مدرس: دکتر فاطمه منصوری

نیم‌سال اول ۱۴۰۲-۱۴۰۳

هوش مصنوعی

تهیه کننده: فاطمه حسین زاده

These lecture notes are heavily based on notes written for the Artificial Intelligence Course at Berkley University

مسائل ارضای محدودیت[1]

در یادداشت قبلی، نحوه پیدا کردن راه حل های بهینه برای مسائل جست و جو، نوعی مسئله برنامه ریزی را آموختیم .اکنون، ما در مورد حل یک کلاس مرتبط از مسائل، مسائل ارضای محدودیت (CSP) یاد خواهیم گرفت .برخلاف مسائل جستجو، CSPها نوعی مسئله شناسایی هستند، مسائلی که در آنها باید به سادگی تشخیص دهیم که آیا یک حالت یک حالت هدف است یا خیر، بدون توجه به اینکه چگونه به آن هدف می رسیم. CSP ها با سه عامل تعریف می شوند:

1. متغیرها: CSP ها دارای مجموعه ای از n متغیر $X_1, ..., X_n$ هستند که هر کدام می توانند یک مقدار مشخص از مجموعه ای از مقادیر تعریف شده را اتخاذ کنند.
2. دامنه: یک مجموعه $\{x_1, ..., x_d\}$ که نشان دهنده تمام مقادیر ممکنی است که یک متغیر CSP می تواند اتخاذ کند.
3. محدودیت ها: یکسری محدودیت را بر روی مقادیر متغیرها، به طور بالقوه با توجه به سایر متغیرها تعریف می کنند.

مسئله n-وریز را در نظر بگیرید: با توجه به صفحه شطرنج N×N ، آیا می توانیم حالتی را پیدا کنیم که در آن n وزیر را روی تخته قرار دهیم به طوری که هیچ دو وزیر یکدیگر را تهدید نکنند؟

ما می توانیم این مسئله را به صورت یک مسئله CSP فرمول‌بندی کنیم:

1. متغیرها: X_{ij} که $0 \le i,j < N$. هر X_{ij} نشان دهنده یک موقعیت در صفحه شطرنج N× N ما است، که i و j به ترتیب عدد ردیف و ستون را مشخص می کنند.
2. دامنه: {0,1}. هر X_{ij} می تواند مقدار ۰ یا ۱ را به خود بگیرد، یک مقدار بولین که نشان دهنده وجود یک وزیر در موقعیت (i,j) روی صفحه است.
3. محدودیت ها:
   - $\forall i,j,k\ (X_{ij},X_{ik}) \in \{(0,0), (0,1), (1,0)\}$. این محدودیت بیان می کند که اگر دو متغیر مقدار یکسانی برای i داشته باشند، تنها یکی از آنها می تواند مقدار ۱ را به خود بگیرد. این محدودیت این شرایط را در بر می گیرد که هیچ دو وزیری نمی توانند در یک ردیف باشند.

