Constraint Satisfaction and Scheduling

Shamelessly lifted from

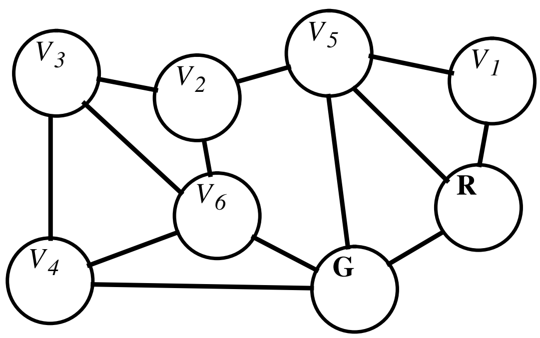
Andrew W. Moore
by David Cohn
cohn+@cs.cmu.edu

http://www.cs.cmu.edu/~cohn

Overview

- CSPs defined
- Using standard search for CSPs
- Blindingly obvious improvements
  - Backtracking search
  - Forward Checking
  - Constraint Propagation
- Some example CSP applications
  - Overview
  - Waltz Algorithm
  - Job Shop Scheduling
- Variable ordering
- Value ordering
- Discussion

A Constraint Sat'n problem



Inside each circle marked V1 .. V6 we must assign an R, a G or a B .

No two connected circles may be assigned the same symbol.

Notice that two circles have already been given an assignment.

Formalized Constraint Satisfaction Problem

A CSP is a triplet { V , D , C }. A CSP has a finite set of variables V = { V1 , V2 .. VN }

Each variable may be assigned a value from a domain D of values .

Each member of C is a pair . The first member of each pair is a set of variables . The second element is a set of legal values which that set may take .

Example:

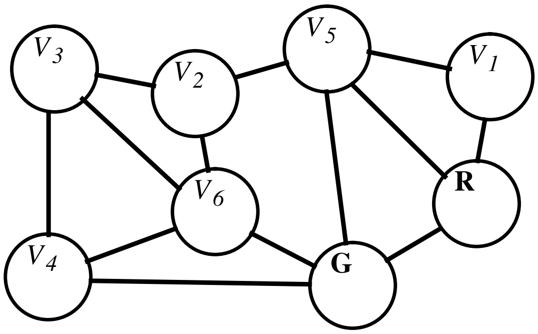
V = { V1 , V2 , V3 , V4 , V5 , V6 }

D = { R , G , B }

C = { (V1,V2) : { (R,G), (R,B), (G ,R),(G,B),(B,R),(B,G) },
{(V1,V3) : {(R,G), (R,B), (G ,R) , (G ,B), (B ,R) , (B ,G) },
:
: }

Obvious point: Usually C isn't represented explicitly , but by a function.

How to solve our CSP?



How about using a search algorithm?

Define: a search state has variables 1 ... k assigned. Values k+1...n, as yet unassigned.

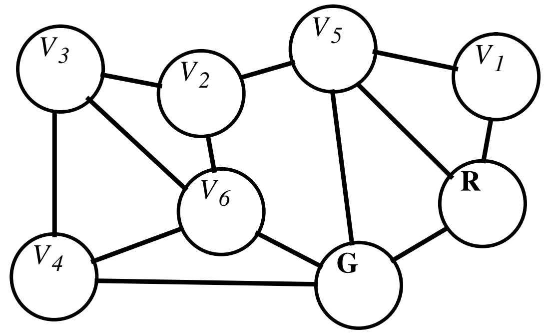
Start state: All unassigned.

Goal state: All assigned, and all constraints satisfied.

Successors of a state with V1 ... Vk assigned and rest unassigned are all states (with V1 .. Vk the same) with Vk+1 assigned a value from D .

Costs on transitions: 0 is fine. We don't care. We just want any solution.

How to solve our CSP?



START =(V1=? V2=? V3=? V4=? V5=? V6=?)

succs(START) =

(V1=R V2=? V3=? V4=? V5=? V6=?)

