



ARTIFICIAL INTELLIGENCE

Prolog & Answer Set
Programming

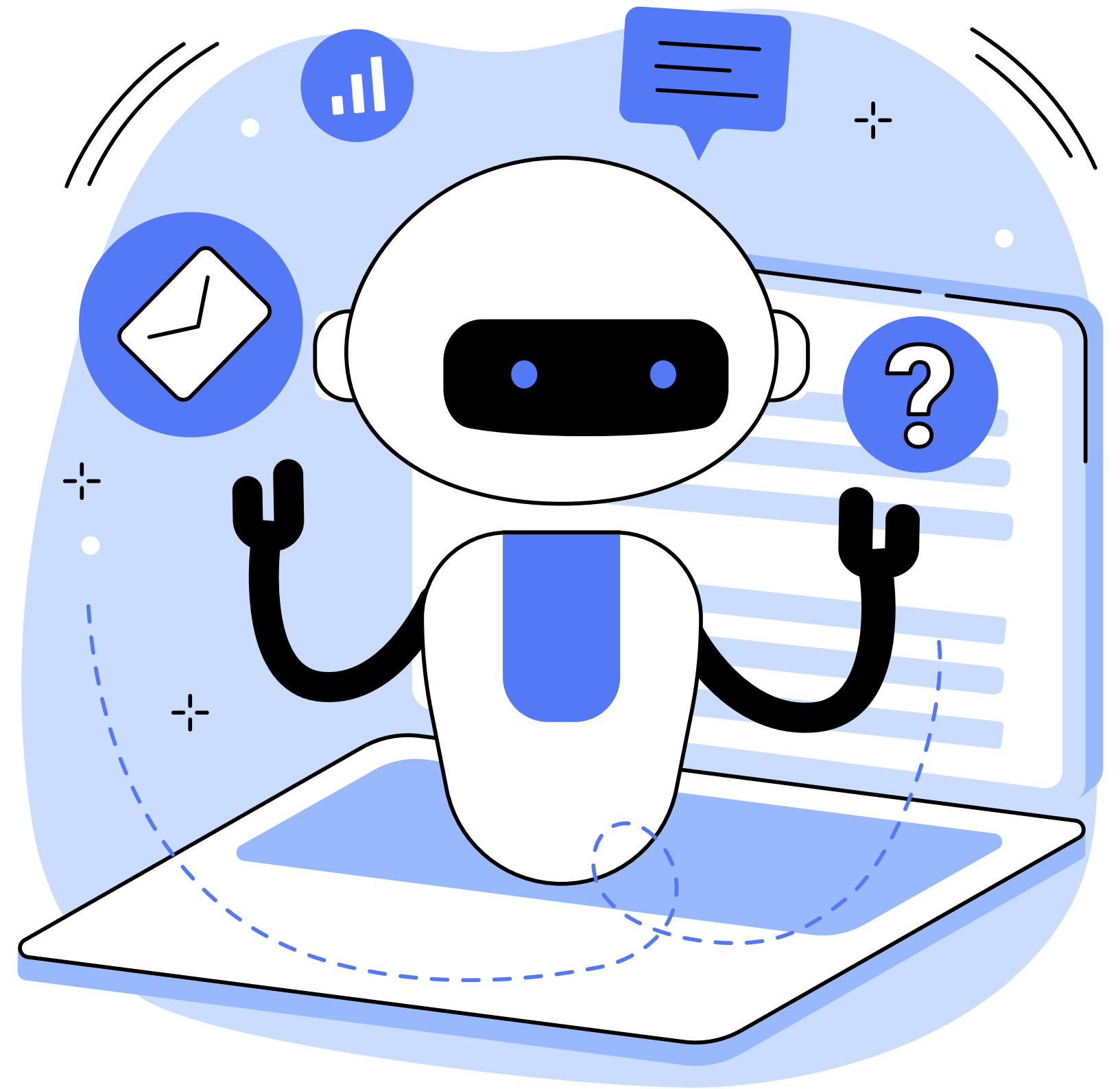
CONTENTS



- 1 Prolog
- 2 Answer Set Programming

PROLOG

Sudoku: il nostro sistema intelligente affronta il problema del Sudoku con due strategie: la prima, basata su un approccio di brute force con backtracking, esplora tutte le possibili combinazioni fino a trovare una soluzione valida. La seconda strategia si concentra sull'individuazione delle celle in cui è possibile determinare un unico valore come soluzione, senza fare backtracking.



ESEGUIRE IL PROGRAMMA

```
cd prolog-project
```

```
swipl
```

```
['startProject9x9facile.pl'].
```

```
main.
```

FILE

start project (facile,
difficile, euristiche)

