جهش [۱] **:** تولید مشخصه های جدید که در والدها وجود ندارد ؛ تغییرات تصادفی اجـزای رمزکننـده در نسل که توسط احتمال کنترل شده است .

الگو [۲] : اجزای مفید یک راه حل می توانند در میان نسل ها باقی بمانند .

انتخاب مناسب ترین فرضیه [۳] **ها**

انتخاب شایستگی مناسب با استفاده از فرمو زیر می تواند منجر به تولید چند کپی منتشرشده شود .

$$P(h_i) = \frac{Fitness\,(h_i)}{\sum_{j=1}^{k} Fitness\,(h_j)}$$

انتخاب مسابقه ای [۴] : دو فرد را به صورت تصادفی و با احتمال یکسان انتخاب نمایید سپس با یک احتمال ثابت ، مورد مناسب تر را انتخاب نمایید .

انتخاب ردیف [۵] **:** همه ی فرضیه ها را براساس شایستگی مرتب نمایید ؛ احتمال انتخـاب بـا ردیـف متناسب است .

crossover

مرکب از تکه های ترکیب کننده ی افراد برای ساختن افراد جدید است .

---

[۱] mutation
[۲] schema
[۳] hypothese
[۴] tournament
[۵] rank

Crossover تک نقطه ای : یک نقطه ی Crossover را انتخاب نمایید و Crossover ها را از آنجا بردارید و قطعه ها را تعویض نمایید ؛ به عنوان مثال :

0111110 101|1110

crossover

1010101 011|0101

گام بعد :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

0111110 101|1110

crossover

1010101 011|0101

گام بعدی :

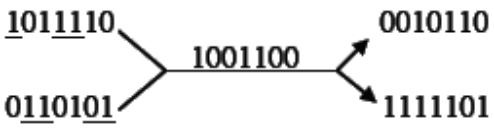
0111110 101|1110

crossover

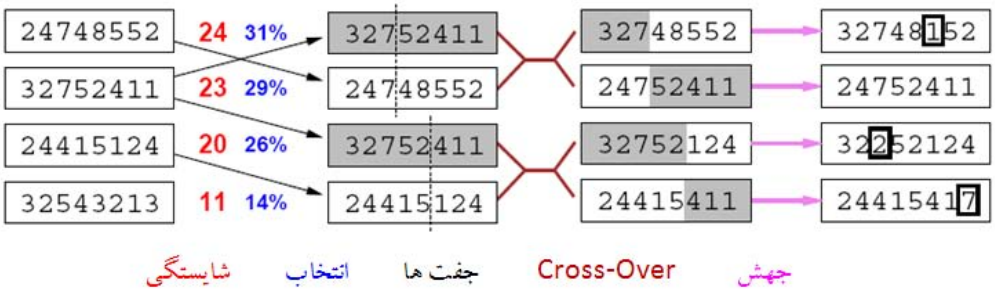
1010101 011|0101

پیـاده سـازی : از یـک ماسـک **crossover** (رشـته ی دودویـی) اسـتفاده نماییـد کـه در مثـال 1110000 می باشد . دو والد h_1 و h_2 ارایـه شـده : $(h_1 \wedge m) \vee (h_2 \wedge \neg m), (h_1 \wedge \neg m) \vee (h_2 \wedge m)$ مـی باشند . Crossover با ماسک های اختیاری به صورت زیر است :

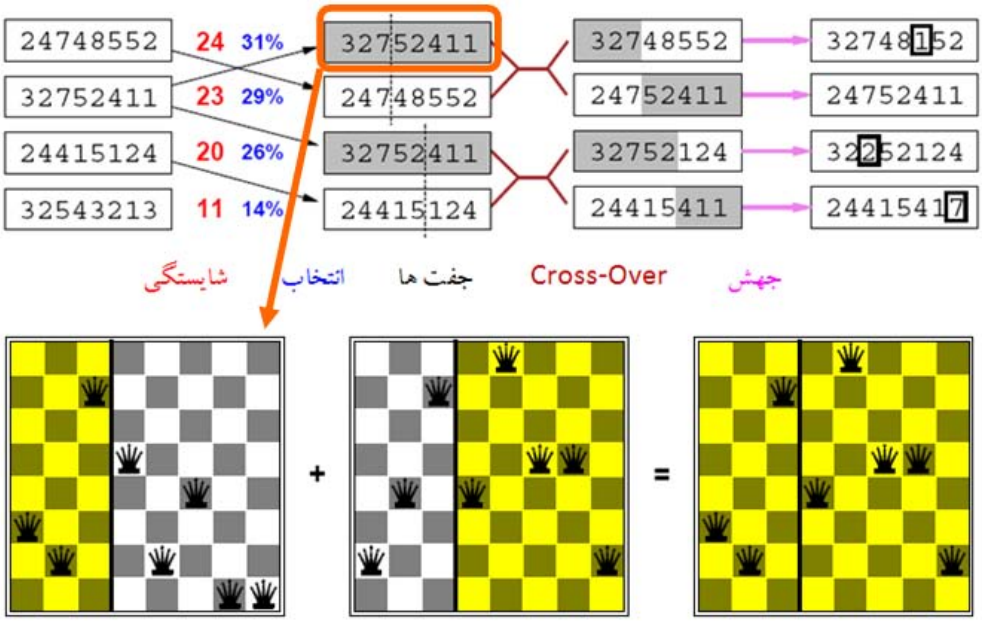
نسل	ماسک	اوّلیه	
0111110 → 1010101	1110000	1011110 → 0110101	تک نقطه
1010110 → 0111101	0011100	1011110 → 0110101	دو نقطه

	یکسان

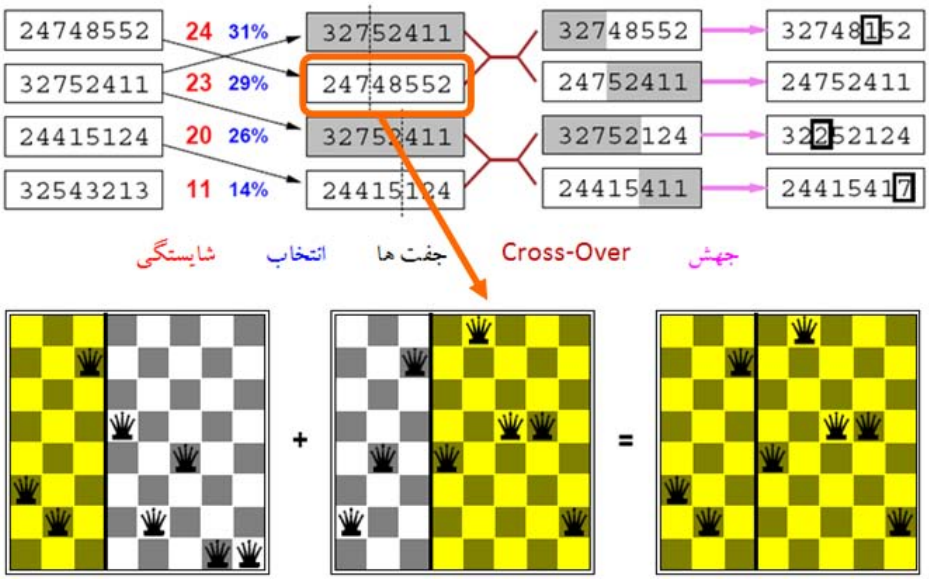
مثال : ۸ – وزیر



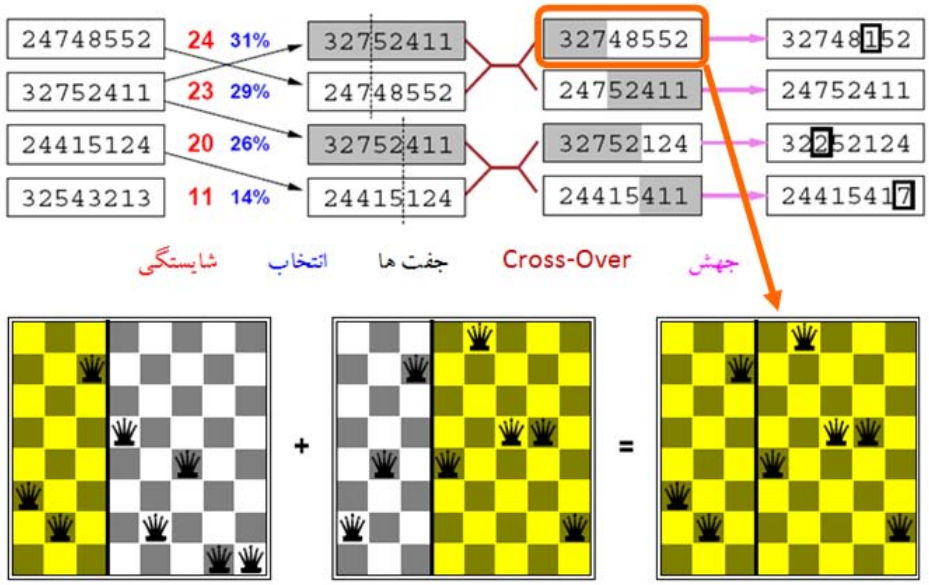
گام بعدی :



