Fast Solutions to CSP's

Based on

PROSSER, P. *Hybrid algorithms for the constraint satisfaction problem*. Computational Intelligence 9 (1993), 268--299

Outline

- Motivation
- Methods of advancing CSP search
  - Backmarking
  - Forward Checking
- Hybrid algorithms
- Conclusion

Problem to be solved

Given a set of n variables *where the i^{th} variable, 1<= i<=n, has a discrete domain of values it can take, domain[i],* **and a set of binary relations** $C = \{C_{1,1} \ldots C_{1,n}\ C_{2,1} \ldots C_{2,n} \ldots C_{n,n}\}$, **find the first consistent instantiation of these variables which satisfies all the relations**.

Notation:

Current variable is indexed with i, V_i

Past variables will be variables that have already been instantiated (those whose index is <i)

Future variables will be those yet to be instantiated (those whose index is >i)

Why Binary CSP's

Every higher order (multiple variables) , finite domain constraint can be reduced to a set of binary constraints if enough auxillary variables are introduced.

What is the forward move and why change it?

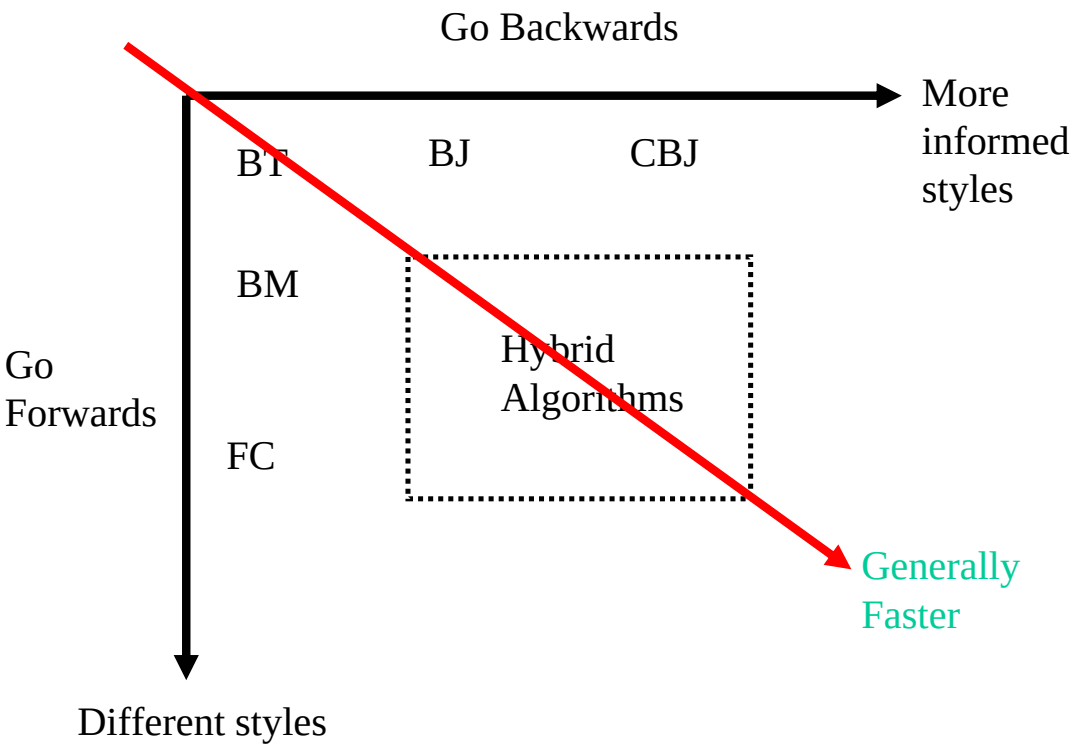
Forward move – the procedure that determines what actions to take (consistency checks, bookkeeping, etc) when the next variable is instantiated.

Goal : avoid unnecessary computation

•Backmarking (BM) – remembers consistency checks it already performed

•Forward Checking (FC) – doesn't expand nodes it knows aren't feasible

The 5 Base styles of search



Outline

- Motivation
- Methods of advancing CSP search
  - Backmarking
  - Forward Checking
- Hybrid algorithms
- Conclusion

What does BM do?

- Objective : BM prevents redundant consistency checks when
  - The current variable is known to fail with its current value because of some variable in the past still has the value that made the current variable fail.
  - The current variable is known to succeed with its current value in a check against a past value that still has the same value that made the current variable succeed.
- Tradeoff : must spend time doing, and allot space for, bookkeeping.

What does BM do?

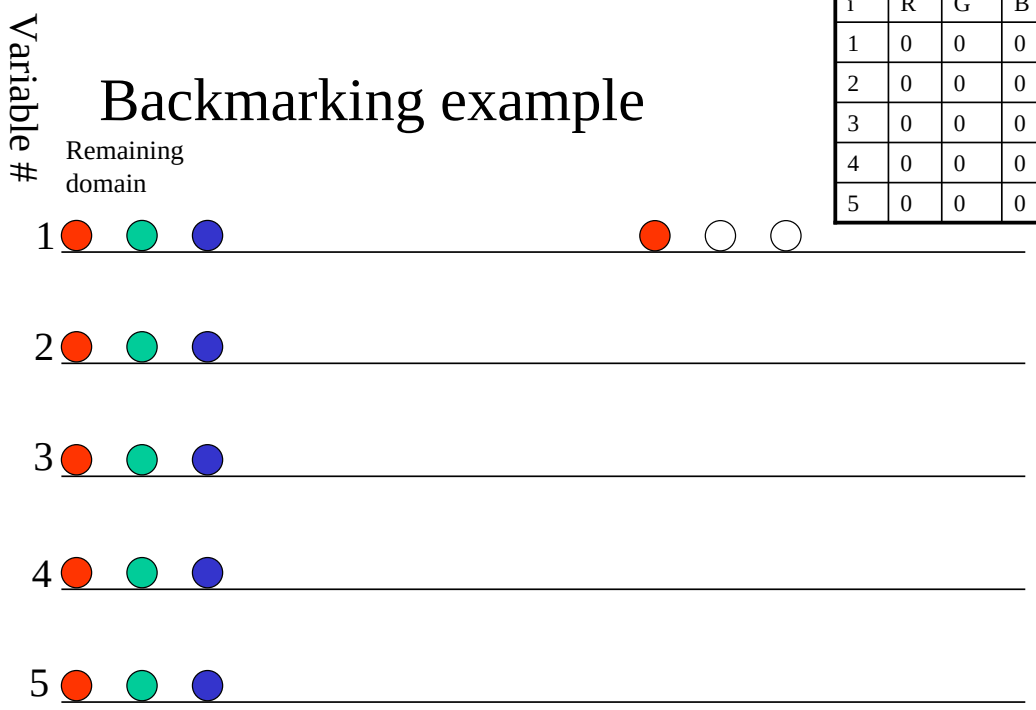
- Maximum checking level (mcl) array
  - Size = Number of variables x domain size
  - mcl[i,k] holds the index of the deepest variable that v[i] = k, k $\in D_i$, was checked against
- Minimum backup level (mbl) array
  - Size = Number of variables x 1
  - mbl[i] the index of the shallowest past variable that has changed value since v[i] was the current variable

What does BM do?

