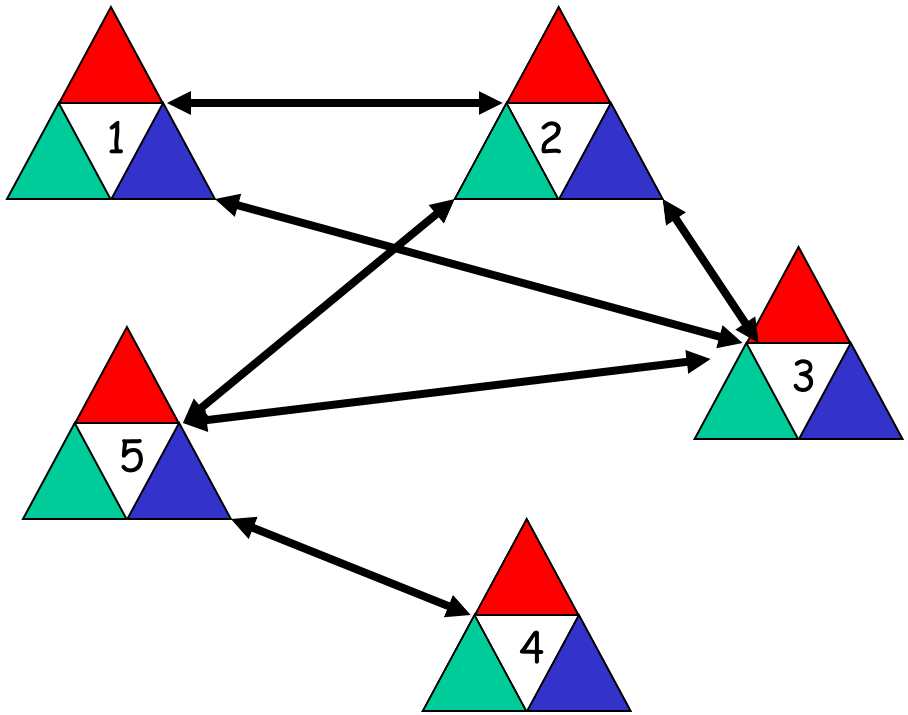
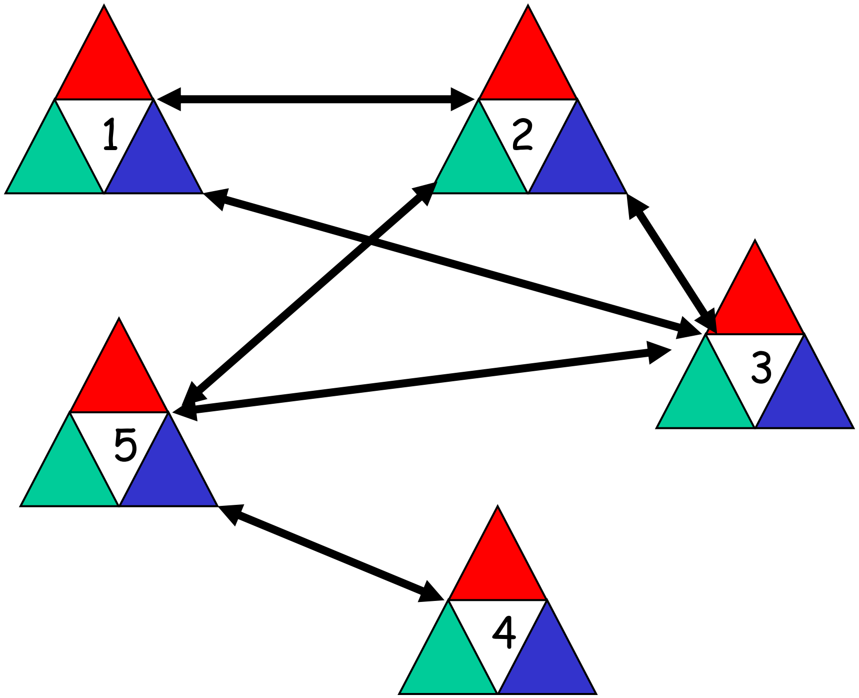
Chronological Backtracking (BT)

An example problem



Colour each of the 5 nodes, such that if they are adjacent, they take different colours

Representation (csp1)



- variables v1, v2, v3, v4, and v5
- domains d1, d2, d3, d4, and d5
- domains are the three colours {R,B,G}
- constraints
    - $v1 \neq v2$
    - $v1 \neq v3$
    - $v2 \neq v3$
    - $v2 \neq v5$
    - $v3 \neq v5$
    - $v4 \neq v5$

Assign a value to each variable, from its domain, such that the constraints are satisfied

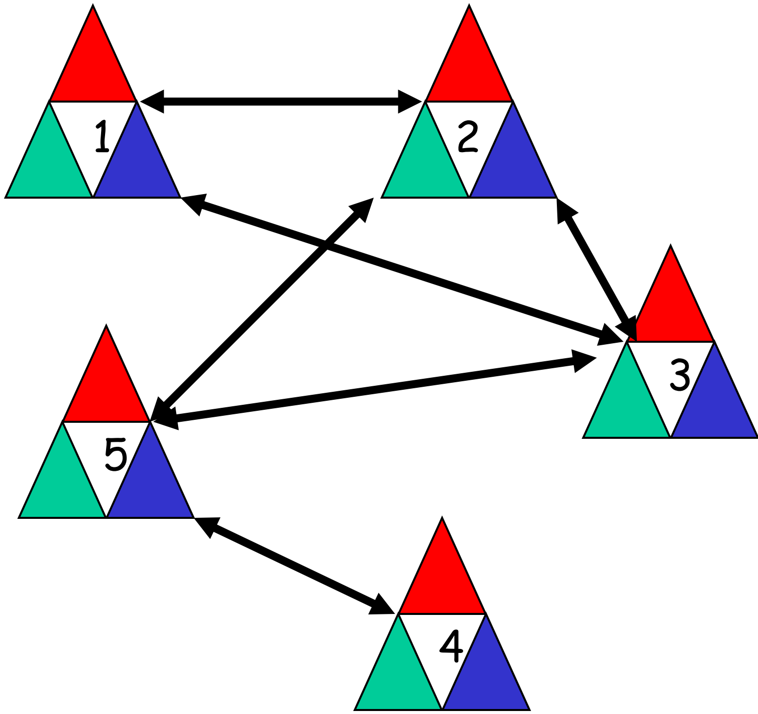
CSP = (V,D,C)

Chronological Backtracking (BT)

As a pair of mutually recursive functions

bt-label

```
bt-label(i,v,d,cd,n)
begin
  if i > n
  then return "solution"
  else begin
         consistent := false;
         for v[i] ∈ cd[i] while ¬consistent
         do begin
              consistent := true;
              for h := 1 to i-1 while consistent
              do consistent := check(v,i,h);
              if ¬consistent
              then cd[i] := cd[i] \ v[i];
              end
         if consistent
         then bt-label(i+1,v,d,cd,n)
         else bt-unlabel(i,v,d,cd,n)
end
```



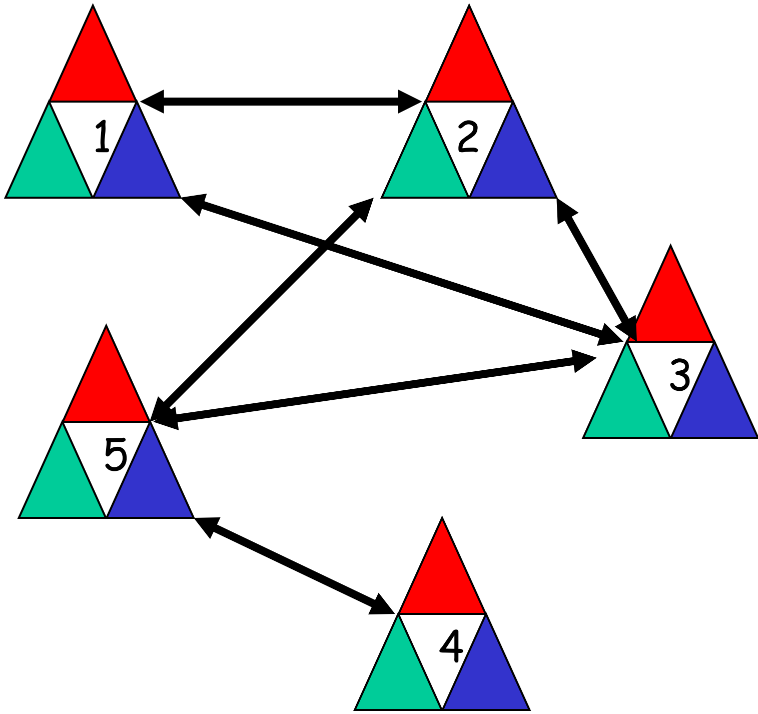
i the index of the current variable
h the index of a past variable (local)
v an array of variables to be instantiated
d an array of domains
cd an array of current domains
n the number of variables

constraints are binary.
consistent is boolean (local)
bt(v,d,n) = for i := 1 to n cd[i] := d[i];
            return bt-label(1,v,d,cd,n)

Chronological Backtracking (BT)

bt-label

```
bt-label(i,v,d,cd,n)
begin
  if i > n
  then return "solution"
  else begin
       consistent := false;
       for v[i] ∈ cd[i] while ¬consistent
       do begin
          consistent := true;
          for h := 1 to i-1 while consistent
          do consistent := check(v,i,h);
          if ¬consistent
          then cd[i] := cd[i] \ v[i];
          end
       if consistent
       then bt-label(i+1,v,d,cd,n)
       else bt-unlabel(i,v,d,cd,n)
end
```



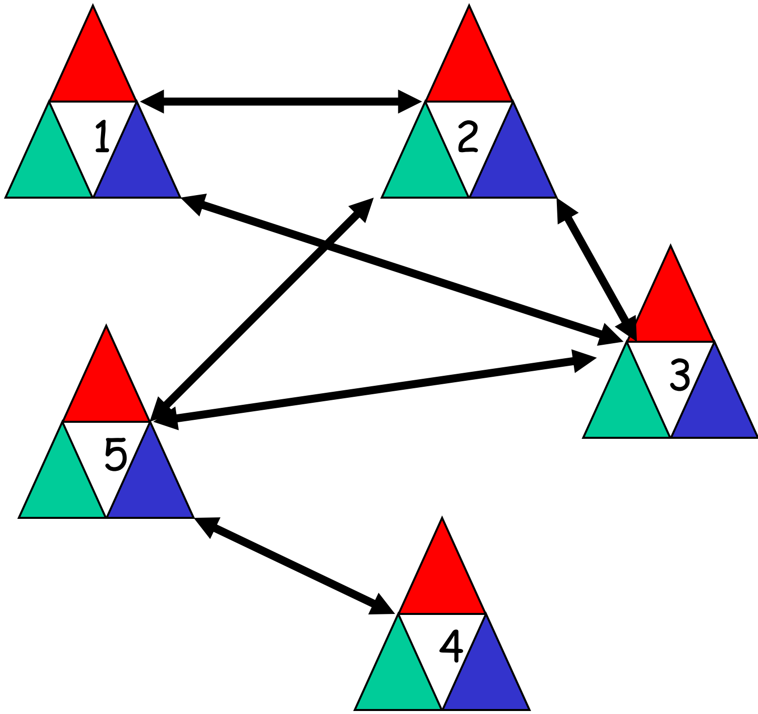
Find a consistent instantiation for v[i]

Chronological Backtracking (BT)

bt-label

```
bt-label(i,v,d,cd,n)
begin
  if i > n
  then return "solution"
  else begin
        consistent := false;
        for v[i] ∈ cd[i] while ¬consistent
        do begin
            consistent := true;
            for h := 1 to i-1 while consistent
            do consistent := check(v,i,h);
            if ¬consistent
            then cd[i] := cd[i] \ v[i];
            end
        if consistent
        then bt-label(i+1,v,d,cd,n)
        else bt-unlabel(i,v,d,cd,n)
end
```

