دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

مسائل ارضای محدودیت

«هوش مصنوعی: رهیافتی نوین»، فصل ۶

مدرس: آرش عبدی هجراندوست

نیم‌سال دوم ۱۴۰۱-۱۴۰۲

رئوس مطالب

- تعریف مسائل ارضای محدودیت (Constraint Satisfaction Problems)
- انتشار محدودیت: استنتاج در CSPها
- جستجوی عقب‌گرد برای CSPها
- جستجوی محلی در CSPها
- ساختار مسائل

تعریف مسائل ارضای محدودیت

تعریف مسائل ارضای محدودیت

- یک مسئله‌ی ارضای محدودیت شامل سه جزء زیر است:
  - مجموعه‌ای از **متغیرها** $X=\{X_1,X_2,\ldots, X_n\}$
  - مجموعه‌ای از **دامنه‌ها** $D=\{D_1,D_2,\ldots, D_n\}$
    - هر دامنه‌ی D_i شامل مجموعه‌ای از مقادیر مجاز $\{v_1,\ldots,v_k\}$ برای متغیر X_i است.
  - مجموعه‌ای از **محدودیت‌ها** $C=\{C_1,C_2,\ldots, C_m\}$
    - هر محدودیت C_i یک زوج <scope, rel> است که scope یک چندتایی است که متغیرهایی که در محدودیت شرکت می‌کنند را مشخص میکند و rel رابطه‌ای است که مقادیری که متغیرها می‌توانند بگیرند را تعریف می‌کند.
    - این رابطه میتواند به صورت صریح لیستی از تمام مجموعه مقادیر scope را بیان کند یا یک رابطه انتزاعی بین متغیرها تعریف کند.
      - برای مثال اگر بخواهیم رابطه‌ی نابرابری بین دو متغیر که هر دو از دامنه‌ی $\{A, B\}$ هستند را نشان دهیم یکی از دو روش زیر را می‌توانیم برای نشان دادن محدودیت مربوطه به کار ببریم: ($\{A, B\}$ مجموعه ای شامل دو عضو است)

$$<(X_1, X_2), X_1 \neq X_2>$$

$$<(X_1, X_2), [(A, B), (B, A)]>$$

تعریف مسائل ارضای محدودیت ...

- برای حل یک مسئله ارضای محدودیت می‌بایست فضای حالت و راه‌حل تعریف شود.
- فضای حالت: هر حالت یک انتساب مقادیر به برخی یا همه متغیرها است
  - $\{X_i=v_i, X_j=v_j,\ldots\}$ (انتساب جزئی یا کامل)
- راه‌حل (هدف): یک انتساب **کامل** و **سازگار** از مقادیر به متغیرها
  - سازگار: انتسابی که هیچ یک از محدودیت‌ها را نقض نمی‌کند.
  - کامل: به هر یک از متغیرها مقداری نسبت داده شده باشد.

مثال: رنگ آمیزی نقشه

- تعریف مسئله رنگ‌آمیزی نقشه
  - متغیرها: هر یک از ناحیه‌ها

$\{WA, NT, Q, NSW, V, SA, T\}$

  - دامنه‌ها: سه رنگ قرمز، سبز و آبی

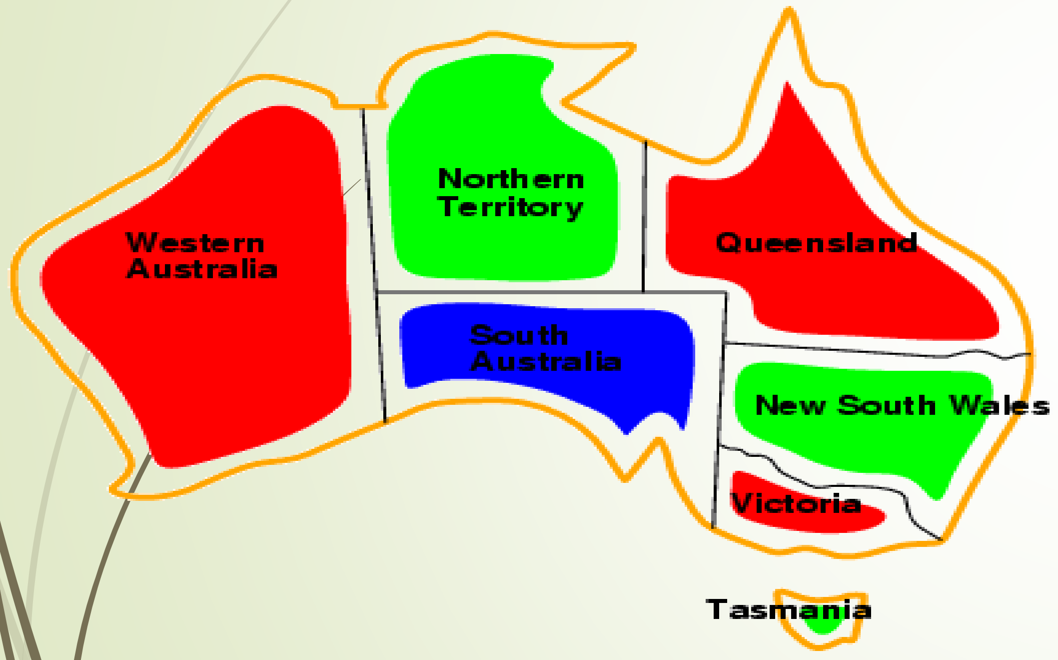
$D_i = \{red, green, blue\}$

  - محدودیت‌ها: نواحی همسایه باید رنگ متفاوتی داشته باشند.

$\{WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, NSW \neq V, NSW \neq Q\}$

مثال: رنگ آمیزی نقشه

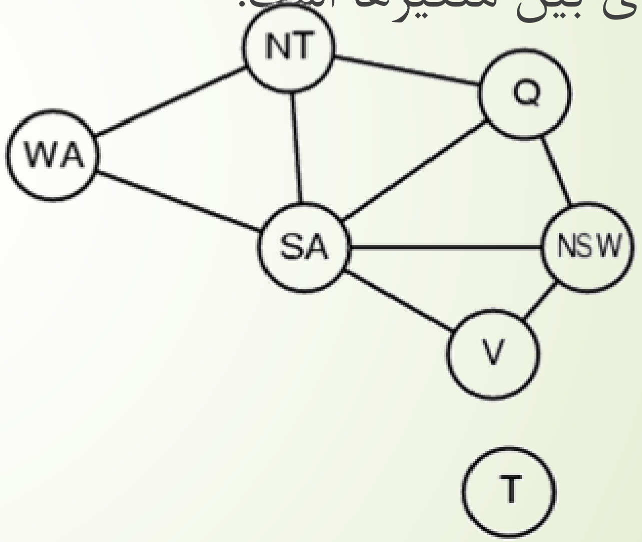
- راه‌حل مسئله‌ی رنگ‌آمیزی
  - یک انتساب کامل و سازگار برای مثال



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

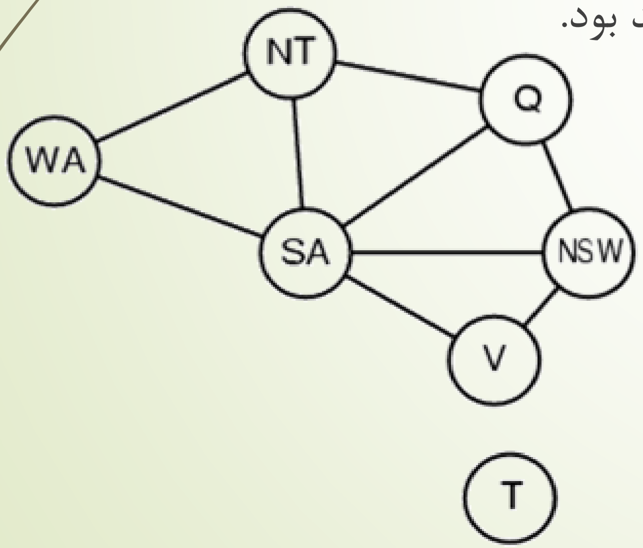
گراف محدودیت

- گراف محدودیت: گرافی که گره‌ها در آن نشان‌دهنده‌ی متغیرها و یال‌ها نشان‌دهنده‌ی محدودیت‌های بین متغیرها است.



دلایل استفاده از CSP ها

- CSP ها یک نمایش طبیعی برای بازه گسترده‌ای از مسائل ارائه می‌کنند.
  - اغلب حل مسئله با استفاده از CSP ها آسان‌تر است.
- حل‌کننده‌های CSP می‌توانند به سرعت بخش بزرگی از فضای حالت را حذف کنند.
  - برای مثال در رنگ‌آمیزی نقشه با انتخاب {SA=blue} می‌توان نتیجه گرفت که هیچ یک از ۵ متغیر همسایه نمی‌تواند رنگ آبی داشته باشد.
  - بدون استفاده از انتشار محدودیت تعداد انتساب ۵ متغیر دیگر ۲۴۳=۳^{۵} خواهد بود.
  - با انتشار محدودیت این تعداد برابر با ۳۲=۲^{۵} می‌شود. (٪۸۷ کاهش)



- بسیاری از مسائلی که برای جستجوی فضای حالت، رام‌نشدنی (intractable) هستند می‌توانند به سرعت هنگامی که به یک CSP فرموله می‌شوند حل گردند.
  - هنگامی که متوجه شدیم یک انتساب جزئی منجر به یک راه‌حل نمی‌شود از ادامه دادن و مقداردهی بیشتر این انتساب خودداری می‌کنیم.

مثال: برنامه‌ریزی وظایف

- یکی از مسائل رایج کارخانه‌ها، برنامه‌ریزی وظایف روزانه‌ی آن‌ها است.
- برای مثال: مسئله‌ی برنامه‌ریزی اسمبل کردن یک ماشین
  - متغیرها: کل کار از وظایفی تشکیل شده است و می‌توان هر وظیفه را به‌عنوان یک متغیر مدل نمود.
  - دامنه‌ها: مقدار هر متغیر زمان شروع وظیفه است. (یک عدد برحسب دقیقه)
  - محدودیت‌ها: یک وظیفه باید قبل از وظیفه دیگر انجام شود (برای مثال یک چرخ باید قبل از قرار گرفتن قالپاق نصب شود)
    - وظایف زیادی می‌توانند به صورت هم‌زمان اجرا شوند.
    - یک وظیفه یک مقدار زمان مشخص برای تکمیل شدن نیاز دارد.

مثال: برنامه‌ریزی وظایف ...

- فرض کنید این مسئله دارای ۱۵ وظیفه زیر است
  - نصب محورهای جلو و عقب
  - اضافه کردن هر چهار چرخ (جلو، عقب، راست و چپ)
  - جای دادن مهره‌ها برای هر چهار چرخ
  - اضافه کردن قالپاق
  - بازرسی مونتاژ نهایی

$$X = \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}$$

مثال: برنامه‌ریزی وظایف ...

- هرگاه وظیفه T_1 باید قبل از وظیفه‌ی T_2 انجام شود و T_1 برای تکمیل شدن به d_1 زمان احتیاج داشته باشد، یک محدودیت محاسباتی به شکل $T_1+d_1 \leq T_2$ خواهیم داشت.

$$
\begin{array}{ll}
Axle_F + 10 \leq Wheel_{RF}; & Axle_F + 10 \leq Wheel_{LF}; \\
Axle_B + 10 \leq Wheel_{RB}; & Axle_B + 10 \leq Wheel_{LB} \\
Wheel_{RF} + 1 \leq Nuts_{RF}; & Nuts_{RF} + 2 \leq Cap_{RF}; \\
Wheel_{LF} + 1 \leq Nuts_{LF}; & Nuts_{LF} + 2 \leq Cap_{LF}; \\
Wheel_{RB} + 1 \leq Nuts_{RB}; & Nuts_{RB} + 2 \leq Cap_{RB}; \\
Wheel_{LB} + 1 \leq Nuts_{LB}; & Nuts_{LB} + 2 \leq Cap_{LB} .
\end{array}
$$

- فرض می‌کنیم چهار کارگر برای نصب چرخ‌ها داریم، اما تنها یک ابزار برای نصب محور دارند. (ترکیب محدودیت‌های محاسباتی و منطقی)

$$(Axle_F + 10 \leq Axle_B) \quad \textbf{or} \quad (Axle_B + 10 \leq Axle_F)$$

مثال: برنامه‌ریزی وظایف ...

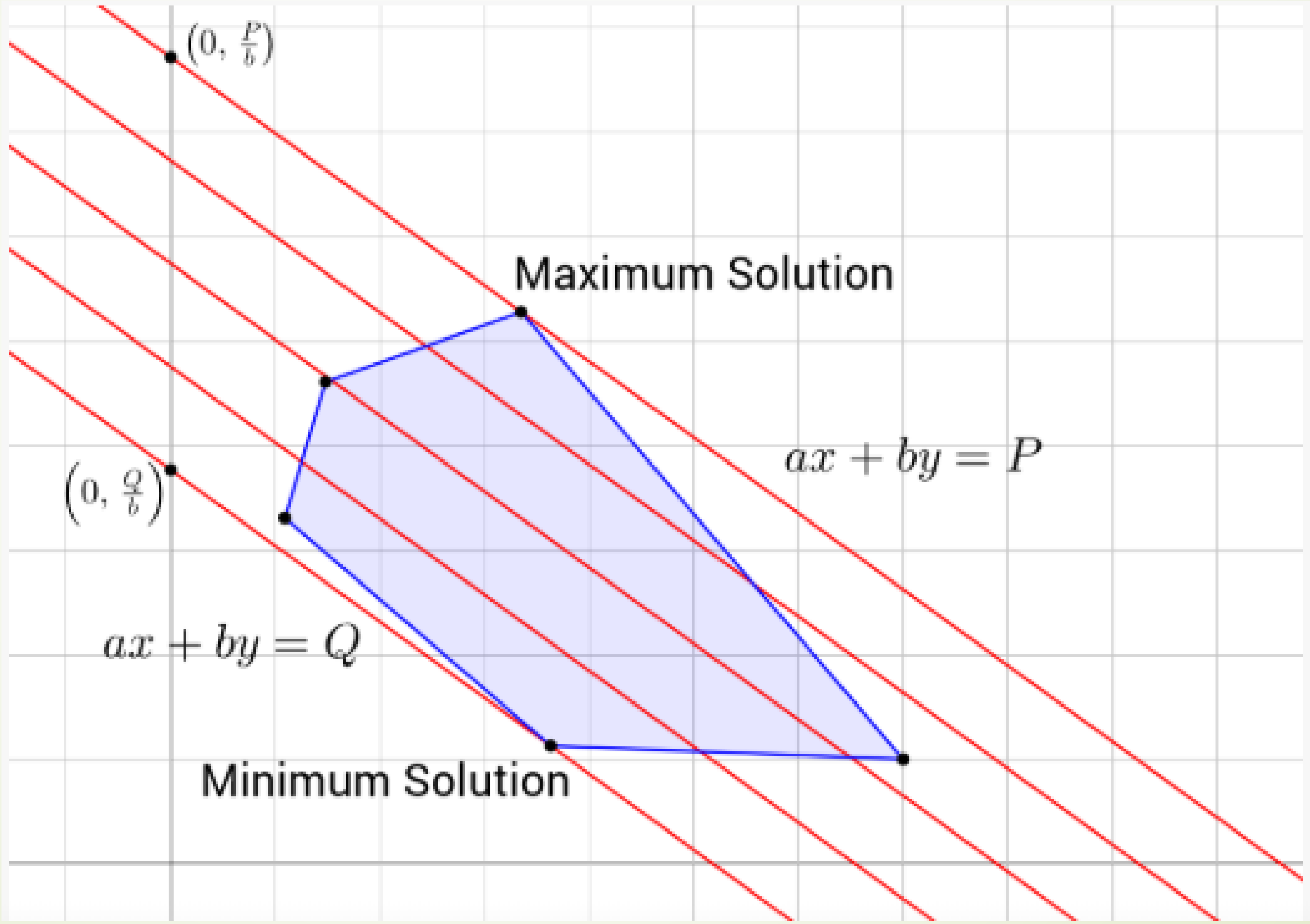
- بازرسی بعد از تمام وظایف انجام می‌شود و ۳ دقیقه طول می‌کشد. پس برای هر متغیر به جز Inspect یک محدودیت به شکل $X+d_X \le Inspect$ اضافه می‌کنیم.
  - مدت زمان انجام متغیر X = d_X
- اگر یک شرط وجود داشته باشد که کل اسمبل کردن باید در ۳۰ دقیقه انجام شود، می‌توان با محدود کردن دامنه‌ی تمام متغیرها به این شرط رسید.

$$D_i=\{1,2,3,\ldots,27\}$$

انواع متغیرهای CSP

- متغیرهای گسسته
  - دامنه‌های محدود
    - n متغیر، اندازه هر دامنه d: تعداد انتساب های کامل $O(d^n)$
    - مثال: ۸ وزیر ، رنگ‌آمیزی نقشه و …
  - دامنه‌های نامحدود
    - اعداد صحیح، رشته‌ها و … مثال: در برنامه‌ریزی وظایف اگر deadline قرار نمی‌دادیم تعداد نامتناهی زمان شروع برای هر متغیر وجود داشت.
    - نیاز به بیان انتزاعی محدودیتها دارند. زیرا امکان بیان صریح تمام حالات ارضا کننده محدودیت وجود ندارد مثال: $StartJob1 + 5 \leq StartJob3$
- متغیرهای پیوسته
  - اگر محدودیت‌ها تنها خطی باشند آن‌گاه مسئله توسط برنامه‌ریزی خطی (Linear Programming) و در زمان چندجمله‌ای نسبت به تعداد متغیرها قابل حل خواهد بود.

برنامه ریزی خطی (اشاره)



انواع محدودیت‌ها از نظر تعداد متغیرها

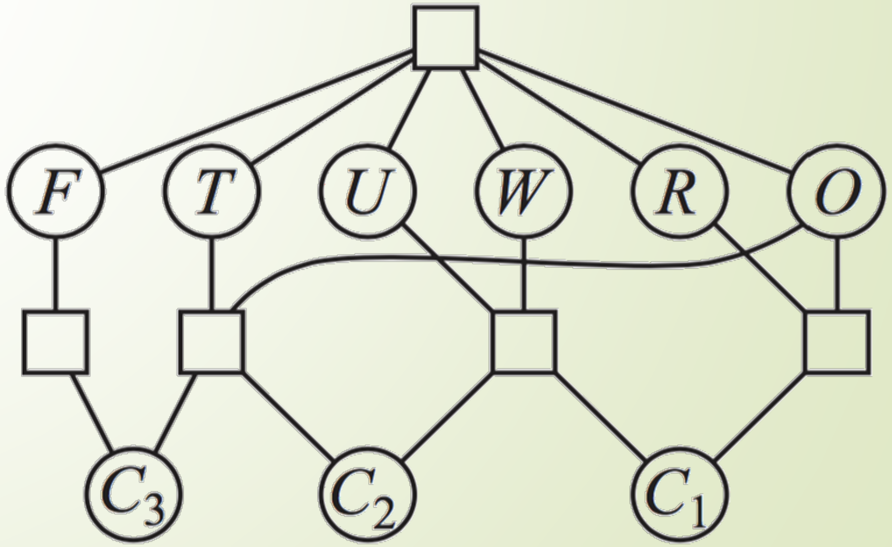
- یگانی (unary) باعث محدود شدن مقدار یک متغیر منفرد می‌شود.
  - مثال: <(SA), SA ≠ green >
- دوگانی (binary) محدودیت شامل یک زوج از متغیرها می‌باشد،
  - مثال: <(SA, WA), SA ≠ WA >
- محدودیت مرتبه بالاتر(Higher-order) شامل سه یا بیشتر متغیر است.
  - مثال: مقدار Y بین X و Z است Between(X, Y, Z)
- محدودیت سراسری شامل تعداد دلخواه از متغیرها است.
  - مثال: Alldiff یعنی تمام متغیرهای موجود در محدودیت باید مقادیر متفاوت داشته باشند.

