

Presentazione progetti Prolog e Clingo

Giulio Taralli & Ismaila Toure

Progetti Prolog

Progetto 1: Labirinto

In questo esercizio veniva proposto di implementare due strategie di ricerca nello spazio degli stati sul dominio di un labirinto.

- Le strategie di ricerca sono: A^* e IDA^*
- Il labirinto doveva avere un punto di entrata e almeno due uscite

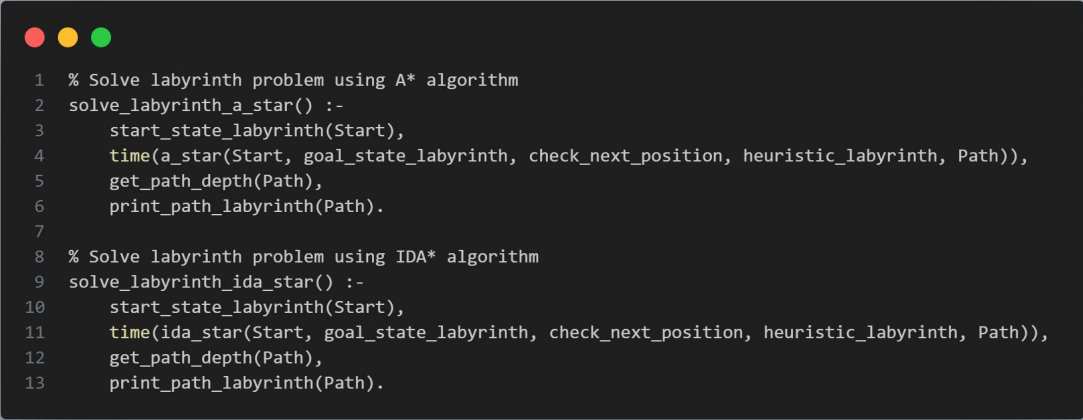
Per mantenere un codice pulito e modulare abbiamo deciso di suddividere il progetto in 3 file:

- **main.pl**: punto di ingresso per richiamare il predicato iniziale di avvio
- **activity1_labyrinth.pl**: contiene la KB e predicati univoci per labirinto, incluse le euristiche
- **search.pl**: contiene le implementazioni di A^* e IDA^*

main.pl

Questo file Prolog è il punto di ingresso per richiamare la risoluzione del problema labirinto (e non solo, sarà utilizzato anche per puzzle) e in base al predicato scelto si sceglierà anche il tipo di ricerca che si vorrà eseguire tra A* e IDA*.

- In tutti i predicati otteniamo il tempo di esecuzione, la profondità del percorso ottenuto e il percorso stesso dall'inizio alla fine



```
1 % Solve labyrinth problem using A* algorithm
2 solve_labyrinth_a_star() :-
3     start_state_labyrinth(Start),
4     time(a_star(Start, goal_state_labyrinth, check_next_position, heuristic_labyrinth, Path)),
5     get_path_depth(Path),
6     print_path_labyrinth(Path).
7
8 % Solve labyrinth problem using IDA* algorithm
9 solve_labyrinth_ida_star() :-
10    start_state_labyrinth(Start),
11    time(ida_star(Start, goal_state_labyrinth, check_next_position, heuristic_labyrinth, Path)),
12    get_path_depth(Path),
13    print_path_labyrinth(Path).
```

activity1_labyrinth.pl

La KB del nostro programma labirinto è descritta attraverso questi 4 fatti principali:

- dim: rappresenta la dimensione del labirinto stesso
- start: rappresenta la posizione iniziale
- goal: rappresenta l'obiettivo (ce ne devono essere almeno due)
- block: rappresenta una posizione inaccessibile del labirinto

In aggiunta vengono definiti dei predicati esclusivi per labirinto:

- Controllo se la nuova posizione è una posizione valida
- La definizione dell'euristica per gli algoritmi di ricerca: siccome il problema è definito su una griglia e le mosse possibili sono su, giù, destra e sinistra è naturale pensare che la distanza di Manhattan sia l'euristica più adeguata per questo tipo di problema



```
1 % Valid movements avoiding blocks and staying in the grid
2 check_move_labyrinth((R_Old, C), (R_New, C)) :- % up and down
3     (R_New is R_Old + 1; R_New is R_Old - 1),
4     check_inside_grid(R_New, C),
5     \+ block(R_New, C).
6
7 check_move_labyrinth((R, C_Old), (R, C_New)) :- % left and right
8     (C_New is C_Old + 1; C_New is C_Old - 1),
9     check_inside_grid(R, C_New),
10    \+ block(R, C_New).
```



```
1 % -----
2 % Heuristic: Manhattan distance between the current position and the goal
3 % -----
4 heuristic_labyrinth((Row, Column), H) :-
5     findall(
6         Distance,
7         (goal(Row_Goal, Column_Goal), Distance is abs(Row - Row_Goal) + abs(Column - Column_Goal)),
8         Distances
9     ),
10    min_list(Distances, H).
```

search.pl

Implementa gli algoritmi di ricerca nello spazio degli stati A^* e IDA^* .

Per rendere il codice il più modulare possibile passiamo come argomenti i predicati specifici del problema che stiamo considerando e che sono definiti in `activity1_labirinth.pl` (o `activity1_puzzle.pl`) e attraverso il predicato built-in `call` verrà eseguito il predicato passato come argomento.