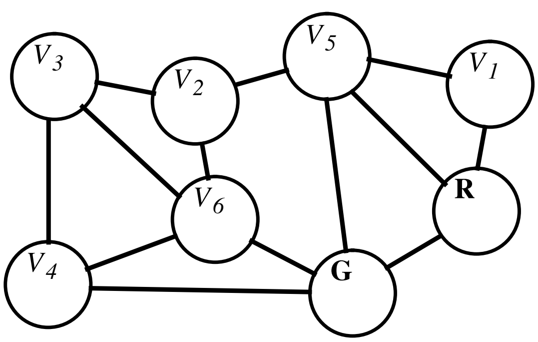
(V1=G V2=? V3=? V4=? V5=? V6=?)

(V1=B V2=? V3=? V4=? V5=? V6=?)

What search algorithms could we use?

It turns out BFS is not a popular choice. Why not?

DFS for CSPs



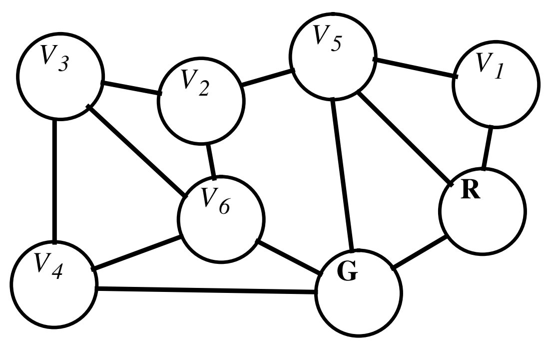
What about DFS?

Much more popular. At least it has a chance of finding an easy answer quickly.

What happens if we we do DFS with the order of assignments as B tried first, then G then R?

This makes DFS look very very stupid!

Blindingly obvious improvement - Consistency Checking: "Backtracking Search"

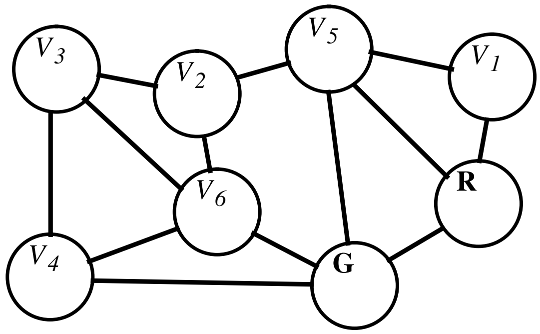


Don't ever try a successor which causes an inconsistency with its neighbors.

Again, what happens if we we do DFS with the order of assignments as B tried first, then G then R?

What's the computational overhead for this?

Obvious improvement - Forward Checking



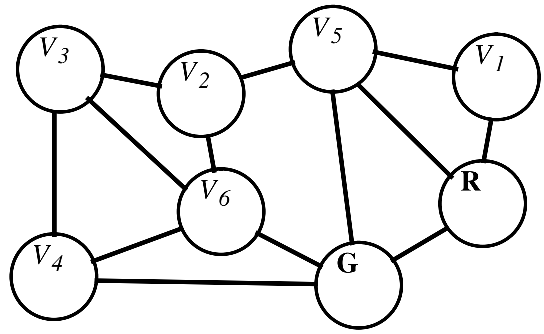
At start, for each variable, record the current set of possible legal values for it.

When you assign a value in the search, update set of legal values for all variables. Backtrack immediately if you empty a variable's constraint set.

Again, what happens if we we do DFS with the order of assignments as B tried first, then G then R?

What's the computational overhead?

Constraint Propagation

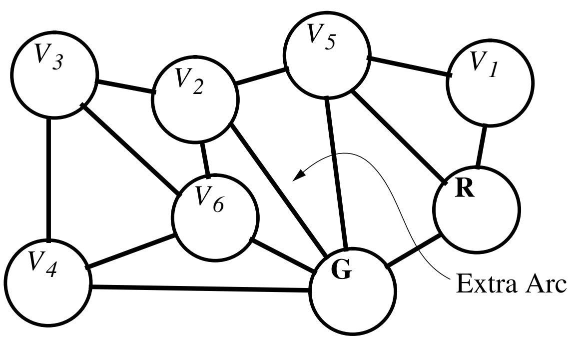


Forward checking computes the domain of each variable independently at the start, and then only updates these domains when assignments are made in the DFS that are directly relevant to the current variable.

