Restricciones Blandas: Modelos y Algoritmos

Javier Larrosa
Dep. LSI, UPC
Jordi Girona Salgado 1-3
08034 Barcelona
larrosa@lsi.upc.es

Pedro Meseguer
IIIA-CSIC
Campus UAB
08193 Bellaterra
pedro@iiia.csic.es

Resumen

El uso de *restricciones blandas* generaliza el modelo CSP clásico permitiendo que el usuario especifique sus preferencias respecto al conjunto de soluciones. En este modelo el problema consiste en encontrar la solución que mejor satisfaga las restricciones según el criterio especificado por el usuario. Este artículo presenta una visión de los trabajos más significativos sobre el procesamiento de restricciones blandas. En primer lugar presentamos los diferentes modelos de restricciones blandas usando los CSPs valuados (VCSPs) como marco unificador. Seguidamente describimos las dos aproximaciones para resolver este tipo de problemas: *búsqueda* e *inferencia*. También presentamos algunas estrategias híbridas.

Palabras clave: Restricciones blandas, optimización combinatoria, búsqueda heurística.

1 Introducción

El modelo CSP permite representar de manera natural, y resolver de manera eficiente, numerosos problemas de Informática en general, y de Inteligencia Artificial en particular [2].

Idealmente, el proceso de resolución de consta de dos pasos: primero se modeliza el problema como un CSP (es decir, se expresa en base a *variables, dominios* y *restricciones*) y a continuación se aplica un algoritmo de resolución, como los que se describen en [18]. Al algoritmo de resolución se le puede pedir que devuelva una solución arbitraria (si es que existe alguna) o bien que devuelva todas las soluciones. A menudo ocurre que este método no satisface al usuario. Una solución arbitraria no suele satisfacerle porque, de tener más alternativas, enseguida encontraría criterios para preferir unas sobre otras. El conjunto de todas las soluciones suele ser muy grande, lo que le hace difícil evaluar todas las alternativas, compararlas, y escoger la mejor. La respuesta de que el problema no tiene solución tampoco suele satisfacer al usuario que, de haberlo sabido, hubiese relajado el problema eliminando una o varias restricciones no críticas para facilitar la existencia de soluciones.

Como consecuencia de éstos inconvenientes, la resolución del problema se convierte en un proceso iterativo en el que el usuario va añadiendo nuevas restricciones (para eliminar soluciones de poca calidad, cuando hay muchas) o eliminando restricciones (para que aparezcan soluciones, cuando no hay ninguna). Éste proceso continua hasta que el usuario se da por satisfecho, sin que exista un criterio preciso de parada.

La problemática anterior pone de manifiesto la existencia de dos tipos esencialmente diferentes de restricciones:

- **Restricciones Duras.** Son condiciones de obligatorio cumplimiento tales como propiedades espaciales (p.ej. propiedades geométricas) o temporales (p.ej. un objeto no puede estar en dos lugares a la vez).
- **Restricciones Blandas**. Son restricciones que en realidad denotan preferencias del usuario y se desea que se cumplan en la me-

dida de lo posible (p.ej. en un problema de generación de horarios no se desea que un profesor imparta dos clases consecutivas en aulas distantes).

El modelo CSP clásico no maneja correctamente las restricciones blandas porque es incapaz de distinguir su importancia relativa. Para resolver esta limitación se han propuesto extensiones del modelo que permiten expresar restricciones blandas con diferentes semánticas, tales como *prioridades* [24], grados de *preferencia* [12], *costes* [14] o *probabilidades* [13]. En los modelos de restricciones blandas el objetivo consiste en encontrar la *mejor solución*, donde el criterio de preferencia entre unas soluciones y otras viene especificado por las restricciones blandas. Por lo tanto, la introducción de restricciones blandas transforma el problema de decisión (CSPs clásicos) en un problema de *optimización*. La resolución de CSPs con restricciones blandas es un problema NP-duro (*NP-hard*).

Recientemente, se han propuesto dos modelos que permiten razonar sobre restricciones blandas de manera abstracta, sin necesidad de conocer su semántica. La ventaja principal de estos modelos es que permiten el desarrollo de propiedades y algoritmos genéricos que luego se pueden especializar para cada caso. Estos dos modelos son los CSPs *valuados* (VCSPs) [23] y los CSP basados en *semi-anillos* (SCSPs) [5]. El modelo SCSP es algo más general porque permite definir problemas en los que el conjunto de soluciones tan solo esté parcialmente ordenado[4]. En este artículo tomaremos el modelo VCSP, conceptualmente más simple, como referencia.

En paralelo a los modelos de restricciones blandas se han desarrollado técnicas algorítmicas que permiten la resolución eficiente de este tipo de problemas. En muchos casos estas técnicas se pueden ver como la generalización de las desarrolladas para CSPs clásicos.

En este artículo hacemos un resumen de los resultados más significativos en el campo de las restricciones blandas incluyendo tanto aspectos de modelización como algorítmicos. El artículo se puede ver como una extensión de [18], que trata el modelo CSP clásico. Se ha procurado mantener la misma notación y estructura con el fin de resaltar las similitudes en ambos casos. Si bien este artículo es autocontenido, se recomienda la lectura previa de [18].

La estructura del artículo es la siguiente: A continuación se dan algunas definiciones básicas. En la Sección 2 se presentan los aspectos esenciales del modelo VCSP y cómo éste se puede instanciar para obtener los modelos de restricciones blandas más conocidos. En la Sección 3 se presentan el esquema de búsqueda como método de resolución. En la Sección 4 se describen los métodos de inferencia tanto completa como incompleta. En la Sección 5 se presentan algunos algoritmos híbridos. Finalmente, en la Sección 6 hacemos un resumen del artículo en el que destacamos los aspectos más relevantes.

1.1 Definiciones básicas

Sea $X = (x_1, x_2, \dots, x_n)$ un conjunto de variables con dominios $D = (D_1, D_2, \dots, D_n)$. La asignación del valor $a \in D_i$ se denota $x_i \leftarrow a$. Sea $Y = (x_{i_1}, x_{i_2}, \dots, x_{i_n})$ un subconjunto de X. Una asignación de las variables de Y se puede escribir como un conjunto ordenado de asignaciones individuales $(x_{i_1} \leftarrow v_{i_1}, x_{i_2} \leftarrow v_{i_2}, \dots, x_{i_n} \leftarrow v_{i_n})$ o, si queda suficientemente claro por el contexto, como una tupla genérica t. Sea t una tupla sobre un conjunto de variables Y y Z un subconjunto de Y. La *proyección* de t sobre Z, que se escribe $t[Z]$, es la subtupla de t con las asignaciones a las variables de Z. La *concatenación* de dos tuplas t y t' se escribe $t \cdot t'$. Para que dos tuplas se puedan concatenar tienen que asignar los mismos valores a las variables comunes.

