Solución de Problemas con CCP
Fortaleza de la propagación

slides basados en el curso “constraint Programming” de
Christian Schulte [2]
Profesor: Camilo Rueda [1]

[1]Universidad Javeriana-Cali,
[2]KTH Royal Institute of Technology, Sweden

PUJ 2008

Dominio Consistencia

Un propagador p con $var(p) = (x_1, ..., x_k)$
es dominio-consistente para la restricción c
si para todo store no fallido s, cumple que
  si el valor $v_i \in p(s)(x_i)$
  entonces existen valores $v_j \in p(s)(x_j)$ (para $i \neq j$) tales que
    $(v_1, ..., v_k) \in sol(c)$

Dominio Consistencia: ejemplo

- Restringirse a valores en las soluciones
- Ejemplo: sea $x = 3y + 5z$

  $s(x) = \{2, ..., 7\}$ $s(y) = \{0, ..., 2\}$ $s(z) = \{-1, ..., 2\}$
- Soluciones

  (x, y, z) : $(3, 1, 0), (5, 0, 1), (6, 2, 0)$
- store resultante

  $s'(x) = \{3, 5, 6\}$

  $s'(y) = \{0, 1, 2\}$

  $s'(z) = \{0, 1\}$

LímiteConsistencia

Sea el universo de valores los conjuntos finitos de enteros ($\mathbb{F}$)

- Rangos:

$$rango \in 2^{\mathbb{F}} \rightarrow 2^{\mathbb{F}}$$
$$rango(c) = \{min(c), ..., max(c)\}$$

- Ejemplo,
  $rango(\{1, 3, 5\}) = \{1, 2, 3, 4, 5\}$
- Ejemplo: sea $x = 3y + 5z$
  $s(x) = \{2, ..., 7\}$ $s(y) = \{0, ..., 2\}$ $s(z) = \{-1, ..., 2\}$
- Un subconjunto c de $\mathbb{F}$ es un rango si
  $rango(c) = c$

LímiteConsistencia: definición

Un propagador p con $var(p) = (x_1, ..., x_k)$
es límite-consistente para la restricción c
si para todo store no fallido s, cumple que
si el valor $v_i \in \{min(\ p(s)(x_i)\ ), max(\ p(s)(x_i)\ )\}$
entonces existen valores $v_j \in rango(\ p(s)(x_j)\ )$ (para $i \neq j$)
tales que

$$(v_1, ..., v_k) \in sol(c)$$

LímiteConsistencia: propiedades

- Un propagador dominio-consistente es idempotente
- Un propagador límite-consistente no necesariamente es idempotente

Tarea: pruebe esto!

Modelar: restricción “element”

- Suponga que las variables modelan lugares en una bodega
  - los valores modelan el producto almacenado en ese lugar
  - productos diferentes tiene precios diferentes
- Cómo propagar el precio mientrasque la variable no tiene todavía asignado un producto?
- Muy común: mapear variable con variable, de acuerdo a los valores

Modelar: restricción “element”, ejemplo

- Suponga que los productos son $0, 1, 2, 3$
- Precios:
  - producto 0 precio 10
  - producto 1 precio 15
  - producto 2 precio 5
  - producto 3 precio 12

Ejemplo, reificando

$$
\begin{aligned}
&\mathit{BoolVar}\ b0 = \mathit{new}\ \mathit{BoolVar}(\mathit{this}\ );\\
&\ldots\\
&\mathit{rel}\ (\mathit{this}\ , g, \mathit{IRT\_EQ}, 0, b0);\\
&\mathit{rel}\ (\mathit{this}\ , p, \mathit{IRT\_EQ}, 10, b0);\\
&\mathit{rel}\ (\mathit{this}\ , g, \mathit{IRT\_EQ}, 1, b1);\\
&\mathit{rel}\ (\mathit{this}\ , p, \mathit{IRT\_EQ}, 15, b1);\\
&\mathit{rel}\ (\mathit{this}\ , g, \mathit{IRT\_EQ}, 2, b2);\\
&\mathit{rel}\ (\mathit{this}\ , p, \mathit{IRT\_EQ}, 5, b2);\\
&\mathit{rel}\ (\mathit{this}\ , g, \mathit{IRT\_EQ}, 3, b3);\\
&\mathit{rel}\ (\mathit{this}\ , p, \mathit{IRT\_EQ}, 12, b3);\\
&b0 + b1 + b2 + b3 = 1;
\end{aligned}
$$

- Tedioso: varios productos pueden tener el mismo precio...
- Ineficiente: muchos propagadores
- Propagación: si llegan a punto fijo. es dominio-consistente

La restricción “element”

- Restricción Element, $a[[x]] = y$
  - Arreglo de enteros a
  - Variables x, y
  - el valor de y es el valor de la x-ésima posición de a
  - en particular, $0 \leq x \leq$ elementos en a
- En Gecode
  - *element*(*this* , a, x, y);
  - también para arreglo de variables

Modelar con “element”

$$\begin{array}{l} int\ \ precios[]\ =\ \{10, 15, 5, 12\} \\ element(this\ ,\ precios,\ g,\ p); \end{array}$$

- Un solo propagador!
- no hay problema si el mismo entero ocurre varias veces en el arreglo

Propagar “element”

- preferible dominio-consistencia
  - límite-consistencia demasiado débil
- Para $a[[x]] = y$ con un store s, propague
  - si $j \in s(y)$ entonces mantenga todo k de $s(x)$ tal que $j = a[k]$
  - si $k \in s(x)$ entonces mantenga todo j de $s(y)$ tal que $j = a[k]$
  - elimine todos los demás valores

Implementar “element”

- Requisito fundamental: nuevos dominios deben calcularse en orden
- Iterar sobre elemetos $k \in s(x)$

  $$\{a[k] \mid k \in s(x)\} \cap s(y)$$
- iterar k de 0 a $n-1$ (tamaño del vector a):
  construir nuevo dominio para x
  - si $a[k] \in s(y)$ entonces mantenga todo k
  - requiere intersección y ordenamiento

Problemas de implemetación de “element”

- Los arreglos en las aplicaciones tienden a ser muy grandes
  - siempre iterar sobre todo el arreglo
  - siempre ordenar (o mantener ordenado)
  - siempre calcular la intersección
- Se puede hacer mejor