Per rappresentare le posizioni all'interno del labirinto utilizziamo un *compound term* personalizzato chiamato *node*:

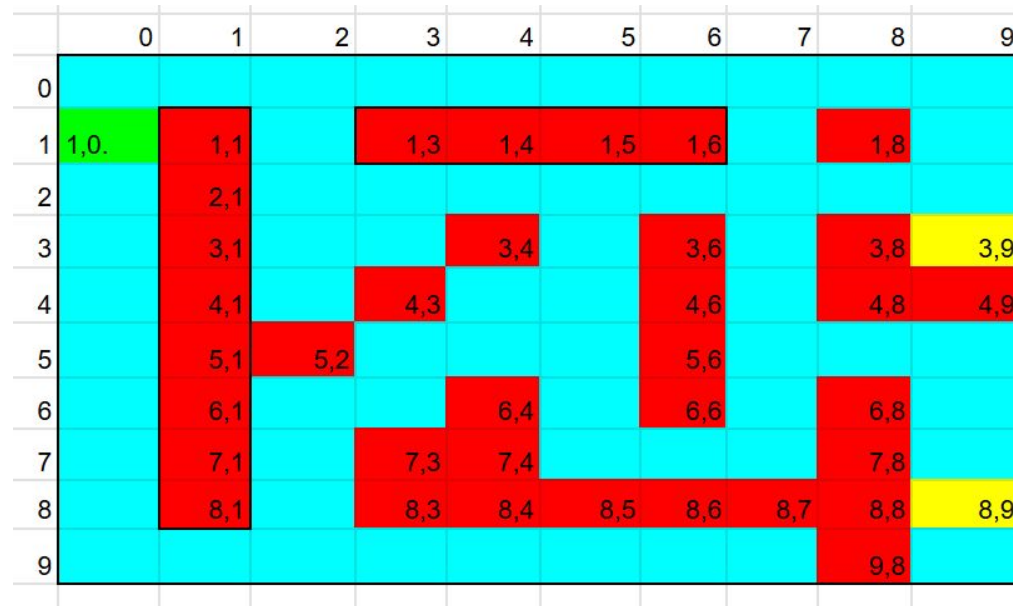
- `node(Position, G, H, F, Path)`

Confronto A* e IDA* nel problema labirinto

Andiamo adesso a stressare gli algoritmi A^* e IDA^* creando 3 configurazioni e confrontando i risultati.

La prima configurazione è quella più semplice tra le tre:

Qui A^* e IDA^* si comportano allo stesso modo sia a livello di soluzione proposta, sia a livello di tempistiche. È possibile notare una differenza sostanziale sul numero di inferenze dei due algoritmi dovuto alla loro diverse caratteristiche.



Possiamo notare che le inferenze, ossia i passi logici eseguiti dal motore di inferenza di Prolog, sono più numerose nell'esecuzione di IDA*. Questo perché l'algoritmo lavora in modo più esplorativo e ripetitivo per ridurre l'uso di memoria, mentre A* utilizza strutture dati come le liste per gestire i nodi, rendendolo più veloce e ottimizzato, ma a costo di un maggiore consumo di memoria.

Con le successive configurazioni questi dettagli saranno ancora più evidenti.

```
4 ?- solve_labyrinth_a_star().  
% 3,531 inferences, 0.000 CPU in 0.001 seconds (0% CPU, Infinite Lips)  
Solution length: 14  
Start (1,0) -> (0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (2,5) -> (2,6) ->  
-> (2,7) -> (2,8) -> (2,9) -> (3,9) Finish.  
true.
```

```
5 ?- solve_labyrinth_ida_star().  
% 4,887 inferences, 0.000 CPU in 0.001 seconds (0% CPU, Infinite Lips)  
Solution length: 14  
Start (1,0) -> (0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (2,5) -> (2,6) ->  
-> (2,7) -> (2,8) -> (2,9) -> (3,9) Finish.  
true.
```

Con questa configurazione cerchiamo di mettere in difficoltà gli algoritmi sulla gestione dell'espansione dei nodi.

In questo caso A^* lavora molto bene in quando controlla se un nodo è già espanso e quindi fa già parte della open list.

Non si può dire lo stesso per IDA* che trova una soluzione ma non in un tempo ragionevole, questo è dovuto principalmente all'assenza di memoria che caratterizza questo algoritmo.



