مسائل ارضاء محدودیت
(CSP)

فصل پنجم
سید ناصر رضوی
Email: razavi@Comp.iust.ac.ir
۱۳۸۴

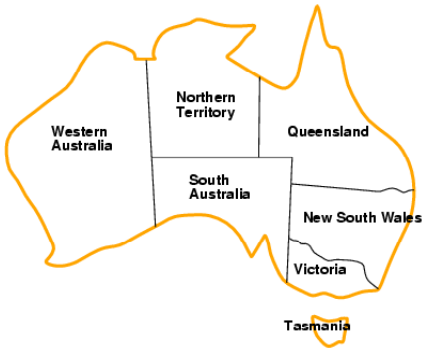
مقدمه

- چند مثال از مسائل ارضاء محدودیت
- کاربرد General Search در حل CSP
- جستجوی عقبگرد (backtracking) برای CSP
- کنترل روبه جلو (forward checking)
- استفاده از هیوریستیک ها در CSP ها
- جستجوی محلی برای مسائل ارضاء محدودیت

مسائل ارضاء محدودیت

- مسأله جستجوی استاندارد
  - حالت یک " جعبه سیاه" است – هر ساختار داده ای که از تابع حالات بعدی، تابع هیوریستیک و تست هدف پشتیبانی کند.
- CSP:
  - حالت توسط متغیرهای X_i تعریف می شود که هر یک مقادیرشان را از یک دامنه D_i اختیار می کنند.
  - تست هدف مجموعه ای از محدودیت ها می باشد که ترکیبات مجاز مقادیر برای زیرمجموعه ای از متغیرها را مشخص می کند.
- مثال ساده ای از یک زبان بازنمایی رسمی
- امکان استفاده از الگوریتم های همه منظوره که نسبت به الگوریتم های استاندارد قدرت بیشتری دارند.

مثال: رنگ آمیزی نقشه



- متغیرها: WA, NT, Q, NSW, V, SA, T
- دامنه ها: {red, green, blue}
- محدودیت ها: نواحی همسایه باید رنگ متفاوتی داشته باشند

مثلا: $WA \neq NT$ ، به بیان دیگر:

{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

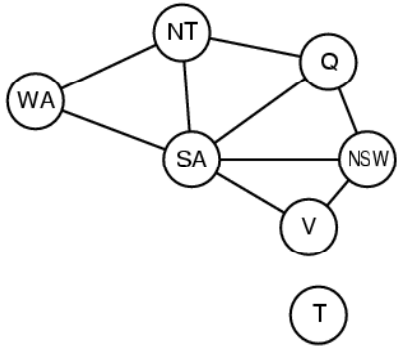
مثال: رنگ آمیزی نقشه

- راه حل ها انتساب های **کامل** و **سازگار** می باشند.
- مثال:

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

گراف محدودیت

- گراف محدودیت:
  - گره ها متغیرها را نشان می دهند.
  - یالهای گراف محدودیت ها ی بین متغیرها را نشان می دهند.



مزایای بیان مسأله به صورت *CSP*

- به دلیل نمایش استاندارد حالت ها (مجموعه متغیرهایی با مقدارشان)، می توان تابع Successor و آزمون هدف را به شکل کلی نوشت به طوریکه برای هر CSP قابل اعمال باشد.

- می توان هیوریستیک های کلی و کارایی ایجاد کرد که نیاز به تخصص اضافی در دامنه خاص مسأله نداشته باشند.

انواع مسائل CSP

- متغیرهای گسسته
  - دامنه های محدود:
    - n متغیر، اندازه هر دامنه d ← تعداد انتساب های کامل $O(d^n)$
    - مثال: CSP های بولین ، n- وزیر ، رنگ آمیزی نقشه و ...
  - دامنه های نامحدود:
    - اعداد صحیح، رشته ها و ...
    - مثال: زمانبندی کارها- متغیرها، زمان شروع/پایان هر کار هستند.
    - نیاز به زبان محدودیت دارند. مثال: $StartJob_1 + 5 \leq StartJob_3$
- متغیرهای پیوسته
  - مثال: زمان های شروع و پایان مشاهدات تلسکوپ فضایی هابل
  - محدودیت های خطی که توسط برنامه ریزی خطی قابل حل در زمان چندجمله ای می باشند.

