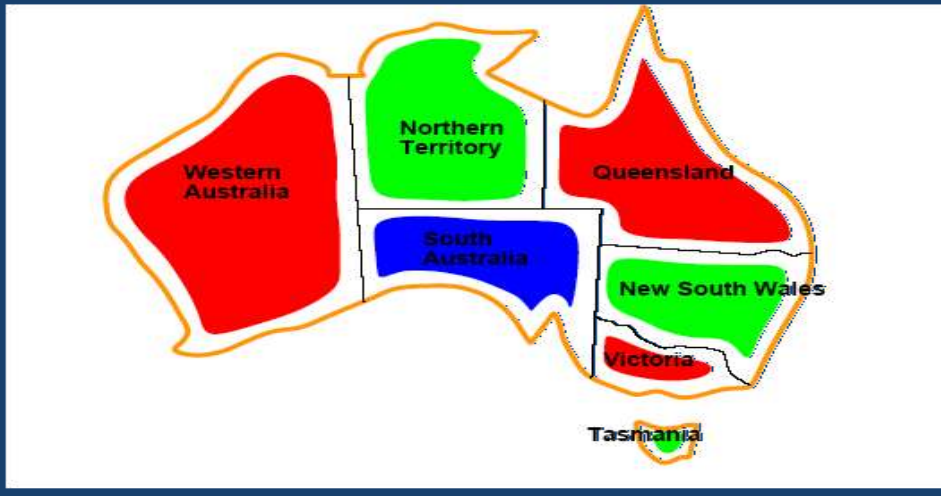
VEŠTAČKA INTELIGENCIJA 2020/21

PROBLEM ZADOVOLJENJA OGRANIČENJA (CONSTRAINT SATISFACTION PROBLEMS)

Sadržaj

- Šta je CSP
- Backtracking za CSP
- Povećanje efikasnosti za CSP



Prof. dr Leonid Stoimenov

Constraint satisfaction problems (CSPs)

- Traženje:
  - Stanje je “black box”, neka struktura podataka koja podržava elemente traženja (test na ciljno stanje, sledbenici,…)

- **CSP:**
  - Stanje je definisano preko promenljivih X_i sa vrednostima iz domena D_i
  - Test na ciljno stanje je skup ograničenja (*constraints)* koja specificiraju dozovoljene kombinacije vrednosti za podskup promenljivih

Definicija CSP

- Konačan **skup promenljivih** $V_1, V_2, \ldots, V_n$
  - (Ne-prazni) domeni mogućih vrednosti za svaku promenljivu: $D_{V1}, D_{V2}, \ldots D_{Vn}$
- Konačan **skup ograničenja:** $C_1, C_2, \ldots, C_m$
  - Svako ograničenje C_i ograničava vrednosti koje promenljive mogu da uzimaju,
    - Primer, $V_1 \neq V_2$

- ***Stanje*** se definiše kao *dodela* vrednosti nekim promenljivima.
- *Konzistentna dodela (Consistent assignment):*
  - Dodela vrednosti ne narušava ograničenja

Definicija CSP (nastavak)

- Dodela je ***kompletna*** kada je su uključene sve promenljive.
- *Rešenje* za CSP je kompletna dodela koja zadovoljava sva ograničenja.
- Neki CSP zahtevaju rešenje koje maksimizuje funkciju cilja.
- Primeri primene CSP:
  - Scheduling (Airline schedules)
  - Kriptografija
  - Computer vision -> image interpretation

Prednosti CSP u odnosu na traženje

- Jednostavan primer formalnog jezika za reprezentaciju
  - Standardni paterni za reprezentaciju
- Generički cilj i funkcije sledbenika
- Generička heuristika (nema domenski specifične ekspertize).
- Dozvoljava primenu algoritama generalne namene koji imaju bolje karakteristike od uobičajenih algoritama traženja

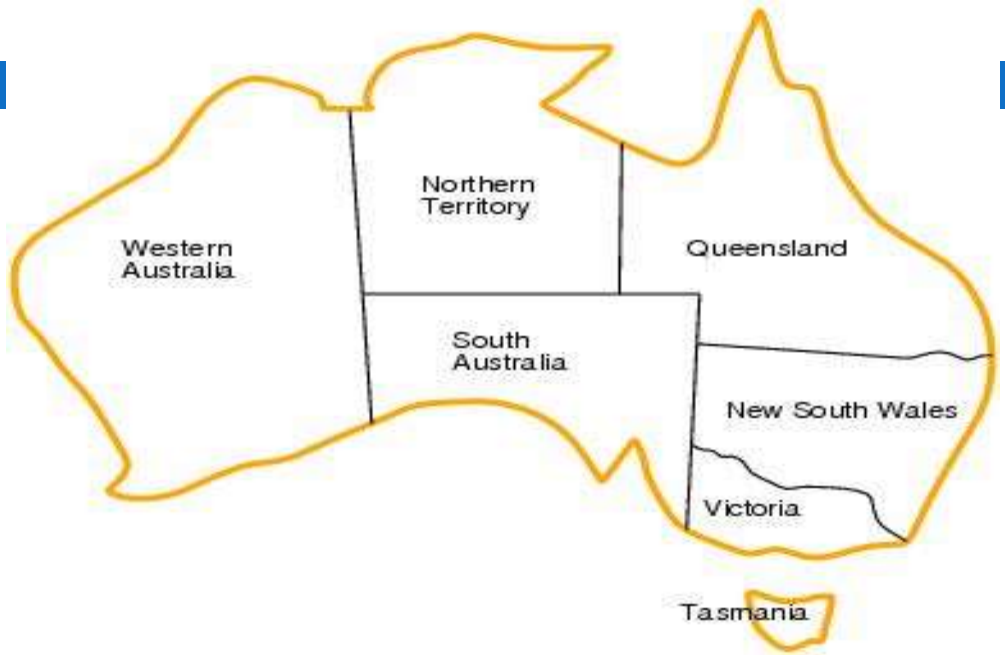
Vrste CSP

- Diskretne promenljive
  - **Konačni domeni**; veličina $d \Rightarrow O(d^n)$ komplenih dodela.
    - Napr. Boolean CSP: Boolean satisfiability (NP-complete).
  - Beskonačni domeni (integer-i, string-ovi, itd.)
    - Napr. job scheduling, promenljive su start/end dani za svaki posao
    - Potreban je jezik za zadavanje ograničenja napr. $StartJob_1 + 5 \leq StartJob_3$.
    - Beskonačno mnogo rešenja
    - Linearna ograničenja
    - Nelinearna: nema generalnog algoritma
- Kontinualne promenljive
  - Napr. Određivanje rasporeda letenja ili rasporeda časova.
  - Linearna ograničenja, rešiv za polinomno vreme korišćenjem metoda linearnog programiranja.

Vrste ograničenja

- ***Unarno*** ograničenje utiče na jednu promenjivu,
  - napr. SA ≠green
- ***Binarno*** ograničenje uključuje par promenjivih,
  - napr. SA ≠ WA
- ***Ograničenje višeg reda***, *globalno* ograničenje uključuje 3 i više promenjivih,
  - napr. Profesori A, B, i C ne mogu biti zajedno u komisiji
  - Uvek može biti predstavljeno preko više binarnih ograničenja
- **Preferenca** (soft ograničenja)
  - Napr. *Crveno* je bolje nego *zeleno* često se mogu predstaviti preko cene za svaku dodelu vrednosti promenljivoj
  - Kombinacija optimizacije sa CSPs

