

CSP Search Techniques

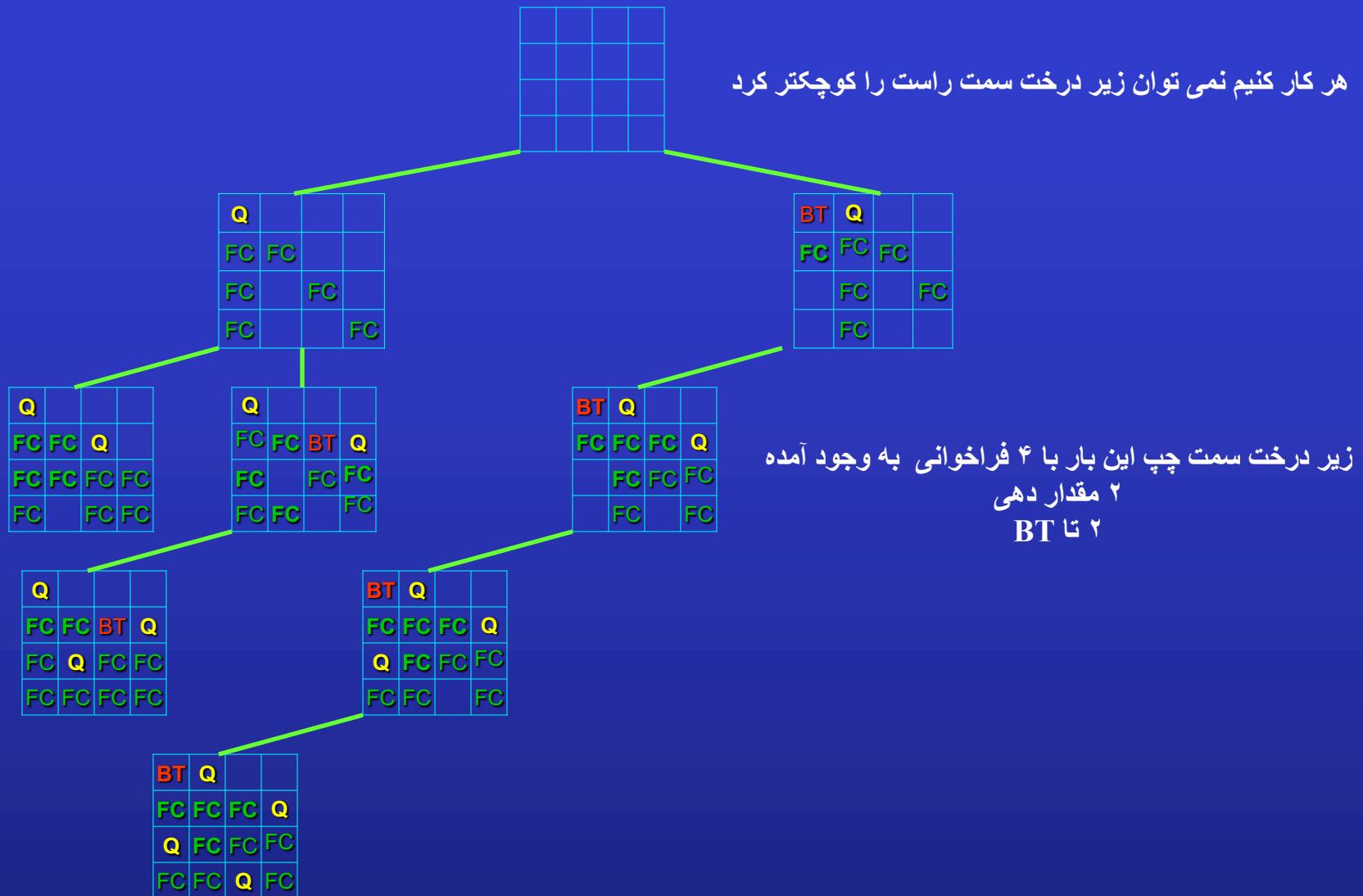
1. Backtracking
2. Forward checking
3. Partially Look Ahead
4. Fully look Ahead
5. Back Checking
6. Back Marking
7. Modified Forward checking