Constraint Propagation carries this further. When you delete a value from your domain, check all variables connected to you. If any of them change, delete all inconsistent values connected to them, etc...

Often constraint propagation is only performed at the start. In our example it is useless. But....

Constraint Propagation being non-useless



In this example, constraint propagation solves the problem without search.

Not always that lucky!

Constraint propagation can be done as a preprocessing step. (Cheap).

Or it can be maintained dynamically during the search. Expensive: when you backtrack, you must undo some of your additional constraints.

Some real CSPs

- Graph coloring is a real, and useful, CSP. Applied to problems with many hundreds of thousands of nodes. Not very AI-esque.
- VLSI or PCB board layout.
- Selecting a move in the game of "minesweeper".

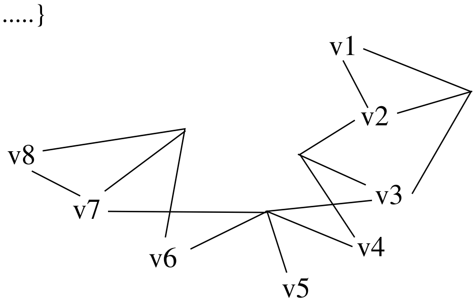
0	0	1			
0	0	1			
0	0	1			
1	1	2			

Which squares have a bomb? Squares with numbers don't. Other squares might. Numbers tell how many of the eight adjacent squares have bombs. We want to find out if a given square can possibly have a bomb....

"Minesweeper" CSP

0	0	1	v1		
0	0	1	v2		
0	0	1	v3		
1	1	2	v4		
v8	v7	v6	v5		

V = { v1 , v2 , v3 , v4 , v5 , v6 , v7 , v8 }
D = { B (bomb) , S (space) }
C = { (v1,v2) : { (B , S) , (S , B) } ,
(v1 , v2 , v3) : { (B ,S,S) , (S ,B,S) , (S , S, B)},
.......}



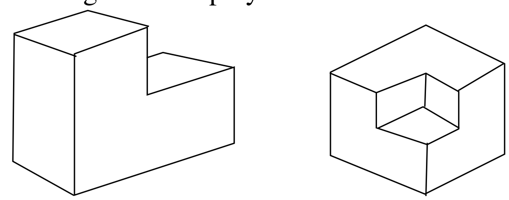
Easy for CSP to show, e.g., that v3=B unsatisfiable.

The Waltz algorithm

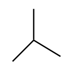
One of the earliest examples of a computation posed as a CSP.

Constraint Propagation works very well.

The Waltz algorithm is for interpreting line drawings of solid polyhedra.



Look at all intersections.



What kind of intersection could this be? A concave intersection of three faces? Or an external convex intersection?

Adjacent intersections impose constraints on each other. Use CSP to find a unique set of labelings. Important step to "understanding" the image.

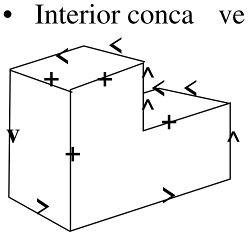
Waltz Alg. on simple scenes

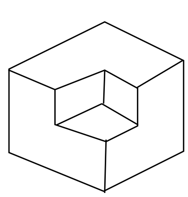
Assume all objects:

- Have no shadows or cracks
- Three-faces vertices
- "General position": no junctions change with small movements of the eye.

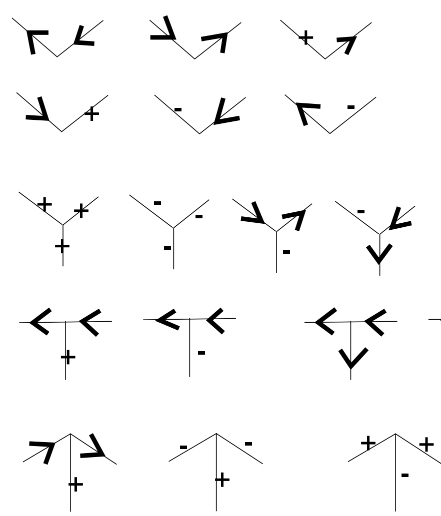
Then each line on image is one of the following:

- Boundary line (edge of an object) (<)
- Interior convex edge (+)
- Interior concave edge (-)





18 legal kinds of junctions



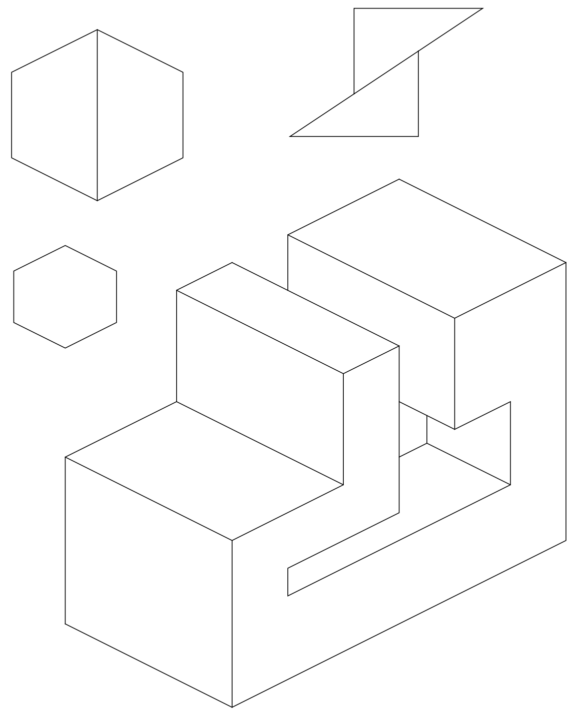
Given a representation of the diagram, label each junction in one of the above manners.

The junctions must be labelled so that lines are labelled consistently at both ends.

Can you formulate that as a CSP?

Constraint Propagation always works perfectly.

Waltz Examples



Scheduling

A very big, important use of CSP methods.

- **Used in many industries. Makes many multi-million dollar decisions.**
- **Used extensively for space mission planning.**
- **Military uses.**

People *really care* about improving scheduling algorithms!

Problems with phenomenally huge state spaces. But for which solutions are needed very quickly.

Many kinds of scheduling problems e.g:

- **Job shop**: Discrete time, weird ordering of operations possible, set of separate jobs.
- **Batch shop**: Discrete or continuous time, restricted operation ordering, grouping is important.
- **Manufacturing cell**: Discrete, automated version of open job shop.

Job Shop scheduling

At a job-shop you make various products. Each product is a "job" to be done. E.G.

Job1 = Make a polished-thing-with-a-hole

Job2 = Paint and drill a hole in a widget

Each job requires several operations. E.G.

Operations for job1: Polish, Drill

Operations for job2: Paint, Drill

Each operation needs several resources. E.G.

Polishing needs the Polishing machine

Polishing needs Pat (a Polishing expert)

Drilling needs the Drill

Drilling needs Pat (also a Drilling expert)

Or Drilling can be done by Chris

Some operations need to be done in a particular order (e.g. Paint after you've drilled)

Job Shop Formalized

A Job Shop problem is a pair (J, RES)

J is a set of jobs $J = \{ j_1, j_2, \ldots j_n \}$

RES is a set of resources $RES = \{ R_1 \ldots R_m \}$

Each job j_l is specified by:

a set of operations $O_l = \{ O^l_1\ O^l_2 \ldots O^l_{n(l)} \}$

and must be carried out between release-date rd_l and due-date dd_l.

and a partial order on operations:
$(O^l_i$ **before** $O^l_j)$, $(O^l_{i'}$ **before** $O^l_{j'})$ etc ...

Each operation O^l_i has a variable start time st^l_i and a fixed duration du^l_i and requires a set of resources. E.G: O^l_i requires $\{ R^l_{i1}, R^l_{i2} \ldots \}$.

Each resource can be accomplished by one of several possible physical resources, e.g. R^l_{ij} might be accomplished by any one of $\{ r^l_{ij1}, r^l_{ij2}, \ldots \}$. Each of the r^l_{ijk}s are a member of RES.