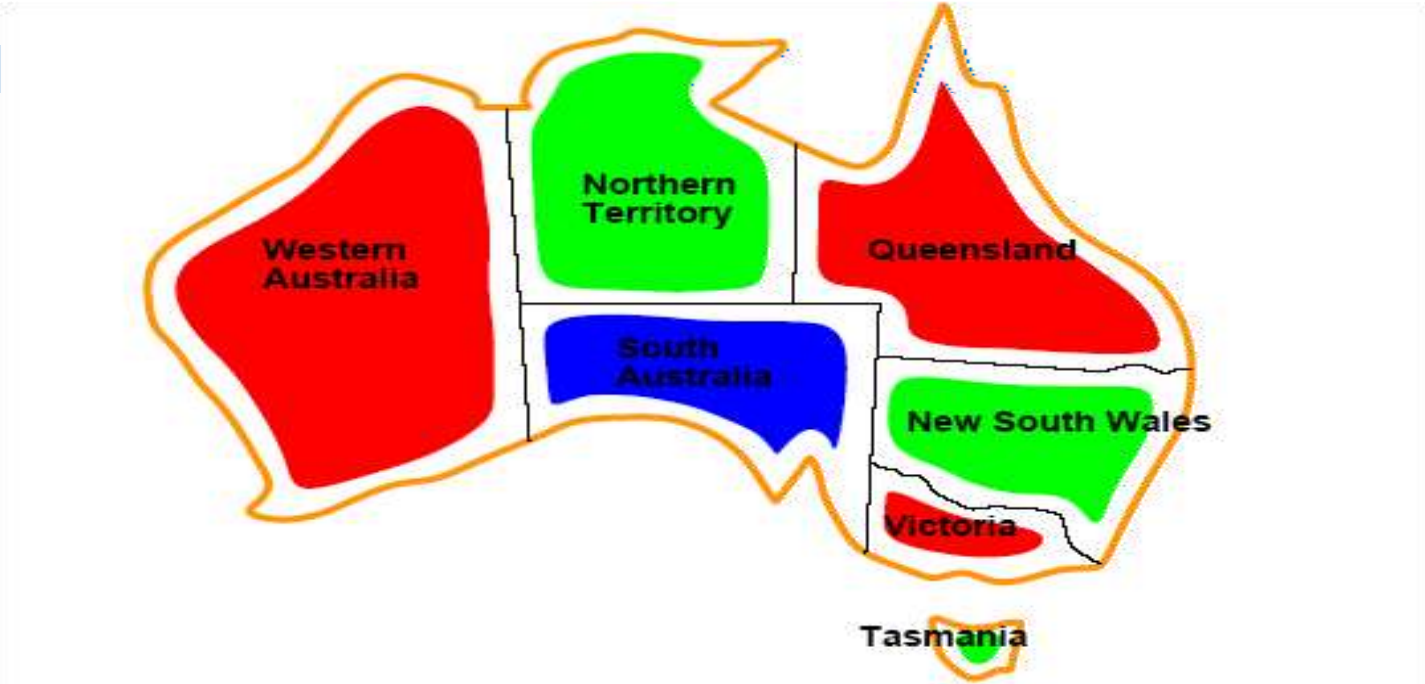
Primer CSP: bojenje mape



Problem predstavljen kao CSP:

- Promenljive: *WA, NT, Q, NSW, V, SA, T*
- Domeni: D_i={*red,green,blue*}
- Ograničenja: susedni regioni moraju da imaju različite boje.
  - Primer: $WA \neq NT$

Primer CSP: bojenje mape



- **Rešenja** su dodele koje zadovoljavaju sva ograničenja, npr.

  {*WA=red,NT=green,Q=red,NSW=green,V=red, SA=blue,T=green*}

(kompletna i konzistentna dodela)

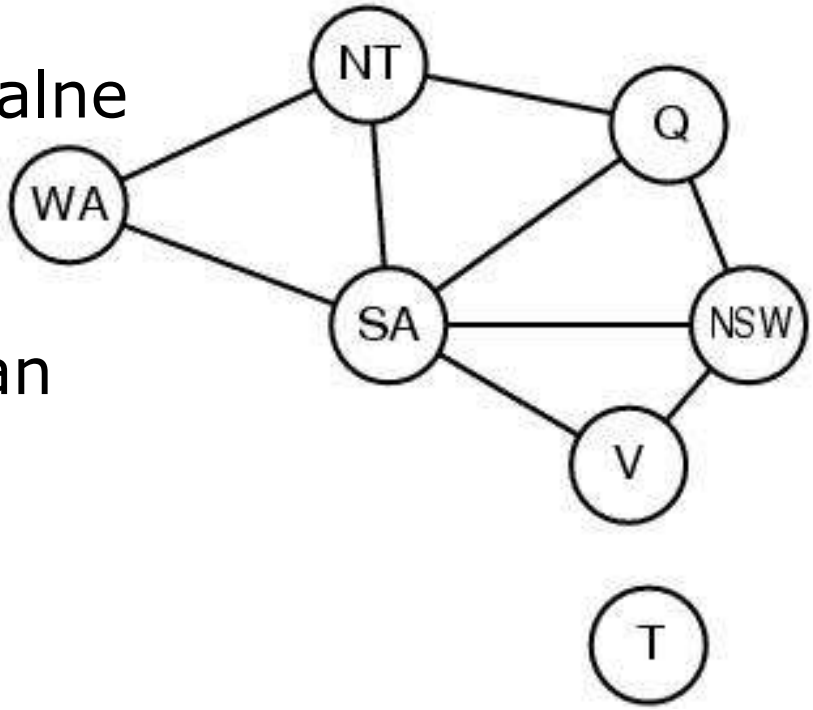
Graf ograničenja (Constraint graph)

- **Binarni CSP**

- **Graf ograničenja** čine:
  - čvorovi - promenljive
  - potezi - binarna ograničenja

- Koriste se algoritmi generalne namene za CSP (da se pojednostavi traženje)

Napr. *Tasmania* je nezavistan podproblem



Primeri primene CSP

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- Notice that many real-world problems involve real-valued variables

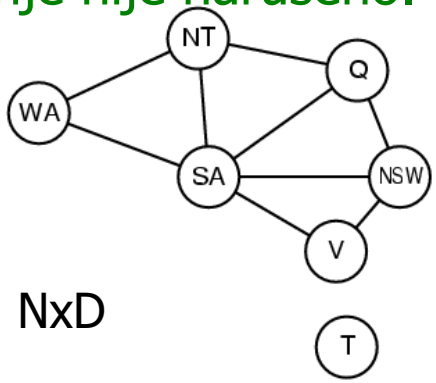
CSP kao standardni problem traženja

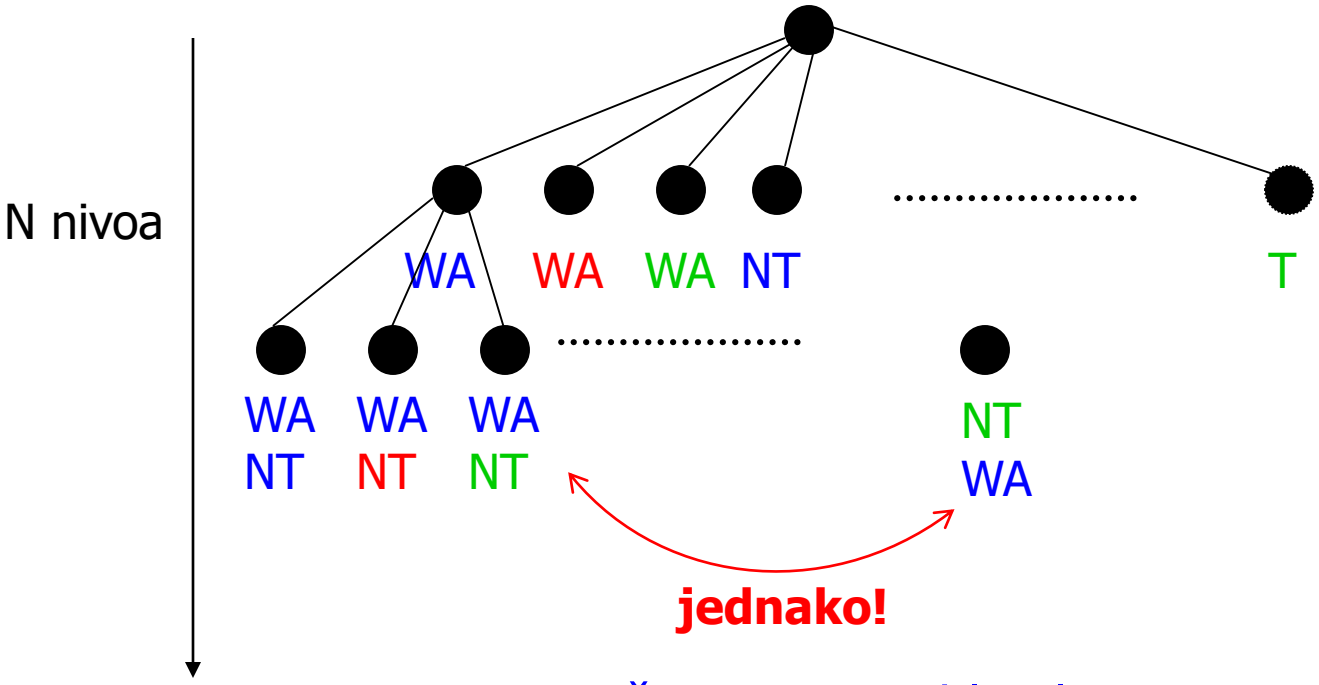
- CSP se jednostavno može predstaviti kao standardni problem traženja.
- Formulacija
  - ***Početno stanje***: prazna dodela {}
  - ***Funkcija sledbenika***: Dodela vrednosti bilo kojoj nedeodeljenoj promenjivoj tako da ne narušava ograničenje
  - ***Test na ciljno stanje***: tekuća dodela je kompletna
  - *Cena puta*: konstantna cena za svaki korak (praktično, nije važno)