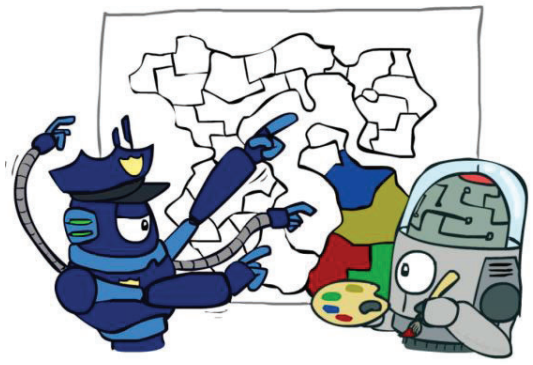
[1] Constraint Satisfaction Problems

- $\forall i,j,k\ (X_{ij},X_{kj}) \in \{(0,0), (0,1), (1,0)\}$. تقریباً مانند محدودیت قبلی، این محدودیت بیان می‌کند که اگر دو متغیر برای j مقدار یکسانی داشته باشند، تنها یکی از آنها می‌تواند مقدار ۱ را اخذ کند، و این شرط را در بر می‌گیرد که هیچ دو وزیری نمی‌توانند در یک ستون باشند.
- $\forall i,j,k\ (X_{ij},X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$. با استدلال مشابه بالا، می‌توانیم ببینیم که این محدودیت و محدودیت بعدی شرایطی را نشان می‌دهند که هیچ دو وزیری نمی‌توانند به ترتیب در مورب‌های اصلی یا فرعی یکسانی قرار بگیرند.
- $\forall i,j,k\ (X_{ij},X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$.
- $\sum_{i,j} Xij = N$. این محدودیت بیان می کند که ما باید دقیقاً N موقعیت داشته باشیم که با ۱ مشخص شده اند و بقیه موقعیت ها با ۰ علامت گذاری شده اند، که این شرط را نشان می دهد که دقیقاً N وزیر روی صفحه وجود دارد.

مسائل ارضای محدودیت NP-hard هستند، که به وضوح به این معنی است که هیچ الگوریتم شناخته شده ای برای یافتن راه حل برای آنها در زمان چند جمله ای وجود ندارد. اگر مسئله ای با N متغیر با دامنه اندازه O(d) برای هر متغیر داشته باشیم، $O(d^N)$ حالت برای بررسی وجود دارد که با توجه به تعداد متغیرها نمایی است .ما اغلب می‌توانیم با فرمول‌بندیCSP ها به‌عنوان مسائل جستجو، تعریف وضعیت‌ها به‌عنوان حالت های جزئی (تخصیص متغیر بهCSP ها که در آن به برخی از متغیرها مقادیری اختصاص داده شده است، در حالی که به برخی دیگر ارزش اختصاص داده نشده‌است)، این مشکل را حل کنیم . به همین ترتیب، تابع جانشین برای یک وضعیتCSP ، همه حالت ها را با یک متغیر جدید اختصاص داده شده، به عنوان خروجی می دهد، و آزمون هدف تأیید می کند که همه متغیرها تخصیص داده شده باشند و همه محدودیت ها در حالتی که در حال آزمایش هستند، ارضا می شوند .مسائل ارضای محدودیت ساختار قابل توجهی نسبت به مسائل جستجوی سنتی دارند و ما می‌توانیم از این ساختار با ترکیب فرمول بالا و با اکتشافی‌های مناسب برای بررسی راه‌حل‌ها در مدت زمان ممکن استفاده کنیم.

گراف های محدودیت[2]

اکنون یک مثال دیگر از CSP را معرفی می کنیم: رنگ‌آمیزی نقشه .رنگ‌آمیزی نقشه مسئله ای را حل می‌کند که در آن مجموعه‌ای از رنگ‌ها به ما داده می‌شود و باید نقشه‌ای را طوری رنگ کنیم که هیچ دو ایالت یا منطقه مجاور هم رنگ نباشند.



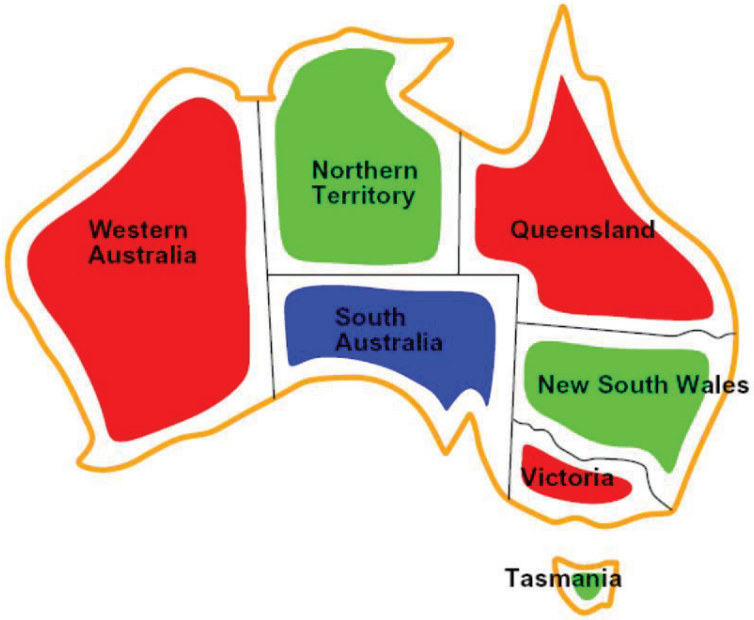
مسائل ارضای محدودیت اغلب به صورت گراف های محدودیت نشان داده می شوند، جایی که گره ها متغیرها را نشان می دهند و یال ها محدودیت های بین آنها را نشان می دهند .انواع مختلفی از محدودیت ها وجود دارد، و هر یک کمی متفاوت است:

- محدودیت های یگانه[3] محدودیت های یگانه شامل یک متغیر منفرد در CSP هستند. آنها در گراف های محدودیت نشان داده نمی شوند، در عوض برای هرس دامنه متغیری که در صورت لزوم محدود می کنند استفاده می شوند.

