Изкуствен интелект - зимен семестър, 2007/2008 учебна година

Лекция 2: Търсене в пространството на състоянията

Пространство на състоянията – основни понятия и задачи

Обща постановка. Решаването на много задачи, традиционно смятани за интелектуални, може да бъде сведено до последователно преминаване от едно описание (*формулировка*) на задачата към друго, *еквивалентно* на първото или *по-просто* от него, докато се стигне до това, което се смята за решение на задачата.

Примери: задачи от областта на т. нар. интелектуални игри, аналитични преобразования на алгебрични и тригонометрични изрази, решаване на уравнения и т.н.

Основни дефиниции

Състояние: едно описание (формулировка) на задачата в процеса на нейното решаване.

Видове състояния: начално, междинни, крайни (целеви).

Оператор: начин (правило, алгоритъм), по който от дадено състояние се получава друго.

Пространство на състоянията (ПС): съвкупността от всички възможни състояния, които могат да се получат от дадено начално състояние.

Представяне на ПС: чрез ориентиран граф (*граф на състоянията, ГС*) с възли – състоянията и дъги – операторите. Когато ПС може да се представи във вид на дърво, се говори за т. нар. *дърво на състоянията (ДС).*

Действия, свързани с ПС

- ***Генериране на състояния***

- генериране на следващ наследник
- генериране на всички наследници

- ***Оценяване на състояние***

- двоични оценки (true/false)
- числови оценки в определен интервал (оценката на целта съвпада с единия край на интервала)

Основни типове задачи, свързани с ПС

- генериране на ПС
- решаване на задачи (търсене) върху генерирано ПС
- комбинирано генериране и търсене в ПС

Стратегиите за решаване на тези задачи са сходни и затова обикновено се говори само за търсене (а се подразбира и/или генериране).

Основни типове задачи за търсене в ПС

1. ***Търсене на път до (определена) цел*** – търси се път от дадено начално състояние до определено целево състояние (целевото състояние е описано явно или може да бъде разпознато). Пътят може да се търси под формата на списък от възли или списък от дъги в ГС.

Варианти: търсене на минимален (най-къс или най-икономичен) път до цел и др.

Примери: търсене на път върху географска карта, задача за търговския пътник и др.

2. ***Търсене на печеливша стратегия*** (при игри за двама играчи).

Примери: шахмат, шашки, “кръстчета и нулички” и др.

3. ***Търсене на цел при спазване на ограничителни условия*** (задачи за удовлетворяване на ограничения).

Примери: задача за осемте царици, решаване на криптограми, съставяне на разписания и др.

Задачи за удовлетворяване на ограничения (Constraint Satisfaction Problems, CSP)

Формулировка

Дадени са:

- множество от ***променливи*** $v_1, v_2, \ldots, v_n$ (със съответни области на допустимите стойности Dv_i – дискретни (крайни или изброими безкрайни) или непрекъснати)
- множество от ***ограничения*** (допустими/недопустими комбинации от стойности на променливите)

Целево състояние (състояния): множество от свързвания със стойности на променливите $\{v_1=c_1, v_2=c_2, \ldots, v_n=c_n\}$, които удовлетворяват всички ограничения.

Примерни задачи от разглеждания тип

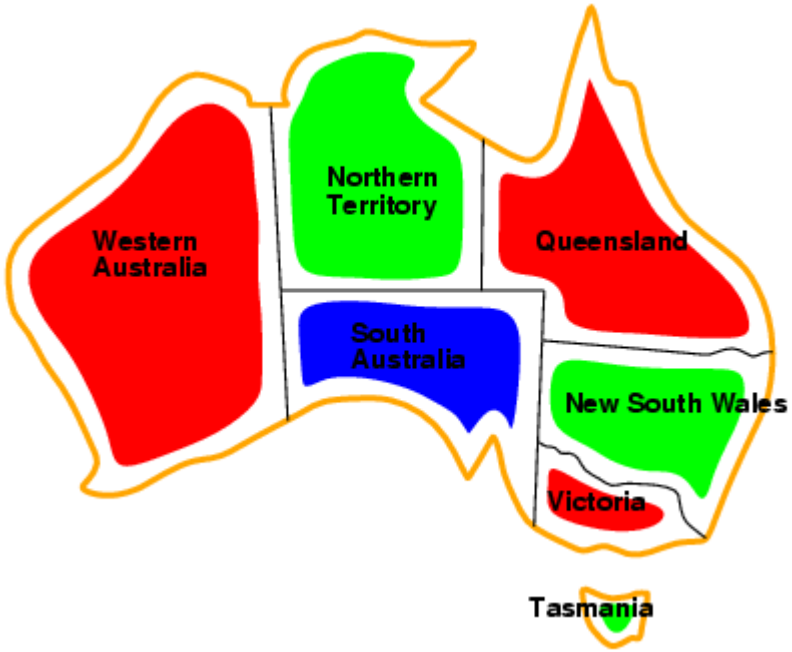
- Задача за осемте (n-те) царици, криптоаритметика, оцветяване на географска карта, пъзели, кръстословици, судоку и др.
- планиране, проектиране, съставяне на разписания и др.