Standardna formulacija za traženje

Potrebno nam je:

- Inicijalno stanje: nijedna promenljiva nema vrednost (boju)
- Sledbenik: jedna od promenljivih bez vrednosti dobija neku vrednost.
- Cilj: sve promenjive imaju vrednost i nijedno ograničenje nije narušeno.





NxD

[NxD]x[(N-1)xD]

N! x D^N

Postoji N! x D^N čvorova u stablu ali samo D^N različitih stanja??

Backtracking (Depth-First) traženje

- Specijalna osobina CSP: dodele su komutativne
  Značenje: redosled dodele vrednosti nije od važnosti.
- Treba razmatrati dodelu samo jednoj promenljivoj u svakom čvoru
- **Depth-first search za CSP sa single-variable dodelama se zove Backtracking search**
  - Backtracking search je osnovni neinformisani algoritam za CSP
  - Može da reši *n-queens* za n ≈ 25
- Bolje stablo traženja: Najpre uredi promenljive, nakon toga im dodeljuje vrednosti jedna-po-jedna.

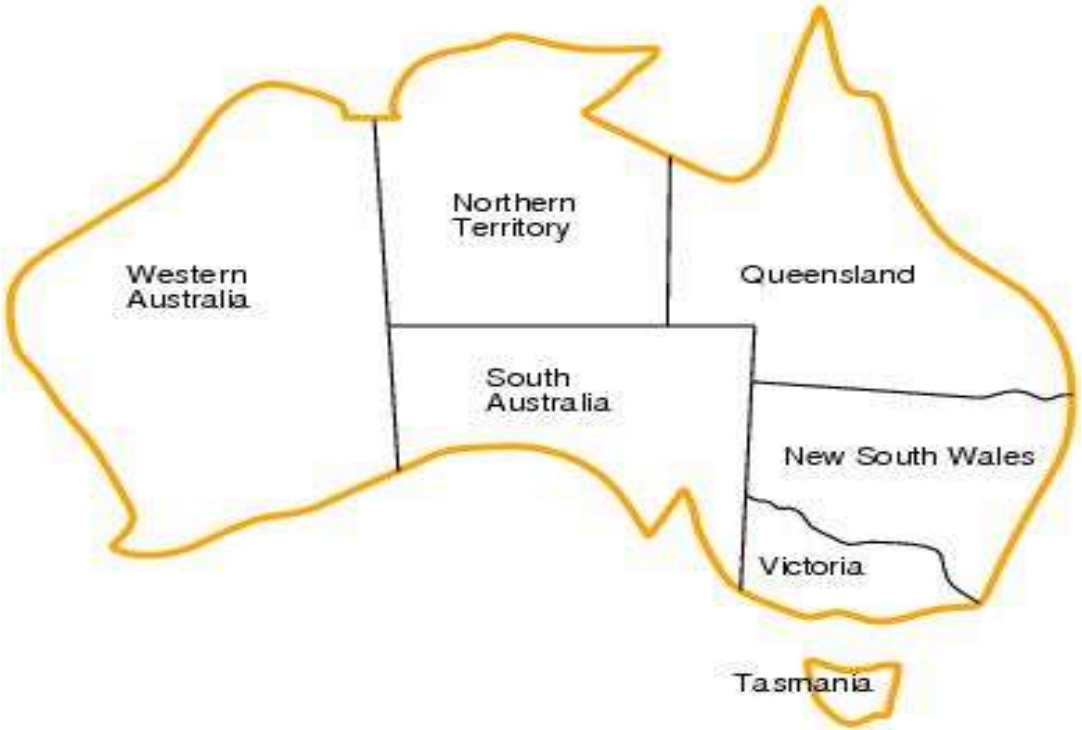
Backtracking traženje

- U osnovi je DFS
- Bira vrednosti za jednu promenljivu i vraća se kada za promenljiva nema legalnih vrednosti za dodelu
- Neinformisani algoritam
  - Generalno nema dobre performanse

