Reasoning with Constraints

Instead of reasoning explicitly in terms of states,

it is typically better to describe states in terms of features and to reason in terms of these features.

Features are described using variables.

Often features are not independent and there are hard constraints

that specify legal combinations of assignments of values to variables.

exploit constraints to solve tasks

In planning and scheduling an agent assigns a time for each action.

These assignments must satisfy constraints on the order actions can be carried out and

constraints specifying that the actions achieve a goal.

Preferences over assignments are specified in terms of soft constraints.

https://artint.info/2e/html/ArtInt2e.Ch4.S12.html

Variables and Worlds

Constraint satisfaction problems are described in terms of variables and possible worlds.

Possible worlds are described by algebraic variables.

An algebraic variable is a symbol used to denote features of possible worlds.

Algebraic variables are written starting with an upper-case letter.

Each algebraic variable X

has an associated domain,

dom(X), which is the set of values the variable can take.

discrete variable is one whose domain is finite or countably infinite

A binary variable is a discrete variable with two values in its domain

Boolean variable is a binary variable, with domain {true, false}

a variable whose domain corresponds to the real line or an interval of the real line is a continuous variable

.

A Possible World

Given a set of variables, an assignment on the set of variables is a function from the variables into the domains of the variables.

Assignment on {X1, X2 ,…, Xk} is

{X1 = v1, X2 = v2,…,Xk = vk}, where vi is in dom(Xi)

A variable can only be assigned one value in an assignment.

total assignment assigns all of the variables

Assignments may be partial.

A possible world is a total assignment

If world w is the assignment {X1 = v1,X2 = v2,…,Xk = vk} ,

variable Xi has value vi in world w

Ex:

The variable Class_time may denote the starting time for a particular class. The domain of Class_time may be the following set of possible times

Dom (Class_time) = {8,9, 10, 11, 12, 1, 2, 3, 4, 5}

Assignment and possible world

variable Height_joe may refer to the height of a particular person at a particular time and

have as its domain the set of real numbers, in some range, that represent the height in centimeters.

Raining may be a random variable with

domain {true, false} , which has value true if it is raining at a particular time.

Assignment

{ Class_time = 11, Height_joe = 165, Raining = false}

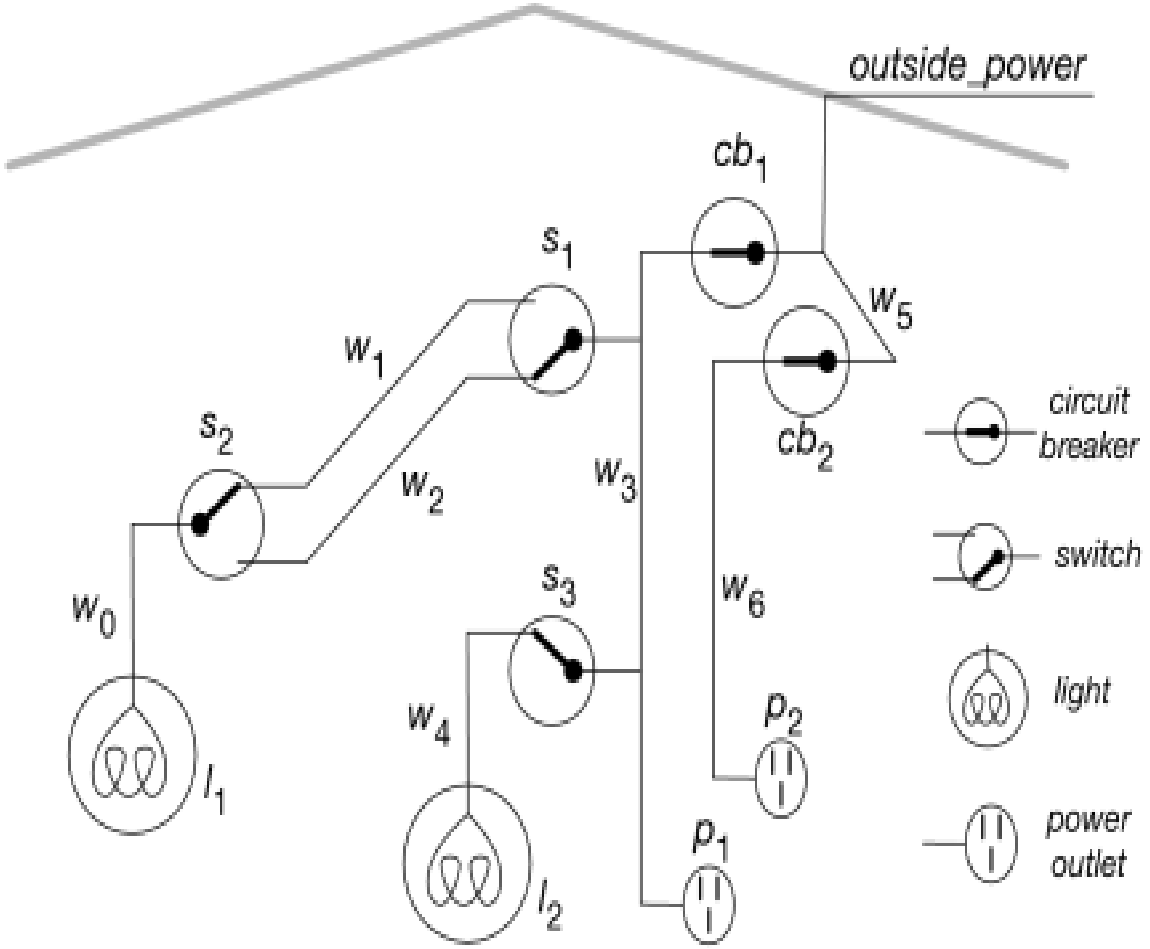
specifies that the class starts at 11, Joe is 165cm tall and it is not raining