2 Problemas de satisfacción de restricciones valuados

El modelo de los CSPs *valuados* (VCSP) [23] extiende el modelo CSP. La principal diferencia es que las restricciones no son relaciones sino *funciones de coste* que expresan el grado de satisfacción de las posibles asignaciones parciales. El *conjunto de costes* se denomina E y debe estar totalmente ordenado. También se tiene una operación $\oplus$ sobre E, que debe ser *commutativa* y *asociativa*. La operación $\oplus$, a la que llamaremos *suma* o *agregación*, permite combinar costes locales provenientes de diferentes restricciones para obtener costes globales con los que poder ordenar el conjunto de soluciones. Los elementos máximo y mínimo de E se denominan $\top$ y $\bot$, respectivamente. El modelo VCSP asume problemas de minimización, por lo que se entiende que valores bajos de E son preferibles a valores altos. El valor $\perp$ se usa para representar máxima satisfacción, mientras que $\top$ representa mínima satisfacción.

Una *red de restricciones valuadas* es una terna (X, D, C). Los conjuntos $X = (x_1, x_2, \ldots, x_n)$ y $D = (D_1, D_2, \ldots, D_n)$ contienen las *variables* y sus *dominios*, de manera similar a las redes de restricciones clásicas. La diferencia radica en C, que ahora es un conjunto de *restricciones posiblemente blandas*. Una restricción c es una función sobre un subconjunto de las variables $var(c) \subseteq X$ (llamado *ámbito* de c). Para cada tupla t que asigne valores a las variables de $var(c)$, $c(t)$ devuelve un elemento de E que indica el grado de satisfacción de la tupla t. Decimos que la restriccion c es *dura* si para toda tupla t, $c(t) \in \{\perp, \top\}$, es decir, todas las tuplas son totalmente satisfactorias o totalmente insatisfactorias. Cuando una restricción no es dura, decimos que es *blanda*.

La *valuación de una tupla* t sobre el conjunto de variables $Y \subset X$ se obtiene agregando los costes de todas las restricciones cuyo ámbito está incluido en Y. Formalmente,

$$\mathcal{V}(t) = \bigoplus_{c \in C, var(c) \subset Y} c(t[var(c)])$$

Una tupla t es *consistente* si no tiene la máxima valuación (es decir, si $\mathcal{V}(t) < \top$). Una tupla t es *solución* del problema si es una asignación total (asigna todas las variables) y es consistente. Resolver una red de restricciones valuadas consiste en *encontrar la mejor solución*, es decir, una solución con valuación mínima. Esto se conoce como problema de satisfacción de restricciones valuadas o VCSP (acrónimo del inglés *valued constraint satisfaction problem*).

A una red de restricciones blandas se le asocia el llamado *grafo de restricciones* de la misma manera que se hace en el modelo CSP clásico. Los nodos del grafo son las variables de la red y se pone una arista entre cada par de variables tal que exista alguna restricción que incluya a ambas variables en su ámbito.

2.1 Justificación y propiedades

Tras las definiciones previas, podemos justificar informalmente las propiedades que se suelen pedro a los modelo CSP con restricciones blandas:

- El conjunto E debe estar totalmente ordenado para que los diferentes niveles de preferencia puedan ser comparados. En la práctica, esta propiedad es muy útil para el desarrollo de algoritmos eficientes.

- La operación $\oplus$ es commutativa y asociativa para que la valuación de las asignaciones totales sólo dependa de las valuaciones locales (dadas por las restricciones) y no del orden en que se han agregado.

- La operación $\oplus$ crece de manera monótona para que una asignación A nunca sea mejor que una asignación contenida en A. Esta propiedad también es muy útil en el desarrollo de algoritmos.

2.2 Instanciaciones habituales

Existen infinitas instanciaciones posibles del conjunto E y la operación $\oplus$. Cada una de ellas da lugar a un modelo diferente de CSPs con restricciones blandas. A continuación enumeramos los casos más conocidos.

- **CSP clásico.** Si tomamos $E = \{cierto, falso\}$ (con $cierto < falso$) y $u \oplus v = u \wedge v$, tenemos el modelo CSP clásico. Las restricciones son funciones booleanas que tan sólo distinguen entre tuplas (totalmente) permitidas y tuplas (totalmente) prohibidas, sin posibilidades intermedias.

- **CSP posibilístico.** Si tomamos $E = [0, 1]$ y $u \oplus v = \max\{u, v\}$ obtenemos el modelo de CSPs *posibilísticos* [24]. Las restricciones devuelven valores en el intervalo $[0, 1]$ y la valuación global de una asignación es la peor (es decir, el coste maximo) de las valuaciones locales. Este modelo está directamente relacionado con los *CSPs difusos* (*fuzzy CSPs* en inglés) [12]. De hecho, ambos modelos tienen la misma capacidad expresiva y una instancia se puede transformar de un modelo al otro.

- **WCSP**. Cuando E es el conjunto de los naturales más el ∞, y $\oplus$ es la suma, tenemos los CSPs *con peso* (también llamados WCSPs, acrónimo del inglés *weighted* CSPs) [23]. Un caso especial de WCSP es en el que todas las restricciones devuelven costes en $\{0, 1\}$. Este es el conocido modelo MAX-CSP.

- **CSP probabilístico**. El caso $E = [0, 1]$ con $u \oplus v = 1 - (1-u)(1-v)$ se corresponde con el modelo de CSPs *probabilísticos* [13] que permite expresar algunos tipos de inferencia probabilística.

Ejemplo 1 *Consideremos VCSPs con tres variables $X = \{x_1, x_2, x_3\}$ todas ellas con el mismo dominio $D_i = \{0, 1\}$.*

- *Supongamos un CSP con pesos que tiene tres restricciones $f_1(x_1, x_2) = 2 - x_1 - x_2$, $f_2(x_1, x_3) = x_1 x_3$ y $f_3(x_2, x_3) = x_2 + x_3$. Una solución óptima es $(x_1 \leftarrow 1, x_2 \leftarrow 0, x_3 \leftarrow 0)$, cuya valuación es 1.*

- *Supongamos un CSP posibilístico también con tres restricciones $f_1(x_1, x_2) = (x_1 + x_2)/2$, $f_2(x_1, x_3) = \min\{x_1, x_3\}$ y $f_3(x_2, x_3) = \max\{x_2, x_3\}$. Una solución óptima es $(x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 0)$, cuya valuación es 0.*

Decimos que un modelo CSP con restricciones blandas es *idempotente*, si el operador $\oplus$ lo es (es decir, $\forall a \in E, (a \oplus a) = a$). Por ejemplo los CSPs clásicos y posibilísticos son modelos idempotentes. Sin embargo, los WCSPs no lo son. Está diferenciación es importante para el desarrollo de algoritmos, pues en general la idempotencia es una propiedad que se puede explotar, produciendo algoritmos conceptualmente más simples y más eficientes para este tipo de problemas.