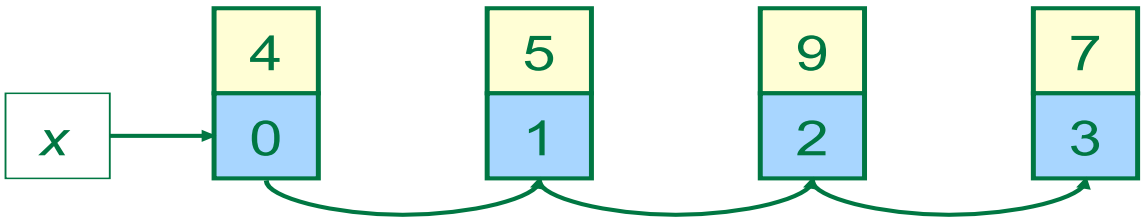
Mejora de “element”, ejemplo

- Considere $a[[x]] = y$, con
  - $a = (4, 5, 9, 7)$
  - $s(x) = \{1, 2, 3\}$
  - $s(y) = \{2, ..., 8\}$
- La propagación obtiene
  - $s(x) = \{1, 3\}$
  - $s(y) = \{5, 7\}$

Mejora de “element”, método

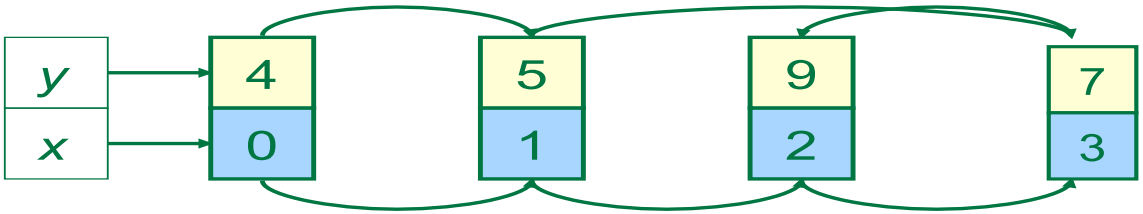
- Construya estructura de datos
  - contiene parejas $(i, a[i])$
  - permite recorrido en orden creciente de i
  - permite recorrido en orden creciente de $a[i]$
  - permite eliminar parejas
- La estructura de datos se construye inicialmente

Construcción de la estructura



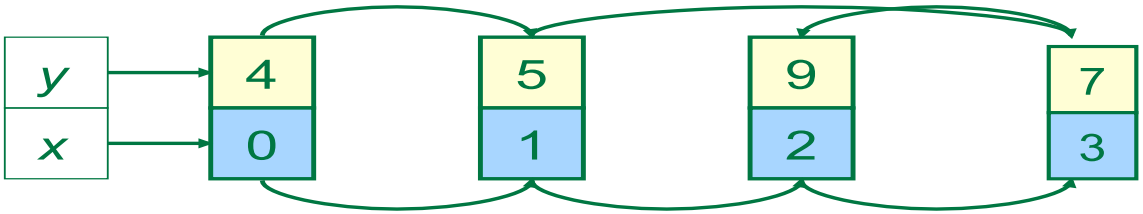
- Itere sobre todo i entre 0 y 3
  - crear nodo $(i, a[i])$
  - crear links en orden de creación (x-links)

Construcción de la estructura



- Crear links para valores de $a[i]$ en orden creciente (y-links)
  - ordenar y crear links

Construcción de la estructura



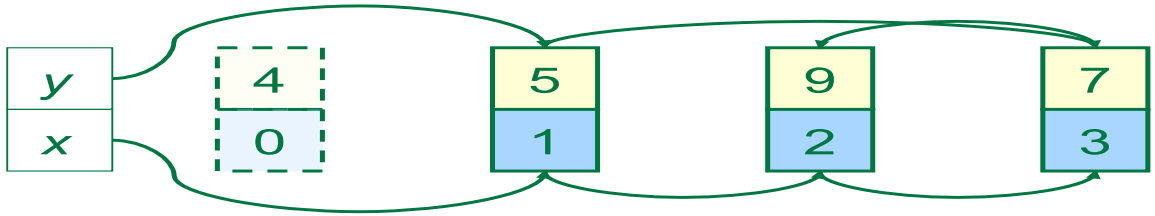
- La estructura permite iteración
  - sobre valores i, en orden seguir x-links
  - sobre valores $a[i]$ en orden seguir y-links

Propagación

- Seguir x-links e iterar valores en $s(x)$
  - si el valor no está en $s(x)$, eliminar el nodo
- Seguir y-links e iterar valores en $s(y)$
  - si el valor no está en $s(y)$, eliminar nodo
- Resultado: nodos con valores correctos permanecen para ambos, x, y
  - En orden creciente!

Seguir x-links, y-links

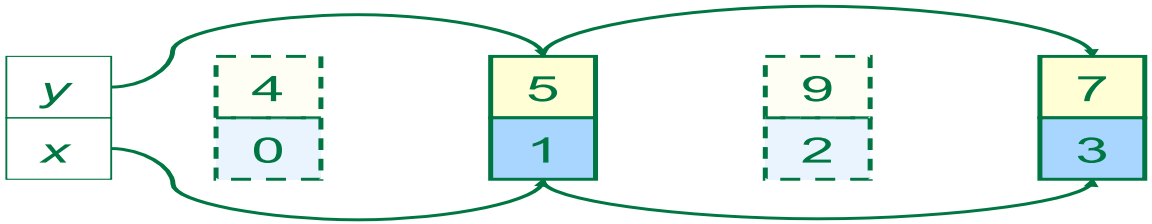
seguir x-links



- store $s(x) = \{1, 2, 3\}$
- eliminar nodo para 0
  - re-encadenando
  - tiempo constante: listas doblemente encadenadas

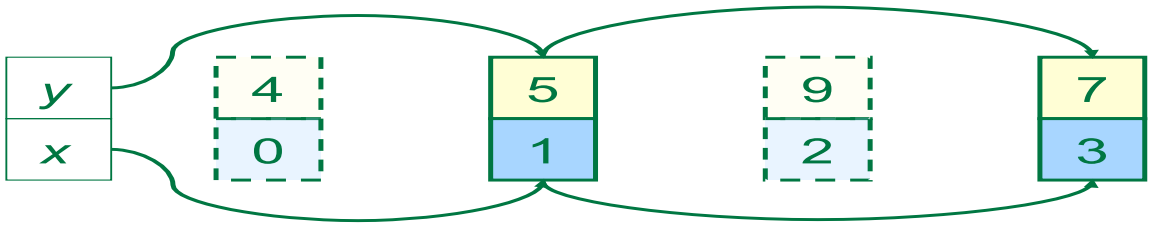
Seguir x-links, y-links

seguir y-links



- store $s(y) = \{2, ..., 8\}$
- eliminar nodo para 9
  - re-encadenando
  - tiempo constante: listas doblemente encadenadas

Leer valores de los dominios



- nuevo store: $s(x) = \{1, 3\}, s(y) = \{5, 7\}$
- siguiendo simplemente los links respectivos
  - están ordenados
  - son más pequeños que los originales

Propagación de restricciones con dominio-consistencia

- necesita considerar todas las soluciones
- ingenua: demasiada memoria y tiempo

Hay otra solución?