An electrical environment for the diagnostic assistant

a variable for the position of each switch that specifies whether the switch is up or down.

a variable for each light that specifies whether it is lit or not.

a variable for each component specifying whether it is working properly or if it is broken



An electrical environment for the diagnostic assistant

- S1_pos *is a binary variable denoting the position of switch* s1 *with*

  domain {up, *down*},

  where S1_pos = up *means switch* s1 *is up, and* S1_pos=down *means switch* s1 *is down.*
- S1_st *is a discrete variable denoting the status of switch* s1 *withdomain* {ok, upside_down, short, intermittent, broken}

  where S1_st = ok *means switch* s1 *is working normally*

  S1_st = upside_down *means it is installed upside down*

  S1_st = short *means it is shorted and*

  it allows electricity to flow whether it is up or down

  S1_st = intermittent *means it is working intermittently*

  S1_st=broken *means it is broken and does not allow electricity to flow.*
- *Number_of_broken_switches is an integer-valued variable denoting the number of switches that are broken.*
- Current_w1 *is a real-valued variable denoting the current, in amps, flowing through wire* w1.
- Current_w1=1.3 *means there are* 1.3 *amps flowing through wire* w1w1.
- *Inequalities between variables and constants are allowed,*
- *(example,* Current_w1≥1.3 *is true when there are at least* 1.3 *amps flowing through wire* w1

An electrical environment for the diagnostic assistant

A world specifies the position of every switch, the status of every device, and so on.

Example:

a world may be described as switch 1 is up, switch 2 is down, fuse 1 is okay, wire 3 is broken, etc.

Crossword puzzle

There are two different representations of crossword puzzles in terms of variables:

- representation1:

the variables are the numbered squares with the direction of the word (down or across), and

the domains are the set of possible words that can be used.

Example: one_across could be a variable with domain {{'ant', 'big', 'bus', 'car', 'has'}}

A possible world corresponds to an assignment of a word for each of the variables.

- representation2:

the variables are the individual squares and

the domain of each variable is the set of letters in the alphabet.

Example: the top-left square could be a variable p00 with domain {a,…,z}{a,…,z}.

A possible world corresponds to an assignment of a letter to each square.

Possible Worlds

number of worlds is the product of the number of values in the domains of the variables.

Ex:

If there are two variables, A with domain {0,1,2} and B with domain {true , false}, there are six possible worlds:

w_0 = {A = 0, B = true}

w1 = {A = 0, B = false}

w2 = {A = 1, B = true}

w3 = {A = 1, B = false}

w4 = {A = 2, B = true}

w5 = {A = 2, B = false}

If there are n variables, each with domain size d, there are d^n possible worlds

main advantage of reasoning in terms of variables is the computational savings.

Many worlds can be described by a few variables

10 binary variables can describe 2^{10} = 1,024210 = 1,024 worlds

30 binary variables can describe 2^{30} = 1,073,741,824230 = 1,073,741,824 worlds

Constraints

In many domains, not all possible assignments of values to variables are permissible.

A hard constraint, or simply constraint,

specifies legal combinations of assignments of values to some of the variables.

A scope is a set of variables

A relation on a scope S is a function from assignments on S to {true , false}

it specifies whether each assignment is true or false.

A constraint c is a scope S and a relation on S.

A constraint is said to involve each of the variables in its scope.

A constraint can be evaluated on any assignment that extends its scope.

Consider constraint c on S.

Assignment A on S′, where $S \subseteq S'$ satisfies c if A, restricted to S, is mapped to true by the relation.

Otherwise, the constraint is violated by the assignment

Constraints and a Possible World

A possible world w **satisfies** a set of constraints if,

for every constraint,

the values assigned in w to the variables in the scope of the constraint

satisfy the constraint.

In this case, the possible world is a **model** of the constraints. or

a **model** is a possible world that satisfies all of the constraints.

Constraints are defined either by their intension, in terms of formulas, or

by their extension, listing all the assignments that are true.

unary constraint is a constraint on a single variable (e.g., $B \leq 3$).

binary constraint is a constraint over a pair of variables (e.g., $A \leq B$).

k-ary constraint has a scope of size k.

Example:

$A+B=C$ is a 3-ary (ternary) constraint

Constraints and a Possible World

Example:

A robot needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.

Each activity has a set of possible times at which it may start.

The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.

For each activity there is a variable that represents the time that it starts.

