Automated Reasoning Project Report

Alessandro Castelli*

Matricola: 147073

July 23, 2024

*Email: 147073@spes.uniud.it

Contents

1 Introduzione

Il presente report si concentra sull'analisi delle soluzioni al problema assegnato, riguardante l'organizzazione di un torneo di calcio a 7. Il problema in questione presenta diverse sfide legate alla suddivisione ottimale dei giocatori in squadre, rispettando specifici vincoli e minimizzando le penalità assegnate alle diverse configurazioni di squadra.

1.1 Problema: Torneo di calcio a 7

> Consideriamo un insieme di n giocatori di calcio $p_1, p_2, \ldots, p_n$. Supponiamo che n sia un multiplo di 7. Ciascuno di loro può giocare come portiere (g), difensore (d), centrocampista (m), attaccante (f). Ognuno di loro sceglie esattamente due di queste opzioni (dati di input). Il problema è quello di dividerli in k squadre dove ciascuna squadra ha esattamente un g, almeno un d, almeno un m e almeno un f. Un ruolo unico viene assegnato a ciascuno di loro tra uno dei due dichiarati in input. Se una squadra ha esattamente 3 d, 2 m e 1 f, la sua penalità è 0. Se una squadra ha esattamente 2 d, 3 m e 1 f, la sua penalità è 0. Decidi (sei libero di assegnare valori) penalità non nulle per tutte le altre conformazioni. Trova una soluzione che minimizzi la somma delle penalità. Viene fornito anche un elenco di vincoli del tipo differentteam (p_i, p_j) in input che costringono p_i e p_j a essere in squadre diverse.

Il problema affrontato è un `Constraint Optimization Problem (COP)`. Per risolverlo, sono stati sviluppati due programmi che impiegano due tecniche di programmazione distinte: **programmazione per vincoli con MiniZinc** e **Answer Set Programming utilizzando il solver Clingo**.
Entrambi gli approcci sono stati selezionati per la loro efficacia nella gestione di problemi complessi e vincolati, sebbene differiscano nelle metodologie e negli strumenti adottati per ottenere la soluzione ottimale.
Nelle sezioni seguenti, verranno fornite descrizioni approfondite di entrambi gli approcci. Inoltre, sarà presentata un'analisi dei risultati ottenuti da ciascun metodo.

2 Modellazione del problema

Prima di iniziare la fase di programmazione, ho proceduto con lo studio e l'analisi del testo. Di seguito riporto alcune assunzioni che ho fatto:
Le assunzioni fatte per la risoluzione del problema sono le seguenti:

- I giocatori iniziali sono n e n deve sempre essere un numero multiplo di 7.
- Per le k squadre, non è necessario che ogni squadra abbia esattamente 7 giocatori; ogni squadra può avere un minimo di 4 giocatori (uno per ogni ruolo) e un massimo di n.
- Il numero di squadre k deve essere scelto in modo tale da avere senso rispetto al numero totale di giocatori; altrimenti, il problema non è soddisfacibile.
- Ogni giocatore può ricoprire solo uno dei due ruoli selezionati per la sua assegnazione finale.
- Ogni squadra deve avere esattamente un portiere (g), almeno un difensore (d), almeno un centrocampista (m) e almeno un attaccante (f).
- È previsto un sistema di penalità per configurazioni di ruolo specifiche, dove determinate combinazioni non comportano penalità.
- Sono forniti vincoli che richiedono che alcune coppie di giocatori siano assegnati a squadre diverse.

3 Soluzione Minizinc

Nella seguente sezione troverai la spiegazione del modello Minzinc che è stato costruito per risolvere il problema.

