Constraint Satisfaction Problems (CSPs)

CS311
David Kauchak
Spring 2013

Some material borrowed from:
Sara Owsley Sood and others

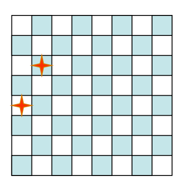
Admin

- Final project comments:
  - Use pre-existing code
  - Get your data now!
    - Use pre-existing data sets
  - Finding good references
    - Google scholar (http://scholar.google.com/)
    - Other papers
- Final project proposals due tomorrow at 6pm
- Some written problems will be posted tomorrow

Quick search recap

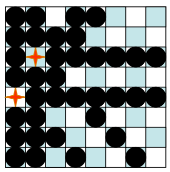
- Search
  - uninformed
    - BFS, DFS, IDS
  - informed
    - A*, IDA*, greedy-search
- Adversarial search
  - assume player makes the optimal move
  - minimax and alpha-beta pruning
- Local search (aka state space search)
  - start random, make small changes
  - dealing with local minima, plateaus, etc.
    - random restart, randomization in the approach, simulated annealing, beam search, genetic algorithms

Intro Example: 8-Queens



Where should I put the queens in columns 3 and 4?

Intro Example: 8-Queens



The decisions you make constrain the possible set of next states

Sudoku

1	6	4						2
2			4		3	9	1	
		5		8		4		7
	9				6	5		
5			1		2			8
		8	9				3	
8		9		4		2		
	7	3	5		9			1
4						6	7	9

What value?

Sudoku

1	6	4						2
2			4		3	9	1	
		5		8		4		7
	9				6	5		
5			1		2			8
		8	9				3	
8		9		4		2		
	7	3	5		9			1
4						6	7	9

1
2
3
4
5
6
7
8
9

Sudoku

1	6	4						2
2			4		3	9	1	
		5		8		4		7
	9				6	5		
5			1		2			8
		8	9				3	
8		9		4		2		
	7	3	5		9			1
4						6	7	9

~~1~~
~~2~~
~~3~~
~~4~~
~~5~~
6
~~7~~
~~8~~
~~9~~

Sudoku

1	6	4						2
2			4		3	9	1	
		5		8		4		7
	9				6	5		
5			1		2			8
		8	9				3	
8		9		4		2		
	7	3	5		9			1
4						6	7	9

We could try and solve this by searching, but the problem constraints may *direct* us better allowing for a much faster solution finding.

Constraint satisfaction problem

Another form of search (more or less)!

- Set of **variables**: $x_1, x_2, \ldots, x_n$
- **Domain** for each variable indicating possible values: $D_{x1}, D_{x2}, \ldots, D_{xn}$
- Set of **constraints**: $C_1, C_2, \ldots, C_m$
  - Each constraint limits the values the variables can take
    - $x_1 \neq x_2$
    - $x_1 < x_2$
    - $x_1 + x_2 = x_3$
    - $x_4 < x_5^2$

Goal: an assignment of values to the variables that satisfies all of the contraints

Applications?

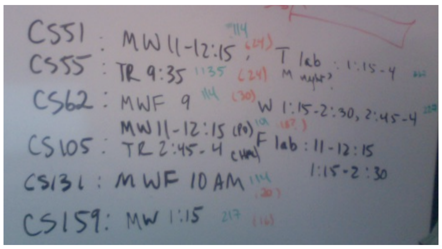
Applications

Scheduling:

Me	I'd like to try and meet this week, just to touch base and see how everything is going. I'm free: Anytime Tue., Wednesday after 4pm, Thursday 1-4pm
S1	I can do Tuesday 11-2:30, 4+, Wednesday 5-6, Thursday 11-2:30
P2	I can do anytime Tuesday (just before or after lunch is best), not Wednesday, or Thursday afternoon.
S2	I'm free Tuesday and Thursday from 2:45-4 or so, and also Wednesday any time after 3.
S3	I can meet from 4-5 on Tuesday or Wednesday after 5.

Applications

Scheduling



Applications

Scheduling

- manufacturing
- Hubble telescope time usage
- Airlines
- Cryptography
- computer vision (image interpretation)
- …

Why CSPs?

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene C. Freuder, Constraints, April 1997

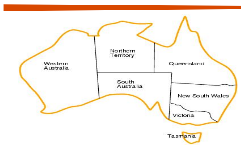
Why CSPs?