انواع محدودیت ها

- **یکانی(Unary) :** روی یک متغیر تعریف می شود،
  - مثال: $SA \neq green$
- **دودویی (Binary):** محدودیت شامل یک زوج از متغیرها می باشد،
  - مثال: $SA \neq WA$
- **مرتبه بالاتر(Higher-order):** محدودیت شامل سه یا بیشتر متغیر است،
  - مثال: محدودیت های موجود در ستون های مسائل ریاضیات رمزی
  - در صورت متناهی بودن دامنه، می تواند به تعدادی محدودیت دودیی کاهش یابد.

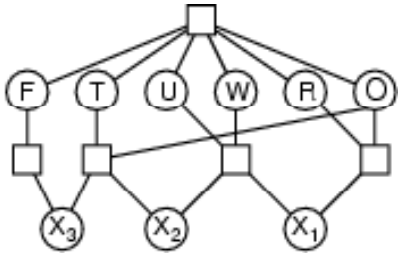
- محدودیت های مثال های بالا همگی از نوع مطلق می باشند. نقض یک محدودیت مطلق به معنای حذف یک راه حل بالقوه می باشد.

محدودیت اولویت دار (نرم):

مثلا، قرمز بر سبز ارجحیت دارد. اغلب بوسیله در نظر گرفتن هزینه برای انتساب متغیرها قابل بازنمایی می باشند. ← Constraint Optimization Problem

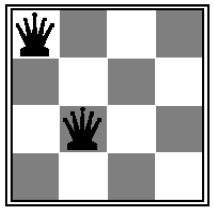
مثال: ریاضیات رمزی

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



- Variables: $F T U W R O X_1 X_2 X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: $Alldiff(F,T,U,W,R,O)$
-
  - $O + O = R + 10 \cdot X_1$
  -
  - $X_1 + W + W = U + 10 \cdot X_2$
  -
  - $X_2 + T + T = O + 10 \cdot X_3$

مثال: ۴-وزیر به عنوان *CSP*



در هر ستون یک وزیر فرض کنید.

متغیرها : Q_1, Q_2, Q_3, Q_4

دامنه ها: $D_i=\{1, 2, 3, 4\}$

محدودیت ها:

$Q_1 = 1 \quad Q_2 = 3$ (دو وزیر نمی توانند در یک سطر قرار بگیرند $Q_i \neq Q_j$

$|Q_i - Q_j| \neq |i - j|$ (یا در یک قطر)

هر محدودیت به مجموعه ای از مقادیر مجاز برای متغیرهایش ترجمه می شود، مثلا:

مقادیر (Q1, Q2) می توانند مانند زیر باشند:

(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)

مسائل ارضاء محدودیت در دنیای واقعی

- مسائل انتسابی (assignment)
  - مثلا، چه کسی چه کلاسی را درس می دهد.
- مسائل تعیین جدول زمانبندی (Timetabling)
  - مثلا، کدام کلاس، کجا و کی ارائه می شود؟
- زمانبندی حمل و نقل (transportation)
- زمانبندی کارخانه (factory scheduling)

- توجه: بسیاری از مسائل در دنیای واقعی شامل متغیرهایی با مقادیر حقیقی می باشند.

فرموله سازی جستجوی استاندارد(افزایشی)

- حالات توسط مقادیری که تا کنون انتساب یافته اند، تعریف می شوند.
- حالت اولیه: تمام متغیرها بدون مقدار، یعنی انتساب تهی {}
- عملگرها: انتساب مقدار به یک متغیر بدون مقدار به طوری که با انتساب فعلی درگیری ایجاد نکند.← در صورت عدم وجود انتساب های مجاز شکست می خورد.
- تست هدف: انتساب فعلی کامل باشد.
- هزینه مسیر: هزینه یکسان برای تمام مراحل