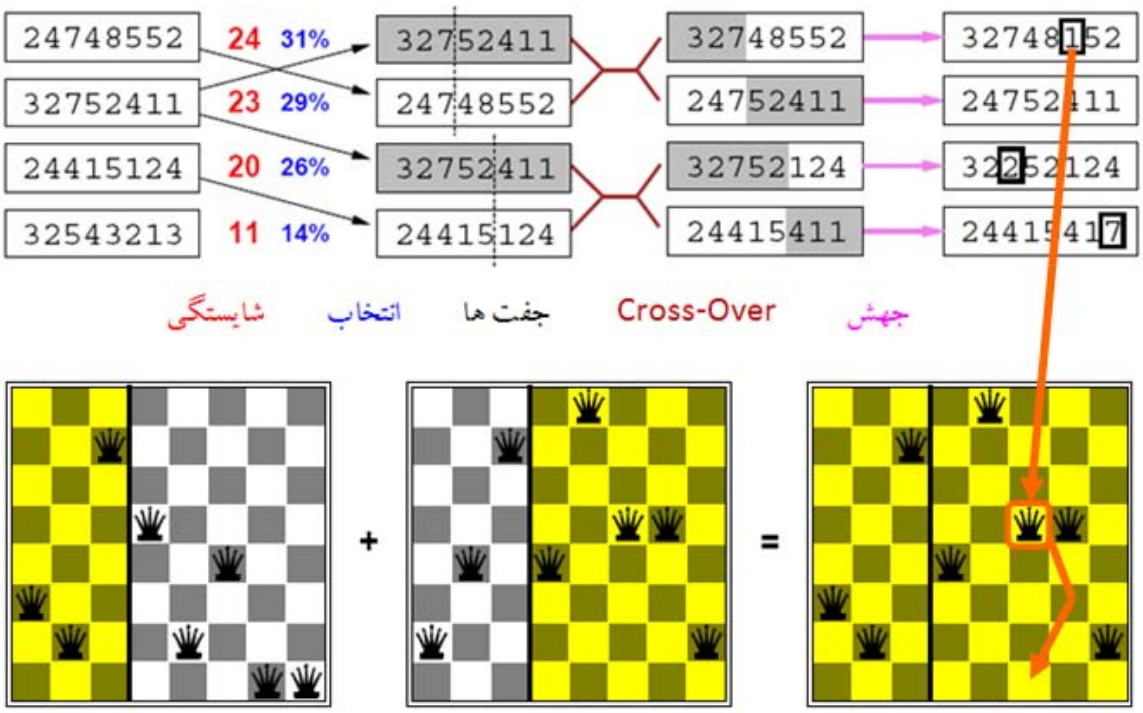
گام بعدی :



گام بعدی :



گام بعدی :



الگوریتم های ژنتیکی به حالت های رمزگذاری شده به صورت رشته نیاز دارند (GP ها از برنامه ها استفاده می کنند) . crossover به زیر رشته های iff ، که اجزایی معنی دار هستند کمک می کند. الگوریتم های ژنتیکی با تکامل مخالفند ؛ عوامل تکامل [۱] واقعی ، تشکیلات [۲] تکراری را رمز گذاری [۳] می کند .

الگوریتم های ژنتیکی به صورت جستجو

---

[۱] genes

[۲] machinery

[۳] encode

وضعیّت ها ، راه حل های ممکن هستند . عملگرهای جستجو : جهش ، دورگه (crossover) و انتخاب هستند . از آنجایی که برای به وجود آوردن جمعیّت ها چند راه حل به صورت موازی به کار گرفته می شوند . با انتخاب تابع مناسب ، مثل روش تپّه نوردی ، بدون سراشیبی می باشد . جهش و دورگه باید به ما اجازه دهند که از کمینه ی (مینیمم) محلّی خارج شویم . وابسته به روش جستجوی شبیه سازی گرم و سرد کردن می باشد .

نکته : برنامه نویسی ژنتیکی وابسته به الگوریتم ژنتیکی می باشد و در آن ها فردها (رشته ها) همان برنامه ها هستند .

کاربردهای الگوریتم ژنتیکی

اغلب برای مسایل بهینه سازی مورد استفاده قرار می گیرد مثل : آرایش مدار ، طراحی سیستم و زمانبندی . به طور خلاصه باید بگوییم که به اندازه ی کافی برای پیداکردن راه حل ، مناسب می باشد ولی بهبود قابل توجّهی در مورد چند نسل ندارد و دارای محدودیّت زمانی هستند .

فضاهای حالت پیوسته – تصوّر کنید که می خواهیم جای سه فرودگاه را در کشور رومانی مشخص نماییم: فضای حالت شش بعدی توسط $(x_1,y_2),(x_2,y_2),(x_3,y_3)$ تعریف می شود . تابع هدف : $f(x_1,y_2,x_2,y_2,x_3,y_3)$ = مجموعه ای از فواصل مربّعی از هر شهر به نزدیک ترین فرودگاه می باشد . متدهای مجزا [1] ، فاصله های پیوسته را به فاصله های مجزا تبدیل می نمایند . گرادیان تجربی با تغییرات $\pm\delta$ در هر وضعیّت متناسب است . گرادیان از فرمول زیر محاسبه می شود:

$$\nabla f = \left(\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial y_1},\frac{\partial f}{\partial x_2},\frac{\partial f}{\partial y_2},\frac{\partial f}{\partial x_3},\frac{\partial f}{\partial y_3}\right)$$

[1] discretization methods

با توجّه به رابطه ی $x \leftarrow x + \alpha \nabla f(x)$ می توانیم مقدار f را افزایش (کاهش) دهیم ، گاهی از اوقات می توان این معادله را به درستی برای حالت $\nabla f(x) = 0$ در صورتی که فقط یک شهر داشته باشیم حل نماییم . در روش نیوتن – رفسون [۱] (1664-1690) برای حل $\nabla f(x) = 0$ در جایی که $H_{ij} = \partial^2 f / \partial x_i \partial x_j$ از فرمول زیر استفاده می شود:

$$x \leftarrow x - H_f^{-1}(x) \nabla f(x)$$

[۱] Newton-Raphson

مترجم: سهراب جلوه گر

ویرایش دوّم ، بهار ۱۳۸۸

فصل ششم

مسایل ارضای محدودیّت [۱]

[۱] Constraint Satisfaction Problems (CSPs)

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

رئوس مطالب

- مثال های مسایل ارضای محدودیّت
- جستجوی معکوس لیست [۱] برای مسایل ارضای محدودیّت
- ساختار مسأله و تجزیه [۲] ی مسأله
- جستجوی محلّی برای مسایل ارضای محدودیّت

مسایل ارضای محدودیّت

مسأله ی جستجوی استاندارد : حالت ، یک " جعبه ی سیاه [۱] " است – هر ساختمان داده ای قدیمی که از تست اهداف ، ارزیابی و جانشین پشتیبانی می کند .