- mcl[i,k] < mbl[i] means the previous consistency check between v[I] = k and some variable in the past of mcl[i,k] failed and will still fail because mcl[i,k] hasn't been changed
- mcl[I,k] >= mbl[i] means v[i] = k passed consistency checking for all variables in the past of mbl[i]. V[i] only needs to be checked for consistency with the new variables, those in the future of mbl[i]

Backmarking example

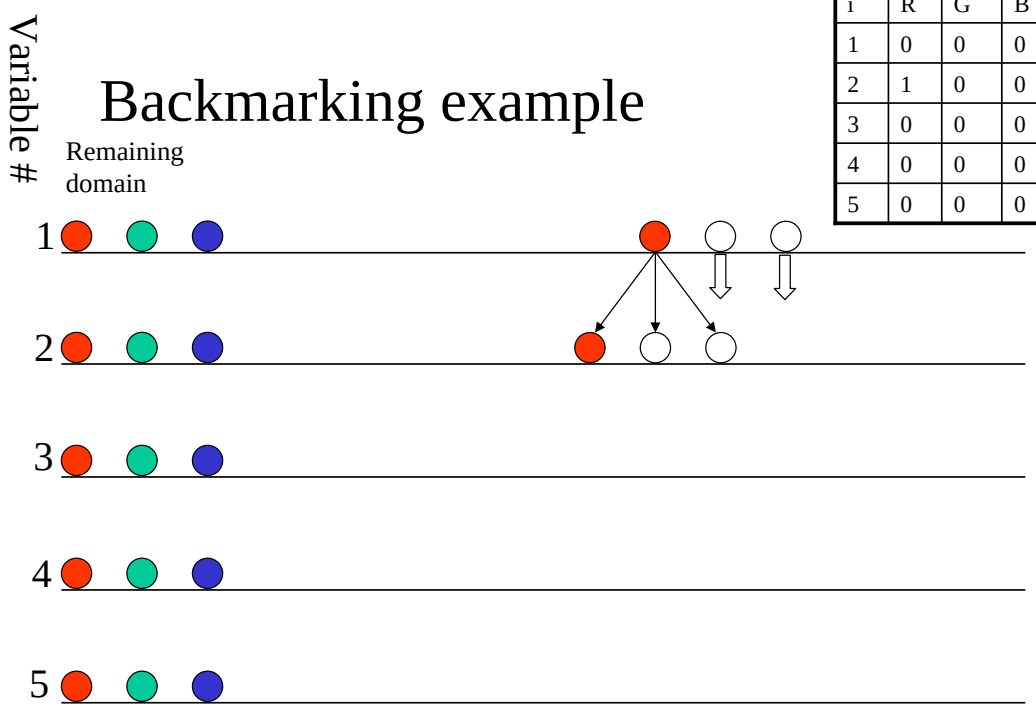
i	R	G	B	mbl
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0



Suppose no element of the domain of the 5th variable is consistent with the first element of the domain of the first variable

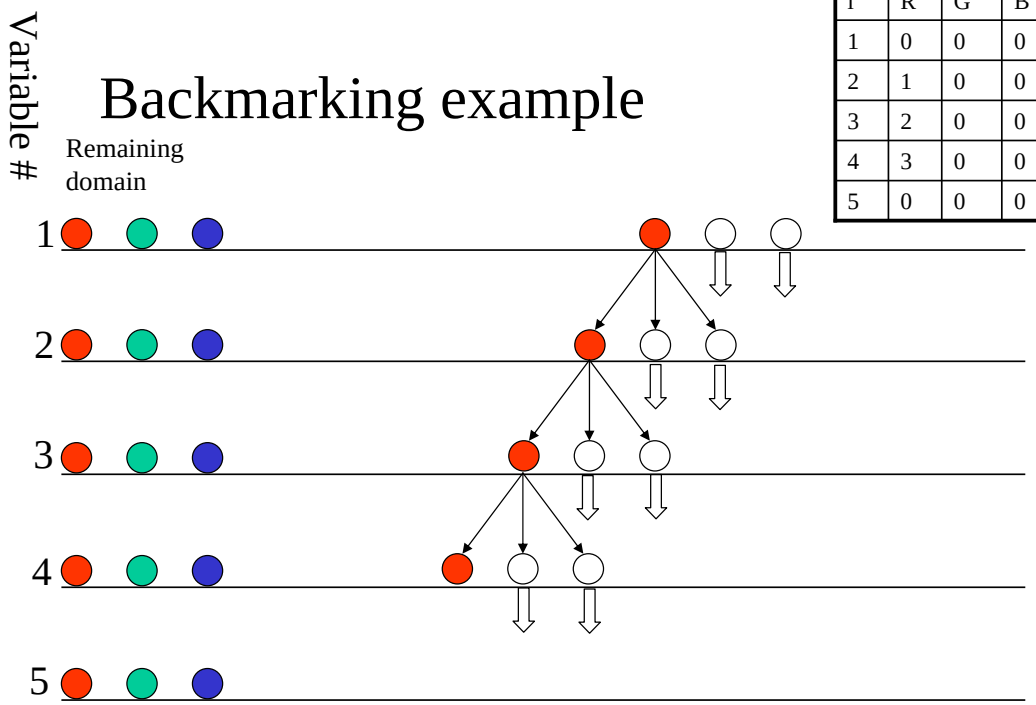
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0



Backmarking example

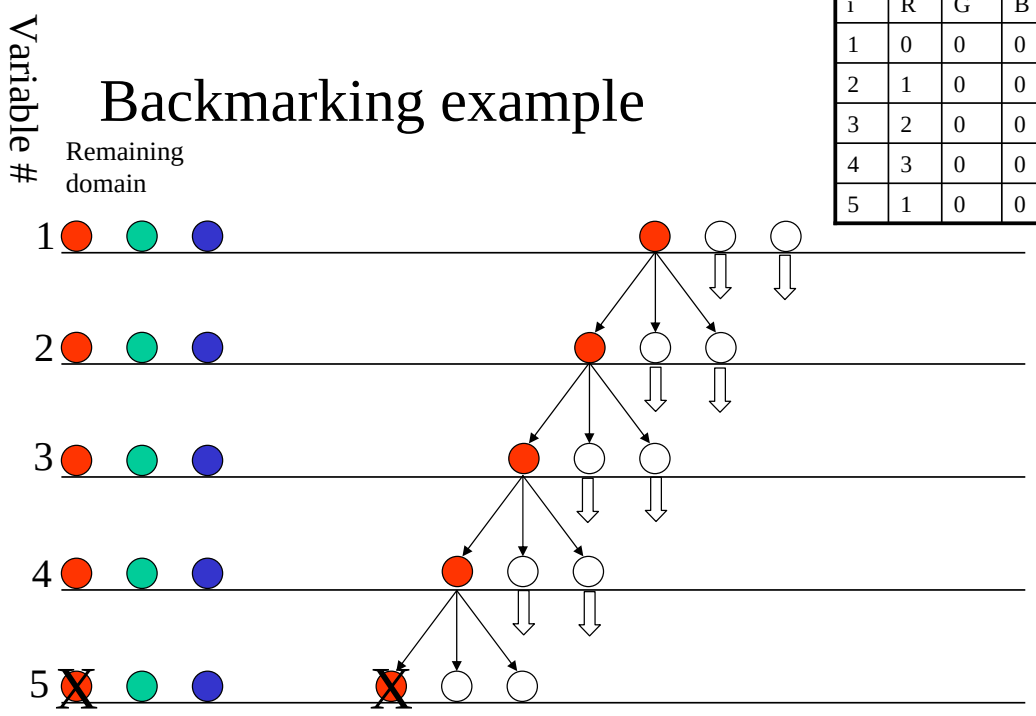
i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	0	0	0
5	0	0	0	0



Skipping a step

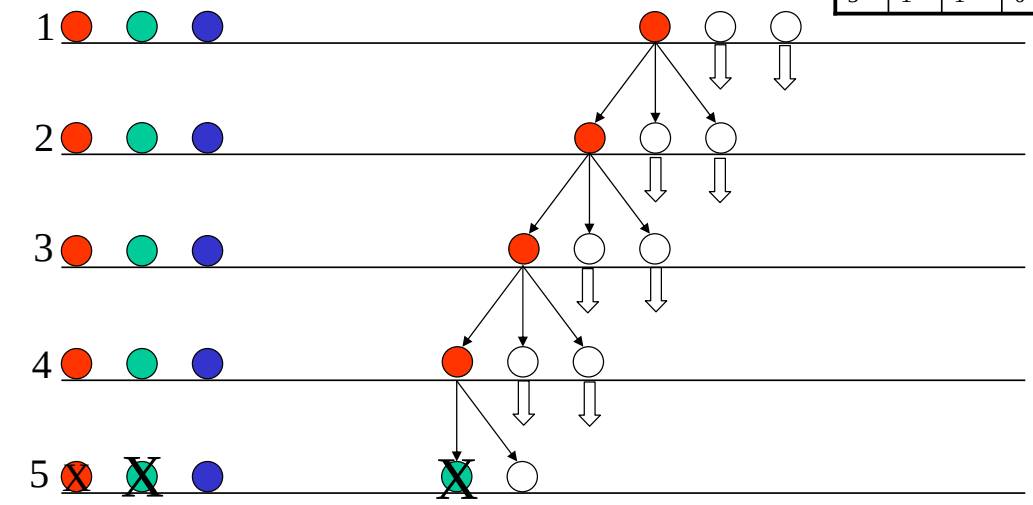
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	0	0	0
5	1	0	0	0



