

Game theory (Ch. 17.5)

| | | Their intent | |
|-------------|---------|---|---|
| | | joke | serious |
| Your answer | joke |  |  |
| | serious |  |  |

Announcements

Writing 2 due Sunday

Repeated games

In repeated games, things are complicated

For example, in the basic PD, there is no benefit to “lying”

| | | PRISONER 2 | |
|------------|---------|------------|--------|
| | | Confess | Lie |
| PRISONER 1 | Confess | -8, -8 | 0, -10 |
| | Lie | -10, 0 | -1, -1 |

However, if you play this game multiple times, it would be beneficial to try and cooperate and stay in the [lie, lie] strategy

Repeated games

One way to do this is the tit-for-tat strategy:

1. Play a cooperative move first turn
2. Play the type of move the opponent last played every turn after (i.e. answer competitive moves with a competitive one)

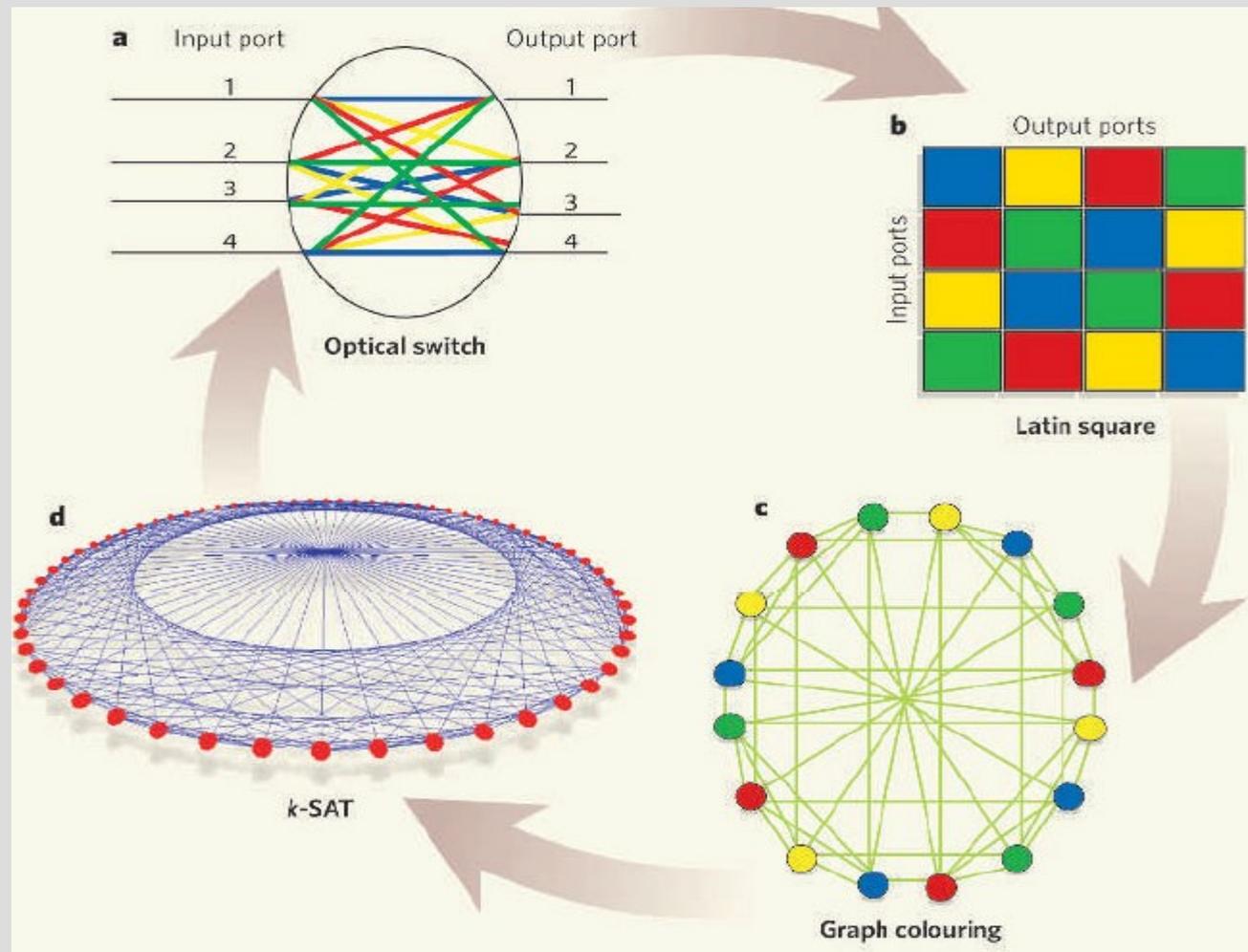
This ensure that no strategy can “take advantage” of this and it is able to reach cooperative outcomes

