Facsímil 02

Inteligencia clásica

Capítulo 06

CSP: variables, dominios y restricciones

686f6c61 · 2026

Capítulo 06: CSP: variables, dominios y restricciones

Entrando en el tema

En el capítulo anterior vimos la idea general: un CSP busca una asignación válida. Ahora toca la parte que parece pequeña y lo cambia todo: cómo eliges las variables, qué valores permites en cada dominio y qué reglas escribes como restricciones.

Parece una decisión de nomenclatura, pero no lo es. Si modelas mal, el solver se ahoga. Si modelas bien, el problema se vuelve transparente. Es la diferencia entre decir “organiza todos los turnos de la semana” y decir “para cada persona y cada día, elige mañana, tarde, noche o libre”.

Un CSP bien modelado no empieza con un algoritmo. Empieza con una pregunta humilde: ¿cuáles son exactamente los huecos que tengo que rellenar?

El problema no es el solver, es el modelado

Piensa en una hoja de cálculo. Cada celda vacía es una decisión pendiente. Algunas celdas solo aceptan ciertos valores. Algunas combinaciones entre celdas están prohibidas. Si rellenas todo sin romper ninguna regla, tienes una solución.

PIEZA DEL CSP	IMAGEN COTIDIANA	EJEMPLO
Variable	Hueco que hay que rellenar.	Hora de la reunión de producto.
Dominio	Valores permitidos para ese hueco.	9:00, 10:00 o 11:00.
Restricción	Regla que descarta combinaciones.	Ana no puede estar en dos reuniones a la vez.
Asignación parcial	Hoja a medio rellenar.	Producto a las 9:00; cliente todavía sin hora.
Solución	Hoja completa sin reglas rotas.	Todas las reuniones colocadas sin solapes.

Esta forma de pensar viene de la programación con restricciones: representar un problema como variables, dominios y relaciones permitidas, y después buscar o propagar hasta encontrar consistencia.[1] La parte difícil rara vez es escribir `for value in domain`. La parte difícil es decidir qué cuenta como variable.

Variables: qué huecos vamos a rellenar

Una variable CSP representa una decisión pendiente. No tiene por qué ser “una cosa del mundo”; puede ser una combinación que nos conviene para modelar. En horarios, una variable puede ser `reunión`, `persona-día`, `aula-hora`, `paquete-versión` o `tarea-máquina`.

Formalmente:

$$X = \{X_1, X_2, \ldots, X_n\}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
X	Conjunto de todas las variables del problema.	Tres cursos que hay que colocar en horario.
X_i	Variable individual número i.	$X_1 = \text{Curso IA}$.
n	Número total de variables.	$n = 3$.

Ejemplo concreto:

VARIABLE	QUÉ DECISIÓN REPRESENTA	COMENTARIO
X_1	Dónde y cuándo colocar “Curso IA”.	Lo imparte Ana.
X_2	Dónde y cuándo colocar “Curso Python”.	También lo imparte Ana.
X_3	Dónde y cuándo colocar “Curso Datos”.	Necesita sala B.

Aquí hemos elegido una variable por curso. Podríamos haber elegido una variable por franja horaria y sala, pero entonces el valor sería “qué curso pongo aquí”. Las dos opciones pueden ser correctas. La buena es la que permite expresar reglas con menos esfuerzo y menos confusión.

Dominios: qué valores puede tomar cada variable

El dominio de una variable es el conjunto de valores que puede tomar. Si la variable es un curso, su dominio puede ser el conjunto de pares `(hora, sala)`.

$$D_i = \{v_{i1}, v_{i2}, \ldots, v_{ik_i}\} \qquad X_i \in D_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
D_i	Dominio de la variable X_i.	Posibles combinaciones de hora y sala.
v_{ij}	Valor j permitido para X_i.	$(9, \text{A})$.
k_i	Tamaño del dominio de X_i.	Si hay 2 horas y 2 salas, $k_i = 4$.
$X_i \in D_i$	La variable debe recibir un valor de su dominio.	“Curso IA” puede ir a $(9, \text{A})$.

Para nuestro ejemplo:

$$D_1 = D_2 = D_3 = \{(9, A), (9, B), (10, A), (10, B)\}$$

Cada curso puede colocarse a las 9:00 o a las 10:00, en sala A o B. Sin restricciones, el número de asignaciones posibles es:

$$|\mathcal{A}| = \prod_{i=1}^{n} |D_i|$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\mathcal{A}$	Conjunto de asignaciones completas candidatas.	Todos los horarios posibles antes de filtrar.
$	\mathcal{A}	$
$	D_i	$
$\prod$	Producto de los tamaños de dominio.	Multiplicar $4 \cdot 4 \cdot 4$.