---

[۱] backtracking

[۲] decomposition

مسأله ی ارضای محدودیّت : حالت ، توسط **متغیّرهای** X_i و با **مقدارهای** درون **دامنه ی** D_i تعریف می شود. تست هدف ، مجموعه ای از محدودیّت های معین کننده ی محدودیّت های مجاز از مقادیر زیرمجموعه ی متغیّر ها می باشد. یک **زبان ارایه ی رسمی** [۲] مثال ساده ای از آن می باشد ، الگوریتم های **همه منظوره** [۳] ی مفید را با توانایی بیش تر از الگوریتم های جستجوی استاندارد سبب می شود .

مثال نقشه ی رنگی : متغیّرها عبارتند از : WA,NT,Q,NSW,V,SA,T . دامنه ها :

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

---

[۱] black box : نوعی از مهندسی معکوس است که هدف اصلی آن بررسی رفتار برنامه بدون توجه به کد برنامه ی آن است . مهندسی معکوس ، از حالت کامپایل در آوردن یا تکه تکه کردن برنامه ی کاربردی بدون توجه به زبان برنامه نویسی ای که توسط آن به وجود آمده است می باشد بنابراین ممکن است به کد اصلی برنامه دسترسی پیدا نماید . (Babylon / Code Analysis)

[۲] formal representation language

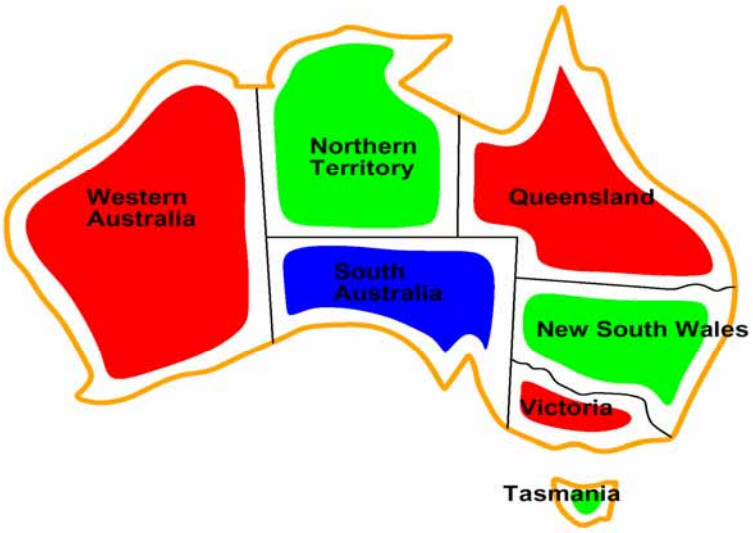
[۳] general-purpose

محدودیّت ها : نواحی مجاور ، باید رنگ های متفاوتی داشته باشند. به عنوان مثال $WA \neq NT$

(اگر زبان این مطلب را اجازه بدهد) ، یا $(WA,NT) \in$

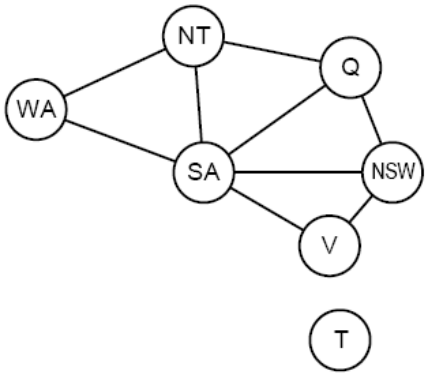
{(red,green),(red,blue),(green,red),(green,blue),…}

گام بعدی :



راه حل ارایه شده همه ی شرایط را لحاظ می کند ، به عنوان مثال ،

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



مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

گراف محدودیّت [۱]

مسأله ی ارضای محدودیّت دودویی [۲]: هر محدودیّت حداکثر دو متغیّر را تشریح می نماید.

گراف محدودیّت : در این گراف ، گره ها ، متغیّرها هستند ، کمان ها [۳]، محدودیّت ها را نشان می دهند. الگوریتم های همه منظوره ی مسأله ی ارضای محدودیّت از ساختار درختی برای بالا بردن سرعت جستجو استفاده می کنند . توجّه نمایید که شهر تاسمانیا یک زیر مسأله ی مستقل است !

انواع مسایل ارضای محدودیّت

مسایل ارضای محدودیّت با متغیّرهای مجزا (گسسته) ، در دامنه های محدود ؛ اندازه d است ← (در نتیجه) $O(d^n)$ و انتساب ها کامل است . به عنوان مثال ، مسایل ارضای محدودیّت بولین [۴] (NP – کامل) . در دامنه های نامحدود (integer ها ، string ها (رشته ها) و .....) به عنوان مثال در زمانبند کار [۵]، متغیّرها روزهای شروع / پایان برای هر کار هستند و به یک زبان محدود [۶] نیازمندند ، به عنوان مثال ، $StartJob_1 + 5 \leq StartJob_3$.

مسایل ارضای محدودیّت با متغیّرهای پیوسته [۱]: به عنوان مثال ، زمان های شروع / پایان برای مشاهدات تلسکوپ [۲] هابل [۳]

---

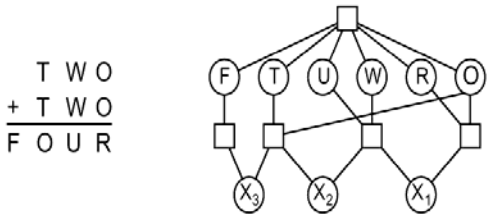
[۱] Constraint graph
[۲] Binary CSP
[۳] arcs
[۴] Boolean CSPs
[۵] job scheduling
[۶] constraint language
[۱] continuous variables

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

تنوع محدودیّت ها

محدودیّت های منحصر به فرد [۳] ، شامل یک متغیّر منفرد می شوند ، به عنوان مثال ، SA≠WA ؛ محدودیّت های دودویی ، شامل جفت هایی از متغیّرها می باشند ، به عنوان مثال ، SA≠WA و محدودیّت های مرتبه ی بالاتر ، شامل تعداد سه یا بیش تر متغیّر می باشند ، به عنوان مثال ، محدودیّت های با ستون های پنهان (رمزی) [۴] .

مثال : مسایل رمزی ، ما باید هر حرف را طوری تغییر دهیم که حاصل جمع به دست آید .



متغیّرها : $X_3\ X_2\ X_1$ F T U W R O . دامنه ها : {۰،۱،۲،۳،۴،۵،۶،۷،۸،۹} . محدودیّت ها : alldiff(F,T,U,W,R,O) و O+O=R+10. X_1 و غیره .

مسایل ارضای محدودیّت در دنیای واقعی

مسایل انتساب ، به عنوان مثال ، چه کسی درس می دهد و در چه کلاسی ؛ مسایل برنامه ی زمانی ، کدام کلاس ارایه می شود ، چه زمانی و کجا ؟ ؛ پیکربندی سخت افزاری - صفحات گسترده [۱]

---

[۱] Telescope

[۲] Hubble

[۳] unary

[۴] cryptarithmetic column constraints

[۱] برنامه ای که امکان اجرای محاسبات روی چندین ستون از اعداد را برقرار کند ، spreadsheet نام دارد

– زمان بند کننده ی انتقال – زمان بند کننده ی مرکز تولید . توجّه کنید که تعدادی از مسایل دنیای واقعی شامل متغیّرهای با مقدار واقعی هستند .

فرمول بندی جستجوی استاندارد ، بیایید به صورت ساده شروع کنیم ، برخورد ساده ای داشته باشیم و سپس آن را اثبات نماییم . تا کنون حالت ها توسط مقادیر نسبت داده شده تعریف می شدند : حالت اوّلیه را برابر با تهی در نظر می گیریم ؛ تابع جایگزین ، یک مقدار را به یک متغیّر فاقد مقدار شده نسبت می دهد طوری که با مقدار فعلی تعارض یا ناسازگاری نداشته باشد . در نتیجه در صورتی که مقداردهی های مجاز وجود نداشته باشد ، رد می شود . آزمایش یا تست هدف : برّرسی می کند که انتساب جاری کامل باشد .