← check backwards, from current to past

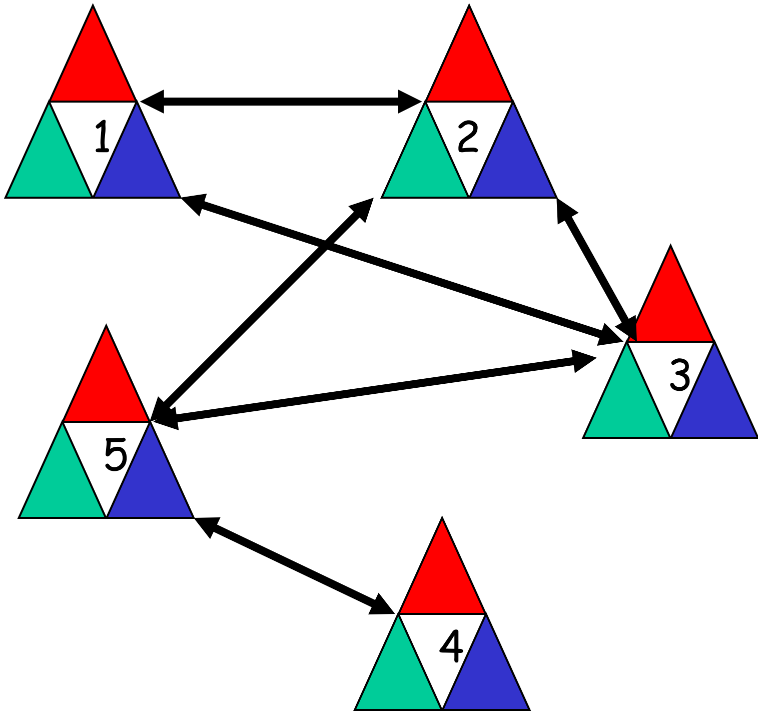


Chronological Backtracking (BT)

bt-label

```
bt-label(i,v,d,cd,n)
begin
  if i > n
  then return "solution"
  else begin
       consistent := false;
       for v[i] ∈ cd[i] while ¬consistent
       do begin
            consistent := true;
            for h := 1 to i-1 while consistent
            do consistent := check(v,i,h);
            if ¬consistent
            then cd[i] := cd[i] \ v[i];
            end
       if consistent
       then bt-label(i+1,v,d,cd,n)
       else bt-unlabel(i,v,d,cd,n)
end
```

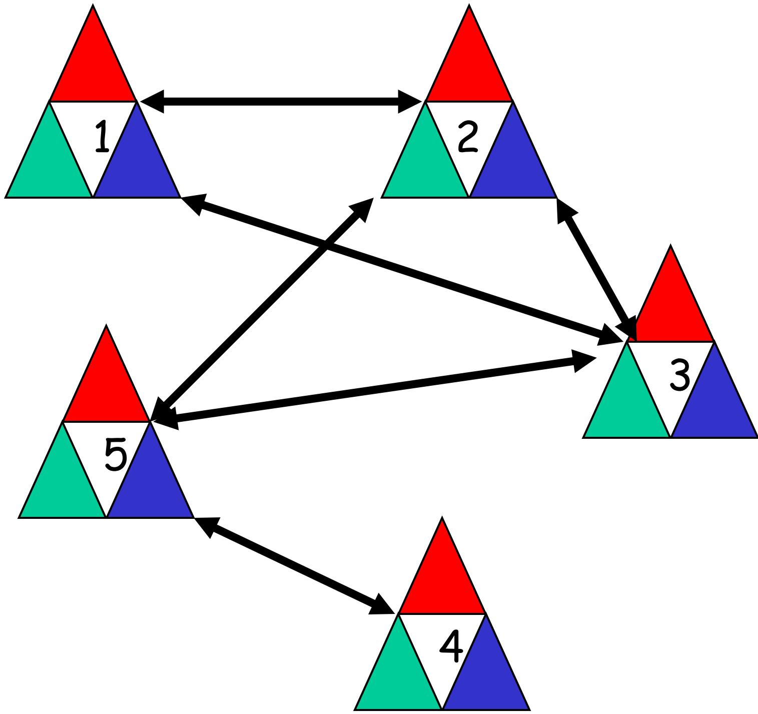
← recurse



Chronological Backtracking (BT)

bt-unlabel

```
bt-unlabel(i,v,d,cd,n)
begin
  if i = 0
  then return "fail"
  else begin
          h := i - 1;
          cd[h] := cd[h] \ v[h];
          cd[i] := d[i];
          if cd[h] ≠ nil
          then bt-label(h,v,d,cd,n)
         else bt-unlabel(h,v,d,cd,n)
         end
end
```

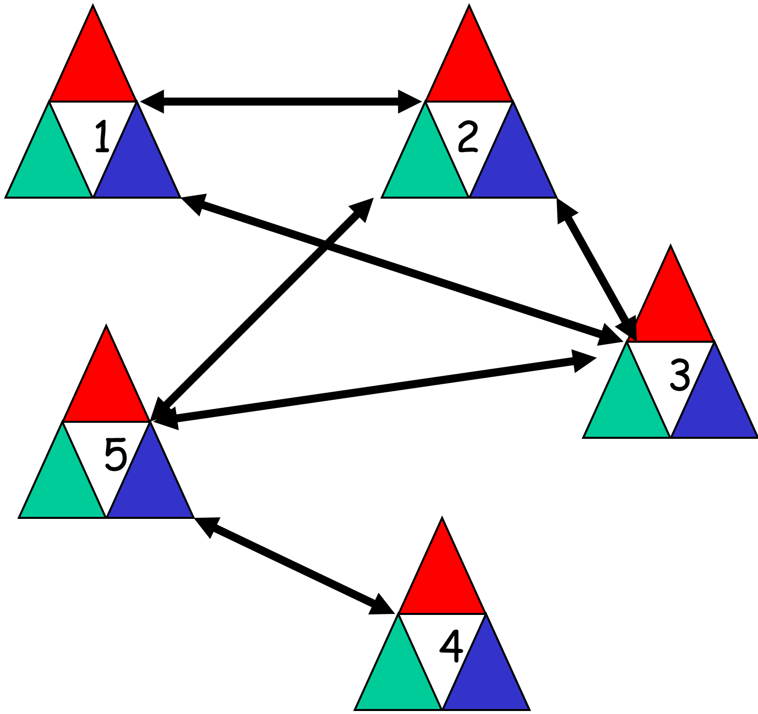


Chronological Backtracking (BT)

bt-unlabel

```
bt-unlabel(i,v,d,cd,n)
begin
  if i = 0
  then return "fail"
  else begin
       h := i - 1;
       cd[h] := cd[h] \ v[h];
       cd[i] := d[i];
       if cd[h] ≠ nil
       then bt-label(h,v,d,cd,n)
      else bt-unlabel(h,v,d,cd,n)
      end
end
```

Past variable

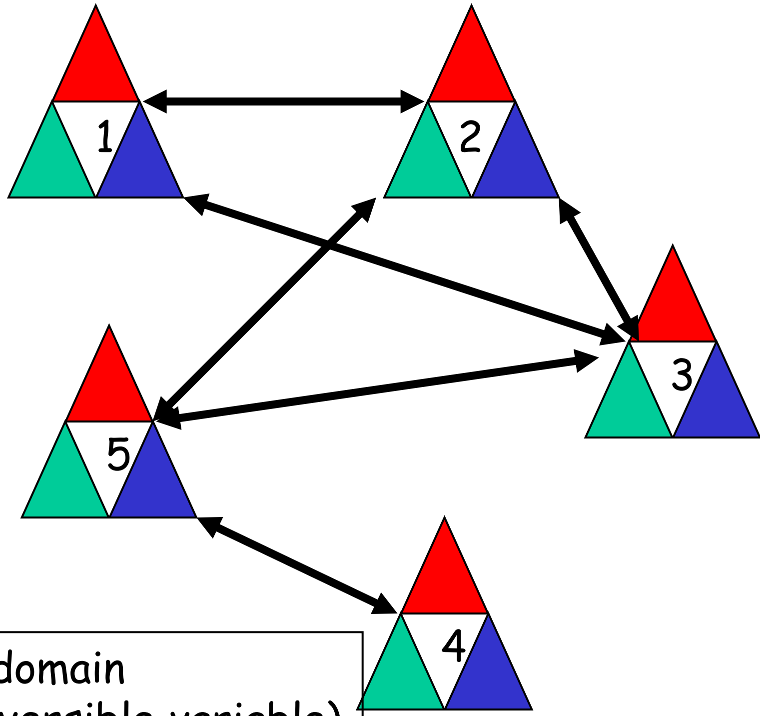


Chronological Backtracking (BT)

bt-unlabel

```
bt-unlabel(i,v,d,cd,n)
begin
  if i = 0
  then return "fail"
  else begin
         h := i – 1;
         cd[h] := cd[h] \ v[h];
         cd[i] := d[i];
         if cd[h] ≠ nil
         then bt-label(h,v,d,cd,n)
        else bt-unlabel(h,v,d,cd,n)
        end
end
```

Reset domain
(bactrackable/reversible variable)

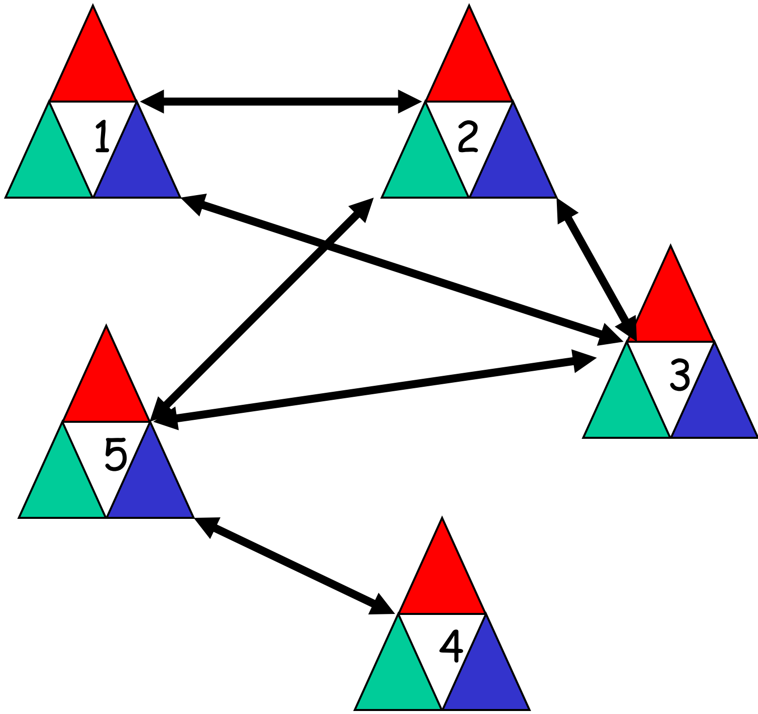


Chronological Backtracking (BT)

bt-unlabel

```
bt-unlabel(i,v,d,cd,n)
begin
  if i = 0
  then return "fail"
  else begin
         h := i - 1;
         cd[h] := cd[h] \ v[h];
         cd[i] := d[i];
         if cd[h] ≠ nil
         then bt-label(h,v,d,cd,n)
        else bt-unlabel(h,v,d,cd,n)
        end
end
```

recurse



Iterative implementation of BT

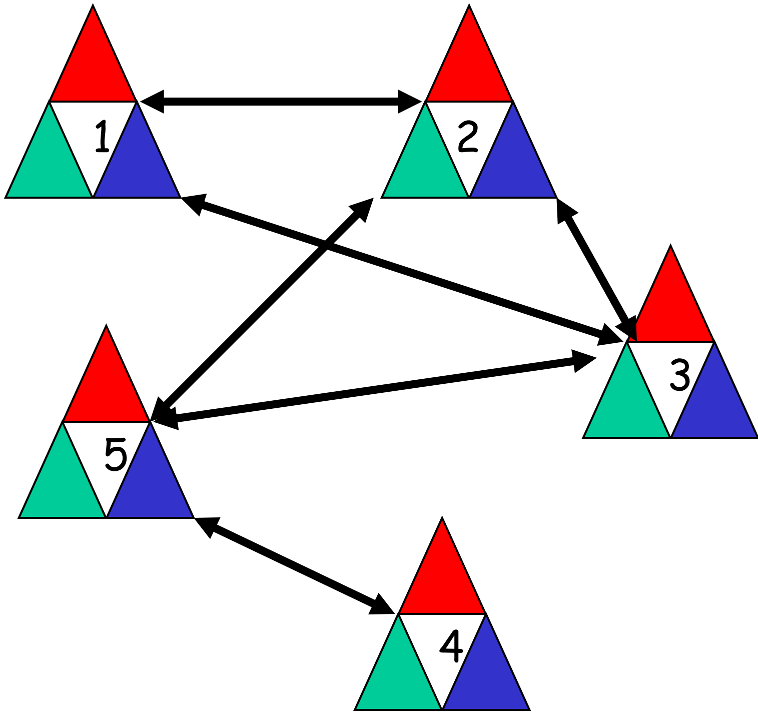
```
search(n,status)
begin
  consistent := true;
  status := "unknown";
  i := 1;
  while status = "unknown"
  do begin
       if consistent
       then i := label(i,consistent)
       else i := unlabel(i,consistent);
       if i > n
      then status = "solution"
      else if i = 0
            then status := "impossible"
      end
end
```

```
bt-label(i,consistent)
begin
  consistent := false;
  for v[i] in cd[i] while ¬consitent
  do begin
       consistent := true;
       for h := 1 to i-1 while consistent
        do consistent := check(v,i,h);
       if ¬consistent
       then cd[i] := cd[i] \ v[i];
       end;
  if consistent then return i+1 else return I
end
```

```
bt-unlabel(i,consistent)
begin
  h := i-1;
  cd[i] := d[i];
  cd[h] := cd[h] \ v[h];
  consistent := cd[h] ≠ nil
  return h;
end
```

This is more realistic. Why?

