Problemas de satisfacción de restricciones

Javier Ramírez Rodríguez

Departamento de Sistemas

Universidad Autónoma Metropolitana

La programación con restricciones (PR) ha generado gran expectación entre expertos de muchas áreas debido a su potencial para la resolución de problemas reales.

La ACM la ha identificado como una de las direcciones estratégicas en la investigación informática.

La idea es resolver problemas declarando restricciones sobre la región factible del mismo y encontrar soluciones que satisfagan todas las restricciones y en su caso optimicen criterios determinados.

PR

- Satisfacción de restricciones.
- Resolución de restricciones.

Un *problema de satisfacción de restricciones* (PSR) está definido por un conjunto de variables X_1, X_2,…, X_n, y un conjunto de restricciones r_1, r_2,…,r_m.

Cada variable X_i tiene un dominio D_i de posibles valores.

Cada restricción r_i comprende algún subconjunto de variables y especifica las combinaciones de valores permisibles para cada subconjunto.

Si los dominios de las variables son discretos o continuos, finitos o infinitos, se pueden distinguir distintos tipos de PSRs, se tratarán problemas con dominios discretos y finitos.

Un estado del problema está definido por una *asignación* de valores a algunas o a todas las variables, $\{X_i=v_i, X_j = v_j,\ldots\}$.

Una asignación que no viola ninguna restricción se llama *consistente* o asignación válida.

Una asignación completa es una en la que cada variable es mencionada, una solución a un PSR es una asignación completa que satisface a todas las restricciones.

Se desea encontrar:

- Una solución, sin preferencia alguna
- Todas las soluciones
- Una óptima, o al menos una buena solución, dando alguna función objetivo definida en términos de algunas o todas las variables, en los problemas que la requieran.

El PSR discreto y finito es un problema de optimización combinatoria NP-Completo; lo que se puede probar basándose en la estrecha relación entre éste y el SAT

Algunas instancias para el PSR

Suponer que se tiene una región de la república mexicana:

Colima, Guanajuato, Jalisco, Michoacán, Nayarit y Zacatecas, se quiere pintar cada estado con rojo verde o azul de tal manera que dos estados vecinos tengan distinto color.


ESTADOS UNIDOS MEXICANOS
32° 43' 06" N
118° 27' 24" O
86° 42' 36" O
14° 32' 27" N
32° 00'
28° 00'
24° 00'
16° 00'
114° 00'
108° 00'
102° 00'
96° 00'
90° 00'
OCEANO PACIFICO
GOLFO DE MEXICO
COORDENADAS EXTREMAS

Para definir este como un PSR, se definen las variables como los estados: Col, Gto, Jal, Mich, Nay y Zac.

El dominio de cada variable es el conjunto {rojo,verde,azul}.

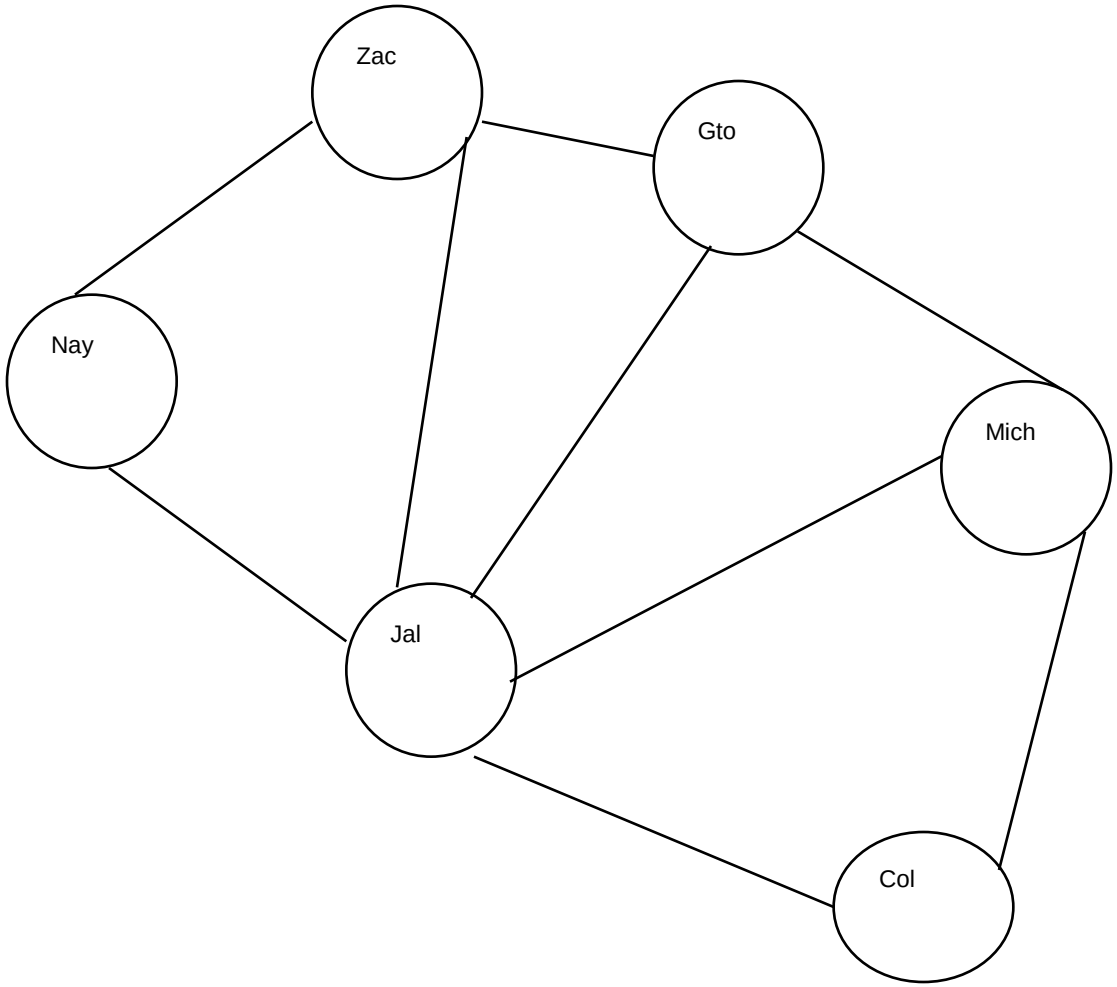
Las restricciones requieren que estados vecinos tengan distinto color.

