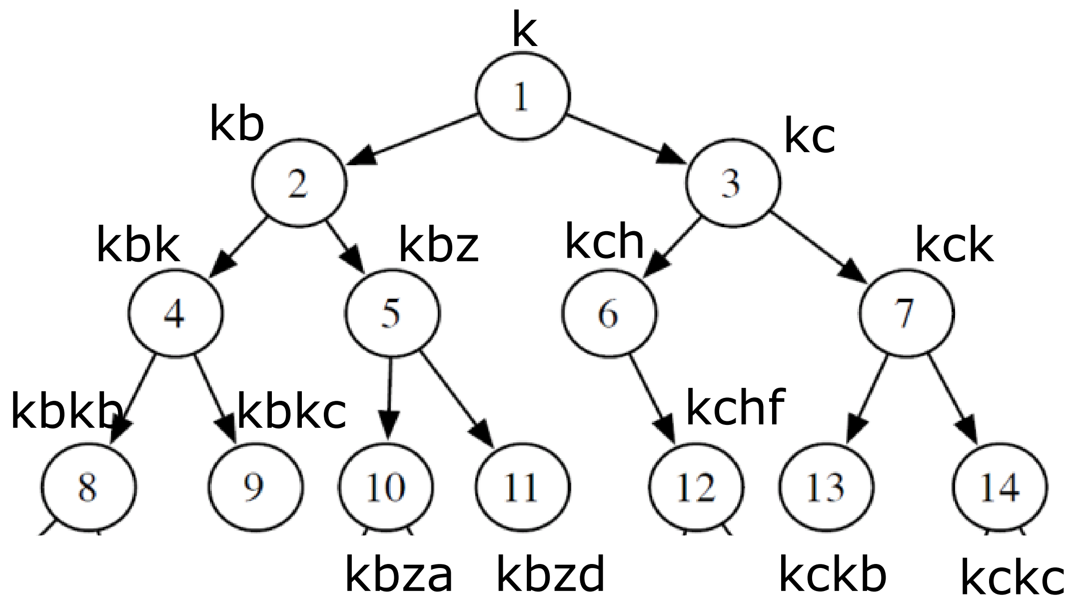
paths

states

Clarification: state space graph vs search tree



State space graph.



Search tree.
(only first 3 levels, of BFS)