3 Búsqueda

La forma más natural de resolver un VCSP es mediante un proceso de búsqueda en el espacio de estados del problema definido por el conjunto de todas las asignaciones posibles de las variables. Igual que para la resolución de CSPs clásicos, hay dos esquemas básicos de búsqueda, la *búsqueda sistemática* y la *búsqueda local*. Los algoritmos de búsqueda sistemática visitan todos los estados significativos y devuelven la solución óptima. Los algoritmos de búsqueda local no garantizan que se visiten todos los estados y devuelven la mejor solución que han podido encontrar, sin garantía de que sea óptima. Si bien los algoritmos de búsqueda local son la mejor opción en muchas aplicaciones prácticas, en este artículo nos limitaremos a tratar los algoritmos de búsqueda sistemática.

El algoritmo sistemático más habitual para resolver VCSPs consiste en aplicar una estrategia de *backtracking*, tal como se hace con los CSPs clásicos. Sin embargo, para resolver VCSPs hay que adaptarla a problemas de optimización. Esto se consigue usando la técnica de *ramificación y poda* (del inglés *branch and bound*) [14]. El algoritmo resultante realiza una búsqueda primero en profundidad en un árbol tal que cada nodo representa una asignación. Los nodos internos representan asignaciones parciales y las hojas representan asignaciones totales. La raiz del árbol representa la asignación vacía. Los hijos de un nodo se obtienen seleccionando una de las variables que todavía no están asignadas y considerando cada uno de los valores de su dominio.

Durante la búsqueda, el algoritmo mantiene el coste de la mejor solución encontrada hasta el momento. Este coste es una *cota superior* CS de la mejor solución del problema. En cada nodo, el algoritmo computa una *cota inferior* CI del mejor coste que se puede obtener en el subárbol correspondiente. Si CI es mayor o igual que CS, el algoritmo hace backtracking (poda el subárbol que pende del nodo actual) porque la mejor solución encontrada hasta el momento no se puede mejorar en ese subárbol. Una cota inferior obvia es la valuación de la asignación del nodo en curso. Desafortunadamente esta cota es de poca calidad (toma valores bajos que pocas veces permiten que se de la condición de poda).

La Figura 1 muestra BB (acrónimo de *branch and bound*), una implementación recursiva del algoritmo básico. Por simplicidad, asumimos que las variables se van asignando en orden lexicográfico. $mejorSol$ y CS son variables globales que guardan la mejor solución encontrada hasta el momento y su coste, respectivamente. Al final, estas variables contienen la mejor solución del problema y su coste. CS se debe inicializar con una cota superior de la solución óptima (si no se conoce ninguna, se puede inicializar con $\top$). Los parámetros de una llamada arbitraria son el índice de la siguiente variable a asignar i y la asignación actual t que da valor a todas las variables anteriores $\{x_1, x_2, \ldots, x_{i-1}\}$. Si $i = n+1$, la asignación actual es total y ésta mejora a la mejor encontrada hasta el momento. Por lo tanto, $mejorSol$ y CS se actualizan (líneas 2,3). Si $i < n+1$, la asignación actual es parcial. En ese caso la nueva variable a asignar es x_i. Se va iterando sobre los valores de su dominio. Para cada valor se calcula la cota inferior del nodo actual (línea 6). Si la cota inferior es menor que la cota superior, se hace la llamada recursiva (línea 8). Sino se poda el subárbol actual y se pasa a considerar el siguiente valor de x_i.

procedure BB(i, t)
1. **if** $(i = n + 1)$**then**
2. $\quad CS := CI$;
3. $\quad mejorSol := t$
4. **else**
5. $\quad$ **for each** $a \in D_i$ **do**
6. $\quad\quad CI := \mathcal{V}(t \cdot (x_i \leftarrow a))$
7. $\quad\quad$ **if** $CI < CS$ **then**
8. $\quad\quad\quad$ BB$(i + 1, t \cdot (x_i \leftarrow a))$
9. $\quad\quad$ **endif**
10. $\quad$ **endfor**
11. **endif**
endprocedure

Figura 1. Branch and Bound genérico para VCSPs.

El coste temporal del algoritmo en el caso peor es proporcional al tamaño del árbol de búsqueda, es decir $O(d^n)$, siendo d la cardinalidad máxima de entre los dominios y n el número de variables. Sin embargo, en la práctica se observa que esta cota es demasiado pesimista y el coste real para instancias concretas suele ser mucho menor. El coste espacial es polinómico.

Existen varias formas de mejorar el rendimiento de los algoritmos de búsqueda. En [19, 1] se proponen cotas inferiores más sofisticadas que, si bien son más costosas de calcular, permiten podar en niveles más altos del árbol con lo que se mejora el rendimiento global del algoritmo. Una práctica común consiste en ejecutar un algoritmo de búsqueda local antes de la búsqueda sistemática. El coste de la mejor solución encontrada se utiliza para inicializar la cota superior CS. De esta manera, CS toma desde el principio un valor bajo, con lo que se facilita la condición de poda y se mejora la eficiencia de la búsqueda. También puede ser muy útil establecer un buen orden en el que asignar las variables y, para cada variable, en el que asignar los valores. Esto se resuelve usando criterios heurísticos, dando lugar a las llamadas heurísticas de *selección de variable* y *selección de valor*. Igual que para los CSP clásicos, las heurísticas pueden ser estáticas o dinámicas, siendo las dinámicas las que suelen dar mejores resultados empíricos.

4 Inferencia

La inferencia en una red de restricciones duras consiste en deducir nuevas restricciones a partir de las restricciones existentes [18]. En las redes de restricciones blandas este concepto es menos preciso. En general podemos decir que un proceso de inferencia consiste en hacer explícitos costes que están implicitos en la red de restricciones. En otras palabras, la inferencia asocia a determinadas tuplas costes necesarios (un coste necesario asociado a una tupla es una cota inferior del coste que tendrá cualquier extensión de la tupla al resto de variables).

Los algoritmos de *inferencia completa* calculan costes necesarios y suficientes (es decir costes exactos) y los usan para sintetizar la solución del problema. Los algoritmos de *inferencia incompleta* computan costes necesarios y los usan para podar valores o aproximar la solución del problema. A continuación presentamos un algoritmos de inferencia completa, y dos tipos diferentes de inferencia incompleta.

4.1 Inferencia completa

Casi todos los algoritmos de inferencia completa para CSPs clásicos se pueden extender al modelo VCSP. En este artículo nos centraremos en un algoritmo denominado BE (acrónimo del inglés *Bucket Elimination*) [6, 3]. BE es la generalización del algoritmo de *Consistencia Adaptativa* [18, 9] al modelo VCSP. El algoritmo es muy utilizado para resolver problemas de razonamiento probabilístico sobre redes bayesianas. BE se basa en dos operaciones básicas sobre funciones: *combinación* (o *suma*) y *eliminación* (o *proyección*). Estas operaciones extiende el join y la proyección definidas sobre relaciones.