Por ejemplo, las combinaciones válidas para Gto y Mich son {(rojo,verde), (rojo,azul), (verde,rojo), (verde,azul), (azul,rojo), (azul,verde)}.

Las restricciones se pueden representar en forma sucinta como Gto $\neq$ Mich si el algoritmo de satisfacción de restricciones tiene alguna manera de evaluar dichas expresiones. Hay muchas posibles soluciones, tales como:

- {Col=rojo, Gto=rojo, Jal=verde, Mich=azul, Nay=rojo, Zac=azul}
- {c(Col)=rojo,c(Gto)=rojo,c(Jal)=verde,...c(Zac)=azul}

Es útil visualizar a un PSR como una gráfica, los nodos corresponden a las variables del problema y las aristas a las restricciones.


Zac
Gto
Nay
Mich
Jal
Col

Ejemplo: considerar una restricción entre cuatro variables x_1, x_2, x_3, x_4, con dominios {1,2}. Donde la suma de las dos primeras es menor o igual que la suma de las dos segundas variables.

La restricción se puede representar como:

- $x_1 + x_2 \leq x_3 + x_4$

  Conjunto de asignaciones válidas

- {(1,1,1,1), (1,1,1,1,2), (1,1,2,1), (1,1,2,2), (2,1,2,2), (1,2,2,2), (1,2,1,2), (1,2,2,1), (2,1,1,2), (2,1,2,1), (2,2,2,2)}

  Conjunto de asignaciones no permitidas

- {(1,2,1,1), (2,1,1,1), (2,2,1,1), (2,2,1,2), (2,2,2,1)}

Tratar un problema como uno de SR proporciona importantes beneficios porque la representación de los estados en un PSR conforma un patrón estándar, es decir, un conjunto de variables con valores asignados, la función sucesora y la prueba meta pueden escribirse en forma genérica de tal manera que se puede usar en todos los PSRs.

Además se pueden desarrollar heurísticas genéricas efectivas que no requieren experiencia adicional sobre el dominio específico.

Finalmente, la estructura de una gráfica puede usarse para simplificar el proceso de solución, dando en algunos casos una reducción exponencial en la complejidad.

Se puede ver que a un PSR se le puede dar una formulación incremental como un problema de búsqueda estándar como sigue:

- *Estado inicial*: la asignación vacía { }, en la cual ninguna variable tiene asignación.
- *Función sucesora*: se le puede asignar un valor a cualquier variable que no tenga asignación, teniendo en cuenta que no está en conflicto con las asignaciones hechas previamente a otras variables.
- Prueba meta: la asignación actual es completa.
- Costo del camino: un costo constante para cada etapa, *e.g.*, 1.

La clase más simple de PSR comprende variables discretas y todas tienen dominios finitos, problemas de coloración de mapas son de esta clase, el problema de las 8 reinas también se puede ver como un PSR con dominio finito, donde las variables r_1,…, r_8 son las posiciones de cada reina en las columnas 1,…,8 y cada variable tiene dominio {1, 2, 3, 4, 5, 6, 7, 8}.

Un caso especial prominente de problemas de coloración de gráficas es el Problema de Cumplimiento Cuasi Grupos (PCCG) el cual es derivado del problema de cuadrados latinos (PCL): Dada una malla cuadriculada $n \times n$ y n colores, el objetivo es asignar un color a cada celda de tal manera que cada renglón y cada columna contenga los n colores.

En el PCCG el objetivo es decidir si una solución parcial del PCL, que es una asignación incompleta de colores a la cuadrícula dada, tales que dos celdas en el mismo renglón o misma columna no tienen el mismo color, puede extenderse a una solución completa, asignando colores al resto de las celdas.

En la formulación del PSR las celdas pre asignadas pueden representarse por restricciones unitarias.

Se sabe que el PCCG es NP-Completo y tiene aplicaciones en áreas tales como longitud de onda dinámica en redes de fibra óptica y diseño estadístico de experimentos.

Si el máximo tamaño del dominio de cualquier variable en un PSR es d, entonces el número de asignaciones completas es $O(d^m)$, *i.e.*, exponencial en el número de variables.

PSR de dominio finito incluyen PSRs booleanos, cuyas variables pueden ser ciertas o falsas, que incluyen como casos especiales algunos problemas NP-Completos, tales como el 3SAT.

Sin embargo, en el peor caso no se puede esperar resolver PSRs con dominio finito en menos de tiempo exponencial.

Las variables discretas también pueden tener dominios infinitos *e.g.*, el conjunto de enteros.

Al planificar tareas en un calendario, la fecha de inicio de cada tarea es una variable y los valores posibles son números enteros de días desde la fecha actual.

Con dominios infinitos, no es posible describir restricciones enumerando todas las combinaciones de valores permitidas, en su lugar se debe usar una restricción de lenguaje.

Por ejemplo, si la $tarea_1$ que dura 5 días, debe preceder a la $tarea_3$ se necesita una restricción de lenguaje de desigualdades algebraicas tales como

$$InicioTarea_1 + 5 \leq InicioTarea_3.$$

Tampoco es posible resolver tales restricciones enumerando todas las posibles asignaciones, porque hay un número infinito de éstas.