Job Shop Example

j_1 = **polished-hole-thing** $= \{ O^1{}_1 , O^1{}_2 \}$

j_2 = **painted-hole-widget** $= \{ O^2{}_1 , O^2{}_2 \}$

RES={ **Pat,Chris,Drill,Paint,Drill,Polisher** }

$O^1{}_1$ = **polish-thing** *: need resources ...*

$\{ R^1{}_{11}$ = **Pat** , $R^1{}_{12}$ = **Polisher** }

$O^1{}_2$ = **drill-thing** *, need resources ....*

$\{R^1{}_{21}=(r^1{}_{211}$ =**Pat** *or*
$r^1{}_{212}$ =**Chris**),$R^1{}_{22}$ =**drill** }

$O^2{}_1$ = **paint-widget,** *need resources ....*

$\{R^2{}_{11}$ = **Paint** }

$O^2{}_2$ = **drill-widget** *, need resources ....*

$\{R^2{}_{21}=(r^2{}_{211}$ =**Pat** *or*
$r^2{}_{212}$ =**Chris**),$R^2{}_{22}$ =**drill** }

Precedence constraints *:* $O^2{}_2$ *before* $O^2{}_1$. All operations take one time unit $du^l{}_i = 1$ forall i,l. Both jobs have release-date $rd^l = 0$ and due-date $dd^l = 1$.

Job-shop: the Variables and Constraints

Variables:

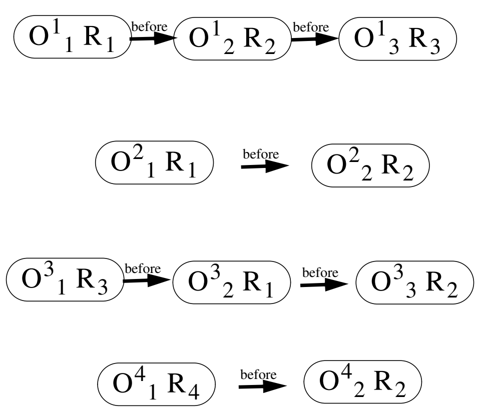
- The operation start times $st^l{}_i$
- The resources $R^l{}_{ij}$ (usually these are obvious from the definition of $O^l{}_i$. Only need to be assigned values when there are alternative physical resources available, e.g. **Pat** or **Chris** for operating the **drill**).

Constraints:

- Precedence constraints. (Some $O^l{}_i$s must be before some other $O^l{}_j$s).
- Capacity constraints. There must never be a pair of operations with overlapping periods of operation that use the same resources.

Non-challenging question. Can you schedule our Job-shop?

A slightly bigger example



See Sadeh+Fox Page 6 Fig 2

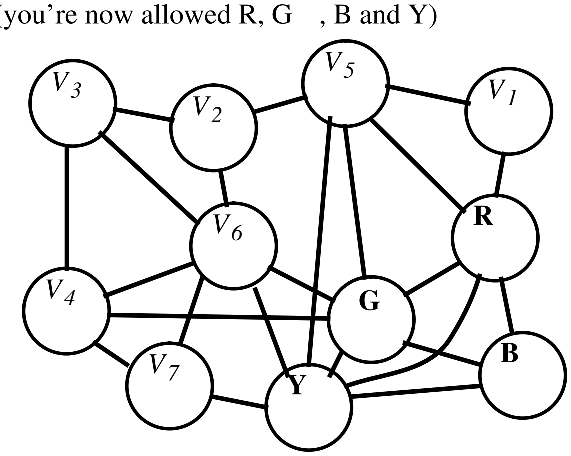
4 jobs. Each 3 units long. All jobs have release date 0 and due date 15. All operations use only one resource each.

This problem is still very small compared with real ones.

Further CSP techniques.

Let's look at some other important CSP methods. Keep the job-shop example in mind.