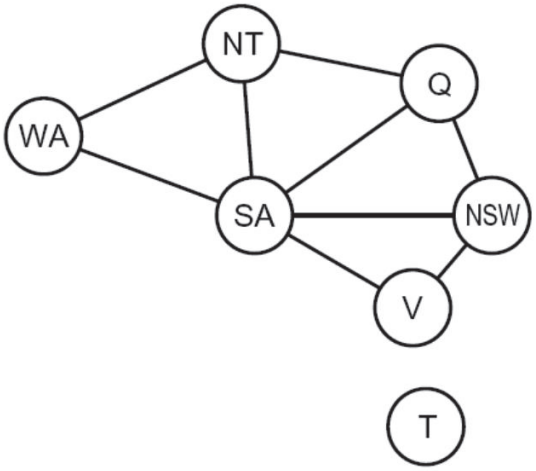
[2] Constraint Graphs
[3] Unary Constraints

- محدودیت های دودویی[4]: محدودیت های دودویی شامل دو متغیر هستند .آنها در گراف های محدودیت به عنوان یال های گراف سنتی نشان داده می شوند.
- محدودیت‌های مرتبه بالاتر[5]: محدودیت‌های شامل سه یا چند متغیر نیز می‌توانند با یال‌هایی در یک گراف CSP نمایش داده شوند، فقط کمی غیر متعارف به نظر می‌رسند.

رنگ آمیزی نقشه استرالیا را در نظر بگیرید:



محدودیت ها در این مسئله عبارت است از اینکه هیچ دو حالت مجاور نمی توانند همرنگ باشند .در نتیجه، با کشیدن یک یال بین هر جفت ایالت که مجاور یکدیگر هستند، می‌توانیم گراف محدودیت را برای رنگ‌آمیزی نقشه استرالیا به صورت زیر ایجاد کنیم:



خوبی گراف های محدودیت این است که می توانیم از آنها برای استخراج اطلاعات مفید در مورد ساختار CSP هایی که در حال حل آنها هستیم استفاده کنیم .با تجزیه و تحلیل گراف یکCSP ، می‌توانیم مواردی را در مورد آن مشخص کنیم، مانند اینکه آیا به صورت

[4] Binary Constraints
[5] Higher-order Constraints

پراکنده یا متراکم متصل/محدود است و آیا ساختار درختی دارد یا خیر .ما این موضوع را با جزئیات بیشتر در مورد حل مسائل ارضای محدودیت بحث خواهیم کرد.

حل مسائل ارضای محدودیت

مسائل ارضای محدودیت به طور سنتی با استفاده از یک الگوریتم جستجو به نام جستجوی عقب گرد[6]حل می شوند .جستجوی عقب گرد یک بهینه سازی در جستجوی اول عمق[7]است که به طور خاص برای مسئله ارضای محدودیت استفاده می شود، با بهبودهایی که از دو اصل اصلی زیر حاصل می شوند:

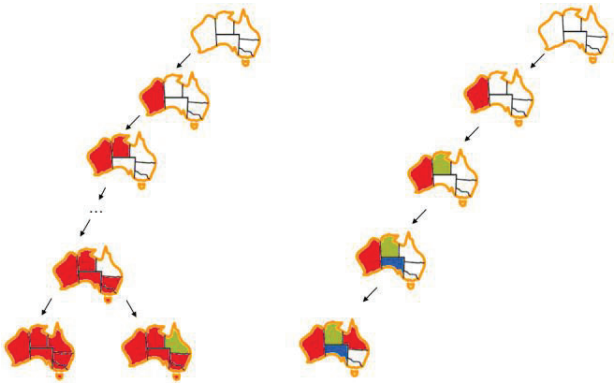
1. یک روش برای مرتب کردن متغیرها انتخاب کنید و مقادیر متغیرها را براساس آن انتخاب کنید .از آنجا که انتساب ها جابجایی هستند(به عنوان مثال اگر قرار دهیم WA=قرمز، NT=سبز، برابر این است که اول قرار دهیم NT=سبز و WA=قرمز)، این یک گزینه معتبر است.
2. هنگام انتخاب مقادیر یک متغیر، فقط مقادیری را انتخاب کنید که با هیچ یک از مقادیر تخصیص داده شده قبلی مغایرت نداشته باشند .اگر چنین مقادیری وجود نداشت، به متغیر قبلی برگردید و مقدار آن را تغییر دهید.

شبه کد زیر چگونگی کارکرد الگوریتم بازگشتی جستجوی عقب گرد را نشان می دهد:

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/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 given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```

برای تجسم نحوه عملکرد این فرآیند، درختان جستجوی جزئی را هم برای جستجوی اول عمق و هم برای جستجوی عقبگرد در رنگ آمیزی نقشه در نظر بگیرید:



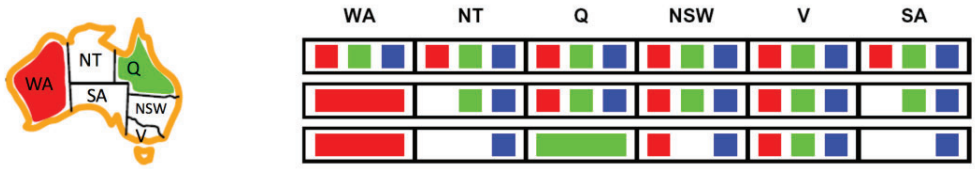
توجه کنید که چگونه DFS متأسفانه همه چیز را قبل از اینکه نیازی به تغییر باشد، قرمز رنگ می کند، و حتی در این صورت در مسیر درست به سمت راه حل حرکت نمی کند .از سوی دیگر، جستجوی عقب گرد تنها زمانی به یک متغیر مقداری اختصاص می‌دهد که آن مقدار هیچ محدودیتی را نقض نکند، که منجر به عقبگردی بسیار کمتری می‌شود .اگرچه جستجوی عقبگرد یک

[6] Backtracking search
[7] Depth first search

پیشرفت بزرگ نسبت به روش بررسی تمام حالات[8]جستجوی اول عمق است، ما می‌توانیم با بهبودهای بیشتر از طریق فیلتر کردن، مرتب‌سازی متغیرها/مقادیر، و بهره‌برداری ساختاری، به سرعت بهتری برسیم.

فیلتر کردن:

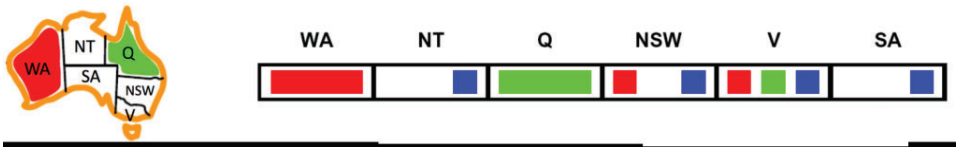
اولین پیشرفتی که در عملکرد CSP در نظر خواهیم گرفت، فیلتر کردن است، که بررسی می‌کند آیا می‌توانیم دامنه‌های متغیرهای اخذ نشده را با حذف مقادیری که می‌دانیم منجر به عقب‌نشینی می‌شود، زودتر از موعد هرس کنیم یا نه .یک روش ساده برای فیلتر کردن، بررسی رو به جلو است، که هرگاه مقداری به یک متغیر X_i اخذ شود، دامنه‌های متغیرهای اخذ نشده ای را که محدودیتی اشتراکی با X_i دارند که در صورت اتخاذ آن مقادیر، محدودیت را نقض می‌کند، هرس می‌کند .هر زمان که یک متغیر جدید اخذ می شود، می توانیم بررسی رو به جلو را اجرا کنیم و دامنه های متغیرهای اخذ نشده مجاور متغیر جدید اخذ شده در گراف محدودیت را هرس کنیم .مثال رنگ آمیزی نقشه را با متغیرهای اخذ نشده و مقادیر بالقوه آنها در نظر بگیرید:



توجه کنید که چگونه با اخذ قرمز برای WA و سپس سبز برای Q، اندازه دامنه هایNT ، NSW، و SA (حالت های مجاورWA ، Qیا هر دو) با حذف مقادیر کاهش می یابد .ایده بررسی رو به جلو را می توان به اصل سازگاری کمان[9]تعمیم داد .برای سازگاری کمان، هر یال بدون جهت گراف محدودیت را برای یک CSP به عنوان دو یال جهت دار که در جهت مخالف قرار دارند تفسیر می کنیم .به هر یک از این یال های جهت دار کمان می گویند .الگوریتم سازگاری کمان به صورت زیر عمل می کند:

- با ذخیره کردن تمام کمان ها در گراف محدودیت برای CSP در یک صف Q شروع کنید.
- به طور مکرر کمان ها را از Q حذف کنید و این شرط را اعمال کنید که در هر کمان حذف شده$X_i \rightarrow X_j$ ، برای هر مقدار باقیمانده v برای متغیر دم X_i، حداقل یک مقدار w باقی مانده برای متغیرسر X_j وجود دارد به طوری که$X_i = v$ ، $X_j = w$ هیچ محدودیتی را نقض نکند .اگر مقداری v برای Xi با هیچ یک از مقادیر باقیمانده برای X_j کار نمی کند، v را از مجموعه مقادیر ممکن برای X_i حذف می کنیم.
- اگر حداقل یک مقدار برای X_i هنگام اعمال الگوریتم برای یک کمان $X_i \rightarrow X_j$ حذف شود، کمان هایی از شکل $X_k \rightarrow X_i$ را به Q، برای همه متغیرهای اخذ نشده X_k اضافه کنید .اگر در این مرحله یک قوس$X_k \rightarrow X_i$ قبلاً در Q باشد، نیازی به افزودن مجدد آن نیست.
- الگوریتم را تا جایی که Q خالی شود، یا دامنه برخی از متغیر ها خالی شود منجر به عقب گرد شود ادامه دهید.

الگوریتم سازگاری کمان معمولاً شهودی ترین الگوریتم نیست، بنابراین بیایید یک مثال سریع با رنگ‌آمیزی نقشه را مرور کنیم:



---

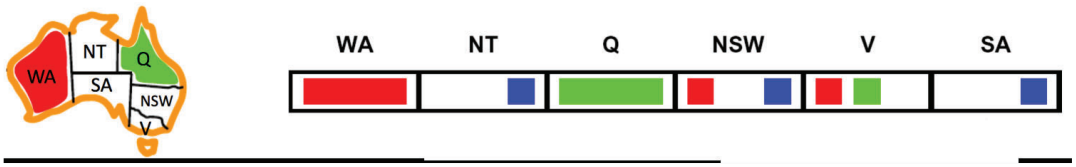
[8] Brute-force

[9] ***الگوریتم سازگاری کمان*** که سازگاری قوس یا سازگاری گنبدی نیز ترجمه شده است .الگوریتمی برای حل مسائل ارضای محدودیت می‌باشد .در این روش سازگاری حالت‌ها با کمان نشان داده شده و سعی می‌شود با حذف تدریجی مقادیر ناسازگار جواب مناسب را پیدا کرد .اگر X و Y دو متغیر در یک مسئله ارضای محدودیت باشند آنگاه X ← Y سازگار کمان است، اگر و فقط اگر به ازای تمامی مقادیر x از X ، مقداری مجاز برای Y موجود باشد.
https://fa.wikipedia.org/wiki/%D8%A7%D9%84%DA%AF%D9%88%D8%B1%DB%8C%D8%AA%D9%85_%D8%B3%D8%A7%D8%B2%DA%AF%D8%A7%D8%B1%DB%8C_%DA%A9%D9%85%D8%A7%D9%86

ما با اضافه کردن تمام کمان ها بین متغیرهای اخذ نشده که یک محدودیت را به یک صف Q به اشتراک می گذارند، شروع می کنیم، پس خواهیم داشت:

$$Q = [SA \rightarrow V, V \rightarrow SA, SA \rightarrow NSW, NSW \rightarrow SA, SA \rightarrow NT, NT \rightarrow SA, V \rightarrow NSW, NSW \rightarrow V]$$