Definición 1 *La* combinación *(o suma) de dos funciones f y g, que denominaremos $(f \oplus g)$, es otra función cuyo ámbito es $var(f) \cup var(g)$ y que devuelve la agregación de los costes de f y g,*

$$(f \oplus g)(t) = f(t[var(f)]) \oplus g(t[var(g)])$$

Definición 2 *La* eliminación *de la variable x_i de la función f, que denominaremos $f \Downarrow x_i$, es otra función cuyo ámbito es $var(f) - \{x_i\}$ y que devuelve para cada tupla t la mejor extensión de t a*

x_i,

$$(f \Downarrow x_i)(t) = \min_{a \in D_i}\{f(t \cdot (x_i \leftarrow a))\}$$

Ejemplo 2 *Consideremos dos funciones $f_1(x_1, x_2) = 2 - x_1 - x_2$ $f_2(x_1, x_3) = x_1 x_3$ sobre variables cuyo dominio es $\{0, 1\}$ en el contexto del modelo WCSP. La suma de f_1 y f_2 es la función $(f_1 + f_2)(x_1, x_2, x_3) = 2 - x_1 - x_2 + x_1 x_3$. La eliminación de x_2 en f_1 da lugar a la función $(f_1 \Downarrow x_2)(x_1) = 1 - x_1$*

Notesé que cuando f es una función unaria (es decir, de aridad uno) eliminar su única variable produce una función constante. La operación básica de BE es la eliminación de variable, que definimos a continuación,

Definición 3 *Sea $P = (X, D, C)$ una instancia de VCSP. Sea $x_i \in X$ una variable arbitraria y sea B_i el conjunto de todas las restricciones que tienen x_i en su ámbito. Se define g_i como la combinación de todas las funciones de B_i y la posterior eliminación de x_i,*

$$g_i = (\bigoplus_{f \in B_i} f) \Downarrow x_i$$

La eliminación de la variable x_i transforma P en $P' = (X - \{x_i\}, D - \{D_i\}, (C - B_i) \cup g_i)$.

En palabras, P' se obtiene reemplazando x_i y todas las funciones de B_i por g_i. P y P' tienen el mismo coste óptimo porque, por construcción, g_i *compensa* la ausencia de x_i. Se puede demostrar que la eliminación de una variable x_i de un problema (X, D, C) tiene un coste temporal y espacial exponencial respecto al número de adyacentes que tiene x_i en el grafo de restricciones.

BE (detallado en la Figura 2) usa un orden o entre las variables que, sin pérdida de generalidad, asumiremos lexicográfico ($o = (x_1, x_2, \ldots, x_n)$). El algoritmo consta de dos fases. En la primera las variables se eliminan una a una, en orden decreciente. La eliminación de x_i consta de dos pasos: (1) Se computa el conjunto B_i que contiene todas las restricciones con x_i en su ámbito y que no tienen ninguna otra variable con índice mayor. A B_i se le llama el *cubo* de x_i (trad. del inglés, *bucket*). (2) Se computa la nueva función g_i que se incorpora al problema sustituyendo a x_i y su cubo B_i.

La eliminación de la última variable x_1 genera una constante que es el óptimo del problema. En la segunda fase BE calcula una asignación de las variables del problema original que produce el coste óptimo. Para ello utiliza los resultado parciales de la primera fase (los diferentes cubos se deben haber conservado). Las variables se van recuperando en orden creciente $x_1, x_2, \ldots, x_n$. El valor óptimo para la variable x_i es la mejor extensión de lo asignado a las variables anteriores respecto al cubo B_i.

Igual que el algoritmo de consistencia adaptativa [18], el coste espacial y temporal de BE es exponencial en la anchura inducida del grafo de restricciones w^*. El alto coste espacial es la principal limitación del algoritmo que en la práctica sólo se puede usar para resolver problemas cuyos grafos sean poco densos y con un nivel de ciclicidad relativamente bajo (en realidad el parámetro w^* mide la ciclicidad del problema [15]) La Figura 2 da una descripción algorítmica de BE.

procedure BE(X, D, C)
    for each $i = n..1$ **do**
        $B_i := \{c \in C | \ x_i \in var(c)\}$
        $g_i := (\sum_{f \in B_i} f) \Downarrow x_i$
        replace x_i and B_i by g_i in the problem
    endfor
    for each $i = 1..n$ **do**
        $x_i \leftarrow$ best extension w.r.t. B_i
    endfor
endprocedure

Figura 2. BE para VCSPs.

Ejemplo 3 *Considerese el CSP con pesos del Ejemplo 1. Sus restricciones son: $f_1(x_1, x_2) = 2 - x_1 - x_2$, $f_2(x_1, x_3) = x_1 x_3$, y $f_3(x_2, x_3) = x_2 + x_3$. BE elimina sus variables en orden decreciente*

Eliminación de x_3. *El cubo de la variable x_3 es $B_3 = \{f_2, f_3\}$. La función g_3 es $(f_2 + f_3) \Downarrow_{x_3}$. Aplicando la definición de suma y eliminación se deduce que $g_3 = x_2$. Por lo tanto, la eliminación de x_3 da lugar a un nuevo problema con dos variables $\{x_1, x_2\}$ y dos restricciones $\{f_1, g_3\}$.*

Eliminación de x_2. *El cubo de la variable x_2 es $B_2 = \{f_1, g_3\}$. La función g_2 es $(2 - x_1 - x_2 + x_2) \Downarrow_{x_2} = 2 - x_1$. Por lo tanto, la eliminación de x_2 da lugar a un nuevo problema con una variable $\{x_1\}$ y una restricción $\{g_2\}$.*

Eliminación de x_1. *El cubo de la variable x_1 es $B_1 = \{g_2\}$. La función g_1 es $(2 - x_1) \Downarrow_{x_1} = 1$.*

Por lo tanto, la eliminación de x_1 da lugar a una constante 1 que es el coste de la solución óptima.

La asignación que da lugar al coste óptimo se obtiene asignando las variables en orden creciente. Para cada variable se mira qué valor es la mejor extensión de acuerdo con su cubo.

Cabe mencionar que, en general, las funciones g_i no se pueden representar como expresiones algebraicas y hay que guardarlas como tablas. Este es el motivo del alto coste temporal y espacial de BE.

4.2 Inferencia incompleta

4.2.1 Consistencia local

Los algoritmos de consistencia local *concentran* en determinadas tuplas costes que están *repartidos* por toda la red de restricciones. Estos costes se utilizan para: detectar que algunos valores no son factibles (por lo que se pueden podar) o, en el mejor de los casos, para detectar que el problema es globalmente inconsistente, es decir, no tiene ninguna solución.