Repeated games

Two “hard” topics (if you are interested) are:

1. We have been talking about how to find best responses, but it is very hard to take advantage if an opponent is playing a sub-optimal strategy
2. How to “learn” or “convince” the opponent to play cooperatively if there is an option that benefits both (yet dominated)

Constraint sat. prob. (Ch. 6)



CSP

A constraint satisfaction problem is when there are a number of variables in a domain with some restrictions

A consistent assignment of variables has no violated constraints

A complete assignment of variables has no unassigned variables

(A solution is complete and consistent)

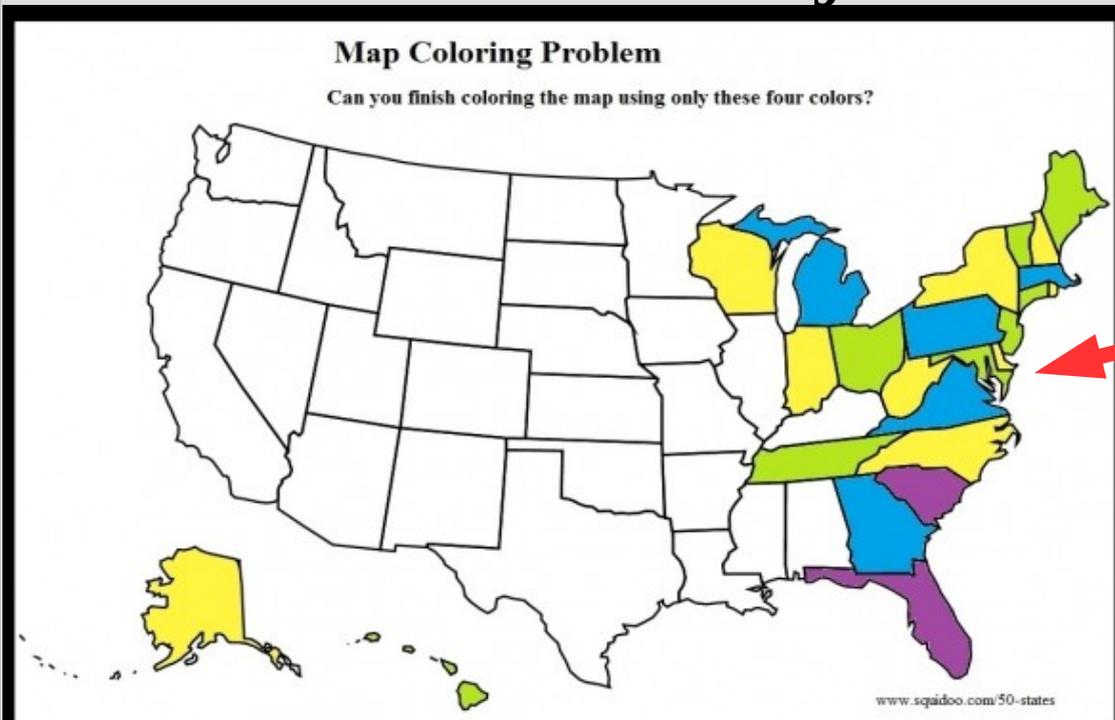
CSP

Map coloring is a famous CSP problem

Variables: each state/country

Domain: {yellow, blue, green, purple} (here)