برای اولین کمان ما، SA→V، می‌بینیم که برای هر مقدار در دامنهSA ، {آبی}، حداقل یک مقدار در دامنهV ، {قرمز، سبز، آبی} وجود دارد که هیچ محدودیتی را نقض نمی‌کند، و بنابراین نیازی به حذف مقادیری نیست . دامنهSA . با این حال، برای کمان بعدی ماSA→ V ، اگر V را آبی قرار بدهیم، می بینیم که SA هیچ مقدار باقیمانده ای نخواهد داشت که هیچ محدودیتی را نقض نمی کند، و بنابراین رنگ آبی را از دامنه V هرس می کنیم.



از آنجا که ما یک مقدار را از دامنه V حذف کردیم، باید تمام کمان ها را با V در سر قرار دهیمV→SA - ، NSW →V. از آنجایی که NSW → V در حال حاضر در Q است، ما فقط باید SA → V را اضافه کنیم، و صف به روز رسانی می کنیم.

$$Q = [SA \rightarrow NSW, NSW \rightarrow SA, SA \rightarrow NT, NT \rightarrow SA, V \rightarrow NSW, NSW \rightarrow V, SA \rightarrow V]$$

می‌توانیم این فرآیند را تا زمانی ادامه دهیم که در نهایت کمان SA→NT را از Q حذف کنیم. اعمال سازگاری کمان در این کمان، رنگ آبی را از دامنه SA حذف می‌کند، آن را خالی می‌گذارد و باعث ایجاد یک عقب گرد می‌شود .توجه داشته باشید که کمان SA → NSW قبل از NT → SA در Q ظاهر می شود و اجرای یکنواخت در این کمان، آبی را از دامنه NSW حذف می کند.

سازگاری کمان معمولاً با الگوریتم AC-3 (الگوریتم سازگاری کمان #۳)[1] پیاده‌سازی می‌شود که شبه کد آن به شرح زیر است:

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do                    tail      head
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue

function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

[1] Arc Consistency Algorithm #3

الگوریتم AC-3 دارای بدترین پیچیدگی زمانی $O(ed^3)$ است، که e تعداد کمان ها (یال های جهت دار) و d اندازه بزرگترین دامنه است .به طور کلی، سازگاری کمان نسبت به بررسی رو به جلو، یک تکنیک هرس دامنه جامع‌تر است و به عقب‌نشینی کمتری منجر می‌شود، اما برای اجرا به محاسبات بسیار بیشتری نیاز دارد .بر این اساس، مهم است که هنگام تصمیم گیری در مورد اینکه کدام تکنیک فیلترینگ را برای CSP که می خواهید حل کنید، این مسئله را در نظر بگیرید.

به عنوان یک بخش جالب یادداشت در مورد سازگاری، سازگاری کمان زیرمجموعه ای از یک مفهوم کلی تر از سازگاری است که به عنوانk -سازگاری[1]شناخته می شود، که وقتی اجرا می شود تضمین می کند که برای هر مجموعه ای از k گره ها درCSP ، یک انتساب ثابت به هر زیر مجموعه ای از گره های k-1 تضمین می کند که گرهk ام حداقل یک مقدار ثابت خواهد داشت .این ایده را می توان از طریق ایده k-سازگاری قوی[2]بیشتر گسترش داد .گرافی که k-سازگار قوی است، دارای این ویژگی است که هر زیر مجموعه ای از k گره ها نه تنها با k سازگار هستند، بلکه همچنین $k-1, k-2, \ldots, 1$ نیز سازگار هستند. جای تعجب نیست که تحمیل درجه بالاتری از سازگاری در CSP هزینه محاسباتی بیشتری دارد .تحت این تعریف تعمیم یافته برای سازگاری، می توانیم ببینیم که سازگاری کمان معادل ۲-سازگاری[3]است.

[1] k-consistency
[1] strong k-consistency
[1] 2-consistency

مرتب‌سازی

در حل یک مسئلهCSP، ما تعیین می‌کنیم که در هر مرحله از روند، کدام متغیر و مقدار باید ثابت شود. در عمل، زمانی‌که برنامه در حال اجرا است، محاسبه تغییر بعدی و مقدار متناظر معمولاً بسیار موثرتر است. این کار با دو اصل عمده، یعنی "حداقل مقادیر باقیمانده" و "حداقل محدودیت مقدار" انجام می‌شود.

