Dpto. Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMES INTEL·LIGENTS

JOC DEL 8-PUZLE

DISSENY, IMPLEMENTACIÓ I AVALUACIÓ D'UN SISTEMA BASAT EN REGLES

Índex

1. Implementació del joc del 8-puzle

En aquest apartat es presenta la implementació del joc del 8-puzle com un SBR. Els elements que intervenen en el problema són:

- **Base de Fets inicial**: situació inicial de partida del puzle
- **Estat Objectiu**: situació final del puzle a la qual es vol arribar
- **Regles**: conjunt de possibles accions a realitzar (moviments del quadre buit)
- **Estratègia de control**: cerca en amplària, cerca en profunditat i cerca heurística (es detallarà en el següent apartat)

1.1 Patrons

Per a la representació del puzle s'ha escollit un patró ordenat on representem els valors de les 9 caselles ordenades per files. Un fet associat a aquest patró representarà un estat concret del problema o configuració del 8-puzle.

Per exemple, la següent configuració o estat del problema:

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

es representarà com:

```
(puzle 2 8 3 1 6 4 7 0 5)
```

Els tres primers valors corresponen a les caselles de la fila 1, els següents tres valors als de la fila 2 i els últims tres valors als de la fila 3.

S'utilitzarà, addicionalment, un conjunt de camps o etiquetes addicionals:

- una etiqueta que indica el ***nivell de profunditat*** del fet associat. Aquesta etiqueta s'emprarà per a dos objectius diferents:
    - d'una banda, per a controlar que no s'expandeixen nodes que superen el límit de profunditat prefixat per l'usuari.
    - d'altra banda, per a retornar el límit de profunditat on la solució ha sigut trobada.

- una etiqueta per a indicar quin ha sigut l'**últim *moviment*** que ha originat tal estat. Aquest camp s'emprará per a evitar els cicles directes en els moviments del puzle.

- una etiqueta per a indicar el ***fet sobre el qual es va aplicar l'últim moviment*** per a generar l'estat actual. Aquest camp s'emprarà per a recuperar el camí seguit per a arribar fins a la solució.

Per tant, un exemple de fet inicial seria:

```
(puzle 2 8 3 1 6 4 7 0 5 nivell 0 moviment nul fet 0)
```

I un exemple de fet final seria:

```
(puzle 1 2 3 8 0 4 7 6 5 nivell 5 moviment dreta fet <Fact-28>)
```

El camp **nivell** és un element constant i l'element que apareix a continuació indica el nivell d'aquest node en l'arbre de cerca.

El camp **moviment** és un element constant i l'element que apareix a continuació guarda el moviment que s'ha realitzat (esquerra, dreta, dalt, baix) per a arribar a aquest estat.

El camp **fet** és un element constant i l'element que apareix a continuació indica l'índex del fet sobre el qual es va aplicar l'últim moviment per a generar aquest estat.

Per tant, l'estructura del nostre patró seria:

(puzle $x1^s$ $x2^s$ $x3^s$ $x4^s$ $x5^s$ $x6^s$ $x7^s$ $x8^s$ $x9^s$ nivell y^s moviment z^s fet w^s)

on:

- **puzle**, **nivell, moviment** i **fet** són quatre camps constants
- els camps que estan etiquetats amb un superíndex ‘s’ (single-valued) indiquen que és un camp mono-valuat; és a dir, un element que ha de ser necessàriament un únic valor. Concretament:
  - $xi^s \in [0\text{-}8] / \forall i,j \ \ xi \neq xj$
  - $y^s \in$ INTEGER
  - $z^s \in$ [dalt, baix, dreta, esquerra]
  - $w^s \in$ fact-index

1.2 Regles

Les regles representen els moviments que es poden realitzar en el puzle. Com tot moviment involucra una fitxa del puzle i la casella en blanc, les regles del problema es plantegen com a moviments de la casella en blanc. Per tant, implementarem els quatre moviments que es poden realitzar amb la casella que conté l'espai en blanc, açò és, **dreta, esquerra, dalt** i **baix**.

Des d'aquest punt de vista, es podrá realitzar un moviment a la **dreta** sempre que l'espai en blanc es trobe en alguna de les posicions indicades per un punt negre en la següent figura:

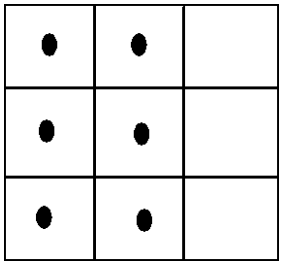

Fig. 1 Posicions des de les quals es pot realitzar un moviment a la dreta

Per tant, tenim que cadascun dels quatre moviments pot realitzar-se a partir de 6 posicions diferents.

Les regles implementen els quatre possibles moviments de l'espai en blanc dins del tauler. Cadascuna de les regles instància en la seua part esquerra la configuració necessària d'un estat per a poder moure l'espai en blanc en el sentit indicat per la regla. Per exemple, la regla `dreta` implementa qualsevol possible moviment del quadre blanc cap a una posició que estiga a la seua dreta. La regla `dreta` té el següent format:

```
(defrule dreta
  ?f<-(puzle $?x 0 ?y $?z nivell ?nivell moviment ?mov fet ?)
  (profunditat-maxima ?prof)
  (test (and (<> (length$ $?x) 2) (<> (length$ $?x) 5)))
  (test (neq ?mov esquerra))
  (test (< ?nivell ?prof))
=>
  (assert (puzle $?x ?y 0 $?z nivell (+ ?nivell 1) moviment dreta fet ?f))
  (bind ?*nod-gen* (+ ?*nod-gen* 1)))
```

Fig. 2 Regla `dreta`

La part esquerra de la regla es compon de dos patrons i tres tests:

1. el primer patró instància tots els fets de la Base de Fets corresponents a les diferents configuracions dels estats del puzle, localitzant la casella en blanc en l'element 0 i la casella que està a la seua dreta en la variable `?y`.