مثال: مسئله ریاضیات رمزی

هر کاراکتر بیانگر یک عدد متفاوت است.

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

- Variables: $F\ T\ U\ W\ \ R\ O\ C_1\ C_2\ C_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: $Alldiff\ (F,T,U,W,R,O)$
  - $O + O = R + 10 \cdot C_1$
  - $C_1 + W + W = U + 10 \cdot C_2$
  - $C_2 + T + T = O + 10 \cdot C_3$
  - $C_3 = F,\ T \neq 0,\ F \neq 0$



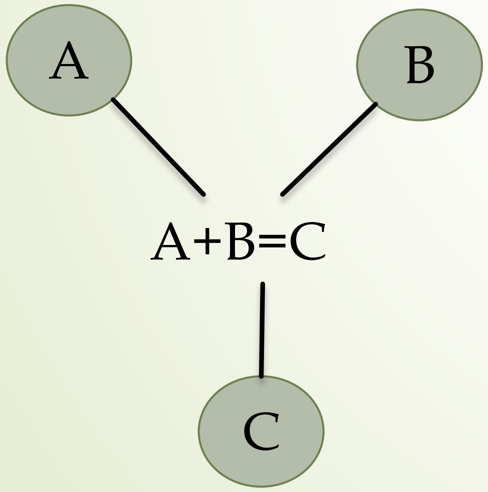
  - $\{C_1, C_2, C_3\} \equiv \{C_{10}, C_{100}, C_{1000}\}$

انواع محدودیت‌ها از نظر اولویت

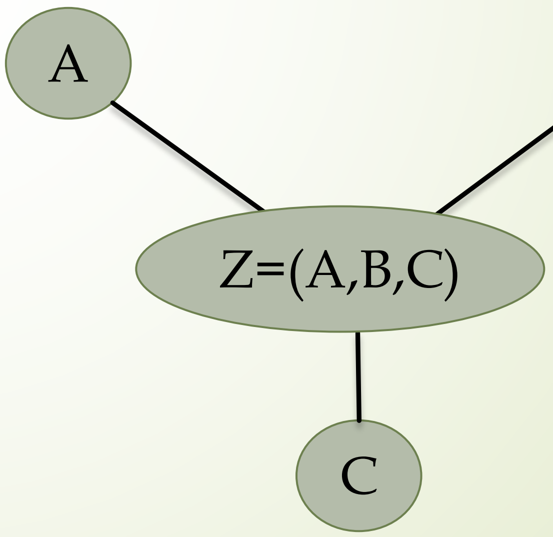
- محدودیت مطلق: محدودیت‌هایی که نباید هیچ یک از آن‌ها در راه‌حل نقض شود.
  - مثال: یک استاد نمی‌تواند به‌طور همزمان در دو کلاس درس دهد.
- محدودیت اولویت‌دار: نشان می‌دهد کدام راه‌حل ارجح است.
  - مثال: استاد X کلاس‌های صبح را ترجیح می‌دهد و استاد Y کلاس‌های عصر را.
  - اگر برنامه‌ی زمانی داشته باشیم که برای استاد X در عصر کلاس گذاشته شده باشد باز هم یک راه‌حل خواهد بود اگر چه راه‌حل بهینه نیست.
- محدودیت‌های اولویت دار می‌توانند به‌صورت هزینه‌هایی بر روی انتساب هر یک از متغیرها کد شوند. با این فرمول‌بندی، CSP را می‌توان با روش‌های بهینه‌سازی مانند Linear Programming حل کرد.
  - کلاس عصر برای استاد X دو امتیاز منفی داشته باشد در حالی که کلاس صبح یک امتیاز مثبت.

باینری CSP

- هر محدودیت n تایی با دامنه‌ی متناهی می‌تواند تبدیل به یک مجموعه از محدودیت‌های باینری شود. چگونه؟
  - معرفی متغیرهای کمکی به‌گونه‌ای که دامنه‌ی آن‌ها برابر با حاصل ضرب کارتزین دامنه‌های اصلی باشد
  - تعریف محدودیت n تایی اصلی به‌عنوان محدودیت یگانی روی متغیرهای کمکی



گراف محدودیت 3گانه



گراف محدودیت باینری

CSP باینری – مثال

- برای مثال اگر دامنه متغیرهای A، B و C به‌صورت زیر داده شده باشد

$D_A=\{0,1\}$ $D_B=\{0,1\}$ $D_C=\{0,1,2\}$

- متغیر کمکی Z را با دامنه‌ی ضرب کارتزین متغیرهای فوق به‌صورت زیر تعریف می‌کنیم:

$D_Z=\{(0,0,0),(0,0,1),(0,0,2),(0,1,0),\ldots,(1,1,2)\}$

- محدودیت nتایی اصلی را بر روی Z به‌صورت یک محدودیت یگانی اعمال می‌کنیم:

$D_Z=\{1,2,3,4\}$ $\mathrm{I}\rightarrow(0,0,0)$ $\mathrm{II}\rightarrow(0,1,1)$ $\mathrm{III}\rightarrow(1,0,1)$ $\mathrm{IV}\rightarrow(1,1,2)$

- آن‌گاه محدودیت‌های باینری را میان زوج متغیرهای {A,Z}، {B,Z} و {C,Z} اعمال می‌کنیم:

<(A,Z),[(0,I),(0,II),(1,III),(1,IV)]>

<(B,Z),[(0,I),(0,III),(1,II),(1, IV)]>

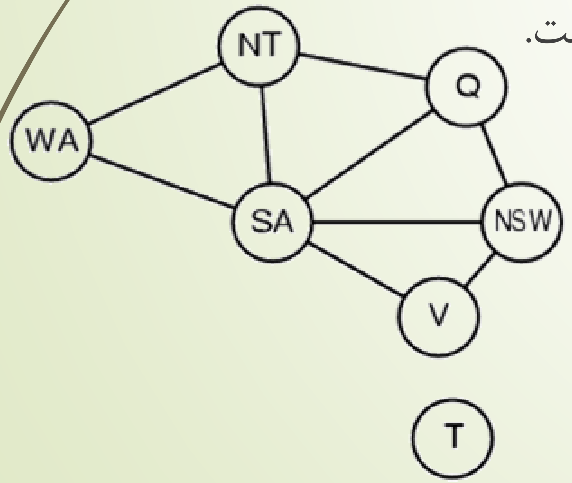
<(C,Z),[(0,I),(1,II),(1,III),(2, IV)]>

حال میتوان انتشار محدودیت را به شکل ساده ای انجام داد.

انتشار محدودیت: استنتاج در CSPها

پایه‌ی استنتاج برای CSP ها: سازگاری محلی

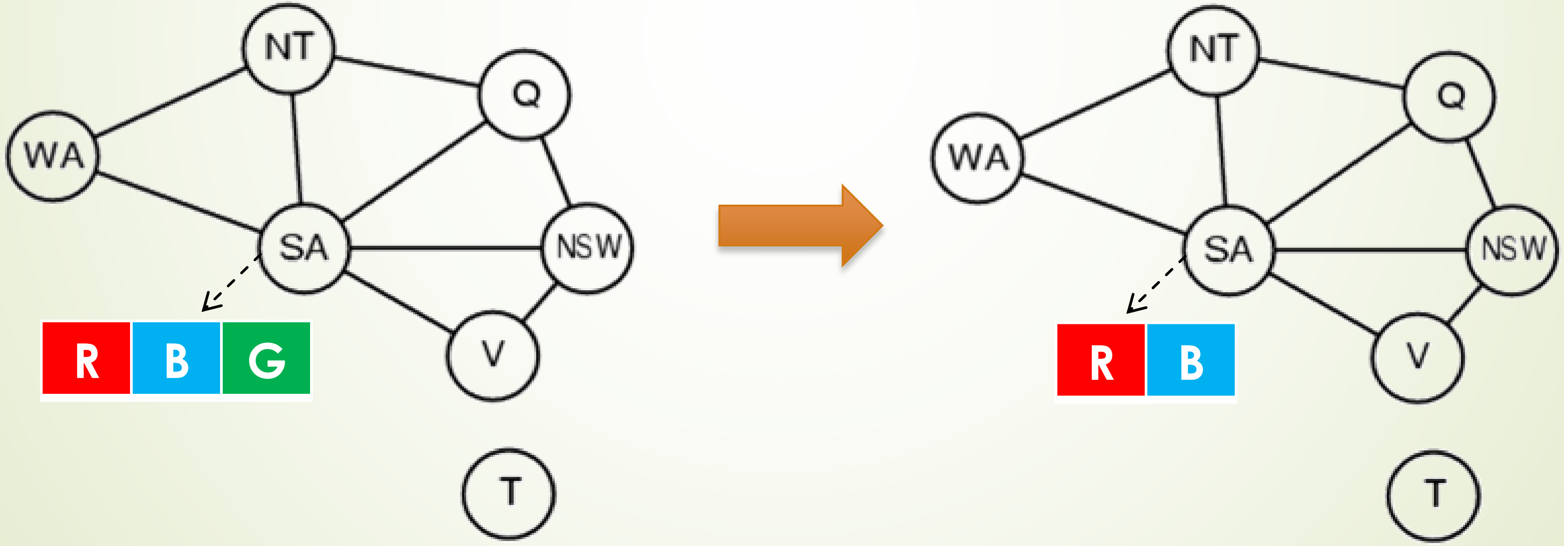
- در جستجوی فضای حالت، یک الگوریتم تنها یک کار می‌تواند انجام دهد: جستجو
- اما در CSPها یک انتخاب برای انجام وجود دارد: جستجو یا انتشار محدودیت
  - جستجو: از مقادیر ممکن برای انتساب متغیر یکی را انتخاب کند.
  - انتشار محدودیت: با استفاده از محدودیت‌ها تعداد مقادیر قانونی برای یک متغیر کم می‌شود که به نوبه‌ی خود می‌تواند مقادیر قانونی برای متغیرهای دیگر را نیز کاهش دهد و ...
  - این دو اقدام می‌تواند در هم تنیده باشند یا انتشار محدودیت مقدمه ای بر جستجو باشد
  - گاهی با انجام انتشار محدودیت، مساله به طور کامل حل میشود و نیازی به جستجو نیست.
- سازگاری محلی (local consistency)
  - فرایند اجرای سازگاری محلی در هر بخش از گراف محدودیت باینری (هر گره یک متغیر و هر یال محدودیتی باینری بین دو متغیر)، باعث می‌شود مقادیر ناسازگار در سراسر گراف زدوده شود.
  - انواع سازگاری محلی: سازگاری نود، سازگاری کمان، ...



سازگاری نود (Node-consistency)

- یک متغیر سازگار نود است اگر تمام مقادیر در دامنه‌ی متغیر با محدودیت‌های یگانی متغیر سازگار باشد.
  - برای مثال اگر در نوعی مسئله رنگ‌آمیزی نقشه، SA رنگ سبز را قبول نکند، می‌توان سازگاری نود را با حذف رنگ سبز از دامنه آن برقرار ساخت.

$$D_{SA}=\{red, blue, green\}\ , SA\neq green \rightarrow D_{SA}=\{red, blue\}$$



اگر همه نودهای گراف سازگار باشند، گراف را سازگارنود می‌نامیم

سازگاری کمان (Arc-consistency)

- یک متغیر سازگار کمان است اگر هر مقدار در دامنه‌ی آن محدودیت‌های دوتایی متغیر را ارضا کند.
- به‌طور دقیق: X_i نسبت به متغیر دیگر X_j سازگار کمان است اگر برای هر مقدار در دامنه‌ی فعلی D_i مقداری در دامنه‌ی D_j وجود داشته باشد که محدودیت باینری روی کمان (X_i,X_j) را ارضا کند.
  - رابطه دو طرفه نیست!
- مثال
  - متغیرها: $\{X, Y\}$
  - دامنه: $\{0,1,2,\ldots,9\}$
  - محدودیت: $Y=X^2$

✓آیا X نسبت به Y سازگار یال است؟
✗خیر باید دامنه X را به $\{0,1,2,3\}$ تبدیل کنیم.
✓آیا Y نسبت به X سازگار است؟
✗خیر باید دامنه Y را به $\{0,1,4,9\}$ تبدیل کنیم.

الگوریتم سازگاری کمان

برای هر کمان (X_i, X_j) در صف آن را از صف خارج کن

X_i را نسبت به X_j سازگار کن

۱- اگر دامنه D_i بدون تغییر ماند آن‌گاه ادامه بده

۲- اگر $|D_i|=0$ آن‌گاه "false" برگردان

۳- هر همسایه X_k از X_i به‌جز X_j را به صورت زیر به صف اضافه کن

(X_k, X_i)

حذف یک مقدار از یک دامنه ممکن است باعث ناسازگاری‌های دیگری در همسایه‌ها (نسبت به گره جاری) شود. بنابراین ما باید رویه را تا زمانی که همه چیز سازگار شود انجام دهیم.

- چرا «به‌جز X_j»؟
- اگر X_j نسبت به X_i سازگار باشد، آیا ممکن در اثر اعمال سازگاری معکوس و کم شدن از دامنه X_i، سازگاری X_j نسبت به X_i نقض شود؟
- هنگامی که صف خالی شد CSP حاصل هم‌ارز با CSP اصلی خواهد بود
  - راه‌حل هر دو یکسان است اما متغیرهای با دامنه‌های کوچکتر منجر به جستجوی سریع‌تر می‌شود.

AC-3 الگوریتم سازگاری کمان

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
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 REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
         add (X_k, X_i) to queue
return true
```

---

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i