3.1 Input

```
n = 7;
k = 2;
roles = [
    {m, f}, % Ruolo giocatore 1
    {d, m}, % Ruolo giocatore 2
    {m, d},
    {d, g},
    {d, g},
    {g, d},
    {d, g}  % Ruolo giocatore n
];
num_pairs = 1;
player1 = [5];
player2 = [1];
```

Listing 1: Esempio di input MiniZinc

Nel codice mostrato in Listing 1, è possibile vedere quali sono i dati che il modello vuole in input. È necessario inserire:

- Il numero di giocatori `n`
- Il numero di squadre `k`
- Il ruolo dei giocatori `roles`
- Il numero di coppie di giocatori che non possono giocare nella stessa squadra `num_pairs`

- `player1` e `player2` contengono i giocatori che non possono giocare insieme. I giocatori contenuti in posizioni uguali negli array non possono giocare nella stessa squadra. Per esempio, in Listing1, il giocatore 5 e il giocatore 1 devono giocare in squadre diverse.

3.2 I vincoli Minizinc

Per la corretta modellazione del problema, sono stati definiti diversi `vincoli` che regolano l'assegnazione dei ruoli ai giocatori all'interno delle squadre, garantendo così il rispetto delle regole del gioco e l'equilibrio tra i team. Di seguito, vengono descritti i principali vincoli implementati:

```
constraint
forall(t in 1..k) (
    % Ogni squadra deve avere esattamente un
        portiere
    sum([assigned_role[j] == g | j in 1..n
        where team[j] == t]) == 1 /\
    % Almeno un difensore
    sum([assigned_role[j] == d | j in 1..n
        where team[j] == t]) >= 1 /\
    % Almeno un centrocampista
    sum([assigned_role[j] == m | j in 1..n
        where team[j] == t]) >= 1 /\
    % Almeno un attaccante
    sum([assigned_role[j] == f | j in 1..n
        where team[j] == t]) >= 1
);
```

Listing 2: Vincolo 1 Minizinc

```
% Ogni giocatore deve essere assegnato a uno
    dei suoi ruoli scelti
constraint
    forall(i in 1..n) (
        assigned_role[i] in roles[i]
    );
```

Listing 3: Vincolo 2 Minizinc

```
    constraint
    forall(i in 1..num_pairs) (
        team[player1[i]] != team[player2[i]]
    );
```

Listing 4: Vincolo 3 Minizinc

Il **Vincolo 1** assicura che ogni squadra abbia esattamente un portiere e almeno un giocatore per ciascuna delle altre posizioni chiave: difensore, centrocampista e attaccante. Questo garantisce che ogni team abbia una struttura bilanciata e conforme alle regole del gioco.
Il **Vincolo 2** specifica che ogni giocatore deve essere assegnato a uno dei ruoli per cui è stato selezionato, assicurando che le preferenze e le competenze dei giocatori siano rispettate nella formazione delle squadre.
Infine, il **Vincolo 3** stabilisce che due giocatori predefiniti non devono appartenere alla stessa squadra. Questo vincolo può essere utilizzato, ad esempio, per evitare che due giocatori incompatibili giochino insieme o per distribuire equamente i giocatori più forti tra le squadre.
Questi vincoli sono fondamentali per garantire una formazione delle squadre che sia equilibrata e conforme alle regole stabilite, facilitando così un'esperienza di gioco equa e competitiva.

3.3 Penalità

Nella competizione è stato previsto l'assegnamento di penalità alle squadre che non rispettano la configurazione richiesta del numero di giocatori nei vari ruoli. Questo criterio garantisce una formazione equilibrata e corretta per ciascuna squadra.
In particolare, le formazioni senza penalità (penalità 0) devono rispettare esattamente uno dei seguenti schemi:

- 3 difensori, 2 centrocampisti, 1 attaccante, 1 portiere
- 2 difensori, 3 centrocampisti, 1 attaccante, 1 portiere

