CS2 9/17/9 Last lecture

Backtracking search:

- one var at a time
- backtrack when no legal vals left

Key functions of alg:

var ← selectUnassignedVar()
foreach val in OrderDomainVals()

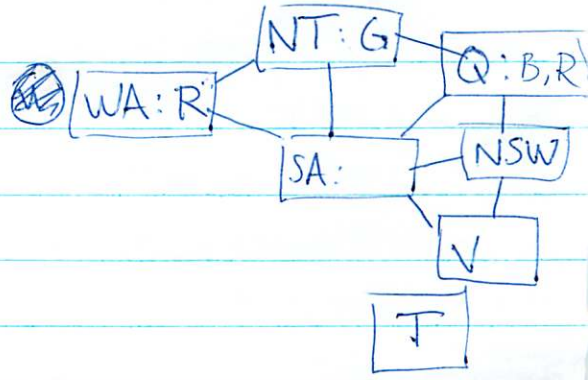
Heuristics:

- Minimum Remaining values (MRV)
  - choose var with fewest legal values
- Degree Heuristic
  - choose var those most constrains others
    ⇒ max reduction in tree size

selecting variables

- Least constraining value
  - choose value that rules out fewest choices for neighbors

How?
check graph & domain vals



→ ⊛ Forward Checking

CS2 ②
CST ⑨

What about the order of the values?

Eg. WA = R NT = G Q = ? B? No. then SA is {}

Choose: Least constraining value

Idea: it is most like to yield a soln.

⊛ Can we do better?

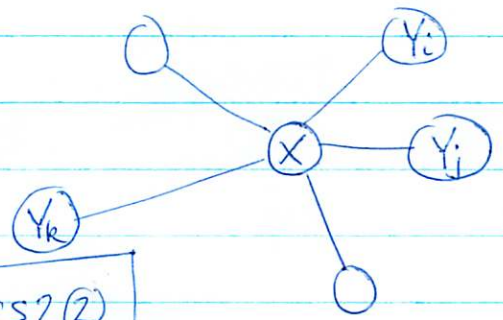
Want do reduce search space as much as possible.

What if we did more intelligent checking?

~~[illegible]~~

Forward checking:

Idea: whenever a var X is assigned,
look at each unassigned var Y that X is connected to
and delete vals from Y's domain, that are inconsistent with X.



→ ⊛ on CS2 ②

delete B
domain empty
⇒ early dike then

	WA	NT	Q	NSW	V	SA	T
Inital domain	RGB	RGB	RGB	RGB	RGB	RGB	RGB
WA = R	① R	② X GB	inconsistent	"	"consistent	③ X GB	"
Q = G	R	④ X B	G	R X B	"not consistent	X B	"
V = B	R	B	G	R X	B	~~B~~ X	"

make consistent by deleting B from domain of NSW

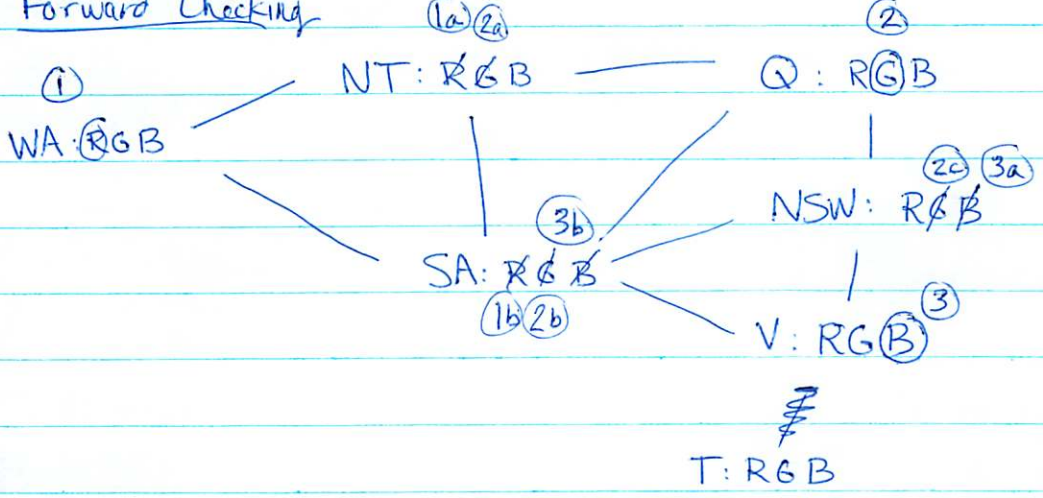
eliminated branching (by propagating information from WA → Q)

no soln ⇒ back track

FC + MRV gives 5-30x speed up

CSP (3) (3)

(*) Forward Checking



Note: MRV heuristic would choose NT or SA after (2) and detect failure sooner than choosing V.

At step (3b) SA domain is empty ⇒ back track immediately.

FC + MRV gives 5-30× speed up

Can we do better than FC?

CS1 (4) (5) 6 Constraint propagation (CP)

FC computes domain of each variable independently at start but doesn't detect all inconsistencies.

@ step (2)

Last example ∧ NT & SA can't both be B.

What if we propagated further by looking more ahead?
(Yeah, but would that be as slow as doing search?)

CP repeatedly enforces constraints

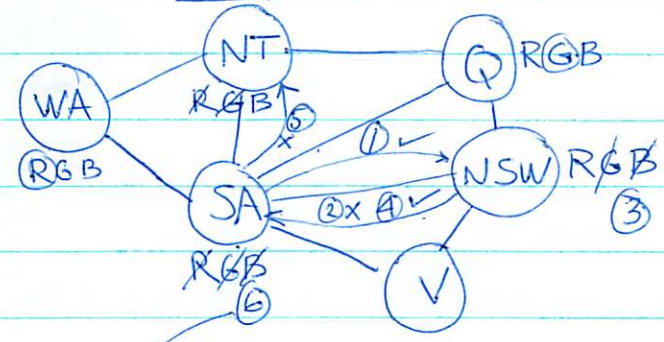
Simplest form: arc consistency

fast method of CP. Much stronger than FC.