```
revised ← false
for each x in D_i do
   if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
return revised
```

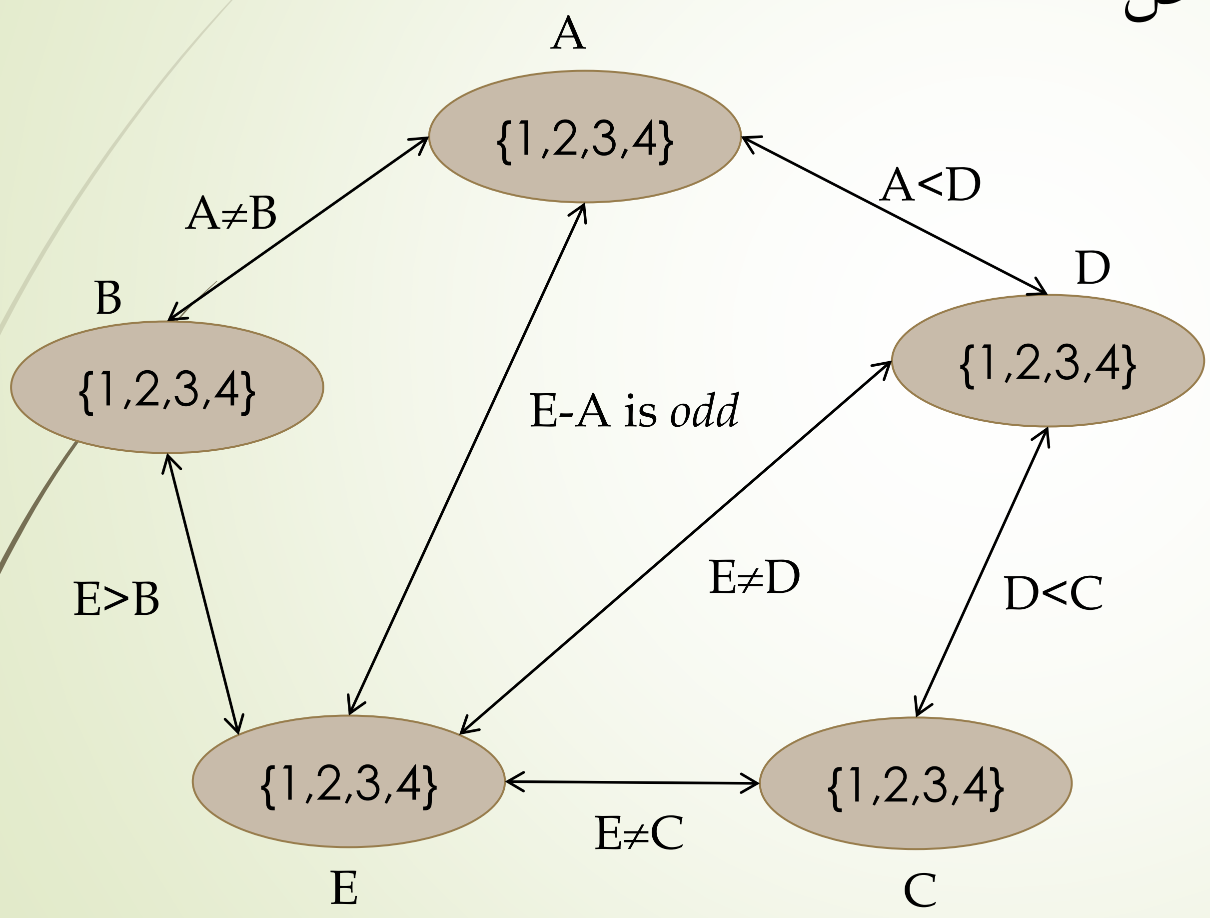
پیچیدگی الگوریتم AC-3

- یک CSP با شرایط زیر درنظر بگیرید
  - تعداد متغیرها: n
  - حداکثر سایز هر متغیر: d
  - تعداد محدودیت‌های دوتایی: c
- هر کمان (X_i, X_k) تنها d بار می‌تواند در صف قرار گیرد.
  - زیرا X_k حداکثر d مقدار برای حذف دارد.
- چک کردن سازگاری کمان: $O(d^2)$ (به ازای هر متغیر در مبدا یک متغیر در مقصد داریم؟)

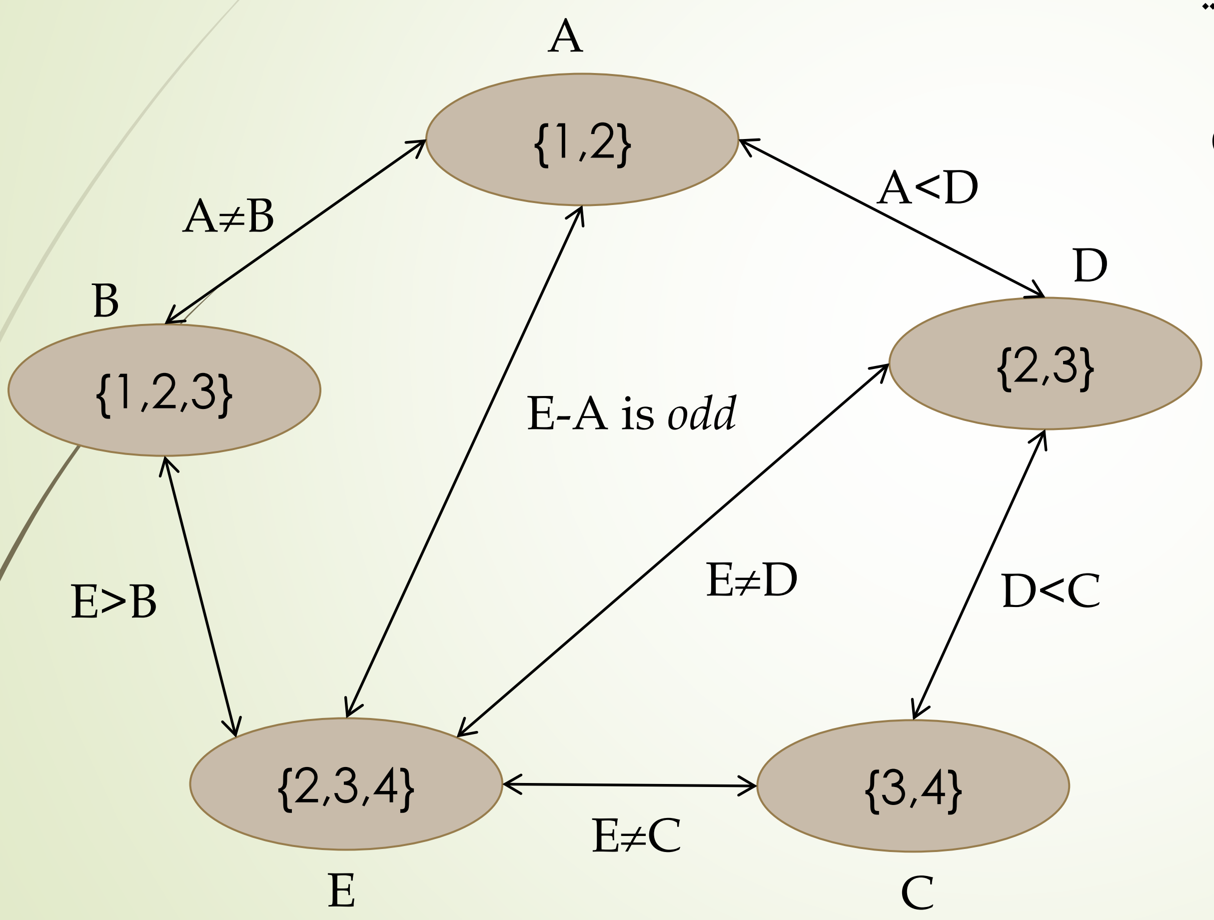
$$O(cd^3)$$

مثال– سازگاری کمان

گراف محدودیت زیر را در نظر بگیرید و مراحل الگوریتم AC-3 را بر روی آن نشان دهید



مثال– سازگاری کمان ...



صف اولیه مجموعه تمامی کمان‌های گراف محدودیت است.

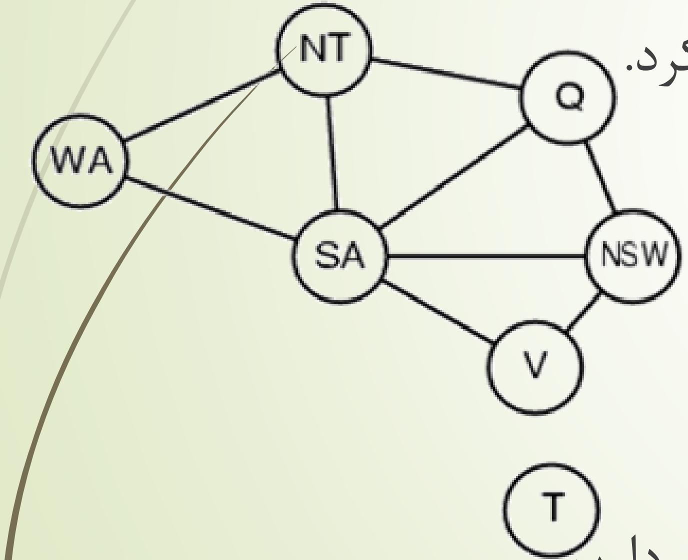
Queue= {<E,B>, <B,E>, <A,B>, <B,A>, …}

خلاصه‌ای از مراحل انجام شده توسط الگوریتم:
نمایش مقادیر حذف شده با بررسی هر کمان

E=1	<E, B>
B=4	<B, E>
D=4	<D, C>
C=1	<C, D>
A=3, A=4	<A, D>
D=1	<D, A>
C=2	<C, D>

سازگاری کمان در مسئله رنگ‌آمیزی نقشه

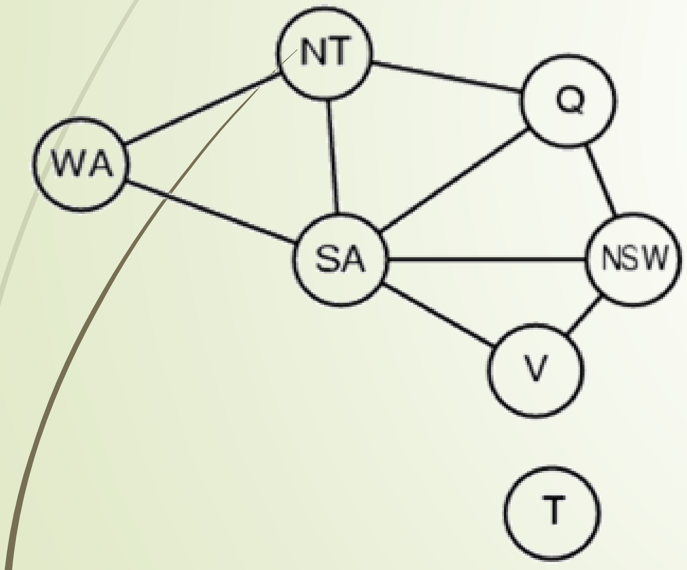
- در مورد یک مسئله رنگ‌آمیزی برای یک نقشه دلخواه، تمامی زوج متغیرها سازگار کمان هستند اگر $|D_i| \geq 2$ برای هر i برقرار باشد.
- مثال: فرض کنید هر ناحیه را بتوان تنها با دو رنگ قرمز و آبی رنگ‌آمیزی کرد.
  - آیا متغیرها در این حالت سازگار کمان هستند؟
  - آیا اعمال سازگاری کمان نیاز است؟
  - آیا راه‌حلی برای مسئله وجود دارد؟



- نتیجه: در برخی مسائل، سازگاری کمان برای استنتاج کمکی به تسریع حل مساله نمی‌کند و ما نیاز به مفهومی قوی‌تر از سازگاری داریم.

سازگاری مسیر (Path-consistency)

- مجموعه‌ی دو متغیره $\{X_i,X_j\}$ نسبت به متغیر سوم X_m سازگار مسیر است اگر برای هر انتساب $\{X_i=a, X_j=b\}$ که محدودیتهای روی X_i,X_j را ارضا می‌کند، یک انتساب برای X_m وجود داشته باشد که محدودیت‌های روی $\{X_i,X_m\}$ و $\{X_m,X_j\}$ را ارضا کنند.
- مثال: اعمال سازگاری مسیر در مورد رنگ‌آمیزی نقشه با دو رنگ
  - سازگاری $\{SA,WA\}$ نسبت به NT؟



انتساب‌های سازگار به $\{SA,WA\}$ به‌صورت زیر است:

$\{SA=blue, WA=red\}$ یا $\{SA=red, WA=blue\}$

در این صورت $NT=\{\}$ خواهد بود و هیچ راه‌حلی برای مسئله وجود ندارد.

سازگاری مرتبه kام (k-consistency)

- یک CSP دارای سازگاری مرتبهٔ k است اگر برای هر مجموعهٔ k-1 عضوی از متغیرها و برای هر انتساب سازگار به آن‌ها، همیشه یک مقدار سازگار یافت شود که بتوان به متغیر kام انتساب داد.
  - سازگاری مرتبهٔ ۱ = سازگاری گره
  - سازگاری مرتبهٔ ۲ = سازگاری کمان
  - سازگاری مرتبهٔ ۳ = سازگاری مسیر

سازگاری قوی مرتبه k (Strongly k-consistent)

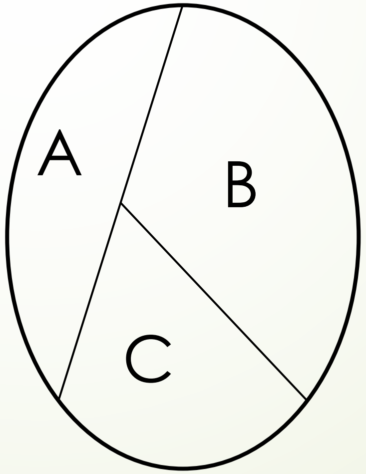
- یک CSP قویاً k سازگار است اگر دارای سازگاری مرتبهٔ k، مرتبهٔ k-1، مرتبهٔ k-2، تا سازگار مرتبهٔ ۱ نیز باشد.
- فرض کنید یک CSP با n گره هر کدام حداکثر با d مقدار داشته باشیم و این CSP دارای سازگاری قوی مرتبهٔ n است.
  - در ابتدا، یک مقدار سازگار را برای X_1 انتخاب می‌کنیم. از آنجا که گراف دارای سازگاری مرتبهٔ ۲ است، مطمئن هستیم که می‌توان مقداری سازگار یافت که بتوان به X_2 اختصاص داد و به ترتیب، می‌توان مقادیر سازگاری برای بقیه متغیرها را یافت.
  - می‌توان این مسأله را بدون انجام عقب‌گرد، حل کرد.
  - راه‌حل مسأله حداکثر با مرتبهٔ زمانی $O(n^2d)$ پیدا می‌شود.
    - n: انتساب مقدار به هر یک از گره های n گانه
    - d: تست d مقدار برای یافتن مقدار ممکن به هر گره
    - n: تست سازگاری (ارضاء محدودیت) با حداکثر n همسایه هنگام ارزیابی امکان پذیر بودن انتساب هر مقدار به هر گره

کدام سطح سازگاری؟

- هر الگوریتم برای برقرای سازگاری مرتبه k در بدترین حالت از نظر زمان و حافظه نمایی است.
- می‌بایست توازنی (trade off) میان زمان مورد نیاز برای برقراری سازگاری kام و مقدار حذفیات از فضای جستجو برقرار کرد.
  - در عمل معمولا سازگاری ۲ انجام می‌شود و کم‌تر از سازگاری ۳ استفاده می‌شود.

تست

مسئله ارضای محدودیت رنگ کردن نقشه زیر را در نظر بگیرید. فرض کنید می‌خواهیم این نقشه را با تنها یک رنگ، رنگ‌آمیزی کنیم (شهرهای مجاور نباید هم‌رنگ باشند). با این فرض، گراف محدودیت این مسئله به ازای چه مقادیری از k دارای خاصیت k-consistency است؟



۱) فقط $k=0$

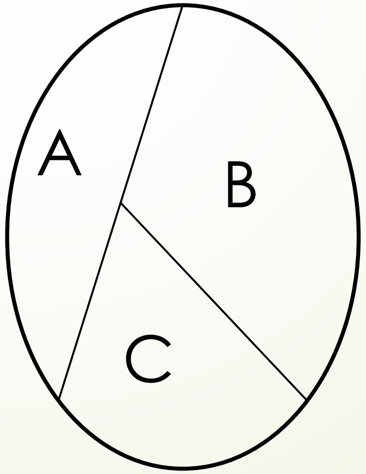
۲) فقط $k=1$

۳) $k=1$ و $k=2$

۴) $k=1$ و $k=3$

تست

مسئله ارضای محدودیت رنگ کردن نقشه زیر را در نظر بگیرید. فرض کنید می‌خواهیم این نقشه را با تنها یک رنگ، رنگ‌آمیزی کنیم (شهرهای مجاور نباید هم‌رنگ باشند). با این فرض، گراف محدودیت این مسئله به ازای چه مقادیری از k دارای خاصیت k-consistency است؟



۱) فقط $k=0$

۲) فقط $k=1$

۳) $k=1$ و $k=2$

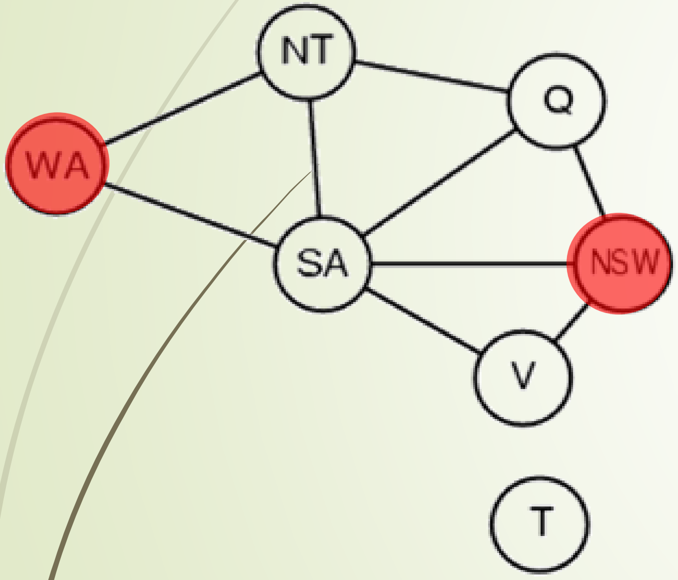
۴) $k=1$ و $k=3$ ✓

برخورد با محدودیت‌های ویژه

۱- محدودیت Alldiff: تمامی متغیرها باید دارای مقادیر متفاوتی باشند.

- یک شکل ساده از کشف ناسازگاری برای این محدودیت
  - اگر m متغیر شامل این محدودیت باشند ولی تنها n مقدار که n<m است برای انتساب موجود باشد دیگر این محدودیت نمی‌تواند برآورده شود.
- یک الگوریتم ساده برای اعمال محدودیت Alldiff