En esta sección, tal como se suele hacer cuando se definen consistencias locales en el modelo CSP clásico, asumiremos que tratamos con problemas binarios. A las restricciones binarias las denominaremos c_{ij} y a las unarias c_i. Los subíndices indican las variables de su ámbito. Sin perdida de generalidad, también asumiremos que: (1) Cada restricción tiene un ámbito diferente (si hay varias restricciones con el mismo ámbito se pueden sumar). (2) Para toda variable x_i existe una restricción unaria c_i (siempre se puede asumir la existencia de una restricción ficticia que devuelve coste $\perp$ a todas las tuplas). (3) Existe una restricción $c_\emptyset$ de aridad cero (es decir, una constante). Inicialmente $c_\emptyset = \perp$.

Los VCSPs binarios se pueden representar gráficamente de la siguiente manera: Cada variable del problema es un vértice de un grafo. Dentro del vertice se ponen los diferentes valores del dominio junto a los costes unarios. Las restricciones binarias se representan mediante aristas que conectan los pares de valores cuyo coste no sea $\perp$. Las aristas se etiquetan con el coste correspondiente.

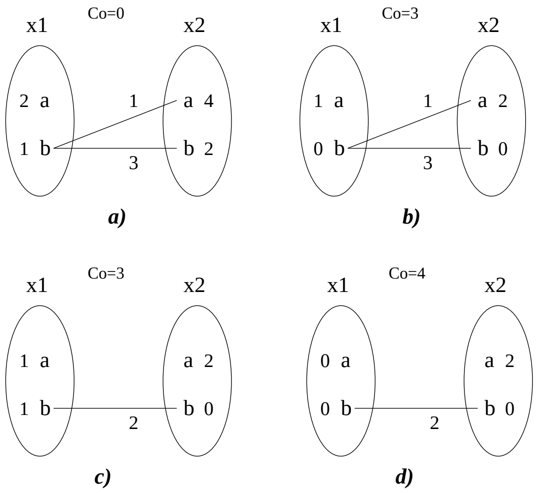

Figura 3. Cuatro WCSPs equivalentes.

Ejemplo 4 *La Figura 3.a representa un WCSP con dos variables y dos valores $\{a, b\}$ en cada dominio. La solución óptima es $t = (x_1 \leftarrow a, x_2 \leftarrow b)$, cuya valuación es $\mathcal{V}(t) = c_1(a) + c_2(b) + c_{12}(a, b) = 2 + 2 + 0 = 4$*

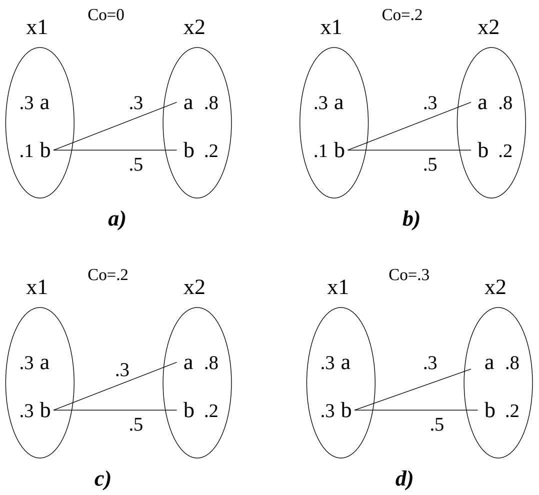

Figura 4. Cuatro CSPs posibilistas equivalentes.

Ejemplo 5 *La Figura 4.a representa un CSP posibilista con dos variables y dos valores $\{a, b\}$ en cada dominio. La solución óptima es $t = (x_1 \leftarrow a, x_2 \leftarrow b)$, cuya valuación es $\mathcal{V}(t) = \max\{c_1(a), c_2(b), c_{12}(a, b)\} = \max\{.3, 0, .2\} = .3$*

La generalización de la noción de consistencia local (k-consistencia) a las redes de restricciones blandas es un tema de investigación en el que todavía quedan muchos interrogantes abiertos. En general, se ha observado que la generalización no plantea demasiadas dificultades en el caso de los modelos idempotentes (p.ej CSPs posibilistas o difusos). Para los modelos no idempotentes (p.ej.

WCSPs) tan sólo se conocen los niveles de consistencia local más básicos: consistencia de nodos y arcos. Además se ha observado que las definiciones no son únicas. A continuación mencionamos dos de estas definiciones.

Para poder definir consistencia de nodos y arcos en el modelo VCSP hace falta que la operación $\oplus$ tenga una operación inversa $\ominus$ que se llama *diferencia* y que se define de la siguiente manera,

Definición 4 *Para todo par $u, v \in E$ tal que $v \leq u$ existe al menos un elemento $z \in E$ tal que $v \oplus z = u$. Sea w, el máximo de todos los elementos que cumplen tal propiedad. Decimos que w es la diferencia entre u y v, y lo escribimos $w = u \ominus v$.*

Propiedad 1 .

- *En el modelo CSP posibilístico ($u \oplus v = \max\{u, v\}$) la diferencia es $u \ominus v = u$*
- *En el modelo WCSP ($u \oplus v = u + v$) la diferencia es $u \ominus v = u - v$*
- *En el modelo CSP clásico ($u \oplus v = u \wedge v$), la diferencia es $u \ominus v = u$*

A continuación damos una definición de consistencia de nodos y arcos. Para ver más detalles, así como definiciones alternativas el lector puede consultar [20].

Consistencia de Nodos [21]. Dada una red de restricciones blandas, una variable x_i es nodo consistente ssi: i) existe un valor $a \in D_i$ al que llamamos *soporte* de x_i tal que $c_\emptyset \oplus c_i(a) = c_\emptyset$, ii) para todo valor $a \in D_i$ se cumple que $c_\emptyset \oplus c_i(a) < \top$. La red de restricciones es nodo consistente ssi todas las variables lo son.

Cuando un problema no es nodo consistente se puede transformar en un problema equivalente que sí lo sea aplicando operaciones locales, tal como se hace en el algoritmo NC de la Figura 5. El algoritmo consta de dos fases. En la primera itera sobre las variables calculando el mínimo coste unario (línea 2). Este coste se le suma a $c_\emptyset$ (línea 3) y, para garantizar que el problema resultante es equivalente, se le resta a todos los costes unarios (línea 4). Al proceso realizado en las líneas 2-6 se le llama *proyección de c_i sobre $c_\emptyset$*. En la segunda fase se vuelve a iterar sobre las variables podando aquellos valores cuyo coste unario es suficientemente alto como para garantizar su inconsistencia (línea 10). El coste del algoritmo es $O(nd)$, siendo n el número de variables y d la cardinalidad máxima de los dominios.

procedure $\texttt{NC}(X, D, C)$
1. **for each** $i = 1..n$ **do**
2. $\quad m := \min_{a \in D_i}\{c_i(a)\}$;
3. $\quad c_\emptyset := c_\emptyset \oplus m$;
4. $\quad$ **for each** $a \in D_i$ **do**
5. $\qquad c_i(a) := c_i(a) \ominus m$;
6. $\quad$ **endfor**
7. **endfor**
8. **for each** $i = 1..n$ **do**
9. $\quad$ **for each** $a \in D_i$ **do**
10. $\qquad$ **if** $c_\emptyset \oplus c_i(a) = \top$ **then** $D_i := D_i - \{a\}$
11. $\quad$ **endfor**
12. **endfor**

endprocedure

Figura 5. Consistencia de Nodos para VCSPs.