Backtracking traženje

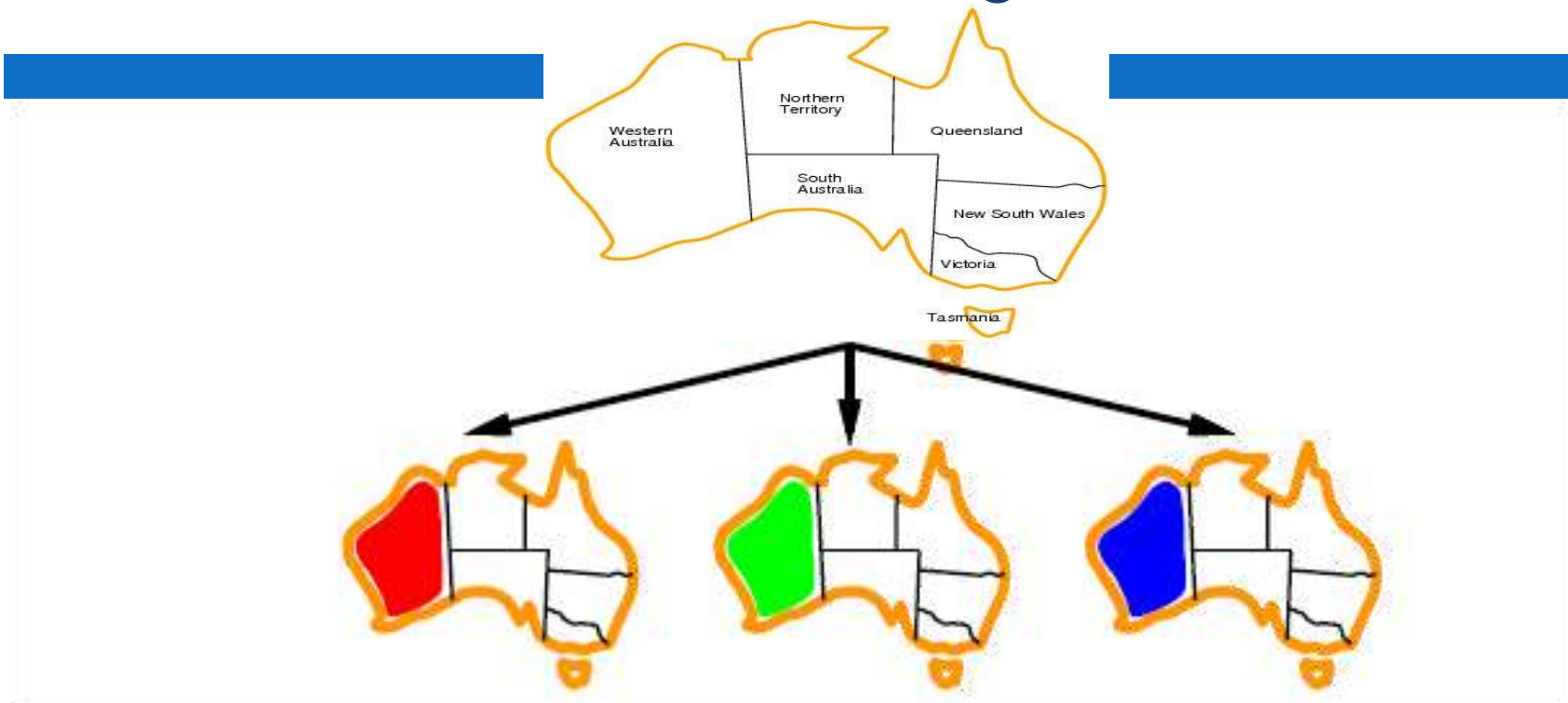
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or
    failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp]
        then
                    add {var=value} to assignment
                    result ← RECURSIVE-BACTRACKING(assignment, csp)
                    if result ≠ failure  then return result
                    remove {var=value} from assignment
    return failure
```

Primer za Backtracking



- Početno stanje

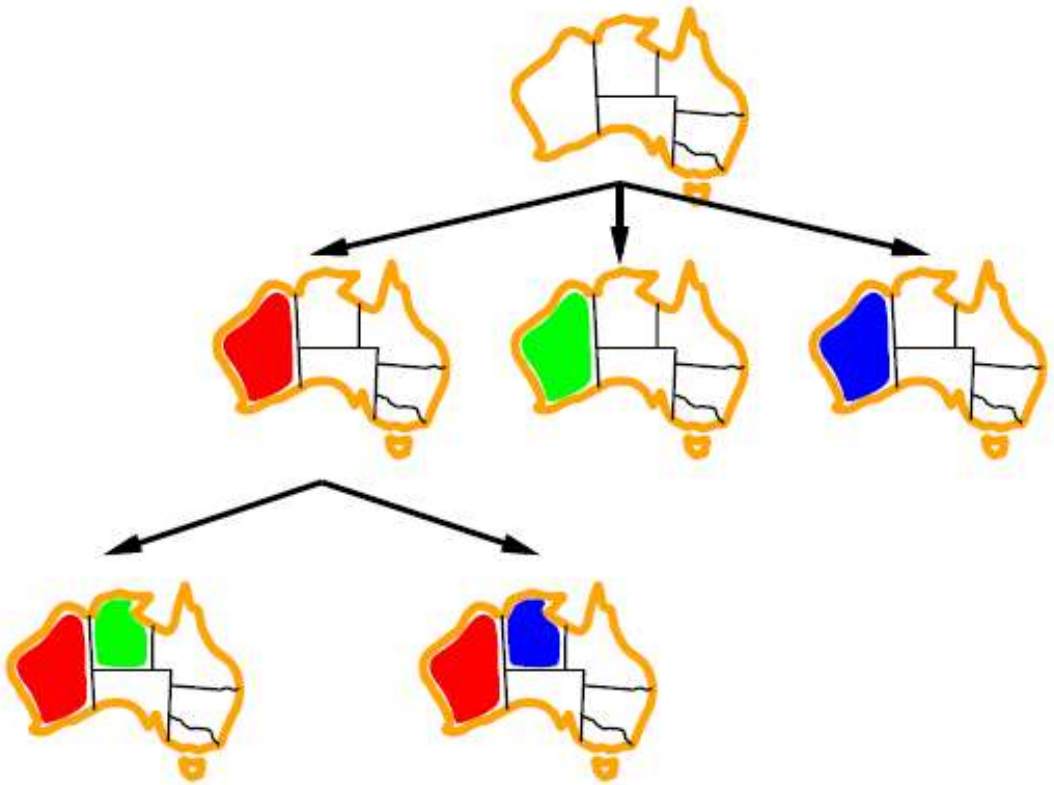
Primer za Backtracking



WA=red WA=green WA=blue

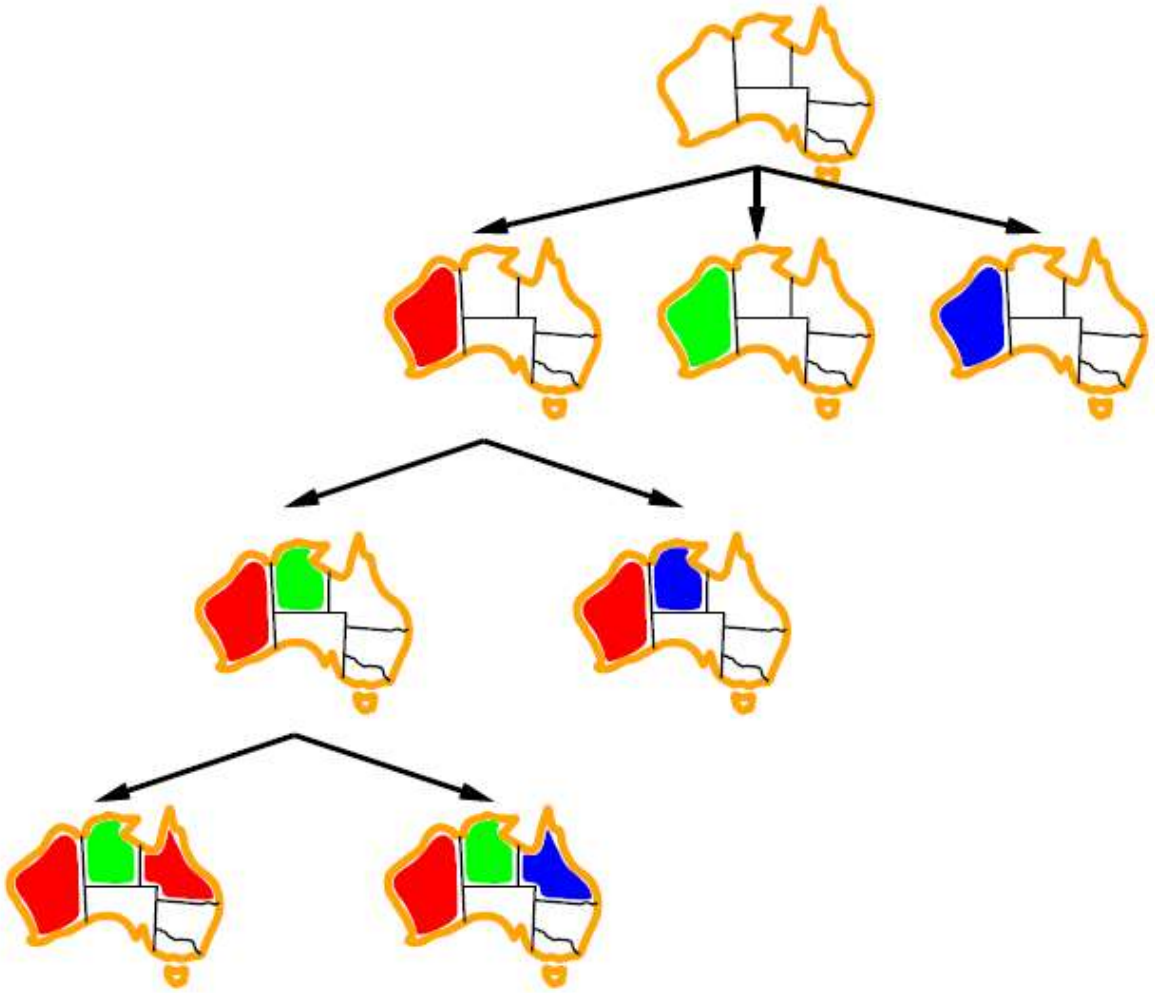
- Dodela (mogući sledbenici)

Primer za Backtracking



NT=green NT=blue

Primer za Backtracking



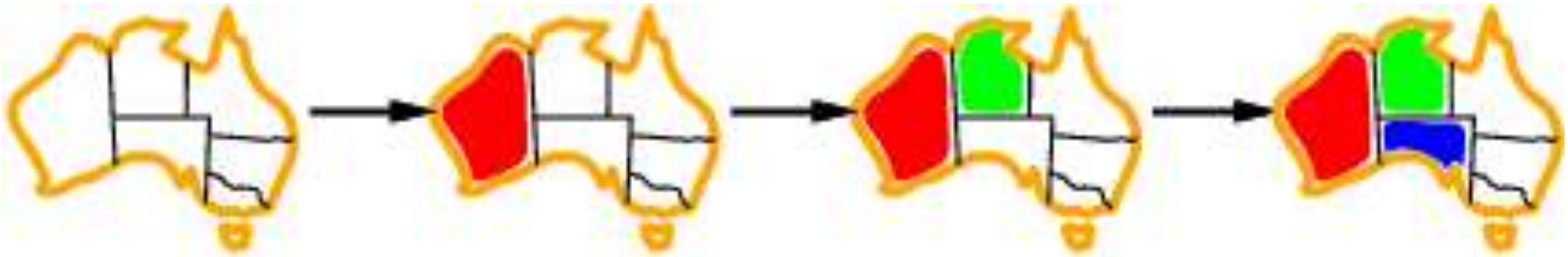
Q=red Q=blue • • •

Povećanje efikasnosti CSP

- Koriste se metode koje su generalne namene, napr.:
  - Koja promenljiva je sledeća za dodelu?
  - U kom redosledu treba uzimati vrednosti za dodelu?
  - Da li se neizbežni neuspeh može detektovati ranije?
  - Možemo li da iskoristimo strukturu problema?

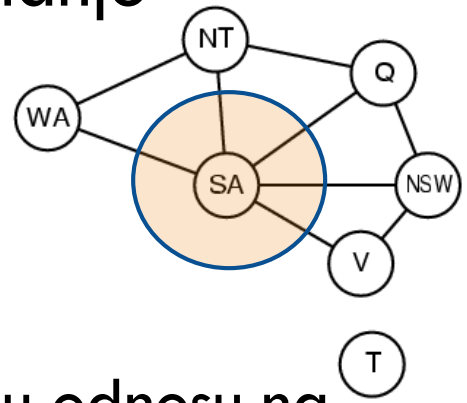
Koja promenljiva je sledeća za dodelu?
Most constrained variable → **MRV** heuristika

- Izaberi promenljivu koja ima **najmanje legalnih vrednosti** (*Most constrained variable*)
- Primer: **Minimum remaining values (MRV)** heuristika
- **Cilj heuristike:** Izaberi promenljivu koja će prouzrokovati grešku što ranije, omogućavajući odsecanje kod stabla traženja.

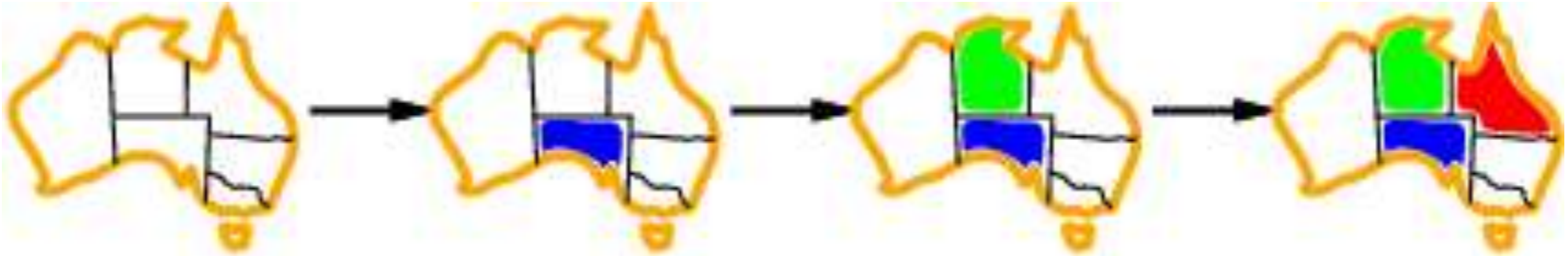


Koja promenljiva je sledeća za dodelu?
Most constrained variable → Degree Heuristic

- Tie-breaker među promenljivima sa najmanje legalnih vrednosti



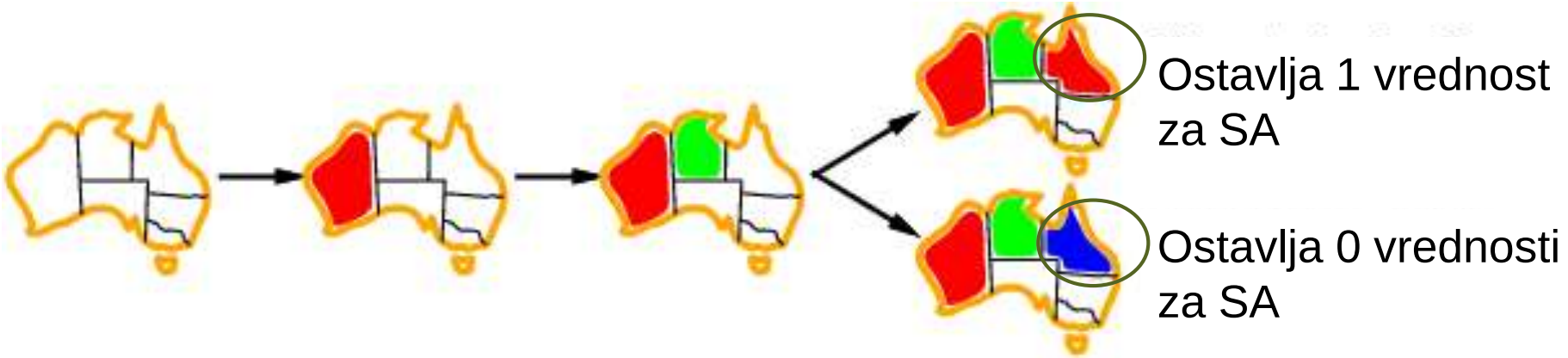
- **Degree heuristics:**