Пример: оцветяване на географска карта. Целта е всяка област (държава) върху картата да бъде оцветена с подходящ цвят по такъв начин, че да няма две съседни (граничещи) области, оцветени с един и същ цвят.

- Променливи: *WA, NT, Q, NSW, V, SA, T*
- Области на допустимите стойности: *Di* = {red,green,blue}
- Ограничения: съседните области трябва да бъдат оцветени с различни цветове, например WA ≠ NT или (WA,NT) ∈ {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

Забележка. Според теоремата за четирите цвята за решаването на задачата в общия случай са достатъчни 4 цвята. В нашия случай картата на Австралия може да бъде оцветена с 3 цвята.

Примерно решение:



Типове ограничения

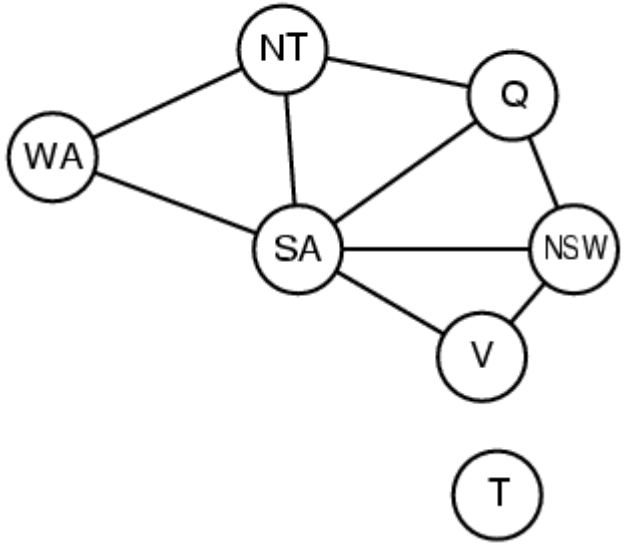
- Унарни ограничения – включват по една променлива
  Например: SA $\neq$ green
- Бинарни ограничения – включват двойки променливи
  Например: SA $\neq$ WA
- Ограничения от по-висок ред – включват по 3 или повече променливи
  Например: аритметичните ограничения в криптограмите
- Предпочитания (слаби ограничения) – определят критерии за избор между няколко решения (дефинират цената на някои свързвания на променливи)

Представяне на ограниченията

Бинарни ограничения: за представянето им се използва граф

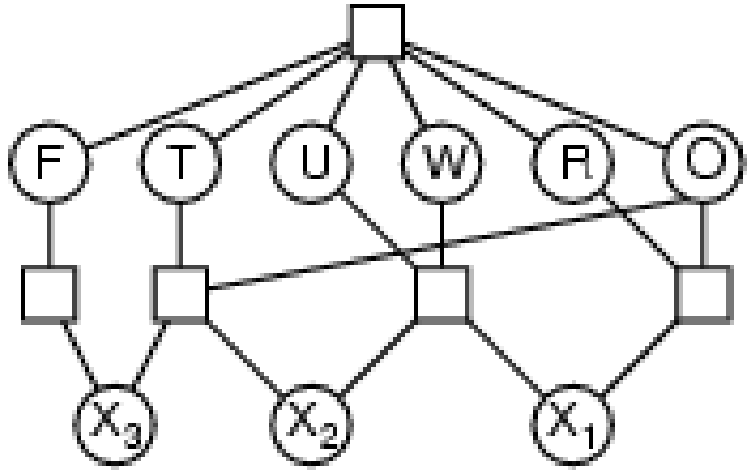
- възли – променливи (X, Y, …)
- дъги – ограничения
  На всяко ограничение p(X,Y) съответстват по две насочени дъги в графа: <X,Y> и <Y,X>
- Дъгата <X,Y> е съвместима, ако за всяка стойност на X от D_X съществува стойност на Y от D_Y, удовлетворяваща ограничението p(X,Y)
- Редуциране: ако дъгата <X,Y> не е съвместима, тогава всички стойности от D_X, за които няма съответна стойност от D_Y, могат да бъдат изтрити от D_X и това би направило дъгата <X,Y> съвместима

Примерен граф на ограниченията (constraint graph):



Пример: криптоаритметика (решаване на криптограма)

```
   T W O
+  T W O
-------
F O U R
```