  - در ابتدا تمام متغیرهایی که دارای دامنهٔ واحد می‌باشند را حذف و مقدار مربوط به دامنه‌ی آن‌ها را نیز از دامنه‌ی سایر متغیرها حذف می‌کنیم.
  - این کار را برای تمام متغیرهای تک مقداره تکرار می‌کنیم.
  - اگر در پایان دامنه‌ی یک متغیر تهی شود یا تعداد متغیر بیشتر از تعداد مقادیر دامنه باقی مانده (برای تمام متغیرها) باشد، ناسازگاری در مسأله تشخیص داده خواهد شد.

برخورد با محدودیت‌های ویژه ...



۱- محدودیت Alldiff: تمامی متغیرها باید دارای مقادیر متفاوتی باشند.

- مثال: رنگ‌آمیزی نقشه
  - فرض کنید انتساب زیر صورت گرفته است

{WA=red, NSW=red}

  - محدودیت سراسری زیر را داریم

Alldiff(SA, NT, Q)

  - بعد از اعمال AC-3 دامنه متغیرها به‌صورت زیر خواهد شد

{green , blue}

  - سه متغیر و دو رنگ داریم پس محدودیت Alldiff ناسازگار می‌شود.

برخورد با محدودیت‌های ویژه ...

۳- محدودیت حدود (Bounds): در مسائلی که نمی‌توان دامنه‌ی هر متغیر را به‌صورت مجموعه‌ای از اعداد صحیح نشان داد، از حدود بالا و پایین برای نمایش استفاده می‌کنند و انتشار محدودیت حدود را به کار می‌برند.

- مثال- برنامه‌ریزی خطوط هوایی (ظرفیت هواپیما)
  - دو پرواز F_1 و F_2 وجود دارد که هر کدام به‌ترتیب ظرفیت ۱۶۵ و ۳۸۵ را دارند.
  - دامنه‌ی اولیه تعداد مسافران برای هر پرواز $D_1=[0,165]$ و $D_2=[0,385]$ است.
  - محدودیت: مجموع تعداد مسافران هر دو پرواز ۴۲۰ نفر شود.
  - انتشار محدودیت حدود: دامنه‌ها به $D_1=[35,165]$ و $D_2=[255,385]$ تبدیل می‌شوند.

جستجوی افزایشی در CSPها

برای حل بسیاری از مسائل CSP علاوه بر استنتاج نیاز به جستجو نیز وجود دارد.

جستجو: افزایشی (انتساب جزئی مقادیر به متغیرها و جلو رفتن و در صورت بن بست، انجام عقبگرد)

جستجو: کامل (انتساب مقدار به همه متغیرها و ارزیابی و تغییر هر مجموعه انتساب (جستجوی محلی))

در این بخش، **جستجوی افزایشی** بررسی می‌شود

فرمول‌بندی افزایشی CSPها

- **حالات:** مجموعه‌ی تمامی انتساب‌های جزئی سازگار
- **حالت اولیه:** تمام متغیرها بدون مقدار، یعنی انتساب تهی
- **اقدام‌ها:** انتساب مقدار به یکی از متغیرهایی که مقدار ندارد به طوری‌که با مقادیر متغیرهای قبلی ناسازگار نباشد.
  - در صورت عدم وجود انتساب‌های مجاز شکست می‌خورد.
- **آزمون هدف:** آیا انتساب فعلی کامل است؟
  - چون عملیات به‌گونه‌ای انجام می‌شود که هر انتساب سازگار باشد در این آزمون تنها بررسی می‌شود که تمام متغیرها مقداردهی شده‌اند یا خیر.
- **هزینه مسیر:** هزینه یکسان برای تمام مراحل (هزینه اهمیت ندارد)

استفاده از الگوریتم‌های جستجوی استاندارد

- برای تمام مسائل CSP این روش قابل استفاده است.
- الگوریتم جستجوی عمق برای یک CSP با n متغیر و سایز دامنه d
  - هر پاسخ در عمق n و با n متغیر ظاهر می‌شود.
  - در سطح یک فاکتور انشعاب برابر است با nd در سطح دو (n-1)d و ...
  - بنابراین تعداد برگ‌های درخت جستجو برابر است با $n!d^n$

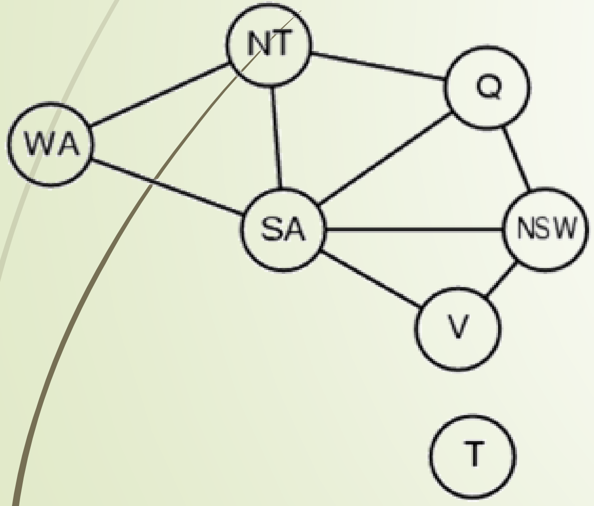
$$(nd) * [(n-1)d] * [(n-2)d] * \ldots d = n!d^n$$

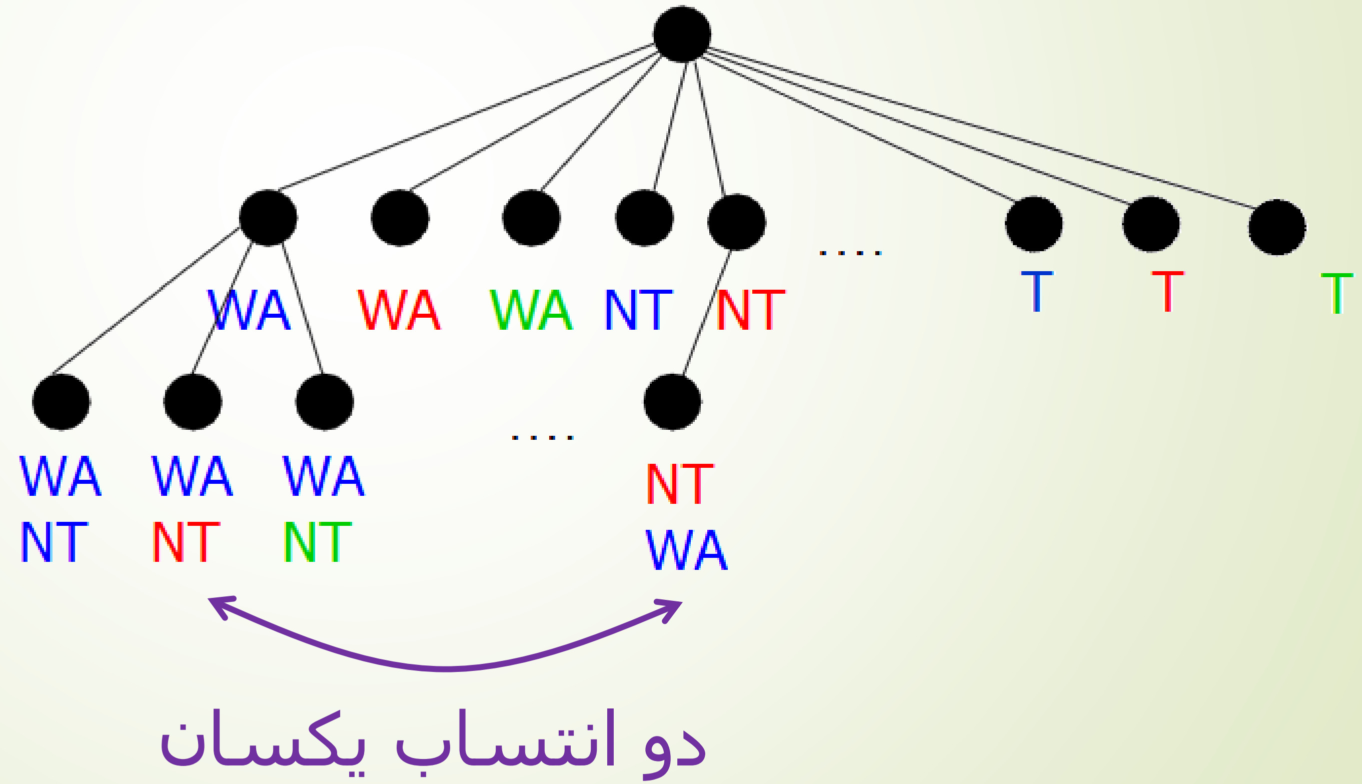
  - با این‌وجود تنها d^n انتساب کامل وجود دارد.

بهبود روش قبل با خاصیت جابه‌جایی CSPها

❖ یک مسئله دارای خاصیت جابه‌جایی (commutativity) است اگر ترتیب اِعمال هر مجموعه از اقدامات هیچ تأثیری بر روی نتیجه ایجاد کند.

❖ مستقل از ترتیب انتساب مقادیر به متغیرها، به انتساب‌های جزئی یکسانی خواهیم رسید.

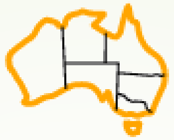




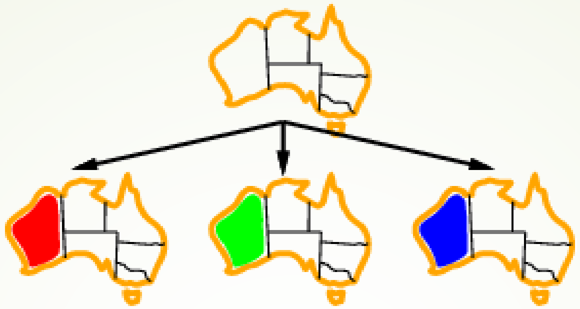
جستجوی عقب‌گرد (Backtracking)

- جستجوی عقب‌گرد، جستجوی اول عمقی است که در هر زمان برای یک متغیر مقداری در نظر گرفته می‌شود و اگر هیچ مقدار مجازی برای انتساب به یک متغیر باقی نمانده باشد به عقب برمی‌گردد.
  - یک متغیر بدون انتساب را انتخاب کن
  - تمام مقادیر دامنه آن متغیر را به نوبت امتحان کن
  - اگر مقداری سازگار با مقادیر انتساب‌یافته قبلی پیدا کردی به آن متغیر انتساب بده در غیر این صورت به عقب برگرد
- تعداد برگ‌ها؟ d^n
  - هرس زیردرخت‌ها

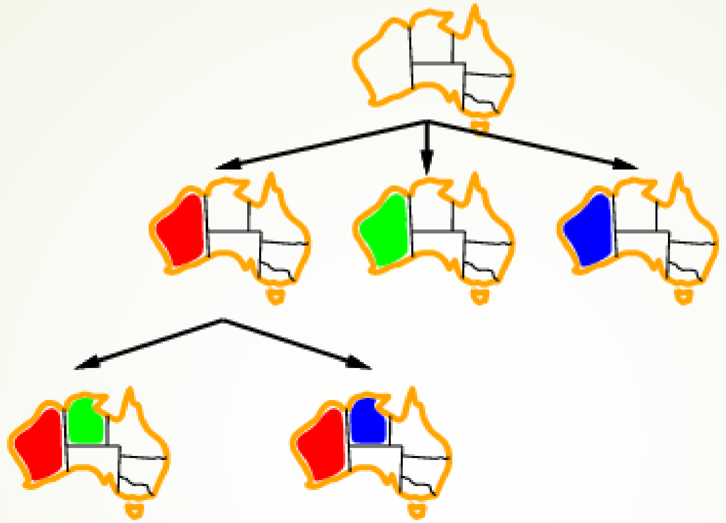
جستجوی عقب‌گرد–مثال رنگ‌آمیزی نقشه



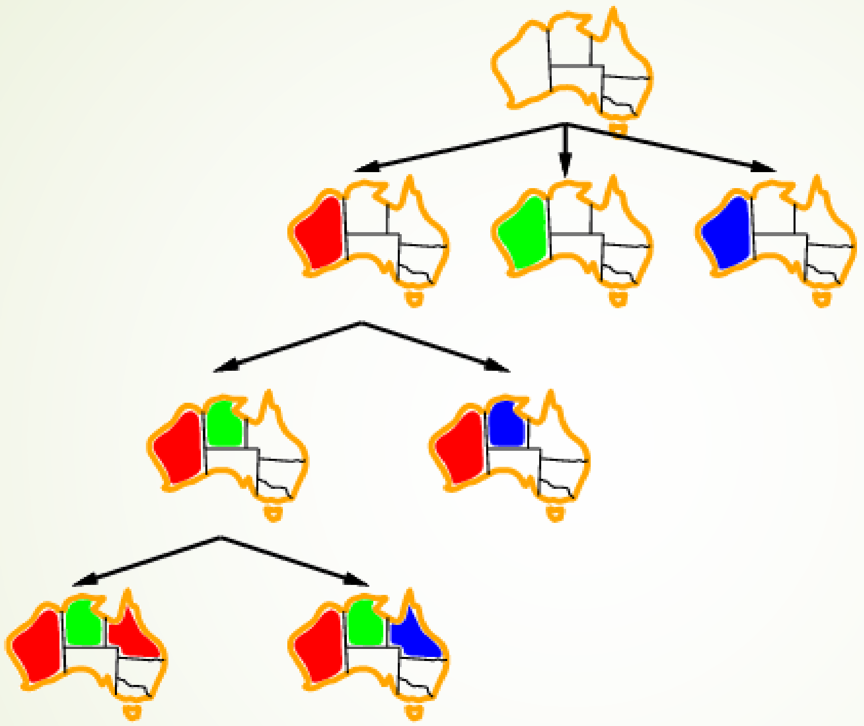
جستجوی عقب‌گرد–مثال رنگ‌آمیزی نقشه



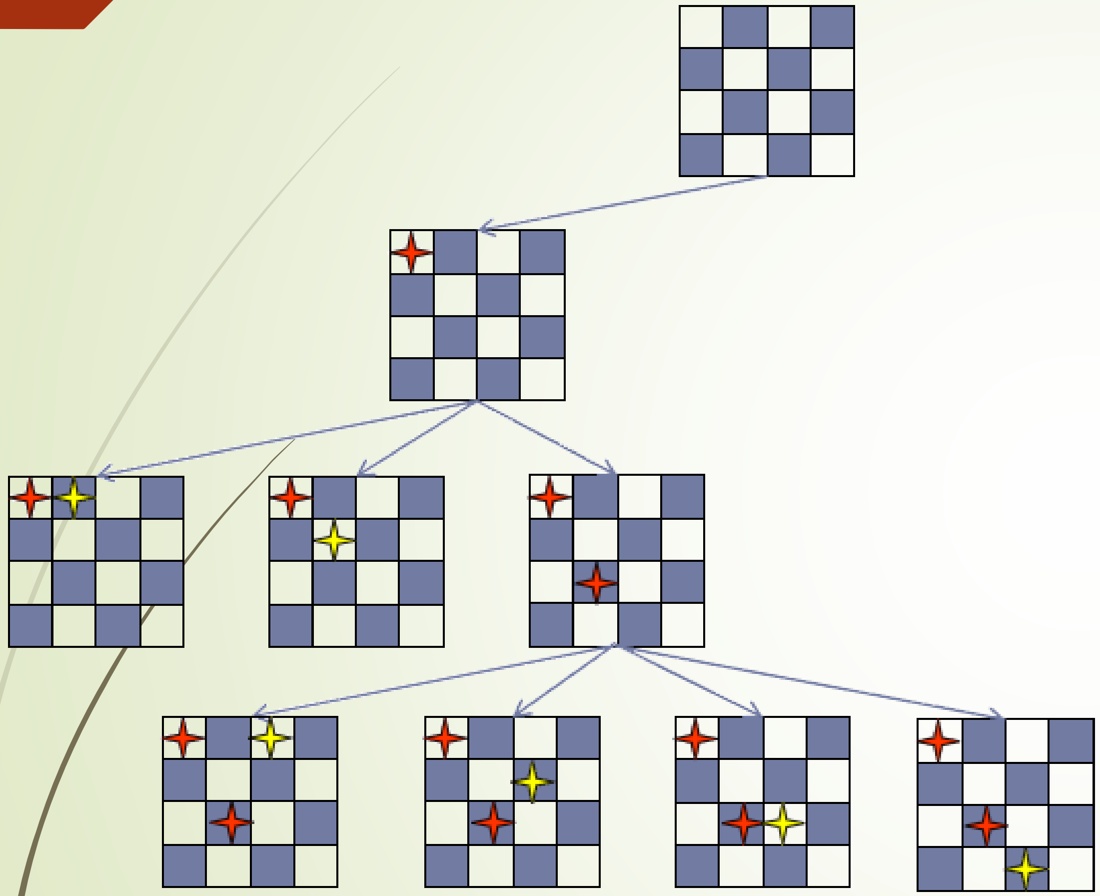
جستجوی عقب‌گرد–مثال رنگ‌آمیزی نقشه



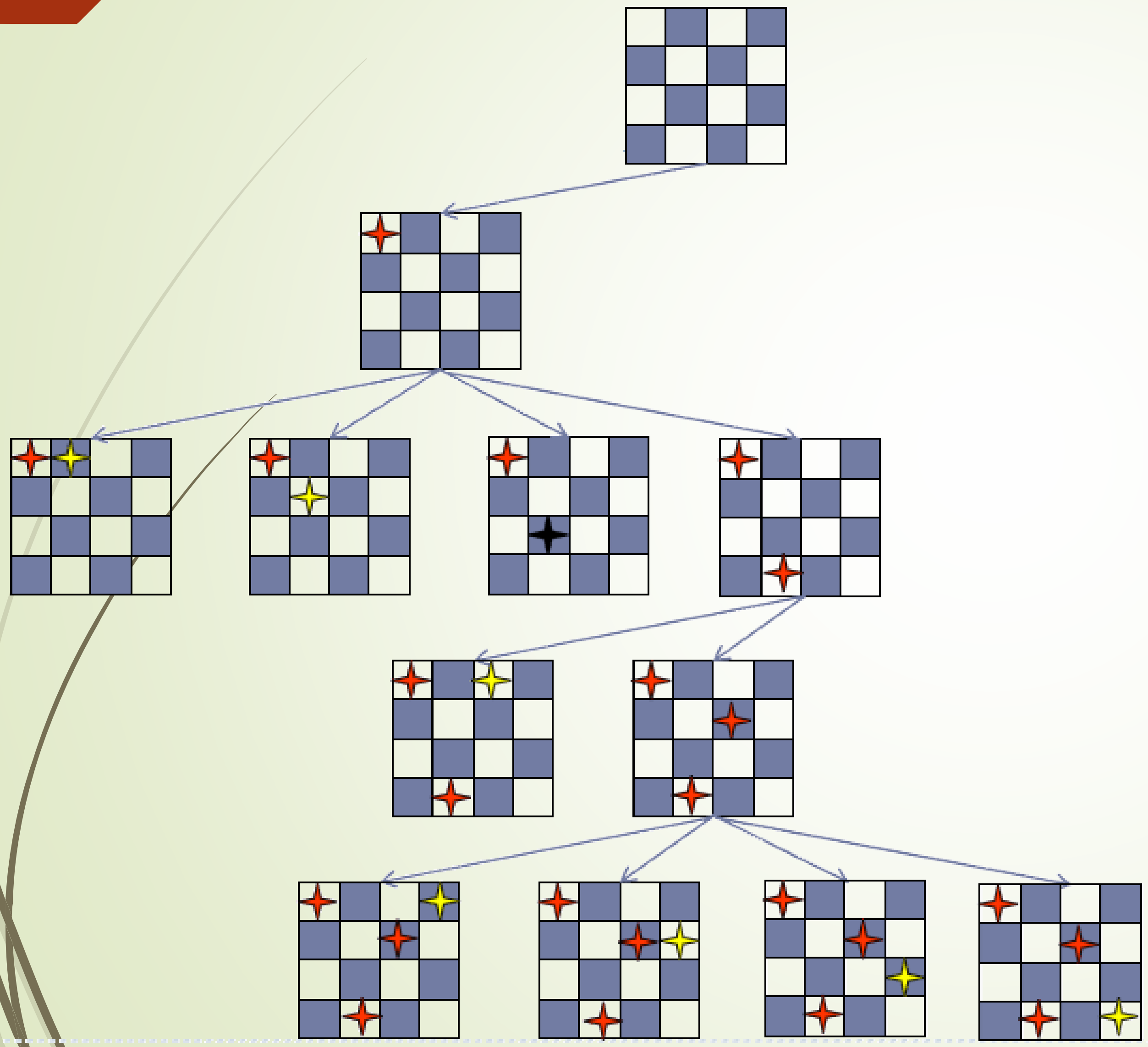
جستجوی عقب‌گرد–مثال رنگ‌آمیزی نقشه



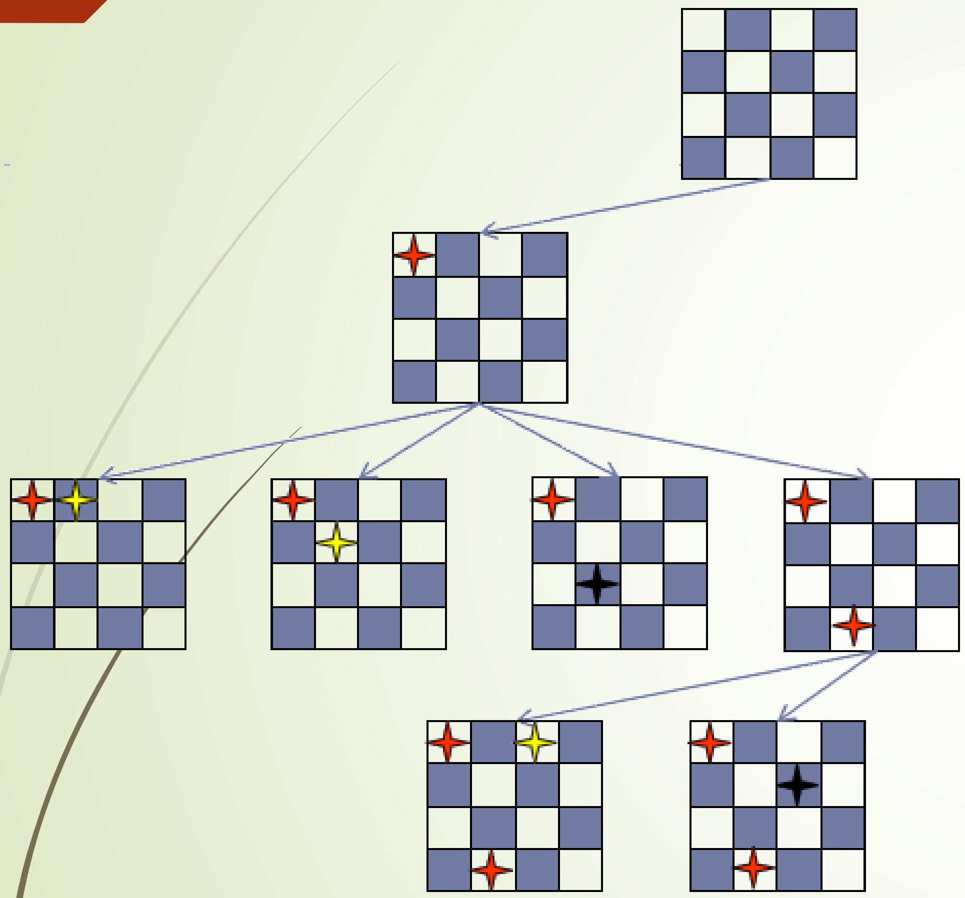
جستجوی عقب‌گرد–مثال ۴ وزیر



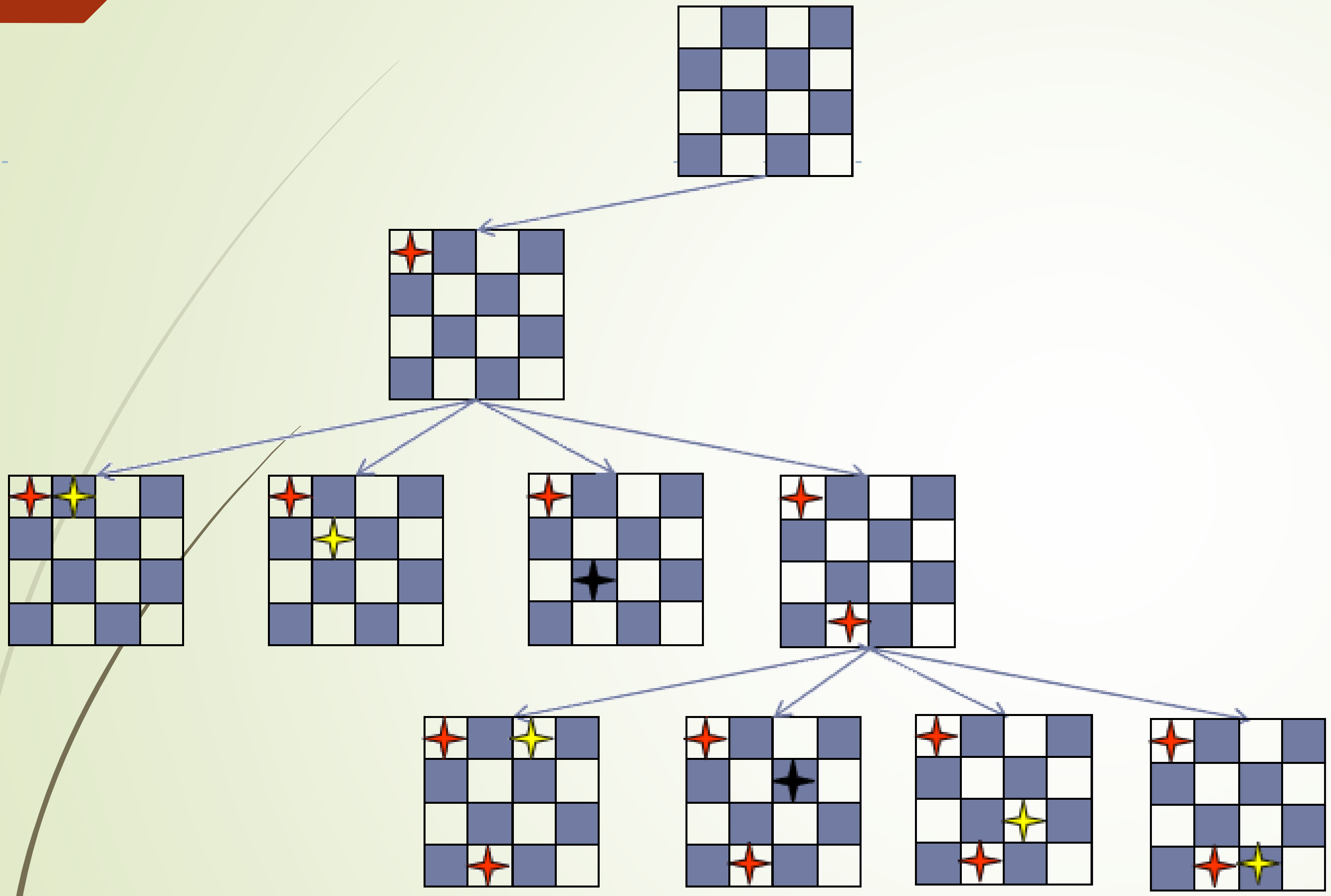
جستجوی عقب‌گرد–مثال ۴ وزیر



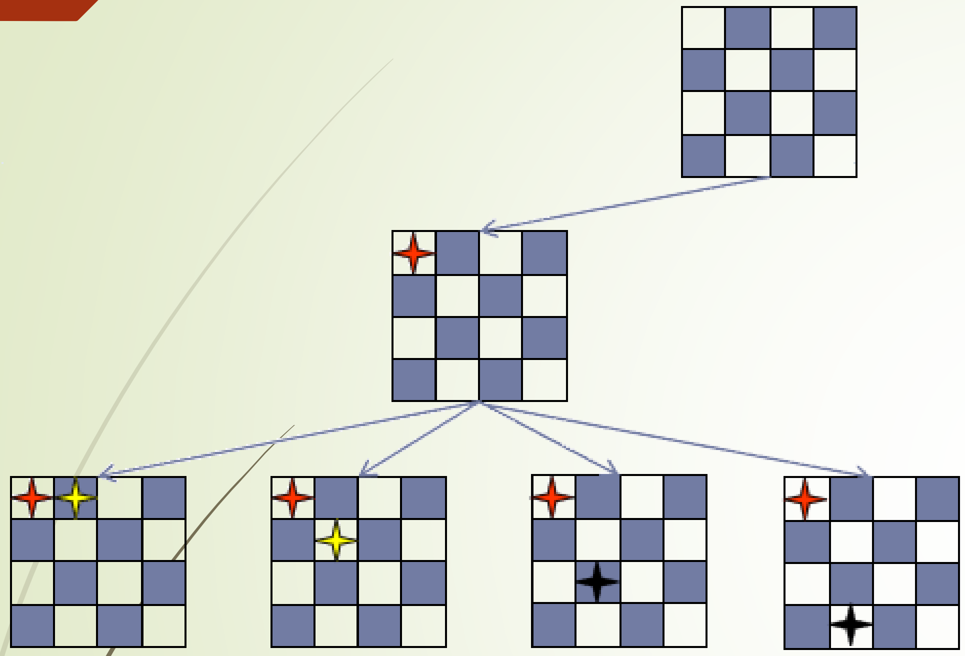
جستجوی عقب‌گرد–مثال ۴ وزیر



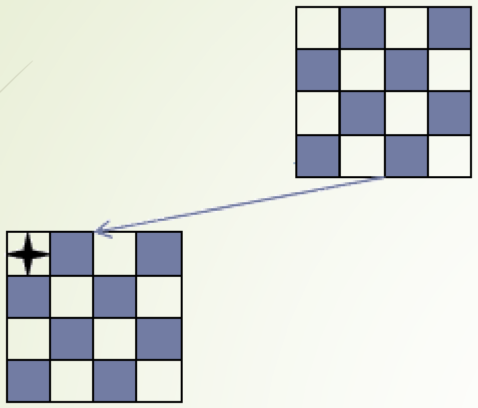
جستجوی عقب‌گرد–مثال ۴ وزیر



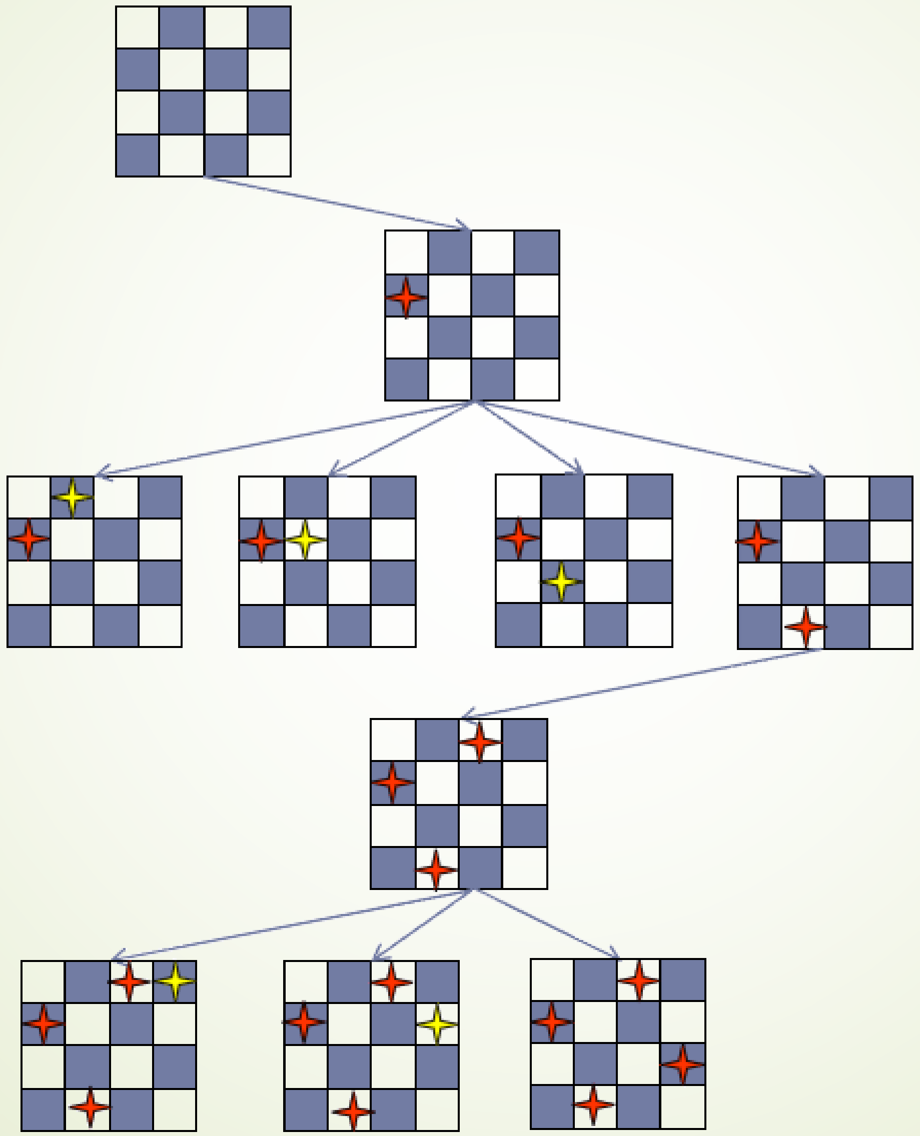
جستجوی عقب‌گرد–مثال ۴ وزیر



جستجوی عقب‌گرد–مثال ۴ وزیر



جستجوی عقب‌گرد–مثال ۴ وزیر



الگوریتم جستجوی عقب‌گرد

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

function BACKTRACK(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment  then
           add {var = value} to assignment
           inferences ← INFERENCE(csp, var, value)
           if inferences ≠  failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result ≠ failure then
                 return result
       remove {var = value} and inferences from assignment
   return failure
```