strategia

dominio 4x4
facile

dominio 4x4
difficile

dominio 9x9
facile

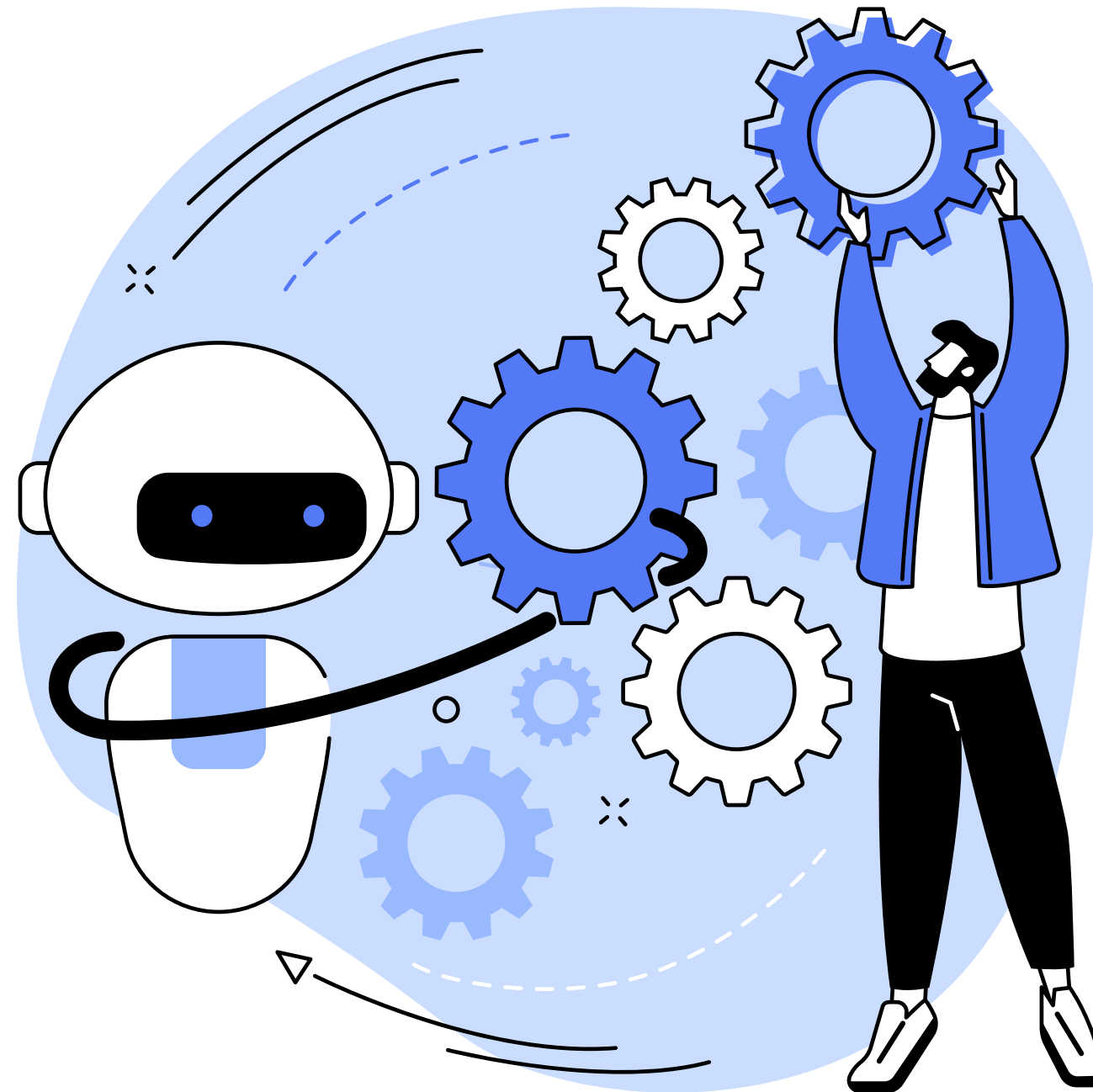
dominio 9x9
difficile

azioni

azioni_v2

azioni con
euristiche

azioni con
euristiche_v2



RAPPRESENTAZIONE DEL DOMINIO

Fatti Iniziali

- **valoreMax**(9).
- **listaPoxVal**([1,2,3,4,5,6,7,8,9]).
- **iniziale**(pos(1,1)).

Celle

- **piena**(pos(1,1),4).
- **vuota**(pos(1,2),0).

Griglie

- **griglia**(1,
[pos(1,1),pos(1,2),pos(1,3),pos(2,1),
pos(2,2),pos(2,3),pos(3,1),pos(3,2),
pos(3,3)]).

Predicato di Uscita

- **finale**:-
 findall(Valore, piena(pos(_,_),Valore);
vuota(pos(_,_),Valore), ListaValoriP),
 sommaLista(ListaValoriP,SommaP),
 sommaValoriCelle(SVC),
 SommaP is SVC.

STRATEGIA DI RICERCA

strategiaDiRicerca(Cammino):-
 iniziale(pos(RStart,CStart)),
 risolvi(pos(RStart,CStart),Cammino).

CASO BASE della RICORSIONE

risolvi(_,[]):-
 finale!,
 listing(vuota/2).

CASO RICORSIVO

risolvi(S,[Azione|ListaAzioni]):-
 applicabile(Azione,S),
 trasforma(Azione,S,SNuovo),
 risolvi(SNuovo,ListaAzioni).

AZIONI

Applicabile(azione,stato)

- data una lista di possibili valori, assegna un valore X ad una cella (in caso di fallimento torna in questo punto ed effettua il **backtracking**)
- scorri riga
- cambia riga

Azioni ausiliarie

- calcolo dei valori possibili di una cella come l'intersezione di tutti i valori possibili per la riga, per la colonna, per la griglia

Trasforma(azione,stato,nuovoStato)

- assegna valore e scorri riga
- assegna valore e cambia riga
- scorri riga
- cambia riga

ASSEGNARE un valore

Se lo stato fa riferimento a una cella vuota con valore 0. Successivamente la cella rimane come fatto vuota ma con valore diverso da 0

SCORRERE la riga

Se lo stato attuale fa riferimento a una cella piena e si può ancora avanzare nella riga

CAMBIARE la riga

Se lo stato attuale fa riferimento a una cella piena ma non si può avanzare nella riga

AZIONI_V2

Nella versione precedente, abbiamo utilizzato il predicato built-in di Prolog, l'if, solo in alcuni punti del nostro programma.

Tuttavia, in questa nuova versione, abbiamo deciso di eliminarlo completamente.

Abbiamo osservato che questa modifica ha portato a un notevole peggioramento delle prestazioni.