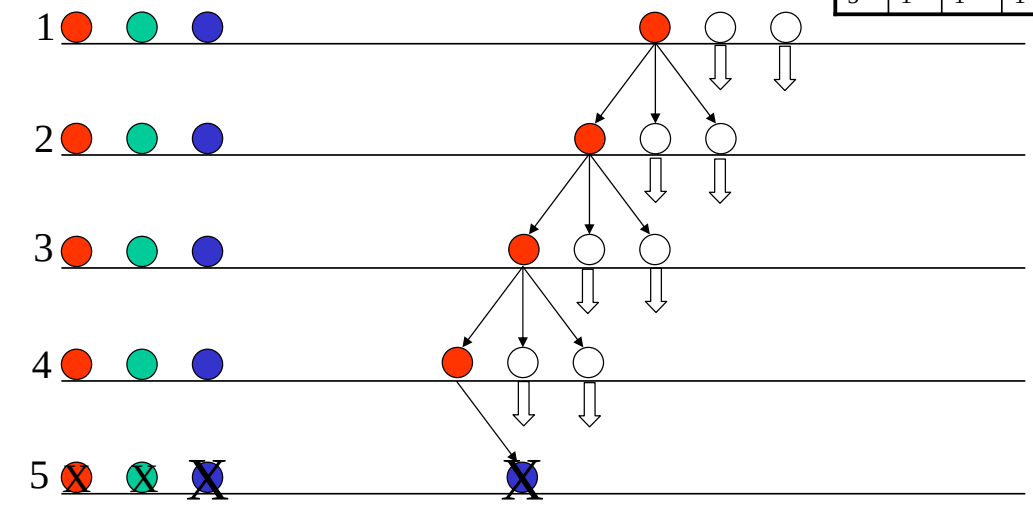
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	0	0	0
5	1	1	0	0



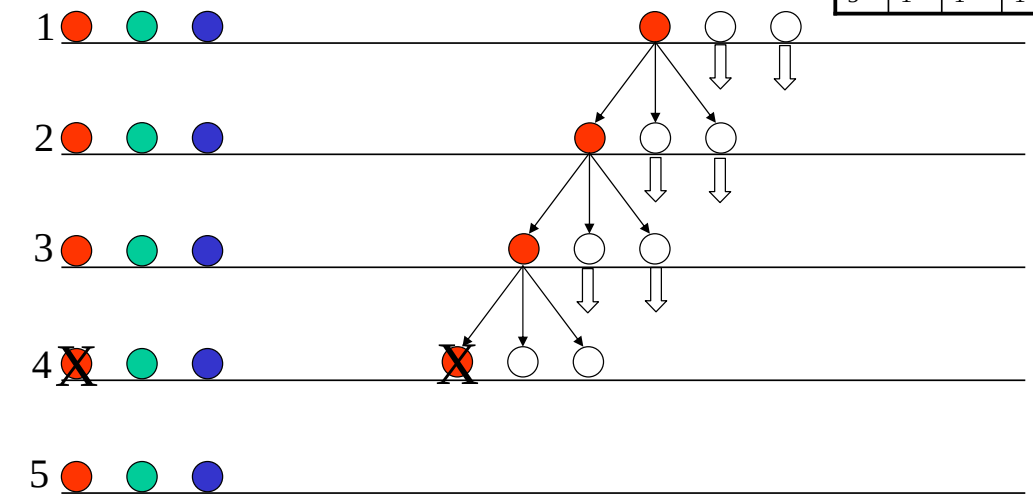
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	0	0	0
5	1	1	1	4



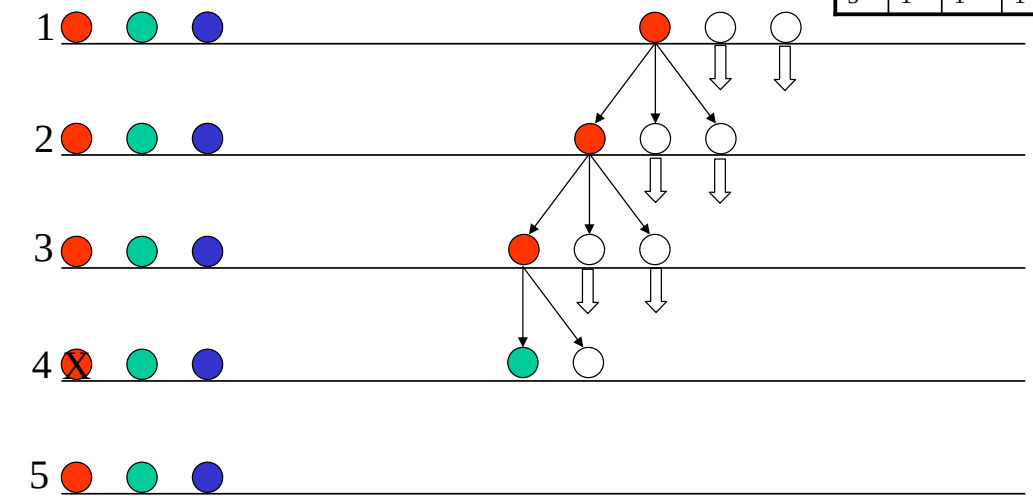
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	0	0	0
5	1	1	1	4



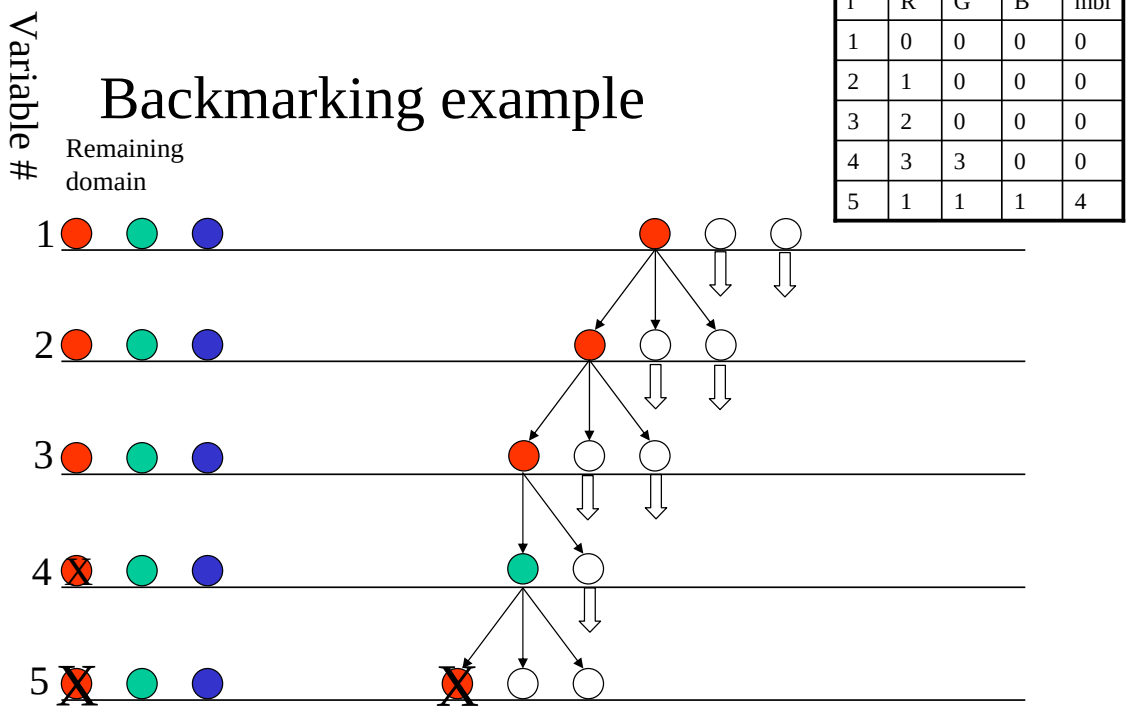
Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	3	0	0
5	1	1	1	4