D represents the start time for the drilling, B start time of the bolting, and C start time for the casting.

Some constraints are -

drilling must start before bolting,

which translates into the constraint $D < B$.

casting and drilling must not start at the same time,

this corresponds to the constraint $C \neq D$.

bolting must start exactly 3 time units after casting starts,

this corresponds to the constraint $B = C + 3$

Constraints and a Possible World

Consider a constraint on the possible dates for three activities.

Let A, B, and C be variables that represent the date of each activity.

Suppose the domain of each variable is {1,2,3,4}

A constraint with scope {A,B,C} could be described by its **intension**,

using a logical formula to specify the legal assignments, such as

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C \leq 3)$$

This formula says that A is on the same date or before B, and B is before day 3, B is before C, and it cannot be that A and B are on the same date and C is on or before day 3

This constraint could instead have its relation defined its extension, as a table specifying the legal assignments:

The first assignment is {A=2,B=2,C=4},

which assigns A *the value 2,* B *the value 2, and* C *the value 4.*

There are four legal assignments of the variables.

The assignment {A=1,B=2,C=3,D=3,E=1} satisfies this constraint

because that assignment restricted to the scope of the relation,

namely {A=1,B=2,C=3}{A=1,B=2,C=3}, is one of the legal assignments in the table

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

Constraints and a Possible World

The assignment {A=1,B=2,C=3,D=3,E=1} satisfies this constraint because that assignment restricted to the scope of the relation,

{A=1,B=2,C=3} is one of the legal assignments in the table

Defining CSP

A factored representation for each state - a set of variables, each of which has a value is used

A problem is solved when each variable has a value that satisfies all the constraints on the variable.

A problem described this way is called a constraint satisfaction problem, or CSP.

CSP search algorithms take advantage of the structure of states and use general-purpose rather than problem specific heuristics to enable the solution of complex problems.

The main idea is to eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints.

Defining CSP

A constraint satisfaction problem consists of three components, X,D, and C:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i.

Each constraint C_i consists of a pair <scope, rel >,

where scope is a tuple of variables that participate in the constraint and

rel is a relation that defines the values that those variables can take on.

A relation can be represented as

an explicit list of all tuples of values that satisfy the constraint, or as

an abstract relation that

supports two operations:

testing if a tuple is a member of the relation and

enumerating the members of the relation.

Solving CSP

Given a CSP, a number of tasks are useful:

- Determine whether or not there is a model.
- Find a model.
- Count the number of models.
- Enumerate all of the models.
- Find a best model, given a measure of how good models are.
- Determine whether some statement holds in all the models.

The multidimensional aspect of CSPs, where each variable is a separate dimension,

makes these tasks difficult to solve, but also provides structure that can be exploited.

To solve a CSP, we need to define a state space and the notion of a solution.

Each state in a CSP is defined by an **assignment** of values to some or all of the variables, {Xi = vi,Xj = vj , . . .}.

An assignment that does not violate any constraints is called a **consistent** or legal assignment.

A **complete assignment** is one in which every variable is assigned, and

a solution to a CSP is a consistent, complete assignment.

A **partial assignment** is one that assigns values to only some of the variables.

Solving CSP - Generate-and-Test Algorithms

A finite CSP could be solved by an exhaustive generate-and-test algorithm.

The assignment space, D , is the set of total assignments.

The algorithm returns one model or all models.

The generate-and-test algorithm to find one model - check each total assignment in turn; if an assignment is found that satisfies all of the constraints, return that assignment.

A generate-and-test algorithm to find all models - saves all of the models found.

Example

Suppose the delivery robot must carry out a number of delivery activities, a, b, c, d, and e.

Suppose that each activity happens at any of times 1, 2, 3, or 4.

Let A be the variable representing the time that activity aa will occur, and similarly for the other activities.

The variable domains, which represent possible times for each of the deliveries, are

Dom(A) ={1,2,3,4}, Dom(B) ={1,2,3,4}, Dom(C) ={1,2,3,4}, Dom(D) ={1,2,3,4}, Dom(E) ={1,2,3,4},

Constraints

{ $(B \neq 3)$, $(C \neq 2)$, $(A \neq B)$, $(B \neq C)$, $(C < D)$, $(A = D)$, $(E < A)$, $(E < B)$, $(E < D)$, $(B < D)$ }

Find a Model:

try to assign a value to each variable that satisfies these constraints.

Generate-and-Test Algorithms

A finite CSP could be solved by an exhaustive generate-and-test algorithm.

The assignment space, D, is the set of total assignments.

The algorithm returns one model or all models

Example assignment space for delivery robo:

D = { {A=1, B=1, C=1, D= 1, E=1}, {A=1, B=1, C=1, D= 1, E=2}, … , {A=4, B=4, C=4, D= 4, E=4} }

Number of assignments $|D| = 4^5$.

If each of the n variable domains has size d,

$|D| = d^n$ elements.

If there are e constraints

the total number of constraints tested is $O(ed^n)$

Solving CSPs Using Search

Generate-and-test algorithms assign values to all variables before checking the constraints.

individual constraints only involve a subset of the variables, some constraints can be tested before all of the variables have been assigned values.