- Променливи: $F, T, U, W, R, O, X_1, X_2, X_3$
- Области на допустимите стойности: {*0,1,2,3,4,5,6,7,8,9*} за променливите, които участват явно във формулиравката на криптограмата, и {*0,1*} за X_i (i=1,2,3).
- Ограничения: *Alldiff (F,T,U,W,R,O)*

  $O + O = R + 10*X_1$

  $X_1 + W + W = U + 10*X_2$

  $X_2 + T + T = O + 10*X_3$

  $X_3 = F$, $T \neq 0$, $F \neq 0$

Алгоритми за решаване на задачи за търсене на цел при спазване на ограничителни условия

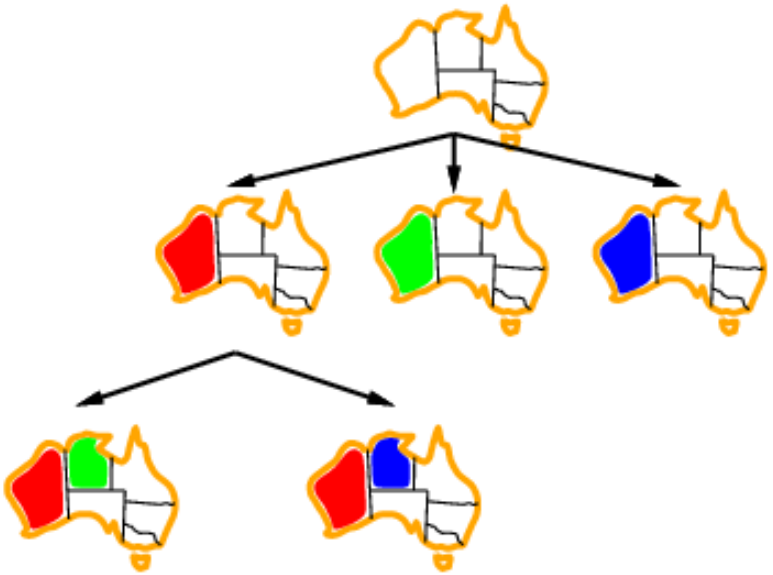
- ***генериране и тестване (generate and test)***
  - *Генерира* се множеството $D = Dv_1 \times Dv_2 \times \ldots \times Dv_n$
  - *Тества* се всяко свързване на променливите с елемент на D за това, дали удовлетворява ограниченията

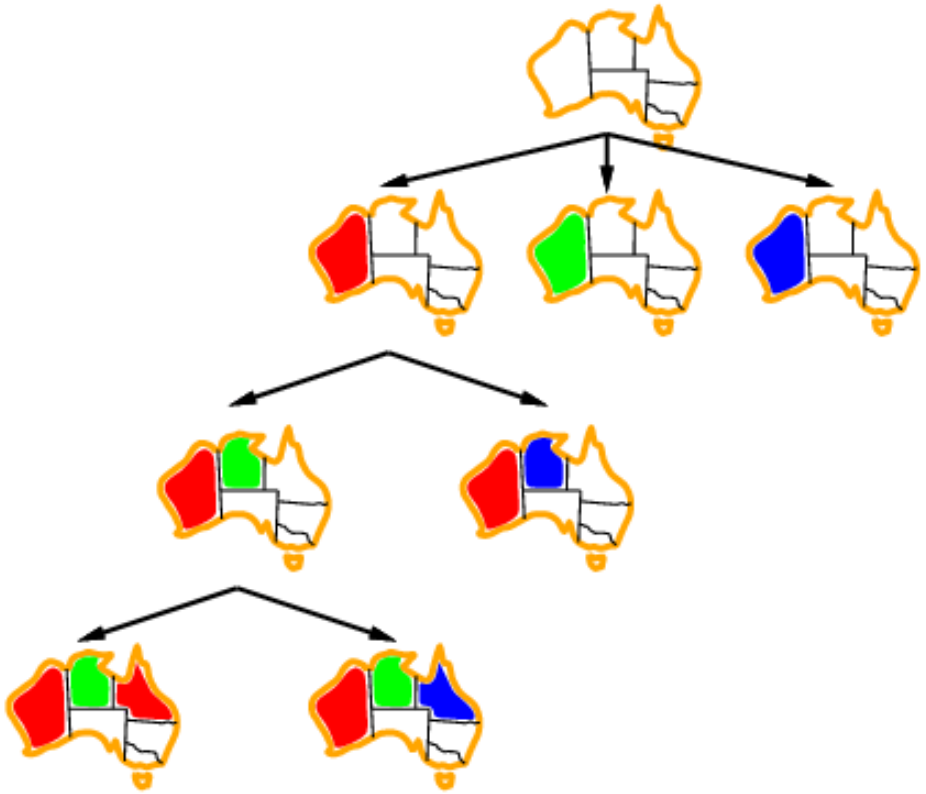
- ***търсене с възврат (backtracking)***
  - Свързване на променливите със стойности в определен ред
  - След свързването на всяка поредна променлива със стойност се извършва проверка дали свързаните до момента променливи удовлетворяват ограниченията:
    - ✓ако ограниченията се удовлетворяват, се продължава с избора на стойност за следващата несвързана променлива
    - ✓ако ограниченията не се удовлетворяват, се избира нова стойност за последната свързана променлива