PSR con dominios continuos son muy comunes en el mundo real y son ampliamente estudiados en Investigación de Operaciones, por ejemplo, la planificación de experimentos en los telescopios requieren tiempos de observación muy precisos el inicio y fin de cada observación y maniobra son variables continuas.

La categoría mejor conocida de PSRs es la programación lineal donde las restricciones deben ser desigualdades lineales formando una región convexa.

Además de examinar los tipos de variables que pueden aparecer en los PSRs, es útil ver el tipo de restricciones.

La restricción más simple es la restricción unaria, que restringe el valor de una variable. Podría ser que a los de Guanajuato (Gto) no les guste el color *rojo*.

Cada restricción unaria se puede eliminar por pre proceso del dominio de la variable correspondiente removiendo cualquier valor que viole la restricción.

Una restricción binaria relaciona dos variables, por ejemplo, Gto $\neq$ Mich es una restricción binaria.

Un PSR binario es uno que solo tiene restricciones binarias y se puede representar como una gráfica de restricciones.

Resolución de un PSR

La resolución de un PSR consta de dos fases:

- Modelar el problema como uno de satisfacción de restricciones. La modelación expresa el problema mediante un conjunto de variables, dominios y restricciones del PSR.
- Procesar el problema de satisfacción de restricciones resultante, para lo que hay dos formas.

1. Técnicas de consistencia. Basadas en la eliminación de valores inconsistentes de los dominios de las variables, en general se combinan con las técnicas de búsqueda, ya que reducen el espacio de soluciones .
2. Algoritmos de búsqueda. Se basan en la exploración sistemática del espacio de soluciones hasta encontrar una solución o probar que no existe alguna.

Búsqueda en PSRs

El conjunto formado por todas las posibles asignaciones de un PSR se representa mediante un árbol de búsqueda.

En cada nivel se asigna un valor a una variable, y los sucesores de un nodo son todos los valores de la variable asociada a este nivel.

Cada camino del nodo raíz a un nodo terminal representa una asignación completa.

Los algoritmos se diferencian en cómo recorren el árbol de búsqueda a la hora de encontrar soluciones: búsqueda hacia atrás, salta hacia atrás, salta hacia atrás directo a conflicto, etc.

Se dio una formulación de los PSRs como problemas de búsqueda, suponer que se aplica el método de búsqueda a lo ancho en el problema genérico formulado en la sección anterior.

Rápidamente se nota algo terrible: el factor de ramificación al inicio es nd, porque cualquiera de los d valores se puede asignar a cualquiera de las n variables;

en el siguiente nivel, el factor de ramificación es $(n-1)d$, y así para los n niveles.

Se genera un árbol con $n! \cdot d^n$ hojas, sin embargo, hay solo d^n asignaciones posibles.

La formulación aparentemente razonable ignoró una propiedad crucial común a todos los PSRs: conmutatividad.

Un problema es conmutativo si el orden de aplicación de cualquier conjunto dado de acciones no tiene efecto en el resultado.

Por lo tanto todos los algoritmos de búsqueda en PSRs generan sucesores considerando posibles asignaciones para una sola variable en cada nodo en el árbol de búsqueda.

Por ejemplo, en el nodo raíz de un árbol de búsqueda para colorear la región considerada se puede escoger entre *Gto = azul, Gto = rojo, Gto = verde*, pero nunca se escogerá entre *Gto = rojo* y *Mich = verde*. Con esta restricción el número de hojas es d^n, como se esperaba.

El término búsqueda hacia atrás es usado para una búsqueda a profundidad que escoge valores para una variable cada vez y regresa cuando no hay valores legales para asignar a una variable.

Se hace notar que el algoritmo usa el método de generación de sucesores uno cada vez.

function búsqueda hacia atrás(PSR) **return** una solución

return regreso-recursivo({ },psr)

function regreso-recursivo(asignación,psr) **return** una solución, o no la encuentra

if se completa asignación **then return** asignación

var ← seleccionar variable no asignada (variables [psr],asignación,psr)