بهبود کارایی جستجوی عقب‌گرد

- جستجوی عقب‌گرد یک الگوریتم ناآگاهانه است و برای مسائل بزرگ کارآمد نیست.
- با پاسخ به سوالات زیر می‌توان به توابع هیوریستیک همه‌منظوره‌ای دست یافت که به جستجوی عقب‌گرد کمک می‌کنند.

۱- در مرحله‌ی بعد، کدام متغیر باید مقداردهی شود؟ (SELECT-UNASSIGNED-VARIABLE)

۲- مقادیر متغیر انتخابی باید به چه ترتیبی امتحان شوند؟ (ORDER-DOMAIN-VALUES)

۳- چه استنتاج‌هایی باید در هر گام در جستجو انجام شود؟ (INFERENCE)

۴- آیا می توان شکست‌های حتمی را زودتر تشخیص داد؟

- فرض کنید در عقبگرد در مسأله ۸- وزیر، ۶ وزیر اول به گونه‌ای قرار گرفته‌اند که قرار دادن هشتمین وزیر را غیر ممکن می‌سازند. عقب‌گرد تمام مکان‌های ممکن برای وزیر هفتم را چک می‌کند، اگرچه مسأله غیر قابل حل است.

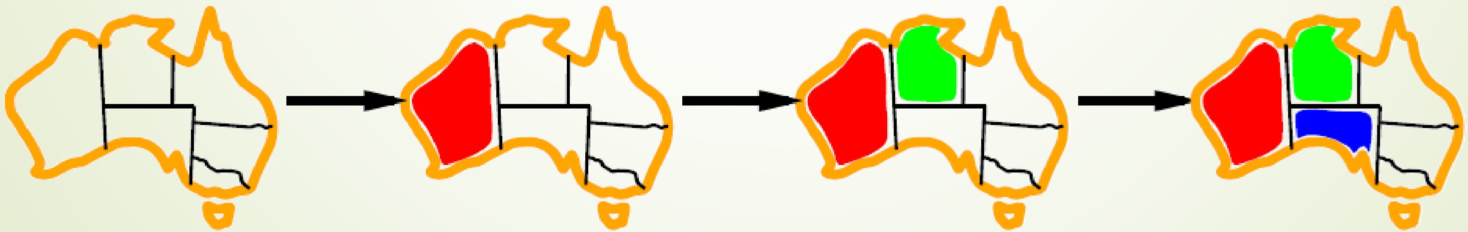
هیوریستیک MRV ”متغیر با کم‌ترین مقدار باقی‌مانده“

❖ هیوریستیک (MRV (Minimum Remaining Value: انتخاب متغیری با کم‌ترین تعداد مقادیر مجاز

❖ نام‌های دیگر: ”متغیر با بیشترین محدودیت“ یا ”اول شکست“

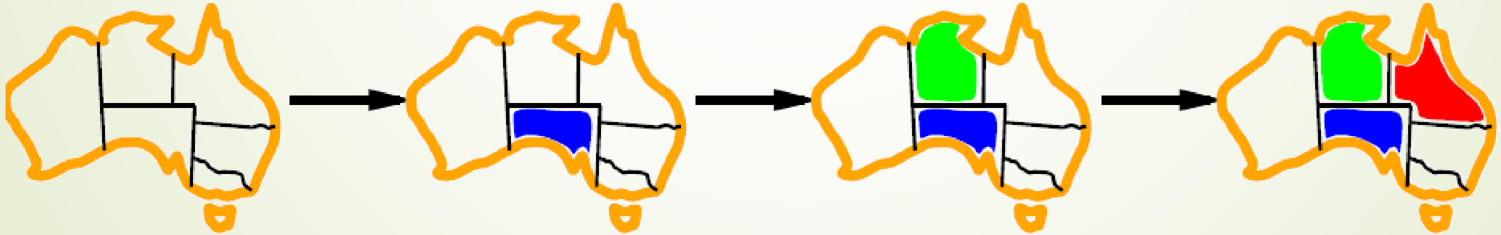
❖ هرس درخت جستجو: متغیری که بیشترین احتمال شکست در مسیر را دارد اول برمی‌گزینند و بدین وسیله درخت جستجو هرس می‌شود.

❖ کشف سریع شکست: اگر در جریان کار متغیر x بدون مقدار مجاز باقی ماند MRV متغیر x را برمی‌گزیند و فورا شکست در جستجو تشخیص داده خواهد شد.



هیوریستیک درجه (degree)

- اولین ناحیه برای رنگ‌آمیزی نقشه کدام باشد بهتر است؟ آیا هیوریستیک MRV در انتخاب آن به ما کمک می‌کند؟
- هیوریستیک درجه: انتخاب متغیری که در تعداد بیشتری از محدودیت‌های متغیرهای بدون انتساب نقش دارد
  - کاربرد: گره‌گشایی برای MRV زمانی‌که تعداد مقادیر مجاز برای چندین متغیر یکسان باشد
  - منجر به کاهش ضریب انشعاب در متغیرهای بعدی می‌شود.

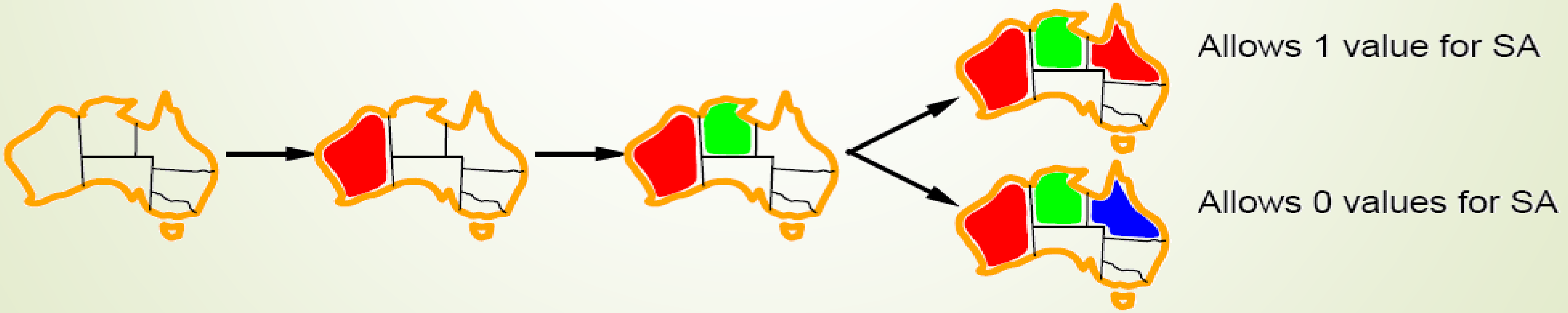


هیوریستیک LCV "مقدار با حداقل محدودیت"

❖ هیوریستیک LCV (Least Constraining Value): مقداری را انتخاب می‌کند که کم‌ترین محدودیت را روی مقادیر معتبر متغیرهای بدون انتساب ایجاد کند.

❖ این هیوریستیک سعی می‌کند بیشترین قابلیت انعطاف را برای انتساب‌های بعدی متغیرها فراهم آورد. به این ترتیب امکان رسیدن به یک انتساب نامعتبر را کم می‌کند و درنتیجه احتمال عقب‌گرد کردن را کم‌تر می‌کند و درنتیجه سریع‌تر به جواب می‌رسیم.

❖ مناسب برای زمانی که فقط به دنبال یک راه‌حل هستیم نه تمام راه‌حل‌ها

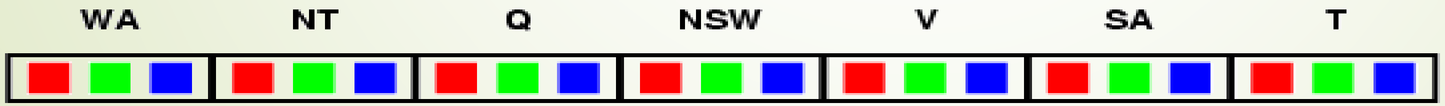


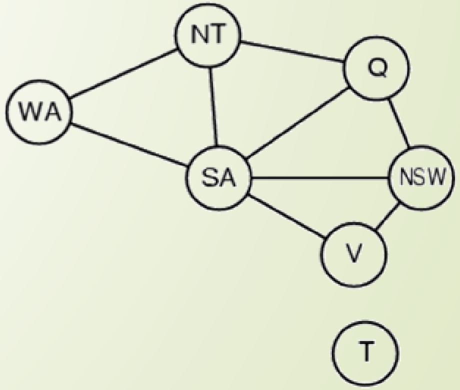
جایگاه استنتاج (inference) در مسائل CSP

- استنتاج به‌عنوان یک مرحله‌ی پیش پردازش
  - الگوریتم سازگاری یال AC-3
    - حذف مقادیر از دامنه‌های تمام متغیرها تا آن‌جا که تمام زوج متغیرها سازگار کمان شوند.
- استنتاج در خلال جستجو
  - وارسی روبه‌جلو (Forward checking)
    - هنگام انتخاب یک مقدار برای یک متغیر، کاهش‌های جدید دامنه بر روی متغیرهای بدون انتساب همسایه را استنتاج می‌کند.

وارسی روبه‌جلو

- هرگاه یک مقدار به متغیر X انتساب داده شد وارسی پیش‌رو سازگاری یال را بر روی آن اجرا می‌کند.
  - برای هر متغیر انتساب‌نیافته‌ی Y که توسط یک محدودیت با X در ارتباط است، هر مقدار ناسازگار با مقدار انتخاب‌شده برای X را از دامنه‌ی Y حذف می‌کند.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده‌ای نداشته باشد، جستجو پایان می‌یابد.



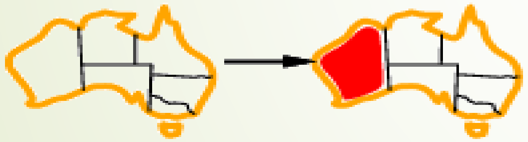


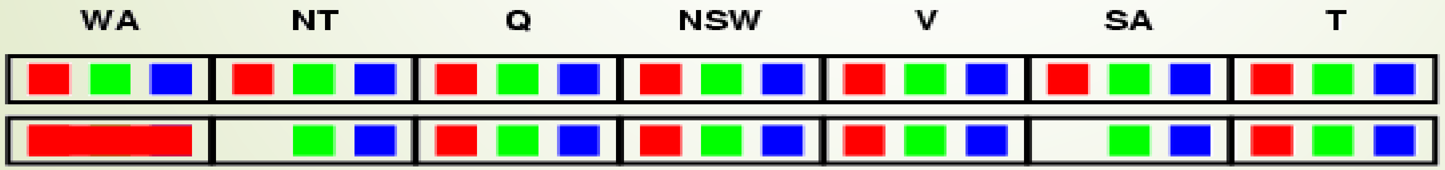
وارسی روبه‌جلو

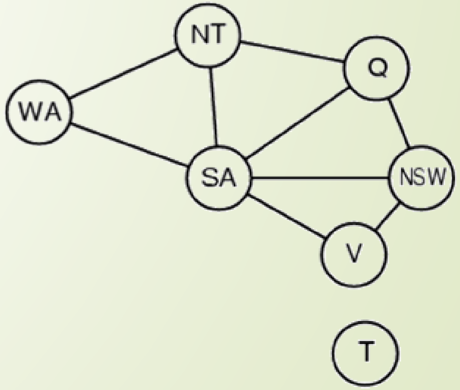
❖ هرگاه یک مقدار به متغیر X انتساب داده شد وارسی پیش‌رو سازگاری یال را بر روی آن اجرا می‌کند.