1. Backtracking

در زیر درخت سمت چپ ۶ فراخوانی داشته ایم . ۶ مقدار دهی و ۲

در این روش زمانی constraint ها را چک می کنیم که نوبت متغیر شده باشد

2. Forward Checking



3. Partially Look Ahead

تاثیر متغیر هایی که مقدار نگرفته اند نسبت به هم بررسی می کنیم

آیا می شود جوری مقدار داد تا مقداری برای متغیر دیگری باقی بماند ؟
در این مثال فرض می کنیم می خواهیم تاثیر هرمتغیر را بر روی بعدی ببینیم

در زیر درخت سمت چپ در گره دوم
خانه سوم از متغیر چهارم حذف نشده

است چون

partially Look Ahead

میباشد

Q			
FC	FC	PL	
FC	PL	FC	
FC			FC

BT	Q		
FC	FC	FC	
FC			FC
FC			

Q			
FC	FC	PL	Q
FC	PL	FC	FC
FC	FC		FC

هر وقت LookAhead داریم حتما
هم داریم !!!

BT	Q		
FC	FC	FC	Q
Q	FC	FC	FC
FC	FC		FC

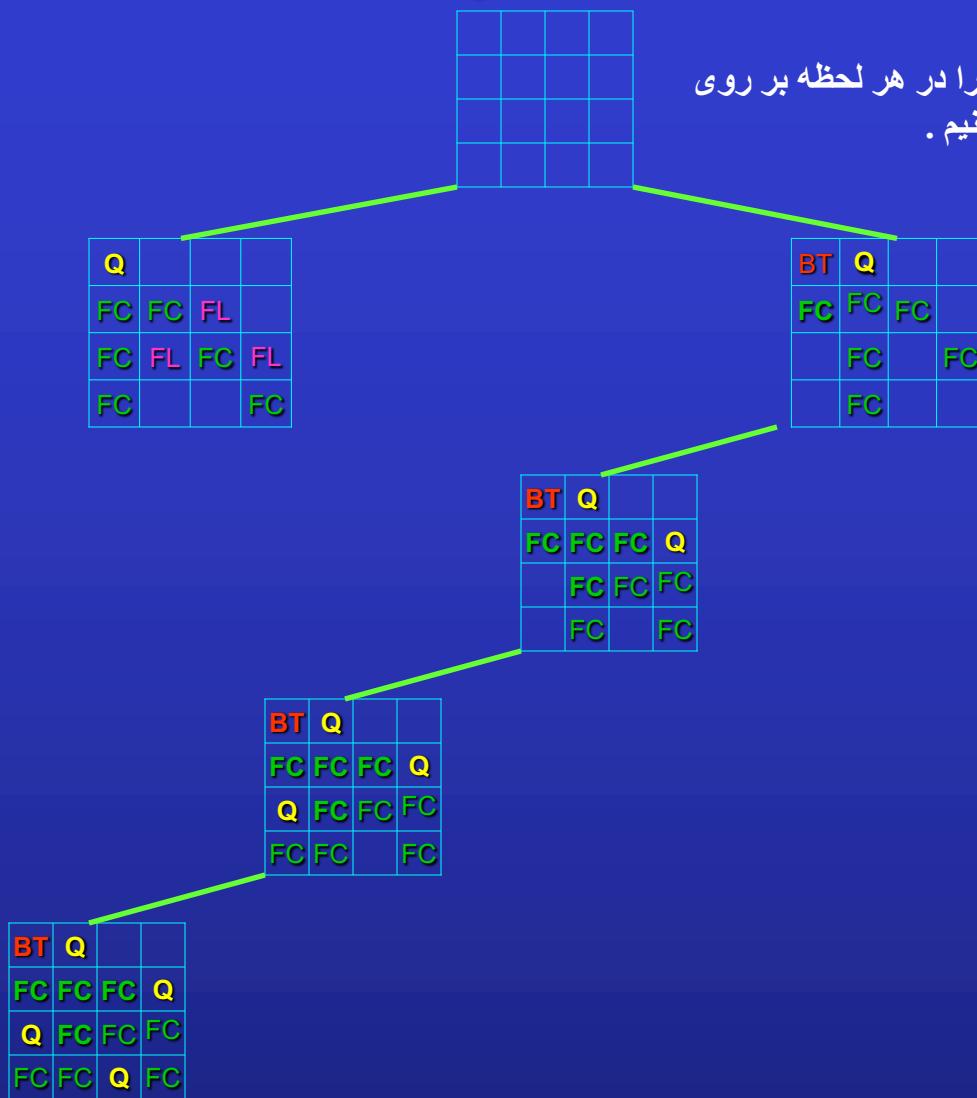
همواره ترکیب های دو تایی را بررسی می کنیم بنابراین همواره
کاری که انجام میدهیم از مرتبه Polynomial می باشد .