۱) برای تمام CSP ها یکسان است!
۲) هر پاسخ در عمق n با n متغیر ظاهر می شود.← استفاده از جستجوی عمقی
۳) در عمق l، فاکتور انشعاب (b) برابر است با $d*(n-l)$.
بنابراین تعداد برگهای درخت جستجو برابر است با $n! * d^n$!!!!
$(nd) * [(n-1)d] * [(n-2)d] * \ldots d = n! d^n$
مثلا در ۸-وزیر: $8!8^8$

پیاده سازی

datatype CSP-STATE
components: UNASSIGNED, a list of variables not yet assigned
ASSIGNED, a list of variable that have value

datatype CSP-VAR
components: NAME, for i/o purpose
DOMAIN, a list of possible values
VALUE, current value (if any)

- محدودیت ها را می توان به دو روش نمایش داد:
  - **صریح:** به عنوان مجموعه ای از مقادیر مجاز، یا
  - **ضمنی:** بوسیله تابعی که ارضاء محدودیت ها را بررسی می کند.

پیچیدگی این رهیافت

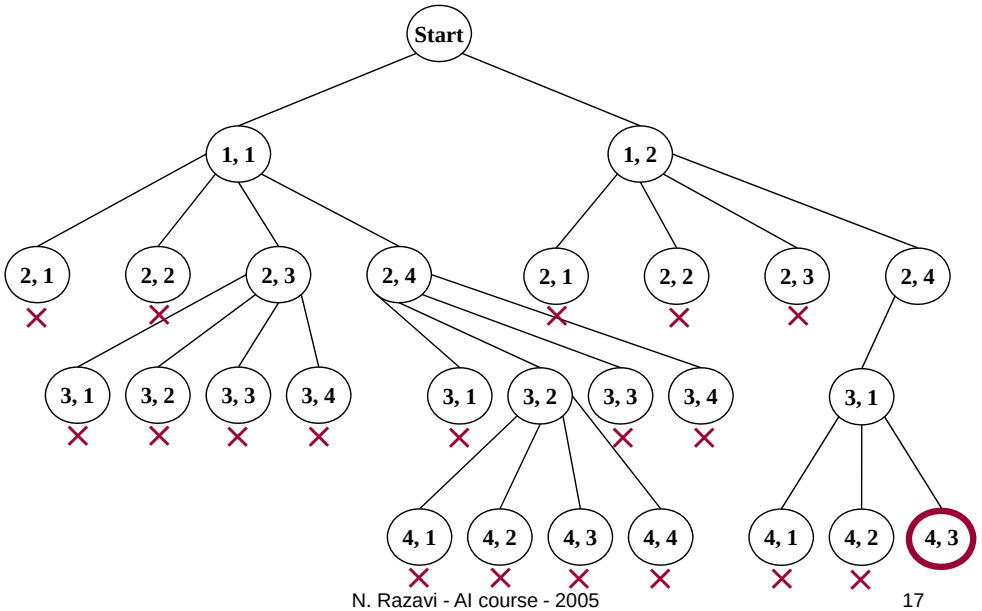
- حداکثر عمق فضا (m) = ؟؟ n (تعداد متغیرها)
- عمق حالت پاسخ (d) = ؟؟ n (تمام متغیرها مقدار دارند)
- الگوریتم جستجو؟؟ جستجوی اول-عمق
- فاکتور انشعاب b ؟؟ $\Sigma_i|D_i|$ (در بالای درخت)

- این ها می توانند با توجه به نکات زیر بهبود یابند:
۱) ترتیب انتساب مقادیر به متغیرها اهمیت ندارد، پس بسیاری از مسیرها معادل یکدیگر می باشند.(در ۸-وزیر اندازه فضای حالت از $8!8^8$ به 8^8 کاهش می یابد)
۲) انتساب های بعدی نمی توانند یک محدودیت نقض شده را تصحیح کنند.(مثلا اگر در ۸-وزیر دو وزیر اول در یک ردیف قرار بگیرند، جستجوی عمقی 8^6 حالت باقیمانده را امتحان می کند تا بفهمد پاسخی وجود ندارد)← جستجوی عقب گرد.