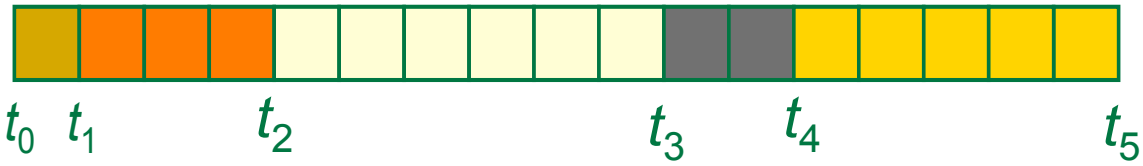
- en general no
- pero para restricciones particulares: si!
- Ya vimos para “element”. Y para otras más complicadas?

Propagación con dominio-consistencia: es necesaria?

se justifica dominio-consistencia,
comparado con propagación ingenua?

Un ejemplo: la regla de Golomb

Regla de Golomb



- encontrar n posiciones t_i tales que
  - las distancias entre posiciones son diferentes
  - la longitud de la regla es mínima
- es un problema muy difícil para casos grandes

Regla de Golomb

n	ingenua	dominio
7	950	474
8	7 622	3 076
9	55 930	16 608
10	413 922	97 782
11	6 330 568	1 448 666

- Impacto del algoritmo de propagación sobre el número de nodos en el espacio de búsqueda
  - OK, parece funcionar

Propagador dominio-consistente para “distinct”

- puede calcularse eficientemente
  - complejidad $O(n^{2,5})$
  - Referencia: Régin, “A fitering algorithm for constraints of difference in CSPs”, AAAI 1994
- Usa algoritmos sobre grafos
  - usa ideas sobre la estructura del problema
  - relación entre soluciones a la restricción y propiedades del grafo

Propagador para “distinct”: ejemplo

- antes de propagación:
  $s: \quad x_0 \to \{0,1\} \quad x_1 \to \{1,2\} \quad x_2 \to \{0,2\}$
  $x_3 \to \{1,3\} \quad x_4 \to \{2,3,4,5\} \quad x_5 \to \{5,6\}$
- después de propagación:
  $s: \quad x_0 \to \{0,1\} \quad x_1 \to \{1,2\} \quad x_2 \to \{0,2\}$
  $x_3 \to \{3\} \quad x_4 \to \{4,5\} \quad x_5 \to \{5,6\}$

Propagador para “distinct”: método

- construir un grafo de variable-valor
  - grafo bipartido
    nodos de variable $\longrightarrow$ nodos de valor
- caracterizar soluciones en el grafo
  - máxima correspondencia (“matching”)
- eliminar arcos que no representen soluciones

Grafo Variable-Valor

$s(x_0)=\{0,1\}$

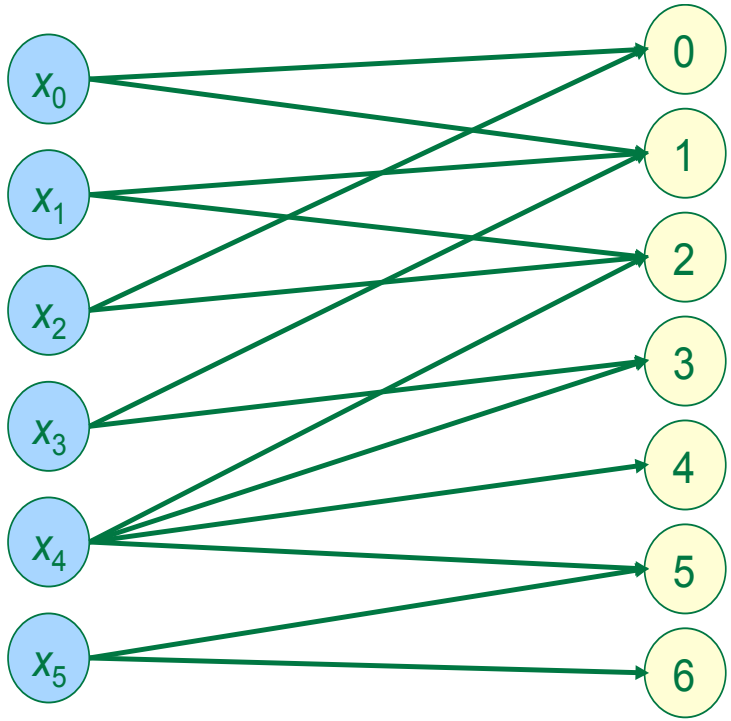
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



Correspondencia o “matching”

- una correspondencia es un subconjunto de arcos del grafo tales que

  ningún nodo tiene dos arcos en común
- la correspondencia es máxima si la cardinalidad es máxima
- en un grafo bipartido, la correspondencia es máxima si cubre un conjunto de nodos
  - en nuestro ejemplo: el conjunto de variables requiere cobertura

Las correspondencias son soluciones

- una asignación a es solución a la restricción “distinct”

