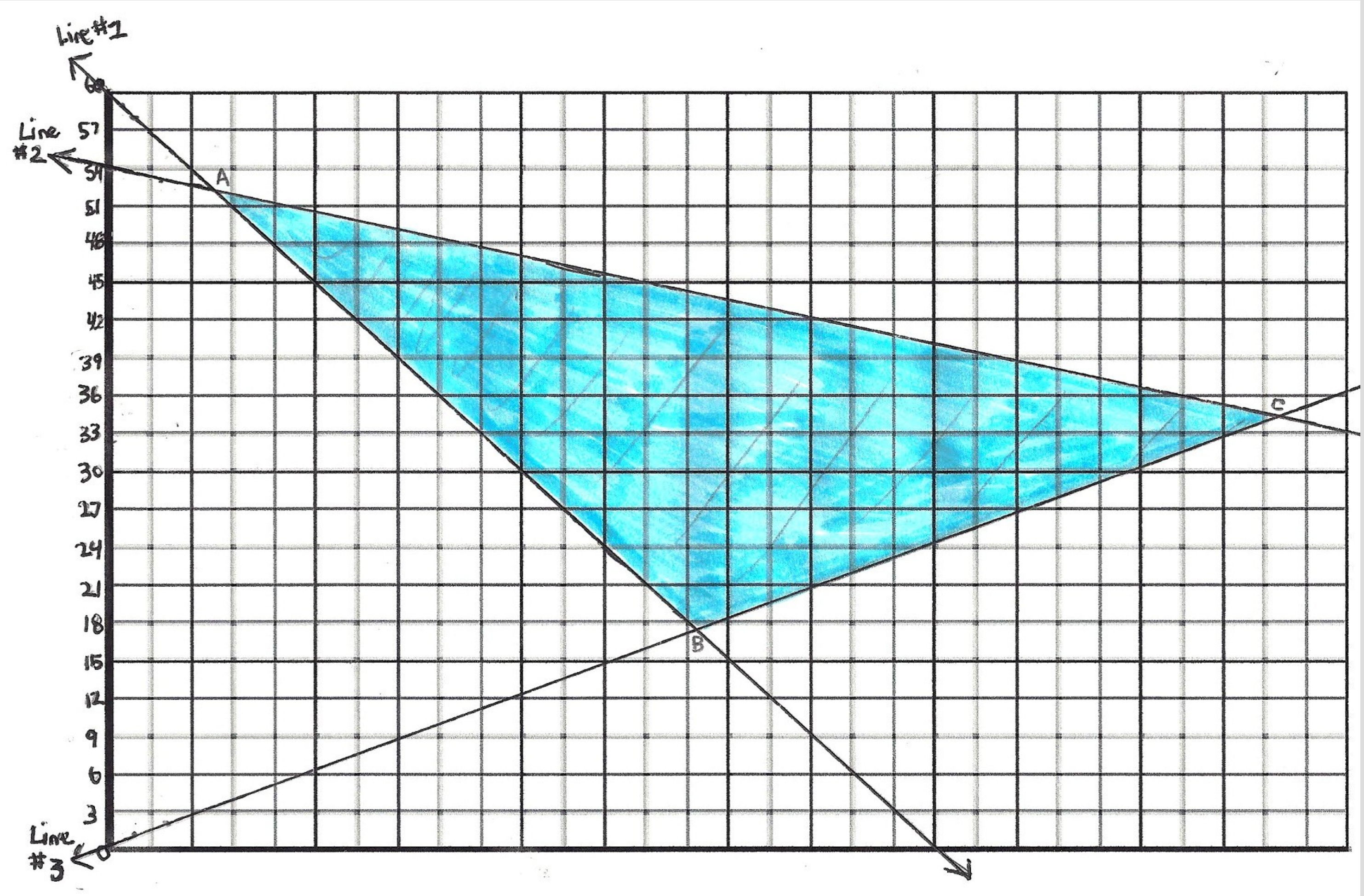
Constraint sat. prob. (Ch. 6)



Announcements

Midterm graded

CSP backtracking

However, this is still hope for searching (called <u>backtracking search</u> (it backups up at conflict))

We will improve it by...

1. The order we pick variables
2. The order we pick values for variables
3. Mix search with inference
4. Smarter backtracking

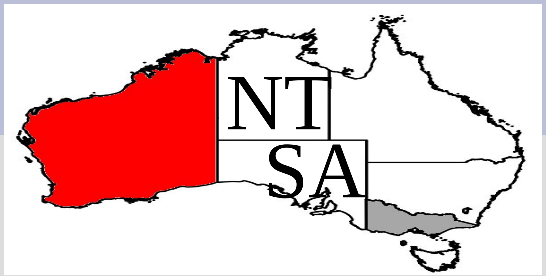
1. What variable?

When picking the variables, we want to the variable with the smallest domain (the most restricted variable)

The best-case is that there is only one value in the domain to remain consistent

By picking the most constrained variables, we fail faster and are able to prune more of the tree

1. What variable?

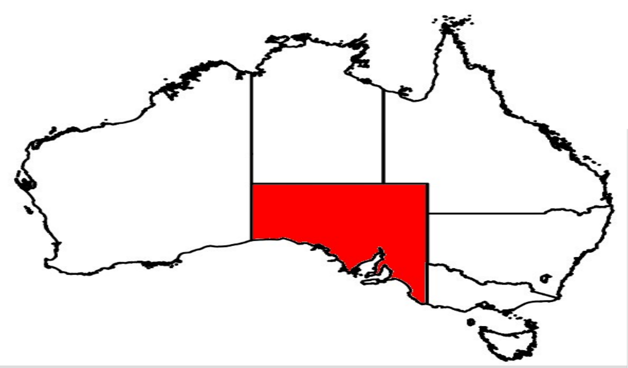


Suppose we pick {WA = red}, it would be silly to try and color V next

Instead we should try to color NT or SA, as these only have 2 possible colorings, while the rest have 3

This will immediately let the computer know that it cannot color NT or SA red (prune these branches right way)

1. What variable?



But we can do even better!

If there is a tie for possible values to take, we pick the variable with the most connections

This ensures that other nodes are more restricted to again prune earlier

For example, we should color SA first as it connects to 5 other provinces

2. What value?

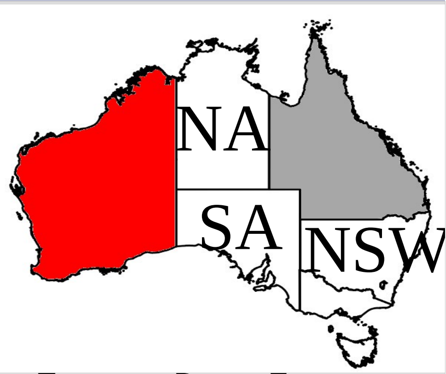
After we picked a variable to look at,
we must assign a value

Here we want to do the opposite: choose the value which constrains the neighbors the <u>least</u>

This is "putting your best foot forward" or trying your best to find a goal (while failing fast helps pruning, we do actually want to find a goal not prune as much as possible)

2. What value?

For example, if we color {WA = red} then pick Q next



Our options for Q are {red, green or blue}, but picking {green or blue} limit NT & SA to only one valid color and NSW to 2

If we pick {Q=red}, then NT, SA & NSW all have 2 valid possibilities (and this happens to be on a solution path)

1. & 2.

An analogy to 1&2 is: “trying our best (2) to solve the weakest link (1)”

By tackling the weakest link first, it will be easier for less constrained nodes to adapt/ pick up the slack

However, we do want to try and solve the problem, not find the quickest way to fail (i.e. always picking blue... ... >.<)

3. Mix search & inference?

We described how AC-3 can use inference to reduce the domain size

Inference does not need to run in isolation;
it works better to assign a value then apply inference to prune before even searching

This works well in combination with 1 as uses the domain size to choose the variable and 3 shrinks domain sizes to be consistent

3. Mix search & inference?

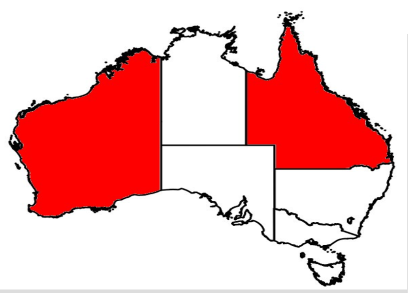
This is somewhat similar to providing
a heuristic for our original search

Inference lets us know an estimation of what colors are left and can be done efficiently

We can use this estimate to guide our search more directly towards the goal

3. Mix search & inference?

In the previous example: {WA = red}, then color Q



We want to choose {Q = red} to allow the most choices for NT and SA

Without inference we could not see this

Instead we would only think all three colors are possible for any uncolored part

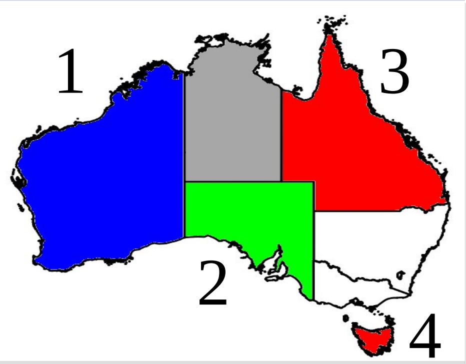
4. Smart backtracking

Instead of moving our search back up a single layer of the tree and picking from there...

We could backup to the first node above the conflict that was actually involved in the conflict

This avoids in-between nodes which did not participate in the conflict

4. Smart backtracking



Suppose we assigned (in this order):
{WA = B, SA = G, Q = R, T = R}
then pick NT

NT has all three colors neighboring it, so a conflict is reached

In normally, we would backtrack and try to change T (i.e. 4), but this was actually not involved in the conflict (1, 2 & 3 were)

Example

Suppose we have the following statement:

```
      T W O
    + T W O
  = F O U R
```

We want to assign each character a single digit to make this a valid math equation (each different letter is a different digit)

How do you represent this as a CSP?