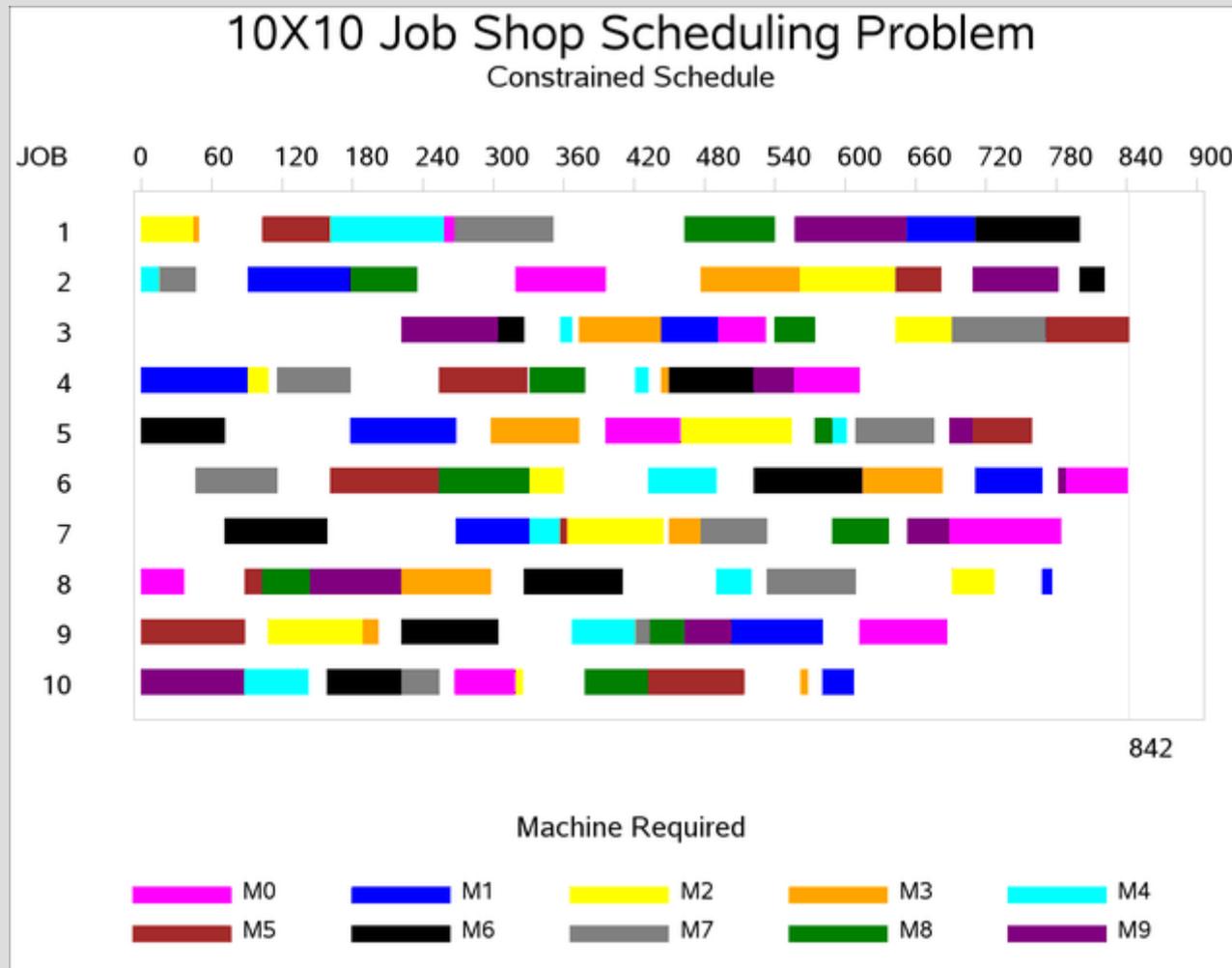
Constraints: No adjacent variables same color



Consistent
but partial

CSP

Another common use of CSP is job scheduling



CSP

Suppose we have 3 jobs: J_1, J_2, J_3

If J_1 takes 20 time units to complete, J_2 takes 30 and J_3 takes 15 but J_1 must be done before J_3