Qualsiasi altra combinazione di giocatori comporta una penalità (penalità 1). Di seguito è riportato il codice MiniZinc utilizzato per calcolare le penalità per ogni squadra. L'array `penalties` contiene le penalità assegnate a ciascuna squadra in base alla conformità con i requisiti sopra descritti.

```
array[1..k] of var 0..1: penalties;
constraint
    forall(t in 1..k) (
        let {
            var int: d_count = sum([
                assigned_role[j] == d | j in
                 1..n where team[j] == t]),
            var int: m_count = sum([
                assigned_role[j] == m | j in
                 1..n where team[j] == t]),
            var int: f_count = sum([
                assigned_role[j] == f | j in
                 1..n where team[j] == t])
        } in
        if d_count == 3 /\ m_count == 2 /\
            f_count == 1 then
             penalties[t] = 0
        elseif d_count == 2 /\ m_count == 3
            /\ f_count == 1 then
             penalties[t] = 0
        else
             penalties[t] = 1
        endif
    );
```

Listing 5: Penalità

3.4 Obiettivo e risultati

L'obiettivo del programma è trovare la soluzione che minimizzi le penalità (vedi Listing 6).

```
solve minimize total_penalty;
```

Listing 6: Obiettivo del problema

Per valutare le performance del modello implementato, sono stati creati tre gruppi di test con livelli di difficoltà differenti: facile, medio e difficile. Ogni gruppo è composto da 10 istanze, ciascuna delle quali è stata generata in modo casuale, includendo variabili come il numero di giocatori, il numero di squadre, i ruoli dei giocatori e l'assegnazione dei giocatori a squadre diverse. I risultati dei test sono stati registrati e confrontati per analizzare l'efficacia dell'algoritmo utilizzato. I file di input e output relativi sono inclusi nel progetto e disponibili per consultazione.

3.5 Struttura dei Test

- **Facile**: 10 istanze con parametri relativamente semplici.
- **Medio**: 10 istanze con parametri di difficoltà intermedia.
- **Difficile**: 10 istanze con parametri complessi.

3.6 Metodologia

Per ciascuna istanza sono stati utilizzati due solver: Gecode e Chuffed. Il tempo massimo di esecuzione (timeout) per ogni istanza è stato impostato a 5 minuti. Se un solver non riesce a trovare una soluzione entro questo tempo, viene segnato come *TIMEOUT*.

3.7 Risultati

I risultati ottenuti sono riportati nella Tabella 1, che mostra il tempo di esecuzione in secondi per ogni istanza risolta dai solver Gecode e Chuffed.
Come è possibile vedere dai riusltati il sover Chuffed giugnge a conclusione molto più velocemente del solver Gecode.
`TIMEOUT` vuol dire che ha superato la soglia massima di tempo che è stata impostata a 5 minuti.

3.8 Analisi dei Risultati

Come si può osservare dalla Tabella 1, il solver Chuffed riesce a concludere i calcoli in tempi significativamente più brevi rispetto al solver Gecode in tutte le istanze. In particolare:

- **Istanze Facili**: Entrambi i solver completano tutte le istanze entro il tempo limite, ma Chuffed risolve le istanze con tempi generalmente inferiori rispetto a Gecode.
- **Istanze Medie**: Gecode fallisce (*TIMEOUT*) in molte delle istanze, mentre Chuffed riesce a completare tutte le istanze entro il tempo limite.
- **Istanze Difficili**: Gecode non riesce a completare la maggior parte delle istanze entro il tempo limite, mentre Chuffed riesce a risolvere alcune istanze ma fallisce (*TIMEOUT*) nelle più complesse.

Si può osservare che, talvolta, istanze simili in `n` e `k` presentano tempi di esecuzione differenti. Questo avviene perché variano i ruoli dei giocatori e le assegnazioni dei giocatori a squadre diverse.