❖ برای هر متغیر انتساب‌نیافته‌ی Y که توسط یک محدودیت با X در ارتباط است، هر مقدار ناسازگار با مقدار انتخاب‌شده برای X را از دامنه‌ی Y حذف می‌کند.

❖ زمانی که یک متغیر هیچ مقدار مجاز باقیمانده‌ای نداشته باشد، جستجو پایان می‌یابد.

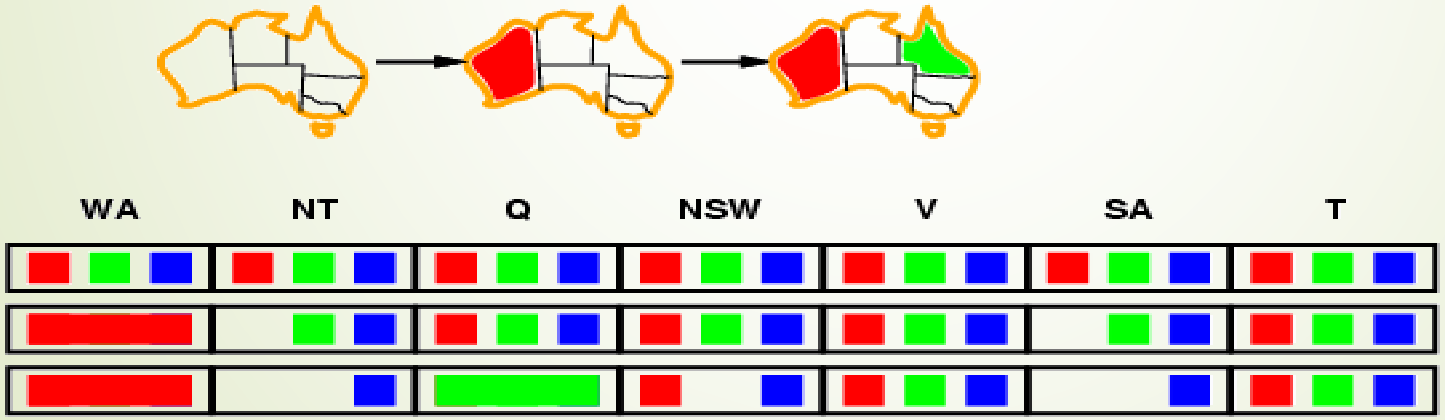


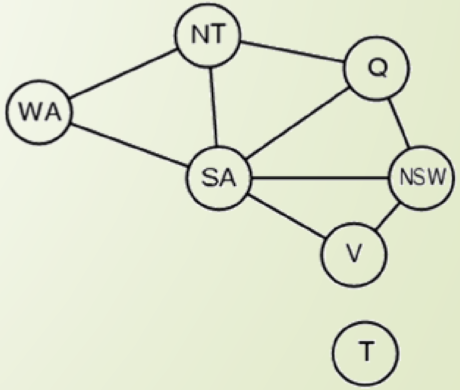




وارسی روبه‌جلو

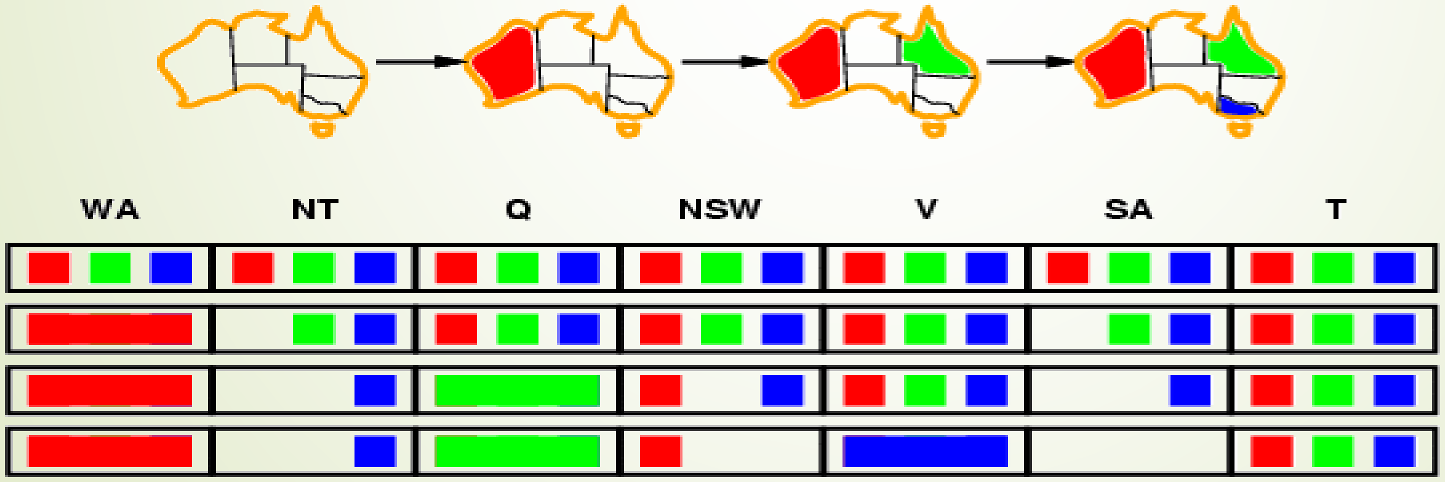
- هرگاه یک مقدار به متغیر X انتساب داده شد وارسی پیش‌رو سازگاری یال را بر روی آن اجرا می‌کند.
  - برای هر متغیر انتساب‌نیافته‌ی Y که توسط یک محدودیت با X در ارتباط است، هر مقدار ناسازگار با مقدار انتخاب‌شده برای X را از دامنه‌ی Y حذف می‌کند.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده‌ای نداشته باشد، جستجو پایان می‌یابد.

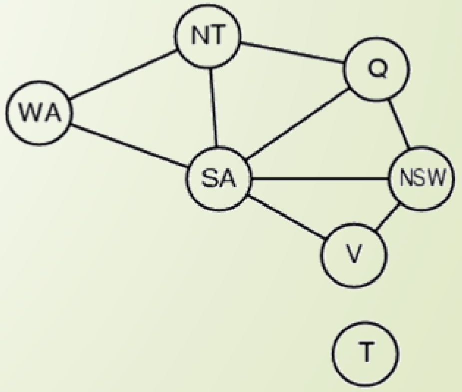




وارسی روبه‌جلو

- هرگاه یک مقدار به متغیر X انتساب داده شد وارسی پیش‌رو سازگاری یال را بر روی آن اجرا می‌کند.
  - برای هر متغیر انتساب‌نیافته‌ی Y که توسط یک محدودیت با X در ارتباط است، هر مقدار ناسازگار با مقدار انتخاب‌شده برای X را از دامنه‌ی Y حذف می‌کند.
  - زمانی که یک متغیر هیچ مقدار مجاز باقیمانده‌ای نداشته باشد، جستجو پایان می‌یابد.

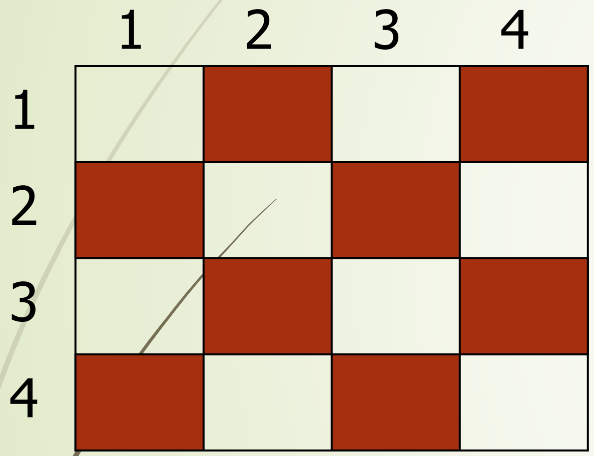


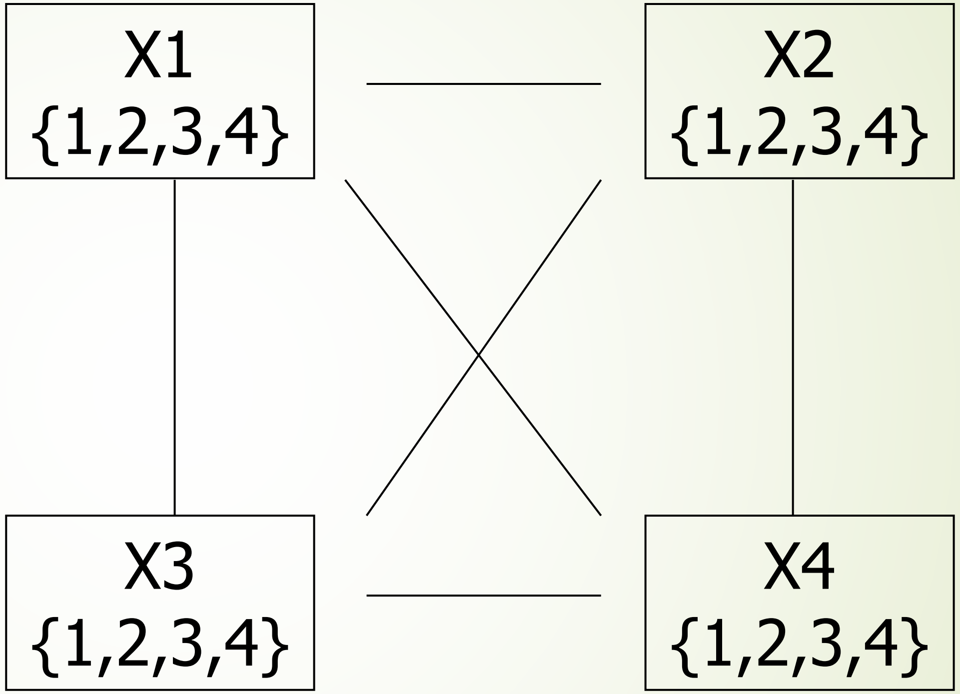


وارسی روبه‌جلو ...

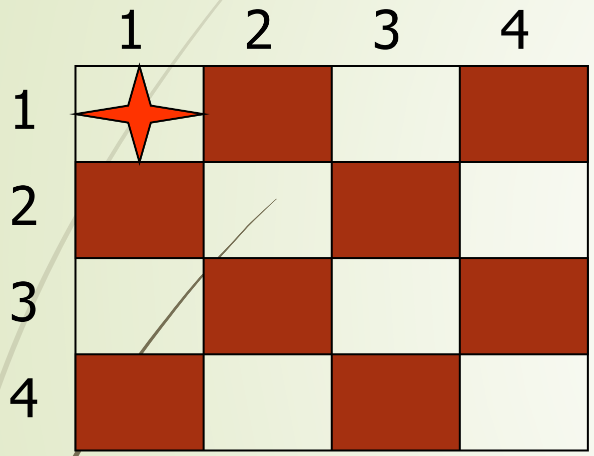
- وارسی روبه‌جلو راجع به انتخاب متغیر بعدی برای مقداردهی یا انتخاب مقدار برای متغیر نظری نمی‌دهد، فقط می‌تواند وقوع تضاد در آینده را تشخیص دهد.
  - در واقع بعد از هر مقداردهی به یک متغیر، وارسی روبه‌جلو را انجام می‌دهیم تا بررسی کنیم مقداردهی اخیر منجر به بن‌بست در آینده می‌شود یا خیر.
- برای بسیاری مسائل ترکیب MRV و وارسی روبه‌جلو کارآمدتر خواهد بود.
  - وارسی روبه‌جلو یک راه کارآمد برای محاسبه افزایشی اطلاعاتی است که MRV نیاز دارد.

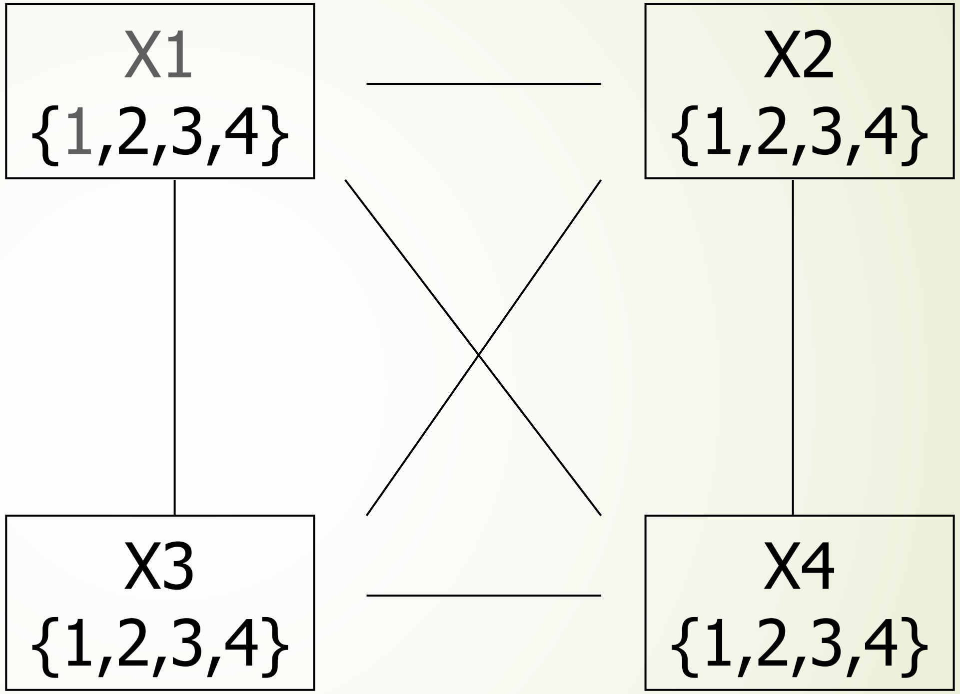
مسئله ۴ وزیر (FC+MRV)



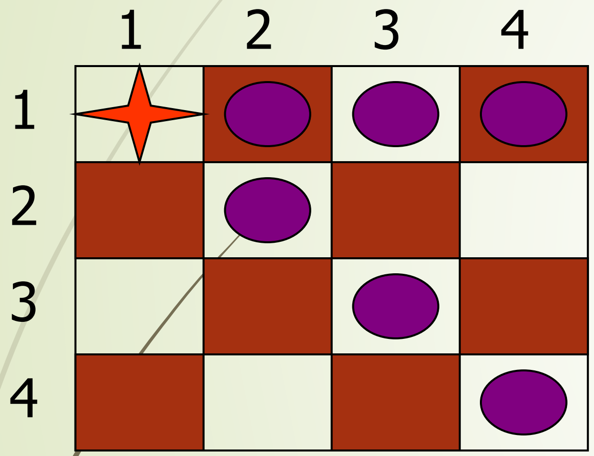


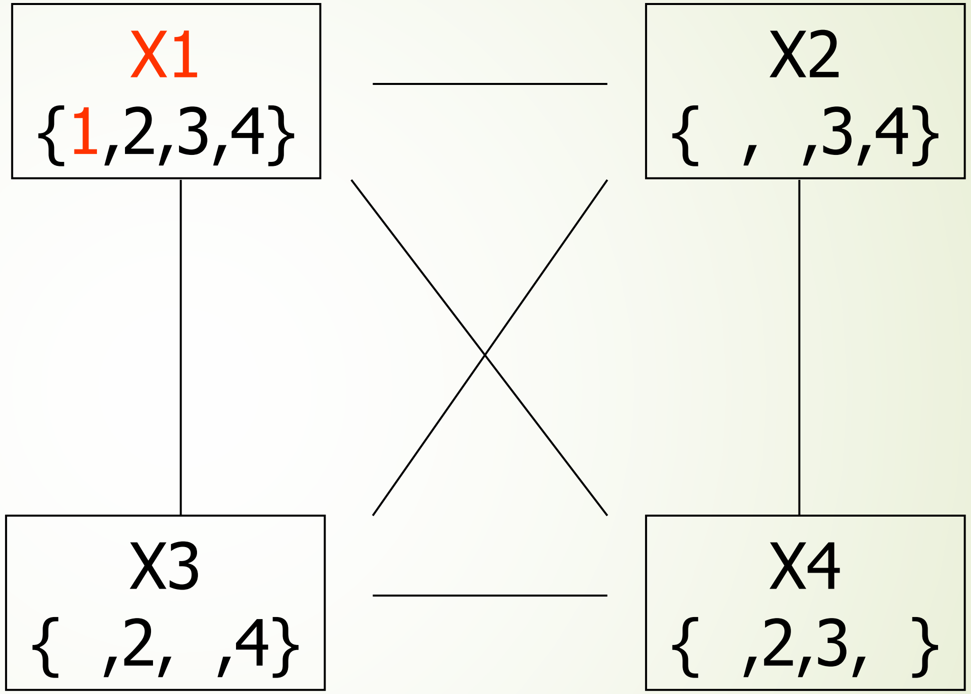
مسئله ۴ وزیر (FC+MRV)



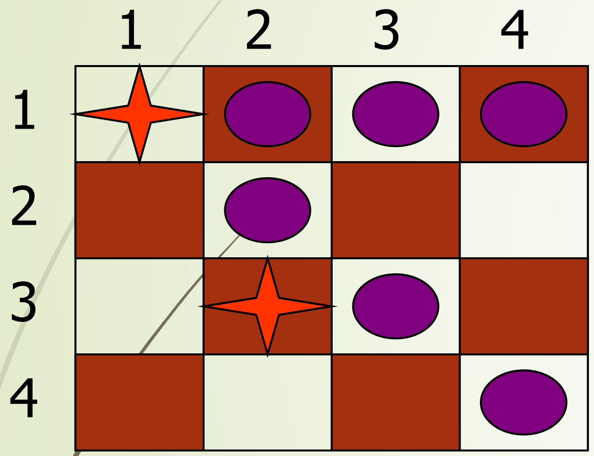


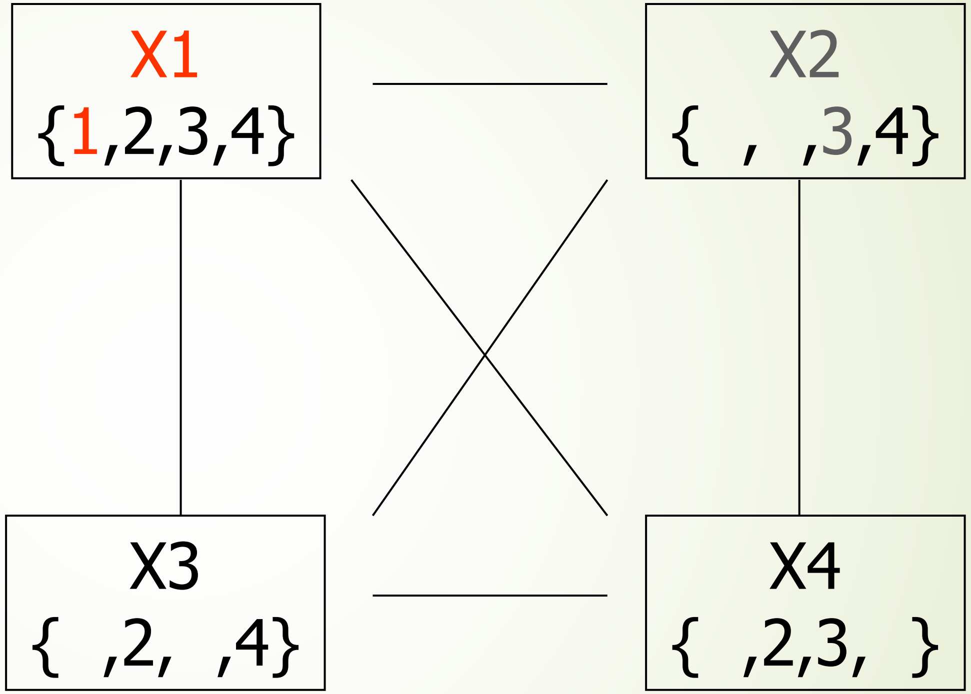
مسئله ۴ وزیر (FC+MRV)