Con tres cursos y cuatro opciones por curso:

$$|\mathcal{A}| = 4^3 = 64$$

Sesenta y cuatro horarios candidatos no parecen muchos. Pero si tienes 30 eventos y cada uno tiene 20 opciones, el espacio sería 20^{30}. Ahí ya no estás rellenando una hoja: estás mirando un océano.

Auditar dominios antes de resolver

Un buen modelo CSP no espera al solver para eliminar lo obvio. Si una restricción unaria dice que Python solo puede ir a las 10:00, no tiene sentido conservar valores de Python a las 9:00 en el dominio inicial. Eso no es “hacer trampa”: es escribir mejor el problema.

Podemos comparar dos tamaños:

$$|\mathcal{A}_{bruta}| = \prod_i |D_i|$$

$$|\mathcal{A}_{podada}| = \prod_i |D'_i|$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
D_i	Dominio original.	Python: $(9, A), (9, B), (10, A), (10, B)$.
D'_i	Dominio podado por reglas unarias.	Python: $(10, A), (10, B)$.
$	\mathcal{A}_{bruta}	$
$	\mathcal{A}_{podada}	$

La diferencia entre 64 y 16 en un ejemplo pequeño no impresiona demasiado. La diferencia entre 20^{30} y algo dos órdenes de magnitud menor sí puede decidir si tu sistema responde hoy o nunca.

Este podado por restricciones unarias tiene nombre propio en la teoría de CSP: **consistencia de nodo**. Un CSP es consistente de nodo cuando ningún dominio guarda valores que ya violan una restricción unaria. Conseguirla es barato, porque basta mirar cada variable por separado, y siempre conviene hacerla antes de buscar. Es el primer escalón de una escalera de consistencias: en el capítulo 7 subiremos al siguiente, la **consistencia de arco**, que en vez de mirar una variable mira pares de variables conectadas por una restricción.

Restricciones: qué combinaciones quedan prohibidas

Una restricción dice qué combinaciones de valores son aceptables. Puede afectar a una variable, a dos o a muchas. Montanari formalizó estas redes de restricciones como relaciones entre variables, una idea que después se volvió central en CSP.[2]

Formalmente:

$$C_k = (S_k, R_k)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
C_k	Restricción número k.	“Ana no puede impartir dos cursos a la misma hora”.
S_k	Alcance: variables afectadas por la restricción.	$S_k = (X_1, X_2)$.
R_k	Relación permitida entre valores.	Hora de X_1 distinta de hora de X_2.

En nuestro horario:

TIPO	REGLA	CÓMO SE LEE
Unaria	$hora(X_2) = 10$	Python solo puede ir a las 10:00.
Unaria	$sala(X_3) = B$	Datos necesita sala B.
Binaria	$hora(X_1) \neq hora(X_2)$	Ana imparte IA y Python; no puede duplicarse.
Global	`all_different((hora, sala))`	Dos cursos no pueden ocupar la misma sala a la misma hora.

Una asignación a es válida cuando todas las restricciones son verdaderas:

$$\text{valida}(a) = \bigwedge_{C_k \in C} C_k(a)$$

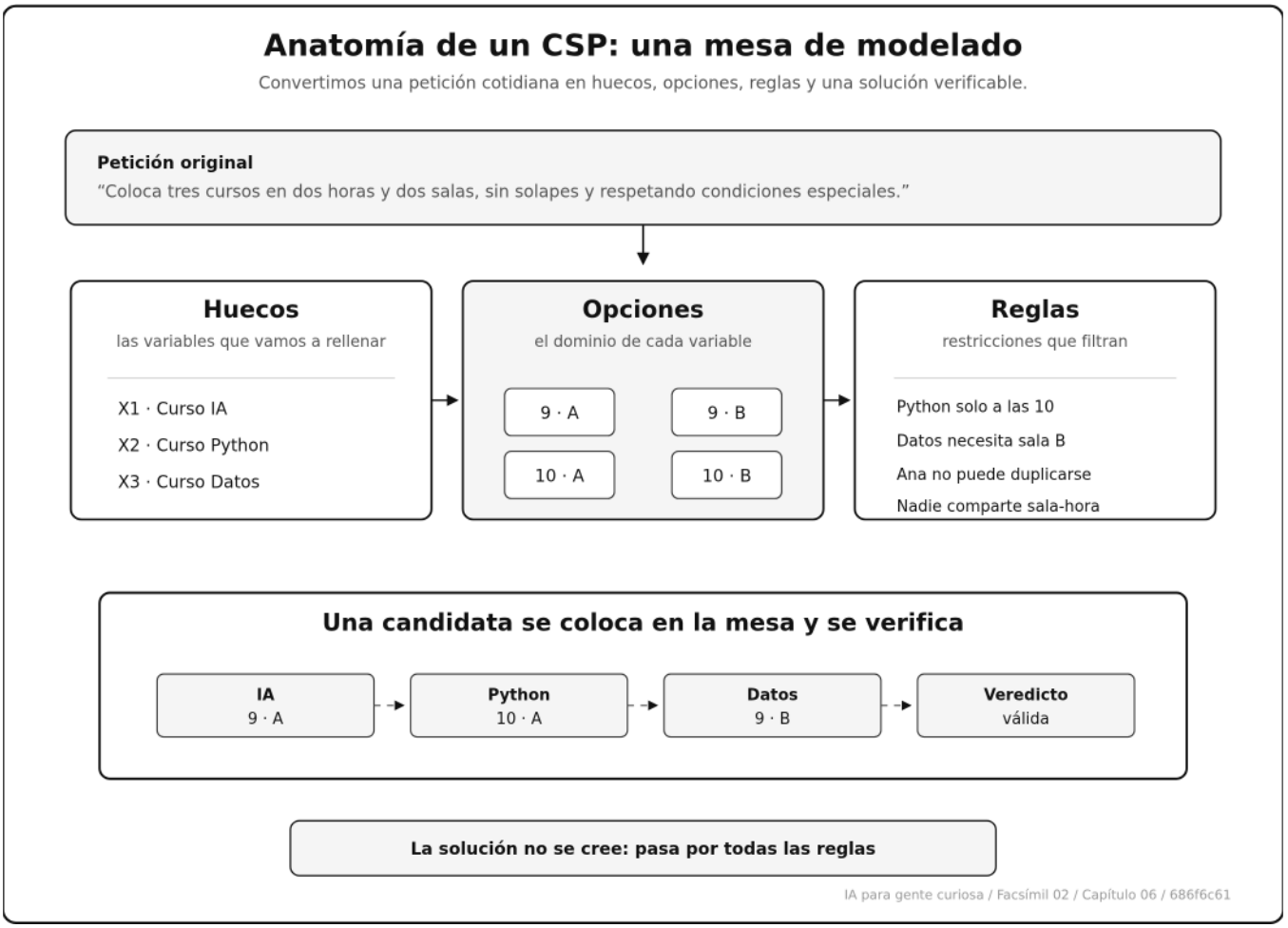
SÍMBOLO	SIGNIFICADO	EJEMPLO
a	Asignación completa de valores.	IA $(9, A)$, Python $(10, A)$, Datos $(9, B)$.
$\text{valida}(a)$	Indica si la asignación cumple todas las reglas.	Verdadero si no hay solapes ni salas incorrectas.
$\bigwedge$	“Y” lógico: todo debe cumplirse.	Si una regla falla, toda la asignación falla.
$C_k(a)$	Resultado de evaluar la restricción k sobre a.	$hora(X_1) \neq hora(X_2)$.

Probemos esta asignación:

$$a(X_1) = (9, A), \qquad a(X_2) = (10, A), \qquad a(X_3) = (9, B)$$

RESTRICCIÓN	SUSTITUCIÓN	RESULTADO
$hora(X_2) = 10$	$10 = 10$	Verdadero
$sala(X_3) = B$	$B = B$	Verdadero
$hora(X_1) \neq hora(X_2)$	$9 \neq 10$	Verdadero
No compartir sala y hora	$(9, A), (10, A), (9, B)$ son distintos	Verdadero

La asignación es válida. No porque “parezca buena”, sino porque supera cada regla.



Restricciones unarias, binarias y globales

Conviene poner nombre a los tipos de restricción porque cada uno cambia cómo se resuelve el problema.

TIPO	AFECTA A	EJEMPLO ENTENDIBLE	PAPEL PRÁCTICO
Unaria	Una variable.	Python solo puede ir a las 10:00.	Reduce un dominio individual.
Binaria	Dos variables.	IA y Python no pueden tener la misma hora.	Conecta dos decisiones.
Global	Muchas variables.	Ningún curso comparte sala y hora.	Expresa reglas de conjunto sin escribir miles de pares.
Blanda	Una o muchas variables, con penalización.	Preferimos sala A, pero sala B vale.	Optimiza sin convertir preferencias en imposibles.