Here's another graph-coloring example (you're now allowed R, G, B and Y)

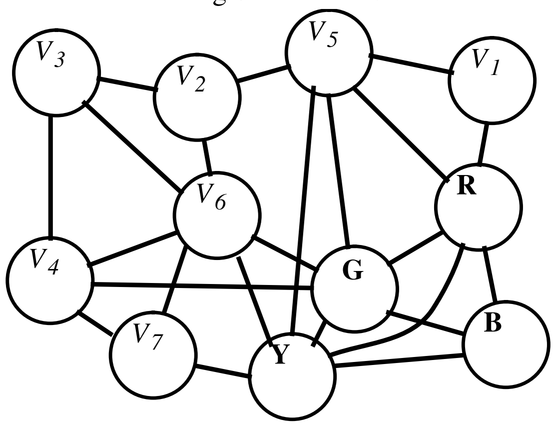


Why do we insist on trying v1 first, then v2 etc?

Are there any heuristics for choosing variable orderings? Yes. And they are called: *Variable Ordering Heuristics*.

General purpose Variable Ordering Heuristics

1. Most constrained variable.
2. Most constraining variable.



There also exist: **Value ordering heuristics.**

A good general purpose one is "least-constrained-value". Choose the value which causes the smallest reduction in available values for your neighbors.

General purpose CSP algorithm

(From Sadeh+Fox)

1. If all values have been succesfully assigned then stop, else go on to 2.
2. Apply the consistency enforcing procedure (e.g. forward-checking if feeling computationally mean, or constraint propagation if extravagant. There are other possibilities too)
3. If a deadend is detected then backtrack (simplest case: DFS-type backtrack. Other options can be tried too). Else go on to step 4.
4. Select the next variable to be assigned (using your variable ordering heuristic)
5. Select a promising value (using your value ordering heuristic)
6. Create a new search state. Cache the info you need for backtracking. And go back to 1.

Job-shop example. Consistency enforcement.

Sadeh claims that generally forward-checking is better, computationally, than full constraint propagation. But it can be supplemented with a Job-shop specific TRICK.

The precedence constraints (i.e. the available times for the operations to start due to the ordering of operations) can be computed exactly, given a partial schedule, very efficiently.

Variable Orderings

Most constrained variable ordering heuristic helps. But often not very useful.

O^1_1 O^1_2 O^1_3 O^3_1 O^3_2 O^3_3 are all jointly most constrained. But O^1_2 and O^3_3 are much harder than O^1_3 and O^3_1. "Survivability" measure is better (see Sadeh+Fox).

Sadeh's variable ordering heuristic

Uses probabilistic reasoning.

For each job, For each resource, for each time, estimates the probability that that resource will be needed by that job at that time.

Thus computes, for each resource, for each time, the overall demand for that resource.

ORR Heuristic: the most critical operation is the one that relies most on the the most contended resource/time interval.

Page 29

Sadeh's value ordering heuristic

Uses the same demand estimates in a different way.

Estimates the probability that a given assignment of an operation and time to a resource "survives" the rest of the scheduling.

EMPIRICAL COMPARISON:

DSR+ABT: Most constrained variable and minimum-spanning-tree-based value ordering.

SMU: A similar method to Sadeh's; without probabilistic approximations.

ORR+FSS: With Sadeh's probabilistic approximations.

Page 30

Reactive CSP solutions

Say you have a built a large schedule.

Disaster! half way through execution, one of the resources breaks down. We have to reschedule!

Bad to have to wait 15 minutes for the scheduler to make a new suggestion.

Important area of research: efficient schedule repair algorithms.

Question: If you expect that resources may sometimes break, what could a scheduling program do to take that into account?

Question: Why has none of this lecture used A*?

Page 31

Other approaches. And What You Should Know

Other Approaches:

- Hill-climbing, Tabu-search, Simulated annealing, Genetic Algorithms. (to be discussed later)

What you should know:

+How to formalize problems as CSPs
+Backtracking Search, Forward Checking, Constraint Propagation
+The Waltz algorithm
+You should understand and appreciate the way job-shop scheduling is formalized. It is an excellent representative example of how important well-studied constraint satisfaction problems are represented.
+Understand examples of Variable ordering and Value ordering heuristics
+Appreciate, without detail, how probabilistic heuristics can help.

In those cases where your lecturer or these handouts are too incomprehensible, consult Chap 5 of the Russell handout. Winston's "Artificial Intelligence" book has good discussion of constraint satisfaction and Waltz algorithm.

Page 32