If a partial assignment is inconsistent with a constraint, any total assignment that extends the partial assignment will also be inconsistent.

assignments A=1 and B=1 are inconsistent with the constraint A≠B

regardless of the values of the other variables.

If the variables A and B are assigned values first,

this inconsistency can be discovered before

any values are assigned to C, D, or E, saving a large amount of work.

Solving CSPs Using Search

construct a search space for the search strategies

- nodes are assignments of values to some subset of the variables.
- neighbors of a node n are obtained by selecting a variable Y that is not assigned in node n and
  by having a neighbor for each assignment of a value to Y that does not violate any constraint.

CSP with the variables A, B, and C each with domain {1,2,3,4)

constraints A<B and B<C

Search tree for the CSP

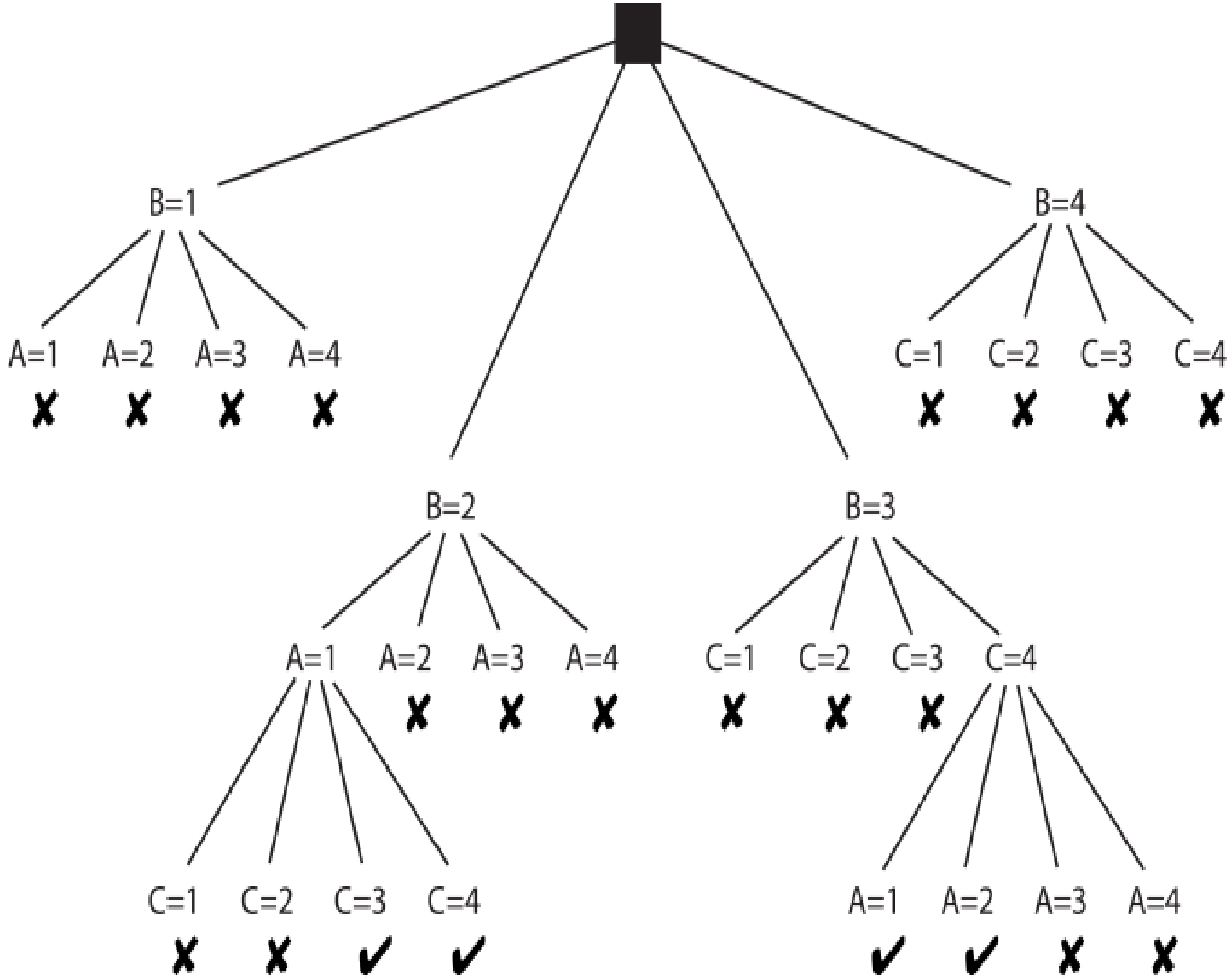
a node corresponds to all of the assignments from the root to that node.

The potential nodes that are pruned because they violate constraints are labeled with ✗.

CSP has four solutions.

inefficiency in back tracking

A = 4 inconsistency is rediscovered for different assignments of B, C.



Solving CSPs Using Search

size of the search tree, and thus the efficiency of the algorithm, depends on which variable is selected at each time.

static ordering: such as always splitting on A then B then C , is usually less efficient than

the dynamic ordering used here,

but it might be more difficult to find the best dynamic ordering than to find the best static ordering.