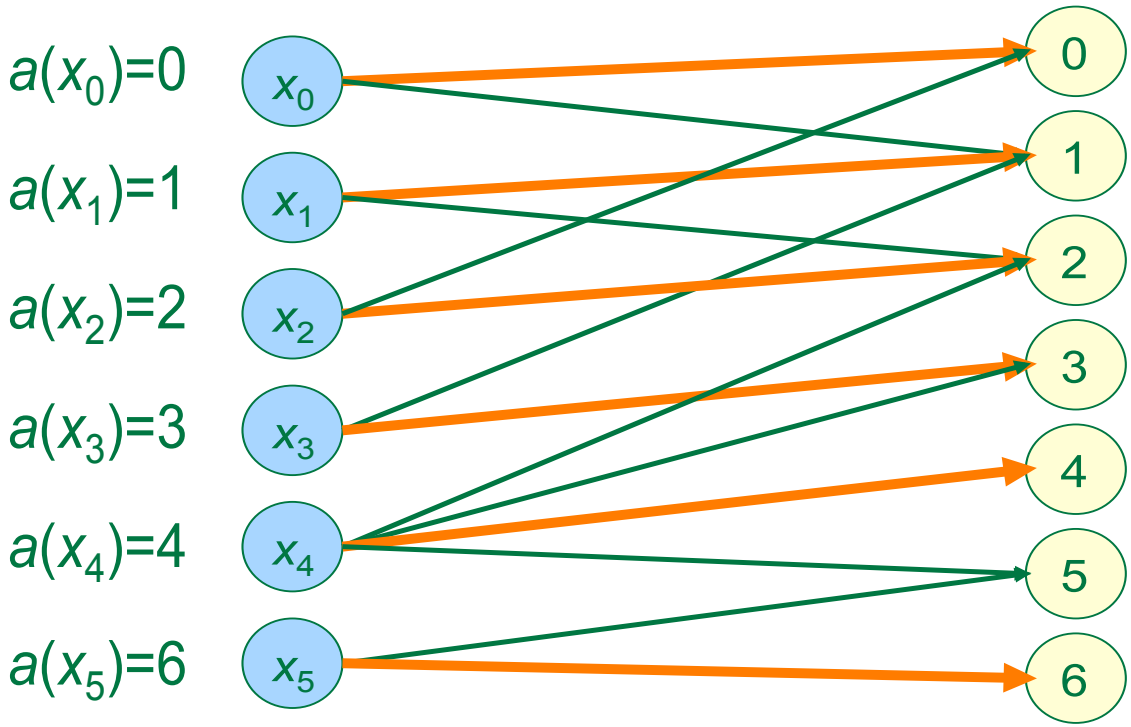
  $\Leftrightarrow$

  todas los nodos-variables con arcos

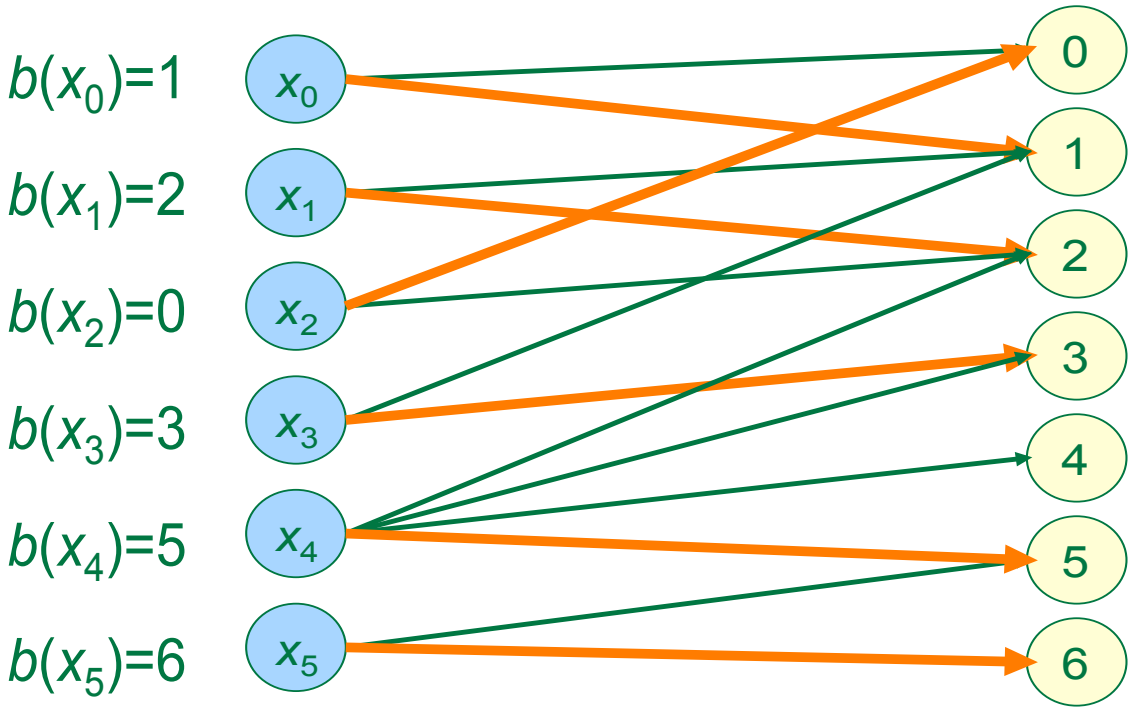
  $x_i \rightarrow a(x_i)$

  es una correspondencia máxima
  - ningún valor tiene dos arcos entrantes
  - todos los nodos-variables están cubiertos