What do edges in the search tree represent in the state

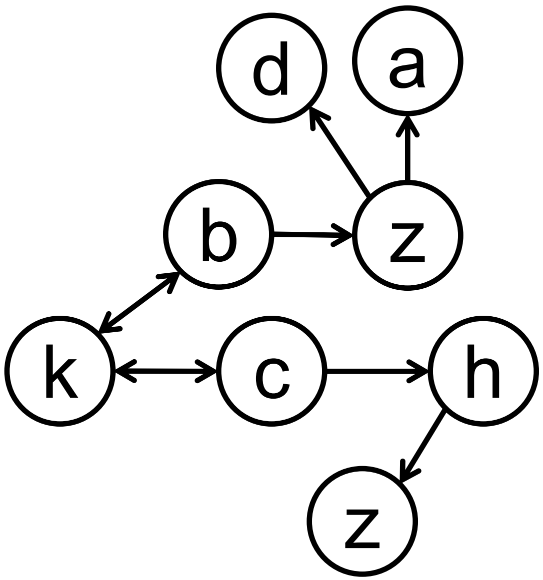
nodes

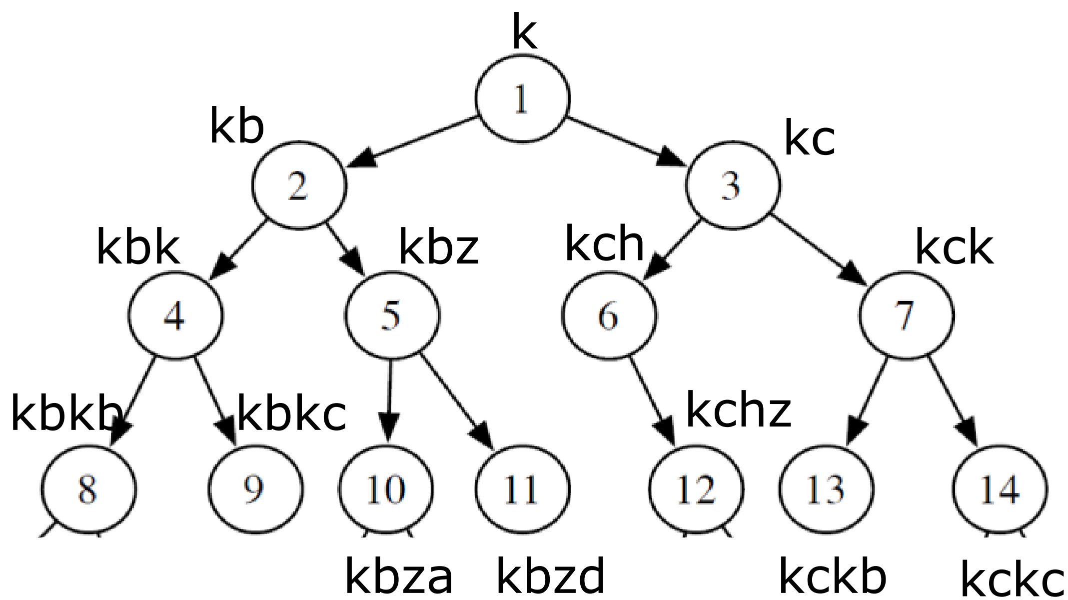
edges

paths

states

Clarification: state space graph vs search tree





State space graph.

Search tree.

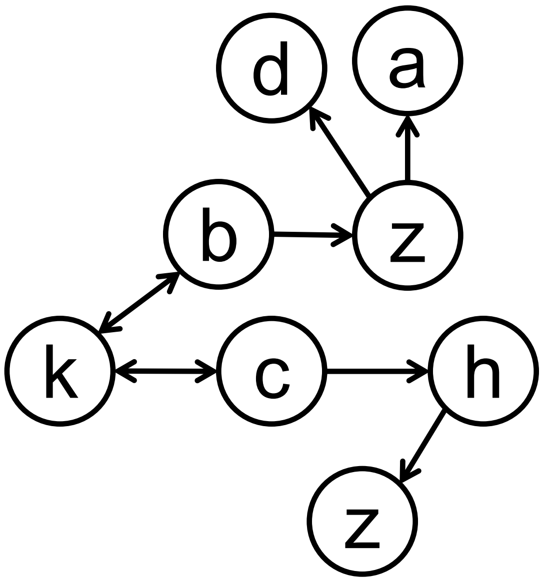
Nodes in this tree correspond to paths in the state space graph

(if multiple start nodes: forest)

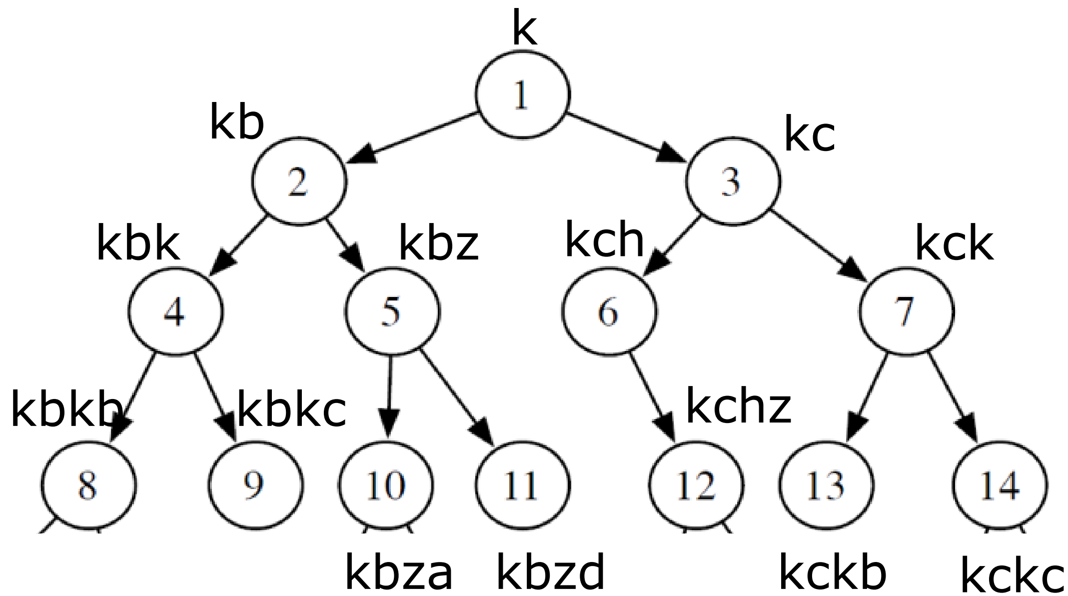
May contain cycles!

Cannot contain cycles!

Clarification: state space graph vs search tree



State space graph.