The set of answers is the same regardless of the variable ordering.

There would be

$4^3 = 64$

total assignments tested in a generate-and-test algorithm.

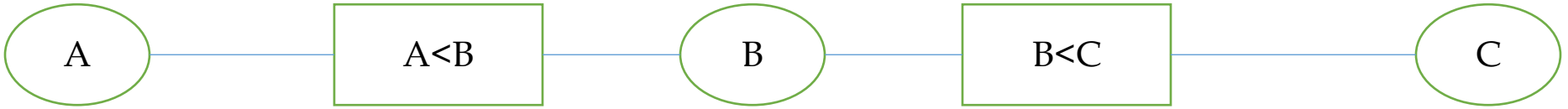
For the search method,

there are 8 total assignments generated, and 16 other partial assignments that were tested for consistency.

Searching this tree with a depth-first search/ backtracking, can be much more efficient than generate and test. Generate and test is equivalent to not checking constraints until reaching the leaves. Checking constraints higher in the tree can prune large subtrees that do not have to be searched.

Constraint graph

Improving efficiency in back tracking using constraints fro pruning domains of variables



Node Consistency/ Domain Consistency

Dom(A) = {1, 2, 3, 4}; c: < A, A ≠ 3> not domain consistent

Arc Consistency

Dom(B) = {1,2,3,4}; c: <(A, B), A<B> for A =4 , no value for B exists in domain of B . Not Arc consistent

Network consistent: all arcs in the network are consistent.

Generalized arc consistency algorithm

Consistency algorithms operate over network of constraints formed by CSP

Propagate arc consistencies to make network arc consistent

All arcs in to-do-arcs list

to-do-arcs:

if not arc consistent remove domain variable and make it arc consistent

if any of previous consistent arcs are now inconsistent, due to removal of domain variable

place them back in the list with new constraint

remove consistent arc from list

Arc Consistency Algorithm

All arcs in to-do list

1. < A, (A< B) > constraint => A= 4 is inconsistent with every possible assignment of B (dom(B) = {1,2,3,4}.

   Remove this value from dom(A). (inefficiency of back tracking search eliminated)

   dom(A) = {1, 2, 3}

2. < B, (A<B) > B = 1 pruned from B — no arc is added to to-do

   dom(B) = {2,3,4}

3. < B, (B<C) > B = 4 pruned — B domain reduced

   dom(B) = {2,3} — <A, (A < B) > added back in to-do

4. <A, (A< B) > A = 3 pruned

   dom(A) = {1,2}

5 remaining arc

< C, (C > B) > C = 1, C = 2 removed — no arc is added as C not involved in other constraints

dom(C) = {3, 4}

to-do empty.

D(A) = {1, 2}, D(B) = {2, 3}, D(C) = {3, 4} - problem simplified. Not solved yet. Can use search now

Arc Consistency Algorithm + Search

D(A) = {1, 2}, D(B) = {2, 3}, D(C) = {3, 4} - problem simplified. Not solved yet. Can use search now

Can use Domain Split/ Case Analysis/ Cut Set

Split the problem into disjoint cases and solve each separately. Combine solutions

B = 2 ➔ A = 2 is pruned ; B = 3 ➔ C = 3 is pruned

C = 3 ➔ B = 3 is pruned

CSP Solving:

1. Use arc consistency & simplify problem
2. Split domain for a variable
3. Solve recursively for each case

Search tree with arc consistency is much smaller.

Structure of the problem, Variable Elimination

Use constraint graph to figure out

independent sub problems / disjoint sets of variables

Can use topological sort of variables, convert/reduce general constrained graphs into trees

Variable elimination: B is implicitly satisfied with A, C domain derived.

A, B, C domain = {1,2,3,4} other variables D, E, F, …

Constraints involving B : A<B, B <C. B does not have constraints in common with D, E, F

A	B
1	2
1	3
1	4
2	3
2	4
3	4

⋈

B	C
1	2
1	3
1	4
2	3
2	4
3	4

=

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

=

A	C
1	3
1	4
2	4

Example

Map Coloring

map of Australia showing each of its states and territories →

We are given the task of coloring each region either red, green, or blue

in such a way that no neighboring regions have the same color.

To formulate this as a CSP,

X = {WA,NT,Q,NSW, V,SA, T}

Di = {red , green, blue}.

constraints require neighboring regions to have distinct colors. Since there are nine places where regions border,

nine constraints:

C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V } .