  - Izaberi promenljivu s **najviše ograničenja** u odnosu na preostale promenljive (najviše potega u grafu)



U kom redosledu treba uzimati vrednosti za dodelu?

→ Least Constraining Value Heuristic (**LCV**)

- *Za promenljivu*, izaberi **najmanje ograničavajuću vrednost:**
  - Ona vrednost koja u preostalim promenljivima najmanje smanjuje broj vrednosti



- Ostavlja maksimalnu fleksibilnost za rešenje.

Obrazloženje za MRV, DH, LCV

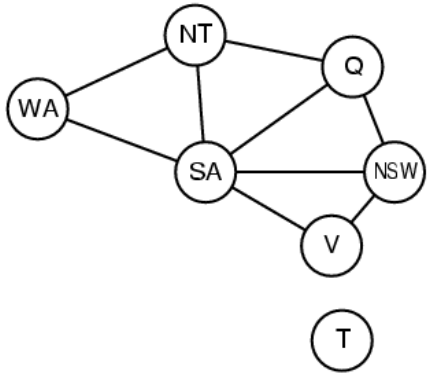
- U svim slučajevima cilj nam je da:
  - izaberemo granu koja najviše obećava,
  - ali takođe želimo da detektujemo neuspeh što ranije.
- MRV+DH: promenljiva koja najverovatnije prouzrokuje neuspeh najpre dobija vrednost.
- LCV: pokušava da izbegne neuspeh dodelom vrednosti koje ostavljaju najviše fleksibilnosti za preostale promenljive.

- Kombinovanjem ovih heuristika problem 1000 kraljica je izvodljiv

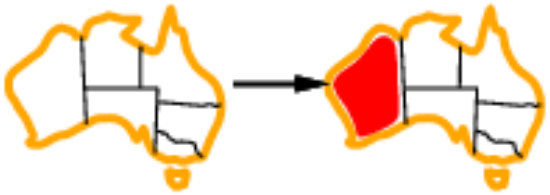
Da li se neizbežni **neuspeh** može detektovati ranije?

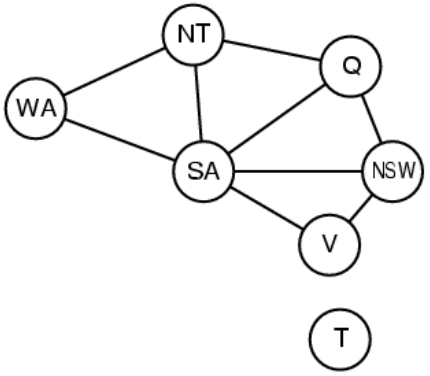
→ Forward Checking (**FC**)

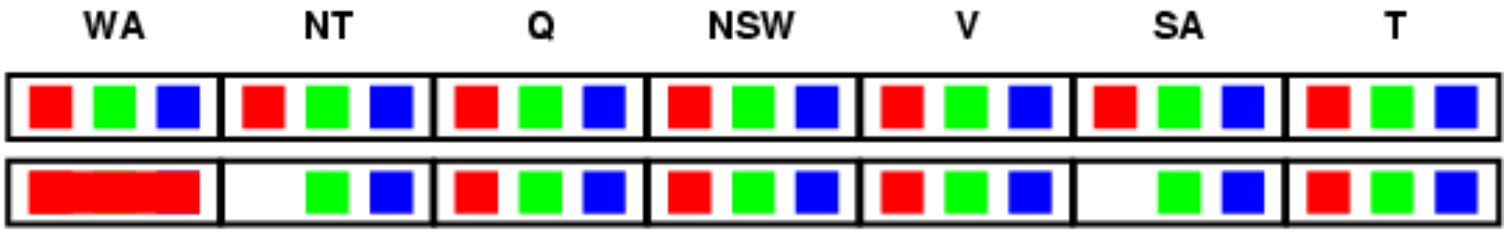
- Ideja:
  - Pratite preostale legalne vrednosti za nedodeljene promenljive koje su povezane sa tekućom promenljivom.
  - Prekinite traženje kad bilo koja promenljiva nema legalne vrednosti za dodelu.



Forward checking - nastavak



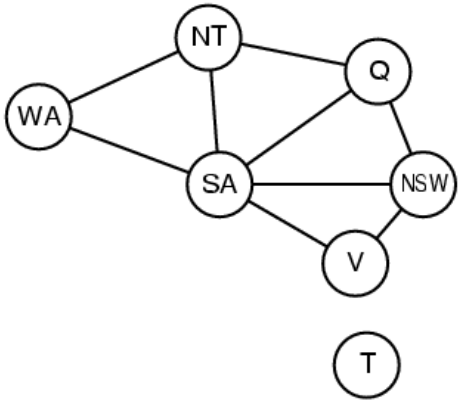




- Dodela {***WA=red***}
- Utiče na promenljive povezane preko ograničenja sa WA
  - *NT više ne može da bude red*
  - *SA više ne može da bude red*

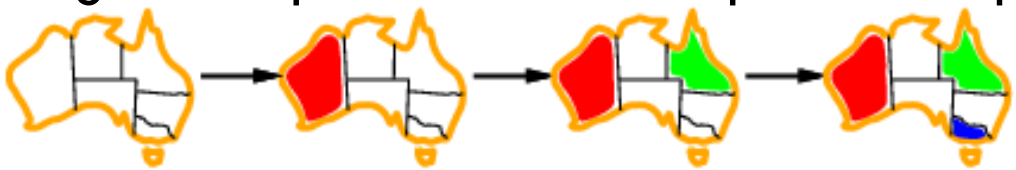
Forward checking - nastavak

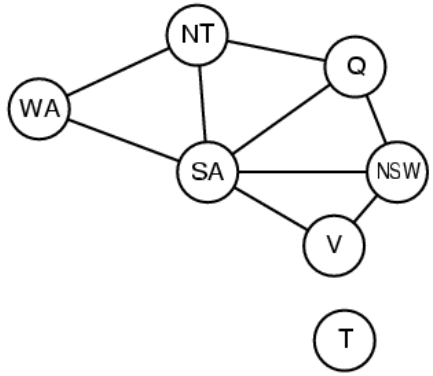
- Dedela {Q=*green*} *(MRV heuristika će automatski selektovati NT ili SA kao sledeću promenljivu, sigurno ne Q!!)*
- Utiče na promenljive povezane ograničenjem sa WA
  - *NT ne može biti green*