ASSEGNARE un valore

Se lo stato fa riferimento a una cella vuota con valore 0. Successivamente la cella rimane come fatto vuota ma con valore diverso da 0

SCORRERE la riga

Se lo stato attuale fa riferimento a una cella piena e si può ancora avanzare nella riga

CAMBIARE la riga

Se lo stato attuale fa riferimento a una cella piena ma non si può avanzare nella riga

AZIONI CON EURISTICHE

- Stessi predicati ma con l'aggiunta dell'euristica "**singoli ovvi**".
- è una delle tecniche più semplici ma efficaci, consiste nel cercare, come prima cosa, le celle che hanno solo una possibile soluzione e la assegna.
- Nell'azione assegna viene assegnato un valore solo se è l'unico possibile, altrimenti si continua la ricerca cercando la cella che ha solo un singolo valore possibile
- E' stata sviluppata anche per questa implementazione la seconda versione sfruttando la sola risoluzione SLD

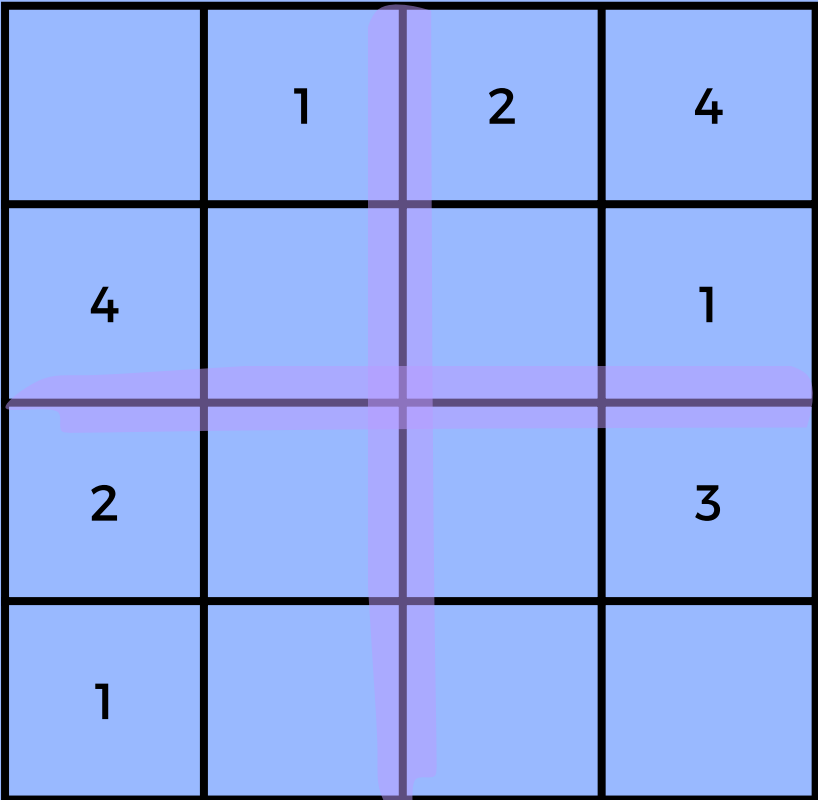
ASSEGNARE un valore

Se lo stato fa riferimento a una cella vuota con valore 0. Successivamente la cella rimane come fatto vuota ma con valore diverso da 0

Ricominciare

Se lo stato attuale fa riferimento all'ultima cella del sudoku, si ricomincia dalla prima