Consider state of Domains during FC

WA=R & Q=G ⇒ both NT & SA are forced B, which is inconsistent.

Arc is a directed arc in constraint graph.



SA → NSW
Arc ∧ is consistent if for every value x of SA there is some val y of NSW that is consistent with x

consistent ∀ x of X ∃ some allowed y of Y.

X → Y

@ step (6) SA is empty ⇒ back track

CS8 (5)

Arc consistency is applied repeatedly until no more consistencies remain.
Why? Any deletion could introduce new inconsistency

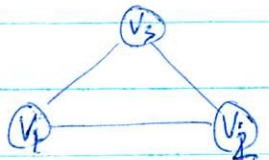
AC-3 alg. uses queue to keep track of arcs that need checking
Seems like a lot of work...
What is (time) complexity?
- binary CSP has at most $O(n^2)$ arcs
- inserted at most d times (only has d values to delete)
- $\Rightarrow$ checking an arc is $O(d^2)$ (potentially every pair in domain must be checked)
  $\Rightarrow$ total cost $O(n^2 d^3)$
  much more expensive than FC, but usually worth it

More general forms and stronger forms of constraint propagation:

k-consistency

CSP is k-consistent if
- for any set of k-1 vars
- and for any consistent assignment of those vars
- a consistent value can be assigned to the kth var.

k=1 1-consistency "node consistency"
k=2 2-consistency same as arc consistency
k=3 3-consistency "path consistency"



any pair can be extended to a third

Trade-off between extra cost of stronger checks & reduction in branching factor

CS2 ⑥ cryptarithmatic problem

```
  TWO
+ TWO
-----
 FOUR
```

Obviously there are constraints
All digits are different.

[?] How do we specify the problem?

Carry out standard arithmatic.

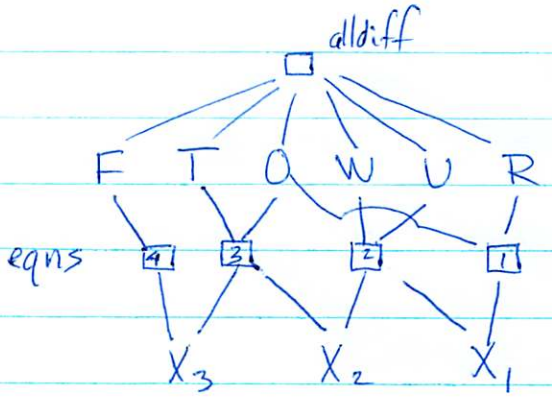
① $O + O = R + (10 \cdot X_1)$ ← we don't know if $O+O \geq 10$ so add auxiliary vars to represent carry over

② $X_1 + W + W = U + 10 \cdot X_2$

③ $X_2 + T + T = O + 10 \cdot X_3$

④ $X_3 = F$

Constraint hypergraph:



CS2 (6)

Example:
Minesweeper

0	0	1	V_1		
0	0	1	V_2		
0	0	1	V_3		
1	1	2	V_4		
V_8	V_7	V_6	V_5		

Which squares have bomb?
- squares with #'s don't
- other squares might
- # is how many bombs in neighboring squares

How do we solve this as a CSP?

[?] What do we want to know? ⟶ values of blank squares

[?] What should we choose as the representation?
All blanks? No
Just ones on border, or ~~#~~ who have non-blank neighbors

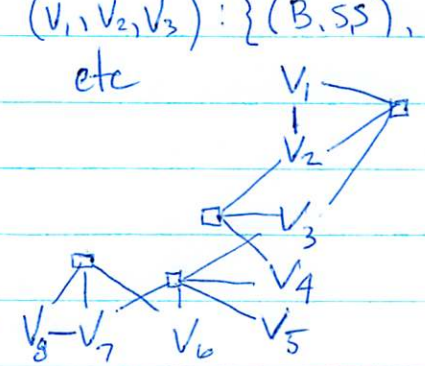
[2] What's the domain?

$$D = \{\text{space}, \text{bomb}\} = \{S, B\}$$

[?] What are the constraints? What values can vars have?

$C = \{(V_1, V_2) : \{(B,S), (S,B)\}$
$(V_1, V_2, V_3) : \{(B,S,S), (S,B,S), (S,S,B)\}$
etc

$V_1 + V_2 = 1$
~~[illegible]~~
$V_1 + V_2 + V_3 = 1$
$V_2 + V_3 + V_4 = 1$
$V_3 + V_4 + V_5 + V_6 + V_7 = 2$
$V_8 + V_7 + V_6 = 1$
$V_7 + V_8 = 1$



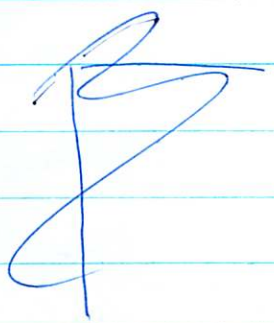
CS2 (7) (8) Local Search for CSPs

HC, SA typically work with "complete states: all vars assigned

To apply to CSPs
- allow states with unsatisfied constraints
- operators reassign var values

Variable selection:
- randomly select any conflicted var

Value selection (min. conflicts heuristic)
- choose value that violates fewest constraints
  - i.e. hill climb with h(n) = # violated constraints



[slide]