Example

Suppose we have the following statement:

$$
\begin{array}{rcccc}
 & & T & W & O \\
 & + & T & W & O \\
= & F & O & U & R
\end{array}
$$

R = O + O mod 10

U = W + W + floor((O+O)/10) mod 10

O = T + T + floor((W+W/10)) mod 10

F = floor((T+T)/10) mod 10

T ≠ W ≠ O ≠ F ≠ U ≠ R

Example

R = O + O mod 10

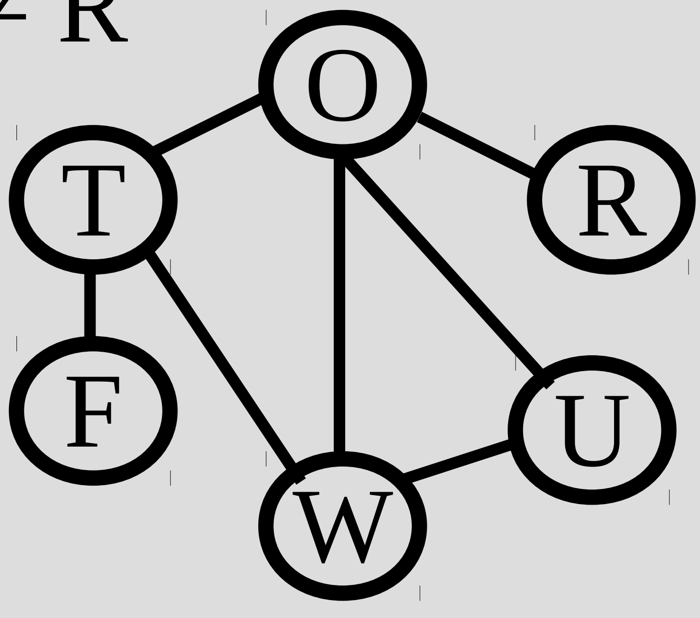
U = W + W + floor((O+O)/10) mod 10

O = T + T + floor((W+W/10)) mod 10

F = floor((T+T)/10) mod 10

T ≠ W ≠ O ≠ F ≠ U ≠ R

Pictorally:
(relationships)



Example

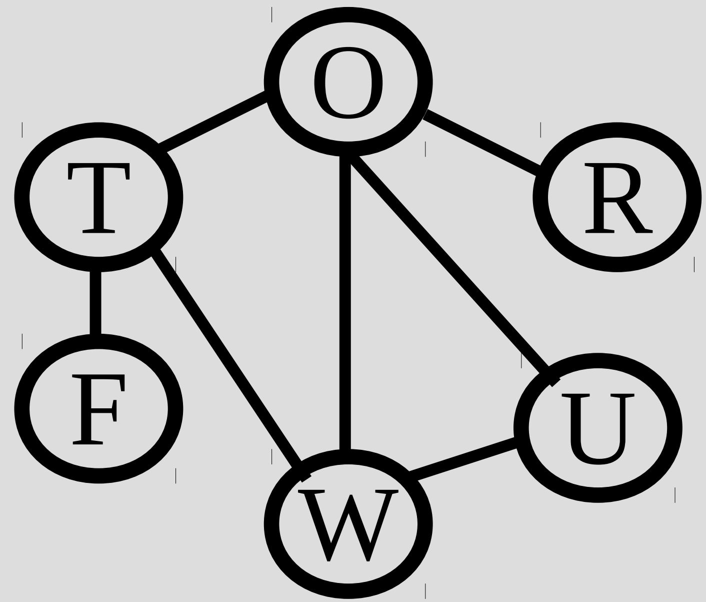
R = O + O (mod 10 on all)

U = W + W + floor((O+O)/10)

O = T + T + floor((W+W/10))

F = floor((T+T)/10)

T ≠ W ≠ O ≠ F ≠ U ≠ R



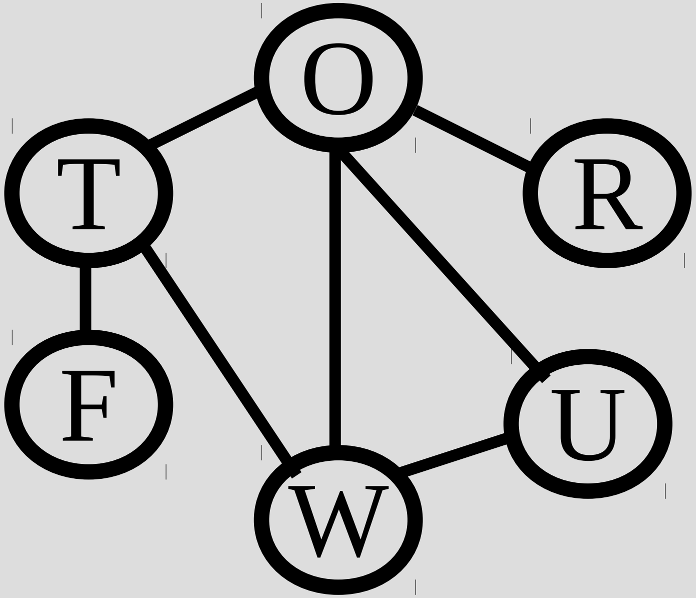
Domains are (as they are digits):

O = R = U = W = {0,1,2,3,4,5,6,7,8,9}

F=T={1,2,3,4,5,6,7,8,9}

(not 0 as leading digit)

Example



R = O + O (mod 10 on all)

U = W + W + floor((O+O)/10)

O = T + T + floor((W+W/10))

F = floor((T+T)/10)

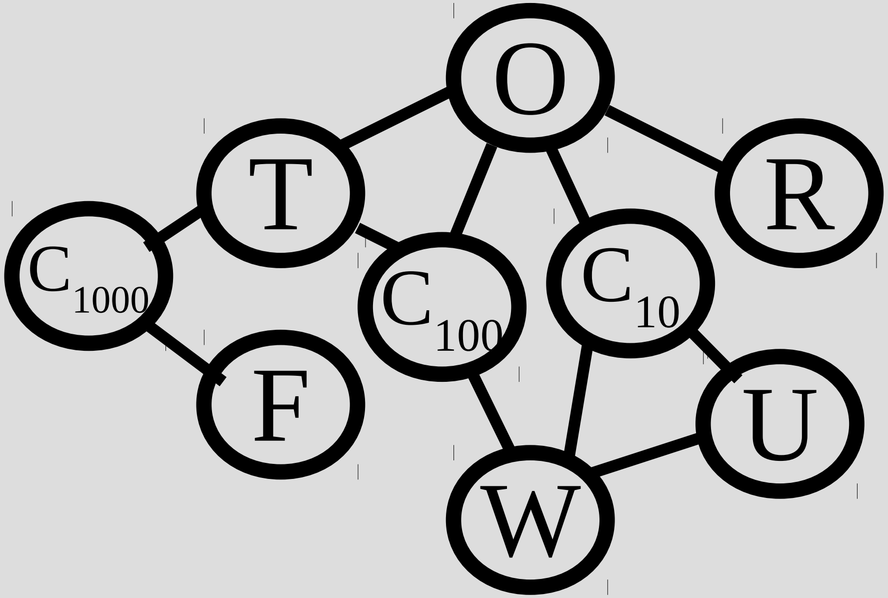
T ≠ W ≠ O ≠ F ≠ U ≠ R

We can simplify the floor by adding auxiliary variables: C_{10}, C_{100} and C_{1000} representing the “carry over” value from the addition

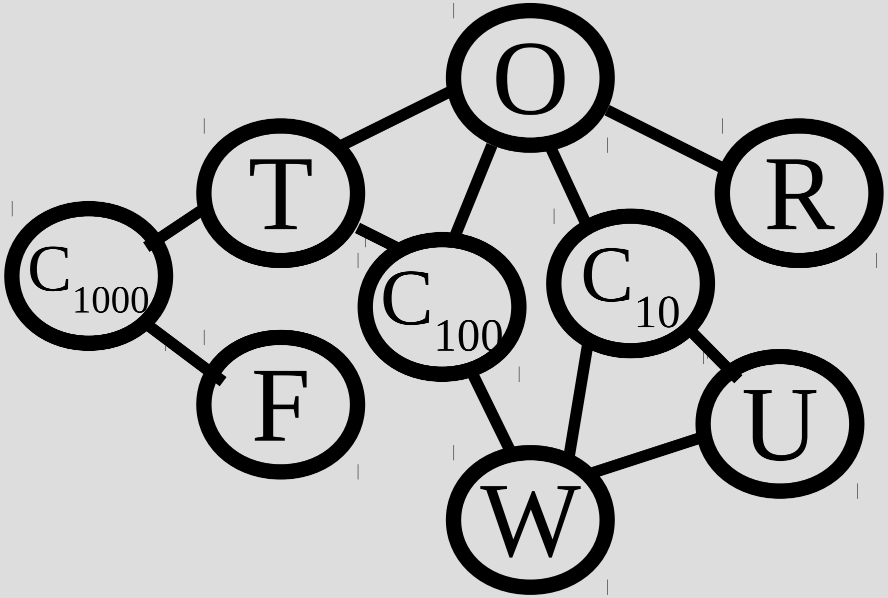
Specifically, floor((O+O/10) = C_{10}

Example

$$R = O + O \bmod 10$$
$$U = W + W + C_{10} \bmod 10$$
$$O = T + T + C_{100} \bmod 10$$
$$F = C_{1000} \bmod 10$$
$$T \neq W \neq O \neq F \neq U \neq R$$
$$C_{10} = \text{floor}((O+O)/10) \bmod 10$$
$$C_{100} = \text{floor}((W+W)/10) \bmod 10$$
$$C_{1000} = \text{floor}((T+T)/10) \bmod 10$$



Example



Domains:

O = R = U = W = {0,1,2,3,4,5,6,7,8,9}

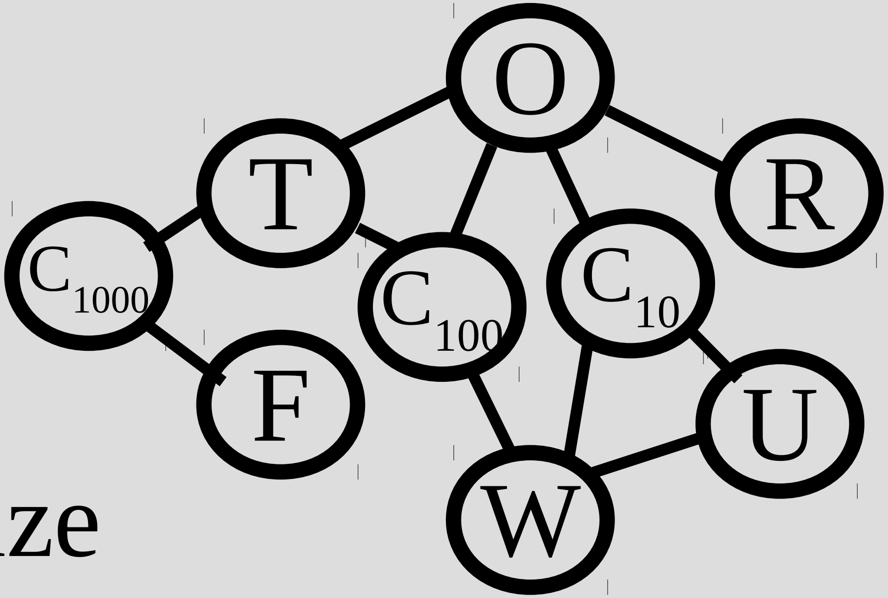
F=T={1,2,3,4,5,6,7,8,9}

$C_{10} = C_{100} = C_{1000} = \{0,1\}$

(as they are the sum of two single digits)

Example

We want to pick the variable with the smallest domain

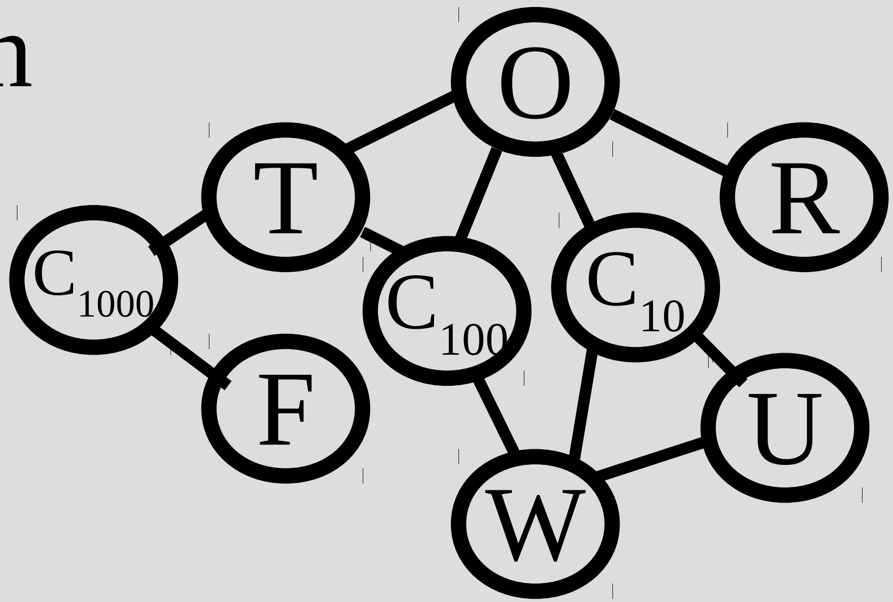


All C_x tie with a domain size of two, so we pick the one with the most connections: Either C_{10} or C_{100} (I will pick C_{10})

So try $C_{10} = 0$

Example

If C_{10} = 0, we see if we can shrink any of the domains that involve C_{10}...