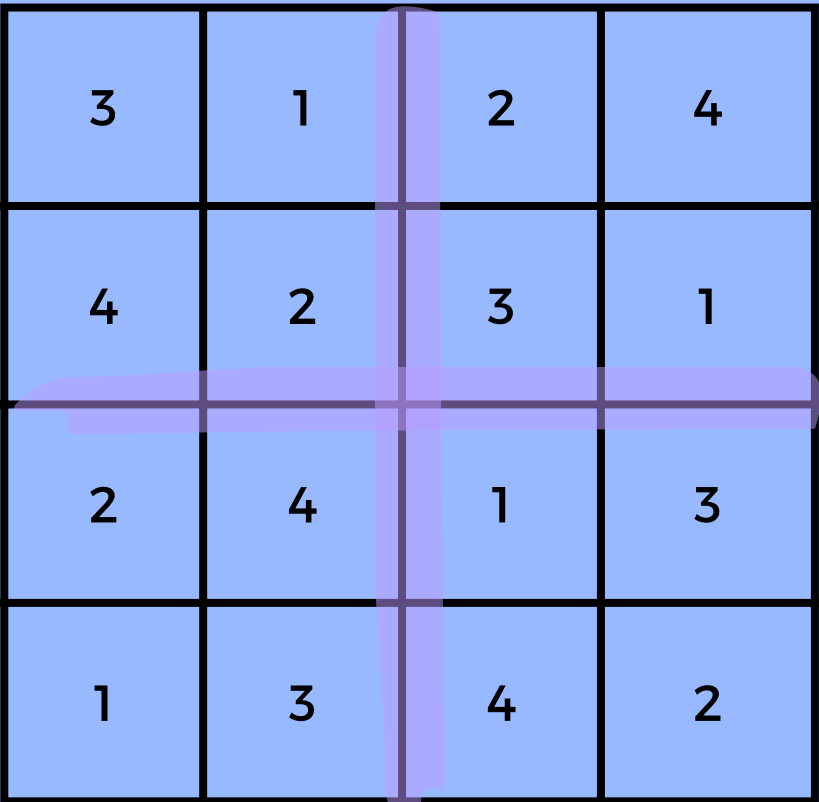
DOMINIO 4X4 FACILE



	1	2	4
4			1
2			3
1			

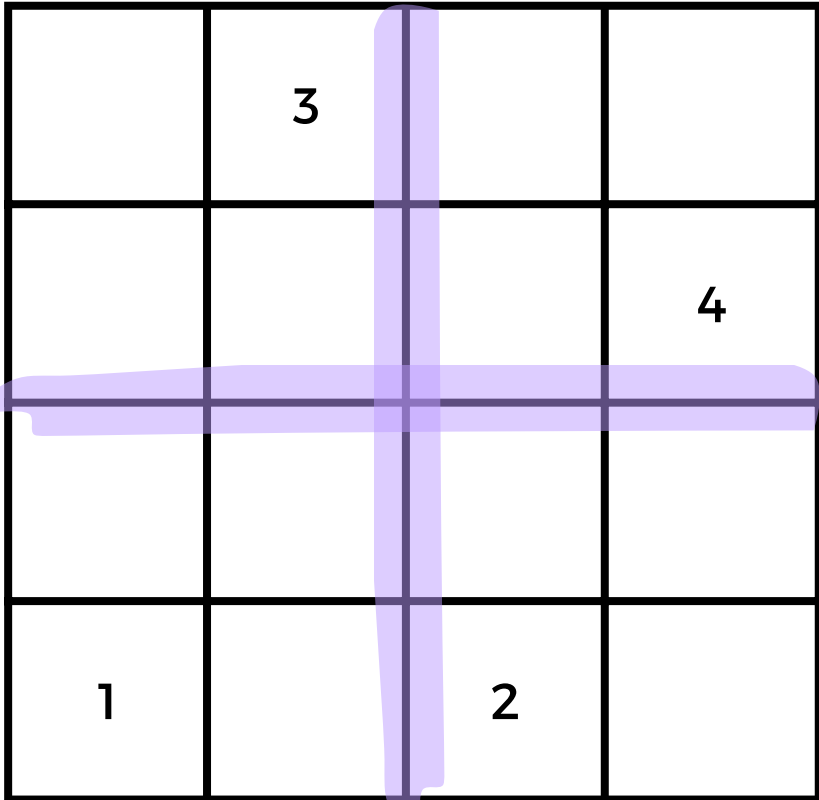
Risoluzione senza
euristica: dai 0.022 ai
0.037 secondi

Risoluzione con
euristica: dai 0.034 ai
0.039 secondi



3	1	2	4
4	2	3	1
2	4	1	3
1	3	4	2

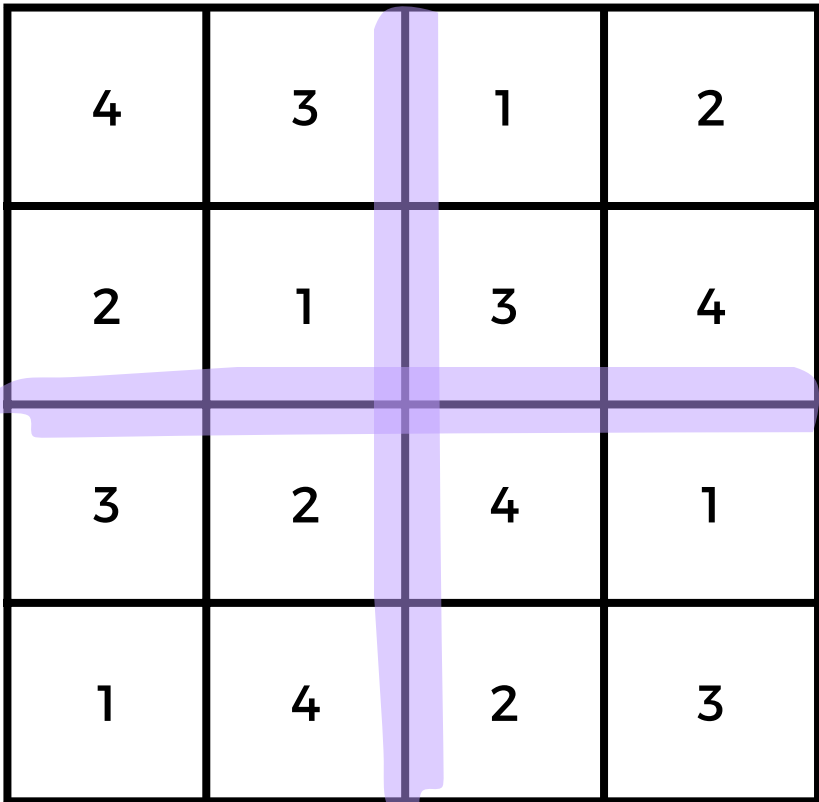
DOMINIO 4X4 DIFFICILE



	3		
			4
1		2	

Risoluzione senza
euristica: dai 0.034 ai
0.039 secondi

Risoluzione con
euristica: dai 0.030 ai
0.041 secondi



4	3	1	2
2	1	3	4
3	2	4	1
1	4	2	3

DOMINIO 9X9 FACILE

	9	1					5	3
		2		3			6	
6		3		1		7	9	2
5	7	9	4	8	2		1	6
2	1	8		9	3	5	4	7
	3	6	5	7	1	2		
9	2	4		5	8	6		1
3	8				6	9	2	5
				2			7	8

Risoluzione senza
euristica: dai 0.035 ai
0.038 secondi

Risoluzione con
euristica: dai 0.027 ai
0.037 secondi

7	9	1	2	6	4	8	5	3
8	5	2	9	3	7	1	6	4
6	4	3	8	1	5	7	9	2
5	7	9	4	8	2	3	1	6
2	1	8	6	9	3	5	4	7
4	3	6	5	7	1	2	8	9
9	2	4	7	5	8	6	3	1
3	8	7	1	4	6	9	2	5
1	6	5	3	2	9	4	7	8

DOMINIO 9X9 DIFFICILE

4							9	
1		9				5		2
		7	9		2		1	
		5	2		3			
							8	
	9		1			4		5
8	7							3
		2				7		
	3					8	5	1

