Tema 10: Aplicaciones de SBC: Problemas de búsqueda

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Numeración romana

- **Enunciado:**
  El siguiente conjunto de reglas lee un número x en notación decimal y, si está entre 1 y 3999, escribe su notación romana:
  - **Si $x > 3999$, entonces escribir "demasiado grande" y terminar.**
  - **Si $x = 0$, entonces terminar.**
  - **Si $1 \leq x \leq 3$, entonces escribir "I" y reducir x en 1.**
  - **Si $x = 4$, entonces escribir "IV" y terminar.**
  - **Si $5 \leq x \leq 8$, entonces escribir "V" y reducir x en 5.**
  - **Si $x = 9$, entonces escribir "IX" y terminar.**
  - **Si $10 \leq x \leq 39$, entonces escribir "X" y reducir x en 10.**
  - **Si $40 \leq x \leq 49$, entonces escribir "LX" y reducir x en 40.**
  - **Si $50 \leq x \leq 89$, entonces escribir "L" y reducir x en 50.**
  - **Si $90 \leq x \leq 99$, entonces escribir "XC" y reducir x en 90.**

Numeración romana

- **Enunciado**
  - **Si $100 \leq x \leq 399$, entonces escribir "C" y reducir x en 100.**
  - **Si $400 \leq x \leq 499$, entonces escribir "CD" y reducir x en 400.**
  - **Si $500 \leq x \leq 899$, entonces escribir "D" y reducir x en 500.**
  - **Si $900 \leq x \leq 999$, entonces escribir "CM" y reducir x en 900.**
  - **Si $1000 \leq x \leq 3999$, entonces escribir "M" y reducir x en 1000.**
- **Sesión**

```
CLIPS> (assert (numero 27))
<Fact-0>
CLIPS> (run)
XXVII
CLIPS> (assert (numero 1996))
<Fact-7>
CLIPS> (run)
MCMXCVI
```

Numeración romana

- **Base de conocimiento**

```
(defrule demasiado-grande
  ?h <- (numero ?x&:(not (integerp ?x))
                 |:(< ?x 0)|:(> ?x 3999))
  =>
  (retract ?h)
  (printout t ?x " no es un número, o es negativo,"
              " o es demasiado grande." crlf))

(defrule inicial
  (numero ?x&:(integerp ?x)
            &:(<= ?x 3999)&:(>= ?x 0))
  =>
  (assert (numero-aux ?x)))

(defrule reduce-0
  ?h <- (numero-aux 0)
  =>
  (retract ?h)
  (printout t "" crlf))

(defrule reduce-1
  ?h <- (numero-aux ?x&:(<= 1 ?x 3))
  =>
  (retract ?h)
  (printout t "I")
  (assert (numero-aux (- ?x 1))))
```