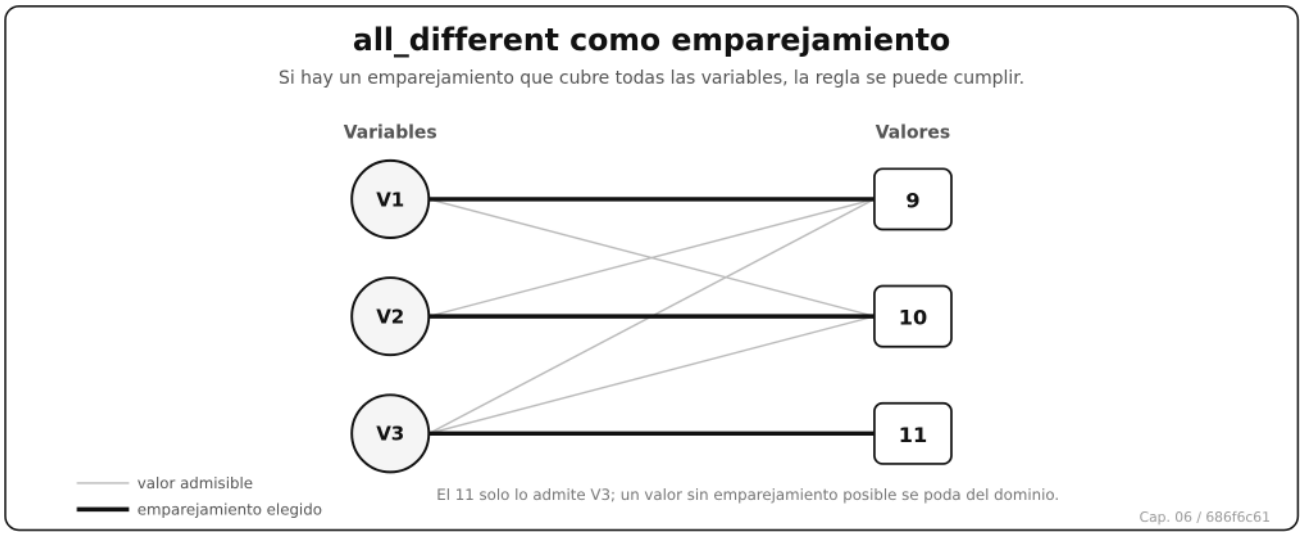
Mackworth mostró que muchas restricciones binarias pueden verse como arcos entre variables y que limpiar inconsistencias locales reduce muchísimo la búsqueda posterior.[3] Esta idea nos llevará al capítulo 7: propagación y *backtracking*.

La **aridad** de una restricción es cuántas variables toca. Una unaria tiene aridad 1; una binaria, aridad 2; una global puede tener aridad n. Esto importa porque afecta a cómo depuras el problema. Una restricción unaria suele explicar fallos muy localmente: “Python no puede ir a las 9”. Una global puede explicar fallos de conjunto: “dos cursos comparten sala-hora”. En sistemas reales conviene que cada restricción tenga identificador, descripción humana y datos suficientes para explicar por qué rechazó una asignación.

Las restricciones globales no son solo abreviatura

Es tentador pensar que una restricción global como `all_different(X_1, ..., X_n)`, que obliga a que todas las variables tomen valores distintos, es solo una forma corta de escribir muchas restricciones binarias $X_i \neq X_j$. Para *expresar* el problema, lo es. Para *resolverlo*, no: una restricción global trae su propio algoritmo de propagación, bastante más potente que mirar los pares uno a uno.

El caso de `all_different` es el ejemplo clásico. Régin demostró en 1994 que se puede propagar viéndolo como un problema de **emparejamiento** en un grafo bipartito: a un lado las variables, al otro los valores, y una arista por cada valor que una variable todavía admite.[4] Por el **teorema de Hall**, existe una asignación que respeta `all_different` si y solo si ese grafo tiene un emparejamiento que cubre todas las variables; y los valores que no pueden formar parte de ningún emparejamiento se eliminan de los dominios de golpe. Las $n(n-1)/2$ restricciones binarias equivalentes nunca habrían podado tanto. Esta es la lección de «lo real»: cuando reconoces un patrón global (todos distintos, capacidad máxima, una secuencia), usar la restricción global con nombre no solo es más legible, sino que el solver razona mejor con ella.