زیر درخت سمت چپ : فقط ۲ فراخوانی
۲ مقدار دهی

BT	Q		
FC	FC	FC	Q
Q	FC	FC	FC
FC	FC	Q	FC

4. Fully Look Ahead

تاثیر همه متغیرهایی که مقدار نگرفته اند را در هر لحظه بر روی یک دیگر بررسی می کنیم.



زیر درخت سمت چپ ۱ فراخوانی

Rest of CLP Search Techniques

5. Back Checking:
Remembering Previous failures
6. Back Marking:
Remembering Previous failures and successes
7. Modified Forward Checking:
Representing constraints as bit patterns and using AND/OR operations to test the patterns

K-Satisfiability

A CLP Problem with n variables is **K-satisfiable**, ($k \leq n$), if for every subset of K variables, there exists a k-label (values for k variables) that satisfies all the problem constraints

اگر در فضای کوچکتر جواب پیدا نشد قطعاً در فضای بزرگتر هم جواب وجود ندارد

If a problem is k-satisfiable, then it is k-1 satisfiable, too.

Consistency

Node consistency

- A node X is **node-consistent** if every value in the domain of X is consistent with X 's unary constraints
- A graph is node-consistent if all nodes are node-consistent

Arc consistency

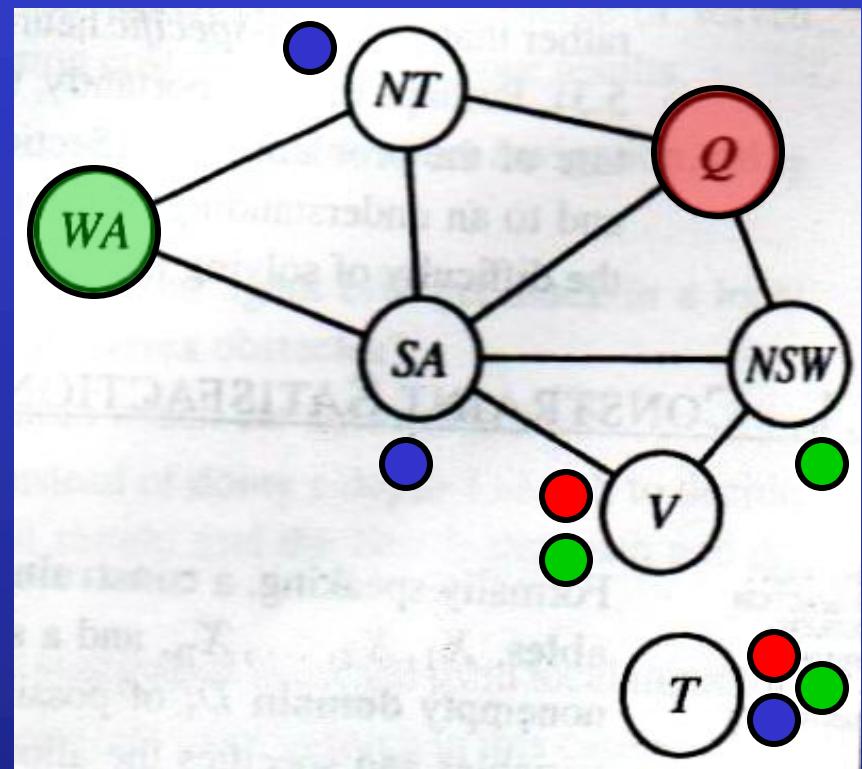
- به ازای هر مقداری که به X می دهیم همواره یک مقدار برای Y باقی بماند
- An arc (X, Y) is **arc-consistent** if, for every value x of X , there is a value y for Y that satisfies the constraint represented by the arc.
 - A graph is arc-consistent if all arcs are arc-consistent

To create arc consistency, we perform **constraint propagation**: that is, we repeatedly reduce the domain of each variable to be consistent with its arcs

Arc Consistency

$A \rightarrow B$ is consistent if for each remaining value in domain of A, there may be a consistent value in domain of B.