جستجوی معکوس لیست – انتساب متغیّرها ، جابجایی پذیر می باشد ، توجّه نمایید که ، [WA=red , NT=green] همانند [NT=green , WA=red] می باشد . فقط لازم است توجّه نمایید که انتساب ها برای یک متغیّر منفرد در هر گره می باشد ، در نتیجه b=d و تعداد d^n گره وجود دارد . ***جستجوی اوّل عمق برای مسایل ارضای محدودیّت با انتساب های متغیّر منفرد ، جستجوی معکوس لیست نام دارد .*** *جستجوی معکوس لیست ، اساس الگوریتم ناآگاهانه برای مسایل ارضای محدودیّت می باشد .* می توان n– وزیر را برای $n \approx 25$ حل نمود .

الگوریتم

تابع Backtracking-Search(csp) ، راه حل یا عدم موفّقیّت را برمی گرداند

Recursive-Backtracking({},csp) را برگردان

تابع Recursive-Backtracking(assignment,csp) ، soln یا عدم موفّقیّت را برمی گرداند .

اگر انتساب (assignment) کامل است انتساب را برگردان

var ← Select-Unassigned-Variable(Variable[csp],assignment,csp)

برای هر مقـدار (value) در Order-Domain-Values(var,assignment,csp) کارهـای زیر را انجام بده

در صورتی که value با assignment ارایه شده در Constraint[csp] سازگار است

{ var = value } را به انتساب (assignment) اضافه نما

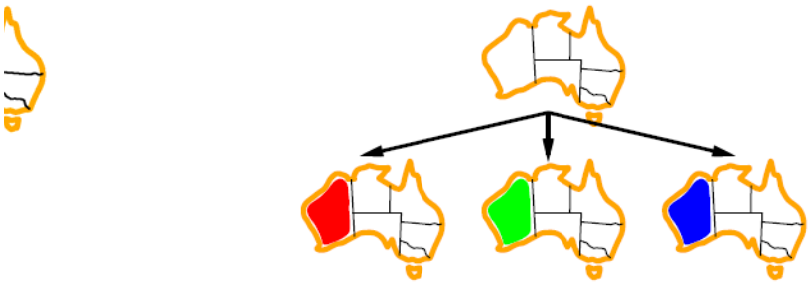
result ← Recursive-Backtracking(assignment,csp)

در صورتی که result ≠ failure است ، result را برگردان

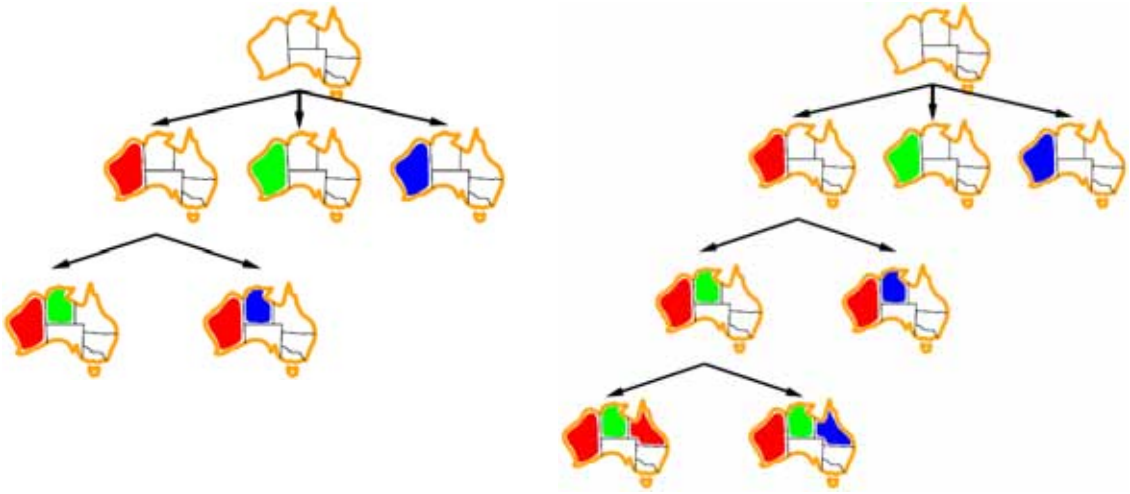
{var = value} را از انتساب [۱] بردار

عدم موفّقیّت را برگردان

مثال لیست معکوس (تصاویر را از چپ به راست دنبال نمایید)



[۱] assignment



بهبود بخشیدن به کارایی لیست معکوس

متدهای همه منظوره می توانند افزایش زیادی در سرعت داشته باشند :

کم ترین مقادیر باقی مانده – کم ترین مقادیر باقی مانده [۱] : متغیّری را با کمترین مقـادیر مجـاز انتخاب نمایید .

a. قرمز دارای محدودترین مقدار می باشد

b. متغیّری با کم ترین مقادیر مجاز را انتخاب نمایید

[۱] Minimum Remaining Values (MRV)

پیداکردن درجه – متغیّرها را در طول MRV به طور تصادفی بشکنید . کشف درجه : متغیّری با بیشترین محدودیّت در متغیّرهای باقی مانده را انتخاب نمایید .

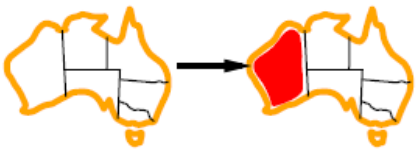
کم ترین مقدار محدود کننده – یک متغیّر ارایه شده ، کم ترین مقدار محدود کننده را انتخاب می نماید

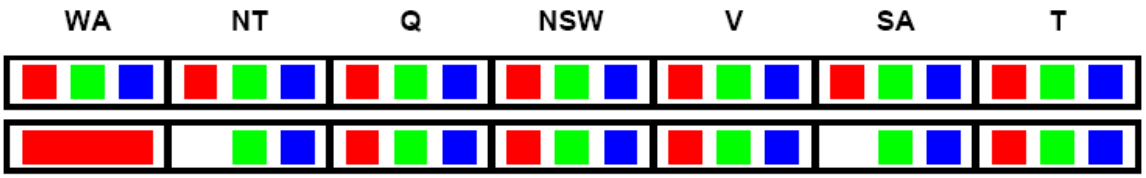
ترکیب کردن این ابتکارات ، داشتن ۱۰۰۰ وزیر را امکان پذیر می سازد .

برّسی مستقیم [1] – مسیر باقی مانده با مقدارهای مجاز را برای متغیّرهای انتساب داده نشده نگهداری نمایید و جستجو را هنگامی که هر متغیّر مقدارهای مجاز ندارد خاتمه دهید .

[1] forward checking

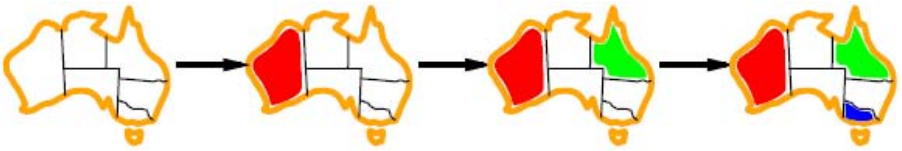
گام بعدی :

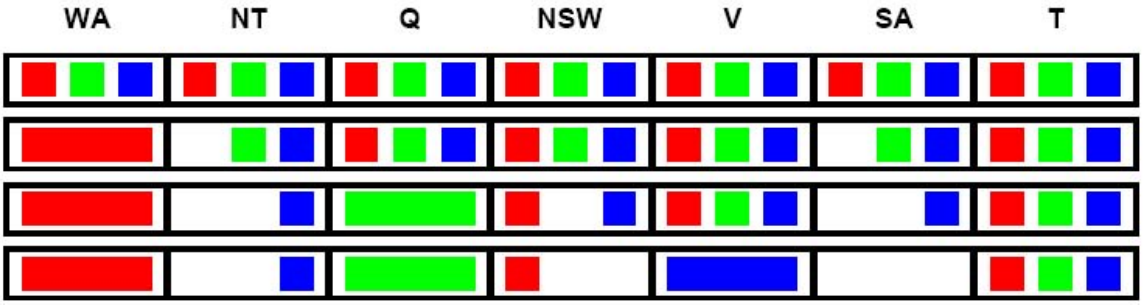




گام بعدی :

گام بعدی :