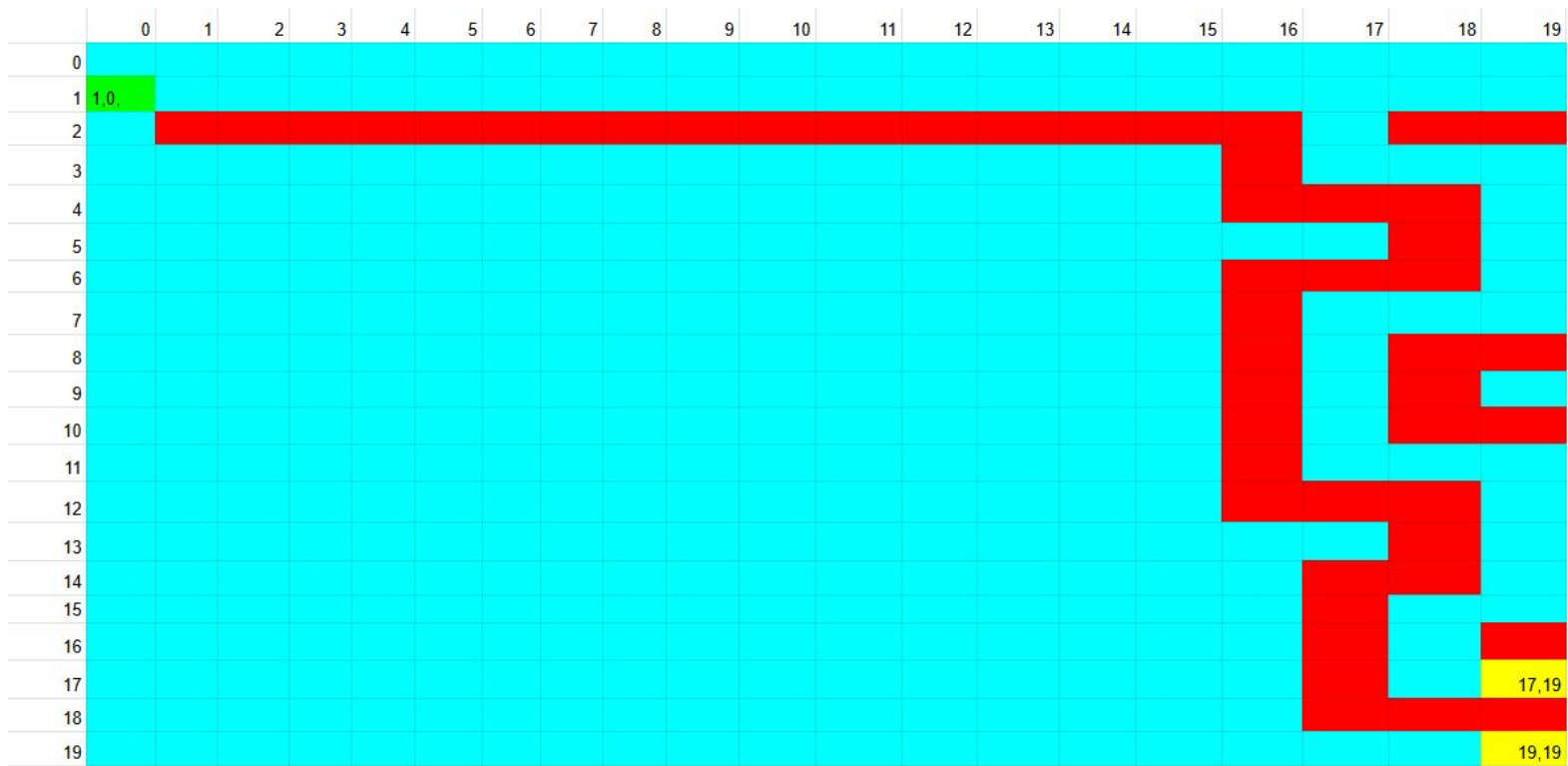
Possiamo notare come A* risolve il problema in tempo in linea con le aspettative, dato che tiene traccia dei nodi da esplorare nella open list si evitano ripetizioni di espansione di nodi.

Non possiamo dire lo stesso per IDA* che, lavora usando meno memoria possibile, ripete sempre molti passi e non solo, va ad espandere molti nodi già espansi in precedenza. Questo porta ad un tempo di esecuzione poco accettabile.

```
2 ?- solve_labyrinth_a_star().
% 75,883 inferences, 0.016 CPU in 0.017 seconds (92% CPU, 4856512 Lips)
Solution length: 36
Start (3,1) -> (3,2) -> (3,3) -> (2,3) -> (1,3) -> (0,3) -> (0,4) -> (0,5) -> (1,5) -> (2,5) ->
(3,5) -> (4,5) -> (5,5) -> (6,5) -> (7,5) -> (7,4) -> (7,3) -> (7,2) -> (7,1) -> (7,0) ->
(8,0) -> (9,0) -> (9,1) -> (9,2) -> (9,3) -> (9,4) -> (9,5) -> (9,6) -> (9,7) -> (8,7) ->
(7,7) -> (7,8) -> (7,9) -> (6,9) -> (5,9) -> (4,9) Finish.
true.
```

```
3 ?- solve_labyrinth_ida_star().
% 2,861,151,189 inferences, 304.750 CPU in 314.697 seconds (97% CPU, 9388519 Lips)
Solution length: 36
Start (3,1) -> (2,1) -> (1,1) -> (0,1) -> (0,2) -> (0,3) -> (0,4) -> (0,5) -> (1,5) -> (2,5) ->
(3,5) -> (4,5) -> (5,5) -> (6,5) -> (7,5) -> (7,4) -> (7,3) -> (7,2) -> (7,1) -> (7,0) ->
(8,0) -> (9,0) -> (9,1) -> (9,2) -> (9,3) -> (9,4) -> (9,5) -> (9,6) -> (9,7) -> (8,7) ->
(7,7) -> (6,7) -> (5,7) -> (4,7) -> (4,8) -> (4,9) Finish.
true.
```

Nell'ultima configurazione ci spostiamo su un labirinto 20x20, cercando di capire se raddoppiando le dimensioni del labirinto gli algoritmi si comportano diversamente.



A* trova la soluzione ottima in un tempo ragionevole riuscendo a gestire bene anche i labirinti di grandi dimensioni.