Máxima correspondencia *a*



Mxima correspondencia *b*



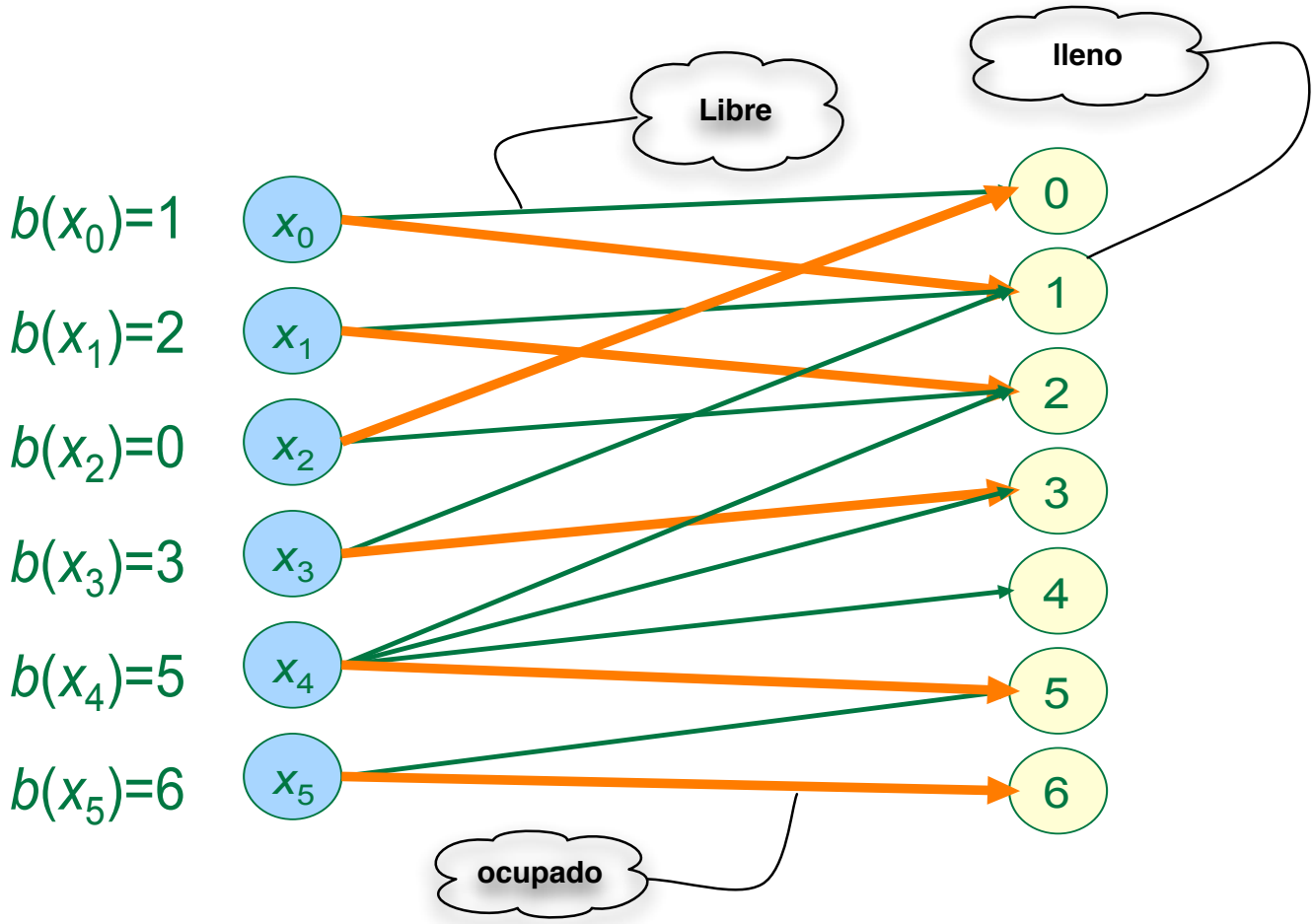
Primera idea

- Calcule todas las correspondencias máximas
  - marque los arcos en la correspondencia
- si no existe correspondencia máxima: falla
  - la correspondencia no cubre todos los nodos
- elimine arcos no marcados
  - el arco no ocurre en ninguna correspondencia máxima
- esto no resuelve el problema: tantas correspondencias máximas como soluciones!
  - pero: al menos se resuelve el problema de espacio

Correspondencia: nociones

- Arco
  - en correspondencia: ocupado
  - de lo contrario: libre
- Nodo
  - incidente a arco ocupado: lleno
  - de lo contrario: libre
- arco Vital: pertenece a cada correspondencia máxima

Libre, ocupado, lleno



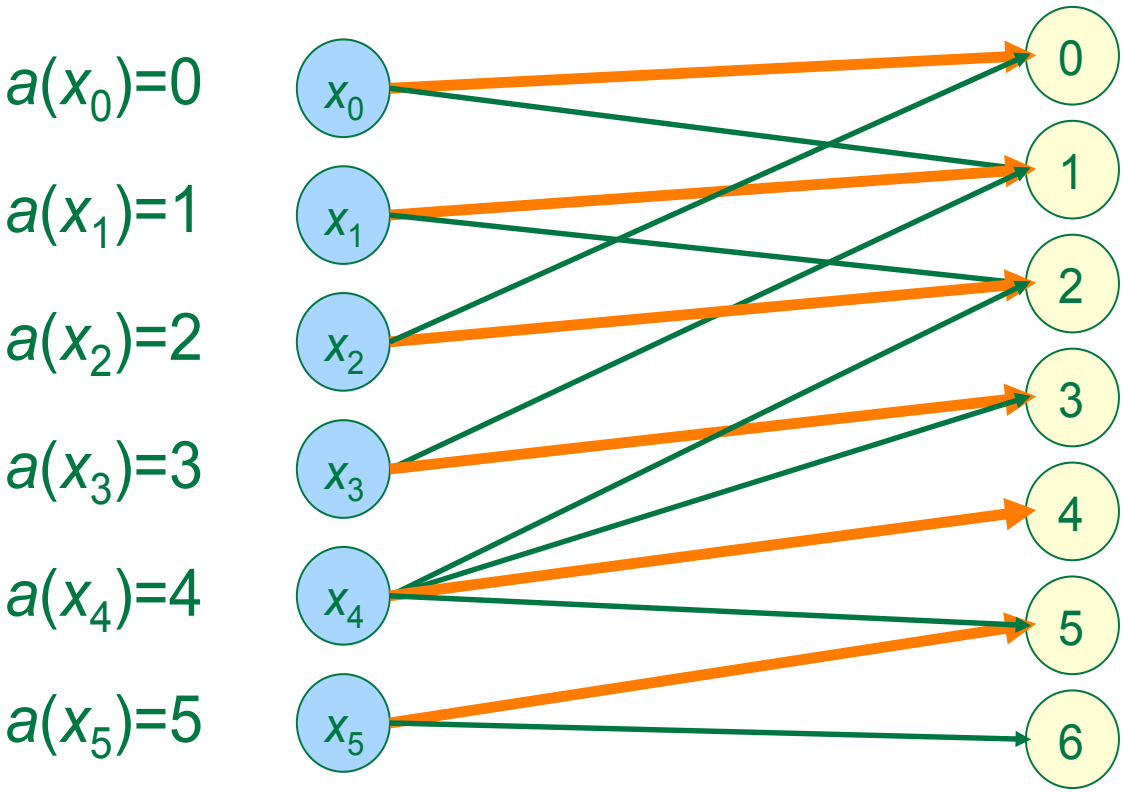
Reformulación mediante grafos

- Arco libre en toda máxima correspondencia:
  - borre el arco
  - es como borrar el valor de la variable
- Arco vital
  - guarde el arco
  - es como asignar el valor a la variable
- Arcos ocupados en algunas pero no todas las correspondencias máximas
  - guarde el arco