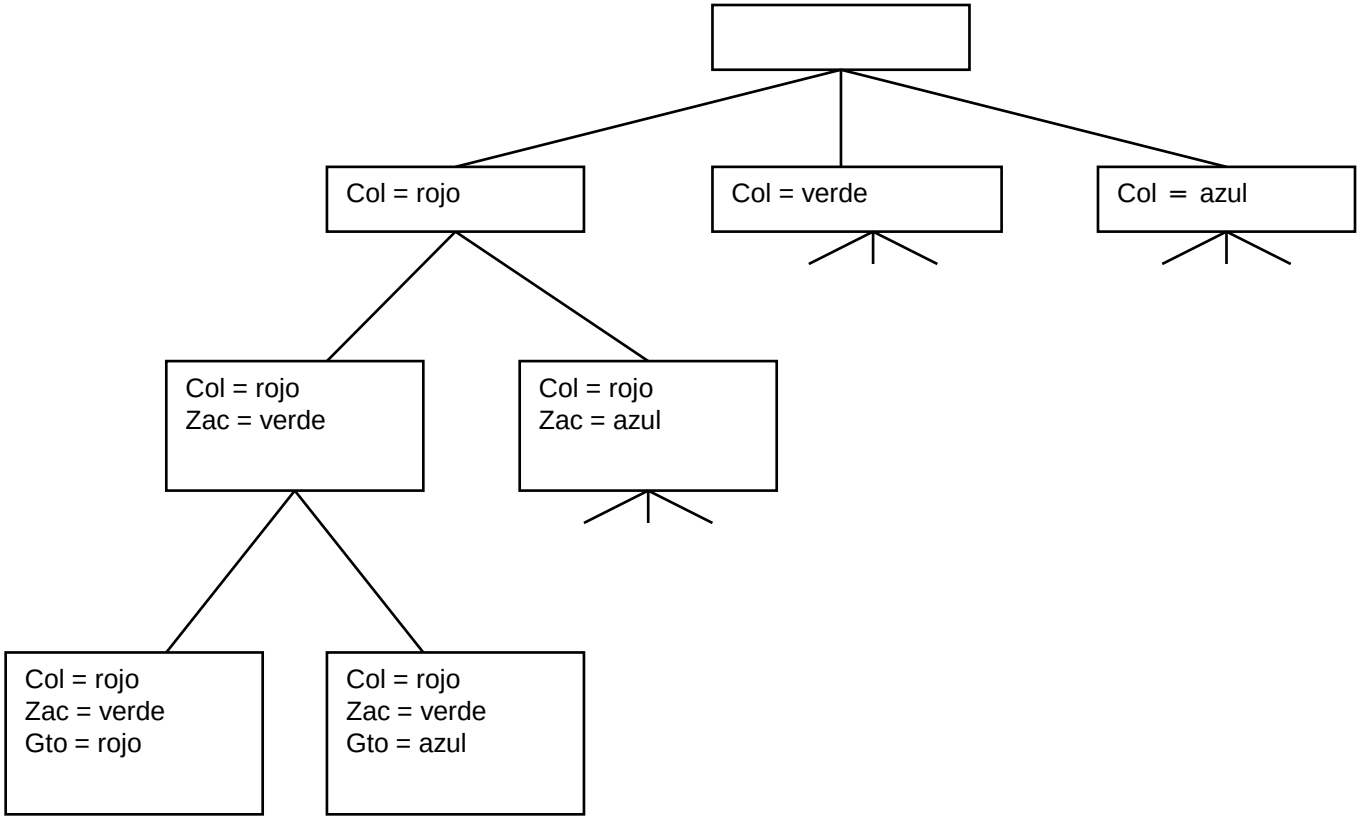
for each valor **in** valores del dominio(*var*, asignación, psr) **do**

if valor consistente con la asignación de acuerdo a restricciones[psr]**then** agregar{*var*=valor} a la asignación
resultado ← regreso-recursivo (asignación, psr)

if resultado ≠ falla **then return** resultado
remover {*var* = valor} desde asignación

return fallo

Algoritmo simple para el PSR


Col = rojo
Col = verde
Col = azul
Col = rojo
Zac = verde
Col = rojo
Zac = azul
Col = rojo
Zac = verde
Gto = rojo
Col = rojo
Zac = verde
Gto = azul

Regreso total es un algoritmo no informado, en el sentido de que lo único que sabe del problema es su definición, por lo que no se espera que sea eficiente en problemas grandes.

La baja eficiencia de estos algoritmos se puede aliviar proporcionándoles funciones heurísticas de dominio específico derivadas del conocimiento que se tenga del problema.

Se pueden resolver PSRs en forma eficiente sin tal conocimiento de dominio específico.

Se encuentran métodos de propósitos generales que hacen las siguientes preguntas:

1. ¿Cuál variable deberá ser asignada y en qué orden serán probados sus valores?

2. ¿Cuáles son las implicaciones de tal asignación para las variables no asignadas?

3. ¿Cuándo un camino falla?, *i.e.*, se llega a un estado en el cual la variable no puede tomar algún valor.

Variable y ordenación de valores

El algoritmo hacia atrás contiene la línea

var ← *Seleccionar-Variable-no-Asignada*(variables[psr],asignación,psr)

que selecciona la siguiente variable no asignada en el orden dado por Variables[psr]. Este orden estático de las variables raras veces es la búsqueda más eficiente.

Se escoge la variable con el menor número de valores posibles, la heurística del mínimo de los valores restantes (MVR) o la variable más restringida o la heurística del primero que falla.

Si hay una variable X con cero valores válidos, la heurística MVR seleccionará a X y la falla se detectará enseguida.

Evitando búsquedas en otras variables que siempre fallarán cuando X sea seleccionada.

La heurística MVR no ayuda a escoger el primer estado a colorear. En este caso la heurística del grado resulta útil.

Intenta reducir el factor de ramificación en selecciones futuras escogiendo la variable que está involucrada en el mayor número de restricciones sobre otras variables no asignadas.

La heurística MVR normalmente es más poderosa pero la heurística del grado puede usarse para romper empates.

Una vez que una variable ha sido seleccionada, el algoritmo debe decidir el orden en el cual examinar sus valores.

Para esto, la heurística del valor menos restringido puede ser efectiva en algunos casos, prefiere el valor que evita la menor selección para las variables vecinas en la gráfica de restricciones

Ejemplo. Suponer que se ha generado la asignación parcial Nay = rojo, Zac = verde y que la siguiente selección es para Gto, azul sería una mala selección porque elimina el último valor válido para el vecino Jal; por lo tanto la heurística del valor menos restringido prefiere el rojo al azul.

Propagación de información a través de restricciones

Hasta ahora el algoritmo de búsqueda considera las restricciones sobre una variable solo en el momento en que ésta es escogida por *Seleccionar-Variable-no-asignada*; pero mirando antes en algunas restricciones de la búsqueda o aún antes de que ésta haya empezado, se puede reducir drásticamente el espacio de búsqueda.

Una forma de hacer mejor uso de las restricciones durante la búsqueda se llama inspección hacia delante. Siempre que una variable X es asignada, el proceso de inspección hacia delante mira a cada variable Y no asignada que está conectada a X por una restricción y borra del dominio de Y cualquier valor que es inconsistente para el valor escogido para X.