Non si può dire lo stesso per IDA* che, dopo più di un ora di esecuzione, non riesce a terminare l'esecuzione. Concludiamo quindi che IDA* soffre ancora di più le configurazioni di grandi dimensioni dove lo spazio degli stati è ancora maggiore.

```
5 ?- solve_labyrinth_a_star().
% 691,950 inferences, 0.062 CPU in 0.071 seconds (88% CPU, 11071200 Lips)
Solution length: 38
Start (1,0) -> (2,0) -> (3,0) -> (4,0) -> (5,0) -> (6,0) -> (7,0) -> (8,0) -> (9,0) -> (10,0) ->
(11,0) -> (12,0) -> (13,0) -> (14,0) -> (15,0) -> (16,0) -> (17,0) -> (17,1) -> (17,2) -> (17,3) ->
(17,4) -> (17,5) -> (17,6) -> (17,7) -> (17,8) -> (17,9) -> (17,10) -> (17,11) -> (17,12) -> (17,13) ->
(17,14) -> (17,15) -> (17,16) -> (18,16) -> (19,16) -> (19,17) -> (19,18) -> (19,19) Finish.
true.
```

```
6 ?- solve_labyrinth_ida_star().
|
```

Progetto 2: Puzzle dell'8

Il secondo esercizio in Prolog richiesto è l'implementazione del gioco dell'8 utilizzando sempre A^* e IDA^* come algoritmi di ricerca nello spazio degli stati.

Anche in questo caso abbiamo modularizzato l'esercizio in 3 file:

- **main.pl**: punto di ingresso per richiamare il predicato iniziale di avvio
- **activity1_puzzle.pl**: contiene la KB e predicati univoci per il puzzle, incluse le euristiche
- **search.pl**: contiene le implementazioni di A^* e IDA^*

main.pl

È lo stesso file dell'esercizio precedente, in questo caso richiameremo due predicati differenti per la risoluzione di questo esercizio con ognuno lo specifico algoritmo di risoluzione.

```
1 % Solve puzzle problem using A* algorithm
2 solve_puzzle_a_star() :-
3     start_state_puzzle(Start),
4     time(a_star(Start, goal_check_puzzle, check_next_configuration, heuristic_puzzle, Path)),
5     get_path_depth(Path),
6     print_puzzle_grid(Path).
7
8 % Solve puzzle problem using IDA* algorithm
9 solve_puzzle_ida_star() :-
10    start_state_puzzle(Start),
11    time(ida_star(Start, goal_check_puzzle, check_next_configuration, heuristic_puzzle, Path)),
12    get_path_depth(Path),
13    print_puzzle_grid(Path).
```

activity1_puzzle.pl

Per rappresentare correttamente la KB di questo problema abbiamo bisogno:

- Della configurazione iniziale del problema
- Della configurazione obiettivo del problema
- Della configurazione degli indici di adiacenza: questo perché abbiamo deciso di rappresentare la **configurazione del puzzle come una lista**

Per i predicati esclusivi dell'esercizio ci saranno:

- Controllo se una mossa è valida
- Implementazione dell'euristica, dove anche in questo caso abbiamo deciso di implementare una versione della distanza di Manhattan
- In questo particolare caso la distanza di Manhattan è relativa a tutte le celle che compongono la configurazione: per ogni cella si calcolerà quanto dista dalla sua posizione desiderata e verrà sommato il totale



```
1 % -----
2 % Puzzle domain definition
3 % -----
4
5 start_state_puzzle([7, 3, 1,
6                     5, 0, 6,
7                     8, 2, 4]).
8 goal_state_puzzle([1, 2, 3,
9                    4, 5, 6,
10                   7, 8, 0]).
11
12 % Index adjacencies in the 3x3 grid
13 neighbour(0, 1). neighbour(0, 3).
14 neighbour(1, 0). neighbour(1, 2). neighbour(1, 4).
15 neighbour(2, 1). neighbour(2, 5).
16 neighbour(3, 0). neighbour(3, 4). neighbour(3, 6).
17 neighbour(4, 1). neighbour(4, 3). neighbour(4, 5). neighbour(4, 7).
18 neighbour(5, 2). neighbour(5, 4). neighbour(5, 8).
19 neighbour(6, 3). neighbour(6, 7).
20 neighbour(7, 4). neighbour(7, 6). neighbour(7, 8).
21 neighbour(8, 5). neighbour(8, 7).
```

search.pl

Grazie alla modularizzazione del codice, il file `search.pl` che contiene le implementazioni degli algoritmi, è pensato per essere eseguito da qualsiasi tipo di problema.

Il codice quindi non è stato modificato rispetto all'esercizio precedente. Quindi per rappresentare le varie combinazioni che compongono una particolare configurazione del puzzle dell'8 utilizziamo un *compound term* personalizzato chiamato *node*:

- `node(Position, G, H, F, Path)`
- Nota bene: si potrebbe ridenominare `Position` con un termine più generale come `State`

Tutte le configurazioni sono risolvibili?

In realtà no, non tutte le configurazioni del gioco dell'8 sono risolvibili.

Per valutare se una configurazione è risolubile dobbiamo usare come criterio di risolubilità le *inversioni*.

Un'inversione è una coppia di tessere (a, b) in adiacenza, tali che:

- a appare prima di b nel vettore della configurazione (escludendo il vuoto)
- ma $a > b$

Più specificamente nel gioco dell'8 il numero delle inversioni deve essere necessariamente pari per essere risolubile. Altrimenti non lo è.

Quindi solo metà delle possibili configurazioni sono effettivamente risolvibili.

- $9! = 362.880$ configurazioni totali, quindi solo 181.440 sono risolvibili

- Questa configurazione ha le caselle 8 e 7 fuori posto, ed è considerata una inversione siccome $8 > 7$.

Abbiamo quindi un numero di inversioni dispari e quindi questa configurazione non è risolvibile.

1	2	3
4	5	6
8	7	0

- Questa configurazione invece ha le caselle 0 (quella libera) e 8 fuori posto, ma $0 < 8$ e quindi non la consideriamo inversione.