Backtracking

- انتساب متغیرها جابه جایی پذیر است مثلا $[Q_1=1,Q_2=3]$ برابر است با $[Q_2=3,Q_1=1]$
- در هر گره تنها باید انتساب های یک متغیر در نظر گرفته شود؛پس، b=d و تعداد برگها=d^n
- جستجوی اول-عمق با انتساب های یک متغیر جداگانه در CSP ها، backtracking نام دارد.
- قرار دادن یک آزمون قبل از بسط گره ها که بررسی می کند آیا تاکنون محدودیتی نقض شده یا خیر. اگر محدودیتی نقض شده باشد، دیگر این گره بسط داده نمی شود و جستجو به عقب باز می گردد.
- می تواند n-وزیر را تا حدود n=25 حل کند.

مثال: جستجوی عقبگرد برای ۴-وزیر



جستجوی عقبگرد

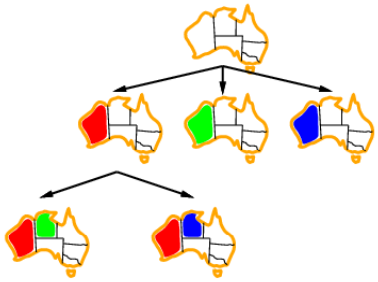
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
   return RECURSIVE-BACKTRACKING({}, csp)

