Satisfacción/Optimización de Restricciones II

Víctor Giménez

Inteligencia Artificial - 2021/2022 1Q

CS - GEI- FIB

Generalización

Hasta ahora hemos escrito el algoritmo teniendo un tipo de propagador de la restricción concreto (arco entre dos variables). ¿Se puede generalizar? ¿Qué aporta generalizarlo?

Componentes del estado hasta ahora:

- Variables
- Dominios (valores posibles para las variables) **EXPLÍCITOS**
- Restricciones **binarias** entre las variables

Algoritmo General de CP (básico)

Constará de 2 fases:

- Propagación de restricciones de forma inteligente (antes, arco-consistencia)
- Búsqueda de valores dentro de los todavía posibles (asignación igual que antes, o incluso más sofisticado)

Propagador

- Representación **funcional** (posiblemente parcial) de una restricción
- Un propagador es una función de D a D (domain-to-domain) que 'reduce' el dominio de una *variable* en función del dominio actual (del resto de variables).
    - En una restricción binaria r, tenemos 2 propagadores, uno para cada variable (X, Y) y que depende de la otra.
    - Ahora los definimos más 'formalmente' en función de la restricción que codifican realmente. Esto requiere de un 'traductor' de restricción a propagadores (antes usábamos una comprobación directa de compatibilidad entre pares de valores).
    - Ejemplo $D'(X) = \{x \in D(X) | \exists y \in D(Y) : \ compatible(x, y)\}$ y viceversa.

Tipos de propagadores

- Un propagador puede ser un propagador de dominio (el más informado) o de otro tipo (como de límites).
- Ejemplo de propagadores para X = |Y|
  - **Dominio** (Equivale a arco-consistencia)
    - $D'(X) = D(X) \cap [D(Y) \cup D(-Y)] \setminus [-\infty.. - 1]$: valores en ambos dominios sin contar signo deben coincidir, y X debe ser positiva.
    - $D'(Y) = D(Y) \cap [D(X) \cup D(-X)]$: lo mismo, pero Y puede ser negativa.
    - Ejemplo para $D(X) = \{-4, 0, 3, 10\}, D(Y) = \{3, 4, 10\}$: $D(Y) = \{3, 4, 10\} \rightarrow D(X) = \{3, 10\} \rightarrow D(Y) = \{3, 10\}$
  - **Límites**
    - $D'(X) = D(X) \cap [0..m], m = max(D(Y), -min(D(Y)))$: X puede ir de 0 hasta el máximo en valor absoluto de Y.
    - $D'(Y) = D(Y) \cap [-max(D(X)), max(D(X))]$: Y puede ir de del máximo de X en negativo a en positivo.
    - Ejemplo para $X = \{-4.,10\}, Y = \{-4.,10\}$: $X = \{0.,10\}, Y = \{-4.,10\}$
- ¿Por qué nos importaría el de límites si el otro reduce mejor los dominios?

Algoritmo de propagación

Function: propagation(F_o, F_n, D)

$\mathsf{F} = \mathsf{F}_o \cup F_n, Q = F_n$
while *not Q.empty()* **do**
  f = choose(Q)
  D' = f(D)
  Q = Q $\cup wake(f, F, D, D')$
  D = D'
return D

- F_o son propagadores que 'inicialmente' no hay que mirar (truco para luego).
- Tenemos una cola (Q) con propagadores que sí hay que mirar
- A cada iteración, escogemos un propagador y lo propagamos. Luego, añadimos propagadores que 'despiertan'. En arcoconsistencia: añadir todos los arcos originados en la variable cambiada.
- Sutileza: ¡no limitamos los propagadores a afectar a una variable! Asumimos que trabajan a nivel del dominio general, y que 'wake' ya sabrá qué toca despertar.
- Cada solver programa esto con sus heurísticos propios

Algoritmo de propagación y Búsqueda

```
Function: prop_search(F_o, F_n, D)
D = propagation(F_o, F_n, D)
if D = ∅ or∀X|D(X)| = 1 then
  └ return D
X = choose(vars(D)\fixed(D))
F = F_o ∪F_n
foreach v ∈ D(X).order_by_heuristic() do
  │ D' = prop_search(F, {new_propagator(X=v)}, D)
  │ if D' ≠ ∅ then
  │   └ return D'
```

- Se llama con $prop\_search(\emptyset, F, D)$, todos los propagadores se tienen que comprobar primero.
- $fixed(D)$ devuelve las variables con dominio 1 (fijadas a un valor).
- $choose$ y $order\_by\_heuristic()$: Hay mejores y peores, definen la estrategia de resolución. La mayoría de programas modernos permiten sobreescribirlas, pero hay defaults.
- *propagation* permite explorar solo los propagadores que me importan: un propagador fija X, y 'wake' se encargará de despertar al resto de propagadores si hace falta.
- Se puede cambiar el bucle de búsqueda para ser más sofisticado, como usar propagadores tipo $new\_propagator(X < v)$ para hacer búsqueda dicotómica.

Optimización

¿Cómo pasar de satisfacción de restricciones a optimización de restricciones?

- De DFS pasamos a BranchBound añadiendo que 'podamos' caminos en función del coste.
- Podemos hacer lo mismo aquí: dada una variable numérica O a maximizar (o minimizar), encontrada una solución de coste v, la guardamos y añadimos a todos los Fs del stack $new\_propagator(O > v)$ para ver si hay algo mejor.
- Si esta es la técnica a seguir, es buena idea configurar los heurísticos para diezmar los valores posibles de la variable objetivo: buscando podar el espacio lo antes posible. Por ejemplo, asignando O la primera con un propagador $new\_propagator(O > mid(O))$. Las mejoras son exponenciales.

Minizinc

MiniZinc es un programa (con IDE) para la descripción en alto nivel de problemas CSP y COP. El lenguaje de alto nivel se compila a restricciones de bajo nivel (con más variables y loop unrolling). Luego, se traduce a un lenguaje de propagadores propio de cada solver disponible (eg. GeCode) optimizado para dichos propagadores.

El lenguaje además permite el uso de constraints 'globales' ultra optimizadas para cada *solver* (como *all_different(Vars)*)