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

function RECURSIVE-BACKTRACKING( assignment,csp) returns 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-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

Пример за търсене с възврат





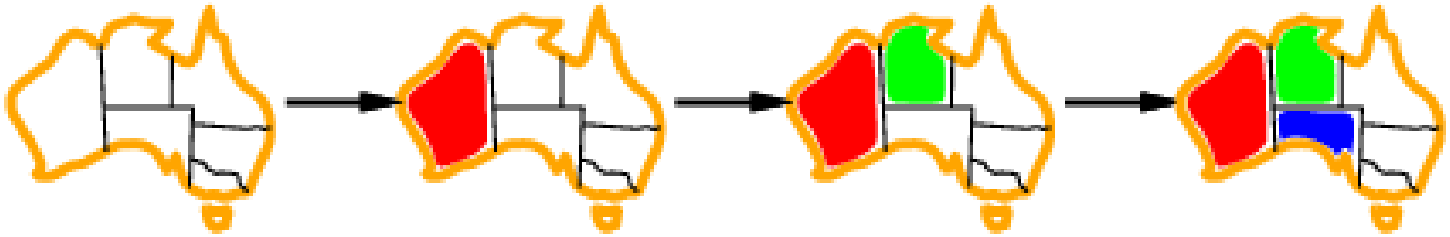
Подобряване на ефективността на търсенето с възврат

Използването на общи методи (какъвто е текущо разглежданият) поставя редица въпроси, свързани с ефективността на търсенето:

- коя променлива трябва да бъде свързана най-напред?
- в какъв ред трябва да бъдат пробвани различните допустими стойности на избраната променлива?
- може ли да се прецени предварително дали дадено свързване ще доведе до неуспех?

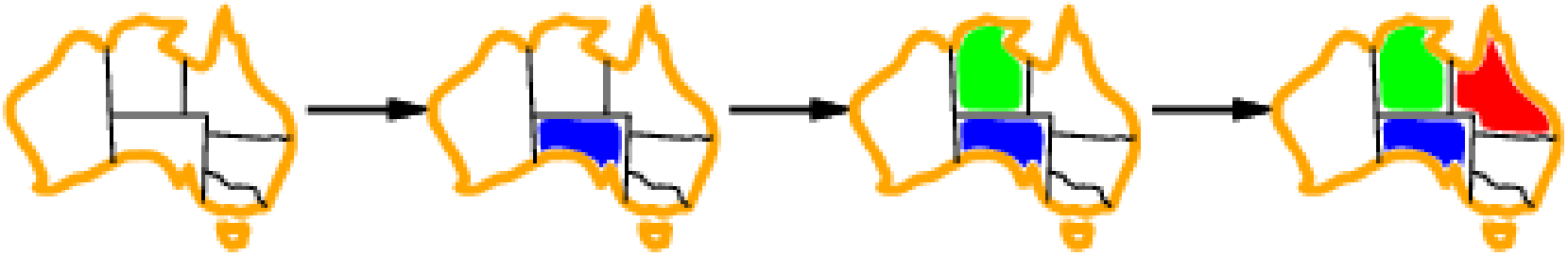
✓ ***Евристика***: избор на най-ограничената променлива

Най-напред се избира за свързване онази променлива, която има най-малък брой допустими стойности