If you can represent it in this standard way (set of variables with a domain of values and constraints), the successor function and goal test can be written in a generic way that applies to **all** CSPs

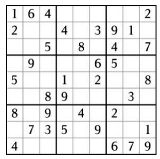
We can develop effective generic heuristics that require **no domain specific expertise**

The **structure** of the constraints can be used to simplify the solution process

Defining CSP problems



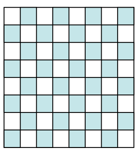
Graph coloring



Sudoku

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

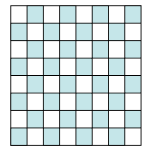
Cryptarithmatic



8-queens

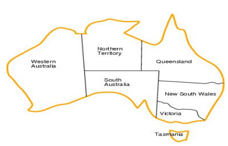
1. variables
2. domains of the variables
3. constraints

Example: 8-Queens Problem



- 8 variables $x_1, x_2, \ldots, x_8$
- Domain for each variable: {1,2,…,8}
- Constraints are of the forms:
  - row constraints: $x_i \neq x_j$ for all i, j where $j \neq i$
  - diagonal constraints: $|i-j| \neq |x_i - x_j|$, for all i, j where $j \neq i$

Example: Map Coloring



- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain {red, green, blue}
- No two adjacent variables have the same value:

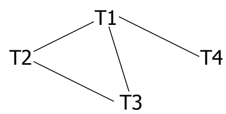
  WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V,Q≠NSW, NSW≠V

CSP Example: Cryptharithmetic puzzle

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

Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
Constraints
alldiff(F,T,U,W,R,O)
$O + O = R + 10 \cdot X_1$, etc.

Example: Task Scheduling



T1 must be done during T3
T2 must be achieved before T1 starts
T2 must overlap with T3
T4 must start after T1 is complete

Many different constraint types

Unary constraints: involve only a single variable (x_1 != green)

Binary constraints: involve two variables

Higher order constraints: involve 3 or more variables (e.g. *all-diff*(a,b,c,d,e))

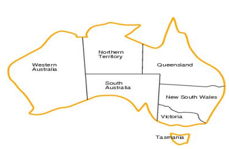
– all higher order constraints can be rewritten as binary constraints by introducing additional variables!

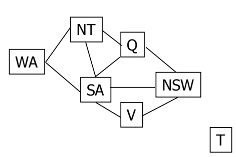
Preference constraints - no absolute - they indicate which solutions are preferred

– I can meet between 3-4, but I'd prefer to meet between 2-3
– Electricity is cheaper at night
– Workers prefer to work in the daytime

Constraint Graph

Binary constraints





Two variables are adjacent or neighbors if they are connected by an edge or an arc

CSP as a Search Problem

Initial state:

– {} no assignments

Successor function:

– any assignment to an unassigned variable that does not conflict

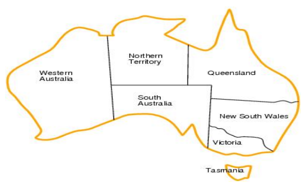
Goal test:

– all variables assigned to?

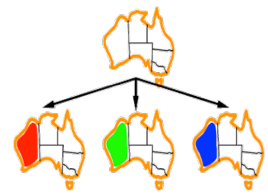
Max search depth?

– number of variables

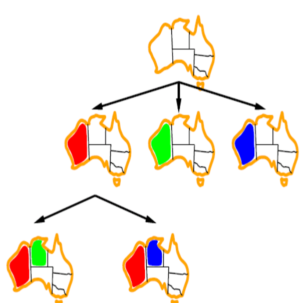
CSP as search



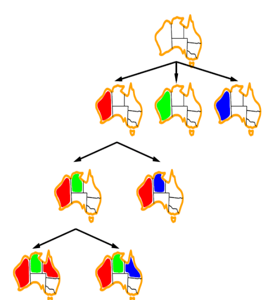
CSP as search



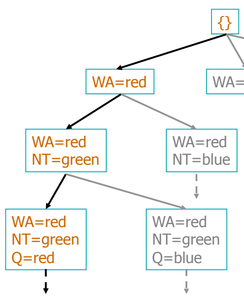
CSP as search

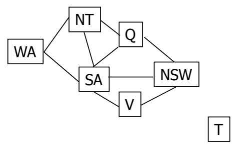


CSP as search



CSP as search





Backtracking Algorithm

CSP-BACKTRACKING(PartialAssignment a)
- If a is complete
  - return a
- x ← select an unassigned variable
- D ← select an ordering for the domain of x
- For each value v in D do
  - If v is consistent with a then
    - Add (x = v) to a
    - result ← CSP-BACKTRACKING(a)
    - If result ≠ *failure* then return result
- Return *failure*