SA ≠ WA → <(SA,WA),SA ≠ WA>, where

SA = WA can be fully enumerated in turn as

{(red , green), (red , blue), (green, red), (green, blue), (blue, red), (blue, green)} .

many possible solutions to this problem, such as

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

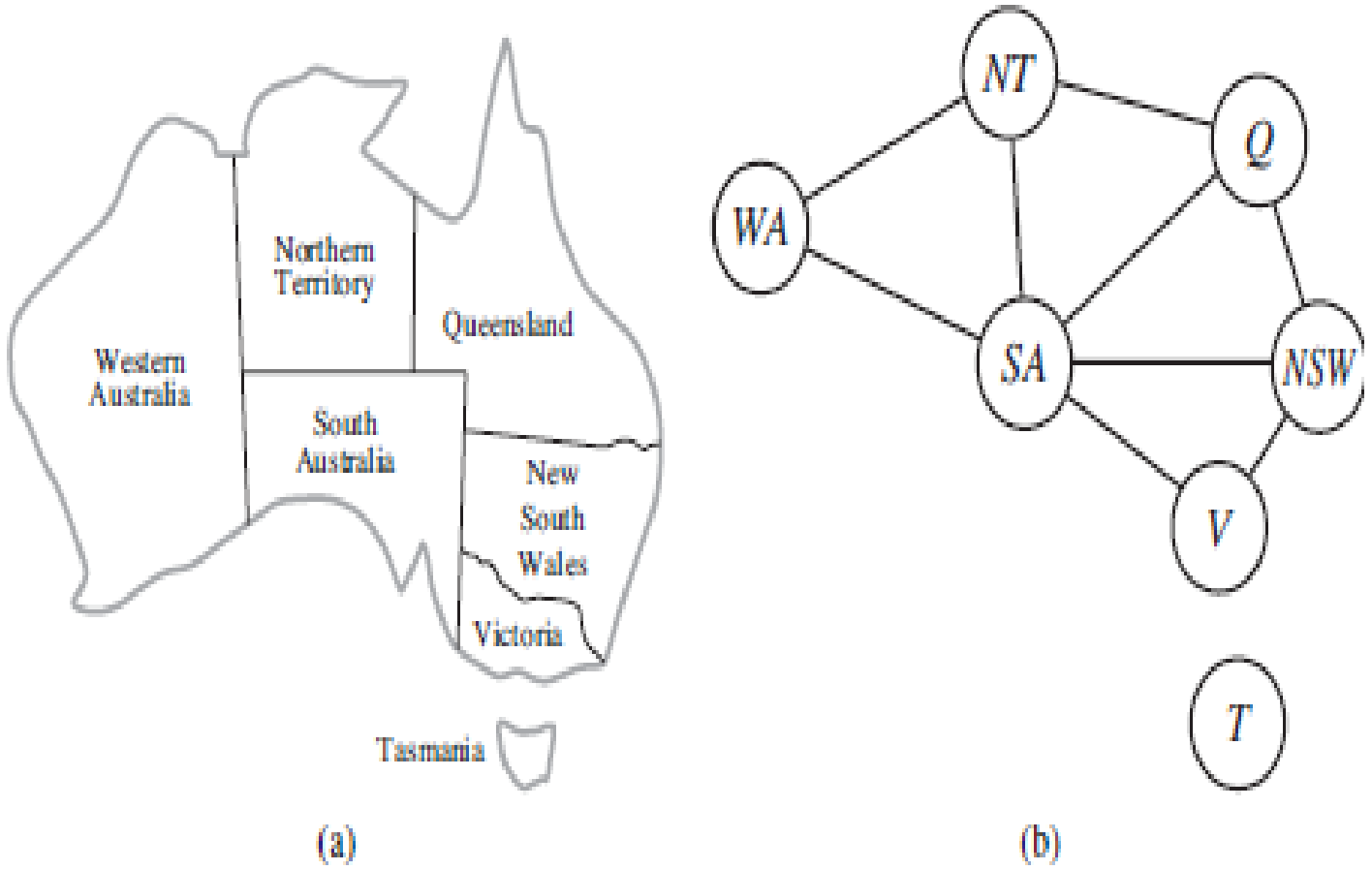
Example

(a) The principal states and territories of Australia.

Coloring this map can be viewed as a constraint satisfaction problem (CSP).