We can represent this as (and them together):

$$J_1 \ \& \ J_2: (J_1 + 20 \leq J_2 \ \underline{\mathbf{or}} \ J_2 + 30 \leq J_1)$$

$$J_1 \ \& \ J_3: (J_1 + 20 \leq J_3)$$

$$J_2 \ \& \ J_3: (J_2 + 30 \leq J_3 \ \underline{\mathbf{or}} \ J_3 + 15 \leq J_2)$$

Types of constraints

A unary constraint is for a single variable
(i.e. J_1 cannot start before time 5)

Binary constraints are between two variables
(i.e. J_1 starts before J_2)

All constraints can be broken down into using
only binary and unary

Types of constraints

K-consistency is:

For any consistent sets size $(k-1)$, there exists a valid value for any other variable (not in set)

1-consistency: All values in the domain satisfy the variable's unary constraints

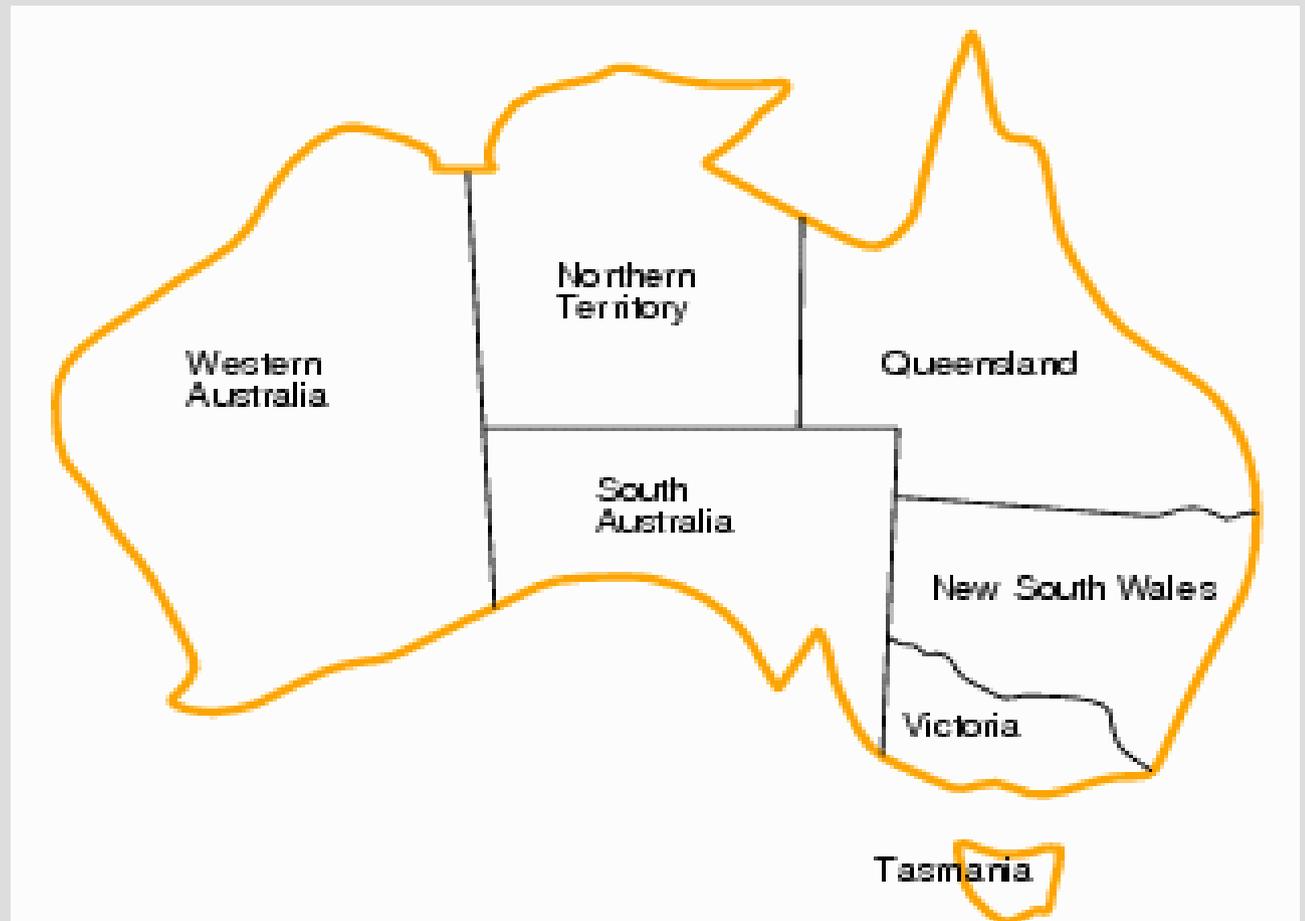
2-consistency: All binary values are in domain