2. el segon patró instància un fet estàtic que indica la profunditat màxima que s'ha d'aconseguir en la cerca.

3. el primer test comprova si l'espai en blanc en el patró es troba en alguna de les 6 posicions requerides per a moure el quadre blanc a la dreta. Per a açò es comprova la longitud de la variable `$?x` que guarda tots els elements a l'esquerra de 0. Per exemple, si la casella en blanc es troba en la posició [1,1] (casella superior esquerra) llavors la longitud de la variable `$?x` serà 0. En aquest cas, solament comprovem que la longitud de la variable `$?x` no siga 2 ni 5, en aquest cas la fitxa en blanc estaria situada en les posicions [1,3] i [2,3], respectivament. No és necessari comprovar que la longitud de `$?x` siga diferent de 8 (en aquest cas la casella blanca estaria en la posició [3,3]) perquè açò no és possible a tenor de les variables del patró.

4. el següent test comprova que la configuració o estat que instància el primer patró no s'haja generat per un moviment a esquerres (per a evitar cicles).

5. l'últim test s'encarrega de filtrar només aquells fets en quins el nivell de profunditat no excedeix l'establit per l'usuari.

En la part dreta de la regla s'insereix el nou estat (fet) que resulta com a conseqüència de moure l'espai en blanc des de la posició en la qual aquest es troba cap a la dreta. Per tant, el nou fet intercanvia les posicions de la casella en blanc (0) amb la casella que està a la seua dreta (`?y`).

Per a les regles `dalt` i `baix` cal tenir en compte que es tracten de moviments verticals en una estructura lineal pel que cal localitzar la fitxa amb la qual s'intercanvia la casella en blanc.

El codi de la regla baix seria:

```
(defrule baix
  ?f<-(puzle $?x 0 ?a ?b ?c $?z nivell ?nivell moviment ?mov fet ?)
 (profunditat-maxima ?prof)
 (test (neq ?mov dalt))
 (test (< ?nivell ?prof))
=>
 (assert (puzle $?x ?c ?a ?b 0 $?z nivell (+ ?nivell 1) moviment baix fet
?f))
 (bind ?*nod-gen* (+ ?*nod-gen* 1)))
```

Fig. 3 Regla `baix`

La casella blanca s'intercanvia sempre amb la fitxa que ocupa la posición de la variable `?c`. Es pot observar que en posar tres variables mono-valuades (`?a ?b ?c`) després del 0, estem exigint que hi haja necessàriament tres elements, la qual cosa evitaria els moviments cap avall que no són possibles (quan l'espai blanc està en l'última fila).

1.3 Dades inicials, objectiu i solució

L'estat **objectiu** amb el qual es treballarà en el problema del puzle és el següent:

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

L'estat objectiu es representa directament amb la regla `objectiu` que es mostra en la figura 4.

```
(defrule objectiu
     (declare (salience 100))
   ?f <-(puzle 1 2 3 8 0 4 7 6 5 nivell ?n moviment ?mov fet ?)
  =>
   (printout t "SOLUCIÓ TROBADA EN EL NIVELL " ?n crlf)
   (printout t "NOMBRE DE NODES EXPANDITS O REGLES DISPARADES " ?*nod-gen*
crlf)
   (printout t "FET OBJECTIU " ?f crlf)
   (halt))
```

Fig. 4 Regla `objectiu`

Aquesta regla té una prioritat màxima (`salience 100`) atès que és la regla que detecta que s'ha arribat a l'estat objectiu. En el moment que es produïsca un fet que unifique amb la configuració objectiu, la regla objectiu es dispararà automàticament concloent així el procés inferencial. El comando (`halt`) deté el Motor d'Inferència.

També es defineix una regla amb una prioritat inferior a la resta de regles per a detectar el cas en el qual no es troba solució al problema.

```
(defrule no_solucio
    (declare (salience -99))
    (puzle $? nivell ?n $?)
=>
    (printout t "SOLUCIÓ NO TROBADA" crlf)
    (printout t "NODES GENERATS: " ?*nod-gen* crlf)
    (halt))
```

Fig. 5 Regla `no_solucio`

Addicionalment, tenim una funció `inici` encarregada de preguntar per la profunditat màxima amb la qual es desitja treballar i el tipus d'estratègia a emprar (de moment, amplària i profunditat). Depenent de la resposta, s'activa l'estratègia de l'agenda corresponent, ja que la estratègia de control (amplària/profunditat) la implementa la pròpia agenda de CLIPS. A més, s'insereix en la base de fets l'estat inicial i un fet estàtic que indica el màxim nivell de profunditat a aconseguir en la cerca.

```
(deffunction inici ()
 (reset)
 (printout t "Profunditat Maxima:= " )
 (bind ?prof (read))
 (printout t "Tipus de Cerca " crlf "1.- Amplària" crlf
             "2.- Profunditat" crlf )
 (bind ?a (read))
 (if (= ?a 1)
       then (set-strategy breadth)
       else (set-strategy depth)))
  (assert (puzle 2 8 3 1 6 4 7 0 5 nivell 0 moviment nul fet 0))
  (assert (profunditat-maxima ?prof))
)
```

Fig. 6 Funció `inici`

Finalment, es disposa de la funció `cami`, que mostra per pantalla la seqüència de moviments aplicats (i sobre quin fet) per a aconseguir la solució. S'invoca de la següent forma: `(cami <fact-index>).`

Per exemple, si després de l'execució d'una configuració del puzle, obtenim una eixida per pantalla del següent tipus (veure regla objectiu en Fig. 4):

```
SOLUCIÓ TROBADA EN EL NIVELL 5
NOMBRE DE NODES EXPANDITS O REGLES DISPARADES 36
FET OBJECTIU <Fact-39>
```

per a recuperar la solució de 5 passos de la configuració provada, devem simplement executar `(cami 39).`

1.4 Configuracions resolubles

En el problema del puzle, pot donar-se el cas de que no existisca solució per a una configuració determinada. Per exemple, donat l'estat inicial:

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

no existeix combinació de moviments dreta-esquerra-dalt-baix que aconseguisca arribar a l'estat final descrit en la secció anterior. En aquest cas, es diu que la configuració és irresoluble.