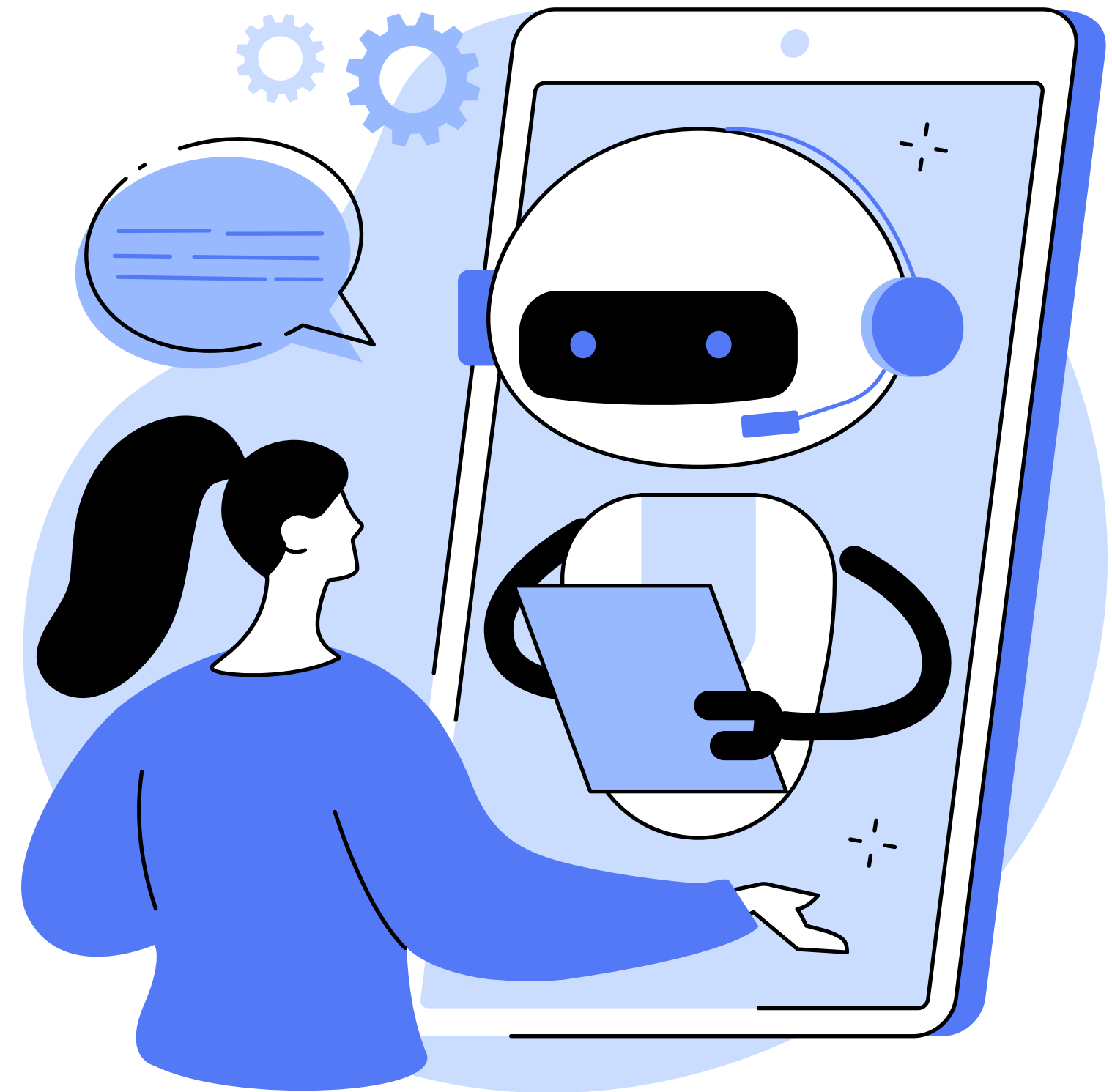
Risoluzione senza
euristica: dai 0.035 ai
0.046 secondi

Risoluzione con
euristica: non trova la
soluzione

4	2	3	5	1	7	6	9	8
1	6	9	3	4	8	5	7	2
5	8	7	9	6	2	3	1	4
7	4	5	2	8	3	1	6	9
3	1	6	4	9	5	2	8	7
2	9	8	1	7	6	4	3	5
8	7	1	6	5	4	9	2	3
9	5	2	8	3	1	7	4	6
6	3	4	7	2	9	8	5	1