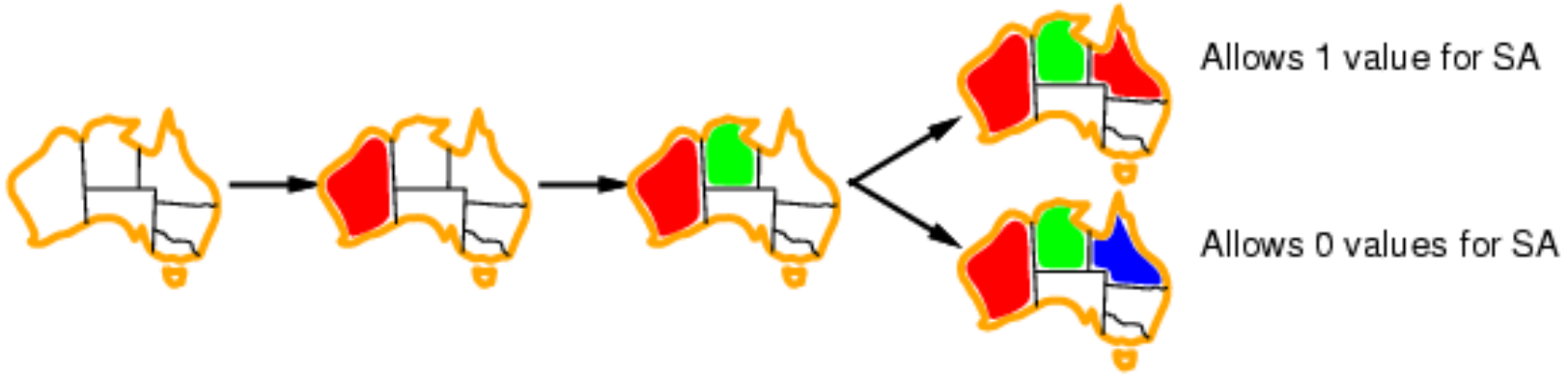
✓ ***Евристика***: избор на най-ограничаващата променлива

Най-напред се избира за свързване онази променлива, която ще наложи най-много ограничения върху оставащите несвързани променливи

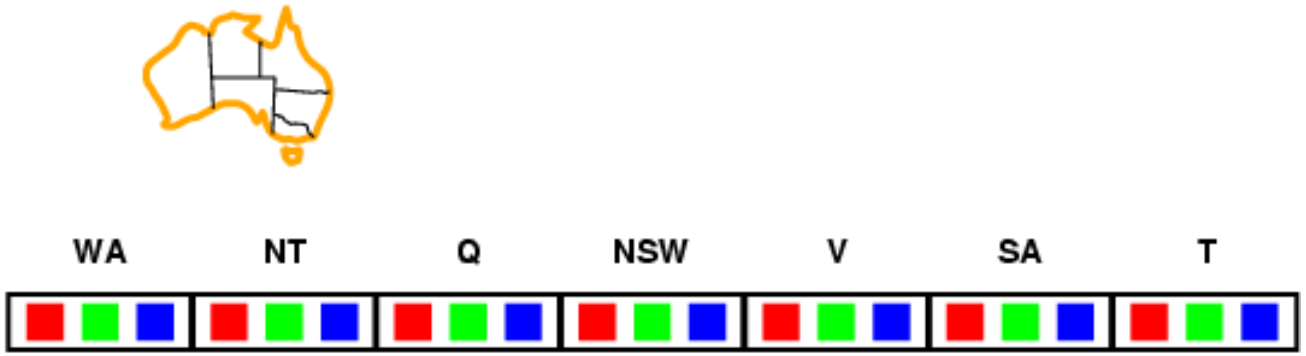


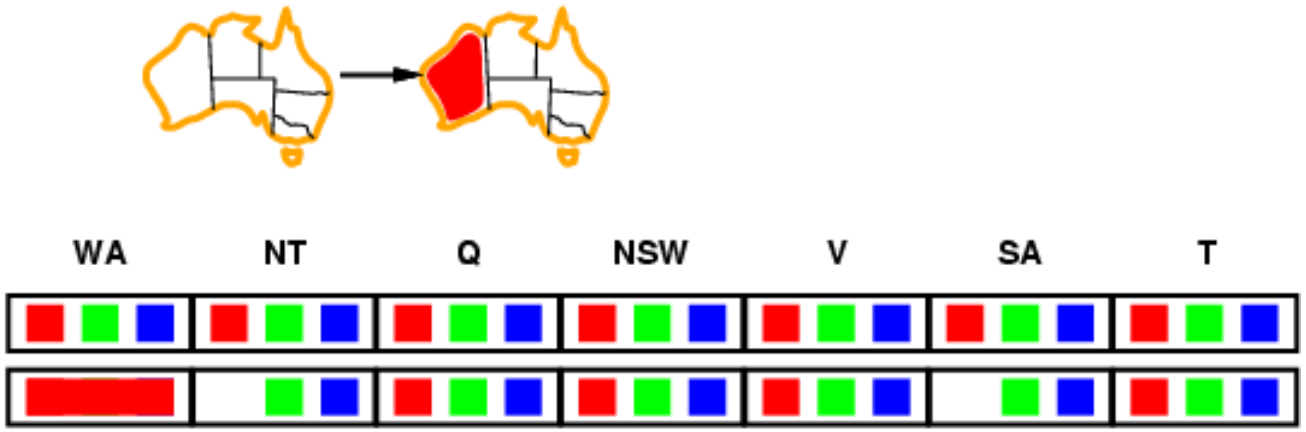
- ✓ ***Евристика***: избор на най-малко ограничаващата променлива