- Consistent:
 - $SA \rightarrow NSW$, $NSW \rightarrow V, \dots$
- Not Consistent:
 - $V \rightarrow NSW$, $NT \rightarrow SA, \dots$



Arc Consistency Checking Algorithm (AC-3)

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if RM-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k in NEIGHBORS[X_i] **do**

 add (X_k, X_i) to *queue*

function RM-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff remove a value

removed $\leftarrow \text{false}$

for each x in DOMAIN[X_i] **do**

if no value y in DOMAIN[X_j] allows (x, y) to satisfy constraint(X_i, X_j)

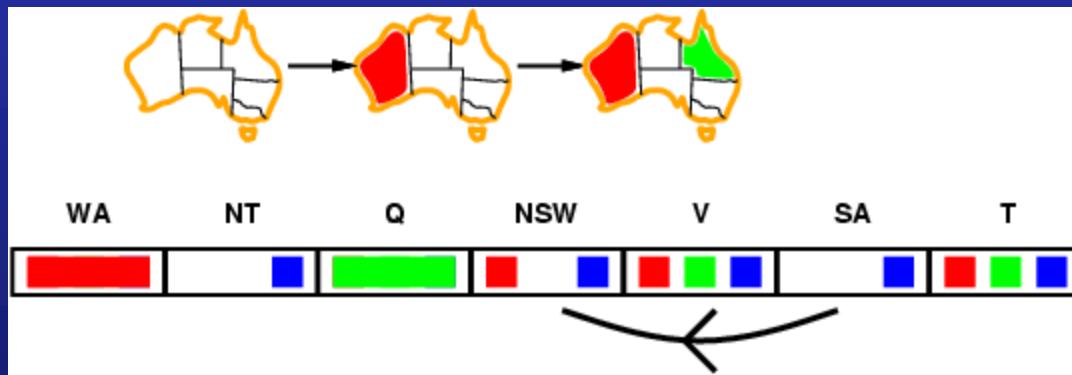
then delete x from DOMAIN[X_i]; *removed* $\leftarrow \text{true}$

return *removed*

Arc consistency

$X \rightarrow Y$ is consistent iff

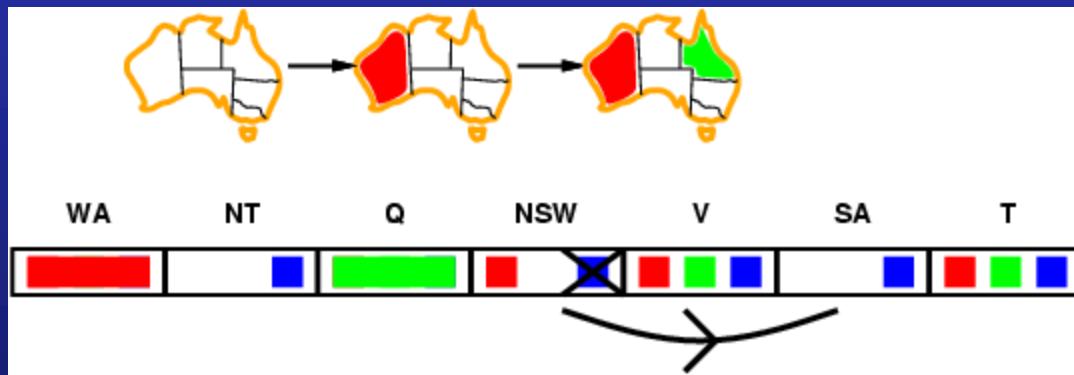
for every value x of X there is some allowed y