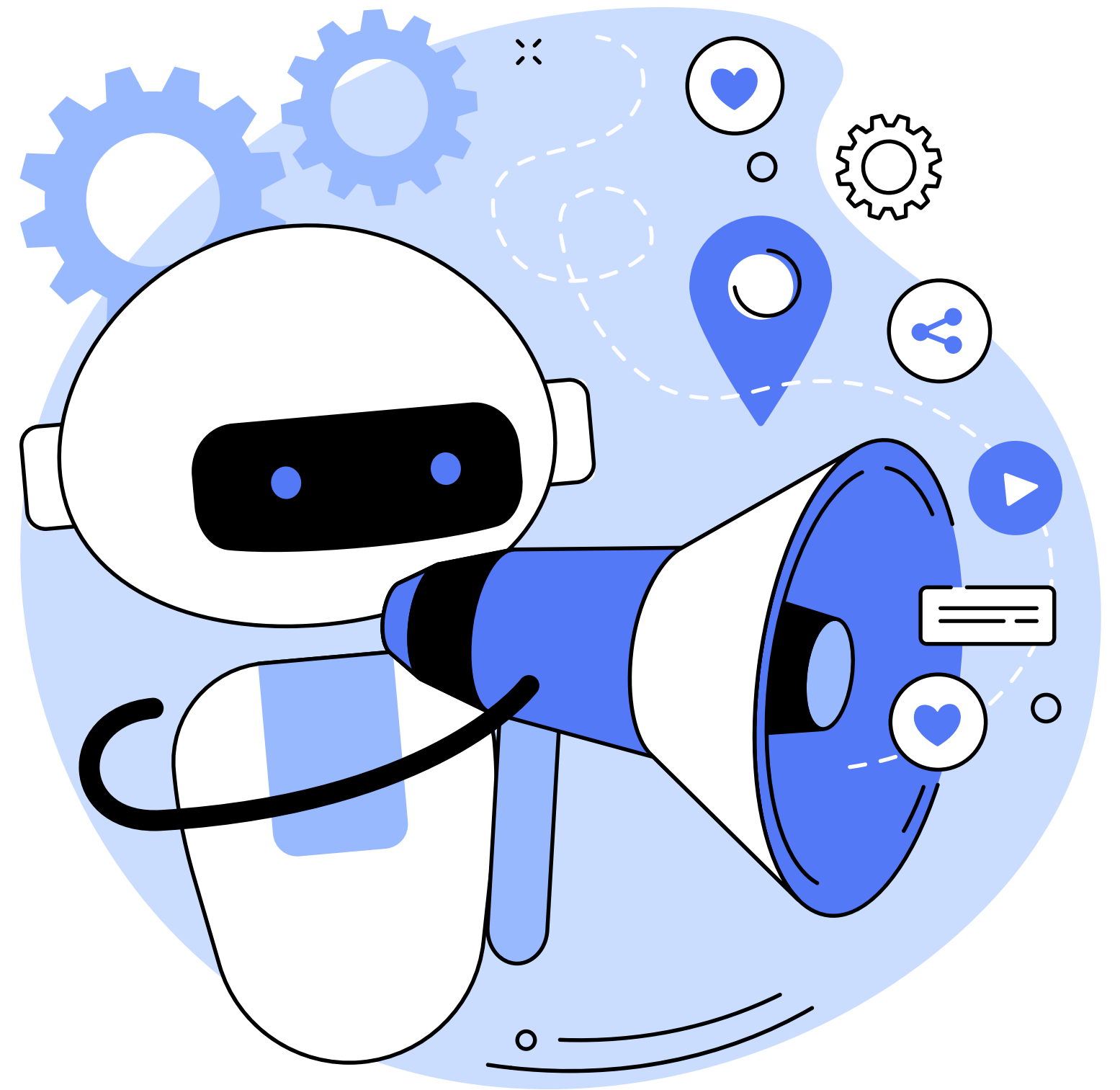
CONSIDERAZIONI FINALI

- Sebbene l'euristica basata sui singoli valori abbia dimostrato di essere efficace in alcuni casi, non ha sempre portato a una soluzione completa. Potrebbero essere necessarie euristiche più complesse per affrontare con successo una più ampia gamma di Sudoku.
- Inoltre, confrontando i tempi di esecuzione, la strategia di brute force con backtracking si è dimostrata leggermente più veloce o almeno equivalente alla strategia basata sui singoli valori, diversamente da quello che ci aspettavamo.
- La scelta dell'euristica dipende dall'esperienza personale poiché rispecchia il nostro approccio personale nel risolvere i Sudoku.



ANSWER SET PROGRAMMING

Sviluppo del progetto per la creazione di un **CALENDARIO SPORTIVO** (in esame, il calendario calcistico della Serie A) in **CLINGO**, applicando il paradigma dell'**Answer Set Programming**



ESEGUIRE IL PROGRAMMA

```
cd asp-project/v1
```

```
clingo campionato-16squadre.cl -t 8
```

CALENDARI SPORTIVI SVILUPPATI

4 squadre

Giornate totali: 6

Partite in una
giornata: **2**

Distanza tra
coppia di gare di
andata e ritorno: **2**

8 squadre

Giornate totali: 14

Partite in una
giornata: **4**

Distanza tra
coppia di gare di
andata e ritorno: **4**

12 squadre

Giornate totali: 22

Partite in una
giornata: **6**

Distanza tra
coppia di gare di
andata e ritorno: **6**

16 squadre

Giornate totali: 30

Partite in una
giornata: **8**

Distanza tra
coppia di gare di
andata e ritorno: **8**

20 squadre

Giornate totali: 38

Partite in una
giornata: **10**

Distanza tra
coppia di gare di
andata e ritorno: **10**

VERSIONE 1

Nella prima versione abbiamo descritto il dominio in maniera più **verbosa**, incrementando di conseguenza il numero di vincoli da definire.

Il codice risulta **molto leggibile** ma **calano drasticamente le prestazioni** all'aumentare delle **squadre**



VERSIONE 2

Nella seconda versione abbiamo descritto il dominio con **meno predicati, diminuendo drasticamente il numero di vincoli** da dichiarare.

Le prestazioni sono aumentate a dismisura, portando alla **terminazione** dei domini con **16 e 20 squadre** (nella prima versione, dopo 2h di esecuzione, non riuscivamo a determinare nessun modello)



VERSIONE 1

(team(juventus).
citta(c_torino).
giocaA(juventus, c_torino)

giornataAndata(1..15).
giornataRitorno(16..30).

8 {partitaAndata(S1,S2,GAndata,C1): team(S1), team(S2), S1<>S2,
giocaA(S1,C1)} 8 :- giornataAndata(GAndata).

8 {partitaRitorno(S1,S2,GRitorno,C1): team(S1),team(S2),S1<>S2,
giocaA(S1,C1)} 8 :- giornataRitorno(GRitorno).

16 squadre

Ciascuna squadra non deve giocare mai più di due partite consecutive in casa

:-partitaAndata(S1,_,G,_),partitaAndata(S1,_,G+1,_),partitaAndata(S1,_,G+2,_).
:-partitaRitorno(S1,_,G,_),partitaRitorno(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
:-partitaAndata(S1,_,G,_),partitaAndata(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
:-partitaAndata(S1,_,G,_),partitaRitorno(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
ALTRETTANTE REGOLE PER IL FUORI CASA.