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
```

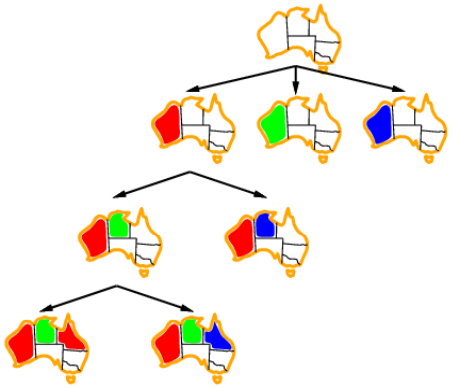

جستجوی عقبگرد

جستجوی عقبگرد

جستجوی عقبگرد



جستجوی عقبگرد



بهبود کارآیی *backtracking*

- افزایش سرعت توسط روش های جستجوی همه منظوره:

۱- در مرحله بعد باید به کدام متغیر مقدار داده شود؟

۲- مقادیر آن باید به چه ترتیبی امتحان شوند؟

۳- آیا می توان شکست های حتمی را زودتر تشخیص داد؟

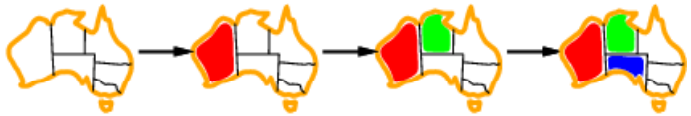
فرض کنید در عقبگرد در مسأله ۸-وزیر، ۶ وزیر اول به گونه ای قرار گرفته اند که قرار دادن هشتمین وزیر را غیر ممکن می سازند. عقب گرد تمام ممکانهای ممکن برای وزیر هفتم را چک می کند، اگرچه مسأله غیر قابل حل می باشد.

۴- آیا می توان از ساختار مسأله بهره گرفت؟

متغیر با بیشترین محدودیت

- متغیر با بیشترین محدودیت :

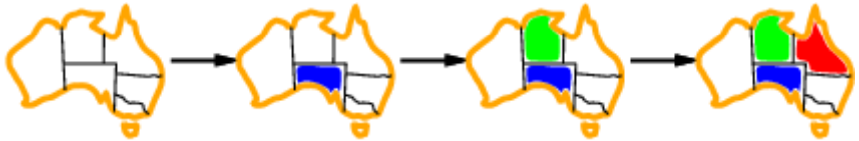
– متغیری را انتخاب کن که کمترین مقادیر معتبر را دارد.



– هیوریستیک **کمترین مقادیر باقیمانده** (MRV)

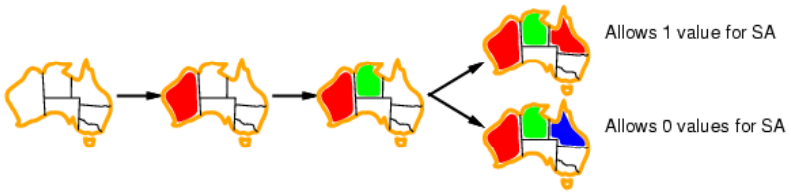
متغیر با بیشترین محدودیت

- حل درگیری (tie) میان متغیرهایی که بیشترین محدودیت را دارند
- متغیر با بیشترین محدودیت: هیوریستیک درجه
  - متغیر با بیشترین محدودیت را روی مقادیر باقیمانده انتخاب کن



مقدار با کمترین محدودیت (Least constraining value)

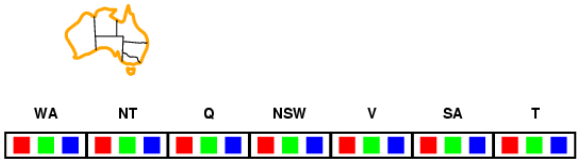
- برای یک متغیر، مقداری را که کمترین محدودیت را ایجاد می کند انتخاب کن.
  - مقداری که کمترین مقادیر را از متغیرهای باقیمانده حذف می کند



- با ترکیب این هیوریستیک ها مسأله ۱۰۰۰-وزیر قابل حل می شود.

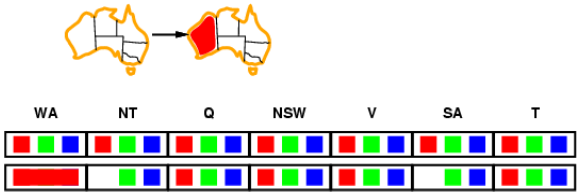
بررسی رو به جلو (Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



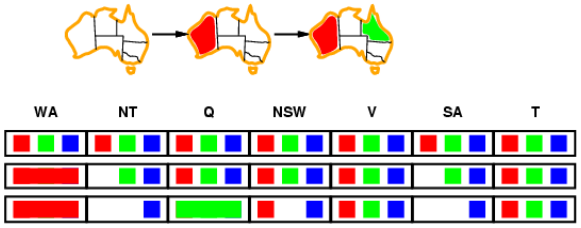