Amb l'objecte de comprovar si una configuració particular és resoluble o no (és a dir, si és possible o no trobar un camí que conduïsca a l'estat objectiu), es proporciona la funció `comprovar_conf` en el fitxer `auxiliar.clp`. Aquesta funció rep com a argument una llista que representa la configuració del puzle que es desitja comprovar i retorna TRUE si la configuració és resoluble i FALSE en cas contrari.

Per exemple:

```
CLIPS> (comprovar_conf (create$ 2 8 3 1 6 4 7 0 5))
TRUE

CLIPS> (comprovar_conf (create$ 2 1 3 8 0 4 7 6 5))
FALSE
```

El fitxer `auxiliar.clp` ha de carregar-se en CLIPS sempre que es vulga realitzar la comprovació d'una determinada configuració del puzle.

2. Estratègies de control *amplària* i *profunditat*

A través de les estratègies de control amplària i profunditat, implementem una cerca en amplària/profunditat en arbre. En aquest punt, convé esmentar un aspecte sobre el control de nodes repetits en CLIPS:

1. En termes generals, CLIPS realitza un control automàtic de nodes repetits perquè, per defecte, no permet duplicitat de fets. Açò significa que si s'intenta inserir un node idèntic a un altre que ja existeix en la BF, CLIPS no ho inserirà.
2. No obstant açò, en el nostre model utilitzem un camp **nivell** que és la profunditat d'un node. Açò implica que un node de nivell 3 que represente el mateix estat que un node de nivell 5 no estaran representats pels mateixos fets i, per tant, CLIPS inserirà tots dos nodes en la BF. A més, s'inclou el camp **fet** (per a la recuperació del camí), la qual cosa suposa un factor totalment distintiu entre dues configuracions idèntiques del puzle.
3. En termes pràctics, per l'exposat en 2, no existeix control de nodes repetits en CLIPS, excepte el control de cicles directes que es realitza a través del camp **moviment**.
4. No obstant açò, si no es mantinguera el **nivell** ni el **moviment** del node ni el **fet pare**, llavors CLIPS realitzaria un control automàtic dels nodes repetits. Però, en dit cas, no podríem conèixer el nivell de profunditat de la solució ni recuperar el camí per a aconseguir la solució.

A continuació es mostra un parell d'iteracions del problema del puzle amb el funcionament de l'agenda en Amplària i la seua equivalència en un procés de cerca en arbre. Inicialment, la situació és la que es mostra en la següent pantalla, que es correspon amb l'estat del procés de cerca de la figura següent.