La siguiente figura muestra el progreso de búsqueda de coloración de un mapa con inspección hacia delante:

	Nay	Zac	Gto	Mich	Col	Jal
Dominio Inicial	RVA	RVA	RVA	RVA	RVA	RVA
Después Nay=Rojo	R	VA	RVA	RVA	RVA	VA
Después Gto=Verde	R	A	V	RA	RVA	A
Después Col=Azul	R	A	V	R	A	

Hay dos puntos importantes a notar en este ejemplo: Primero, después de asignar Nay = rojo y Gto = verde los dominios de Zac y Jal son reducidos a un solo valor, se ha eliminado la ramificación de estas variables juntas propagando información desde Nay y Gto. La heurística MVR que es un medio natural para inspeccionar hacia delante, seleccionará automáticamente Jal y Zac enseguida.

Segundo después de que Col = azul el dominio de Jal está vacío, así, inspeccionando hacia delante ha detectado que la asignación parcial {Nay=rojo, Gto=verde,Col=azul} es inconsistente con las restricciones del problema y el algoritmo regresará inmediatamente.

Propagación de restricciones

Aunque inspeccionado hacia delante detecta inconsistencias, no detecta a todas.

Ejemplo, el cuarto renglón de la tabla muestra que cuando Nay=rojo y Gto = verde, Zac y Jal son forzados a ser ambos azules, pero son adyacentes y por lo tanto no pueden tener el mismo color.

Esta inconsistencia no es detectada porque no mira suficientemente lejos.

Propagación de restricciones es el término general para propagar la implicación de una restricción para una variable sobre las otras.

En este caso se necesita propagar desde Nay y Gto sobre Zac y Jal y después sobre la restricción de Zac y Jal para detectar la inconsistencia.

Se quiere hacer esto rápido, no es bueno reducir el tiempo de búsqueda si se gasta más tiempo propagando restricciones que haciendo una simple búsqueda.

La idea de consistencia de arcos proporciona un método rápido de propagación de restricciones que es más fuerte que inspección hacia delante.

Dados los dominios actuales de Jal y Mich, el arco es consistente para cada valor *x* de Jal, hay algún valor *y* de Mich que es consistente con *x*. En el cuarto renglón de la tabla los dominios de Jal y Mich son {azul} y {rojo, azul} respectivamente.

Para Jal = azul, hay una asignación consistente para Mich=rojo; por lo tanto el arco de Jal a Mich es consistente. Por otro lado, el arco de Mich a Jal no es consistente

También se puede aplicar consistencia de arcos al que va de Jal a Zac en la misma etapa en el proceso de búsqueda, el cuarto renglón de la tabla muestra que ambas variables tienen dominio {azul}.

El resultado es que azul debe borrarse del dominio de Jal dejándolo vacío.

Aplicando la consistencia de arcos ha resultado en la detección temprana de una inconsistencia que no es detectada por inspección hacia delante.

función CA (*psr*) **regresa** el PSR posiblemente con dominios reducidos

inputs: *psr*, un PSR con variables $\{X_1, X_2, ..., X_n\}$

variables locales: *fila*, una fila de aristas, inicialmente todas las aristas en psr

mientras *fila* no es vacía **hacer**

$(X_i, X_j) \leftarrow$ **Remover**(*fila*)

si Remover-Valores-Inconsistentes(X_i, X_j) **entonces**

para cada X_k en Vecinos[X_i] - {X_j} **hacer**

agregar (X_k, X_i) a la *fila*

función Remover-Valores-Inconsistentes(X_i, X_j) **regresa** cierto ssi se ha retirado un valor

retirado ← *falso*

para cada *x* **en** dominio de [X_i] **hacer**

si no hay *y* en dominio [X_j] (*x*,*y*) que satisface la restricción entre X_i y X_j

entonces borrar x de dominio [X_i]; *retirado* ← *cierto*

regresa *retirado*

Cada arco (X_i, X_j) en su turno es retirado de la agenda e inspeccionado, si es necesario borrar algunos valores del dominio de X_i entonces cada arco (X_k, X_i) debe insertarse en la fila para inspección.

La complejidad de inspección de consistencia de arcos puede analizarse como sigue: un PSR binario tiene a lo más $O(n^2)$ aristas, cada arista (X_k, X_i) puede ser incluida en la agenda sólo *d* veces porque X_i tiene a lo más d valores para borrar.

Verificar inconsistencia de una arista puede hacerse en un tiempo $O(d^2)$; por lo que el tiempo para el peor caso es $O(n^2d^3)$.

Debido a que el PSR incluye al 3SAT como un caso especial, no se espera encontrar un algoritmo polinomial que pueda decidir si un PSR es consistente.

De aquí, se puede deducir que consistencia de arcos no revela cada inconsistencia.

Ejemplo, la asignación parcial {Nay = rojo, Mich = rojo} es inconsistente pero el algoritmo no la revela.

Se pueden definir formas de propagación más fuertes usando k-consistencia.

Un PSR es k-consistente si para cada conjunto de k-1 variables y para cualquier asignación consistente a esas variables, siempre se puede asignar un valor consistente a cualquier k-ésima variable

Ejemplo, 1-consistencia significa que cada variable es consistente, 2-consistencia es lo mismo que consistencia de arcos. 3-consistencia significa que cualquier par de variables adyacentes siempre se puede ampliar a una tercera variable vecina, esto también se llama consistencia de caminos.

Una gráfica es fuertemente k-consistente si es k-consistente, (k-1)-consistente,..., 1-consistente.

En un problema de este tipo se puede encontrar una solución en un tiempo O(*nd*).

Hay una gran diferencia entre n-consistencia y consistencia de arcos: usar verificación de consistencia fuerte puede tomar más tiempo pero tendrá mayor efecto al reducir el factor de ramificación detectando asignaciones parcialmente inconsistentes.

Manejo de restricciones especiales

Cierto tipo de restricciones ocurre frecuentemente en problemas reales y pueden tratarse utilizando algoritmos con propósitos especiales que son más eficientes que los métodos de propósitos generales descritos.