20 squadre

idem

VERSIONE 2

(team(juventus).
citta(c_torino).
giocaA(juventus, c_torino)

giornata(1..38).
giornata_andata(1..19).
giornata_ritorno(20..38).

10 {partita(andata, SquadraA, SquadraB, Citta, N):
team(SquadraA),
team(SquadraB),
gioca_a(SquadraA,Citta),
SquadraA <> SquadraB} 10 :- giornata_andata(N).

% le partite di ritorno sono speculari alle partite di andata

10 {partita(ritorno, SquadraA, SquadraB, Citta, N):
team(SquadraA),
team(SquadraB),
gioca_a(SquadraA, Citta),
partita(andata, SquadraB, SquadraA, Citta2, _),
gioca_a(SquadraB,Citta2)} 10:- giornata_ritorno(N).

PRINCIPALI MODIFICHE

% Non si può giocare andata e ritorno nella stessa giornata di andata

```
:- partitaAndata(S1,S2,GAndata,C1),
   partitaAndata(S2,S1,GAndata,C2),giornataAndata(GAndata).
```

% Non si può giocare andata e ritorno nella stessa giornata di ritorno

```
:- partitaRitorno(S1,S2,GRitorno,C1),
   partitaRitorno(S2,S1,GRitorno,C2),giornataRitorno(GRitorno).
```

Ciascuna squadra non deve giocare ma più di due partite consecutive in casa

```
:-partitaAndata(S1,_,G,_),partitaAndata(S1,_,G+1,_),partitaAndata(S1,_,G+2,_).
:-partitaRitorno(S1,_,G,_),partitaRitorno(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
:-partitaAndata(S1,_,G,_),partitaAndata(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
:-partitaAndata(S1,_,G,_),partitaRitorno(S1,_,G+1,_),partitaRitorno(S1,_,G+2,_).
:-partitaAndata(_,S1,G,_),partitaRitorno(_,S1,G+1,_),partitaRitorno(_,S1,G+2,_).
```

Altrettante regole per fuori casa

% Non si può giocare andata e ritorno nella stessa giornata

```
:- team(Squadra),
   giornata(Giornata),
   Numero_Partite_A = #count{SquadraA: partita(_,SquadraA,SquadraA,_, Giornata)},
   Numero_Partite_B = #count{SquadraB: partita(_,SquadraB,Squadra,_, Giornata)},
   Numero_Partite_A + Numero_Partite_B <> 1.
```

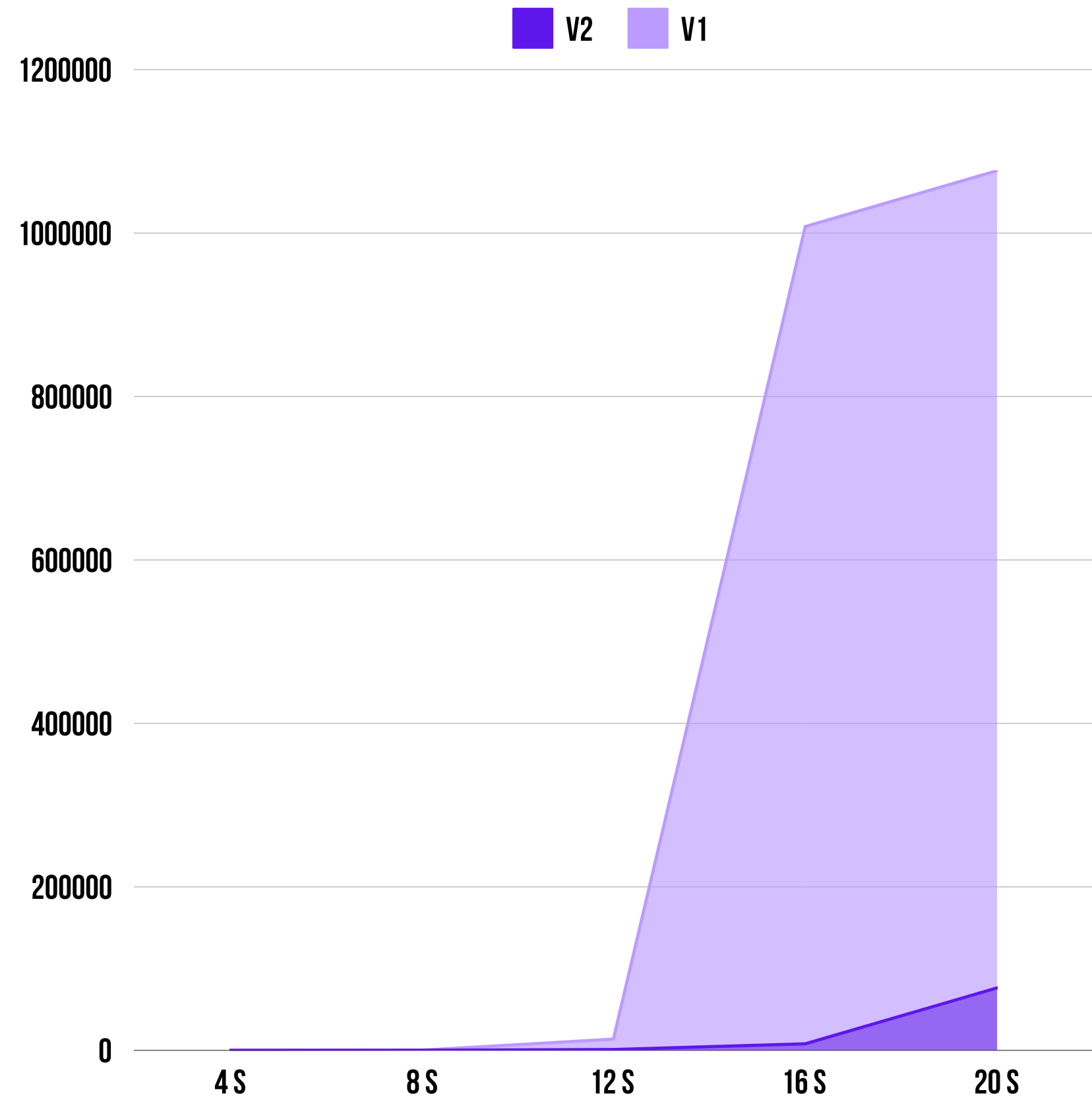
Ciascuna squadra non deve giocare mai più di due partite consecutive in casa o fuori casa;

```
:- partita(_, Squadra, _, _, G1),partita(_, Squadra, _, _, G1+1),partita(_, Squadra, _, _, G1+2).
:- partita(_, _, Squadra, _, G1),partita(_, _, Squadra, _, G1+1),partita(_, _, Squadra, _, G1+2).
```