Le inversioni sono 0 che è un numero pari, perciò questa configurazione è risolvibile

1	2	3
4	5	6
7	0	8

Confronto A* e IDA* nel problema puzzle

Eseguiamo 3 configurazioni diverse e osserviamo se ci sono delle differenze sostanziali tra A* e IDA* per la risoluzione di questo tipo di problema.

Partiamo da una configurazione molto semplice :

1	2	3
4	5	6
0	7	8

```
3 ?- solve_puzzle_a_star().
% 33,821 inferences, 0.016 CPU in 0.039 seconds (40% CPU, 2164544 Lips)
Solution length: 3
[1,2,3]    [1,2,3]    [1,2,3]
[4,5,6] -> [4,5,6] -> [4,5,6]
[0,7,8]    [7,0,8]    [7,8,0]
true.
```

```
4 ?- solve_puzzle_ida_star().
% 2,017 inferences, 0.000 CPU in 0.001 seconds (0% CPU, Infinite Lips)
Solution length: 3
[1,2,3]    [1,2,3]    [1,2,3]
[4,5,6] -> [4,5,6] -> [4,5,6]
[0,7,8]    [7,0,8]    [7,8,0]
true.
```

Notiamo che IDA* si comporta meglio sia in termini di tempo, sia in termini numero di inferenze.

Nelle configurazioni successive sarà ancora più evidente.

Eseguiamo adesso la configurazione nella consegna:

E osserviamo i seguenti risultati:

7	3	1
5	0	6
8	2	4

```
2 ?- solve_puzzle_a_star().
% 747,575 inferences, 0.062 CPU in 0.103 seconds (61% CPU, 11961200 Lips)
Solution length: 21
[7,3,1] [7,3,1] [7,3,1] [7,3,1] [7,3,1] [7,0,1] [7,1,0] [7,1,2]
[5,0,6] -> [5,2,6] -> [5,2,6] -> [5,2,0] -> [5,0,2] -> [5,3,2] -> [5,3,2] -> [5,3,0] ->
[8,2,4] [8,0,4] [8,4,0] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6]

[7,1,2] [7,1,2] [0,1,2] [1,0,2] [1,2,0] [1,2,3] [1,2,3] [1,2,3]
[5,0,3] -> [0,5,3] -> [7,5,3] -> [7,5,3] -> [7,5,3] -> [7,5,0] -> [7,0,5] -> [7,4,5] ->
[8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,0,6]

[1,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3]
[7,4,5] -> [0,4,5] -> [4,0,5] -> [4,5,0] -> [4,5,6]
[0,8,6] [7,8,6] [7,8,6] [7,8,6] [7,8,0]

true.
```

```
3 ?- solve_puzzle_ida_star().
% 631,589 inferences, 0.031 CPU in 0.094 seconds (33% CPU, 20210848 Lips)
Solution length: 21
[7,3,1] [7,3,1] [7,3,1] [7,3,1] [7,3,1] [7,0,1] [7,1,0] [7,1,2]
[5,0,6] -> [5,2,6] -> [5,2,6] -> [5,2,0] -> [5,0,2] -> [5,3,2] -> [5,3,2] -> [5,3,0] ->
[8,2,4] [8,0,4] [8,4,0] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6]

[7,1,2] [7,1,2] [0,1,2] [1,0,2] [1,2,0] [1,2,3] [1,2,3] [1,2,3]
[5,0,3] -> [0,5,3] -> [7,5,3] -> [7,5,3] -> [7,5,3] -> [7,5,0] -> [7,0,5] -> [7,4,5] ->
[8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,4,6] [8,0,6]

[1,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3]
[7,4,5] -> [0,4,5] -> [4,0,5] -> [4,5,0] -> [4,5,6]
[0,8,6] [7,8,6] [7,8,6] [7,8,6] [7,8,0]

true.
```

Anche nelle configurazioni di media difficoltà IDA* si comporta leggermente meglio di A* sia nel tempo, sia nelle inferenze.

Come ultima configurazione ne proponiamo una difficile e osserviamo se riesce a risolverla nel minor numero di passi, ossia 31:

Questo per verificare anche l'ottimalità degli algoritmi.

8	6	7
2	5	4
3	0	1

```
9 ?- solve_puzzle_a_star().
% 752,165,704 inferences, 57.297 CPU in 109.623 seconds (52% CPU, 13127517 Lips)
Solution length: 32
[8,6,7] [8,6,7] [8,6,7] [8,6,7] [8,6,7] [8,6,7] [8,6,7] [8,6,0]
[2,5,4] -> [2,5,4] -> [0,5,4] -> [5,0,4] -> [5,3,4] -> [5,3,4] -> [5,3,0] -> [5,3,7] ->
[3,0,1] [0,3,1] [2,3,1] [2,3,1] [2,0,1] [2,1,0] [2,1,4] [2,1,4]

[8,0,6] [8,3,6] [8,3,6] [8,3,6] [8,3,6] [8,3,6] [8,3,6] [8,3,0]
[5,3,7] -> [5,0,7] -> [0,5,7] -> [2,5,7] -> [2,5,7] -> [2,5,7] -> [2,5,0] -> [2,5,6] ->
[2,1,4] [2,1,4] [2,1,4] [0,1,4] [1,0,4] [1,4,0] [1,4,7] [1,4,7]

[8,0,3] [0,8,3] [2,8,3] [2,8,3] [2,8,3] [2,8,3] [2,8,3] [2,8,3]
[2,5,6] -> [2,5,6] -> [0,5,6] -> [1,5,6] -> [1,5,6] -> [1,5,6] -> [1,5,0] -> [1,0,5] ->
[1,4,7] [1,4,7] [1,4,7] [0,4,7] [4,0,7] [4,7,0] [4,7,6] [4,7,6]

[2,0,3] [0,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3]
[1,8,5] -> [1,8,5] -> [0,8,5] -> [4,8,5] -> [4,8,5] -> [4,0,5] -> [4,5,0] -> [4,5,6] ->
[4,7,6] [4,7,6] [4,7,6] [0,7,6] [7,0,6] [7,8,6] [7,8,6] [7,8,0]

true.
```