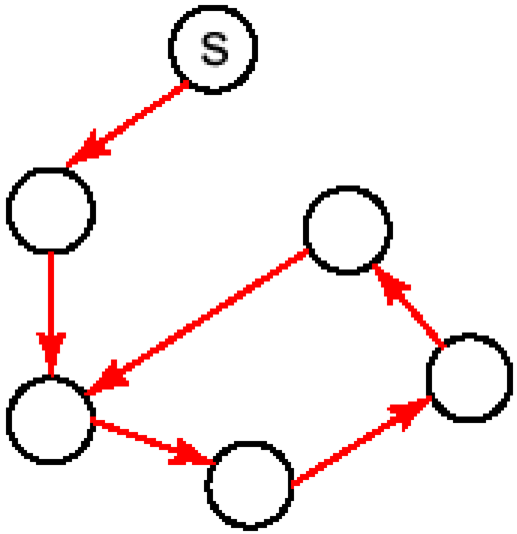
Search tree.

Nodes in this tree correspond to paths in the state space graph

Why don't we just eliminate cycles?

Sometimes (but not always) we want multiple solution paths

Cycle Checking: if we only want optimal solutions



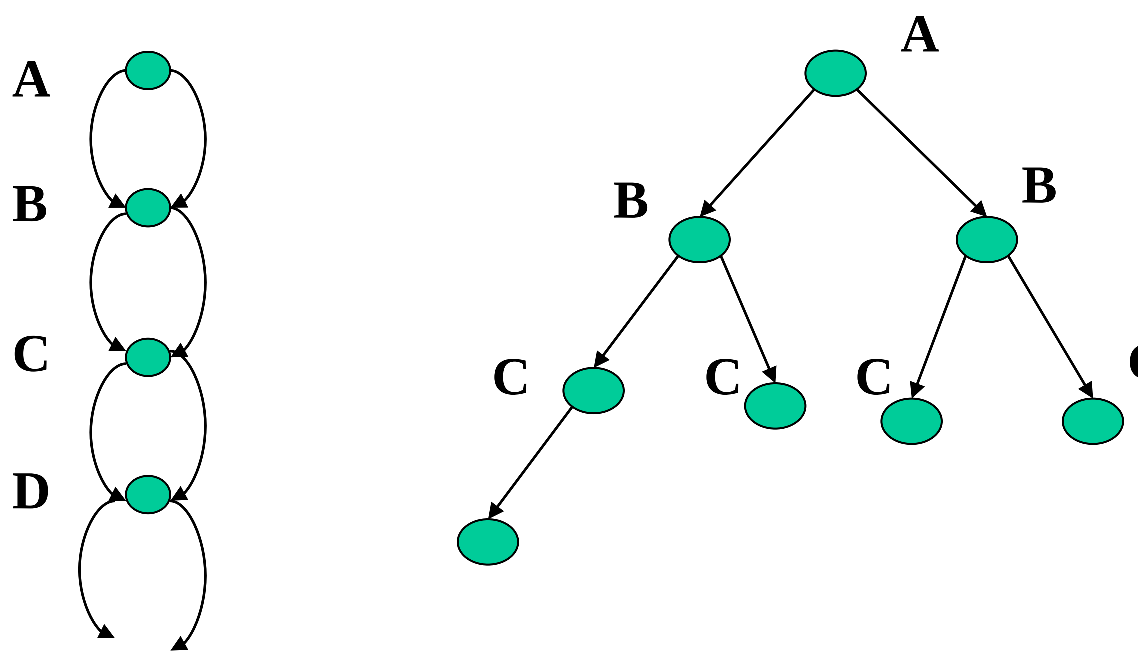
- You can prune a node *n* that is on the path from the start node to n.
- This pruning cannot remove an optimal solution $\Rightarrow$ cycle check

- Using depth-first methods, with the graph explicitly stored, this can be done in constant time
  - Only one path being explored at a time

- Other methods: cost is linear in path length
  - (check each node in the path)