بررسی رو به جلو (Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



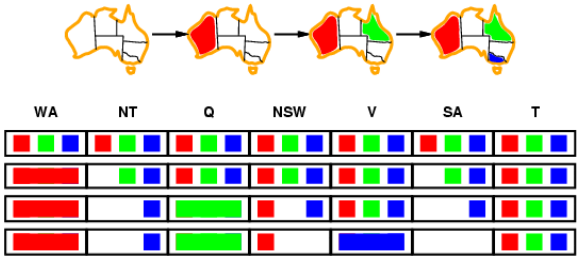
بررسی رو به جلو
(Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



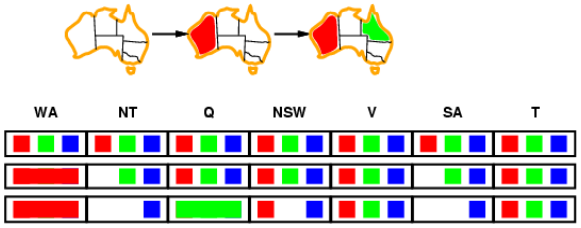
بررسی رو به جلو
(Forward checking)

- **ایده:**
  - مراقب مقادیر مجاز باقیمانده برای متغیرهای بدون مقدار باش.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ای ندارد، به جستجو پایان بده.



انتشار محدودیت

- بررسی رو به جلو محدودیت ها را از متغیرهای انتساب یافته به متغیر های انتساب نیافته منتشر می کند، اما تمام شکست ها را نمی تواند در زود ترین زمان ممکن تشخیص دهد.

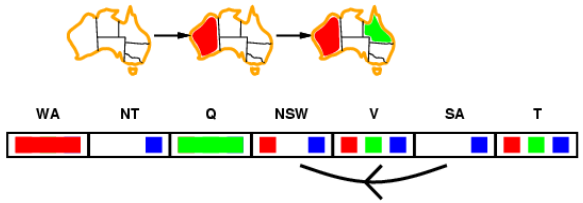


- NT و SA هر دو نمی توانند آبی باشند!
- انتشار محدودیت به طور مکرر بر محدودیت ها به طور محلی تأکید دارد.

سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- $X \rightarrow Y$ سازگار است اگر و فقط اگر

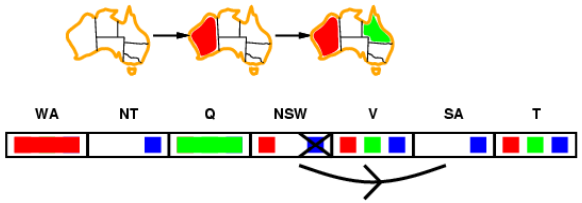
بازاء هر مقدار x برخی مقادیر مجاز y موجود باشند.



سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

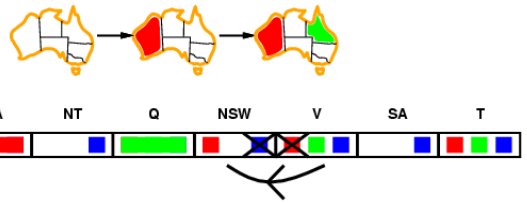
بازاء هر مقدار X برخی مقادیر مجاز Y موجود باشند.



سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

بازاء هر مقدار X برخی مقادیر مجاز Y موجود باشند.

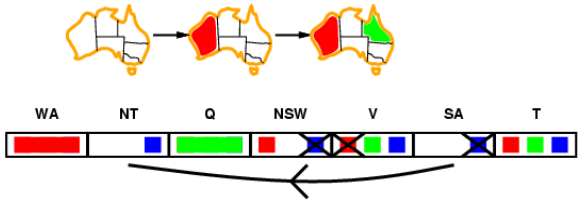


اگر X مقداری را از دست بدهد، همسایه های X نیاز به بررسی مجدد دارند.

سازگاری کمان
(Arc consistency)

- ساده ترین شکل انتشار، هر کمان را سازگار می کند.
- X→Y سازگار است اگر و فقط اگر

بازاء هر مقدار X برخی مقادیر مجاز Y موجود باشند.



سازگاری کمان شکست ها را زودتر از FC تشخیص می دهد.

و می تواند به عنوان پیش پردازنده و یا بعد از هر انتساب اجرا شود.

الگوریتم سازگاری کمان AC-3