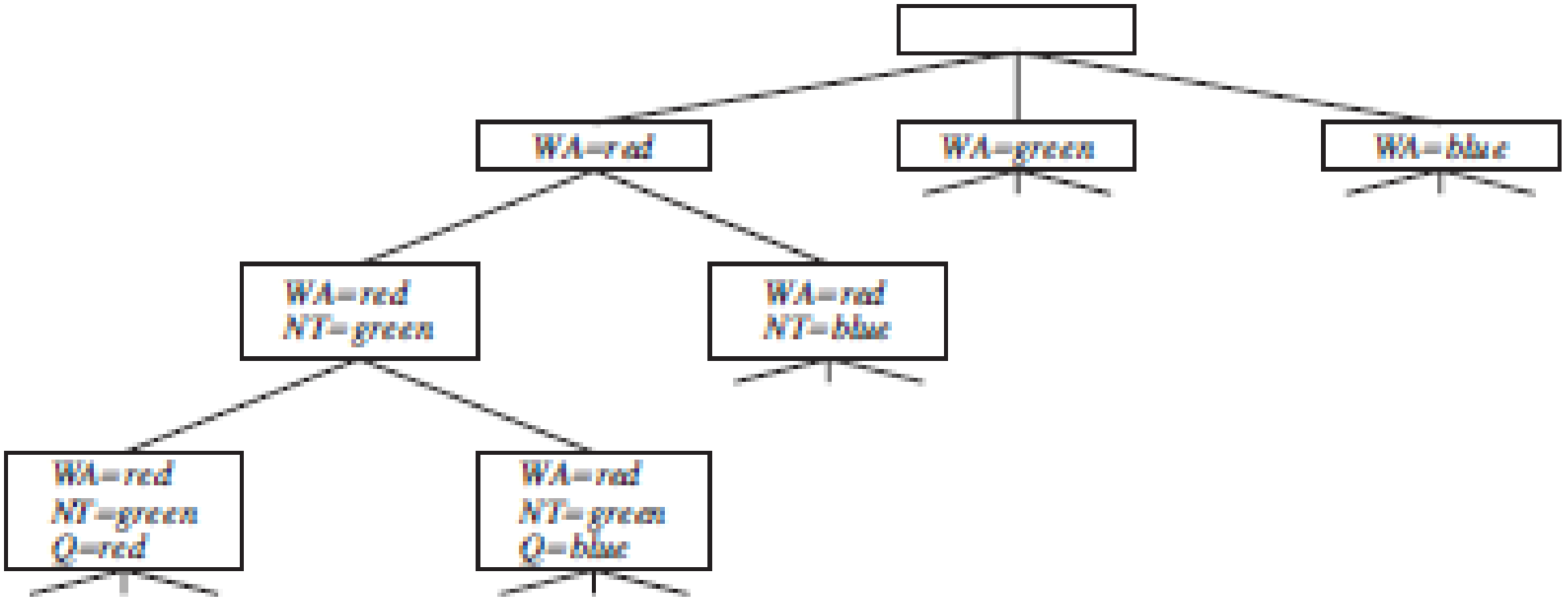
The goal is to assign colors to each region so that no neighboring regions have the same color.

(b) The map-coloring problem represented as a constraint graph.



Search by DFS/Back tracking

Search tree



Constraint propagation

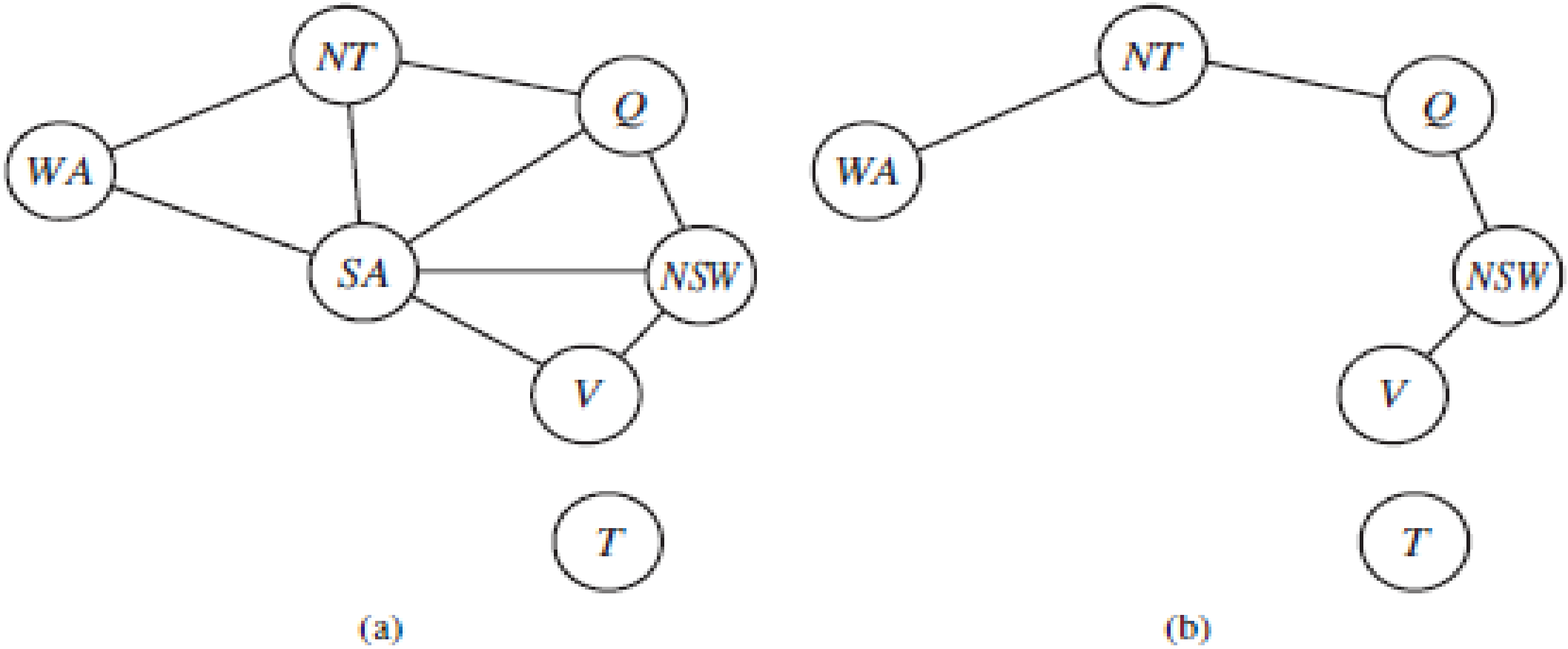
SA = blue, SA has 5 neighbors, remove blue from other domains. - constraint propagation

Assignments for neighbors $2^5 = 32$

without constraint propagation

$3^5 = 243$ assignments

Tree CSP Solver



(a) The original constraint graph.