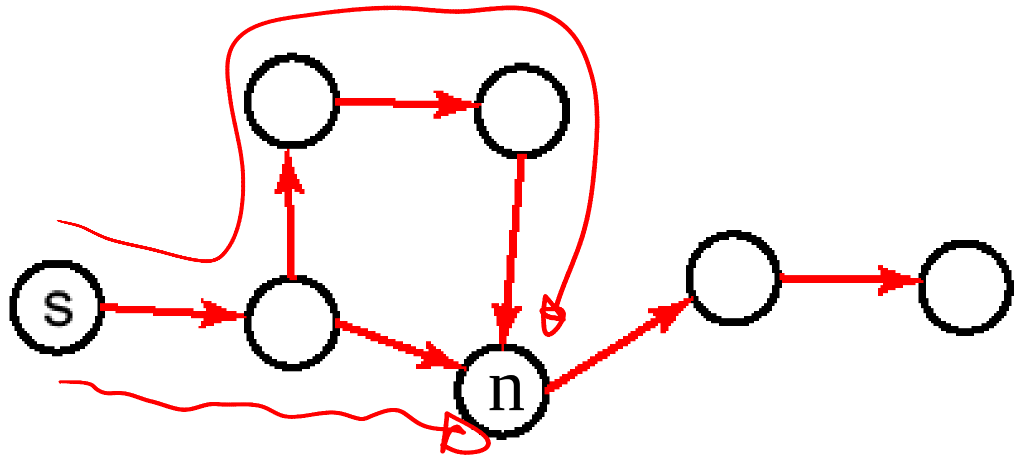
Size of search space vs search tree

- With cycles, search tree can be exponential in the state space
  - E.g. state space with 2 actions from each state to next
  - With d + 1 states, search tree has depth d



- **2^d possible paths through the search space => exponentially larger search tree!**

Multiple Path Pruning

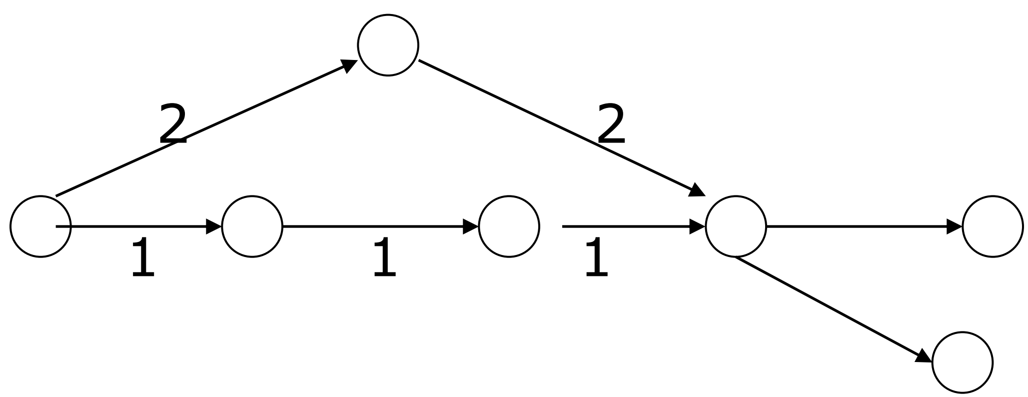


- If we only want one path to the solution
- Can prune path to a node *n* that has already been reached via a previous path
  - Store S := {all nodes n that have been expanded}
  - For newly expanded path p = $(n_1,\ldots,n_k,n)$
    - Check whether $n \in S$
    - Subsumes cycle check
- Can implement by storing the path to each expanded node

Multiple-Path Pruning & Optimal Solutions

- Problem: what if a subsequent path to *n* is shorter than the first path to n, and we want an optimal solution ?

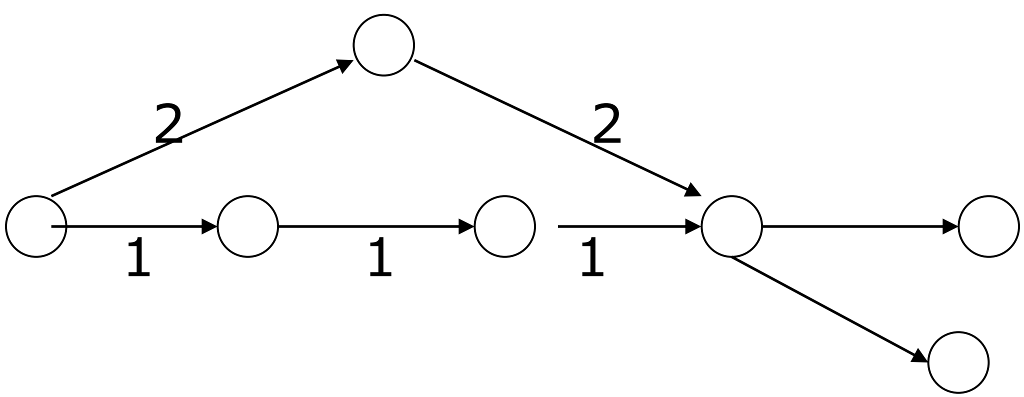
- Can remove all paths from the frontier that use the longer path. (these can't be optimal)



Multiple-Path Pruning & Optimal Solutions

- Problem: what if a subsequent path to *n* is shorter than the first path to n, and we want just the optimal solution ?