Anche in questo caso IDA* nettamente migliore sia a livello di tempo che di inferenze generate.

```
10 ?- solve_puzzle_ida_star().
% 49,849,890 inferences, 3.719 CPU in 6.924 seconds (54% CPU, 13405012 Lips)
Solution length: 32
[8,6,7] [8,6,7] [8,0,7] [0,8,7] [2,8,7] [2,8,7] [2,8,7] [2,8,7]
[2,5,4] -> [2,0,4] -> [2,6,4] -> [2,6,4] -> [0,6,4] -> [3,6,4] -> [3,6,4] -> [3,6,4] ->
[3,0,1] [3,5,1] [3,5,1] [3,5,1] [3,5,1] [3,5,1] [3,5,1] [3,5,1]

[2,8,7] [2,8,0] [2,0,8] [2,6,8] [2,6,8] [2,6,8] [2,6,8] [2,6,8]
[3,6,0] -> [3,6,7] -> [3,6,7] -> [3,6,7] -> [0,3,7] -> [5,3,7] -> [5,3,7] -> [5,3,7] ->
[5,1,4] [5,1,4] [5,1,4] [5,1,4] [5,1,4] [5,1,4] [0,1,4] [1,0,4] [1,4,0]

[2,6,8] [2,6,0] [2,0,6] [2,3,6] [2,3,6] [2,3,6] [2,3,6] [2,3,6]
[5,3,0] -> [5,3,8] -> [5,3,8] -> [5,0,8] -> [0,5,8] -> [1,5,8] -> [1,5,8] -> [1,5,8] ->
[1,4,7] [1,4,7] [1,4,7] [1,4,7] [1,4,7] [1,4,7] [0,4,7] [4,0,7] [4,7,0]

[2,3,6] [2,3,0] [2,0,3] [0,2,3] [1,2,3] [1,2,3] [1,2,3] [1,2,3]
[1,5,0] -> [1,5,6] -> [1,5,6] -> [1,5,6] -> [1,5,6] -> [4,5,6] -> [4,5,6] -> [4,5,6] ->
[4,7,8] [4,7,8] [4,7,8] [4,7,8] [4,7,8] [4,7,8] [0,7,8] [7,0,8] [7,8,0]

true.
```

Conclusioni

Alla luce dei due esercizi fatti possiamo trarre delle conclusioni:

- A* si comporta meglio nel problema del labirinto
 - Questo perché lo spazio degli stati è molto ampio (10x10 o 20x20) ma piatto cioè senza grosse ramificazioni
 - In questi contesti è molto utile avere strutture dati che facilitano l'esplorazione (evitando i duplicati usando la memoria)
- IDA* si comporta meglio nel problema del gioco dell'8
 - Questo perché IDA* eccelle nei problemi dove lo spazio degli stati è stretto (3x3) e profondo (molti duplicati), cioè con molte ramificazioni
 - In questi contesti è meglio evitare al minimo la memoria per minimizzare il rischio di overload

Progetti Clingo

Scheduler competizione sportiva

Il primo progetto da sviluppare in Clingo consisteva nella generazione di un calendario di una competizione sportiva con i suoi relativi vincoli.

Per la risoluzione di questo esercizio si è deciso di suddividere in tre macro parti:

- Assegnazione delle squadre ai gironi
- Generazione delle partite
- Formattazione output risultati tramite uno script python per una visualizzazione migliore
 - `clingo calendar_generator.lp > output.txt`
 - `python3 format_results.py`

Assegnazione squadre ai gironi

- 1 choice rule per assegnare una squadra a un girone
- 1 regola di deduzione per derivare il fatto che in quel girone è presente una squadra di quel continente
- 2 hard constraint:
 - Ogni girone ha esattamente una squadra per fascia
 - Ogni gruppo deve avere almeno 3 squadre provenienti da continenti diversi



```
1  % ==== ASSIGNMENT OF TEAMS TO THE GROUPS ====
2
3  % Each team in exactly one group
4  { assigned(T,G) : group(G) } = 1 :- team(T,_,_).
5
6  % Each group has exactly one team from each pot
7  :- group(G), pot(P), #count { T : team(T,_,P), assigned(T,G) } !=
8  1.
9
10 % Which continents appear in a circle
11
12 % Each group must have at least 3 distinct continents
13 :- group(G), #count { C : cont_in(G,C) } < 3.
```