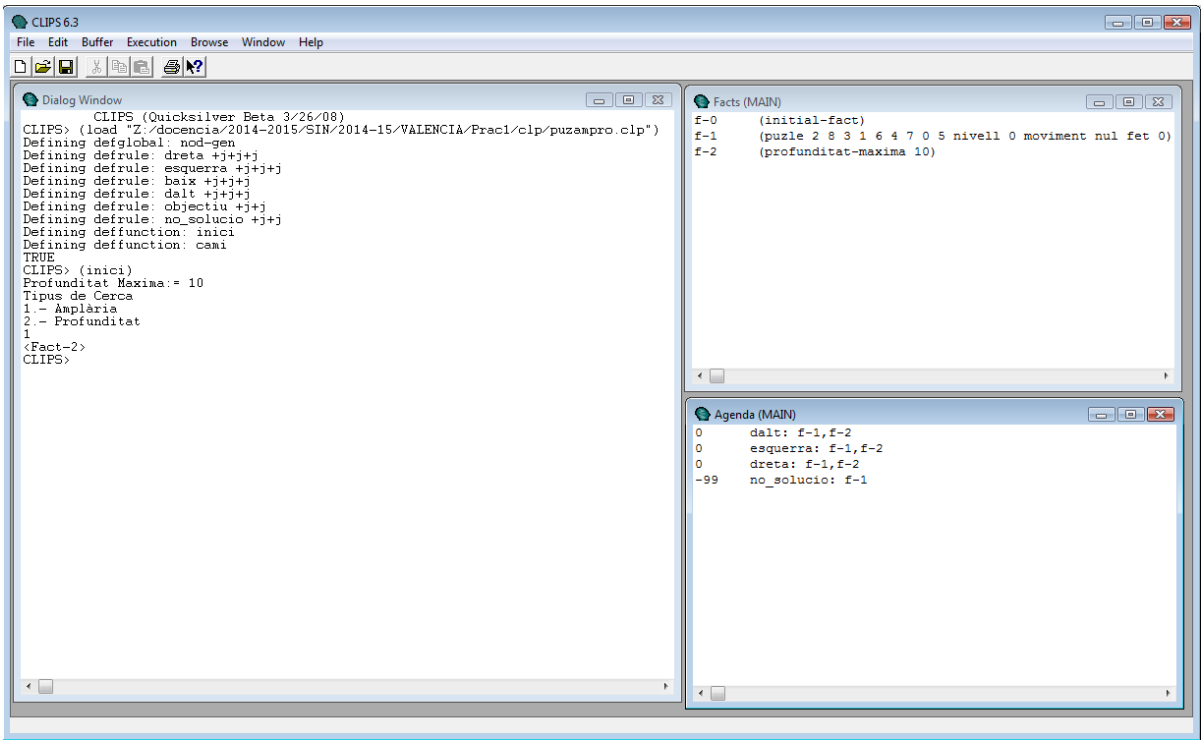

Fig. 7 Situació inicial de l'Agenda i Base de Fets

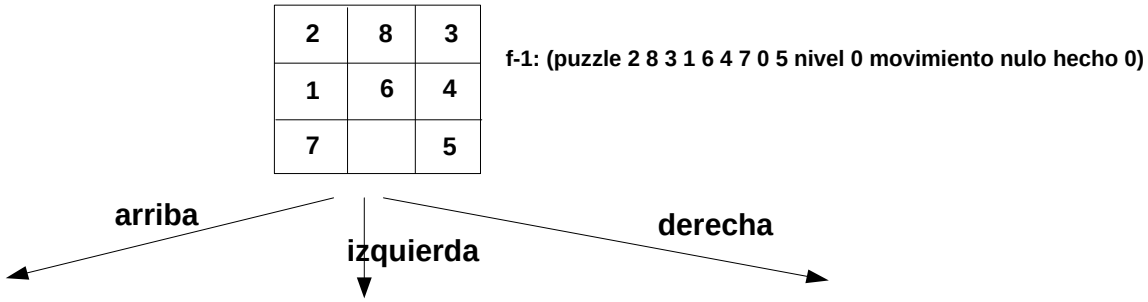

Fig. 8 Estat del procés de cerca corresponent a la Fig. 7

Les tres activacions de l'Agenda representen els tres possibles moviments que es poden aplicar en la configuració inicial. A diferència d'un procés de cerca clàssic, els nodes successors encara no s'han generat (solament està el fet `f-1` en la Base de Fets).

Si executem un pas del procés inferencial, obtenim:

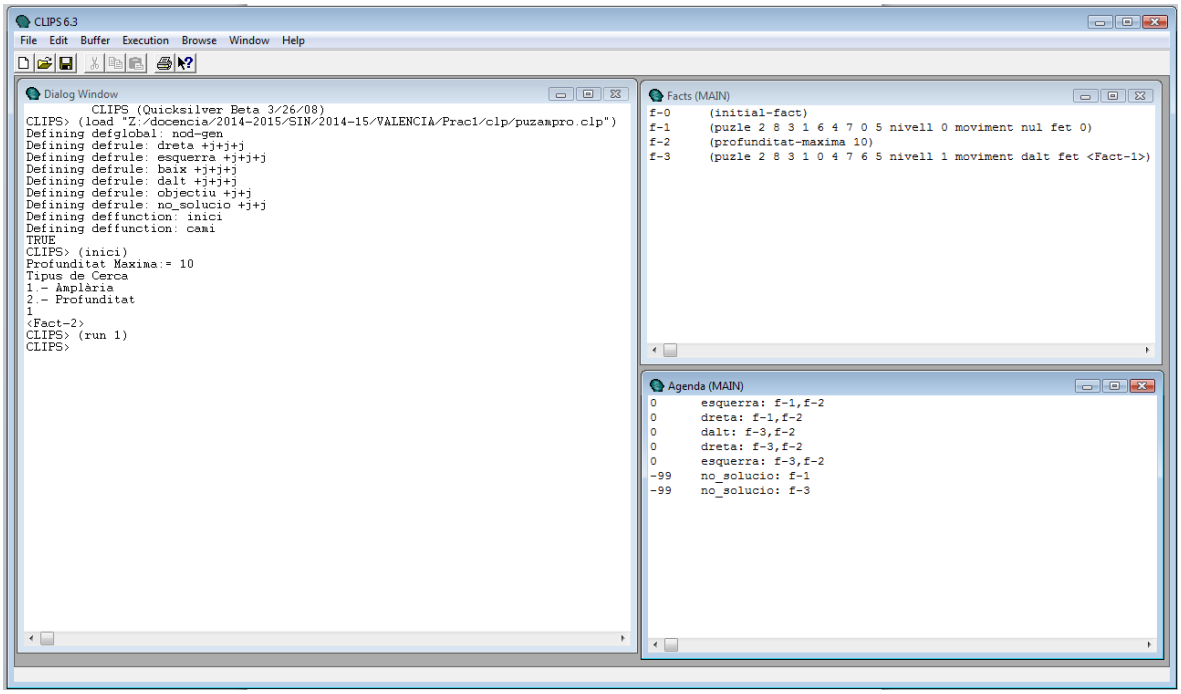

Fig. 9 Situació de l'Agenda i BF després d'un cicle inferencial

que es correspon amb:

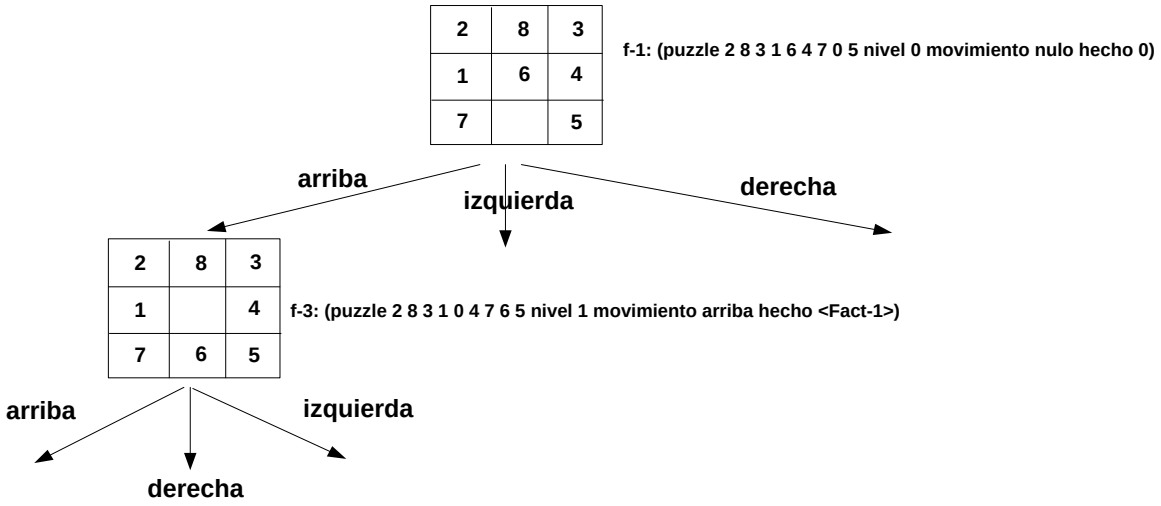

Fig. 10 Estat del procés de cerca corresponent a la Fig. 9

A continuació es dispararia la regla `esquerra` del node arrel, i després la regla `dreta`, i després la regla a `dalt` del node representat amb el fet `f-3`, i així successivament, implementant un procés de cerca en amplària. Quan finalitza el procés de cerca, obtindríem una situació com a la que es mostra en la figura 11.

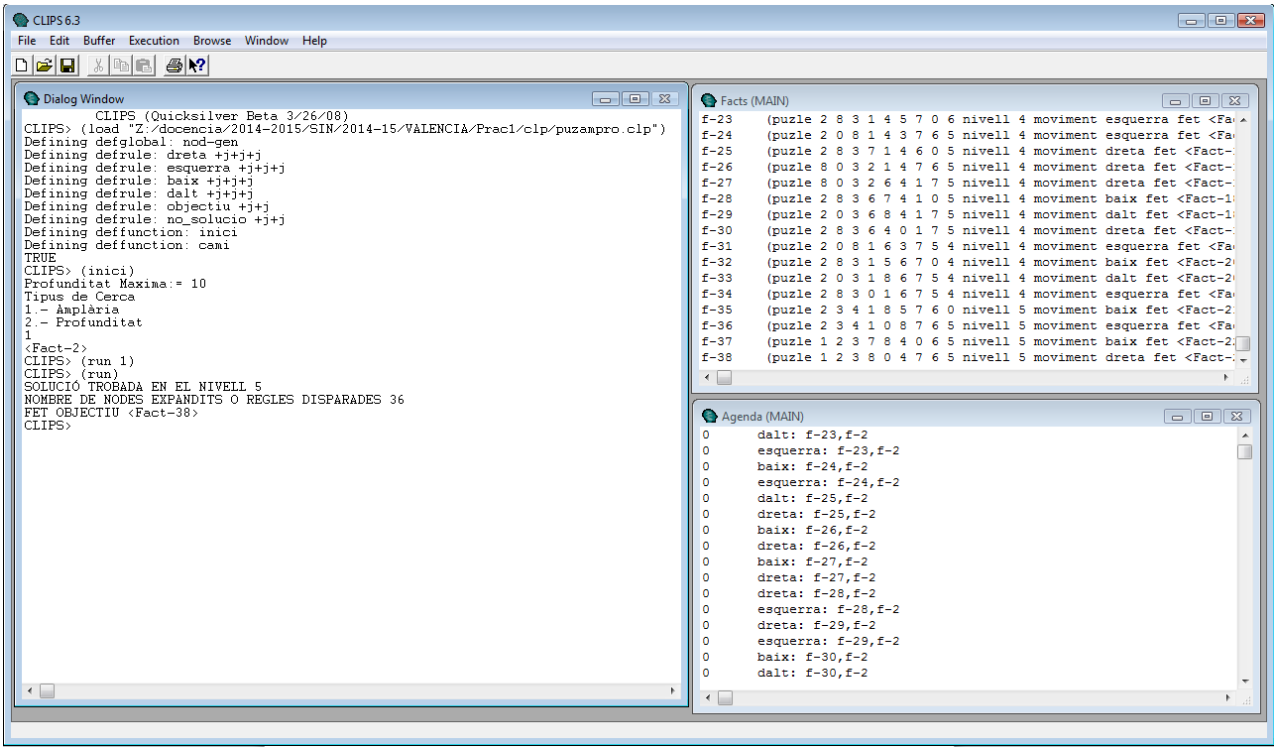

Fig. 11 Situació de l'Agenda i BF en finalitzar l'ejecució

En aquest cas, si utilitzem la funció `(cami 38)`, on 38 és el nombre del fet objectiu, obtindríem aquesta eixida:

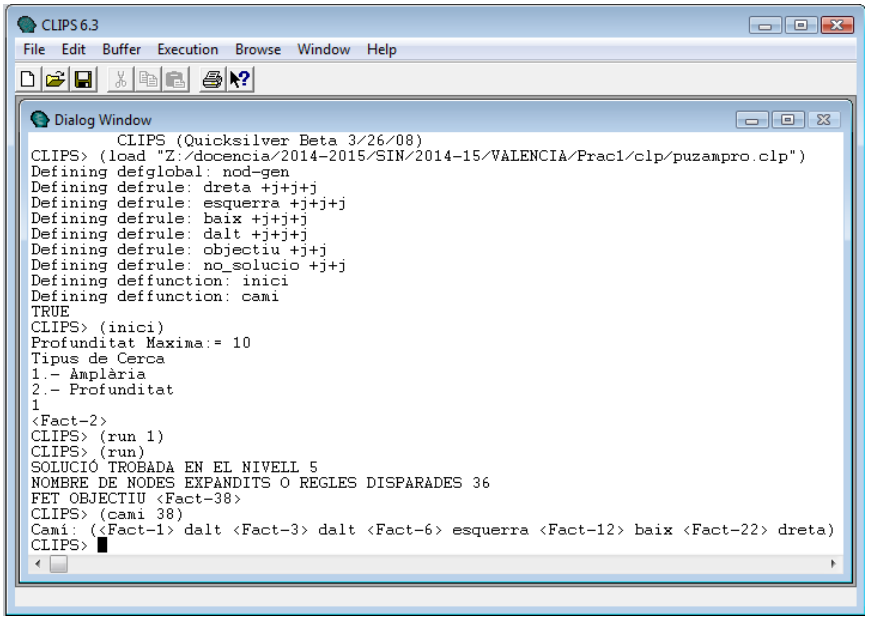

Fig. 12 Obtenció del camí solució per al puzle de la figura 8

3. Cerca heurística

La clau de l'estratègia heurística és seleccionar la regla més 'prometedora', és a dir aquella que genera el 'millor' node d'acord al valor d'una funció heurística.

La idea consisteix per tant a disparar o executar sempre primer la regla que ofereix el millor valor de la funció heurística, màxim o mínim segons es tracte de maximitzar o minimitzar la funció. Per a açò les regles activades han d'introduir-se en l'agenda ordenades d'acord al valor de la funció heurística.

La funció heurística que es presenta com a exemple es correspon amb:

$$f(n)= g(n)+h(n) \implies f(n)=p(n)+W(n)$$

on:

p(n): indica el nivell de profunditat del fet corresponent al node o configuració del puzle que s'està estudiant.

W(n): nombre de fitxes descol·locades en el fet corresponent al node o configuració del puzle que s'està estudiant.

La idea consisteix per tant a minimitzar el valor d'aquesta funció i ordenar les regles en l'agenda pel menor valor de la mateixa. Abans d'entrar en els detalls d'implementació, vegem quins resultats obtindríem en aplicar la cerca heurística a l'exemple de la figura 8.

Si apliquem la funció heurística a l'estat inicial de la figura 8 obtindrem un valor de 4 que correspon a p(n)=0 i W(n)=4.

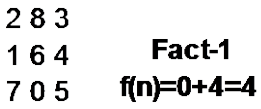

Fig. 13 Valor heurístic per a l'estat inicial

Existeixen tres regles possibles que es poden activar per a l'estat inicial (dalt, dreta, esquerra). Si calculem prèviament el valor heurístic de l'estat al que ens conduiria cada regla podem llavors assignar a cada regla una ***prioritat equivalent al valor obtingut en la funció heurística***. Com ens interessen els valors mínims d'aquesta funció i les regles s'introdueixen en l'agenda per ordre de màxima prioritat, convertim el valor obtingut en un nombre negatiu per a així inserir les regles en l'agenda ordenades pel menor valor de la funció.

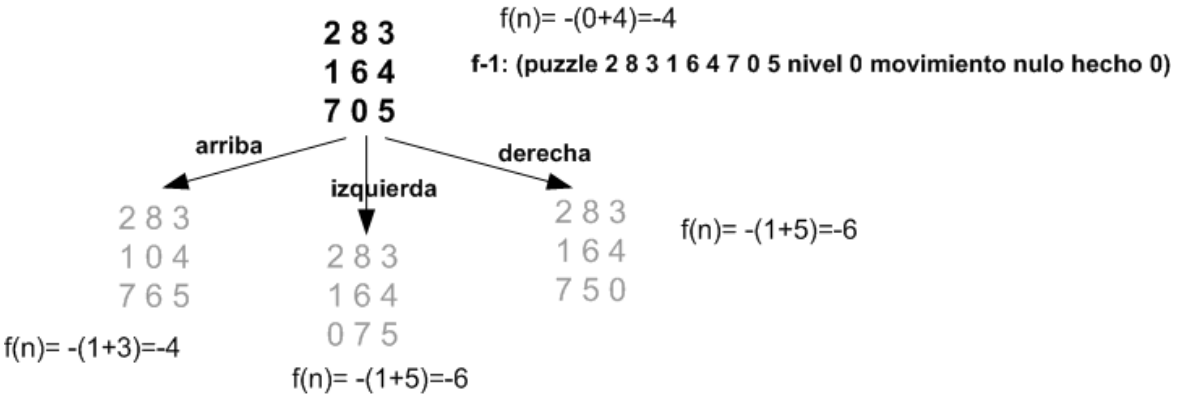

Fig. 14 Càlcul de la prioritat de cada regla abans de ser introduïda en l'agenda

Note's que els estats en gris no són generats. L'únic fet del que disposem és `f-1` i a partir d'aquest es calcula el valor de l'heurística per a cada regla. Els valors obtinguts de la funció heurística determinen automàticament l'ordre en el qual s'insereixen les regles en l'agenda (`dalt, esquerra, dreta`) i conseqüentment quin serà la primera regla a disparar (`dalt`).