Backmarking example

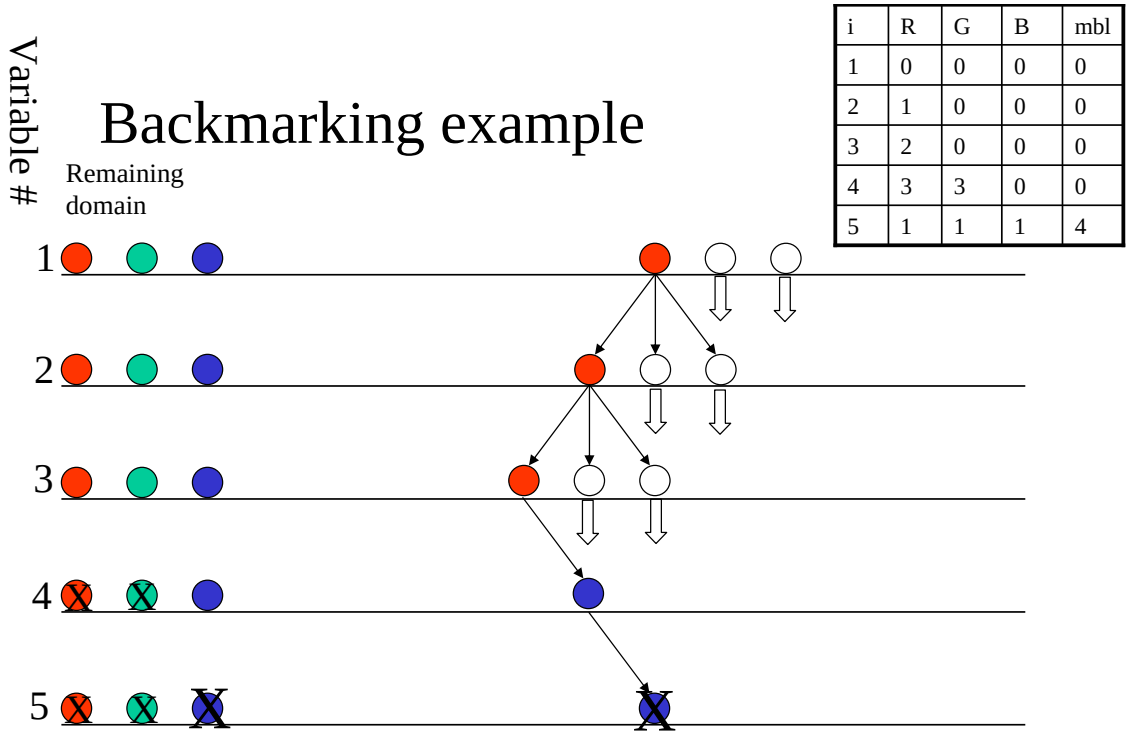
i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	3	0	0
5	1	1	1	4



No consistency checks at level 5 will be performed until the value of the first variable is changed

Backmarking example

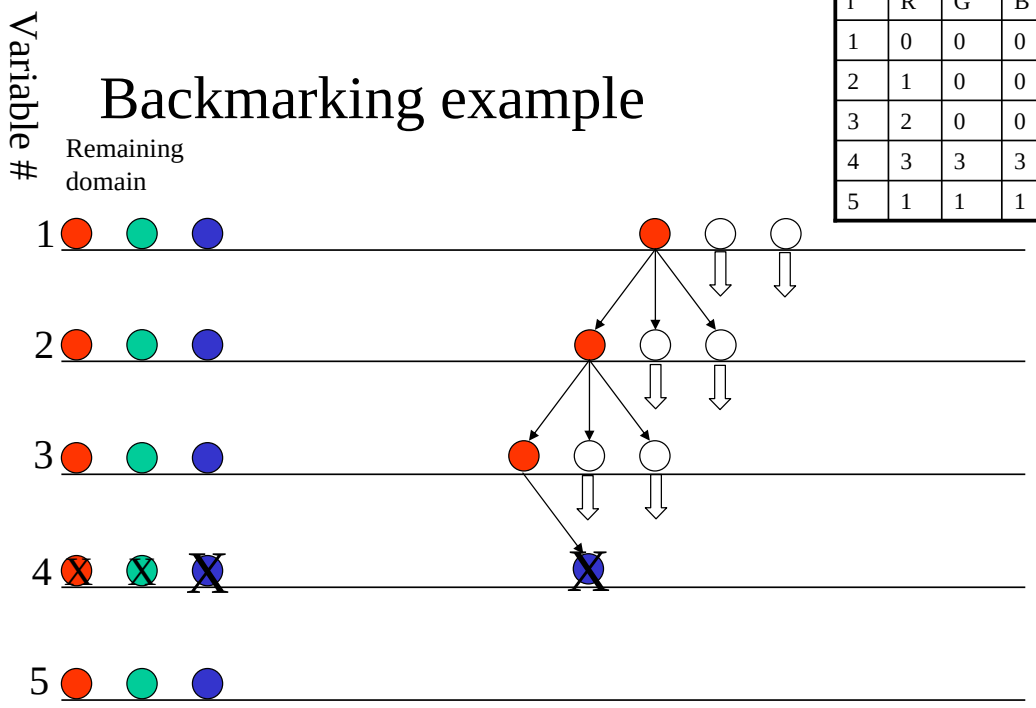
i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	3	0	0
5	1	1	1	4



Eventually variable 4 also encounters DWO

Backmarking example

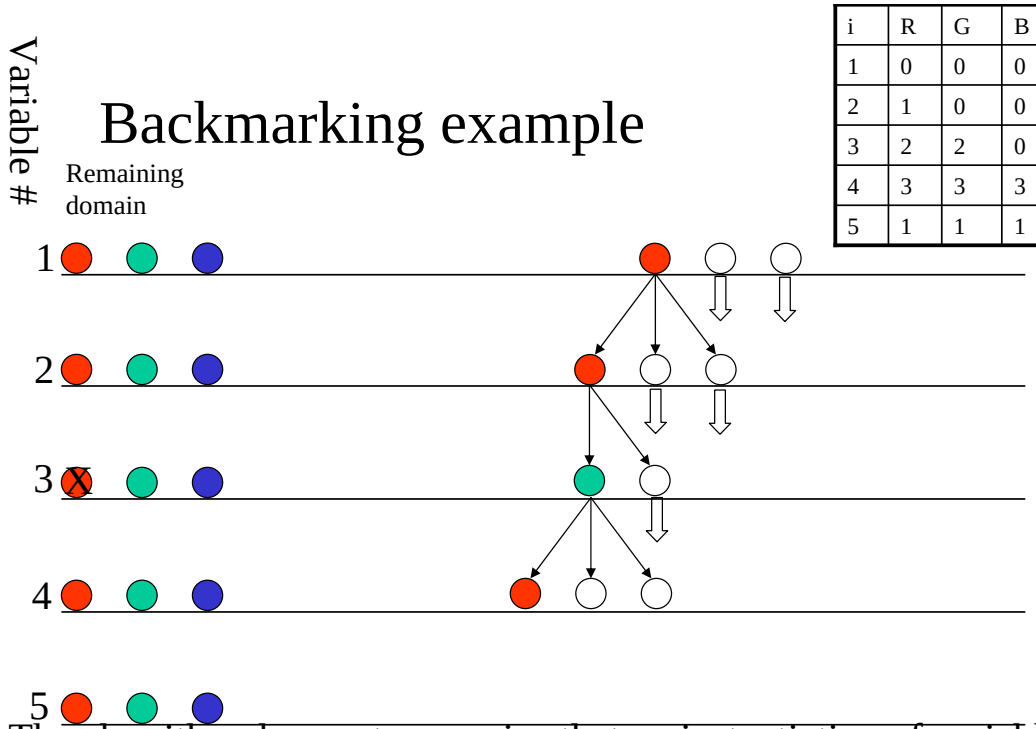
i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	0	0	0
4	3	3	3	3
5	1	1	1	4