Table 1: Performance Minizinc

Livello	n	k	Gecode (s)	Chuffed (s)
Facile 1	7	2	0.59	0.16
Facile 2	7	1	0.18	0.13
Facile 3	7	2	0.28	0.17
Facile 4	7	2	0.23	0.18
Facile 5	7	1	0.14	0.13
Facile 6	7	2	0.31	0.18
Facile 7	7	1	0.16	0.13
Facile 8	7	2	0.17	0.17
Facile 9	7	2	0.47	0.17
Facile 10	7	2	0.18	0.19
Medio 1	14	3	TIMEOUT	1.77
Medio 2	14	3	TIMEOUT	3.12
Medio 3	21	2	15.22	0.82
Medio 4	14	3	TIMEOUT	0.27
Medio 5	21	3	2.15	0.34
Medio 6	21	3	23.8	0.37
Medio 7	14	3	TIMEOUT	0.44
Medio 8	14	3	TIMEOUT	1.18
Medio 9	14	3	TIMEOUT	1.96
Medio 10	14	3	TIMEOUT	1.96
Difficile 1	28	2	83.84	0.53
Difficile 2	28	3	TIMEOUT	15.71
Difficile 3	28	4	TIMEOUT	1.27
Difficile 4	28	4	TIMEOUT	10.02
Difficile 5	28	2	TIMEOUT	0.55
Difficile 6	28	4	TIMEOUT	30.36
Difficile 7	28	3	TIMEOUT	TIMEOUT
Difficile 8	35	3	TIMEOUT	13.43
Difficile 9	28	4	TIMEOUT	TIMEOUT
Difficile 10	28	4	TIMEOUT	TIMEOUT

4 ASP

In questa sezione verrà spiegato come risolvere il problema di ottimizzazione vincolata (COP) utilizzando il risolutore ASP Clingo.

4.1 Input

Come si può osservare nel Listing 7, i file di input per Clingo presentano una struttura simile a quella dei file MiniZinc (Listing 1), ma sono adattati alla sintassi specifica della programmazione logica. Le differenze risiedono principalmente nei paradigmi di programmazione utilizzati, che richiedono una modifica della sintassi per soddisfare i requisiti di Clingo.

```
num_squadre(2).
player(1, difensore, portiere).
player(2, difensore, portiere).
player(3, portiere, attaccante).
player(4, difensore, portiere).
player(5, difensore, centrocampista).
player(6, centrocampista, difensore).
player(7, difensore, centrocampista).
player(8, centrocampista, portiere).
player(9, portiere, difensore).
player(10, portiere, difensore).
player(11, portiere, centrocampista).
player(12, centrocampista, difensore).
player(13, portiere, centrocampista).
player(14, portiere, difensore).
different_team(1, 2).
```

Listing 7: ASP Input

4.2 Modello ASP

Il modello ASP utilizza la sintassi della programmazione logica per definire i vincoli e le penalità associati alla configurazione delle squadre. La struttura del modello è la seguente:

- **Definizione dei Ruoli:** Viene definito l'insieme dei ruoli disponibili (attaccante, difensore, centrocampista e portiere).
- **Definizione delle Squadre:** Le squadre sono definite utilizzando il numero totale di squadre fornito nel file di input.
- **Assegnazione dei Giocatori:** Ogni giocatore deve essere assegnato a una sola squadra e a un solo ruolo. Inoltre, deve essere uno dei ruoli che preferisce.
- **Vincoli delle Squadre:** Ogni squadra deve avere esattamente un portiere e almeno un attaccante, un difensore e un centrocampista. Inoltre, i vincoli di separazione tra giocatori sono rispettati.
- **Calcolo delle Penalità:** Le penalità sono calcolate in base alla configurazione dei ruoli nelle squadre, con penalità zero per configurazioni specifiche e penalità non nulle per tutte le altre conformazioni.

4.3 Vincoli e Penalità

Per garantire che il programma soddisfi i requisiti specificati, sono stati introdotti dei vincoli simili a quelli utilizzati nel programma MiniZinc ma con la sintassi adattata a questo tipo di programmazione.

I vincoli imposti sono i seguenti:

- Ogni giocatore deve essere assegnato a una squadra e deve avere un ruolo.
- Il ruolo assegnato a un giocatore deve corrispondere a uno dei ruoli scelti dal giocatore stesso.
- Ogni squadra deve avere esattamente un portiere.
- Ogni squadra deve includere almeno un centrocampista, un attaccante e un difensore.
- Alcuni giocatori non possono essere assegnati alla stessa squadra.