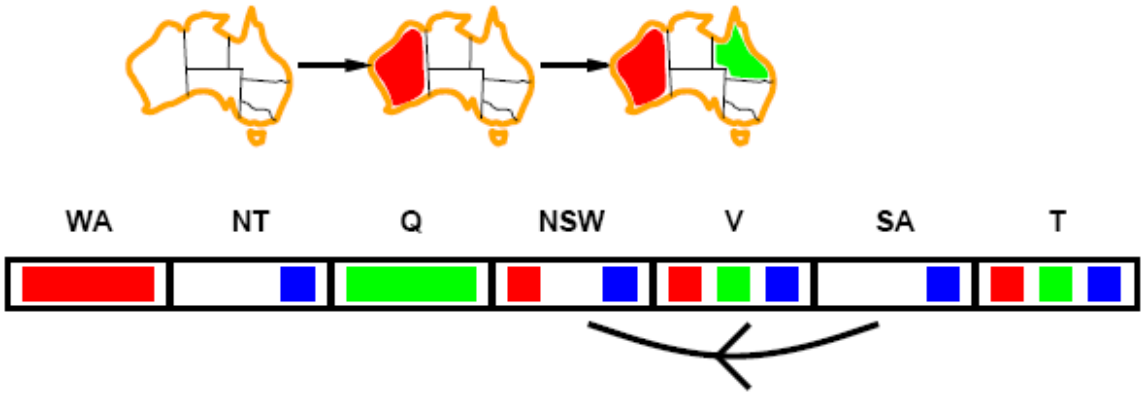




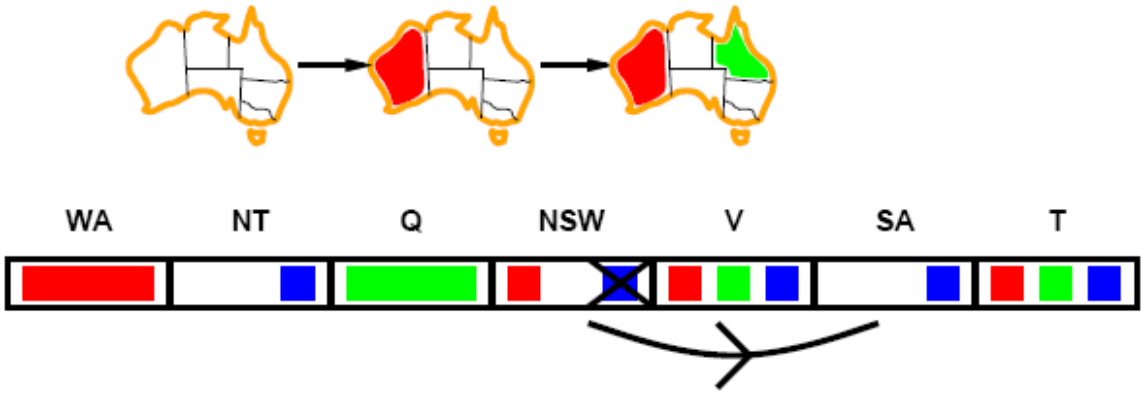
توزیع محدود – برّرسی اطّلاعات توزیع های مستقیم از متغیّرهای منتسب به متغیّرهای منتسب نشده ، به صورت سریع برای همه ی عدم موفّقیّت ها عمل نمی کند :

NT و SA هر دو نمی توانند آبی باشند ! توزیع محدود ، به صورت تکراری ، محدودیّت ها را به صورت محلّی اجرا می کند .

سازگاری قوس [۱] – ساده ترین توزیع ، سازگاری هر قوس را سبب می شود . $X \rightarrow Y$ سازگار می باشد اگر برای هر مقدار x از X تعدادی برای y مجاز باشند .

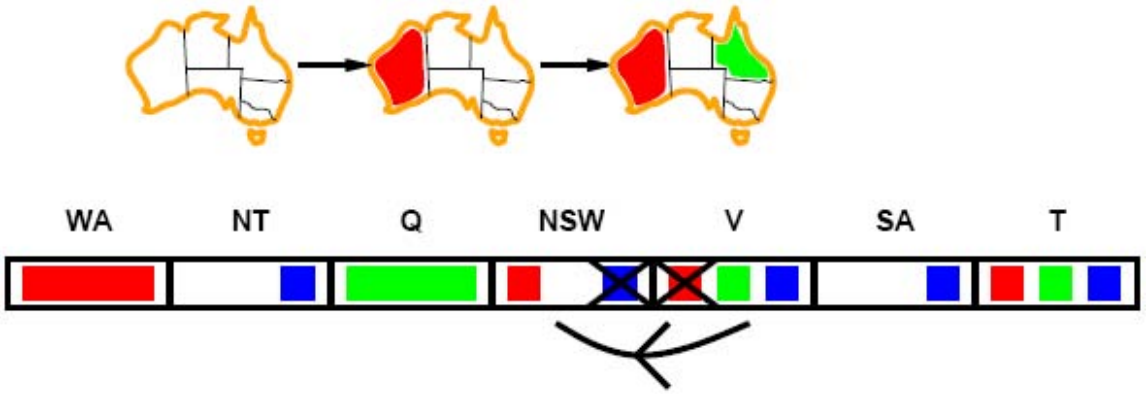


گام بعدی :

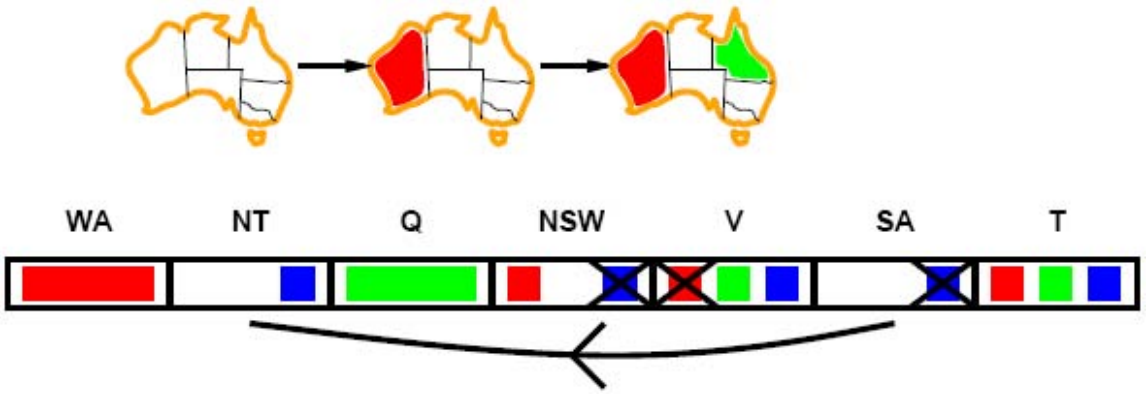


گام بعدی :

[۱] Arc consistency (AC)



اگر X یک مقدار را از دست داد ، همسایه های X باید دوباره برّرسی شوند .



اگر X یک مقدار را از دست داد ، همسایه های X باید دوباره برّرسی شوند. سازگاری قوس زودتر از بررسی مستقیم ، عدم موفّقیّت پیدا می کند و می تواند به صورت یک پیش پردازنده و یا بعد از هر انتساب اجرا شود .

الگوریتم سازگاری قوس

تابع AC-3(csp) ، CSP و شاید دامنه های کاهش داده شده را برگرداند

ورودی ها ،CSP که یک مسأله ی ارضای محدودیّت دودویی با متغیّرهای $\{X_1, X_2, ..., X_n\}$ می باشد

متغیّرهای محلّی ، queue ، که یک صف از قوس ها که همه به صورت اوّلیه قوس هایی در csp هستند

مادامی که queue خالی نیست کارهای زیر را انجام بده

$(X_i, X_j) \leftarrow$ Remove-First(queue)

اگر Remove-Inconsistent-Values (X_i, X_j) آن گاه

برای هر X_k در X_iNeighbours(کارهای زیر را انجام بده)

(X_k, X_i) را به queue اضافه کن

تابع Remove-Inconsistent-Values(X_i, X_j) موفّقیّت های صحیح iff را برمی گرداند

remove ← false

برای هر x در $Domain[X_i]$ کارهای زیر را انجام بده

در صورتی که هیچ مقداری در $Domain[X_j]$ اجازه نمی دهد که (x,y) محدودیّت $X_i \leftrightarrow X_j$ را ارضا کند آن گاه x را از $Domain[X_i]$ حذف کن ؛ removed ← true

removed را برگردان

$O(n^2d^3)$ می تواند به $O(n^2d^2)$ کاهش داده شود.