Método

- Teoría de correspondencias: conexión entre arcos en una correspondencia y arcos en algunas correspondencias
- Calcular una correspondencia maximal
  - calcular qué arcos ocurren en al menos una correspondencia
  - no hay necesidad de calcular más de una correspondencia

Correspondencia Máxima empleada



Camino y ciclo alternante

- Camino alternante
  - camino simple con arcos que alternan entre libre y ocupado
- Ciclo alternante
  - ciclo con arcos que alternan entre libre y ocupado
- Longitud de camino o ciclo
  - número de arcos
- Par: de longitud par

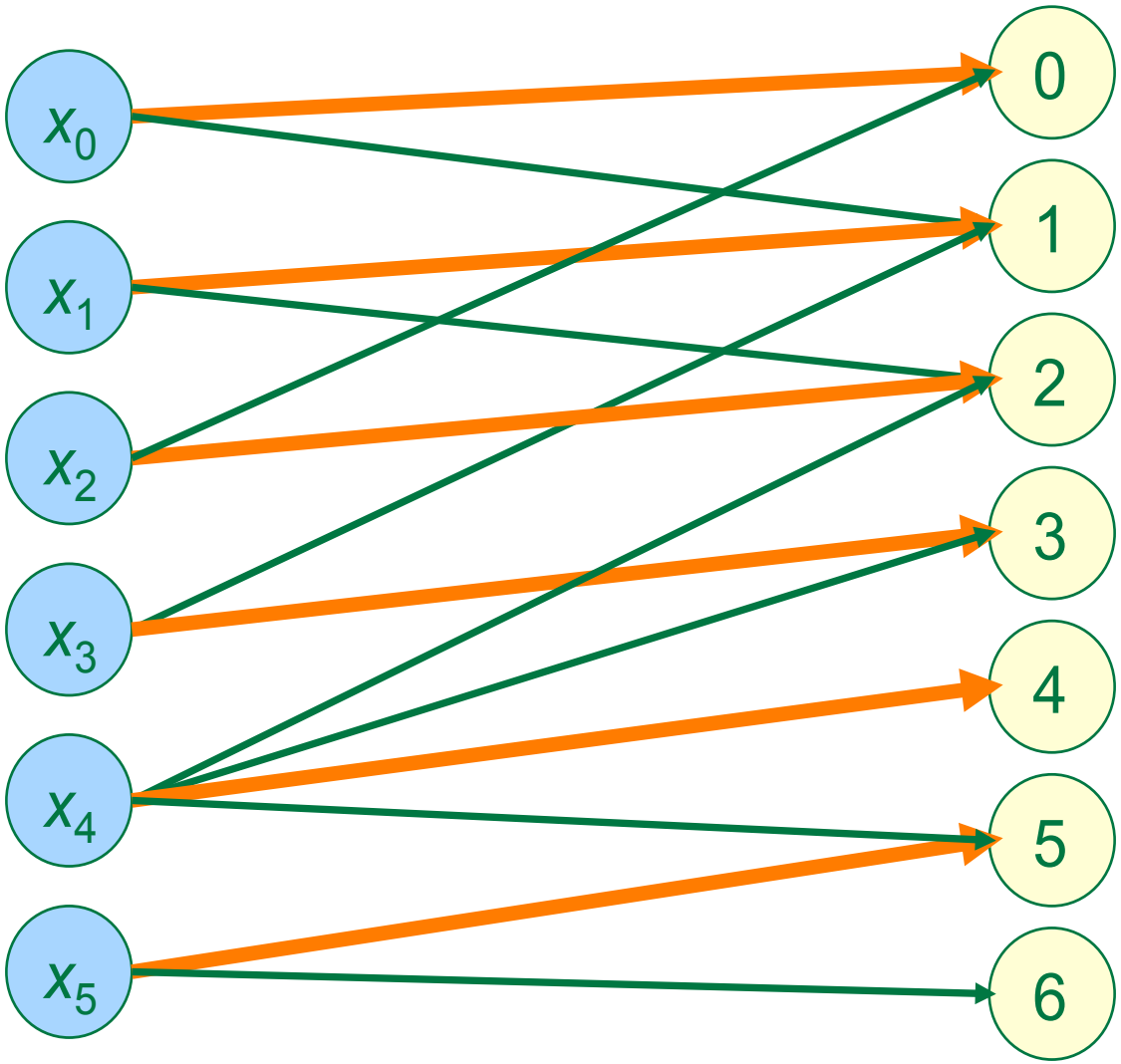
Teoría de correspondencia

- un arco e pertenece a alguna correspondencia $\Leftrightarrow$ para alguna correspondencia arbitraria M,
  - o el arco e pertenece a un camino alternante par que empieza en un nodo libre
  - o el arco e pertenece a un ciclo alternante par

Alternación: orientar arcos

- Como interesan los caminos alternantes pares, orientamos arcos, para simplificar
- para una correspondencia máxima dada
  - de variable a valor lleno
  - de valor a variable libre

Grafo orientado



Caminos alternantes

- Empezar en nodo libre, buscar todos los nodos en un camino simple directo
  - marcar todos los arcos en el camino
  - la alternación queda por construccion
- Empezar desde nodos-valor solamente
  - todos los nodos-variable están marcados!