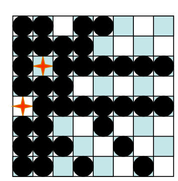
CSP-BACKTRACKING({})

Questions

CSP-BACKTRACKING(PartialAssignment a)
- If a is complete
  - return a
- x ← select an unassigned variable
- D ← select an ordering for the domain of x
- For each value v in D do
  - If v is consistent with a then
    - Add (x = v) to a
    - result ← CSP-BACKTRACKING(a)
    - If result ≠ *failure* then return result
- Return *failure*

- Which variable *x* should be assigned a value next?
- In which order should its domain D be sorted?
- How do choices made affect assignments for unassigned variables?

Choice of Variable

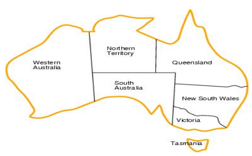


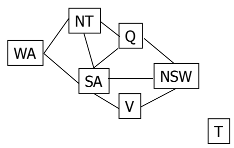
x ← select an unassigned variable

Which variable should we pick?

The most constrained variable, i.e. the one with the fewest remaining values – column 3

Choice of Variable



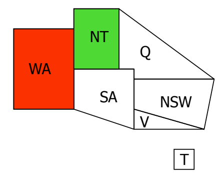


x ← select an unassigned variable

Which variable should we start with?

The variable involved with the most constraints - SA

Choice of Variable

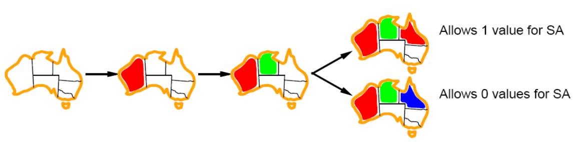


D ← select an ordering for the domain of x

Which value should we pick for Q?

Least constraining value - RED

Least constraining value



Prefer the value that leaves the largest subset of legal values for other unassigned variables

Why CSPs?

Notice that our heuristics work for **any** CSP problem formulation

– unlike our previous search problems!

– does not require any domain knowledge

- mancala heuristics
- straight-line distance

Eliminating wasted search

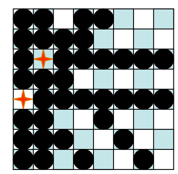
One of the other important characteristics of CSPs is that we can prune the domain values without actually searching (searching implies guessing)

Our goal is to avoid searching branches that will ultimately dead-end

How can we use the information available at early on to help with this process?

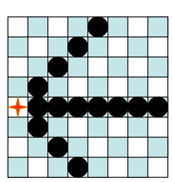
Constraint Propagation …

… is the process of determining how the possible values of one variable affect the possible values (domains) of other variables

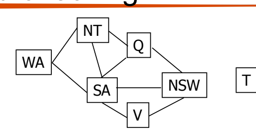


Forward Checking

After a variable X is assigned a value v, look at each unassigned variable Y that is connected to X by a constraint and delete from Y's domain any value that is inconsistent with v



Forward checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

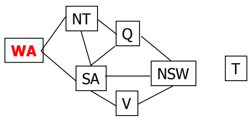
Can we detect inevitable failure early?

– *And avoid it later?*

Forward checking idea: keep track of remaining legal values for unassigned variables.

Terminate search when any variable has no legal values.

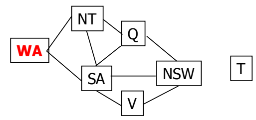
Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

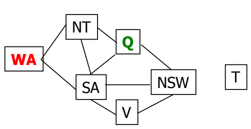
Pick red for WA… how does it change the domains?

Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB

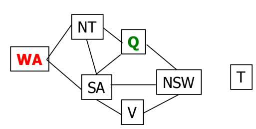
Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB

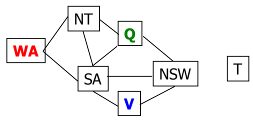
Pick green for Q… how does it change the domains?

Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

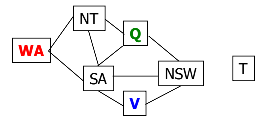
Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

Pick blue for V… how does it change the domains?