Numeración romana

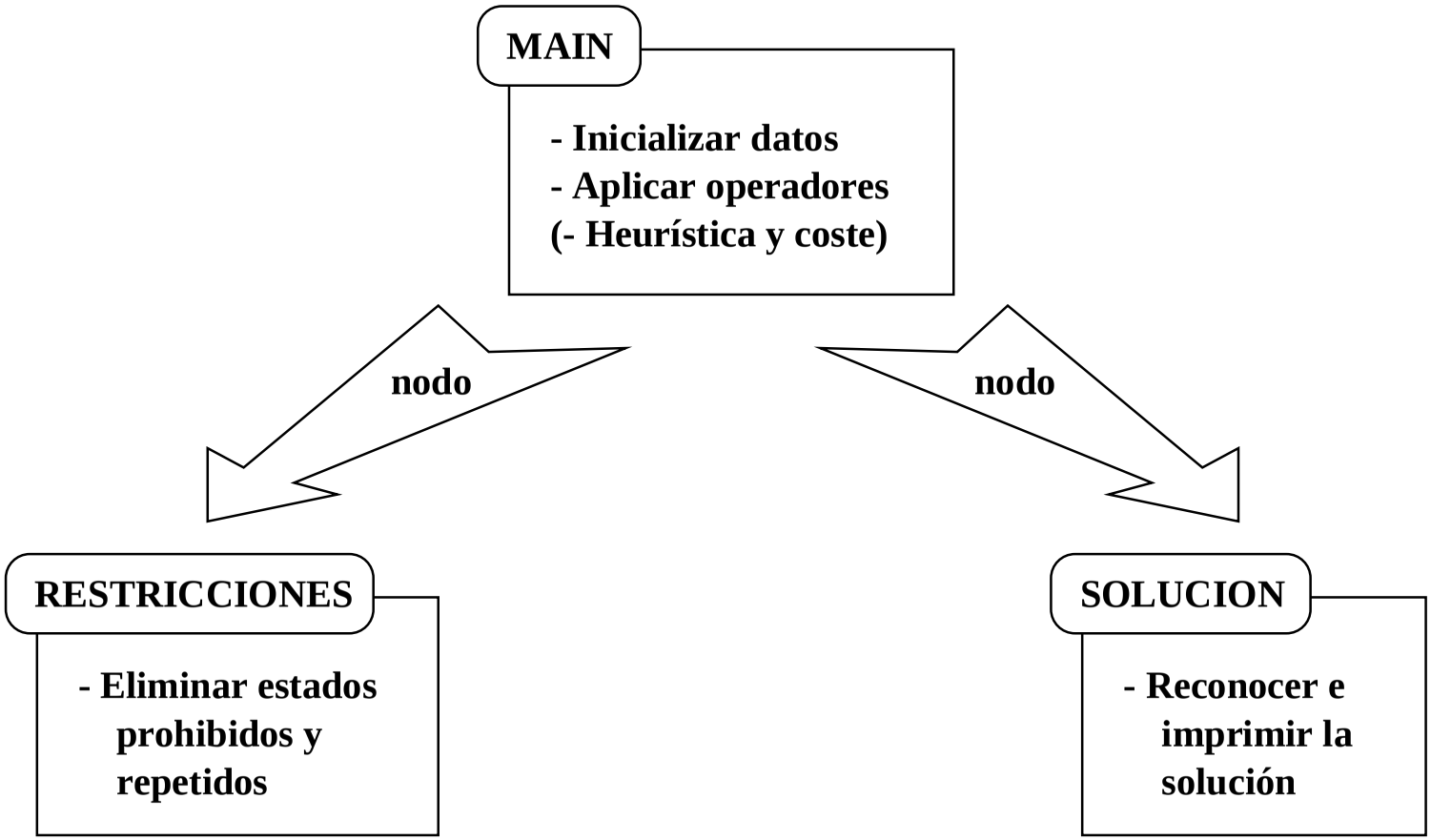
- **Base de conocimiento**

```
(defrule reduce-4
  ?h <- (numero-aux 4)
  =>
  (retract ?h)
  (printout t "IV" crlf))

(defrule reduce-5
  ?h <- (numero-aux ?x&:(<= 5 ?x 8))
  =>
  (retract ?h)
  (printout t "V")
  (assert (numero-aux (- ?x 5))))

(defrule reduce-9    ... )
(defrule reduce-10   ... )
(defrule reduce-40   ... )
(defrule reduce-50   ... )
(defrule reduce-90   ... )
(defrule reduce-100  ... )
(defrule reduce-400  ... )
(defrule reduce-500  ... )
(defrule reduce-900  ... )
(defrule reduce-1000 ... )
```

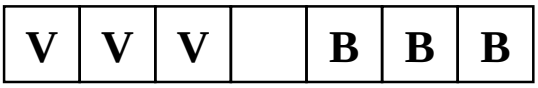
Esquema general de los problemas de búsqueda



El problema de las fichas

- **Enunciado**
  - **La situación inicial es**

  - **La situación final es**



  - **Los movimientos permitidos consisten en desplazar una ficha al hueco saltando, como máximo, sobre otras dos.**

- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino))

(deffacts MAIN::nodo-inicial
  (nodo (estado B B B H V V V)
        (camino "B B B H V V V")))
```

El problema de las fichas

- **Módulo MAIN**

```
;;; REGLA: movimiento-izquierda
;;; SI
;;;   en un nodo, una ficha tiene el hueco a su
;;;     izquierda, a una distancia de, como máximo,
;;;     dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo en el que movemos la
;;;     ficha al hueco.