```
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
```

- پیچیدگی زمانی: $O(n^2d^3)$

جستجوی محلی برای CSP

- تپه نوردی و SA با حالات کامل کار می کنند؛ یعنی تمام متغیرها باید دارای مقدار باشند.

- برای اعمال آنها بر مسائل CSP:
  - داشتن حالاتی با محدودیتهای نقض شده باید مجاز باشد.
  - عملگرها باید بتوانند به متغیرها مقادیر جدید بدهند.

- انتخاب متغیر: به صورت تصادفی یک متغیر درگیر را انتخاب کن
- هیوریستیک *min-conflicts*:
  - مقداری را انتخاب کن که کمترین تعداد محدودیت ها را نقض می کند،

یعنی، تپه نوردی با هیوریستیک " تعداد کل محدودیت های نقض شده"

الگوریتم *Min-Conflicts*

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
    if current is a solution for csp then return current
    var ← a randomly chosen, conflicted variable from VARIABLES[csp]
    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set var = value in current
  return failure
```

Figure 5.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

مثال : ۸-وزیر

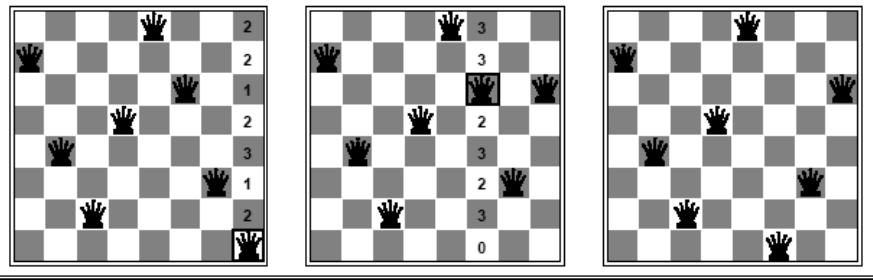


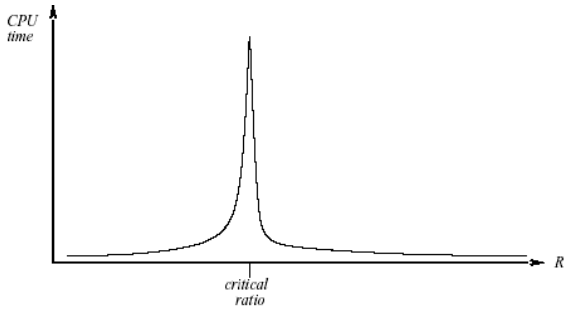
Figure 5.9 A two-step solution for an 8-queens problem using min-conflicts. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflict square, breaking ties randomly.

کارآیی *min-conflicts*

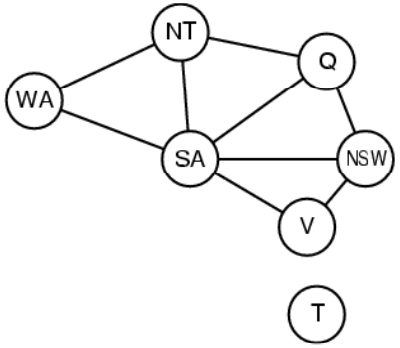
حل n- وزیر با n دلخواه (مثلا 10,000,000) در یک زمان تقریبا ثابت با شروع از یک وضعیت تصادفی با احتمال بالا(میانگین ۵۰ مرحله)

همین موضوع به نظر می رسد برای هر CSP با شروع تصادفی درست باشد، به غیر از یک ناحیه باریک از نسبت R.

R = (num. of constraints)/(num. of variables)



ساختار مسأله



- Tasmania و جزیره اصلی زیرمسایل مستقل هستند.
- و با توجه به مولفه های همبند در گراف محدودیت قابل شناسایی هستند.

ساختار مسأله (ادامه)

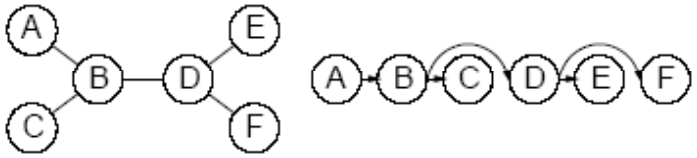
- فرض کنید هر زیرمسأله شامل c متغیر از کل n متغیر باشد
- هزینه راه حل در بدترین حالت خطی(برحسب n) و برابر $n/c \cdot d^c$ می باشد

- مثال: $n = 80,\ d = 2,\ c = 20$

- 2^{80} = 4 billion years at 10 million nodes/sec
- 4.2^{20} = 0.4 seconds at 10 million nodes/sec

الگوریتم برای *CSP* های دارای ساختار درختی

۱. یک متغیر را به عنوان ریشه انتخاب کن و سپس متغیرها را از ریشه تا برگها به گونه ای مرتب کن که والد هر گره قبل از آن گره قرار بگیرد.



۲. بازاء j از n تا 2 عمل زیر را انجام بده

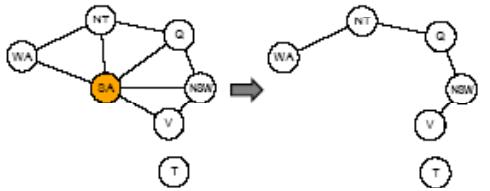
REMOVEINCONSISTENT($Parent(X_j)$, X_j)

۳. برای j از یک تا n ، مقدار X_j را به طور سازگار با $Parent(X_j)$ تعیین کن

پیچیدگی زمانی: $O(n \cdot d^2)$

CSP های دارای ساختار شبه درختی

- شرطی سازی: به یک متغیر مقدار بده؛ دامنه همسایه های آن را هرس کن



- شرط برشی: مجموعه ای از متغیرها (در تمام حالات ممکن) را مقدار دهی کن به طوریکه گراف گراف محدودیت باقیمانده یک درخت شود.