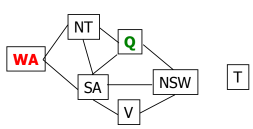
Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R	B		RGB

Only picked 3 colors, but already know we're at a dead end!

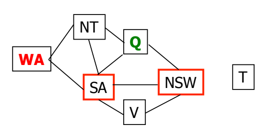
Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

After just selecting 2… anything wrong with this?

Map Coloring



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

After just selecting 2… anything wrong with this?

Removal of Arc Inconsistencies

Given two variables x_j and x_k that are connected by some constraint

We have the current remaining domains D_{xj} and D_{xk}

For every possible label in D_{xj}
- if using that label leaves no possible labels in D_{xk}
- Then get rid of that possible label

See full pseudocode in the book

Arc consistency: AC-3 algorithm

What happens if we remove a possible value during an arc consistency check?
- may cause other domains to change!

When do we stop?
- keep running repeatedly until no inconsistencies remain
- can get very complicated to keep track of which to check

Arc consistency: AC-3 algorithm

systematic way to keep track of which arcs still need to be checked

AC-3
- keep track of the set of possible constraints/arcs that may need to be checked
- grab one from this set
- if we make changes to variable's domain, add all of it's constraints into the set
- keep doing this until no constraints exist

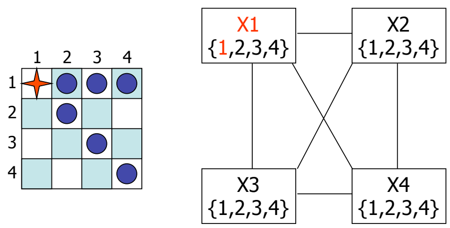
Solving a CSP

Search:
- can find good solutions, but must examine non-solutions along the way

Constraint Propagation:
- can rule out non-solutions, but this is not the same as finding solutions

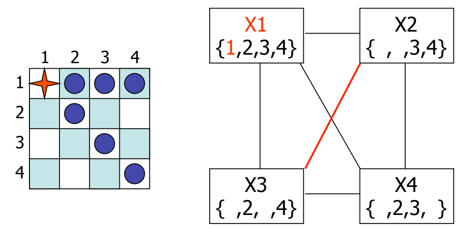
Interweave constraint propagation and search
- Perform constraint propagation at each search step.

4-Queens Problem



What can we remove with forward checking?

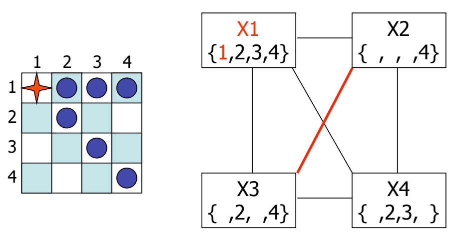
4-Queens Problem



Anything else with arc consistency?

Can't have X2 = 3!

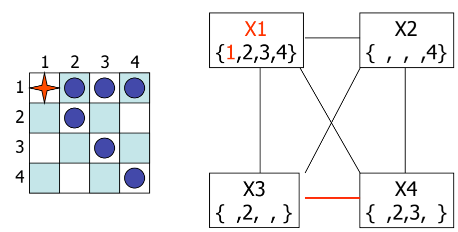
4-Queens Problem



Anything else?

Can't have X3 = 4!

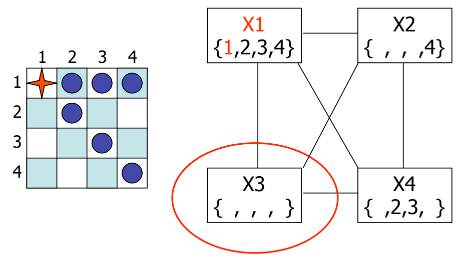
4-Queens Problem



Anything else?

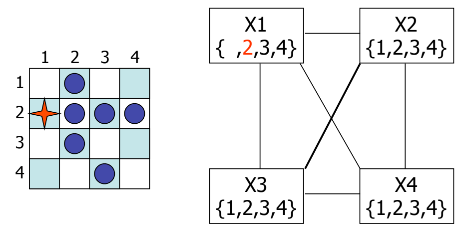
Can't have X3 = 2!