(b) The constraint graph after the removal of SA.

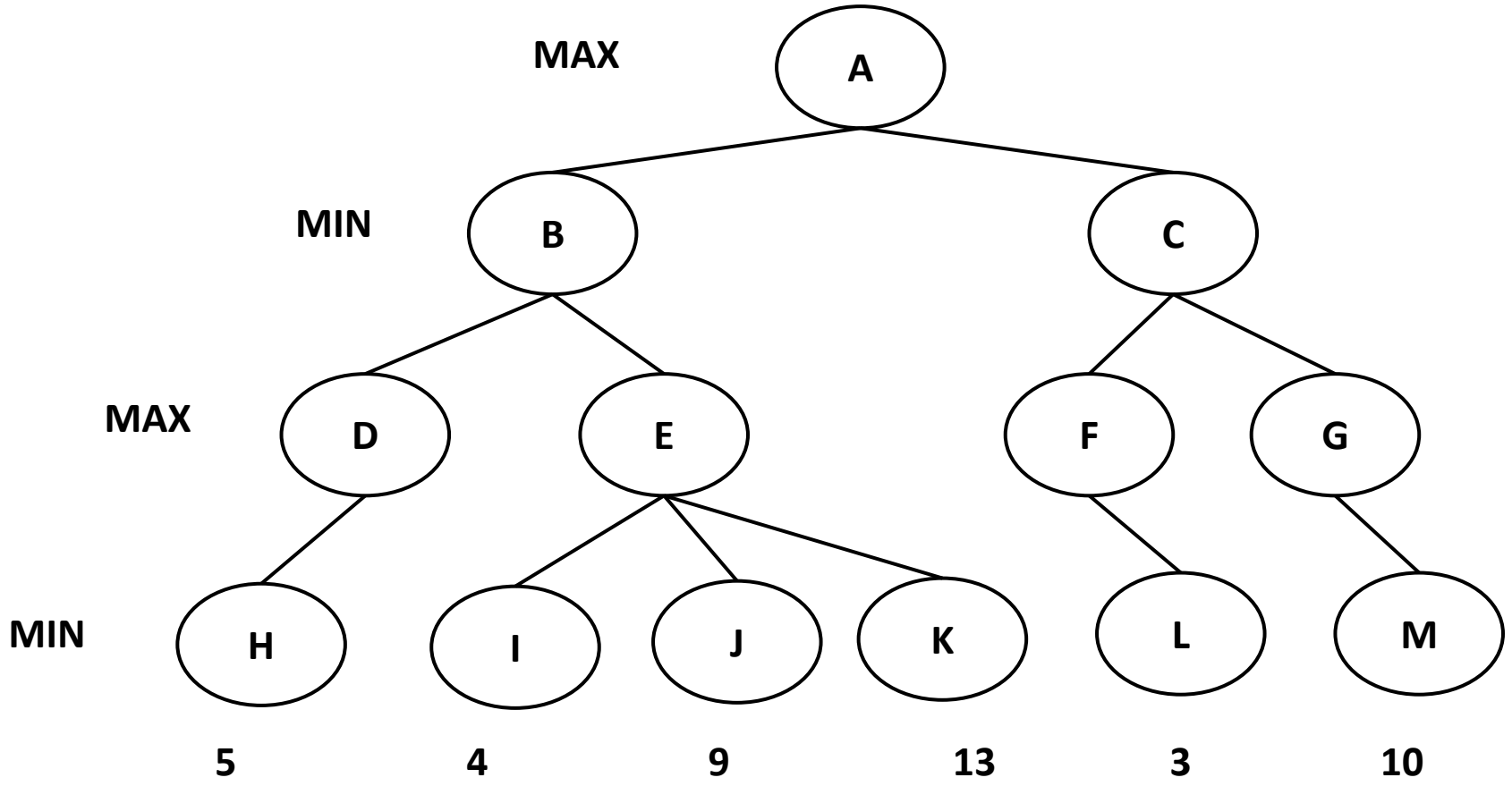
Homework

Pose and solve the crypt-arithmetic problem SEND+MORE=MONEYSEND+MORE=MONEY as a CSP. In a crypt arithmetic problem, each letter represents a different digit, the leftmost digit cannot be zero (because then it would not be there), and the sum must be correct considering each sequence of letters as a base ten numeral.

In this example, you know that Y=(D+E)mod10Y=(D+E)mod10 and that

E=(N+R+((D+E)÷10))mod10E=(N+R+((D+E)÷10)) mod10, and so on.

Homework



Homework


Northern
Territory
Queensland
Western
Australia
South Australia
North South wales
Victoria
Tasmania

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	g	20	21	22	23	24
25	26						32
33		35	36	37	38	39	40
41	42		s	45	46		48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	g	20	21	22	23	24
25	26						32
33		35	36	37	38	39	40
41	42		s	45	46		48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64