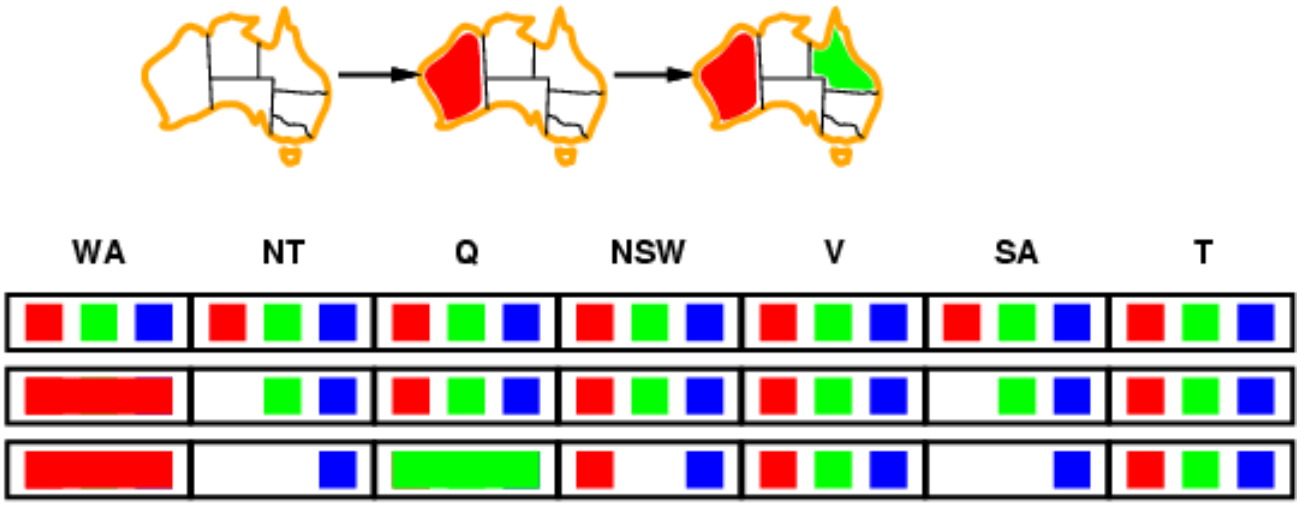
Най-напред се избира за свързване онази променлива, която ще наложи най-малко ограничения върху оставащите несвързани променливи

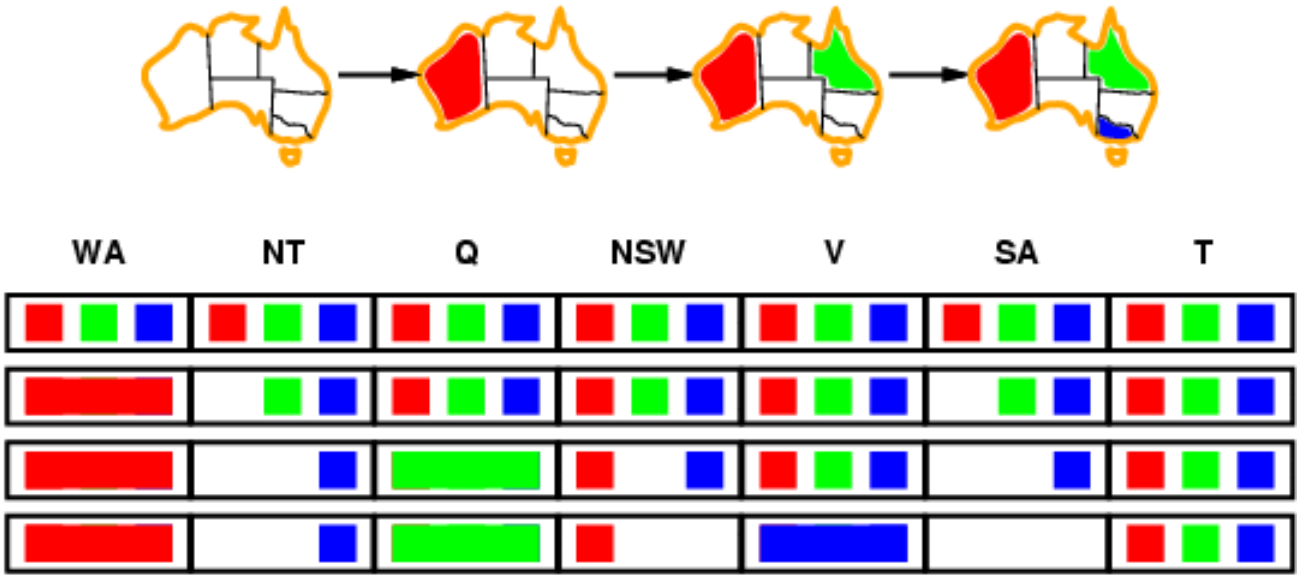


- ***разпространяване на ограниченията (forward checking)***
  - Следи се за оставащите допустими стойности за несвързаните променливи. Търсенето се прекратява, ако на дадена стъпка се окаже, че няма допустими стойности за някоя променлива.