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x
                H
                $?y&:(<= (length $?y) 2)
                ?ficha
                $?z)
        (camino $?movimientos))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado)))))
```

El problema de las fichas

- **Módulo MAIN**

```
;;; REGLA: movimiento-derecha
;;; SI
;;;   en un nodo, una ficha tiene el hueco a su
;;;     derecha, a una distancia de, como máximo,
;;;     dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo en el que movemos la
;;;     ficha al hueco.

(defrule MAIN::movimiento-derecha
  (nodo (estado $?x
                ?ficha
                $?y&:(<= (length $?y) 2)
                H
                $?z)
        (camino $?movimientos))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado)))))
```

El problema de las fichas

- **Módulo RESTRICCIONES**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

;;; REGLA: repeticion-de-nodo
;;; SI
;;;   hay un nodo repetido
;;; ENTONCES
;;;   eliminamos el de mayor profundidad.

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (camino $?movimientos))
  ?nodo <- (nodo (estado $?actual)
                 (camino $?movimientos ? $?))
  =>
  (retract ?nodo))
```

El problema de las fichas

- **Módulo SOLUCION**

```
(defmodule SOLUCION
  (import MAIN deftemplate nodo))

;;; REGLA: reconoce-solucion
;;; SI
;;;   hay un nodo en el que todas las fichas blancas
;;;     están a la derecha y todas las verdes a la
;;;     izquierda
;;; ENTONCES
;;;   almacenamos su camino como solución.

(defrule SOLUCION::reconoce-solucion
  (declare (auto-focus TRUE))
  ?nodo <- (nodo (estado V V V H B B B)
                 (camino $?estados))
  =>
  (retract ?nodo)
  (assert (solucion $?estados)))
```

El problema de las fichas

- **Módulo SOLUCION**

```
;;; REGLA: escribe-solucion
;;; SI
;;;   hemos obtenido una solución al problema
;;; ENTONCES
;;;   escribimos la solución obtenida en la pantalla.

(defrule SOLUCION::escribe-solucion
  (solucion $?estados)
  =>
  (bind ?longitud (length $?estados))
  (printout t "La solucion, de longitud " ?longitud
              " es " crlf)
  (loop-for-count (?i 1 ?longitud)
    (bind ?estado (nth ?i $?estados))
    (printout t ?estado crlf))
  (retract *))
```

El problema de las fichas

- **Sesión con estadística**

```
CLIPS> (load "fichas-1.clp")
+%$*******
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 83 es:
B B B H V V V
H B B B V V V
B H B B V V V
B B H B V V V
B B V B H V V
.............
V B V V H B B
V H V V B B B
H V V V B B B
V V H V B B B
V V V H B B B
238 rules fired        Run time is 34.50 seconds.
6.898550724637682 rules per second.
43 mean number of facts (84 maximum).
1 mean number of instances (1 maximum).
103 mean number of activations (205 maximum).
```

Fichas con heurística

- **Heurística:**
  - **Definición: La heurística de un estado es la suma de piezas blancas situadas a la izquierda de cada una de las piezas verdes.**
  - **Ejemplo: La heurística del siguiente estado es 1+2+2 = 5.**

B	**V**	**B**		**V**	**V**	**B**

- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot clase (default abierto)))

(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))
```

Fichas con heurística

- **Módulo MAIN**

```
(deffunction MAIN::heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes))
        else (if (eq ?ficha V)
                 then (bind ?verdes
                            (- ?verdes 1)))))
  ?resultado)

;;; REGLA: inicial
;;;   creamos el nodo inicial cuyo estado es el
;;;     estado inicial, su camino es la lista vacía,
;;;     su heurística es la del estado inicial y
;;;     pertenece a la clase cerrados.