سازگاری قوس فقط مقادیر ناسازگار را برمی دارد و با این کار فضای جستجو را هرس می کند.

مثال : $X \neq Y$ و $X \neq Z$ و $Y \neq Z$ با دامنه ی {۱و۲} دارای سازگاری قوس می باشد امّا راه حل نیست .

سازگاری قوس هدایتی [۱]

سازگاری قوس ، تجدید نظر را تکرار می کند (تعداد دفعات وابسته به قوس ها و دامنه ها می باشد) ولی سازگاری قوس هدایتی ، فقط یک بار بازنگری می کند و یک ترتیب متغیّر را بازبینی می کند (<) ؛ مسأله ی ارضای محدودیّت در صورتی یک سازگاری قوس هدایتی است اگر هر قوس (X_i, X_j) که $X_i < X_j$ سازگاری قوس باشد . توجّه کنید که سازگاری قوس ، کلّی تر از سازگاری قوس هدایتی است و برای درخت ها مناسب می باشد .

سازگاری مسیر [۲]

یک مسیر $(X_1,...,X_m)$ یک مسیر سازگار است اگر برای هر جفت از مقادیر x_1 و x_m راضی کننده ی همه ی محدودیّت های x_1 و x_m باشد ؛ یک انتساب برای $X_2,...,X_{m-1}$ که راضی کننده ی محدودیّت های (X_i, X_{i+1})باشد وجود دارد . مسأله ی ارضای محدودیّت با مسیر سازگار است در صورتی که هر مسیر ، سازگار باشد . در واقع ، شما فقط سازگاری همه ی مسیرهای با طول ۲ را احتیاج دارید . سازگاری مسیر جفت های مقادیر را برمی دارد و محدودیّت ها را صریح می سازد (به عنوان مثال ، $X<Y$ و $X+1<Z \Leftarrow Y<Z$

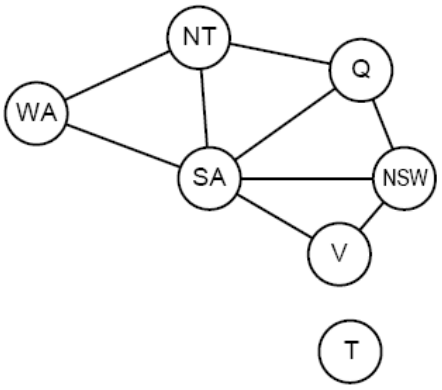
K – سازگاری [۱]

---

[۱] Directional Arc Consistency (DAC)

[۲] Path Consistency (PC)

k – سازگاری ، اَشکال سازگاری را تعمیم می دهد . مسأله ی ارضای محدودیّت ، k- سازگار است اگر برای هر مجموعه ی k-1 متغیّری با انتساب سازگار و یک مقدار سازگار بتواند به هر k انتساب داده شود . در صورتی که هر متغیّر با خودش سازگار باشد ، سازگار می باشد و دارای سازگاری قوس و سازگاری مسیر هم هست .



K – سازگار قوی - یک مسأله ی ارضای محدودیّت k – سازگار قوی است در صورتی که k – سازگار و k-1 – سازگار و ... و ۱- سازگار باشد .

ساختار مسأله [۲]

تاسمانیا و مین لند [۳] زیر مسأله هایی مستقل هستند و به صورت اجزای متصل شده به گراف محدودیّت قابل تعریف اند حال تصوّر کنید که هر زیر مسأله دارای متغیّرهای c تا حد مجموع n باشد . در بد ترین حالت ، هزینه ی راه حل برابر است با $n/c.d^c$ و به صورت خطی بر حسب مقدار n می

---

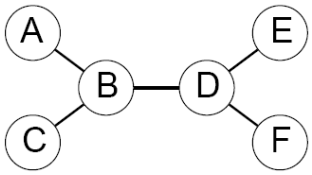
۱ K-consistency

۲ problem structure

۳ mainland

باشد . به عنوان مثال ، برای $n = 80 , d = 2 , c = 20$ داریم : 2^{80} = چهار بیلیون سال با سرعت ده میلیون گره بر ثانیه . 4.2^{20} = چهار دهم ثانیه با سرعت ده میلیون گره بر ثانیه .

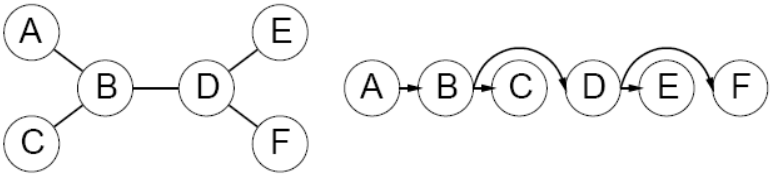
مسایل ارضای محدودیّت درختی ساختیافته [۱]



قضیّه : *در صورتی که گراف محدودیّت هیچ حلقه ای نداشته باشد ، مسأله ی ارضای محدودیّت می تواند در زمان* $O(nd^2)$ *حل شود* . این زمان را در جایی که در بدترین حالت ، زمان برابر $O(d^n)$ است مقایسه نمایید . این خصوصیت همچنین برای استدلال های منطقی و احتمالاتی به کار برده می شود : یک مثال مهم ، ارتباط میان محدودیّت ها و پیچیدگی استدلال است.

الگوریتمی برای مسایل ارضای محدودیّت درختی ساختیافته

۱ . متغیّری را به صورت ریشه انتخاب نمایید ، متغیّرها را از ریشه به برگ ها مرتّب نمایید تا این که هر گره مقدم والد آن به صورت منظم در آید ،



۲ . برای j از n تا ۲ (به صورت کاهشی) ، تابع RemoveInconsistent $(Parent(X_j), X_j)$ را به کار ببرید

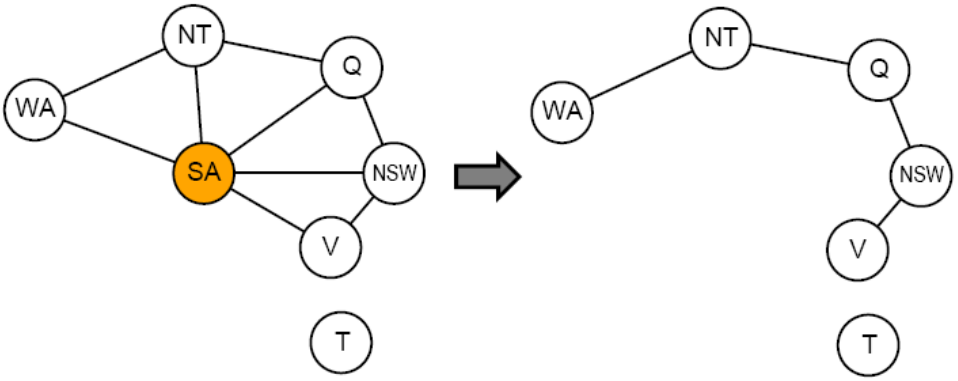
---

[۱] Tree-structured CSPs

۳. برای j از ۱ تا n ، X_j را بدون تناقض با $Parent(X_j)$ منتسب نمایید.

مسایل ارضای محدودیّت درختی ساختیافته ی تقریبی [۱]

شایسته سازی [۲] : معرّفی یک متغیّر و هرس کردن دامنه ی همسایه های آن .



شایسته سازی برش [۳] : معرّفی (در همه ی موارد) یک مجموعه از متغیّرها تا این که گراف محدودیّت ، به یک درخت تبدیل شود . اگر اندازه ی برش c باشد ، در نتیجه ، زمان اجرای $O(d^c.(n-c)d^2)$ را خواهیم داشت ، برای c کوچک خیلی سریع است .