اندازه برش = c ← زمان اجرا $O(d^c \cdot (n - c)\ d^2)$

رهیافت دوم: تجزیه درختی

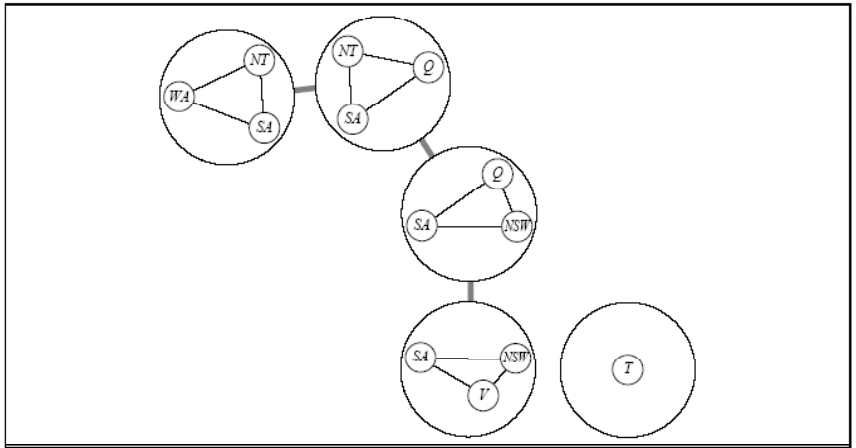


Figure 5.12 A tree decomposition of the constraint graph in Figure 5.11(a).

رهیافت دوم: تجزیه درختی

- شرایط تجزیه درختی:
  - هر متغیر در مسأله اصلی باید حداقل در یک زیرمسأله ظاهر شود
  - اگر دو متغیر به وسیله محدودیتی در مسأله اصلی متصل شده باشند، آنگاه آن دو متغیربا هم(به همراه محدودیت) باید حداقل در یک زیر مسأله ظاهر شوند.
  - اگر متغیری در درخت در دو زیر مسأله ظاهر شده باشد، آنگاه باید در هر زیرمسأله در طول مسیری که زیرمسایل را متصل می کند، ظاهر شود.

رهیافت دوم: تجزیه درختی

- حل تجزیه درختی:
  - هر زیر مسأله را به صورت یک mega-variable در نظر می گیریم که دامنه آن مجموعه تمام راه حل های ممکن آن زیر مسأله می باشد.
  - سپس محدودیت های میان زیر مسایل را با استفاده از الگوریتم کارای درخت حل می کنیم. مثلا اگر پاسخ اولین زیر مسأله انتساب زیر باشد:

$$\{WA = red, SA = blue, NT = green\}$$

آنگاه تنها پاسخ سازگار برای زیر مسأله بعدی انتساب زیر می باشد:

$$\{SA = blue, NT = green, Q = red\}$$

پیچیدگی زمانی: $O(n \cdot d^{w+1})$

خلاصه

- مسائل CSP نوع خاصی از مسائل می باشند:
  - حالات توسط مقادیر مجموعه ای از متغیرها تعریف می شوند.
  - تست هدف توسط محدودیت ها روی مقادیر متغیرها تعریف می شود.
- جستجوی عقبگرد = جستجوی اول-عمق که در هر گره یک متغیر مقدار گرفته باشد.
- هیوریستیک های ترتیب متغیرها و انتخاب مقادیر بسیار کمک کننده می باشند.
- بررسی رو به جلو از انتساب های منجر به شکست جلوگیری می کند.
- انتشار محدودیت کارهای بیشتری برای محدود کردن مقادیر و تشخیص ناسازگاری ها انجام می دهد.
- در عمل معمولا جستجوی محلی min-conflicts مؤثر می باشد.