Chronological Backtracking (BT)



Three Different Views of Search

a trace
a tree
past, current, future

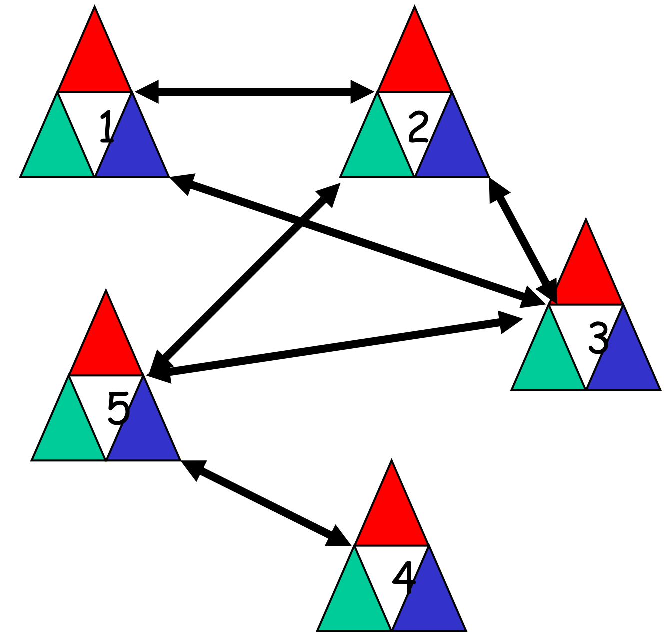
A Trace of BT (assume domain ordered {R,B,G})

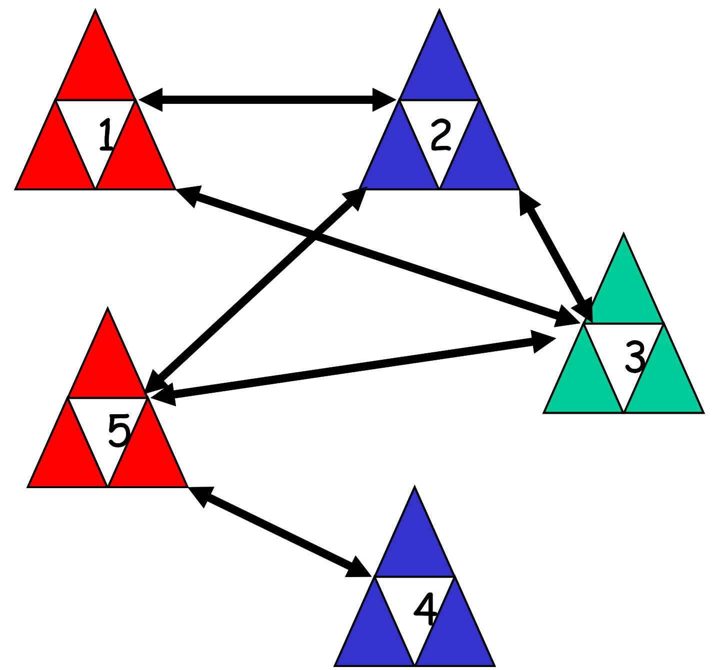
- V[1] := R
- V[2] := R
  - check(v[1],v[2]) fails
- V[2] := B
  - check(v[1],v[2]) good
- V[3] := R
  - check(v[1],v[3]) fails
- V[3] := B
  - check(v[1],v[3]) good
  - check(v[2],v[3]) fails
- V[3] := G
  - check(v[1],v[3]) good
  - check(v[2],v[3]) good
- V[4] := R
- V[5] := R
  - check(v[2],v[5]) good
  - check(v[3],v[5]) good
  - check(v[4],v[5]) fails
- V[5] := B
  - check(v[2],v[5]) fails
- V[5] := G
  - check(v[2],v[5]) good
  - check(v[3],v[5]) fails

- backtrack!

- V[4] := B
- V[5] := R
  - check(v[2],v[5]) good
  - check(v[3],v[5]) good
  - check(v[4],v[5]) good

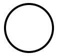
- solution found





16 checks and 12 nodes

A Tree Trace of BT (assume domain ordered {R,B,G})