درخت تجزیّه [۴]

مسأله را به زیر مسأله های پیوسته تجزیّه می نماید که زیر مسأله ها همان گره ها هستند . احتیاجات (نیازهای) ما عبارتند از : برای هر متغیّر ، لااقل یک گره باید داشته باشیم ؛ متغیّرهای پیوسته باید لااقل در یک

---

۱ Nearly tree-structured CSPs

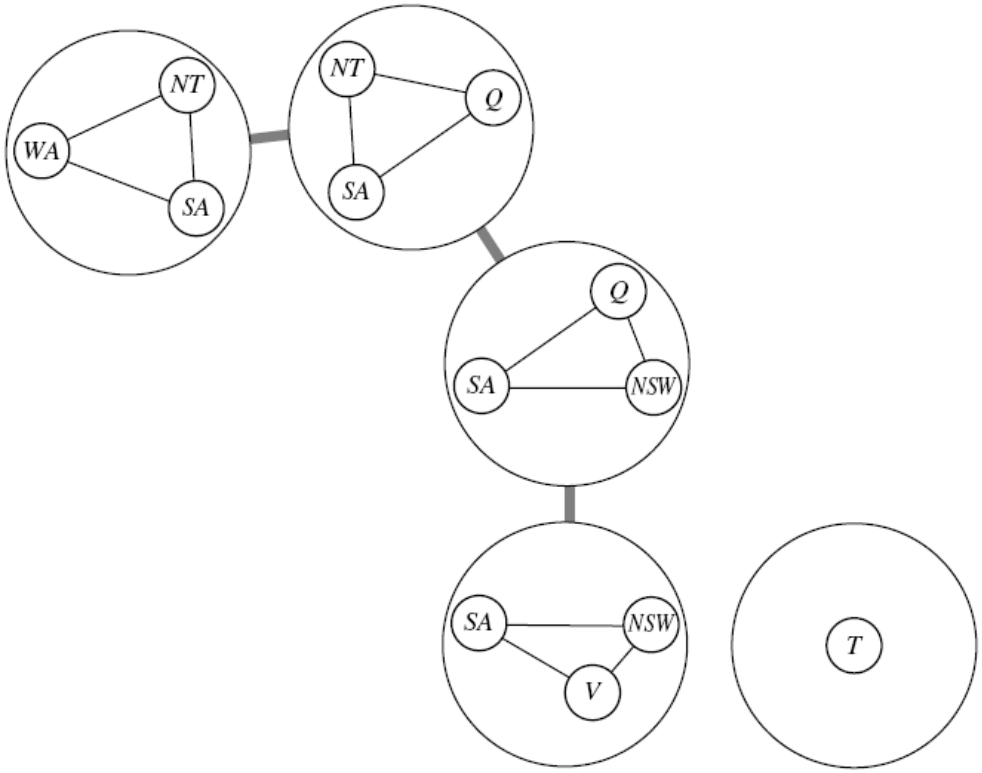
۲ Conditioning

۳ Cutset conditioning

۴ Tree decomposition

گره با هم به نظر برسند و متغیّری که به نظر می رسد در دو گره است باید در هر گره دیگر در مسیر وجود داشته باشد .

مثال درخت تجزیّه



توجّه کنید که محدودیّت میان گره ها ، سازگاری متغیّرها را سبب می شود و تجزیه منحصر به فرد نمی باشد در ضمن زیر مسأله ها باید تا حد امکان کوچک باشند . زیرمسأله ها به صورت مستقل حل می شوند و سپس به وسیله ی متغیّرهای مشترک به هم چسبانده [1] می شوند . *در صورتی که یک زیرمسأله دارای*

[1] glued

هیچ راه حلی نباشد آن گاه کل مسأله دارای هیچ راه حلی نمی باشد . در صورتی که اندازه ی بزرگ ترین مسأله ، w باشد ، آن گاه مسأله می تواند در $O(nd^w)$ حل شود .

الگوریتم های تکراری برای مسایل ارضای محدودیّت – تپّه نوردی و شبیه سازی گرم و سرد کردن معمولی با حالت های کامل کار می باشد ، توجّه کنید که همه ی متغیّرها نسبت داده شده اند . برای به کاربردن با مسایل ارضای محدودیّت ، باید حالت های مجاز با محدودیّت های عملگرهای ناخوشایند ، دوباره به مقدار متغیّرها نسبت داده شوند .

انتخاب متغیّر : هر متغیّر ناسازگار را به صورت تصادفی انتخاب نمایید .

ابتکار انتخاب مقدار با کم ترین ناسازگاری : مقداری را که از کم ترین محدودیّت ها تخلف می کند را انتخاب نمایید . توجّه نمایید ، در تپّه نوردی ، $h(n)$ برابر با تعداد مجموع محدودیّت های متخلف می باشد .

مثال : چهار وزیر

تصوّر نمایید که یک وزیر در هر ستون وجود دارد .

متغیّرها : Q_1 و Q_2 و Q_3 و Q_4

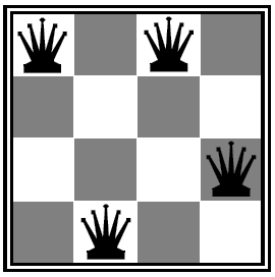
دامنه ها : $D_i = \{1,2,3,4\}$

محدودیّت ها :

$Q_i \neq Q_j$ (نمی توانند در ردیف یکسان باشند)

$|Q_i - Q_j| \neq |i - j|$ (در یک قطر هم نمی توانند باشند)

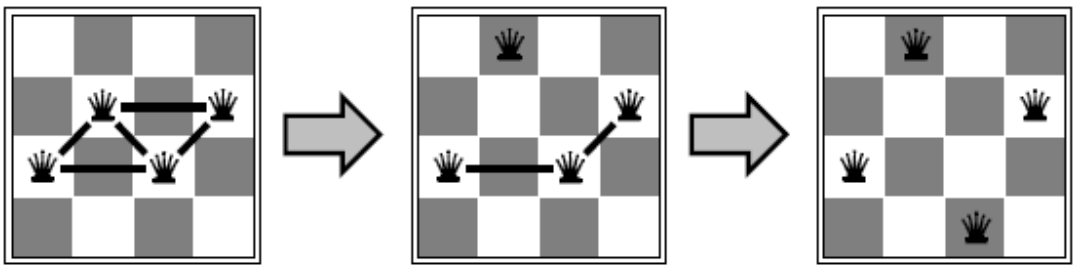
هر محدودیّت را به مجموعه ای از مقادیر مجاز برای متغیّرشان ترجمه نمایید :



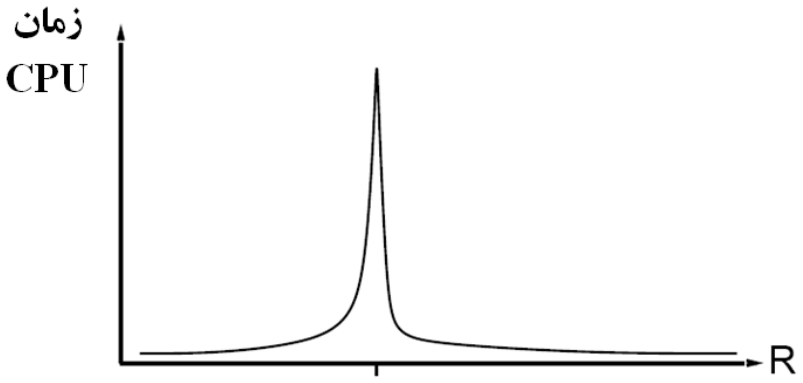
به عنوان مثال ، برای (Q_1, Q_2) مقادیر عبارتند از (۳و۱) و (۴و۱) و (۴و۲) و (۱و۳) و (۱و۴) و (۲و۴)

.