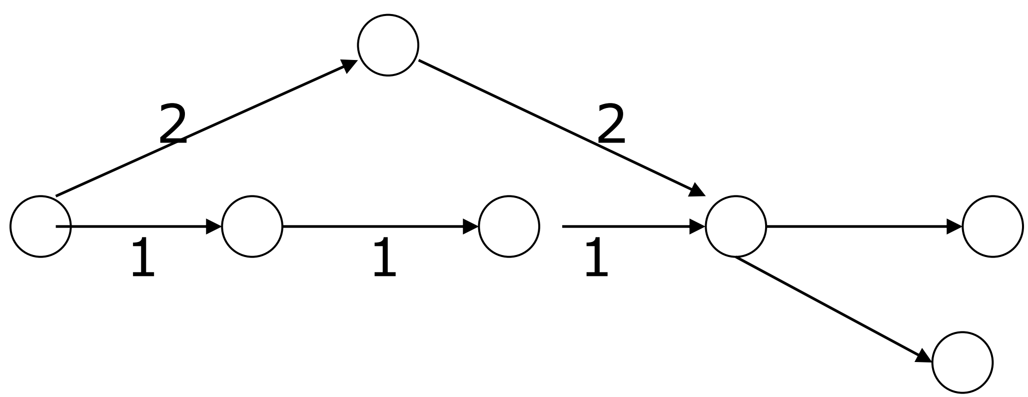
- Can change the initial segment of the paths on the frontier to use the shorter path



Multiple-Path Pruning & Optimal Solutions

- Problem: what if a subsequent path to *n* is shorter than the first path to n, and we want just the optimal solution ?

- Can prove that this can't happen for an algorithm



- Which of the following algorithms always find the shortest path to nodes on the frontier first?

Least Cost Search First

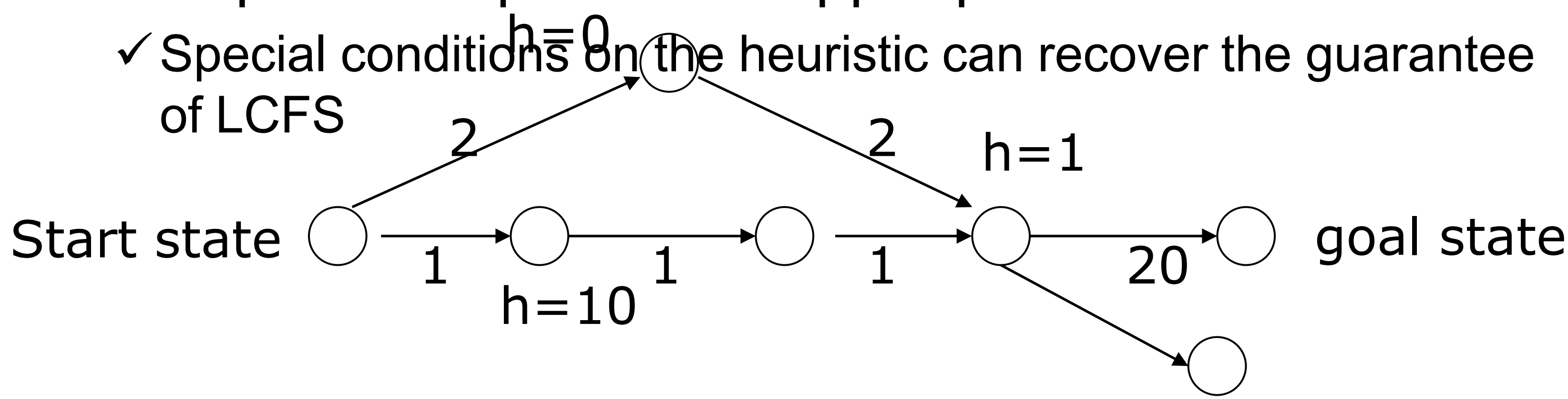
A*

Both of the above

None of the above

- Which of the following algorithms always find the shortest path to nodes on the frontier first?
  - Only Least Cost First Search (like Dijkstra's algorithm)
  - For A* this is only guaranteed for nodes on the optimal solution path

  - Example: A* expands the upper path first
    - ✓ Special conditions on the heuristic can recover the guarantee of LCFS



Summary: pruning

- Sometimes we don't want pruning
  - Actually want multiple solutions (including non-optimal ones)
- Search tree can be exponentially larger than search space
  - So pruning is often important
- In DFS-type search algorithms
  - We can do cheap cycle checks: O(1)
- BFS-type search algorithms are memory-heavy