Il Listing 8 mostra la definizione dei vincoli.

```
% Assegnazione di un giocatore a una
    squadra e a un ruolo
{assigned(P, T, R) : team(T), role(R)} =
    1 :- player(P, _, _).

% Il giocatore deve ricoprire uno dei
    ruoli che ha scelto
:- player(P, R1, R2), assigned(P, T, R),
    R != R1, R != R2.

% Ogni squadra deve avere esattamente un
    portiere
1 {assigned(P, T, portiere) : player(P,
    _, _)} 1 :- team(T).

% Ogni squadra deve avere almeno un
    attaccante, un difensore e un
    centrocampista
1 {assigned(P, T, attaccante) : player(P
    , _, _)} :- team(T).
1 {assigned(P, T, difensore) : player(P,
    _, _)} :- team(T).
1 {assigned(P, T, centrocampista) :
    player(P, _, _)} :- team(T).

% Vincoli di esclusione per giocatori
    appartenenti a squadre diverse
:- different_team(P1, P2), assigned(P1,
    T, _), assigned(P2, T, _).
```

Listing 8: Vincoli ASP

Sono state anche introdotte delle penalità, come indicato nel Listing 9. Le penalità sono calcolate in base alla configurazione dei membri all'interno della squadra:

```
% Calcolo delle penalità per
    configurazioni ottimali
penalty(T, 0) :-
    team(T),
    #count{P : assigned(P, T, difensore)
        } == 3,
    #count{P : assigned(P, T,
        centrocampista)} == 2,
    #count{P : assigned(P, T, attaccante
        )} == 1.

penalty(T, 0) :-
    team(T),
    #count{P : assigned(P, T, difensore)
        } == 2,
    #count{P : assigned(P, T,
        centrocampista)} == 3,
    #count{P : assigned(P, T, attaccante
        )} == 1.

% Penalità per configurazioni non
    ottimali
penalty(T, 1) :-
    team(T),
    not penalty(T, 0).

% Calcolo della penalità totale per
    tutte le squadre
total_penalty(P) :-
    P = #sum { P1 : penalty(T, P1), team
        (T) }.
```

Listing 9: Penalità ASP

Questi vincoli e penalità assicurano che ogni squadra soddisfi le condizioni ottimali e penalizzano le configurazioni non conformi ai requisiti stabiliti.

4.4 Obiettivo e risultati

Anche in questo caso sono stati creati 3 gruppi di test con livelli di difficoltà crescenti:

- Facile: 10 istanze
- Medio: 10 istanze
- Difficile: 10 istanze

Anche in questo caso il tempo massimo per ogni istanza è stato impostato a 5 minuti. Se non viene trovato un risultato entro il tempo limite, viene segnato `TIMEOUT`. Nella Tabella 2 puoi vedere i risultati delle performance del modello utilizzando la stessa tipologia dei dati di input che sono stati utilizzati nel programma Minzinc. Come è possibile osservare, ASP va generalmente meglio di MiniZinc; per questo motivo ho creato un altro set di dati difficili dove eseguire questo modello per vedere quali tipologie di dati lo portano in `TIMEOUT`. Nella Tabella 3 è possibile vedere i risultati con dati di input più complessi, dove ho aumentato sia n che k che il numero di vincoli `all_different` (i file di input completi si possono trovare nella documentazione del progetto).