Por ejemplo, las restricciones *todasdif* dicen que todas las variables consideradas deben tener distintos valores.

Una forma simple de detectar inconsistencia en estas restricciones es la siguiente: si se consideran m variables en las restricciones y tienen n posibles valores distintos y $m > n$, entonces las restricciones no se pueden satisfacer.

Esto da lugar al siguiente algoritmo:

- Remover cualquier variable que tiene dominio con un elemento y borrarlo del dominio de las variables restantes.

- Repetir el paso anterior mientras haya variables que tienen dominio con un elemento.

Si en algún momento se produce un dominio vacío o hay más variables que valores de dominio a retirar, entonces se ha detectado una inconsistencia.

Se puede usar este método para detectar inconsistencia en la asignación parcial {Nay = rojo, Mich = rojo}, notar que las variables Jal, Zac y Gto están efectivamente conectadas por una restricción *todasdif* porque cada par debe ser de color diferente.

Después de aplicar CA en la asignación parcial, el dominio de cada variable se reduce a {verde, azul}, i.e., se tienen tres variables y solo hay dos colores, la restricción *todasdif* es violada.

Un procedimiento de consistencia para restricciones de orden mayor es algunas veces más efectivo que aplicar consistencia de arista a un conjunto equivalente de restricciones binarias.

Quizá la restricción de orden mayor más importante es la restricción de recursos.

Ejemplo: sean PA_1, ..., PA_4 el número de personas asignadas a cada una de las cuatro tareas, la restricción de que no se debe asignar en total más de 10 personas se escribe como *alomas*(10, PA_1, PA_2, PA_3, PA_4).

Una inconsistencia se puede detectar verificando la suma de los valores mínimos de los dominios actuales, *e.g.*, si cada variable tiene dominio {3,4,5,6} la restricción no se puede cumplir.

Se puede reforzar la consistencia borrando el valor máximo de cualquier dominio si no es consistente con los valores mínimos de los otros dominios.

Si cada variable tiene dominio {2,3,4,5,6} los valores 5 y 6 se pueden borrar de cada dominio.

Para problemas grandes de recursos limitados con valores enteros, tales como problemas logísticos que comprenden mover miles de personas en cientos de vehículos, normalmente no es posible representar el dominio de cada variable como un conjunto grande de enteros y gradualmente reducirlos verificando consistencia.

Los dominios son representados por cotas inferiores y superiores y se manipulan por propagación de cotas.

Ejemplo, suponer que hay dos vuelos 271 y 172 cuyos aviones tienen capacidades de 165 y 385 pasajero respectivamente; el dominio inicial para cada vuelo es

$$vuelo271 \in [0,165] \quad vuelo272 \in [0,385]$$

Suponer que tienen una restricción adicional de que los dos vuelos deben llevar 420 pasajeros

$$vuelo271 + vuelo272 \in [420,420]$$

propagando restricciones de cotas, los dominios se reducen a

$$vuelo271 \in [35,165] \quad vuelo272 \in [255,385]$$

se dice que el PSR es cotas-consistente para cada variable X y para valores de X en ambas cotas hay algún valor de Y que satisface la restricción entre X y Y, para cada variable Y.

Regresos inteligentes

El algoritmo Búsqueda-hacia-atrás, tiene una política muy simple para lo que hay que hacer cuando la modificación falla: regresar a la variable precedente e intentar diferentes valores para ésta, lo que se llama regreso cronológico, porque se vuelve a visitar el punto donde se tomó la decisión más reciente, se verá que hay mejores formas para esto.

Considerar lo que sucede cuando se aplica el regreso simple en el problema de coloración al que se ha agregado Tlaxcala (Tlax), dado un orden de las variables Gto, Mich, Col, Tlax, Jal, Nay, Zac.

Suponer que se ha generado la asignación parcial {Gto = rojo, Mich = verde, Col = azul, Tlax = rojo}. Cuando se intenta la siguiente variable, Jal, se ve que cada valor viola una restricción. Regresando a Tlax e intentando un nuevo color para ésta, no resuelve el problema con Jal.

Un regreso inteligente es recorrer todo el camino de un conjunto de variables que causó la falla, este conjunto se llama *conjunto conflicto* (CC) aquí el CC para Jal es {Gto, Mich, Col}.

Generalmente el CC para una variable es el conjunto de variables previamente asignadas que están conectadas a X por restricciones.

Búsqueda local para PSRs

Los algoritmos de búsqueda local son muy eficientes para resolver algunos problemas de satisfacción de restricciones.

Usan una formulación completa de estados: el estado inicial asigna un valor a cada variable y la función que sigue normalmente cambia el valor de una variable cada vez.

Por ejemplo, en el problema de las ocho reinas el estado inicial puede ser una configuración de 8 reinas en 8 columnas y la función siguiente escoge una reina y considera moverla a algún otro lugar de la columna.

Otra posibilidad sería, empezar con las 8 reinas, una por columna en una permutación de los 8 renglones y generar un sucesor intercambiando las de 2 renglones.

Para escoger un nuevo valor para una variable, la heurística más obvia es seleccionar el valor que resulta en el mínimo número de conflictos con otras variables, algoritmo *conf-min*.

función conf-min(psr,max-etapas) **regresa** una solución o falla

entradas: psr, un PSRs

actual ← asignación inicial completa para psr

para i = 1 a max-etapas **hacer**

si actual es una solución para psr **entonces** regresa actual

var ← selección aleatoria de variable conflictiva de Variable[psr]

valor← el valor v para *var* que minimice Conflictivas(var,v,actual,psr)

hacer var = valor en actual

regresar falla

Estructura de problemas

Se analizan formas que pueden usarse para representar al problema para poder encontrar soluciones rápidamente.