(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica
                      (heuristica ?*estado-inicial*))
                (clase cerrado))))
```

Fichas con heurística

- **Módulo MAIN**

```
;;; REGLA: movimiento-izquierda
;;; SI
;;;   en un nodo de la clase cerrados, una ficha tiene
;;;     el hueco a su izquierda, a una distancia de,
;;;     como máximo, dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo de la clase abiertos en
;;;     el que movemos la ficha al hueco.

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x
                H
                $?y&:(<= (length $?y) 2)
                ?ficha
                $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Fichas con heurística

- **Módulo MAIN**

```
;;; REGLA: movimiento-derecha
;;; SI
;;;   en un nodo de la clase cerrados, una ficha tiene
;;;     el hueco a su derecha, a una distancia de,
;;;     como máximo, dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo de la clase abiertos en
;;;     el que movemos la ficha al hueco.

(defrule MAIN::movimiento-derecha
  (nodo (estado $?x
                ?ficha
                $?y&:(<= (length $?y) 2)
                H
                $?z)
        (camino $?movimientos)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Fichas con heurística

- **Módulo MAIN**

```
;;; REGLA: pasa-el-mejor-a-cerrado
;;; SI
;;;   no quedan nodos de la clase cerrados por
;;;     desarrollar y
;;;   tenemos un nodo de la clase abiertos con
;;;     valor heurístico mínimo
;;; ENTONCES
;;;   pasamos dicho nodo a la clase cerrados.

(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  =>
  (modify ?nodo (clase cerrado)))
```

- **Módulo RESTRICCIONES: Ningún cambio**
- **Módulo SOLUCION: Ningún cambio**

Fichas con heurística

- **Sesión con estadística**

```
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 15 es:
B B B H V V V
B B B V V V H
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
B V V V B B H
B V V V B H B
B V V V H B B
B V V H V B B
H V V B V B B
V V H B V B B
V V V B H B B
V V V H B B B
93 rules fired        Run time is 1.83 seconds.
50.7272727270043 rules per second.
26 mean number of facts (49 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (12 maximum).
```

Fichas con heurística y coste

- **Coste de un nodo = número de movimientos**
- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate nodo))

(deftemplate MAIN::nodo
  (multislot estado)
  (multislot camino)
  (slot heuristica)
  (slot coste)
  (slot clase (default abierto)))

(defglobal MAIN
  ?*estado-inicial* = (create$ B B B H V V V))

(deffunction MAIN::heuristica ($?estado)
  (bind ?resultado 0)
  (bind ?verdes 3)
  (loop-for-count (?i 1 7)
    (bind ?ficha (nth ?i $?estado))
    (if (eq ?ficha B)
        then (bind ?resultado
                   (+ ?resultado ?verdes))
        else (if (eq ?ficha V)
                 then (bind ?verdes
                            (- ?verdes 1)))))
  ?resultado)
```

Fichas con heurística y coste

- **Módulo MAIN**

```
;;; REGLA: inicial
;;;   creamos el nodo inicial cuyo estado es el
;;;     estado inicial, su camino es la lista vacía,
;;;     su heurística es la del estado inicial, su
;;;     coste es 0 y pertenece a la clase cerrados.

(defrule MAIN::inicial
  =>
  (assert (nodo (estado ?*estado-inicial*)
                (camino (implode$ ?*estado-inicial*))
                (heuristica
                      (heuristica ?*estado-inicial*))
                (coste 0)
                (clase cerrado))))
```

Fichas con heurística y coste

- **Módulo MAIN**

```
;;; REGLA: movimiento-izquierda
;;; SI
;;;   en un nodo de la clase cerrados, una ficha tiene
;;;     el hueco a su izquierda, a una distancia de,
;;;     como máximo, dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo de la clase abiertos en
;;;     el que movemos la ficha al hueco.

(defrule MAIN::movimiento-izquierda
  (nodo (estado $?x
                H
                $?y&:(<= (length $?y) 2)
                ?ficha
                $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x ?ficha $?y H $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (coste (+ ?coste 1))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Fichas con heurística y coste

- **Módulo MAIN**

```
;;; REGLA: movimiento-derecha
;;; SI
;;;   en un nodo de la clase cerrados, una ficha tiene
;;;     el hueco a su derecha, a una distancia de,
;;;     como máximo, dos fichas
;;; ENTONCES
;;;   creamos un nuevo nodo de la clase abiertos en
;;;     el que movemos la ficha al hueco.