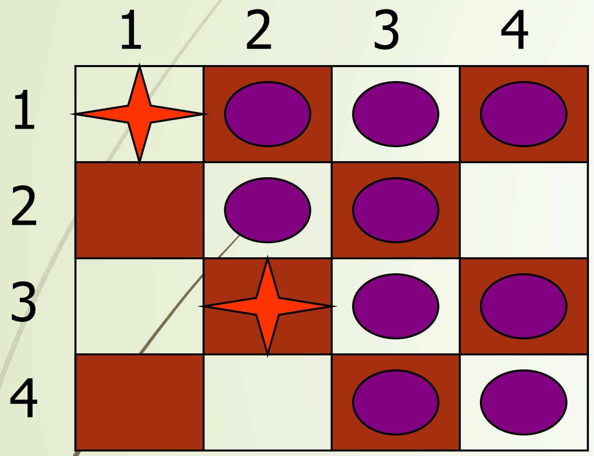


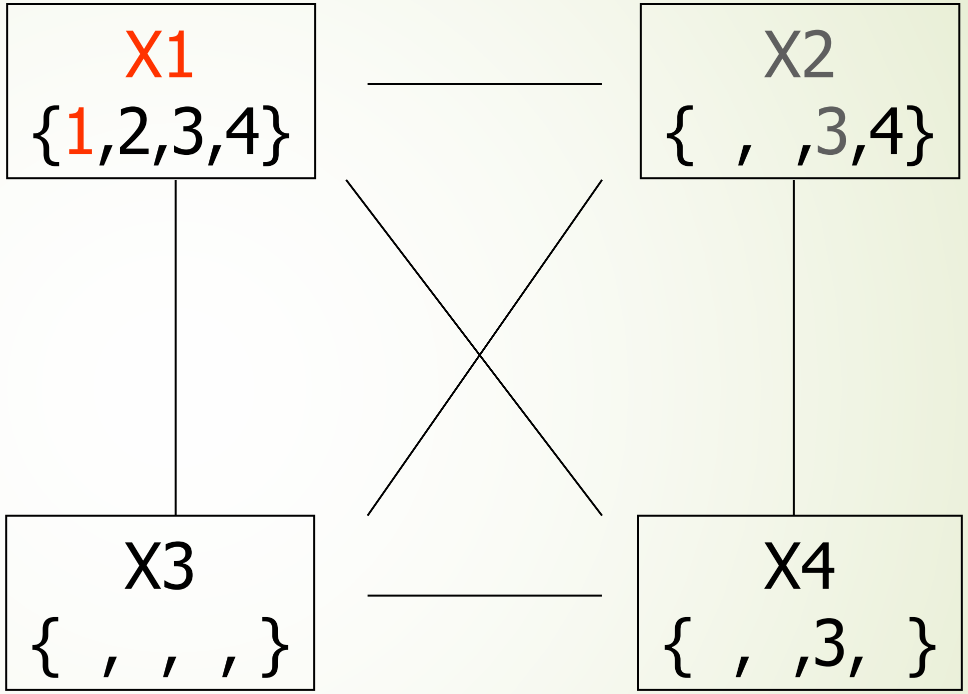
مسئله ۴ وزیر (FC+MRV)



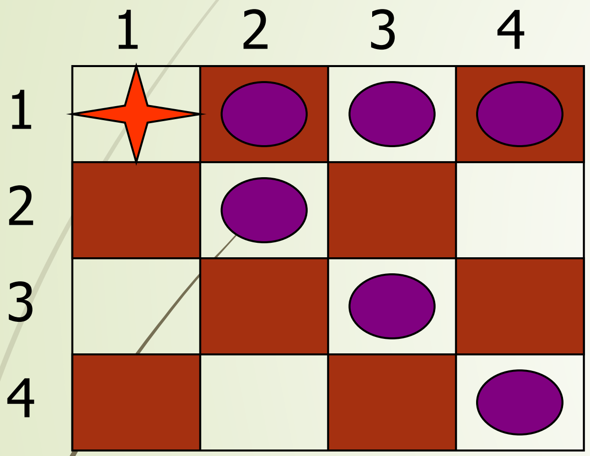


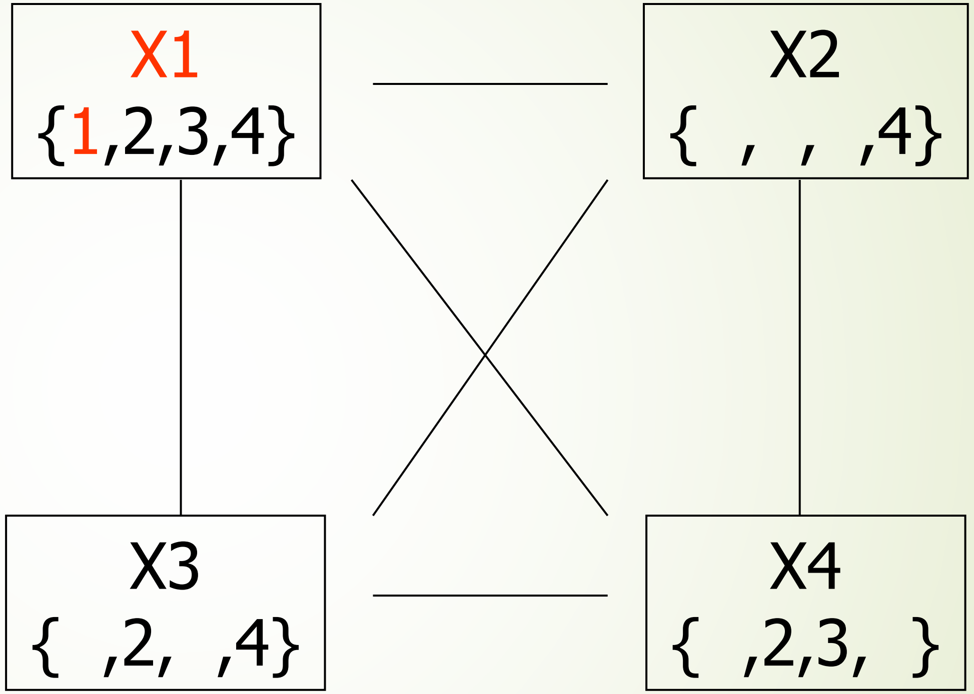
مسئله ۴ وزیر (FC+MRV)



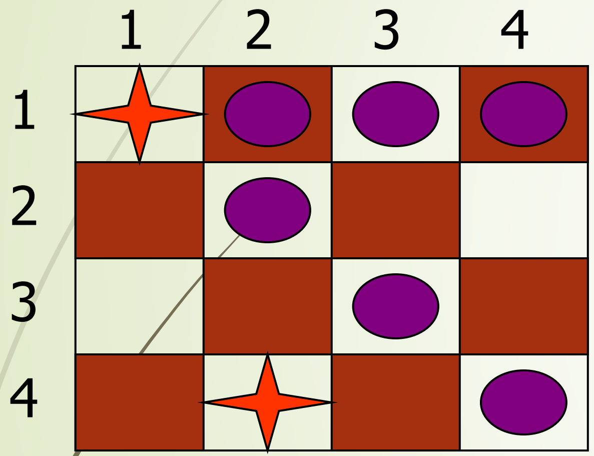


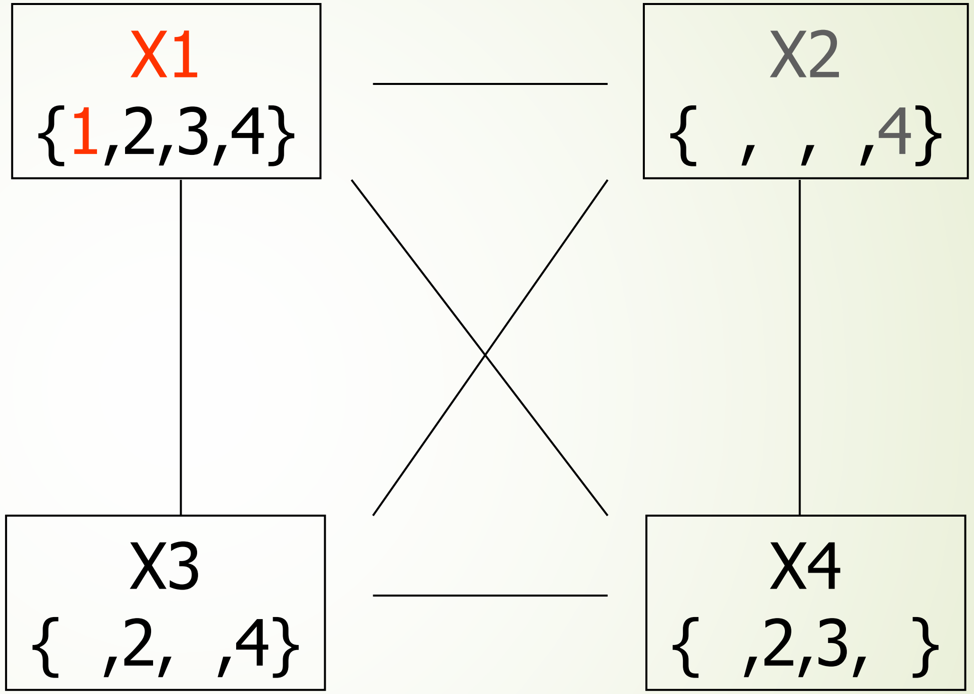
مسئله ۴ وزیر (FC+MRV)



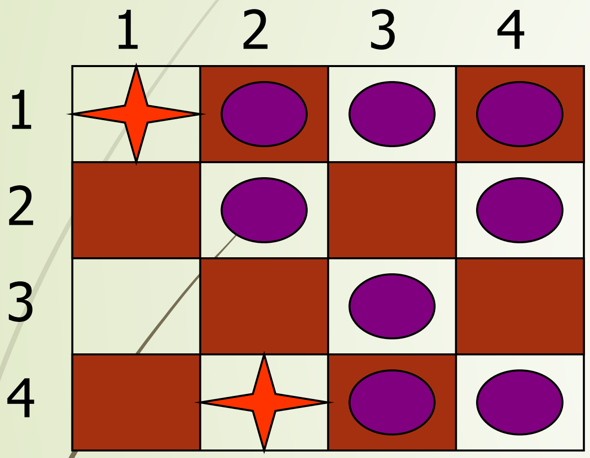


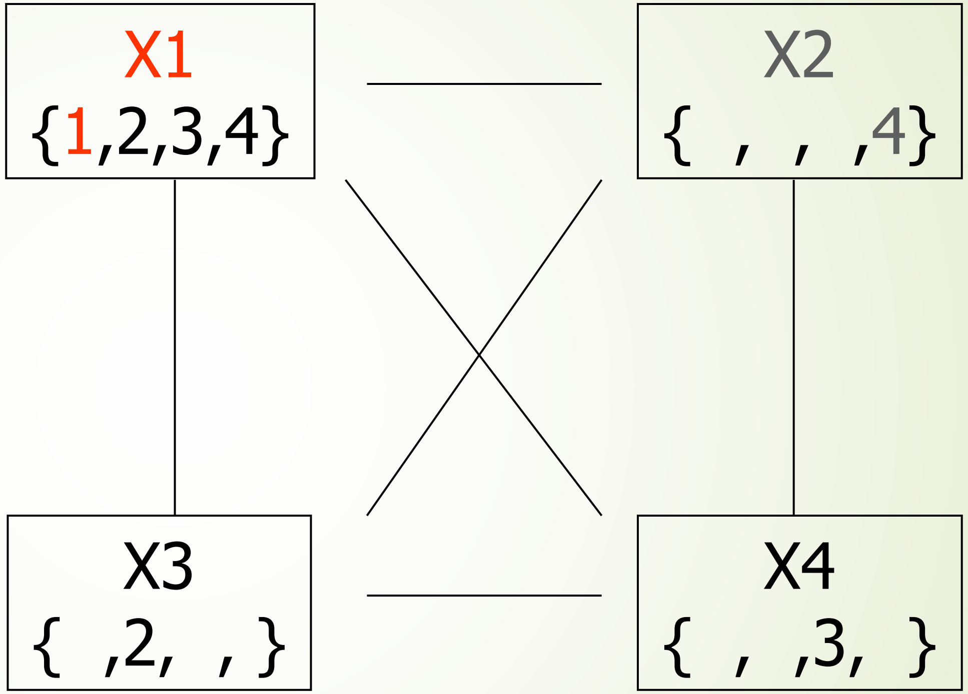
مسئله ۴ وزیر (FC+MRV)



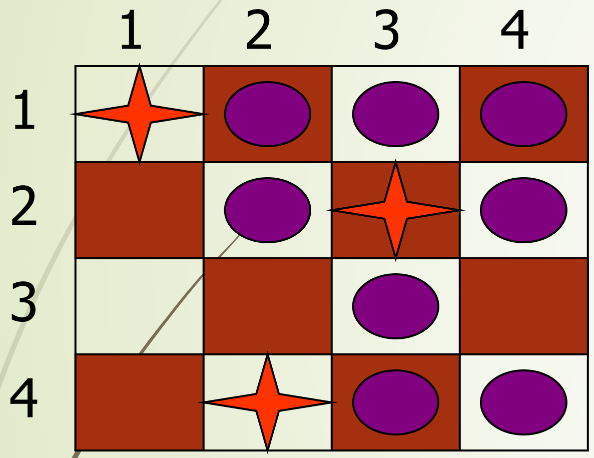


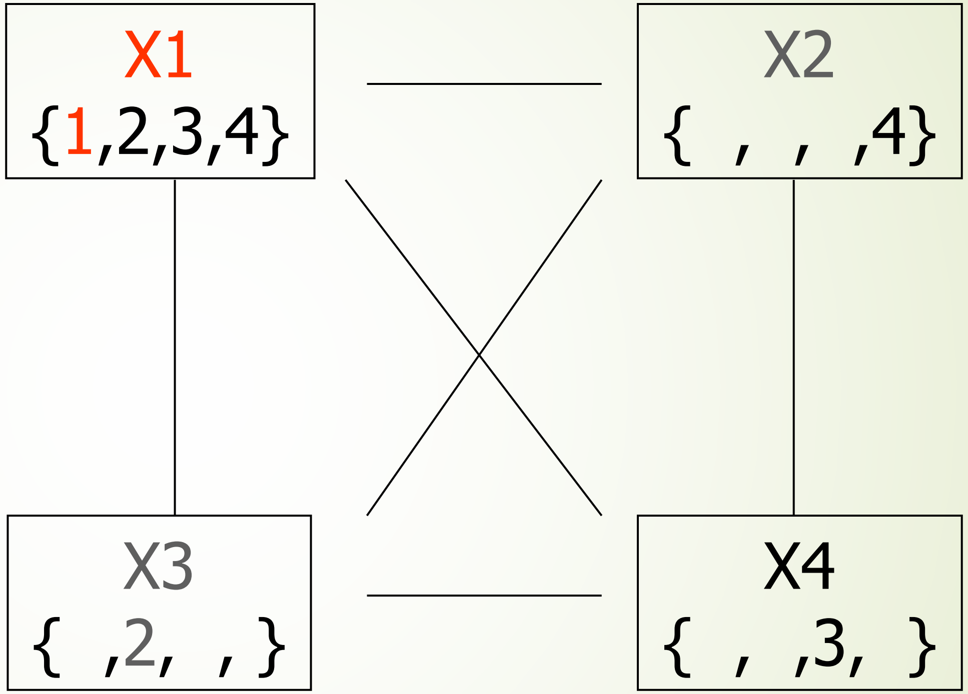
مسئله ۴ وزیر (FC+MRV)



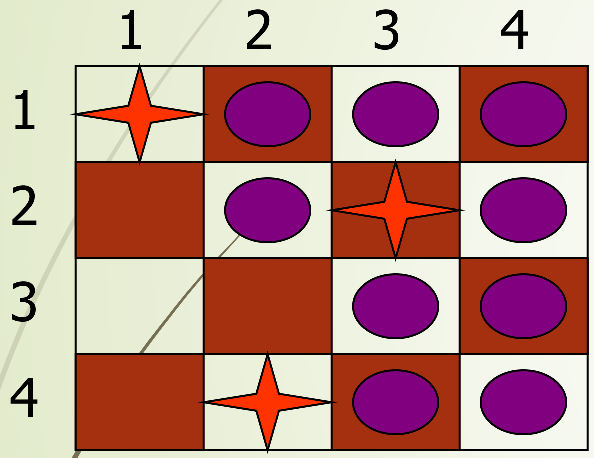


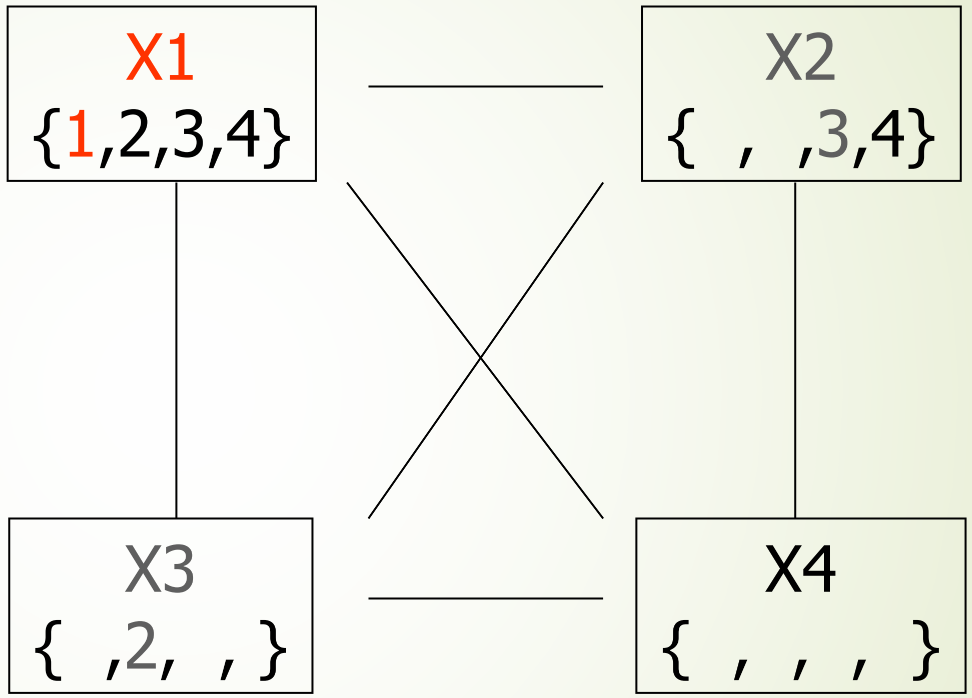
مسئله ۴ وزیر (FC+MRV)





مسئله ۴ وزیر (FC+MRV)





عقب‌گرد ساده

- در الگوریتم BACKTRACKING-SEARCH هنگامی که یکی از شاخه‌ها با شکست در جستجو مواجه می‌شود، الگوریتم به متغیر قبلی باز می‌گردد و مقداری جدید برای آن در نظر می‌گیرد.
  - مثال: فرض کنید ترتیب رنگ ناحیه‌ها به‌صورت زیر باشد

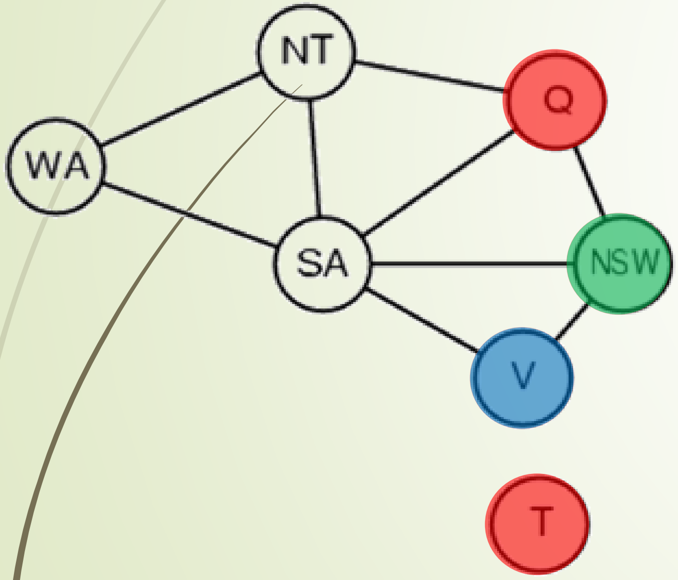
Q=red, NSW=green, V=blue, T=red

و فرض کنید متغیر بعدی برای مقداردهی SA باشد.