Generazione delle partite

- 1 choice rule per generare le partite
- 2 regole deduttive per ottenere i fatti partite simmetriche
- 2 hard constraint:
 - Ogni squadra può giocare una sola partita a giornata
 - In ogni gruppo, per ogni girone ci devono essere esattamente due partite

```
1 % ==== GAME GENERATION ====
2
3 % Each pair of teams from the same group plays exactly once, in one of the 3 matchdays
4 { play(T1,T2,D,G) : matchdays(D) } = 1 :- assigned(T1,G), assigned(T2,G), T1 < T2, group(G).
5
6 % Normalize the match in both directions to count the games per team
7 play_norm(T1,T2,D,G) :- play(T1,T2,D,G).
8 play_norm(T2,T1,D,G) :- play(T1,T2,D,G).
9
10 % Each team plays exactly 1 match per matchday
11 :- team(T,_), group(G), matchdays(D), assigned(T,G), #count { T2 : play_norm(T,T2,D,G) } !=
12 1.
13
14 % In each group, on each matchdays, there must be exactly 2 matches
15 :- group(G), matchdays(D), #count { T1,T2 : play(T1,T2,D,G) } != 2.
```

Risultati

```
=====
      GIRONI
=====

Girone G1:
- australia
- brazil
- croatia
- qatar

Girone G2:
- germany
- japan
- paraguay
- peru

Girone G3:
- italy
- new_zealand
- senegal
- united_states
```

```
Girone G4:
- canada
- chile
- egypt
- england

Girone G5:
- netherlands
- nigeria
- sweden
- uruguay

Girone G6:
- morocco
- south_corea
- spain
- switzerland
```

```
Girone G7:
- argentina
- ecuador
- iran
- mexico

Girone G8:
- austria
- colombia
- costa_rica
- france
```

```
=====
      TEMPO DI ESECUZIONE
=====
Time           : 0.411s (Solving: 0.01s 1st Model: 0.01s Unsat: 0.00s)
```

=====
CALENDARIO
=====

Giornata 1:

[G1] australia vs brazil
[G1] croatia vs qatar
[G2] germany vs paraguay
[G2] japan vs peru
[G3] italy vs senegal
[G3] new_zeland vs united_states
[G4] canada vs chile
[G4] egypt vs england
[G5] netherlands vs uruguay
[G5] nigeria vs sweden
[G6] morocco vs south_corea
[G6] spain vs switzerland
[G7] argentina vs iran
[G7] ecuador vs mexico
[G8] austria vs costa_rica
[G8] colombia vs france

Giornata 2:

[G1] australia vs croatia
[G1] brazil vs qatar
[G2] germany vs japan
[G2] paraguay vs peru
[G3] italy vs new_zeland
[G3] senegal vs united_states
[G4] canada vs egypt
[G4] chile vs england
[G5] netherlands vs sweden
[G5] nigeria vs uruguay
[G6] morocco vs spain
[G6] south_corea vs switzerland
[G7] argentina vs mexico
[G7] ecuador vs iran
[G8] austria vs colombia
[G8] costa_rica vs france

Giornata 3:

[G1] australia vs qatar
[G1] brazil vs croatia
[G2] germany vs peru
[G2] japan vs paraguay
[G3] italy vs united_states
[G3] new_zeland vs senegal
[G4] canada vs england
[G4] chile vs egypt
[G5] netherlands vs nigeria
[G5] sweden vs uruguay
[G6] morocco vs switzerland
[G6] south_corea vs spain
[G7] argentina vs ecuador
[G7] iran vs mexico
[G8] austria vs france
[G8] colombia vs costa_rica

Scheduler master UniTo

Il secondo e ultimo progetto in Clingo consisteva nella generazione di un calendario di un Master UniTo con i suoi relativi vincoli.

Anche in questo caso si è deciso di suddividere l'esercizio in tre macro parti:

- Definizione dei vincoli strutturali
- Definizione dei vincoli in relativi alla consegna
- Formattazione output risultati tramite uno script python per una visualizzazione migliore
 - `clingo master_program_scheduler.lp > output.txt`
 - `python3 format_results.py`

Vincoli strutturali

All'interno dei vincoli strutturali definiamo tutte le regole di derivazione che ci permettono di identificare i giorni e gli orari validi secondo la consegna.

- Choice rule per assegnare una lezione a un orario
- Una lezione è un fatto ed ha una durata di un'ora
- 5 vincoli di derivazione per ottenere giorni e orari validi
- 1 hard constraint per evitare sovrapposizioni tra lezioni

```
1  % — STRUCTURAL CONSTRAINTS AND TIMETABLES —
2
3  % Valid days per week
4  valid_day(W,D) :- week(W), fulltime_week(W), day(D).
5  valid_day(W,fri) :- week(W), not fulltime_week(W).
6  valid_day(W,sat) :- week(W), not fulltime_week(W).
7
8  % Valid hours per day
9  valid_hour(W,D,H) :- valid_day(W,D), D != sat, hour(H).
10 valid_hour(W,sat,H) :- valid_day(W,sat), hour_saturday(H).
11
12 % Each lesson can take place in a valid position on the calendar
13 { lesson(C,W,D,H) : valid_hour(W,D,H) } = Hours :- course(C,_, Hour
s).
14
15 % Avoid overlaps
16 :- lesson(C1,W,D,H), lesson(C2,W,D,H), C1 < C2.
```


Vincoli di consegna: analizziamo due vincoli interessanti

- 1 regola di deduzione per ottenere gli spot da un'ora liberi
- 1 regola di deduzione per ottenere i blocchi liberi composti da spot liberi consecutivi (2 ore)
- 1 hard constraint per evitare le sovrapposizioni
- 1 hard constraint per avere almeno 6 blocchi liberi diversi composti da 2 ore



```
1 % — CONSTRAINT 4: The calendar must include at least 6 free blocks of 2 hours each for any recoveries
2 %           of lessons cancelled or postponed
3 free_spot(W,D,H) :-
4     valid_hour(W,D,H),
5     not lesson(_,W,D,H),
6     not presentation(W,D,H).
7
8 free_block(W,D,H) :-
9     free_spot(W,D,H),
10    free_spot(W,D,H+1),
11    valid_hour(W,D,H),
12    valid_hour(W,D,H+1).
13
14 % Avoid overlaps
15 :- free_block(W,D,H), free_block(W,D,H+1).
16 :- free_block(W,D,H), free_block(W,D,H-1).
17
18 :- #count { W,D,H : free_block(W,D,H) } < 6.
```