Si se considera Tlax en el problema de coloración, bastaría ver la gráfica de restricciones (estructura) para darse cuenta que Tlax no está conectado a la otra región, así que colorear Tlax y después la otra región son subproblemas independientes.

La independencia se puede identificar viendo las componentes conexas en la gráfica de restricciones.

Problemas completamente independientes son apreciados, pero son raros, en la mayoría de los casos están conectados.

El caso más simple es cuando la gráfica de restricciones forma un árbol, cualquier par de variables está conectado por un solo camino.

Se demostrará que cualquier PSR árbol-estructurado puede resolverse en tiempo lineal en el número de variables. El algoritmo tiene las siguientes etapas:

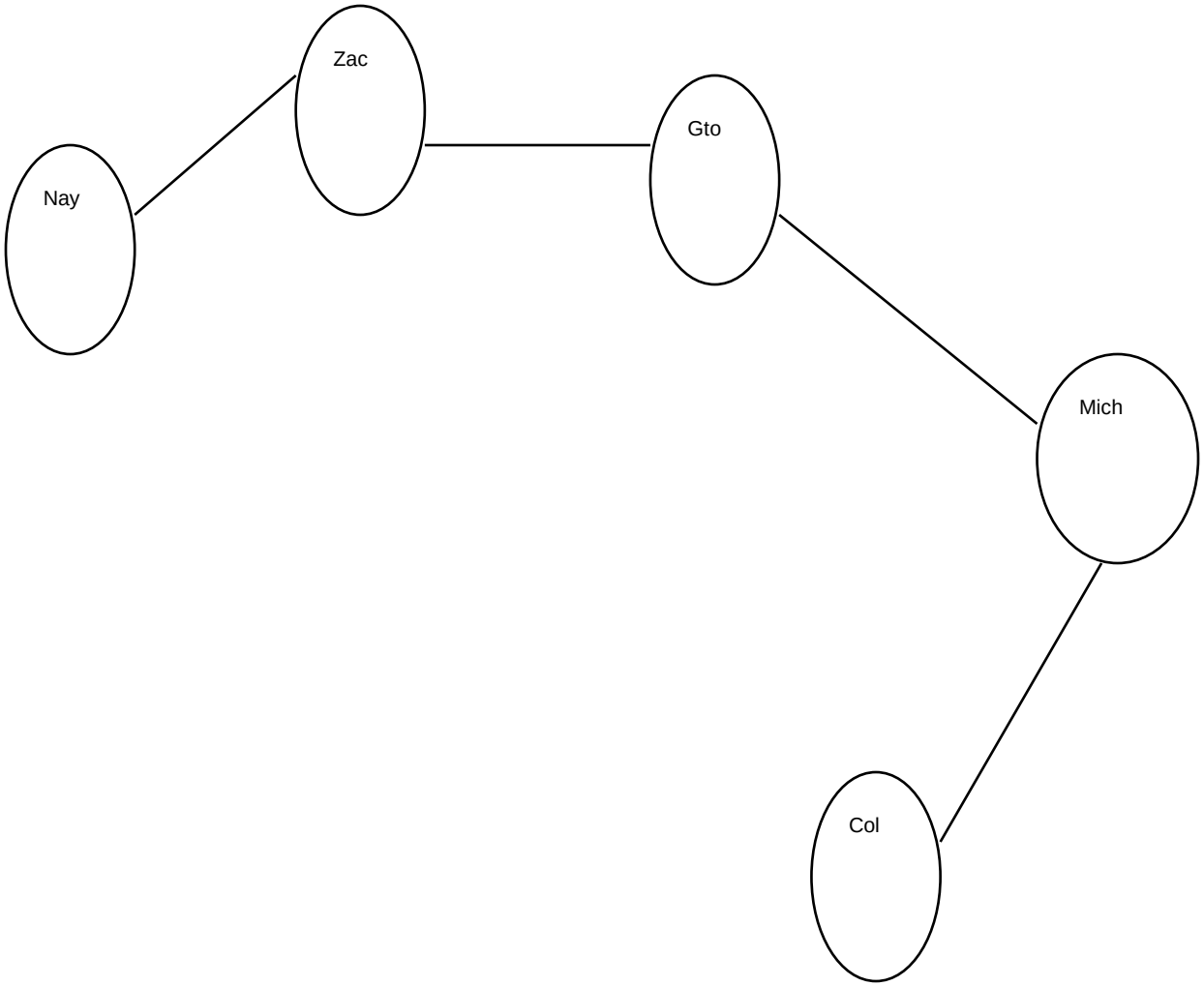
1. Escoger cualquier variable como raíz del árbol y ordenar las variables de la raíz a las hojas de tal manera que cada padre preceda a sus hijos en el árbol. Etiquetar las variables X_1, ..., X_n en orden. Cada variable excepto la raíz tiene exactamente una variable padre.
2. Desde J = n, 2, aplicar arco consistencia a (X_i,X_j), donde X_i es el padre de X_j, retirar valores de Dominio[X_i] tanto como sea necesario.
3. Para j = 1,n asignar cualquier valor a X_j consistente con los valores asignados a X_i, donde X_i es el padre de X_j.

El algoritmo completo corre en un tiempo $O(nd^2)$. Se puede considerar si gráficas de restricciones más generales se pueden reducir a árboles de alguna manera.

Hay dos formas de hacer esto, una basada en quitar nodos y otra basada en juntar nodos.

El primer enfoque consiste en asignar valores a algunas variables de tal manera que las restantes formen un árbol.

Si en la gráfica de restricciones del problema de coloración se puede retirar Jal, la gráfica sería un árbol


Zac
Nay
Gto
Mich
Col

Cualquier solución para el PSR después de retirar Jal y sus restricciones será consistente con el valor asignado a Jal.

Esto funciona para PSRs binarios, la situación es más complicada con restricciones de orden mayor. Por lo tanto se puede resolver el árbol restante con el algoritmo dado y con esto resolver el problema.