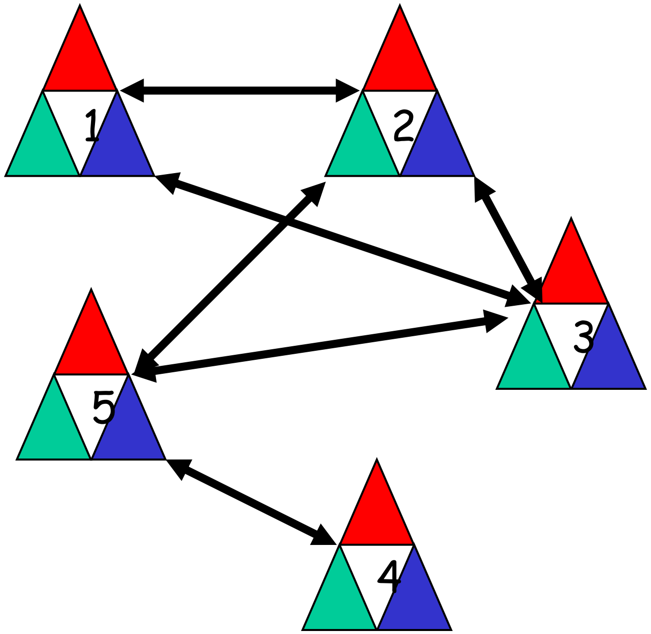
v1

v2

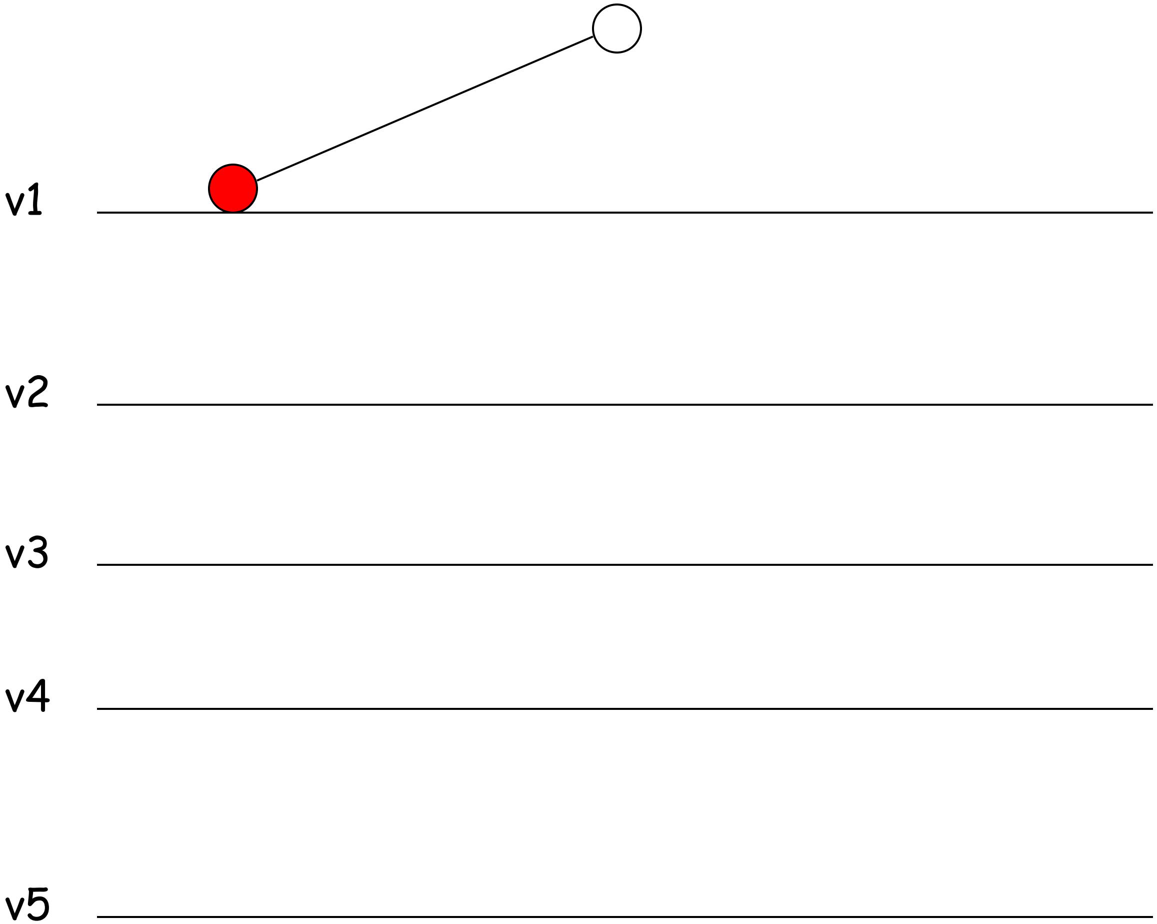
v3

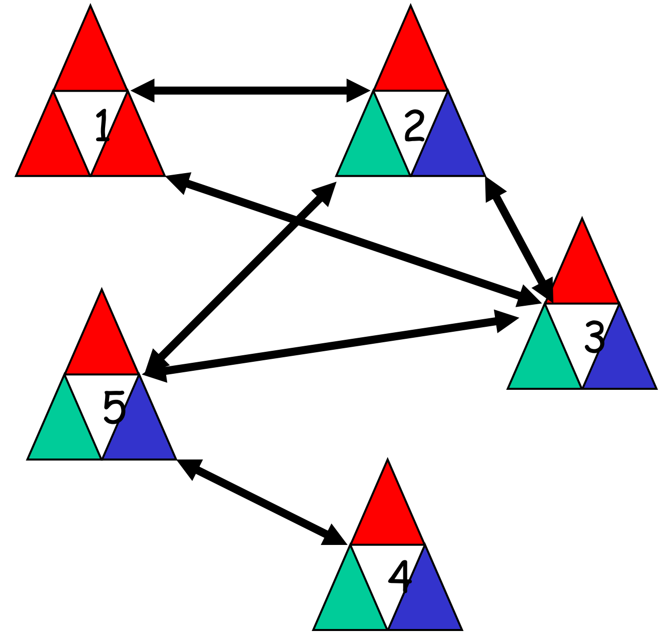
v4

v5

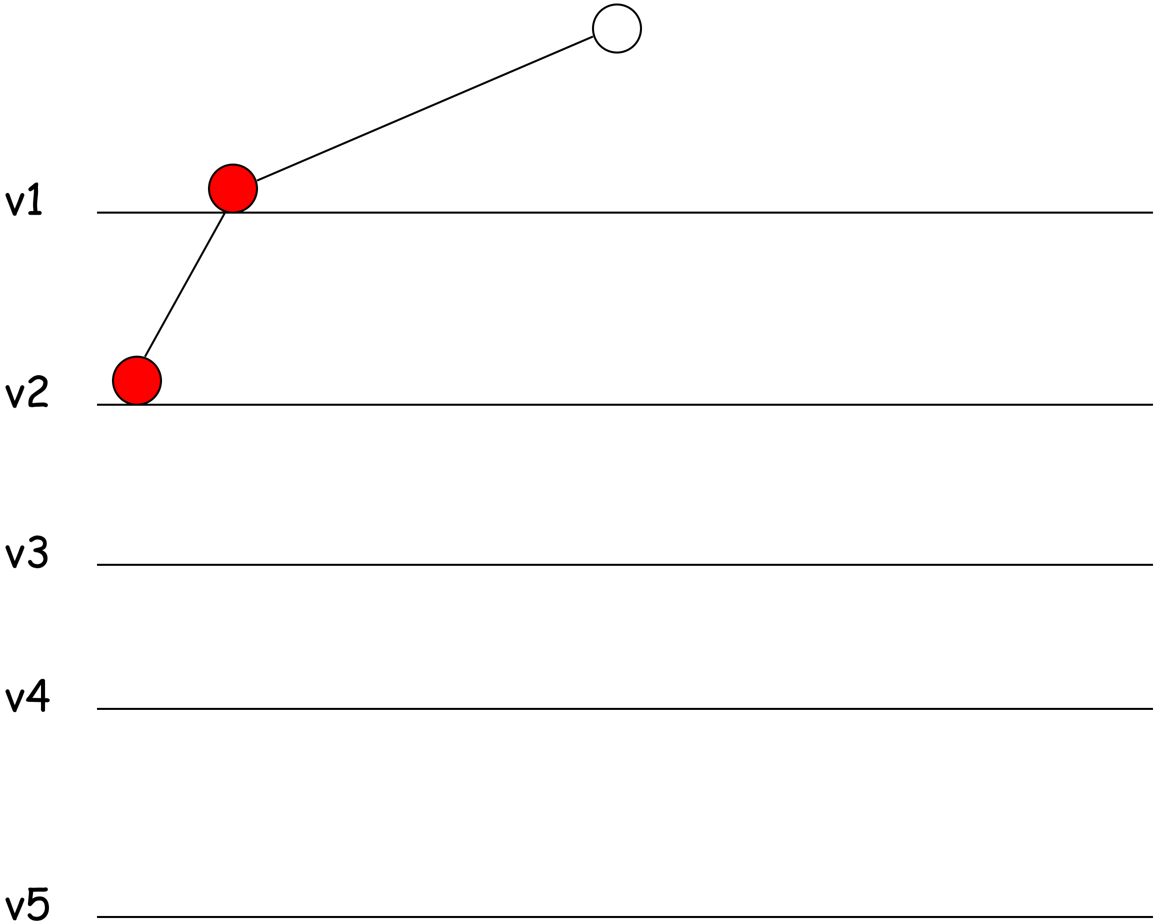


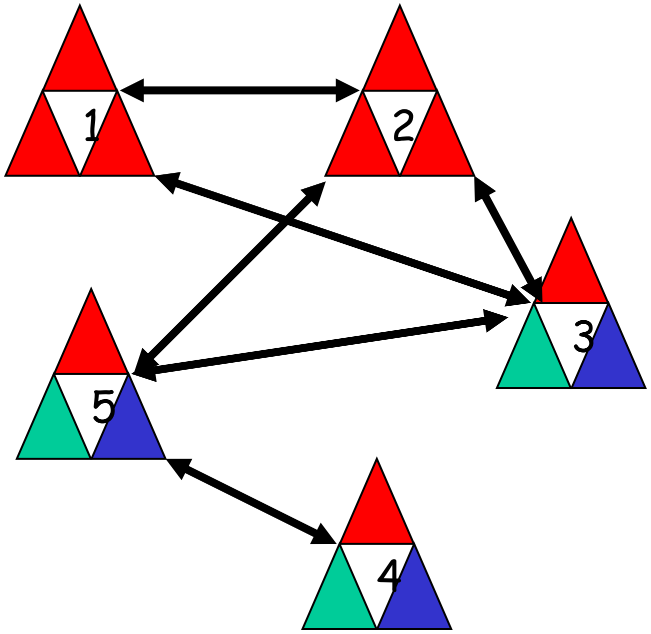
A Tree Trace of BT (assume domain ordered {R,B,G})



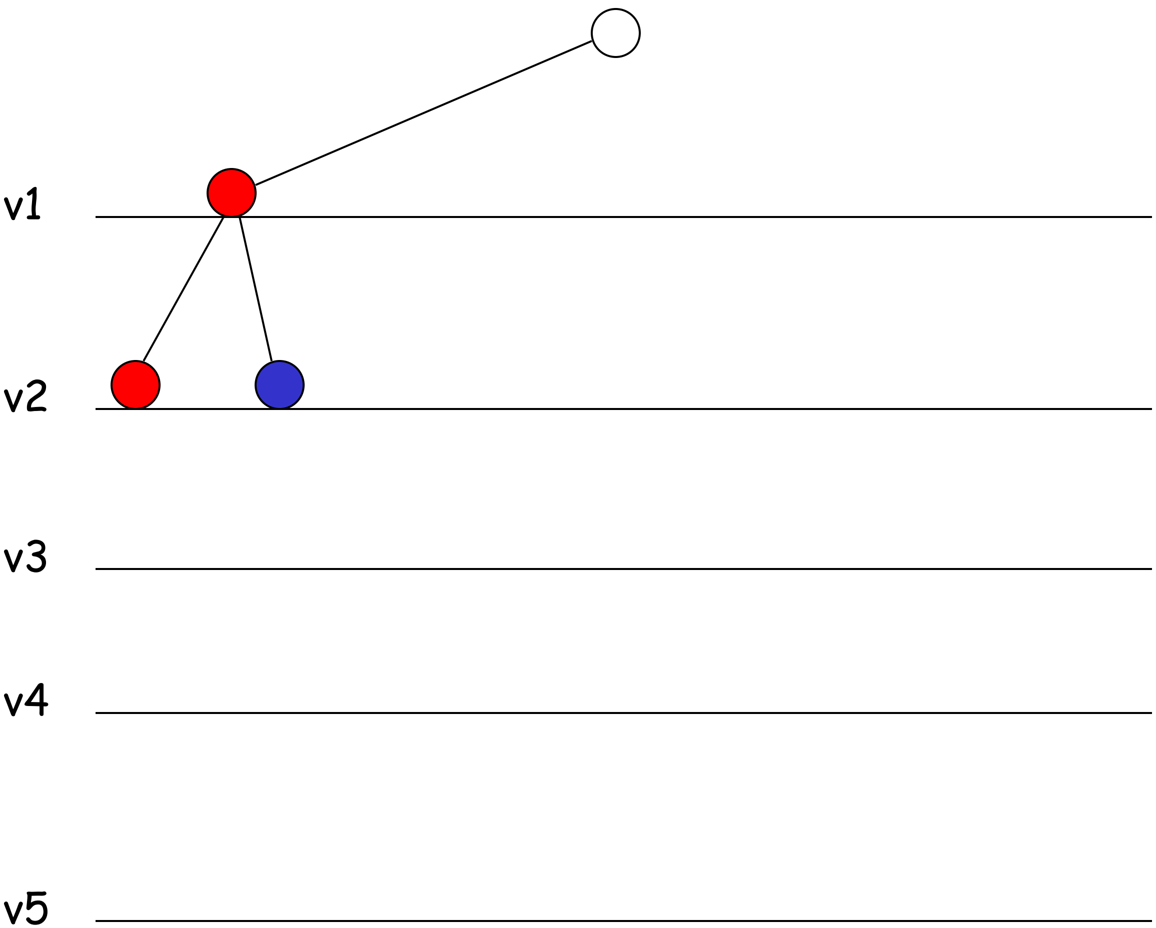


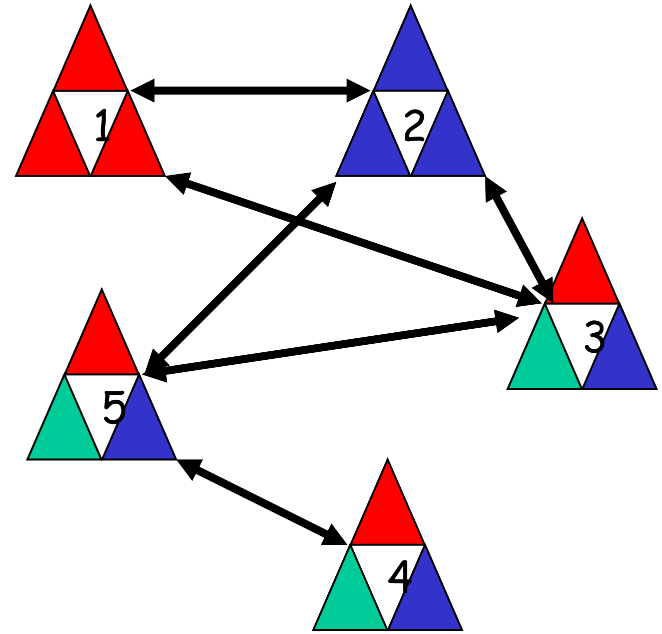
A Tree Trace of BT (assume domain ordered {R,B,G})



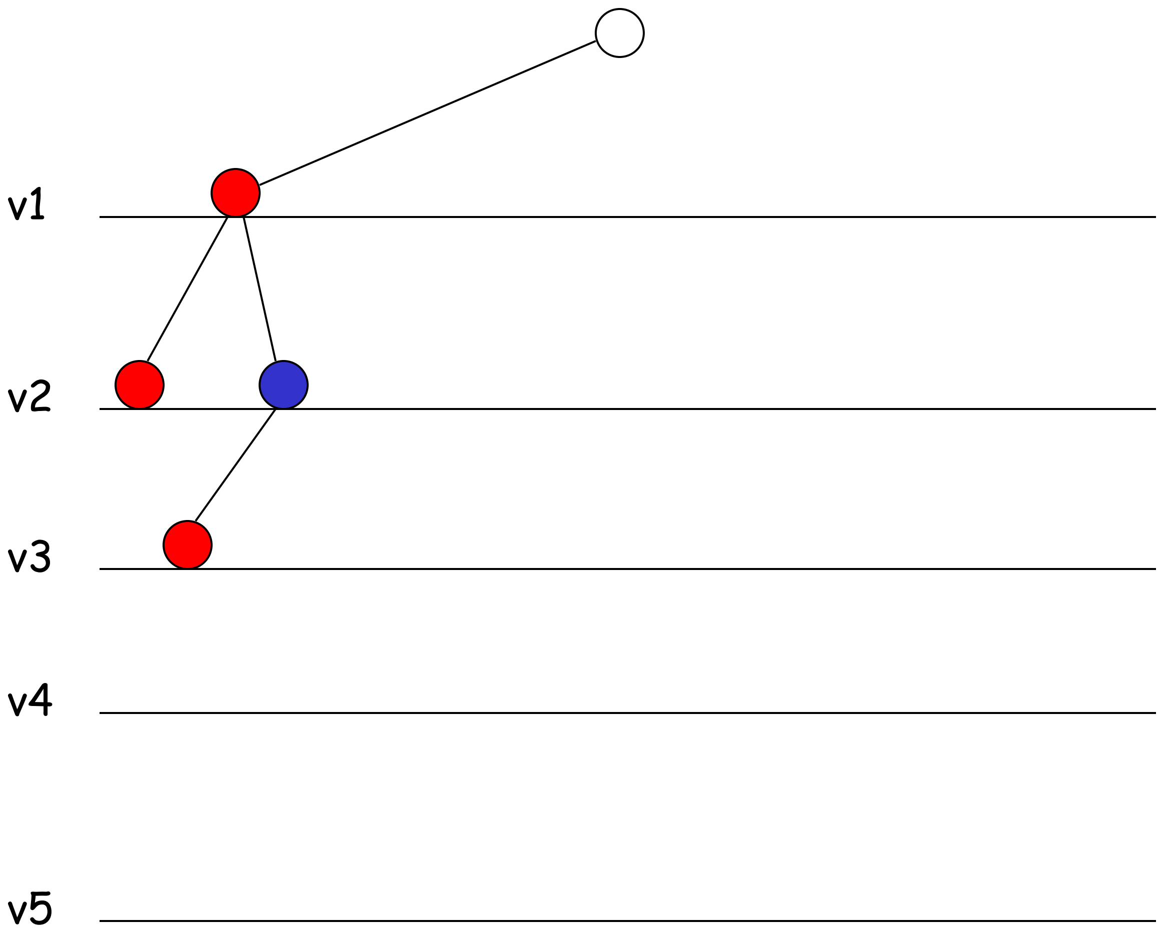


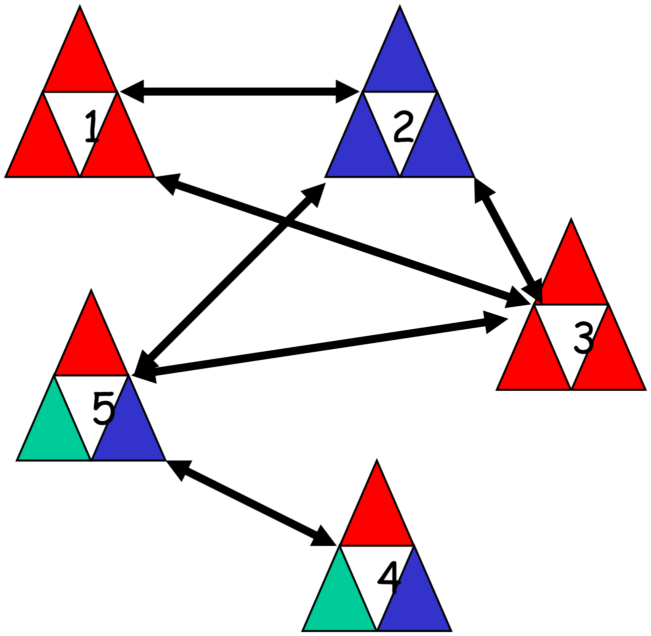
A Tree Trace of BT (assume domain ordered {R,B,G})