Table 2: Performance ASP

Livello	n	k	Time (s)
Facile 1	14	1	0.004
Facile 2	14	2	0.02
Facile 3	14	2	0.005
Facile 4	14	2	0.005
Facile 5	14	1	0.004
Facile 6	14	1	0.004
Facile 7	14	2	0.005
Facile 8	14	2	0.005
Facile 9	14	2	0.005
Facile 10	14	2	0.005
Medio 1	14	2	0.005
Medio 2	14	3	0.181
Medio 3	14	2	0.005
Medio 4	14	2	0.006
Medio 5	14	2	0.006
Medio 6	14	2	0.005
Medio 7	14	3	0.007
Medio 8	21	3	0.008
Medio 9	14	3	0.007
Medio 10	14	3	0.007
Difficile 1	28	4	0.015
Difficile 2	35	2	0.013
Difficile 3	28	3	48.409
Difficile 4	28	3	6.160
Difficile 5	28	2	0.017
Difficile 6	28	3	76.641
Difficile 7	28	2	0.007
Difficile 8	28	3	62.021
Difficile 9	28	3	75.452
Difficile 10	35	3	104.743

Livello	**n**	**k**	**Time (s)**
Facile 1	2	21	0.013
Facile 2	1	21	0.005
Facile 3	2	21	0.006
Facile 4	1	21	0.005
Facile 5	2	21	0.028
Facile 6	2	21	0.091
Facile 7	2	21	0.025
Facile 8	2	21	0.011
Facile 9	2	21	0.007
Medio 1	1	21	0.005
Medio 2	3	28	0.017
Medio 3	4	28	0.012
Medio 4	2	28	0.008
Medio 5	4	28	0.014
Difficile 1	3	28	12.623
Difficile 2	3	28	78.036
Difficile 3	4	28	0.012
Difficile 4	4	28	0.012
Difficile 5	2	35	0.01
Difficile 6	2	35	0.01
Difficile 7	5	63	111.509
Difficile 8	6	70	TIMEOUT
Difficile 9	7	63	TIMEOUT
Difficile 10	8	63	TIMEOUT
Difficile 11	5	63	TIMEOUT
Difficile 12	8	70	TIMEOUT
Difficile 13	4	63	0.378
Difficile 14	6	63	TIMEOUT
Difficile 15	7	63	TIMEOUT
Difficile 16	7	63	TIMEOUT

Table 3: Performance ASP 2

5 Generazione dei File di Input

Per facilitare la sperimentazione e la valutazione del problema di ottimizzazione vincolata (COP) descritto, è stato creato un sistema automatizzato per la generazione dei file di input necessari per i risolutori MiniZinc e Clingo. Questo processo è stato realizzato tramite uno script Python personalizzato, progettato per generare istanze di difficoltà variabile (facili, medie e difficili) e per riempire i file con dati casuali.

5.1 Script di Generazione in Python

Lo script Python utilizzato è stato sviluppato per creare automaticamente i file di input in base a parametri specifici. Il codice sorgente è stato progettato per generare file di input sia per MiniZinc che per Clingo, garantendo la compatibilità con i requisiti di entrambi i risolutori. Di seguito, vengono descritti i principali aspetti dello script:

- **Generazione di Giocatori e Ruoli:** Per ciascun giocatore, lo script assegna due ruoli preferiti tra portiere (g), difensore (d), centrocampista (m) e attaccante (f). Le preferenze sono generate in modo casuale, rispettando le probabilità specificate.
- **Generazione di Vincoli:** Lo script crea vincoli di separazione tra giocatori selezionati casualmente.
- **Creazione dei File di Input:** Dopo la generazione dei dati, alcuni script compilano i file di input per MiniZinc e Clingo nei formati richiesti e creano dei file di output.

5.2 Istanze e Difficoltà

Le istanze generate dallo script sono classificate in tre categorie di difficoltà:

- **Facili:** Queste istanze hanno una configurazione di base con vincoli semplici e una distribuzione equilibrata dei ruoli tra i giocatori. Sono progettate per essere risolvibili con tempi di esecuzione brevi e sono utili per testare la correttezza del modello.
- **Medie:** Le istanze medie presentano una complessità maggiore, con vincoli più restrittivi e una distribuzione dei ruoli più variabile. Queste istanze offrono una sfida maggiore rispetto a quelle facili e sono utilizzate per valutare le prestazioni e l'efficienza del risolutore.
- **Difficili:** Le istanze difficili sono progettate con una complessità elevata, inclusi vincoli complessi e una distribuzione dei ruoli altamente variabile. Queste istanze sono utili per testare i limiti del risolutore e per valutare la robustezza della soluzione proposta.