El color escogido para Jal podría ser el equivocado, por lo que habría que intentarlo con cada uno.

El algoritmo es el siguiente:

1. Escoger un subconjunto S de variables[psr] tal que la gráfica de restricciones se convierta en árbol después de retirar a S. S se llama conjunto corte ciclo.
2. Para cada posible asignación de las variables en S que satisfaga todas las restricciones en S
   a. Retirar de los dominios de las variables restantes los valores que son inconsistentes con la asignación de S, y
   b. Si el PSR restante tiene una solución, darla con la asignación de S.

Si el conjunto corte ciclo tiene tamaño c, entonces el algoritmo corre en un tiempo $O(d^c \cdot (n-c)d^2)$. Si la gráfica es casi un árbol entonces c será pequeño y lo guardado en los regresos directos será grande.

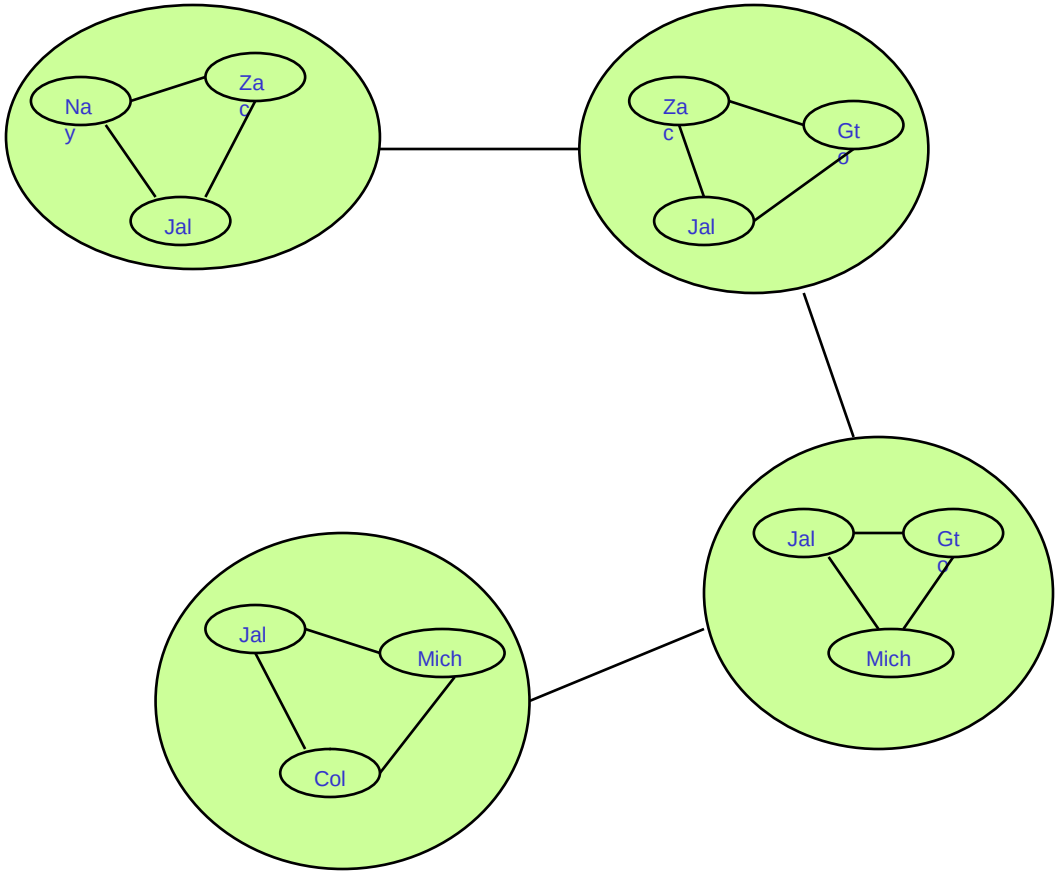
En el peor caso, c puede ser tan grande como (n-2). Encontrar el conjunto corte ciclo más pequeño es NP-Completo, pero hay varios algoritmos de aproximación eficientes.

El segundo enfoque está basado en la construcción de un árbol de descomposición de la gráfica de restricciones en un conjunto de sub problemas conectados.

Cada sub problema es resuelto independientemente y las soluciones resultantes se combinan.

Como la mayoría de los algoritmos divide y vencerás, esto funciona si los sub problemas no son muy grandes.

En la siguiente figura se muestra la descomposición árbol del problema de coloración en 4 sub problemas


Nay
Zac
Jal
Zac
Gto
Jal
Jal
Gto
Mich
Jal
Mich
Col

Se resuelve cada problema en forma independiente, si alguno no tiene solución, se sabe que el problema no tiene solución.

Si se pueden resolver todos los sub problemas, se intenta construir una solución global como sigue, primero se ve a cada sub problema como una mega-variable cuyo dominio es el conjunto de todas las soluciones para el sub problema.

Por ejemplo, cualquier sub problema de la figura 4 es uno de coloración de un mapa con 3 variables y 6 soluciones, una es {Nay = rojo, Jal = azul, Zac = verde}. Se resuelven las restricciones conectando los sub problemas y usando el algoritmo eficiente para árboles.

Bibliografía:

1. Barber F. Salido M A. Introducción a la programación de restricciones. http://users.dsic.upv.es/~msalido/papers/aepia-introduccion.pdf.
2. Larrosa J. Meseguer P. (2003). Algoritmos para satisfacción de restricciones. Revista Iberoamericana de Inteligencia Artificial, Volume Otoño, Number 20, p.31-42.
3. Manyà F. Gomes C. Técnicas de resolución de problemas de satisfacción de restricciones. Revista Iberoamericana de Inteligencia Artificial, No. 19, pp. 169-180.