حالت ها (وضعیّت ها) : چهار وزیر در چهار ستون ($4^4 = 256$ حالت) | عملگرها : وزیر را در ستون حرکت دهید | تست هدف : هیچ حمله ای وجود نداشته باشد . | ارزیابی : $h(n)$ = تعداد حملات



عمل با کمترین ناسازگاری – حالت اوّلیه ی تصادفی ارایه شده ، می تواند n – وزیر را در ثابت زمانی تقریبی دلخواه n با احتمال زیاد حل کند ($n = 10{,}000{,}000$) . بیان های همانند می توانند برای هر مسأله ی ارضای محدودیت به صورت تصادفی صحیح باشند **به جز** در یک محدوده ی باریک از نسبت زیر : **R = تعداد محدودیت ها / تعداد متغیّرها**



نسبت بحرانی

محدودیّت های دودویی

بیش تر تکنیک های گفته شده فقط با محدودیّت های دودویی کار می کنند . تعدادی از مسایل به طور طبیعی به صورت محدودیّت های n تایی [۱] ، رمزگذاری می شوند ، به عنوان مثال : X+Y=Z . توجّه نمایید که گره با سازگاری زوج ، دودویی نمی باشد ؛ به عنوان مثال $X \leq 12$ و محدودیت های یکانی به سادگی می توانند با کاهش دامنه به کار گرفته شوند :

$$D_X = \{1,5,24\} \text{ و } X \leq 12 \leftarrow D'_X = \{1,5\}$$

کاهش محدودیّت های n تایی

هر مسأله ی ارضای محدودیت می تواند به یک مسأله ی ارضای محدودیّت دودویی معادل تبدیل شود . برای این کار چند روش وجود دارد ؛ در این جا ما روش **رمزکننده ی متغیّر پنهان** [۲] را بررسی می کنیم ؛ در این روش محدودیّت های n تایی با متغیّرهای جدید ارایه می شوند ؛ دامنه ی متغیّرهای جدید ،

---

[۱] n-ary

[۲] hidden variable encoding

چندتایی [۱] (n تایی) هستند ؛ محدودیّت های دودویی متغیّرهای " اصلی " را به اجزای چندتایی ها بچسبانید . مثال : X+Y=Z ، با $X,Y,Z \in N$. در این صورت متغیّر جدید U با دامنه ی $D_U = \{(x_1, x_2, x_3) \in N^3 \mid x_1 + x_2 = x_3\}$ خواهد بود . از محدودیت $\pi_i(0,0)$ برای چسباندن مقدار I اُم استفاده نمایید ؛ به عنوان مثال :

$$\pi_2(U,Y) = \{((x_1, x_2, x_3), x_2) \in N^3 \times N\}$$

سختی مسأله ی ارضای محدودیّت

مسأله ی ارضای محدودیّت ، NP – کامل و سخت است ، برای آن ها از الگوریتم های مکاشفه ای استفاده کنید (همان طوری که در درس مطرح شده است) . در بدترین حالت ، پیچیدگی به صورت نمایی می باشد ، امّا در مواردی دارای رفتار خوبی می باشد . الگوریتم های مکاشفه ای ، کلاس هایی از مسایل نرم و رام شدنی را تعریف می نماید و دارای زمان چندجمله ای با محدودیت هایی در مسایل است . به عنوان مثال ، مسایل با ساختار درختی .

برنامه نویسی براساس محدودیّت

محدودیّت ها ، یک راه طبیعی برای بیان مسایل می باشند . مسأله ی ارضای محدودیّت ، با یک زبان برنامه نویسی حل می شود . برنامه نویسی با زبان های منطقی نظیر پرولوگ انجام می شود که دارای محیط اختصاصی می باشد (مثل موزارت [۲]) . پرولوگ به صورت های تجاری و رایگان ارایه شده است (به عنوان مثال SICStus یا GNU-Prolog) . برای اطّلاعات بیش تر نگاه کنید به راهنمای برخط برای برنامه

---

[۱] tuple

[۲] Mozart نامی برای نسخه ی ۷.۵ سیستم عامل شرکت Apple برای کامپیوترهای MAC و PowerPC . نسخه های جدید دارای خصوصیات چند وظیفه ای ، بهبود شبکه ، پشتیبانی چند رسانه ای بهتر و امکانات سیستم عامل های Dos و Windows می باشند . (Babylon / Jensen's Technology Glossary)

نویسی محدود با آدرس اینترنتی http://kti.mff.cuni.cz/~Ebartak/constraints/ و آرشیو محدودیت ها با آدرس اینترنتی http://www.cs.unh.edu/ccc/archive .

خلاصه ی فصل

مسایل ارضای محدودیّت ، نوع خاصی از مسایل هستند که در آن ها حالت ها توسط مقدارهای یک مجموعه ی ثابت از متغیّر ها تعریف می شوند . و هدف آزمایش ، توسط محدودیت هایی بر مقدار متغیّرها تعریف می شود . پیمایش معکوس ، جستجوی اوّل – عمق با یک متغیّر انتساب داده شده به هر گره می باشد . مرتب کردن متغیّر و انتخاب ابتکاری مقدارها به طور بسیار مطلوبی به ما کمک می کند . بررسی مستقیم ، از مقداردهی هایی که خرابی آینده را به وجود می آورند یا تضمین می کنند جلوگیری می کند . پخش محدود ، کاری اضافی را برای مقدارهای محدود و کشف ناسازگاری ها ، انجام می دهد . بیان مسأله به صورت مسأله ی ارضای محدودیت ، تحلیل ساختار مسأله را اجازه می دهد . درخت ساختیافته ی مسایل ارضای محدودیت می تواند در زمانی که به صورت خطی می باشد ، حل شود .

مترجم: سهراب جلوه گر
ویرایش دوّم ، بهار ۱۳۸۸

مسأله ی بهینه سازی محدودیّت [۱]

مسأله ی ارضای محدودیت استاندارد ، مجموعه ای از متغیّرهای مربوط به هم و یک مجموعه از محدودیت هاست . مسأله ی بهینه سازی محدودیّت همان مسأله ی ارضای محدودیّت استاندارد است به اضافه ی تابع هدف نسبت دهنده (نگاشت کننده) ی هر راه حل به یک مقدار عددی . راه حلّ این نوع مسایل ، انتساب کامل محدودیت های ارضا کننده و بیشینه سازی یا کمینه سازی مسأله است . به عنوان مثال در رنگ آمیزی نقشه ، هزینه ، وابسته به رنگ های استفاده شده است و چاپ با چند رنگ دارای هزینه ی بیش تری می باشد .

الگوریتم های مسایل بهینه سازی محدودیّت

الگوریتم ساده ، همه ی راه حل ها را به حساب می آورد و از تمام روش هایی که تا کنون توضیح داده شده است استفاده می کند . الگوریتم در زمانی که تعداد زیادی راه حل وجود داشته باشد بد است (به عنوان مثال ، n – وزیر) . بهبودهای الگوریتم مسأله ی ارضای محدودیّت برای اجتناب از جستجو در شاخه

---

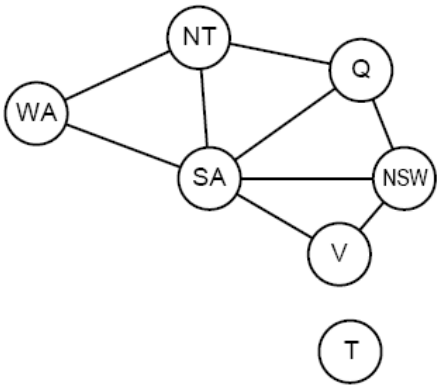
[۱] Constraint Optimisation Problem (COP)

های بدون راه حل طراحی می شود . ایده ی کلیدی این است که شاخه ها را با زیر راه حل های بهینه هرس نمایید و از مکاشفه برای هرس فضای جستجو استفاده نمایید .

ابتکارها (مکاشفه ها)

تابع هدف ، مقداردهی های کامل را ارزیابی می کند و تابع مکاشفه ای ، انتساب های جزیی را ارزیابی می نماید و به مقادیر عددی ، نسبت می دهد . در تخمین تابع هدف ، به همه ی راه حل های توسعه دهنده ی یک انتساب جزیی توجّه نمایید باید بهترین مقدار این راه حل ها را تخمین بزنند . توجّه کنید که مقداردهی های با مقادیر ابتکاری (مکاشفه ای) بد را توسعه ندهید . هرس اضافی و هرس استاندارد مسأله ی ارضای محدودیت هم می تواند به کار گرفته شود .

مسأله ی ارضای محدودیت به صورت برنامه نویسی منطقی




```
rgb(red).
rgb(green).
rgb(blue).
```