- 2 regole di deduzione per ottenere la prima e l'ultima lezione (con giorno in stringa perché usiamo il fatto in output)
- 1 hard constraint per verificare se il corso non eccede le 8 settimane andando a contare quando settimane passano dalla prima all'ultima lezione
- span_info per una visualizzazione migliore nell'output

```

1  % — CONSTRAINT 7: the distance between the first and last lesson of each course must not exceed 8 week
  s
2
3  % Find the start and end week of each course
4  first_week(C,W,D,H) :- course(C,_,_), (W,N,H) = #min {(W1,N1,H1) : lesson(C,W1,D1,H1), day_num(D1,N1)},
5                          day_num(D,N).
6  last_week(C,W,D,H) :- course(C,_,_), (W,N,H) = #max {(W2,N2,H2) : lesson(C,W2,D2,H2), day_num(D2,N2)},
7                          day_num(D,N).
8
9  % constraint: the number of weeks in the interval [W1..W2] cannot be > 9
10 :- first_week(C,W1,_,_), last_week(C,W2,_,_),
11     #count {W : week(W), W1 <= W, W <= W2} > 9.
12     % count how many of those weeks fall in the range W1 to W2
13
14 % Another way is:
15 % :- first_week(C,W1,_,_), last_week(C,W2,_,_), week(W1), week(W2), W2 - W1 > 8.
16
17 span_info(C, W1, W2, N) :-
18     first_week(C,W1,_,_),
19     last_week(C,W2,_,_),
20     N = #count { W : week(W), W1 <= W, W <= W2 }.

```

Risultati

Settimana 3

Fri

Slot 1: project_management - Prof. muzzetto
Slot 2: project_management - Prof. muzzetto
Slot 3: progettazione_di_basi_di_dati - Prof. mazzei
Slot 4: project_management - Prof. muzzetto
Slot 5: project_management - Prof. muzzetto
Slot 6: progettazione_di_basi_di_dati - Prof. mazzei
Slot 7: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta
Slot 8: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta

Sat

Slot 1: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta
Slot 2: risorse_digitali_per_il_progetto_collaborazione_e_documentazione - Prof. boniolo
Slot 3: risorse_digitali_per_il_progetto_collaborazione_e_documentazione - Prof. boniolo
Slot 4: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta
Slot 5: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta
Slot 6: acquisizione_ed_elaborazione_di_immagini_statiche_grafica - Prof. zanchetta

```
=====
          INIZIO E FINE CORSI
=====
risorse_digitali_per_il_progetto_collaborazione_e_documentazione
  Start: settimana 1, fri, ora 3
  End:   settimana 8, sat, ora 3
  Time span: 7 settimane

project_management
  Start: settimana 1, fri, ora 4
  End:   settimana 4, fri, ora 2
  Time span: 3 settimane

acquisizione_ed_elaborazione_del_suono
  Start: settimana 1, fri, ora 5
  End:   settimana 7, fri, ora 6
  Time span: 6 settimane
```

Numero blocchi liberi: 9

=====

TEMPO DI ESECUZIONE

=====

Time : 12.058s (Solving: 0.87s 1st Model: 0.87s Unsat: 0.00s)

Settimana 16

Mon

- Slot 1: crossmedia_articolazione_delle_scritture_multimediali - Prof. taddeo
- Slot 2: crossmedia_articolazione_delle_scritture_multimediali - Prof. taddeo
- Slot 3: crossmedia_articolazione_delle_scritture_multimediali - Prof. taddeo
- Slot 4: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio
- Slot 5: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio
- Slot 7: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio
- Slot 8: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio

Tue

- Slot 1: crossmedia_articolazione_delle_scritture_multimediali - Prof. taddeo
- Slot 2: crossmedia_articolazione_delle_scritture_multimediali - Prof. taddeo
- Slot 3: acquisizione_ed_elaborazione_di_sequenze_di_immagini_digitali - Prof. ghidelli
- Slot 4: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio
- Slot 5: ambienti_di_sviluppo_e_linguaggi_client_side_per_il_web - Prof. micalizio
- Slot 7: acquisizione_ed_elaborazione_di_sequenze_di_immagini_digitali - Prof. ghidelli
- Slot 8: acquisizione_ed_elaborazione_di_sequenze_di_immagini_digitali - Prof. ghidelli

Wed

- Slot 1: introduzione_al_social_media_management - Prof. suppini
- Slot 2: acquisizione_ed_elaborazione_di_sequenze_di_immagini_digitali - Prof. ghidelli
- Slot 3: strumenti_e_metodi_di_interazione_nei_social_media - Prof. giordani
- Slot 4: strumenti_e_metodi_di_interazione_nei_social_media - Prof. giordani
- Slot 5: strumenti_e_metodi_di_interazione_nei_social_media - Prof. giordani
- Slot 7: acquisizione_ed_elaborazione_di_sequenze_di_immagini_digitali - Prof. ghidelli
- Slot 8: introduzione_al_social_media_management - Prof. suppini

Grazie per l'attenzione