El grafo de restricciones

Un CSP con restricciones binarias se dibuja de forma natural como un **grafo de restricciones**: cada variable es un nodo y cada restricción binaria es una arista entre los dos nodos que conecta. Las restricciones unarias se anotan en su propio nodo; las globales, que tocan más de dos variables, son **hiperaristas** que envuelven a todas las que afectan. Esta imagen no es decoración: es la estructura sobre la que trabajan los algoritmos del capítulo 7. La consistencia de arco, por ejemplo, recorre precisamente las aristas de este grafo.

Ver el problema como grafo cambia cómo lo piensas. Una variable con muchas aristas es un cuello de botella y conviene asignarla pronto. Una parte del grafo que no se conecta con el resto es un subproblema independiente que puede resolverse por separado. Y un fallo se explica señalando la arista, o la hiperarista, que ninguna asignación consigue satisfacer.

En el día a día

En producto, los dominios aparecen como opciones de configuración. Un plan puede aceptar o no ciertos módulos, regiones, monedas, permisos o límites. Cada elección abre y cierra posibilidades.

En operaciones, los dominios aparecen como calendarios, turnos, recursos y máquinas disponibles. Si el dominio incluye valores que nunca deberían usarse, el solver perderá tiempo y quizás proponga soluciones absurdas.

En agentes con LLMs, las variables pueden ser más abstractas: qué herramienta usar, qué permisos pedir, qué paso ejecutar después, qué formato devolver. Nilsson ya presentaba búsqueda, planificación y agentes como problemas de decisión secuencial; los CSP añaden una capa útil cuando esas decisiones deben respetar restricciones explícitas.[5]

Por qué debería importarte

Porque modelar un CSP es diseñar el contrato entre el mundo y el solver. Si haces variables demasiado grandes, el dominio explota. Si haces variables demasiado pequeñas, las restricciones se vuelven difíciles de leer. Si confundes reglas duras con preferencias, el sistema rechaza soluciones útiles o acepta soluciones peligrosas.

Russell y Norvig insisten en que la representación importa tanto como el algoritmo: un buen espacio de estados o una buena formulación de restricciones puede hacer que una búsqueda difícil se vuelva manejable.[6] En IA aplicada, esa frase se traduce así: antes de pedirle inteligencia al solver, dale un problema bien escrito.

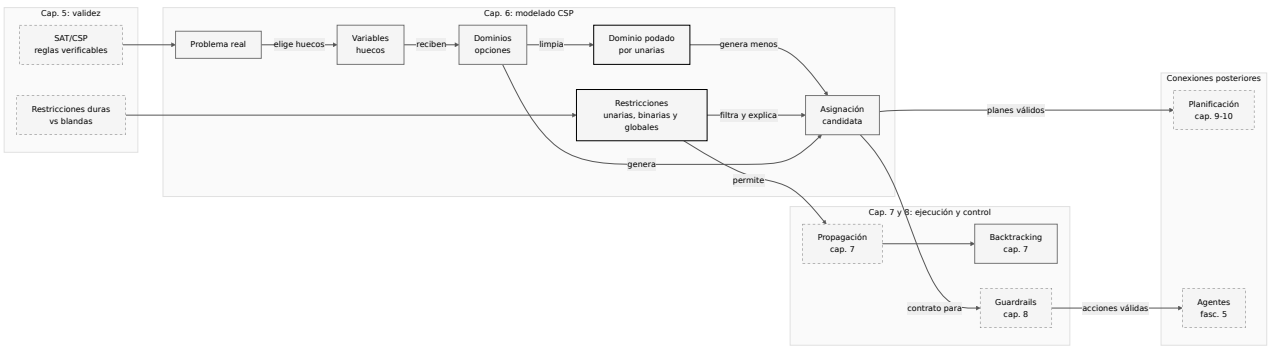
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Elegir variables demasiado grandes	Una variable “horario completo de la semana” tiene un dominio gigantesco y opaco.	Divide en decisiones pequeñas: curso, reunión, persona-día, tarea-máquina.
Meter valores imposibles en el dominio	Si Ana nunca trabaja de noche, no pongas “noche” en su dominio para filtrarlo después.	Limpia dominios antes de escribir restricciones complejas.
Escribir restricciones repetidas a mano	Cien reglas binarias pueden ocultar una regla global sencilla.	Busca patrones: “todos distintos”, “exactamente dos”, “al menos uno”, “capacidad máxima”.
Confundir ausencia de solución con fallo del solver	A veces el problema está realmente sobre-restringido.	Prueba con menos restricciones y añade reglas una a una para localizar la contradicción.