Eventually variable 4 also encounters DWO

Backmarking example

i	R	G	B	mbl
1	0	0	0	0
2	1	0	0	0
3	2	2	0	0
4	3	3	3	3
5	1	1	1	4

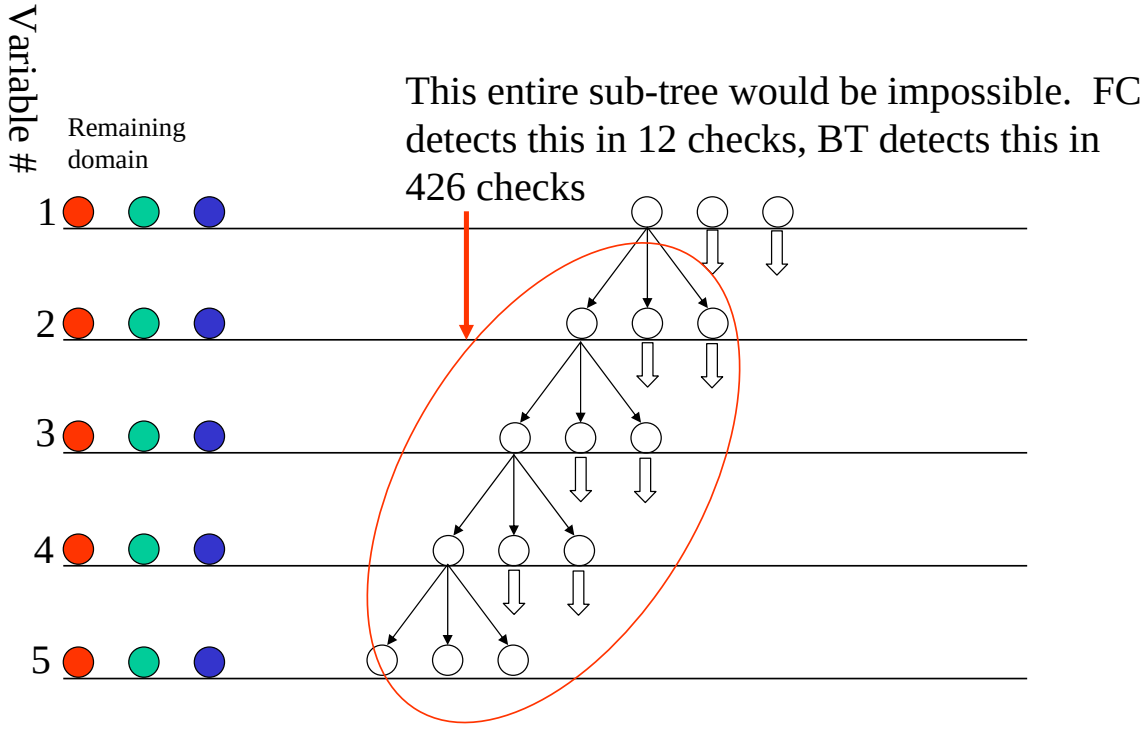


The algorithm does not recognize that no instantiation of variable 4 works while variable 1 is red, it does however save on consistency checks between variable 4 and all variables previous to the shallowest change since variable 4 was last tested

Forward Checking

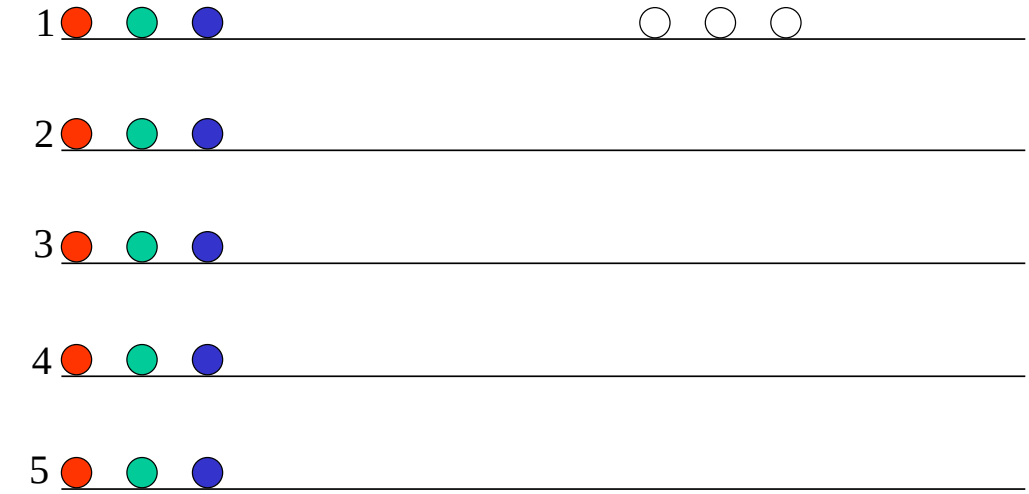
- Objective : look ahead to find impossible combinations as soon as possible; “fail early”
  - Remove inconsistent values from the domains of all future variables
  - If domain of future variable is null, then backtrack
- Trade off : perform speculative checks hoping they will reduce the total number of checks you must perform

Suppose no element of the domain of the 5th variable is consistent with the first element of the domain of the first variable