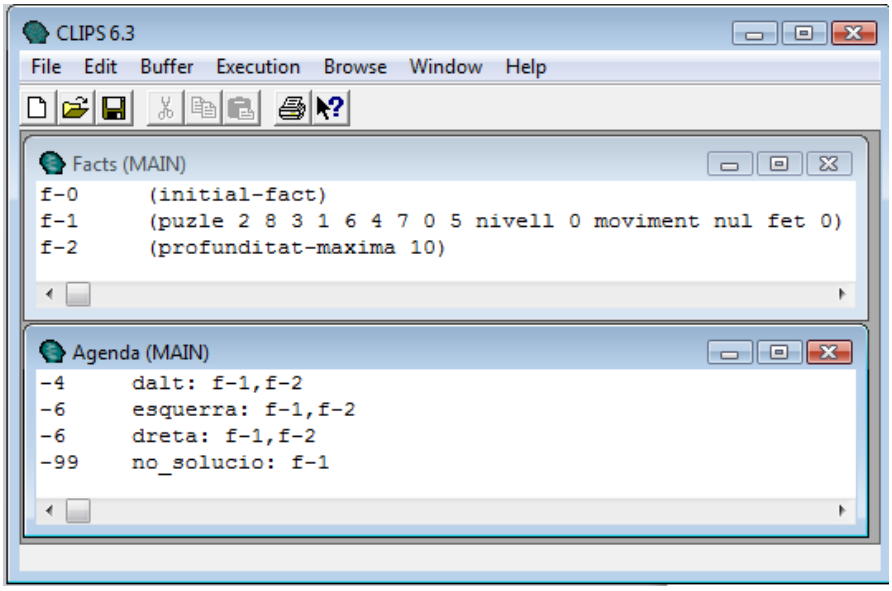

Fig. 15 Ordenació de les regles activades segons prioritat

Després de l'aplicació de la regla `dalt`, la situació és com es mostra a continuació:

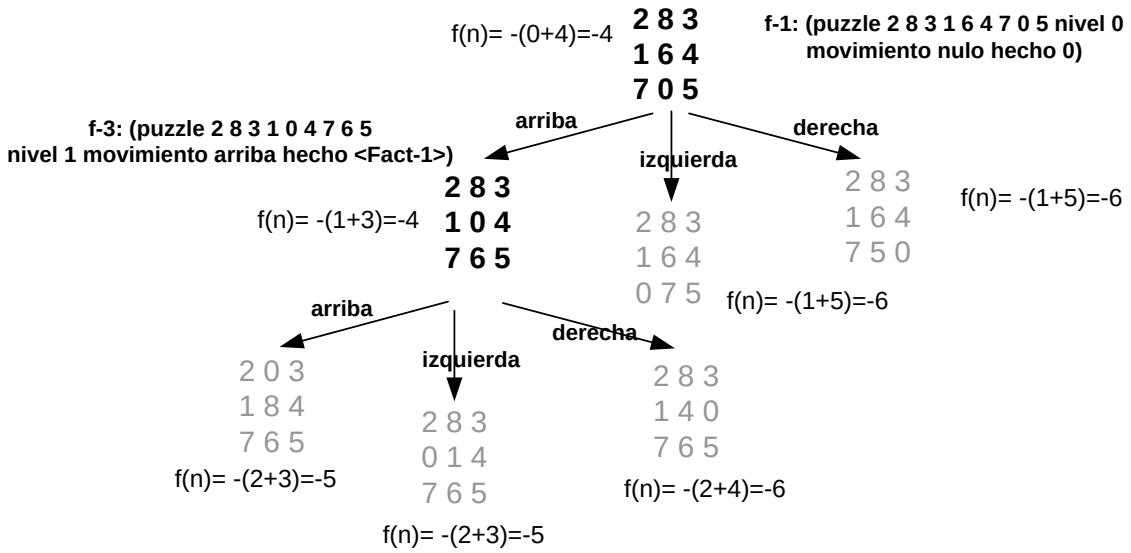

Fig. 16 Situació tras disparar-se la regla `dalt`

Es calcula el valor de la funció heurística en aquells nodes que resultarien en aplicar les regles que s'activen amb el fet `f-3`. Dos de les regles ens retorna el valor -5 i l'altra el valor -6. Disposem ara de 5 regles aplicables o activades en l'Agenda (nodes fulles de l'arbre), tres de les quals tenen associat un valor -6 i dos un valor -5. Entre aquelles regles amb la mateixa prioritat, la que s'insereix primer depèn de l'estratègia amb la qual estiga funcionant l'agenda (DEPTH o BREADTH). Recordem que l'estratègia heurística anul·la per complet l'estratègia per defecte de l'agenda a l'hora de decidir què regla ha de disparar-se ja que aquestes s'ordenen per ordre de prioritat; però l'estratègia de l'agenda segueix vigent a l'hora de determinar per exemple què regla introduir primer entre diverses amb la mateixa prioritat.

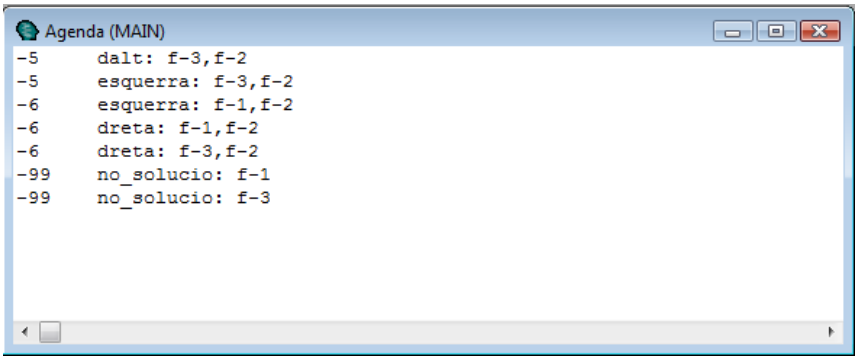

Fig. 17 Resultat de l'agenda després de disparar-se de la regla `dalt`

En la figura 17 es pot observar que totes les regles s'instancien amb dos fets, i que totes elles involucren el fet `f-2`. En aquest punt cal recordar que el fet `f-2` és el que estableix el màxim nivell de profunditat. En aquest cas concret seria
`f-2: (profunditat-maxima 10)`, raó per la qual el fet corresponent al següent estat generat seria `f-3`.