Arc consistency

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y

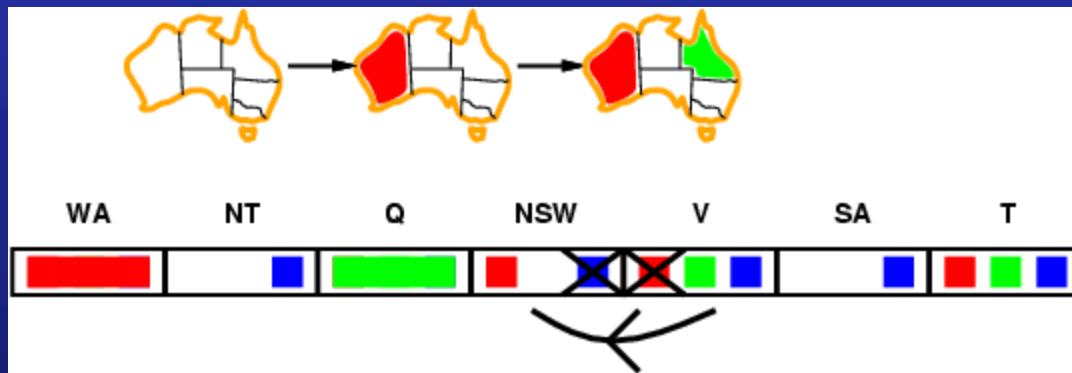


Arc consistency

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y

If X loses a value, neighbors of X need to be rechecked



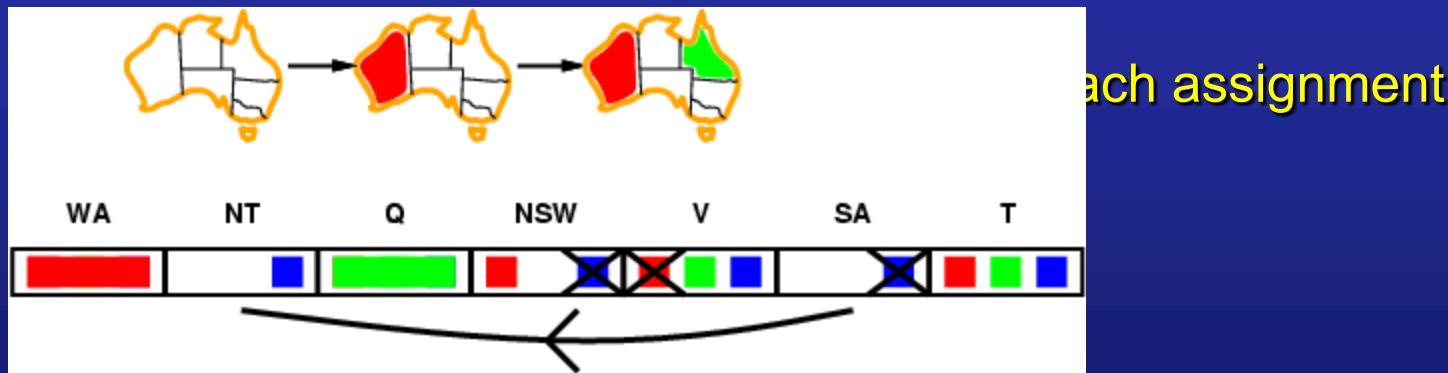
Arc consistency

$X \rightarrow Y$ is consistent iff

for every value x of X there is some allowed y

If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking



K-consistency

K-consistency generalizes the notion of arc consistency to sets of more than two variables.

- A graph is K-consistent if, for legal values of any K-1 variables in the graph, and for any Kth variable V_k , there is a legal value for V_k

Strong K-consistency = J-consistency for all $J \leq K$

Node consistency = strong 1-consistency

Arc consistency = strong 2-consistency

Path consistency = strong 3-consistency

بنابراین گره هایی وجود دارد که پایی به گره های بعدی و بالی هم به گره های قبلی دارند
به ازای هر NODE تعداد پال هایی که به node های قبلی دارد را عرض می کویند
به ازای ترتیب NODE های درون ترتیب را عرض ترتیب می کوییم
در هر گراف مینیمم روی عرض ترتیب را عرض گراف می کوییم

Width of a constraint graph

- We can have different orderings of a constraint graph
- The number of backward arcs of a node, in an specific ordering, is called the node's **width**
- An ordering width is the maximum width of its nodes
- A graph width is the minimum width of its different orderings

Why do we care?