  - *NSW ne može biti green*
  - *SA ne može biti green*



Forward checking - nastavak

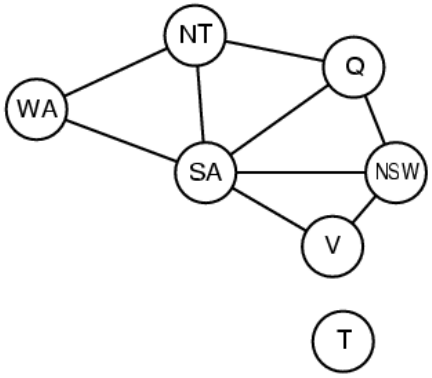
- Ako *V* dobije vrednost *blue*, *V*=*blue*
- Utiče na promenljive:
  - *NSW ne može biti blue*
  - *SA je prazno!*
- FC detektuje da je parcijalna dodela ***nekonzistentna*** sa ograničenjima i aktivira se povratak tj backtracking.





Forward checking - nastavak

- *Forward checking* jedino posmatra promenljive povezane sa tekućom vrednošću u grafu ograničenja.



- Rešvanje CSP korišćenjem kombincije **heuristika + FC** je efikasnije od svakog pristupa pojedinačno
- FC **ne** detektuje sve neuspehe:
  - NT i SA ne mogu obe da bude plave!

Propagacija ograničenja
Constraint propagation (CP)

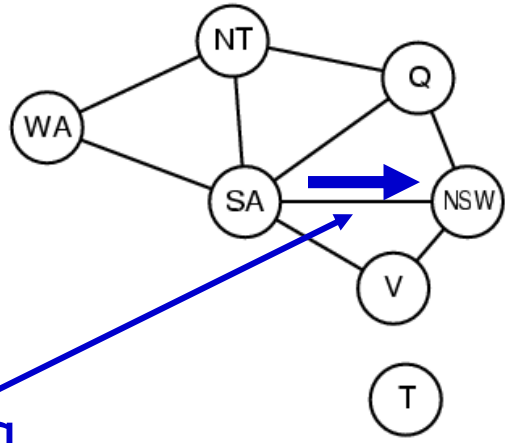
- Propagacija ograničenja (CP) propagira **konzistentnost potega** kroz graf.
- Tehnike kao što su CP i FC eliminišu deo prostora traženja
- CP ide dalje u odnosu na FC pošto više puta lokalno izvršava ograničenja
- **Arc-consistency** (**AC**) ili **konzistentnost potega**, je sistematska procedura (postupak) za propagaciju ograničenja

Konzistentnost potega – Arc consistency (**AC**)

- *X* → *Y* je **konzistentan** *akko*

  za **svaku** vrednost ***x*** iz ***X*** postoji **neko** dozvoljeno ***y***

- Jednostavna forma propagacije čini svaki poteg **konzistentnim**

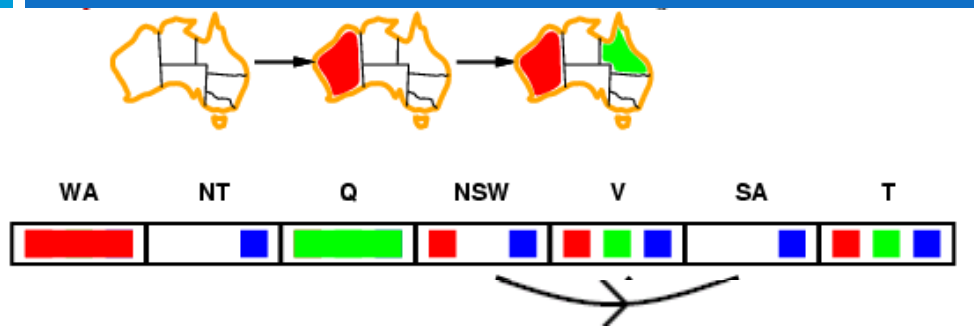


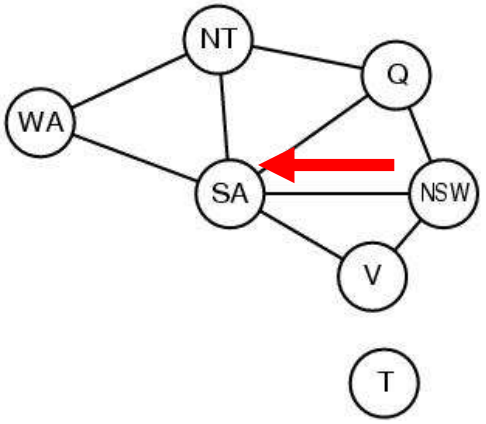
konzistentni poteg

- Stanje nakon dodele vrednosti za WA i Q:

  SA → *NSW* je konzistentno **ako** *SA=blue* i *NSW=red*

Arc consistency - nastavak



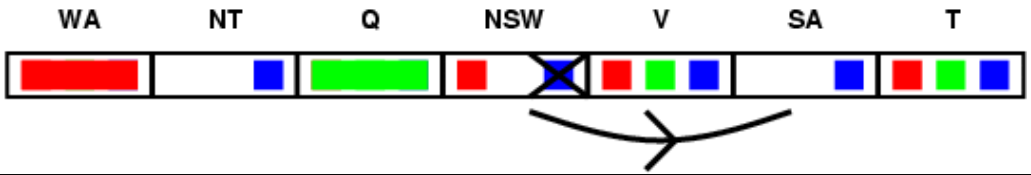


Provera:

- *NSW* → *SA* je konzistentno ako

  NSW=red i *SA=blue*

  NSW=blue i *SA=???*



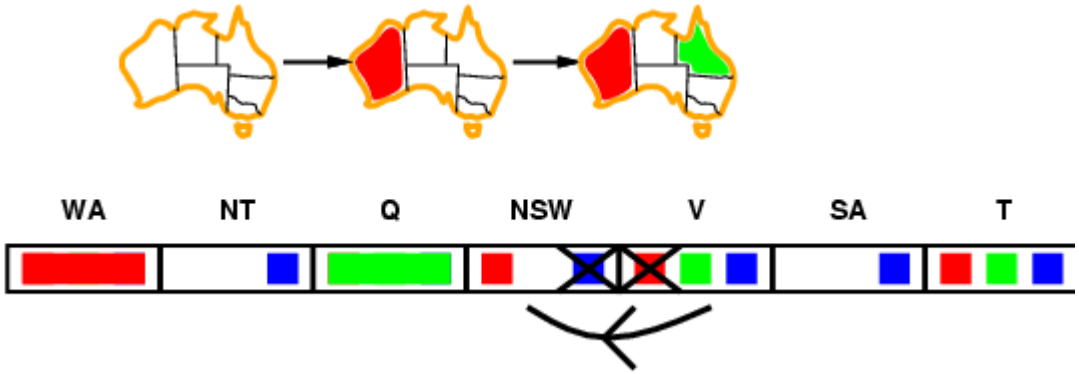