PROIEZIONE PRESTAZIONI TEMPORALI

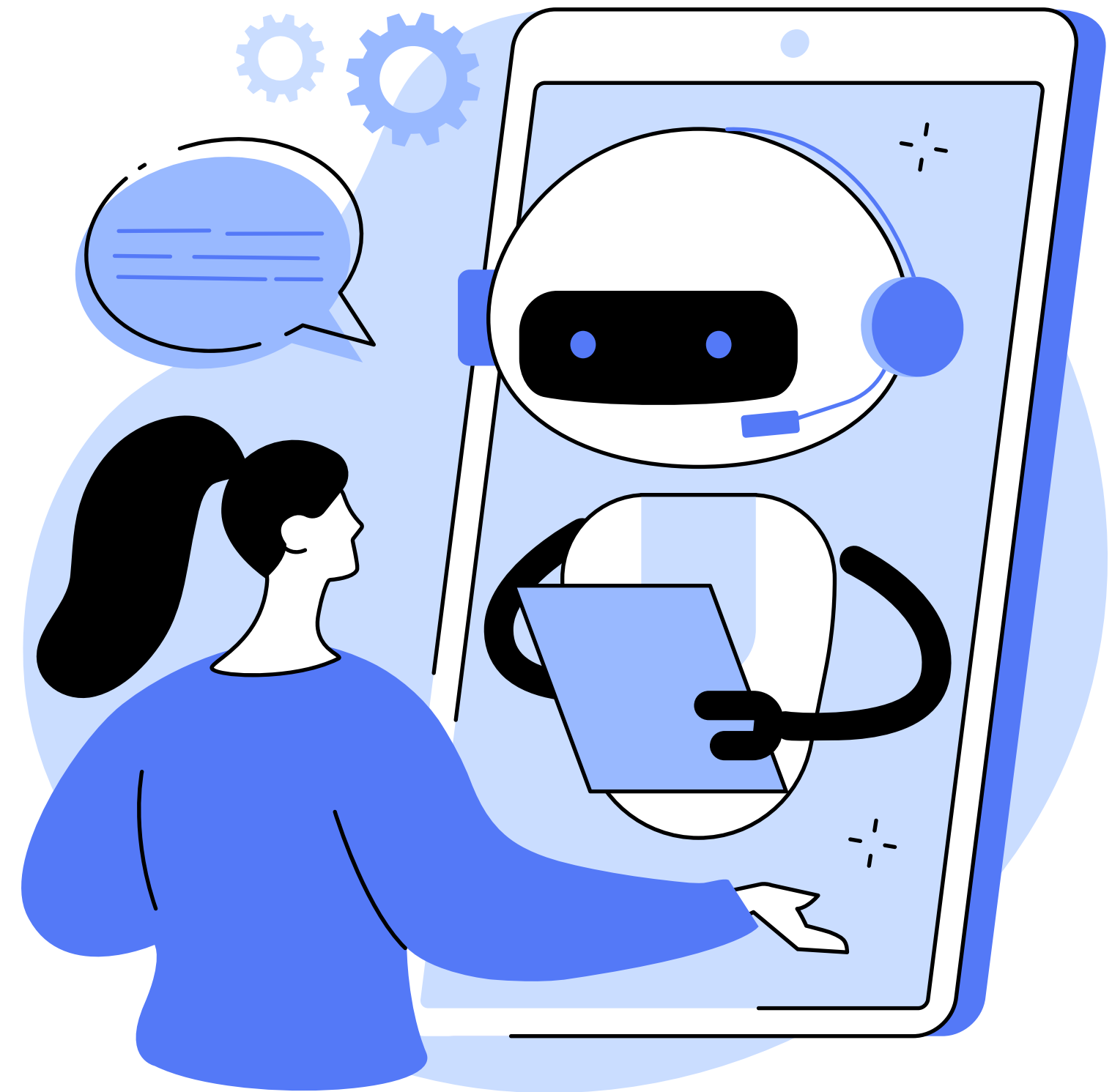
- Sulla retta delle ordinate troviamo i **millesimi di secondo** impiegati dal **ASP Solver** per trovare **un modello**.
- Dal grafico si evince come nella versione 2 le prestazioni migliorino drasticamente

NOTA: Le esecuzioni sono state effettuate tramite la libreria **clingo-dl**



CONSIDERAZIONI FINALI

- Lo studio dell'Answer Set Programming ci ha portato alla prima versione della modellazione del dominio
- Purtroppo, la v1 non è abbastanza efficiente da poter eseguire i vincoli facoltativi in un tempo ragionevole, perciò abbiamo riscritto la modellazione nella seconda versione
- Infine, nella seconda versione abbiamo notato come il numero di predicati possa incidere sulle prestazioni al crescere del task in esame.
- Siamo quindi riusciti nell'intento di migliorare l'esecuzione, che ora termina.
- Inoltre, lo studio di approfondimento ci ha portato alla scoperta di librerie nuove e di come possono essere applicate ottimizzazioni in base al tipo di dominio.





**THANK YOU FOR
LISTENING!**