چه مقداری را می‌توان برای SA برگزید؟ هیچ مقداری موجود نیست.

متغیری که به آن برگشت انجام می‌شود کدام است؟ T

آیا انتخاب این متغیر برای برگشت کمکی می‌کند؟ خیر



پرش رو به عقب

- یک روش عقب‌گرد هوشمندانه‌تر آن است که تمام مسیر را تا رسیدن به مجموعه‌ای از متغیرها که باعث شکست شده‌اند (مجموعه تناقض)، به عقب باز گردیم.
  - مجموعه تناقض (Conflict set) برای متغیر X عبارت است از مجموعه‌ای از متغیرهایی که قبلاً مقداردهی شده‌اند و به واسطهٔ یک محدودیت با X در ارتباطند.

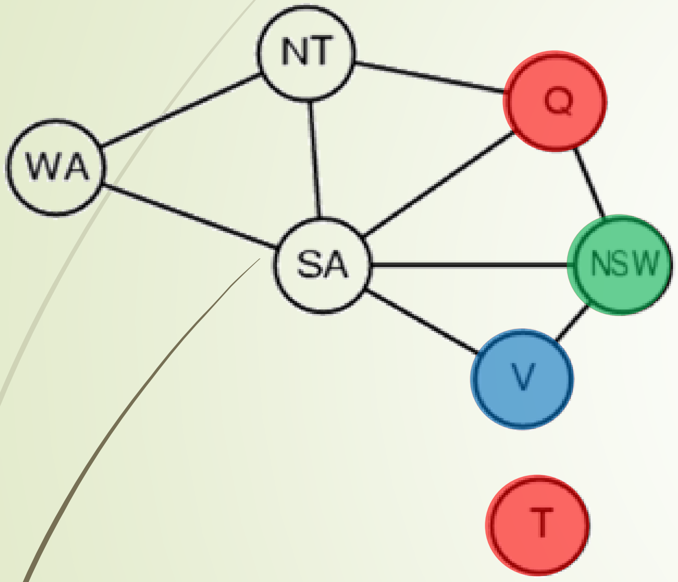
- روش پرش رو به عقب (Back jumping)
  - قبل از مقداردهی به یک متغیر: "مجموعه تناقض" را برای آن متغیر محاسبه و ذخیره کن
  - اگر تمام مقادیر متغیر فعلی X با انتساب‌های قبلی ناسازگار باشد: مسیر پیموده‌شده را تا رسیدن به آخرین متغیری که در مجموعهٔ تناقض مقداردهی شده است، به عقب برو و مقدار دیگری از آن متغیر را بررسی کن.

پرش رو به عقب ...

- مثال: ترتیب رنگ ناحیه‌ها را مانند مثال قبل فرض کنید

Q=red, NSW=green, V=blue, T=red

مجموعه تناقض را قبل از مقداردهی هر متغیر ذکر می‌کنیم



Conf(Q)={}
Conf(NSW)={Q}
Conf(V)={NSW}
Conf(T)={}
Conf(SA)={Q,NSW,V}

مقدار سازگاری برای SA وجود ندارد پس به متغیر V برمی‌گردیم.

پرش رو به عقب ...

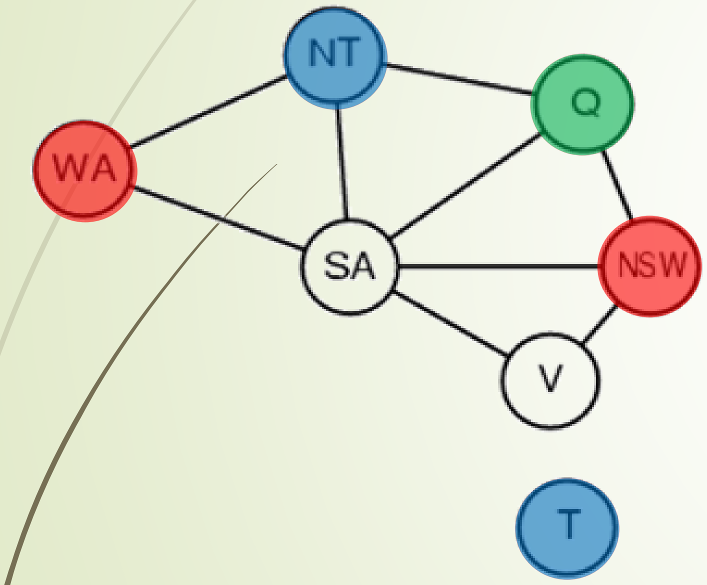
- پرش رو به عقب هنگامی رخ می‌دهد که هر مقدار متعلق به دامنه با انتساب انجام شده دارای تناقض باشد. وارسی رو به جلو این حالت را از قبل تشخیص می‌دهد و از رسیدن به آن جلوگیری می‌کند.
  - هر شاخه‌ای که به وسیلهٔ پرش رو به عقب هرس می‌شود، می‌تواند به وسیله انجام وارسی رو به جلو نیز هرس شود.

پرش رو به عقب با هدایت برخورد

❖ مثال: ترتیب رنگ ناحیه‌ها را به صورت زیر فرض کنید

WA=red, NSW=red, T=blue, NT=blue, Q=green, SA=?

مجموعه تناقض را قبل از مقداردهی هر متغیر ذکر می‌کنیم



Conf(WA)={}
Conf(NSW)={}
Conf(T)={}
Conf(NT)={WA}
Conf(Q)={NSW, NT}
Conf(SA)={WA, NSW, NT, Q}

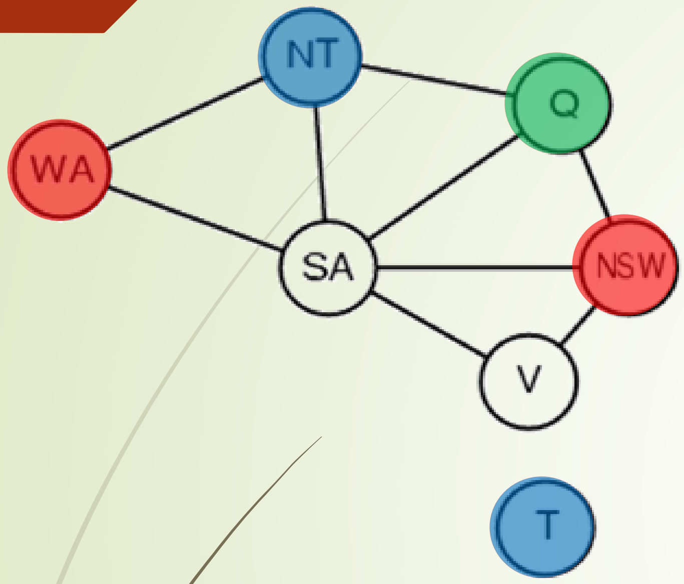
مقدار سازگاری برای SA وجود ندارد پس به متغیر Q برمی‌گردیم. اما راهگشا نیست.

پرش رو به عقب با هدایت برخورد ...

- تعریف دقیق مجموعه برخورد: مجموعه برخورد متغیر x عبارت است از مجموعه متغیرهایی که قبل از x مقدار گرفته‌اند و باعث می‌شوند متغیر x به همراه متغیرهای بعدی فاقد راه‌حل سازگار باشد.
- پرش رو به عقب با هدایت برخورد (Conflict-directed Backjumping)
  - در نظر بگیرید که X_j متغیر فعلی و $conf(X_j)$ مجموعهٔ تناقض آن باشد.
  - اگر تمام مقادیر ممکن برای X_j با شکست مواجه شوند، الگوریتم به X_i برخواهد گشت که آخرین متغیر مقدار داده شده و متعلق به مجموعهٔ تناقض X_j می‌باشد و انتساب زیر را انجام می‌دهیم:

$$conf\ (X_i) \leftarrow conf\ (X_i) \cup conf\ (X_j) - \{X_i\}$$

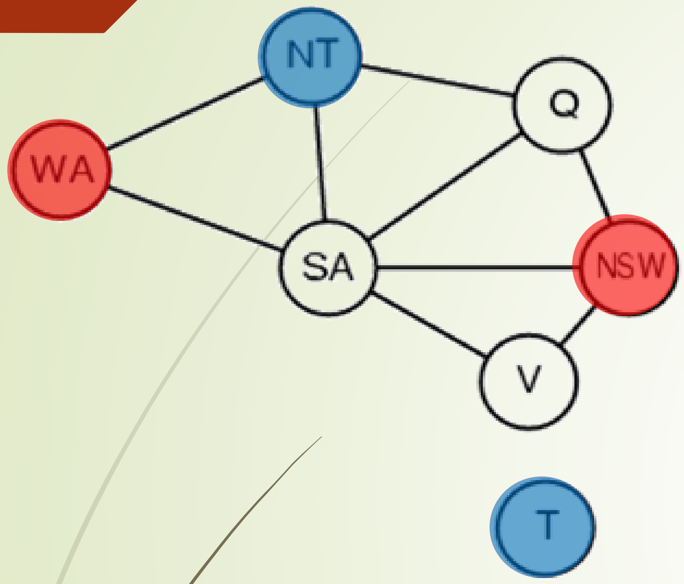
پرش رو به عقب با هدایت برخورد ...



Conf(WA)={}
Conf(NSW)={}
Conf(T)={}
Conf(NT)={WA}
Conf(Q)=~~{NSW, NT}~~ → {WA, NSW, NT}
Conf(SA)={WA, NSW, NT, Q}

- دامنه‌ی SA خالی می‌شود
- SA به جدیدترین متغیر در Conf(SA) یعنی Q پرش می‌کند.
- به‌روز کردن Conf(Q)=Conf(Q) ∪ Conf(SA)-{Q}={WA,NSW,NT}
- یعنی هیچ راه‌حل سازگاری با انتساب‌های فعلی WA, NSW, NT از Q=green به بعد وجود ندارد.

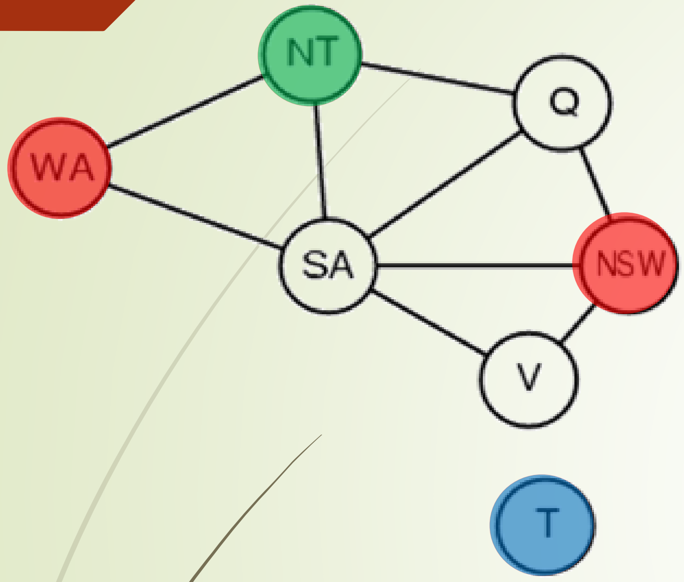
پرش رو به عقب با هدایت برخورد ...



Conf(WA)={}
Conf(NSW)={}
Conf(T)={}
Conf(NT)=~~{WA}~~ → {WA, NSW}
Conf(Q)={WA, NSW, NT}
Conf(SA)={WA, NSW, NT, Q}

- دامنه‌ی Q خالی می‌شود
- Q به جدیدترین متغیر در Conf(Q) یعنی NT پرش می‌کند.
  - به‌روز کردن $Conf(NT)=Conf(NT) \cup Conf(Q)-\{NT\}=\{WA,NSW\}$
  - یعنی هیچ راه‌حل سازگاری با انتساب‌های فعلی WA, NSW از NT=blue به بعد وجود ندارد.

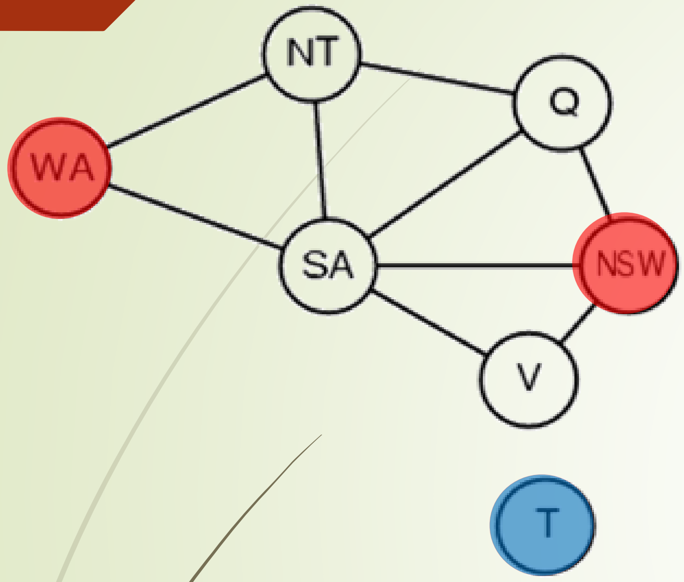
پرش رو به عقب با هدایت برخورد ...



Conf(WA)={}
Conf(NSW)={}
Conf(T)={}
Conf(NT)={WA, NSW}
Conf(Q)={WA, NSW, NT}
Conf(SA)={WA, NSW, NT, Q}

- مقدار NT=green را امتحان می‌کنیم. اما باز هم متوجه می‌شویم مقدار سازگاری برای Q و SA یافت نمی‌شود.
  - یعنی هیچ راه‌حل سازگاری با انتساب‌های فعلی WA, NSW از NT=green به بعد وجود ندارد.
- دامنه‌ی NT خالی می‌شود

پرش رو به عقب با هدایت برخورد ...



Conf(WA)={}
Conf(NSW)=~~{}~~ → {WA}
Conf(T)={}
Conf(NT)={WA, NSW}
Conf(Q)={WA, NSW, NT}
Conf(SA)={WA, NSW, NT, Q}

- NT به جدیدترین متغیر در Conf(NT) یعنی NSW پرش می‌کند.
  - به‌روز کردن Conf(NSW)=Conf(NSW) ∪ Conf(NT)-{NSW}={WA}
  - ارتباط میان NSW و WA کشف می‌شود، بنابراین NSW=green امتحان می‌شود و ...
  - ...
  - چقدر بهتر از عقبگرد ساده (یا با جهش رو به عقب ساده) است؟

جستجوی کامل (محلی) برای CSPها

حل CSPها با الگوریتم‌های جستجوی محلی

- در فرمولهسازی CSP بهصورت یک مسئله جستجو، مسیر اهمیتی ندارد بنابراین میتوانیم از فرمولهسازی حالت کامل استفاده کنیم.
  - حالت اولیه: انتساب مقادیر به تمام متغیرها (مثلا بهصورت تصادفی)
  - اقدامها: انتساب یک مقدار جدید به یکی از متغیرها
  - تابع هیوریستیک $h(s)$: تعداد محدودیتهای نقضشده
  - کمینه سراسری: $h(s)=0$

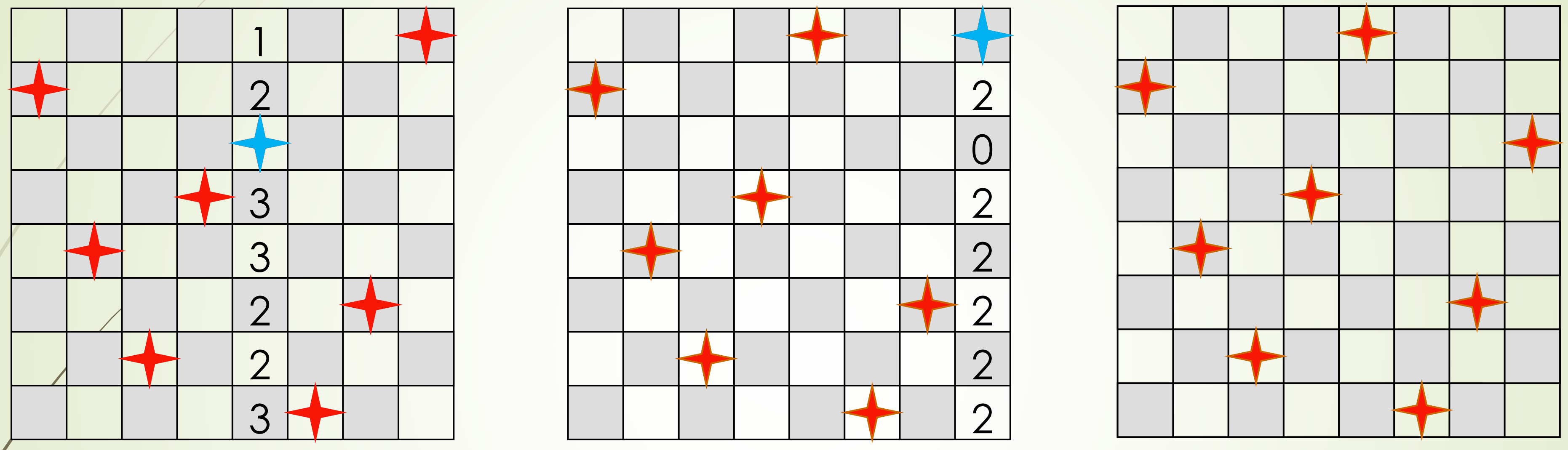
الگوریتم Min-Conflicts

- ❖ هیوریستیک min-conflicts
  - ❖ مقداری را انتخاب کن که کمترین تعداد محدودیت‌ها را نقض کند،
  - ❖ یعنی، تپه نوردی با هیوریستیک " تعداد کل محدودیت‌های نقض شده "

```
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 csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
   return failure
```

حل ۸–وزیر با فرمول‌سازی کامل CSP



- پس از جای‌گذاری اولیه مهره‌ها، این روش حتی با در نظر گرفتن تعداد یک میلیون وزیر، مسئله را به طور متوسط در ۵۰ مرحله حل می‌کند.
- از آن‌جا که در مسئله n وزیر، حالت‌های هدف در فضای حالت توزیع شده‌اند، روش جستجوی محلی درمورد آن بسیار مفید واقع می‌شود.
  - می‌توان گفت در نزدیکی هر حالت اولیه یک حالت هدف وجود دارد.