Tal como se muestra en los siguientes ejemplos al alcanzar consistencia de nodos se acumulan sobre $c_\emptyset$ costes que están implícitos en las restricciones unarias.

Ejemplo 6 *El problema de la Figura 3.a no es nodo consistente porque ninguna de sus dos variables tiene soporte. Para transformarlo en un problema nodo consistente podemos ejecutar el algoritmo NC de la Figura 5. Puesto que el problema es un WCSP, las líneas 3-6 se instancian en,*

3. $\quad c_\emptyset := c_\emptyset + m$;
4. $\quad$ ***for each*** $a \in D_i$ ***do***
5. $\qquad c_i(a) := c_i(a) - m$
6. $\quad$ ***endfor***

El resultado es el problema de la Figura 3.b, que sí es nodo consistente. El nuevo problema es más explícito porque $c_\emptyset = 3$ nos da una cota inferior del coste óptimo.

Ejemplo 7 *El problema de la Figura 4.a tampoco es nodo consistente porque ninguna de sus dos variables tiene soporte. Puesto que el problema es un CSP posibilista, las líneas 3-6 se instancian en,*

3. $\quad c_\emptyset := \max\{c_\emptyset, m\}$;
4. $\quad$ ***for each*** $a \in D_i$ ***do***
5. $\qquad c_i(a) := c_i(a)$
6. $\quad$ ***endfor***

Obviamente, las líneas 4 y 5 no tienen ningún efecto y se pueden eliminar. El resultado es el problema de la Figura 4.b, que sí es nodo consistente. El nuevo problema es más explícito porque $c_\emptyset = .2$ nos da una cota inferior del coste óptimo.

Consistencia de Arcos [22, 21]. La consistencia de arcos introduce información proveniente de las restricciones binarias. Dada una red de restricciones blandas, una variable x_i es arco consistente respecto a una restricción binaria c_{ij} ssi para todo $a \in D_i$ existe un valor $b \in D_j$ al que llamamos *soporte* de a en c_{ij} tal que $c_i(a) \oplus c_{ij}(a,b) = c_i(a)$. Una variable es arco consistente ssi lo es respecto a todas las restricciones en las que está implicada. La red de restricciones es arco consistente ssi es nodo consistente y todas las variables son arco consistentes.

Cuando un problema no es arco consistente se puede transformar en un problema equivalente que sí lo sea aplicando operaciones locales en las que sólo intervienen las restricciones unarias y binarias. En particular, la consistencia de arcos se consigue generalizando el concepto de proyección a las restricciones binarias: Cuando un valor $a \in D_i$ no tiene soporte en c_{ij}, se calcula $m = \min_{b \in D_j}\{c_{ij}(a,b)\}$. m se suma a $c_i(a)$ y se resta de los costes binarios $c_{ij}(a,b)$.

Ejemplo 8 *El problema de la Figura 3.b no es arco consistente porque $b \in x_1$ no tiene soporte en c_{12}. Para hacerlo arco consistente hay que sumar 1 a $c_1(b)$ y restarlo de $c_{12}(b,a)$ y $c_{12}(b,b)$. El problema resultante (Figura 3.c) tampoco es arco consistente porque la variable x_1 ha dejado de tener soporte. Para hacerlo nodo consistente hay que sumar 1 a $c_\emptyset$ y restar 1 a $c_1(a)$ y $c_1(b)$. El resultado, en la Figura 3.d sí es arco consistente. En el nuevo problema, la solución óptima con coste 4 se puede obtener de manera trivial.*

Ejemplo 9 *El problema de la Figura 4.b no es arco consistente porque $b \in x_1$ no tiene soporte en c_{12}. Para hacerlo arco consistente hay que sumar .3 a $c_1(b)$ y restarlo de $c_{12}(b,a)$ y $c_{12}(b,b)$. El problema resultante tampoco es arco consistente porque ha dejado de ser nodo consistente. Para hacerlo nodo consistente hay que sumar .3 a $c_\emptyset$ y restar .3 a $c_1(a)$ y $c_1(b)$. El resultado, en la Figura 3.c sí es arco consistente. En el nuevo problema, la solución óptima con coste .3 se puede obtener de manera trivial.*

En [21] se presenta un algoritmo de consistencia de arcos con coste $O(ed^3)$, siendo e el número de restricciones y d la cardinalidad máxima de los dominios.

4.2.2 Mini buckets

Otro tipo de inferencia incompleta es la que computa el algoritmo MB (acrónimo del inglés *Mini-Buckets*) [10]. Este algoritmo es una aproximación de BE que calcula una cota inferior de la solución óptima del problema. MB es un algoritmo parametrizado por k que indica el nivel de precisión deseada. Cuanto mayor sea el valor de k, mejor (mayor) será la calidad de la cota. En el caso extremo $k = n$, MB es equivalente a BE y por lo tanto calcula el valor óptimo. La complejidad temporal y espacial de MB es exponencial en k por lo que hay que encontrar el equilibrio entre los recursos disponibles y la precisión deseada del resultado.

MB funciona de manera similar a BE. Primero se establece un orden entre las variables $o = (x_1, x_2, \ldots, x_n)$. Seguidamente se procede a eliminar las variables una a una en orden decreciente. La diferencia entre BE y MB está en el proceso de eliminación de variables. Si la eliminación de la variable x_i es demasiado costosa (espacial y/o temporalmente), su cubo B_i se parte en trozos más pequeños y manejables. La eliminación de x_i se hace en cada trozo por separado. De esta manera se pierde la corrección del algoritmo, pero se mantiene la garantía de que el resultado final es una cota inferior del óptimo.

El parametro k sirve para especificar el esfuerzo máximo que se está dispuesto a hacer en cada eliminación. Consideremos la eliminación de una variable x_i, cuyo cubo es $B_i = \{f_{i_1}, \ldots, f_{i_k}\}$. La eliminación exacta que realiza BE es,

$$g_i = (\sum_{f \in B_i} f) \Downarrow x_i$$

La complejidad espacial y temporal de este proceso es exponencial en la aridad de g_i. Si esta aridad es mayor que el parámetro k, MB hace una partición de B_i en subconjuntos $B_{i_1}, \ldots, B_{i_k}$ tales que el número de variables que aparecen en los ámbitos de cada subconjunto sea menor o igual a k. Entonces MB calcula un conjunto de funciones g_{i_j},

$$g_{i_j} = (\sum_{f \in B_{i_j}} f) \Downarrow x_i, \quad j = 1..k$$

donde, por construcción, cada g_{i_j} tiene aridad acotada. Puesto que,

$$(\overbrace{\sum_{f\in B_i} f) \Downarrow x_i}^{g_i} \geq \sum_{j=1}^{k} (\overbrace{\sum_{f\in B_{i_j}} f) \Downarrow x_i}^{g_{i_j}}$$

este método genera una subestimación del coste óptimo del problema original. Es obvio que valores mayores de k permitiran tratar mini-cubos de tamaño mayor y no se perderá tanta precisión en el resultado. Por lo tanto es de esperar que incrementando k se consiga más precisión a costa de consumir más tiempo y espacio.