3-consistency: Given consistent 2 variables, there is a value for a third variable(i.e. if $\{A,B\}$ is consistent, then exists C s.t. $\{A,C\}$ & $\{B,C\}$)

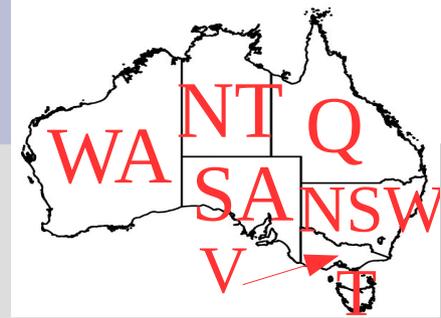
Types of constraints

- Rules: 1. Tasmania cannot be red
2. Neighboring providences cannot share colors

2 Colors:
red
green



Types of constraints



WA = {red, green}

NT = {red, green}

Q = {red, green}

SA = {red, green}

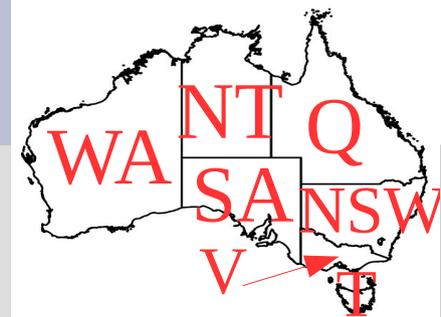
NSW = {red, green}

V = {red, green}

T = {red, green}

Not 1-consistent as we need T to not be red
(i.e. rule #2 eliminates T=red)

Types of constraints



$WA = NT = Q = SA = NSW = V$
 $= \{\text{red, green}\}$
 $T = \{\text{green}\}$

1-consistent now

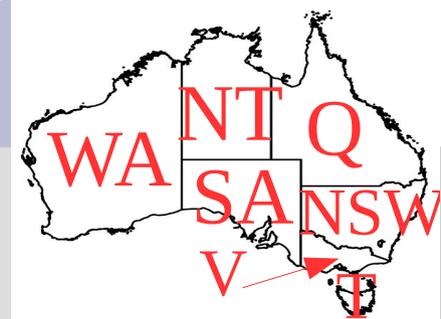
Also 2-consistent, for example:

Pick WA as “set k-1”, then try to pick NT...

If WA=green, then we can make NT=red

if WA=red, NT=green (true for all pairs)

Types of constraints



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

Not 3-consistent!

Pick (WA, SA) and add NT... If NT=green, will not work with either: (WA=red, SA=green) or (WA=green, SA=red)... NT=red also will not work, so NT's domain is empty and not 3-cons.

Applying constraints

We can repeatedly apply our constraint rules to shrink the domain of variables (we just shrunk NT's domain to nothing)

This reduces the size of the domain, making it easier to check:

- If the domain size is zero, there are no solutions for this problem
- If the domain size is one, this variable must take on that value (the only one in domain)

Applying constraints

AC-3 checks all 2-consistency constraints:

1. Add all binary constraints to queue
2. Pick a binary constraint (X_i, Y_j) from queue
3. If x in $\text{domain}(X_i)$ and no consistent y in $\text{domain}(Y_j)$, then remove x from $\text{domain}(X_i)$
4. If you removed in step 3, update all other binary constraints involving X_i (i.e. (X_i, X_k))
5. Goto step 2 until queue empty

Applying constraints

Some problems can be solved by applying constraint restrictions (such as sudoku) (i.e. the size of domain is one after reduction)

Harder problems this is insufficient and we will need to search to find a solution

Which is Thursday's topic!