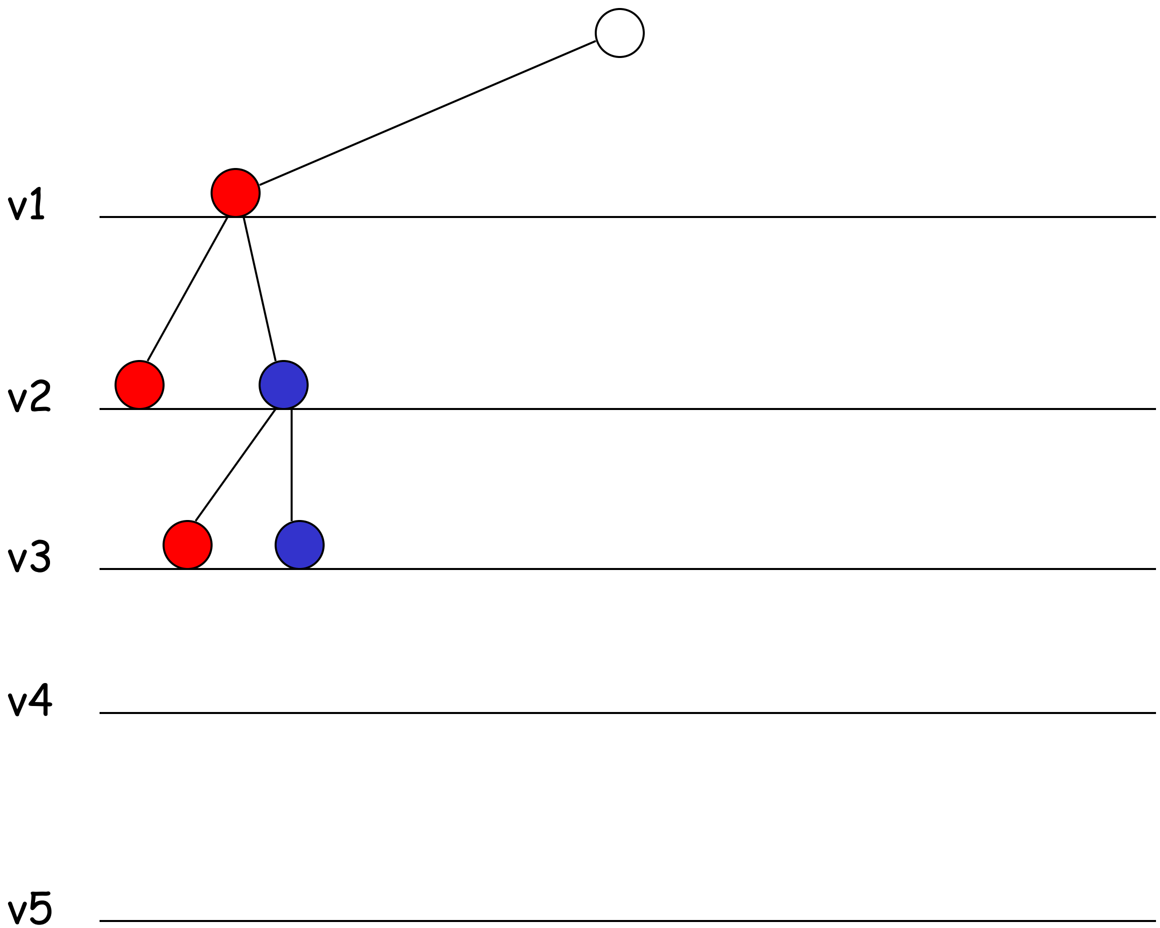


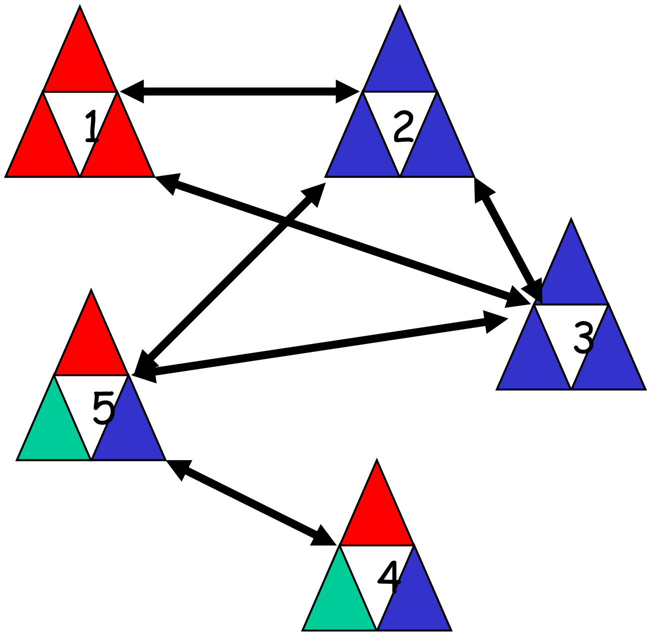
A Tree Trace of BT (assume domain ordered {R,B,G})



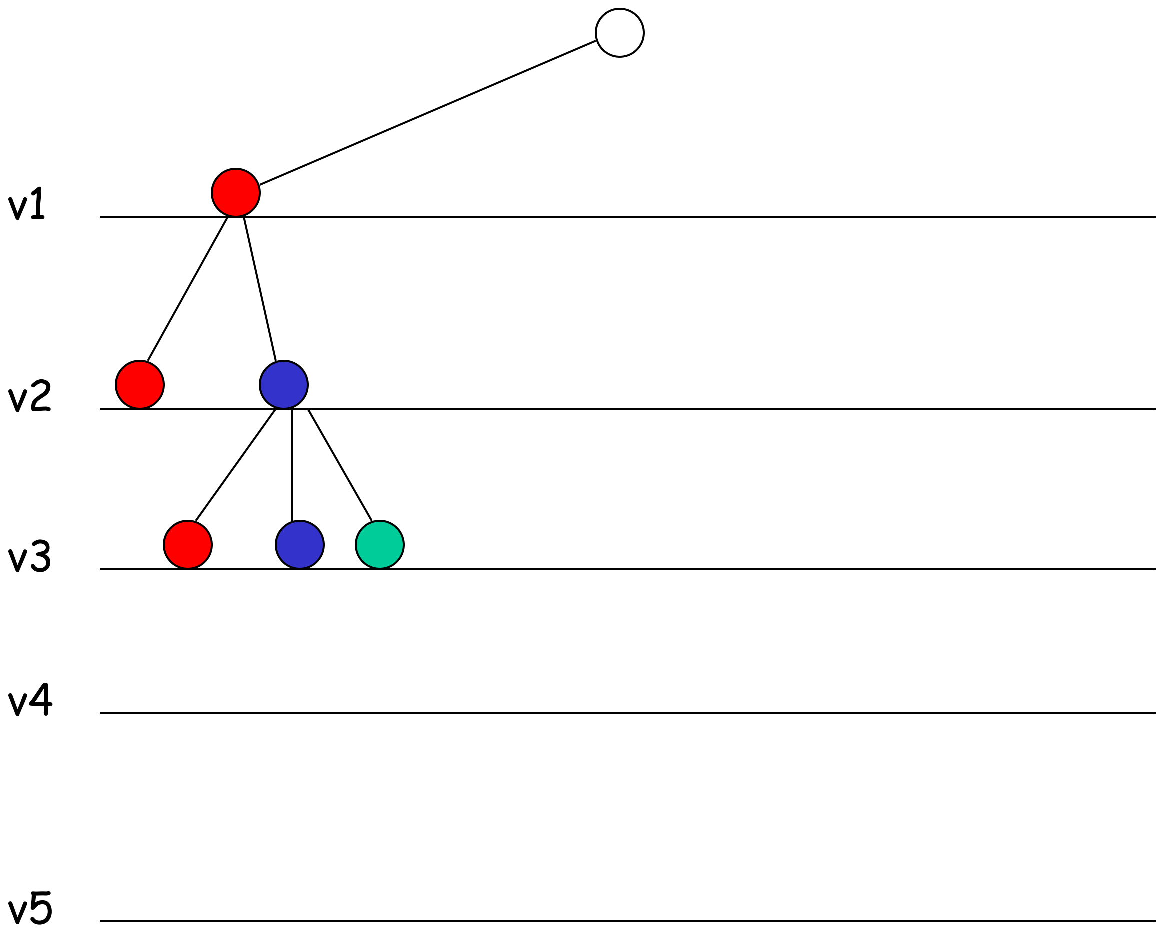


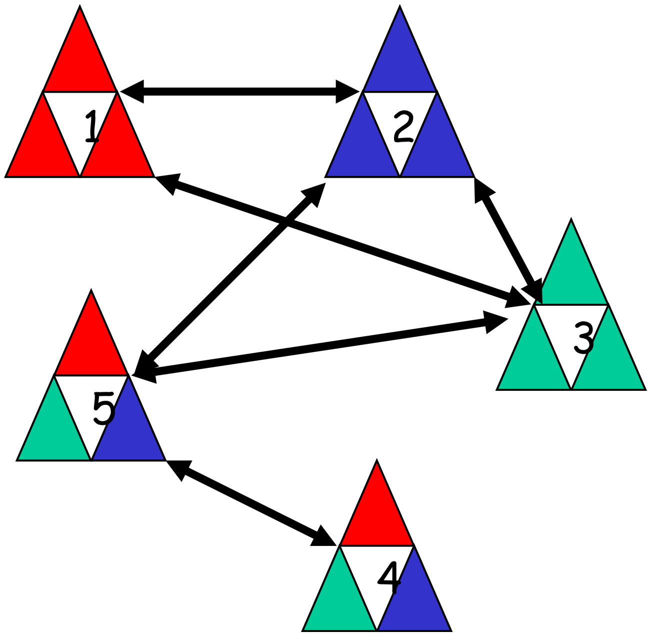
A Tree Trace of BT (assume domain ordered {R,B,G})



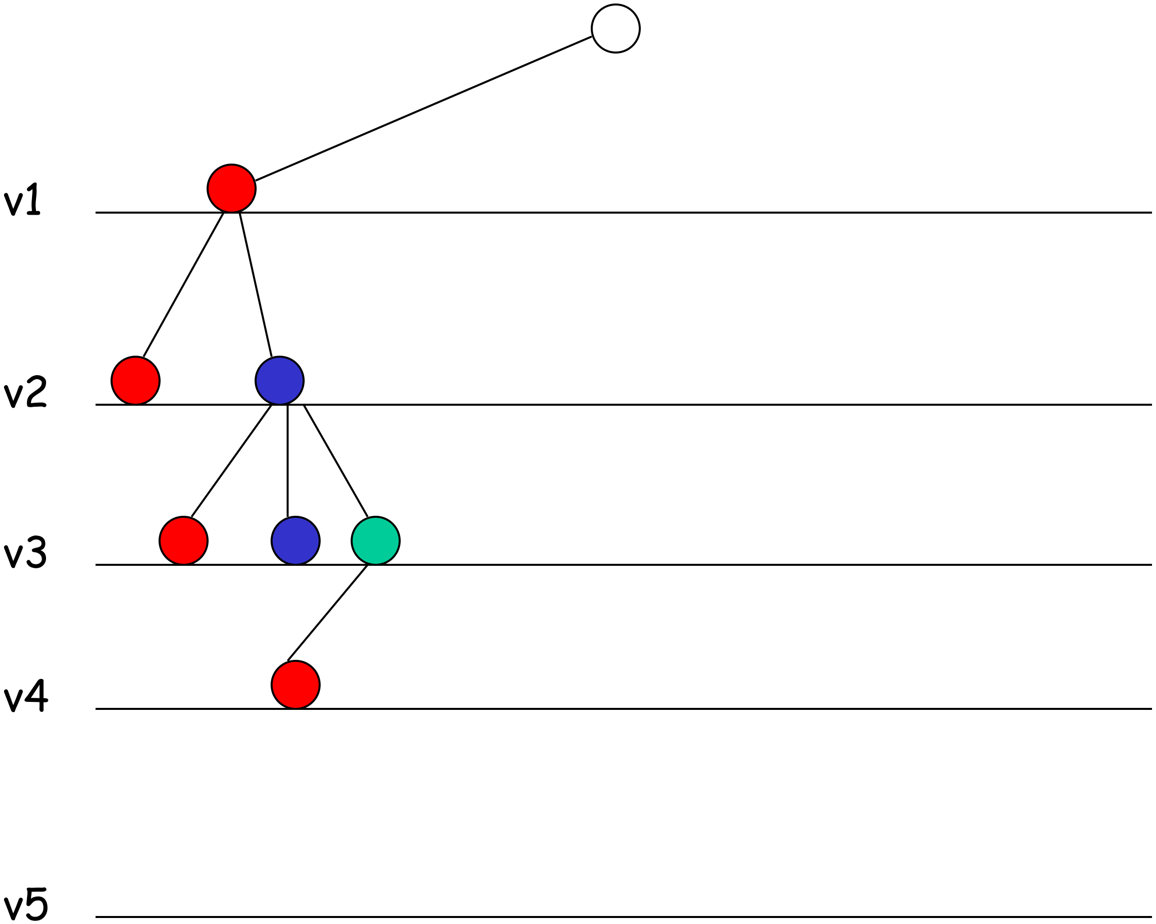


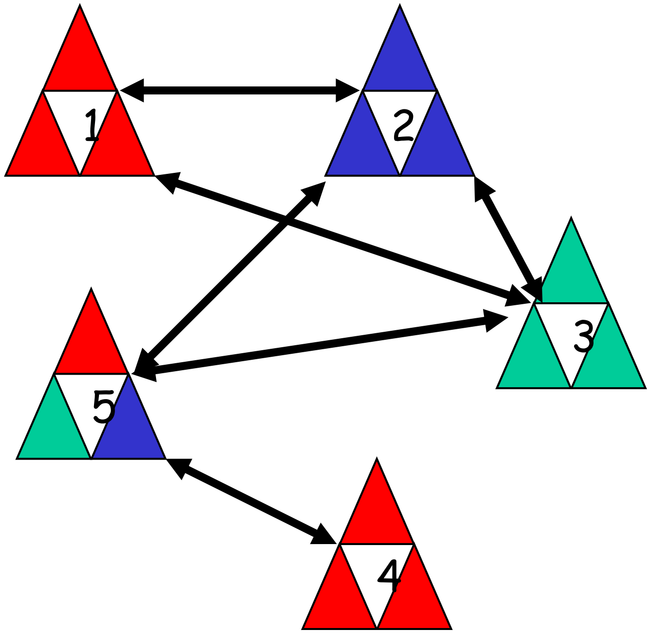
A Tree Trace of BT (assume domain ordered {R,B,G})