5.3 Riempimento con Dati Casuali

Per garantire la variabilità e la rappresentatività delle istanze, lo script riempie i file di input con dati casuali. I dati casuali sono generati in modo da simulare situazioni realistiche e sfidanti, mantenendo al contempo un controllo sulla distribuzione dei ruoli e sui vincoli imposti. Questo approccio consente di creare un ampio set di istanze per testare diverse configurazioni e strategie di risoluzione.

L'automazione della generazione dei file di input tramite lo script Python ha facilitato notevolmente la creazione di istanze variabili e la sperimentazione con diversi scenari. I file generati forniscono una base solida per l'analisi e la risoluzione del problema di ottimizzazione vincolata utilizzando MiniZinc e Clingo, contribuendo così a una valutazione completa delle soluzioni proposte.

6 Conclusioni

In questo report abbiamo esplorato il problema di organizzazione di un torneo di calcio a 7 attraverso due approcci distinti: **Programmazione per Vincoli con MiniZinc** e **Answer Set Programming (ASP) con Clingo**. Entrambi i metodi sono stati impiegati per risolvere il problema di ottimizzazione vincolata, mirato a minimizzare le penalità associate alle configurazioni delle squadre, rispettando al contempo una serie di vincoli prestabiliti.

6.1 Sintesi dei Risultati

I risultati ottenuti con MiniZinc e Clingo mostrano chiaramente le differenze di performance tra i due approcci:

- **MiniZinc**: I test condotti con MiniZinc hanno rivelato che il solver Chuffed, utilizzato per la risoluzione del problema, è significativamente più veloce rispetto a Gecode, specialmente con istanze di difficoltà media e difficile. I tempi di esecuzione mostrano che Chuffed riesce a gestire più efficacemente la complessità crescente delle istanze, risolvendo la maggior parte dei casi difficili entro il tempo limite, mentre Gecode spesso incontra difficoltà, portando a *TIMEOUT* in molti casi.
- **ASP con Clingo**: Clingo, utilizzato per la risoluzione tramite programmazione logica, ha dimostrato una notevole efficienza anche per istanze più complesse. Sebbene il tempo di esecuzione possa variare in base alla specificità dei vincoli e delle penalità, Clingo ha generalmente ottenuto risultati competitivi, risolvendo in tempi accettabili anche le istanze più difficili.

6.2 Riflessioni sui Metodi Utilizzati

- **MiniZinc**: La programmazione per vincoli con MiniZinc si è rivelata un metodo potente per affrontare il problema, soprattutto per la sua capacità di gestire vincoli complessi e penalità multiple. Tuttavia, la scelta del solver influisce notevolmente sulle prestazioni, come evidenziato dalla differenza tra Chuffed e Gecode.
- **ASP con Clingo**: L'approccio basato su ASP ha dimostrato una buona capacità di adattamento ai vincoli specifici del problema. La flessibilità della programmazione logica consente una definizione chiara dei vincoli e delle penalità, sebbene la performance possa dipendere dalla complessità del problema e dalla qualità del modello ASP definito.

6.3 Considerazioni Finali

Entrambi gli approcci hanno mostrato la loro validità nella risoluzione del problema di assegnazione delle squadre, con MiniZinc particolarmente efficace per problemi con vincoli più rigidi e penalità specifiche, e Clingo eccellente per la sua flessibilità nella definizione e gestione di vincoli complessi. La scelta tra MiniZinc e ASP dipende in gran parte dalle specifiche esigenze del problema e dalle preferenze dell'utente in termini di sintassi e metodologie di risoluzione.
In conclusione, il report ha dimostrato che la modellazione e la risoluzione di problemi complessi come quello dell'organizzazione di tornei possono beneficiare notevolmente dall'utilizzo di tecniche avanzate di programmazione e ottimizzazione, offrendo soluzioni pratiche e efficienti a problemi reali.