$U = W + W + C_{10} \bmod 10$

$C_{10} = \mathrm{floor}((O+O)/10) \bmod 10$

U and W we cannot shrink, but we can for O: O={0,1,2,3,4}

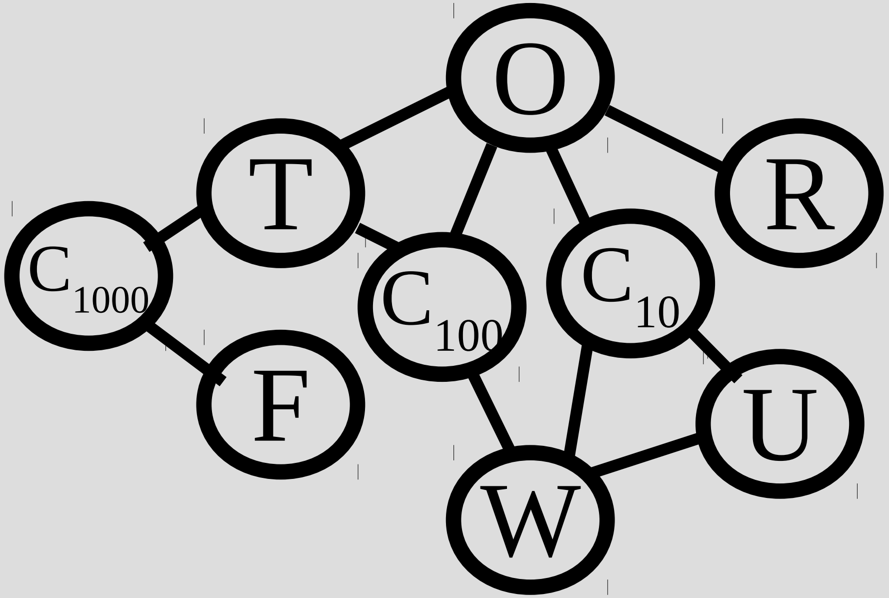
Example

Then pick next:

C_{100} = 0, then infer

W={0,1,2,3,4}

O and T no change



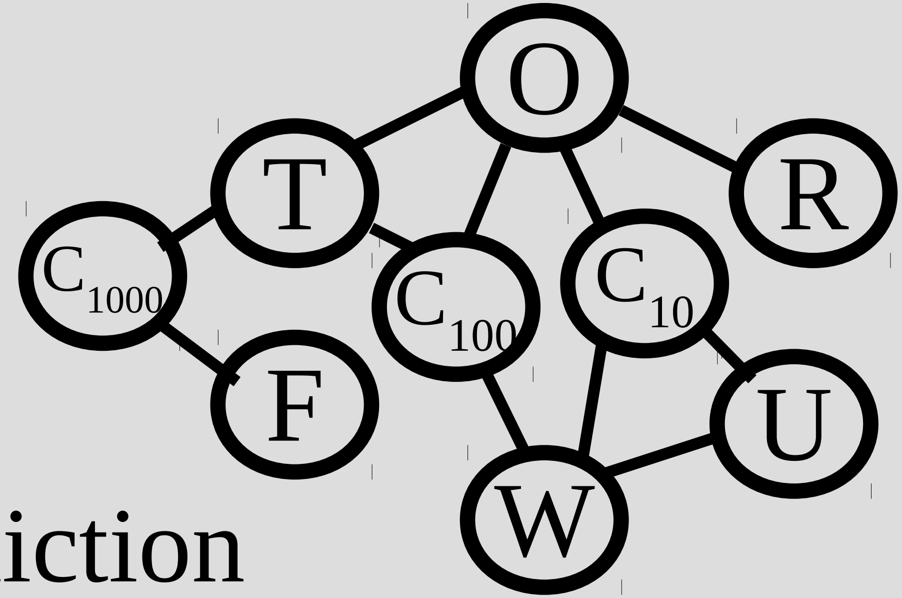
(You could do further inference to reduce U by using MAC inference, but I only shrink domains of things directly related to the pick)

Example

Then pick next:

C_{1000} = 0, then infer

F = { }, a contradiction

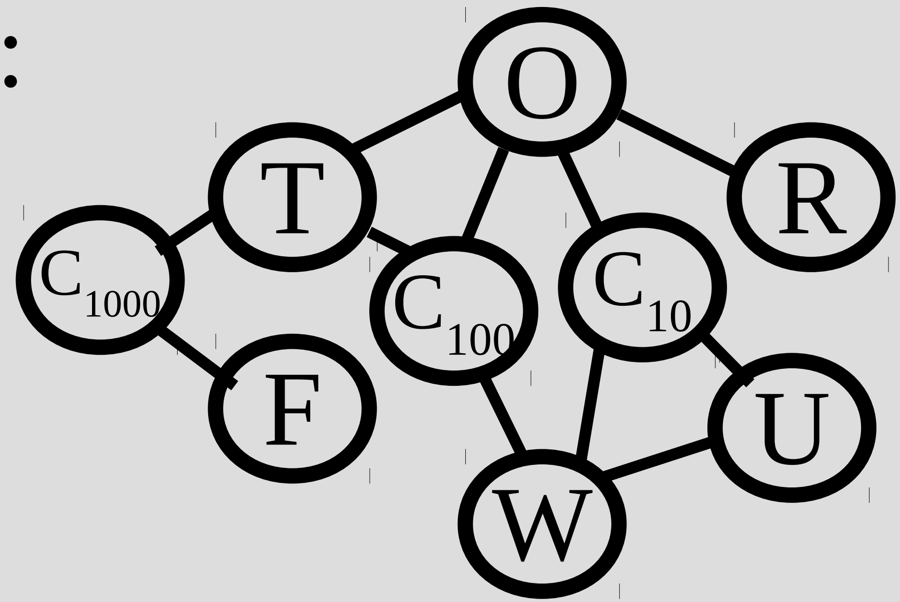


So backup... This contradiction involved C_{1000} and F, so we just need to re-pick C_{1000}, C_{1000}=1

Thus we can infer:

F={1}, T = {5,6,7,8,9}

Example



At this point our picks are:

$C_{10} = 0$

$C_{100} = 0$

$C_{1000} = 1$

Domains:

F = {1}

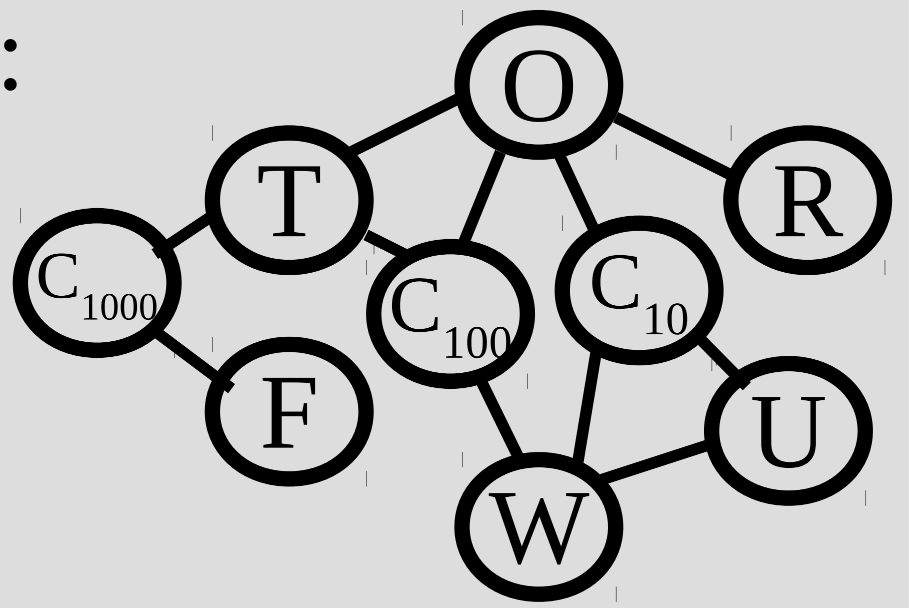
T = {5,6,7,8,9}

W = O = {0,1,2,3,4}

U = R = {0,1,2,3,4,5,6,7,8,9}

Example

Next smallest domain is F:
Only one pick, F=1

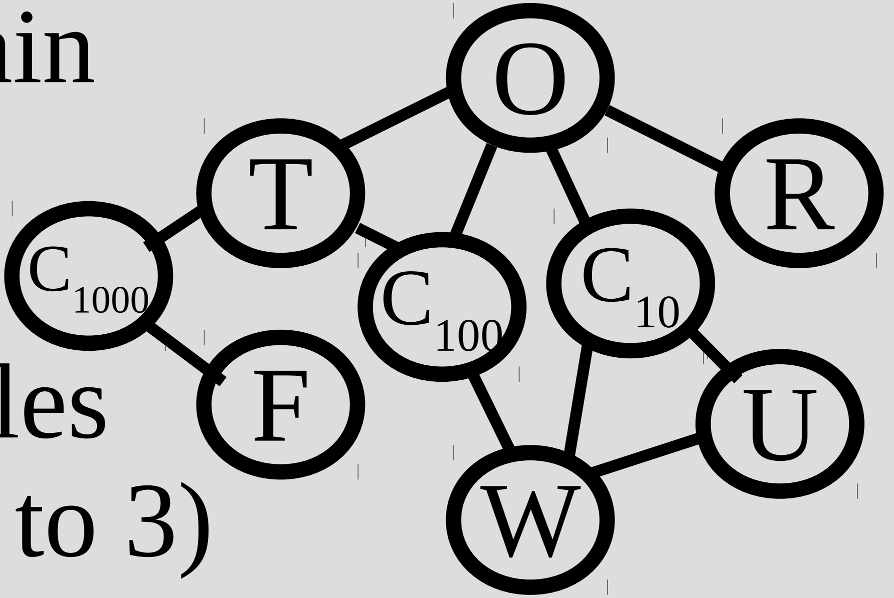


Since F has to be a unique digit we can infer:

W = O = {0,2,3,4}
U = R = {0,2,3,4,5,6,7,8,9}
T unchanged = {5,6,7,8,9}

Example

Tie for next smallest domain between W and O

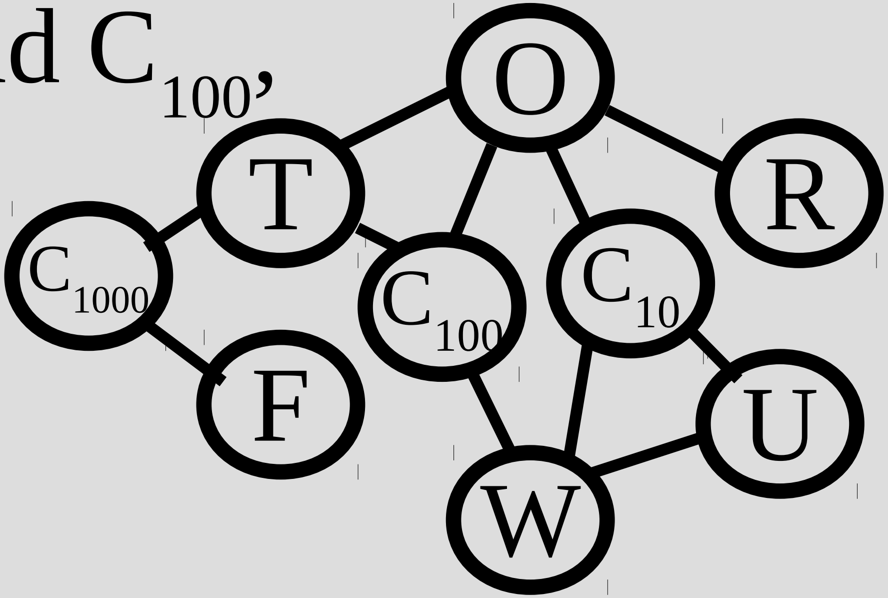


O is connected to 4 variables so pick over W(connected to 3) (other than the “unique” criteria)

Try O=0 and infer:
W = {2,3,4}, R = { } ← Invalid
U = {2,3,4,5,6,7,8,9,}, T={ } ← Invalid

Example

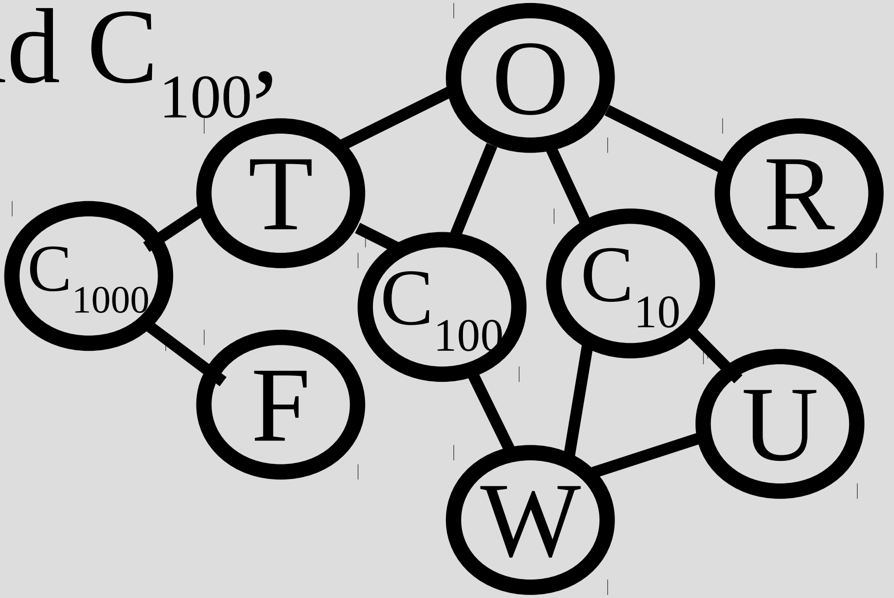
Conflict: T involving O and C_{100},
most recent pick is O



Change to O=2, infer:

T={ } ← Invalid
W = {0,3,4}, R = { 4 }
U = {0,3,4,5,6,7,8,9,}

Example



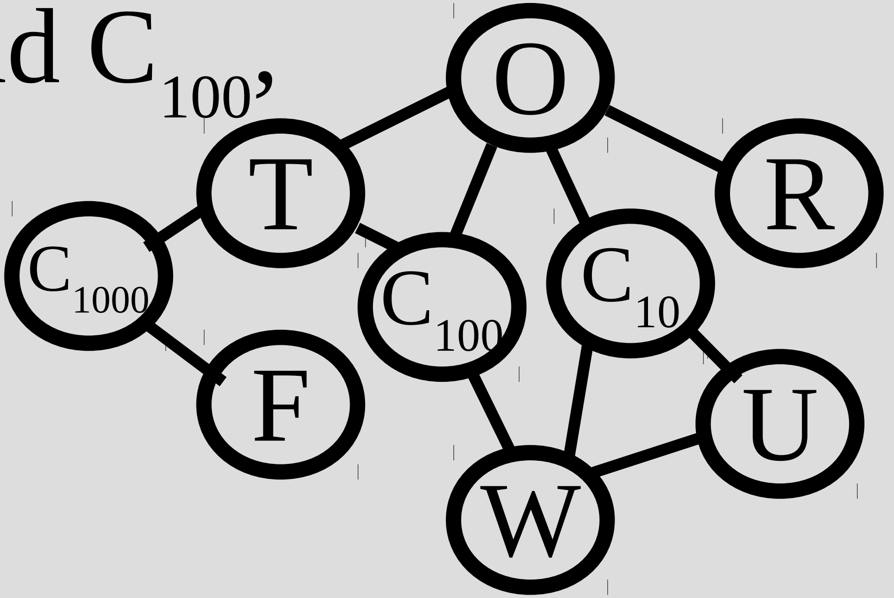
Conflict: T involving O and C_{100},
most recent pick is O

Change to O=3, infer:

T={ } ← Invalid
W = {0,2,4}, R = { 6 }
U = {0,2,4,5,6,7,8,9,}

Example

Conflict: T involving O and C_{100},
most recent pick is O

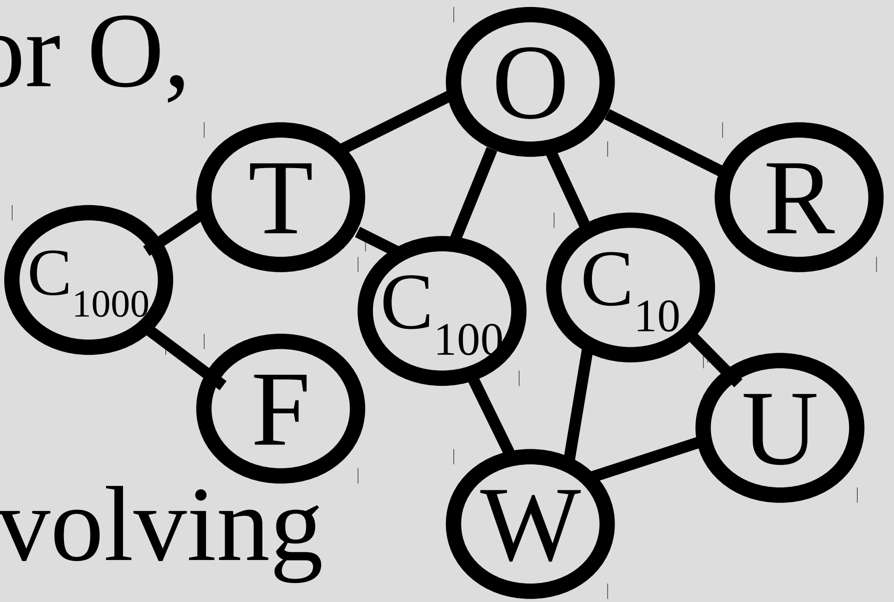


Change to O=4, infer:

T={ } ← Invalid
W = {0,2,3}, R = { 8 }
U = {0,2,3,5,6,7,8,9,}

Example

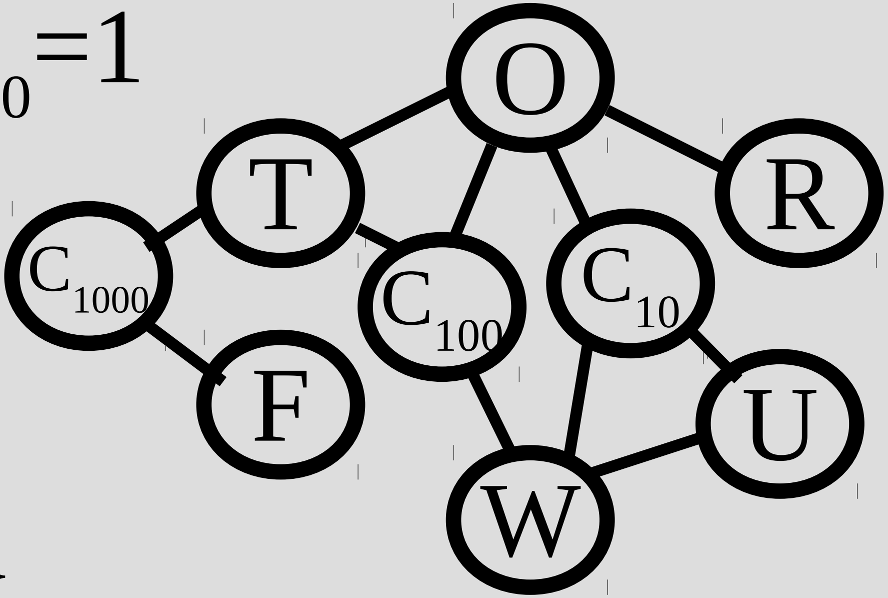
Tried all possible values for O, none worked so we need to backtrack



The conflict was with T involving O and C_{100}, so we will go back and choose $C_{100} = 1$

Example

Currently have: C_{10}=0, C_{100}=1



Domains:

$C_{1000} = \{0, 1\}$

$F = T = \{1,2,3,4,5,6,7,8,9\}$

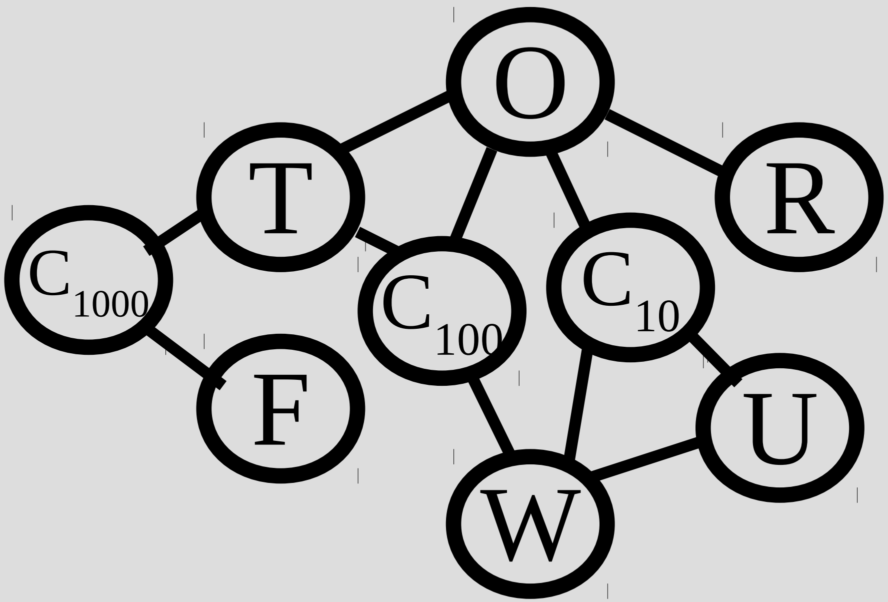
$U = R = \{0,1,2,3,4,5,6,7,8,9\}$

$O = \{0,1,2,3,4\}$, $W = \{5,6,7,8,9\}$

We will again pick C_{1000}=0, conflict, pick C_{1000}=1, pick F=1... just as before

Example

Tie for smallest domain,
O has more connections:

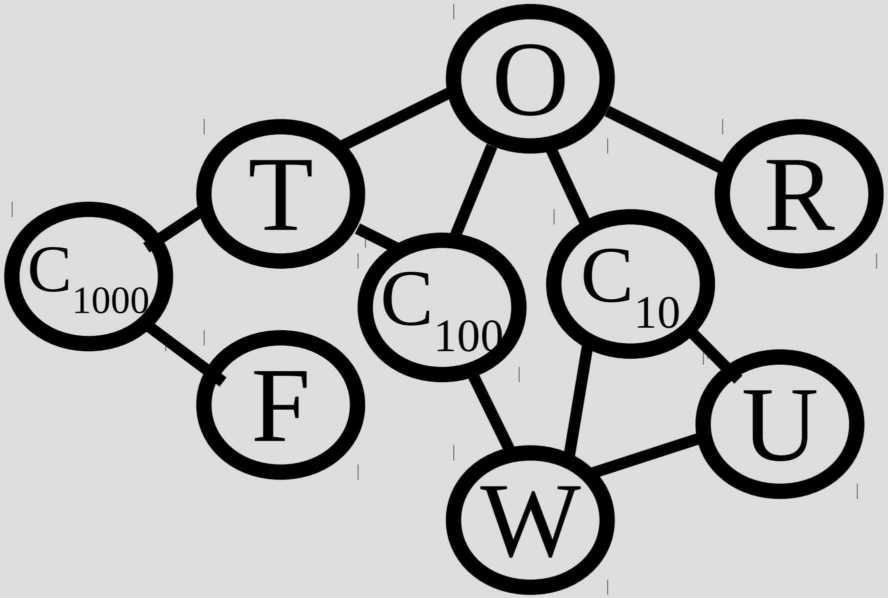


Pick O=0

Domains:
W = {5,6,7,8,9}, R = { } ← Invalid
U = {2,3,4,5,6,7,8,9}, T = { } ← Invalid

Example

Conflict: R with O



Pick O=2

Domains:

W = {5,6,7,8,9}, R = { 4 }

U = {0,3,4,5,6,7,8,9}, T = { } ← Invalid

(as C_{100} = 1, we claim T+T+1=2 mod 10...)

Example

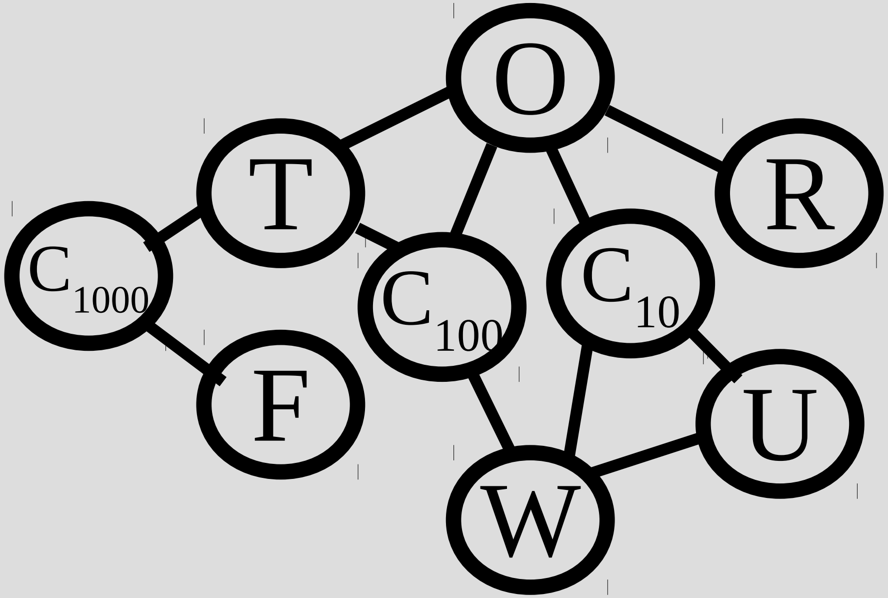
Conflict: T with O

Pick O=3

Domains:

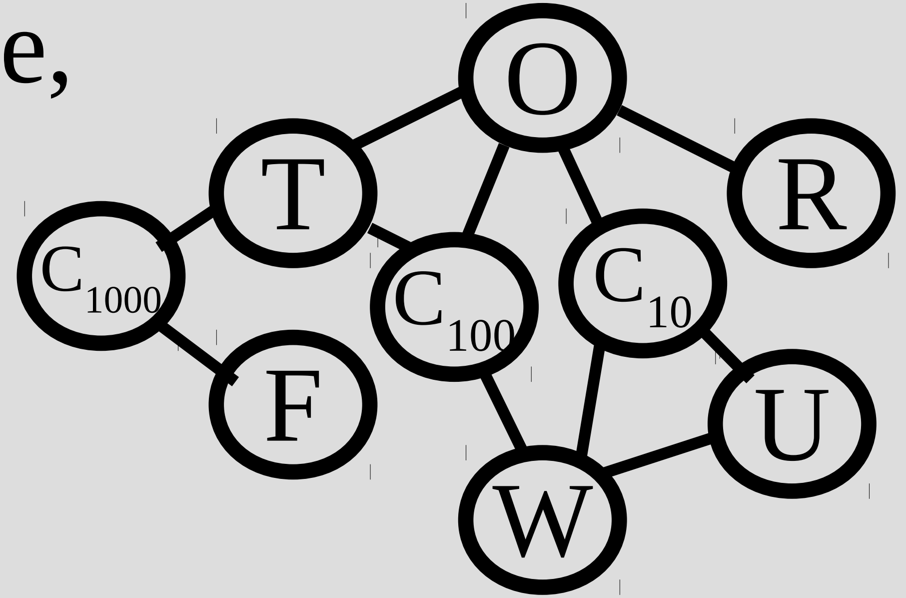
W = {5,6,7,8,9}, R = { 6 }

U = {0,2,4,5,6,7,8,9}, T = { 6 }



Example

Next smallest domain is tie,
T has more connections

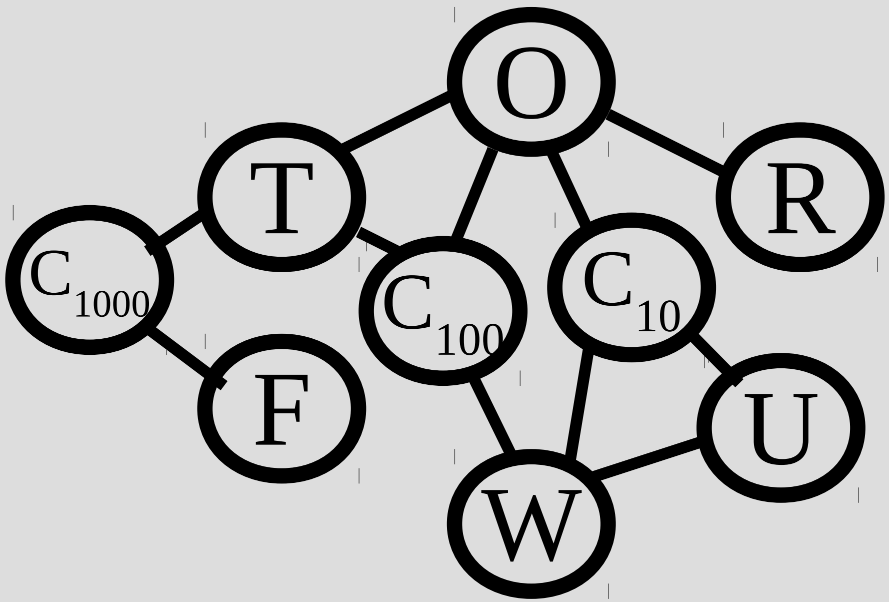


Pick T=6

Domains:
W = {5,7,8,9}, R = { } ← Invalid
U = {0,2,4,5,7,8,9}

Example

Conflict: R with T and O,
T has no other options,
so we go back to O

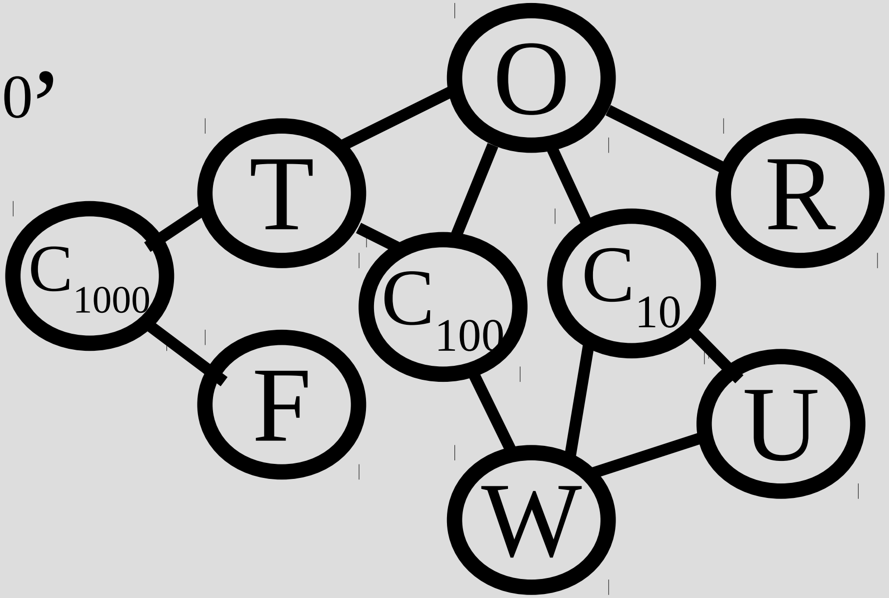


Pick O=4

Domains:
W = {5,7,8,9}, R = { 8 }
U = {0,2,3,5,7,8,9}, T = { } ← Invalid

Example

Conflict: T with O and C_{100},
no other options for C_{100},
so have to go back and
pick $C_{10}=1$



Domains:

$C_{100}=C_{1000}=\{0,1\}$, $O=\{5,6,7,8,9\}$

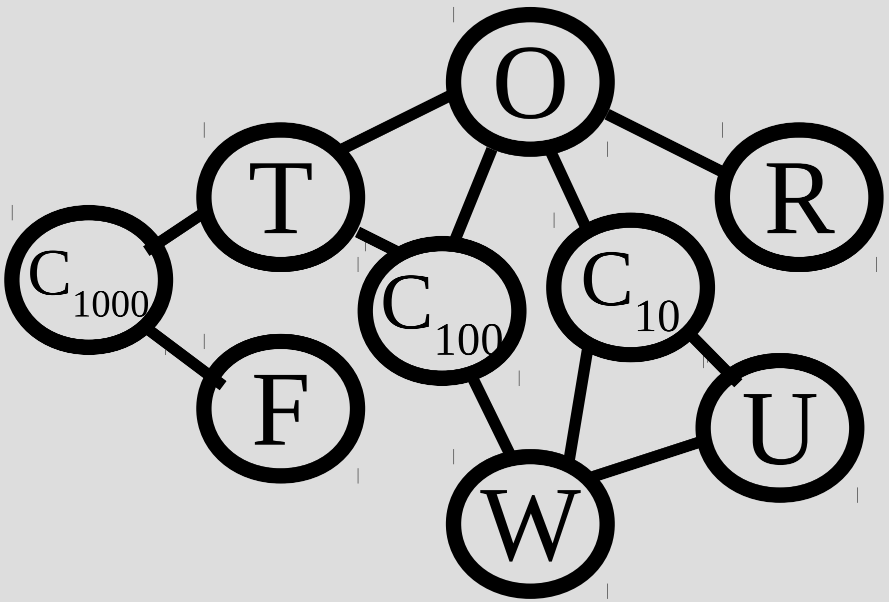
$F = T = \{1,2,3,4,5,6,7,8,9\}$

$W = U = R = \{0,1,2,3,4,5,6,7,8,9\}$

Example

Pick C_{100}=0, do part with F and C_{100} to find

C_{100}=F=1



Domains:

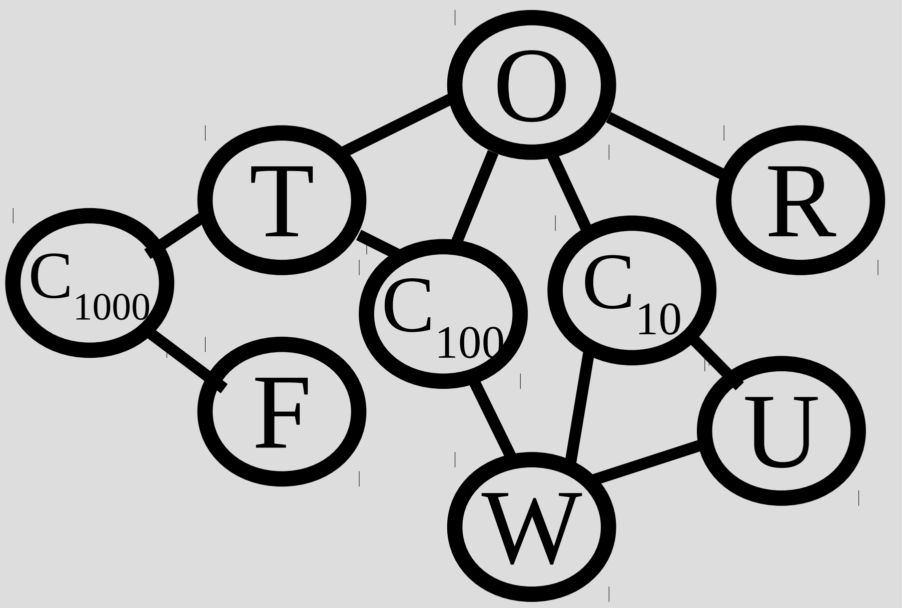
T=O={5,6,7,8,9}

W = {0,2,3,4}

U = R = {0,2,3,4,5,6,7,8,9}

Example

Tie for smallest domain,
O has more connections:



Pick O=5

Domains:

T = { } ← Invalid

W = {0,2,3,4}

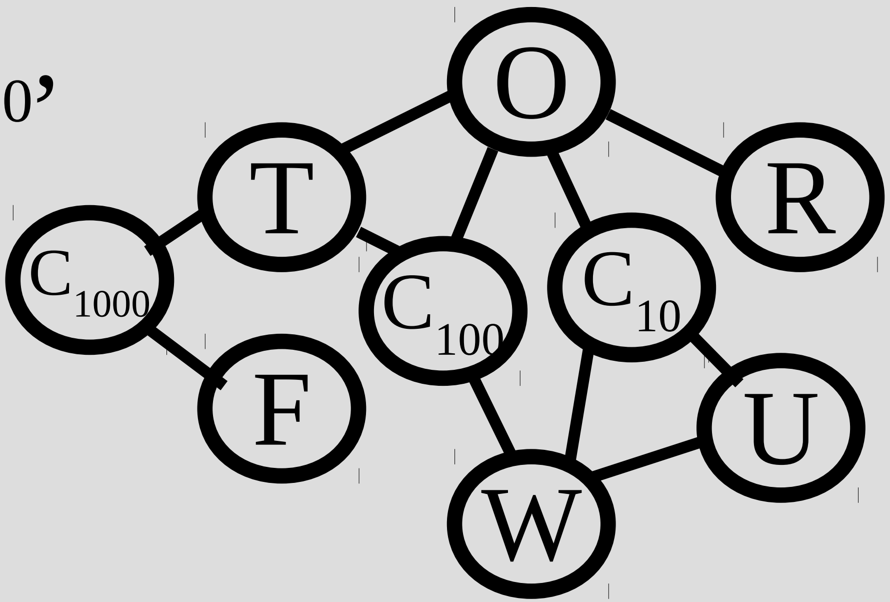
U = {0,2,3,4,6,7,8,9}, R = {0}

Example

Conflict: T with O and C_{100},
re-pick O...

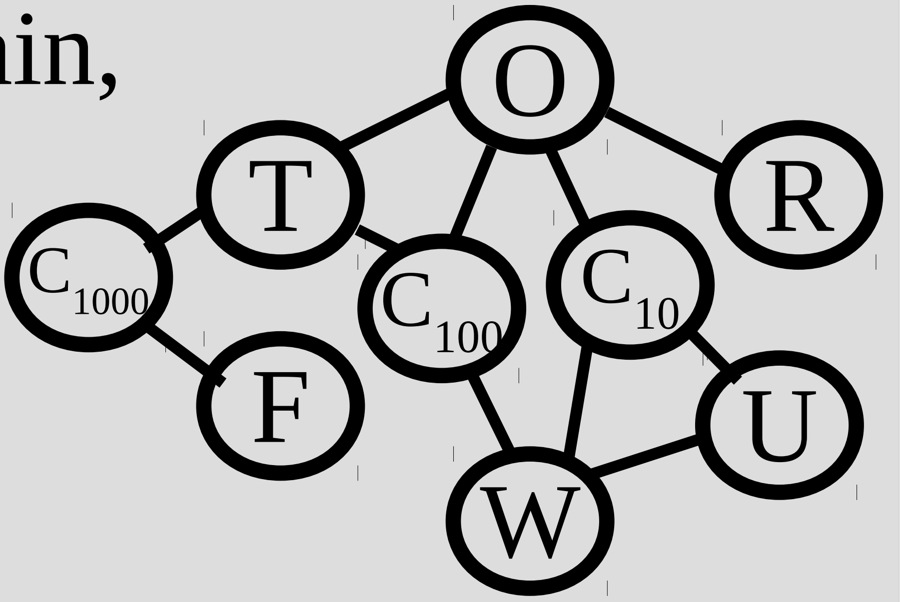
Pick O=6

Domains:
T = { 8 }
W = {0,2,3,4}
U = {0,2,3,4,5,7,8,9}, R = {2}



Example

Tie for next smallest domain,
T has more connections



Pick T=8

Domains:
W = {0,2,3,4}
U = {0,2,3,4,5,7,9}, R = {2}

Example

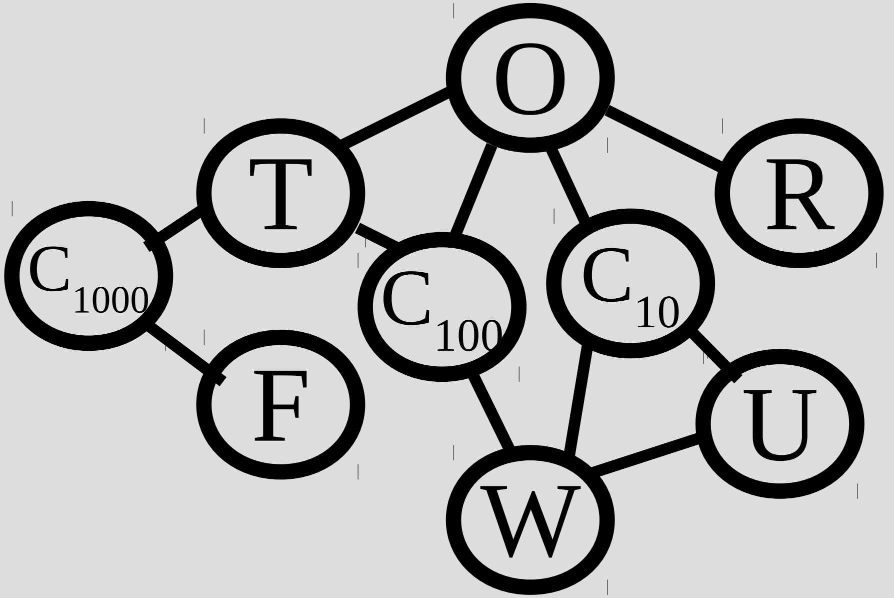
Next smallest domain is R

Pick R=2

Domains:

W = {0,3,4}

U = {0,3,4,5,7,9}

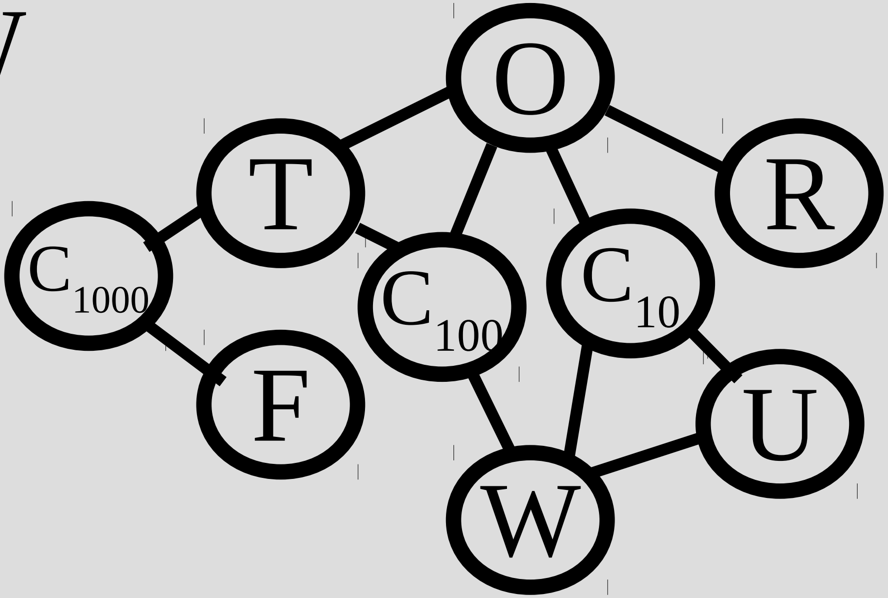


Example

Next smallest domain is W

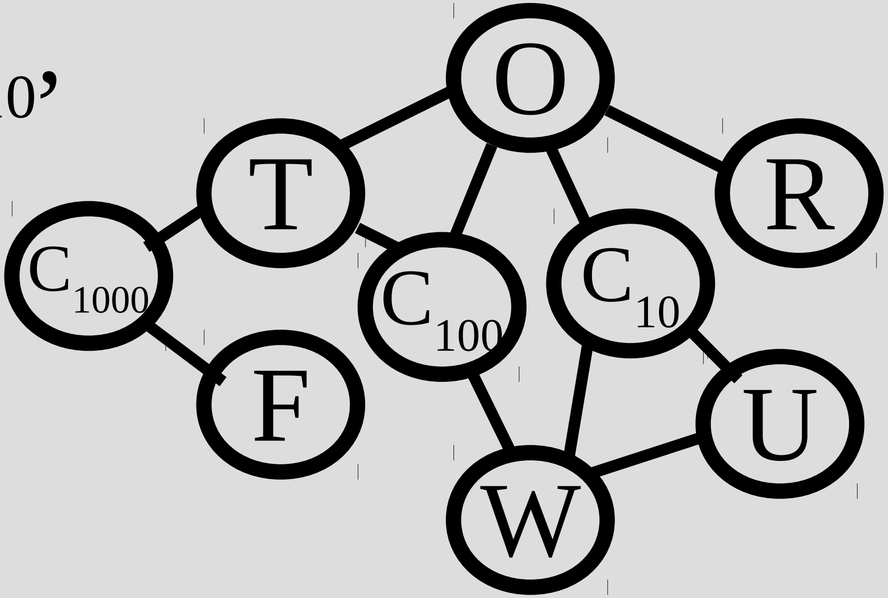
Pick W=0

Domains:
U = { } ← Invalid



Example

Conflict: U with W and C_{10},
most recent is W...