5 Algoritmos híbridos

Una técnica muy habitual en la resolución de VCSPs consiste en combinar algoritmos de búsqueda con algoritmos de inferencia. Puesto que las técnicas de inferencia se definen sobre instancias de VCSP es conveniente modificar el algoritmo de búsqueda sistemática de la Figura 1 para que cada nodo del espacio de búsqueda se corresponda con un subproblema. Intuitivamente, cuando se visita un nodo del árbol de búsqueda, el problema actual es el problema original sujeto a las asignaciones que se han ido haciendo en la rama que va desde la raiz hasta el nodo actual. Empezamos definiendo la operación *asignación* de una función,

Definición 5 *La* asignación $x_i \leftarrow a$ *en la función* f*, que denominamos* $f(x_i \leftarrow a)$*, es otra función cuyo ámbito es* $var(f) - \{x_i\}$ *y que devuelve para cada tupla* t *el coste de* $t \cdot (x_i \leftarrow a)$*,*

$$(f(x_i \leftarrow a))(t) = f(t \cdot (x_i \leftarrow a))$$

Ejemplo 10 *La asignación de* $x_1 \leftarrow 1$ *en la función* $f(x_1, x_2) = 2 - x_1 - x_2$ *da lugar a,* $(f(x_1 \leftarrow 1))(x_2) = 1 - x_2$

Notesé que cuando f es una función unaria asignar su única variable produce una función constante. La Figura 6 muestra el pseudo-código de BB', el algoritmo BB (Figura 1) modificado. BB' recibe como parámetro el subproblema a resolver (X, D, C) formado por las variables que aún no se han asignado, junto con los dominios y restricciones correspondientes. También recibe la asignación parcial t que se ha ido haciendo en la rama actual del árbol de búsqueda. En la llamada inicial t es la tupla vacía y (X, D, C) es el problema original. Si $X = \emptyset$ (línea 1) el problema actual no tiene variables, por lo que la única restricción que puede haber en C es $c_\emptyset$ cuyo valor es la solución del problema. En este caso se actualiza la cota superior y la mejor solución encontrada hasta el momento (líneas 2,3). Si $X \neq \emptyset$ se selecciona una variable x_i y se itera sobre sus valores. Para cada valor $a \in D_i$, se actualiza la asignación t (línea 7) y se modifica el problema actual para que incluya la nueva asignación $x_i \leftarrow a$. Esto se consigue asignando $(x_i \leftarrow a)$ en todas las restricciones que tienen x_i en su ámbito: En la línea 8, se asigna la restricción c_i, que se transforma en una constante que se suma a $c_\emptyset$; en la línea 9, se asignan todas las restricciones c_{ij} que pasan a ser restricciones unarias con ámbito x_j y que se suman a c_j. El nuevo problema es $(X - \{x_i\}, D, C)$. A continuación (línea 10) se calcula la cota inferior asociada al problema (una cota inferior trivial es $c_\emptyset$). Si la cota inferior es menor que la cota superior se resuelve recursivamente el nuevo problema actual, sino se poda el subárbol y se procede con el siguiente valor de x_i. Observesé que antes de pasar al nuevo valor hay que restaurar el contexto, es decir, recuperar el problema actual (X, D, C).

procedure `BB'`$(t, (X, D, C))$
1. **if** $(X = \emptyset)$**then**
2. $CS := c_\emptyset$
3. $mejorSol := t$
4. **else**
5. $x_i :=$ `SelectVar`(X)
6. **for each** $a \in D_i$ **do**
7. $t := t \cdot (x_i \leftarrow a)$
8. $c_\emptyset := c_\emptyset \oplus c_i(x_i \leftarrow a)$
9. $\forall c_{ij} \in C$ **do** $c_j := c_j \oplus c_{ij}(x_i \leftarrow a)$
10. $CI := c_\emptyset$
11. **if** $CI < CS$ **then**
12. `BB'`$(t, X - \{x_i\}, D, C))$
13. `restore context`
14. **endfor**
15. **endif**
endprocedure

Figura 6. Branch and bound genérico para CSPs valuados. En esta versión cada nodo del árbol de búsqueda es un subproblema del problema original.

5.1 Inferencia incompleta

Posiblemente la combinación búsqueda/inferencia más habitual sea la de entrelazar los algoritmos de búsqueda sistemática con inferencia incompleta. Esta idea, que está totalmente aceptada para el modelo CSP clásico, suele ser también una opción muy efectiva para el modelo VCSP.

Una posibilidad consiste en combinar la búsqueda con algún tipo de consistencia local como por ejemplo la consistencia de nodos o arcos descrita en la Sección 4.2.1. Ello se consigue insertando antes de la línea 10 del algoritmo BB' (Figura 6) una llamada al algoritmo de consistencia local correspondiente (p.ej. NC tal como aparece en la Figura 5). El algoritmo de consistencia local posiblemente incremente el valor de $c_\emptyset$ (y por lo tanto la cota inferior) con lo que se facilita la condición de poda. Como efecto lateral, el algoritmo puede podar valores de los dominios con lo que se reduce el espacio de búsqueda del problema en curso. Como cada tipo de consistencia local tiene un algoritmo diferente con un coste diferente, hay que decidir cuál es el más apropiado para el problema que se desee resolver. Para más detalles se puede consultar [20].

Una opción alternativa consiste en ejecutar para cada subproblema el algoritmo MB (Sección 4.2.2) y usar en la línea 10 la cota inferior resultante. Puesto que MB tiene un parámetro que permite especificar la precisión deseada, habrá que decidir el valor óptimo para el problema que se desee resolver. En general, contra más difícil es el problema, más efectivo resulta usar un valor alto del parámetro k. En [16, 7] se pueden encontrar más detalles sobre esta aproximación.

5.2 Inferencia completa

Otra aproximación híbrida que puede dar buenos resultados consiste en combinar búsqueda sistemática con inferencia completa. La aplicación más conocida de esta idea es el algoritmo de *corte de ciclos* (del inglés *cycle cutset* [8], que usa principalmente una estrategia de búsqueda, pero que activa BE para resolver el subproblema en curso cuando detecta que éste es acíclico. La motivación de esta estrategia es que para problemas acíclicos (es decir, cuando el grafo de restricciones del problema es un árbol) la estrategia de búsqueda tiene coste exponencial, mientras que BE tiene coste cuadrático.