nekonzistentni poteg!!

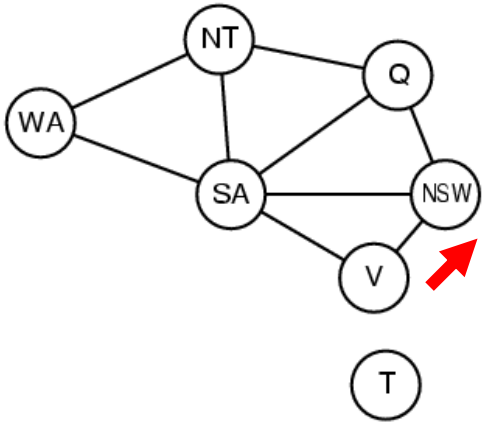
brisanje *blue* → konzistentni poteg

Arc consistency – nastavak

- **Ako *X* gubi vrednost**, susede od *X* treba ponovo proveriti: **dolazni potezi** mogu postati nekonzistentni ponovo (odlazni potezi ostaju konzistentni).

- U navedenom primeru se može forsirati konzistentnost potega: poteg može postati konzisten
  blue iz NSW (naredni slajd, nastavak)

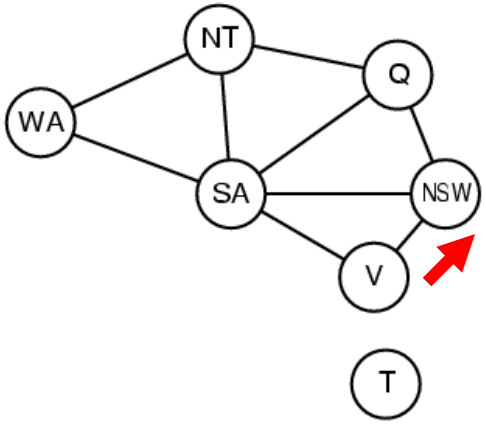


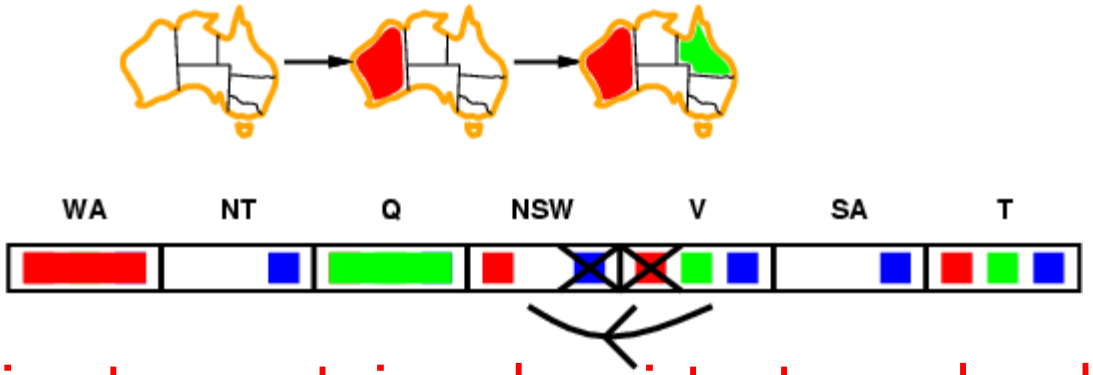


Arc consistency – nastavak

- Nastavlja se propagacija ograničenja….
  - Provera $V \rightarrow NSW$
  - Nije konzistentno za V = red
  - Postaje konzistntno:

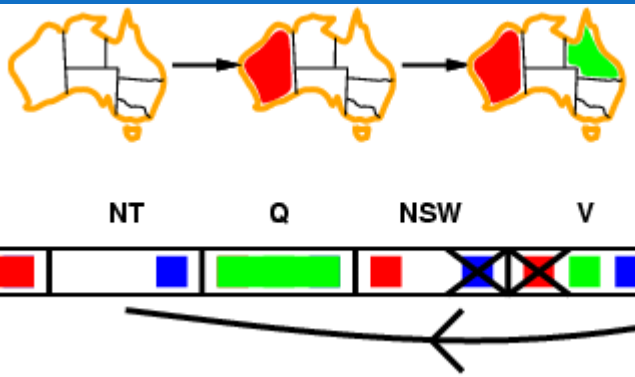
    Izbaci vrednost *red* iz *V*

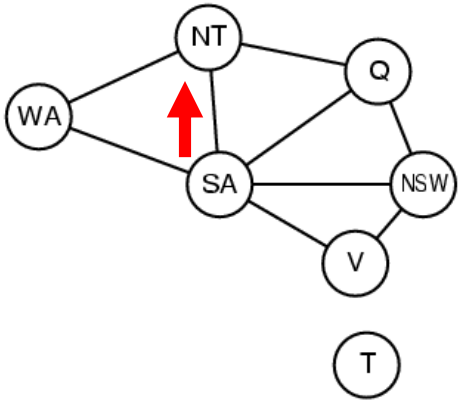




Ovaj poteg postaje nekonzistentan nakon brisanja *blue* iz NSW
i ponovo konzistentan nakon brisanja *red* iz V

Arc consistency - nastavak





- Propagacija ograničenja….
  - *SA* $\rightarrow$ *NT* nije konzistentno!
  - … **i ne može se učiniti konzistentnim**
  - **NEUSPEH!!**

Složenost: $O(n^2d^3)$

- AC detektuje neuspeh ranije nego FC
- Može da se izvrši kao pretprocesoranje ili nakon svake dodele

Arc Consistency - analiza

- Propagacioni algoritam – kao slanje poruka susedima u grafu! Kako možemo da planiramo takve poruke?
- Svaki put kada se domen promenljive menja, sve dolazne poruke se moraju ponovo slati. Ponavljati sve dok nema poruka koje menjaju bilo koji domen.
- Prazni domen znači da nema mogućeg rešenja – povratak iz te grane!
- **Forward checking** je jednostavno slanje poruka promenljivoj koja je upravo dobila svoju vrednost. Praktično **prvi korak iz arc-consistency**.

Arc consistency algoritam (AC-3)

function AC-3(*csp*) **return** the CSP, possibly with reduced domains