4-Queens Problem

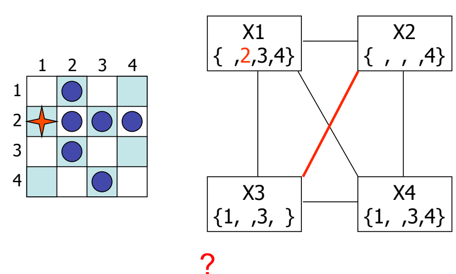


Technically no search over values was involved. Only looked at constraints.

4-Queens Problem

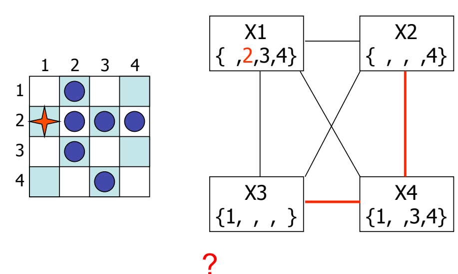


4-Queens Problem



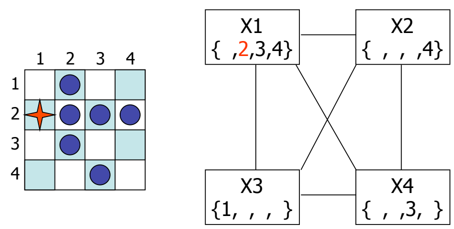
Can’t have X3 = 3

4-Queens Problem

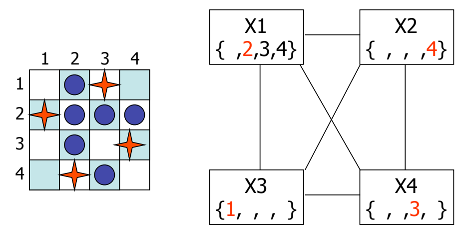


Can’t have X4 = 1 or X4 = 4

4-Queens Problem



4-Queens Problem



Only searched 2 nodes!

CSP Summary

Key: allow us to use heuristics that are problem independent

CSP as a search problem
- Backtracking algorithm
- General heuristics

Forward checking

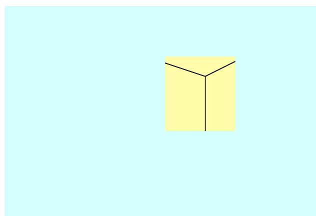
Constraint propagation

Interweaving CP and backtracking

Edge Labeling in Computer Vision

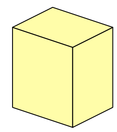
How do you know what the 3-D shape looks like?

Edge Labeling



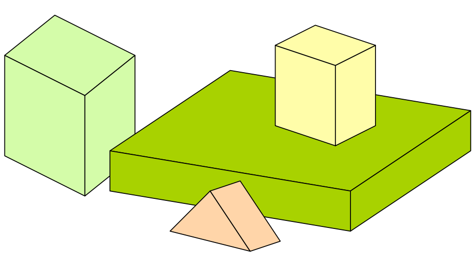
In or out?

Edge Labeling



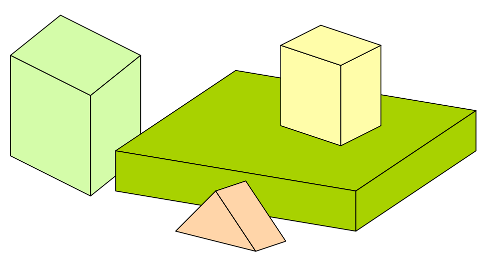
In or out?

Edge Labeling



In or out?

Edge Labeling



Information about the other edges
constrains the possibilities

Labels of Edges

Convex edge:
- two surfaces intersecting at an angle greater than 180°
- often, “sticking out”, “towards us”

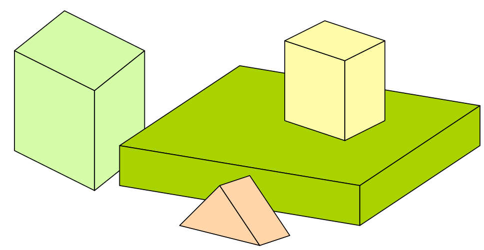
Concave edge
- two surfaces intersecting at an angle less than 180°
- often, “folded in”, “away from us”