La idea del corte de cíclicos se puede generalizar de dos maneras complementarias entre sí. Por ejemplo, en [17] se propone BE-BB(k) un algoritmo híbrido parametrizado por k. El algoritmo sigue una estrategia de búsqueda, pero en cada subproblema, antes de seleccionar la siguiente variable y proceder a asignar los diferentes valores, elimina aquellas variables que tengan como mucho k adyacentes en el grafo de restricciones. La eliminación de variables se hace tal como la haría BE (Sección 4.1). El comportamiento del algoritmo variará en función del valor de k. Cuando k toma su valor mínimo -1 no se permite ninguna eliminación de variable y BE-BB sigue una estrategia de búsqueda pura. Cuando $k = 1$ se puede demostrar que BE-BB es equivalente al algoritmo de corte de ciclos. Cuando k toma su valor máximo n todas las variables se pueden eliminar, y BE-BB degenera en BE. El parámetro k permite encontrar la combinación más apropiada de estas dos estrategias algorítmicas. BE-BB(k) se puede obtener insertando las siguientes líneas de código antes de la línea 1 en el algoritmo BT' de la Figura 6 (X^k denota el conjunto de variables de X cuyo número de adyacentes en el grafo de restricciones está acotado por k).

while $(X^k \neq \emptyset)$**do**
  $x_i :=$ `SelectVar`(X^k);
  `ElimVar`$(x_i, (X, D, C))$;
endwhile

En [11] se propone SBE(k) (acrónimo del inglés, *super-bucket elimination*), un algoritmo híbrido parametrizado complementario a BE-BB. SBE(k) sigue una estrategia de eliminación de variable, pero cuando se encuentra una variable x_i cuya eliminación es demasiado costosa en espacio (x_i tiene más de k adyacentes en el grafo de restricciones) trata de eliminar varias variables simultáneamente. La eliminación de un conjunto de variables Y se realiza de la siguiente manera: Primero se calcula el *super cubo* B_Y que contiene todas las restricciones del problema que tienen a alguna variable de Y en su ámbito. A continuación se calcula g_Y,

$$g_Y = (\bigoplus_{f \in B_Y} f) \Downarrow Y$$

Por definición de eliminación, la expresión anterior se puede escribir como,

$$g_Y(t) = \min_{t' \in \prod_{x_i \in Y} D_i} \{(\bigoplus_{f \in B_Y} f)(t \cdot t')\}$$

En realidad, calcular $g_Y(t)$ no es más que resolver un problema de optimización para cada tupla t y SBE los resuelve uno a uno usando un algoritmo de búsqueda.

El comportamiento del algoritmo variará en función del valor de k. Cuando k toma su valor mínimo 0 la única eliminación que se permite es la de todas las variables a la vez ($Y = X$). En este caso g_X es una constante y su valor es la asignación total t de coste mínimo. Por lo tanto SBE resuelve todo el problema mediante búsqueda. Cuando k toma su valor máximo n todas las variables se pueden eliminar individualmente, y SBE degenera en BE. Valores intermedios de k dan lugar a diferentes formas de combinar las dos estrategias.

6 Resumen

El uso de restricciones blandas aumenta la expresividad de los problemas de satisfacción de restricciones permitiendo que el usuario especifique la importancia de las restricciones. Resolver el problema consiste en encontrar la solución que *mejor* satisfaga las restricciones.

En este artículo hemos presentado los aspectos más relevantes de la modelización y resolución de problemas con restricciones blandas. Respecto a la modelización, hemos visto cómo el modelo VCSP unifica, bajo un modelo algebraico, los diferentes modelos que se han propuesto en la literatura, como los CSPs posibilisticos, probabilísticos, difusos, con pesos, *etc.*. Tal como hemos visto, el uso de este modelo permite describir las diferentes aproximaciones algorítmicas de manera unificada, sin tener que preocuparse por la semántica de las restricciones blandas.

Respecto a los algoritmos de resolución, hemos presentado las dos aproximaciones principales: *búsqueda* e *inferencia*. La búsqueda puede ser sistemática o no sistemática (no tratada en este artículo). La inferencia puede ser completa o incompleta. Como paradigma de búsqueda sistemática hemos descrito un algoritmo que combina las técnicas de *vuelta atrás* y *ramificación y poda*. Como paradigma de inferencia completa hemos descrito el algoritmo de *eliminación de cubos* (o *bucket elimination*). Estos dos algoritmos permiten resolver el problema de forma exacta. Sus propiedades son complementarias: La búsqueda tiene un coste temporal en el caso peor muy alto, pero un coste espacial muy favorable. La inferencia exacta tiene un coste temporal que puede ser bueno, pero a costa de consumir muchos recursos espaciales.

Tambien hemos descrito dos tipos de inferencia incompleta: consistencia local (de nodos y arcos) y el algoritmo parametrizado de *mini-cubos* (o *mini-buckets*). Estos algoritmos no resuelven el problema de forma exacta, pero deducen información útil (p.ej. cotas de la solución). Finalmente hemos visto algunas aproximaciones híbridas que combinan diferentes técnicas con el objetivo de obtener lo mejor de cada una de ellas.

Referencias

[1] M. S. Affane and H. Bennaceur. A weighted arc consistency technique for Max-CSP. In *Proc. of the 13^{th} ECAI*, pages 209–213, Brighton, United Kingdom, 1998.

[2] F. Barber and M.A. Salido. La programacion de restricciones. una introduccion. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 20, 2003.

[3] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[4] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4:199–240, 1999.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[6] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[7] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower bound computation in constraint optimization. In *CP-2001*, pages 346–360, 2001.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[9] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[10] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of the 13th UAI-97*, pages 132–141, San Francisco, 1997. Morgan Kaufmann Publishers.

[11] Rina Dechter and Yousri El Fatah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125(1–2):93–118, 2001.

[12] D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *IEEE conference on fuzzy sets*, 1993.

[13] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *ECSQARU*, 1993.

[14] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, December 1992.

[15] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*, pages 394–399, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[16] K. Kask. New search heuristics for max-csp. In *Proc. of the 6^{th} CP*, pages 262–277, Singapore, 2000. LNCS 1894. Springer Verlag.

[17] J. Larrosa and R. Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326, 2003.

[18] J. Larrosa and P. Meseguer. Satisfaccion de restricciones. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 20, 2003.

[19] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.

[20] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. In *Proc. of the 18^{th} IJCAI*, Acapulco, Mexico, August 2003.

[21] Javier Larrosa. Node and arc consistency in weighted csp. In *Proceedings of the 18th AAAI*, 2002.

[22] T. Schiex. Arc consistency for soft constraints. In *CP-2000*, pages 411–424, Singapore, 2000.

[23] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *IJCAI-95*, pages 631–637, Montréal, Canada, August 1995.

[24] Thomas Schiex. Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". In *UAI*, pages 268–275, Stanford, CA, 1992.