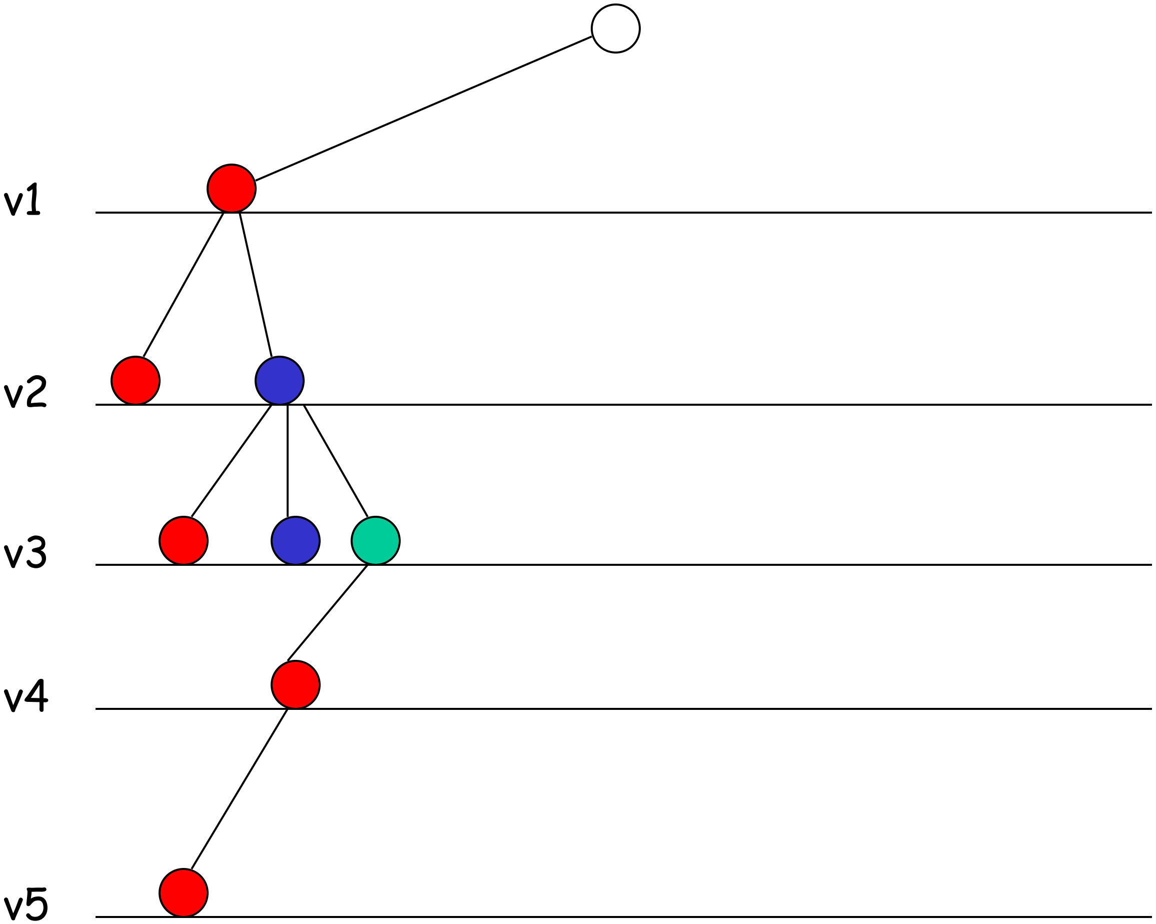


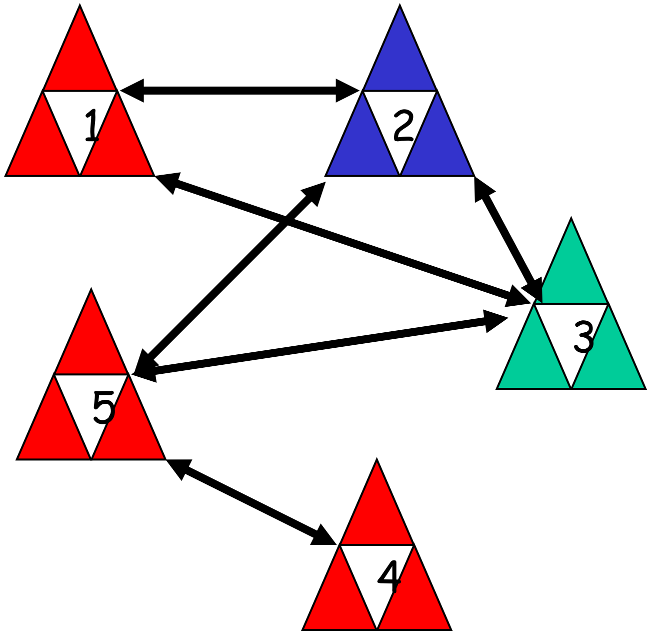
A Tree Trace of BT (assume domain ordered {R,B,G})



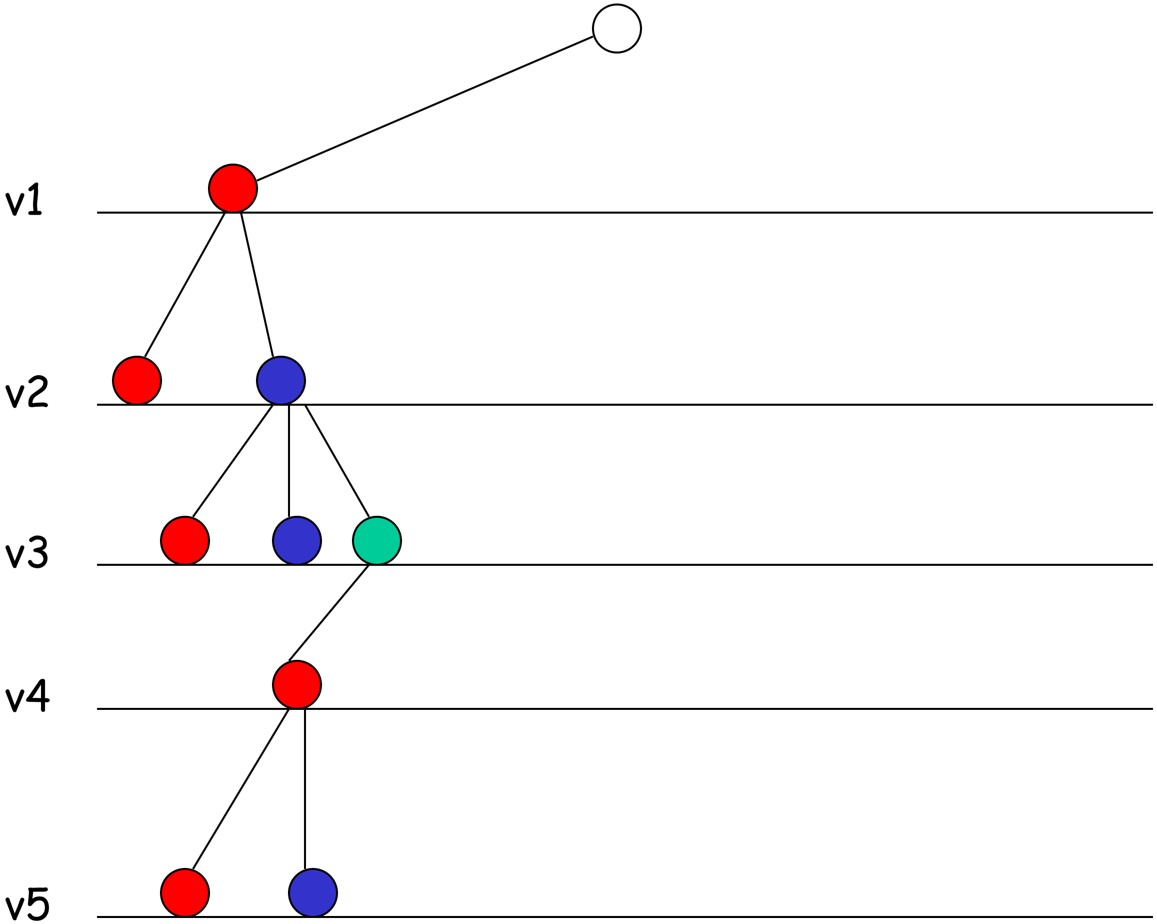


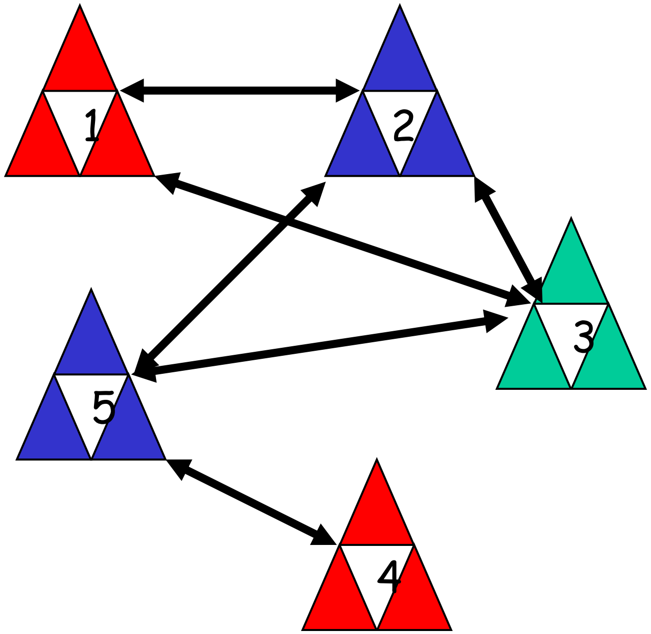
A Tree Trace of BT (assume domain ordered {R,B,G})