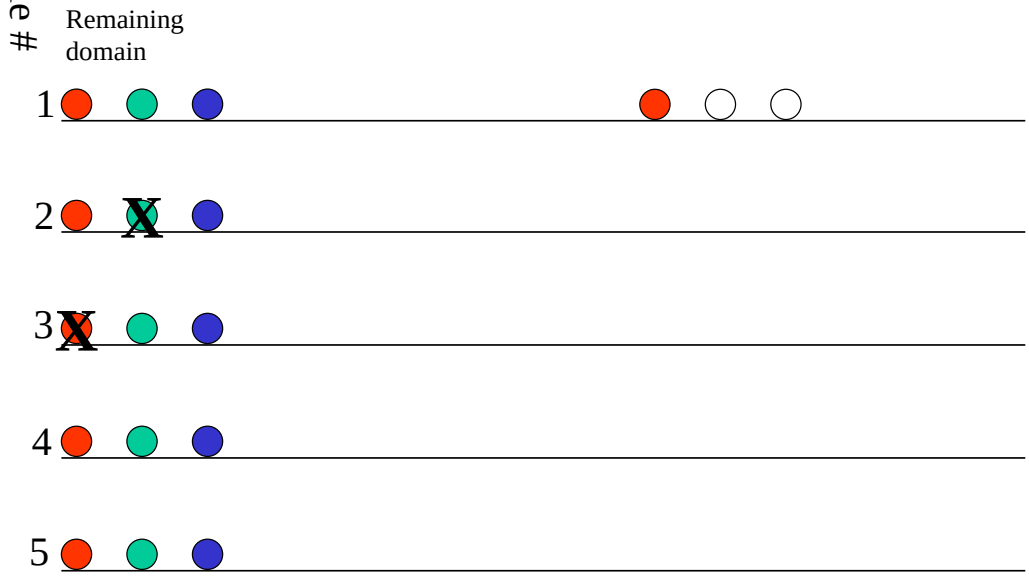


Forward checking example

5 variables to be instantiated, each has domain {red,green,blue}



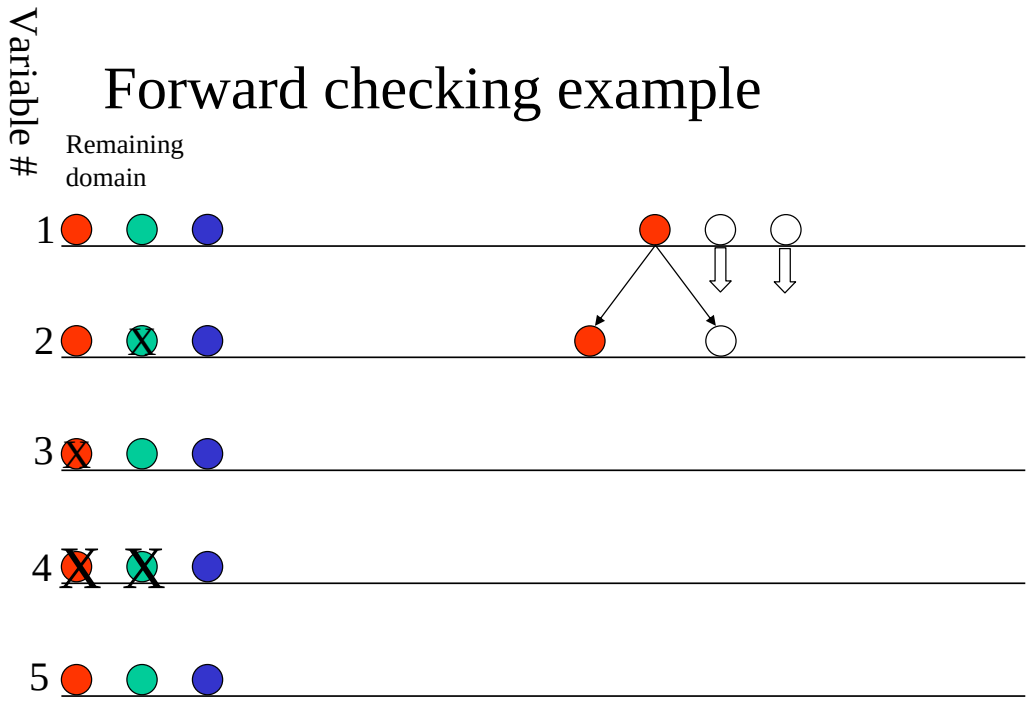
Forward checking example



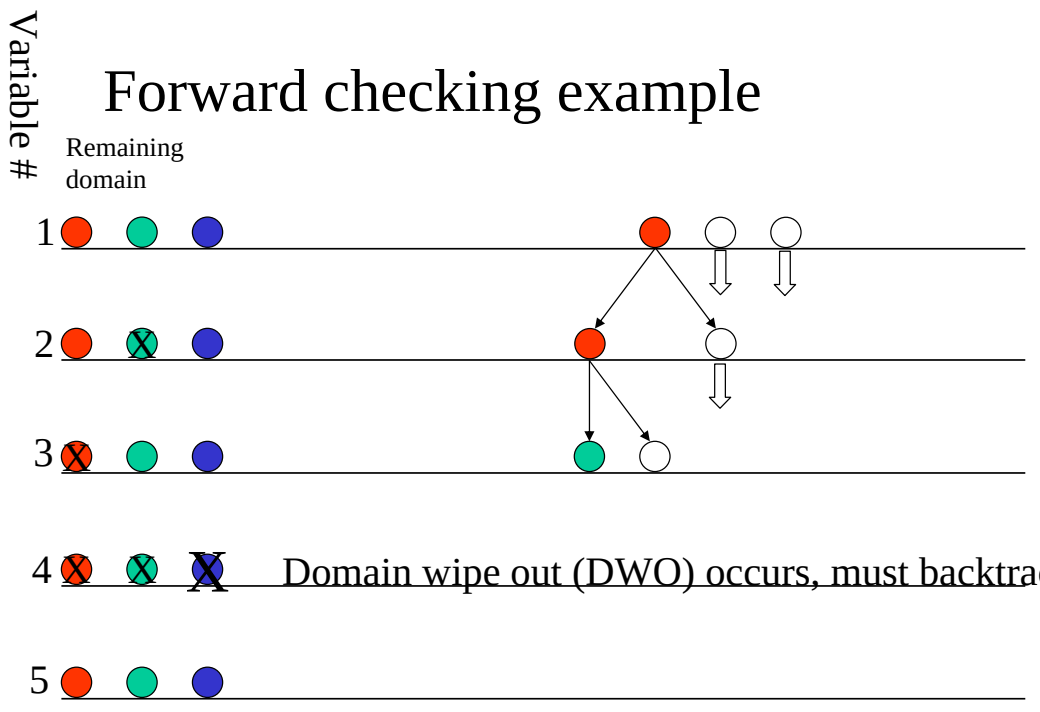
First step does 3 x 4 consistency checks

In general, n^{th} step can require up to (N-n) x max domain size consistency checks, N = total number of variables