(defrule MAIN::movimiento-derecha
  (nodo (estado $?x
                ?ficha
                $?y&:(<= (length $?y) 2)
                H
                $?z)
        (camino $?movimientos)
        (coste ?coste)
        (clase cerrado))
  =>
  (bind $?nuevo-estado
        (create$ $?x H $?y ?ficha $?z))
  (assert (nodo (estado $?nuevo-estado)
                (camino $?movimientos
                        (implode$ $?nuevo-estado))
                (coste (+ ?coste 1))
                (heuristica
                      (heuristica $?nuevo-estado)))))
```

Fichas con heurística y coste

- **Módulo MAIN**

```
;;; REGLA: pasa-el-mejor-a-cerrado
;;; SI
;;;   no quedan nodos de la clase cerrados por
;;;     desarrollar y
;;;   tenemos un nodo de la clase abiertos con
;;;     valor heurístico mínimo y
;;;   de entre todos los nodos con el mismo
;;;     valor heurístico, tenemos el de coste
;;;     mínimo
;;; ENTONCES
;;;   pasamos dicho nodo a la clase cerrados.

(defrule MAIN::pasa-el-mejor-a-cerrado
  (declare (salience -10))
  ?nodo <- (nodo (clase abierto)
                 (heuristica ?h1)
                 (coste ?c1))
  (not (nodo (clase abierto)
             (heuristica ?h2&:(< ?h2 ?h1))))
  (not (nodo (clase abierto) (heuristica ?h1)
             (coste ?c2&:(< ?c2 ?c1))))
  =>
  (modify ?nodo (clase cerrado)))
```

Fichas con heurística y coste

- **Módulo RESTRICCIONES**

```
(defmodule RESTRICCIONES
  (import MAIN deftemplate nodo))

;;; REGLA: repeticion-de-nodo
;;; SI
;;;   hay un nodo repetido
;;; ENTONCES
;;;   eliminamos el de mayor coste.

(defrule RESTRICCIONES::repeticion-de-nodo
  (declare (auto-focus TRUE))
  (nodo (estado $?actual)
        (coste ?coste-1))
  ?nodo <- (nodo (estado $?actual)
                 (coste ?coste-2&:(> ?coste-2 ?coste-1)))
  =>
  (retract ?nodo))
```

- **Módulo SOLUCION: Ningún cambio**

Fichas con heurística y coste

- **Sesión con estadística**

```
CLIPS> (load "fichas-3.clp")
+%:!*********
TRUE
CLIPS> (watch statistics)
CLIPS> (reset)
CLIPS> (run)
La solucion, de longitud 13 es:
B B B H V V V
B B B V V H V
B B H V V B V
B B V V H B V
B H V V B B V
B V V H B B V
H V V B B B V
V V H B B B V
V V B H B B V
V V B V B B H
V V B V B H B
V V H V B B B
V V V H B B B
120 rules fired        Run time is 0.65 seconds.
184.6153846170378 rules per second.
25 mean number of facts (47 maximum).
1 mean number of instances (1 maximum).
5 mean number of activations (11 maximum).
```

Comparación de soluciones

	Solución 1	Solución 2 (heurística)	Solución 3 (heurística y coste)
Longitud de la solución	83	15	13
Tiempo (segundos)	34.50	1.83	0.65
Número de disparos	238	93	120
Número máximo de hechos	84	49	47
Número medio de hechos	43	26	25
Número máximo de activaciones	205	12	11
Número medio de activaciones	103	5	5

Problema del 8-puzzle

- **Enunciado**

2	**8**	**3**
1	**6**	**4**
7		**5**

Estado inicial

1	**2**	**3**
8		**4**
7	**6**	**5**

Estado final

- **Heurística**
  - **Definición: número de piezas descolocadas**
  - **Heurística del estado inicial: 4**