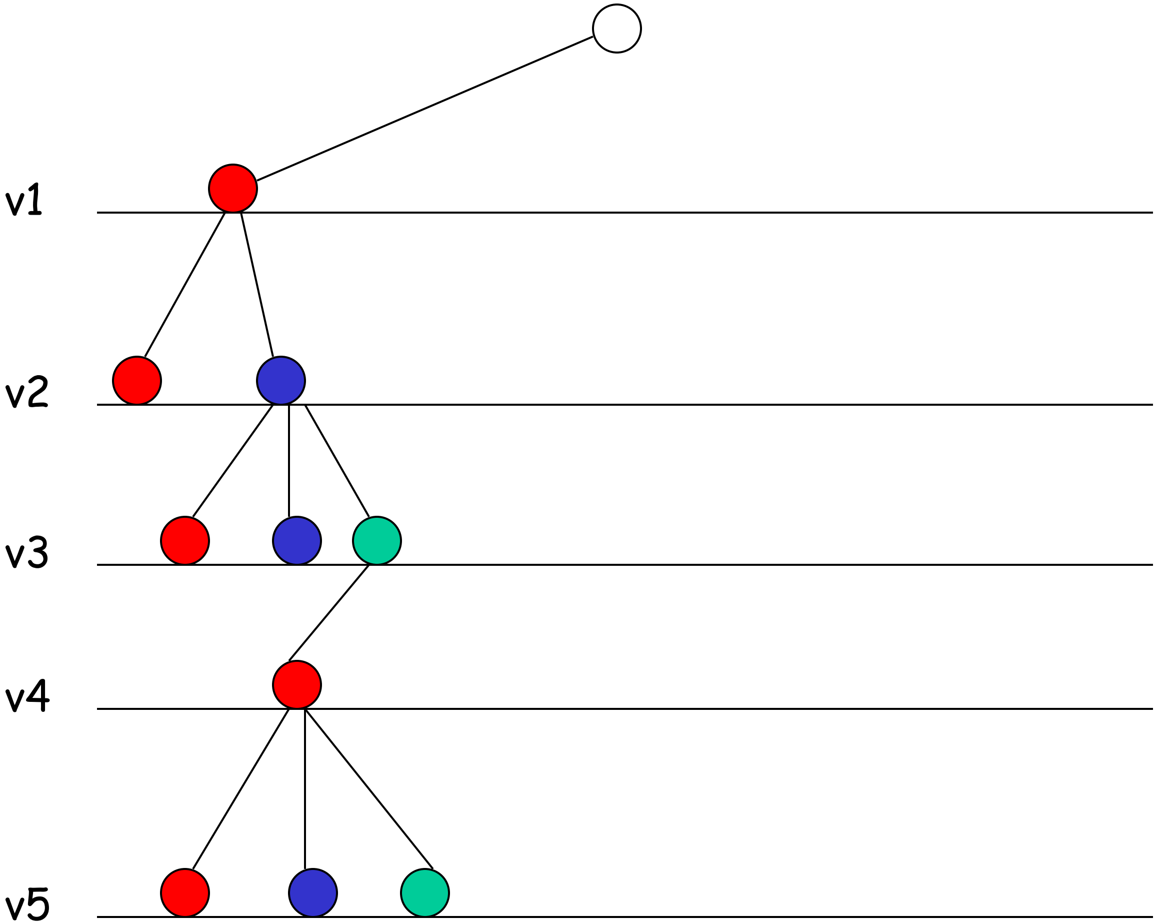


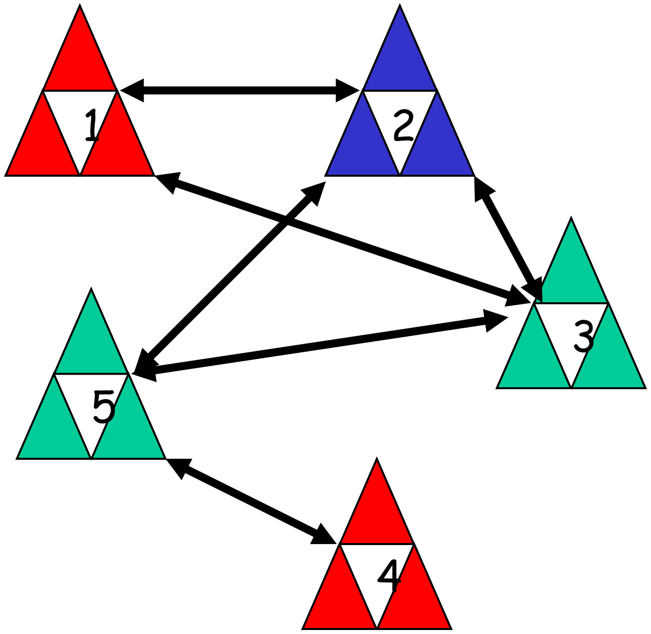
A Tree Trace of BT (assume domain ordered {R,B,G})



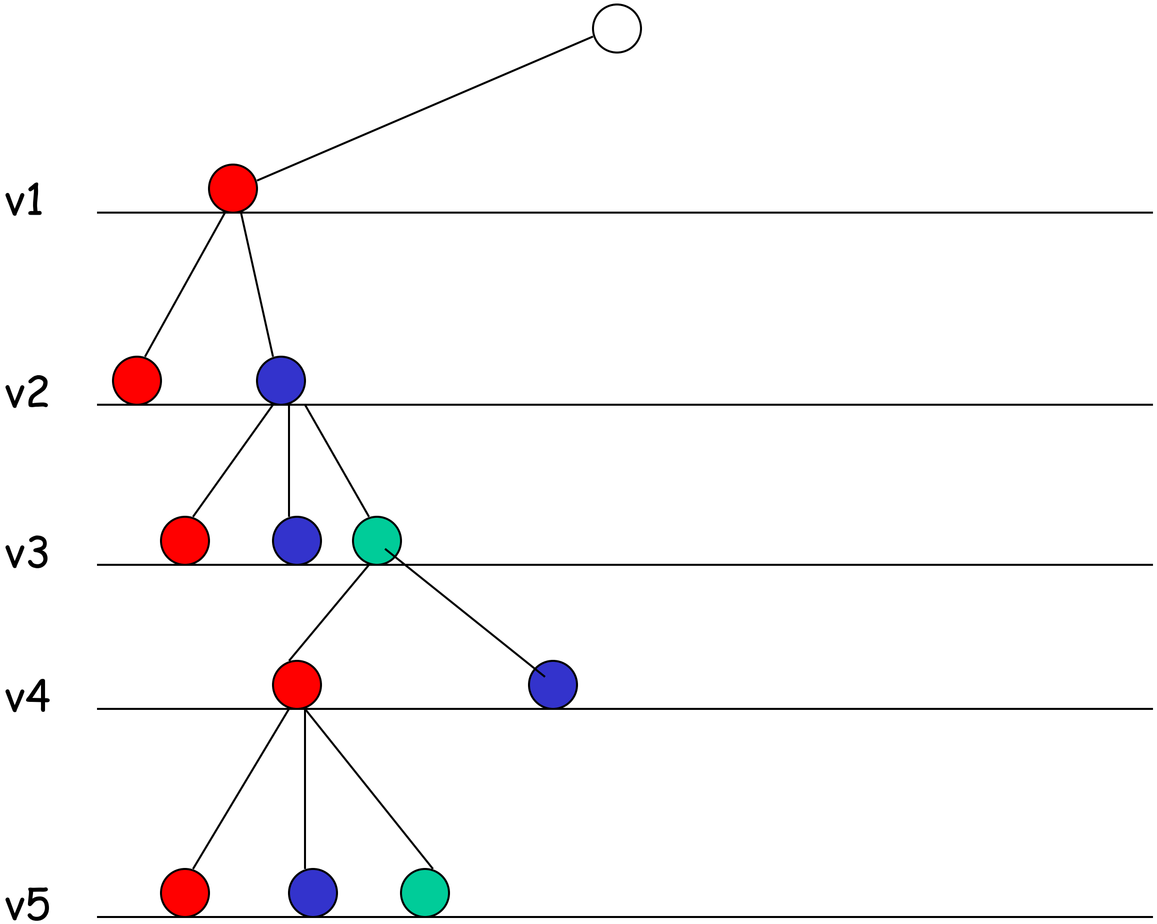


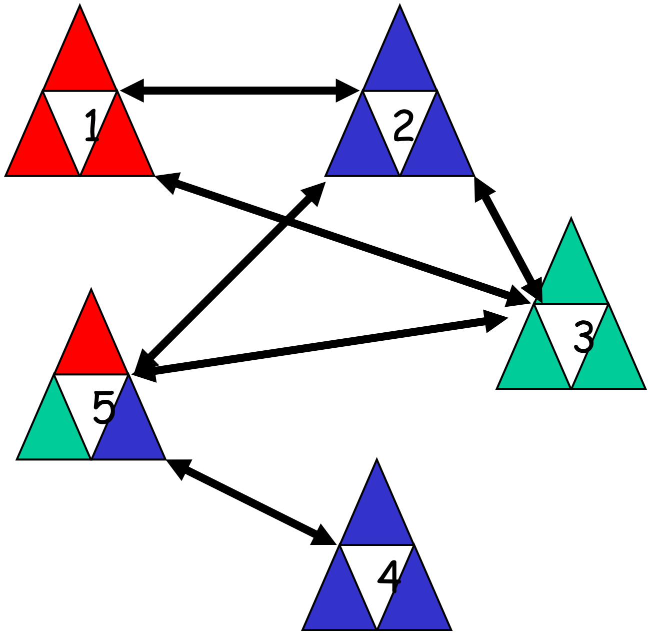
A Tree Trace of BT (assume domain ordered {R,B,G})



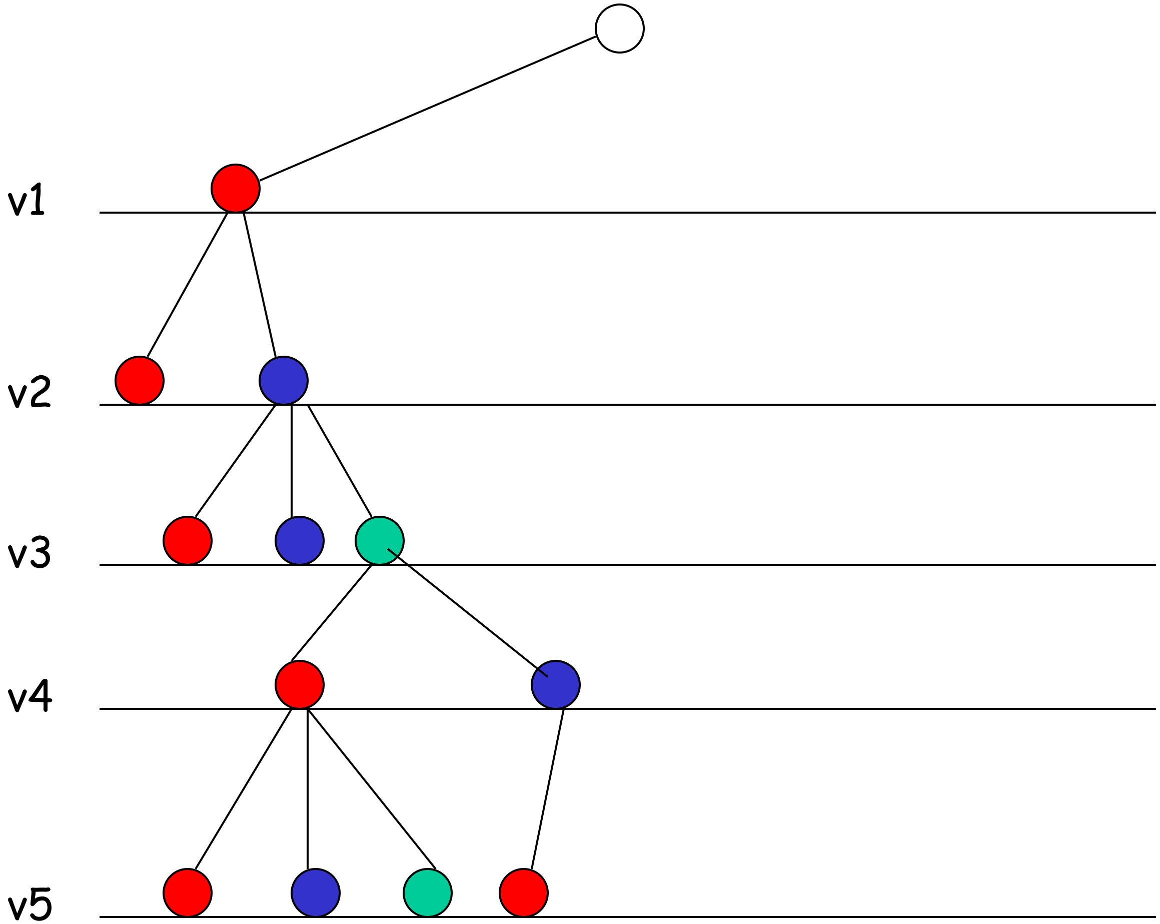


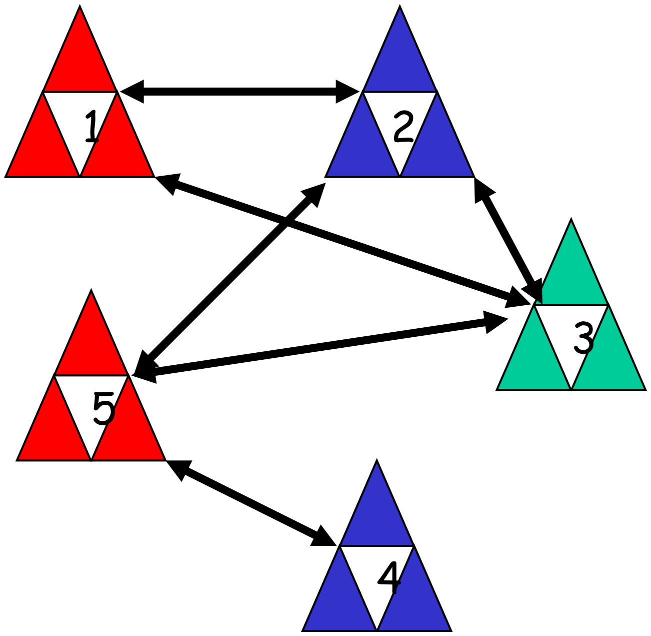
A Tree Trace of BT (assume domain ordered {R,B,G})