Forward checking example

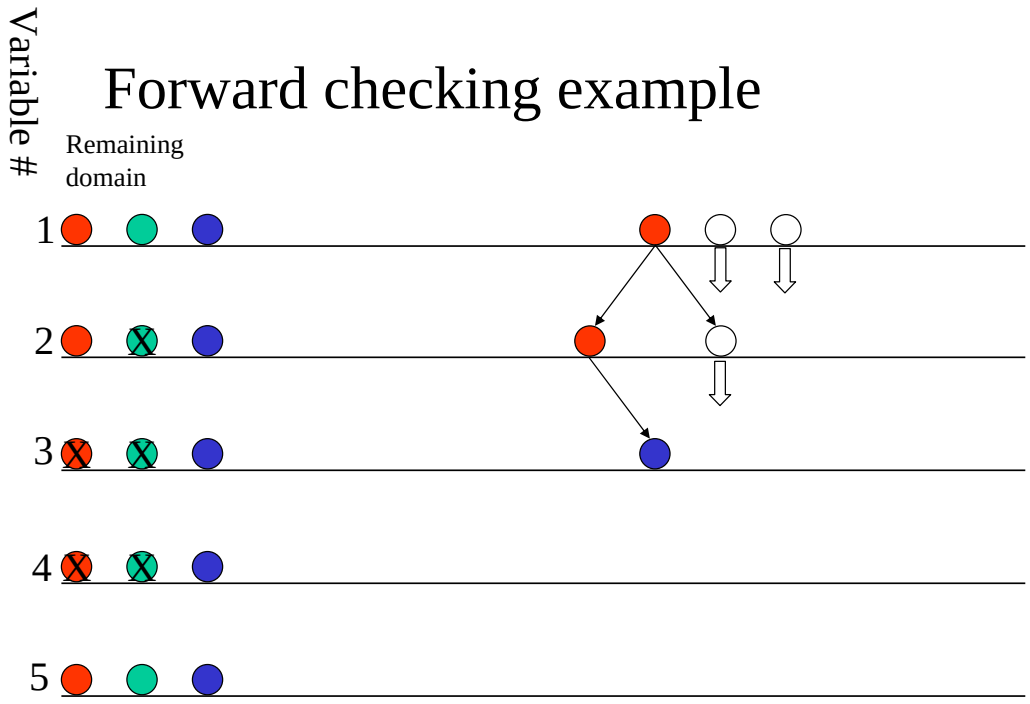


Forward checking example

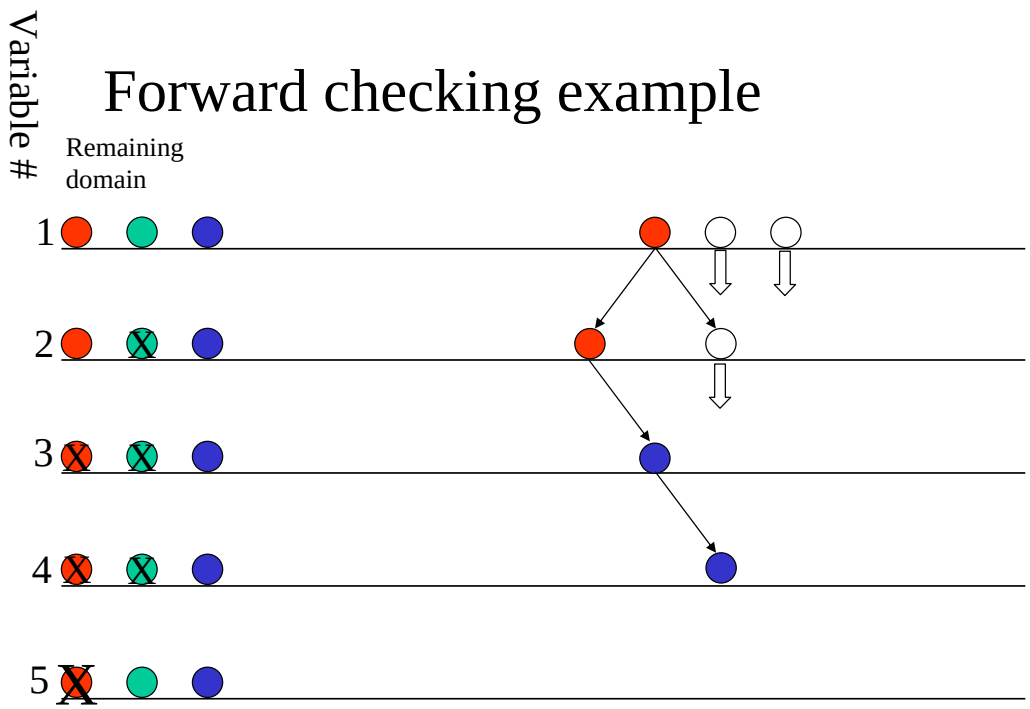


Domain wipe out (DWO) occurs, must backtrack

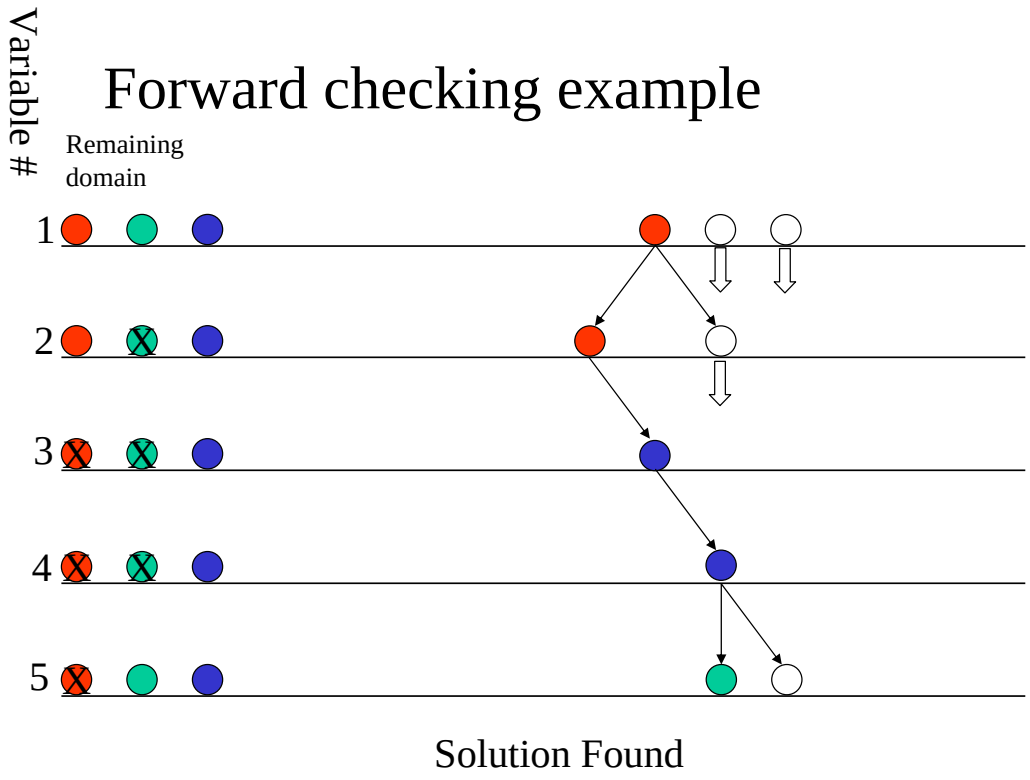
Forward checking example



Forward checking example



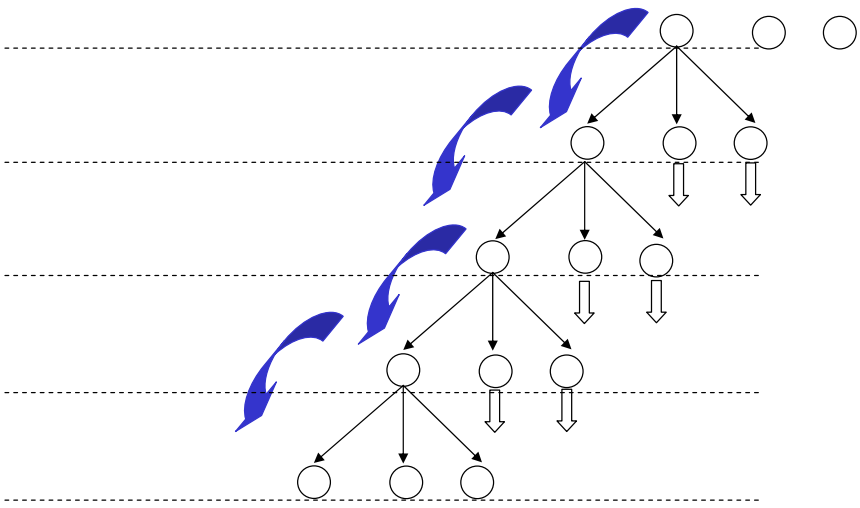
Forward checking example



Solution Found

Example when FC is beat by BT

A problem with 5 variables, each with domain of size = 3



BT would pick the first element in each variable's domain and find the solution while performing the minimum possible number of consistency checks (4+3+2+1 = 10 consistency checks)
FC would propagate the domain reductions downward which would be fruitless effort (4x3 + 3x3 + 2x3 + 1x3 = 30 consistency checks)

Outline

- Motivation
- Methods of advancing CSP search
  - Backmarking
  - Forward Checking
- Hybrid algorithms
- Conclusion

Why create a hybrid algorithm?

•Objective : create better CSP algorithms by mixing forward and backward algorithms

•Requires that the backward and forward algorithms be compatible : information stored by the forward algorithm won't be corrupted by the actions of the backward algorithm and vice versa

How do you create a hybrid algorithm?

- Implement each algorithm iteratively instead of recursively
  - Two functions : move-forward( ) and move-backward( )
  - WHILE (!solution and !impossible) {

    IF consistent then move-forward( )

    ELSE move-backward( )

    }
- Maintain the information required by each algorithm in the forward and backward move

Hybrid Algorithms Implemented by Prosser

Backmark Jumping (BMJ)

Backmarking and Conflict Directed Backjumping (BM-CBJ)

Forward Checking and Backjumping (FC-BJ)

Forward Checking and Conflict Directed Backjumping (FC-CBJ)

He compared these with the base algorithms BT, BJ, CBJ, BM, FC on the ZEBRA problem

Results: Consistency checks (from Prosser)

Algorithm	mean	Std. Dev.	Min	Max
BT	3,858,989	9,616,407	1773	102,267,383
BJ	503,324	1,524,193	358	19,324,081
CBJ	63,212	193,846	339	19,324,081
BM	396,945	1,276,415	401	18,405,514
BMJ	125,474	361,595	300	5,214,608
BM-CBJ	25,470	72,004	297	1,237,283
FC	35,582	71,012	262	802,069
FC-BJ	16,839	29,977	262	280,302
FC-CBJ	10,361	16,383	262	119,767