If the width of the constraint graph of a CSP is D and it is **strongly K-consistent**, then if $K > D$, we can solve the CSP without backtracking, if we use an **appropriate variable ordering** (i.e., one with Minimal width ordering)

اگر قرار است عرض گراف D باشد یک ترتیبی باید وجود داشته باشد که در آن ترتیب هر node ای حداقل به D تا قبیل خود بyal داشته باشد ، داخل آن ترتیب

همان ترتیب را در نظر می گیریم یعنی ترتیب مقدار دادن به متغیرها را همان ترتیبی در نظر می گیریم که عرض D را برای ما ایجاد کرده

شروع می کنیم طبق همان ترتیب به متغیر ها مقدار دهیم :
متغیر اول هر مقداری که بخواهد را میگیرد

حال به صورت استقرایی فرض می کنیم به n متغیر مقدار داده ایم و می خواهیم به متغیر $n+1$ ام هم مقدار بدهیم :

طبق تعریفی که از عرض گراف داشتیم این متغیر $i+1$ امی مقدارش حداقل به D متغیر وابسته است ولی از طرفی ما می دانیم که $k > D$ برای k برابر باشد k consistensy برقرار است بعضی به هر شکل به آن D متغیر مقدار بدهیم چون $D < k$ بنا براین برای متغیر جدید یک مقدار پائی می داند همان مقدار را به متغیر جدید می دهیم این کار را می توان آن قدر ادامه داد تا متغیرهای ما تمام شوند (اثبات این که چرا اگر شرایط بالا برقرار باشد می توان بدون backtracking مسئله را حل کرد .)

Intelligent Backtracking

در اینجا نباید به تصمیم قبلی backtrack کرد چون که هیچ تأثیری بر روی متغیر که هیچ مقداری برای آن باقی نمانده ندارد.

$Q \leftarrow \text{Red}$

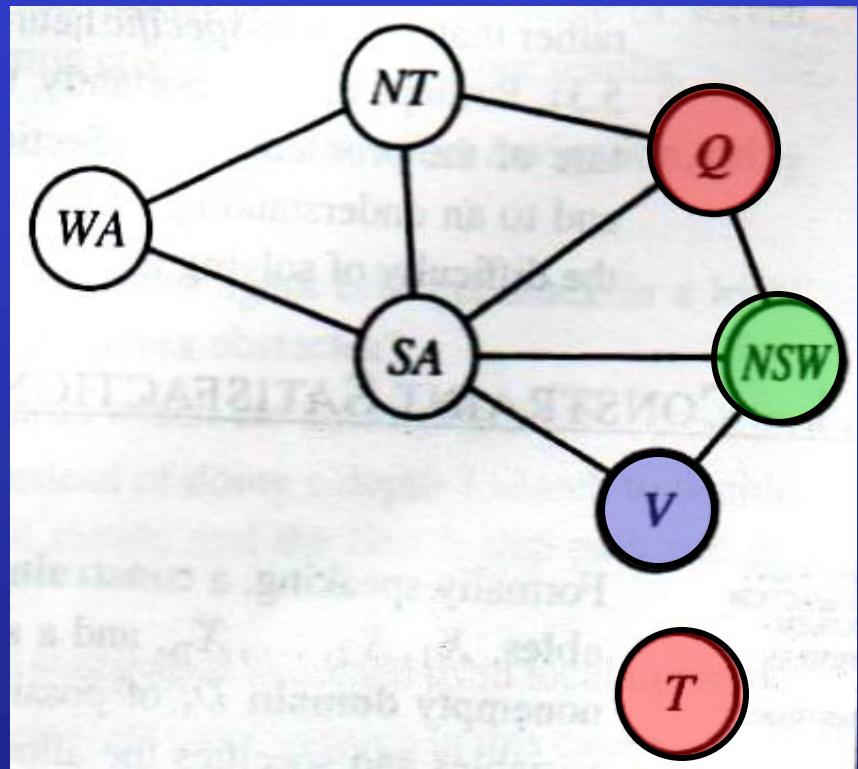
$\text{NSW} \leftarrow \text{Green}$

$V \leftarrow \text{Blue}$

$T \leftarrow \text{Red}$

$\text{SA} \leftarrow ?$

Chronological Backtracking



Back Jumping

از روی :

1- constraints

2- values assigned to previous nodes

3- constraint propagation methods

باید استنتاج کرد که کدام مقادیر برای ما مشکل ایجاد می‌کند

$Q \leftarrow \text{Red}$

$\text{NSW} \leftarrow \text{Green}$

$V \leftarrow \text{Blue}$

$T \leftarrow \text{Red}$

$\text{SA} \leftarrow ?$

Conflict Set of SA:

{Q, NSW, V}

NOTE: Back Jumping
doesn't help Forward
Checking.