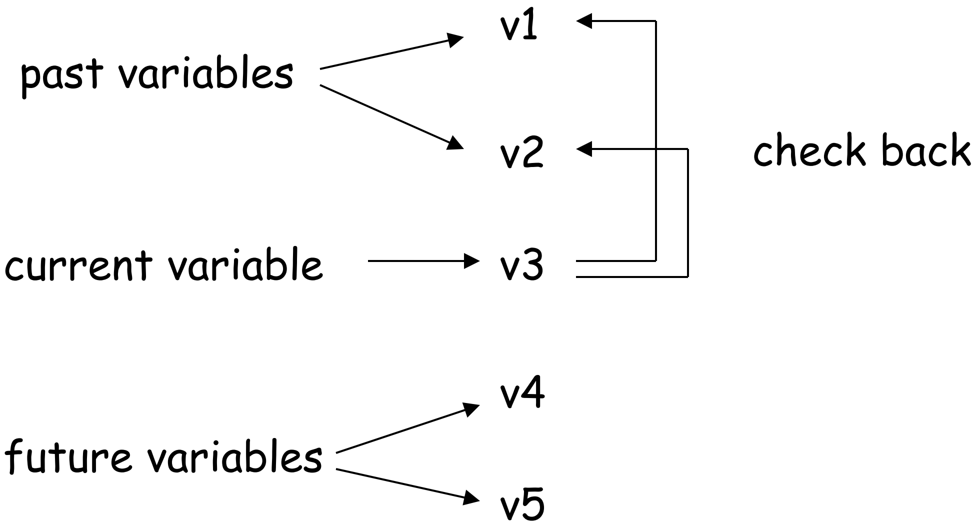


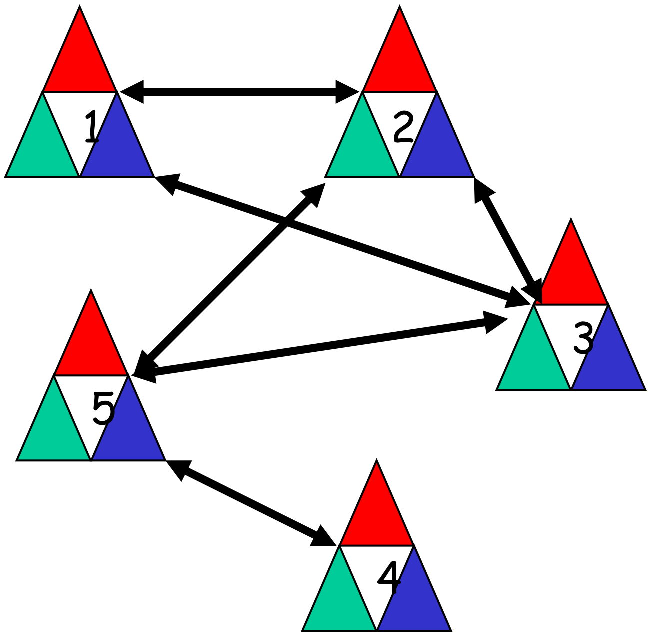
A Tree Trace of BT (assume domain ordered {R,B,G})



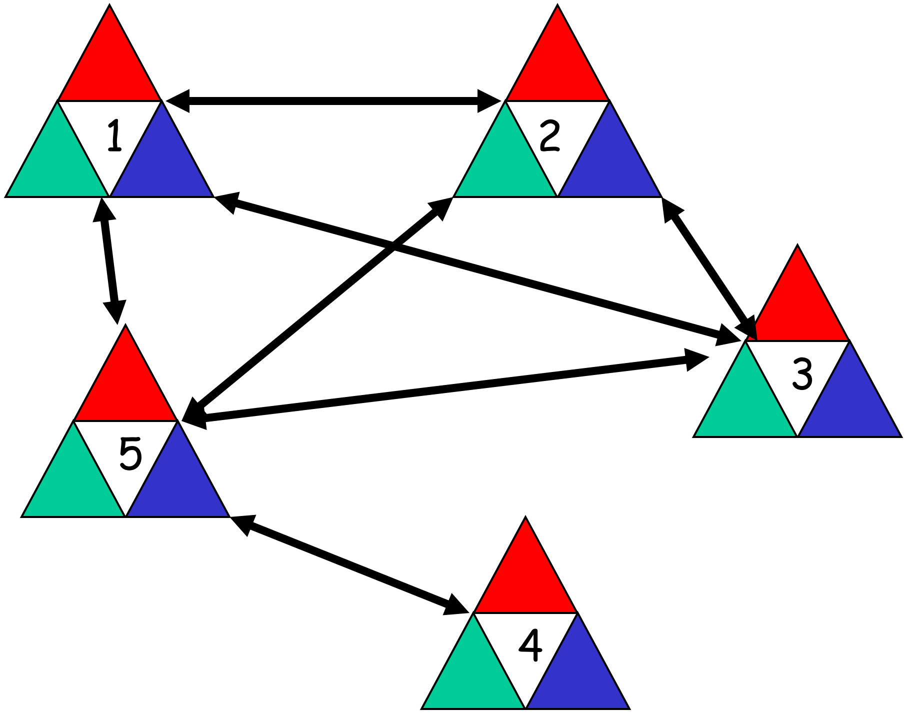


Another view

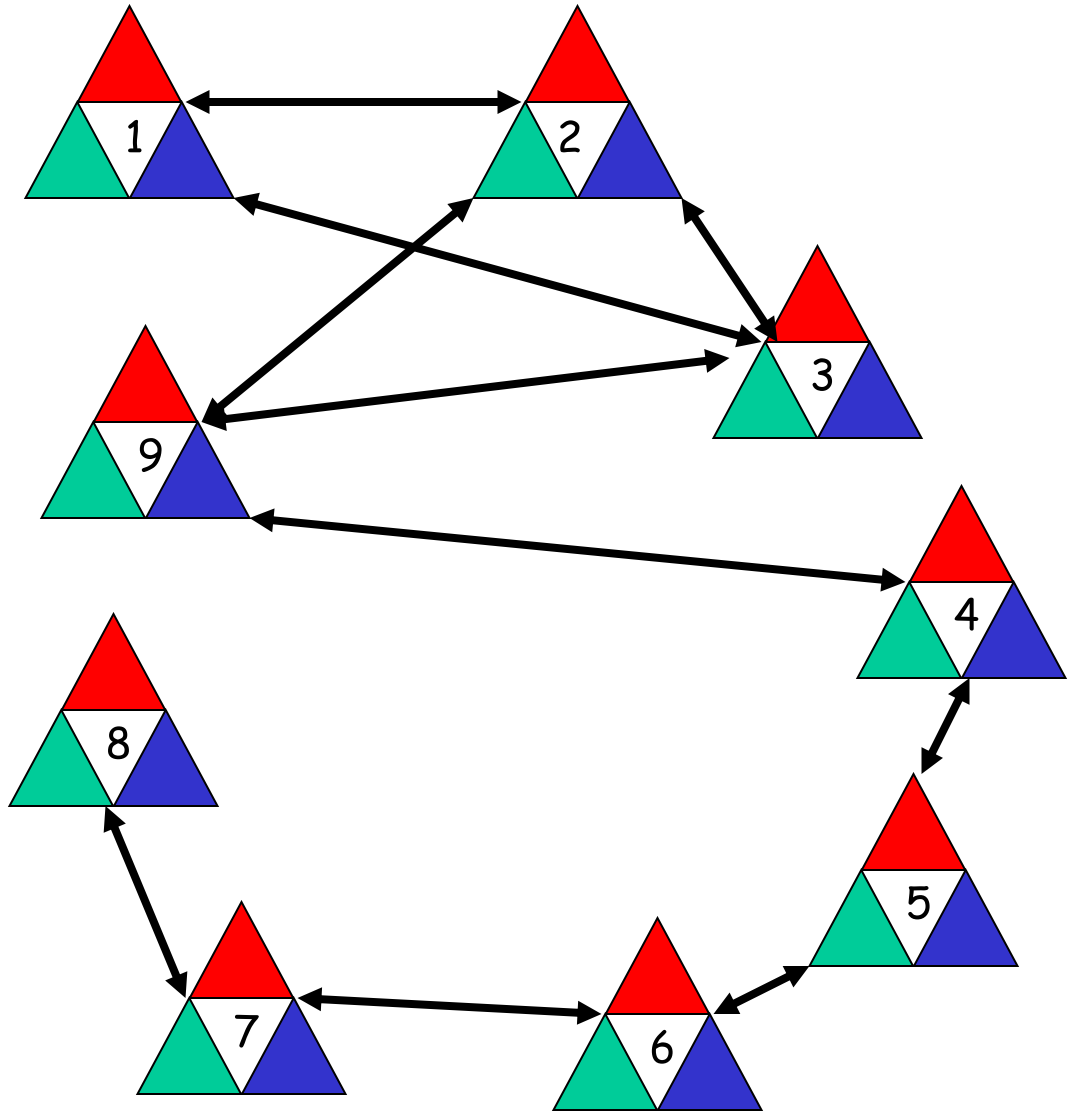




Can you solve this (csp2)?

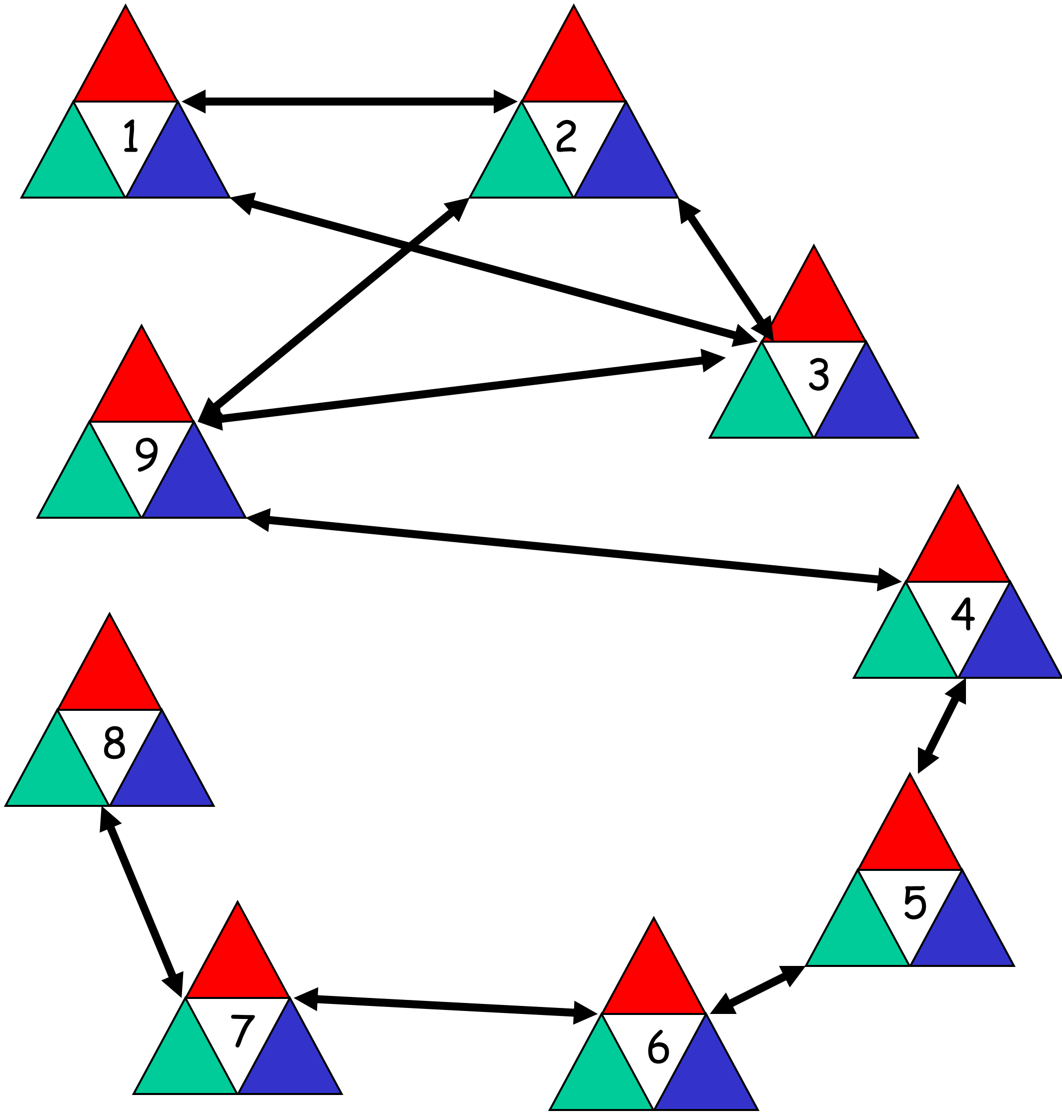


Thrashing? (csp3c)



Thrashing? (csp3c)

V1 = R
v2 = B
v3 = G
v4 = R
v5 = B
v6 = R
v7 = B
v8 = R
v9 = conflict



The cause of the conflict with v9 is v4, but what will bt do?

Find out how it goes with bt3 and csp4

questions

- Why measure checks and nodes?
    - What is a node anyway?
    - Who cares about checks?
- Why instantiate variables in lex order?
    - Would a different order make a difference?
- How many nodes could there be?
- How many checks could there be?
- Are all csp's binary?
- How can we represent constraints?
- Is it reasonable to separate V from D?
    - Why not have variables with domains
- What is a *constraint graph*?
    - How can (V,D,C) be a graph?
- What is BT "thinking about"? i.e. why is it so dumb?
- What could we do to make BT smarter?
- Is BT doing any inferencing/propagation?
- What's the 1st reference to BT?
    - Golomb & Baumert JACM 12, 1965?
    - The Minataur?