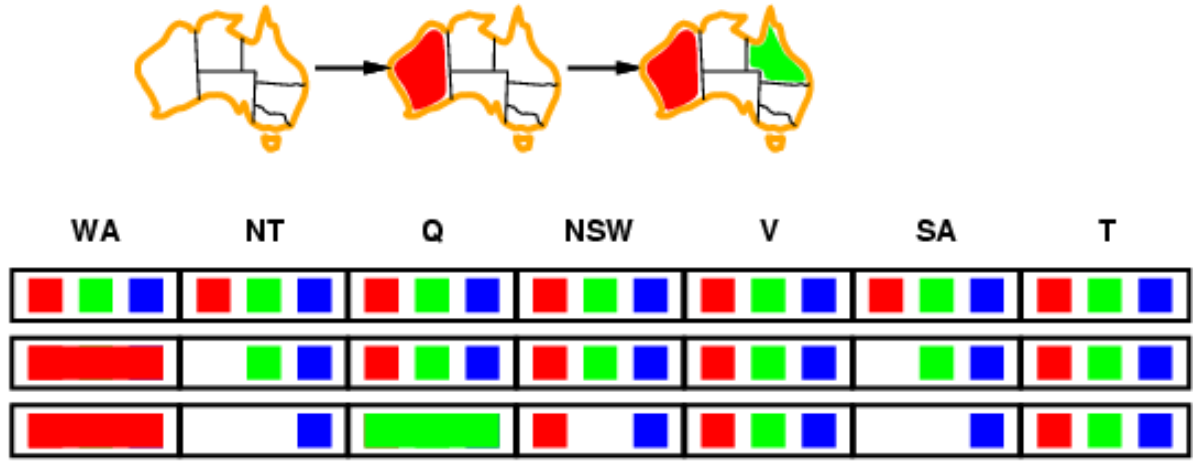



WA
NT
Q
NSW
V
SA
T


WA
NT
Q
NSW
V
SA
T


WA
NT
Q
NSW
V
SA
T

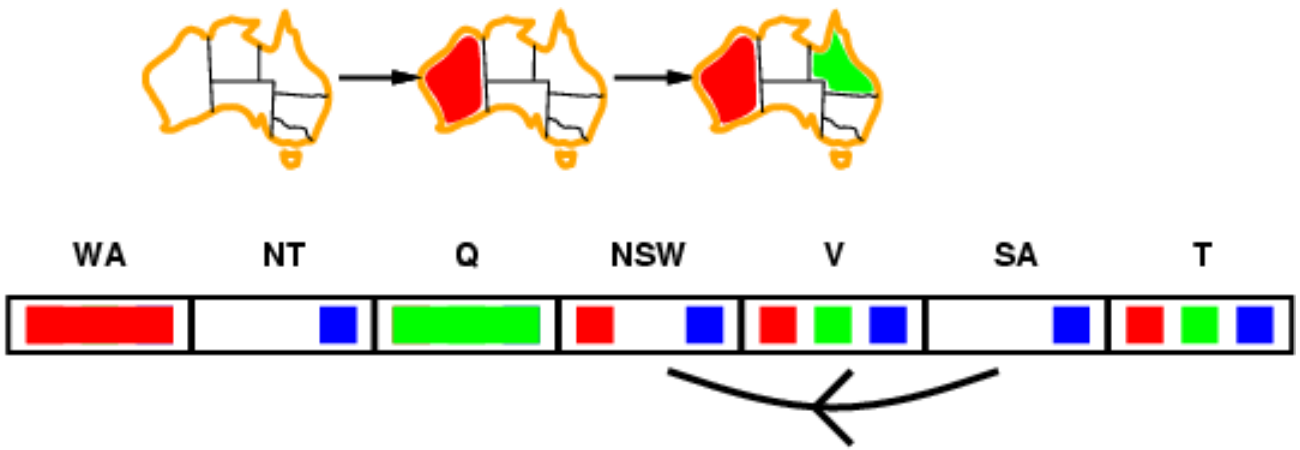
- ***разпространяване на ограниченията (constraint propagation)***
  - о *Forward checking* разпространява информация (ограничения) от свързаните към несвързаните променливи, но не открива всички случаи, които ще доведат до неуспех:

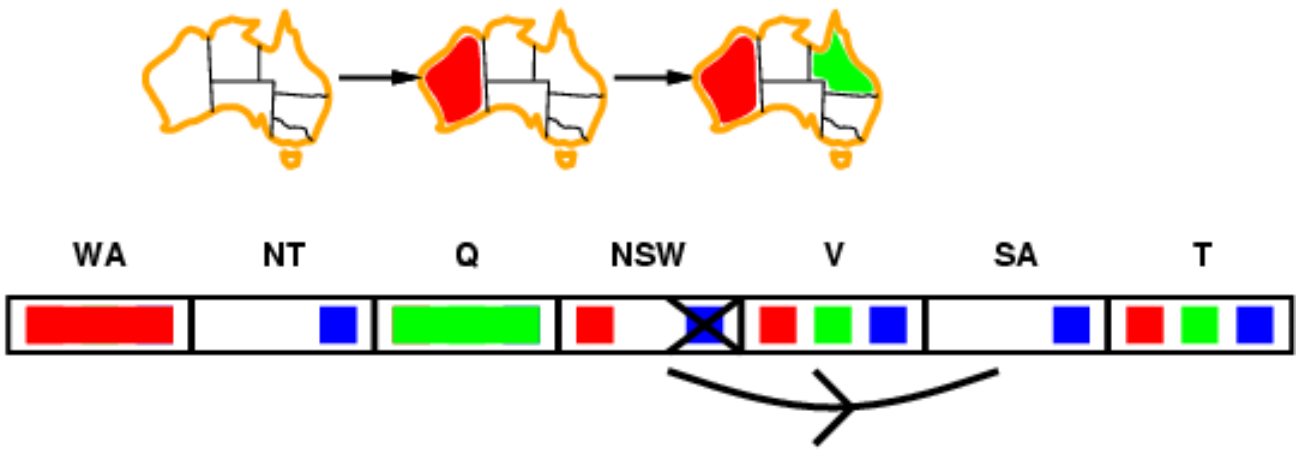


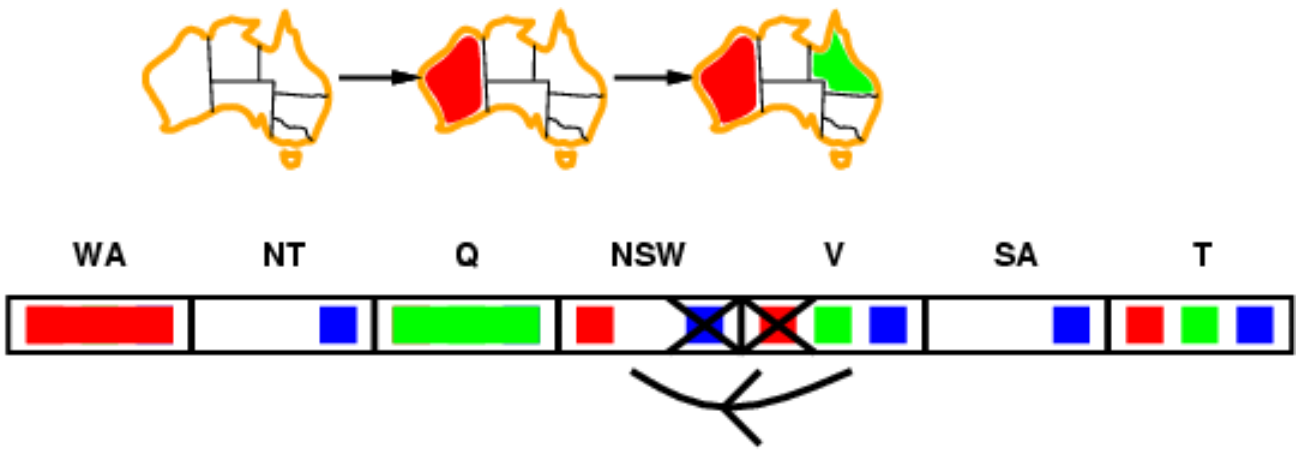
- *NT* и *SA* не могат едновременно да имат стойност *blue*.
- *Constraint propagation* е общото понятие за разпространяване на ограничения към от една променлива към други.
- *Constraint propagation* изисква *многократно засилване на ограниченията на локално равнище*.

Съвместимост на дъгите

Проверката за съвместимост и осигуряването на съвместимост на дъгите чрез *редукция* е бърз метод за разпространяване на ограниченията (при задачи с бинарни ограничения), който е по-силен от *forward checking*.


WA
NT
Q
NSW
V
SA
T


WA
NT
Q
NSW
V
SA
T


WA
NT
Q
NSW
V
SA
T

Алгоритъм AC-3 за установяване на съвместимост на дъгите

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X_1, X_2, ..., 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) ← REMOVE-FIRST(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 ← 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 ← true
   return removed
```