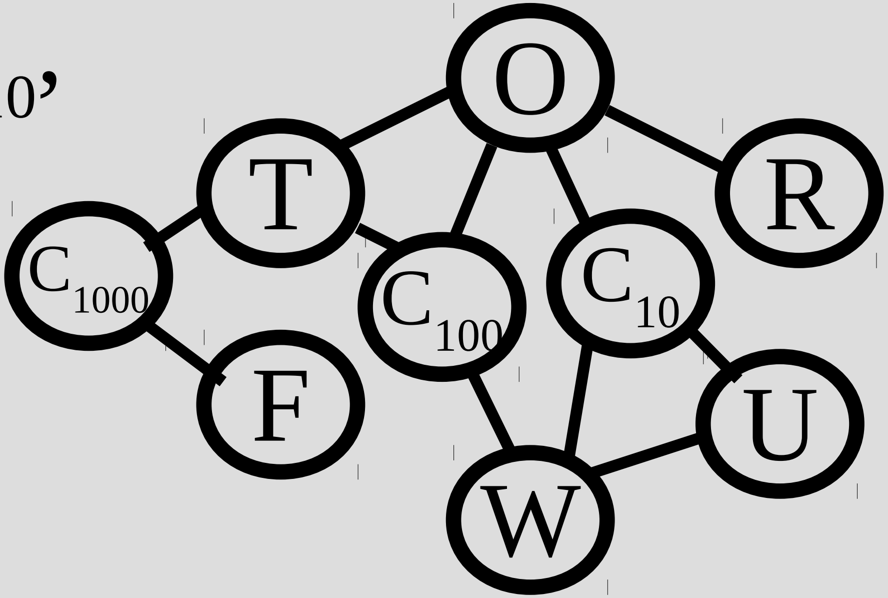


Pick W=3

Domains:
U = { } ← Invalid

Example

Conflict: U with W and C_{10}, most recent is W...



Pick W=4

Domains:

U = { } ← Invalid

Example

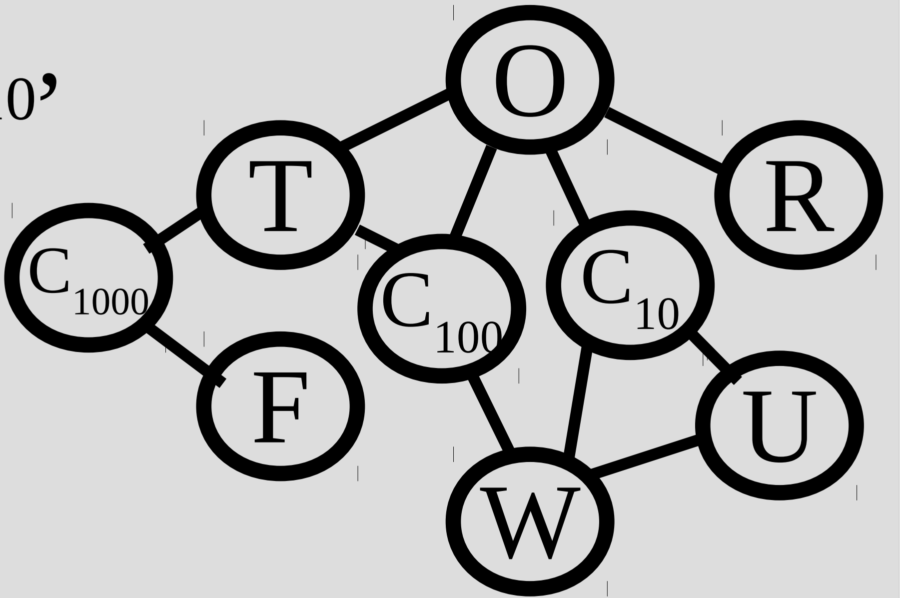
Conflict: U with W and C_{10}, most recent is W...

W has no more choices, (nor does R or T) so pick O = 7

Domains:

W = {0,2,3,4}, T = { } ← Invalid

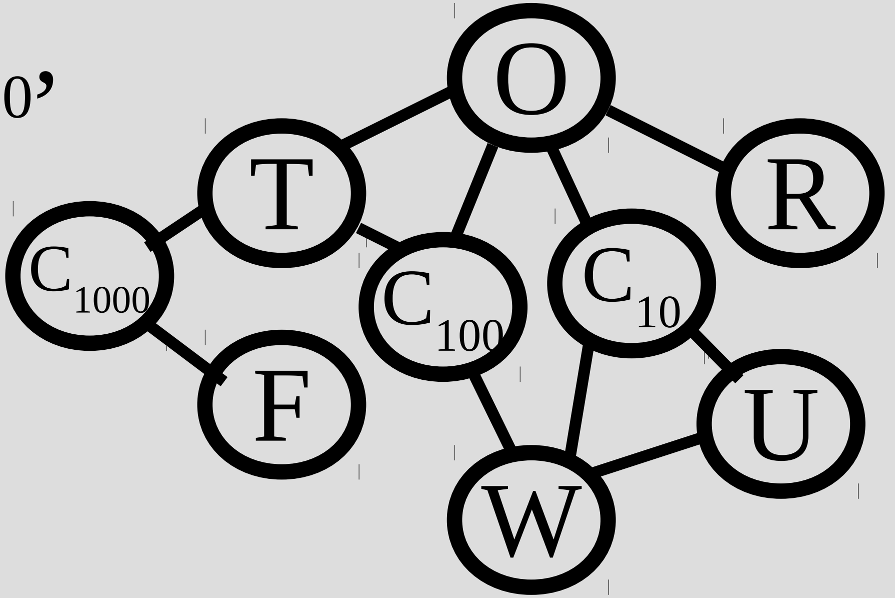
U = {0,2,3,4,5,6,8,9}, R = {4}



Example

Conflict: T with O and C_{100}, most recent is O

W has no more choices, (nor does R or T) so pick O = 8



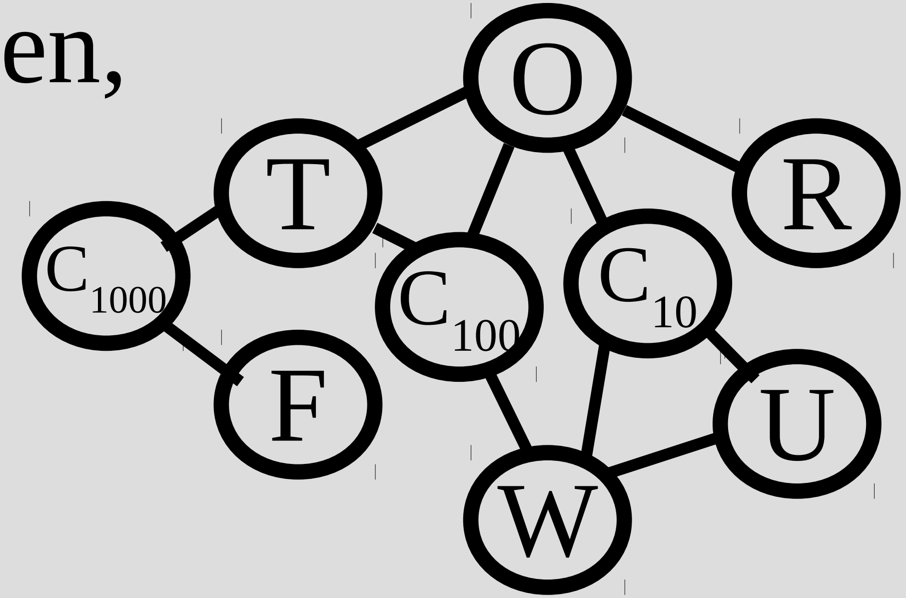
Domains:

W = {0,2,3,4}, T = { 9 }

U = {0,2,3,4,5,6,7,9}, R = {6 }

Example

Tie for domain size between, T and R, but T has more connections



Pick T = 9

Domains:

W = {0,2,3,4},

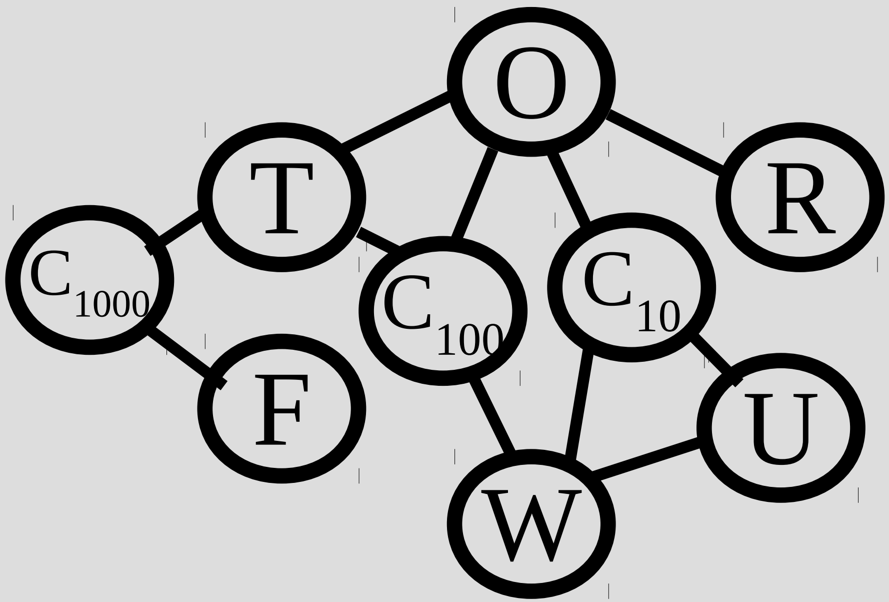
U = {0,2,3,4,5,6,7}, R = {6 }

Example

R has smallest domain

Pick R = 6

Domains:
W = {0,2,3,4},
U = {0,2,3,4,5,7}



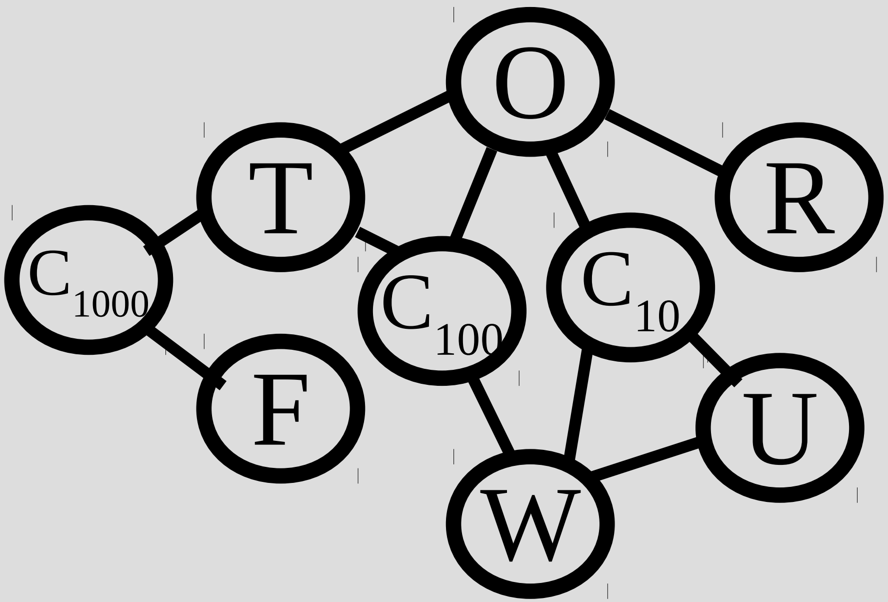
Example

W has smallest domain

Pick W = 0

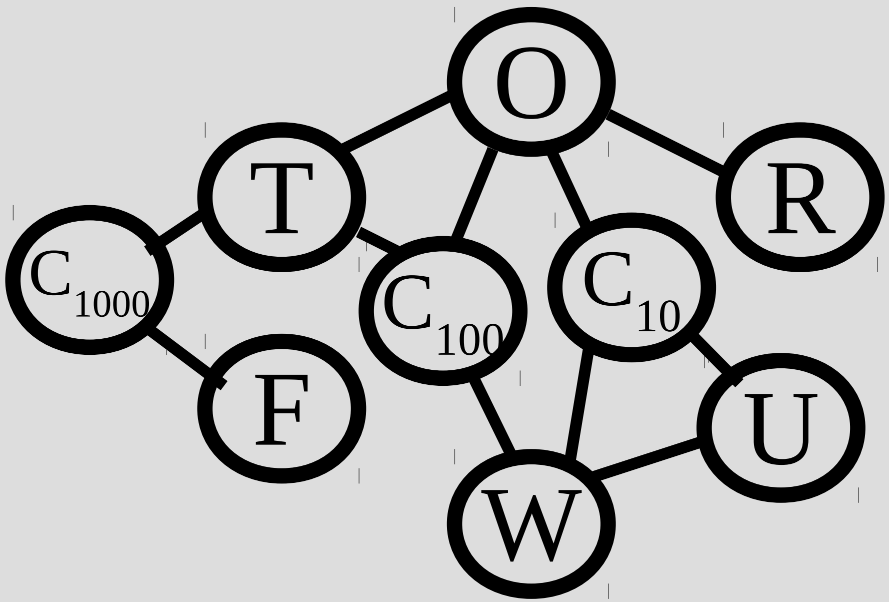
Domains:

U = { } ← Invalid



Example

Conflict with W and C_{10},
W most recent...



Pick W = 2

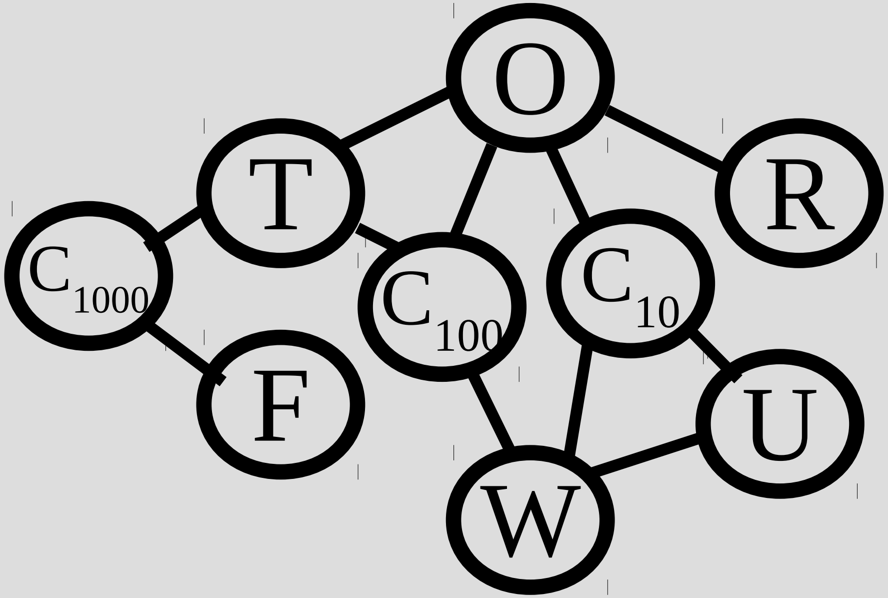
Domains:
U = { 5 }

Example

U has smallest domain (and only left)

Pick U = 5

Done!



Example

$C_{10} = 1$, $C_{100} = 0$, $C_{1000} = 1$

U=5, W=2, R=6, T=9, O=8, F=1

So:	T W O		928
	+ T W O	...becomes...	+ 928
	= F O U R		=1856

Complete-state CSP

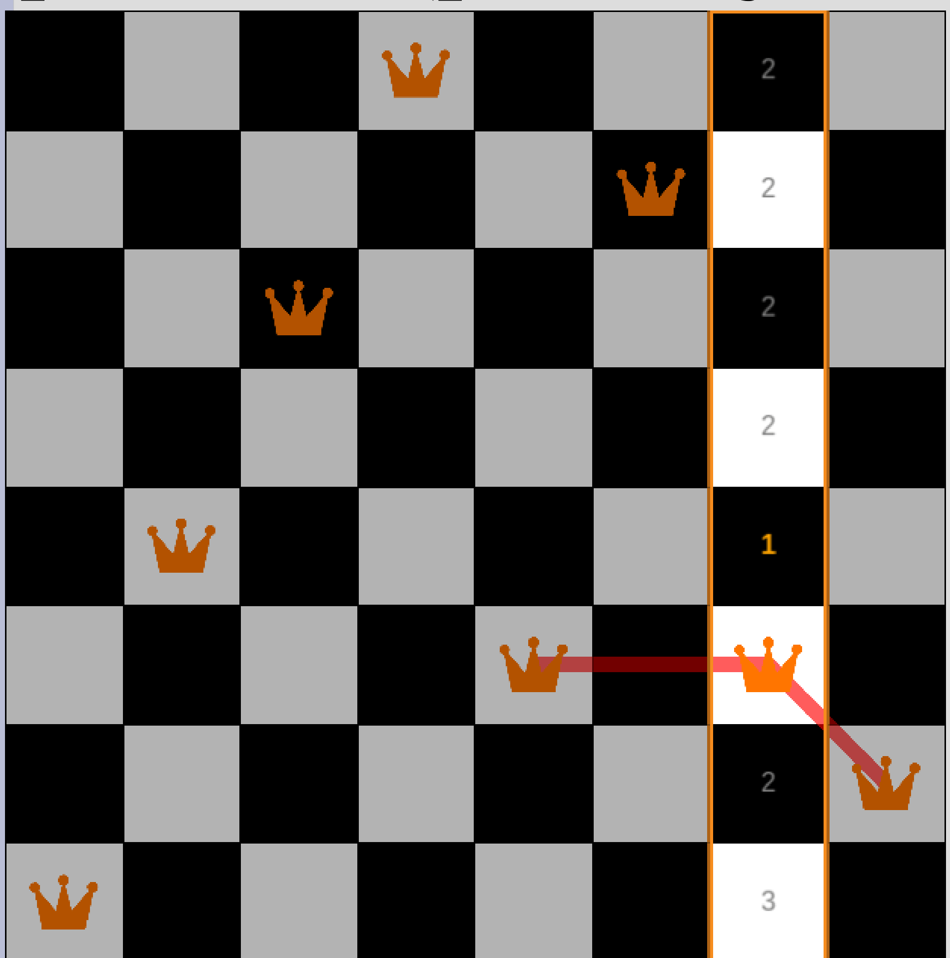
So far we have been looking at incremental search (adding one value at a time)

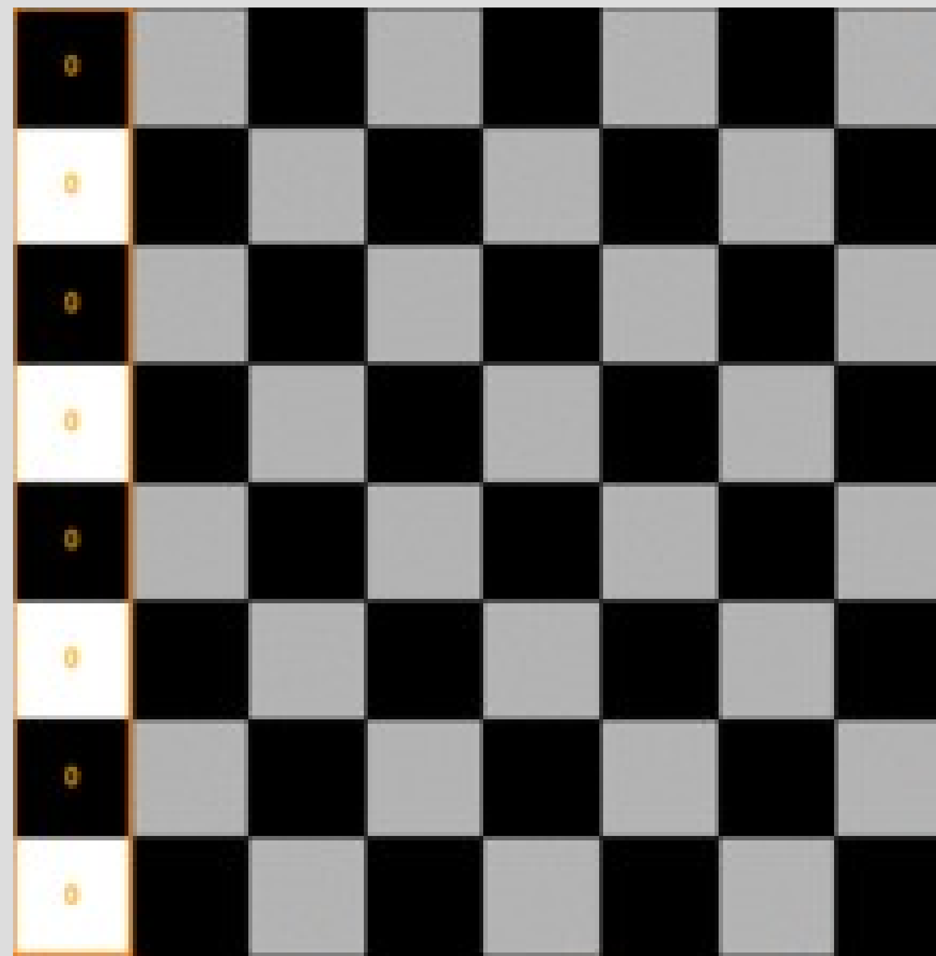
Complete-state searches are also possible in CSPs and can be quite effective

A popular method is to find the <u>min-conflict</u>, where you pick a random variable and update the choice to be one that creates the least number of conflicts

Complete-state CSP

This works incredibly well for the n-queens problem (partially due to dense solutions)





Complete-state CSP

As with most local searches (hill-climbing), this method has issues with plateaus

This can be mitigated by avoiding recently assigned variables (forces more exploration)

You can also apply weights to constraints and update them based on how often they are violated (to estimate which constraints are more restrictive than others)

Complete-state CSP

Local search does not have “locally optimal” solution our general search does

As we have a CSP, the “local optimal” may occur, but if it is not 0 then we know we are not satisfied (unless we searched the whole space and find no goal)

This is almost as if we had an almost perfect heuristic built in to the problem!