Cómo encaja todo

Este mapa se lee como una cadena de modelado. El capítulo 5 nos dio la idea general de restricción; este capítulo baja al contrato operativo de un CSP: qué variables existen, qué valores pueden tomar y qué reglas eliminan combinaciones.

La decisión importante no es elegir solver todavía. Es escribir un modelo que se pueda auditar, podar y explicar antes de buscar soluciones.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Variable CSP	Decisión pendiente que hay que rellenar con un valor.
Dominio	Conjunto de valores permitidos para una variable.
Restricción unaria	Regla que afecta a una sola variable.
Restricción binaria	Regla que relaciona dos variables.
Restricción global	Regla que afecta a muchas variables a la vez, con algoritmo de propagación propio (como `all_different`).
Grafo de restricciones	Representación del CSP: variables como nodos, restricciones binarias como aristas.
Consistencia de nodo	Estado en el que ningún dominio guarda valores que violan una restricción unaria.
Asignación parcial	Estado donde algunas variables ya tienen valor y otras no.
Asignación completa	Estado donde todas las variables tienen valor.
Solución	Asignación completa que cumple todas las restricciones duras.
Aridad	Número de variables que toca una restricción.
Dominio podado	Dominio reducido antes de buscar, normalmente por restricciones unarias.

Antes de pasar página

- [ ] ¿Puedo explicar qué es una variable CSP usando el ejemplo de una agenda? (Si no, vuelve a «Variables: qué huecos vamos a rellenar».)

- [ ] ¿Sé calcular el número de candidatos con $|\mathcal{A}| = \prod_i |D_i|$? (Si no, vuelve a «Dominios: qué valores puede tomar cada variable».)

- [ ] ¿Entiendo qué es la consistencia de nodo y por qué podar dominios antes de buscar? (Si no, vuelve a «Auditar dominios antes de resolver».)

- [ ] ¿Distingo una restricción unaria de una binaria y una global? (Si no, vuelve a «Restricciones unarias, binarias y globales».)
- [ ] ¿Sé dibujar el grafo de restricciones de un problema pequeño? (Si no, vuelve a «El grafo de restricciones».)
- [ ] ¿Entiendo por qué elegir variables demasiado grandes puede hacer explotar el problema? (Si no, vuelve a «Dónde solía tropezar yo».)
- [ ] ¿Sé explicar la diferencia entre el espacio bruto y el dominio podado? (Si no, vuelve a «Dominios: qué valores puede tomar cada variable».)

En resumen

IDEA FUERZA	DETALLE
Modelar es elegir huecos.	Las variables son las decisiones que el solver debe rellenar.
El dominio controla el tamaño del problema.	Tres variables con cuatro opciones generan $4^3 = 64$ candidatos; con muchas variables, el crecimiento explota.
Podar dominios también es ingeniería.	Las restricciones unarias pueden eliminar valores antes de que empiece la búsqueda.
Las restricciones convierten candidatos en soluciones.	Una asignación solo vale si pasa todas las reglas duras.
La calidad del CSP depende del modelado.	Un solver bueno no compensa variables mal elegidas ni dominios llenos de valores imposibles.

Para saber más

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1), 99-118. https://doi.org/10.1016/0004-3702(77)90007-8

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95-132. https://doi.org/10.1016/0020-0255(74)90008-5

Nilsson, N. J. (1998). *Artificial intelligence: A new synthesis*. Morgan Kaufmann.

Poole, D., Mackworth, A. y Goebel, R. (1998). *Computational intelligence: a logical approach*. Oxford University Press.

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. En *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* (pp. 362-367). AAAI Press.

Rossi, F., van Beek, P. y Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier.

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Notas

1. Rossi, F., van Beek, P. y Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier. El manual organiza el campo precisamente alrededor de modelado, propagación, búsqueda y optimización. ↩

2. Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7, 95-132. https://doi.org/10.1016/0020-0255(74)90008-5 ↩

3. Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1), 99-118. https://doi.org/10.1016/0004-3702(77)90007-8 ↩

4. Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. En *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* (pp. 362-367). AAAI Press. El artículo introduce el filtrado de `all_different` por emparejamiento en grafos bipartitos, hoy estándar en los solvers de restricciones. ↩

5. Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Morgan Kaufmann. ↩

6. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. ↩