Com es pot observar, la regla `dalt` amb el fet 3 (`f-3`) és la primera regla de l'agenda i per tant la que es dispararà a continuació (el funcionament de l'agenda és BREADTH i només té efecte entre regles amb la mateixa prioritat).

Implementació de l'estratègia de cerca heurística

Per a implementar la prioritat amb la qual s'han d'activar les regles en l'agenda es declara una prioritat dinàmica a les mateixes (`salience`). Aquesta prioritat es determina en el moment que la regla és susceptible de poder aplicar-se i apareix en la part esquerra de les regles. Exemple:

```
(defrule dreta
  (declare (salience (- 0 ?*f*)))
  ?f<-(puzle $?x 0 ?y $?z nivell ?nivell moviment ?mov fet ?)
  (profunditat-maxima ?prof)
  (test (and (<> (length$ $?x) 2) (<> (length$ $?x) 5)))
  (test (neq ?mov esquerra))
  (test (< ?nivell ?prof))
  (test (control (create$ $?x ¿y 0 $?z) ?nivell))

  =>
```

Fig. 18 LHS de la regla `dreta` en la cerca heurística

Per a calcular la prioritat de la regla es declara un `salience` dinàmic que, inicialment, pren el valor d'una variable global `?*f*` **`(declare (salience (- 0 ?*f*)))`**. Aquest valor s'actualitza al final de la LHS de la regla, concretament en l'últim `test` de la regla:

```
(test (control (create$ $?x ?y 0 $?z) ?nivell))
```

L'objectiu d'aquest `test` és invocar a una funció anomenada `control` que és l'encarregada de calcular el nombre de peces descol·locades que resultarien de moure l'espai blanc a la dreta (aquest seria el valor $h(n)$), i sumar el factor $g(n)$ del node corresponent `(+ ?nivell 1)`. Cal observar que la regla està calculant en la seua part esquerra el valor f(n) del node fill que es generarà en la part dreta de la mateixa; per aquesta raó:

1) el valor $h(n)$ és el nombre de peces descol·locades del node fill, és a dir del node que resulta després de realitzar el moviment corresponent (`dreta, esquerra, dalt, baix`). Per això, la llista que se li pasa a `control és la del node fill.`
2) el valor $g(n)$ és el nivell del node actual (node pare) més 1; és a dir, `(+ ?nivell 1)`.

Una vegada s'ha executat la funció `control`, el valor de la variable global `?*f*` s'actualitza amb el valor f(n) del node que es generarà quan s'execute la instància d'aquesta regla; i aquest valor serà la prioritat amb la qual la instància de la regla s'introduïsca en l'agenda (`salience`). El còdi de la funció `control` és el següent:

```
(deffunction control (?estat ?nivell)
     (bind ?*f* (descolocades ?estat))
     (bind ?*f* (+ ?*f* ?nivell 1)))
```

Fig. 19 Funció `control` per a calcular el valor del `salience` de les regles

La funció `control` crida a la funció `descolocades` que és l'encarregada de calcular el nombre de peces descol·locades.

Com la prioritat de les regles és dinàmica, aquesta ha de calcular-se en el moment que la regla s'activa. Per a açò, s'informa a l'agenda de tal fet en la funció `inici`:

```
(deffunction inici ()
   (set-salience-evaluation when-activated)
   (reset)
   (printout t "Profunditat Maxima:= " )
   (bind ?prof (read))
   (printout t " Executa run per a engegar el programa " crlf)
   (assert (puzle 2 8 3 1 6 4 7 0 5 nivell 0 moviment nul fet 0))
   (assert (profunditat-maxima ?prof)))
```

Fig. 20 Funció `inici` on es determina que el valor del `salience` s'ha de calcular cada vegada que una regla s'activa.

4. Execució del sistema basat en regles

Es proporciona un conjunt de fitxers:

- `puzampro.clp`, `puzdesco.clp`: corresponden a diferents implementacions del sistema de producció del puzle: amplària, profunditat i amb l'heurística descol·locades, respectivament.

- `auxiliar.clp`: conté la funció `comprovar_conf` per a comprovar si la configuració que es passa com a paràmetre és resoluble.

Per a realitzar una execució, es carregarà el fitxer corresponent al sistema basat en regles (`puzampro.clp`, `puzdesco.clp`), modificant la configuració inicial del problema en la funció `inici`. Prèviament, es pot comprovar mitjançant la funció del fitxer `auxiliar.clp` si la configuració del puzle a provar té solució.

5. Proves d'avaluació

Per a l'execució d'una configuració particular del puzle, modificarem l'estat inicial en la funció inici del SBR corresponent. Per exemple:

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

La situació inicial d'aquest puzle es correspon amb el següent fet:

```
(puzle 8 1 3 7 2 5 4 0 6 nivell 0 moviment nul fet 0))
```

Els resultats per a diferents execucions, tant en amplària, profunditat com amb cerca heurística i amb diferents nivells de profunditat, són els següents:

Estratègia	*Nivell màxim de profunditat*	*Nivell de la solució*	*Nombre de nodes expandits*
Amplària	10	9	442
Amplària	20	9	442
Profunditat	10	9	479
Profunditat	20	15	68318
Heurística descol·locades	10	9	19
Heurística descol·locades	20	9	19