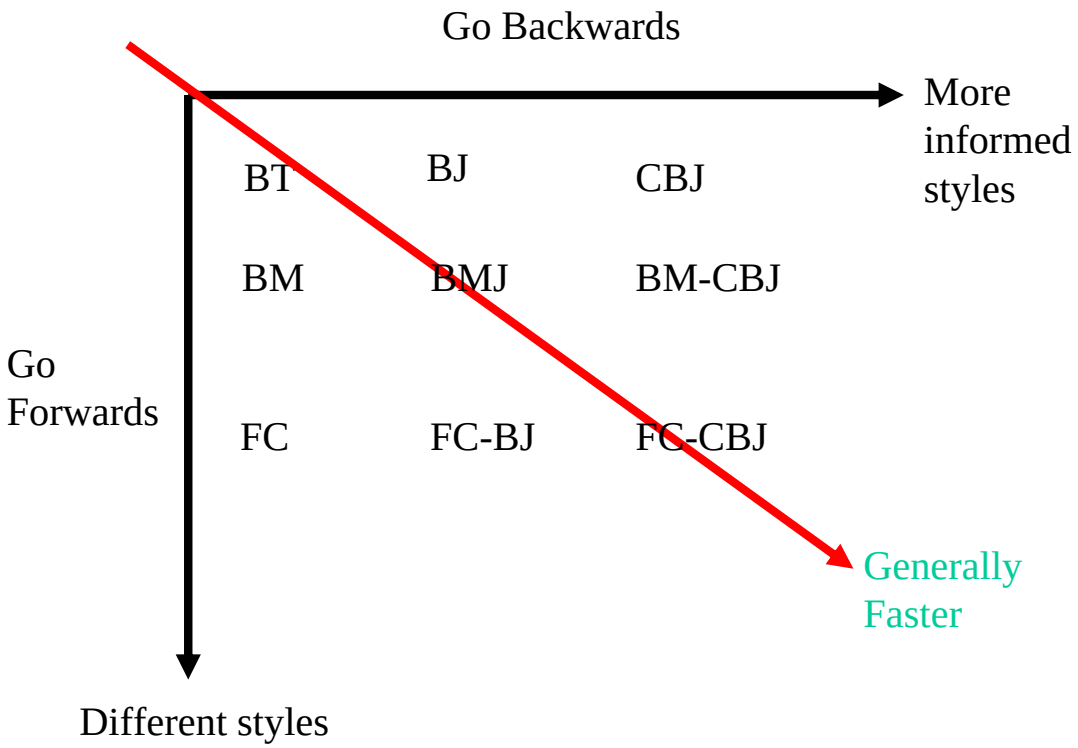
Consistency checks and Nodes visited are two possible, implementation independent metrics for algorithm efficiency. FC's performance is unfairly enhanced by the Nodes Visited metric since it prunes the tree. Also, consistency checks could possibly be computationally intense for more difficult problems and therefore is the preferable metric

How Often One Algorithm (Row) Bettered Another (Column) (from Prosser)

	BT	BJ	CBJ	BM	BMJ	BM-CBJ	FC	FC-BJ	FC-CBJ
BT	----------	0	0	0	0	0	0	0	0
BJ	450	----------	0	132	0	0	0	0	0
CBJ	450	450	----------	370	280	0	130	35	5
BM	450	318	80	----------	31	8	13	5	2
BMJ	450	450	170	419	----------	17	29	7	3
BM-CBJ	450	450	450	442	433	----------	286	117	35
FC	450	450	320	437	421	163	----------	0	0
FC-BJ	450	450	415	445	443	333	438	----------	0
FC-CBJ	450	450	445	448	447	415	440	388	----------

If (i,j) does not equal (j,i) then there were ties

The 5 Base styles of search



Beware

- If one part of the hybrid performs poorly on a problem, then its possible that the hybrid will perform poorly as well.
  - CBJ is usually better than BM, but when BM beats CBJ, its possible that BM might out perform BM-CBJ
    - In jumping back you change some past variable further up the tree than you would with BM alone. This causes you to forget about the previous consistency checks that have been performed since mbl[i] < mcl[i,k]

Why do BM schemes sometimes outperform FC-CBJ?

- FC performs speculative consistency checks, these may pare down the tree (detect DWO) or they may just be a waste of time
- Can you detect DWO without performing excess consistency checks?
  - No, but Minimal Forward Checking (MFC) will reduce the number of excess consistency checks

MFC

- To detect DWO you only need to check for existence of a single consistent instantiation for a variable
  - Use lazy evaluation for all consistency checks beyond those necessary to find the first consistent instantiation
- MFC = adding DWO detection to BM with lazy evaluation of consistency checks (hybrid of FC and BM)
- Results : MFC usually performs better than BM, FC, and FC-CBJ

Outline

- Motivation
- Methods of advancing CSP search
  - Backmarking
  - Forward Checking
- Hybrid algorithms
- Conclusion

Conclusion

- BM and FC are powerful in their ability to reduce consistency checks
- Hybrids generally perform better than their constituent parts
  - Poor performance of one algorithm in the hybrid might mean poor hybrid performance
- Can even make hybrids within the forward move (MFC)

Information on MFC from

Bacchus, F. & Grove, A. *On the Forward Checking Algorithm*. In Proceedings the First International Conference on Principle and Practice of Constraint Programming, 292-309, 1995

Framework for the iterative Constraint Satisfaction Search Problem

```
1   PROCEDURE bcssp(n,status)
2   BEGIN
3           consistent = true
4           status = “unknown”
5           i = 1
6           WHILE status == “unknown”
7           DO BEGIN
8                   IF consistent
9                   THEN i=label(i,consistent)
10                  ELSE i = unlabel(i,consistent)
11                  IF i>n
12                  THEN status = “solution”
13                  ELSE IF i=0
14                          THEN status = “impossible”
15          END
16  END
```

```
1   FUNCTION bm-label(I,consistent) return INTEGER
2   BEGIN
3           consistent = false
4           FOR k = EACH ELEMENT OF current-domain[I]    WHILE not
    consistent
5           DO BEGIN
6                   consistent = mcl[I,k] >= mbl[I]
7                   FOR h=mbl[I] TO I-1 WHILE consistent
8                   DO BEGIN
9                           v[I] = k
10                          consistent = check(I,h)
11                          mcl[I,k] = h
12                  END
13                  IF not consistent
14                  THEN current-domain[I] = remove(v[I],current-domain[I])
15          END
16          IF consistent THEN return(I+1) ELSE return(I)
17  END
```

```
1   FUNCTION bm-unlabel(I,consistent) return INTEGER
2   BEGIN
3           h = I-1
4           current-domain[I] = domain[I]
5           mbl[I] = h
6           FOR j = h+1 TO n DO mbl[j] = min(mbl[j],h)
7           current-domain[h] = remove(v[h],current-domain[h])
8           consistent = current-domain[h] != null
9           return(h)
10  END
```

```
1   FUNCTION check-forward(I,j) returns BOOLEAN
2   BEGIN
3           reduction = null
4           FOR v[j] = EACH ELEMENT OF current-domain[j]
5           DO IF not check(I,j)
6                   THEN push(v[j],reduction)
7           IF reduction != null
8           THEN BEGIN
9                   current-domain[j] =  set-difference(current-domain[j],reduction)
10                  push(reduction,reductions[j]
11                  push(j,future-fc[I])
12                  push(I,past-fc[j])
13          END
14          return (current-domain[j] != null)
15  END
```

```
1   PROCEDURE update-current-domain(I)
2   BEGIN
3           current-domain[I] = domain[I]
4           FOR reduction = EACH ELEMENT OF reductions[I]
5           DO current-domain[i] = set-difference(current-domain[I],reduction)
6   END
```

```
1   FUNCTION fc-label(I,consistent) returns INTEGER
2   BEGIN
3           consistent = false
4           FOR k = EACH ELEMENT OF current-domain[I]    WHILE not
    consistent
5           DO BEGIN
6                   consistent = true
7                   FOR j = I+1 TO n WHILE consistent
8                   DO consistent = check-forward(I,j)
9                   IF not consistent
10                  THEN BEGIN
11                          current-domain[I] = remove(v[I],current-domain[I])
12                          undo-reductions(I)
13                  END
14          END
15          IF consistent THEN return(I+1) ELSE return(I)
16  END
```

```
1   FUNCTION fc-unlabel(I,consistent) returns INTEGER
2   BEGIN
3           h = I-1
4           Undo-reductions(h)
5           Update-current-domain(I)
6           current-domain[h] = remove(v[h],current-domain[h])
7           consistent = current-domain[h] != null
8           return(h)
9   END
```