-حداقل مقادیر باقی‌مانده:[1] هنگام انتخاب متغیر بعدی برای اختصاص، استفاده از سیاست MRV به معنای انتخاب یک متغیر از بین متغیرهای تخصیص نیافته‌ای است که کمترین مقدار باقی‌مانده معتبر را دارد (متغیری که بیشترین محدودیت را دارد). این انتخاب از نظر شهودی نشان می‌دهد که متغیر با بیشترین محدودیت احتمالاً از مقادیر ممکن خارج می‌شود و در صورت عدم اختصاص، منجر به بازگشت به عقب می‌شود. بنابراین، بهتر است به زودی به آن مقداری اختصاص داده شود تا به تأخیر بیفتد.

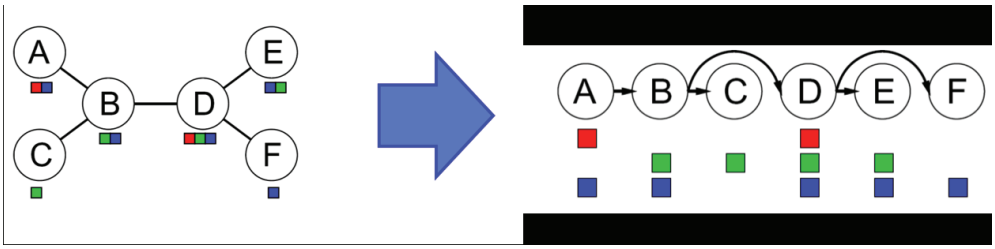
-حداقل محدودیت مقدار:[2] به طور مشابه، در هنگام انتخاب مقداربعدی برای اختصاص، سیاست خوب دیگر این است که مقداری را انتخاب کنید که از بین **دامنه مقادیر باقی‌مانده اختصاص‌نیافته** کمترین تعداد مقادیر را برمی‌دارد. به طور قابل توجهی، این کار نیاز به محاسبات اضافی دارد، اما بسته به استفاده ممکن است هنوز سرعت را افزایش دهد.

ساختار

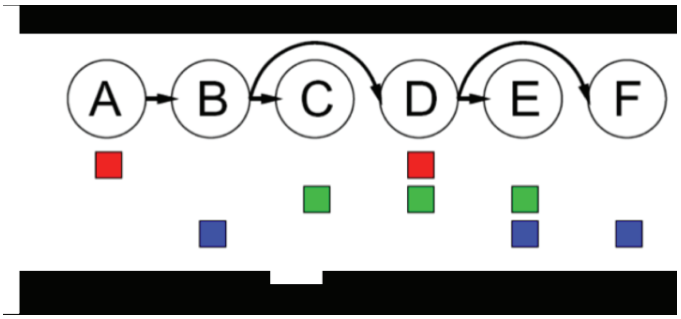
آخرین مراحل پیشرفته برای حل مشکلات رضایت از محدودیت‌ها، آنهایی هستند که از ساختار مسائل استفاده می‌کنند. به طور خاص، اگر بخواهیم یک CSP با ساختار درختی(که در گراف آن حلقه وجود ندارد) را حل کنیم، می‌توانیم زمان اجرا برای یافتن راه‌حل را از $O(d^N)$ تا $O(nd^2)$کاهش دهیم. این کار با الگوریتم CSP با ساختار درختی، که در زیر شرح داده شده است، امکان پذیر است:

- ابتدا یک گره دلخواه در گراف محدودیت CSP انتخاب کنید تا به عنوان ریشه درخت عمل کند. (مهم نیست کدام گره، زیرا نظریه اصلی گراف به ما می گوید هر گره درخت می تواند به عنوان ریشه عمل کند).
- تمام یال‌های بدون جهت درخت را به یال‌های جهت‌دار تبدیل کنید که از ریشه دور می‌شوند. سپس گراف جهت‌دار حاصل را خطی (یا بر اساس توپولوژی مرتب‌سازی) کنید. به طور ساده، این به معنای مرتب کردن گره‌های گراف به گونه‌ای است که همه‌ی یال‌ها به سمت راست اشاره کنند. با توجه به اینکه گره A را به عنوان ریشه انتخاب می کنیم و تمام یال‌ها را دور از A هدایت می کنیم، این فرآیند منجر به تبدیل زیر برای ارائه شده در زیر می شود:

[1] Minimum Remaining Values
[2] Least Constraining Value

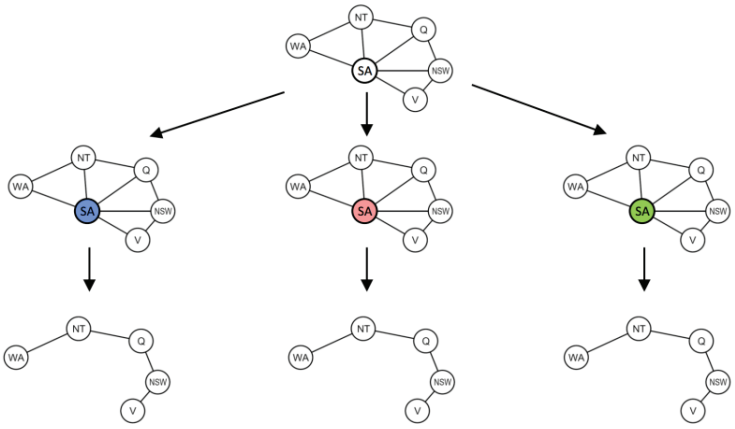


- جستجوی عقب‌گردی را برای سازگاری کمان‌ها انجام دهید. با تکرار از $i = n$ تا $i = 2$، سازگاری را برای همه کمان ها به صورت $\text{Parent}(X_i) \rightarrow X_i$ اعمال کنید. برای CSP خطی سازگار شده از بالا، با پاک کردن چند مقدار، مابقی مقادیر به شرح زیر خواهند بود:



- و در آخر، نوبت به جلو رفتن روی کمان‌ها می‌رسد. با شروع از X_1 و رفتن به X_n، به هر X_i مقداری منطبق با مقدار والد خود اختصاص دهید. چون ما سازگاری را روی همه این کمان ها اعمال کرده ایم، مهم نیست که چه مقداری را برای هر گره انتخاب می کنیم، می دانیم که فرزندان آن هر کدام حداقل یک مقدار سازگار خواهند داشت.[3]

الگوریتم ساختار-یافته درختی می‌تواند به مسائل CSP تبدیل شود که با استفاده از مجموعه‌ای به نام مجموعه برشی[4] به ساختار درختی نزدیک هستند. مجموعه برشی اولین مجموعه‌ای است که شامل یافتن کوچکترین زیر مجموعه متغیرها در یک گراف محدودیت باشد به گونه ای که حذف آنها منجر به یک درخت شود. (چنین زیر مجموعه ای به عنوان cutset برای گراف شناخته می شود) به عنوان مثال، در مثال رنگ آمیزی نقشه، SA کوچکترین cutset ممکن است:



[3] برای اطلاعات بیشتر این سایت را بررسی کنید.
[4] Cutset conditioning

هنگامی که کوچکترین مجموعه برشی پیدا شد، همه متغیرها را به آن اختصاص می دهیم و دامنه همه گره‌های همسایه را هرس می کنیم. آنچه باقی می‌ماند یک CSP با ساختار درختی است که می توانیم با الگوریتم ساختار-یافته درختی CSP آن را حل کنیم! اختصاص اولیه به یک مجموعه برشی با اندازه c ممکن است باعث بروز مشکلی در CSP با ساختار درختی شده و بعد از هرس کردن، هیچ راه‌حل معتبری باقی نماند. پس نیاز است بازگشت به عقب را d^c بار تکرار کنیم. از آنجایی که حذف مجموعه برشی ما را با یک CSP با ساختار درختی با متغیرهای (n-c)مواجه می‌کند، می‌دانیم که این می‌تواند در زمان $O((n-c)d^2)$ حل شود. بنابراین، زمان اجرای مجموعه برشی برای CSPها$O(d_c(n-c)d_2)$ است. هرچه c کوچک‌تر باشد، بهتر است.