  inputs: *csp*, a binary csp with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs initially the arcs in *csp*
  while queue is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS[X_i] **do**
      add (X_i, X_j) to queue

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **return** *true* iff we remove a value
  removed $\leftarrow$ *false*
  for each *x* **in** DOMAIN[X_i] **do**
    if no value *y* in DOMAIN[X_j] allows (x,y) to satisfy the constraints between X_i and X_j
    then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

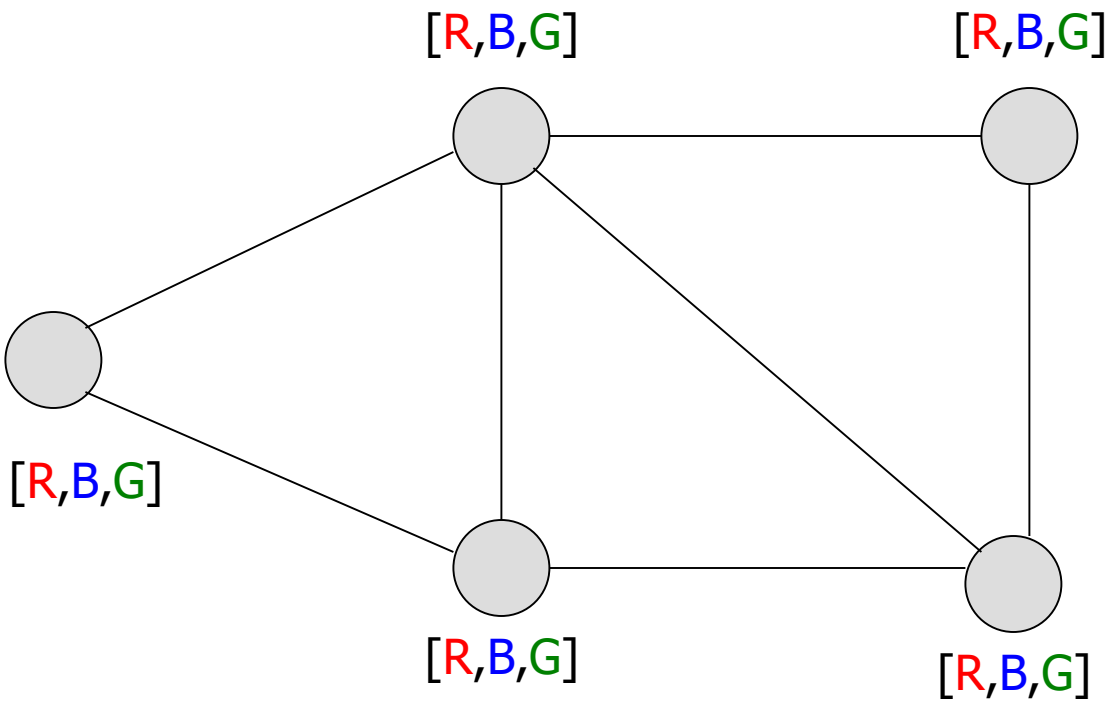
(Mackworth, 1977)

Primer za vežbu: bojenje mape

Zadatak:

1) Definisati problem kao CSP

2) Primenite sve heuristike za rešavanje ovog problema!



Definicija CSP??

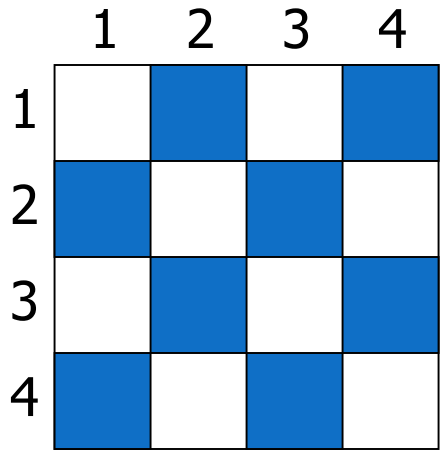
1. **Promenljive?**
2. **Domeni?**
3. **Ograničenja?**

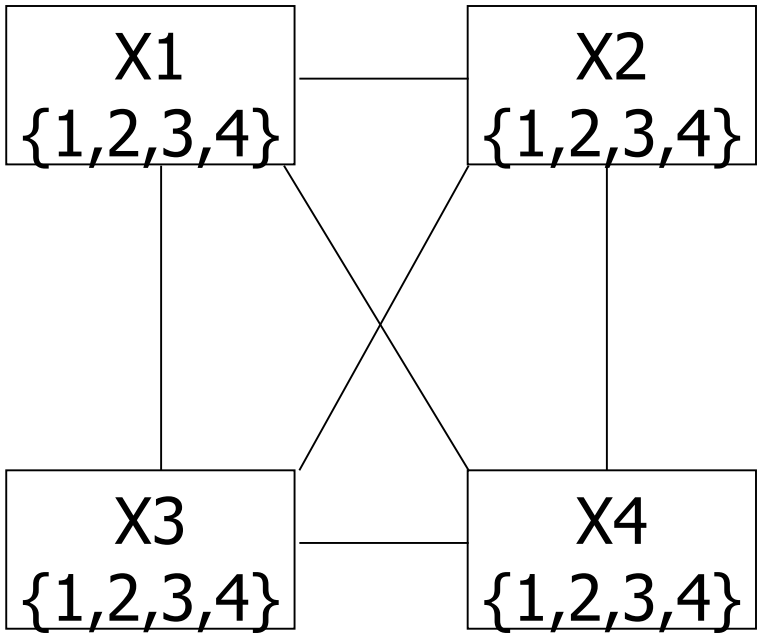
Heuristike

1. **MRV**
2. **DH**
3. **LCV**

Primer za vežbu: 4-Queens

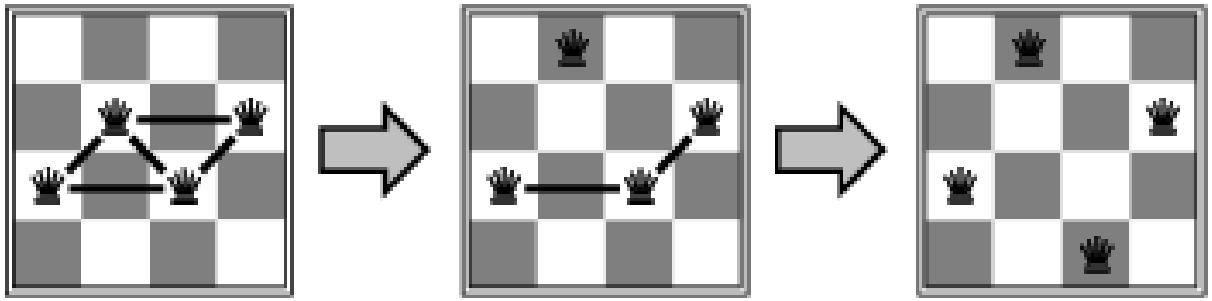
- Stanja: kraljica u nekoj vrsti
- Akcije: dodaj novu kraljicu
- Test na ciljno stanje: nema napada
- **Definicija CSP??**
  - **Promenljive?** (Promenljive X1, X2, X3 ili X4 koje se odnosi na odg. vrstu
  - **Domeni?** (1,2,3,4 tj vrednost kolone)
  - **Ograničenja?**





Primer za vežbu: 4-Queens

- Stanja: 4 kraljice u 4 kolone ($4^4 = 256$ stanja)
- Akcije: pomeri kraljicu u koloni
- Test na ciljno stanje: nema napada
- Evaluacija: heuristika $h(n)$ = broj napada



Definicija CSP??
Promenljive?
Domeni?
Ograničenja?

- Može se primeniti neki od Local Search algoritama, napr Hill Climbing!!
- ZAŠTO? (kompletna dodela, uz dozvolu da se tolerišu nekonzistentne dodele; potrebna heuristika tj provera „konzistentnosti“ dodele)

PITANJA?

Dileme?

Komentari?