+ convex edge, both surfaces visible

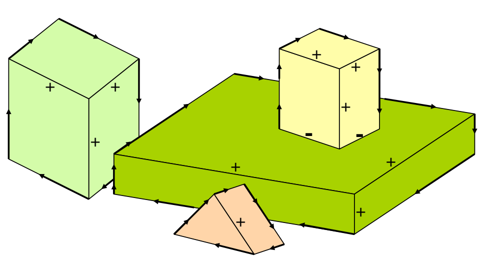
− concave edge, both surfaces visible

← convex edge, only one surface is visible and it is on the right side of ←

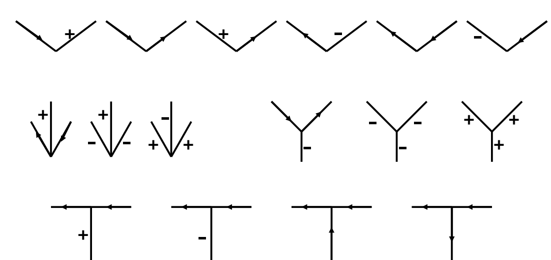
Edge Labeling



Edge Labeling



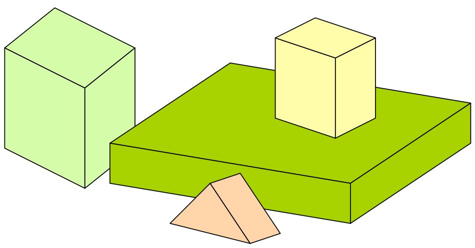
Junction Label Sets



(Waltz, 1975; Mackworth, 1977)

Edge Labeling

CSP?



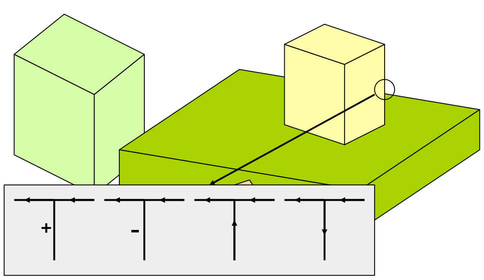
Edge Labeling as a CSP

A variable is associated with each junction

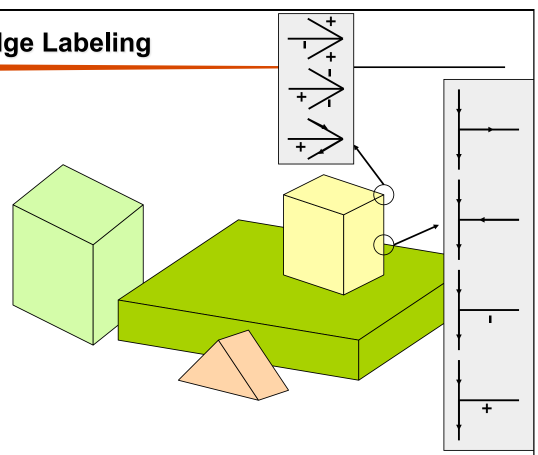
The domain of a variable is the label set of the corresponding junction

Each constraint imposes that the values given to two adjacent junctions give the same label to the joining edge

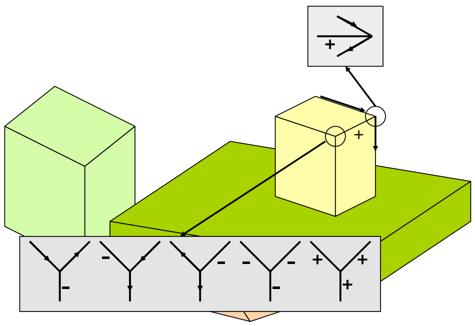
Edge Labeling



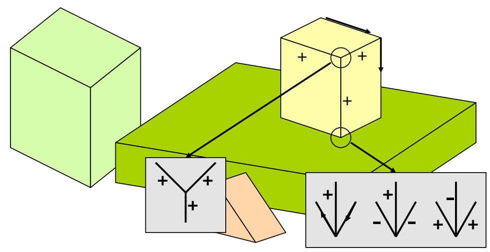
Edge Labeling



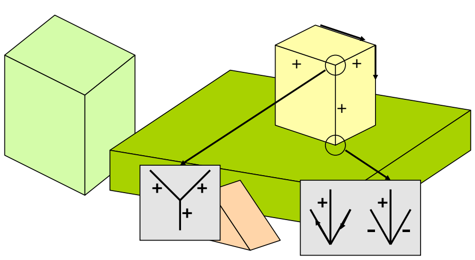
Edge Labeling



Edge Labeling



Edge Labeling



Edge Labeling