Caminos alternantes

único nodo libre: 6

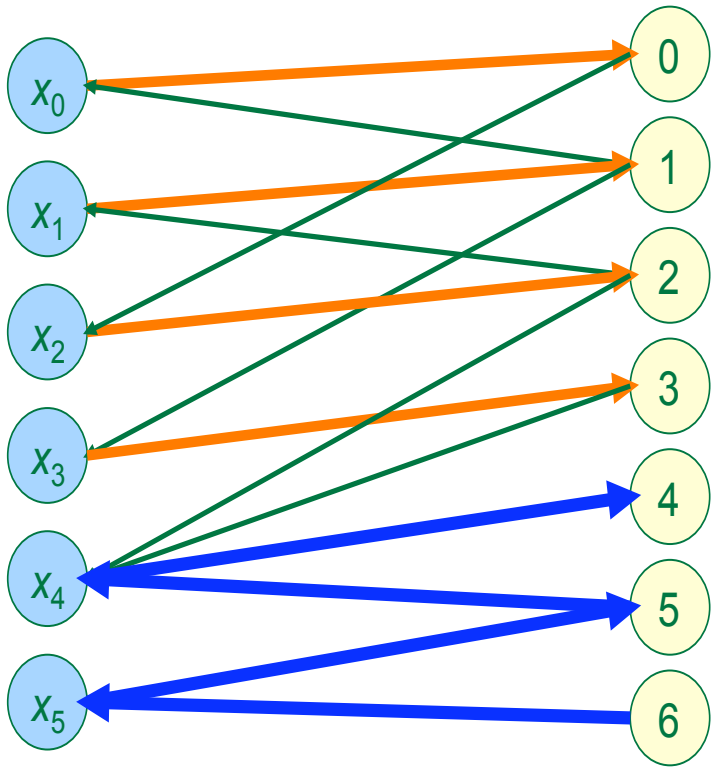
marque 6 --> X5

marque X5 --> 5

marque 5 --> X4

marque x4 --> 4

intuición: los nodos pueden permutarse



Ciclos alternantes

- Calcular componentes fuertemente conexos (CFC)
  - dos nodos *a* y *b* son fuertemente conexos si hay camino de *a* hasta *b* y de *b* hasta *a*
  - componente fuertemente conexo: cualquier par de nodos es fuertemente conexo
- Marcar todos los arcos que enlazan nodos en el mismo componente fuertemente conexo

Ciclos alternantes

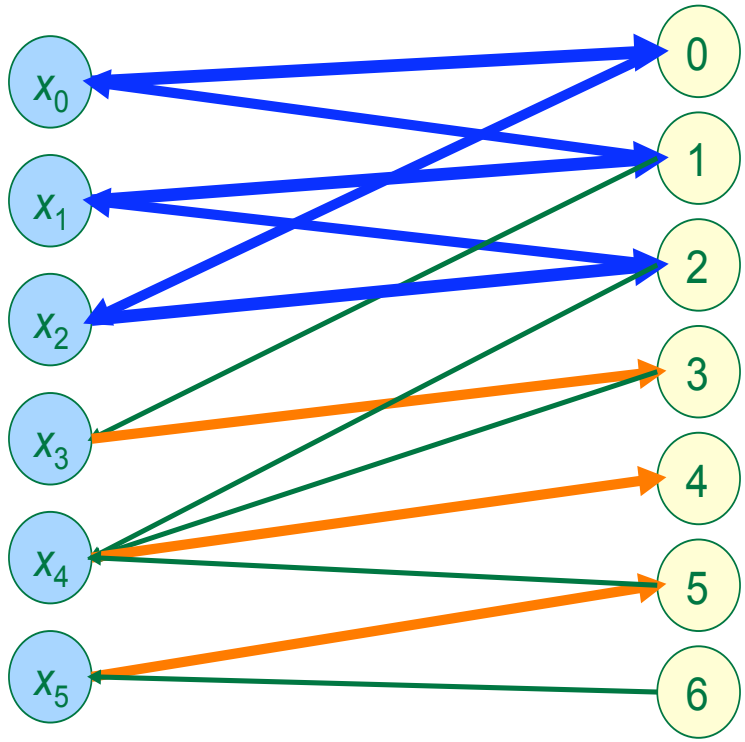
nodos en CFC:

X0,X1,X2,

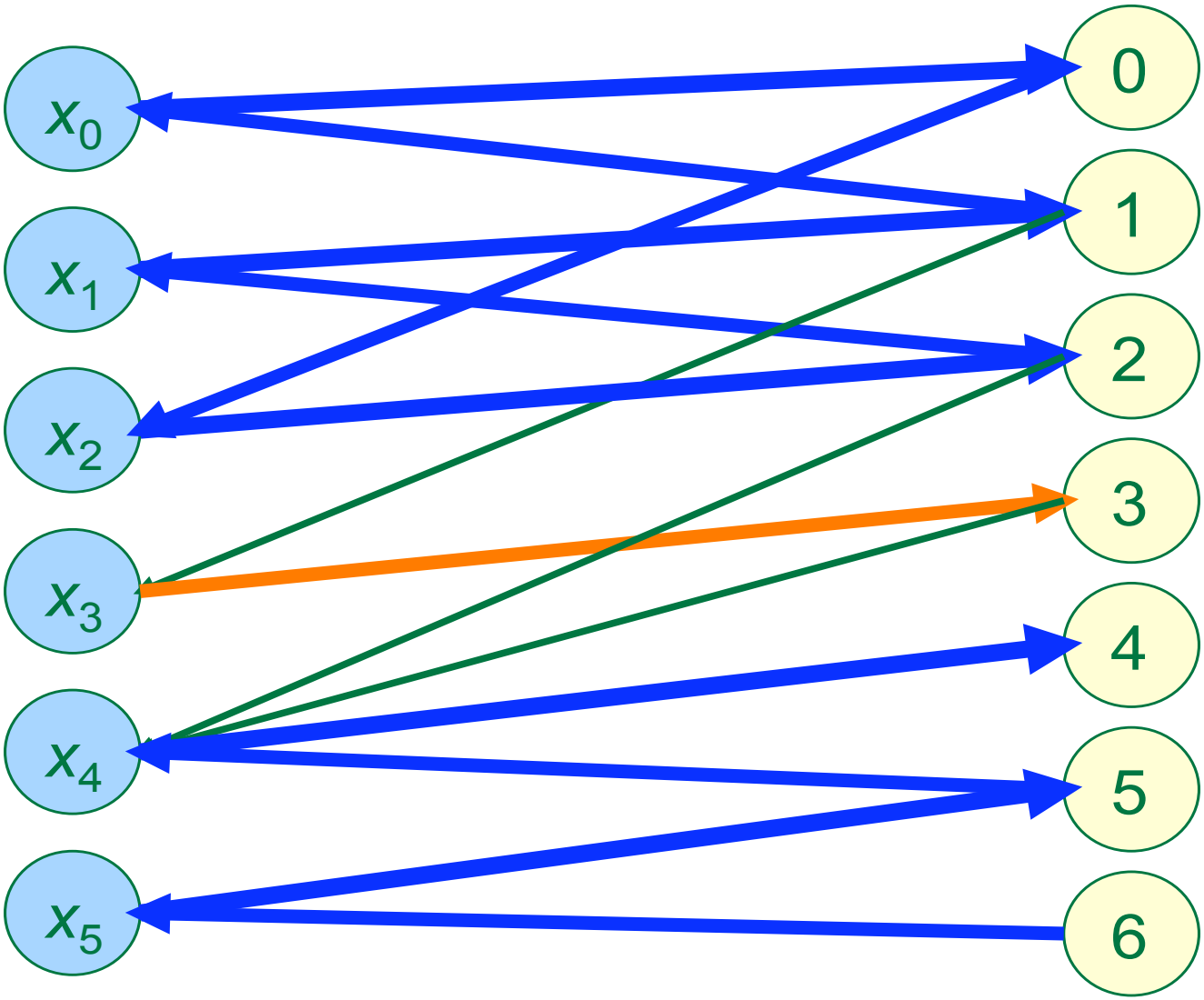
0,1,2

marcar arcos que se juntan

intuición: las variables toman todos los valores en el CFC



Todos los nodos marcados



Eliminar arcos

- Se eliminan aquellos que
  - están libres y no marcados
  - libre: no ocurre in muestra máxima correspondencia
  - no marcado: no ocurre en ninguna correspondencia
- Arcos vitales
  - no marcados y ocupados

Eliminar arcos

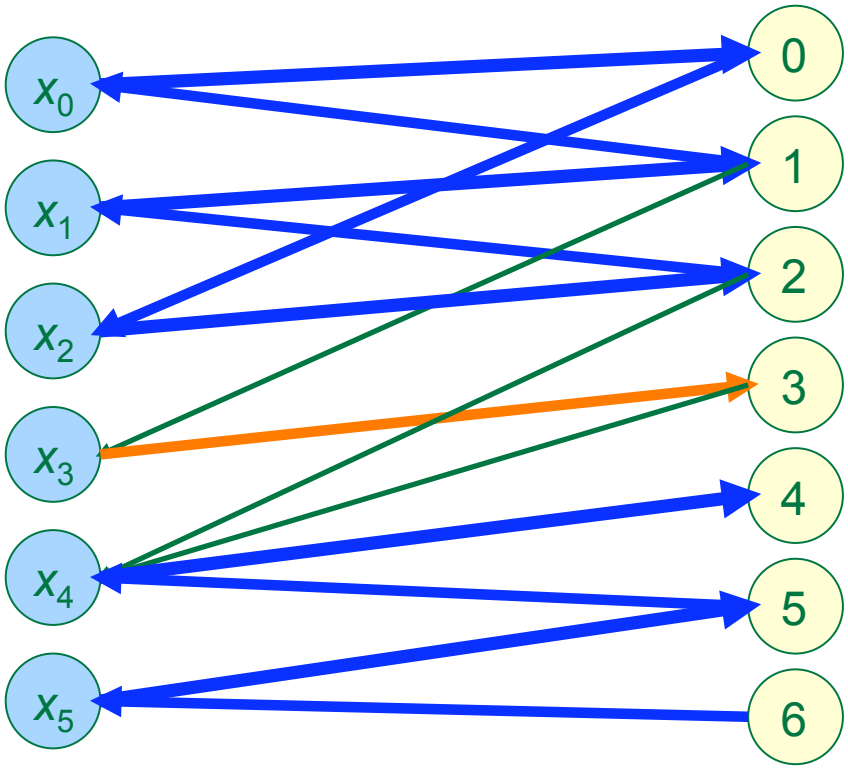
eliminar:

1-->X3
2-->X4
3-->X4

guardar:

X3-->3

ocupado!



Arcos eliminados

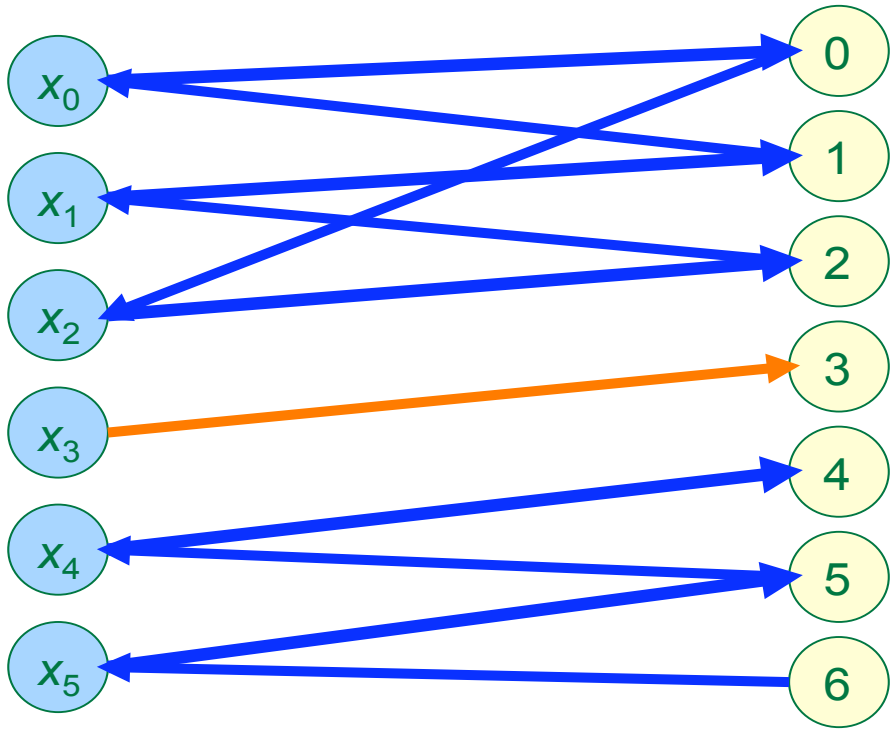
eliminar:

1-->X3
2-->X4
3-->X4

guardar:

X3-->3

ocupado!



Resumen del algoritmo

- Construir grafo variable-valor
- Encontrar correspondencia maximal
- Orientar el grafo (se hace al calcular correspondencia maximal)
- Marcar arcos que salen de nodos libres mediante búsqueda en el grafo
- Calcular CFC: marcar arcos de unión
- Eliminar arcos

Complejidad

- Lineal en el número de arcos
  - construcción, marcado, CFC
- Correspondencia maximal: muchos algoritmos conocidos
  - $O(n^{2,5})$ donde n es el máximo entre el número de variables y de valores